FORM

4.2

Generated by Doxygen 1.9.1

Chapter 1

Data Structure Index

1.1 Data Structures

Here are the data structures with brief descriptions:

Chapter 2

File Index

2.1 File List

Here is a list of all documented files with brief descriptions:

Chapter 3

Data Structure Documentation

3.1 AllGlobals Struct Reference

`#include <structs.h>`

Public Member Functions

- **PADPOSITION** (0, 0, 0, 0, sizeof(struct P_const)+sizeof(struct T_const)+sizeof(struct X_const))

Data Fields

- struct M_const **M**
- struct C_const **C**
- struct S_const **S**
- struct R_const **R**
- struct N_const **N**
- struct O_const **O**
- struct P_const **P**
- struct T_const **T**
- struct X_const **X**

3.1.1 Detailed Description

Without pthreads (FORM) the ALLGLOBALS struct has all the global variables

Definition at line 2457 of file structs.h.

The documentation for this struct was generated from the following file:

- structs.h

3.2 ARGBUFFER Struct Reference

Data Fields

- WORD * **buffer**
- DOLLARS **dollar**
- LONG **size**
- int **type**
- int **dummy**

3.2.1 Detailed Description

Definition at line 601 of file ratio.c.

The documentation for this struct was generated from the following file:

- ratio.c

3.3 bit_field Struct Reference

```
#include <structs.h>
```

Data Fields

- UBYTE **bit_0**: 1
- UBYTE **bit_1**: 1
- UBYTE **bit_2**: 1
- UBYTE **bit_3**: 1
- UBYTE **bit_4**: 1
- UBYTE **bit_5**: 1
- UBYTE **bit_6**: 1
- UBYTE **bit_7**: 1

3.3.1 Detailed Description

The struct bit_field is used by set_in, set_set, set_del and set_sub. They in turn are used in pre.c to toggle bits that indicate whether a character can be used as a separator of function arguments. This facility is used in the communication with external channels.

Definition at line 878 of file structs.h.

The documentation for this struct was generated from the following file:

- structs.h

3.4 BrAcKeTiNdEx Struct Reference

Public Member Functions

- **PADPOSITION** (0, 2, 0, 0, 0)

Data Fields

- POSITION **start**
- POSITION **next**
- LONG **bracket**
- LONG **termsinbracket**

3.4.1 Detailed Description

Definition at line 316 of file structs.h.

The documentation for this struct was generated from the following file:

- structs.h

3.5 BracketInfo Struct Reference

Public Member Functions

- **BracketInfo** (const std::vector< int > &pattern, int num_terms, const poly *p)
- bool **operator<** (const BracketInfo &rhs) const

Data Fields

- std::vector< int > **pattern**
- int **num_terms**
- int **dummy**
- const poly * **p**

3.5.1 Detailed Description

Definition at line 1391 of file polygcd.cc.

The documentation for this struct was generated from the following file:

- polygcd.cc

3.6 BrAcKeTiNfO Struct Reference

Data Fields

- BRACKETINDEX * indexbuffer
- WORD * bracketbuffer
- LONG **bracketbuffersize**
- LONG **indexbuffersize**
- LONG **bracketfill**
- LONG **indexfill**
- WORD SortType

3.6.1 Detailed Description

Definition at line 328 of file structs.h.

3.6.2 Field Documentation

3.6.2.1 indexbuffer

```
BRACKETINDEX* BrAcKeTiNfO::indexbuffer
```

[D]

Definition at line 329 of file structs.h.

3.6.2.2 bracketbuffer

```
WORD* BrAcKeTiNfO::bracketbuffer
```

[D]

Definition at line 330 of file structs.h.

3.6.2.3 SortType

```
WORD BrAcKeTiNfO::SortType
```

The sorting criterium used (like POWERFIRST etc)

Definition at line 335 of file structs.h.

The documentation for this struct was generated from the following file:

- structs.h

3.7 buflPstruct Struct Reference

Data Fields

- LONG **i**
- struct ExPrEsSiOn **e**

3.7.1 Detailed Description

Definition at line 3908 of file parallel.c.

The documentation for this struct was generated from the following file:

- parallel.c

3.8 C_const Struct Reference

```
#include <structs.h>
```

Public Member Functions

- **PADPOSITION** (47, 8+3 ∗MAXNEST, 70, 48+3 ∗MAXNEST+MAXREPEAT, COMMERCIALSIZE+MAXFLAGS+4+sizeof(LIST) ∗17)

Data Fields

- set_of_char separators
- POSITION **StoreFileSize**
- NAMETREE ∗ dollarnames
- NAMETREE ∗ exprnames
- NAMETREE ∗ varnames
- LIST ChannelList
- LIST DubiousList
- LIST FunctionList
- LIST ExpressionList
- LIST IndexList
- LIST SetElementList
- LIST SetList
- LIST SymbolList
- LIST VectorList
- LIST PotModDolList
- LIST ModOptDolList
- LIST TableBaseList
- LIST cbufList
- LIST **AutoSymbolList**
- LIST **AutoIndexList**
- LIST **AutoVectorList**
- LIST **AutoFunctionList**

- NAMETREE * autonames
- LIST * **Symbols**
- LIST * **Indices**
- LIST * **Vectors**
- LIST * **Functions**
- NAMETREE ** **activenames**
- STREAM * Streams
- STREAM * CurrentStream
- SWITCH * **SwitchArray**
- WORD * **SwitchHeap**
- LONG * termstack
- LONG * termsortstack
- UWORD * cmod
- UWORD * powmod
- UWORD * modpowers
- UWORD * **halfmod**
- WORD * **ProtoType**
- WORD * **WildC**
- LONG * IfHeap
- LONG * IfCount
- LONG * IfStack
- UBYTE * iBuffer
- UBYTE * iPointer
- UBYTE * iStop
- UBYTE ** LabelNames
- WORD * FixIndices
- WORD * termsumcheck
- UBYTE * WildcardNames
- int * Labels
- SBYTE * tokens
- SBYTE * toptokens
- SBYTE * endoftokens
- WORD * tokenarglevel
- UWORD * **modinverses**
- UBYTE * **Fortran90Kind**
- WORD ** **MultiBracketBuf**
- UBYTE * **extrasym**
- WORD * **doloopstack**
- WORD * **doloopnest**
- char * CheckpointRunAfter
- char * CheckpointRunBefore
- WORD * IfSumCheck
- WORD * **CommuteInSet**
- UBYTE * **TestValue**
- LONG **argstack** [MAXNEST]
- LONG **insidestack** [MAXNEST]
- LONG **inexprstack** [MAXNEST]
- LONG **iBufferSize**
- LONG **TransEname**
- LONG **ProcessBucketSize**
- LONG **mProcessBucketSize**
- LONG **CModule**
- LONG **ThreadBucketSize**
- LONG CheckpointStamp
- LONG CheckpointInterval

- int cbufnum
- int **AutoDeclareFlag**
- int NoShowInput
- int **ShortStats**
- int **compiletype**
- int **firstconstindex**
- int **insidefirst**
- int **minsidefirst**
- int **wildflag**
- int **NumLabels**
- int **MaxLabels**
- int **IDefDim**
- int **IDefDim4**
- int **NumWildcardNames**
- int **WildcardBufferSize**
- int **MaxIf**
- int **NumStreams**
- int **MaxNumStreams**
- int **firstctypemessage**
- int **tablecheck**
- int **idoption**
- int **BottomLevel**
- int **CompileLevel**
- int **TokensWriteFlag**
- int **UnsureDollarMode**
- int **outsidefun**
- int **funpowers**
- int **WarnFlag**
- int **StatsFlag**
- int **NamesFlag**
- int **CodesFlag**
- int **SetupFlag**
- int **SortType**
- int **ISortType**
- int **ThreadStats**
- int **FinalStats**
- int **OldParallelStats**
- int **ThreadsFlag**
- int **ThreadBalancing**
- int **ThreadSortFileSynch**
- int **ProcessStats**
- int **BracketNormalize**
- int **maxtermlevel**
- int **dumnumflag**
- int **bracketindexflag**
- int **parallelflag**
- int **mparallelflag**
- int **inparallelflag**
- int **partodoflag**
- int **properorderflag**
- int **vetofilling**
- int **tablefilling**
- int **vetotablebasefill**
- int **exprfillwarning**
- int **lhdollarflag**

- int **NoCompress**
- int **IsFortran90**
- int **MultiBracketLevels**
- int **topolynomialflag**
- int **ffbufnum**
- int **OldFactArgFlag**
- int **MemDebugFlag**
- int **OldGCDflag**
- int **WTimeStatsFlag**
- int **doloopstacksize**
- int **dolooplevel**
- int CheckpointFlag
- int **SizeCommuteInSet**
- int **origin**
- int **vectorlikeLHS**
- WORD **argsumcheck** [MAXNEST]
- WORD **insidesumcheck** [MAXNEST]
- WORD **inexprsumcheck** [MAXNEST]
- WORD **RepSumCheck** [MAXREPEAT]
- WORD **IUniTrace** [4]
- WORD **RepLevel**
- WORD **arglevel**
- WORD **insidelevel**
- WORD **inexprlevel**
- WORD **termlevel**
- WORD **MustTestTable**
- WORD **DumNum**
- WORD **ncmod**
- WORD **npowmod**
- WORD **modmode**
- WORD **nhalfmod**
- WORD **DirtPow**
- WORD **IUnitTrace**
- WORD **NwildC**
- WORD **ComDefer**
- WORD **CollectFun**
- WORD **AltCollectFun**
- WORD **OutputMode**
- WORD **Cnumpows**
- WORD **OutputSpaces**
- WORD **OutNumberType**
- WORD **DidClean**
- WORD **IfLevel**
- WORD **WhileLevel**
- WORD **SwitchLevel**
- WORD **SwitchInArray**
- WORD **MaxSwitch**
- WORD **LogHandle**
- WORD **LineLength**
- WORD **StoreHandle**
- WORD **HideLevel**
- WORD **IPolyFun**
- WORD **IPolyFunInv**
- WORD **IPolyFunType**
- WORD **IPolyFunExp**

- WORD **IPolyFunVar**
- WORD **IPolyFunPow**
- WORD **SymChangeFlag**
- WORD **CollectPercentage**
- WORD **ShortStatsMax**
- WORD **extrasymbols**
- WORD **PolyRatFunChanged**
- WORD **ToBeInFactors**
- WORD **InnerTest**
- UBYTE **Commercial** [COMMERCIALSIZE+2]
- UBYTE **debugFlags** [MAXFLAGS+2]

3.8.1 Detailed Description

The C_const struct is part of the global data and resides in the ALLGLOBALS struct #A under the name #C. We see it used with the macro #AC as in AC.exprnames. It contains variables that involve the compiler and objects set during compilation.

Definition at line 1613 of file structs.h.

3.8.2 Field Documentation

3.8.2.1 separators

```
set_of_char C_const::separators
```

Separators in #call and #do

Definition at line 1614 of file structs.h.

3.8.2.2 dollarnames

```
NAMETREE* C_const::dollarnames
```

[D] Names of dollar variables

Definition at line 1616 of file structs.h.

3.8.2.3 exprnames

```
NAMETREE* C_const::exprnames
```

[D] Names of expressions

Definition at line 1617 of file structs.h.

3.8.2.4 varnames

```
NAMETREE* C_const::varnames
```

[D] Names of regular variables

Definition at line 1618 of file structs.h.

3.8.2.5 ChannelList

```
LIST C_const::ChannelList
```

Used for the #write statement. Contains CHANNEL

Definition at line 1619 of file structs.h.

3.8.2.6 DubiousList

```
LIST C_const::DubiousList
```

List of dubious variables. Contains #DUBIOUSV. If not empty -> no execution

Definition at line 1621 of file structs.h.

3.8.2.7 FunctionList

```
LIST C_const::FunctionList
```

List of functions and properties. Contains FUNCTIONS

Definition at line 1623 of file structs.h.

3.8.2.8 ExpressionList

```
LIST C_const::ExpressionList
```

List of expressions, locations etc.

Definition at line 1624 of file structs.h.

3.8.2.9 IndexList

```
LIST C_const::IndexList
```

List of indices

Definition at line 1625 of file structs.h.

3.8.2.10 SetElementList

```
LIST C_const::SetElementList
```

List of all elements of all sets

Definition at line 1626 of file structs.h.

3.8.2.11 SetList

```
LIST C_const::SetList
```

List of the sets

Definition at line 1627 of file structs.h.

3.8.2.12 SymbolList

```
LIST C_const::SymbolList
```

List of the symbols and their properties

Definition at line 1628 of file structs.h.

3.8.2.13 VectorList

```
LIST C_const::VectorList
```

List of the vectors

Definition at line 1629 of file structs.h.

3.8.2.14 PotModDolList

```
LIST C_const::PotModDolList
```

Potentially changed dollars

Definition at line 1630 of file structs.h.

3.8.2.15 ModOptDolList

```
LIST C_const::ModOptDolList
```

Module Option Dollars list

Definition at line 1631 of file structs.h.

3.8.2.16 TableBaseList

```
LIST C_const::TableBaseList
```

TableBase list

Definition at line 1632 of file structs.h.

3.8.2.17 cbufList

```
LIST C_const::cbufList
```

List of compiler buffers

Definition at line 1636 of file structs.h.

3.8.2.18 autonames

```
NAMETREE* C_const::autonames
```

[D] Names in autodeclare

Definition at line 1644 of file structs.h.

3.8.2.19 Streams

```
STREAM* C_const::Streams
```

(C) Pointer for AutoDeclare statement. Points either to varnames or autonames. [D] The input streams.

Definition at line 1653 of file structs.h.

3.8.2.20 CurrentStream

```
STREAM* C_const::CurrentStream
```

(C) The current input stream. Streams are: do loop, file, prevariable. points into Streams memory.

Definition at line 1654 of file structs.h.

3.8.2.21 termstack

```
LONG* C_const::termstack
```

[D] Last term statement {offset}

Definition at line 1658 of file structs.h.

3.8.2.22 termsortstack

```
LONG* C_const::termsortstack
```

[D] Last sort statement {offset}

Definition at line 1659 of file structs.h.

3.8.2.23 cmod

```
UWORD* C_const::cmod
```

[D] Local setting of modulus. Pointer to value.

Definition at line 1660 of file structs.h.

3.8.2.24 powmod

```
UWORD* C_const::powmod
```

Local setting printing as powers. Points into cmod memory

Definition at line 1661 of file structs.h.

3.8.2.25 modpowers

```
UWORD* C_const::modpowers
```

[D] The conversion table for mod-> powers.

Definition at line 1662 of file structs.h.

3.8.2.26 IfHeap

```
LONG* C_const::IfHeap
```

[D] Keeps track of where to go in if

Definition at line 1666 of file structs.h.

3.8.2.27 IfCount

```
LONG* C_const::IfCount
```

[D] Keeps track of where to go in if

Definition at line 1667 of file structs.h.

3.8.2.28 IfStack

```
LONG* C_const::IfStack
```

Keeps track of where to go in if. Points into IfHeap-memory

Definition at line 1668 of file structs.h.

3.8.2.29 iBuffer

```
UBYTE* C_const::iBuffer
```

[D] Compiler input buffer

Definition at line 1669 of file structs.h.

3.8.2.30 iPointer

```
UBYTE* C_const::iPointer
```

Running pointer in the compiler input buffer

Definition at line 1670 of file structs.h.

3.8.2.31 iStop

```
UBYTE* C_const::iStop
```

Top of iBuffer

Definition at line 1671 of file structs.h.

3.8.2.32 LabelNames

```
UBYTE** C_const::LabelNames
```

[D] List of names in label statements

Definition at line 1672 of file structs.h.

3.8.2.33 FixIndices

```
WORD* C_const::FixIndices
```

[D] Buffer of fixed indices

Definition at line 1673 of file structs.h.

3.8.2.34 termsumcheck

```
WORD* C_const::termsumcheck
```

[D] Checking of nesting

Definition at line 1674 of file structs.h.

3.8.2.35 WildcardNames

```
UBYTE* C_const::WildcardNames
```

[D] Names of ?a variables

Definition at line 1675 of file structs.h.

3.8.2.36 Labels

```
int* C_const::Labels
```

Label information for during run. Pointer into LabelNames memory.

Definition at line 1676 of file structs.h.

3.8.2.37 tokens

```
SBYTE* C_const::tokens
```

[D] Array with tokens for tokenizer

Definition at line 1677 of file structs.h.

3.8.2.38 toptokens

```
SBYTE* C_const::toptokens
```

Top of tokens

Definition at line 1678 of file structs.h.

3.8.2.39 endoftokens

```
SBYTE* C_const::endoftokens
```

End of the actual tokens

Definition at line 1679 of file structs.h.

3.8.2.40 tokenarglevel

```
WORD* C_const::tokenarglevel
```

[D] Keeps track of function arguments

Definition at line 1680 of file structs.h.

3.8.2.41 CheckpointRunAfter

```
char* C_const::CheckpointRunAfter
```

[D] Filename of script to be executed *before* creating the snapshot. =0 if no script shall be executed.

Definition at line 1687 of file structs.h.

3.8.2.42 CheckpointRunBefore

```
char* C_const::CheckpointRunBefore
```

[D] Filename of script to be executed *after* having created the snapshot. =0 if no script shall be executed.

Definition at line 1689 of file structs.h.

3.8.2.43 IfSumCheck

```
WORD* C_const::IfSumCheck
```

[D] Keeps track of if-nesting

Definition at line 1691 of file structs.h.

3.8.2.44 CheckpointStamp

```
LONG C_const::CheckpointStamp
```

Timestamp of the last created snapshot (set to Timer(0)).

Definition at line 1711 of file structs.h.

3.8.2.45 CheckpointInterval

```
LONG C_const::CheckpointInterval
```

Time interval in milliseconds for snapshots. =0 if snapshots shall be created at the end of *every* module.

Definition at line 1712 of file structs.h.

3.8.2.46 cbufnum

```
int C_const::cbufnum
```

Current compiler buffer

Definition at line 1714 of file structs.h.

3.8.2.47 NoShowInput

```
int C_const::NoShowInput
```

(C) Mode of looking for names. Set to NOAUTO (=0) or WITHAUTO (=2), cf. AutoDeclare statement

Definition at line 1717 of file structs.h.

3.8.2.48 CheckpointFlag

```
int C_const::CheckpointFlag
```

Tells preprocessor whether checkpoint code must executed. -1 : do recovery from snapshot, set by command line option; 0 : do nothing; 1 : create snapshots, set by On checkpoint statement

Definition at line 1781 of file structs.h.

The documentation for this struct was generated from the following file:

- structs.h

3.9 CbUf Struct Reference

`#include <structs.h>`

Data Fields

- WORD * Buffer
- WORD * Top
- WORD * Pointer
- WORD ** lhs
- WORD ** rhs
- LONG * CanCommu
- LONG * NumTerms
- WORD * numdum
- WORD * dimension
- COMPTREE * boomlijst
- LONG BufferSize
- int **numlhs**
- int **numrhs**
- int **maxlhs**
- int **maxrhs**
- int **mnumlhs**
- int **mnumrhs**
- int **numtree**
- int **rootnum**
- int **MaxTreeSize**

3.9.1 Detailed Description

The CBUF struct is used by the compiler. It is a compiler buffer of which since version 3.0 there can be many.

Definition at line 938 of file structs.h.

3.9.2 Field Documentation

3.9.2.1 Buffer

`WORD* CbUf::Buffer`

[D] Size in BufferSize

Definition at line 939 of file structs.h.

3.9.2.2 Top

```
WORD* CbUf::Top
```

pointer to the end of the Buffer memory

Definition at line 940 of file structs.h.

3.9.2.3 Pointer

```
WORD* CbUf::Pointer
```

pointer into the Buffer memory

Definition at line 941 of file structs.h.

3.9.2.4 lhs

```
WORD** CbUf::lhs
```

[D] Size in maxlhs. list of pointers into Buffer.

Definition at line 942 of file structs.h.

3.9.2.5 rhs

```
WORD** CbUf::rhs
```

[D] Size in maxrhs. list of pointers into Buffer.

Definition at line 943 of file structs.h.

3.9.2.6 CanCommu

```
LONG* CbUf::CanCommu
```

points into rhs memory behind WORD* area.

Definition at line 944 of file structs.h.

3.9.2.7 NumTerms

```
LONG* CbUf::NumTerms
```

points into rhs memory behind CanCommu area

Definition at line 945 of file structs.h.

3.9.2.8 numdum

```
WORD* CbUf::numdum
```

points into rhs memory behind NumTerms

Definition at line 946 of file structs.h.

3.9.2.9 dimension

```
WORD* CbUf::dimension
```

points into rhs memory behind numdum

Definition at line 947 of file structs.h.

3.9.2.10 boomlijst

```
COMPTREE* CbUf::boomlijst
```

[D] Number elements in MaxTreeSize

Definition at line 948 of file structs.h.

3.9.2.11 BufferSize

```
LONG CbUf::BufferSize
```

Number of allocated WORD's in Buffer

Definition at line 949 of file structs.h.

The documentation for this struct was generated from the following file:

- structs.h

3.10 ChAnNeL Struct Reference

`#include <structs.h>`

Data Fields

- char * name
- int handle

3.10.1 Detailed Description

When we read input from text files we have to remember not only their handle but also their name. This is needed for error messages. Hence we call such a file a channel and reserve a struct of type CHANNEL to allow to lay this link.

Definition at line 969 of file structs.h.

3.10.2 Field Documentation

3.10.2.1 name

`char* ChAnNeL::name`

[D] Name of the channel

Definition at line 970 of file structs.h.

3.10.2.2 handle

`int ChAnNeL::handle`

File handle

Definition at line 971 of file structs.h.

The documentation for this struct was generated from the following file:

- structs.h

3.11 COST Struct Reference

Data Fields

- LONG **add**
- LONG **mul**
- LONG **div**
- LONG **pow**

3.11.1 Detailed Description

Definition at line 1241 of file structs.h.

The documentation for this struct was generated from the following file:

- structs.h

3.12 CSEEq Struct Reference

Public Member Functions

- bool **operator()** (const vector< WORD > &lhs, const vector< WORD > &rhs) const

3.12.1 Detailed Description

Definition at line 1013 of file optimize.cc.

The documentation for this struct was generated from the following file:

- optimize.cc

3.13 CSEHash Struct Reference

Public Member Functions

- size_t **operator()** (const vector< WORD > &n) const

3.13.1 Detailed Description

Definition at line 1007 of file optimize.cc.

The documentation for this struct was generated from the following file:

- optimize.cc

3.14 dbase Struct Reference

Data Fields

- INIINFO **info**
- MLONG **mode**
- MLONG **tablenamessize**
- MLONG **topnumber**
- MLONG **tablenamefill**
- INDEXBLOCK ∗∗ **iblocks**
- NAMESBLOCK ∗∗ **nblocks**
- FILE ∗ **handle**
- char ∗ **name**
- char ∗ **fullname**
- char ∗ **tablenames**

3.14.1 Detailed Description

Definition at line 120 of file minos.h.

The documentation for this struct was generated from the following file:

- minos.h

3.15 DICTIONARY Struct Reference

Data Fields

- DICTIONARY_ELEMENT ∗∗ **elements**
- UBYTE ∗ **name**
- int **sizeelements**
- int **numelements**
- int **numbers**
- int **variables**
- int **characters**
- int **funwith**
- int **gnumelements**
- int **ranges**

3.15.1 Detailed Description

Definition at line 1307 of file structs.h.

The documentation for this struct was generated from the following file:

- structs.h

3.16 DICTIONARY_ELEMENT Struct Reference

Data Fields

- WORD * **lhs**
- WORD * **rhs**
- int **type**
- int **size**

3.16.1 Detailed Description

Definition at line 1300 of file structs.h.

The documentation for this struct was generated from the following file:

- structs.h

3.17 DiStRiBuTe Struct Reference

`#include <structs.h>`

Data Fields

- WORD * **obj1**
- WORD * **obj2**
- WORD * **out**
- WORD **sign**
- WORD **n1**
- WORD **n2**
- WORD **n**
- WORD **cycle** [MAXMATCH]

3.17.1 Detailed Description

The struct DISTRIBUTE is used to help the pattern matcher when matching antisymmetric tensors.

Definition at line 1049 of file structs.h.

The documentation for this struct was generated from the following file:

- structs.h

3.18 dollar_buf Struct Reference

3.18.1 Detailed Description

Definition at line 2468 of file parallel.c.

The documentation for this struct was generated from the following file:

- parallel.c

3.19 DoLIArS Struct Reference

Data Fields

- WORD * **where**
- FACDOLLAR * **factors**
- LONG **size**
- LONG **name**
- WORD **type**
- WORD **node**
- WORD **index**
- WORD **zero**
- WORD **numdummies**
- WORD **nfactors**

3.19.1 Detailed Description

Definition at line 528 of file structs.h.

The documentation for this struct was generated from the following file:

- structs.h

3.20 DoLoOp Struct Reference

```
#include <structs.h>
```

Data Fields

- PRELOAD p
- UBYTE * name
- UBYTE * **vars**
- UBYTE * **contents**
- UBYTE * dollarname
- LONG **startlinenumber**
- LONG **firstnum**
- LONG **lastnum**
- LONG **incnum**
- int **type**
- int **NoShowInput**
- int **errorsinloop**
- int **firstloopcall**
- WORD **firstdollar**
- WORD **lastdollar**
- WORD **incdollar**
- WORD **NumPreTypes**
- WORD **PreIfLevel**
- WORD **PreSwitchLevel**

3.20.1 Detailed Description

Each preprocessor do loop has a struct of type DOLOOP to keep track of all relevant parameters like where the beginning of the loop is, what the boundaries, increment and value of the loop parameter are, etc. Also we keep the whole loop inside a buffer of type PRELOAD

Definition at line 848 of file structs.h.

3.20.2 Field Documentation

3.20.2.1 p

```
PRELOAD DoLoOp::p
```

size, name and buffer

Definition at line 849 of file structs.h.

3.20.2.2 name

```
UBYTE* DoLoOp::name
```

pointer into PRELOAD buffer

Definition at line 850 of file structs.h.

3.20.2.3 dollarname

```
UBYTE* DoLoOp::dollarname
```

For loop over terms in expression. Allocated with Malloc1()

Definition at line 853 of file structs.h.

The documentation for this struct was generated from the following file:

- structs.h

3.21 DuBiOuS Struct Reference

Data Fields

- LONG **name**
- WORD **node**
- WORD **dummy**

3.21.1 Detailed Description

Definition at line 513 of file structs.h.

The documentation for this struct was generated from the following file:

- structs.h

3.22 EEdge Class Reference

Data Fields

- int **nodes** [2]
- int **ext**
- int **momn**
- char **momc** [2]
- char **padding** [6]

3.22.1 Detailed Description

Definition at line 24 of file gentopo.h.

The documentation for this class was generated from the following file:

- gentopo.h

3.23 EGraph Class Reference

Public Member Functions

- **EGraph** (int nnodes, int nedges, int mxdeg)
- void **print** (void)
- void **init** (int pid, long gid, int ∗∗adjmat, int sopi, BigInt nsym, BigInt esym)
- void **setExtern** (int nd, int val)
- void **endSetExtern** (void)

Data Fields

- long **gId**
- int **pId**
- int **nNodes**
- int **nEdges**
- int **maxdeg**
- int **nExtern**
- int **opi**
- BigInt **nsym**
- BigInt **esym**
- ENode ∗ **nodes**
- EEdge ∗ **edges**

3.23.1 Detailed Description

Definition at line 34 of file gentopo.h.

The documentation for this class was generated from the following files:

- gentopo.h
- gentopo.cc

3.24 ENode Class Reference

Data Fields

- int **deg**
- int **ext**
- int ∗ **edges**

3.24.1 Detailed Description

Definition at line 17 of file gentopo.h.

The documentation for this class was generated from the following file:

- gentopo.h

3.25 ExPrEsSiOn Struct Reference

Public Member Functions

- **PADPOSITION** (5, 2, 0, 12, 0)

Data Fields

- POSITION **onfile**
- POSITION **prototype**
- POSITION **size**
- RENUMBER **renum**
- BRACKETINFO * **bracketinfo**
- BRACKETINFO * **newbracketinfo**
- WORD * renumlists
- WORD * **inmem**
- LONG **counter**
- LONG **name**
- WORD **hidelevel**
- WORD **vflags**
- WORD **printflag**
- WORD **status**
- WORD **replace**
- WORD **node**
- WORD **whichbuffer**
- WORD **namesize**
- WORD **compression**
- WORD **numdummies**
- WORD **numfactors**
- WORD **sizeprototype**

3.25.1 Detailed Description

Definition at line 390 of file structs.h.

3.25.2 Field Documentation

3.25.2.1 renumlists

```
WORD* ExPrEsSiOn::renumlists
```

Allocated only for threaded version if variables exist, else points to AN.dummyrenumlist

Definition at line 397 of file structs.h.

The documentation for this struct was generated from the following file:

- structs.h

3.26 FaCdOlLaR Struct Reference

Data Fields

- WORD * **where**
- LONG **size**
- WORD **type**
- WORD **value**

3.26.1 Detailed Description

Definition at line 520 of file structs.h.

The documentation for this struct was generated from the following file:

- structs.h

3.27 factorized_poly Class Reference

Public Member Functions

- void **add_factor** (const poly &f, int p=1)
- const std::string **tostring** () const

Data Fields

- std::vector< poly > **factor**
- std::vector< int > **power**

Friends

- std::ostream & **operator**<< (std::ostream &out, const poly &p)

3.27.1 Detailed Description

Definition at line 54 of file polyfact.h.

The documentation for this class was generated from the following files:

- polyfact.h
- polyfact.cc

3.28 FiLe Struct Reference

```
#include <structs.h>
```

Public Member Functions

- **PADPOSITION** (5, 3, 2, 0, 0)

Data Fields

- POSITION **POposition**
- POSITION **filesize**
- WORD * **PObuffer**
- WORD * **POstop**
- WORD * **POfill**
- WORD * **POfull**
- char * **name**
- ULONG **numblocks**
- ULONG **inbuffer**
- LONG **POsize**
- int handle
- int **active**

3.28.1 Detailed Description

The type FILEHANDLE is the struct that controls all relevant information of a file, whether it is open or not. The file may even not yet exist. There is a system of caches (PObuffer) and as long as the information to be written still fits inside the cache the file may never be created. There are variables that can store information about different types of files, like scratch files or sort files. Depending on what is available in the system we may also have information about gzip compression (currently sort file only) or locks (TFORM).

Definition at line 633 of file structs.h.

3.28.2 Field Documentation

3.28.2.1 handle

```
int FiLe::handle
```

Our own handle. Equal -1 if no file exists.

Definition at line 661 of file structs.h.

The documentation for this struct was generated from the following file:

- structs.h

3.29 FiLeDaTa Struct Reference

Public Member Functions

- **PADPOSITION** (0, 0, 0, 2, 0)

Data Fields

- FILEINDEX **Index**
- POSITION **Fill**
- POSITION **Position**
- WORD **Handle**
- WORD **dirtyflag**

3.29.1 Detailed Description

Definition at line 150 of file structs.h.

The documentation for this struct was generated from the following file:

- structs.h

3.30 FiLeInDeX Struct Reference

```
#include <structs.h>
```

Data Fields

- POSITION next
- POSITION number
- INDEXENTRY expression [INFILEINDEX]
- SBYTE empty [EMPTYININDEX]

3.30.1 Detailed Description

Defines the structure of a file index in store-files and save-files.

It contains several entries (see struct InDeXeNtRy) up to a maximum of INFILEINDEX.

The variable number has been made of type POSITION to avoid padding problems with some types of computers/↩ OS and keep system independence of the .sav files.

This struct is always 512 bytes long.

Definition at line 137 of file structs.h.

3.30.2 Field Documentation

3.30.2.1 next

`POSITION FiLeInDeX::next`

Position of next FILEINDEX if any

Definition at line 138 of file structs.h.

3.30.2.2 number

`POSITION FiLeInDeX::number`

Number of used entries in this index

Definition at line 139 of file structs.h.

3.30.2.3 expression

`INDEXENTRY FiLeInDeX::expression[INFILEINDEX]`

File index entries

Definition at line 140 of file structs.h.

3.30.2.4 empty

`SBYTE FiLeInDeX::empty[EMPTYININDEX]`

Padding to 512 bytes

Definition at line 141 of file structs.h.

The documentation for this struct was generated from the following file:

- structs.h

3.31 FixedGlobals Struct Reference

`#include <structs.h>`

Data Fields

- WCN **Operation** [8]
- WCN2 **OperaFind** [6]
- char * **VarType** [10]
- char * **ExprStat** [21]
- char * **FunNam** [2]
- char * **swmes** [3]
- char * **fname**
- char * **fname2**
- UBYTE * **s_one**
- WORD **fnamebase**
- WORD **fname2base**
- UINT **cTable** [256]

3.31.1 Detailed Description

The FIXEDGLOBALS struct is an anachronism. It started as the struct with global variables that needed initialization. It contains the elements Operation and OperaFind which define a very early way of automatically jumping to the proper operation. We find the results of it in parts of the file opera.c Later operations were treated differently in a more transparent way. We never changed the existing code. The most important part is currently the cTable which is used intensively in the compiler.

Definition at line 2509 of file structs.h.

The documentation for this struct was generated from the following file:

- structs.h

3.32 fixedset Struct Reference

Data Fields

- char * **name**
- char * **description**
- int **type**
- int **dimension**

3.32.1 Detailed Description

Definition at line 569 of file structs.h.

The documentation for this struct was generated from the following file:

- structs.h

3.33 FUN_INFO Struct Reference

`#include <structs.h>`

Data Fields

- WORD * **location**
- int **numargs**
- int **numfunnies**
- int **numwildcards**
- int **symmet**
- int **tensor**
- int **commute**

3.33.1 Detailed Description

The struct FUN_INFO is used for information about functions in the file smart.c which is supposed to intelligently look for patterns in complicated wildcard situations involving symmetric functions.

Definition at line 606 of file structs.h.

The documentation for this struct was generated from the following file:

- structs.h

3.34 FuNcTiOn Struct Reference

`#include <structs.h>`

Data Fields

- TABLES tabl
- LONG symminfo
- LONG name
- WORD commute
- WORD complex
- WORD number
- WORD flags
- WORD spec
- WORD symmetric
- WORD node
- WORD namesize
- WORD **dimension**
- WORD **maxnumargs**
- WORD **minnumargs**

3.34.1 Detailed Description

Contains all information about a function. Also used for tables. It is used in the LIST elements of #AC.

Definition at line 475 of file structs.h.

3.34.2 Field Documentation

3.34.2.1 tabl

```
TABLES FuNcTiOn::tabl
```

Used if redefined as table. != 0 if function is a table

Definition at line 476 of file structs.h.

3.34.2.2 symminfo

```
LONG FuNcTiOn::symminfo
```

Info regarding symm properties offset in buffer

Definition at line 477 of file structs.h.

3.34.2.3 name

```
LONG FuNcTiOn::name
```

Location in namebuffer of NAMETREE

Definition at line 478 of file structs.h.

3.34.2.4 commute

```
WORD FuNcTiOn::commute
```

Commutation properties

Definition at line 479 of file structs.h.

3.34.2.5 complex

`WORD FuNcTiOn::complex`

Properties under complex conjugation

Definition at line 480 of file structs.h.

3.34.2.6 number

`WORD FuNcTiOn::number`

Number when stored in file

Definition at line 481 of file structs.h.

3.34.2.7 flags

`WORD FuNcTiOn::flags`

Used to indicate usage when storing

Definition at line 482 of file structs.h.

3.34.2.8 spec

`WORD FuNcTiOn::spec`

Regular, Tensor, etc. See FunSpecs.

Definition at line 483 of file structs.h.

3.34.2.9 symmetric

`WORD FuNcTiOn::symmetric`

0 if symmetric properties

Definition at line 484 of file structs.h.

3.34.2.10 node

```
WORD FuNcTiOn::node
```

Location in namenode of NAMETREE

Definition at line 485 of file structs.h.

3.34.2.11 namesize

```
WORD FuNcTiOn::namesize
```

Length of the name

Definition at line 486 of file structs.h.

The documentation for this struct was generated from the following file:

- structs.h

3.35 gcd_heuristic_failed Class Reference

3.35.1 Detailed Description

Definition at line 33 of file polygcd.h.

The documentation for this class was generated from the following file:

- polygcd.h

3.36 HANDLERS Struct Reference

```
#include <structs.h>
```

Data Fields

- WORD **newlogonly**
- WORD **newhandle**
- WORD **oldhandle**
- WORD **oldlogonly**
- WORD **oldprinttype**
- WORD **oldsilent**

3.36.1 Detailed Description

The struct HANDLERS is used in the communication with external channels.

Definition at line 915 of file structs.h.

The documentation for this struct was generated from the following file:

- structs.h

3.37 InDeX Struct Reference

Data Fields

- LONG **name**
- WORD **type**
- WORD **dimension**
- WORD **number**
- WORD **flags**
- WORD **nmin4**
- WORD **node**
- WORD **namesize**

3.37.1 Detailed Description

Definition at line 443 of file structs.h.

The documentation for this struct was generated from the following file:

- structs.h

3.38 indexblock Struct Reference

Data Fields

- MLONG **flags**
- MLONG **previousblock**
- MLONG **position**
- OBJECTS **objects** [NUMOBJECTS]

3.38.1 Detailed Description

Definition at line 107 of file minos.h.

The documentation for this struct was generated from the following file:

- minos.h

3.39 InDeXeNtRy Struct Reference

`#include <structs.h>`

Public Member Functions

- **PADPOSITION** (0, 1, 0, 5, MAXENAME+1)

Data Fields

- POSITION position
- POSITION length
- POSITION variables
- LONG CompressSize
- WORD nsymbols
- WORD nindices
- WORD nvectors
- WORD nfunctions
- WORD size
- SBYTE name [MAXENAME+1]

3.39.1 Detailed Description

Defines the structure of an entry in a file index (see struct FiLeInDeX).

It represents one expression in the file.

Definition at line 99 of file structs.h.

3.39.2 Field Documentation

3.39.2.1 position

`POSITION InDeXeNtRy::position`

Position of the expression itself

Definition at line 100 of file structs.h.

3.39.2.2 length

```
POSITION InDeXeNtRy::length
```

Length of the expression itself

Definition at line 101 of file structs.h.

3.39.2.3 variables

```
POSITION InDeXeNtRy::variables
```

Position of the list with variables

Definition at line 102 of file structs.h.

3.39.2.4 CompressSize

```
LONG InDeXeNtRy::CompressSize
```

Size of buffer before compress

Definition at line 103 of file structs.h.

3.39.2.5 nsymbols

```
WORD InDeXeNtRy::nsymbols
```

Number of symbols in the list

Definition at line 104 of file structs.h.

3.39.2.6 nindices

```
WORD InDeXeNtRy::nindices
```

Number of indices in the list

Definition at line 105 of file structs.h.

3.39.2.7 nvectors

```
WORD InDeXeNtRy::nvectors
```

Number of vectors in the list

Definition at line 106 of file structs.h.

3.39.2.8 nfunctions

```
WORD InDeXeNtRy::nfunctions
```

Number of functions in the list

Definition at line 107 of file structs.h.

3.39.2.9 size

```
WORD InDeXeNtRy::size
```

Size of variables field

Definition at line 108 of file structs.h.

3.39.2.10 name

```
SBYTE InDeXeNtRy::name[MAXENAME+1]
```

Name of expression

Definition at line 109 of file structs.h.

The documentation for this struct was generated from the following file:

- structs.h

3.40 iniinfo Struct Reference

Data Fields

- MLONG **entriesinindex**
- MLONG **numberofindexblocks**
- MLONG **firstindexblock**
- MLONG **lastindexblock**
- MLONG **numberoftables**
- MLONG **numberofnamesblocks**
- MLONG **firstnameblock**
- MLONG **lastnameblock**

3.40.1 Detailed Description

Definition at line 82 of file minos.h.

The documentation for this struct was generated from the following file:

- minos.h

3.41 INSIDEINFO Struct Reference

`#include <structs.h>`

Data Fields

- WORD * **buffer**
- int **oldcompiletype**
- int **oldparallelflag**
- int **oldnumpotmoddollars**
- WORD **size**
- WORD **numdollars**
- WORD **oldcbuf**
- WORD **oldrbuf**
- WORD **inscbuf**
- WORD **oldcnumlhs**

3.41.1 Detailed Description

Used for #inside

Definition at line 805 of file structs.h.

The documentation for this struct was generated from the following file:

- structs.h

3.42 KEYWORD Struct Reference

`#include <structs.h>`

Data Fields

- char * **name**
- TFUN **func**
- int **type**
- int **flags**

3.42.1 Detailed Description

The KEYWORD struct defines names of commands/statements and the routine to be called when they are encountered by the compiler or preprocessor.

Definition at line 221 of file structs.h.

The documentation for this struct was generated from the following file:

- structs.h

3.43 KEYWORDV Struct Reference

`#include <structs.h>`

Data Fields

- char * **name**
- int * **var**
- int **type**
- int **flags**

3.43.1 Detailed Description

The KEYWORDV struct defines names of commands/statements and the variable to be affected when they are encountered by the compiler or preprocessor.

Definition at line 233 of file structs.h.

The documentation for this struct was generated from the following file:

- structs.h

3.44 LIST Struct Reference

`#include <structs.h>`

Data Fields

- void * lijst
- char * message
- int num
- int maxnum
- int size
- int numglobal
- int numtemp
- int numclear

3.44.1 Detailed Description

Much information is stored in arrays of which we can double the size if the array proves to be too small. Such arrays are controled by a variable of type LIST. The routines that expand the lists are in the file tools.c

Definition at line 204 of file structs.h.

3.44.2 Field Documentation

3.44.2.1 lijst

```
void* LIST::lijst
```

[D] Holds space for "maxnum" elements of size "size" each

Definition at line 205 of file structs.h.

3.44.2.2 message

```
char* LIST::message
```

Text for Malloc1 when allocating lijst. Set to constant string.

Definition at line 206 of file structs.h.

3.44.2.3 num

```
int LIST::num
```

Number of elements in lijst.

Definition at line 207 of file structs.h.

3.44.2.4 maxnum

```
int LIST::maxnum
```

Maximum number of elements in lijst.

Definition at line 208 of file structs.h.

3.44.2.5 size

```
int LIST::size
```

Size of one element in lijst.

Definition at line 209 of file structs.h.

3.44.2.6 numglobal

```
int LIST::numglobal
```

Marker for position when .global is executed.

Definition at line 210 of file structs.h.

3.44.2.7 numtemp

```
int LIST::numtemp
```

At the moment only needed for sets and setstore.

Definition at line 211 of file structs.h.

3.44.2.8 numclear

```
int LIST::numclear
```

Only for the clear instruction.

Definition at line 212 of file structs.h.

The documentation for this struct was generated from the following file:

- structs.h

3.45 longMultiStruct Struct Reference

Data Fields

- UBYTE * **buffer**
- int **bufpos**
- int **packpos**
- int **nPacks**
- int **lastLen**
- struct longMultiStruct * **next**

3.45.1 Detailed Description

Definition at line 1013 of file mpi.c.

The documentation for this struct was generated from the following file:

- mpi.c

3.46 M_const Struct Reference

`#include <structs.h>`

Public Member Functions

- **PADPOSITION** (17, 24, 62, 83, 0)

Data Fields

- POSITION **zeropos**
- SORTING * S0
- UWORD * gcmod
- UWORD * gpowmod
- UBYTE * **TempDir**
- UBYTE * **TempSortDir**
- UBYTE * **IncDir**
- UBYTE * **InputFileName**
- UBYTE * **LogFileName**
- UBYTE * **OutBuffer**
- UBYTE * **Path**
- UBYTE * **SetupDir**
- UBYTE * **SetupFile**
- UBYTE * **gFortran90Kind**
- UBYTE * **gextrasym**
- UBYTE * **ggextrasym**
- UBYTE * **oldnumextrasymbols**
- SPECTATOR * **SpectatorFiles**
- LONG **MaxTer**
- LONG **CompressSize**
- LONG **ScratSize**
- LONG **HideSize**
- LONG **SizeStoreCache**
- LONG **MaxStreamSize**
- LONG **SIOsize**
- LONG **SLargeSize**
- LONG **SSmallEsize**
- LONG **SSmallSize**
- LONG **STermsInSmall**
- LONG **MaxBracketBufferSize**
- LONG **hProcessBucketSize**
- LONG **gProcessBucketSize**

- LONG **shmWinSize**
- LONG **OldChildTime**
- LONG **OldSecTime**
- LONG **OldMilliTime**
- LONG **WorkSize**
- LONG **gThreadBucketSize**
- LONG **ggThreadBucketSize**
- LONG **SumTime**
- LONG **SpectatorSize**
- LONG **TimeLimit**
- int **FileOnlyFlag**
- int **Interact**
- int **MaxParLevel**
- int **OutBufSize**
- int **SMaxFpatches**
- int **SMaxPatches**
- int **StdOut**
- int **ginsidefirst**
- int **gDefDim**
- int **gDefDim4**
- int **NumFixedSets**
- int **NumFixedFunctions**
- int **rbufnum**
- int **dbufnum**
- int **sbufnum**
- int **zbufnum**
- int **SkipClears**
- int **gTokensWriteFlag**
- int **gfunpowers**
- int **gStatsFlag**
- int **gNamesFlag**
- int **gCodesFlag**
- int **gSortType**
- int **gproperorderflag**
- int **hparallelflag**
- int **gparallelflag**
- int **totalnumberofthreads**
- int **gSizeCommuteInSet**
- int **gThreadStats**
- int **ggThreadStats**
- int **gFinalStats**
- int **ggFinalStats**
- int **gThreadsFlag**
- int **ggThreadsFlag**
- int **gThreadBalancing**
- int **ggThreadBalancing**
- int **gThreadSortFileSynch**
- int **ggThreadSortFileSynch**
- int **gProcessStats**
- int **ggProcessStats**
- int **gOldParallelStats**
- int **ggOldParallelStats**
- int **maxFlevels**
- int **resetTimeOnClear**
- int **gcNumDollars**

- int **MultiRun**
- int **gNoSpacesInNumbers**
- int **ggNoSpacesInNumbers**
- int **gIsFortran90**
- int **PrintTotalSize**
- int **fbuffersize**
- int **gOldFactArgFlag**
- int **ggOldFactArgFlag**
- int **gnumextrasym**
- int **ggnumextrasym**
- int **NumSpectatorFiles**
- int **SizeForSpectatorFiles**
- int **gOldGCDflag**
- int **ggOldGCDflag**
- int **gWTimeStatsFlag**
- int **ggWTimeStatsFlag**
- int **jumpratio**
- WORD **MaxTal**
- WORD **IndDum**
- WORD **DumInd**
- WORD **WilInd**
- WORD **gncmod**
- WORD **gnpowmod**
- WORD **gmodmode**
- WORD **gUnitTrace**
- WORD **gOutputMode**
- WORD **gOutputSpaces**
- WORD **gOutNumberType**
- WORD **gCnumpows**
- WORD **gUniTrace** [4]
- WORD **MaxWildcards**
- WORD **mTraceDum**
- WORD **OffsetIndex**
- WORD **OffsetVector**
- WORD **RepMax**
- WORD **LogType**
- WORD **ggStatsFlag**
- WORD **gLineLength**
- WORD **qError**
- WORD **FortranCont**
- WORD **HoldFlag**
- WORD **Ordering** [15]
- WORD **silent**
- WORD **tracebackflag**
- WORD **expnum**
- WORD **denomnum**
- WORD **facnum**
- WORD **invfacnum**
- WORD **sumnum**
- WORD **sumpnum**
- WORD **OldOrderFlag**
- WORD **termfunnum**
- WORD **matchfunnum**
- WORD **countfunnum**
- WORD **gPolyFun**

- WORD **gPolyFunInv**
- WORD **gPolyFunType**
- WORD **gPolyFunExp**
- WORD **gPolyFunVar**
- WORD **gPolyFunPow**
- WORD **dollarzero**
- WORD **atstartup**
- WORD **exitflag**
- WORD **NumStoreCaches**
- WORD **gIndentSpace**
- WORD **ggIndentSpace**
- WORD gShortStatsMax
- WORD ggShortStatsMax
- WORD **gextrasymbols**
- WORD **ggextrasymbols**
- WORD **zerorhs**
- WORD **onerhs**
- WORD **havesortdir**
- WORD **vectorzero**
- WORD **ClearStore**
- WORD **BracketFactors** [8]

3.46.1 Detailed Description

The M_const struct is part of the global data and resides in the ALLGLOBALS struct #A under the name #M. We see it used with the macro #AM as in AM.S0. It contains global settings at startup or .clear.

Definition at line 1360 of file structs.h.

3.46.2 Field Documentation

3.46.2.1 S0

```
SORTING* M_const::S0
```

[D] The main sort buffer

Definition at line 1362 of file structs.h.

3.46.2.2 gcmod

```
UWORD* M_const::gcmod
```

Global setting of modulus. Uses AC.cmod's memory

Definition at line 1363 of file structs.h.

3.46.2.3 gpowmod

```
UWORD* M_const::gpowmod
```

Global setting printing as powers. Uses AC.cmod's memory

Definition at line 1364 of file structs.h.

3.46.2.4 gShortStatsMax

```
WORD M_const::gShortStatsMax
```

For On FewerStatistics 10;

Definition at line 1521 of file structs.h.

3.46.2.5 ggShortStatsMax

```
WORD M_const::ggShortStatsMax
```

For On FewerStatistics 10;

Definition at line 1522 of file structs.h.

The documentation for this struct was generated from the following file:

- structs.h

3.47 MGraph Class Reference

Public Member Functions

- **MGraph** (int pid, int ncl, int ∗cldeg, int ∗clnum, int ∗clext, int sopi)
- long **generate** (void)

Data Fields

- int **pld**
- int **nNodes**
- int **nEdges**
- int **nLoops**
- MNode ∗∗ **nodes**
- int ∗ **clist**
- int **nClasses**
- int **mindeg**
- int **maxdeg**
- int **selOPl**
- long **ndiag**
- long **n1Pl**
- int **nBridges**
- int **c1Pl**
- int ∗∗ **adjMat**
- BigInt **nsym**
- BigInt **esym**
- Fraction **wsum**
- Fraction **wsopi**
- MNodeClass ∗ **curcl**
- EGraph ∗ **egraph**
- long **ngen**
- long **ngconn**
- long **nCallRefine**
- long **discardOrd**
- long **discardRefine**
- long **discardDisc**
- long **discardIso**

3.47.1 Detailed Description

Definition at line 84 of file gentopo.h.

The documentation for this class was generated from the following files:

- gentopo.h
- gentopo.cc

3.48 MiNmAx Struct Reference

Data Fields

- WORD mini
- WORD maxi
- WORD size

3.48.1 Detailed Description

Definition at line 306 of file structs.h.

3.48.2 Field Documentation

3.48.2.1 mini

`WORD MiNmAx::mini`

Minimum value

Definition at line 307 of file structs.h.

3.48.2.2 maxi

`WORD MiNmAx::maxi`

Maximum value

Definition at line 308 of file structs.h.

3.48.2.3 size

`WORD MiNmAx::size`

Value of one unit in this position.

Definition at line 309 of file structs.h.

The documentation for this struct was generated from the following file:

- structs.h

3.49 MNode Class Reference

Public Member Functions

- **MNode** (int id, int deg, int ext, int clss)

Data Fields

- int **id**
- int **deg**
- int **clss**
- int **ext**
- int **freelg**
- int **visited**

3.49.1 Detailed Description

Definition at line 61 of file gentopo.h.

The documentation for this class was generated from the following files:

- gentopo.h
- gentopo.cc

3.50 MNodeClass Class Reference

Public Member Functions

- **MNodeClass** (int nnodes, int ncl)
- void **init** (int ∗cl, int mxdeg, int ∗∗adjmat)
- void **copy** (MNodeClass ∗mnc)
- int **clCmp** (int nd0, int nd1, int cn)
- void **printMat** (void)
- void **mkFlist** (void)
- void **mkNdCl** (void)
- void **mkClMat** (int ∗∗adjmat)
- void **incMat** (int nd, int td, int val)
- Bool **chkOrd** (int nd, int ndc, MNodeClass ∗cl, int ∗dtcl)
- int **cmpArray** (int ∗a0, int ∗a1, int ma)

Data Fields

- int **nNodes**
- int **nClasses**
- int ∗ **clist**
- int ∗ **ndcl**
- int ∗∗ **clmat**
- int ∗ **flist**
- int **maxdeg**
- int **forallignment**

3.50.1 Detailed Description

Definition at line 248 of file gentopo.cc.

The documentation for this class was generated from the following file:

- gentopo.cc

3.51 MODNUM Struct Reference

Data Fields

- UWORD * **a**
- UWORD * **m**
- WORD **na**
- WORD **nm**

3.51.1 Detailed Description

Definition at line 1248 of file structs.h.

The documentation for this struct was generated from the following file:

- structs.h

3.52 MoDoPtDoLIArS Struct Reference

Data Fields

- WORD **number**
- WORD **type**

3.52.1 Detailed Description

Definition at line 554 of file structs.h.

The documentation for this struct was generated from the following file:

- structs.h

3.53 monomial_larger Class Reference

Public Member Functions

- bool **operator()** (const WORD *a, const WORD *b)

3.53.1 Detailed Description

Definition at line 169 of file poly.h.

The documentation for this class was generated from the following file:

- poly.h

3.54 N_const Struct Reference

`#include <structs.h>`

Public Member Functions

- **PADPOSITION** (50, 7, 23, 26, sizeof(SHvariables))

Data Fields

- POSITION **OldPosIn**
- POSITION **OldPosOut**
- POSITION **theposition**
- WORD * **EndNest**
- WORD * **Frozen**
- WORD * **FullProto**
- WORD * **cTerm**
- int * **RepPoint**
- WORD * **WildValue**
- WORD * **WildStop**
- WORD * **argaddress**
- WORD * **RepFunList**
- WORD * **patstop**
- WORD * **terstop**
- WORD * **terstart**
- WORD * **terfirstcomm**
- WORD * **DumFound**
- WORD ** **DumPlace**
- WORD ** **DumFunPlace**
- WORD * **UsedSymbol**
- WORD * **UsedVector**
- WORD * **UsedIndex**
- WORD * **UsedFunction**
- WORD * **MaskPointer**
- WORD * **ForFindOnly**
- WORD * **findTerm**
- WORD * **findPattern**
- WORD * **dummyrenumlist**
- int * **funargs**
- WORD ** **funlocs**
- int * **funinds**
- UWORD * **NoScrat2**

- WORD * **ReplaceScrat**
- TRACES * **tracestack**
- WORD * **selecttermundo**
- WORD * **patternbuffer**
- WORD * **termbuffer**
- WORD ** **PoinScratch**
- WORD ** **FunScratch**
- WORD * **RenumScratch**
- FUN_INFO * **FunInfo**
- WORD ** **SplitScratch**
- WORD ** **SplitScratch1**
- SORTING ** **FunSorts**
- UWORD * **SoScratC**
- WORD * **listinprint**
- WORD * **currentTerm**
- WORD ** **arglist**
- int * **tlistbuf**
- WORD * **compressSpace**
- UWORD * **SHcombi**
- WORD * **poly_vars**
- UWORD * **cmod**
- SHvariables **SHvar**
- LONG **deferskipped**
- LONG **lnScratch**
- LONG **SplitScratchSize**
- LONG **lnScratch1**
- LONG **SplitScratchSize1**
- LONG **ninterms**
- LONG **SHcombisize**
- int **NumTotWildArgs**
- int **UseFindOnly**
- int **UsedOtherFind**
- int **ErrorInDollar**
- int **numfargs**
- int **numflocs**
- int **nargs**
- int **tohunt**
- int **numoffuns**
- int **funisize**
- int **RSsize**
- int **numtracesctack**
- int **intracestack**
- int **numfuninfo**
- int **NumFunSorts**
- int **MaxFunSorts**
- int **arglistsize**
- int **tlistsize**
- int **filenum**
- int **compressSize**
- int **polysortflag**
- int **nogroundlevel**
- int **subsubveto**
- WORD **MaxRenumScratch**
- WORD **oldtype**
- WORD **oldvalue**

- WORD **NumWild**
- WORD **RepFunNum**
- WORD **DisOrderFlag**
- WORD **WildDirt**
- WORD **NumFound**
- WORD **WildReserve**
- WORD **TeInFun**
- WORD **TeSuOut**
- WORD **WildArgs**
- WORD **WildEat**
- WORD **PolyNormFlag**
- WORD **PolyFunTodo**
- WORD **sizeselecttermundo**
- WORD **patternbuffersize**
- WORD **numlistinprint**
- WORD **ncmod**
- WORD **ExpectedSign**
- WORD SignCheck
- WORD IndDum
- WORD **poly_num_vars**
- WORD **idfunctionflag**
- WORD **poly_vars_type**
- WORD **tryterm**

3.54.1 Detailed Description

The N_const struct is part of the global data and resides either in the ALLGLOBALS struct A, or the ALLPRIVATES struct B (TFORM) under the name N We see it used with the macro AN as in AN.RepFunNum It has variables that are private to each thread and are used as temporary storage during the expansion of the terms tree.

Definition at line 2133 of file structs.h.

3.54.2 Field Documentation

3.54.2.1 SignCheck

```
WORD N_const::SignCheck
```

Used in pattern matching of antisymmetric functions

Definition at line 2257 of file structs.h.

3.54.2.2 IndDum

```
WORD N_const::IndDum
```

Used in pattern matching of antisymmetric functions

Definition at line 2258 of file structs.h.

The documentation for this struct was generated from the following file:

- structs.h

3.55 nameblock Struct Reference

Data Fields

- MLONG **previousblock**
- MLONG **position**
- char **names** [NAMETABLESIZE]

3.55.1 Detailed Description

Definition at line 114 of file minos.h.

The documentation for this struct was generated from the following file:

- minos.h

3.56 NaMeNode Struct Reference

```
#include <structs.h>
```

Data Fields

- LONG name
- WORD parent
- WORD left
- WORD right
- WORD balance
- WORD type
- WORD number

3.56.1 Detailed Description

The names of variables are kept in an array. Elements of type NAMENODE define a tree (that is kept balanced) that make it easy and fast to look for variables. See also NAMETREE.

Definition at line 246 of file structs.h.

3.56.2 Field Documentation

3.56.2.1 name

```
LONG NaMeNode::name
```

Offset into NAMETREE::namebuffer.

Definition at line 247 of file structs.h.

3.56.2.2 parent

```
WORD NaMeNode::parent
```

=-1 if no parent.

Definition at line 248 of file structs.h.

3.56.2.3 left

```
WORD NaMeNode::left
```

=-1 if no child.

Definition at line 249 of file structs.h.

3.56.2.4 right

```
WORD NaMeNode::right
```

=-1 if no child.

Definition at line 250 of file structs.h.

3.56.2.5 balance

```
WORD NaMeNode::balance
```

Used for the balancing of the tree.

Definition at line 251 of file structs.h.

3.56.2.6 type

```
WORD NaMeNode::type
```

Type associated with the name. See compiler types.

Definition at line 252 of file structs.h.

3.56.2.7 number

```
WORD NaMeNode::number
```

Number of variable in LIST's like for example C_const::SymbolList.

Definition at line 253 of file structs.h.

The documentation for this struct was generated from the following file:

- structs.h

3.57 NaMeTree Struct Reference

```
#include <structs.h>
```

Data Fields

- NAMENODE * namenode
- UBYTE * namebuffer
- LONG nodesize
- LONG nodefill
- LONG namesize
- LONG namefill
- LONG oldnamefill
- LONG oldnodefill
- LONG globalnamefill
- LONG globalnodefill
- LONG clearnamefill
- LONG clearnodefill
- WORD headnode

3.57.1 Detailed Description

A struct of type NAMETREE controls a complete (balanced) tree of names for the compiler. The compiler maintains several of such trees and the system has been set up in such a way that one could define more of them if we ever want to work with local name spaces.

Definition at line 264 of file structs.h.

3.57.2 Field Documentation

3.57.2.1 namenode

```
NAMENODE* NaMeTree::namenode
```

[D] Vector of NAMENODE's. Number of elements is nodesize. =0 if no memory has been allocated.

Definition at line 265 of file structs.h.

3.57.2.2 namebuffer

```
UBYTE* NaMeTree::namebuffer
```

[D] Buffer that holds all the name strings refered to by the NAMENODE's. Allocation size is namesize. =0 if no memory has been allocated.

Definition at line 267 of file structs.h.

3.57.2.3 nodesize

```
LONG NaMeTree::nodesize
```

Maximum number of elements in namenode.

Definition at line 270 of file structs.h.

3.57.2.4 nodefill

```
LONG NaMeTree::nodefill
```

Number of currently used nodes in namenode.

Definition at line 271 of file structs.h.

3.57.2.5 namesize

```
LONG NaMeTree::namesize
```

Allocation size of namebuffer in bytes.

Definition at line 272 of file structs.h.

3.57.2.6 namefill

```
LONG NaMeTree::namefill
```

Number of bytes occupied.

Definition at line 273 of file structs.h.

3.57.2.7 oldnamefill

```
LONG NaMeTree::oldnamefill
```

UNUSED

Definition at line 274 of file structs.h.

3.57.2.8 oldnodefill

```
LONG NaMeTree::oldnodefill
```

UNUSED

Definition at line 275 of file structs.h.

3.57.2.9 globalnamefill

```
LONG NaMeTree::globalnamefill
```

Set by .global statement to the value of namefill. When a .store command is processed, this value will be used to reset namefill.

Definition at line 276 of file structs.h.

3.57.2.10 globalnodefill

```
LONG NaMeTree::globalnodefill
```

Same usage as globalnamefill, but for nodefill.

Definition at line 278 of file structs.h.

3.57.2.11 clearnamefill

```
LONG NaMeTree::clearnamefill
```

Marks the reset point used by the .clear statement.

Definition at line 279 of file structs.h.

3.57.2.12 clearnodefill

```
LONG NaMeTree::clearnodefill
```

Marks the reset point used by the .clear statement.

Definition at line 280 of file structs.h.

3.57.2.13 headnode

```
WORD NaMeTree::headnode
```

Offset in namenode of head node. =-1 if tree is empty.

Definition at line 281 of file structs.h.

The documentation for this struct was generated from the following file:

- structs.h

3.58 NeStInG Struct Reference

```
#include <structs.h>
```

Data Fields

- WORD * **termsize**
- WORD * **funsize**
- WORD * **argsize**

3.58.1 Detailed Description

The NESTING struct is used when we enter the argument of functions and there is the possibility that we have to change something there. Because functions can be nested we have to keep track of all levels of functions in case we have to move the outer layers to make room for a larger function argument.

Definition at line 999 of file structs.h.

The documentation for this struct was generated from the following file:

- structs.h

3.59 node Struct Reference

Public Member Functions

- **node** (const WORD *data)
- int **cmp** (const struct node *rhs) const
- bool **operator()** (const struct node *lhs, const struct node *rhs) const
- void **calcHash** ()

Data Fields

- const WORD * **data**
- struct node * **l**
- struct node * **r**
- WORD **sign**
- UWORD **hash**

3.59.1 Detailed Description

Definition at line 1449 of file optimize.cc.

The documentation for this struct was generated from the following file:

- optimize.cc

3.60 NoDe Struct Reference

Data Fields

- struct NoDe * **left**
- struct NoDe * **rght**
- int **lloser**
- int **rloser**
- int **lsrc**
- int **rsrc**

3.60.1 Detailed Description

A node for the tree of losers in the final sorting on the master.

Definition at line 265 of file parallel.c.

The documentation for this struct was generated from the following file:

- parallel.c

3.61 NodeEq Struct Reference

Public Member Functions

- bool **operator()** (const NODE *lhs, const NODE *rhs) const

3.61.1 Detailed Description

Definition at line 1511 of file optimize.cc.

The documentation for this struct was generated from the following file:

- optimize.cc

3.62 NodeHash Struct Reference

Public Member Functions

- size_t **operator()** (const NODE *n) const

3.62.1 Detailed Description

Definition at line 1505 of file optimize.cc.

The documentation for this struct was generated from the following file:

- optimize.cc

3.63 O_const Struct Reference

`#include <structs.h>`

Public Member Functions

- **PADPOSITION** (25, 4, 35, 17, 1)

Data Fields

- FILEDATA **SaveData**
- STOREHEADER **SaveHeader**
- OPTIMIZERESULT **OptimizeResult**
- UBYTE * **OutputLine**
- UBYTE * **OutStop**
- UBYTE * **OutFill**
- WORD * **bracket**
- WORD * **termbuf**
- WORD * **tabstring**
- UBYTE * **wpos**
- UBYTE * **wpoin**
- UBYTE * **DollarOutBuffer**
- UBYTE * **CurBufWrt**
- VOID(* **FlipWORD**)(UBYTE *)
- VOID(* **FlipLONG**)(UBYTE *)
- VOID(* **FlipPOS**)(UBYTE *)
- VOID(* **FlipPOINTER**)(UBYTE *)
- VOID(* **ResizeData**)(UBYTE *, int, UBYTE *, int)
- VOID(* **ResizeWORD**)(UBYTE *, UBYTE *)
- VOID(* **ResizeNCWORD**)(UBYTE *, UBYTE *)
- VOID(* **ResizeLONG**)(UBYTE *, UBYTE *)
- VOID(* **ResizePOS**)(UBYTE *, UBYTE *)
- VOID(* **ResizePOINTER**)(UBYTE *, UBYTE *)
- VOID(* **CheckPower**)(UBYTE *)
- VOID(* **RenumberVec**)(UBYTE *)
- DICTIONARY ** **Dictionaries**
- UBYTE * **tensorList**
- WORD * **inscheme**
- LONG **NumInBrack**
- LONG **wlen**
- LONG **DollarOutSizeBuffer**
- LONG **DollarInOutBuffer**

- OPTIMIZE **Optimize**
- int **OutInBuffer**
- int **NoSpacesInNumbers**
- int **BlockSpaces**
- int **CurrentDictionary**
- int **SizeDictionaries**
- int **NumDictionaries**
- int **CurDictNumbers**
- int **CurDictVariables**
- int **CurDictSpecials**
- int **CurDictFunWithArgs**
- int **CurDictNumberWarning**
- int **CurDictNotInFunctions**
- int **CurDictInDollars**
- int **gNumDictionaries**
- WORD **schemenum**
- WORD **transFlag**
- WORD **powerFlag**
- WORD **mpower**
- WORD **resizeFlag**
- WORD **bufferedInd**
- WORD **OutSkip**
- WORD **IsBracket**
- WORD **InFbrack**
- WORD **PrintType**
- WORD **FortFirst**
- WORD **DoubleFlag**
- WORD **IndentSpace**
- WORD **FactorMode**
- WORD **FactorNum**
- WORD **ErrorBlock**
- WORD **OptimizationLevel**
- UBYTE **FortDotChar**

3.63.1 Detailed Description

The O_const struct is part of the global data and resides in the ALLGLOBALS struct A under the name O We see it used with the macro AO as in AO.OutputLine It contains variables that involve the writing of text output and save/store files.

Definition at line 2310 of file structs.h.

The documentation for this struct was generated from the following file:

- structs.h

3.64 objects Struct Reference

Data Fields

- MLONG **position**
- MLONG **size**
- MLONG **date**
- MLONG **tablenumber**
- MLONG **uncompressed**
- MLONG **spare1**
- MLONG **spare2**
- MLONG **spare3**
- char **element** [ELEMENTSIZE]

3.64.1 Detailed Description

Definition at line 94 of file minos.h.

The documentation for this struct was generated from the following file:

- minos.h

3.65 optimization Class Reference

Public Member Functions

- bool **operator**< (const optimization &a) const

Data Fields

- int **type**
- int **arg1**
- int **arg2**
- int **improve**
- vector< WORD > **coeff**
- vector< int > **eqnidxs**

3.65.1 Detailed Description

class Optimization

3.65.2 Description

This object represents an optimization. Its type is a number in the range 0 to 5. Depending on this type, the variables arg1, arg2 and coeff indicate:

type==0 : optimization of the form x[arg1] ^ arg2 (coeff=empty) type==1 : optimization of the form x[arg1] * x[arg2] (coeff=empty) type==2 : optimization of the form x[arg1] * coeff (arg2=0) type==3 : optimization of the form x[arg1] + coeff (arg2=0) type==4 : optimization of the form x[arg1] + x[arg2] (coeff=empty) type==5 : optimization of the form x[arg1] - x[arg2] (coeff=empty)

Here, "x[arg]" represents a symbol (if positive) or an extrasymbol (if negative). The represented symbol's id is ABS(x[arg])-1.

"eqns" is a list of equation, where this optimization can be performed.

"improve" is the total improvement of this optimization.

Definition at line 2623 of file optimize.cc.

The documentation for this class was generated from the following file:

- optimize.cc

3.66 OPTIMIZE Struct Reference

Data Fields

- union {
  float **fval**
  int **ival** [2]
  } **mctsconstant**
- int **horner**
- int **hornerdirection**
- int **method**
- int **mctstimelimit**
- int **mctsnumexpand**
- int **mctsnumkeep**
- int **mctsnumrepeat**
- int **greedytimelimit**
- int **greedyminnum**
- int **greedymaxperc**
- int **printstats**
- int **debugflags**
- int **schemeflags**
- int **mctsdecaymode**
- int **salter**
- union {
  float **fval**
  int **ival** [2]
  } **saMaxT**
- union {
  float **fval**
  int **ival** [2]
  } **saMinT**

3.66.1 Detailed Description

Definition at line 1259 of file structs.h.

The documentation for this struct was generated from the following file:

- structs.h

3.67 OPTIMIZERESULT Struct Reference

Public Member Functions

- **PADPOSITION** (2, 1, 0, 3, 0)

Data Fields

- WORD * **code**
- UBYTE * **nameofexpr**
- LONG **codesize**
- WORD **exprnr**
- WORD **minvar**
- WORD **maxvar**

3.67.1 Detailed Description

Definition at line 1289 of file structs.h.

The documentation for this struct was generated from the following file:

- structs.h

3.68 P_const Struct Reference

```
#include <structs.h>
```

Data Fields

- LIST **DollarList**
- LIST **PreVarList**
- LIST **LoopList**
- LIST **ProcList**
- INSIDEINFO **inside**
- UBYTE ∗∗ **PreSwitchStrings**
- UBYTE ∗ **preStart**
- UBYTE ∗ **preStop**
- UBYTE ∗ **preFill**
- UBYTE ∗ **procedureExtension**
- UBYTE ∗ **cprocedureExtension**
- LONG ∗ **PreAssignStack**
- int ∗ **PreIfStack**
- int ∗ **PreSwitchModes**
- int ∗ **PreTypes**
- LONG **StopWatchZero**
- LONG **InOutBuf**
- LONG **pSize**
- int **PreAssignFlag**
- int **PreContinuation**
- int **PreproFlag**
- int **iBufError**
- int **PreOut**
- int **PreSwitchLevel**
- int **NumPreSwitchStrings**
- int **MaxPreTypes**
- int **NumPreTypes**
- int **MaxPreIfLevel**
- int **PreIfLevel**
- int **PreInsideLevel**
- int **DelayPrevar**
- int **AllowDelay**
- int **lhdollarerror**
- int **eat**
- int **gNumPre**
- int **PreDebug**
- int **OpenDictionary**
- int **PreAssignLevel**
- int **MaxPreAssignLevel**
- WORD **DebugFlag**
- WORD **preError**
- UBYTE **ComChar**
- UBYTE **cComChar**

3.68.1 Detailed Description

The P_const struct is part of the global data and resides in the ALLGLOBALS struct A under the name P We see it used with the macro AP as in AP.InOutBuf It contains objects that have dealings with the preprocessor.

Definition at line 1548 of file structs.h.

The documentation for this struct was generated from the following file:

- structs.h

3.69 ParallelVars Struct Reference

Public Member Functions

- **PADPOSITION** (2, 0, 8, 2, 0)

Data Fields

- FILEHANDLE **slavebuf**
- PF_BUFFER * **sbuf**
- PF_BUFFER ** **rbufs**
- int **me**
- int **numtasks**
- int **parallel**
- int **rhsInParallel**
- int **mkSlaveInfile**
- int **exprbufsize**
- int **exprtodo**
- int **log**
- WORD **numsbufs**
- WORD **numrbufs**

3.69.1 Detailed Description

Definition at line 168 of file parallel.h.

The documentation for this struct was generated from the following file:

- parallel.h

3.70 PaRtl Struct Reference

```
#include <structs.h>
```

Data Fields

- WORD * **psize**
- WORD * **args**
- WORD * **nargs**
- WORD * **nfun**
- WORD **numargs**
- WORD **numpart**
- WORD **where**

3.70.1 Detailed Description

The struct PARTI is used to help determining whether a partition_ function can be replaced.

Definition at line 1066 of file structs.h.

The documentation for this struct was generated from the following file:

- structs.h

3.71 PeRmUtE Struct Reference

`#include <structs.h>`

Data Fields

- WORD * **objects**
- WORD **sign**
- WORD **n**
- WORD **cycle** [MAXMATCH]

3.71.1 Detailed Description

The struct PERM is used to generate all permutations when the pattern matcher has to try to match (anti)symmetric functions.

Definition at line 1024 of file structs.h.

The documentation for this struct was generated from the following file:

- structs.h

3.72 PeRmUtEp Struct Reference

`#include <structs.h>`

Data Fields

- WORD ** **objects**
- WORD **sign**
- WORD **n**
- WORD **cycle** [MAXMATCH]

3.72.1 Detailed Description

Like struct PERM but works with pointers.

Definition at line 1036 of file structs.h.

The documentation for this struct was generated from the following file:

- structs.h

3.73 PF_BUFFER Struct Reference

`#include <parallel.h>`

Data Fields

- WORD ** **buff**
- WORD ** **fill**
- WORD ** **full**
- WORD ** **stop**
- MPI_Status * **status**
- MPI_Status * **retstat**
- MPI_Request * **request**
- MPI_Datatype * **type**
- int * **index**
- int * **tag**
- int * **from**
- int **numbufs**
- int **active**

3.73.1 Detailed Description

A struct for nonblocking, unbuffered send of the sorted terms in the PObuffers back to the master using several "rotating" PObuffers.

Definition at line 146 of file parallel.h.

The documentation for this struct was generated from the following file:

- parallel.h

3.74 poly Class Reference

Public Member Functions

- **poly** (PHEAD int, WORD=-1, WORD=1)
- **poly** (PHEAD const UWORD *, WORD, WORD=-1, WORD=1)
- **poly** (const poly &, WORD=-1, WORD=1)
- poly & **operator+=** (const poly &)
- poly & **operator-=** (const poly &)
- poly & **operator*=** (const poly &)
- poly & **operator/=** (const poly &)
- poly & **operator%=** (const poly &)
- const poly **operator+** (const poly &) const
- const poly **operator-** (const poly &) const
- const poly **operator*** (const poly &) const
- const poly **operator/** (const poly &) const
- const poly **operator%** (const poly &) const
- bool **operator==** (const poly &) const
- bool **operator!=** (const poly &) const
- poly & **operator=** (const poly &)
- WORD & **operator[ ]** (int)
- const WORD & **operator[ ]** (int) const
- void **termscopy** (const WORD *, int, int)
- void **check_memory** (int)
- void **expand_memory** (int)
- bool **is_zero** () const
- bool **is_one** () const
- bool **is_integer** () const
- bool **is_monomial** () const
- int is_dense_univariate () const
- int **sign** () const
- int **degree** (int) const
- int **total_degree** () const
- int **first_variable** () const
- int **number_of_terms** () const
- const std::vector< int > **all_variables** () const
- const poly **integer_lcoeff** () const
- const poly **lcoeff_univar** (int) const
- const poly **lcoeff_multivar** (int) const
- const poly **coefficient** (int, int) const
- const poly **derivative** (int) const
- void **setmod** (WORD, WORD=1)
- void **coefficients_modulo** (UWORD *, WORD, bool)
- int **size_of_form_notation** () const
- int **size_of_form_notation_with_den** (WORD) const
- const poly & **normalize** ()
- const std::string **to_string** () const
- WORD **last_monomial_index** () const
- WORD * **last_monomial** () const
- int **compare_degree_vector** (const poly &) const
- std::vector< int > **degree_vector** () const
- int **compare_degree_vector** (const std::vector< int > &) const

Static Public Member Functions

- static const poly **simple_poly** (PHEAD int, int=0, int=1, int=0, int=1)
- static const poly **simple_poly** (PHEAD int, const poly &, int=1, int=0, int=1)
- static void **get_variables** (PHEAD std::vector< WORD * >, bool, bool)
- static const poly **argument_to_poly** (PHEAD WORD *, bool, bool, poly *den=NULL)
- static void **poly_to_argument** (const poly &, WORD *, bool)
- static void **poly_to_argument_with_den** (const poly &, WORD, const UWORD *, WORD *, bool)
- static const poly **from_coefficient_list** (PHEAD const std::vector< WORD > &, int, WORD)
- static const std::vector< WORD > **to_coefficient_list** (const poly &)
- static const std::vector< WORD > **coefficient_list_divmod** (const std::vector< WORD > &, const std::vector< WORD > &, WORD, int)
- static int **monomial_compare** (PHEAD const WORD *, const WORD *)
- static void **add** (const poly &, const poly &, poly &)
- static void **sub** (const poly &, const poly &, poly &)
- static void mul (const poly &, const poly &, poly &)
- static void **div** (const poly &, const poly &, poly &)
- static void **mod** (const poly &, const poly &, poly &)
- static void divmod (const poly &, const poly &, poly &, poly &, bool only_divides)
- static bool **divides** (const poly &, const poly &)
- static void **mul_one_term** (const poly &, const poly &, poly &)
- static void **mul_univar** (const poly &, const poly &, poly &, int)
- static void mul_heap (const poly &, const poly &, poly &)
- static void **divmod_one_term** (const poly &, const poly &, poly &, poly &, bool)
- static void divmod_univar (const poly &, const poly &, poly &, poly &, int, bool)
- static void divmod_heap (const poly &, const poly &, poly &, poly &, bool, bool, bool &)
- static void **push_heap** (PHEAD WORD **, int)
- static void **pop_heap** (PHEAD WORD **, int)

Data Fields

- WORD * **terms**
- LONG **size_of_terms**
- WORD **modp**
- WORD **modn**

3.74.1 Detailed Description

Definition at line 49 of file poly.h.

3.74.2 Member Function Documentation

3.74.2.1 is_dense_univariate()

```
int poly::is_dense_univariate ( ) const
```

Dense univariate detection

3.74.3 Description

This method returns whether the polynomial is dense and univariate. The possible return values are:

-2 is not dense univariate -1 is no variables n>=0 is univariate in n

3.74.4 Notes

A univariate polynomial is considered dense iff more than half of the coefficients a_0...a_deg are non-zero.

Definition at line 2278 of file poly.cc.

Referenced by mul().

3.74.4.1 mul()

```
void poly::mul (
            const poly & a,
            const poly & b,
            poly & c )  [static]
```

Polynomial multiplication

3.74.5 Description

This routine determines which multiplication routine to use for multiplying two polynomials. The logic is as follows:

- If a or b consists of only one term, call mul_one_term;
- Otherwise, if both are univariate and dense, call mul_univar;
- Otherwise, call mul_heap.

Definition at line 1191 of file poly.cc.

References is_dense_univariate(), and mul_heap().

3.74.5.1 divmod()

```
void poly::divmod (
            const poly & a,
            const poly & b,
            poly & q,
            poly & r,
            bool only_divides )  [static]
```

Polynomial division

3.74.6 Description

This routine determines which division routine to use for dividing two polynomials. The logic is as follows:

- If b consists of only one term, call divmod_one_term;
- Otherwise, if both are univariate and dense, call divmod_univar;
- Otherwise, call divmod_heap.

Definition at line 1852 of file poly.cc.

3.74.6.1 mul_heap()

```
void poly::mul_heap (
            const poly & a,
            const poly & b,
            poly & c )  [static]
```

Multiplication of polynomials with a heap

3.74.7 Description

Multiplies two multivariate polynomials. The next element of the product is efficiently determined by using a heap. If the product of the maximum power in all variables is small, a hash table is used to add equal terms for extra speed.

A heap element h is formatted as follows:

- h[0] = index in a
- h[1] = index in b
- h[2] = hash code (-1 if no hash is used)
- h[3] = length of coefficient with sign
- h[4...4+AN.poly_num_vars-1] = powers
- h[4+AN.poly_num_vars...4+h[3]-1] = coefficient

Definition at line 957 of file poly.cc.

References RaisPowCached().

Referenced by mul().

3.74.7.1 divmod_univar()

```
void poly::divmod_univar (
             const poly & a,
             const poly & b,
             poly & q,
             poly & r,
             int var,
             bool only_divides )  [static]
```

Division of dense univariate polynomials.

3.74.8 Description

Divides two dense univariate polynomials. For each power, the method collects all terms that result in that power.

Relevant formula [Q=A/B, P=SUM(p_i∗x^i), n=deg(A), m=deg(B)]: q_k = [a_{m+k} - SUM(i=k+1...n-m, b_{m+k-i}∗q_i)] / b_m

Definition at line 1372 of file poly.cc.

References RaisPowCached().

3.74.8.1 divmod_heap()

```
void poly::divmod_heap (
             const poly & a,
             const poly & b,
             poly & q,
             poly & r,
             bool only_divides,
             bool check_div,
             bool & div_fail )  [static]
```

Division of polynomials with a heap

3.74.9 Description

Divides two multivariate polynomials. The next element of the quotient/remainder is efficiently determined by using a heap. If the product of the maximum power in all variables is small, a hash table is used to add equal terms for extra speed.

If the input flag check_div is set then if the result of any coefficient division results in a non-zero remainder (indicating that division has failed over the integers) the output flag div_fail will be set and the division will be terminated early (q, r will be incorrect). If check_div is not set then non-zero remainders from coefficient division will be written into r.

A heap element h is formatted as follows:

- h[0] = index in a

- h[1] = index in b
- h[2] = -1 (no hash is used)
- h[3] = length of coefficient with sign
- h[4...4+AN.poly_num_vars-1] = powers
- h[4+AN.poly_num_vars...4+h[3]-1] = coefficient

Note: the hashing trick as in multiplication cannot be used easily, since there is no tight upperbound on the exponents in the answer.

For details, see M. Monagan, "Polynomial Division using Dynamic Array, Heaps, and Packed Exponent Vectors"

Definition at line 1575 of file poly.cc.

The documentation for this class was generated from the following files:

- poly.h
- poly.cc

3.75 POLYMOD Struct Reference

`#include <structs.h>`

Data Fields

- WORD * **coefs**
- WORD **numsym**
- WORD **arraysize**
- WORD **polysize**
- WORD **modnum**

3.75.1 Detailed Description

The POLYMOD struct controls one univariate polynomial of which the coefficients have been taken modulus a (prime) number that fits inside a variable of type WORD. The polynomial is stored as an array of coefficients of size WORD.

Definition at line 1215 of file structs.h.

The documentation for this struct was generated from the following file:

- structs.h

3.76 PoSiTiOn Struct Reference

Data Fields

- off_t **p1**

3.76.1 Detailed Description

Definition at line 58 of file structs.h.

The documentation for this struct was generated from the following file:

- structs.h

3.77 PRELOAD Struct Reference

`#include <structs.h>`

Data Fields

- UBYTE * **buffer**
- LONG **size**

3.77.1 Detailed Description

Used by the preprocessor to load the contents of a doloop or a procedure. The struct PRELOAD is used both in the DOLOOP and PROCEDURE structs.

Definition at line 824 of file structs.h.

The documentation for this struct was generated from the following file:

- structs.h

3.78 pReVaR Struct Reference

`#include <structs.h>`

Data Fields

- UBYTE * name
- UBYTE * value
- UBYTE * argnames
- int nargs
- int wildarg

3.78.1 Detailed Description

An element of the type PREVAR is needed for each preprocessor variable.

Definition at line 792 of file structs.h.

3.78.2 Field Documentation

3.78.2.1 name

```
UBYTE* pReVaR::name
```

allocated

Definition at line 793 of file structs.h.

3.78.2.2 value

```
UBYTE* pReVaR::value
```

points into memory of name

Definition at line 794 of file structs.h.

3.78.2.3 argnames

```
UBYTE* pReVaR::argnames
```

names of arguments, zero separated. points into memory of name

Definition at line 795 of file structs.h.

3.78.2.4 nargs

```
int pReVaR::nargs
```

0 = regular, >= 1: number of macro arguments. total number

Definition at line 796 of file structs.h.

3.78.2.5 wildarg

```
int pReVaR::wildarg
```

The number of a potential ?var. If none: 0. wildarg<nargs

Definition at line 797 of file structs.h.

The documentation for this struct was generated from the following file:

- structs.h

3.79 PROCEDURE Struct Reference

```
#include <structs.h>
```

Data Fields

- PRELOAD **p**
- UBYTE * **name**
- int **loadmode**

3.79.1 Detailed Description

An element of the type PROCEDURE is needed for each procedure in the system.

Definition at line 834 of file structs.h.

The documentation for this struct was generated from the following file:

- structs.h

3.80 R_const Struct Reference

```
#include <structs.h>
```

Public Member Functions

- **PADPOSITION** (8, 7, 7, 24, 0)

Data Fields

- FILEDATA **StoreData**
- FILEHANDLE **Fscr** [3]
- FILEHANDLE **FoStage4** [2]
- POSITION **DefPosition**
- FILEHANDLE * **infile**
- FILEHANDLE * **outfile**
- FILEHANDLE * **hidefile**
- WORD * **CompressBuffer**
- WORD * **ComprTop**
- WORD * **CompressPointer**
- COMPARE **CompareRoutine**
- ULONG * **wranfia**
- LONG **OldTime**
- LONG **InInBuf**
- LONG **InHiBuf**
- LONG **pWorkSize**
- LONG **lWorkSize**
- LONG **posWorkSize**
- ULONG **wranfseed**
- int **NoCompress**
- int **gzipCompress**
- int **Cnumlhs**
- int **outtohide**
- int **wranfcall**
- int **wranfnpair1**
- int **wranfnpair2**
- WORD **GetFile**
- WORD **KeptInHold**
- WORD **BracketOn**
- WORD **MaxBracket**
- WORD **CurDum**
- WORD **DeferFlag**
- WORD **TePos**
- WORD **sLevel**
- WORD **Stage4Name**
- WORD **GetOneFile**
- WORD **PolyFun**
- WORD **PolyFunInv**
- WORD **PolyFunType**
- WORD **PolyFunExp**
- WORD **PolyFunVar**
- WORD **PolyFunPow**
- WORD **Eside**
- WORD **MaxDum**
- WORD **level**
- WORD **expchanged**
- WORD **expflags**
- WORD **CurExpr**
- WORD **SortType**
- WORD **ShortSortCount**

3.80.1 Detailed Description

The R_const struct is part of the global data and resides either in the ALLGLOBALS struct A, or the ALLPRIVATES struct B (TFORM) under the name R We see it used with the macro AR as in AR.infile It has the variables that define the running environment and that should be transferred with a term in a multithreaded run.

Definition at line 1914 of file structs.h.

The documentation for this struct was generated from the following file:

- structs.h

3.81 ReNuMbEr Struct Reference

`#include <structs.h>`

Public Member Functions

- **PADPOSITION** (4, 0, 0, 0, sizeof(VARRENUM) ∗4)

Data Fields

- POSITION **startposition**
- VARRENUM symb
- VARRENUM indi
- VARRENUM vect
- VARRENUM func
- WORD ∗ symnum
- WORD ∗ indnum
- WORD ∗ vecnum
- WORD ∗ funnum

3.81.1 Detailed Description

Only symb.lo gets dynamically allocated. All other pointers points into this memory.

Definition at line 177 of file structs.h.

3.81.2 Field Documentation

3.81.2.1 symb

```
VARRENUM ReNuMbEr::symb
```

Symbols

Definition at line 180 of file structs.h.

3.81.2.2 indi

```
VARRENUM ReNuMbEr::indi
```

Indices

Definition at line 181 of file structs.h.

3.81.2.3 vect

```
VARRENUM ReNuMbEr::vect
```

Vectors

Definition at line 182 of file structs.h.

3.81.2.4 func

```
VARRENUM ReNuMbEr::func
```

Functions

Definition at line 183 of file structs.h.

3.81.2.5 symnum

```
WORD* ReNuMbEr::symnum
```

Renumbered symbols

Definition at line 185 of file structs.h.

3.81.2.6 indnum

```
WORD* ReNuMbEr::indnum
```

Renumbered indices

Definition at line 186 of file structs.h.

3.81.2.7 vecnum

```
WORD* ReNuMbEr::vecnum
```

Renumbered vectors

Definition at line 187 of file structs.h.

3.81.2.8 funnum

```
WORD* ReNuMbEr::funnum
```

Renumbered functions

Definition at line 188 of file structs.h.

The documentation for this struct was generated from the following file:

- structs.h

3.82 S_const Struct Reference

```
#include <structs.h>
```

Public Member Functions

- **PADPOSITION** (3, 0, 4, 2, 0)

Data Fields

- POSITION **MaxExprSize**
- POSITION * **OldOnFile**
- WORD * **OldNumFactors**
- WORD * **Oldvflags**
- int **NumOldOnFile**
- int **NumOldNumFactors**
- int **MultiThreaded**
- int **Balancing**
- WORD **ExecMode**
- WORD **CollectOverFlag**

3.82.1 Detailed Description

The S_const struct is part of the global data and resides in the ALLGLOBALS struct A under the name S We see it used with the macro AS as in AS.ExecMode It has some variables used by the master in multithreaded runs

Definition at line 1866 of file structs.h.

The documentation for this struct was generated from the following file:

- structs.h

3.83 SeTs Struct Reference

Data Fields

- LONG **name**
- WORD **type**
- WORD **first**
- WORD **last**
- WORD **node**
- WORD **namesize**
- WORD **dimension**
- WORD **flags**

3.83.1 Detailed Description

Definition at line 497 of file structs.h.

The documentation for this struct was generated from the following file:

- structs.h

3.84 SETUPPARAMETERS Struct Reference

`#include <structs.h>`

Data Fields

- UBYTE * **parameter**
- int **type**
- int **flags**
- LONG **value**

3.84.1 Detailed Description

Each setup parameter has one element of the struct SETUPPARAMETERS assigned to it. By binary search in the array of them we can then locate the proper element by name. We have to assume that two ints make a long and either one or two longs make a pointer. The long before the ints and padding gives a problem in the initialization.

Definition at line 984 of file structs.h.

The documentation for this struct was generated from the following file:

- structs.h

3.85 SHvariables Struct Reference

Data Fields

- WORD * **outterm**
- WORD * **outfun**
- WORD * **incoef**
- WORD * **stop1**
- WORD * **stop2**
- WORD * **ststop1**
- WORD * **ststop2**
- FINISHUFFLE **finishuf**
- DO_UFFLE **do_uffle**
- LONG **combilast**
- WORD **nincoef**
- WORD **level**
- WORD **thefunction**
- WORD **option**

3.85.1 Detailed Description

Definition at line 1223 of file structs.h.

The documentation for this struct was generated from the following file:

- structs.h

3.86 sOrT Struct Reference

```
#include <structs.h>
```

Public Member Functions

- **PADPOSITION** (25, 12, 12, 3, 0)

Data Fields

- FILEHANDLE **file**
- POSITION **SizeInFile** [3]
- WORD * **lBuffer**
- WORD * **lTop**
- WORD * **lFill**
- WORD * **used**
- WORD * **sBuffer**
- WORD * **sTop**
- WORD * **sTop2**
- WORD * **sHalf**
- WORD * **sFill**
- WORD ** **sPointer**
- WORD ** **PoinFill**
- WORD ** **SplitScratch**
- WORD * **cBuffer**
- WORD ** **Patches**
- WORD ** **pStop**
- WORD ** **poina**
- WORD ** **poin2a**
- WORD * **ktoi**
- WORD * **tree**
- POSITION * **fPatches**
- POSITION * **inPatches**
- POSITION * **fPatchesStop**
- POSITION * **iPatches**
- FILEHANDLE * **f**
- FILEHANDLE ** **ff**
- LONG **sTerms**
- LONG **LargeSize**
- LONG **SmallSize**
- LONG **SmallEsize**
- LONG **TermsInSmall**
- LONG **Terms2InSmall**
- LONG **GenTerms**
- LONG **TermsLeft**
- LONG **GenSpace**
- LONG **SpaceLeft**
- LONG **putinsize**
- LONG **ninterms**
- int **MaxPatches**
- int **MaxFpatches**
- int **type**
- int **lPatch**
- int **fPatchN1**
- int **PolyWise**
- int **PolyFlag**
- int **cBufferSize**
- int **maxtermsize**
- int **newmaxtermsize**
- int **outputmode**
- int **stagelevel**
- WORD **fPatchN**
- WORD **inNum**
- WORD **stage4**

3.86.1 Detailed Description

The struct SORTING is used to control a sort operation. It includes a small and a large buffer and arrays for keeping track of various stages of the (merge) sorts. Each sort level has its own struct and different levels can have different sizes for its arrays. Also different threads have their own set of SORTING structs.

Definition at line 1086 of file structs.h.

The documentation for this struct was generated from the following file:

- structs.h

3.87 SpecTatoR Struct Reference

Public Member Functions

- **PADPOSITION** (2, 0, 0, 2, 0)

Data Fields

- POSITION **position**
- POSITION **readpos**
- FILEHANDLE * **fh**
- char * **name**
- WORD **exprnumber**
- WORD **flags**

3.87.1 Detailed Description

Definition at line 716 of file structs.h.

The documentation for this struct was generated from the following file:

- structs.h

3.88 StOrEcAcHe Struct Reference

```
#include <structs.h>
```

Public Member Functions

- **PADPOSITION** (1, 0, 0, 2, 0)

Data Fields

- POSITION **position**
- POSITION **toppos**
- struct StOrEcAcHe * **next**
- WORD **buffer** [2]

3.88.1 Detailed Description

The struct of type STORECACHE is used by a caching system for reading terms from stored expressions. Each thread should have its own system of caches.

Definition at line 1011 of file structs.h.

The documentation for this struct was generated from the following file:

- structs.h

3.89 STOREHEADER Struct Reference

```
#include <structs.h>
```

Data Fields

- UBYTE headermark [8]
- UBYTE lenWORD
- UBYTE lenLONG
- UBYTE lenPOS
- UBYTE lenPOINTER
- UBYTE endianness [16]
- UBYTE sSym
- UBYTE sInd
- UBYTE sVec
- UBYTE sFun
- UBYTE maxpower [16]
- UBYTE wildoffset [16]
- UBYTE revision
- UBYTE reserved [512-8-4-16-4-16-16-1]

3.89.1 Detailed Description

Defines the structure of the file header for store-files and save-files.

The first 8 bytes serve as a unique mark to identity save-files that contain such a header. Older versions of FORM don't have this header and will write the POSITION of the next file index (struct FiLeInDeX) here, which is always different from this pattern.

It is always 512 bytes long.

Definition at line 74 of file structs.h.

3.89.2 Field Documentation

3.89.2.1 headermark

```
UBYTE STOREHEADER::headermark[8]
```

Pattern for header identification. Old versions of FORM have a maximum sizeof(POSITION) of 8

Definition at line 75 of file structs.h.

3.89.2.2 lenWORD

```
UBYTE STOREHEADER::lenWORD
```

Number of bytes for WORD

Definition at line 77 of file structs.h.

3.89.2.3 lenLONG

```
UBYTE STOREHEADER::lenLONG
```

Number of bytes for LONG

Definition at line 78 of file structs.h.

3.89.2.4 lenPOS

```
UBYTE STOREHEADER::lenPOS
```

Number of bytes for POSITION

Definition at line 79 of file structs.h.

3.89.2.5 lenPOINTER

```
UBYTE STOREHEADER::lenPOINTER
```

Number of bytes for void $*$

Definition at line 80 of file structs.h.

3.89.2.6 endianness

```
UBYTE STOREHEADER::endianness[16]
```

Used to determine endianness, sizeof(int) should be <= 16

Definition at line 81 of file structs.h.

3.89.2.7 sSym

```
UBYTE STOREHEADER::sSym
```

sizeof(struct SyMbOl)

Definition at line 82 of file structs.h.

3.89.2.8 sInd

```
UBYTE STOREHEADER::sInd
```

sizeof(struct InDeX)

Definition at line 83 of file structs.h.

3.89.2.9 sVec

```
UBYTE STOREHEADER::sVec
```

sizeof(struct VeCtOr)

Definition at line 84 of file structs.h.

3.89.2.10 sFun

```
UBYTE STOREHEADER::sFun
```

sizeof(struct FuNcTiOn)

Definition at line 85 of file structs.h.

3.89.2.11 maxpower

```
UBYTE STOREHEADER::maxpower[16]
```

Maximum power, see #MAXPOWER

Definition at line 86 of file structs.h.

3.89.2.12 wildoffset

```
UBYTE STOREHEADER::wildoffset[16]
```

#WILDOFFSET macro

Definition at line 87 of file structs.h.

3.89.2.13 revision

```
UBYTE STOREHEADER::revision
```

Revision number of save-file system

Definition at line 88 of file structs.h.

3.89.2.14 reserved

```
UBYTE STOREHEADER::reserved[512-8-4-16-4-16-16-1]
```

Padding to 512 bytes

Definition at line 89 of file structs.h.

The documentation for this struct was generated from the following file:

- structs.h

3.90 StreaM Struct Reference

```
#include <structs.h>
```

Public Member Functions

- **PADPOSITION** (6, 3, 9, 0, 4)

Data Fields

- off_t **fileposition**
- off_t **linenumber**
- off_t **prevline**
- UBYTE * buffer
- UBYTE * pointer
- UBYTE * top
- UBYTE * FoldName
- UBYTE * name
- UBYTE * pname
- LONG **buffersize**
- LONG **bufferposition**
- LONG **inbuffer**
- int **previous**
- int **handle**
- int **type**
- int **prevars**
- int **previousNoShowInput**
- int **eqnum**
- int **afterwards**
- int **olddelay**
- int **oldnoshowinput**
- UBYTE **isnextchar**
- UBYTE **nextchar** [2]
- UBYTE **reserved**

3.90.1 Detailed Description

Input is read from 'streams' which are represented by objects of type STREAM. A stream can be a file, a do-loop, a procedure, the string value of a preprocessor variable ..... When a new stream is opened we have to keep information about where to fall back in the parent stream to allow this to happen even in the middle of reading names etc as would be the case with a`i'b

Definition at line 687 of file structs.h.

3.90.2 Field Documentation

3.90.2.1 buffer

```
UBYTE* StreaM::buffer
```

[D] Size in buffersize

Definition at line 691 of file structs.h.

3.90.2.2 pointer

```
UBYTE* StreaM::pointer
```

pointer into buffer memory

Definition at line 692 of file structs.h.

3.90.2.3 top

```
UBYTE* StreaM::top
```

pointer into buffer memory

Definition at line 693 of file structs.h.

3.90.2.4 FoldName

```
UBYTE* StreaM::FoldName
```

[D]

Definition at line 694 of file structs.h.

3.90.2.5 name

```
UBYTE* StreaM::name
```

[D]

Definition at line 695 of file structs.h.

3.90.2.6 pname

```
UBYTE* StreaM::pname
```

for DOLLARSTREAM and PREVARSTREAM it points always to name, else it is undefined

Definition at line 696 of file structs.h.

The documentation for this struct was generated from the following file:

- structs.h

3.91 SuBbUf Struct Reference

Data Fields

- WORD **subexpnum**
- WORD **buffernum**

3.91.1 Detailed Description

Definition at line 239 of file compiler.c.

The documentation for this struct was generated from the following file:

- compiler.c

3.92 SWITCH Struct Reference

Data Fields

- SWITCHTABLE * **table**
- SWITCHTABLE **defaultcase**
- SWITCHTABLE **endswitch**
- WORD **typetable**
- WORD **maxcase**
- WORD **mincase**
- WORD **numcases**
- WORD **tablesize**
- WORD **caseoffset**
- WORD **iflevel**
- WORD **whilelevel**
- WORD **nestingsum**
- WORD **padding**

3.92.1 Detailed Description

Definition at line 1326 of file structs.h.

The documentation for this struct was generated from the following file:

- structs.h

3.93 SWITCHTABLE Struct Reference

Data Fields

- WORD **ncase**
- WORD **value**
- WORD **compbuffer**

3.93.1 Detailed Description

Definition at line 1320 of file structs.h.

The documentation for this struct was generated from the following file:

- structs.h

3.94 SyMbOl Struct Reference

Data Fields

- LONG **name**
- WORD **minpower**
- WORD **maxpower**
- WORD **complex**
- WORD **number**
- WORD **flags**
- WORD **node**
- WORD **namesize**
- WORD **dimension**

3.94.1 Detailed Description

Definition at line 426 of file structs.h.

The documentation for this struct was generated from the following file:

- structs.h

3.95 T_const Struct Reference

```
#include <structs.h>
```

Data Fields

- SORTING * **S0**
- SORTING * **SS**
- NESTING **Nest**
- NESTING **NestStop**
- NESTING **NestPoin**
- WORD * **BrackBuf**
- STORECACHE **StoreCache**
- STORECACHE **StoreCacheAlloc**
- WORD ** **pWorkSpace**
- LONG * **lWorkSpace**
- POSITION * **posWorkSpace**
- WORD * **WorkSpace**
- WORD * **WorkTop**
- WORD * **WorkPointer**
- int * **RepCount**
- int * **RepTop**
- WORD * **WildArgTaken**
- UWORD * **factorials**
- WORD * **small_power_n**
- UWORD ** **small_power**
- UWORD * **bernoullis**
- WORD * **primelist**
- LONG * **pfac**
- LONG * **pBer**
- WORD * **TMaddr**
- WORD * **WildMask**
- WORD * **previousEfactor**
- WORD ** **TermMemHeap**
- UWORD ** **NumberMemHeap**
- UWORD ** **CacheNumberMemHeap**
- BRACKETINFO * **bracketinfo**
- WORD ** **ListPoly**
- WORD * **ListSymbols**
- UWORD * **NumMem**
- WORD * **TopologiesTerm**
- WORD * **TopologiesStart**
- PARTI **partitions**
- LONG **sBer**
- LONG **pWorkPointer**
- LONG **lWorkPointer**
- LONG **posWorkPointer**
- LONG **lnNumMem**
- int **sfact**
- int **mfac**
- int **ebufnum**
- int **fbufnum**
- int **allbufnum**
- int **aebufnum**
- int **idallflag**
- int **idallnum**
- int **idallmaxnum**
- int **WildcardBufferSize**
- int **TermMemMax**

- int **TermMemTop**
- int **NumberMemMax**
- int **NumberMemTop**
- int **CacheNumberMemMax**
- int **CacheNumberMemTop**
- int **bracketindexflag**
- int **optimtimes**
- int **ListSymbolsSize**
- int **NumListSymbols**
- int **numpoly**
- int **LeaveNegative**
- int **TrimPower**
- WORD **small_power_maxx**
- WORD **small_power_maxn**
- WORD **dummysubexp** [SUBEXPSIZE+4]
- WORD **comsym** [8]
- WORD **comnum** [4]
- WORD **comfun** [FUNHEAD+4]
- WORD **comind** [7]
- WORD **MinVecArg** [7+ARGHEAD]
- WORD **FunArg** [4+ARGHEAD+FUNHEAD]
- WORD **locwildvalue** [SUBEXPSIZE]
- WORD **mulpat** [SUBEXPSIZE+5]
- WORD **proexp** [SUBEXPSIZE+5]
- WORD **TMout** [40]
- WORD **TMbuff**
- WORD **TMdolfac**
- WORD **nfac**
- WORD **nBer**
- WORD **mBer**
- WORD **PolyAct**
- WORD **RecFlag**
- WORD **inprimelist**
- WORD **sizeprimelist**
- WORD **fromindex**
- WORD **setinterntopo**
- WORD **setexterntopo**
- WORD **TopologiesLevel**
- WORD **TopologiesOptions** [2]

3.95.1 Detailed Description

The T_const struct is part of the global data and resides either in the ALLGLOBALS struct A, or the ALLPRIVATES struct B (TFORM) under the name T We see it used with the macro AT as in AT.WorkPointer It has variables that are private to each thread, most of which have to be defined at startup.

Definition at line 2002 of file structs.h.

The documentation for this struct was generated from the following file:

- structs.h

3.96 TaBlEbAsE Struct Reference

Public Member Functions

- **PADPOSITION** (3, 0, 1, 0, 0)

Data Fields

- POSITION **fillpoint**
- POSITION **current**
- UBYTE * **name**
- int * **tablenumbers**
- TABLEBASESUBINDEX * **subindex**
- int **numtables**

3.96.1 Detailed Description

Definition at line 590 of file structs.h.

The documentation for this struct was generated from the following file:

- structs.h

3.97 TaBlEbAsEsUbInDeX Struct Reference

Public Member Functions

- **PADPOSITION** (0, 1, 0, 0, 0)

Data Fields

- POSITION **where**
- LONG **size**

3.97.1 Detailed Description

Definition at line 580 of file structs.h.

The documentation for this struct was generated from the following file:

- structs.h

3.98 TaBlEs Struct Reference

```
#include <structs.h>
```

Data Fields

- WORD * tablepointers
- WORD * prototype
- WORD * pattern
- MINMAX * mm
- WORD * flags
- COMPTREE * boomlijst
- UBYTE * argtail
- struct TaBlEs * spare
- WORD * buffers
- LONG totind
- LONG reserved
- LONG defined
- LONG mdefined
- int prototypeSize
- int numind
- int bounds
- int strict
- int sparse
- int numtree
- int rootnum
- int MaxTreeSize
- WORD bufnum
- WORD bufferssize
- WORD buffersfill
- WORD tablenum
- WORD mode
- WORD **numdummies**

3.98.1 Detailed Description

buffers, mm, flags, and prototype are always dynamically allocated, tablepointers only if needed (=0 if unallocated), boomlijst and argtail only for sparse tables.

Allocation is done for both the normal and the stub instance (spare), except for prototype and argtail which share memory.

Definition at line 349 of file structs.h.

3.98.2 Field Documentation

3.98.2.1 tablepointers

```
WORD* TaBlEs::tablepointers
```

[D] Start in tablepointers table.

Definition at line 350 of file structs.h.

3.98.2.2 prototype

```
WORD* TaBlEs::prototype
```

[D] The wildcard prototyping for arguments

Definition at line 355 of file structs.h.

3.98.2.3 pattern

```
WORD* TaBlEs::pattern
```

The pattern with which to match the arguments

Definition at line 356 of file structs.h.

3.98.2.4 mm

```
MINMAX* TaBlEs::mm
```

[D] Array bounds, dimension by dimension. # elements = numind.

Definition at line 358 of file structs.h.

3.98.2.5 flags

```
WORD* TaBlEs::flags
```

[D] Is element in use ? etc. # elements = numind.

Definition at line 359 of file structs.h.

3.98.2.6 boomlijst

```
COMPTREE* TaBlEs::boomlijst
```

[D] Tree for searching in sparse tables

Definition at line 360 of file structs.h.

3.98.2.7 argtail

```
UBYTE* TaBlEs::argtail
```

[D] The arguments in characters. Starts for tablebase with parenthesis to indicate tail

Definition at line 361 of file structs.h.

3.98.2.8 spare

```
struct TaBlEs* TaBlEs::spare
```

[D] For tablebase. Alternatingly stubs and real

Definition at line 363 of file structs.h.

3.98.2.9 buffers

```
WORD* TaBlEs::buffers
```

[D] When we use more than one compiler buffer.

Definition at line 364 of file structs.h.

3.98.2.10 totind

```
LONG TaBlEs::totind
```

Total number requested

Definition at line 365 of file structs.h.

3.98.2.11 reserved

```
LONG TaBlEs::reserved
```

Total reservation in tablepointers for sparse

Definition at line 366 of file structs.h.

3.98.2.12 defined

```
LONG TaBlEs::defined
```

Number of table elements that are defined

Definition at line 367 of file structs.h.

3.98.2.13 mdefined

```
LONG TaBlEs::mdefined
```

Same as defined but after .global

Definition at line 368 of file structs.h.

3.98.2.14 prototypeSize

```
int TaBlEs::prototypeSize
```

Size of allocated memory for prototype in bytes.

Definition at line 369 of file structs.h.

3.98.2.15 numind

```
int TaBlEs::numind
```

Number of array indices

Definition at line 370 of file structs.h.

3.98.2.16 bounds

```
int TaBlEs::bounds
```

Array bounds check on/off.

Definition at line 371 of file structs.h.

3.98.2.17 strict

```
int TaBlEs::strict
```

>0: all must be defined. <0: undefined not substitute

Definition at line 372 of file structs.h.

3.98.2.18 sparse

```
int TaBlEs::sparse
```

0 --> sparse table

Definition at line 373 of file structs.h.

3.98.2.19 numtree

```
int TaBlEs::numtree
```

For the tree for sparse tables

Definition at line 374 of file structs.h.

3.98.2.20 rootnum

```
int TaBlEs::rootnum
```

For the tree for sparse tables

Definition at line 375 of file structs.h.

3.98.2.21 MaxTreeSize

```
int TaBlEs::MaxTreeSize
```

For the tree for sparse tables

Definition at line 376 of file structs.h.

3.98.2.22 bufnum

`WORD TaBlEs::bufnum`

Each table potentially its own buffer

Definition at line 377 of file structs.h.

3.98.2.23 bufferssize

`WORD TaBlEs::bufferssize`

When we use more than one compiler buffer

Definition at line 378 of file structs.h.

3.98.2.24 buffersfill

`WORD TaBlEs::buffersfill`

When we use more than one compiler buffer

Definition at line 379 of file structs.h.

3.98.2.25 tablenum

`WORD TaBlEs::tablenum`

For testing of tableuse

Definition at line 380 of file structs.h.

3.98.2.26 mode

`WORD TaBlEs::mode`

0: normal, 1: stub

Definition at line 381 of file structs.h.

The documentation for this struct was generated from the following file:

- structs.h

3.99 ToPoTyPe Struct Reference

Data Fields

- int **cldeg** [MAXPOINTS]
- int **clnum** [MAXPOINTS]
- int **clext** [MAXPOINTS]
- int **ncl**
- int **nloops**
- int **nlegs**
- int **npadding**
- WORD ∗ **vert**
- WORD ∗ **vertmax**
- WORD **nvert**
- WORD **sopi**

3.99.1 Detailed Description

Definition at line 46 of file topowrap.cc.

The documentation for this struct was generated from the following file:

- topowrap.cc

3.100 TrAcEn Struct Reference

`#include <structs.h>`

Data Fields

- WORD ∗ **accu**
- WORD ∗ **accup**
- WORD ∗ **termp**
- WORD ∗ **perm**
- WORD ∗ **inlist**
- WORD **sgn**
- WORD **num**
- WORD **level**
- WORD **factor**
- WORD **allsign**

3.100.1 Detailed Description

The struct TRACEN keeps track of the progress during the expansion of a 4-dimensional trace. Each time a term gets generated the expansion tree continues in the next statement. When it returns it has to know where to continue.

Definition at line 773 of file structs.h.

The documentation for this struct was generated from the following file:

- structs.h

3.101 TrAcEs Struct Reference

`#include <structs.h>`

Data Fields

- WORD * **accu**
- WORD * **accup**
- WORD * **termp**
- WORD * **perm**
- WORD * **inlist**
- WORD * **nt3**
- WORD * **nt4**
- WORD * **j3**
- WORD * **j4**
- WORD * **e3**
- WORD * **e4**
- WORD * **eers**
- WORD * **mepf**
- WORD * **mdel**
- WORD * **pepf**
- WORD * **pdel**
- WORD **sgn**
- WORD **stap**
- WORD **step1**
- WORD **kstep**
- WORD **mdum**
- WORD **gamm**
- WORD **ad**
- WORD **a3**
- WORD **a4**
- WORD **lc3**
- WORD **lc4**
- WORD **sign1**
- WORD **sign2**
- WORD **gamma5**
- WORD **num**
- WORD **level**
- WORD **factor**
- WORD **allsign**
- WORD **finalstep**

3.101.1 Detailed Description

The struct TRACES keeps track of the progress during the expansion of a 4-dimensional trace. Each time a term gets generated the expansion tree continues in the next statement. When it returns it has to know where to continue. The 4-dimensional traces are more complicated than the n-dimensional traces (see TRACEN) because of the extra tricks that can be used. They are responsible for the shorter final expressions.

Definition at line 740 of file structs.h.

The documentation for this struct was generated from the following file:

- structs.h

3.102 tree Struct Reference

`#include <structs.h>`

Data Fields

- int parent
- int left
- int right
- int value
- int blnce
- int usage

3.102.1 Detailed Description

The subexpressions in the compiler are kept track of in a (balanced) tree to reduce the need for subexpressions and hence save much space in large rhs expressions (like when we have xxxxxxx occurrences of objects like f(x+1,x+1) in which each x+1 becomes a subexpression. The struct that controls this tree is COMPTREE.

Definition at line 293 of file structs.h.

3.102.2 Field Documentation

3.102.2.1 parent

`int tree::parent`

Index of parent

Definition at line 294 of file structs.h.

3.102.2.2 left

`int tree::left`

Left child (if not -1)

Definition at line 295 of file structs.h.

3.102.2.3 right

```
int tree::right
```

Right child (if not -1)

Definition at line 296 of file structs.h.

3.102.2.4 value

```
int tree::value
```

The object to be sorted and searched

Definition at line 297 of file structs.h.

3.102.2.5 blnce

```
int tree::blnce
```

Balance factor

Definition at line 298 of file structs.h.

3.102.2.6 usage

```
int tree::usage
```

Number of uses in some types of trees

Definition at line 299 of file structs.h.

The documentation for this struct was generated from the following file:

- structs.h

3.103 tree_node Class Reference

Public Member Functions

- **tree_node** (int _var=0)

Data Fields

- vector< tree_node > **childs**
- double **sum_results**
- int **num_visits**
- WORD **var**
- bool **finished**

3.103.1 Detailed Description

Definition at line 121 of file optimize.cc.

The documentation for this class was generated from the following file:

- optimize.cc

3.104 VaRrEnUm Struct Reference

```
#include <structs.h>
```

Data Fields

- WORD * start
- WORD * lo
- WORD * hi

3.104.1 Detailed Description

Contains the pointers to an array in which a binary search will be performed.

Definition at line 165 of file structs.h.

3.104.2 Field Documentation

3.104.2.1 start

```
WORD* VaRrEnUm::start
```

Start point for search. Points inbetween lo and hi

Definition at line 166 of file structs.h.

3.104.2.2 lo

```
WORD* VaRrEnUm::lo
```

Start of memory area

Definition at line 167 of file structs.h.

3.104.2.3 hi

```
WORD* VaRrEnUm::hi
```

End of memory area

Definition at line 168 of file structs.h.

The documentation for this struct was generated from the following file:

- structs.h

3.105 VeCtOr Struct Reference

Data Fields

- LONG **name**
- WORD **complex**
- WORD **number**
- WORD **flags**
- WORD **node**
- WORD **namesize**
- WORD **dimension**

3.105.1 Detailed Description

Definition at line 459 of file structs.h.

The documentation for this struct was generated from the following file:

- structs.h

3.106 X_const Struct Reference

```
#include <structs.h>
```

Data Fields

- UBYTE * **currentPrompt**
- UBYTE * **shellname**
- UBYTE * **stderrname**
- int **timeout**
- int **killSignal**
- int **killWholeGroup**
- int **daemonize**
- int **currentExternalChannel**

3.106.1 Detailed Description

The X_const struct is part of the global data and resides in the ALLGLOBALS struct A under the name X We see it used with the macro AX as in AX.timeout It contains variables that involve communication with external programs

Definition at line 2401 of file structs.h.

The documentation for this struct was generated from the following file:

- structs.h

Chapter 4

File Documentation

4.1 argument.c File Reference

```
#include "form3.h"
```

Macros

- #define NEWORDER

Functions

- WORD **execarg** (PHEAD WORD ∗term, WORD level)
- WORD **execterm** (PHEAD WORD ∗term, WORD level)
- int **ArgumentImplode** (PHEAD WORD ∗term, WORD ∗thelist)
- int **ArgumentExplode** (PHEAD WORD ∗term, WORD ∗thelist)
- int **ArgFactorize** (PHEAD WORD ∗argin, WORD ∗argout)
- WORD FindArg (PHEAD WORD ∗a)
- WORD InsertArg (PHEAD WORD ∗argin, WORD ∗argout, int par)
- int CleanupArgCache (PHEAD WORD bufnum)
- int **ArgSymbolMerge** (WORD ∗t1, WORD ∗t2)
- int **ArgDotproductMerge** (WORD ∗t1, WORD ∗t2)
- WORD ∗ TakeArgContent (PHEAD WORD ∗argin, WORD ∗argout)
- WORD ∗ MakeInteger (PHEAD WORD ∗argin, WORD ∗argout, WORD ∗argfree)
- WORD ∗ MakeMod (PHEAD WORD ∗argin, WORD ∗argout, WORD ∗argfree)
- void SortWeights (LONG ∗weights, LONG ∗extraspace, WORD number)

4.1.1 Detailed Description

Contains the routines that deal with the execution phase of the argument and related statements (like term)

4.1.2 Macro Definition Documentation

4.1.2.1 NEWORDER

```
#define NEWORDER
```

Factorizes an argument in general notation (meaning that the first word of the argument is a positive size indicator) Input (argin): pointer to the complete argument Output (argout): Pointer to where the output should be written. This is in the WorkSpace Return value should be negative if anything goes wrong.

The notation of the output should be a string of arguments terminated by the number zero.

Originally we sorted in a way that the constants came last. This gave conflicts with the dollar and expression factorizations (in the expressions we wanted the zero first and then followed by the constants).

Definition at line 2022 of file argument.c.

4.1.3 Function Documentation

4.1.3.1 FindArg()

```
WORD FindArg (
            PHEAD WORD * a )
```

Looks the argument up in the (workers) table. If it is found the number in the table is returned (plus one to make it positive). If it is not found we look in the compiler provided table. If it is found - the number in the table is returned (minus one to make it negative). If in neither table we return zero.

Definition at line 2472 of file argument.c.

4.1.3.2 InsertArg()

```
WORD InsertArg (
            PHEAD WORD * argin,
            WORD * argout,
            int par )
```

Inserts the argument into the (workers) table. If the table is too full we eliminate half of it. The eliminated elements are the ones that have not been used most recently, weighted by their total use and age(?). If par == 0 it inserts in the regular factorization cache If par == 1 it inserts in the cache defined with the FactorCache statement

Definition at line 2496 of file argument.c.

4.1.3.3 CleanupArgCache()

```
int CleanupArgCache (
            PHEAD WORD bufnum )
```

Cleans up the argument factorization cache. We throw half the elements. For a weight of what we want to keep we use the product of usage and the number in the buffer.

Definition at line 2531 of file argument.c.

4.1.3.4 TakeArgContent()

```
WORD* TakeArgContent (
            PHEAD WORD * argin,
            WORD * argout )
```

Implements part of the old ExecArg in which we take common factors from arguments with more than one term. The common pieces are put in argout as a sequence of arguments. The part with the multiple terms that are now relative prime is put in argfree which is allocated via TermMalloc and is given as the return value. The difference with the old code is that negative powers are always removed. Hence it is as in MakeInteger in which only numerators will be left: now only zero or positive powers will be remaining.

Definition at line 2725 of file argument.c.

4.1.3.5 MakeInteger()

```
WORD* MakeInteger (
            PHEAD WORD * argin,
            WORD * argout,
            WORD * argfree )
```

For normalizing everything to integers we have to determine for all elements of this argument the LCM of the denominators and the GCD of the numerators. The input argument is in argin. The number that comes out should go to argout. The new pointer in the argout buffer is the return value. The normalized argument is in argfree.

Definition at line 3271 of file argument.c.

4.1.3.6 MakeMod()

```
WORD* MakeMod (
            PHEAD WORD * argin,
            WORD * argout,
            WORD * argfree )
```

Similar to MakeInteger but now with modulus arithmetic using only a one WORD 'prime'. We make the coefficient of the first term in the argument equal to one. Already the coefficients are taken modulus AN.cmod and AN.ncmod == 1

Definition at line 3442 of file argument.c.

4.1.3.7 SortWeights()

```
void SortWeights (
            LONG * weights,
            LONG * extraspace,
            WORD number )
```

Sorts an array of LONGS in the same way SplitMerge (in sort.c) works We use gradual division in two.

Definition at line 3487 of file argument.c.

4.2 bugtool.c File Reference

```
#include "form3.h"
```

Functions

- void **ExprStatus** (EXPRESSIONS e)

4.2.1 Detailed Description

Low level routines for debugging

4.3 checkpoint.c File Reference

```
#include "form3.h"
#include <errno.h>
```

Macros

- #define **CACHED_SNAPSHOT**
- #define **CACHE_SIZE** 4096
- #define **R_FREE**(ARG) if (ARG) M_free(ARG, #ARG);
- #define **R_FREE_NAMETREE**(ARG)
- #define **R_FREE_STREAM**(ARG)
- #define **R_SET**(VAR, TYPE) VAR = ∗((TYPE∗)p); p = (unsigned char∗)p + sizeof(TYPE);
- #define **R_COPY_B**(VAR, SIZE, CAST)
- #define **S_WRITE_B**(BUF, LEN) if (fwrite_cached(BUF, 1, LEN, fd) != (size_t)(LEN)) return(__LINE__);
- #define **S_FLUSH_B** if (flush_cache(fd) != 1) return(__LINE__);
- #define **R_COPY_S**(VAR, CAST)
- #define **S_WRITE_S**(STR)
- #define **R_COPY_LIST**(ARG)
- #define **S_WRITE_LIST**(LST)
- #define **R_COPY_NAMETREE**(ARG)
- #define **S_WRITE_NAMETREE**(ARG)
- #define **S_WRITE_DOLLAR**(ARG)
- #define **ANNOUNCE**(str)

Functions

- int CheckRecoveryFile ()
- void DeleteRecoveryFile ()
- char * RecoveryFilename ()
- void InitRecovery ()
- size_t **fwrite_cached** (const void *ptr, size_t size, size_t nmemb, FILE *fd)
- size_t **flush_cache** (FILE *fd)
- int DoRecovery (int *moduletype)
- void DoCheckpoint (int moduletype)

Variables

- unsigned char **cache_buffer** [CACHE_SIZE]
- size_t **cache_fill** = 0

4.3.1 Detailed Description

Contains all functions that deal with the recovery mechanism controlled and activated by the On Checkpoint switch.

The main function are DoCheckpoint, DoRecovery, and DoSnapshot. If the checkpoints are activated DoCheckpoint is called every time a module is finished executing. If the conditions for the creation of a recovery snapshot are met DoCheckpoint calls DoSnapshot. DoRecovery is called once when FORM starts up with the command line argument -R. Most of the other code contains debugging facilities that are only compiled if the macro PRINTDEBUG is defined.

The recovery mechanism is atomic, i.e. only if everything went well, the final recovery file is created (and the older one overwritten) in a single step (copying). If some errors occur, a warning is issued and the program continues without having created a new recovery file. The only situation in which the creation of the recovery data leads to a termination of the running program is if not enough disk or memory space is left.

For ParFORM each slave creates its own recovery file, sends it to the master and then it deletes the recovery file. The master stores all the recovery files and on recovery it feeds these files to the slaves. It is nearly impossible to recover after some MPI fault so ParFORM terminates on any recovery failure.

DoRecovery and DoSnapshot do the loading and saving of the recovery data, respectively. Every change in one functions needs to be accompanied by the appropriate change in the other function. The structure of both functions is quite similar. They handle the relevant global structs one after the other and then care about the copying of the hide and scratch files.

The names of the recovery, scratch and hide files are hard-coded in the variables in fold "filenames and system commands".

If the global structs AM,AP,AC,AR are changed, DoRecovery and DoSnapshot usually also have to be changed. Some structs are read/written as a whole (AP,AC), some are read/written only partly as a selection of their individual elements (AM,AR). If AM or AR have been changed by adding or removing an element that is important for the runtime status, then the reading/writing statements have to be added to or removed from DoRecovery and Do↩ Snapshot. If AP or AC are changed, then for non-pointer variables (in the case of a struct it also means that none of its elements is a pointer) nothing has to be changed in the functions here. If pointers are involved, extra code has to be added (or removed). See the comments of DoRecovery and DoSnapshot.

4.3.2 Macro Definition Documentation

4.3.2.1 R_FREE_NAMETREE

```
#define R_FREE_NAMETREE(
                ARG )
```

Value:

```
    R_FREE(ARG->namenode); \
    R_FREE(ARG->namebuffer); \
    R_FREE(ARG);
```

Definition at line 1281 of file checkpoint.c.

4.3.2.2 R_FREE_STREAM

```
#define R_FREE_STREAM(
                ARG )
```

Value:

```
    R_FREE(ARG.buffer); \
    R_FREE(ARG.FoldName); \
    R_FREE(ARG.name);
```

Definition at line 1286 of file checkpoint.c.

4.3.2.3 R_COPY_B

```
#define R_COPY_B(
                VAR,
                SIZE,
                CAST )
```

Value:

```
    VAR = (CAST)Malloc1(SIZE,#VAR); \
    memcpy(VAR, p, SIZE); p = (unsigned char*)p + SIZE;
```

Definition at line 1298 of file checkpoint.c.

4.3.2.4 R_COPY_S

```
#define R_COPY_S(
                VAR,
                CAST )
```

Value:

```
    if ( VAR ) { \
        VAR = (CAST)Malloc1(strlen(p)+1,"R_COPY_S"); \
        strcpy((char*)VAR, p); p = (unsigned char*)p + strlen(p) + 1; \
    }
```

Definition at line 1310 of file checkpoint.c.

4.3.2.5 S_WRITE_S

```
#define S_WRITE_S(
                STR )
```

Value:

```
    if ( STR ) { \
        l = strlen((char*)STR) + 1; \
        if ( fwrite_cached(STR, 1, l, fd) != (size_t)l ) return(__LINE__); \
    }
```

Definition at line 1316 of file checkpoint.c.

4.3.2.6 R_COPY_LIST

```
#define R_COPY_LIST(
                ARG )
```

Value:

```
    if ( ARG.maxnum ) { \
        R_COPY_B(ARG.lijst, ARG.size*ARG.maxnum, void*) \
    }
```

Definition at line 1324 of file checkpoint.c.

4.3.2.7 S_WRITE_LIST

```
#define S_WRITE_LIST(
                LST )
```

Value:

```
    if ( LST.maxnum ) { \
        S_WRITE_B((char*)LST.lijst, LST.maxnum*LST.size) \
    }
```

Definition at line 1329 of file checkpoint.c.

4.3.2.8 R_COPY_NAMETREE

```
#define R_COPY_NAMETREE(
                ARG )
```

Value:

```
    R_COPY_B(ARG, sizeof(NAMETREE), NAMETREE*); \
    if ( ARG->namenode ) { \
        R_COPY_B(ARG->namenode, ARG->nodesize*sizeof(NAMENODE), NAMENODE*); \
    } \
    if ( ARG->namebuffer ) { \
        R_COPY_B(ARG->namebuffer, ARG->namesize, UBYTE*); \
    }
```

Definition at line 1336 of file checkpoint.c.

4.3.2.9 S_WRITE_NAMETREE

```
#define S_WRITE_NAMETREE(
                ARG )
```

Value:

```
    S_WRITE_B(ARG, sizeof(NAMETREE)); \
    if ( ARG->namenode ) { \
        S_WRITE_B(ARG->namenode, ARG->nodesize*sizeof(struct NaMeNode)); \
    } \
    if ( ARG->namebuffer ) { \
        S_WRITE_B(ARG->namebuffer, ARG->namesize); \
    }
```

Definition at line 1345 of file checkpoint.c.

4.3.2.10 S_WRITE_DOLLAR

```
#define S_WRITE_DOLLAR(
                ARG )
```

Value:

```
    if ( ARG.size && ARG.where && ARG.where != &(AM.dollarzero) ) { \
        S_WRITE_B(ARG.where, ARG.size*sizeof(WORD)) \
    }
```

Definition at line 1356 of file checkpoint.c.

4.3.3 Function Documentation

4.3.3.1 CheckRecoveryFile()

```
int CheckRecoveryFile ( )
```

Checks whether a snapshot/recovery file exists. Returns 1 if it exists, 0 otherwise.

Definition at line 278 of file checkpoint.c.

4.3.3.2 DeleteRecoveryFile()

```
void DeleteRecoveryFile ( )
```

Deletes the recovery files. It is called by CleanUp() in the case of a successful completion.

Definition at line 333 of file checkpoint.c.

4.3.3.3 RecoveryFilename()

```
char* RecoveryFilename ( )
```

Returns pointer to recovery filename.

Definition at line 364 of file checkpoint.c.

4.3.3.4 InitRecovery()

```
void InitRecovery ( )
```

Sets up the strings for the filenames of the recovery files. This functions should only be called once to avoid memory leaks and after AM.TempDir has been initialized.

Definition at line 399 of file checkpoint.c.

4.3.3.5 DoRecovery()

```
int DoRecovery (
            int * moduletype )
```

Reads from the recovery file and restores all necessary variables and states in FORM, so that the execution can recommence in preprocessor() as if no restart of FORM had occurred.

The recovery file is read into memory as a whole. The pointer p then points into this memory at the next non-processed data. The macros by which variables are restored, like R_SET, automatically increase p appropriately.

If something goes wrong, the function returns with a non-zero value.

Allocated memory that would be lost when overwriting the global structs with data from the file is freed first. A major part of the code deals with the restoration of pointers. The idiom we use is to memorize the original pointer value (org), allocate new memory and copy the data from the file into this memory, calculate the offset between the old pointer value and the new allocated memory position (ofs), and then correct all affected pointers (+=ofs).

We rely on the fact that several variables (especially in AM) are already assigned the correct values by the startup functions. That means, in principle, that a change in the setup files between snapshot creation and recovery will be noticed.

Definition at line 1399 of file checkpoint.c.

4.3.3.6 DoCheckpoint()

```
void DoCheckpoint (
            int moduletype )
```

Checks whether a snapshot should be done. Calls DoSnapshot() to create the snapshot.

Definition at line 3102 of file checkpoint.c.

References TimeWallClock().

4.4 comexpr.c File Reference

```
#include "form3.h"
```

Data Structures

- struct **id_options**

Functions

- int **CoLocal** (UBYTE ∗inp)
- int **CoGlobal** (UBYTE ∗inp)
- int **CoLocalFactorized** (UBYTE ∗inp)
- int **CoGlobalFactorized** (UBYTE ∗inp)
- int **DoExpr** (UBYTE ∗inp, int type, int par)
- int **CoIdOld** (UBYTE ∗inp)
- int **CoId** (UBYTE ∗inp)
- int **CoIdNew** (UBYTE ∗inp)
- int **CoDisorder** (UBYTE ∗inp)
- int **CoMany** (UBYTE ∗inp)
- int **CoMulti** (UBYTE ∗inp)
- int **CoIfMatch** (UBYTE ∗inp)
- int **CoIfNoMatch** (UBYTE ∗inp)
- int **CoOnce** (UBYTE ∗inp)
- int **CoOnly** (UBYTE ∗inp)
- int **CoSelect** (UBYTE ∗inp)
- int **CoIdExpression** (UBYTE ∗inp, int type)
- int **CoMultiply** (UBYTE ∗inp)
- int **CoFill** (UBYTE ∗inp)
- int **CoFillExpression** (UBYTE ∗inp)
- int **CoPrintTable** (UBYTE ∗inp)
- int **CoAssign** (UBYTE ∗inp)
- int **CoDeallocateTable** (UBYTE ∗inp)

4.4.1 Detailed Description

Compiler routines for statements that involve algebraic expressions. These involve definitions, id-statements, the multiply statement and the fill statement.

4.5 compcomm.c File Reference

```
#include "form3.h"
#include "comtool.h"
```

Functions

- int **CoCollect** (UBYTE ∗s)
- int **setonoff** (UBYTE ∗s, int ∗flag, int onvalue, int offvalue)
- int **CoCompress** (UBYTE ∗s)
- int **CoFlags** (UBYTE ∗s, int value)
- int **CoOff** (UBYTE ∗s)
- int **CoOn** (UBYTE ∗s)
- int **CoInsideFirst** (UBYTE ∗s)
- int **CoProperCount** (UBYTE ∗s)
- int **CoDelete** (UBYTE ∗s)
- int **CoFormat** (UBYTE ∗s)
- int **CoKeep** (UBYTE ∗s)
- int **CoFixIndex** (UBYTE ∗s)
- int **CoMetric** (UBYTE ∗s)
- int **DoPrint** (UBYTE ∗s, int par)
- int **CoPrint** (UBYTE ∗s)
- int **CoPrintB** (UBYTE ∗s)
- int **CoNPrint** (UBYTE ∗s)
- int **CoPushHide** (UBYTE ∗s)
- int **CoPopHide** (UBYTE ∗s)
- int **SetExprCases** (int par, int setunset, int val)
- int **SetExpr** (UBYTE ∗s, int setunset, int par)
- int **CoDrop** (UBYTE ∗s)
- int **CoNoDrop** (UBYTE ∗s)
- int **CoSkip** (UBYTE ∗s)
- int **CoNoSkip** (UBYTE ∗s)
- int **CoHide** (UBYTE ∗inp)
- int **CoIntoHide** (UBYTE ∗inp)
- int **CoNoHide** (UBYTE ∗inp)
- int **CoUnHide** (UBYTE ∗inp)
- int **CoNoUnHide** (UBYTE ∗inp)
- void **AddToCom** (int n, WORD ∗array)
- int **AddComString** (int n, WORD ∗array, UBYTE ∗thestring, int par)
- int **Add2ComStrings** (int n, WORD ∗array, UBYTE ∗string1, UBYTE ∗string2)
- int **CoDiscard** (UBYTE ∗s)
- int **CoContract** (UBYTE ∗s)
- int **CoGoTo** (UBYTE ∗inp)
- int **CoLabel** (UBYTE ∗inp)
- int **DoArgument** (UBYTE ∗s, int par)
- int **CoArgument** (UBYTE ∗s)
- int **CoEndArgument** (UBYTE ∗s)
- int **CoInside** (UBYTE ∗s)
- int **CoEndInside** (UBYTE ∗s)
- int **CoNormalize** (UBYTE ∗s)
- int **CoMakeInteger** (UBYTE ∗s)
- int **CoSplitArg** (UBYTE ∗s)
- int **CoSplitFirstArg** (UBYTE ∗s)
- int **CoSplitLastArg** (UBYTE ∗s)
- int **CoFactArg** (UBYTE ∗s)
- int **DoSymmetrize** (UBYTE ∗s, int par)
- int **CoSymmetrize** (UBYTE ∗s)
- int **CoAntiSymmetrize** (UBYTE ∗s)
- int **CoCycleSymmetrize** (UBYTE ∗s)
- int **CoRCycleSymmetrize** (UBYTE ∗s)

- int **CoWrite** (UBYTE ∗s)
- int **CoNWrite** (UBYTE ∗s)
- int **CoRatio** (UBYTE ∗s)
- int **CoRedefine** (UBYTE ∗s)
- int **CoRenumber** (UBYTE ∗s)
- int **CoSum** (UBYTE ∗s)
- int **CoToTensor** (UBYTE ∗s)
- int **CoToVector** (UBYTE ∗s)
- int **CoTrace4** (UBYTE ∗s)
- int **CoTraceN** (UBYTE ∗s)
- int **CoChisholm** (UBYTE ∗s)
- int **DoChain** (UBYTE ∗s, int option)
- int **CoChainin** (UBYTE ∗s)
- int **CoChainout** (UBYTE ∗s)
- int **CoExit** (UBYTE ∗s)
- int **CoInParallel** (UBYTE ∗s)
- int **CoNotInParallel** (UBYTE ∗s)
- int **DoInParallel** (UBYTE ∗s, int par)
- int **CoInExpression** (UBYTE ∗s)
- int **CoEndInExpression** (UBYTE ∗s)
- int **CoSetExitFlag** (UBYTE ∗s)
- int **CoTryReplace** (UBYTE ∗p)
- int **CoModulus** (UBYTE ∗inp)
- int **CoRepeat** (UBYTE ∗inp)
- int **CoEndRepeat** (UBYTE ∗inp)
- int **DoBrackets** (UBYTE ∗inp, int par)
- int **CoBracket** (UBYTE ∗inp)
- int **CoAntiBracket** (UBYTE ∗inp)
- int **CoMultiBracket** (UBYTE ∗inp)
- WORD ∗ **CountComp** (UBYTE ∗inp, WORD ∗to)
- int **CoIf** (UBYTE ∗inp)
- int **CoElse** (UBYTE ∗p)
- int **CoElseIf** (UBYTE ∗inp)
- int **CoEndIf** (UBYTE ∗inp)
- int **CoWhile** (UBYTE ∗inp)
- int **CoEndWhile** (UBYTE ∗inp)
- int **DoFindLoop** (UBYTE ∗inp, int mode)
- int **CoFindLoop** (UBYTE ∗inp)
- int **CoReplaceLoop** (UBYTE ∗inp)
- int **CoFunPowers** (UBYTE ∗inp)
- int **CoUnitTrace** (UBYTE ∗s)
- int **CoTerm** (UBYTE ∗s)
- int **CoEndTerm** (UBYTE ∗s)
- int **CoSort** (UBYTE ∗s)
- int **CoPolyFun** (UBYTE ∗s)
- int **CoPolyRatFun** (UBYTE ∗s)
- int **CoMerge** (UBYTE ∗inp)
- int **CoStuffle** (UBYTE ∗inp)
- int **CoProcessBucket** (UBYTE ∗s)
- int **CoThreadBucket** (UBYTE ∗s)
- int **DoArgPlode** (UBYTE ∗s, int par)
- int **CoArgExplode** (UBYTE ∗s)
- int **CoArgImplode** (UBYTE ∗s)
- int **CoClearTable** (UBYTE ∗s)
- int **CoDenominators** (UBYTE ∗s)

- int **CoDropCoefficient** (UBYTE ∗s)
- int **CoDropSymbols** (UBYTE ∗s)
- int **CoToPolynomial** (UBYTE ∗inp)
- int **CoFromPolynomial** (UBYTE ∗inp)
- int **CoArgToExtraSymbol** (UBYTE ∗s)
- int **CoExtraSymbols** (UBYTE ∗inp)
- WORD ∗ **GetIfDollarFactor** (UBYTE ∗∗inp, WORD ∗w)
- UBYTE ∗ **GetDoParam** (UBYTE ∗inp, WORD ∗∗wp, int par)
- int **CoDo** (UBYTE ∗inp)
- int **CoEndDo** (UBYTE ∗inp)
- int **CoFactDollar** (UBYTE ∗inp)
- int **CoFactorize** (UBYTE ∗s)
- int **CoNFactorize** (UBYTE ∗s)
- int **CoUnFactorize** (UBYTE ∗s)
- int **CoNUnFactorize** (UBYTE ∗s)
- int **DoFactorize** (UBYTE ∗s, int par)
- int **CoOptimizeOption** (UBYTE ∗s)
- int **CoPutInside** (UBYTE ∗inp)
- int **CoAntiPutInside** (UBYTE ∗inp)
- int **DoPutInside** (UBYTE ∗inp, int par)
- int **CoSwitch** (UBYTE ∗s)
- int **CoCase** (UBYTE ∗s)
- int **CoBreak** (UBYTE ∗s)
- int **CoDefault** (UBYTE ∗s)
- int **CoEndSwitch** (UBYTE ∗s)

4.5.1 Detailed Description

Compiler routines for most statements that don't involve algebraic expressions. Exceptions: all routines involving declarations are in the file names.c When making new statements one can add the compiler routines here and have a look whether there is already a routine that is similar. In that case one can make a copy and modify it.

4.6 compiler.c File Reference

```
#include "form3.h"
```

Data Structures

- struct SuBbUf

Macros

- #define **OPTION0** 1
- #define **OPTION1** 2
- #define **OPTION2** 3
- #define **REDUCESUBEXPBUFFERS**

Typedefs

- typedef struct SuBbUf **SUBBUF**

Functions

- VOID **inictable** ()
- KEYWORD * **findcommand** (UBYTE *in)
- int **ParenthesesTest** (UBYTE *sin)
- UBYTE * **SkipAName** (UBYTE *s)
- UBYTE * **IsRHS** (UBYTE *s, UBYTE c)
- int **IsIdStatement** (UBYTE *s)
- int **CompileAlgebra** (UBYTE *s, int leftright, WORD *prototype)
- int **CompileStatement** (UBYTE *in)
- int **TestTables** ()
- int **CompileSubExpressions** (SBYTE *tokens)
- int **CodeGenerator** (SBYTE *tokens)
- int **CompleteTerm** (WORD *term, UWORD *numer, UWORD *denom, WORD nnum, WORD nden, int sign)
- int **CodeFactors** (SBYTE *tokens)
- WORD **GenerateFactors** (WORD n, WORD inc)

Variables

- int **alfatable1** [27]
- SUBBUF * **subexpbuffers** = 0
- SUBBUF * **topsubexpbuffers** = 0
- LONG **insubexpbuffers** = 0

4.6.1 Detailed Description

The heart of the compiler. It contains the tables of statements. It finds the statements in the tables and calls the proper routines. For algebraic expressions it runs the compilation by first calling the tokenizer, splitting things into subexpressions and generating the code. There is a system for recognizing already existing subexpressions. This economizes on the length of the output.

Note: the compiler of FORM doesn't attempt to normalize the input. Hence x+1 and 1+x are different objects during compilation. Similarly (a+b-b) will not be simplified to (a).

4.6.2 Macro Definition Documentation

4.6.2.1 REDUCESUBEXPBUFFERS

```
#define REDUCESUBEXPBUFFERS
```

Value:

```
{ if ( (topsubexpbuffers-subexpbuffers) > 256 ) {\
M_free(subexpbuffers,"subexpbuffers");\
subexpbuffers = (SUBBUF *)Malloc1(256*sizeof(SUBBUF),"subexpbuffers");\
topsubexpbuffers = subexpbuffers+256; } insubexpbuffers = 0; }
```

Definition at line 248 of file compiler.c.

4.7 compress.c File Reference

```
#include "form3.h"
```

4.7.1 Detailed Description

The routines for the use of gzip (de)compression of the information in the sort file.

4.8 comtool.c File Reference

```
#include "form3.h"
```

Functions

- int inicbufs (VOID)
- void finishcbuf (WORD num)
- void clearcbuf (WORD num)
- WORD * DoubleCbuffer (int num, WORD *w, int par)
- WORD * AddLHS (int num)
- WORD * AddRHS (int num, int type)
- int AddNtoL (int n, WORD *array)
- int AddNtoC (int bufnum, int n, WORD *array, int par)
- int **InsTree** (int bufnum, int h)
- int **FindTree** (int bufnum, WORD *subexpr)
- void **RedoTree** (CBUF *C, int size)
- void **ClearTree** (int i)
- int IniFbuffer (WORD bufnum)
- LONG **numcommute** (WORD *terms, LONG *numterms)

4.8.1 Detailed Description

Utility routines for the compiler.

4.8.2 Function Documentation

4.8.2.1 inicbufs()

```
int inicbufs (
            VOID  )
```

Creates a new compiler buffer and returns its ID number.

Returns

The ID number for the new compiler buffer.

Definition at line 47 of file comtool.c.

4.8.2.2 finishcbuf()

```
void finishcbuf (
            WORD num )
```

Frees a compiler buffer.

Parameters

num	The ID number for the buffer to be freed.

Definition at line 89 of file comtool.c.

4.8.2.3 clearcbuf()

```
void clearcbuf (
            WORD num )
```

Clears contents in a compiler buffer.

Parameters

num	The ID number for the buffer to be cleared.

Definition at line 116 of file comtool.c.

4.8.2.4 DoubleCbuffer()

```
WORD* DoubleCbuffer (
            int num,
```

```
            WORD * w,
            int par )
```

Doubles a compiler buffer.

Parameters

num	The ID number for the buffer to be doubled.
w	The pointer to the end (exclusive) of the current buffer. The contents in the range of [cbuf[num].Buffer,w) will be kept.

Definition at line 143 of file comtool.c.

4.8.2.5 AddLHS()

```
WORD* AddLHS (
            int num )
```

Adds an LHS to a compiler buffer and returns the pointer to a buffer for the new LHS.

Parameters

num	The ID number for the buffer to get another LHS.

Definition at line 188 of file comtool.c.

4.8.2.6 AddRHS()

```
WORD* AddRHS (
            int num,
            int type )
```

Adds an RHS to a compiler buffer and returns the pointer to a buffer for the new RHS.

Parameters

num	The ID number for the buffer to get another RHS.
type	If 0, the subexpression tree will be reallocated.

Definition at line 214 of file comtool.c.

4.8.2.7 AddNtoL()

```
int AddNtoL (
```

```
            int n,
            WORD * array )
```

Adds an LHS with the given data to the current compiler buffer.

Parameters

n	The length of the data.
array	The data to be added.

Returns

0 if succeeds.

Definition at line 288 of file comtool.c.

4.8.2.8 AddNtoC()

```
int AddNtoC (
            int bufnum,
            int n,
            WORD * array,
            int par )
```

Adds the given data to the last LHS/RHS in a compiler buffer.

Parameters

bufnum	The ID number for the buffer where the data will be added.
n	The length of the data.
array	The data to be added.

Returns

0 if succeeds.

Definition at line 317 of file comtool.c.

4.8.2.9 IniFbuffer()

```
int IniFbuffer (
            WORD bufnum )
```

Initialize a factorization cache buffer. We set the size of the rhs and boomlijst buffers immediately to their final values.

Definition at line 614 of file comtool.c.

4.9 comtool.h File Reference

```
#include "form3.h"
```

4.9.1 Detailed Description

Utility routines for the compiler.

4.10 declare.h File Reference

Macros

- #define **MaX**(x, y) ((x) > (y) ? (x): (y))
- #define **MiN**(x, y) ((x) < (y) ? (x): (y))
- #define **ABS**(x) ((x) < 0 ? -(x): (x))
- #define **SGN**(x) ((x) > 0 ? 1 : (x) < 0 ? -1 : 0)
- #define **REDLENG**(x) ((((x)<0)?((x)+1):((x)-1))/2)
- #define **INCLENG**(x) (((x)<0)?(((x)∗2)-1):(((x)∗2)+1))
- #define **GETCOEF**(x, y) x += ∗x;y = x[-1];x -= ABS(y);y=REDLENG(y)
- #define **GETSTOP**(x, y) y=x+(∗x)-1;y -= ABS(∗y)-1
- #define **StuffAdd**(x, y) (((x)<0?-1:1)∗(y)+((y)<0?-1:1)∗(x))
- #define **EXCHN**(t1, t2, n) { WORD a,i; for(i=0;i<n;i++){a=t1[i];t1[i]=t2[i];t2[i]=a;} }
- #define **EXCH**(x, y) { WORD a = (x); (x) = (y); (y) = a; }
- #define **TOKENTOLINE**(x, y)
- #define **UngetFromStream**(stream, c) ((stream)->nextchar[(stream)->isnextchar++]=c)
- #define **AddLineFeed**(s, n) { (s)[(n)++] = LINEFEED; }
- #define **TryRecover**(x) Terminate(-1)
- #define **UngetChar**(c) { pushbackchar = c; }
- #define **ParseNumber**(x, s) {(x)=0;while(∗(s)>='0'&&∗(s)<='9')(x)=10∗(x)+∗(s)++ -'0';}
- #define **ParseSign**(sgn, s)
- #define **ParseSignedNumber**(x, s)
- #define **NCOPY**(s, t, n) while (--n >= 0) ∗s++ = ∗t++;
- #define **NCOPYI**(s, t, n) while (--n >= 0) ∗s++ = ∗t++;
- #define **NCOPYB**(s, t, n) while (--n >= 0) ∗s++ = ∗t++;
- #define **NCOPYI32**(s, t, n) while (--n >= 0) ∗s++ = ∗t++;
- #define **WCOPY**(s, t, n) { int nn=n; WORD ∗ss=(WORD ∗)s, ∗tt=(WORD ∗)t; while (--nn >= 0) ∗ss++=∗tt++; }
- #define **NeedNumber**(x, s, err)
- #define **SKIPBLANKS**(s) { while (∗(s) == ' ' || ∗(s) == '\t') (s)++; }
- #define **FLUSHCONSOLE** if (AP.InOutBuf > 0) CharOut(LINEFEED)
- #define **SKIPBRA1**(s)
- #define **SKIPBRA2**(s)
- #define **SKIPBRA3**(s)
- #define **SKIPBRA4**(s)
- #define **SKIPBRA5**(s)
- #define **CYCLE1**(t, a, i) {t iX=∗a; WORD jX; for(jX=1;jX<i;jX++)a[jX-1]=a[jX]; a[i-1]=iX;}
- #define **AddToCB**(c, wx)
- #define **EXCHINOUT**
- #define **BACKINOUT**

- #define **CopyArg**(to, from)
- #define **FILLARG**(w)
- #define **COPYARG**(w, t)
- #define **ZEROARG**(w)
- #define **FILLFUN**(w)
- #define **COPYFUN**(w, t)
- #define **COPYFUN3**(w, t)
- #define **FILLFUN3**(w)
- #define **FILLSUB**(w)
- #define **COPYSUB**(w, ww)
- #define **FILLEXPR**(w)
- #define **NEXTARG**(x) if(*x>0) x += *x; else if(*x <= -FUNCTION)x++; else x += 2;
- #define **COPY1ARG**(s1, t1)
- #define ZeroFillRange(w, begin, end)
- #define **TABLESIZE**(a, b) (((WORD)sizeof(a))/((WORD)sizeof(b)))
- #define **WORDDIF**(x, y) (WORD)(x-y)
- #define **wsizeof**(a) ((WORD)sizeof(a))
- #define **VARNAME**(type, num) (AC.varnames->namebuffer+type[num].name)
- #define **DOLLARNAME**(type, num) (AC.dollarnames->namebuffer+type[num].name)
- #define **EXPRNAME**(num) (AC.exprnames->namebuffer+Expressions[num].name)
- #define **PREV**(x) prevorder?prevorder:x
- #define **SETERROR**(x) { Terminate(-1); return(-1); }
- #define **DUMMYUSE**(x) (void)(x);
- #define **ADDPOS**(pp, x) (pp).p1 = ((pp).p1+(LONG)(x))
- #define **SETBASELENGTH**(ss, x) (ss).p1 = (LONG)(x)
- #define **SETBASEPOSITION**(pp, x) (pp).p1 = (LONG)(x)
- #define **ISEQUALPOSINC**(pp1, pp2, x) ((pp1).p1 == ((pp2).p1+(LONG)(x)))
- #define **ISGEPOSINC**(pp1, pp2, x) ((pp1).p1 >= ((pp2).p1+(LONG)(x)))
- #define **DIVPOS**(pp, n) ((pp).p1/(LONG)(n))
- #define **MULPOS**(pp, n) (pp).p1 *= (LONG)(n)
- #define **DIFPOS**(ss, pp1, pp2) (ss).p1 = ((pp1).p1-(pp2).p1)
- #define **DIFBASE**(pp1, pp2) ((pp1).p1-(pp2).p1)
- #define **ADD2POS**(pp1, pp2) (pp1).p1 += (pp2).p1
- #define **PUTZERO**(pp) (pp).p1 = 0
- #define **BASEPOSITION**(pp) ((pp).p1)
- #define **SETSTARTPOS**(pp) (pp).p1 = -2
- #define **NOTSTARTPOS**(pp) ((pp).p1 > -2)
- #define **ISMINPOS**(pp) ((pp).p1 == -1)
- #define **ISEQUALPOS**(pp1, pp2) ((pp1).p1 == (pp2).p1)
- #define **ISNOTEQUALPOS**(pp1, pp2) ((pp1).p1 != (pp2).p1)
- #define **ISLESSPOS**(pp1, pp2) ((pp1).p1 < (pp2).p1)
- #define **ISGEPOS**(pp1, pp2) ((pp1).p1 >= (pp2).p1)
- #define **ISNOTZEROPOS**(pp) ((pp).p1 != 0)
- #define **ISZEROPOS**(pp) ((pp).p1 == 0)
- #define **ISPOSPOS**(pp) ((pp).p1 > 0)
- #define **ISNEGPOS**(pp) ((pp).p1 < 0)
- #define **TOLONG**(x) ((LONG)(x))
- #define **Add2Com**(x) { WORD cod[2]; cod[0] = x; cod[1] = 2; AddNtoL(2,cod); }
- #define **Add3Com**(x1, x2) { WORD cod[3]; cod[0] = x1; cod[1] = 3; cod[2] = x2; AddNtoL(3,cod); }
- #define **Add4Com**(x1, x2, x3)
- #define **Add5Com**(x1, x2, x3, x4)
- #define **WantAddPointers**(x)
- #define **WantAddLongs**(x)
- #define **WantAddPositions**(x)
- #define **FORM_INLINE** inline

- #define **MEMORYMACROS**
- #define **TermMalloc**(x) ((AT.TermMemTop <= 0) ? TermMallocAddMemory(BHEAD0), AT.TermMemHeap[--AT.TermMemTop]: AT.TermMemHeap[--AT.TermMemTop])
- #define **NumberMalloc**(x) ((AT.NumberMemTop <= 0) ? NumberMallocAddMemory(BHEAD0), AT.↩ NumberMemHeap[--AT.NumberMemTop]: AT.NumberMemHeap[--AT.NumberMemTop])
- #define **CacheNumberMalloc**(x) ((AT.CacheNumberMemTop <= 0) ? CacheNumberMallocAdd↩ Memory(BHEAD0), AT.CacheNumberMemHeap[--AT.CacheNumberMemTop]: AT.CacheNumberMemHeap[--AT.CacheNumberMemTop])
- #define **TermFree**(TermMem, x) AT.TermMemHeap[AT.TermMemTop++] = (WORD *)(TermMem)
- #define **NumberFree**(NumberMem, x) AT.NumberMemHeap[AT.NumberMemTop++] = (UWORD *)(NumberMem)
- #define **CacheNumberFree**(NumberMem, x) AT.CacheNumberMemHeap[AT.CacheNumberMemTop++] = (UWORD *)(NumberMem)
- #define **NestingChecksum**() (AC.IfLevel + AC.RepLevel + AC.arglevel + AC.insidelevel + AC.termlevel + AC.inexprlevel + AC.dolooplevel +AC.SwitchLevel)
- #define **MesNesting**() MesPrint("&Illegal nesting of if, repeat, argument, inside, term, inexpression and do")
- #define **MarkPolyRatFunDirty**(T)
- #define **PUSHPREASSIGNLEVEL**
- #define **POPPREASSIGNLEVEL**
- #define EXTERNLOCK(x)
- #define **INILOCK**(x)
- #define **LOCK**(x)
- #define **UNLOCK**(x)
- #define **EXTERNRWLOCK**(x)
- #define **INIRWLOCK**(x)
- #define **RWLOCKR**(x)
- #define **RWLOCKW**(x)
- #define **UNRWLOCK**(x)
- #define **MLOCK**(x)
- #define **MUNLOCK**(x)
- #define **GETIDENTITY**
- #define **GETBIDENTITY**
- #define **M_alloc**(x) malloc((size_t)(x))
- #define **CompareTerms** ((COMPARE)AR.CompareRoutine)
- #define **FiniShuffle** AN.SHvar.finishuf
- #define **DoShtuffle** ((DO_UFFLE)AN.SHvar.do_uffle)

Typedefs

- typedef int(* **WRITEBUFTOEXTCHANNEL**) (char *, size_t)
- typedef int(* **GETCFROMEXTCHANNEL**) (VOID)
- typedef int(* **SETTERMINATORFOREXTERNALCHANNEL**) (char *)
- typedef int(* **SETKILLMODEFOREXTERNALCHANNEL**) (int, int)
- typedef LONG(* **WRITEFILE**) (int, UBYTE *, LONG)
- typedef WORD(* **GETTERM**) (PHEAD WORD *)

Functions

- VOID **TELLFILE** (int, POSITION *)
- VOID StartVariables ()
- VOID **setSignalHandlers** (VOID)
- UBYTE * **CodeToLine** (WORD, UBYTE *)
- UBYTE * **AddArrayIndex** (WORD, UBYTE *)
- INDEXENTRY * **FindInIndex** (WORD, FILEDATA *, WORD, WORD)
- INDEXENTRY * **NextFileIndex** (POSITION *)
- WORD * PasteTerm (PHEAD WORD, WORD *, WORD *, WORD, WORD)
- UBYTE * **StrCopy** (UBYTE *, UBYTE *)
- UBYTE * **WrtPower** (UBYTE *, WORD)
- WORD **AccumGCD** (PHEAD UWORD *, WORD *, UWORD *, WORD)
- VOID AddArgs (PHEAD WORD *, WORD *, WORD *)
- WORD AddCoef (PHEAD WORD **, WORD **)
- WORD **AddLong** (UWORD *, WORD, UWORD *, WORD, UWORD *, WORD *)
- WORD **AddPLon** (UWORD *, WORD, UWORD *, WORD, UWORD *, WORD *)
- WORD AddPoly (PHEAD WORD **, WORD **)
- WORD **AddRat** (PHEAD UWORD *, WORD, UWORD *, WORD, UWORD *, WORD *)
- VOID **AddToLine** (UBYTE *)
- WORD **AddWild** (PHEAD WORD, WORD, WORD)
- WORD **BigLong** (UWORD *, WORD, UWORD *, WORD)
- WORD **BinomGen** (PHEAD WORD *, WORD, WORD **, WORD, WORD, WORD, WORD, WORD, UWORD *, WORD)
- WORD **CheckWild** (PHEAD WORD, WORD, WORD, WORD *)
- WORD **Chisholm** (PHEAD WORD *, WORD)
- WORD **CleanExpr** (WORD)
- VOID **CleanUp** (WORD)
- VOID **ClearWild** (PHEAD0)
- WORD **CompareFunctions** (WORD *, WORD *)
- WORD **Commute** (WORD *, WORD *)
- WORD **DetCommu** (WORD *)
- WORD **DoesCommu** (WORD *)
- int **CompArg** (WORD *, WORD *)
- WORD CompCoef (WORD *, WORD *)
- WORD **CompGroup** (PHEAD WORD, WORD **, WORD *, WORD *, WORD)
- WORD Compare1 (WORD *, WORD *, WORD)
- WORD **CountDo** (WORD *, WORD *)
- WORD **CountFun** (WORD *, WORD *)
- WORD **DimensionSubterm** (WORD *)
- WORD **DimensionTerm** (WORD *)
- WORD **DimensionExpression** (PHEAD WORD *)
- WORD Deferred (PHEAD WORD *, WORD)
- WORD **DeleteStore** (WORD)
- WORD **DetCurDum** (PHEAD WORD *)
- VOID **DetVars** (WORD *, WORD)
- WORD **Distribute** (DISTRIBUTE *, WORD)
- WORD **DivLong** (UWORD *, WORD, UWORD *, WORD, UWORD *, WORD *, UWORD *, WORD *)
- WORD **DivRat** (PHEAD UWORD *, WORD, UWORD *, WORD, UWORD *, WORD *)
- WORD **Divvy** (PHEAD UWORD *, WORD *, UWORD *, WORD)
- WORD **DoDelta** (WORD *)
- WORD **DoDelta3** (PHEAD WORD *, WORD)
- WORD **TestPartitions** (WORD *, PARTI *)
- WORD **DoPartitions** (PHEAD WORD *, WORD)
- int **CoCanonicalize** (UBYTE *)

- int **DoCanonicalize** (PHEAD WORD *, WORD *)
- WORD **GenTopologies** (PHEAD WORD *, WORD)
- WORD **GenDiagrams** (PHEAD WORD *, WORD)
- int **DoTopologyCanonicalize** (PHEAD WORD *, WORD, WORD, WORD *)
- int **DoShattering** (PHEAD WORD *, WORD *, WORD *, WORD)
- WORD **GenerateTopologies** (PHEAD WORD, WORD, WORD, WORD)
- WORD **DoTableExpansion** (WORD *, WORD)
- WORD **DoDistrib** (PHEAD WORD *, WORD)
- WORD **DoShuffle** (WORD *, WORD, WORD, WORD)
- WORD **DoPermutations** (PHEAD WORD *, WORD)
- int **Shuffle** (WORD *, WORD *, WORD *)
- int **FinishShuffle** (WORD *)
- WORD **DoStuffle** (WORD *, WORD, WORD, WORD)
- int **Stuffle** (WORD *, WORD *, WORD *)
- int **FinishStuffle** (WORD *)
- WORD * **StuffRootAdd** (WORD *, WORD *, WORD *)
- WORD **TestUse** (WORD *, WORD)
- DBASE * **FindTB** (UBYTE *)
- int **CheckTableDeclarations** (DBASE *)
- WORD **Apply** (WORD *, WORD)
- int **ApplyExec** (WORD *, int, WORD)
- WORD **ApplyReset** (WORD)
- WORD **TableReset** (VOID)
- VOID **ReWorkT** (WORD *, WORD *, WORD)
- WORD **GetIfDollarNum** (WORD *, WORD *)
- int **FindVar** (WORD *, WORD *)
- WORD **DoIfStatement** (PHEAD WORD *, WORD *)
- WORD DoOnePow (PHEAD WORD *, WORD, WORD, WORD *, WORD *, WORD, WORD *)
- void **DoRevert** (WORD *, WORD *)
- WORD **DoSumF1** (PHEAD WORD *, WORD *, WORD, WORD)
- WORD **DoSumF2** (PHEAD WORD *, WORD *, WORD, WORD)
- WORD **DoTheta** (PHEAD WORD *)
- LONG EndSort (PHEAD WORD *, int)
- WORD **EntVar** (WORD, UBYTE *, WORD, WORD, WORD, WORD)
- WORD **EpfCon** (PHEAD WORD *, WORD *, WORD, WORD)
- WORD **EpfFind** (PHEAD WORD *, WORD *)
- WORD **EpfGen** (WORD, WORD *, WORD *, WORD *, WORD)
- WORD **EqualArg** (WORD *, WORD, WORD)
- WORD **Evaluate** (UBYTE **)
- int **Factorial** (PHEAD WORD, UWORD *, WORD *)
- int **Bernoulli** (WORD, UWORD *, WORD *)
- int **FactorIn** (PHEAD WORD *, WORD)
- int **FactorInExpr** (PHEAD WORD *, WORD)
- WORD **FindAll** (PHEAD WORD *, WORD *, WORD, WORD *)
- WORD **FindMulti** (PHEAD WORD *, WORD *)
- WORD **FindOnce** (PHEAD WORD *, WORD *)
- WORD **FindOnly** (PHEAD WORD *, WORD *)
- WORD **FindRest** (PHEAD WORD *, WORD *)
- WORD **FindSpecial** (WORD *)
- WORD **FindrNumber** (WORD, VARRENUM *)
- VOID **FiniLine** (VOID)
- WORD FiniTerm (PHEAD WORD *, WORD *, WORD *, WORD, WORD)
- WORD FlushOut (POSITION *, FILEHANDLE *, int)
- VOID **FunLevel** (PHEAD WORD *)
- VOID **AdjustRenumScratch** (PHEAD0)

- VOID **GarbHand** (VOID)
- WORD **GcdLong** (PHEAD UWORD *, WORD, UWORD *, WORD, UWORD *, WORD *)
- WORD **LcmLong** (PHEAD UWORD *, WORD, UWORD *, WORD, UWORD *, WORD *)
- VOID **GCD** (UWORD *, WORD, UWORD *, WORD, UWORD *, WORD *)
- ULONG **GCD2** (ULONG, ULONG)
- WORD Generator (PHEAD WORD *, WORD)
- WORD **GetBinom** (UWORD *, WORD *, WORD, WORD)
- WORD **GetFromStore** (WORD *, POSITION *, RENUMBER, WORD *, WORD)
- WORD **GetLong** (UBYTE *, UWORD *, WORD *)
- WORD **GetMoreTerms** (WORD *)
- WORD **GetMoreFromMem** (WORD *, WORD **)
- WORD **GetOneTerm** (PHEAD WORD *, FILEHANDLE *, POSITION *, int)
- RENUMBER **GetTable** (WORD, POSITION *, WORD)
- WORD **GetTerm** (PHEAD WORD *)
- WORD **Glue** (PHEAD WORD *, WORD *, WORD *, WORD)
- WORD InFunction (PHEAD WORD *, WORD *)
- VOID **IniLine** (WORD)
- WORD **IniVars** (VOID)
- VOID **Initialize** (VOID)
- WORD InsertTerm (PHEAD WORD *, WORD, WORD, WORD *, WORD *, WORD)
- VOID **LongToLine** (UWORD *, WORD)
- WORD **MakeDirty** (WORD *, WORD *, WORD)
- VOID **MarkDirty** (WORD *, WORD)
- VOID **PolyFunDirty** (PHEAD WORD *)
- VOID **PolyFunClean** (PHEAD WORD *)
- WORD **MakeModTable** (VOID)
- WORD **MatchE** (PHEAD WORD *, WORD *, WORD *, WORD)
- int **MatchCy** (PHEAD WORD *, WORD *, WORD *, WORD)
- int **FunMatchCy** (PHEAD WORD *, WORD *, WORD *, WORD)
- int **FunMatchSy** (PHEAD WORD *, WORD *, WORD *, WORD)
- int **MatchArgument** (PHEAD WORD *, WORD *)
- WORD **MatchFunction** (PHEAD WORD *, WORD *, WORD *)
- WORD MergePatches (WORD)
- WORD **MesCerr** (char *, UBYTE *)
- WORD **MesComp** (char *, UBYTE *, UBYTE *)
- WORD **Modulus** (WORD *)
- VOID **MoveDummies** (PHEAD WORD *, WORD)
- WORD **MulLong** (UWORD *, WORD, UWORD *, WORD, UWORD *, WORD *)
- WORD **MulRat** (PHEAD UWORD *, WORD, UWORD *, WORD, UWORD *, WORD *)
- WORD **Mully** (PHEAD UWORD *, WORD *, UWORD *, WORD)
- WORD **MultDo** (PHEAD WORD *, WORD *)
- WORD NewSort (PHEAD0)
- WORD **ExtraSymbol** (WORD, WORD, WORD, WORD *, WORD *)
- WORD **Normalize** (PHEAD WORD *)
- WORD **BracketNormalize** (PHEAD WORD *)
- VOID **DropCoefficient** (PHEAD WORD *)
- VOID **DropSymbols** (PHEAD WORD *)
- int **PutInside** (PHEAD WORD *, WORD *)
- WORD **OpenTemp** (VOID)
- VOID **Pack** (UWORD *, WORD *, UWORD *, WORD)
- LONG PasteFile (PHEAD WORD, WORD *, POSITION *, WORD **, RENUMBER, WORD *, WORD)
- WORD **Permute** (PERM *, WORD)
- WORD **PermuteP** (PERMP *, WORD)
- WORD PolyFunMul (PHEAD WORD *)
- WORD **PopVariables** (VOID)

- WORD PrepPoly (PHEAD WORD *, WORD)
- WORD **Processor** (VOID)
- WORD **Product** (UWORD *, WORD *, WORD)
- VOID **PrtLong** (UWORD *, WORD, UBYTE *)
- VOID **PrtTerms** (VOID)
- VOID **PrintRunningTime** (VOID)
- LONG **GetRunningTime** (VOID)
- WORD **PutBracket** (PHEAD WORD *)
- LONG PutIn (FILEHANDLE *, POSITION *, WORD *, WORD **, int)
- WORD **PutInStore** (INDEXENTRY *, WORD)
- WORD PutOut (PHEAD WORD *, POSITION *, FILEHANDLE *, WORD)
- UWORD **Quotient** (UWORD *, WORD *, WORD)
- WORD **RaisPow** (PHEAD UWORD *, WORD *, UWORD)
- VOID RaisPowCached (PHEAD WORD, WORD, UWORD **, WORD *)
- WORD **RaisPowMod** (WORD, WORD, WORD)
- int NormalModulus (UWORD *, WORD *)
- int **MakeInverses** (VOID)
- int GetModInverses (WORD, WORD, WORD *, WORD *)
- int **GetLongModInverses** (PHEAD UWORD *, WORD, UWORD *, WORD, UWORD *, WORD *, UWORD *, WORD *)
- VOID **RatToLine** (UWORD *, WORD)
- WORD **RatioFind** (PHEAD WORD *, WORD *)
- WORD **RatioGen** (PHEAD WORD *, WORD *, WORD, WORD)
- WORD **ReNumber** (PHEAD WORD *)
- WORD **ReadSnum** (UBYTE **)
- WORD **Remain10** (UWORD *, WORD *)
- WORD **Remain4** (UWORD *, WORD *)
- WORD **ResetScratch** (VOID)
- WORD **ResolveSet** (PHEAD WORD *, WORD *, WORD *)
- WORD **RevertScratch** (VOID)
- WORD **ScanFunctions** (PHEAD WORD *, WORD *, WORD)
- VOID **SeekScratch** (FILEHANDLE *, POSITION *)
- VOID **SetEndScratch** (FILEHANDLE *, POSITION *)
- VOID **SetEndHScratch** (FILEHANDLE *, POSITION *)
- WORD **SetFileIndex** (VOID)
- WORD Sflush (FILEHANDLE *)
- WORD **Simplify** (PHEAD UWORD *, WORD *, UWORD *, WORD *)
- WORD SortWild (WORD *, WORD)
- FILE * **LocateBase** (char **, char **)
- LONG SplitMerge (PHEAD WORD **, LONG)
- WORD StoreTerm (PHEAD WORD *)
- VOID **SubPLon** (UWORD *, WORD, UWORD *, WORD, UWORD *, WORD *)
- VOID **Substitute** (PHEAD WORD *, WORD *, WORD)
- WORD **SymFind** (PHEAD WORD *, WORD *)
- WORD **SymGen** (PHEAD WORD *, WORD *, WORD, WORD)
- WORD **Symmetrize** (PHEAD WORD *, WORD *, WORD, WORD, WORD)
- int **FullSymmetrize** (PHEAD WORD *, int)
- WORD **TakeModulus** (UWORD *, WORD *, UWORD *, WORD, WORD)
- WORD **TakeNormalModulus** (UWORD *, WORD *, UWORD *, WORD, WORD)
- VOID **TalToLine** (UWORD)
- WORD **TenVec** (PHEAD WORD *, WORD *, WORD, WORD)
- WORD **TenVecFind** (PHEAD WORD *, WORD *)
- WORD TermRenumber (WORD *, RENUMBER, WORD)
- VOID **TestDrop** (VOID)
- VOID **PutInVflags** (WORD)

- WORD TestMatch (PHEAD WORD *, WORD *)
- WORD TestSub (PHEAD WORD *, WORD)
- LONG TimeCPU (WORD)
- LONG **TimeChildren** (WORD)
- LONG TimeWallClock (WORD)
- LONG **Timer** (int)
- int **GetTimerInfo** (LONG **, LONG **)
- void **WriteTimerInfo** (LONG *, LONG *)
- LONG **GetWorkerTimes** (VOID)
- WORD **ToStorage** (EXPRESSIONS, POSITION *)
- VOID **TokenToLine** (UBYTE *)
- WORD **Trace4** (PHEAD WORD *, WORD *, WORD, WORD)
- WORD **Trace4Gen** (PHEAD TRACES *, WORD)
- WORD **Trace4no** (WORD, WORD *, TRACES *)
- WORD **TraceFind** (PHEAD WORD *, WORD *)
- WORD **TraceN** (PHEAD WORD *, WORD *, WORD, WORD)
- WORD **TraceNgen** (PHEAD TRACES *, WORD)
- WORD **TraceNno** (WORD, WORD *, TRACES *)
- WORD **Traces** (PHEAD WORD *, WORD *, WORD, WORD)
- WORD **Trick** (WORD *, TRACES *)
- WORD **TryDo** (PHEAD WORD *, WORD *, WORD)
- VOID **UnPack** (UWORD *, WORD, WORD *, WORD *)
- WORD **VarStore** (UBYTE *, WORD, WORD, WORD)
- WORD **WildFill** (PHEAD WORD *, WORD *, WORD *)
- WORD **WriteAll** (VOID)
- WORD **WriteOne** (UBYTE *, int, int, WORD)
- VOID **WriteArgument** (WORD *)
- WORD **WriteExpression** (WORD *, LONG)
- WORD **WriteInnerTerm** (WORD *, WORD)
- VOID **WriteLists** (VOID)
- VOID **WriteSetup** (VOID)
- VOID WriteStats (POSITION *, WORD)
- WORD **WriteSubTerm** (WORD *, WORD)
- WORD **WriteTerm** (WORD *, WORD *, WORD, WORD, WORD)
- WORD **execarg** (PHEAD WORD *, WORD)
- WORD **execterm** (PHEAD WORD *, WORD)
- VOID **SpecialCleanup** (PHEAD0)
- void **SetMods** ()
- void **UnSetMods** ()
- WORD **DoExecute** (WORD, WORD)
- VOID **SetScratch** (FILEHANDLE *, POSITION *)
- VOID **Warning** (char *)
- VOID **HighWarning** (char *)
- int **SpareTable** (TABLES)
- UBYTE * **strDup1** (UBYTE *, char *)
- VOID * **Malloc** (LONG)
- VOID * **Malloc1** (LONG, const char *)
- int **DoTail** (int, UBYTE **)
- int **OpenInput** (VOID)
- int PutPreVar (UBYTE *, UBYTE *, UBYTE *, int)
- VOID **Error0** (char *)
- VOID **Error1** (char *, UBYTE *)
- VOID **Error2** (char *, char *, UBYTE *)
- UBYTE **ReadFromStream** (STREAM *)
- UBYTE **GetFromStream** (STREAM *)

- UBYTE **LookInStream** (STREAM *)
- STREAM * **OpenStream** (UBYTE *, int, int, int)
- int **LocateFile** (UBYTE **, int)
- STREAM * **CloseStream** (STREAM *)
- VOID **PositionStream** (STREAM *, LONG)
- int **ReverseStatements** (STREAM *)
- int **ProcessOption** (UBYTE *, UBYTE *, int)
- int **DoSetups** (VOID)
- VOID **Terminate** (int)
- NAMENODE * **GetNode** (NAMETREE *, UBYTE *)
- int **AddName** (NAMETREE *, UBYTE *, WORD, WORD, int *)
- int **GetName** (NAMETREE *, UBYTE *, WORD *, int)
- UBYTE * **GetFunction** (UBYTE *, WORD *)
- UBYTE * **GetNumber** (UBYTE *, WORD *)
- int **GetLastExprName** (UBYTE *, WORD *)
- int **GetAutoName** (UBYTE *, WORD *)
- int **GetVar** (UBYTE *, WORD *, WORD *, int, int)
- int **MakeDubious** (NAMETREE *, UBYTE *, WORD *)
- int **GetOName** (NAMETREE *, UBYTE *, WORD *, int)
- VOID **DumpTree** (NAMETREE *)
- VOID **DumpNode** (NAMETREE *, WORD, WORD)
- VOID **LinkTree** (NAMETREE *, WORD, WORD)
- VOID **CopyTree** (NAMETREE *, NAMETREE *, WORD, WORD)
- int **CompactifyTree** (NAMETREE *, WORD)
- NAMETREE * **MakeNameTree** (VOID)
- VOID **FreeNameTree** (NAMETREE *)
- int **AddExpression** (UBYTE *, int, int)
- int **AddSymbol** (UBYTE *, int, int, int, int)
- int **AddDollar** (UBYTE *, WORD, WORD *, LONG)
- int **ReplaceDollar** (WORD, WORD, WORD *, LONG)
- int **DollarRaiseLow** (UBYTE *, LONG)
- int **AddVector** (UBYTE *, int, int)
- int **AddDubious** (UBYTE *)
- int **AddIndex** (UBYTE *, int, int)
- UBYTE * **DoDimension** (UBYTE *, int *, int *)
- int **AddFunction** (UBYTE *, int, int, int, int, int, int, int)
- int **CoCommuteInSet** (UBYTE *)
- int **CoFunction** (UBYTE *, int, int)
- int **TestName** (UBYTE *)
- int **AddSet** (UBYTE *, WORD)
- int **DoElements** (UBYTE *, SETS, UBYTE *)
- int **DoTempSet** (UBYTE *, UBYTE *)
- int **NameConflict** (int, UBYTE *)
- int **OpenFile** (char *)
- int **OpenAddFile** (char *)
- int **ReOpenFile** (char *)
- int **CreateFile** (char *)
- int **CreateLogFile** (char *)
- VOID **CloseFile** (int)
- int CopyFile (char *, char *)
- int **CreateHandle** (VOID)
- LONG **ReadFile** (int, UBYTE *, LONG)
- LONG **ReadPosFile** (PHEAD FILEHANDLE *, UBYTE *, LONG, POSITION *)
- LONG **WriteFileToFile** (int, UBYTE *, LONG)
- VOID **SeekFile** (int, POSITION *, int)

- LONG **TellFile** (int)
- void **FlushFile** (int)
- int **GetPosFile** (int, fpos_t *)
- int **SetPosFile** (int, fpos_t *)
- VOID **SynchFile** (int)
- VOID **TruncateFile** (int)
- int **GetChannel** (char *, int)
- int **GetAppendChannel** (char *)
- int **CloseChannel** (char *)
- VOID **inictable** (VOID)
- KEYWORD * **findcommand** (UBYTE *)
- int inicbufs (VOID)
- VOID **StartFiles** (VOID)
- UBYTE * **MakeDate** (VOID)
- VOID **PreProcessor** (VOID)
- VOID * **FromList** (LIST *)
- VOID * **From0List** (LIST *)
- VOID * **FromVarList** (LIST *)
- int **DoubleList** (VOID ***, int *, int, char *)
- int **DoubleLList** (VOID ***, LONG *, int, char *)
- void **DoubleBuffer** (void **, void **, int, char *)
- void **ExpandBuffer** (void **, LONG *, int)
- LONG **iexp** (LONG, int)
- int **IsLikeVector** (WORD *)
- int **AreArgsEqual** (WORD *, WORD *)
- int **CompareArgs** (WORD *, WORD *)
- UBYTE * **SkipField** (UBYTE *, int)
- int **StrCmp** (UBYTE *, UBYTE *)
- int **StrICmp** (UBYTE *, UBYTE *)
- int **StrHICmp** (UBYTE *, UBYTE *)
- int **StrICont** (UBYTE *, UBYTE *)
- int **CmpArray** (WORD *, WORD *, WORD)
- int **ConWord** (UBYTE *, UBYTE *)
- int **StrLen** (UBYTE *)
- UBYTE * **GetPreVar** (UBYTE *, int)
- void **ToGeneral** (WORD *, WORD *, WORD)
- WORD **ToPolyFunGeneral** (PHEAD WORD *)
- int **ToFast** (WORD *, WORD *)
- SETUPPARAMETERS * **GetSetupPar** (UBYTE *)
- int **RecalcSetups** (VOID)
- int **AllocSetups** (VOID)
- SORTING * **AllocSort** (LONG, LONG, LONG, LONG, int, int, LONG)
- VOID **AllocSortFileName** (SORTING *)
- UBYTE * **LoadInputFile** (UBYTE *, int)
- UBYTE **GetInput** (VOID)
- VOID **ClearPushback** (VOID)
- UBYTE **GetChar** (int)
- VOID **CharOut** (UBYTE)
- VOID **UnsetAllowDelay** (VOID)
- VOID **PopPreVars** (int)
- VOID **IniModule** (int)
- VOID **IniSpecialModule** (int)
- int **ModuleInstruction** (int *, int *)
- int **PreProInstruction** (VOID)
- int **LoadInstruction** (int)

- int **LoadStatement** (int)
- KEYWORD * **FindKeyWord** (UBYTE *, KEYWORD *, int)
- KEYWORD * **FindInKeyWord** (UBYTE *, KEYWORD *, int)
- int **DoDefine** (UBYTE *)
- int **DoRedefine** (UBYTE *)
- int TheDefine (UBYTE *, int)
- int **TheUndefine** (UBYTE *)
- int **ClearMacro** (UBYTE *)
- int **DoUndefine** (UBYTE *)
- int **DoInclude** (UBYTE *)
- int **DoReverseInclude** (UBYTE *)
- int **Include** (UBYTE *, int)
- int **DoExternal** (UBYTE *)
- int **DoToExternal** (UBYTE *)
- int **DoFromExternal** (UBYTE *)
- int **DoPrompt** (UBYTE *)
- int **DoSetExternal** (UBYTE *)
- int **DoSetExternalAttr** (UBYTE *)
- int **DoRmExternal** (UBYTE *)
- int **DoFactDollar** (UBYTE *)
- WORD **GetDollarNumber** (UBYTE **, DOLLARS)
- int **DoSetRandom** (UBYTE *)
- int **DoOptimize** (UBYTE *)
- int **DoClearOptimize** (UBYTE *)
- int **DoSkipExtraSymbols** (UBYTE *)
- int **DoTimeOutAfter** (UBYTE *)
- int **DoMessage** (UBYTE *)
- int **DoPreOut** (UBYTE *)
- int **DoPreAppend** (UBYTE *)
- int **DoPreCreate** (UBYTE *)
- int **DoPreAssign** (UBYTE *)
- int **DoPreBreak** (UBYTE *)
- int **DoPreDefault** (UBYTE *)
- int **DoPreSwitch** (UBYTE *)
- int **DoPreEndSwitch** (UBYTE *)
- int **DoPreCase** (UBYTE *)
- int **DoPreShow** (UBYTE *)
- int **DoPreExchange** (UBYTE *)
- int **DoSystem** (UBYTE *)
- int **DoPipe** (UBYTE *)
- VOID **StartPrepro** (VOID)
- int **DoIfdef** (UBYTE *, int)
- int **DoIfydef** (UBYTE *)
- int **DoIfndef** (UBYTE *)
- int **DoElse** (UBYTE *)
- int **DoElseif** (UBYTE *)
- int **DoEndif** (UBYTE *)
- int **DoTerminate** (UBYTE *)
- int **DoIf** (UBYTE *)
- int **DoCall** (UBYTE *)
- int **DoDebug** (UBYTE *)
- int **DoDo** (UBYTE *)
- int **DoBreakDo** (UBYTE *)
- int **DoEnddo** (UBYTE *)
- int **DoEndprocedure** (UBYTE *)

- int **DoInside** (UBYTE *)
- int **DoEndInside** (UBYTE *)
- int **DoProcedure** (UBYTE *)
- int **DoPrePrintTimes** (UBYTE *)
- int **DoPreWrite** (UBYTE *)
- int **DoPreClose** (UBYTE *)
- int **DoPreRemove** (UBYTE *)
- int **DoCommentChar** (UBYTE *)
- int **DoPrcExtension** (UBYTE *)
- int **DoPreReset** (UBYTE *)
- VOID **WriteString** (int, UBYTE *, int)
- VOID **WriteUnfinString** (int, UBYTE *, int)
- UBYTE * **AddToString** (UBYTE *, UBYTE *, int)
- UBYTE * **PreCalc** (VOID)
- UBYTE * **PreEval** (UBYTE *, LONG *)
- VOID **NumToStr** (UBYTE *, LONG)
- int **PreCmp** (int, int, UBYTE *, int, int, UBYTE *, int)
- int **PreEq** (int, int, UBYTE *, int, int, UBYTE *, int)
- UBYTE * **pParseObject** (UBYTE *, int *, LONG *)
- UBYTE * **PreIfEval** (UBYTE *, int *)
- int **EvalPreIf** (UBYTE *)
- int **PreLoad** (PRELOAD *, UBYTE *, UBYTE *, int, char *)
- int **PreSkip** (UBYTE *, UBYTE *, int)
- UBYTE * **EndOfToken** (UBYTE *)
- VOID **SetSpecialMode** (int, int)
- VOID **MakeGlobal** (VOID)
- int **ExecModule** (int)
- int **ExecStore** (VOID)
- VOID **FullCleanUp** (VOID)
- int **DoExecStatement** (VOID)
- int **DoPipeStatement** (VOID)
- int **DoPolyfun** (UBYTE *)
- int **DoPolyratfun** (UBYTE *)
- int **CompileStatement** (UBYTE *)
- UBYTE * **ToToken** (UBYTE *)
- int **GetDollar** (UBYTE *)
- int **MesWork** (VOID)
- int **MesPrint** (const char *,...)
- int **MesCall** (char *)
- UBYTE * **NumCopy** (WORD, UBYTE *)
- char * **LongCopy** (LONG, char *)
- char * **LongLongCopy** (off_t *, char *)
- VOID **ReserveTempFiles** (int)
- VOID **PrintTerm** (WORD *, char *)
- VOID **PrintTermC** (WORD *, char *)
- VOID **PrintSubTerm** (WORD *, char *)
- VOID **PrintWords** (WORD *, LONG)
- void **PrintSeq** (WORD *, char *)
- int **ExpandTripleDots** (int)
- LONG ComPress (WORD **, LONG *)
- VOID StageSort (FILEHANDLE *)
- void **M_free** (VOID *, const char *)
- void **ClearWildcardNames** (VOID)
- int **AddWildcardName** (UBYTE *)
- int **GetWildcardName** (UBYTE *)

- void **Globalize** (int)
- void **ResetVariables** (int)
- void **AddToPreTypes** (int)
- void **MessPreNesting** (int)
- LONG **GetStreamPosition** (STREAM *)
- WORD * DoubleCbuffer (int, WORD *, int)
- WORD * AddLHS (int)
- WORD * AddRHS (int, int)
- int AddNtoL (int, WORD *)
- int AddNtoC (int, int, WORD *, int)
- VOID **DoubleIfBuffers** (VOID)
- STREAM * **CreateStream** (UBYTE *)
- int **setonoff** (UBYTE *, int *, int, int)
- int **DoPrint** (UBYTE *, int)
- int **SetExpr** (UBYTE *, int, int)
- void **AddToCom** (int, WORD *)
- int **Add2ComStrings** (int, WORD *, UBYTE *, UBYTE *)
- int **DoSymmetrize** (UBYTE *, int)
- int **DoArgument** (UBYTE *, int)
- int **ArgFactorize** (PHEAD WORD *, WORD *)
- WORD * TakeArgContent (PHEAD WORD *, WORD *)
- WORD * MakeInteger (PHEAD WORD *, WORD *, WORD *)
- WORD * MakeMod (PHEAD WORD *, WORD *, WORD *)
- WORD FindArg (PHEAD WORD *)
- WORD InsertArg (PHEAD WORD *, WORD *, int)
- int CleanupArgCache (PHEAD WORD)
- int **ArgSymbolMerge** (WORD *, WORD *)
- int **ArgDotproductMerge** (WORD *, WORD *)
- void SortWeights (LONG *, LONG *, WORD)
- int **DoBrackets** (UBYTE *, int)
- int **DoPutInside** (UBYTE *, int)
- WORD * **CountComp** (UBYTE *, WORD *)
- int **CoAntiBracket** (UBYTE *)
- int **CoAntiSymmetrize** (UBYTE *)
- int **DoArgPlode** (UBYTE *, int)
- int **CoArgExplode** (UBYTE *)
- int **CoArgImplode** (UBYTE *)
- int **CoArgument** (UBYTE *)
- int **CoInside** (UBYTE *)
- int **ExecInside** (UBYTE *)
- int **CoInExpression** (UBYTE *)
- int **CoInParallel** (UBYTE *)
- int **CoNotInParallel** (UBYTE *)
- int **DoInParallel** (UBYTE *, int)
- int **CoEndInExpression** (UBYTE *)
- int **CoBracket** (UBYTE *)
- int **CoPutInside** (UBYTE *)
- int **CoAntiPutInside** (UBYTE *)
- int **CoMultiBracket** (UBYTE *)
- int **CoCFunction** (UBYTE *)
- int **CoCTensor** (UBYTE *)
- int **CoCollect** (UBYTE *)
- int **CoCompress** (UBYTE *)
- int **CoContract** (UBYTE *)
- int **CoCycleSymmetrize** (UBYTE *)

- int **CoDelete** (UBYTE *)
- int **CoTableBase** (UBYTE *)
- int **CoApply** (UBYTE *)
- int **CoDenominators** (UBYTE *)
- int **CoDimension** (UBYTE *)
- int **CoDiscard** (UBYTE *)
- int **CoDisorder** (UBYTE *)
- int **CoDrop** (UBYTE *)
- int **CoDropCoefficient** (UBYTE *)
- int **CoDropSymbols** (UBYTE *)
- int **CoElse** (UBYTE *)
- int **CoElseIf** (UBYTE *)
- int **CoEndArgument** (UBYTE *)
- int **CoEndInside** (UBYTE *)
- int **CoEndIf** (UBYTE *)
- int **CoEndRepeat** (UBYTE *)
- int **CoEndTerm** (UBYTE *)
- int **CoEndWhile** (UBYTE *)
- int **CoExit** (UBYTE *)
- int **CoFactArg** (UBYTE *)
- int **CoFactDollar** (UBYTE *)
- int **CoFactorize** (UBYTE *)
- int **CoNFactorize** (UBYTE *)
- int **CoUnFactorize** (UBYTE *)
- int **CoNUnFactorize** (UBYTE *)
- int **DoFactorize** (UBYTE *, int)
- int **CoFill** (UBYTE *)
- int **CoFillExpression** (UBYTE *)
- int **CoFixIndex** (UBYTE *)
- int **CoFormat** (UBYTE *)
- int **CoGlobal** (UBYTE *)
- int **CoGlobalFactorized** (UBYTE *)
- int **CoGoTo** (UBYTE *)
- int **CoId** (UBYTE *)
- int **CoIdNew** (UBYTE *)
- int **CoIdOld** (UBYTE *)
- int **CoIf** (UBYTE *)
- int **CoIfMatch** (UBYTE *)
- int **CoIfNoMatch** (UBYTE *)
- int **CoIndex** (UBYTE *)
- int **CoInsideFirst** (UBYTE *)
- int **CoKeep** (UBYTE *)
- int **CoLabel** (UBYTE *)
- int **CoLoad** (UBYTE *)
- int **CoLocal** (UBYTE *)
- int **CoLocalFactorized** (UBYTE *)
- int **CoMany** (UBYTE *)
- int **CoMerge** (UBYTE *)
- int **CoStuffle** (UBYTE *)
- int **CoMetric** (UBYTE *)
- int **CoModOption** (UBYTE *)
- int **CoModuleOption** (UBYTE *)
- int **CoModulus** (UBYTE *)
- int **CoMulti** (UBYTE *)
- int **CoMultiply** (UBYTE *)

- int **CoNFunction** (UBYTE *)
- int **CoNPrint** (UBYTE *)
- int **CoNTensor** (UBYTE *)
- int **CoNWrite** (UBYTE *)
- int **CoNoDrop** (UBYTE *)
- int **CoNoSkip** (UBYTE *)
- int **CoNormalize** (UBYTE *)
- int **CoMakeInteger** (UBYTE *)
- int **CoFlags** (UBYTE *, int)
- int **CoOff** (UBYTE *)
- int **CoOn** (UBYTE *)
- int **CoOnce** (UBYTE *)
- int **CoOnly** (UBYTE *)
- int **CoOptimizeOption** (UBYTE *)
- int **CoOptimize** (UBYTE *)
- int **CoPolyFun** (UBYTE *)
- int **CoPolyRatFun** (UBYTE *)
- int **CoPrint** (UBYTE *)
- int **CoPrintB** (UBYTE *)
- int **CoProperCount** (UBYTE *)
- int **CoUnitTrace** (UBYTE *)
- int **CoRCycleSymmetrize** (UBYTE *)
- int **CoRatio** (UBYTE *)
- int **CoRedefine** (UBYTE *)
- int **CoRenumber** (UBYTE *)
- int **CoRepeat** (UBYTE *)
- int **CoSave** (UBYTE *)
- int **CoSelect** (UBYTE *)
- int **CoSet** (UBYTE *)
- int **CoSetExitFlag** (UBYTE *)
- int **CoSkip** (UBYTE *)
- int **CoProcessBucket** (UBYTE *)
- int **CoPushHide** (UBYTE *)
- int **CoPopHide** (UBYTE *)
- int **CoHide** (UBYTE *)
- int **CoIntoHide** (UBYTE *)
- int **CoNoHide** (UBYTE *)
- int **CoUnHide** (UBYTE *)
- int **CoNoUnHide** (UBYTE *)
- int **CoSort** (UBYTE *)
- int **CoSplitArg** (UBYTE *)
- int **CoSplitFirstArg** (UBYTE *)
- int **CoSplitLastArg** (UBYTE *)
- int **CoSum** (UBYTE *)
- int **CoSymbol** (UBYTE *)
- int **CoSymmetrize** (UBYTE *)
- int **DoTable** (UBYTE *, int)
- int **CoTable** (UBYTE *)
- int **CoTerm** (UBYTE *)
- int **CoNTable** (UBYTE *)
- int **CoCTable** (UBYTE *)
- void **EmptyTable** (TABLES)
- int **CoToTensor** (UBYTE *)
- int **CoToVector** (UBYTE *)
- int **CoTrace4** (UBYTE *)

- int **CoTraceN** (UBYTE *)
- int **CoChisholm** (UBYTE *)
- int **CoTransform** (UBYTE *)
- int **CoClearTable** (UBYTE *)
- int **DoChain** (UBYTE *, int)
- int **CoChainin** (UBYTE *)
- int **CoChainout** (UBYTE *)
- int **CoTryReplace** (UBYTE *)
- int **CoVector** (UBYTE *)
- int **CoWhile** (UBYTE *)
- int **CoWrite** (UBYTE *)
- int **CoAuto** (UBYTE *)
- int **CoSwitch** (UBYTE *)
- int **CoCase** (UBYTE *)
- int **CoBreak** (UBYTE *)
- int **CoDefault** (UBYTE *)
- int **CoEndSwitch** (UBYTE *)
- int **CoTBaddto** (UBYTE *)
- int **CoTBaudit** (UBYTE *)
- int **CoTBcleanup** (UBYTE *)
- int **CoTBcreate** (UBYTE *)
- int **CoTBenter** (UBYTE *)
- int **CoTBhelp** (UBYTE *)
- int **CoTBload** (UBYTE *)
- int **CoTBoff** (UBYTE *)
- int **CoTBon** (UBYTE *)
- int **CoTBopen** (UBYTE *)
- int **CoTBreplace** (UBYTE *)
- int **CoTBuse** (UBYTE *)
- int **CoTestUse** (UBYTE *)
- int **CoThreadBucket** (UBYTE *)
- int **AddComString** (int, WORD *, UBYTE *, int)
- int **CompileAlgebra** (UBYTE *, int, WORD *)
- int **IsIdStatement** (UBYTE *)
- UBYTE * **IsRHS** (UBYTE *, UBYTE)
- int **ParenthesesTest** (UBYTE *)
- int **tokenize** (UBYTE *, WORD)
- void **WriteTokens** (SBYTE *)
- int **simp1token** (SBYTE *)
- int **simpwtoken** (SBYTE *)
- int **simp2token** (SBYTE *)
- int **simp3atoken** (SBYTE *, int)
- int **simp3btoken** (SBYTE *, int)
- int **simp4token** (SBYTE *)
- int **simp5token** (SBYTE *, int)
- int **simp6token** (SBYTE *, int)
- UBYTE * **SkipAName** (UBYTE *)
- int **TestTables** (VOID)
- int **GetLabel** (UBYTE *)
- int **ColdExpression** (UBYTE *, int)
- int **CoAssign** (UBYTE *)
- int **DoExpr** (UBYTE *, int, int)
- int **CompileSubExpressions** (SBYTE *)
- int **CodeGenerator** (SBYTE *)
- int **CompleteTerm** (WORD *, UWORD *, UWORD *, WORD, WORD, int)

- int **CodeFactors** (SBYTE *s)
- WORD **GenerateFactors** (WORD, WORD)
- int **InsTree** (int, int)
- int **FindTree** (int, WORD *)
- void **RedoTree** (CBUF *, int)
- void **ClearTree** (int)
- int **CatchDollar** (int)
- int **AssignDollar** (PHEAD WORD *, WORD)
- UBYTE * **WriteDollarToBuffer** (WORD, WORD)
- UBYTE * **WriteDollarFactorToBuffer** (WORD, WORD, WORD)
- void **AddToDollarBuffer** (UBYTE *)
- int **PutTermInDollar** (WORD *, WORD)
- void **TermAssign** (WORD *)
- void **WildDollars** (PHEAD WORD *)
- LONG **numcommute** (WORD *, LONG *)
- int **FullRenumber** (PHEAD WORD *, WORD)
- int **Lus** (WORD *, WORD, WORD, WORD, WORD, WORD)
- int **FindLus** (int, int, int)
- int **CoReplaceLoop** (UBYTE *)
- int **CoFindLoop** (UBYTE *)
- int **DoFindLoop** (UBYTE *, int)
- int **CoFunPowers** (UBYTE *)
- int **SortTheList** (int *, int)
- int **MatchIsPossible** (WORD *, WORD *)
- int **StudyPattern** (WORD *)
- WORD **DolToTensor** (PHEAD WORD)
- WORD **DolToFunction** (PHEAD WORD)
- WORD **DolToVector** (PHEAD WORD)
- WORD **DolToNumber** (PHEAD WORD)
- WORD **DolToSymbol** (PHEAD WORD)
- WORD **DolToIndex** (PHEAD WORD)
- LONG **DolToLong** (PHEAD WORD)
- int **DollarFactorize** (PHEAD WORD)
- int **CoPrintTable** (UBYTE *)
- int **CoDeallocateTable** (UBYTE *)
- void **CleanDollarFactors** (DOLLARS)
- WORD * **TakeDollarContent** (PHEAD WORD *, WORD **)
- WORD * MakeDollarInteger (PHEAD WORD *, WORD **)
- WORD * MakeDollarMod (PHEAD WORD *, WORD **)
- int **GetDolNum** (PHEAD WORD *, WORD *)
- void AddPotModdollar (WORD)
- int Optimize (WORD, int)
- int **ClearOptimize** (VOID)
- int **LoadOpti** (WORD)
- int **PutObject** (WORD *, int)
- void **CleanOptiBuffer** (VOID)
- int **PrintOptima** (WORD)
- int **FindScratchName** (VOID)
- WORD **MaxPowerOpti** (LONG)
- WORD **HuntNumFactor** (LONG, WORD *, int)
- WORD **HuntFactor** (LONG, WORD *, int)
- void **HuntPairs** (LONG, WORD)
- void **HuntBrackets** (LONG)
- int **AddToOpti** (WORD *, int)
- LONG **TestNewSca** (LONG, WORD *, WORD *)

- void **NormOpti** (WORD *)
- void **SortOpti** (LONG)
- void **SplitOpti** (WORD **, LONG)
- void **CombiOpti** (VOID)
- int **TakeLongRoot** (UWORD *, WORD *, WORD)
- int **TakeRatRoot** (UWORD *, WORD *, WORD)
- int **MakeRational** (WORD, WORD, WORD *, WORD *)
- int **MakeLongRational** (PHEAD UWORD *, WORD, UWORD *, WORD, UWORD *, WORD *)
- void **HuntPowers** (LONG, WORD)
- void **HuntNumBrackets** (LONG)
- void **ClearTableTree** (TABLES)
- int **InsTableTree** (TABLES, WORD *)
- void **RedoTableTree** (TABLES, int)
- int **FindTableTree** (TABLES, WORD *, int)
- void finishcbuf (WORD)
- void clearcbuf (WORD)
- void CleanUpSort (int)
- FILEHANDLE * **AllocFileHandle** (WORD, char *)
- VOID **DeAllocFileHandle** (FILEHANDLE *)
- VOID **LowerSortLevel** (VOID)
- WORD * **PolyRatFunSpecial** (PHEAD WORD *, WORD *)
- VOID **SimpleSplitMergeRec** (WORD *, WORD, WORD *)
- VOID **SimpleSplitMerge** (WORD *, WORD)
- WORD **BinarySearch** (WORD *, WORD, WORD)
- int **InsideDollar** (PHEAD WORD *, WORD)
- DOLLARS **DolToTerms** (PHEAD WORD)
- WORD EvalDoLoopArg (PHEAD WORD *, WORD)
- int **SetExprCases** (int, int, int)
- int **TestSelect** (WORD *, WORD *)
- VOID **SubsInAll** (PHEAD0)
- VOID **TransferBuffer** (int, int, int)
- int **TakeIDfunction** (PHEAD WORD *)
- int **MakeSetupAllocs** (VOID)
- int **TryFileSetups** (VOID)
- void **ExchangeExpressions** (int, int)
- void **ExchangeDollars** (int, int)
- int **GetFirstBracket** (WORD *, int)
- int **GetFirstTerm** (WORD *, int)
- int **GetContent** (WORD *, int)
- int **CleanupTerm** (WORD *)
- WORD **ContentMerge** (PHEAD WORD *, WORD *)
- UBYTE * **PrelfDollarEval** (UBYTE *, int *)
- LONG **TermsInDollar** (WORD)
- LONG **SizeOfDollar** (WORD)
- LONG **TermsInExpression** (WORD)
- LONG **SizeOfExpression** (WORD)
- WORD * **TranslateExpression** (UBYTE *)
- int **IsSetMember** (WORD *, WORD)
- int **IsMultipleOf** (WORD *, WORD *)
- int **TwoExprCompare** (WORD *, WORD *, int)
- void **UpdatePositions** (VOID)
- void **M_check** (VOID)
- void **M_print** (VOID)
- void **M_check1** (VOID)
- void **PrintTime** (UBYTE *)

- POSITION * **FindBracket** (WORD, WORD *)
- VOID **PutBracketInIndex** (PHEAD WORD *, POSITION *)
- void **ClearBracketIndex** (WORD)
- VOID **OpenBracketIndex** (WORD)
- int **DoNoParallel** (UBYTE *)
- int **DoParallel** (UBYTE *)
- int **DoModSum** (UBYTE *)
- int **DoModMax** (UBYTE *)
- int **DoModMin** (UBYTE *)
- int **DoModLocal** (UBYTE *)
- UBYTE * **DoModDollar** (UBYTE *, int)
- int **DoProcessBucket** (UBYTE *)
- int **DoinParallel** (UBYTE *)
- int **DonotinParallel** (UBYTE *)
- int **FlipTable** (FUNCTIONS, int)
- int **ChainIn** (PHEAD WORD *, WORD)
- int **ChainOut** (PHEAD WORD *, WORD)
- int **ArgumentImplode** (PHEAD WORD *, WORD *)
- int **ArgumentExplode** (PHEAD WORD *, WORD *)
- int **DenToFunction** (WORD *, WORD)
- WORD **HowMany** (PHEAD WORD *, WORD *)
- VOID **RemoveDollars** (VOID)
- LONG **CountTerms1** (PHEAD0)
- LONG **TermsInBracket** (PHEAD WORD *, WORD)
- int **Crash** (VOID)
- char * **str_dup** (char *)
- void **convertblock** (INDEXBLOCK *, INDEXBLOCK *, int)
- void **convertnamesblock** (NAMESBLOCK *, NAMESBLOCK *, int)
- void **convertiniinfo** (INIINFO *, INIINFO *, int)
- int **ReadIndex** (DBASE *)
- int **WriteIndexBlock** (DBASE *, MLONG)
- int **WriteNamesBlock** (DBASE *, MLONG)
- int **WriteIndex** (DBASE *)
- int **WriteIniInfo** (DBASE *)
- int **ReadIniInfo** (DBASE *)
- int **AddToIndex** (DBASE *, MLONG)
- DBASE * **GetDbase** (char *)
- DBASE * **OpenDbase** (char *)
- char * **ReadObject** (DBASE *, MLONG, char *)
- char * **ReadijObject** (DBASE *, MLONG, MLONG, char *)
- int **ExistsObject** (DBASE *, MLONG, char *)
- int **DeleteObject** (DBASE *, MLONG, char *)
- int **WriteObject** (DBASE *, MLONG, char *, char *, MLONG)
- MLONG **AddObject** (DBASE *, MLONG, char *, char *)
- int **Cleanup** (DBASE *)
- DBASE * **NewDbase** (char *, MLONG)
- void **FreeTableBase** (DBASE *)
- int **ComposeTableNames** (DBASE *)
- int **PutTableNames** (DBASE *)
- MLONG **AddTableName** (DBASE *, char *, TABLES)
- MLONG **GetTableName** (DBASE *, char *)
- MLONG **FindTableNumber** (DBASE *, char *)
- int **TryEnvironment** (VOID)
- int **CopyExpression** (FILEHANDLE *, FILEHANDLE *)
- int **set_in** (UBYTE, set_of_char)

- one_byte **set_set** (UBYTE, set_of_char)
- one_byte **set_del** (UBYTE, set_of_char)
- one_byte **set_sub** (set_of_char, set_of_char, set_of_char)
- int **DoPreAddSeparator** (UBYTE *)
- int **DoPreRmSeparator** (UBYTE *)
- int **openExternalChannel** (UBYTE *, int, UBYTE *, UBYTE *)
- int **initPresetExternalChannels** (UBYTE *, int)
- int **closeExternalChannel** (int)
- int **selectExternalChannel** (int)
- int **getCurrentExternalChannel** (VOID)
- VOID **closeAllExternalChannels** (VOID)
- UBYTE * **defineChannel** (UBYTE *, HANDLERS *)
- int **writeToChannel** (int, UBYTE *, HANDLERS *)
- int **writeBufToExtChannelOk** (char *, size_t)
- int **getcFromExtChannelOk** (VOID)
- int **setKillModeForExternalChannelOk** (int, int)
- int **setTerminatorForExternalChannelOk** (char *)
- int **getcFromExtChannelFailure** (VOID)
- int **setKillModeForExternalChannelFailure** (int, int)
- int **setTerminatorForExternalChannelFailure** (char *)
- int **writeBufToExtChannelFailure** (char *, size_t)
- int **ReleaseTB** (VOID)
- int SymbolNormalize (WORD *)
- int **TestFunFlag** (PHEAD WORD *)
- WORD CompareSymbols (WORD *, WORD *, WORD)
- WORD CompareHSymbols (WORD *, WORD *, WORD)
- WORD NextPrime (PHEAD WORD)
- UWORD **wranf** (PHEAD0)
- UWORD **iranf** (PHEAD UWORD)
- void **iniwranf** (PHEAD0)
- UBYTE * **PreRandom** (UBYTE *)
- WORD * **PolyNormPoly** (PHEAD WORD)
- WORD * **EvaluateGcd** (PHEAD WORD *)
- int **TreatPolyRatFun** (PHEAD WORD *)
- WORD **ReadSaveHeader** (VOID)
- WORD ReadSaveIndex (FILEINDEX *)
- WORD ReadSaveExpression (UBYTE *, UBYTE *, LONG *, LONG *)
- UBYTE * ReadSaveTerm32 (UBYTE *, UBYTE *, UBYTE **, UBYTE *, UBYTE *, int)
- WORD ReadSaveVariables (UBYTE *, UBYTE *, LONG *, LONG *, INDEXENTRY *, LONG *)
- WORD WriteStoreHeader (WORD)
- void **InitRecovery** (VOID)
- int **CheckRecoveryFile** (VOID)
- void **DeleteRecoveryFile** (VOID)
- char * **RecoveryFilename** (VOID)
- int DoRecovery (int *)
- void DoCheckpoint (int)
- VOID **NumberMallocAddMemory** (PHEAD0)
- VOID **CacheNumberMallocAddMemory** (PHEAD0)
- VOID **TermMallocAddMemory** (PHEAD0)
- void **ExprStatus** (EXPRESSIONS)
- VOID **iniTools** (VOID)
- int TestTerm (WORD *)
- WORD **RunTransform** (PHEAD WORD *term, WORD *params)
- WORD **RunEncode** (PHEAD WORD *fun, WORD *args, WORD *info)
- WORD **RunDecode** (PHEAD WORD *fun, WORD *args, WORD *info)

- WORD **RunReplace** (PHEAD WORD *fun, WORD *args, WORD *info)
- WORD **RunImplode** (WORD *fun, WORD *args)
- WORD **RunExplode** (PHEAD WORD *fun, WORD *args)
- int **TestArgNum** (int n, int totarg, WORD *args)
- WORD **PutArgInScratch** (WORD *arg, UWORD *scrat)
- UBYTE * **ReadRange** (UBYTE *s, WORD *out, int par)
- int **FindRange** (PHEAD WORD *, WORD *, WORD *, WORD)
- WORD **RunPermute** (PHEAD WORD *fun, WORD *args, WORD *info)
- WORD **RunReverse** (PHEAD WORD *fun, WORD *args)
- WORD **RunCycle** (PHEAD WORD *fun, WORD *args, WORD *info)
- WORD **RunAddArg** (PHEAD WORD *fun, WORD *args)
- WORD **RunMulArg** (PHEAD WORD *fun, WORD *args)
- WORD **RunIsLyndon** (PHEAD WORD *fun, WORD *args, int par)
- WORD **RunToLyndon** (PHEAD WORD *fun, WORD *args, int par)
- WORD **RunDropArg** (PHEAD WORD *fun, WORD *args)
- WORD **RunSelectArg** (PHEAD WORD *fun, WORD *args)
- WORD **RunDedup** (PHEAD WORD *fun, WORD *args)
- int **NormPolyTerm** (PHEAD WORD *)
- WORD **ComparePoly** (WORD *, WORD *, WORD)
- int **ConvertToPoly** (PHEAD WORD *, WORD *, WORD *, WORD)
- int LocalConvertToPoly (PHEAD WORD *, WORD *, WORD, WORD)
- int **ConvertFromPoly** (PHEAD WORD *, WORD *, WORD, WORD, WORD, WORD)
- WORD **FindSubterm** (WORD *)
- WORD **FindLocalSubterm** (PHEAD WORD *, WORD)
- void **PrintSubtermList** (int, int)
- void **PrintExtraSymbol** (int, WORD *, int)
- WORD **FindSubexpression** (WORD *)
- void **UpdateMaxSize** (VOID)
- int **CoToPolynomial** (UBYTE *)
- int **CoFromPolynomial** (UBYTE *)
- int **CoArgToExtraSymbol** (UBYTE *)
- int **CoExtraSymbols** (UBYTE *)
- UBYTE * **GetDoParam** (UBYTE *, WORD **, int)
- WORD * **GetIfDollarFactor** (UBYTE **, WORD *)
- int **CoDo** (UBYTE *)
- int **CoEndDo** (UBYTE *)
- int **ExtraSymFun** (PHEAD WORD *, WORD)
- int **PruneExtraSymbols** (WORD)
- int IniFbuffer (WORD)
- void **IniFbufs** (VOID)
- int **GCDfunction** (PHEAD WORD *, WORD)
- WORD * **GCDfunction3** (PHEAD WORD *, WORD *)
- WORD * **GCDfunction4** (PHEAD WORD *, WORD *)
- int **ReadPolyRatFun** (PHEAD WORD *)
- int **FromPolyRatFun** (PHEAD WORD *, WORD **, WORD **)
- void **PRFnormalize** (PHEAD WORD *)
- WORD * **PRFadd** (PHEAD WORD *, WORD *)
- WORD * **PolyDiv** (PHEAD WORD *, WORD *, char *)
- WORD * **PolyGCD** (PHEAD WORD *, WORD *)
- WORD * **PolyAdd** (PHEAD WORD *, WORD *)
- void **GCDclean** (PHEAD WORD *, WORD *)
- int **RatFunNormalize** (PHEAD WORD *)
- WORD * TakeSymbolContent (PHEAD WORD *, WORD *)
- int **GCDterms** (PHEAD WORD *, WORD *, WORD *)
- WORD * **PutExtraSymbols** (PHEAD WORD *, WORD, int *)

- WORD * **TakeExtraSymbols** (PHEAD WORD *, WORD)
- WORD * **MultiplyWithTerm** (PHEAD WORD *, WORD *, WORD)
- WORD * TakeContent (PHEAD WORD *, WORD *)
- int **MergeSymbolLists** (PHEAD WORD *, WORD *, int)
- int **MergeDotproductLists** (PHEAD WORD *, WORD *, int)
- WORD * **CreateExpression** (PHEAD WORD)
- int **DIVfunction** (PHEAD WORD *, WORD, int)
- WORD * **MULfunc** (PHEAD WORD *, WORD *)
- WORD * **ConvertArgument** (PHEAD WORD *, int *)
- int **ExpandRat** (PHEAD WORD *)
- int **InvPoly** (PHEAD WORD *, WORD, WORD)
- WORD **TestDoLoop** (PHEAD WORD *, WORD)
- WORD **TestEndDoLoop** (PHEAD WORD *, WORD)
- WORD * poly_gcd (PHEAD WORD *, WORD *, WORD)
- WORD * **poly_div** (PHEAD WORD *, WORD *, WORD)
- WORD * **poly_rem** (PHEAD WORD *, WORD *, WORD)
- WORD * **poly_inverse** (PHEAD WORD *, WORD *)
- WORD * **poly_mul** (PHEAD WORD *, WORD *)
- WORD * poly_ratfun_add (PHEAD WORD *, WORD *)
- int poly_ratfun_normalize (PHEAD WORD *)
- int poly_factorize_argument (PHEAD WORD *, WORD *)
- WORD * poly_factorize_dollar (PHEAD WORD *)
- int poly_factorize_expression (EXPRESSIONS)
- int **poly_unfactorize_expression** (EXPRESSIONS)
- void **poly_free_poly_vars** (PHEAD const char *)
- VOID optimize_print_code (int)
- int **DoPreAdd** (UBYTE *s)
- int **DoPreUseDictionary** (UBYTE *s)
- int **DoPreCloseDictionary** (UBYTE *s)
- int **DoPreOpenDictionary** (UBYTE *s)
- void **RemoveDictionary** (DICTIONARY *dict)
- void **UnSetDictionary** (VOID)
- int **SetDictionaryOptions** (UBYTE *options)
- int **SelectDictionary** (UBYTE *name, UBYTE *options)
- int **AddToDictionary** (DICTIONARY *dict, UBYTE *left, UBYTE *right)
- int **AddDictionary** (UBYTE *name)
- int **FindDictionary** (UBYTE *name)
- UBYTE * **IsExponentSign** (VOID)
- UBYTE * **IsMultiplySign** (VOID)
- VOID **TransformRational** (UWORD *a, WORD na)
- void **WriteDictionary** (DICTIONARY *)
- void **ShrinkDictionary** (DICTIONARY *)
- void **MultiplyToLine** (VOID)
- UBYTE * **FindSymbol** (WORD num)
- UBYTE * **FindVector** (WORD num)
- UBYTE * **FindIndex** (WORD num)
- UBYTE * **FindFunction** (WORD num)
- UBYTE * **FindFunWithArgs** (WORD *t)
- UBYTE * **FindExtraSymbol** (WORD num)
- LONG **DictToBytes** (DICTIONARY *dict, UBYTE *buf)
- DICTIONARY * **DictFromBytes** (UBYTE *buf)
- int **CoCreateSpectator** (UBYTE *inp)
- int **CoToSpectator** (UBYTE *inp)
- int **CoRemoveSpectator** (UBYTE *inp)
- int **CoEmptySpectator** (UBYTE *inp)

- int **CoCopySpectator** (UBYTE *inp)
- int **PutInSpectator** (WORD *, WORD)
- void **ClearSpectators** (WORD)
- WORD **GetFromSpectator** (WORD *, WORD)
- void **FlushSpectators** (VOID)
- WORD * **PreGCD** (PHEAD WORD *, WORD *, int)
- WORD * **FindCommonVariables** (PHEAD int, int)
- VOID **AddToSymbolList** (PHEAD WORD)
- int **AddToListPoly** (PHEAD0)
- int **ReadFromScratch** (FILEHANDLE *, POSITION *, UBYTE *, POSITION *)
- int **AddToScratch** (FILEHANDLE *, POSITION *, UBYTE *, POSITION *, int)
- int DoPreAppendPath (UBYTE *)
- int DoPrePrependPath (UBYTE *)
- int **DoSwitch** (PHEAD WORD *, WORD *)
- int **DoEndSwitch** (PHEAD WORD *, WORD *)
- SWITCHTABLE * **FindCase** (WORD, WORD)
- VOID **SwitchSplitMergeRec** (SWITCHTABLE *, WORD, SWITCHTABLE *)
- VOID **SwitchSplitMerge** (SWITCHTABLE *, WORD)
- int **DoubleSwitchBuffers** (VOID)
- int **DistrN** (int, int *, int, int *)

4.10.1 Detailed Description

Contains macros and function declarations.

4.10.2 Macro Definition Documentation

4.10.2.1 TOKENTOLINE

```
#define TOKENTOLINE(
                x,
                y )
```

Value:

```
if ( AC.OutputSpaces == NOSPACEFORMAT ) { \
TokenToLine((UBYTE *)(y)); } else { TokenToLine((UBYTE *)(x)); }
```

Definition at line 53 of file declare.h.

4.10.2.2 ParseSign

```
#define ParseSign(
                sgn,
                s )
```

Value:

```
{(sgn)=0;while(*(s)=='-'||*(s)=='+'){\
if ( *(s)++ == '-' ) sgn ^= 1;}}
```

Definition at line 65 of file declare.h.

4.10.2.3 ParseSignedNumber

```
#define ParseSignedNumber(
              x,
              s )
```

Value:

```
{ int sgn; ParseSign(sgn,s)\
ParseNumber(x,s) if ( sgn ) x = -x; }
```

Definition at line 67 of file declare.h.

4.10.2.4 NeedNumber

```
#define NeedNumber(
              x,
              s,
              err )
```

Value:

```
{ int sgn = 1;                                    \
while ( *s == ' ' || *s == '\t' || *s == '-' || *s == '+' ) {     \
    if ( *s == '-' ) sgn = -sgn; s++; }                           \
if ( chartype[*s] != 1 ) goto err;                                \
ParseNumber(x,s)                                                  \
if ( sgn < 0 ) (x) = -(x); while ( *s == ' ' || *s == '\t' ) s++;\
}
```

Definition at line 77 of file declare.h.

4.10.2.5 SKIPBRA1

```
#define SKIPBRA1(
              s )
```

Value:

```
{ int lev1=0; s++; while(*s) { if(*s=='[')lev1++; \
else if(*s==']'&&--lev1<0)break; s++;} }
```

Definition at line 87 of file declare.h.

4.10.2.6 SKIPBRA2

```
#define SKIPBRA2(
              s )
```

Value:

```
{ int lev2=0; s++; while(*s) { if(*s=='{')lev2++; \
else if(*s=='}'&&--lev2<0)break; \
else if(*s=='[')SKIPBRA1(s) s++;} }
```

Definition at line 89 of file declare.h.

4.10.2.7 SKIPBRA3

```
#define SKIPBRA3(
                s )
```

Value:

```
{ int lev3=0; s++; while(*s) { if(*s=='(')lev3++; \
else if(*s==')'&&--lev3<0)break; \
else if(*s=='{')SKIPBRA2(s) \
else if(*s=='[')SKIPBRA1(s) s++;} }
```

Definition at line 92 of file declare.h.

4.10.2.8 SKIPBRA4

```
#define SKIPBRA4(
                s )
```

Value:

```
{ int lev4=0; s++; while(*s) { if(*s=='(')lev4++; \
else if(*s==')'&&--lev4<0)break; \
else if(*s=='[')SKIPBRA1(s) s++;} }
```

Definition at line 96 of file declare.h.

4.10.2.9 SKIPBRA5

```
#define SKIPBRA5(
                s )
```

Value:

```
{ int lev5=0; s++; while(*s) { if(*s=='{')lev5++; \
else if(*s=='}'&&--lev5<0)break; \
else if(*s=='(')SKIPBRA4(s) \
else if(*s=='[')SKIPBRA1(s) s++;} }
```

Definition at line 99 of file declare.h.

4.10.2.10 AddToCB

```
#define AddToCB(
                c,
                wx )
```

Value:

```
if(c->Pointer>=c->Top) \
DoubleCbuffer(c->cbuf,c->Pointer,21); \
*(c->Pointer)++ = wx;
```

Definition at line 109 of file declare.h.

4.10.2.11 EXCHINOUT

```
#define EXCHINOUT
```

Value:

```
    { FILEHANDLE *ffFi = AR.outfile; \
    AR.outfile = AR.infile; AR.infile = ffFi; }
```

Definition at line 113 of file declare.h.

4.10.2.12 BACKINOUT

```
#define BACKINOUT
```

Value:

```
    { FILEHANDLE *ffFi = AR.outfile; POSITION posi; \
    AR.outfile = AR.infile; AR.infile = ffFi; \
    SetEndScratch(AR.infile,&posi); }
```

Definition at line 115 of file declare.h.

4.10.2.13 CopyArg

```
#define CopyArg(
                to,
                from )
```

Value:

```
        { if ( *from > 0 ) { int ica = *from; NCOPY(to,from,ica) } \
        else if ( *from <= -FUNCTION ) *to++ = *from++;  \
        else { *to++ = *from++; *to++ = *from++; } }
```

Definition at line 119 of file declare.h.

4.10.2.14 COPY1ARG

```
#define COPY1ARG(
                s1,
                t1 )
```

Value:

```
        { int ica; if ( (ica=*t1) > 0 ) { NCOPY(s1,t1,ica) } \
        else if(*t1<=-FUNCTION){*s1++=*t1++;} else{*s1++=*t1++;*s1++=*t1++;} }
```

Definition at line 164 of file declare.h.

4.10.2.15 ZeroFillRange

```
#define ZeroFillRange(
                w,
                begin,
                end )
```

Value:

```
    do { \
    int tmp_i; \
    for ( tmp_i = begin; tmp_i < end; tmp_i++ ) { (w)[tmp_i] = 0; } \
} while (0)
```

Fills a buffer by zero in the range [begin,end).

Parameters

w	The buffer.
begin	The index for the beginning of the range.
end	The index for the end of the range (exclusive).

Definition at line 174 of file declare.h.

4.10.2.16 Add4Com

```
#define Add4Com(
                x1,
                x2,
                x3 )
```

Value:

```
{ WORD cod[4]; cod[0] = x1; cod[1] = 4; \
cod[2] = x2; cod[3] = x3; AddNtoL(4,cod); }
```

Definition at line 248 of file declare.h.

4.10.2.17 Add5Com

```
#define Add5Com(
                x1,
                x2,
                x3,
                x4 )
```

Value:

```
{ WORD cod[5]; cod[0] = x1; cod[1] = 5; \
cod[2] = x2; cod[3] = x3; cod[4] = x4; AddNtoL(5,cod); }
```

Definition at line 250 of file declare.h.

4.10.2.18 WantAddPointers

```
#define WantAddPointers(
                x )
```

Value:

```
 while((AT.pWorkPointer+(x))>AR.pWorkSize){WORD ***ppp=&AT.pWorkSpace;\
 ExpandBuffer((void **)ppp,&AR.pWorkSize,(int)(sizeof(WORD *)));}
```

Definition at line 256 of file declare.h.

4.10.2.19 WantAddLongs

```
#define WantAddLongs(
              x )
```

Value:

```
    while((AT.lWorkPointer+(x))>AR.lWorkSize){LONG **ppp=&AT.lWorkSpace;\
    ExpandBuffer((void **)ppp,&AR.lWorkSize,sizeof(LONG));}
```

Definition at line 258 of file declare.h.

4.10.2.20 WantAddPositions

```
#define WantAddPositions(
              x )
```

Value:

```
    while((AT.posWorkPointer+(x))>AR.posWorkSize){POSITION **ppp=&AT.posWorkSpace;\
    ExpandBuffer((void **)ppp,&AR.posWorkSize,sizeof(POSITION));}
```

Definition at line 260 of file declare.h.

4.10.2.21 MarkPolyRatFunDirty

```
#define MarkPolyRatFunDirty(
              T )
```

Value:

```
{if(*T&&AR.PolyFunType==2){WORD *TP,*TT;TT=T+*T;TT-=ABS(TT[-1]);\
TP=T+1;while(TP<TT){if(*TP==AR.PolyFun){TP[2]|=(DIRTYFLAG|MUSTCLEANPRF);}TP+=TP[1];}}}
```

Definition at line 312 of file declare.h.

4.10.2.22 PUSHPREASSIGNLEVEL

```
#define PUSHPREASSIGNLEVEL
```

Value:

```
    AP.PreAssignLevel++; { GETIDENTITY \
    if ( AP.PreAssignLevel >= AP.MaxPreAssignLevel ) { int i; \
        LONG *ap = (LONG *)Malloc1(2*AP.MaxPreAssignLevel*sizeof(LONG *),"PreAssignStack"); \
        for ( i = 0; i < AP.MaxPreAssignLevel; i++ ) ap[i] = AP.PreAssignStack[i]; \
        M_free(AP.PreAssignStack,"PreAssignStack"); \
        AP.MaxPreAssignLevel *= 2; AP.PreAssignStack = ap; \
    } \
    *AT.WorkPointer++ = AP.PreContinuation; AP.PreContinuation = 0; \
    AP.PreAssignStack[AP.PreAssignLevel] = AC.iPointer - AC.iBuffer; }
```

Definition at line 319 of file declare.h.

4.10.2.23 POPPREASSIGNLEVEL

```
#define POPPREASSIGNLEVEL
```

Value:

```
    if ( AP.PreAssignLevel > 0 ) { GETIDENTITY \
    AC.iPointer = AC.iBuffer + AP.PreAssignStack[AP.PreAssignLevel--]; \
    AP.PreContinuation = *--AT.WorkPointer; \
    *AC.iPointer = 0; }
```

Definition at line 329 of file declare.h.

4.10.2.24 EXTERNLOCK

```
#define EXTERNLOCK(
              x )
```

NOTE: We have replaced LOCK(ErrorMessageLock) and UNLOCK(ErrorMessageLock) by MLOCK(Error↩ MessageLock) and MUNLOCK(ErrorMessageLock). They are used for the synchronised output in ParFORM. (TU 28 May 2011)

Definition at line 445 of file declare.h.

4.10.3 Function Documentation

4.10.3.1 StartVariables()

```
VOID StartVariables ( )
```

All functions (well, nearly all) are declared here.

Definition at line 840 of file startup.c.

4.10.3.2 PasteTerm()

```
WORD* PasteTerm (
             PHEAD WORD number,
             WORD * accum,
             WORD * position,
             WORD times,
             WORD divby )
```

Puts the term at position in the accumulator accum at position 'number+1'. if times $>$ 0 the coefficient of this term is multiplied by times/divby.

Parameters

number	The number of term fragments in accum that should be skipped
accum	The accumulator of term fragments
position	A position in (typically) a compiler buffer from where a (piece of a) term comes.
times	Multiply the result by this
divby	Divide the result by this.

This routine is typically used when we have to replace a (sub)expression pointer by a power of a (sub)expression. This uses mostly a binomial expansion and the new term is the old term multiplied one by one by terms of the new expression. The factors times and divby keep track of the binomial coefficient. Once this is complete, the routine FiniTerm will make the contents of the accumulator into a proper term that still needs to be normalized.

Definition at line 2837 of file proces.c.

4.10.3.3 AddArgs()

```
VOID AddArgs (
            PHEAD WORD * s1,
            WORD * s2,
            WORD * m )
```

Adds the arguments of two occurrences of the PolyFun.

Parameters

s1	Pointer to the first occurrence.
s2	Pointer to the second occurrence.
m	Pointer to where the answer should be.

Definition at line 2251 of file sort.c.

4.10.3.4 AddCoef()

```
WORD AddCoef (
            PHEAD WORD ** ps1,
            WORD ** ps2 )
```

Adds the coefficients of the terms ∗ps1 and ∗ps2. The problem comes when there is not enough space for a new longer coefficient. First a local solution is tried. If this is not succesfull we need to move terms around. The possibility of a garbage collection should not be ignored, as avoiding this costs very much extra space which is nearly wasted otherwise.

If the return value is zero the terms cancelled.

The resulting term is left in ∗ps1.

Definition at line 1962 of file sort.c.

4.10.3.5 AddPoly()

```
WORD AddPoly (
             PHEAD WORD ** ps1,
             WORD ** ps2 )
```

Routine should be called when S->PolyWise != 0. It points then to the position of AR.PolyFun in both terms.

We add the contents of the arguments of the two polynomials. Special attention has to be given to special arguments. We have to reserve a space equal to the size of one term + the size of the argument of the other. The addition has to be done in this routine because not all objects are reentrant.

Newer addition (12-nov-2007). The PolyFun can have two arguments. In that case S->PolyFlag is 2 and we have to call the routine for adding rational polynomials. We have to be rather careful what happens with: The location of the output The order of the terms in the arguments At first we allow only univariate polynomials in the PolyFun. This restriction will be lifted a.s.a.p.

Parameters

ps1	A pointer to the postion of the first term
ps2	A pointer to the postion of the second term

Returns

If zero the terms cancel. Otherwise the new term is in ∗ps1.

Definition at line 2089 of file sort.c.

4.10.3.6 CompCoef()

```
WORD CompCoef (
             WORD * term1,
             WORD * term2 )
```

```
Routine takes a1 mod m1 and a2 mod m2 and returns a mod m1*m2 with
a mod m1 = a1 and a mod m2 = a2
```

Chinese remainder: a%(m1∗m2) = q1∗m1+a1 a%(m1∗m2) = q2∗m2+a2 Compute n1 such that (n1∗m1)m2 is one Compute n2 such that (n2∗m2)m1 is one Then (a1∗n2∗m2+a2∗n1∗m1)%(m1∗m2) is a%(m1∗m2)

Definition at line 3037 of file reken.c.

4.10.3.7 Compare1()

```
WORD Compare1 (
             WORD * term1,
             WORD * term2,
             WORD level )
```

Compares two terms. The answer is: 0 equal (with exception of the coefficient if level == 0.) >0 term1 comes first. <0 term2 comes first. Some special precautions may be needed to keep the CompCoef routine from generating overflows, although this is very unlikely in subterms. This routine should not return an error condition.

Originally this routine was called Compare. With the treatment of special polynomials with terms that contain only symbols and the need for extreme speed for the polynomial routines we made a special compare routine and now we store the address of the current compare routine in AR.CompareRoutine and have a macro Compare which makes all existing code work properly and we can just replace the routine on a thread by thread basis (each thread has its own AR struct).

Parameters

term1	First input term
term2	Second input term
level	The sorting level (may influence on the result)

Returns

0 equal (with exception of the coefficient if level == 0.) >0 term1 comes first. <0 term2 comes first.

Definition at line 2536 of file sort.c.

4.10.3.8 Deferred()

```
WORD Deferred (
             PHEAD WORD * term,
             WORD level )
```

Picks up the deferred brackets. These are the bracket contents of which we postpone the reading when we use the 'Keep Brackets' statement. These contents are multiplying the terms just before they are sent to the sorting system. Special attention goes to having it thread-safe We have to lock positioning the file and reading it in a thread specific buffer.

Parameters

term	The term that must be multiplied by the contents of the current bracket
level	The compiler level. This is needed because after multiplying term by term we call Generator again.

Definition at line 4616 of file proces.c.

4.10.3.9 DoOnePow()

```
WORD DoOnePow (
              PHEAD WORD * term,
              WORD power,
              WORD nexp,
              WORD * accum,
              WORD * aa,
              WORD level,
              WORD * freeze )
```

Routine gets one power of an expression in the scratch system. If there are more powers needed there will be a recursion.

No attempt is made to use binomials because we have no information about commutating properties.

There is a searching for the contents of brackets if needed. This searching may be rather slow because of the single links.

Parameters

term	is the term we are adding to.
power	is the power of the expression that we need.
nexp	is the number of the expression.
accum	is the accumulator of terms. It accepts the termfragments that are made into a proper term in FiniTerm
aa	points to the start of the entire accumulator. In $*$aa we store the number of term fragments that are in the accumulator.
level	is the current depth in the tree of statements. It is needed to continue to the next operation/substitution with each generated term
freeze	is the pointer to the bracket information that should be matched.

Definition at line 4395 of file proces.c.

4.10.3.10 EndSort()

```
LONG EndSort (
              PHEAD WORD * buffer,
              int par )
```

Finishes a sort. At AR.sLevel == 0 the output is to the regular output stream. When AR.sLevel $>$ 0, the parameter par determines the actual output. The AR.sLevel will be popped. All ongoing stages are finished and if the sortfile is open it is closed. The statistics are printed when AR.sLevel == 0 par == 0 Output to the buffer. par == 1 Sort for function arguments. The output will be copied into the buffer. It is assumed that this is in the WorkSpace. par == 2 Sort for $-variable. We return the address of the buffer that contains the output in buffer (treated like WORD $**$). We first catch the output in a file (unless we can intercept things after the small buffer has been sorted) Then we read from the file into a buffer. Only when par == 0 data compression can be attempted at AT.SS==AT.S0.

Parameters

buffer	buffer for output when needed
par	See above

Returns

If negative: error. If positive: number of words in output.

Definition at line 682 of file sort.c.

4.10.3.11 FiniTerm()

```
WORD FiniTerm (
              PHEAD WORD * term,
              WORD * accum,
              WORD * termout,
              WORD number,
              WORD tepos )
```

Concatenates the contents of the accumulator into a single legal term, which replaces the subexpression pointer

Parameters

term	the input term with the (sub)expression subterm
accum	the accumulator with the term fragments
termout	the location where the output should be written
number	the number of term fragments in the accumulator
tepos	the position of the subterm in term to be replaced

Definition at line 2902 of file proces.c.

4.10.3.12 FlushOut()

```
WORD FlushOut (
              POSITION * position,
              FILEHANDLE * fi,
              int compr )
```

Completes output to an output file and writes the trailing zero.

Parameters

position	The position in the file after writing
fi	The file (or its cache)
compr	Indicates whether there should be compression with gzip.

Returns

Regular conventions (OK -> 0).

Definition at line 1748 of file sort.c.

4.10.3.13 Generator()

```
WORD Generator (
            PHEAD WORD * term,
            WORD level )
```

The heart of the program. Here the expansion tree is set up in one giant recursion

Parameters

term	the input term. may be overwritten
level	the level in the compiler buffer (number of statement)

Returns

Normal conventions (OK = 0).

The routine looks first whether there are unsubstituted (sub)expressions. If so, one of them gets inserted term by term and the new term is used in a renewed call to Generator. If there are no (sub)expressions, the term is normalized, the compiler level is raised (next statement) and the program looks what type of statement this is. If this is a special statement it is either treated on the spot or the appropriate routine is called. If it is a substitution, the pattern matcher is called (TestMatch) which tells whether there was a match. If so we need to call TestSub again to test for (sub)expressions. If we run out of levels, the term receives a final treatment for modulus calculus and/or brackets and is then sent off to the sorting routines.

Definition at line 3101 of file proces.c.

4.10.3.14 InFunction()

```
WORD InFunction (
            PHEAD WORD * term,
            WORD * termout )
```

Makes the replacement of the subexpression with the number 'replac' in a function argument. Additional information is passed in some of the AR, AN, AT variables.

Parameters

term	The input term
termout	The output term

Returns

0: everything is fine, Negative: fatal, Positive: error.

Special attention should be given to nested functions!

Definition at line 2033 of file proces.c.

4.10.3.15 InsertTerm()

```
WORD InsertTerm (
             PHEAD WORD * term,
             WORD replac,
             WORD extractbuff,
             WORD * position,
             WORD * termout,
             WORD tepos )
```

Puts the terms 'term' and 'position' together into a single legal term in termout. replac is the number of the subexpression that should be replaced. It must be a positive term. When action is needed in the argument of a function all terms in that argument are dealt with recursively. The subexpression is sorted. Only one subexpression is done at a time this way.

Parameters

term	the input term
replac	number of the subexpression pointer to replace
extractbuff	number of the compiler buffer replac refers to
position	position from where to take the term in the compiler buffer
termout	the output term
tepos	offset in term where the subexpression is.

Returns

Normal conventions (OK = 0).

Definition at line 2579 of file proces.c.

4.10.3.16 MergePatches()

```
WORD MergePatches (
             WORD par )
```

The general merge routine. Can be used for the large buffer and the file merging. The array S->Patches tells where the patches start S->pStop tells where they end (has to be computed first). The end of a 'line to be merged' is indicated by a zero. If the end is reached without running into a zero or a term runs over the boundary of a patch it is a file merging operation and a new piece from the file is read in.

Parameters

par	If par == 0 the sort is for file -> outputfile. If par == 1 the sort is for large buffer -> sortfile. If par == 2 the sort is for large buffer -> outputfile.

Definition at line 3577 of file sort.c.

4.10.3.17 NewSort()

```
WORD NewSort (
             PHEAD0  )
```

Starts a new sort. At the lowest level this is a 'main sort' with the struct according to the parameters in S0. At higher levels this is a sort for functions, subroutines or dollars. We prepare the arrays and structs.

Returns

Regular convention (OK -> 0)

Definition at line 592 of file sort.c.

4.10.3.18 PasteFile()

```
LONG PasteFile (
             PHEAD WORD number,
             WORD * accum,
             POSITION * position,
             WORD ** accfill,
             RENUMBER renumber,
             WORD * freeze,
             WORD nexpr )
```

Gets a term from stored expression expr and puts it in the accumulator at position number. It returns the length of the term that came from file.

Parameters

number	number of partial terms to skip in accum
accum	the accumulator
position	file position from where to get the stored term
accfill	returns tail position in accum
renumber	the renumber struct for the variables in the stored expression
freeze	information about if we need only the contents of a bracket
nexpr	the number of the stored expression

Returns

Normal conventions (OK = 0).

Definition at line 2715 of file proces.c.

4.10.3.19 PolyFunMul()

```
WORD PolyFunMul (
              PHEAD WORD * term )
```

Multiplies the arguments of multiple occurrences of the polyfun. In this routine we do the original PolyFun with one argument only. The PolyRatFun (PolyFunType = 2) is done in a dedicated routine in the file polywrap.cc The new result is written over the old result.

Parameters

term	It contains the input term and later the output.

Returns

Normal conventions (OK = 0).

Definition at line 5132 of file proces.c.

4.10.3.20 PrepPoly()

```
WORD PrepPoly (
              PHEAD WORD * term,
              WORD par )
```

Routine checks whether the count of function AR.PolyFun is zero or one. If it is one and it has one scalarlike argument the coefficient of the term is pulled inside the argument. If the count is zero a new function is made with the coefficient as its only argument. The function should be placed at its proper position.

When this function is active it places the PolyFun as last object before the coefficient. This is needed because otherwise the compress algorithm has problems in MergePatches.

The bracket routine should also place the PolyFun at a comparable spot. The compression should then stop at the PolyFun. It doesn't really have to stop when writing the final result but this may be too complicated.

The parameter par tells whether we are at groundlevel or inside a function or dollar variable.

Definition at line 4744 of file proces.c.

4.10.3.21 PutIn()

```
LONG PutIn (
            FILEHANDLE * file,
            POSITION * position,
            WORD * buffer,
            WORD ** take,
            int npat )
```

Reads a new patch from position in file handle. It is put at buffer, anything after take is moved forward. This would be part of a term that hasn't been used yet. Because of this there should be some space before the start of the buffer

Parameters

file	The file system from which to read
position	The position from which to read
buffer	The buffer into which to read
take	The unused tail should be moved before the buffer
npat	The number of the patch. Is needed if the information was compressed with gzip, because each patch has its own independent gzip encoding.

Definition at line 1259 of file sort.c.

4.10.3.22 PutOut()

```
WORD PutOut (
             PHEAD WORD * term,
             POSITION * position,
             FILEHANDLE * fi,
             WORD ncomp )
```

Routine writes one term to file handle at position. It returns the new value of the position.

NOTE: For 'final output' we may have to index the brackets. See the struct BRACKETINDEX. We should maintain: 1: a list with brackets array with the brackets 2: a list of objects of type BRACKETINDEX. It contains array with either pointers or offsets to the list of brackets. starting positions in the file. The index may be tied to a maximum size. In that case we may have to prune the list occasionally.

Parameters

term	The term to be written
position	The position in the file. Afterwards it is updated
fi	The file (or its cache) to which should be written
ncomp	Information about what type of compression should be used

Definition at line 1405 of file sort.c.

4.10.3.23 RaisPowCached()

```
VOID RaisPowCached (
            PHEAD WORD x,
            WORD n,
            UWORD ** c,
            WORD * nc )
```

Computes power x^n and caches the value

4.10.4 Description

Calculates the power x^n and stores the results for caching purposes. The pointer c (i.e., the pointer, and not what it points to) is overwritten. What it points to should not be overwritten in the calling function.

4.10.5 Notes

- Caching is done in AT.small_power[]. This array is extended if necessary.

Definition at line 1286 of file reken.c.

Referenced by poly::divmod_univar(), and poly::mul_heap().

4.10.5.1 NormalModulus()

```
int NormalModulus (
            UWORD * a,
            WORD * na )
```

Brings a modular representation in the range -p/2 to +p/2 The return value tells whether anything was done. Routine made in the general modulus revamp of July 2008 (JV).

Definition at line 1393 of file reken.c.

4.10.5.2 GetModInverses()

```
int GetModInverses (
            WORD m1,
            WORD m2,
            WORD * im1,
            WORD * im2 )
```

Input m1 and m2, which are relative prime. determines a∗m1+b∗m2 = 1 (and 1 is the gcd of m1 and m2) then a∗m1 = 1 mod m2 and hence im1 = a. and b∗m2 = 1 mod m1 and hence im2 = b. Set m1 = 0∗m1+1∗m2 = a1∗m1+b1∗m2 m2 = 1∗m1+0∗m2 = a2∗m1+b2∗m2 If everything is OK, the return value is zero

Definition at line 1466 of file reken.c.

4.10.5.3 Sflush()

```
WORD Sflush (
            FILEHANDLE * fi )
```

Puts the contents of a buffer to output Only to be used when there is a single patch in the large buffer.

Parameters

fi	The filesystem (or its cache) to which the patch should be written

Definition at line 1319 of file sort.c.

4.10.5.4 SortWild()

```
WORD SortWild (
            WORD * w,
            WORD nw )
```

Sorts the wildcard entries in the parameter w. Double entries are removed. Full space taken is nw words. Routine serves for the reading of wildcards in the compiler. The entries come in the format: (type,4,number,0) in which the zero is reserved for the future replacement of 'number'.

Parameters

w	buffer with wildcard entries.
nw	number of wildcard entries.

Returns

Normal conventions (OK -> 0)

Definition at line 4552 of file sort.c.

4.10.5.5 SplitMerge()

```
LONG SplitMerge (
            PHEAD WORD ** Pointer,
            LONG number )
```

Algorithm by J.A.M.Vermaseren (31-7-1988)

Note that AN.SplitScratch and AN.InScratch are used also in GarbHand

Merge sort in memory. The input is an array of pointers. Sorting is done recursively by dividing the array in two equal parts and calling SplitMerge for each. When the parts are small enough we can do the compare and take the appropriate action. An addition is that we look for 'runs'. Sequences that are already ordered. This happens a lot when there is very little action in a module. This made FORM faster by a few percent.

Parameters

Pointer	The array of pointers to the terms to be sorted.
number	The number of pointers in Pointer.

The terms are supposed to be sitting in the small buffer and there is supposed to be an extension to this buffer for when there are two terms that should be added and the result takes more space than each of the original terms. The notation guarantees that the result never needs more space than the sum of the spaces of the original terms.

Definition at line 3240 of file sort.c.

4.10.5.6 StoreTerm()

```
WORD StoreTerm (
            PHEAD WORD * term )
```

The central routine to accept terms, store them and keep things at least partially sorted. A call to EndSort will then complete storing and sorting.

Parameters

term	The term to be stored

Returns

Regular return conventions (OK -> 0)

Definition at line 4333 of file sort.c.

4.10.5.7 TermRenumber()

```
WORD TermRenumber (
            WORD * term,
            RENUMBER renumber,
            WORD nexpr )
```

!! WORD *memterm=term; static LONG ctrap=0; !!!

!! ctrap++; !!!

Definition at line 2407 of file store.c.

4.10.5.8 TestMatch()

```
WORD TestMatch (
              PHEAD WORD * term,
              WORD * level )
```

This routine governs the pattern matching. If it decides that a substitution should be made, this can be either the insertion of a right hand side (C->rhs) or the automatic generation of terms as a result of an operation (like trace). The object to be replaced is removed from term and a subexpression pointer is inserted. If the substitution is made more than once there can be more subexpression pointers. Its number is positive as it corresponds to the level at which the C->rhs can be found in the compiler output. The subexpression pointer contains the wildcard substitution information. The power is found in ∗AT.TMout. For operations the subexpression pointer is negative and corresponds to an address in the array AT.TMout. In this array are then the instructions for the routine to be called and its number in the array 'Operations' The format is here: length,functionnumber,length-2 parameters

There is a certain complexity wrt repeat levels. Another complication is the poking of the wildcard values in the subexpression prototype in the compiler buffer. This was how things were done in the past with sequential FORM, but with the advent of TFORM this cannot be maintained. Now, for TFORM we make a copy of it. 7-may-2008 (JV): We cannot yet guarantee that this has been done 100% correctly. There are errors that occur in TFORM only and that may indicate problems.

Definition at line 97 of file pattern.c.

4.10.5.9 TestSub()

```
WORD TestSub (
              PHEAD WORD * term,
              WORD level )
```

TestSub hunts for subexpression pointers. If one is found its power is given in AN.TeSuOut. and the returnvalue is 'expressionnumber'. If the expression number is negative it is an expression on disk.

In addition this routine tries to locate subexpression pointers in functions. It also notices that action must be taken with any of the special functions.

Parameters

term	The term in which TestSub hunts for potential action
level	The number of the 'level' in the compiler buffer.

Returns

The number of the (sub)expression that was encountered.

Other values that are returned are in AN.TeSuOut, AR.TePos, AT.TMbuff, AN.TeInFun, AN.Frozen, AT.TMaddr

The level in the compiler buffer is more or less the number of the statement in the module. Hence it refers to the element in the lhs array.

This routine is one of the most important routines in FORM.

Definition at line 681 of file proces.c.

4.10.5.10 TimeCPU()

```
LONG TimeCPU (
            WORD par )
```

Returns the CPU time.

Parameters

par	If zero, the CPU time will be reset to 0.

Returns

The CPU time in milliseconds.

Definition at line 3478 of file tools.c.

4.10.5.11 TimeWallClock()

```
LONG TimeWallClock (
            WORD par )
```

Returns the wall-clock time.

Parameters

par	If zero, the wall-clock time will be reset to 0.

Returns

The wall-clock time in centiseconds.

Definition at line 3404 of file tools.c.

Referenced by DoCheckpoint().

4.10.5.12 WriteStats()

```
VOID WriteStats (
            POSITION * plspace,
            WORD par )
```

Writes the statistics.

Parameters

plspace	The size in bytes currently occupied
par	par = 0 after a splitmerge. par = 1 after merge to sortfile. par = 2 after the sort

current expression is to be found in AR.CurExpr. terms are in S->TermsLeft. S->GenTerms.

Definition at line 93 of file sort.c.

4.10.5.13 PutPreVar()

```
int PutPreVar (
            UBYTE * name,
            UBYTE * value,
            UBYTE * args,
            int mode )
```

Inserts/Updates a preprocessor variable in the name administration.

Parameters

name	Character string with the variable name.
value	Character string with a possible value. Special case: if this argument is zero, then we have no value. Note: This is different from having an empty argument! This should only occur when the name starts with a ?
args	Character string with possible arguments.
mode	=0: always create a new name entry, =1: try to do a redefinition if possible.

Returns

Index of used entry in name list.

Definition at line 642 of file pre.c.

4.10.5.14 CopyFile()

```
int CopyFile (
            char * source,
            char * dest )
```

Copies a file with name ∗source to a file named ∗dest. The involved files must not be open. Returns non-zero if an error occurred. Uses if possible the combined large and small sorting buffers as cache.

Definition at line 1029 of file tools.c.

4.10.5.15 inicbufs()

```
int inicbufs (
            VOID  )
```

Creates a new compiler buffer and returns its ID number.

Returns

The ID number for the new compiler buffer.

Definition at line 47 of file comtool.c.

4.10.5.16 TheDefine()

```
int TheDefine (
            UBYTE * s,
            int mode )
```

Preprocessor assignment. Possible arguments and values are treated and the new preprocessor variable is put into the name administration.

Parameters

s	Pointer to the character string following the preprocessor command.
mode	Bitmask. 0-bit clear: always create a new name entry, 0-bit set: try to redefine an existing name, 1-bit set: ignore preprocessor if/switch status.

Returns

zero: no errors, negative number: errors.

Definition at line 1942 of file pre.c.

4.10.5.17 ComPress()

```
LONG ComPress (
            WORD ** ss,
            LONG * n )
```

Gets a list of pointers to terms and compresses the terms. In n it collects the number of terms and the return value of the function is the space that is occupied.

We have to pay some special attention to the compression of terms with a PolyFun. This PolyFun should occur only straight before the coefficient, so we can use the same trick as for the coefficient to sabotage compression of this object (Replace in the history the function pointer by zero. This is safe, because terms that would be identical otherwise would have been added).

Parameters

ss	Array of pointers to terms to be compressed.
n	Number of pointers in ss.

Returns

Total number of words needed for the compressed result.

Definition at line 3074 of file sort.c.

4.10.5.18 StageSort()

```
VOID StageSort (
            FILEHANDLE * fout )
```

Prepares a stage 4 or higher sort. Stage 4 sorts occur when the sort file contains more patches than can be merged in one pass.

Definition at line 4453 of file sort.c.

4.10.5.19 DoubleCbuffer()

```
WORD* DoubleCbuffer (
            int num,
            WORD * w,
            int par )
```

Doubles a compiler buffer.

Parameters

num	The ID number for the buffer to be doubled.
w	The pointer to the end (exclusive) of the current buffer. The contents in the range of [cbuf[num].Buffer,w) will be kept.

Definition at line 143 of file comtool.c.

4.10.5.20 AddLHS()

```
WORD* AddLHS (
            int num )
```

Adds an LHS to a compiler buffer and returns the pointer to a buffer for the new LHS.

Parameters

num	The ID number for the buffer to get another LHS.

Definition at line 188 of file comtool.c.

4.10.5.21 AddRHS()

```
WORD* AddRHS (
            int num,
            int type )
```

Adds an RHS to a compiler buffer and returns the pointer to a buffer for the new RHS.

Parameters

num	The ID number for the buffer to get another RHS.
type	If 0, the subexpression tree will be reallocated.

Definition at line 214 of file comtool.c.

4.10.5.22 AddNtoL()

```
int AddNtoL (
            int n,
            WORD * array )
```

Adds an LHS with the given data to the current compiler buffer.

Parameters

n	The length of the data.
array	The data to be added.

Returns

0 if succeeds.

Definition at line 288 of file comtool.c.

4.10.5.23 AddNtoC()

```
int AddNtoC (
             int bufnum,
             int n,
             WORD * array,
             int par )
```

Adds the given data to the last LHS/RHS in a compiler buffer.

Parameters

bufnum	The ID number for the buffer where the data will be added.
n	The length of the data.
array	The data to be added.

Returns

0 if succeeds.

Definition at line 317 of file comtool.c.

4.10.5.24 TakeArgContent()

```
WORD* TakeArgContent (
             PHEAD WORD * argin,
             WORD * argout )
```

Implements part of the old ExecArg in which we take common factors from arguments with more than one term. The common pieces are put in argout as a sequence of arguments. The part with the multiple terms that are now relative prime is put in argfree which is allocated via TermMalloc and is given as the return value. The difference with the old code is that negative powers are always removed. Hence it is as in MakeInteger in which only numerators will be left: now only zero or positive powers will be remaining.

Definition at line 2725 of file argument.c.

4.10.5.25 MakeInteger()

```
WORD* MakeInteger (
             PHEAD WORD * argin,
             WORD * argout,
             WORD * argfree )
```

For normalizing everything to integers we have to determine for all elements of this argument the LCM of the denominators and the GCD of the numerators. The input argument is in argin. The number that comes out should go to argout. The new pointer in the argout buffer is the return value. The normalized argument is in argfree.

Definition at line 3271 of file argument.c.

4.10.5.26 MakeMod()

```
WORD* MakeMod (
             PHEAD WORD * argin,
             WORD * argout,
             WORD * argfree )
```

Similar to MakeInteger but now with modulus arithmetic using only a one WORD 'prime'. We make the coefficient of the first term in the argument equal to one. Already the coefficients are taken modulus AN.cmod and AN.ncmod == 1

Definition at line 3442 of file argument.c.

4.10.5.27 FindArg()

```
WORD FindArg (
             PHEAD WORD * a )
```

Looks the argument up in the (workers) table. If it is found the number in the table is returned (plus one to make it positive). If it is not found we look in the compiler provided table. If it is found - the number in the table is returned (minus one to make it negative). If in neither table we return zero.

Definition at line 2472 of file argument.c.

4.10.5.28 InsertArg()

```
WORD InsertArg (
             PHEAD WORD * argin,
             WORD * argout,
             int par )
```

Inserts the argument into the (workers) table. If the table is too full we eliminate half of it. The eliminated elements are the ones that have not been used most recently, weighted by their total use and age(?). If par == 0 it inserts in the regular factorization cache If par == 1 it inserts in the cache defined with the FactorCache statement

Definition at line 2496 of file argument.c.

4.10.5.29 CleanupArgCache()

```
int CleanupArgCache (
             PHEAD WORD bufnum )
```

Cleans up the argument factorization cache. We throw half the elements. For a weight of what we want to keep we use the product of usage and the number in the buffer.

Definition at line 2531 of file argument.c.

4.10.5.30 SortWeights()

```
void SortWeights (
             LONG * weights,
             LONG * extraspace,
             WORD number )
```

Sorts an array of LONGS in the same way SplitMerge (in sort.c) works We use gradual division in two.

Definition at line 3487 of file argument.c.

4.10.5.31 MakeDollarInteger()

```
WORD* MakeDollarInteger (
             PHEAD WORD * bufin,
             WORD ** bufout )
```

For normalizing everything to integers we have to determine for all elements of this argument the LCM of the denominators and the GCD of the numerators. The input argument is in bufin. The number that comes out is the return value. The normalized argument is in bufout.

Definition at line 3622 of file dollar.c.

4.10.5.32 MakeDollarMod()

```
WORD* MakeDollarMod (
             PHEAD WORD * buffer,
             WORD ** bufout )
```

Similar to MakeDollarInteger but now with modulus arithmetic using only a one WORD 'prime'. We make the coefficient of the first term in the argument equal to one. Already the coefficients are taken modulus AN.cmod and AN.ncmod == 1

Definition at line 3796 of file dollar.c.

4.10.5.33 AddPotModdollar()

```
void AddPotModdollar (
             WORD numdollar )
```

Adds a $-variable specified by *numdollar* to the list of potentially modified $-variables unless it has already been included in the list.

Parameters

numdollar	The index of the $-variable to be added.

Definition at line 3954 of file dollar.c.

4.10.5.34 Optimize()

```
int Optimize (
             WORD exprnr,
             int do_print )
```

Optimization of expression

4.10.6 Description

This method takes an input expression and generates optimized code to calculate its value. The following methods are called to do so:

(1) get_expression : to read to expression

(2) get_brackets : find brackets for simultaneous optimization

(3) occurrence_order or find_Horner_MCTS : to determine (the) Horner scheme(s) to use; this depends on AO.↩ optimize.horner

(4) optimize_expression_given_Horner : to do the optimizations for each Horner scheme; this method does either CSE or greedy optimizations dependings on AO.optimize.method

(5) generate_output : to format the output in Form notation and store it in a buffer

(6a) optimize_print_code : to print the expression (for "Print") or (6b) generate_expression : to modify the expression (for "#Optimize")

On ParFORM, all the processes must call this function at the same time. Then

(1) Because only the master can access to the expression to be optimized, the master broadcast the expression to all the slaves after reading the expression (PF_get_expression).

(2) get_brackets reads optimize_expr as the input and it works also on the slaves. We leave it although the bracket information is not needed on the slaves (used in (5) on the master).

(3) and (4) find_Horner_MCTS and optimize_expression_given_Horner are parallelized.

(5), (6a) and (6b) are needed only on the master.

Definition at line 4587 of file optimize.cc.

4.10.6.1 finishcbuf()

```
void finishcbuf (
             WORD num )
```

Frees a compiler buffer.

Parameters

num	The ID number for the buffer to be freed.

Definition at line 89 of file comtool.c.

4.10.6.2 clearcbuf()

```
void clearcbuf (
            WORD num )
```

Clears contents in a compiler buffer.

Parameters

num	The ID number for the buffer to be cleared.

Definition at line 116 of file comtool.c.

4.10.6.3 CleanUpSort()

```
void CleanUpSort (
            int num )
```

Partially or completely frees function sort buffers.

Definition at line 4644 of file sort.c.

4.10.6.4 EvalDoLoopArg()

```
WORD EvalDoLoopArg (
            PHEAD WORD * arg,
            WORD par )
```

Evaluates one argument of a do loop. Such an argument is constructed from SNUMBERs DOLLAREXPRESSIONs and possibly DOLLAREXPR2s which indicate factors of the preceeding dollar. Hence we have SNUMBER,num DOLLAREXPRESSION,numdollar DOLLAREXPRESSION,numdollar,DOLLAREXPR2,numfactor DOLLAREXPRESSION,numdollar,DOLLAREXPR2,numfactor,DOLLAREXPR2,numfactor etc. Because we have a do-loop at every stage we should have a number. The notation in DOLLAREXPR2 is that $>=0$ is number of yet another dollar and <0 is -n-1 with n the array element or zero. The return value is the (short) number. The routine works its way through the list in a recursive manner.

Definition at line 2646 of file dollar.c.

4.10.6.5 SymbolNormalize()

```
int SymbolNormalize (
            WORD * term )
```

Routine normalizes terms that contain only symbols. Regular minimum and maximum properties are ignored.

We check whether there are negative powers in the output. This is not allowed.

Definition at line 5000 of file normal.c.

4.10.6.6 CompareSymbols()

```
WORD CompareSymbols (
            WORD * term1,
            WORD * term2,
            WORD par )
```

Compares the terms, based on the value of AN.polysortflag. If term1 $<$ term2 the return value is -1 If term1 $>$ term2 the return value is 1 If term1 = term2 the return value is 0 The coefficients may differ. The terms contain only a single subterm of type SYMBOL. If AN.polysortflag = 0 it is a 'regular' compare. If AN.polysortflag = 1 the sum of the powers is more important par is a dummy parameter to make the parameter field identical to that of Compare1 which is the regular compare routine in sort.c

Definition at line 2976 of file sort.c.

4.10.6.7 CompareHSymbols()

```
WORD CompareHSymbols (
            WORD * term1,
            WORD * term2,
            WORD par )
```

Compares terms that can have only SYMBOL and HAAKJE subterms. If term1 $<$ term2 the return value is -1 If term1 $>$ term2 the return value is 1 If term1 = term2 the return value is 0 par is a dummy parameter to make the parameter field identical to that of Compare1 which is the regular compare routine in sort.c

Definition at line 3020 of file sort.c.

4.10.6.8 NextPrime()

```
WORD NextPrime (
            PHEAD WORD num )
```

Gives the next prime number in the list of prime numbers.

If the list isn't long enough we expand it. For ease in ParForm and because these lists shouldn't be very big we let each worker keep its own list.

The list is cut off at MAXPOWER, because we don't want to get into trouble that the power of a variable gets larger than the prime number.

Definition at line 3654 of file reken.c.

4.10.6.9 ReadSaveIndex()

```
WORD ReadSaveIndex (
            FILEINDEX * fileind )
```

Reads a FILEINDEX from the open save file specified by AO.SaveData.Handle. Translations for adjusting endianness and data sizes are done if necessary.

Depends on the assumption that sizeof(FILEINDEX) is the same everywhere. If FILEINDEX or INDEXENTRY change, then this functions has to be adjusted.

Called by CoLoad() and FindInIndex().

Parameters

fileind	contains the read FILEINDEX after succesful return. must point to allocated, big enough memory.

Returns

= 0 everything okay, != 0 an error occurred

Definition at line 4137 of file store.c.

4.10.6.10 ReadSaveExpression()

```
WORD ReadSaveExpression (
            UBYTE * buffer,
            UBYTE * top,
            LONG * size,
            LONG * outsize )
```

Reads an expression from the open file specified by AO.SaveData.Handle. The endianness flip and a resizing without renumbering is done in this function. Thereafter the buffer consists of chunks with a uniform maximal word size (32bit at the moment). The actual renumbering is then done by calling the function ReadSaveTerm32(). The result is returned in *buffer*.

If the translation at some point doesn't fit into the buffer anymore, the function returns and must be called again. In any case *size* returns the number of successfully read bytes, *outsize* returns the number of successfully written bytes, and the file will be positioned at the next byte after the successfully read data.

It is called by PutInStore().

Parameters

buffer	output buffer, holds the (translated) expression
top	end of buffer
size	number of read bytes
outsize	number of written bytes

Returns

= 0 everything okay, != 0 an error occurred

Definition at line 5092 of file store.c.

4.10.6.11 ReadSaveTerm32()

```
UBYTE* ReadSaveTerm32 (
            UBYTE * bin,
            UBYTE * binend,
            UBYTE ** bout,
            UBYTE * boutend,
            UBYTE * top,
            int terminbuf )
```

Reads a single term from the given buffer at *bin* and write the translated term back to this buffer at *bout*.

ReadSaveTerm32() is currently the only instantiation of a ReadSaveTerm-function. It only deals with data that already has the correct endianness and that is resized to 32bit words but without being renumbered or translated in any other way. It uses the compress buffer AR.CompressBuffer.

The function is reentrant in order to cope with nested function arguments. It is called by ReadSaveExpression() and itself.

The *return value* indicates the position in the input buffer up to which the data has already been successfully processed. The parameter *bout* returns the corresponding position in the output buffer.

Parameters

bin	start of the input buffer
binend	end of the input buffer
bout	as input points to the beginning of the output buffer, as output points behind the already translated data in the output buffer
boutend	end of already decompressed data in output buffer
top	end of output buffer
terminbuf	flag whether decompressed data is already in the output buffer. used in recursive calls

Returns

pointer to the next unprocessed data in the input buffer

Definition at line 4683 of file store.c.

4.10.6.12 ReadSaveVariables()

```
WORD ReadSaveVariables (
            UBYTE * buffer,
```

```
            UBYTE * top,
            LONG * size,
            LONG * outsize,
            INDEXENTRY * ind,
            LONG * stage )
```

Reads the variables from the open file specified by AO.SaveData.Handle. It reads the ∗size bytes and writes them to the ∗buffer. It is called by PutInStore().

If translation is necessary, the data might shrink or grow in size, then ∗*size* is adjusted so that the reading and writing fits into the memory from the buffer to the top. The actual number of read bytes is returned in ∗*size*, the number of written bytes is returned in ∗*outsize*.

If the ∗size is smaller than the actual size of the variables, this function will be called several times and needs to remember the current position in the variable structure. The parameter *stage* does this job. When ReadSaveVariables() is called for the first time, this parameter should have the value -1.

The parameter *ind* is used to get the number of variables.

Parameters

buffer	read variables are written into this allocated memory
top	upper end of allocated memory
size	number of bytes to read. might return a smaller number of read bytes if translation was necessary
outsize	if translation has be done, outsize contains the number of written bytes
ind	pointer of INDEXENTRY for the current expression. read-only
stage	should be -1 for the first call, will be increased by ReadSaveVariables to memorize the position in the variable structure

Returns

= 0 everything okay, != 0 an error occurred

Definition at line 4323 of file store.c.

4.10.6.13 WriteStoreHeader()

```
WORD WriteStoreHeader (
            WORD handle )
```

Writes header with information about system architecture and FORM revision to an open store file.

Called by SetFileIndex().

Parameters

handle	specifies open file to which header will be written

Returns

= 0 everything okay, != 0 an error occurred

Definition at line 3926 of file store.c.

4.10.6.14 DoRecovery()

```
int DoRecovery (
            int * moduletype )
```

Reads from the recovery file and restores all necessary variables and states in FORM, so that the execution can recommence in preprocessor() as if no restart of FORM had occurred.

The recovery file is read into memory as a whole. The pointer p then points into this memory at the next non-processed data. The macros by which variables are restored, like R_SET, automatically increase p appropriately.

If something goes wrong, the function returns with a non-zero value.

Allocated memory that would be lost when overwriting the global structs with data from the file is freed first. A major part of the code deals with the restoration of pointers. The idiom we use is to memorize the original pointer value (org), allocate new memory and copy the data from the file into this memory, calculate the offset between the old pointer value and the new allocated memory position (ofs), and then correct all affected pointers (+=ofs).

We rely on the fact that several variables (especially in AM) are already assigned the correct values by the startup functions. That means, in principle, that a change in the setup files between snapshot creation and recovery will be noticed.

Definition at line 1399 of file checkpoint.c.

4.10.6.15 DoCheckpoint()

```
void DoCheckpoint (
            int moduletype )
```

Checks whether a snapshot should be done. Calls DoSnapshot() to create the snapshot.

Definition at line 3102 of file checkpoint.c.

References TimeWallClock().

4.10.6.16 TestTerm()

```
int TestTerm (
            WORD * term )
```

Tests the consistency of the term. Returns 0 when the term is OK. Any nonzero value is trouble. In the current version the testing isn't 100% complete. For instance, we don't check the validity of the symbols nor do we check the range of their powers. Etc. This should be extended when the need is there.

Parameters

term	the term to be tested

Definition at line 3789 of file tools.c.

4.10.6.17 LocalConvertToPoly()

```
int LocalConvertToPoly (
             PHEAD WORD * term,
             WORD * outterm,
             WORD startebuf,
             WORD par )
```

Converts a generic term to polynomial notation in which there are only symbols and brackets. During conversion there will be only symbols. Brackets are stripped. Objects that need 'translation' are put inside a special compiler buffer and represented by a symbol. The numbering of the extra symbols is down from the maximum. In principle there can be a problem when running into the already assigned ones. This uses the FindTree for searching in the global tree and then looks further in the AT.ebufnum. This allows fully parallel processing. Hence we need no locks. Cannot be used in the same module as ConvertToPoly.

Definition at line 510 of file notation.c.

4.10.6.18 IniFbuffer()

```
int IniFbuffer (
             WORD bufnum )
```

Initialize a factorization cache buffer. We set the size of the rhs and boomlijst buffers immediately to their final values.

Definition at line 614 of file comtool.c.

4.10.6.19 TakeSymbolContent()

```
WORD* TakeSymbolContent (
             PHEAD WORD * in,
             WORD * term )
```

Implements part of the old ExecArg in which we take common factors from arguments with more than one term. We allow only symbols as this code is used for the polyratfun only. We have a special routine, because the generic TakeContent does too much work and speed is at a premium here. Input: in is the input expression as a sequence of terms. Output: term: the content return value: the contentfree expression. it is in new allocation, made by TermMalloc. (should be in a TermMalloc space?)

Definition at line 2434 of file ratio.c.

4.10.6.20 TakeContent()

```
WORD* TakeContent (
             PHEAD WORD * in,
             WORD * term )
```

Implements part of the old ExecArg in which we take common factors from arguments with more than one term. Here the input is a sequence of terms in 'in' and the answer is a content-free sequence of terms. This sequence has been allocated by the Malloc1 routine in a call to EndSort, unless the expression was already content-free. In that case the input pointer is returned. The content is returned in term. This is supposed to be a separate allocation, made by TermMalloc in the calling routine.

Definition at line 1376 of file ratio.c.

4.10.6.21 poly_gcd()

```
WORD* poly_gcd (
             PHEAD WORD * a,
             WORD * b,
             WORD fit )
```

Polynomial gcd

4.10.7 Description

This method calculates the greatest common divisor of two polynomials, given by two zero-terminated Form-style term lists.

4.10.8 Notes

- The result is written at newly allocated memory
- Called from ratio.c
- Calls polygcd::gcd

Definition at line 124 of file polywrap.cc.

4.10.8.1 poly_ratfun_add()

```
WORD* poly_ratfun_add (
             PHEAD WORD * t1,
             WORD * t2 )
```

Addition of PolyRatFuns

4.10.9 Description

This method gets two pointers to polyratfuns with up to two arguments each and calculates the sum.

4.10.10 Notes

- The result is written at the workpointer
- Called from sort.c and threads.c
- Calls poly::operators and polygcd::gcd

Definition at line 600 of file polywrap.cc.

4.10.10.1 poly_ratfun_normalize()

```
int poly_ratfun_normalize (
            PHEAD WORD * term )
```

Multiplication/normalization of PolyRatFuns

4.10.11 Description

This method seaches a term for multiple polyratfuns and multiplies their contents. The result is properly normalized. Normalization also works for terms with a single polyratfun.

4.10.12 Notes

- The result overwrites the original term
- Called from proces.c
- Calls poly::operators and polygcd::gcd

Definition at line 719 of file polywrap.cc.

4.10.12.1 poly_factorize_argument()

```
int poly_factorize_argument (
            PHEAD WORD * argin,
            WORD * argout )
```

Factorization of function arguments

4.10.13 Description

This method factorizes the Form-style argument argin.

4.10.14 Notes

- The result is written at argout
- Called from argument.c
- Calls poly_factorize

Definition at line 1047 of file polywrap.cc.

4.10.14.1 poly_factorize_dollar()

```
WORD* poly_factorize_dollar (
            PHEAD WORD * argin )
```

Factorization of dollar variables

4.10.15 Description

This method factorizes a dollar variable.

4.10.16 Notes

- The result is written at newly allocated memory.
- Called from dollar.c
- Calls poly_factorize

Definition at line 1074 of file polywrap.cc.

4.10.16.1 poly_factorize_expression()

```
int poly_factorize_expression (
            EXPRESSIONS expr )
```

Factorization of expressions

4.10.17 Description

This method factorizes an expression.

4.10.18 Notes

- The result overwrites the input expression
- Called from proces.c
- Calls polyfact::factorize

Definition at line 1100 of file polywrap.cc.

4.10.18.1 optimize_print_code()

```
VOID optimize_print_code (
             int print_expr )
```

Print optimized code

4.10.19 Description

This method prints the optimized code via "PrintExtraSymbol". Depending on the flag, the original expression is printed (for "Print") or not (for "#Optimize / #write "O").

Definition at line 4474 of file optimize.cc.

4.10.19.1 DoPreAppendPath()

```
int DoPreAppendPath (
             UBYTE * s )
```

Appends the given path (absolute or relative to the current file directory) to the FORM path.

Syntax: #appendpath <path>

Definition at line 6954 of file pre.c.

4.10.19.2 DoPrePrependPath()

```
int DoPrePrependPath (
            UBYTE * s )
```

Prepends the given path (absolute or relative to the current file directory) to the FORM path.

Syntax: #prependpath <path>

Definition at line 6971 of file pre.c.

4.11 diagrams.c File Reference

```
#include "form3.h"
```

Functions

- int **CoCanonicalize** (UBYTE *s)
- int **DoCanonicalize** (PHEAD WORD *term, WORD *params)
- WORD **GenTopologies** (PHEAD WORD *term, WORD level)
- WORD **GenDiagrams** (PHEAD WORD *term, WORD level)
- int **DoTopologyCanonicalize** (PHEAD WORD *term, WORD vert, WORD edge, WORD *args)
- int **DoShattering** (PHEAD WORD *connect, WORD *environ, WORD *partitions, WORD nvert)

4.11.1 Detailed Description

Contains the wrapper routines for diagram manipulations.

4.12 dict.c File Reference

```
#include "form3.h"
```

Functions

- VOID **TransformRational** (UWORD *a, WORD na)
- UBYTE * **IsMultiplySign** (VOID)
- UBYTE * **IsExponentSign** (VOID)
- UBYTE * **FindSymbol** (WORD num)
- UBYTE * **FindVector** (WORD num)
- UBYTE * **FindIndex** (WORD num)
- UBYTE * **FindFunction** (WORD num)
- UBYTE * **FindFunWithArgs** (WORD *t)
- UBYTE * **FindExtraSymbol** (WORD num)
- int **FindDictionary** (UBYTE *name)
- int **AddDictionary** (UBYTE *name)
- int **AddToDictionary** (DICTIONARY *dict, UBYTE *left, UBYTE *right)
- int **UseDictionary** (UBYTE *name, UBYTE *options)
- int **SetDictionaryOptions** (UBYTE *options)
- void **UnSetDictionary** (VOID)
- void **RemoveDictionary** (DICTIONARY *dict)
- void **ShrinkDictionary** (DICTIONARY *dict)
- int **DoPreOpenDictionary** (UBYTE *s)
- int **DoPreCloseDictionary** (UBYTE *s)
- int **DoPreUseDictionary** (UBYTE *s)
- int **DoPreAdd** (UBYTE *s)
- LONG **DictToBytes** (DICTIONARY *dict, UBYTE *buf)
- DICTIONARY * **DictFromBytes** (UBYTE *buf)

4.12.1 Detailed Description

Contains the code pertaining to dictionaries Commands are: #opendictionary name #closedictionary #selectdictionary name <options> There can be several dictionaries, but only one can be active. Defining elements is done with #add object: "replacement" Replacements are strings when a dictionary is for output translation. Objects can be 1: a number (rational) 2: a variable 3: * ^ 4: a function with arguments

4.13 dollar.c File Reference

```
#include "form3.h"
```

Macros

- #define STEP2

Functions

- int **CatchDollar** (int par)
- int **AssignDollar** (PHEAD WORD ∗term, WORD level)
- UBYTE ∗ **WriteDollarToBuffer** (WORD numdollar, WORD par)
- UBYTE ∗ **WriteDollarFactorToBuffer** (WORD numdollar, WORD numfac, WORD par)
- void **AddToDollarBuffer** (UBYTE ∗s)
- void **TermAssign** (WORD ∗term)
- int **PutTermInDollar** (WORD ∗term, WORD numdollar)
- void **WildDollars** (PHEAD WORD ∗term)
- WORD **DolToTensor** (PHEAD WORD numdollar)
- WORD **DolToFunction** (PHEAD WORD numdollar)
- WORD **DolToVector** (PHEAD WORD numdollar)
- WORD **DolToNumber** (PHEAD WORD numdollar)
- WORD **DolToSymbol** (PHEAD WORD numdollar)
- WORD **DolToIndex** (PHEAD WORD numdollar)
- DOLLARS **DolToTerms** (PHEAD WORD numdollar)
- LONG **DolToLong** (PHEAD WORD numdollar)
- int **ExecInside** (UBYTE ∗s)
- int **InsideDollar** (PHEAD WORD ∗ll, WORD level)
- void **ExchangeDollars** (int num1, int num2)
- LONG **TermsInDollar** (WORD num)
- LONG **SizeOfDollar** (WORD num)
- UBYTE ∗ **PrelfDollarEval** (UBYTE ∗s, int ∗value)
- WORD ∗ **TranslateExpression** (UBYTE ∗s)
- int **IsSetMember** (WORD ∗buffer, WORD numset)
- int **IsMultipleOf** (WORD ∗buf1, WORD ∗buf2)
- int **TwoExprCompare** (WORD ∗buf1, WORD ∗buf2, int oprtr)
- int **DollarRaiseLow** (UBYTE ∗name, LONG value)
- WORD EvalDoLoopArg (PHEAD WORD ∗arg, WORD par)
- WORD **TestDoLoop** (PHEAD WORD ∗lhsbuf, WORD level)
- WORD **TestEndDoLoop** (PHEAD WORD ∗lhsbuf, WORD level)
- int **DollarFactorize** (PHEAD WORD numdollar)
- void **CleanDollarFactors** (DOLLARS d)
- WORD ∗ **TakeDollarContent** (PHEAD WORD ∗dollarbuffer, WORD ∗∗factor)
- WORD ∗ MakeDollarInteger (PHEAD WORD ∗bufin, WORD ∗∗bufout)
- WORD ∗ MakeDollarMod (PHEAD WORD ∗buffer, WORD ∗∗bufout)
- int **GetDolNum** (PHEAD WORD ∗t, WORD ∗tstop)
- void AddPotModdollar (WORD numdollar)

4.13.1 Detailed Description

The routines that deal with the dollar variables. The name administration is to be found in the file names.c

4.13.2 Macro Definition Documentation

4.13.2.1 STEP2

```
#define STEP2
```

Factors a dollar expression. Notation: d->nfactors becomes nonzero. if the number of factors is one, we leave d->factors zero. Otherwise factors is an array of pointers to the factors. These are pointers of the type FACDOLLAR. fd->where pointer to contents in term notation fd->size size of the buffer fd->where points to fd->type DOLNUMBER or DOLTERMS fd->value value if type is DOLNUMBER and it fits in a WORD.

Definition at line 2948 of file dollar.c.

4.13.3 Function Documentation

4.13.3.1 EvalDoLoopArg()

```
WORD EvalDoLoopArg (
              PHEAD WORD * arg,
              WORD par )
```

Evaluates one argument of a do loop. Such an argument is constructed from SNUMBERs DOLLAREXPRESSIONs and possibly DOLLAREXPR2s which indicate factors of the preceeding dollar. Hence we have SNUMBER,num DOLLAREXPRESSION,numdollar DOLLAREXPRESSION,numdollar,DOLLAREXPR2,numfactor DOLLAREXPRESSION,numdollar,DOLLAREXPR2,numfactor,DOLLAREXPR2,numfactor etc. Because we have a do-loop at every stage we should have a number. The notation in DOLLAREXPR2 is that >= 0 is number of yet another dollar and < 0 is -n-1 with n the array element or zero. The return value is the (short) number. The routine works its way through the list in a recursive manner.

Definition at line 2646 of file dollar.c.

4.13.3.2 MakeDollarInteger()

```
WORD* MakeDollarInteger (
              PHEAD WORD * bufin,
              WORD ** bufout )
```

For normalizing everything to integers we have to determine for all elements of this argument the LCM of the denominators and the GCD of the numerators. The input argument is in bufin. The number that comes out is the return value. The normalized argument is in bufout.

Definition at line 3622 of file dollar.c.

4.13.3.3 MakeDollarMod()

```
WORD* MakeDollarMod (
            PHEAD WORD * buffer,
            WORD ** bufout )
```

Similar to MakeDollarInteger but now with modulus arithmetic using only a one WORD 'prime'. We make the coefficient of the first term in the argument equal to one. Already the coefficients are taken modulus AN.cmod and AN.ncmod == 1

Definition at line 3796 of file dollar.c.

4.13.3.4 AddPotModdollar()

```
void AddPotModdollar (
            WORD numdollar )
```

Adds a $-variable specified by *numdollar* to the list of potentially modified $-variables unless it has already been included in the list.

Parameters

numdollar	The index of the $-variable to be added.

Definition at line 3954 of file dollar.c.

4.14 execute.c File Reference

```
#include "form3.h"
```

Macros

- #define **CURRENTBRACKET** 1
- #define **BRACKETCURRENTEXPR** 2
- #define **BRACKETOTHEREXPR** 3
- #define **NOBRACKETACTIVE** 4

Functions

- WORD **CleanExpr** (WORD par)
- WORD **PopVariables** ()
- VOID **MakeGlobal** ()
- VOID **TestDrop** ()
- void **PutInVflags** (WORD nexpr)
- WORD **DoExecute** (WORD par, WORD skip)

- WORD **PutBracket** (PHEAD WORD ∗termin)
- VOID **SpecialCleanup** (PHEAD0)
- void **SetMods** ()
- void **UnSetMods** ()
- void **ExchangeExpressions** (int num1, int num2)
- int **GetFirstBracket** (WORD ∗term, int num)
- int **GetFirstTerm** (WORD ∗term, int num)
- int **GetContent** (WORD ∗content, int num)
- int **CleanupTerm** (WORD ∗term)
- WORD **ContentMerge** (PHEAD WORD ∗content, WORD ∗term)
- LONG **TermsInExpression** (WORD num)
- LONG **SizeOfExpression** (WORD num)
- void **UpdatePositions** ()
- LONG **CountTerms1** (PHEAD0)
- LONG **TermsInBracket** (PHEAD WORD ∗term, WORD level)

4.14.1 Detailed Description

The routines that start the execution phase of a module. It also contains the routines for placing the bracket subterm.

4.15 extcmd.c File Reference

```
#include "form3.h"
```

Functions

- int **openExternalChannel** (UBYTE ∗cmd, int daemonize, UBYTE ∗shellname, UBYTE ∗stderrname)
- int **initPresetExternalChannels** (UBYTE ∗theline, int thetimeout)
- int **closeExternalChannel** (int n)
- int **selectExternalChannel** (int n)
- int **getCurrentExternalChannel** ()
- void **closeAllExternalChannels** ()

4.15.1 Detailed Description

The system that takes care of communication with external programs.

4.16 factor.c File Reference

```
#include "form3.h"
```

Functions

- int **FactorIn** (PHEAD WORD *term, WORD level)
- int **FactorInExpr** (PHEAD WORD *term, WORD level)

4.16.1 Detailed Description

The routines for finding (one term) factors in dollars and expressions.

4.17 findpat.c File Reference

```
#include "form3.h"
```

Functions

- WORD **FindOnly** (PHEAD WORD *term, WORD *pattern)
- WORD **FindOnce** (PHEAD WORD *term, WORD *pattern)
- WORD **FindMulti** (PHEAD WORD *term, WORD *pattern)
- WORD **FindRest** (PHEAD WORD *term, WORD *pattern)

4.17.1 Detailed Description

Pattern matching of symbols and dotproducts. There are various routines because of the options in the id-statements like once, only, multi and many. These are amoung the oldest routines in FORM and that can be noticed, because the interplay with the function matching is not complete. When we match functions and halfway we fail we can backtrack properly. With the symbols, the dotproducts and the vectors (in pattern.c) there is no proper backtracking. Hence the routines here need still quite some work or may even have to be rewritten.

4.18 form3.h File Reference

```
#include <stddef.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>
#include <limits.h>
#include "ftypes.h"
#include "fsizes.h"
#include "minos.h"
#include "structs.h"
#include "declare.h"
#include "variable.h"
```

Macros

- #define **MAJORVERSION** 4
- #define **MINORVERSION** 2
- #define **PRODUCTIONDATE** "06-jul-2017"
- #define **inline**
- #define **NDEBUG**
- #define **STATIC_ASSERT**(condition) STATIC_ASSERT__1(condition,__LINE__)
- #define **STATIC_ASSERT__1**(X, L) STATIC_ASSERT__2(X,L)
- #define **STATIC_ASSERT__2**(X, L) STATIC_ASSERT__3(X,L)
- #define **STATIC_ASSERT__3**(X, L) typedef char static_assertion_failed_##L[(!!(X))∗2-1]
- #define **TOPBITONLY** ((ULONG)1 << (BITSINWORD - 1)) /∗ 0x00008000UL ∗/
- #define **TOPLONGBITONLY** ((ULONG)1 << (BITSINLONG - 1)) /∗ 0x80000000UL ∗/
- #define **SPECMASK** ((UWORD)1 << (BITSINWORD - 1)) /∗ 0x8000U ∗/
- #define **WILDMASK** ((UWORD)1 << (BITSINWORD - 2)) /∗ 0x4000U ∗/
- #define **WORDMASK** ((ULONG)FULLMAX - 1) /∗ 0x0000FFFFUL ∗/
- #define **AWORDMASK** (WORDMASK << BITSINWORD) /∗ 0xFFFF0000UL ∗/
- #define **FULLMAX** ((LONG)1 << BITSINWORD) /∗ 0x00010000L ∗/
- #define **MAXPOSITIVE** ((LONG)(TOPBITONLY - 1)) /∗ 0x00007FFFL ∗/
- #define **MAXLONG** ((LONG)(TOPLONGBITONLY - 1)) /∗ 0x7FFFFFFFL ∗/
- #define **MAXPOSITIVE2** (MAXPOSITIVE / 2) /∗ 0x00003FFFL ∗/
- #define **MAXPOSITIVE4** (MAXPOSITIVE / 4) /∗ 0x00001FFFL ∗/
- #define **alignof**(type) offsetof(struct { char c_; type x_; }, x_)
- #define **PADDUMMY**(type, size) UBYTE d_u_m_m_y[alignof(type) - ((size) & (alignof(type) - 1))]
- #define **PADPOSITION**(ptr_, long_, int_, word_, byte_)
- #define **PADPOINTER**(long_, int_, word_, byte_)
- #define **PADLONG**(int_, word_, byte_)
- #define **PADINT**(word_, byte_)
- #define **PADWORD**(byte_)
- #define **WITHSORTBOTS**
- #define **FILES** FILE
- #define **Uopen**(x, y) fopen(x,y)
- #define **Uflush**(x) fflush(x)
- #define **Uclose**(x) fclose(x)
- #define **Uread**(x, y, z, u) fread(x,y,z,u)
- #define **Uwrite**(x, y, z, u) fwrite(x,y,z,u)
- #define **Usetbuf**(x, y) setbuf(x,y)
- #define **Useek**(x, y, z) fseek(x,y,z)
- #define **Utell**(x) ftell(x)
- #define **Ugetpos**(x, y) fgetpos(x,y)
- #define **Usetpos**(x, y) fsetpos(x,y)
- #define **Usync**(x) fflush(x)
- #define **Utruncate**(x) _chsize(_fileno(x),0)
- #define **Ustdout** stdout
- #define **MAX_OPEN_FILES** FOPEN_MAX
- #define **bzero**(b, len) (memset((b), 0, (len)), (void)0)
- #define **GetPID**() ((LONG)GetCurrentProcessId())

Typedefs

- typedef void **VOID**
- typedef signed char **SBYTE**
- typedef unsigned char **UBYTE**
- typedef unsigned int **UINT**
- typedef ULONG **RLONG**
- typedef INT64 **MLONG**

Functions

- **STATIC_ASSERT** (sizeof(WORD) ∗8==BITSINWORD)
- **STATIC_ASSERT** (sizeof(LONG) ∗8==BITSINLONG)
- **STATIC_ASSERT** (sizeof(LONG) >=sizeof(int ∗))
- **STATIC_ASSERT** (sizeof(INT16)==2)
- **STATIC_ASSERT** (sizeof(INT32)==4)
- **STATIC_ASSERT** (sizeof(INT64)==8)

4.18.1 Detailed Description

Contains critical defines for the compilation process Also contains the inclusion of all necessary header files. There are also some system dependencies concerning file functions.

4.18.2 Macro Definition Documentation

4.18.2.1 PADPOSITION

```
#define PADPOSITION(
              ptr_,
              long_,
              int_,
              word_,
              byte_ )
```

Value:

```
    PADDUMMY(off_t, \
        + sizeof(int *) * (ptr_) \
        + sizeof(LONG)  * (long_) \
        + sizeof(int)   * (int_) \
        + sizeof(WORD)  * (word_) \
        + sizeof(UBYTE) * (byte_) \
    )
```

Definition at line 381 of file form3.h.

4.18.2.2 PADPOINTER

```
#define PADPOINTER(
              long_,
              int_,
              word_,
              byte_ )
```

Value:

```
    PADDUMMY(int *, \
        + sizeof(LONG)  * (long_) \
        + sizeof(int)   * (int_) \
        + sizeof(WORD)  * (word_) \
        + sizeof(UBYTE) * (byte_) \
    )
```

Definition at line 389 of file form3.h.

4.18.2.3 PADLONG

```
#define PADLONG(
                int_,
                word_,
                byte_ )
```

Value:

```
    PADDUMMY(LONG, \
        + sizeof(int)   * (int_) \
        + sizeof(WORD)  * (word_) \
        + sizeof(UBYTE) * (byte_) \
    )
```

Definition at line 396 of file form3.h.

4.18.2.4 PADINT

```
#define PADINT(
                word_,
                byte_ )
```

Value:

```
    PADDUMMY(int, \
        + sizeof(WORD)  * (word_) \
        + sizeof(UBYTE) * (byte_) \
    )
```

Definition at line 402 of file form3.h.

4.18.2.5 PADWORD

```
#define PADWORD(
                byte_ )
```

Value:

```
    PADDUMMY(WORD, \
        + sizeof(UBYTE) * (byte_) \
    )
```

Definition at line 407 of file form3.h.

4.19 fsizes.h File Reference

Macros

- #define **MAXPRENAMESIZE** 128
- #define **MAXPOWER** 10000
- #define **MAXVARIABLES** 8050
- #define **MAXDOLLARVARIABLES** 32000
- #define **WILDOFFSET** 6100
- #define **MAXINNAMETREE** 32768
- #define **MAXDUMMIES** 1000
- #define **WORKBUFFER** 10000000
- #define **MAXTER** 10000
- #define **HALFMAX** 0x100
- #define **MAXSUBEXPRESSIONS** 0x3FFF
- #define **MAXFILESTREAMSIZE** 1048576
- #define **MAXENAME** 16
- #define **MAXSAVEFUNCTION** 16384
- #define **MAXPARLEVEL** 100
- #define **MAXNUMBERSIZE** 200
- #define **MAXREPEAT** 100
- #define **NORMSIZE** 1000
- #define **INITNODESIZE** 10
- #define **INITNAMESIZE** 100
- #define **NUMFIXED** 128
- #define **MAXNEST** 100
- #define **MAXMATCH** 30
- #define **MAXIF** 20
- #define **SIZEFACS** 640L
- #define **NUMFACS** 50
- #define **MAXLOOPS** 30
- #define **MAXLABELS** 20
- #define **COMMERCIALSIZE** 24
- #define **MAXFLAGS** 16
- #define **COMPRESSBUFFER** 90000
- #define **FORTRANCONTINUATIONLINES** 15
- #define **MAXLEVELS** 2000
- #define **MAXLHS** 400
- #define **MAXWILDC** 100
- #define **NUMTABLEENTRIES** 1000
- #define **COMPILERBUFFER** 20000
- #define **SMALLBUFFER** 10000000L
- #define **SMALLOVERFLOW** 20000000L
- #define **TERMSSMALL** 100000L
- #define **LARGEBUFFER** 50000000L
- #define **SCRATCHSIZE** 50000000L
- #define **MAXPATCHES** 256
- #define **MAXFPATCHES** 256
- #define **SORTIOSIZE** 200000L
- #define **SSMALLBUFFER** 500000L
- #define **SSMALLOVERFLOW** 800000L
- #define **STERMSSMALL** 10000L
- #define **SLARGEBUFFER** 4000000L
- #define **SMAXPATCHES** 64

- #define **SMAXFPATCHES** 64
- #define **SSORTIOSIZE** 32768L
- #define **SPECTATORSIZE** 1048576L
- #define **MAXFLEVELS** 30
- #define **COMPINC** 2
- #define **MAXNUMSIZE** 10
- #define **MAXBRACKETBUFFERSIZE** 200000
- #define **SFHSIZE** 40
- #define **DEFAULTPROCESSBUCKETSIZE** 1000
- #define **SHMWINSIZE** 65536L
- #define **TABLEEXTENSION** 6
- #define **GZIPDEFAULT** 6
- #define **DEFAULTTHREADS** 0
- #define **DEFAULTTHREADBUCKETSIZE** 500
- #define **DEFAULTTHREADLOADBALANCING** 1
- #define **THREADSCRATCHSIZE** 100000L
- #define **THREADSCRATCHOUTSIZE** 2500000L
- #define **MAXTABLECOMBUF** 1000000L
- #define **MAXCOMBUFRHS** 32500L
- #define **NUMSTORECACHES** 4
- #define **SIZESTORECACHE** 32768
- #define **INDENTSPACE** 3
- #define **MULTIINDENTSPACE** 1
- #define **MAXMULTIBRACKETLEVELS** 25
- #define **FBUFFERSIZE** 1026
- #define **NPAIR1** 38
- #define **NPAIR2** 89
- #define **MAXLINELENGTH** 256
- #define **MINALLOC** 32
- #define **JUMPRATIO** 4

4.19.1 Detailed Description

The definition the default values for certain buffer sizes etc.

4.20 ftypes.h File Reference

Macros

- #define WITHOUTERROR 0
- #define **WITHERROR** 1
- #define **FILESTREAM** 0
- #define **PREVARSTREAM** 1
- #define **PREREADSTREAM** 2
- #define **PIPESTREAM** 3
- #define **PRECALCSTREAM** 4
- #define **DOLLARSTREAM** 5
- #define **PREREADSTREAM2** 6
- #define **EXTERNALCHANNELSTREAM** 7
- #define **PREREADSTREAM3** 8
- #define **REVERSEFILESTREAM** 9

- #define **ENDOFSTREAM** 0xFF
- #define **ENDOFINPUT** 0xFF
- #define **SUBROUTINEFILE** 0
- #define **PROCEDUREFILE** 1
- #define **HEADERFILE** 2
- #define **SETUPFILE** 3
- #define **TABLEBASEFILE** 4
- #define **FIRSTMODULE** -1
- #define **GLOBALMODULE** 0
- #define **SORTMODULE** 1
- #define **STOREMODULE** 2
- #define **CLEARMODULE** 3
- #define **ENDMODULE** 4
- #define **POLYFUN** 0
- #define **NOPARALLEL_DOLLAR** 0x0001
- #define **NOPARALLEL_RHS** 0x0002
- #define **NOPARALLEL_CONVPOLY** 0x0004
- #define **NOPARALLEL_SPECTATOR** 0x0008
- #define **NOPARALLEL_USER** 0x0010
- #define **NOPARALLEL_TBLDOLLAR** 0x0100
- #define **NOPARALLEL_NPROC** 0x0200
- #define **PARALLELFLAG** 0x0000
- #define **PRENOACTION** 0
- #define **PRERAISEAFTER** 1
- #define **PRELOWERAFTER** 2
- #define **WITHSEMICOLON** 0
- #define **WITHOUTSEMICOLON** 1
- #define **MODULEINSTACK** 8
- #define **EXECUTINGIF** 0
- #define **LOOKINGFORELSE** 1
- #define **LOOKINGFORENDIF** 2
- #define **NEWSTATEMENT** 1
- #define **OLDSTATEMENT** 0
- #define **EXECUTINGPRESWITCH** 0
- #define **SEARCHINGPRECASE** 1
- #define **SEARCHINGPREENDSWITCH** 2
- #define **PREPROONLY** 1
- #define **DUMPTOCOMPILER** 2
- #define **DUMPOUTTERMS** 4
- #define **DUMPINTERMS** 8
- #define **DUMPTOSORT** 16
- #define **DUMPTOPARALLEL** 32
- #define **THREADSDEBUG** 64
- #define **ERROROUT** 0
- #define **INPUTOUT** 1
- #define **STATSOUT** 2
- #define **EXPRSOUT** 3
- #define **WRITEOUT** 4
- #define **EXTERNALCHANNELOUT** 5
- #define **NUMERICALLOOP** 0
- #define **LISTEDLOOP** 1
- #define **ONEEXPRESSION** 2
- #define **PRETYPENONE** 0
- #define **PRETYPEIF** 1
- #define **PRETYPEDO** 2

- #define **PRETYPEPROCEDURE** 3
- #define **PRETYPESWITCH** 4
- #define **PRETYPEINSIDE** 5
- #define **DECLARATION** 1
- #define **SPECIFICATION** 2
- #define **DEFINITION** 3
- #define **STATEMENT** 4
- #define **TOOUTPUT** 5
- #define **ATENDOFMODULE** 6
- #define **MIXED** 9
- #define **VOID** void
- #define **NOAUTO** 0
- #define **PARTEST** 1
- #define **WITHAUTO** 2
- #define **ALLVARIABLES** -1
- #define **SYMBOLONLY** 1
- #define **INDEXONLY** 2
- #define **VECTORONLY** 4
- #define **FUNCTIONONLY** 8
- #define **SETONLY** 16
- #define **EXPRESSIONONLY** 32

Defines: compiler types

Type of variable found by the compiler.

- #define **CDELETE** -1
- #define **ANYTYPE** -1
- #define **CSYMBOL** 0
- #define **CINDEX** 1
- #define **CVECTOR** 2
- #define **CFUNCTION** 3
- #define **CSET** 4
- #define **CEXPRESSION** 5
- #define **CDOTPRODUCT** 6
- #define **CNUMBER** 7
- #define **CSUBEXP** 8
- #define **CDELTA** 9
- #define **CDOLLAR** 10
- #define **CDUBIOUS** 11
- #define **CRANGE** 12
- #define **CVECTOR1** 21
- #define **CDOUBLEDOT** 22
- #define **TSYMBOL** -1
- #define **TINDEX** -2
- #define **TVECTOR** -3
- #define **TFUNCTION** -4
- #define **TSET** -5
- #define **TEXPRESSION** -6
- #define **TDOTPRODUCT** -7
- #define **TNUMBER** -8
- #define **TSUBEXP** -9
- #define **TDELTA** -10
- #define **TDOLLAR** -11
- #define **TDUBIOUS** -12
- #define **LPARENTHESIS** -13
- #define **RPARENTHESIS** -14
- #define **TWILDCARD** -15
- #define **TWILDARG** -16
- #define **TDOT** -17

- #define **LBRACE** -18
- #define **RBRACE** -19
- #define **TCOMMA** -20
- #define **TFUNOPEN** -21
- #define **TFUNCLOSE** -22
- #define **TMULTIPLY** -23
- #define **TDIVIDE** -24
- #define **TPOWER** -25
- #define **TPLUS** -26
- #define **TMINUS** -27
- #define **TNOT** -28
- #define **TENDOFIT** -29
- #define **TSETOPEN** -30
- #define **TSETCLOSE** -31
- #define **TGENINDEX** -32
- #define **TCONJUGATE** -33
- #define **LRPARENTHESES** -34
- #define **TNUMBER1** -35
- #define **TPOWER1** -36
- #define **TEMPTY** -37
- #define **TSETNUM** -38
- #define **TSGAMMA** -39
- #define **TSETDOL** -40
- #define **TYPEISFUN** 0
- #define **TYPEISSUB** 1
- #define **TYPEISMYSTERY** -1
- #define **LHSIDEX** 2
- #define **LHSIDE** 1
- #define **RHSIDE** 0
- #define **FORTRANMODE** 1
- #define **REDUCEMODE** 2
- #define **MAPLEMODE** 3
- #define **MATHEMATICAMODE** 4
- #define **CMODE** 5
- #define **VORTRANMODE** 6
- #define **PFORTRANMODE** 7
- #define **DOUBLEFORTRANMODE** 33
- #define **DOUBLEPRECISIONFLAG** 32
- #define **NODOUBLEMASK** 31
- #define **QUADRUPLEFORTRANMODE** 65
- #define **QUADRUPLEPRECISIONFLAG** 64
- #define **ALLINTEGERDOUBLE** 128
- #define **NOQUADMASK** 63
- #define **NORMALFORMAT** 0
- #define **NOSPACEFORMAT** 1
- #define **ISNOTFORTRAN90** 0
- #define **ISFORTRAN90** 1
- #define **ALSOREVERSE** 1
- #define **CHISHOLM** 2
- #define **NOTRICK** 16
- #define **SORTLOWFIRST** 0
- #define **SORTHIGHFIRST** 1
- #define **SORTPOWERFIRST** 2
- #define **SORTANTIPOWER** 3
- #define **NMIN4SHIFT** 4
- #define **SYMBOL** 1
- #define **DOTPRODUCT** 2
- #define **VECTOR** 3
- #define **INDEX** 4
- #define **EXPRESSION** 5
- #define **SUBEXPRESSION** 6
- #define **DOLLAREXPRESSION** 7
- #define **SETSET** 8
- #define **ARGWILD** 9

- #define **MINVECTOR** 10
- #define **SETEXP** 11
- #define **DOLLAREXPR2** 12
- #define **HAAKJE0** 9
- #define **FUNCTION** 20
- #define **TMPPOLYFUN** 14
- #define **ARGFIELD** 15
- #define **SNUMBER** 16
- #define **LNUMBER** 17
- #define **HAAKJE** 18
- #define **DELTA** 19
- #define **EXPONENT** 20
- #define **DENOMINATOR** 21
- #define **SETFUNCTION** 22
- #define **GAMMA** 23
- #define **GAMMAI** 24
- #define **GAMMAFIVE** 25
- #define **GAMMASIX** 26
- #define **GAMMASEVEN** 27
- #define **SUMF1** 28
- #define **SUMF2** 29
- #define **DUMFUN** 30
- #define **REPLACEMENT** 31
- #define **REVERSEFUNCTION** 32
- #define **DISTRIBUTION** 33
- #define **DELTA3** 34
- #define **DUMMYFUN** 35
- #define **DUMMYTEN** 36
- #define **LEVICIVITA** 37
- #define **FACTORIAL** 38
- #define **INVERSEFACTORIAL** 39
- #define **BINOMIAL** 40
- #define **NUMARGSFUN** 41
- #define **SIGNFUN** 42
- #define **MODFUNCTION** 43
- #define **MOD2FUNCTION** 44
- #define **MINFUNCTION** 45
- #define **MAXFUNCTION** 46
- #define **ABSFUNCTION** 47
- #define **SIGFUNCTION** 48
- #define **INTFUNCTION** 49
- #define **THETA** 50
- #define **THETA2** 51
- #define **DELTA2** 52
- #define **DELTAP** 53
- #define **BERNOULLIFUNCTION** 54
- #define **COUNTFUNCTION** 55
- #define **MATCHFUNCTION** 56
- #define **PATTERNFUNCTION** 57
- #define **TERMFUNCTION** 58
- #define **CONJUGATION** 59
- #define **ROOTFUNCTION** 60
- #define **TABLEFUNCTION** 61
- #define **FIRSTBRACKET** 62
- #define **TERMSINEXPR** 63
- #define **NUMTERMSFUN** 64
- #define **GCDFUNCTION** 65
- #define **DIVFUNCTION** 66
- #define **REMFUNCTION** 67
- #define **MAXPOWEROF** 68
- #define **MINPOWEROF** 69
- #define **TABLESTUB** 70
- #define **FACTORIN** 71
- #define **TERMSINBRACKET** 72

- #define **WILDARGFUN** 73
- #define **SQRTFUNCTION** 74
- #define **LNFUNCTION** 75
- #define **SINFUNCTION** 76
- #define **COSFUNCTION** 77
- #define **TANFUNCTION** 78
- #define **ASINFUNCTION** 79
- #define **ACOSFUNCTION** 80
- #define **ATANFUNCTION** 81
- #define **ATAN2FUNCTION** 82
- #define **SINHFUNCTION** 83
- #define **COSHFUNCTION** 84
- #define **TANHFUNCTION** 85
- #define **ASINHFUNCTION** 86
- #define **ACOSHFUNCTION** 87
- #define **ATANHFUNCTION** 88
- #define **LI2FUNCTION** 89
- #define **LINFUNCTION** 90
- #define **EXTRASYMFUN** 91
- #define **RANDOMFUNCTION** 92
- #define **RANPERM** 93
- #define **NUMFACTORS** 94
- #define **FIRSTTERM** 95
- #define **CONTENTTERM** 96
- #define **PRIMENUMBER** 97
- #define **EXTEUCLIDEAN** 98
- #define **MAKERATIONAL** 99
- #define **INVERSEFUNCTION** 100
- #define **IDFUNCTION** 101
- #define **PUTFIRST** 102
- #define **PERMUTATIONS** 103
- #define **PARTITIONS** 104
- #define **MULFUNCTION** 105
- #define **TOPOLOGIES** 106
- #define **DIAGRAMS** 107
- #define **VERTEX** 108
- #define **EDGE** 109
- #define **SIZEOFFUNCTION** 110
- #define **MAXBUILTINFUNCTION** 110
- #define **FIRSTUSERFUNCTION** 150
- #define **ISYMBOL** 0
- #define **PISYMBOL** 1
- #define **COEFFSYMBOL** 2
- #define **NUMERATORSYMBOL** 3
- #define **DENOMINATORSYMBOL** 4
- #define **WILDARGSYMBOL** 5
- #define **DIMENSIONSYMBOL** 6
- #define **FACTORSYMBOL** 7
- #define **SEPARATESYMBOL** 8
- #define **BUILTINSYMBOLS** 9
- #define **FIRSTUSERSYMBOL** 20
- #define **BUILTININDICES** 1
- #define **BUILTINVECTORS** 1
- #define **BUILTINDOLLARS** 1
- #define **WILDARGVECTOR** 0
- #define **WILDARGINDEX** 0
- #define **TYPEEXPRESSION** 0
- #define **TYPEIDNEW** 1
- #define **TYPEIDOLD** 2
- #define **TYPEOPERATION** 3
- #define **TYPEREPEAT** 4
- #define **TYPEENDREPEAT** 5
- #define **TYPECOUNT** 20
- #define **TYPEMULT** 21

- #define **TYPEGOTO** 22
- #define **TYPEDISCARD** 23
- #define **TYPEIF** 24
- #define **TYPEELSE** 25
- #define **TYPEELIF** 26
- #define **TYPEENDIF** 27
- #define **TYPESUM** 28
- #define **TYPECHISHOLM** 29
- #define **TYPEREVERSE** 30
- #define **TYPEARG** 31
- #define **TYPENORM** 32
- #define **TYPENORM2** 33
- #define **TYPENORM3** 34
- #define **TYPEEXIT** 35
- #define **TYPESETEXIT** 36
- #define **TYPEPRINT** 37
- #define **TYPEFPRINT** 38
- #define **TYPEREDEFPRE** 39
- #define **TYPESPLITARG** 40
- #define **TYPESPLITARG2** 41
- #define **TYPEFACTARG** 42
- #define **TYPEFACTARG2** 43
- #define **TYPETRY** 44
- #define **TYPEASSIGN** 45
- #define **TYPERENUMBER** 46
- #define **TYPESUMFIX** 47
- #define **TYPEFINDLOOP** 48
- #define **TYPEUNRAVEL** 49
- #define **TYPEADJUSTBOUNDS** 50
- #define **TYPEINSIDE** 51
- #define **TYPETERM** 52
- #define **TYPESORT** 53
- #define **TYPEDETCURDUM** 54
- #define **TYPEINEXPRESSION** 55
- #define **TYPESPLITFIRSTARG** 56
- #define **TYPESPLITLASTARG** 57
- #define **TYPEMERGE** 58
- #define **TYPETESTUSE** 59
- #define **TYPEAPPLY** 60
- #define **TYPEAPPLYRESET** 61
- #define **TYPECHAININ** 62
- #define **TYPECHAINOUT** 63
- #define **TYPENORM4** 64
- #define **TYPEFACTOR** 65
- #define **TYPEARGIMPLODE** 66
- #define **TYPEARGEXPLODE** 67
- #define **TYPEDENOMINATORS** 68
- #define **TYPESTUFFLE** 69
- #define **TYPEDROPCOEFFICIENT** 70
- #define **TYPETRANSFORM** 71
- #define **TYPETOPOLYNOMIAL** 72
- #define **TYPEFROMPOLYNOMIAL** 73
- #define **TYPEDOLOOP** 74
- #define **TYPEENDDOLOOP** 75
- #define **TYPEDROPSYMBOLS** 76
- #define **TYPEPUTINSIDE** 77
- #define **TYPETOSPECTATOR** 78
- #define **TYPEARGTOEXTRASYMBOL** 79
- #define **TYPECANONICALIZE** 80
- #define **TYPESWITCH** 81
- #define **TYPEENDSWITCH** 82
- #define **TAKETRACE** 1
- #define **CONTRACT** 2
- #define **RATIO** 3

- #define **SYMMETRIZE** 4
- #define **TENVEC** 5
- #define **SUMNUM1** 6
- #define **SUMNUM2** 7
- #define **WILDDUMMY** 0
- #define **SYMTONUM** 1
- #define **SYMTOSYM** 2
- #define **SYMTOSUB** 3
- #define **VECTOMIN** 4
- #define **VECTOVEC** 5
- #define **VECTOSUB** 6
- #define **INDTOIND** 7
- #define **INDTOSUB** 8
- #define **FUNTOFUN** 9
- #define **ARGTOARG** 10
- #define **ARLTOARL** 11
- #define **EXPTOEXP** 12
- #define **FROMBRAC** 13
- #define **FROMSET** 14
- #define **SETTONUM** 15
- #define **WILDCARDS** 16
- #define **SETNUMBER** 17
- #define **LOADDOLLAR** 18
- #define **NUMTONUM** 20
- #define **NUMTOSYM** 21
- #define **NUMTOIND** 22
- #define **NUMTOSUB** 23
- #define **CLEANFLAG** 0
- #define **DIRTYFLAG** 1
- #define **DIRTYSYMFLAG** 2
- #define **MUSTCLEANPRF** 4
- #define **SUBTERMUSED1** 8
- #define **SUBTERMUSED2** 16
- #define **ALLDIRTY** (DIRTYFLAG|DIRTYSYMFLAG)
- #define **ARGHEAD** 2
- #define **FUNHEAD** 3
- #define **SUBEXPSIZE** 5
- #define **EXPRHEAD** 5
- #define **TYPEARGHEADSIZE** 6
- #define **SKIP** 1
- #define **DROP** 2
- #define **HIDE** 3
- #define **UNHIDE** 4
- #define **INTOHIDE** 5
- #define **LOCALEXPRESSION** 0
- #define **SKIPLEXPRESSION** 1
- #define **DROPLEXPRESSION** 2
- #define **DROPPEDEXPRESSION** 3
- #define **GLOBALEXPRESSION** 4
- #define **SKIPGEXPRESSION** 5
- #define **DROPGEXPRESSION** 6
- #define **STOREDEXPRESSION** 8
- #define **HIDDENLEXPRESSION** 9
- #define **HIDDENGEXPRESSION** 13
- #define **INCEXPRESSION** 9
- #define **HIDELEXPRESSION** 10
- #define **HIDEGEXPRESSION** 14
- #define **DROPHLEXPRESSION** 11
- #define **DROPHGEXPRESSION** 15
- #define **UNHIDELEXPRESSION** 12
- #define **UNHIDEGEXPRESSION** 16
- #define **INTOHIDELEXPRESSION** 17
- #define **INTOHIDEGEXPRESSION** 18
- #define **SPECTATOREXPRESSION** 19

- #define **DROPSPECTATOREXPRESSION** 20
- #define **SKIPUNHIDELEXPRESSION** 21
- #define **SKIPUNHIDEGEXPRESSION** 22
- #define **PRINTOFF** 0
- #define **PRINTON** 1
- #define **PRINTCONTENTS** 2
- #define **PRINTCONTENT** 3
- #define **PRINTLFILE** 4
- #define **PRINTONETERM** 8
- #define **PRINTONEFUNCTION** 16
- #define **PRINTALL** 32
- #define **REGULAREXPRESSION** -1
- #define **REDEFINEDEXPRESSION** -2
- #define **NEWLYDEFINEDEXPRESSION** -3

Defines: function specs

Function specifications.

- #define **GENERALFUNCTION** 0
- #define **FASTFUNCTION** 1
- #define **TENSORFUNCTION** 2
- #define **GAMMAFUNCTION** 4
- #define **POS_** 0 /∗ integer > 0 ∗/
- #define **POS0_** 1 /∗ integer >= 0 ∗/
- #define **NEG_** 2 /∗ integer < 0 ∗/
- #define **NEG0_** 3 /∗ integer <= 0 ∗/
- #define **EVEN_** 4 /∗ integer (even) ∗/
- #define **ODD_** 5 /∗ integer (odd) ∗/
- #define **Z_** 6 /∗ integer ∗/
- #define **SYMBOL_** 7 /∗ symbol only ∗/
- #define **FIXED_** 8 /∗ fixed index ∗/
- #define **INDEX_** 9 /∗ index only ∗/
- #define **Q_** 10 /∗ rational ∗/
- #define **DUMMYINDEX_** 11 /∗ dummy index only ∗/
- #define **VECTOR_** 12 /∗ vector only ∗/
- #define **GAMMA1** 0
- #define **GAMMA5** -1
- #define **GAMMA6** -2
- #define **GAMMA7** -3
- #define **FUNNYVEC** -4
- #define **FUNNYWILD** -5
- #define **SUMMEDIND** -6
- #define **NOINDEX** -7
- #define **FUNNYDOLLAR** -8
- #define **EMPTYINDEX** -9
- #define **MINSPEC** -10
- #define **USEDFLAG** 2
- #define **DUMMYFLAG** 1
- #define **MAINSORT** 0
- #define **FUNCTIONSORT** 1
- #define **SUBSORT** 2
- #define **FLOATMODE** 1
- #define **RATIONALMODE** 0
- #define **NUMSPECSETS** 10
- #define **EATTENSOR** 0x2000
- #define **ISZERO** 1
- #define **ISUNMODIFIED** 2
- #define **ISCOMPRESSED** 4
- #define **ISINRHS** 8
- #define **ISFACTORIZED** 16
- #define **TOBEFACTORED** 32
- #define **TOBEUNFACTORED** 64
- #define **KEEPZERO** 128

- #define **VARTYPENONE** 0
- #define **VARTYPECOMPLEX** 1
- #define **VARTYPEIMAGINARY** 2
- #define **VARTYPEROOTOFUNITY** 4
- #define **VARTYPEMINUS** 8
- #define **CYCLESYMMETRIC** 1
- #define **RCYCLESYMMETRIC** 2
- #define **SYMMETRIC** 3
- #define **ANTISYMMETRIC** 4
- #define **REVERSEORDER** 256
- #define **SUBMULTI** 1
- #define **SUBONCE** 2
- #define **SUBONLY** 3
- #define **SUBMANY** 4
- #define **SUBVECTOR** 5
- #define **SUBSELECT** 6
- #define **SUBALL** 7
- #define **SUBMASK** 15
- #define **SUBDISORDER** 16
- #define **SUBAFTER** 32
- #define **SUBAFTERNOT** 64
- #define **IDHEAD** 6
- #define **DOLLARFLAG** 1
- #define **NORMALIZEFLAG** 2
- #define **GIDENT** 1
- #define **GFIVE** 4
- #define **GPLUS** 3
- #define **GMINUS** 2
- #define **LONGNUMBER** 1
- #define **MATCH** 2
- #define **COEFFI** 3
- #define **SUBEXPR** 4
- #define **MULTIPLEOF** 5
- #define **IFDOLLAR** 6
- #define **IFEXPRESSION** 7
- #define **IFDOLLAREXTRA** 8
- #define **IFISFACTORIZED** 9
- #define **IFOCCURS** 10
- #define **GREATER** 0
- #define **GREATEREQUAL** 1
- #define **LESS** 2
- #define **LESSEQUAL** 3
- #define **EQUAL** 4
- #define **NOTEQUAL** 5
- #define **ORCOND** 6
- #define **ANDCOND** 7
- #define **DUMMY** 1
- #define **SORT** 1
- #define **STORE** 2
- #define **END** 3
- #define **GLOBAL** 4
- #define **CLEAR** 5
- #define **VECTBIT** 1
- #define **DOTPBIT** 2
- #define **FUNBIT** 4
- #define **SETBIT** 8
- #define **EXTRAPARAMETER** 0x4000
- #define **GENCOMMUTE** 0
- #define **GENNONCOMMUTE** 0x2000
- #define **NAMENOTFOUND** -9
- #define **DOLUNDEFINED** 0
- #define **DOLNUMBER** 1
- #define **DOLARGUMENT** 2
- #define **DOLSUBTERM** 3

- #define **DOLTERMS** 4
- #define **DOLWILDARGS** 5
- #define **DOLINDEX** 6
- #define **DOLZERO** 7
- #define **FINDLOOP** 0
- #define **REPLACELOOP** 1
- #define **NOFUNPOWERS** 0
- #define **COMFUNPOWERS** 1
- #define **ALLFUNPOWERS** 2
- #define **PROPERORDERFLAG** 0
- #define **REGULAR** 0
- #define **FINISH** 1
- #define **POLYADD** 1
- #define **POLYSUB** 2
- #define **POLYMUL** 3
- #define **POLYDIV** 4
- #define **POLYREM** 5
- #define **POLYGCD** 6
- #define **POLYINTFAC** 7
- #define **POLYNORM** 8
- #define **MODNONE** 0
- #define **MODSUM** 1
- #define **MODMAX** 2
- #define **MODMIN** 3
- #define **MODLOCAL** 4
- #define **ELEMENTUSED** 1
- #define **ELEMENTLOADED** 2
- #define **POSNEG** 0x1
- #define **INVERSETABLE** 0x2
- #define **COEFFICIENTSONLY** 0x4
- #define **ALSOPOWERS** 0x8
- #define **ALSOFUNARGS** 0x10
- #define **ALSODOLLARS** 0x20
- #define **NOINVERSES** 0x40
- #define **POSITIVEONLY** 0
- #define **UNPACK** 0x80
- #define **NOUNPACK** 0
- #define **FROMFUNCTION** 0x100
- #define **VARNAMES** 0
- #define **AUTONAMES** 1
- #define **EXPRNAMES** 2
- #define **DOLLARNAMES** 3
- #define **DUMMYBUFFER** 1
- #define **ALLARGS** 1
- #define **NUMARG** 2
- #define **ARGRANGE** 3
- #define **MAKEARGS** 4
- #define **MAXRANGEINDICATOR** 4
- #define **REPLACEARG** 5
- #define **ENCODEARG** 6
- #define **DECODEARG** 7
- #define **IMPLODEARG** 8
- #define **EXPLODEARG** 9
- #define **PERMUTEARG** 10
- #define **REVERSEARG** 11
- #define **CYCLEARG** 12
- #define **ISLYNDON** 13
- #define **ISLYNDONR** 14
- #define **TOLYNDON** 15
- #define **TOLYNDONR** 16
- #define **ADDARG** 17
- #define **MULTIPLYARG** 18
- #define **DROPARG** 19
- #define **SELECTARG** 20

- #define **DEDUPARG** 21
- #define **BASECODE** 1
- #define **YESLYNDON** 1
- #define **NOLYNDON** 2
- #define **TOPOLYNOMIALFLAG** 1
- #define **FACTARGFLAG** 2
- #define **OLDFACTARG** 1
- #define **NEWFACTARG** 0
- #define **FROMMODULEOPTION** 0
- #define **FROMPOINTINSTRUCTION** 1
- #define **EXTRASYMBOL** 0
- #define **REGULARSYMBOL** 1
- #define **EXPRESSIONNUMBER** 2
- #define **O_NONE** 0
- #define **O_CSE** 1
- #define **O_CSEGREEDY** 2
- #define **O_GREEDY** 3
- #define **O_OCCURRENCE** 0
- #define **O_MCTS** 1
- #define **O_SIMULATED_ANNEALING** 2
- #define **O_FORWARD** 0
- #define **O_BACKWARD** 1
- #define **O_FORWARDORBACKWARD** 2
- #define **O_FORWARDANDBACKWARD** 3
- #define **OPTHEAD** 3
- #define **DOALL** 1
- #define **ONLYFUNCTIONS** 2
- #define **INUSE** 1
- #define **COULDCOMMUTE** 2
- #define **DOESNOTCOMMUTE** 4
- #define **DICT_NONUMBERS** 0
- #define **DICT_INTEGERONLY** 1
- #define **DICT_RATIONALONLY** 2
- #define **DICT_ALLNUMBERS** 3
- #define **DICT_NOVARIABLES** 0
- #define **DICT_DOVARIABLES** 1
- #define **DICT_NOSPECIALS** 0
- #define **DICT_DOSPECIALS** 1
- #define **DICT_NOFUNWITHARGS** 0
- #define **DICT_DOFUNWITHARGS** 1
- #define **DICT_NOTINDOLLARS** 0
- #define **DICT_INDOLLARS** 1
- #define **DICT_INTEGERNUMBER** 1
- #define **DICT_RATIONALNUMBER** 2
- #define **DICT_SYMBOL** 3
- #define **DICT_VECTOR** 4
- #define **DICT_INDEX** 5
- #define **DICT_FUNCTION** 6
- #define **DICT_FUNCTION_WITH_ARGUMENTS** 7
- #define **DICT_SPECIALCHARACTER** 8
- #define **DICT_RANGE** 9
- #define **READSPECTATORFLAG** 3
- #define **GLOBALSPECTATORFLAG** 1
- #define **ORDEREDSET** 1
- #define **DENSETABLE** 1
- #define **SPARSETABLE** 0

Typedefs

- typedef VOID(∗ **PVFUNWP**) ()
- typedef VOID(∗ **PVFUNV**) ()
- typedef int(∗ **CFUN**) ()
- typedef int(∗ **TFUN**) ()
- typedef int(∗ **TFUN1**) ()

4.20.1 Detailed Description

Contains the definitions of many internal codes Rather than using numbers directly we do this by defines, making it much easier to change things. Changing things is sometimes also a good way of testing the code.

4.20.2 Macro Definition Documentation

4.20.2.1 WITHOUTERROR

```
#define WITHOUTERROR 0
```

The next macros were introduced when TFORM was programmed. In the case of workers, each worker may need some private data. These can in principle be accessed by some posix calls but that is unnecessarily slow. The passing of a pointer to the complete data struct with private data will be much faster. And anyway, there would have to be a macro that either makes the posix call (TFORM) or doesn't (FORM). The solution by having macro's that either pass the pointer (TFORM) or don't pass it (FORM) is seen as the best solution.

In the declarations and the calling of the functions we have to use the PHEAD or the BHEAD macro, respectively, if the pointer is to be passed. These macro's contain the comma as well. Hence we need special macro's if there are no other arguments. These are called PHEAD0 and BHEAD0.

Definition at line 51 of file ftypes.h.

4.21 function.c File Reference

```
#include "form3.h"
```

Functions

- WORD **MakeDirty** (WORD ∗term, WORD ∗x, WORD par)
- void **MarkDirty** (WORD ∗term, WORD flags)
- void **PolyFunDirty** (PHEAD WORD ∗term)
- void **PolyFunClean** (PHEAD WORD ∗term)
- WORD **Symmetrize** (PHEAD WORD ∗func, WORD ∗Lijst, WORD ngroups, WORD gsize, WORD type)
- WORD **CompGroup** (PHEAD WORD type, WORD ∗∗args, WORD ∗a1, WORD ∗a2, WORD num)
- int **FullSymmetrize** (PHEAD WORD ∗fun, int type)
- WORD **SymGen** (PHEAD WORD ∗term, WORD ∗params, WORD num, WORD level)
- WORD **SymFind** (PHEAD WORD ∗term, WORD ∗params)
- int **ChainIn** (PHEAD WORD ∗term, WORD funnum)
- int **ChainOut** (PHEAD WORD ∗term, WORD funnum)
- WORD **MatchFunction** (PHEAD WORD ∗pattern, WORD ∗interm, WORD ∗wilds)
- WORD **ScanFunctions** (PHEAD WORD ∗inpat, WORD ∗inter, WORD par)

4.21.1 Detailed Description

The file with the central routines for the pattern matching of functions and their arguments. The file also contains the routines for the execution of the Symmetrize statement and its variations (like antisymmetrize etc).

4.22 fwin.h File Reference

Macros

- #define **LINEFEED** '\n'
- #define **CARRIAGERETURN** 0x0D
- #define **WITHRETURN**
- #define **WITHSYSTEM**
- #define **P_term**(code) exit((int)(code<0?-code:code))
- #define **SEPARATOR** '\\'
- #define **ALTSEPARATOR** '/'
- #define **PATHSEPARATOR** ';'
- #define **WITH_ENV**

4.22.1 Detailed Description

Settings for Windows computers.

4.23 if.c File Reference

```
#include "form3.h"
```

Functions

- WORD **GetIfDollarNum** (WORD *ifp, WORD *ifstop)
- int **FindVar** (WORD *v, WORD *term)
- WORD **DoIfStatement** (PHEAD WORD *ifcode, WORD *term)
- WORD **HowMany** (PHEAD WORD *ifcode, WORD *term)
- VOID **DoubleIfBuffers** ()
- int **DoSwitch** (PHEAD WORD *term, WORD *lhs)
- int **DoEndSwitch** (PHEAD WORD *term, WORD *lhs)
- SWITCHTABLE * **FindCase** (WORD nswitch, WORD ncase)
- int **DoubleSwitchBuffers** ()
- VOID **SwitchSplitMergeRec** (SWITCHTABLE *array, WORD num, SWITCHTABLE *auxarray)
- VOID **SwitchSplitMerge** (SWITCHTABLE *array, WORD num)

4.23.1 Detailed Description

Routines for the dealing with if statements.

4.24 index.c File Reference

```
#include "form3.h"
```

Functions

- POSITION * **FindBracket** (WORD nexp, WORD *bracket)
- VOID **PutBracketInIndex** (PHEAD WORD *term, POSITION *newpos)
- void **ClearBracketIndex** (WORD numexp)
- VOID **OpenBracketIndex** (WORD nexpr)
- int **PutInside** (PHEAD WORD *term, WORD *code)

4.24.1 Detailed Description

The routines that deal with bracket indexing It creates the bracket index and it can find the brackets using the index.

4.25 inivar.h File Reference

Data Structures

- struct **fixedfun**

Variables

- FIXEDGLOBALS **FG**
- ALLGLOBALS **A**
- FIXEDSET **fixedsets** [ ]
- UBYTE **BufferForOutput** [MAXLINELENGTH+14]
- char * **setupfilename** = "form.set"

4.25.1 Detailed Description

Contains the initialization of a number of structs at compile time This file should only be included in the file startup.c !!!

4.25.2 Variable Documentation

4.25.2.1 fixedsets

```
FIXEDSET fixedsets[]
```

Initial value:

```
= {
     {"pos_",   "integers > 0",  CSYMBOL, 0}
    ,{"pos0_",  "integers >= 0", CSYMBOL, 0}
    ,{"neg_",   "integers < 0",  CSYMBOL, 0}
    ,{"neg0_",  "integers <= 0", CSYMBOL, 0}
    ,{"even_",  "even integers", CSYMBOL, 0}
    ,{"odd_",   "odd integers",  CSYMBOL, 0}
    ,{"int_",   "all integers",  CSYMBOL, 0}
    ,{"symbol_","only symbols",  CSYMBOL, MAXPOSITIVE}
    ,{"fixed_", "fixed indices", CINDEX, 0}
    ,{"index_", "all indices",   CINDEX, 0}
    ,{"number_","all rationals", CSYMBOL, 0}
    ,{"dummyindices_", "dummy indices", CINDEX, 0}
    ,{"vector_","only vectors",  CVECTOR, 0}
}
```

Definition at line 241 of file inivar.h.

4.26 lus.c File Reference

```
#include "form3.h"
```

Functions

- int **Lus** (WORD ∗term, WORD funnum, WORD loopsize, WORD numargs, WORD outfun, WORD mode)
- int **FindLus** (int from, int level, int openindex)
- int **SortTheList** (int ∗slist, int num)

4.26.1 Detailed Description

Routines to find loops in index contractions. These routines allow for a category of topological statements. They were originally developed for the color library.

4.27 message.c File Reference

```
#include "form3.h"
```

Functions

- VOID **Error0** (char ∗s)
- VOID **Error1** (char ∗s, UBYTE ∗t)
- VOID **Error2** (char ∗s1, char ∗s2, UBYTE ∗t)
- int **MesWork** ()
- int **MesPrint** (va_alist)
- VOID **Warning** (char ∗s)
- VOID **HighWarning** (char ∗s)
- int **MesCall** (char ∗s)
- WORD **MesCerr** (char ∗s, UBYTE ∗t)
- WORD **MesComp** (char ∗s, UBYTE ∗p, UBYTE ∗q)
- VOID **PrintTerm** (WORD ∗term, char ∗where)
- VOID **PrintTermC** (WORD ∗term, char ∗where)
- VOID **PrintSubTerm** (WORD ∗term, char ∗where)
- VOID **PrintWords** (WORD ∗buffer, LONG number)
- void **PrintSeq** (WORD ∗a, char ∗text)

4.27.1 Detailed Description

Contains the routines that write messages. This includes the very important routine MesPrint which is the FORM equivalent of printf but then with escape sequences that are relevant for symbolic manipulation. The FORM statement Print "...." is passed almost literally to MesPrint.

4.28 minos.c File Reference

```
#include "form3.h"
#include "minos.h"
```

Macros

- #define **CFD**(y, s, type, x, j)
- #define **CTD**(y, s, type, x, j)

Functions

- int **minosread** (FILE *f, char *buffer, MLONG size)
- int **minoswrite** (FILE *f, char *buffer, MLONG size)
- char * **str_dup** (char *str)
- void **convertblock** (INDEXBLOCK *in, INDEXBLOCK *out, int mode)
- void **convertnamesblock** (NAMESBLOCK *in, NAMESBLOCK *out, int mode)
- void **convertiniinfo** (INIINFO *in, INIINFO *out, int mode)
- FILE * **LocateBase** (char **name, char **newname)
- int **ReadIndex** (DBASE *d)
- int **WriteIndexBlock** (DBASE *d, MLONG num)
- int **WriteNamesBlock** (DBASE *d, MLONG num)
- int **WriteIndex** (DBASE *d)
- int **WriteIniInfo** (DBASE *d)
- int **ReadIniInfo** (DBASE *d)
- DBASE * **GetDbase** (char *filename)
- DBASE * **NewDbase** (char *name, MLONG number)
- void **FreeTableBase** (DBASE *d)
- int **ComposeTableNames** (DBASE *d)
- DBASE * **OpenDbase** (char *filename)
- MLONG **AddTableName** (DBASE *d, char *name, TABLES T)
- MLONG **GetTableName** (DBASE *d, char *name)
- int **PutTableNames** (DBASE *d)
- int **AddToIndex** (DBASE *d, MLONG number)
- MLONG **AddObject** (DBASE *d, MLONG tablenumber, char *arguments, char *rhs)
- MLONG **FindTableNumber** (DBASE *d, char *name)
- int **WriteObject** (DBASE *d, MLONG tablenumber, char *arguments, char *rhs, MLONG number)
- char * **ReadObject** (DBASE *d, MLONG tablenumber, char *arguments)
- char * **ReadijObject** (DBASE *d, MLONG i, MLONG j, char *arguments)
- int **ExistsObject** (DBASE *d, MLONG tablenumber, char *arguments)
- int **DeleteObject** (DBASE *d, MLONG tablenumber, char *arguments)

Variables

- int **withoutflush** = 0

4.28.1 Detailed Description

These are the low level functions for the database part of the tablebases. These routines have been copied (and then adapted) from the minos database program. This file goes together with minos.h

4.28.2 Macro Definition Documentation

4.28.2.1 CFD

```
#define CFD(
                y,
                s,
                type,
                x,
                j )
```

Value:

```
for(x=0,j=0;j<((int)sizeof(type));j++) \
x=(x<<8)+((*s++)&0x00FF); y=x;
```

Definition at line 55 of file minos.c.

4.28.2.2 CTD

```
#define CTD(
                y,
                s,
                type,
                x,
                j )
```

Value:

```
x=y;for(j=sizeof(type)-1;j>=0;j--){s[j]=x&0xFF; \
x>>=8;} s += sizeof(type);
```

Definition at line 57 of file minos.c.

4.29 minos.h File Reference

```
#include <stdio.h>
#include <stdlib.h>
#include <ctype.h>
#include <string.h>
#include <time.h>
```

Data Structures

- struct iniinfo
- struct objects
- struct indexblock
- struct nameblock
- struct dbase

Macros

- #define **MAXBASES** 16
- #define **NUMOBJECTS** 100
- #define **MAXINDEXSIZE** 33000000L
- #define **NAMETABLESIZE** 1008
- #define **ELEMENTSIZE** 128
- #define **TODISK** 0
- #define **FROMDISK** 1
- #define **MDIRTYFLAG** 1
- #define **MCLEANFLAG** (∼MDIRTYFLAG)
- #define **INANDOUT** 0
- #define **INPUTONLY** 1
- #define **OUTPUTONLY** 2
- #define **NOCOMPRESS** 4

Typedefs

- typedef struct iniinfo **INIINFO**
- typedef struct objects **OBJECTS**
- typedef struct indexblock **INDEXBLOCK**
- typedef struct nameblock **NAMESBLOCK**
- typedef struct dbase **DBASE**

Functions

- int **minosread** (FILE ∗f, char ∗buffer, MLONG size)
- int **minoswrite** (FILE ∗f, char ∗buffer, MLONG size)

Variables

- int **withoutflush**

4.29.1 Detailed Description

Contains all needed declarations and definitions for the tablebase low level file routines. These have been taken from the minos database system and modified somewhat.

!!!CAUTION!!! Changes in this file will most likely have consequences for the recovery mechanism (see checkpoint.c). You need to care for the code in checkpoint.c as well and modify the code there accordingly!

4.30 module.c File Reference

```
#include "form3.h"
```

Functions

- int **ModuleInstruction** (int *moduletype, int *specialtype)
- int **CoModuleOption** (UBYTE *s)
- int **CoModOption** (UBYTE *s)
- VOID **SetSpecialMode** (int moduletype, int specialtype)
- int **ExecModule** (int moduletype)
- int **ExecStore** ()
- VOID **FullCleanUp** ()
- int **DoPolyfun** (UBYTE *s)
- int **DoPolyratfun** (UBYTE *s)
- int **DoNoParallel** (UBYTE *s)
- int **DoParallel** (UBYTE *s)
- int **DoModSum** (UBYTE *s)
- int **DoModMax** (UBYTE *s)
- int **DoModMin** (UBYTE *s)
- int **DoModLocal** (UBYTE *s)
- int **DoProcessBucket** (UBYTE *s)
- UBYTE * **DoModDollar** (UBYTE *s, int type)
- int **DoinParallel** (UBYTE *s)
- int **DonotinParallel** (UBYTE *s)
- int **DoExecStatement** ()
- int **DoPipeStatement** ()

4.30.1 Detailed Description

A number of routines that deal with the moduleoption statement and the execution of modules. Additionally there are the execution of the exec and pipe instructions.

4.31 mpi.c File Reference

```
#include <limits.h>
#include "form3.h"
```

Data Structures

- struct longMultiStruct

Macros

- #define **PF_PACKSIZE** 1600
- #define MPI_ERRCODE_CHECK(err)

Typedefs

- typedef struct longMultiStruct **PF_LONGMULTI**

Functions

- LONG PF_RealTime (int i)
- int PF_LibInit (int *argcp, char ***argvp)
- int PF_LibTerminate (int error)
- int PF_Probe (int *src)
- int PF_ISendSbuf (int to, int tag)
- int PF_RecvWbuf (WORD *b, LONG *s, int *src)
- int PF_IRecvRbuf (PF_BUFFER *r, int bn, int from)
- int PF_WaitRbuf (PF_BUFFER *r, int bn, LONG *size)
- int PF_Bcast (void *buffer, int count)
- int PF_RawSend (int dest, void *buf, LONG l, int tag)
- LONG PF_RawRecv (int *src, void *buf, LONG thesize, int *tag)
- int PF_RawProbe (int *src, int *tag, int *bytesize)
- int PF_PrintPackBuf (char *s, int size)
- int PF_PreparePack (void)
- int PF_Pack (const void *buffer, size_t count, MPI_Datatype type)
- int PF_Unpack (void *buffer, size_t count, MPI_Datatype type)
- int PF_PackString (const UBYTE *str)
- int PF_UnpackString (UBYTE *str)
- int PF_Send (int to, int tag)
- int PF_Receive (int src, int tag, int *psrc, int *ptag)
- int PF_Broadcast (void)
- int PF_PrepareLongSinglePack (void)
- int PF_LongSinglePack (const void *buffer, size_t count, MPI_Datatype type)
- int PF_LongSingleUnpack (void *buffer, size_t count, MPI_Datatype type)
- int PF_LongSingleSend (int to, int tag)
- int PF_LongSingleReceive (int src, int tag, int *psrc, int *ptag)
- int PF_PrepareLongMultiPack (void)
- int PF_LongMultiPackImpl (const void *buffer, size_t count, size_t eSize, MPI_Datatype type)
- int PF_LongMultiUnpackImpl (void *buffer, size_t count, size_t eSize, MPI_Datatype type)
- int PF_LongMultiBroadcast (void)

Variables

- LONG **PF_maxDollarChunkSize** = 0

4.31.1 Detailed Description

MPI dependent functions of parform

This file contains all the functions for the parallel version of form3 that explicitly need to call mpi routines. This is the only file that really needs to be linked to the mpi-library!

4.31.2 Macro Definition Documentation

4.31.2.1 MPI_ERRCODE_CHECK

```
#define MPI_ERRCODE_CHECK(
                err )
```

Value:

```
    do { \
        int _tmp_err = (err); \
        if ( _tmp_err != MPI_SUCCESS ) return _tmp_err != 0 ? _tmp_err : -1; \
    } while (0)
```

A macro which exits the caller with a non-zero return value if *err* is not MPI_SUCCESS.

Parameters

err	The return value of a MPI function to be checked.

Remarks

The MPI standard defines MPI_SUCCESS == 0. Then (_tmp_err == 0) appears twice and we can expect the second evaluation will be eliminated by the compiler optimization.

Definition at line 84 of file mpi.c.

4.31.3 Function Documentation

4.31.3.1 PF_RealTime()

```
LONG PF_RealTime (
              int i )
```

Returns the realtime in 1/100 sec. as a LONG.

Parameters

i	the timer will be reset if i == 0.

Returns

the real elapsed time in 1/100 second.

Definition at line 101 of file mpi.c.

4.31.3.2 PF_LibInit()

```
int PF_LibInit (
              int * argcp,
              char *** argvp )
```

Performs all library dependent initializations.

Parameters

argcp	pointer to the number of arguments.
argvp	pointer to the arguments.

Returns

0 if OK, nonzero on error.

Definition at line 123 of file mpi.c.

4.31.3.3 PF_LibTerminate()

```
int PF_LibTerminate (
            int error )
```

Exits mpi, when there is an error either indicated or happening, returnvalue is 1, else returnvalue is 0.

Parameters

error	an error code.

Returns

0 if OK, nonzero on error.

Definition at line 209 of file mpi.c.

Referenced by PF_Terminate().

4.31.3.4 PF_Probe()

```
int PF_Probe (
            int * src )
```

Probes the next incoming message. If src == PF_ANY_SOURCE this operation is blocking, otherwise nonbloking.

Parameters

in,out	*src*	the source process number. In output, the process number of actual found source.

Returns

the tag value of the next incoming message if found, 0 if a nonbloking probe (input src != PF_ANY_SOURCE) did not find any messages. A negative returned value indicates an error.

Definition at line 230 of file mpi.c.

4.31.3.5 PF_ISendSbuf()

```
int PF_ISendSbuf (
            int to,
            int tag )
```

Nonblocking send operation. It sends everything from `buff` to `fill` of the active buffer. Depending on *tag* it also can do waiting for other sends to finish or set the active buffer to the next one.

Parameters

to	the destination process number.
tag	the message tag.

Returns

0 if OK, nonzero on error.

Definition at line 261 of file mpi.c.

4.31.3.6 PF_RecvWbuf()

```
int PF_RecvWbuf (
            WORD * b,
            LONG * s,
            int * src )
```

Blocking receive of a `WORD` buffer.

Parameters

`out`	*b*	the buffer to store the received data.
`in,out`	*s*	the size of the buffer. The output value is the actual size of the received data.
`in,out`	*src*	the source process number. The output value is the process number of actual source.

Returns

the received message tag. A negative value indicates an error.

Definition at line 337 of file mpi.c.

4.31.3.7 PF_IRecvRbuf()

```
int PF_IRecvRbuf (
            PF_BUFFER * r,
            int bn,
            int from )
```

Posts nonblocking receive for the active receive buffer. The buffer is filled from `full` to `stop`.

Parameters

r	the PF_BUFFER struct for the nonblocking receive.
bn	the index of the cyclic buffer.
from	the source process number.

Returns

0 if OK, nonzero on error.

Definition at line 366 of file mpi.c.

4.31.3.8 PF_WaitRbuf()

```
int PF_WaitRbuf (
            PF_BUFFER * r,
            int bn,
            LONG * size )
```

Waits for the buffer *bn* to finish a pending nonblocking receive. It returns the received tag and in `*size` the number of field received. If the receive is already finished, just return the flag and size, else wait for it to finish, but also check for other pending receives.

Parameters

	r	the PF_BUFFER struct for the pending nonblocking receive.
	bn	the index of the cyclic buffer.
out	*size*	the actual size of received data.

Returns

the received message tag. A negative value indicates an error.

Definition at line 400 of file mpi.c.

4.31.3.9 PF_Bcast()

```
int PF_Bcast (
             void * buffer,
             int count )
```

Broadcasts a message from the master to slaves.

Parameters

in,out	*buffer*	the starting address of buffer. The contents in this buffer on the master will be transferred to those on the slaves.
	count	the length of the buffer in bytes.

Returns

0 if OK, nonzero on error.

Definition at line 440 of file mpi.c.

4.31.3.10 PF_RawSend()

```
int PF_RawSend (
             int dest,
             void * buf,
             LONG l,
             int tag )
```

Sends *l* bytes from *buf* to *dest*. Returns 0 on success, or -1.

Parameters

dest	the destination process number.
buf	the send buffer.
l	the size of data in the send buffer in bytes.
tag	the message tag.

Returns

0 if OK, nonzero on error.

Definition at line 463 of file mpi.c.

Referenced by PF_SendFile().

4.31.3.11 PF_RawRecv()

```
LONG PF_RawRecv (
            int * src,
            void * buf,
            LONG thesize,
            int * tag )
```

Receives not more than *thesize* bytes from *src*, returns the actual number of received bytes, or -1 on failure.

Parameters

in,out	*src*	the source process number. In output, that of the actual received message.
out	*buf*	the receive buffer.
	thesize	the size of the receive buffer in bytes.
out	*tag*	the message tag of the actual received message.

Returns

the actual sizeof received data in bytes, or -1 on failure.

Definition at line 484 of file mpi.c.

4.31.3.12 PF_RawProbe()

```
int PF_RawProbe (
            int * src,
            int * tag,
            int * bytesize )
```

Probes an incoming message.

Parameters

in,out	*src*	the source process number. In output, that of the actual received message.
in,out	*tag*	the message tag. In output, that of the actual received message.
out	*bytesize*	the size of incoming data in bytes.

Returns

0 if OK, nonzero on error.

Definition at line 508 of file mpi.c.

4.31.3.13 PF_PrintPackBuf()

```
int PF_PrintPackBuf (
            char * s,
            int size )
```

Prints the contents in the pack buffer.

Parameters

s	a message to be shown.
size	the length of the buffer to be shown.

Returns

0 if OK, nonzero on error.

Definition at line 594 of file mpi.c.

4.31.3.14 PF_PreparePack()

```
int PF_PreparePack (
            void  )
```

Prepares for the next pack operations on the sender.

Returns

0 if OK, nonzero on error.

Definition at line 624 of file mpi.c.

4.31.3.15 PF_Pack()

```
int PF_Pack (
            const void * buffer,
            size_t count,
            MPI_Datatype type )
```

Adds data into the pack buffer.

Parameters

buffer	the pointer to the buffer storing the data to be packed.
count	the number of elements in the buffer.
type	the data type of elements in the buffer.

Returns

0 if OK, nonzero on error.

Definition at line 642 of file mpi.c.

4.31.3.16 PF_Unpack()

```
int PF_Unpack (
            void * buffer,
            size_t count,
            MPI_Datatype type )
```

Retrieves the next data in the pack buffer.

Parameters

out	*buffer*	the pointer to the buffer to store the unpacked data.
	count	the number of elements of data to be received.
	type	the data type of elements of data to be received.

Returns

0 if OK, nonzero on error.

Definition at line 671 of file mpi.c.

4.31.3.17 PF_PackString()

```
int PF_PackString (
            const UBYTE * str )
```

Packs a string *str* into the packed buffer PF_packbuf, including the trailing zero.

The first element (PF_INT) is the length of the packed portion of the string. If the string does not fit to the buffer PF↩ _packbuf, the function packs only the initial portion. It returns the number of packed bytes, so if (str[length-1]=='\0') then the whole string fits to the buffer, if not, then the rest (str+length) bust be packed and send again. On error, the function returns the negative error code.

One exception: the string "\0!\0" is used as an image of the NULL, so all 3 characters will be packed.

Parameters

str	a string to be packed.

Returns

the number of packed bytes, or a negative value on failure.

Definition at line 706 of file mpi.c.

4.31.3.18 PF_UnpackString()

```
int PF_UnpackString (
            UBYTE * str )
```

Unpacks a string to *str* from the packed buffer PF_packbuf, including the trailing zero.

It returns the number of unpacked bytes, so if (str[length-1]=='\0') then the whole string was unpacked, if not, then the rest must be appended to (str+length). On error, the function returns the negative error code.

Parameters

out	*str*	the buffer to store the unpacked string

Returns

the number of unpacked bytes, or a negative value on failure.

Definition at line 774 of file mpi.c.

4.31.3.19 PF_Send()

```
int PF_Send (
            int to,
            int tag )
```

Sends the contents in the pack buffer to the process specified by *to*.

Example:

```
if ( PF.me == SRC ) {
  PF_PreparePack();
  // Packing operations here...
  PF_Send(DEST, TAG);
}
else if ( PF.me == DEST ) {
  PF_Receive(SRC, TAG, &actual_src, &actual_tag);
  // Unpacking operations here...
}
```

Parameters

to	the destination process number.
tag	the message tag.

Returns

0 if OK, nonzero on error.

Definition at line 822 of file mpi.c.

4.31.3.20 PF_Receive()

```
int PF_Receive (
            int src,
            int tag,
            int * psrc,
            int * ptag )
```

Receives data into the pack buffer from the process specified by *src*. This function allows &src == psrc or &tag == ptag. Either *psrc* or *ptag* can be NULL.

See the example of PF_Send().

Parameters

	src	the source process number (can be PF_ANY_SOURCE).
	tag	the source message tag (can be PF_ANY_TAG).
out	*psrc*	the actual source process number of received message.
out	*ptag*	the received message tag.

Returns

0 if OK, nonzero on error.

Definition at line 848 of file mpi.c.

4.31.3.21 PF_Broadcast()

```
int PF_Broadcast (
            void  )
```

Broadcasts the contents in the pack buffer on the master to those on the slaves.

Example:

```
if ( PF.me == MASTER ) {
  PF_PreparePack();
  // Packing operations here...
}
PF_Broadcast();
if ( PF.me != MASTER ) {
  // Unpacking operations here...
}
```

Returns

0 if OK, nonzero on error.

Definition at line 883 of file mpi.c.

4.31.3.22 PF_PrepareLongSinglePack()

```
int PF_PrepareLongSinglePack (
            void  )
```

Prepares for the next long-single-pack operations on the sender.

Returns

0 if OK, nonzero on error.

Definition at line 1451 of file mpi.c.

4.31.3.23 PF_LongSinglePack()

```
int PF_LongSinglePack (
            const void * buffer,
            size_t count,
            MPI_Datatype type )
```

Adds data into the "long single" pack buffer.

Parameters

buffer	the pointer to the buffer storing the data to be packed.
count	the number of elements in the buffer.
type	the data type of elements in the buffer.

Returns

0 if OK, nonzero on error.

Definition at line 1469 of file mpi.c.

4.31.3.24 PF_LongSingleUnpack()

```
int PF_LongSingleUnpack (
            void * buffer,
            size_t count,
            MPI_Datatype type )
```

Retrieves the next data in the "long single" pack buffer.

Parameters

out	*buffer*	the pointer to the buffer to store the unpacked data.
	count	the number of elements of data to be received.
	type	the data type of elements of data to be received.

Returns

0 if OK, nonzero on error.

Definition at line 1503 of file mpi.c.

4.31.3.25 PF_LongSingleSend()

```
int PF_LongSingleSend (
              int to,
              int tag )
```

Sends the contents in the "long single" pack buffer to the process specified by *to*.

Example:

```
if ( PF.me == SRC ) {
  PF_PrepareLongSinglePack();
  // Packing operations here...
  PF_LongSingleSend(DEST, TAG);
}
else if ( PF.me == DEST ) {
  PF_LongSingleReceive(SRC, TAG, &actual_src, &actual_tag);
  // Unpacking operations here...
}
```

Parameters

to	the destination process number.
tag	the message tag.

Returns

0 if OK, nonzero on error.

Definition at line 1540 of file mpi.c.

4.31.3.26 PF_LongSingleReceive()

```
int PF_LongSingleReceive (
              int src,
              int tag,
              int * psrc,
              int * ptag )
```

Receives data into the "long single" pack buffer from the process specified by *src*. This function allows &src == psrc or &tag == ptag. Either *psrc* or *ptag* can be NULL.

See the example of PF_LongSingleSend().

Parameters

	src	the source process number (can be PF_ANY_SOURCE).
	tag	the source message tag (can be PF_ANY_TAG).
out	*psrc*	the actual source process number of received message.
out	*ptag*	the received message tag.

Returns

0 if OK, nonzero on error.

Definition at line 1583 of file mpi.c.

4.31.3.27 PF_PrepareLongMultiPack()

```
int PF_PrepareLongMultiPack (
            void  )
```

Prepares for the next long-multi-pack operations on the sender.

Returns

0 if OK, nonzero on error.

Definition at line 1643 of file mpi.c.

4.31.3.28 PF_LongMultiPackImpl()

```
int PF_LongMultiPackImpl (
            const void * buffer,
            size_t count,
            size_t eSize,
            MPI_Datatype type )
```

Adds data into the "long multi" pack buffer.

Parameters

buffer	the pointer to the buffer storing the data to be packed.
count	the number of elements in the buffer.
eSize	the byte size of each element of data.
type	the data type of elements in the buffer.

Returns

0 if OK, nonzero on error.

Definition at line 1662 of file mpi.c.

4.31.3.29 PF_LongMultiUnpackImpl()

```
int PF_LongMultiUnpackImpl (
            void * buffer,
            size_t count,
            size_t eSize,
            MPI_Datatype type )
```

Retrieves the next data in the "long multi" pack buffer.

Parameters

out	*buffer*	the pointer to the buffer to store the unpacked data.
	count	the number of elements of data to be received.
	eSize	the byte size of each element of data.
	type	the data type of elements of data to be received.

Returns

0 if OK, nonzero on error.

Definition at line 1721 of file mpi.c.

4.31.3.30 PF_LongMultiBroadcast()

```
int PF_LongMultiBroadcast (
            void  )
```

Broadcasts the contents in the "long multi" pack buffer on the master to those on the slaves.

Example:

```
if ( PF.me == MASTER ) {
  PF_PrepareLongMultiPack();
  // Packing operations here...
}
PF_LongMultiBroadcast();
if ( PF.me != MASTER ) {
  // Unpacking operations here...
}
```

Returns

0 if OK, nonzero on error.

Definition at line 1807 of file mpi.c.

4.32 mpidbg.h File Reference

```
#include <stdarg.h>
#include <stdio.h>
#include <string.h>
#include <mpi.h>
```

Macros

- #define **MPIDBG_RANK** MPIDBG_Get_rank()
- #define **MPIDBG_Out**(...) MPIDBG_Out(file, line, func, __VA_ARGS__)
- #define **MPIDBG_EXTARG** const char *file, int line, const char *func
- #define **MPI_Send**(...) MPIDBG_Send(__VA_ARGS__, __FILE__, __LINE__, __func__)
- #define **MPI_Recv**(...) MPIDBG_Recv(__VA_ARGS__, __FILE__, __LINE__, __func__)
- #define **MPI_Bsend**(...) MPIDBG_Bsend(__VA_ARGS__, __FILE__, __LINE__, __func__)
- #define **MPI_Ssend**(...) MPIDBG_Ssend(__VA_ARGS__, __FILE__, __LINE__, __func__)
- #define **MPI_Rsend**(...) MPIDBG_Rsend(__VA_ARGS__, __FILE__, __LINE__, __func__)
- #define **MPI_Isend**(...) MPIDBG_Isend(__VA_ARGS__, __FILE__, __LINE__, __func__)
- #define **MPI_Ibsend**(...) MPIDBG_Ibsend(__VA_ARGS__, __FILE__, __LINE__, __func__)
- #define **MPI_Issend**(...) MPIDBG_Issend(__VA_ARGS__, __FILE__, __LINE__, __func__)
- #define **MPI_Irsend**(...) MPIDBG_Irsend(__VA_ARGS__, __FILE__, __LINE__, __func__)
- #define **MPI_Irecv**(...) MPIDBG_Irecv(__VA_ARGS__, __FILE__, __LINE__, __func__)
- #define **MPI_Wait**(...) MPIDBG_Wait(__VA_ARGS__, __FILE__, __LINE__, __func__)
- #define **MPI_Test**(...) MPIDBG_Test(__VA_ARGS__, __FILE__, __LINE__, __func__)
- #define **MPI_Waitany**(...) MPIDBG_Waitany(__VA_ARGS__, __FILE__, __LINE__, __func__)
- #define **MPI_Testany**(...) MPIDBG_Testany(__VA_ARGS__, __FILE__, __LINE__, __func__)
- #define **MPI_Waitall**(...) MPIDBG_Waitall(__VA_ARGS__, __FILE__, __LINE__, __func__)
- #define **MPI_Testall**(...) MPIDBG_Testall(__VA_ARGS__, __FILE__, __LINE__, __func__)
- #define **MPI_Waitsome**(...) MPIDBG_Waitsome(__VA_ARGS__, __FILE__, __LINE__, __func__)
- #define **MPI_Testsome**(...) MPIDBG_Testsome(__VA_ARGS__, __FILE__, __LINE__, __func__)
- #define **MPI_Iprobe**(...) MPIDBG_Iprobe(__VA_ARGS__, __FILE__, __LINE__, __func__)
- #define **MPI_Probe**(...) MPIDBG_Probe(__VA_ARGS__, __FILE__, __LINE__, __func__)
- #define **MPI_Cancel**(...) MPIDBG_Cancel(__VA_ARGS__, __FILE__, __LINE__, __func__)
- #define **MPI_Test_cancelled**(...) MPIDBG_Test_cancelled(__VA_ARGS__, __FILE__, __LINE__, __func↩ __)
- #define **MPI_Barrier**(...) MPIDBG_Barrier(__VA_ARGS__, __FILE__, __LINE__, __func__)
- #define **MPI_Bcast**(...) MPIDBG_Bcast(__VA_ARGS__, __FILE__, __LINE__, __func__)

4.32.1 Detailed Description

MPI APIs with the logging feature. NOTE: This file needs C99.

4.33 names.c File Reference

```
#include "form3.h"
```

Functions

- NAMENODE * **GetNode** (NAMETREE *nametree, UBYTE *name)
- int **AddName** (NAMETREE *nametree, UBYTE *name, WORD type, WORD number, int *nodenum)
- int **GetName** (NAMETREE *nametree, UBYTE *name, WORD *number, int par)
- UBYTE * **GetFunction** (UBYTE *s, WORD *funnum)
- UBYTE * **GetNumber** (UBYTE *s, WORD *num)
- int **GetLastExprName** (UBYTE *name, WORD *number)
- int **GetOName** (NAMETREE *nametree, UBYTE *name, WORD *number, int par)
- int **GetAutoName** (UBYTE *name, WORD *number)
- int **GetVar** (UBYTE *name, WORD *type, WORD *number, int wantedtype, int par)
- WORD **EntVar** (WORD type, UBYTE *name, WORD x, WORD y, WORD z, WORD d)
- int **GetDollar** (UBYTE *name)
- VOID **DumpTree** (NAMETREE *nametree)
- VOID **DumpNode** (NAMETREE *nametree, WORD node, WORD depth)
- int **CompactifyTree** (NAMETREE *nametree, WORD par)
- VOID **CopyTree** (NAMETREE *newtree, NAMETREE *oldtree, WORD node, WORD par)
- VOID **LinkTree** (NAMETREE *tree, WORD offset, WORD numnodes)
- NAMETREE * **MakeNameTree** ()
- VOID **FreeNameTree** (NAMETREE *n)
- void **ClearWildcardNames** ()
- int **AddWildcardName** (UBYTE *name)
- int **GetWildcardName** (UBYTE *name)
- int **AddSymbol** (UBYTE *name, int minpow, int maxpow, int cplx, int dim)
- int **CoSymbol** (UBYTE *s)
- int **AddIndex** (UBYTE *name, int dim, int dim4)
- int **CoIndex** (UBYTE *s)
- UBYTE * **DoDimension** (UBYTE *s, int *dim, int *dim4)
- int **CoDimension** (UBYTE *s)
- int **AddVector** (UBYTE *name, int cplx, int dim)
- int **CoVector** (UBYTE *s)
- int **AddFunction** (UBYTE *name, int comm, int istensor, int cplx, int symprop, int dim, int argmax, int argmin)
- int **CoCommuteInSet** (UBYTE *s)
- int **CoFunction** (UBYTE *s, int comm, int istensor)
- int **CoNFunction** (UBYTE *s)
- int **CoCFunction** (UBYTE *s)
- int **CoNTensor** (UBYTE *s)
- int **CoCTensor** (UBYTE *s)
- int **DoTable** (UBYTE *s, int par)
- int **CoTable** (UBYTE *s)
- int **CoNTable** (UBYTE *s)
- int **CoCTable** (UBYTE *s)
- void **EmptyTable** (TABLES T)
- int **AddSet** (UBYTE *name, WORD dim)
- int **DoElements** (UBYTE *s, SETS set, UBYTE *name)
- int **CoSet** (UBYTE *s)
- int **DoTempSet** (UBYTE *from, UBYTE *to)
- int **CoAuto** (UBYTE *inp)
- int **AddDollar** (UBYTE *name, WORD type, WORD *start, LONG size)
- int **ReplaceDollar** (WORD number, WORD newtype, WORD *newstart, LONG newsize)
- int **AddDubious** (UBYTE *name)
- int **MakeDubious** (NAMETREE *nametree, UBYTE *name, WORD *number)
- int **NameConflict** (int type, UBYTE *name)
- int **AddExpression** (UBYTE *name, int x, int y)
- int **GetLabel** (UBYTE *name)
- void **ResetVariables** (int par)
- void **RemoveDollars** ()
- void **Globalize** (int par)
- int **TestName** (UBYTE *name)

4.33.1 Detailed Description

The complete names administration. All variables with a name have to pass here to be properly registered, have structs of the proper type assigned to them etc. The file also contains the utility routines for maintaining the balanced trees that make searching for names rather fast.

4.34 normal.c File Reference

```
#include "form3.h"
```

Macros

- #define **MAXNUMBEROFNONCOMTERMS** 2

Functions

- WORD **CompareFunctions** (WORD ∗fleft, WORD ∗fright)
- WORD **Commute** (WORD ∗fleft, WORD ∗fright)
- WORD **Normalize** (PHEAD WORD ∗term)
- WORD **ExtraSymbol** (WORD sym, WORD pow, WORD nsym, WORD ∗ppsym, WORD ∗ncoef)
- WORD **DoTheta** (PHEAD WORD ∗t)
- WORD **DoDelta** (WORD ∗t)
- void **DoRevert** (WORD ∗fun, WORD ∗tmp)
- WORD **DetCommu** (WORD ∗terms)
- WORD **DoesCommu** (WORD ∗term)
- int **TreatPolyRatFun** (PHEAD WORD ∗prf)
- void **DropCoefficient** (PHEAD WORD ∗term)
- void **DropSymbols** (PHEAD WORD ∗term)
- int SymbolNormalize (WORD ∗term)
- int **TestFunFlag** (PHEAD WORD ∗tfun)
- WORD **BracketNormalize** (PHEAD WORD ∗term)

4.34.1 Detailed Description

Mainly the routine Normalize. This routine brings terms to standard FORM. Currently it has one serious drawback. Its buffers are all in the stack. This means these buffers have a fixed size (NORMSIZE). In the past this has caused problems and NORMSIZE had to be increased.

It is not clear whether Normalize can be called recursively.

4.34.2 Function Documentation

4.34.2.1 SymbolNormalize()

```
int SymbolNormalize (
            WORD * term )
```

Routine normalizes terms that contain only symbols. Regular minimum and maximum properties are ignored.

We check whether there are negative powers in the output. This is not allowed.

Definition at line 5000 of file normal.c.

4.35 notation.c File Reference

```
#include "form3.h"
```

Functions

- int **NormPolyTerm** (PHEAD WORD ∗term)
- int **ConvertToPoly** (PHEAD WORD ∗term, WORD ∗outterm, WORD ∗comlist, WORD par)
- int LocalConvertToPoly (PHEAD WORD ∗term, WORD ∗outterm, WORD startebuf, WORD par)
- int **ConvertFromPoly** (PHEAD WORD ∗term, WORD ∗outterm, WORD from, WORD to, WORD offset, WORD par)
- WORD **FindSubterm** (WORD ∗subterm)
- WORD **FindLocalSubterm** (PHEAD WORD ∗subterm, WORD startebuf)
- void **PrintSubtermList** (int from, int to)
- void **PrintExtraSymbol** (int num, WORD ∗terms, int par)
- WORD **FindSubexpression** (WORD ∗subexpr)
- int **ExtraSymFun** (PHEAD WORD ∗term, WORD level)
- int **PruneExtraSymbols** (WORD downto)

4.35.1 Detailed Description

Contains the functions that deal with the rewriting and manipulation of expressions/terms in polynomial representation.

4.35.2 Function Documentation

4.35.2.1 LocalConvertToPoly()

```
int LocalConvertToPoly (
            PHEAD WORD * term,
            WORD * outterm,
            WORD startebuf,
            WORD par )
```

Converts a generic term to polynomial notation in which there are only symbols and brackets. During conversion there will be only symbols. Brackets are stripped. Objects that need 'translation' are put inside a special compiler buffer and represented by a symbol. The numbering of the extra symbols is down from the maximum. In principle there can be a problem when running into the already assigned ones. This uses the FindTree for searching in the global tree and then looks further in the AT.ebufnum. This allows fully parallel processing. Hence we need no locks. Cannot be used in the same module as ConvertToPoly.

Definition at line 510 of file notation.c.

4.36 opera.c File Reference

```
#include "form3.h"
```

Functions

- WORD **EpfFind** (PHEAD WORD ∗term, WORD ∗params)
- WORD **EpfCon** (PHEAD WORD ∗term, WORD ∗params, WORD num, WORD level)
- WORD **EpfGen** (WORD number, WORD ∗inlist, WORD ∗kron, WORD ∗perm, WORD sgn)
- WORD **Trick** (WORD ∗in, TRACES ∗t)
- WORD **Trace4no** (WORD number, WORD ∗kron, TRACES ∗t)
- WORD **Trace4** (PHEAD WORD ∗term, WORD ∗params, WORD num, WORD level)
- WORD **Trace4Gen** (PHEAD TRACES ∗t, WORD number)
- WORD **TraceNno** (WORD number, WORD ∗kron, TRACES ∗t)
- WORD **TraceN** (PHEAD WORD ∗term, WORD ∗params, WORD num, WORD level)
- WORD **TraceNgen** (PHEAD TRACES ∗t, WORD number)
- WORD **Traces** (PHEAD WORD ∗term, WORD ∗params, WORD num, WORD level)
- WORD **TraceFind** (PHEAD WORD ∗term, WORD ∗params)
- WORD **Chisholm** (PHEAD WORD ∗term, WORD level)
- WORD **TenVecFind** (PHEAD WORD ∗term, WORD ∗params)
- WORD **TenVec** (PHEAD WORD ∗term, WORD ∗params, WORD num, WORD level)

4.36.1 Detailed Description

Contains the 'operations' These are the trace routines, the contractions of the Levi-Civita tensors and the tensor to vector/vector to tensor routines. The trace and contraction routines are done in a special way (see commentary with the FIXEDGLOBALS struct)

4.37 optimize.cc File Reference

```
#include <vector>
#include <stack>
#include <algorithm>
#include <set>
#include <map>
#include <climits>
#include <cmath>
#include <string>
#include <iostream>
#include <tr1/unordered_map>
#include <tr1/unordered_set>
#include "form3.h"
#include "mytime.h"
```

Data Structures

- class tree_node
- struct CSEHash
- struct CSEEq
- struct node
- struct NodeHash
- struct NodeEq
- class optimization

Typedefs

- typedef struct node **NODE**

Functions

- void **print_instr** (const vector< WORD > &instr, WORD num)
- template<class RandomAccessIterator >
  void my_random_shuffle (PHEAD RandomAccessIterator fr, RandomAccessIterator to)
- LONG get_expression (int exprnr)
- vector< vector< WORD > > get_brackets ()
- int count_operators (const WORD *expr, bool print=false)
- int count_operators (const vector< WORD > &instr, bool print=false)
- vector< WORD > occurrence_order (const WORD *expr, bool rev)
- WORD getpower (const WORD *term, int var, int pos)
- void fixarg (UWORD *t, WORD &n)
- void **GcdLong_fix_args** (PHEAD UWORD *a, WORD na, UWORD *b, WORD nb, UWORD *c, WORD *nc)
- void **DivLong_fix_args** (UWORD *a, WORD na, UWORD *b, WORD nb, UWORD *c, WORD *nc, UWORD *d, WORD *nd)
- void build_Horner_tree (const WORD **terms, int numterms, int var, int maxvar, int pos, vector< WORD > *res)
- bool term_compare (const WORD *a, const WORD *b)
- vector< WORD > Horner_tree (const WORD *expr, const vector< WORD > &order)
- void **print_tree** (const vector< WORD > &tree)

- template<typename T >
  size_t **hash_range** (T *array, int size)
- vector< WORD > generate_instructions (const vector< WORD > &tree, bool do_CSE)
- int count_operators_cse (const vector< WORD > &tree)
- NODE * **buildTree** (vector< WORD > &tree)
- int **count_operators_cse_topdown** (vector< WORD > &tree)
- vector< WORD > **simulated_annealing** ()
- void **find_Horner_MCTS_expand_tree** ()
- void find_Horner_MCTS ()
- vector< WORD > merge_operators (const vector< WORD > &all_instr, bool move_coeff)
- vector< optimization > find_optimizations (const vector< WORD > &instr)
- bool do_optimization (const optimization optim, vector< WORD > &instr, int newid)
- int partial_factorize (vector< WORD > &instr, int n, int improve)
- vector< WORD > optimize_greedy (vector< WORD > instr, LONG time_limit)
- vector< WORD > recycle_variables (const vector< WORD > &all_instr)
- void optimize_expression_given_Horner ()
- VOID generate_output (const vector< WORD > &instr, int exprnr, int extraoffset, const vector< vector< WORD > > &brackets)
- WORD generate_expression (WORD exprnr)
- VOID optimize_print_code (int print_expr)
- int Optimize (WORD exprnr, int do_print)
- int ClearOptimize ()

Variables

- const WORD **OPER_ADD** = -1
- const WORD **OPER_MUL** = -2
- const WORD **OPER_COMMA** = -3
- WORD * **optimize_expr**
- vector< vector< WORD > > **optimize_best_Horner_schemes**
- int **optimize_num_vars**
- int **optimize_best_num_oper**
- vector< WORD > **optimize_best_instr**
- vector< WORD > **optimize_best_vars**
- bool **mcts_factorized**
- bool **mcts_separated**
- vector< WORD > **mcts_vars**
- tree_node **mcts_root**
- int **mcts_expr_score**
- set< pair< int, vector< WORD > > > **mcts_best_schemes**

4.37.1 Detailed Description

experimental routines for the optimization of FORTRAN or C output.

4.37.2 Function Documentation

4.37.2.1 my_random_shuffle()

```
template<class RandomAccessIterator >
void my_random_shuffle (
            PHEAD RandomAccessIterator fr,
            RandomAccessIterator to )
```

Random shuffle

4.37.3 Description

Randomly permutes elements in the range [fr,to). Functionality is the same as C++'s "random_shuffle", but it uses Form's "wranf".

Definition at line 180 of file optimize.cc.

4.37.3.1 get_expression()

```
LONG get_expression (
            int exprnr )
```

Get expression

4.37.4 Description

Reads an expression from the input file into a buffer (called optimize_expr). This buffer is used during the optimization process. Non-symbols are removed by ConvertToPoly and are put in temporary symbols.

The return value is the length of the expression in WORDs, or a negative number if it failed.

Definition at line 204 of file optimize.cc.

4.37.4.1 get_brackets()

```
vector<vector<WORD> > get_brackets ( )
```

Get brackets

4.37.5 Description

Checks whether the input expression (stored in optimize_expr) contains brackets. If so, this method replaces terms outside brackets by powers of SEPERATESYMBOL (equal brackets have equal powers) and the brackets are returned. If not, the result is empty.

Brackets are used for simultaneous optimization. The symbol SEPARATESYMBOL is always the first one used in a Horner scheme.

Definition at line 281 of file optimize.cc.

4.37.5.1 count_operators() [1/2]

```
int count_operators (
            const WORD * expr,
            bool print = false )
```

Count operators

4.37.6 Description

Counts the number of operators in a Form-style expression.

Definition at line 401 of file optimize.cc.

Referenced by optimize_greedy().

4.37.6.1 count_operators() [2/2]

```
int count_operators (
            const vector< WORD > & instr,
            bool print = false )
```

Count operators

4.37.7 Description

Counts the number of operators in a vector of instructions

Definition at line 455 of file optimize.cc.

4.37.7.1 occurrence_order()

```
vector<WORD> occurrence_order (
            const WORD * expr,
            bool rev )
```

Occurrence order

4.37.8 Description

Extracts all variables from an expression and sorts them with most occurring first (or last, with rev=true)

Definition at line 498 of file optimize.cc.

4.37.8.1 getpower()

```
WORD getpower (
             const WORD * term,
             int var,
             int pos )
```

Horner tree building

4.37.9 Description

Given a Form-style expression (in a buffer in memory), this builds an expression tree. The tree is determined by a multivariate Horner scheme, i.e., something of the form:

1+y+x∗(2+y∗(1+y)+x∗(3-y∗(...)))

The order of the variables is given to the method "Horner_tree", which renumbers ad reorders the terms of the expression. Next, the recursive method "build_Horner_tree" does the actual tree construction.

The tree is represented in postfix notation. Tokens are of the following forms:

- SNUMBER tokenlength num den coefflength
- SYMBOL tokenlength variable power
- OPER_ADD or OPER_MUL

4.37.10 Note

Sets AN.poly_num_vars and allocates AN.poly_vars. The latter should be freed later. Get power of variable (helper function for build_Horner_tree)

4.37.11 Description

Returns the power of the variable "var", which is at position "pos" in this term, if it is present.

Definition at line 579 of file optimize.cc.

4.37.11.1 fixarg()

```
void fixarg (
             UWORD * t,
             WORD & n )
```

Call GcdLong/DivLong with leading zeroes

4.37.12 Description

These method remove leading zeroes, so that GcdLong and DivLong can safely be called.

Definition at line 593 of file optimize.cc.

4.37.12.1 build_Horner_tree()

```
void build_Horner_tree (
             const WORD ** terms,
             int numterms,
             int var,
             int maxvar,
             int pos,
             vector< WORD > * res )
```

Build the Horner tree

4.37.13 Description

Constructs the Horner tree. The method processes one variable and continues recursively until the Horner scheme is completed.

"terms" is a pointer to the starts of the terms. "numterms" is the number of terms to be processed. "var" is the next variable to be processed (index between 0 and #maxvar) and "maxvar" is the last variable to be processed, so that partial Horner trees can also be constructed. "pos" is the position that the power of "var" should be in (one level further in the recursion, "pos" has increased by 0 or 1 depending on whether the previous power was 0 or not). The result is written at the pointer "res".

This method also factors out gcds of the coefficients. The result should end with "gcd OPER_MUL" at all times, so that one level higher gcds can be extracted again.

Definition at line 631 of file optimize.cc.

4.37.13.1 term_compare()

```
bool term_compare (
             const WORD * a,
             const WORD * b )
```

Term compare (helper function for Horner_tree)

4.37.14 Description

Compares two terms of the form "L SYM 4 x n coeff" or "L coeff". Lower powers of lower-indexed symbols come first. This is used to sort the terms in correct order.

Definition at line 831 of file optimize.cc.

4.37.14.1 Horner_tree()

```
vector<WORD> Horner_tree (
            const WORD * expr,
            const vector< WORD > & order )
```

Prepare Horner tree building

4.37.15 Description

This method renumbers the variables to 0...#vars-1 according to the specified order. Next, it stored pointer to individual terms and sorts the terms with higher power first. Then the sorted lists of power is used for the construction of the Horner tree.

Definition at line 852 of file optimize.cc.

4.37.15.1 generate_instructions()

```
vector<WORD> generate_instructions (
            const vector< WORD > & tree,
            bool do_CSE )
```

Generate instructions

4.37.16 Description

Converts the expression tree to a list of instructions that directly translate to code. Instructions are of the form:

expr.nr operator length [operands]+ trailing.zero

The operands are of the form:

length [(EXTRA)SYMBOL length variable power] coeff

This method only generates binary operators. Merging is done later. The method also checks for common subexpressions and eliminates them and the flag "do_CSE" is set.

4.37.17 Implementation details

The map "ID" keeps track of which subexpressions already exist. The key is formatted as one of the following:

SYMBOL x n SNUMBER coeff OPERATOR LHS RHS

with LHS/RHS formatted as one of the following:

SNUMBER idx 0 (EXTRA)SYMBOL x n

ID[symbol] or ID[operator] equals a subexpression number. ID[coeff] equals the position of the number in the input.

The stack s is used the process the postfix expression tree. Three-word tokens of the form:

SNUMBER idx.of.coeff 0 SYMBOL x n EXTRASYMBOL x 1

are pushed onto it. Operators pop two operands and push the resulting expression.

(Extra)symbols are 1-indexed, because -X is also needed to represent -1 times this term.

There is currently a bug. The notation cannot tell if there is a single bracket and then ignores the bracket.

TODO: check if this method performs properly if do_CSE=false

Definition at line 1086 of file optimize.cc.

4.37.17.1 count_operators_cse()

```
int count_operators_cse (
            const vector< WORD > & tree )
```

Count number of operators in a binary tree, while removing CSEs on the fly. The instruction set is not created, which makes this method slightly faster.

A hash is created on the fly and is passed through the stack. TODO: find better hash functions

Definition at line 1295 of file optimize.cc.

4.37.17.2 find_Horner_MCTS()

```
void find_Horner_MCTS ( )
```

Find best Horner schemes using MCTS

4.37.18 Description

The method governs the MCTS for the best Horner schemes. It does some pre-processing, calls "find_Horner_↩ MCTS_expand_tree" a number of times and does some post-processing.

Definition at line 2208 of file optimize.cc.

4.37.18.1 merge_operators()

```
vector<WORD> merge_operators (
            const vector< WORD > & all_instr,
            bool move_coeff )
```

Merge operators

4.37.19 Description

The input instructions form a binary DAG. This method merges expressions like

Z1 = a+b; Z2 = Z1+c;

into

Z2 = a+b+c;

An instruction is merged iff it only has one parent and the operator equals its parent's operator.

This still leaves some freedom: where should the coefficients end up in cases as:

Z1 = Z2 + x <=> Z1 = 2∗Z2 + x Z2 = 2∗x∗y Z2 = x∗y

Both are relevant, e.g. for CSE of the form "2∗x" and "2∗Z2". The flag "move_coeff" moves coefficients from LHS-like expressions to RHS-like expressions.

Furthermore, this method removes empty equation (Z1=0), that are introduced by some "optimize_greedy" substitutions.

4.37.20 Implementation details

Expressions are mostly traversed via a stack, so that parents are evaluated before their children.

With "move_coeff" set coefficients are moved, but this leads to some tricky cases, e.g.

Z1 = Z2 + x Z2 = 2∗y

Here Z2 reduces to the trivial equation Z2=y, which should be eliminated. Here the array skip[i] comes in.

Furthermore in the case

Z1 = Z2 + x Z2 = 2∗Z3 Z3 = x∗Z4 Z4 = y∗z

after substituting Z1 = 2∗Z3 + x, the parent expression for Z4 becomes Z3 instead of Z2. This is where renum_par[i] comes in.

Finally, once a coefficient has been moved, skip_coeff[i] is set and this coefficient is copied into the new expression anymore.

Definition at line 2356 of file optimize.cc.

4.37.20.1 find_optimizations()

```
vector<optimization> find_optimizations (
            const vector< WORD > & instr )
```

Find optimizations

4.37.21 Description

This method find all optimization of the form described in "class Optimization". It process every equation, looking for possible optimizations and stores them in a fast-access data structure to count the total improvement of an optimization.

Definition at line 2651 of file optimize.cc.

4.37.21.1 do_optimization()

```
bool do_optimization (
            const optimization optim,
            vector< WORD > & instr,
            int newid )
```

Do optimization

4.37.22 Description

This method performs an optimization. It scans through the equations of "optim.eqnidxs" and looks in which this optimization can still be performed (due to other performed optimizations this isn't always the case). If possible, it substitutes the common subexpression by a new extra symbol numbered "newid". Finally, the new extrasymbol is defined accordingly.

Substitutions may lead to trivial equations of the form "Zi=Zj", but these are removed in the end of the method. The method returns whether the substitution has been done once or more (or not).

Definition at line 2884 of file optimize.cc.

4.37.22.1 partial_factorize()

```
int partial_factorize (
            vector< WORD > & instr,
            int n,
            int improve )
```

Partial factorization of instructions

4.37.23 Description

This method performs partial factorization of instructions. In particular the following instructions

Z1 = x∗a∗b Z2 = x∗c∗d∗e Z3 = 2∗x + Z1 + Z2 + more

are replaced by

Z1 = a∗b Z2 = c∗d∗e Z3 = Zj + more Zi = 2 + Z1 + Z2 Zj = x∗Zi

Here it is necessary that no other equations refer to Z1 and Z2. The generation of trivial instructions (Zi=Zj or Zi=x) is prevented.

Definition at line 3433 of file optimize.cc.

4.37.23.1 optimize_greedy()

```
vector<WORD> optimize_greedy (
            vector< WORD > instr,
            LONG time_limit )
```

Optimize instructions greedily

4.37.24 Description

This method optimizes an expression greedily. It calls "find_optimizations" to obtain candidates and performs the best one(s) by calling "do_optimization".

How many different optimization are done, before "find_optimization" is called again, is determined by the settings "greedyminnum" and "greedymaxperc".

During the optimization process, sequences of zeroes are introduced in the instructions, since moving all instructions when one gets optimized, is very costly. Therefore, in the end, the instructions are "compressed" again to remove these extra zeroes.

Definition at line 3727 of file optimize.cc.

References count_operators().

4.37.24.1 recycle_variables()

```
vector<WORD> recycle_variables (
             const vector< WORD > & all_instr )
```

Recycle variables

4.37.25 Description

The current input uses many temporary variables. Many of them become obsolete at some point during the evaluation of the code, so can be recycled. This method renumbers the temporary variables, so that they are recycled. Furthermore, the input is order in depth-first order, so that the instructions can be performed consecutively.

4.37.26 Implementation details

First, for each subDAG, an estimate for the number of variables needed is made. This is done by the following recursive formula:

#vars(x) = max(#vars(ch_i(x)) + i),

with ch_i(x) the i-th child of x, where the childs are ordered w.r.t. #vars(ch_i). This formula is exact if the input forms a tree, and otherwise gives a reasonable estimate.

Then, the instructions are reordered in a depth-first order with childs ordered w.r.t. #vars. Next, the times that variables become obsolete are found. Each LHS of an instruction is renumbered to the lowest-numbered temporary variable that is available at that time.

Definition at line 3867 of file optimize.cc.

4.37.26.1 optimize_expression_given_Horner()

```
void optimize_expression_given_Horner ( )
```

Optimize expression given a Horner scheme

4.37.27 Description

This method picks one Horner scheme from the list of best Horner schemes, applies this scheme to the expression and then, depending on optimize.settings, does a common subexpression elimination (CSE) or performs greedy optimizations.

CSE is fast, while greedy might be slow. CSE followed by greedy is faster than greedy alone, but typically results in slightly worse code (not proven; just observed). eventually do greedy optimations

Definition at line 4014 of file optimize.cc.

4.37.27.1 generate_output()

```
VOID generate_output (
            const vector< WORD > & instr,
            int exprnr,
            int extraoffset,
            const vector< vector< WORD > > & brackets )
```

Generate output

4.37.28 Description

This method prepares the instructions for printing. They are stored in Form format, so that they can be printed by "PrintExtraSymbol". The results are stored in the buffer AO.OptimizeResult.

Definition at line 4245 of file optimize.cc.

4.37.28.1 generate_expression()

```
WORD generate_expression (
            WORD exprnr )
```

Generate expression

4.37.29 Description

This method modifies the original optimized expression by an expression with extra symbols. This is used for "#Optimize".

Definition at line 4394 of file optimize.cc.

4.37.29.1 optimize_print_code()

```
VOID optimize_print_code (
            int print_expr )
```

Print optimized code

4.37.30 Description

This method prints the optimized code via "PrintExtraSymbol". Depending on the flag, the original expression is printed (for "Print") or not (for "#Optimize / #write "O").

Definition at line 4474 of file optimize.cc.

4.37.30.1 Optimize()

```
int Optimize (
            WORD exprnr,
            int do_print )
```

Optimization of expression

4.37.31 Description

This method takes an input expression and generates optimized code to calculate its value. The following methods are called to do so:

(1) get_expression : to read to expression

(2) get_brackets : find brackets for simultaneous optimization

(3) occurrence_order or find_Horner_MCTS : to determine (the) Horner scheme(s) to use; this depends on AO.↩ optimize.horner

(4) optimize_expression_given_Horner : to do the optimizations for each Horner scheme; this method does either CSE or greedy optimizations dependings on AO.optimize.method

(5) generate_output : to format the output in Form notation and store it in a buffer

(6a) optimize_print_code : to print the expression (for "Print") or (6b) generate_expression : to modify the expression (for "#Optimize")

On ParFORM, all the processes must call this function at the same time. Then

(1) Because only the master can access to the expression to be optimized, the master broadcast the expression to all the slaves after reading the expression (PF_get_expression).

(2) get_brackets reads optimize_expr as the input and it works also on the slaves. We leave it although the bracket information is not needed on the slaves (used in (5) on the master).

(3) and (4) find_Horner_MCTS and optimize_expression_given_Horner are parallelized.

(5), (6a) and (6b) are needed only on the master.

Definition at line 4587 of file optimize.cc.

4.37.31.1 ClearOptimize()

```
int ClearOptimize ( )
```

Optimization of expression

4.37.32 Description

Clears the buffers that were used for optimization output. Clears the expression from the buffers (marks it to be dropped). Note: we need to use the expression by its name, because numbers may change if we drop other expressions between when we do the optimizations and clear the results (in execute.c). Also this is not 100% safe, because we could overwrite the optimized expression. But that can be done only in a Local or Global statement and hence we only have to test there that we might have to call ClearOptimize first. (in file comexpr.c)

Definition at line 4924 of file optimize.cc.

4.38 parallel.c File Reference

```
#include "form3.h"
#include "vector.h"
```

Data Structures

- struct NoDe
- struct dollar_buf
- struct bufIPstruct

Macros

- #define **PRINTFBUF**(TEXT, TERM, SIZE) {}
- #define SWAP(x, y)
- #define PACK_LONG(p, n)
- #define UNPACK_LONG(p, n)
- #define CHECK(condition) _CHECK(condition, __FILE__, __LINE__)
- #define **_CHECK**(condition, file, line) __CHECK(condition, file, line)
- #define **__CHECK**(condition, file, line)
- #define **DBGOUT**(lv1, lv2, a) do { if (lv1 >= lv2) { printf a; fflush(stdout); } } while (0)
- #define **DBGOUT_NINTERMS**(lv, a)
- #define **PF_STATS_SIZE** 5
- #define **PF_SNDFILEBUFSIZE** 4096
- #define **recvBuffer** logBuffer /∗ (master) The buffer for receiving messages. ∗/

Typedefs

- typedef struct NoDe NODE
- typedef struct bufIPstruct **bufIPstruct_t**

Functions

- LONG PF_RealTime (int)
- int PF_LibInit (int *, char ***)
- int PF_LibTerminate (int)
- int PF_Probe (int *)
- int PF_RecvWbuf (WORD *, LONG *, int *)
- int PF_IRecvRbuf (PF_BUFFER *, int, int)
- int PF_WaitRbuf (PF_BUFFER *, int, LONG *)
- int PF_RawSend (int dest, void *buf, LONG l, int tag)
- LONG PF_RawRecv (int *src, void *buf, LONG thesize, int *tag)
- int PF_RawProbe (int *src, int *tag, int *bytesize)
- int PF_EndSort (void)
- WORD PF_Deferred (WORD *term, WORD level)
- int PF_Processor (EXPRESSIONS e, WORD i, WORD LastExpression)
- int PF_Init (int *argc, char ***argv)
- int PF_Terminate (int errorcode)
- LONG PF_GetSlaveTimes (void)
- LONG PF_BroadcastNumber (LONG x)
- void PF_BroadcastBuffer (WORD **buffer, LONG *length)
- int PF_BroadcastString (UBYTE *str)
- int PF_BroadcastPreDollar (WORD **dbuffer, LONG *newsize, int *numterms)
- int PF_CollectModifiedDollars (void)
- int PF_BroadcastModifiedDollars (void)
- int PF_BroadcastRedefinedPreVars (void)
- int **PF_StoreInsideInfo** (void)
- int **PF_RestoreInsideInfo** (void)
- int PF_BroadcastCBuf (int bufnum)
- int PF_BroadcastExpFlags (void)
- int PF_BroadcastExpr (EXPRESSIONS e, FILEHANDLE *file)
- int PF_BroadcastRHS (void)
- int PF_InParallelProcessor (void)
- int PF_SendFile (int to, FILE *fd)
- int PF_RecvFile (int from, FILE *fd)
- void PF_MLock (void)
- void PF_MUnlock (void)
- LONG PF_WriteFileToFile (int handle, UBYTE *buffer, LONG size)
- void PF_FlushStdOutBuffer (void)
- void PF_FreeErrorMessageBuffers (void)

Variables

- PARALLELVARS **PF**

4.38.1 Detailed Description

Message passing library independent functions of parform

This file contains functions needed for the parallel version of form3 these functions need no real link to the message passing libraries, they only need some interface dependent preprocessor definitions (check parallel.h). So there still need two different objectfiles to be compiled for mpi and pvm!

4.38.2 Macro Definition Documentation

4.38.2.1 SWAP

```
#define SWAP(
               x,
               y )
```

Value:

```
    do { \
        char swap_tmp__[sizeof(x) == sizeof(y) ? (int)sizeof(x) : -1]; \
        memcpy(swap_tmp__, &y, sizeof(x)); \
        memcpy(&y, &x, sizeof(x)); \
        memcpy(&x, swap_tmp__, sizeof(x)); \
    } while (0)
```

Swaps the variables *x* and *y*. If sizeof(x) != sizeof(y) then a compilation error will occur. A set of memcpy calls with constant sizes is expected to be inlined by the optimisation.

Definition at line 124 of file parallel.c.

4.38.2.2 PACK_LONG

```
#define PACK_LONG(
               p,
               n )
```

Value:

```
    do { \
        *(p)++ = (UWORD)((ULONG)(n) & (ULONG)WORDMASK); \
        *(p)++ = (UWORD)(((ULONG)(n) » BITSINWORD) & (ULONG)WORDMASK); \
    } while (0)
```

Packs a LONG value *n* to a WORD buffer *p*.

Definition at line 135 of file parallel.c.

4.38.2.3 UNPACK_LONG

```
#define UNPACK_LONG(
               p,
               n )
```

Value:

```
    do { \
        (n) = (LONG)((((ULONG)(p)[1] & (ULONG)WORDMASK) « BITSINWORD) | ((ULONG)(p)[0] & (ULONG)WORDMASK));
       \
        (p) += 2; \
    } while (0)
```

Unpacks a LONG value *n* from a WORD buffer *p*.

Definition at line 144 of file parallel.c.

4.38.2.4 CHECK

```
#define CHECK(
              condition ) _CHECK(condition, __FILE__, __LINE__)
```

A simple check for unrecoverable errors.

Definition at line 153 of file parallel.c.

4.38.2.5 __CHECK

```
#define __CHECK(
              condition,
              file,
              line )
```

Value:

```
    do { \
        if ( !(condition) ) { \
            Error0("Fatal error at " file ":" #line); \
            Terminate(-1); \
        } \
    } while (0)
```

Definition at line 155 of file parallel.c.

4.38.3 Typedef Documentation

4.38.3.1 NODE

```
typedef struct NoDe NODE
```

A node for the tree of losers in the final sorting on the master.

4.38.4 Function Documentation

4.38.4.1 PF_RealTime()

```
LONG PF_RealTime (
             int i )
```

Returns the realtime in 1/100 sec. as a LONG.

Parameters

i	the timer will be reset if i == 0.

Returns

the real elapsed time in 1/100 second.

Definition at line 101 of file mpi.c.

4.38.4.2 PF_LibInit()

```
int PF_LibInit (
            int * argcp,
            char *** argvp )
```

Performs all library dependent initializations.

Parameters

argcp	pointer to the number of arguments.
argvp	pointer to the arguments.

Returns

0 if OK, nonzero on error.

Definition at line 123 of file mpi.c.

4.38.4.3 PF_LibTerminate()

```
int PF_LibTerminate (
            int error )
```

Exits mpi, when there is an error either indicated or happening, returnvalue is 1, else returnvalue is 0.

Parameters

error	an error code.

Returns

0 if OK, nonzero on error.

Definition at line 209 of file mpi.c.

Referenced by PF_Terminate().

4.38.4.4 PF_Probe()

```
int PF_Probe (
              int * src )
```

Probes the next incoming message. If src == PF_ANY_SOURCE this operation is blocking, otherwise nonbloking.

Parameters

in,out	*src*	the source process number. In output, the process number of actual found source.

Returns

the tag value of the next incoming message if found, 0 if a nonbloking probe (input src != PF_ANY_SOURCE) did not find any messages. A negative returned value indicates an error.

Definition at line 230 of file mpi.c.

4.38.4.5 PF_RecvWbuf()

```
int PF_RecvWbuf (
              WORD * b,
              LONG * s,
              int * src )
```

Blocking receive of a WORD buffer.

Parameters

out	*b*	the buffer to store the received data.
in,out	*s*	the size of the buffer. The output value is the actual size of the received data.
in,out	*src*	the source process number. The output value is the process number of actual source.

Returns

the received message tag. A negative value indicates an error.

Definition at line 337 of file mpi.c.

4.38.4.6 PF_IRecvRbuf()

```
int PF_IRecvRbuf (
              PF_BUFFER * r,
              int bn,
              int from )
```

Posts nonblocking receive for the active receive buffer. The buffer is filled from `full` to `stop`.

Parameters

r	the PF_BUFFER struct for the nonblocking receive.
bn	the index of the cyclic buffer.
from	the source process number.

Returns

0 if OK, nonzero on error.

Definition at line 366 of file mpi.c.

4.38.4.7 PF_WaitRbuf()

```
int PF_WaitRbuf (
              PF_BUFFER * r,
              int bn,
              LONG * size )
```

Waits for the buffer *bn* to finish a pending nonblocking receive. It returns the received tag and in `*size` the number of field received. If the receive is already finished, just return the flag and size, else wait for it to finish, but also check for other pending receives.

Parameters

	r	the PF_BUFFER struct for the pending nonblocking receive.
	bn	the index of the cyclic buffer.
out	*size*	the actual size of received data.

Returns

the received message tag. A negative value indicates an error.

Definition at line 400 of file mpi.c.

4.38.4.8 PF_RawSend()

```
int PF_RawSend (
            int dest,
            void * buf,
            LONG l,
            int tag )
```

Sends *l* bytes from *buf* to *dest*. Returns 0 on success, or -1.

Parameters

dest	the destination process number.
buf	the send buffer.
l	the size of data in the send buffer in bytes.
tag	the message tag.

Returns

0 if OK, nonzero on error.

Definition at line 463 of file mpi.c.

Referenced by PF_SendFile().

4.38.4.9 PF_RawRecv()

```
LONG PF_RawRecv (
            int * src,
            void * buf,
            LONG thesize,
            int * tag )
```

Receives not more than *thesize* bytes from *src*, returns the actual number of received bytes, or -1 on failure.

Parameters

in,out	*src*	the source process number. In output, that of the actual received message.
out	*buf*	the receive buffer.
	thesize	the size of the receive buffer in bytes.
out	*tag*	the message tag of the actual received message.

Returns

the actual sizeof received data in bytes, or -1 on failure.

Definition at line 484 of file mpi.c.

4.38.4.10 PF_RawProbe()

```
int PF_RawProbe (
            int * src,
            int * tag,
            int * bytesize )
```

Probes an incoming message.

Parameters

in,out	*src*	the source process number. In output, that of the actual received message.
in,out	*tag*	the message tag. In output, that of the actual received message.
out	*bytesize*	the size of incoming data in bytes.

Returns

0 if OK, nonzero on error.

Definition at line 508 of file mpi.c.

4.38.4.11 PF_EndSort()

```
int PF_EndSort (
            void  )
```

Finishes a master sorting with collecting terms from slaves. Called by EndSort().

If this is not the masterprocess, just initialize the sendbuffers and return 0, else PF_EndSort() sends the rest of the terms in the sendbuffer to the next slave and a dummy message to all slaves with tag PF_ENDSORT_MSGTAG. Then it receives the sorted terms, sorts them using a recursive 'tree of losers' (PF_GetLoser()) and writes them to the outputfile.

Returns

1 if the sorting on the master was done. 0 if EndSort() still must perform a regular sorting becuase it is not at the ground level or not on the master or in the sequential mode or in the InParallel mode. -1 if an error occured.

Remarks

The slaves will send the sorted terms back to the master in the regular sorting (after the initialization of the send buffer in PF_EndSort()). See PutOut() and FlushOut().

This function has been changed such that when it returns 1, AM.S0->TermsLeft is set correctly. But AM.S0->GenTerms is not set: it will be set after collecting the statistics from the slaves at the end of PF_Processor(). (TU 30 Jun 2011)

Definition at line 864 of file parallel.c.

4.38.4.12 PF_Deferred()

```
WORD PF_Deferred (
            WORD * term,
            WORD level )
```

Replaces Deferred() on the slaves.

Parameters

term	the term that must be multiplied by the contents of the current bracket.
level	the compiler level.

Returns

0 if OK, nonzero on error.

Definition at line 1208 of file parallel.c.

4.38.4.13 PF_Processor()

```
int PF_Processor (
            EXPRESSIONS e,
            WORD i,
            WORD LastExpression )
```

Replaces parts of Processor() on the masters and slaves. On the master PF_Processor() is responsible for proper distribution of terms from the input file to the slaves. On the slaves it calls Generator() for all the terms that this process gets, but PF_GetTerm() gets terms from the master (not directly from infile).

Parameters

e	The pointer to the current expression.
i	The index for the current expression.
LastExpression	The flag indicating whether it is the last expression.

Returns

0 if OK, nonzero on error.

Definition at line 1540 of file parallel.c.

4.38.4.14 PF_Init()

```
int PF_Init (
            int * argc,
            char *** argv )
```

All the library independent stuff. PF_LibInit() should do all library dependent initializations.

Parameters

argc	pointer to the number of arguments.
argv	pointer to the arguments.

Returns

0 if OK, nonzero on error.

Definition at line 1953 of file parallel.c.

4.38.4.15 PF_Terminate()

```
int PF_Terminate (
            int errorcode )
```

Performs the finalization of ParFORM. To be called by Terminate().

Parameters

error	an error code.

Returns

0 if OK, nonzero on error.

Definition at line 2047 of file parallel.c.

References PF_LibTerminate().

4.38.4.16 PF_GetSlaveTimes()

```
LONG PF_GetSlaveTimes (
            void  )
```

Returns the total CPU time of all slaves together. This function must be called on the master and all slaves.

Returns

on the master, the sum of CPU times on all slaves.

Definition at line 2063 of file parallel.c.

4.38.4.17 PF_BroadcastNumber()

```
LONG PF_BroadcastNumber (
            LONG x )
```

Broadcasts a LONG value from the master to the all slaves.

Parameters

x	the number to be broadcast (set on the master).

Returns

the synchronised result.

Definition at line 2083 of file parallel.c.

4.38.4.18 PF_BroadcastBuffer()

```
void PF_BroadcastBuffer (
            WORD ** buffer,
            LONG * length )
```

Broadcasts a buffer from the master to all the slaves.

Parameters

in,out	*buffer*	on the master, the buffer to be broadcast. On the slaves, the buffer will be allocated if the length is greater than 0. The caller must free it.
in,out	*length*	on the master, the length of the buffer to be broadcast. On the slaves, it receives the length of transfered buffer. The actual transfer occurs only if the length is greater than 0.

Definition at line 2110 of file parallel.c.

4.38.4.19 PF_BroadcastString()

```
int PF_BroadcastString (
            UBYTE * str )
```

Broadcasts a string from the master to all slaves.

Parameters

in,out	*str*	The pointer to a null-terminated string.

Returns

0 if OK, nonzero on error.

Definition at line 2152 of file parallel.c.

4.38.4.20 PF_BroadcastPreDollar()

```
int PF_BroadcastPreDollar (
            WORD ** dbuffer,
            LONG * newsize,
            int * numterms )
```

Broadcasts dollar variables set as a preprocessor variables. Only the master is able to make an assignment like #$a=g; where g is an expression: only the master has an access to the expression. So, the master broadcasts the result to slaves.

The result is in ∗dbuffer of the size is ∗newsize (in number of WORDs), +1 for trailing zero. For slave newsize and numterms are output parameters.

Parameters

in,out	*dbuffer*	the buffer for a dollar variable.
in,out	*newsize*	the size of the dollar variable in WORDs.
in,out	*numterms*	the number of terms in the dollar variable.

Returns

0 if OK, nonzero on error.

Definition at line 2207 of file parallel.c.

4.38.4.21 PF_CollectModifiedDollars()

```
int PF_CollectModifiedDollars (
            void  )
```

Combines modified dollar variables on the all slaves, and store them into those on the master.

The potentially modified dollar variables are given in PotModdollars, and the number of them is given by NumPot↩ Moddollars.

The current module could be executed in parallel only if all potentially modified variables are listed in ModOptdollars, otherwise the module was switched to the sequential mode.

Returns

0 if OK, nonzero on error.

Definition at line 2495 of file parallel.c.

4.38.4.22 PF_BroadcastModifiedDollars()

```
int PF_BroadcastModifiedDollars (
            void  )
```

Broadcasts modified dollar variables on the master to the all slaves.

The potentially modified dollar variables are given in PotModdollars, and the number of them is given by NumPot↩ Moddollars.

The current module could be executed in parallel only if all potentially modified variables are listed in ModOptdollars, otherwise the module was switched to the sequential mode. In either cases, we need to broadcast them.

Returns

0 if OK, nonzero on error.

Definition at line 2774 of file parallel.c.

4.38.4.23 PF_BroadcastRedefinedPreVars()

```
int PF_BroadcastRedefinedPreVars (
            void  )
```

Broadcasts preprocessor variables, which were changed by the Redefine statements in the current module, from the master to the all slaves.

The potentially redefined preprocessor variables are given in AC.pfirstnum, and the number of them is given by AC.numpfirstnum. For an actually redefined variable, the corresponding value in AC.inputnumbers is non-negative.

Returns

0 if OK, nonzero on error.

Definition at line 2991 of file parallel.c.

4.38.4.24 PF_BroadcastCBuf()

```
int PF_BroadcastCBuf (
            int bufnum )
```

Broadcasts a compiler buffer specified by *bufnum* from the master to the all slaves.

Parameters

bufnum	The index of the compiler buffer to be broadcast.

Returns

0 if OK, nonzero on error.

Definition at line 3133 of file parallel.c.

4.38.4.25 PF_BroadcastExpFlags()

```
int PF_BroadcastExpFlags (
            void  )
```

Broadcasts AR.expflags and several properties of each expression, e.g., e->vflags, from the master to all slaves.

Returns

0 if OK, nonzero on error.

Definition at line 3244 of file parallel.c.

4.38.4.26 PF_BroadcastExpr()

```
int PF_BroadcastExpr (
            EXPRESSIONS e,
            FILEHANDLE * file )
```

Broadcasts an expression from the master to the all slaves.

Parameters

e	The expression to be broadcast.
file	The file in which the expression is sitting.

Returns

0 if OK, nonzero on error.

Definition at line 3536 of file parallel.c.

4.38.4.27 PF_BroadcastRHS()

```
int PF_BroadcastRHS (
            void  )
```

Broadcasts expressions appearing in the right-hand side from the master to the all slaves.

Returns

0 if OK, nonzero on error.

Definition at line 3564 of file parallel.c.

4.38.4.28 PF_InParallelProcessor()

```
int PF_InParallelProcessor (
            void  )
```

Processes expressions in the InParallel mode, i.e., dividing expressions marked by partodo over the slaves.

Returns

0 if OK, nonzero on error.

Definition at line 3611 of file parallel.c.

4.38.4.29 PF_SendFile()

```
int PF_SendFile (
            int to,
            FILE * fd )
```

Sends a file to the process specified by *to*.

Parameters

to	the destination process number.
fd	the file to be sent.

Returns

the size of sent data in bytes, or -1 on error.

Definition at line 4207 of file parallel.c.

References PF_RawSend().

4.38.4.30 PF_RecvFile()

```
int PF_RecvFile (
            int from,
            FILE * fd )
```

Receives a file from the process specified by *from*.

Parameters

from	the source process number.
fd	the file to save the received data.

Returns

the size of received data in bytes, or -1 on error.

Definition at line 4245 of file parallel.c.

4.38.4.31 PF_MLock()

```
void PF_MLock (
            void  )
```

A function called by MLOCK(ErrorMessageLock) for slaves.

Definition at line 4326 of file parallel.c.

4.38.4.32 PF_MUnlock()

```
void PF_MUnlock (
            void  )
```

A function called by MUNLOCK(ErrorMessageLock) for slaves.

Definition at line 4342 of file parallel.c.

4.38.4.33 PF_WriteFileToFile()

```
LONG PF_WriteFileToFile (
            int handle,
            UBYTE * buffer,
            LONG size )
```

Replaces WriteFileToFile() on the master and slaves.

It copies the given buffer into internal buffers if called between MLOCK(ErrorMessageLock) and MUNLOCK(Error↩MessageLock) for slaves and handle is StdOut or LogHandle, otherwise calls WriteFileToFile().

Parameters

handle	a file handle that specifies the output.
buffer	a pointer to the source buffer containing the data to be written.
size	the size of data to be written in bytes.

Returns

the actual size of data written to the output in bytes.

Definition at line 4371 of file parallel.c.

4.38.4.34 PF_FlushStdOutBuffer()

```
void PF_FlushStdOutBuffer (
            void  )
```

Explicitly Flushes the buffer for the standard output on the master, which is used if PF_ENABLE_STDOUT_↩ BUFFERING is defined.

Definition at line 4465 of file parallel.c.

4.38.4.35 PF_FreeErrorMessageBuffers()

```
void PF_FreeErrorMessageBuffers (
            void  )
```

Frees the buffers allocated for the synchronized output.

Currently, not used anywhere, but could be used in PF_Terminate().

Definition at line 4601 of file parallel.c.

References VectorFree.

4.39 parallel.h File Reference

```
#include <mpi.h>
```

Data Structures

- struct PF_BUFFER
- struct ParallelVars

Macros

- #define **MASTER** 0
- #define **PF_RESET** 0
- #define **PF_TIME** 1
- #define **PF_TERM_MSGTAG** 10 /* master -> slave: sending terms */
- #define **PF_ENDSORT_MSGTAG** 11 /* master -> slave: no more terms to be distributed, slave -> master: after EndSort() */
- #define **PF_DOLLAR_MSGTAG** 12 /* slave -> master: sending $-variables */
- #define **PF_BUFFER_MSGTAG** 20 /* slave -> master: sending sorted terms, or in PF_SendFile()/PF_RecvFile() */
- #define **PF_ENDBUFFER_MSGTAG** 21 /* same as PF_BUFFER_MSGTAG, but indicates the end of operation */
- #define **PF_READY_MSGTAG** 30 /* slave -> master: slave is idle and can accept terms */
- #define **PF_DATA_MSGTAG** 50 /* InParallel, DoCheckpoint() */
- #define **PF_EMPTY_MSGTAG** 52 /* InParallel, DoCheckpoint(), PF_SendFile(), PF_RecvFile() */
- #define **PF_STDOUT_MSGTAG** 60 /* slave -> master: sending text to the stdout */
- #define **PF_LOG_MSGTAG** 61 /* slave -> master: sending text to the log file */
- #define **PF_OPT_MCTS_MSGTAG** 70 /* master <-> slave: optimization */
- #define **PF_OPT_HORNER_MSGTAG** 71 /* master <-> slave: optimization */
- #define **PF_OPT_COLLECT_MSGTAG** 72 /* slave -> master: optimization */
- #define **PF_MISC_MSGTAG** 100
- #define **GNUC_PREREQ**(major, minor, patchlevel) 0
- #define **indices** ((INDICES)(AC.IndexList.lijst))
- #define **PF_ANY_SOURCE** MPI_ANY_SOURCE
- #define **PF_ANY_MSGTAG** MPI_ANY_TAG
- #define **PF_COMM** MPI_COMM_WORLD
- #define **PF_BYTE** MPI_BYTE
- #define **PF_INT** MPI_INT
- #define **PF_LongMultiPack**(buffer, count, type) PF_LongMultiPackImpl(buffer, count, sizeof_datatype(type), type)
- #define **PF_LongMultiUnpack**(buffer, count, type) PF_LongMultiUnpackImpl(buffer, count, sizeof_datatype(type), type)

Typedefs

- typedef struct ParallelVars **PARALLELVARS**

Functions

- int PF_ISendSbuf (int, int)
- int PF_Bcast (void *buffer, int count)
- int PF_RawSend (int, void *, LONG, int)
- LONG PF_RawRecv (int *, void *, LONG, int *)
- int PF_PreparePack (void)
- int PF_Pack (const void *buffer, size_t count, MPI_Datatype type)
- int PF_Unpack (void *buffer, size_t count, MPI_Datatype type)
- int PF_PackString (const UBYTE *str)
- int PF_UnpackString (UBYTE *str)
- int PF_Send (int to, int tag)
- int PF_Receive (int src, int tag, int *psrc, int *ptag)
- int PF_Broadcast (void)
- int PF_PrepareLongSinglePack (void)

- int PF_LongSinglePack (const void *buffer, size_t count, MPI_Datatype type)
- int PF_LongSingleUnpack (void *buffer, size_t count, MPI_Datatype type)
- int PF_LongSingleSend (int to, int tag)
- int PF_LongSingleReceive (int src, int tag, int *psrc, int *ptag)
- int PF_PrepareLongMultiPack (void)
- int PF_LongMultiPackImpl (const void *buffer, size_t count, size_t eSize, MPI_Datatype type)
- int PF_LongMultiUnpackImpl (void *buffer, size_t count, size_t eSize, MPI_Datatype type)
- int PF_LongMultiBroadcast (void)
- int PF_EndSort (void)
- WORD PF_Deferred (WORD *, WORD)
- int PF_Processor (EXPRESSIONS, WORD, WORD)
- int PF_Init (int *, char ***)
- int PF_Terminate (int)
- LONG PF_GetSlaveTimes (void)
- LONG PF_BroadcastNumber (LONG)
- void PF_BroadcastBuffer (WORD **buffer, LONG *length)
- int PF_BroadcastString (UBYTE *)
- int PF_BroadcastPreDollar (WORD **, LONG *, int *)
- int PF_CollectModifiedDollars (void)
- int PF_BroadcastModifiedDollars (void)
- int PF_BroadcastRedefinedPreVars (void)
- int PF_BroadcastCBuf (int bufnum)
- int PF_BroadcastExpFlags (void)
- int **PF_StoreInsideInfo** (void)
- int **PF_RestoreInsideInfo** (void)
- int PF_BroadcastExpr (EXPRESSIONS e, FILEHANDLE *file)
- int PF_BroadcastRHS (void)
- int PF_InParallelProcessor (void)
- int PF_SendFile (int to, FILE *fd)
- int PF_RecvFile (int from, FILE *fd)
- void PF_MLock (void)
- void PF_MUnlock (void)
- LONG PF_WriteFileToFile (int, UBYTE *, LONG)
- void PF_FlushStdOutBuffer (void)

Variables

- PARALLELVARS **PF**
- LONG **PF_maxDollarChunkSize**

4.39.1 Detailed Description

Header file with things relevant to ParForm.

4.39.2 Function Documentation

4.39.2.1 PF_ISendSbuf()

```
int PF_ISendSbuf (
            int to,
            int tag )
```

Nonblocking send operation. It sends everything from `buff` to `fill` of the active buffer. Depending on *tag* it also can do waiting for other sends to finish or set the active buffer to the next one.

Parameters

to	the destination process number.
tag	the message tag.

Returns

0 if OK, nonzero on error.

Definition at line 261 of file mpi.c.

4.39.2.2 PF_Bcast()

```
int PF_Bcast (
            void * buffer,
            int count )
```

Broadcasts a message from the master to slaves.

Parameters

in,out	*buffer*	the starting address of buffer. The contents in this buffer on the master will be transferred to those on the slaves.
	count	the length of the buffer in bytes.

Returns

0 if OK, nonzero on error.

Definition at line 440 of file mpi.c.

4.39.2.3 PF_RawSend()

```
int PF_RawSend (
            int dest,
            void * buf,
            LONG l,
            int tag )
```

Sends *l* bytes from *buf* to *dest*. Returns 0 on success, or -1.

Parameters

dest	the destination process number.
buf	the send buffer.
l	the size of data in the send buffer in bytes.
tag	the message tag.

Returns

0 if OK, nonzero on error.

Definition at line 463 of file mpi.c.

4.39.2.4 PF_RawRecv()

```
LONG PF_RawRecv (
            int * src,
            void * buf,
            LONG thesize,
            int * tag )
```

Receives not more than *thesize* bytes from *src*, returns the actual number of received bytes, or -1 on failure.

Parameters

in,out	*src*	the source process number. In output, that of the actual received message.
out	*buf*	the receive buffer.
	thesize	the size of the receive buffer in bytes.
out	*tag*	the message tag of the actual received message.

Returns

the actual sizeof received data in bytes, or -1 on failure.

Definition at line 484 of file mpi.c.

4.39.2.5 PF_PreparePack()

```
int PF_PreparePack (
            void  )
```

Prepares for the next pack operations on the sender.

Returns

0 if OK, nonzero on error.

Definition at line 624 of file mpi.c.

4.39.2.6 PF_Pack()

```
int PF_Pack (
            const void * buffer,
            size_t count,
            MPI_Datatype type )
```

Adds data into the pack buffer.

Parameters

buffer	the pointer to the buffer storing the data to be packed.
count	the number of elements in the buffer.
type	the data type of elements in the buffer.

Returns

0 if OK, nonzero on error.

Definition at line 642 of file mpi.c.

4.39.2.7 PF_Unpack()

```
int PF_Unpack (
            void * buffer,
            size_t count,
            MPI_Datatype type )
```

Retrieves the next data in the pack buffer.

Parameters

out	*buffer*	the pointer to the buffer to store the unpacked data.
	count	the number of elements of data to be received.
	type	the data type of elements of data to be received.

Returns

0 if OK, nonzero on error.

Definition at line 671 of file mpi.c.

4.39.2.8 PF_PackString()

```
int PF_PackString (
            const UBYTE * str )
```

Packs a string *str* into the packed buffer PF_packbuf, including the trailing zero.

The first element (PF_INT) is the length of the packed portion of the string. If the string does not fit to the buffer PF↩ _packbuf, the function packs only the initial portion. It returns the number of packed bytes, so if (str[length-1]=='\0') then the whole string fits to the buffer, if not, then the rest (str+length) bust be packed and send again. On error, the function returns the negative error code.

One exception: the string "\0!\0" is used as an image of the NULL, so all 3 characters will be packed.

Parameters

str	a string to be packed.

Returns

the number of packed bytes, or a negative value on failure.

Definition at line 706 of file mpi.c.

4.39.2.9 PF_UnpackString()

```
int PF_UnpackString (
            UBYTE * str )
```

Unpacks a string to *str* from the packed buffer PF_packbuf, including the trailing zero.

It returns the number of unpacked bytes, so if (str[length-1]=='\0') then the whole string was unpacked, if not, then the rest must be appended to (str+length). On error, the function returns the negative error code.

Parameters

out	*str*	the buffer to store the unpacked string

Returns

the number of unpacked bytes, or a negative value on failure.

Definition at line 774 of file mpi.c.

4.39.2.10 PF_Send()

```
int PF_Send (
            int to,
            int tag )
```

Sends the contents in the pack buffer to the process specified by *to*.

Example:

```
if ( PF.me == SRC ) {
  PF_PreparePack();
  // Packing operations here...
  PF_Send(DEST, TAG);
}
else if ( PF.me == DEST ) {
  PF_Receive(SRC, TAG, &actual_src, &actual_tag);
  // Unpacking operations here...
}
```

Parameters

to	the destination process number.
tag	the message tag.

Returns

0 if OK, nonzero on error.

Definition at line 822 of file mpi.c.

4.39.2.11 PF_Receive()

```
int PF_Receive (
            int src,
            int tag,
            int * psrc,
            int * ptag )
```

Receives data into the pack buffer from the process specified by *src*. This function allows &src == psrc or &tag == ptag. Either *psrc* or *ptag* can be NULL.

See the example of PF_Send().

Parameters

	src	the source process number (can be PF_ANY_SOURCE).
	tag	the source message tag (can be PF_ANY_TAG).
out	*psrc*	the actual source process number of received message.
out	*ptag*	the received message tag.

Returns

0 if OK, nonzero on error.

Definition at line 848 of file mpi.c.

4.39.2.12 PF_Broadcast()

```
int PF_Broadcast (
            void  )
```

Broadcasts the contents in the pack buffer on the master to those on the slaves.

Example:

```
if ( PF.me == MASTER ) {
  PF_PreparePack();
  // Packing operations here...
}
PF_Broadcast();
if ( PF.me != MASTER ) {
  // Unpacking operations here...
}
```

Returns

0 if OK, nonzero on error.

Definition at line 883 of file mpi.c.

4.39.2.13 PF_PrepareLongSinglePack()

```
int PF_PrepareLongSinglePack (
            void  )
```

Prepares for the next long-single-pack operations on the sender.

Returns

0 if OK, nonzero on error.

Definition at line 1451 of file mpi.c.

4.39.2.14 PF_LongSinglePack()

```
int PF_LongSinglePack (
            const void * buffer,
            size_t count,
            MPI_Datatype type )
```

Adds data into the "long single" pack buffer.

Parameters

buffer	the pointer to the buffer storing the data to be packed.
count	the number of elements in the buffer.
type	the data type of elements in the buffer.

Returns

0 if OK, nonzero on error.

Definition at line 1469 of file mpi.c.

4.39.2.15 PF_LongSingleUnpack()

```
int PF_LongSingleUnpack (
            void * buffer,
```

```
            size_t count,
            MPI_Datatype type )
```

Retrieves the next data in the "long single" pack buffer.

Parameters

out	*buffer*	the pointer to the buffer to store the unpacked data.
	count	the number of elements of data to be received.
	type	the data type of elements of data to be received.

Returns

0 if OK, nonzero on error.

Definition at line 1503 of file mpi.c.

4.39.2.16 PF_LongSingleSend()

```
int PF_LongSingleSend (
            int to,
            int tag )
```

Sends the contents in the "long single" pack buffer to the process specified by *to*.

Example:

```
if ( PF.me == SRC ) {
  PF_PrepareLongSinglePack();
  // Packing operations here...
  PF_LongSingleSend(DEST, TAG);
}
else if ( PF.me == DEST ) {
  PF_LongSingleReceive(SRC, TAG, &actual_src, &actual_tag);
  // Unpacking operations here...
}
```

Parameters

to	the destination process number.
tag	the message tag.

Returns

0 if OK, nonzero on error.

Definition at line 1540 of file mpi.c.

4.39.2.17 PF_LongSingleReceive()

```
int PF_LongSingleReceive (
            int src,
            int tag,
            int * psrc,
            int * ptag )
```

Receives data into the "long single" pack buffer from the process specified by *src*. This function allows &src == psrc or &tag == ptag. Either *psrc* or *ptag* can be NULL.

See the example of PF_LongSingleSend().

Parameters

	src	the source process number (can be PF_ANY_SOURCE).
	tag	the source message tag (can be PF_ANY_TAG).
out	*psrc*	the actual source process number of received message.
out	*ptag*	the received message tag.

Returns

0 if OK, nonzero on error.

Definition at line 1583 of file mpi.c.

4.39.2.18 PF_PrepareLongMultiPack()

```
int PF_PrepareLongMultiPack (
            void  )
```

Prepares for the next long-multi-pack operations on the sender.

Returns

0 if OK, nonzero on error.

Definition at line 1643 of file mpi.c.

4.39.2.19 PF_LongMultiPackImpl()

```
int PF_LongMultiPackImpl (
            const void * buffer,
            size_t count,
            size_t eSize,
            MPI_Datatype type )
```

Adds data into the "long multi" pack buffer.

Parameters

buffer	the pointer to the buffer storing the data to be packed.
count	the number of elements in the buffer.
eSize	the byte size of each element of data.
type	the data type of elements in the buffer.

Returns

0 if OK, nonzero on error.

Definition at line 1662 of file mpi.c.

4.39.2.20 PF_LongMultiUnpackImpl()

```
int PF_LongMultiUnpackImpl (
            void * buffer,
            size_t count,
            size_t eSize,
            MPI_Datatype type )
```

Retrieves the next data in the "long multi" pack buffer.

Parameters

out	*buffer*	the pointer to the buffer to store the unpacked data.
	count	the number of elements of data to be received.
	eSize	the byte size of each element of data.
	type	the data type of elements of data to be received.

Returns

0 if OK, nonzero on error.

Definition at line 1721 of file mpi.c.

4.39.2.21 PF_LongMultiBroadcast()

```
int PF_LongMultiBroadcast (
            void  )
```

Broadcasts the contents in the "long multi" pack buffer on the master to those on the slaves.

Example:

```
if ( PF.me == MASTER ) {
  PF_PrepareLongMultiPack();
  // Packing operations here...
}
PF_LongMultiBroadcast();
if ( PF.me != MASTER ) {
  // Unpacking operations here...
}
```

Returns

0 if OK, nonzero on error.

Definition at line 1807 of file mpi.c.

4.39.2.22 PF_EndSort()

```
int PF_EndSort (
            void  )
```

Finishes a master sorting with collecting terms from slaves. Called by EndSort().

If this is not the masterprocess, just initialize the sendbuffers and return 0, else PF_EndSort() sends the rest of the terms in the sendbuffer to the next slave and a dummy message to all slaves with tag PF_ENDSORT_MSGTAG. Then it receives the sorted terms, sorts them using a recursive 'tree of losers' (PF_GetLoser()) and writes them to the outputfile.

Returns

1 if the sorting on the master was done. 0 if EndSort() still must perform a regular sorting becuase it is not at the ground level or not on the master or in the sequential mode or in the InParallel mode. -1 if an error occured.

Remarks

The slaves will send the sorted terms back to the master in the regular sorting (after the initialization of the send buffer in PF_EndSort()). See PutOut() and FlushOut().

This function has been changed such that when it returns 1, AM.S0->TermsLeft is set correctly. But AM.S0->GenTerms is not set: it will be set after collecting the statistics from the slaves at the end of PF_Processor(). (TU 30 Jun 2011)

Definition at line 864 of file parallel.c.

4.39.2.23 PF_Deferred()

```
WORD PF_Deferred (
            WORD * term,
            WORD level )
```

Replaces Deferred() on the slaves.

Parameters

term	the term that must be multiplied by the contents of the current bracket.
level	the compiler level.

Returns

0 if OK, nonzero on error.

Definition at line 1208 of file parallel.c.

4.39.2.24 PF_Processor()

```
int PF_Processor (
            EXPRESSIONS e,
            WORD i,
            WORD LastExpression )
```

Replaces parts of Processor() on the masters and slaves. On the master PF_Processor() is responsible for proper distribution of terms from the input file to the slaves. On the slaves it calls Generator() for all the terms that this process gets, but PF_GetTerm() gets terms from the master (not directly from infile).

Parameters

e	The pointer to the current expression.
i	The index for the current expression.
LastExpression	The flag indicating whether it is the last expression.

Returns

0 if OK, nonzero on error.

Definition at line 1540 of file parallel.c.

4.39.2.25 PF_Init()

```
int PF_Init (
            int * argc,
            char *** argv )
```

All the library independent stuff. PF_LibInit() should do all library dependent initializations.

Parameters

argc	pointer to the number of arguments.
argv	pointer to the arguments.

Returns

0 if OK, nonzero on error.

Definition at line 1953 of file parallel.c.

4.39.2.26 PF_Terminate()

```
int PF_Terminate (
            int errorcode )
```

Performs the finalization of ParFORM. To be called by Terminate().

Parameters

error	an error code.

Returns

0 if OK, nonzero on error.

Definition at line 2047 of file parallel.c.

References PF_LibTerminate().

4.39.2.27 PF_GetSlaveTimes()

```
LONG PF_GetSlaveTimes (
            void  )
```

Returns the total CPU time of all slaves together. This function must be called on the master and all slaves.

Returns

on the master, the sum of CPU times on all slaves.

Definition at line 2063 of file parallel.c.

4.39.2.28 PF_BroadcastNumber()

```
LONG PF_BroadcastNumber (
            LONG x )
```

Broadcasts a LONG value from the master to the all slaves.

Parameters

x	the number to be broadcast (set on the master).

Returns

the synchronised result.

Definition at line 2083 of file parallel.c.

4.39.2.29 PF_BroadcastBuffer()

```
void PF_BroadcastBuffer (
            WORD ** buffer,
            LONG * length )
```

Broadcasts a buffer from the master to all the slaves.

Parameters

in,out	*buffer*	on the master, the buffer to be broadcast. On the slaves, the buffer will be allocated if the length is greater than 0. The caller must free it.
in,out	*length*	on the master, the length of the buffer to be broadcast. On the slaves, it receives the length of transfered buffer. The actual transfer occurs only if the length is greater than 0.

Definition at line 2110 of file parallel.c.

4.39.2.30 PF_BroadcastString()

```
int PF_BroadcastString (
            UBYTE * str )
```

Broadcasts a string from the master to all slaves.

Parameters

in,out	*str*	The pointer to a null-terminated string.

Returns

0 if OK, nonzero on error.

Definition at line 2152 of file parallel.c.

4.39.2.31 PF_BroadcastPreDollar()

```
int PF_BroadcastPreDollar (
            WORD ** dbuffer,
            LONG * newsize,
            int * numterms )
```

Broadcasts dollar variables set as a preprocessor variables. Only the master is able to make an assignment like #$a=g; where g is an expression: only the master has an access to the expression. So, the master broadcasts the result to slaves.

The result is in ∗dbuffer of the size is ∗newsize (in number of WORDs), +1 for trailing zero. For slave newsize and numterms are output parameters.

Parameters

in,out	*dbuffer*	the buffer for a dollar variable.
in,out	*newsize*	the size of the dollar variable in WORDs.
in,out	*numterms*	the number of terms in the dollar variable.

Returns

0 if OK, nonzero on error.

Definition at line 2207 of file parallel.c.

4.39.2.32 PF_CollectModifiedDollars()

```
int PF_CollectModifiedDollars (
            void  )
```

Combines modified dollar variables on the all slaves, and store them into those on the master.

The potentially modified dollar variables are given in PotModdollars, and the number of them is given by NumPot↩ Moddollars.

The current module could be executed in parallel only if all potentially modified variables are listed in ModOptdollars, otherwise the module was switched to the sequential mode.

Returns

0 if OK, nonzero on error.

Definition at line 2495 of file parallel.c.

4.39.2.33 PF_BroadcastModifiedDollars()

```
int PF_BroadcastModifiedDollars (
            void  )
```

Broadcasts modified dollar variables on the master to the all slaves.

The potentially modified dollar variables are given in PotModdollars, and the number of them is given by NumPot↩ Moddollars.

The current module could be executed in parallel only if all potentially modified variables are listed in ModOptdollars, otherwise the module was switched to the sequential mode. In either cases, we need to broadcast them.

Returns

0 if OK, nonzero on error.

Definition at line 2774 of file parallel.c.

4.39.2.34 PF_BroadcastRedefinedPreVars()

```
int PF_BroadcastRedefinedPreVars (
            void  )
```

Broadcasts preprocessor variables, which were changed by the Redefine statements in the current module, from the master to the all slaves.

The potentially redefined preprocessor variables are given in AC.pfirstnum, and the number of them is given by AC.numpfirstnum. For an actually redefined variable, the corresponding value in AC.inputnumbers is non-negative.

Returns

0 if OK, nonzero on error.

Definition at line 2991 of file parallel.c.

4.39.2.35 PF_BroadcastCBuf()

```
int PF_BroadcastCBuf (
            int bufnum )
```

Broadcasts a compiler buffer specified by *bufnum* from the master to the all slaves.

Parameters

bufnum	The index of the compiler buffer to be broadcast.

Returns

0 if OK, nonzero on error.

Definition at line 3133 of file parallel.c.

4.39.2.36 PF_BroadcastExpFlags()

```
int PF_BroadcastExpFlags (
            void  )
```

Broadcasts AR.expflags and several properties of each expression, e.g., e->vflags, from the master to all slaves.

Returns

0 if OK, nonzero on error.

Definition at line 3244 of file parallel.c.

4.39.2.37 PF_BroadcastExpr()

```
int PF_BroadcastExpr (
            EXPRESSIONS e,
            FILEHANDLE * file )
```

Broadcasts an expression from the master to the all slaves.

Parameters

e	The expression to be broadcast.
file	The file in which the expression is sitting.

Returns

0 if OK, nonzero on error.

Definition at line 3536 of file parallel.c.

4.39.2.38 PF_BroadcastRHS()

```
int PF_BroadcastRHS (
            void  )
```

Broadcasts expressions appearing in the right-hand side from the master to the all slaves.

Returns

0 if OK, nonzero on error.

Definition at line 3564 of file parallel.c.

4.39.2.39 PF_InParallelProcessor()

```
int PF_InParallelProcessor (
            void  )
```

Processes expressions in the InParallel mode, i.e., dividing expressions marked by partodo over the slaves.

Returns

0 if OK, nonzero on error.

Definition at line 3611 of file parallel.c.

4.39.2.40 PF_SendFile()

```
int PF_SendFile (
            int to,
            FILE * fd )
```

Sends a file to the process specified by *to*.

Parameters

to	the destination process number.
fd	the file to be sent.

Returns

the size of sent data in bytes, or -1 on error.

Definition at line 4207 of file parallel.c.

References PF_RawSend().

4.39.2.41 PF_RecvFile()

```
int PF_RecvFile (
            int from,
            FILE * fd )
```

Receives a file from the process specified by *from*.

Parameters

from	the source process number.
fd	the file to save the received data.

Returns

the size of received data in bytes, or -1 on error.

Definition at line 4245 of file parallel.c.

4.39.2.42 PF_MLock()

```
void PF_MLock (
            void  )
```

A function called by MLOCK(ErrorMessageLock) for slaves.

Definition at line 4326 of file parallel.c.

4.39.2.43 PF_MUnlock()

```
void PF_MUnlock (
            void  )
```

A function called by MUNLOCK(ErrorMessageLock) for slaves.

Definition at line 4342 of file parallel.c.

4.39.2.44 PF_WriteFileToFile()

```
LONG PF_WriteFileToFile (
            int handle,
            UBYTE * buffer,
            LONG size )
```

Replaces WriteFileToFile() on the master and slaves.

It copies the given buffer into internal buffers if called between MLOCK(ErrorMessageLock) and MUNLOCK(Error↩MessageLock) for slaves and handle is StdOut or LogHandle, otherwise calls WriteFileToFile().

Parameters

handle	a file handle that specifies the output.
buffer	a pointer to the source buffer containing the data to be written.
size	the size of data to be written in bytes.

Returns

the actual size of data written to the output in bytes.

Definition at line 4371 of file parallel.c.

4.39.2.45 PF_FlushStdOutBuffer()

```
void PF_FlushStdOutBuffer (
            void  )
```

Explicitly Flushes the buffer for the standard output on the master, which is used if PF_ENABLE_STDOUT_↩ BUFFERING is defined.

Definition at line 4465 of file parallel.c.

4.40 pattern.c File Reference

```
#include "form3.h"
```

Macros

- #define **PutInBuffers**(pow)

Functions

- WORD TestMatch (PHEAD WORD ∗term, WORD ∗level)
- VOID **Substitute** (PHEAD WORD ∗term, WORD ∗pattern, WORD power)
- WORD **FindAll** (PHEAD WORD ∗term, WORD ∗pattern, WORD level, WORD ∗par)
- int **TestSelect** (WORD ∗term, WORD ∗setp)
- VOID **SubsInAll** (PHEAD0)
- VOID **TransferBuffer** (int from, int to, int spectator)
- int **TakeIDfunction** (PHEAD WORD ∗term)

4.40.1 Detailed Description

Top level pattern matching routines. More pattern matching is found in findpat.c, function.c, symmetr.c and smart.c. The last three files contain the matching inside functions. The file pattern.c contains also the very important routine Substitute. All regular pattern matching is just the finding of the pattern and indicating what are the wildcards etc. The routine Substitute does the actual removal of the pattern and replaces it by a subterm of the type SUBEXPRESSION.

4.40.2 Macro Definition Documentation

4.40.2.1 PutInBuffers

```
#define PutInBuffers(
              pow )
```

Value:

```
AddRHS(AT.ebufnum,1); \
*out++ = SUBEXPRESSION; \
*out++ = SUBEXPSIZE; \
*out++ = C->numrhs; \
*out++ = pow; \
*out++ = AT.ebufnum; \
FILLSUB(out) \
r = AT.pWorkSpace[rhs+i]; \
if ( *r > 0 ) { \
    oldinr = r[*r]; r[*r] = 0; \
    AddNtoC(AT.ebufnum,(*r+1-ARGHEAD),(r+ARGHEAD),14); \
    r[*r] = oldinr; \
} \
else { \
    ToGeneral(r,buffer,1); \
    buffer[buffer[0]] = 0; \
    AddNtoC(AT.ebufnum,buffer[0]+1,buffer,15); \
}
```

Definition at line 2193 of file pattern.c.

4.40.3 Function Documentation

4.40.3.1 TestMatch()

```
WORD TestMatch (
             PHEAD WORD * term,
             WORD * level )
```

This routine governs the pattern matching. If it decides that a substitution should be made, this can be either the insertion of a right hand side (C->rhs) or the automatic generation of terms as a result of an operation (like trace). The object to be replaced is removed from term and a subexpression pointer is inserted. If the substitution is made more than once there can be more subexpression pointers. Its number is positive as it corresponds to the level at which the C->rhs can be found in the compiler output. The subexpression pointer contains the wildcard substitution information. The power is found in *AT.TMout. For operations the subexpression pointer is negative and corresponds to an address in the array AT.TMout. In this array are then the instructions for the routine to be called and its number in the array 'Operations' The format is here: length,functionnumber,length-2 parameters

There is a certain complexity wrt repeat levels. Another complication is the poking of the wildcard values in the subexpression prototype in the compiler buffer. This was how things were done in the past with sequential FORM, but with the advent of TFORM this cannot be maintained. Now, for TFORM we make a copy of it. 7-may-2008 (JV): We cannot yet guarantee that this has been done 100% correctly. There are errors that occur in TFORM only and that may indicate problems.

Definition at line 97 of file pattern.c.

4.41 polyfact.cc File Reference

```
#include "poly.h"
#include "polygcd.h"
#include "polyfact.h"
#include <cmath>
#include <vector>
#include <iostream>
#include <algorithm>
#include <climits>
```

Functions

- ostream & **operator**<< (ostream &out, const factorized_poly &a)
- template<class T >
  ostream & **operator**<< (ostream &out, const vector< T > &v)

4.41.1 Detailed Description

Contains the routines for factorizing multivariate polynomials

4.42 polygcd.cc File Reference

```
#include "poly.h"
#include "polygcd.h"
#include <iostream>
#include <vector>
#include <cmath>
#include <map>
#include <algorithm>
```

Data Structures

- struct BracketInfo

Functions

- bool gcd_heuristic_possible (const poly &a)
- const poly **gcd_linear_helper** (const poly &a, const poly &b)

4.42.1 Detailed Description

Contains the routines for calculating greatest commons divisors of multivariate polynomials

4.42.2 Function Documentation

4.42.2.1 gcd_heuristic_possible()

```
bool gcd_heuristic_possible (
            const poly & a )
```

Heuristic greatest common divisor of multivariate polynomials

4.42.3 Description

Checks whether the heuristic seems possible by estimating

MAX_{terms} (coeff ^ PROD_{i=1..#vars} (pow_i+1))

and comparing this with GCD_HEURISTIC_MAX_DIGITS.

4.42.4 Notes

- For small polynomials, this consumes time and never triggers.

Definition at line 1144 of file polygcd.cc.

4.43 polywrap.cc File Reference

```
#include "poly.h"
#include "polygcd.h"
#include "polyfact.h"
#include <iostream>
#include <vector>
#include <map>
#include <climits>
#include <cassert>
```

Functions

- WORD poly_determine_modulus (PHEAD bool multi_error, bool is_fun_arg, string message)
- WORD ∗ poly_gcd (PHEAD WORD ∗a, WORD ∗b, WORD fit)
- WORD ∗ **poly_divmod** (PHEAD WORD ∗a, WORD ∗b, int divmod, WORD fit)
- WORD ∗ **poly_div** (PHEAD WORD ∗a, WORD ∗b, WORD fit)
- WORD ∗ **poly_rem** (PHEAD WORD ∗a, WORD ∗b, WORD fit)
- void poly_ratfun_read (WORD ∗a, poly &num, poly &den)
- void poly_sort (PHEAD WORD ∗a)
- WORD ∗ poly_ratfun_add (PHEAD WORD ∗t1, WORD ∗t2)
- int poly_ratfun_normalize (PHEAD WORD ∗term)
- void **poly_fix_minus_signs** (factorized_poly &a)
- WORD ∗ poly_factorize (PHEAD WORD ∗argin, WORD ∗argout, bool with_arghead, bool is_fun_arg)
- int poly_factorize_argument (PHEAD WORD ∗argin, WORD ∗argout)
- WORD ∗ poly_factorize_dollar (PHEAD WORD ∗argin)
- int poly_factorize_expression (EXPRESSIONS expr)
- int **poly_unfactorize_expression** (EXPRESSIONS expr)
- WORD ∗ **poly_inverse** (PHEAD WORD ∗arga, WORD ∗argb)
- WORD ∗ **poly_mul** (PHEAD WORD ∗a, WORD ∗b)
- void **poly_free_poly_vars** (PHEAD const char ∗text)

Variables

- const int **POLYWRAP_DENOMPOWER_INCREASE_FACTOR** = 2

4.43.1 Detailed Description

Contains methods to call the polynomial methods (written in C++) from the rest of Form (written in C). These include polynomial gcd computation, factorization and polyratfuns.

4.43.2 Function Documentation

4.43.2.1 poly_determine_modulus()

```
WORD poly_determine_modulus (
             PHEAD bool multi_error,
             bool is_fun_arg,
             string message )
```

Modulus for polynomial algebra

4.43.3 Description

This method determines whether polynomial algebra is done with a modulus or not. This depends on AC.ncmod. If only_funargs is set it also depends on (AC.modmode & ALSOFUNARGS).

The program terminates if the feature is not implemented. Polynomial algebra modulo M $>$ WORDSIZE in not implemented. If multi_error is set, multivariate algebra mod M is not implemented.

4.43.4 Notes

- If AC.ncmod$>$0 and only_funargs=true and AC.modmode&ALSOFUNARGS=false, AN.ncmod is set to zero, for otherwise RaisPow calculates mod M.

Definition at line 79 of file polywrap.cc.

4.43.4.1 poly_gcd()

```
WORD* poly_gcd (
             PHEAD WORD * a,
             WORD * b,
             WORD fit )
```

Polynomial gcd

4.43.5 Description

This method calculates the greatest common divisor of two polynomials, given by two zero-terminated Form-style term lists.

4.43.6 Notes

- The result is written at newly allocated memory
- Called from ratio.c
- Calls polygcd::gcd

Definition at line 124 of file polywrap.cc.

4.43.6.1 poly_ratfun_read()

```
void poly_ratfun_read (
             WORD * a,
             poly & num,
             poly & den )
```

Read a PolyRatFun

4.43.7 Description

This method reads a polyratfun starting at the pointer a. The resulting numerator and denominator are written in num and den. If MUSTCLEANPRF, the result is normalized.

4.43.8 Notes

- Calls polygcd::gcd

Definition at line 471 of file polywrap.cc.

4.43.8.1 poly_sort()

```
void poly_sort (
             PHEAD WORD * a )
```

Sort the polynomial terms

4.43.9 Description

Sorts the terms of a polynomial in Form poly(rat)fun order, i.e. lexicographical order with highest degree first.

4.43.10 Notes

- Uses Form sort routines with custom compare

Definition at line 557 of file polywrap.cc.

4.43.10.1 poly_ratfun_add()

```
WORD* poly_ratfun_add (
             PHEAD WORD * t1,
             WORD * t2 )
```

Addition of PolyRatFuns

4.43.11 Description

This method gets two pointers to polyratfuns with up to two arguments each and calculates the sum.

4.43.12 Notes

- The result is written at the workpointer
- Called from sort.c and threads.c
- Calls poly::operators and polygcd::gcd

Definition at line 600 of file polywrap.cc.

4.43.12.1 poly_ratfun_normalize()

```
int poly_ratfun_normalize (
             PHEAD WORD * term )
```

Multiplication/normalization of PolyRatFuns

4.43.13 Description

This method seaches a term for multiple polyratfuns and multiplies their contents. The result is properly normalized. Normalization also works for terms with a single polyratfun.

4.43.14 Notes

- The result overwrites the original term
- Called from proces.c
- Calls poly::operators and polygcd::gcd

Definition at line 719 of file polywrap.cc.

4.43.14.1 poly_factorize()

```
WORD* poly_factorize (
            PHEAD WORD * argin,
            WORD * argout,
            bool with_arghead,
            bool is_fun_arg )
```

Factorization of function arguments / dollars

4.43.15 Description

This method factorizes a Form style argument or zero-terminated term list.

4.43.16 Notes

- Called from poly_factorize_{argument,dollar}
- Calls polyfact::factorize

Definition at line 922 of file polywrap.cc.

4.43.16.1 poly_factorize_argument()

```
int poly_factorize_argument (
            PHEAD WORD * argin,
            WORD * argout )
```

Factorization of function arguments

4.43.17 Description

This method factorizes the Form-style argument argin.

4.43.18 Notes

- The result is written at argout
- Called from argument.c
- Calls poly_factorize

Definition at line 1047 of file polywrap.cc.

4.43.18.1 poly_factorize_dollar()

```
WORD* poly_factorize_dollar (
            PHEAD WORD * argin )
```

Factorization of dollar variables

4.43.19 Description

This method factorizes a dollar variable.

4.43.20 Notes

- The result is written at newly allocated memory.
- Called from dollar.c
- Calls poly_factorize

Definition at line 1074 of file polywrap.cc.

4.43.20.1 poly_factorize_expression()

```
int poly_factorize_expression (
            EXPRESSIONS expr )
```

Factorization of expressions

4.43.21 Description

This method factorizes an expression.

4.43.22 Notes

- The result overwrites the input expression
- Called from proces.c
- Calls polyfact::factorize

Definition at line 1100 of file polywrap.cc.

4.44 portsignals.h File Reference

```
#include <signal.h>
```

Macros

- #define **FATAL_SIG_ERROR** 4
- #define **NSIG** (1024)
- #define **SIGSEGV** (NSIG+1)
- #define **SIGFPE** (NSIG+2)
- #define **SIGILL** (NSIG+3)
- #define **SIGEMT** (NSIG+4)
- #define **SIGSYS** (NSIG+5)
- #define **SIGPIPE** (NSIG+6)
- #define **SIGLOST** (NSIG+7)
- #define **SIGXCPU** (NSIG+8)
- #define **SIGXFSZ** (NSIG+9)
- #define **SIGTERM** (NSIG+10)
- #define **SIGINT** (NSIG+11)
- #define **SIGQUIT** (NSIG+12)
- #define **SIGHUP** (NSIG+13)
- #define **SIGALRM** (NSIG+14)
- #define **SIGVTALRM** (NSIG+15)
- #define **SIGPROF** (NSIG+16)

4.44.1 Detailed Description

Contains definitions for signals used/intercepted in FORM.

Some systems (especially LINUX) have not enough signals available so some of the (!documented!) signals are not defined. This file contains the definition of all signals used in the program. If the signal is not defined we define it as unused (>NSIG).

The include of signal.h must be first, before we try to define undefined signals.

4.45 pre.c File Reference

```
#include "form3.h"
#include "vector.h"
```

Macros

- #define **STRINGIFY**(x) STRINGIFY__(x)
- #define **STRINGIFY__**(x) #x
- #define **SKIPBUFSIZE** 20
- #define **KILL** "kill"
- #define **KILLALL** "killall"
- #define **DAEMON** "daemon"
- #define **SHELL** "shell"
- #define **STDERR** "stderr"
- #define **TRUE_EXPR** "true"
- #define **FALSE_EXPR** "false"
- #define **NOSHELL** "noshell"
- #define **TERMINAL** "terminal"

Functions

- UBYTE **GetInput** ()
- VOID **ClearPushback** ()
- UBYTE **GetChar** (int level)
- VOID **CharOut** (UBYTE c)
- VOID **UnsetAllowDelay** ()
- UBYTE ∗ **GetPreVar** (UBYTE ∗name, int flag)
- int PutPreVar (UBYTE ∗name, UBYTE ∗value, UBYTE ∗args, int mode)
- VOID **PopPreVars** (int tonumber)
- VOID **IniModule** (int type)
- VOID **IniSpecialModule** (int type)
- VOID **PreProcessor** ()
- int **PreProInstruction** ()
- int **LoadInstruction** (int mode)
- int **LoadStatement** (int type)
- int **ExpandTripleDots** (int par)
- KEYWORD ∗ **FindKeyWord** (UBYTE ∗theword, KEYWORD ∗table, int size)
- KEYWORD ∗ **FindInKeyWord** (UBYTE ∗theword, KEYWORD ∗table, int size)
- int TheDefine (UBYTE ∗s, int mode)
- int **DoCommentChar** (UBYTE ∗s)
- int **DoPreAssign** (UBYTE ∗s)
- int **DoDefine** (UBYTE ∗s)
- int **DoRedefine** (UBYTE ∗s)
- int **ClearMacro** (UBYTE ∗name)
- int **TheUndefine** (UBYTE ∗name)
- int **DoUndefine** (UBYTE ∗s)
- int **DoInclude** (UBYTE ∗s)
- int **DoReverseInclude** (UBYTE ∗s)
- int **Include** (UBYTE ∗s, int type)
- int **DoPreExchange** (UBYTE ∗s)

- int **DoCall** (UBYTE *s)
- int **DoDebug** (UBYTE *s)
- int **DoTerminate** (UBYTE *s)
- int **DoDo** (UBYTE *s)
- int **DoBreakDo** (UBYTE *s)
- int **DoElse** (UBYTE *s)
- int **DoElseif** (UBYTE *s)
- int **DoEnddo** (UBYTE *s)
- int **DoEndif** (UBYTE *s)
- int **DoEndprocedure** (UBYTE *s)
- int **DoIf** (UBYTE *s)
- int **DoIfdef** (UBYTE *s, int par)
- int **DoIfydef** (UBYTE *s)
- int **DoIfndef** (UBYTE *s)
- int **DoInside** (UBYTE *s)
- int **DoEndInside** (UBYTE *s)
- int **DoMessage** (UBYTE *s)
- int **DoPipe** (UBYTE *s)
- int **DoPrcExtension** (UBYTE *s)
- int **DoPreOut** (UBYTE *s)
- int **DoPrePrintTimes** (UBYTE *s)
- int **DoPreAppend** (UBYTE *s)
- int **DoPreCreate** (UBYTE *s)
- int **DoPreRemove** (UBYTE *s)
- int **DoPreClose** (UBYTE *s)
- int **DoPreWrite** (UBYTE *s)
- int **DoProcedure** (UBYTE *s)
- int **DoPreBreak** (UBYTE *s)
- int **DoPreCase** (UBYTE *s)
- int **DoPreDefault** (UBYTE *s)
- int **DoPreEndSwitch** (UBYTE *s)
- int **DoPreSwitch** (UBYTE *s)
- int **DoPreShow** (UBYTE *s)
- int **DoSystem** (UBYTE *s)
- int **PreLoad** (PRELOAD *p, UBYTE *start, UBYTE *stop, int mode, char *message)
- int **PreSkip** (UBYTE *start, UBYTE *stop, int mode)
- VOID **StartPrepro** ()
- int **EvalPrelf** (UBYTE *s)
- UBYTE * **PrelfEval** (UBYTE *s, int *value)
- int **PreCmp** (int type, int val, UBYTE *t, int type2, int val2, UBYTE *t2, int cmpop)
- int **PreEq** (int type, int val, UBYTE *t, int type2, int val2, UBYTE *t2, int eqop)
- UBYTE * **pParseObject** (UBYTE *s, int *type, LONG *val2)
- UBYTE * **PreCalc** ()
- UBYTE * **PreEval** (UBYTE *s, LONG *x)
- void **AddToPreTypes** (int type)
- void **MessPreNesting** (int par)
- int **DoPreAddSeparator** (UBYTE *s)
- int **DoPreRmSeparator** (UBYTE *s)
- int **DoExternal** (UBYTE *s)
- int **DoPrompt** (UBYTE *s)
- int **DoSetExternal** (UBYTE *s)
- int **DoSetExternalAttr** (UBYTE *s)
- int **DoRmExternal** (UBYTE *s)
- int **DoFromExternal** (UBYTE *s)
- int **DoToExternal** (UBYTE *s)

- UBYTE * **defineChannel** (UBYTE *s, HANDLERS *h)
- int **writeToChannel** (int wtype, UBYTE *s, HANDLERS *h)
- int **DoFactDollar** (UBYTE *s)
- WORD **GetDollarNumber** (UBYTE **inp, DOLLARS d)
- int **DoSetRandom** (UBYTE *s)
- int **DoOptimize** (UBYTE *s)
- int **DoClearOptimize** (UBYTE *s)
- int **DoSkipExtraSymbols** (UBYTE *s)
- int **DoPreReset** (UBYTE *s)
- int DoPreAppendPath (UBYTE *s)
- int DoPrePrependPath (UBYTE *s)
- int **DoTimeOutAfter** (UBYTE *s)

4.45.1 Detailed Description

This is the preprocessor and all its routines.

4.45.2 Function Documentation

4.45.2.1 PutPreVar()

```
int PutPreVar (
            UBYTE * name,
            UBYTE * value,
            UBYTE * args,
            int mode )
```

Inserts/Updates a preprocessor variable in the name administration.

Parameters

name	Character string with the variable name.
value	Character string with a possible value. Special case: if this argument is zero, then we have no value. Note: This is different from having an empty argument! This should only occur when the name starts with a ?
args	Character string with possible arguments.
mode	=0: always create a new name entry, =1: try to do a redefinition if possible.

Returns

Index of used entry in name list.

Definition at line 642 of file pre.c.

4.45.2.2 TheDefine()

```
int TheDefine (
            UBYTE * s,
            int mode )
```

Preprocessor assignment. Possible arguments and values are treated and the new preprocessor variable is put into the name administration.

Parameters

s	Pointer to the character string following the preprocessor command.
mode	Bitmask. 0-bit clear: always create a new name entry, 0-bit set: try to redefine an existing name, 1-bit set: ignore preprocessor if/switch status.

Returns

zero: no errors, negative number: errors.

Definition at line 1942 of file pre.c.

4.45.2.3 DoPreAppendPath()

```
int DoPreAppendPath (
            UBYTE * s )
```

Appends the given path (absolute or relative to the current file directory) to the FORM path.

Syntax: #appendpath <path>

Definition at line 6954 of file pre.c.

4.45.2.4 DoPrePrependPath()

```
int DoPrePrependPath (
            UBYTE * s )
```

Prepends the given path (absolute or relative to the current file directory) to the FORM path.

Syntax: #prependpath <path>

Definition at line 6971 of file pre.c.

4.46 proces.c File Reference

```
#include "form3.h"
```

Functions

- WORD Processor ()
- WORD TestSub (PHEAD WORD *term, WORD level)
- WORD InFunction (PHEAD WORD *term, WORD *termout)
- WORD InsertTerm (PHEAD WORD *term, WORD replac, WORD extractbuff, WORD *position, WORD *termout, WORD tepos)
- LONG PasteFile (PHEAD WORD number, WORD *accum, POSITION *position, WORD **accfill, RENUMBER renumber, WORD *freeze, WORD nexpr)
- WORD * PasteTerm (PHEAD WORD number, WORD *accum, WORD *position, WORD times, WORD divby)
- WORD FiniTerm (PHEAD WORD *term, WORD *accum, WORD *termout, WORD number, WORD tepos)
- WORD Generator (PHEAD WORD *term, WORD level)
- WORD DoOnePow (PHEAD WORD *term, WORD power, WORD nexp, WORD *accum, WORD *aa, WORD level, WORD *freeze)
- WORD Deferred (PHEAD WORD *term, WORD level)
- WORD PrepPoly (PHEAD WORD *term, WORD par)
- WORD PolyFunMul (PHEAD WORD *term)

Variables

- WORD **printscratch** [2]

4.46.1 Detailed Description

Contains the central terms processor routines. This is the core of the virtual machine. All other files are to help these routines.

4.46.2 Function Documentation

4.46.2.1 Processor()

```
WORD Processor ( )
```

This is the central processor. It accepts a stream of Expressions which is accessed by calls to GetTerm. The expressions reside either in AR.infile or AR.hidefile The definitions of an expression are seen as an id-statement, so the primary Expressions should be written to the system of scratch files as single terms with an expression pointer. Each expression is terminated with a zero and the whole is terminated by two zeroes.

The routine DoExecute should determine whether results are to be printed, should revert the scratch I/O directions etc. In principle it is DoExecute that calls Processor.

Returns

if everything OK: 0. Otherwise error. The preprocessor may continue with compilation though. Really fatal errors should return on the spot by calling Terminate.

Definition at line 64 of file proces.c.

4.46.2.2 TestSub()

```
WORD TestSub (
             PHEAD WORD * term,
             WORD level )
```

TestSub hunts for subexpression pointers. If one is found its power is given in AN.TeSuOut. and the returnvalue is 'expressionnumber'. If the expression number is negative it is an expression on disk.

In addition this routine tries to locate subexpression pointers in functions. It also notices that action must be taken with any of the special functions.

Parameters

term	The term in which TestSub hunts for potential action
level	The number of the 'level' in the compiler buffer.

Returns

The number of the (sub)expression that was encountered.

Other values that are returned are in AN.TeSuOut, AR.TePos, AT.TMbuff, AN.TeInFun, AN.Frozen, AT.TMaddr

The level in the compiler buffer is more or less the number of the statement in the module. Hence it refers to the element in the lhs array.

This routine is one of the most important routines in FORM.

Definition at line 681 of file proces.c.

4.46.2.3 InFunction()

```
WORD InFunction (
             PHEAD WORD * term,
             WORD * termout )
```

Makes the replacement of the subexpression with the number 'replac' in a function argument. Additional information is passed in some of the AR, AN, AT variables.

Parameters

term	The input term
termout	The output term

Returns

0: everything is fine, Negative: fatal, Positive: error.

Special attention should be given to nested functions!

Definition at line 2033 of file proces.c.

4.46.2.4 InsertTerm()

```
WORD InsertTerm (
             PHEAD WORD * term,
             WORD replac,
             WORD extractbuff,
             WORD * position,
             WORD * termout,
             WORD tepos )
```

Puts the terms 'term' and 'position' together into a single legal term in termout. replac is the number of the subexpression that should be replaced. It must be a positive term. When action is needed in the argument of a function all terms in that argument are dealt with recursively. The subexpression is sorted. Only one subexpression is done at a time this way.

Parameters

term	the input term
replac	number of the subexpression pointer to replace
extractbuff	number of the compiler buffer replac refers to
position	position from where to take the term in the compiler buffer
termout	the output term
tepos	offset in term where the subexpression is.

Returns

Normal conventions (OK = 0).

Definition at line 2579 of file proces.c.

4.46.2.5 PasteFile()

```
LONG PasteFile (
             PHEAD WORD number,
             WORD * accum,
             POSITION * position,
             WORD ** accfill,
             RENUMBER renumber,
             WORD * freeze,
             WORD nexpr )
```

Gets a term from stored expression expr and puts it in the accumulator at position number. It returns the length of the term that came from file.

Parameters

number	number of partial terms to skip in accum
accum	the accumulator
position	file position from where to get the stored term
accfill	returns tail position in accum
renumber	the renumber struct for the variables in the stored expression
freeze	information about if we need only the contents of a bracket
nexpr	the number of the stored expression

Returns

Normal conventions (OK = 0).

Definition at line 2715 of file proces.c.

4.46.2.6 PasteTerm()

```
WORD* PasteTerm (
              PHEAD WORD number,
              WORD * accum,
              WORD * position,
              WORD times,
              WORD divby )
```

Puts the term at position in the accumulator accum at position 'number+1'. if times $>$ 0 the coefficient of this term is multiplied by times/divby.

Parameters

number	The number of term fragments in accum that should be skipped
accum	The accumulator of term fragments
position	A position in (typically) a compiler buffer from where a (piece of a) term comes.
times	Multiply the result by this
divby	Divide the result by this.

This routine is typically used when we have to replace a (sub)expression pointer by a power of a (sub)expression. This uses mostly a binomial expansion and the new term is the old term multiplied one by one by terms of the new expression. The factors times and divby keep track of the binomial coefficient. Once this is complete, the routine FiniTerm will make the contents of the accumulator into a proper term that still needs to be normalized.

Definition at line 2837 of file proces.c.

4.46.2.7 FiniTerm()

```
WORD FiniTerm (
              PHEAD WORD * term,
```

```
            WORD * accum,
            WORD * termout,
            WORD number,
            WORD tepos )
```

Concatenates the contents of the accumulator into a single legal term, which replaces the subexpression pointer

Parameters

term	the input term with the (sub)expression subterm
accum	the accumulator with the term fragments
termout	the location where the output should be written
number	the number of term fragments in the accumulator
tepos	the position of the subterm in term to be replaced

Definition at line 2902 of file proces.c.

4.46.2.8 Generator()

```
WORD Generator (
            PHEAD WORD * term,
            WORD level )
```

The heart of the program. Here the expansion tree is set up in one giant recursion

Parameters

term	the input term. may be overwritten
level	the level in the compiler buffer (number of statement)

Returns

Normal conventions (OK = 0).

The routine looks first whether there are unsubstituted (sub)expressions. If so, one of them gets inserted term by term and the new term is used in a renewed call to Generator. If there are no (sub)expressions, the term is normalized, the compiler level is raised (next statement) and the program looks what type of statement this is. If this is a special statement it is either treated on the spot or the appropriate routine is called. If it is a substitution, the pattern matcher is called (TestMatch) which tells whether there was a match. If so we need to call TestSub again to test for (sub)expressions. If we run out of levels, the term receives a final treatment for modulus calculus and/or brackets and is then sent off to the sorting routines.

Definition at line 3101 of file proces.c.

4.46.2.9 DoOnePow()

```
WORD DoOnePow (
              PHEAD WORD * term,
              WORD power,
              WORD nexp,
              WORD * accum,
              WORD * aa,
              WORD level,
              WORD * freeze )
```

Routine gets one power of an expression in the scratch system. If there are more powers needed there will be a recursion.

No attempt is made to use binomials because we have no information about commutating properties.

There is a searching for the contents of brackets if needed. This searching may be rather slow because of the single links.

Parameters

term	is the term we are adding to.
power	is the power of the expression that we need.
nexp	is the number of the expression.
accum	is the accumulator of terms. It accepts the termfragments that are made into a proper term in FiniTerm
aa	points to the start of the entire accumulator. In $*$aa we store the number of term fragments that are in the accumulator.
level	is the current depth in the tree of statements. It is needed to continue to the next operation/substitution with each generated term
freeze	is the pointer to the bracket information that should be matched.

Definition at line 4395 of file proces.c.

4.46.2.10 Deferred()

```
WORD Deferred (
              PHEAD WORD * term,
              WORD level )
```

Picks up the deferred brackets. These are the bracket contents of which we postpone the reading when we use the 'Keep Brackets' statement. These contents are multiplying the terms just before they are sent to the sorting system. Special attention goes to having it thread-safe We have to lock positioning the file and reading it in a thread specific buffer.

Parameters

term	The term that must be multiplied by the contents of the current bracket
level	The compiler level. This is needed because after multiplying term by term we call Generator again.

Definition at line 4616 of file proces.c.

4.46.2.11 PrepPoly()

```
WORD PrepPoly (
            PHEAD WORD * term,
            WORD par )
```

Routine checks whether the count of function AR.PolyFun is zero or one. If it is one and it has one scalarlike argument the coefficient of the term is pulled inside the argument. If the count is zero a new function is made with the coefficient as its only argument. The function should be placed at its proper position.

When this function is active it places the PolyFun as last object before the coefficient. This is needed because otherwise the compress algorithm has problems in MergePatches.

The bracket routine should also place the PolyFun at a comparable spot. The compression should then stop at the PolyFun. It doesn't really have to stop when writing the final result but this may be too complicated.

The parameter par tells whether we are at groundlevel or inside a function or dollar variable.

Definition at line 4744 of file proces.c.

4.46.2.12 PolyFunMul()

```
WORD PolyFunMul (
            PHEAD WORD * term )
```

Multiplies the arguments of multiple occurrences of the polyfun. In this routine we do the original PolyFun with one argument only. The PolyRatFun (PolyFunType = 2) is done in a dedicated routine in the file polywrap.cc The new result is written over the old result.

Parameters

term	It contains the input term and later the output.

Returns

Normal conventions (OK = 0).

Definition at line 5132 of file proces.c.

4.47 ratio.c File Reference

```
#include "form3.h"
```

Data Structures

- struct ARGBUFFER

Functions

- WORD **RatioFind** (PHEAD WORD *term, WORD *params)
- WORD **RatioGen** (PHEAD WORD *term, WORD *params, WORD num, WORD level)
- WORD **BinomGen** (PHEAD WORD *term, WORD level, WORD **tstops, WORD x1, WORD x2, WORD pow1, WORD pow2, WORD sign, UWORD *coef, WORD ncoef)
- WORD **DoSumF1** (PHEAD WORD *term, WORD *params, WORD replac, WORD level)
- WORD **Glue** (PHEAD WORD *term1, WORD *term2, WORD *sub, WORD insert)
- WORD **DoSumF2** (PHEAD WORD *term, WORD *params, WORD replac, WORD level)
- int **GCDfunction** (PHEAD WORD *term, WORD level)
- WORD * **GCDfunction3** (PHEAD WORD *in1, WORD *in2)
- WORD * **PutExtraSymbols** (PHEAD WORD *in, WORD startebuf, int *actionflag)
- WORD * **TakeExtraSymbols** (PHEAD WORD *in, WORD startebuf)
- WORD * **MultiplyWithTerm** (PHEAD WORD *in, WORD *term, WORD par)
- WORD * TakeContent (PHEAD WORD *in, WORD *term)
- int **MergeSymbolLists** (PHEAD WORD *old, WORD *extra, int par)
- int **MergeDotproductLists** (PHEAD WORD *old, WORD *extra, int par)
- WORD * **CreateExpression** (PHEAD WORD nexp)
- int **GCDterms** (PHEAD WORD *term1, WORD *term2, WORD *termout)
- int **ReadPolyRatFun** (PHEAD WORD *term)
- int **FromPolyRatFun** (PHEAD WORD *fun, WORD **numout, WORD **denout)
- WORD * TakeSymbolContent (PHEAD WORD *in, WORD *term)
- void **GCDclean** (PHEAD WORD *num, WORD *den)
- WORD * **PolyDiv** (PHEAD WORD *a, WORD *b, char *text)
- int **DIVfunction** (PHEAD WORD *term, WORD level, int par)
- WORD * **MULfunc** (PHEAD WORD *p1, WORD *p2)
- WORD * **ConvertArgument** (PHEAD WORD *arg, int *type)
- int **ExpandRat** (PHEAD WORD *fun)
- int **InvPoly** (PHEAD WORD *inpoly, WORD maxpow, WORD sym)

Variables

- WORD **divrem** [4] = { DIVFUNCTION, REMFUNCTION, INVERSEFUNCTION, MULFUNCTION }
- char * **TheErrorMessage** [ ]

4.47.1 Detailed Description

A variety of routines: The ratio command for partial fractioning (rather old. Schoonschip inheritance) The sum routines.

4.47.2 Function Documentation

4.47.2.1 TakeContent()

```
WORD* TakeContent (
            PHEAD WORD * in,
            WORD * term )
```

Implements part of the old ExecArg in which we take common factors from arguments with more than one term. Here the input is a sequence of terms in 'in' and the answer is a content-free sequence of terms. This sequence has been allocated by the Malloc1 routine in a call to EndSort, unless the expression was already content-free. In that case the input pointer is returned. The content is returned in term. This is supposed to be a separate allocation, made by TermMalloc in the calling routine.

Definition at line 1376 of file ratio.c.

4.47.2.2 TakeSymbolContent()

```
WORD* TakeSymbolContent (
            PHEAD WORD * in,
            WORD * term )
```

Implements part of the old ExecArg in which we take common factors from arguments with more than one term. We allow only symbols as this code is used for the polyratfun only. We have a special routine, because the generic TakeContent does too much work and speed is at a premium here. Input: in is the input expression as a sequence of terms. Output: term: the content return value: the contentfree expression. it is in new allocation, made by TermMalloc. (should be in a TermMalloc space?)

Definition at line 2434 of file ratio.c.

4.47.3 Variable Documentation

4.47.3.1 TheErrorMessage

```
char* TheErrorMessage[]
```

Initial value:

```
= {
     "PolyRatFun not of a type that FORM will expand: incorrect variable inside."
    ,"Division by zero in PolyRatFun encountered in ExpandRat."
    ,"Irregular code in PolyRatFun encountered in ExpandRat."
    ,"Called from ExpandRat."
    ,"WorkSpace overflow. Change parameter WorkSpace in setup file?"
    }
```

Definition at line 3105 of file ratio.c.

4.48 reken.c File Reference

```
#include "form3.h"
#include <math.h>
```

Macros

- #define **GCDMAX** 3
- #define **NEWTRICK** 1
- #define **COPYLONG**(x1, nx1, x2, nx2) { int i; for(i=0;i<ABS(nx2);i++)x1[i]=x2[i];nx1=nx2; }
- #define **WARMUP** 6

Functions

- VOID **Pack** (UWORD *a, WORD *na, UWORD *b, WORD nb)
- VOID **UnPack** (UWORD *a, WORD na, WORD *denom, WORD *numer)
- WORD **Mully** (PHEAD UWORD *a, WORD *na, UWORD *b, WORD nb)
- WORD **Divvy** (PHEAD UWORD *a, WORD *na, UWORD *b, WORD nb)
- WORD **AddRat** (PHEAD UWORD *a, WORD na, UWORD *b, WORD nb, UWORD *c, WORD *nc)
- WORD **MulRat** (PHEAD UWORD *a, WORD na, UWORD *b, WORD nb, UWORD *c, WORD *nc)
- WORD **DivRat** (PHEAD UWORD *a, WORD na, UWORD *b, WORD nb, UWORD *c, WORD *nc)
- WORD **Simplify** (PHEAD UWORD *a, WORD *na, UWORD *b, WORD *nb)
- WORD **AccumGCD** (PHEAD UWORD *a, WORD *na, UWORD *b, WORD nb)
- int **TakeRatRoot** (UWORD *a, WORD *n, WORD power)
- WORD **AddLong** (UWORD *a, WORD na, UWORD *b, WORD nb, UWORD *c, WORD *nc)
- WORD **AddPLon** (UWORD *a, WORD na, UWORD *b, WORD nb, UWORD *c, WORD *nc)
- VOID **SubPLon** (UWORD *a, WORD na, UWORD *b, WORD nb, UWORD *c, WORD *nc)
- WORD **MulLong** (UWORD *a, WORD na, UWORD *b, WORD nb, UWORD *c, WORD *nc)
- WORD **BigLong** (UWORD *a, WORD na, UWORD *b, WORD nb)
- WORD **DivLong** (UWORD *a, WORD na, UWORD *b, WORD nb, UWORD *c, WORD *nc, UWORD *d, WORD *nd)
- WORD **RaisPow** (PHEAD UWORD *a, WORD *na, UWORD b)
- VOID RaisPowCached (PHEAD WORD x, WORD n, UWORD **c, WORD *nc)
- WORD **RaisPowMod** (WORD x, WORD n, WORD m)
- int NormalModulus (UWORD *a, WORD *na)
- int MakeInverses ()
- int GetModInverses (WORD m1, WORD m2, WORD *im1, WORD *im2)
- int **GetLongModInverses** (PHEAD UWORD *a, WORD na, UWORD *b, WORD nb, UWORD *ia, WORD *nia, UWORD *ib, WORD *nib)
- WORD **Product** (UWORD *a, WORD *na, WORD b)
- UWORD **Quotient** (UWORD *a, WORD *na, WORD b)
- WORD **Remain10** (UWORD *a, WORD *na)
- WORD **Remain4** (UWORD *a, WORD *na)
- VOID **PrtLong** (UWORD *a, WORD na, UBYTE *s)
- WORD **GetLong** (UBYTE *s, UWORD *a, WORD *na)
- WORD **GcdLong** (PHEAD UWORD *a, WORD na, UWORD *b, WORD nb, UWORD *c, WORD *nc)
- WORD **GetBinom** (UWORD *a, WORD *na, WORD i1, WORD i2)
- WORD **LcmLong** (PHEAD UWORD *a, WORD na, UWORD *b, WORD nb, UWORD *c, WORD *nc)
- int **TakeLongRoot** (UWORD *a, WORD *n, WORD power)
- int **MakeRational** (WORD a, WORD m, WORD *b, WORD *c)
- int **MakeLongRational** (PHEAD UWORD *a, WORD na, UWORD *m, WORD nm, UWORD *b, WORD *nb)
- WORD CompCoef (WORD *term1, WORD *term2)
- WORD **Modulus** (WORD *term)
- WORD **TakeModulus** (UWORD *a, WORD *na, UWORD *cmodvec, WORD ncmod, WORD par)
- WORD **TakeNormalModulus** (UWORD *a, WORD *na, UWORD *c, WORD nc, WORD par)
- WORD **MakeModTable** ()
- int **Factorial** (PHEAD WORD n, UWORD *a, WORD *na)
- int **Bernoulli** (WORD n, UWORD *a, WORD *na)
- WORD NextPrime (PHEAD WORD num)
- void **iniwranf** (PHEAD0)
- UWORD **wranf** (PHEAD0)
- UWORD **iranf** (PHEAD UWORD imax)
- UBYTE * **PreRandom** (UBYTE *s)

4.48.1 Detailed Description

This file contains the numerical routines. The arithmetic in FORM is normally over the rational numbers. Hence there are routines for dealing with integers and with rational of 'arbitrary precision' (within limits) There are also routines for that calculus modulus an integer. In addition there are the routines for factorials and bernoulli numbers. The random number function is currently only for internal purposes.

4.48.2 Function Documentation

4.48.2.1 RaisPowCached()

```
VOID RaisPowCached (
            PHEAD WORD x,
            WORD n,
            UWORD ** c,
            WORD * nc )
```

Computes power x^n and caches the value

4.48.3 Description

Calculates the power x^n and stores the results for caching purposes. The pointer c (i.e., the pointer, and not what it points to) is overwritten. What it points to should not be overwritten in the calling function.

4.48.4 Notes

- Caching is done in AT.small_power[]. This array is extended if necessary.

Definition at line 1286 of file reken.c.

Referenced by poly::divmod_univar(), and poly::mul_heap().

4.48.4.1 NormalModulus()

```
int NormalModulus (
            UWORD * a,
            WORD * na )
```

Brings a modular representation in the range -p/2 to +p/2 The return value tells whether anything was done. Routine made in the general modulus revamp of July 2008 (JV).

Definition at line 1393 of file reken.c.

4.48.4.2 MakeInverses()

```
int MakeInverses ( )
```

Makes a table of inverses in modular calculus The modulus is in AC.cmod and AC.ncmod One should notice that the table of inverses can only be made if the modulus fits inside a single FORM word. Otherwise the table lookup becomes too difficult and the table too long.

Definition at line 1430 of file reken.c.

4.48.4.3 GetModInverses()

```
int GetModInverses (
            WORD m1,
            WORD m2,
            WORD * im1,
            WORD * im2 )
```

Input m1 and m2, which are relative prime. determines a∗m1+b∗m2 = 1 (and 1 is the gcd of m1 and m2) then a∗m1 = 1 mod m2 and hence im1 = a. and b∗m2 = 1 mod m1 and hence im2 = b. Set m1 = 0∗m1+1∗m2 = a1∗m1+b1∗m2 m2 = 1∗m1+0∗m2 = a2∗m1+b2∗m2 If everything is OK, the return value is zero

Definition at line 1466 of file reken.c.

4.48.4.4 CompCoef()

```
WORD CompCoef (
            WORD * term1,
            WORD * term2 )

Routine takes a1 mod m1 and a2 mod m2 and returns a mod m1*m2 with
a mod m1 = a1 and a mod m2 = a2
```

Chinese remainder: a%(m1∗m2) = q1∗m1+a1 a%(m1∗m2) = q2∗m2+a2 Compute n1 such that (n1∗m1)m2 is one Compute n2 such that (n2∗m2)m1 is one Then (a1∗n2∗m2+a2∗n1∗m1)%(m1∗m2) is a%(m1∗m2)

Definition at line 3037 of file reken.c.

4.48.4.5 NextPrime()

```
WORD NextPrime (
            PHEAD WORD num )
```

Gives the next prime number in the list of prime numbers.

If the list isn't long enough we expand it. For ease in ParForm and because these lists shouldn't be very big we let each worker keep its own list.

The list is cut off at MAXPOWER, because we don't want to get into trouble that the power of a variable gets larger than the prime number.

Definition at line 3654 of file reken.c.

4.49 reshuf.c File Reference

```
#include "form3.h"
```

Functions

- WORD **ReNumber** (PHEAD WORD ∗term)
- VOID **FunLevel** (PHEAD WORD ∗term)
- WORD **DetCurDum** (PHEAD WORD ∗t)
- int **FullRenumber** (PHEAD WORD ∗term, WORD par)
- VOID **MoveDummies** (PHEAD WORD ∗term, WORD shift)
- void **AdjustRenumScratch** (PHEAD0)
- WORD **CountDo** (WORD ∗term, WORD ∗instruct)
- WORD **CountFun** (WORD ∗term, WORD ∗countfun)
- WORD **DimensionSubterm** (WORD ∗subterm)
- WORD **DimensionTerm** (WORD ∗term)
- WORD **DimensionExpression** (PHEAD WORD ∗expr)
- WORD **MultDo** (PHEAD WORD ∗term, WORD ∗pattern)
- WORD **TryDo** (PHEAD WORD ∗term, WORD ∗pattern, WORD level)
- WORD **DoDistrib** (PHEAD WORD ∗term, WORD level)
- WORD **EqualArg** (WORD ∗parms, WORD num1, WORD num2)
- WORD **DoDelta3** (PHEAD WORD ∗term, WORD level)
- WORD **TestPartitions** (WORD ∗tfun, PARTI ∗parti)
- WORD **DoPartitions** (PHEAD WORD ∗term, WORD level)
- WORD **DoPermutations** (PHEAD WORD ∗term, WORD level)
- WORD **DoShuffle** (WORD ∗term, WORD level, WORD fun, WORD option)
- int **Shuffle** (WORD ∗from1, WORD ∗from2, WORD ∗to)
- int **FinishShuffle** (WORD ∗fini)
- WORD **DoStuffle** (WORD ∗term, WORD level, WORD fun, WORD option)
- int **Stuffle** (WORD ∗from1, WORD ∗from2, WORD ∗to)
- int **FinishStuffle** (WORD ∗fini)
- WORD ∗ **StuffRootAdd** (WORD ∗t1, WORD ∗t2, WORD ∗to)

4.49.1 Detailed Description

Mixed routines: Routines for relabelling dummy indices. The multiply command The distrib_ function The tryreplace statement

4.50 sch.c File Reference

```
#include "form3.h"
```

Macros

- #define **va_dcl** int va_alist;
- #define **va_start**(list) list = (UBYTE ∗) &va_alist
- #define **va_end**(list)
- #define **va_arg**(list, mode) (((mode ∗)(list += sizeof(mode)))[-1])

Typedefs

- typedef UBYTE * **va_list**

Functions

- UBYTE * **StrCopy** (UBYTE *from, UBYTE *to)
- VOID **AddToLine** (UBYTE *s)
- VOID **FiniLine** ()
- VOID **IniLine** (WORD extrablank)
- VOID **LongToLine** (UWORD *a, WORD na)
- VOID **RatToLine** (UWORD *a, WORD na)
- VOID **TalToLine** (UWORD x)
- VOID **TokenToLine** (UBYTE *s)
- UBYTE * **CodeToLine** (WORD number, UBYTE *Out)
- void **MultiplyToLine** ()
- UBYTE * **AddArrayIndex** (WORD num, UBYTE *out)
- VOID **PrtTerms** ()
- UBYTE * **WrtPower** (UBYTE *Out, WORD Power)
- void **PrintTime** (UBYTE *mess)
- VOID **WriteLists** ()
- void **WriteDictionary** (DICTIONARY *dict)
- VOID **WriteArgument** (WORD *t)
- WORD **WriteSubTerm** (WORD *sterm, WORD first)
- WORD **WriteInnerTerm** (WORD *term, WORD first)
- WORD **WriteTerm** (WORD *term, WORD *lbrac, WORD first, WORD prtf, WORD br)
- WORD **WriteExpression** (WORD *terms, LONG ltot)
- WORD **WriteAll** ()
- WORD **WriteOne** (UBYTE *name, int alreadyinline, int nosemi, WORD plus)

4.50.1 Detailed Description

Contains the functions that deal with the writing of expressions/terms in a textual representation. (Dutch schrijven = to write)

4.51 setfile.c File Reference

```
#include "form3.h"
```

Macros

- #define **NUMERICALVALUE** 0
- #define **STRINGVALUE** 1
- #define **PATHVALUE** 2
- #define **ONOFFVALUE** 3
- #define **DEFINEVALUE** 4
- #define **SETBUFSIZE** 257

Functions

- int **DoSetups** ()
- int **ProcessOption** (UBYTE ∗s1, UBYTE ∗s2, int filetype)
- SETUPPARAMETERS ∗ **GetSetupPar** (UBYTE ∗s)
- int **RecalcSetups** ()
- int **AllocSetups** ()
- VOID **WriteSetup** ()
- SORTING ∗ **AllocSort** (LONG LargeSize, LONG SmallSize, LONG SmallEsize, LONG TermsInSmall, int MaxPatches, int MaxFpatches, LONG IOsize)
- VOID **AllocSortFileName** (SORTING ∗sort)
- FILEHANDLE ∗ **AllocFileHandle** (WORD par, char ∗name)
- void **DeAllocFileHandle** (FILEHANDLE ∗fh)
- int **MakeSetupAllocs** ()
- int **TryFileSetups** ()
- int **TryEnvironment** ()

Variables

- char **curdirp** [ ] = "."
- char **cursortdirp** [ ] = "."
- char **commentchar** [ ] = "∗"
- char **dotchar** [ ] = "_"
- char **highfirst** [ ] = "highfirst"
- char **lowfirst** [ ] = "lowfirst"
- char **procedureextension** [ ] = "prc"
- SETUPPARAMETERS **setupparameters** [ ]

4.51.1 Detailed Description

The routines that deal with the setup parameters.

4.52 smart.c File Reference

```
#include "form3.h"
```

Functions

- int **StudyPattern** (WORD ∗lhs)
- int **MatchIsPossible** (WORD ∗pattern, WORD ∗term)

4.52.1 Detailed Description

The functions for smart pattern searches in combinations of functions. When many wildcards are involved and the functions are (anti)symmetric an exhaustive search for all possibilities may take very much time (like factorial in the number of wildcards) while a human can often see immediately that there cannot be a match. The routines here try to make FORM a bit smarter in this respect.

This is just the beginning. It still needs lots of work!

4.53 sort.c File Reference

```
#include "form3.h"
```

Macros

- #define **NEWSPLITMERGE**
- #define **INSLENGTH**(x) w[1] = FUNHEAD+ARGHEAD+x; w[FUNHEAD] = ARGHEAD+x;

Functions

- VOID WriteStats (POSITION *plspace, WORD par)
- WORD NewSort (PHEAD0)
- LONG EndSort (PHEAD WORD *buffer, int par)
- LONG PutIn (FILEHANDLE *file, POSITION *position, WORD *buffer, WORD **take, int npat)
- WORD Sflush (FILEHANDLE *fi)
- WORD PutOut (PHEAD WORD *term, POSITION *position, FILEHANDLE *fi, WORD ncomp)
- WORD FlushOut (POSITION *position, FILEHANDLE *fi, int compr)
- WORD AddCoef (PHEAD WORD **ps1, WORD **ps2)
- WORD AddPoly (PHEAD WORD **ps1, WORD **ps2)
- VOID AddArgs (PHEAD WORD *s1, WORD *s2, WORD *m)
- WORD Compare1 (WORD *term1, WORD *term2, WORD level)
- WORD CompareSymbols (WORD *term1, WORD *term2, WORD par)
- WORD CompareHSymbols (WORD *term1, WORD *term2, WORD par)
- LONG ComPress (WORD **ss, LONG *n)
- LONG SplitMerge (PHEAD WORD **Pointer, LONG number)
- VOID GarbHand ()
- WORD MergePatches (WORD par)
- WORD StoreTerm (PHEAD WORD *term)
- VOID StageSort (FILEHANDLE *fout)
- WORD SortWild (WORD *w, WORD nw)
- void CleanUpSort (int num)
- VOID LowerSortLevel ()
- WORD * **PolyRatFunSpecial** (PHEAD WORD *t1, WORD *t2)
- VOID **SimpleSplitMergeRec** (WORD *array, WORD num, WORD *auxarray)
- VOID **SimpleSplitMerge** (WORD *array, WORD num)
- WORD **BinarySearch** (WORD *array, WORD num, WORD x)

Variables

- LONG **numcompares**
- char * **toterms** [ ] = { " ", " >>", "-->" }

4.53.1 Detailed Description

Contains the sort routines. We distinguish levels of sorting. The ground level is the sorting of terms in an expression. When a term has functions, the arguments can contain terms that need sorting, which this then done by raising the level. This can happen recursively. NewSort and EndSort automatically raise and lower the level. Because the ground level does some special things, sometimes we have to raise immediately to the second level skipping the ground level.

Special routines for the parallel sorting are in the file threads.c Also the sorting of terms in polynomials is special but most of that is controled by changing the address of the compare routine. Other routines relevant for adding rational polynomials are in the file polynito.c

4.53.2 Function Documentation

4.53.2.1 WriteStats()

```
VOID WriteStats (
            POSITION * plspace,
            WORD par )
```

Writes the statistics.

Parameters

plspace	The size in bytes currently occupied
par	par = 0 after a splitmerge. par = 1 after merge to sortfile. par = 2 after the sort

current expression is to be found in AR.CurExpr. terms are in S->TermsLeft. S->GenTerms.

Definition at line 93 of file sort.c.

4.53.2.2 NewSort()

```
WORD NewSort (
            PHEAD0  )
```

Starts a new sort. At the lowest level this is a 'main sort' with the struct according to the parameters in S0. At higher levels this is a sort for functions, subroutines or dollars. We prepare the arrays and structs.

Returns

Regular convention (OK -> 0)

Definition at line 592 of file sort.c.

4.53.2.3 EndSort()

```
LONG EndSort (
            PHEAD WORD * buffer,
            int par )
```

Finishes a sort. At AR.sLevel == 0 the output is to the regular output stream. When AR.sLevel $>$ 0, the parameter par determines the actual output. The AR.sLevel will be popped. All ongoing stages are finished and if the sortfile is open it is closed. The statistics are printed when AR.sLevel == 0 par == 0 Output to the buffer. par == 1 Sort for function arguments. The output will be copied into the buffer. It is assumed that this is in the WorkSpace. par == 2 Sort for $-variable. We return the address of the buffer that contains the output in buffer (treated like WORD $**$). We first catch the output in a file (unless we can intercept things after the small buffer has been sorted) Then we read from the file into a buffer. Only when par == 0 data compression can be attempted at AT.SS==AT.S0.

Parameters

buffer	buffer for output when needed
par	See above

Returns

If negative: error. If positive: number of words in output.

Definition at line 682 of file sort.c.

4.53.2.4 PutIn()

```
LONG PutIn (
            FILEHANDLE * file,
            POSITION * position,
            WORD * buffer,
            WORD ** take,
            int npat )
```

Reads a new patch from position in file handle. It is put at buffer, anything after take is moved forward. This would be part of a term that hasn't been used yet. Because of this there should be some space before the start of the buffer

Parameters

file	The file system from which to read
position	The position from which to read
buffer	The buffer into which to read
take	The unused tail should be moved before the buffer
npat	The number of the patch. Is needed if the information was compressed with gzip, because each patch has its own independent gzip encoding.

Definition at line 1259 of file sort.c.

4.53.2.5 Sflush()

```
WORD Sflush (
            FILEHANDLE * fi )
```

Puts the contents of a buffer to output Only to be used when there is a single patch in the large buffer.

Parameters

fi	The filesystem (or its cache) to which the patch should be written

Definition at line 1319 of file sort.c.

4.53.2.6 PutOut()

```
WORD PutOut (
              PHEAD WORD * term,
              POSITION * position,
              FILEHANDLE * fi,
              WORD ncomp )
```

Routine writes one term to file handle at position. It returns the new value of the position.

NOTE: For 'final output' we may have to index the brackets. See the struct BRACKETINDEX. We should maintain: 1: a list with brackets array with the brackets 2: a list of objects of type BRACKETINDEX. It contains array with either pointers or offsets to the list of brackets. starting positions in the file. The index may be tied to a maximum size. In that case we may have to prune the list occasionally.

Parameters

term	The term to be written
position	The position in the file. Afterwards it is updated
fi	The file (or its cache) to which should be written
ncomp	Information about what type of compression should be used

Definition at line 1405 of file sort.c.

4.53.2.7 FlushOut()

```
WORD FlushOut (
              POSITION * position,
              FILEHANDLE * fi,
              int compr )
```

Completes output to an output file and writes the trailing zero.

Parameters

position	The position in the file after writing
fi	The file (or its cache)
compr	Indicates whether there should be compression with gzip.

Returns

Regular conventions (OK -> 0).

Definition at line 1748 of file sort.c.

4.53.2.8 AddCoef()

```
WORD AddCoef (
            PHEAD WORD ** ps1,
            WORD ** ps2 )
```

Adds the coefficients of the terms ∗ps1 and ∗ps2. The problem comes when there is not enough space for a new longer coefficient. First a local solution is tried. If this is not succesfull we need to move terms around. The possibility of a garbage collection should not be ignored, as avoiding this costs very much extra space which is nearly wasted otherwise.

If the return value is zero the terms cancelled.

The resulting term is left in ∗ps1.

Definition at line 1962 of file sort.c.

4.53.2.9 AddPoly()

```
WORD AddPoly (
            PHEAD WORD ** ps1,
            WORD ** ps2 )
```

Routine should be called when S->PolyWise != 0. It points then to the position of AR.PolyFun in both terms.

We add the contents of the arguments of the two polynomials. Special attention has to be given to special arguments. We have to reserve a space equal to the size of one term + the size of the argument of the other. The addition has to be done in this routine because not all objects are reentrant.

Newer addition (12-nov-2007). The PolyFun can have two arguments. In that case S->PolyFlag is 2 and we have to call the routine for adding rational polynomials. We have to be rather careful what happens with: The location of the output The order of the terms in the arguments At first we allow only univariate polynomials in the PolyFun. This restriction will be lifted a.s.a.p.

Parameters

ps1	A pointer to the postion of the first term
ps2	A pointer to the postion of the second term

Returns

If zero the terms cancel. Otherwise the new term is in ∗ps1.

Definition at line 2089 of file sort.c.

4.53.2.10 AddArgs()

```
VOID AddArgs (
            PHEAD WORD * s1,
```

```
            WORD * s2,
            WORD * m )
```

Adds the arguments of two occurrences of the PolyFun.

Parameters

s1	Pointer to the first occurrence.
s2	Pointer to the second occurrence.
m	Pointer to where the answer should be.

Definition at line 2251 of file sort.c.

4.53.2.11 Compare1()

```
WORD Compare1 (
            WORD * term1,
            WORD * term2,
            WORD level )
```

Compares two terms. The answer is: 0 equal (with exception of the coefficient if level == 0.) >0 term1 comes first. <0 term2 comes first. Some special precautions may be needed to keep the CompCoef routine from generating overflows, although this is very unlikely in subterms. This routine should not return an error condition.

Originally this routine was called Compare. With the treatment of special polynomials with terms that contain only symbols and the need for extreme speed for the polynomial routines we made a special compare routine and now we store the address of the current compare routine in AR.CompareRoutine and have a macro Compare which makes all existing code work properly and we can just replace the routine on a thread by thread basis (each thread has its own AR struct).

Parameters

term1	First input term
term2	Second input term
level	The sorting level (may influence on the result)

Returns

0 equal (with exception of the coefficient if level == 0.) >0 term1 comes first. <0 term2 comes first.

Definition at line 2536 of file sort.c.

4.53.2.12 CompareSymbols()

```
WORD CompareSymbols (
            WORD * term1,
```

```
            WORD * term2,
            WORD par )
```

Compares the terms, based on the value of AN.polysortflag. If term1 $<$ term2 the return value is -1 If term1 $>$ term2 the return value is 1 If term1 = term2 the return value is 0 The coefficients may differ. The terms contain only a single subterm of type SYMBOL. If AN.polysortflag = 0 it is a 'regular' compare. If AN.polysortflag = 1 the sum of the powers is more important par is a dummy parameter to make the parameter field identical to that of Compare1 which is the regular compare routine in sort.c

Definition at line 2976 of file sort.c.

4.53.2.13 CompareHSymbols()

```
WORD CompareHSymbols (
            WORD * term1,
            WORD * term2,
            WORD par )
```

Compares terms that can have only SYMBOL and HAAKJE subterms. If term1 $<$ term2 the return value is -1 If term1 $>$ term2 the return value is 1 If term1 = term2 the return value is 0 par is a dummy parameter to make the parameter field identical to that of Compare1 which is the regular compare routine in sort.c

Definition at line 3020 of file sort.c.

4.53.2.14 ComPress()

```
LONG ComPress (
            WORD ** ss,
            LONG * n )
```

Gets a list of pointers to terms and compresses the terms. In n it collects the number of terms and the return value of the function is the space that is occupied.

We have to pay some special attention to the compression of terms with a PolyFun. This PolyFun should occur only straight before the coefficient, so we can use the same trick as for the coefficient to sabotage compression of this object (Replace in the history the function pointer by zero. This is safe, because terms that would be identical otherwise would have been added).

Parameters

ss	Array of pointers to terms to be compressed.
n	Number of pointers in ss.

Returns

Total number of words needed for the compressed result.

Definition at line 3074 of file sort.c.

4.53.2.15 SplitMerge()

```
LONG SplitMerge (
            PHEAD WORD ** Pointer,
            LONG number )
```

Algorithm by J.A.M.Vermaseren (31-7-1988)

Note that AN.SplitScratch and AN.InScratch are used also in GarbHand

Merge sort in memory. The input is an array of pointers. Sorting is done recursively by dividing the array in two equal parts and calling SplitMerge for each. When the parts are small enough we can do the compare and take the appropriate action. An addition is that we look for 'runs'. Sequences that are already ordered. This happens a lot when there is very little action in a module. This made FORM faster by a few percent.

Parameters

Pointer	The array of pointers to the terms to be sorted.
number	The number of pointers in Pointer.

The terms are supposed to be sitting in the small buffer and there is supposed to be an extension to this buffer for when there are two terms that should be added and the result takes more space than each of the original terms. The notation guarantees that the result never needs more space than the sum of the spaces of the original terms.

Definition at line 3240 of file sort.c.

4.53.2.16 GarbHand()

```
VOID GarbHand ( )
```

Garbage collection that takes place when the small extension is full and we need to place more terms there. When this is the case there are many holes in the small buffer and the whole can be compactified. The major complication is the buffer for SplitMerge. There are to options for temporary memory: 1: find some buffer that has enough space (maybe in the large buffer). 2: allocate a buffer. Give it back afterwards of course. If the small extension is properly dimensioned this routine should be called very rarely. Most of the time it will be called when the polyfun or polyratfun is active.

Definition at line 3462 of file sort.c.

4.53.2.17 MergePatches()

```
WORD MergePatches (
            WORD par )
```

The general merge routine. Can be used for the large buffer and the file merging. The array S->Patches tells where the patches start S->pStop tells where they end (has to be computed first). The end of a 'line to be merged' is indicated by a zero. If the end is reached without running into a zero or a term runs over the boundary of a patch it is a file merging operation and a new piece from the file is read in.

Parameters

par	If par == 0 the sort is for file -> outputfile. If par == 1 the sort is for large buffer -> sortfile. If par == 2 the sort is for large buffer -> outputfile.

Definition at line 3577 of file sort.c.

4.53.2.18 StoreTerm()

```
WORD StoreTerm (
            PHEAD WORD * term )
```

The central routine to accept terms, store them and keep things at least partially sorted. A call to EndSort will then complete storing and sorting.

Parameters

term	The term to be stored

Returns

Regular return conventions (OK -> 0)

Definition at line 4333 of file sort.c.

4.53.2.19 StageSort()

```
VOID StageSort (
            FILEHANDLE * fout )
```

Prepares a stage 4 or higher sort. Stage 4 sorts occur when the sort file contains more patches than can be merged in one pass.

Definition at line 4453 of file sort.c.

4.53.2.20 SortWild()

```
WORD SortWild (
            WORD * w,
            WORD nw )
```

Sorts the wildcard entries in the parameter w. Double entries are removed. Full space taken is nw words. Routine serves for the reading of wildcards in the compiler. The entries come in the format: (type,4,number,0) in which the zero is reserved for the future replacement of 'number'.

Parameters

w	buffer with wildcard entries.
nw	number of wildcard entries.

Returns

Normal conventions (OK -> 0)

Definition at line 4552 of file sort.c.

4.53.2.21 CleanUpSort()

```
void CleanUpSort (
            int num )
```

Partially or completely frees function sort buffers.

Definition at line 4644 of file sort.c.

4.53.2.22 LowerSortLevel()

```
VOID LowerSortLevel ( )
```

Lowers the level in the sort system.

Definition at line 4727 of file sort.c.

4.54 spectator.c File Reference

```
#include "form3.h"
```

Functions

- int **CoCreateSpectator** (UBYTE ∗inp)
- int **CoToSpectator** (UBYTE ∗inp)
- int **CoRemoveSpectator** (UBYTE ∗inp)
- int **CoEmptySpectator** (UBYTE ∗inp)
- int **PutInSpectator** (WORD ∗term, WORD specnum)
- void **FlushSpectators** (VOID)
- int **CoCopySpectator** (UBYTE ∗inp)
- WORD **GetFromSpectator** (WORD ∗term, WORD specnum)
- void **ClearSpectators** (WORD par)

4.54.1 Detailed Description

File contains the code for the spectator files and their control.

4.55 startup.c File Reference

```
#include "form3.h"
#include "inivar.h"
#include <signal.h>
```

Macros

- #define **STRINGIFY**(x) STRINGIFY__(x)
- #define **STRINGIFY__**(x) #x
- #define **FORMNAME** "FORM"
- #define **VERSIONSTR__** STRINGIFY(MAJORVERSION) "." STRINGIFY(MINORVERSION)
- #define **VERSIONSTR** FORMNAME " " VERSIONSTR__ " (" PRODUCTIONDATE ")"
- #define **TAKEPATH**(x) if(s[1]== '='){x=s+2;} else{x=∗argv++;argc--;}

Functions

- int **DoTail** (int argc, UBYTE ∗∗argv)
- int **OpenInput** ()
- VOID **ReserveTempFiles** (int par)
- VOID StartVariables ()
- VOID **StartMore** ()
- WORD **IniVars** ()
- int **main** (int argc, char ∗∗argv)
- VOID **CleanUp** (WORD par)
- VOID **Terminate** (int errorcode)
- VOID **PrintRunningTime** ()
- LONG **GetRunningTime** ()

Variables

- UBYTE ∗ **emptystring** = (UBYTE ∗)"."
- UBYTE ∗ **defaulttempfilename** = (UBYTE ∗)"xformxxx.str"

4.55.1 Detailed Description

This file contains the main program. It also deals with the very early stages of the startup of FORM and the final stages when the program attemps some cleanup. Here is the routine that analyses the command tail.

4.55.2 Function Documentation

4.55.2.1 StartVariables()

```
VOID StartVariables ( )
```

All functions (well, nearly all) are declared here.

Definition at line 840 of file startup.c.

4.56 store.c File Reference

```
#include "form3.h"
```

Macros

- #define **SAVEREVISION** 0x02

Functions

- WORD **OpenTemp** ()
- VOID **SeekScratch** (FILEHANDLE *fi, POSITION *pos)
- VOID **SetEndScratch** (FILEHANDLE *f, POSITION *position)
- VOID **SetEndHScratch** (FILEHANDLE *f, POSITION *position)
- VOID **SetScratch** (FILEHANDLE *f, POSITION *position)
- WORD **RevertScratch** ()
- WORD **ResetScratch** ()
- int **ReadFromScratch** (FILEHANDLE *fi, POSITION *pos, UBYTE *buffer, POSITION *length)
- int **AddToScratch** (FILEHANDLE *fi, POSITION *pos, UBYTE *buffer, POSITION *length, int withflush)
- int **CoSave** (UBYTE *inp)
- int **CoLoad** (UBYTE *inp)
- WORD **DeleteStore** (WORD par)
- WORD **PutInStore** (INDEXENTRY *ind, WORD num)
- WORD **GetTerm** (PHEAD WORD *term)
- WORD **GetOneTerm** (PHEAD WORD *term, FILEHANDLE *fi, POSITION *pos, int par)
- WORD **GetMoreTerms** (WORD *term)
- WORD **GetMoreFromMem** (WORD *term, WORD **tpoin)
- WORD **GetFromStore** (WORD *to, POSITION *position, RENUMBER renumber, WORD *InCompState, WORD nexpr)
- VOID **DetVars** (WORD *term, WORD par)
- WORD **ToStorage** (EXPRESSIONS e, POSITION *length)
- INDEXENTRY * **NextFileIndex** (POSITION *indexpos)
- WORD SetFileIndex ()
- WORD **VarStore** (UBYTE *s, WORD n, WORD name, WORD namesize)
- WORD TermRenumber (WORD *term, RENUMBER renumber, WORD nexpr)
- WORD **FindrNumber** (WORD n, VARRENUM *v)
- INDEXENTRY * **FindInIndex** (WORD expr, FILEDATA *f, WORD par, WORD mode)
- RENUMBER **GetTable** (WORD expr, POSITION *position, WORD mode)
- int **CopyExpression** (FILEHANDLE *from, FILEHANDLE *to)
- WORD WriteStoreHeader (WORD handle)
- WORD ReadSaveHeader ()
- WORD ReadSaveIndex (FILEINDEX *fileind)
- WORD ReadSaveVariables (UBYTE *buffer, UBYTE *top, LONG *size, LONG *outsize, INDEXENTRY *ind, LONG *stage)
- UBYTE * ReadSaveTerm32 (UBYTE *bin, UBYTE *binend, UBYTE **bout, UBYTE *boutend, UBYTE *top, int terminbuf)
- WORD ReadSaveExpression (UBYTE *buffer, UBYTE *top, LONG *size, LONG *outsize)

4.56.1 Detailed Description

Contains all functions that deal with store-files and the system independent save-files.

4.56.2 Function Documentation

4.56.2.1 SetFileIndex()

```
WORD SetFileIndex ( )
```

Reads the next file index and puts it into AR.StoreData.Index. TODO

Returns

= 0 everything okay, != 0 an error occurred

Definition at line 2300 of file store.c.

4.56.2.2 TermRenumber()

```
WORD TermRenumber (
            WORD * term,
            RENUMBER renumber,
            WORD nexpr )
```

!! WORD *memterm=term; static LONG ctrap=0; !!!

!! ctrap++; !!!

Definition at line 2407 of file store.c.

4.56.2.3 WriteStoreHeader()

```
WORD WriteStoreHeader (
            WORD handle )
```

Writes header with information about system architecture and FORM revision to an open store file.

Called by SetFileIndex().

Parameters

handle	specifies open file to which header will be written

Returns

= 0 everything okay, != 0 an error occurred

Definition at line 3926 of file store.c.

4.56.2.4 ReadSaveHeader()

```
WORD ReadSaveHeader ( )
```

Reads the header in the save file and sets function pointers and flags according to the information found there. Must be called before any other ReadSave... function.

Currently works only for the exchange between 32bit and 64bit machines (WORD size must be 2 or 4 bytes)!

It is called by CoLoad().

Returns

= 0 everything okay, != 0 an error occurred

Definition at line 4019 of file store.c.

4.56.2.5 ReadSaveIndex()

```
WORD ReadSaveIndex (
            FILEINDEX * fileind )
```

Reads a FILEINDEX from the open save file specified by AO.SaveData.Handle. Translations for adjusting endianness and data sizes are done if necessary.

Depends on the assumption that sizeof(FILEINDEX) is the same everywhere. If FILEINDEX or INDEXENTRY change, then this functions has to be adjusted.

Called by CoLoad() and FindInIndex().

Parameters

fileind	contains the read FILEINDEX after succesful return. must point to allocated, big enough memory.

Returns

= 0 everything okay, != 0 an error occurred

Definition at line 4137 of file store.c.

4.56.2.6 ReadSaveVariables()

```
WORD ReadSaveVariables (
            UBYTE * buffer,
            UBYTE * top,
            LONG * size,
            LONG * outsize,
            INDEXENTRY * ind,
            LONG * stage )
```

Reads the variables from the open file specified by AO.SaveData.Handle. It reads the ∗size bytes and writes them to the ∗buffer. It is called by PutInStore().

If translation is necessary, the data might shrink or grow in size, then ∗*size* is adjusted so that the reading and writing fits into the memory from the buffer to the top. The actual number of read bytes is returned in ∗*size*, the number of written bytes is returned in ∗*outsize*.

If the ∗size is smaller than the actual size of the variables, this function will be called several times and needs to remember the current position in the variable structure. The parameter *stage* does this job. When ReadSaveVariables() is called for the first time, this parameter should have the value -1.

The parameter *ind* is used to get the number of variables.

Parameters

buffer	read variables are written into this allocated memory
top	upper end of allocated memory
size	number of bytes to read. might return a smaller number of read bytes if translation was necessary
outsize	if translation has be done, outsize contains the number of written bytes
ind	pointer of INDEXENTRY for the current expression. read-only
stage	should be -1 for the first call, will be increased by ReadSaveVariables to memorize the position in the variable structure

Returns

= 0 everything okay, != 0 an error occurred

Definition at line 4323 of file store.c.

4.56.2.7 ReadSaveTerm32()

```
UBYTE* ReadSaveTerm32 (
            UBYTE * bin,
            UBYTE * binend,
            UBYTE ** bout,
            UBYTE * boutend,
            UBYTE * top,
            int terminbuf )
```

Reads a single term from the given buffer at *bin* and write the translated term back to this buffer at *bout*.

ReadSaveTerm32() is currently the only instantiation of a ReadSaveTerm-function. It only deals with data that already has the correct endianness and that is resized to 32bit words but without being renumbered or translated in any other way. It uses the compress buffer AR.CompressBuffer.

The function is reentrant in order to cope with nested function arguments. It is called by ReadSaveExpression() and itself.

The *return value* indicates the position in the input buffer up to which the data has already been successfully processed. The parameter *bout* returns the corresponding position in the output buffer.

Parameters

bin	start of the input buffer
binend	end of the input buffer
bout	as input points to the beginning of the output buffer, as output points behind the already translated data in the output buffer
boutend	end of already decompressed data in output buffer
top	end of output buffer
terminbuf	flag whether decompressed data is already in the output buffer. used in recursive calls

Returns

pointer to the next unprocessed data in the input buffer

Definition at line 4683 of file store.c.

4.56.2.8 ReadSaveExpression()

```
WORD ReadSaveExpression (
            UBYTE * buffer,
            UBYTE * top,
            LONG * size,
            LONG * outsize )
```

Reads an expression from the open file specified by AO.SaveData.Handle. The endianness flip and a resizing without renumbering is done in this function. Thereafter the buffer consists of chunks with a uniform maximal word size (32bit at the moment). The actual renumbering is then done by calling the function ReadSaveTerm32(). The result is returned in *buffer*.

If the translation at some point doesn't fit into the buffer anymore, the function returns and must be called again. In any case *size* returns the number of successfully read bytes, *outsize* returns the number of successfully written bytes, and the file will be positioned at the next byte after the successfully read data.

It is called by PutInStore().

Parameters

buffer	output buffer, holds the (translated) expression
top	end of buffer
size	number of read bytes
outsize	number of written bytes

Returns

= 0 everything okay, != 0 an error occurred

Definition at line 5092 of file store.c.

4.57 structs.h File Reference

Data Structures

- struct PoSiTiOn
- struct STOREHEADER
- struct InDeXeNtRy
- struct FiLeInDeX
- struct FiLeDaTa
- struct VaRrEnUm
- struct ReNuMbEr
- struct LIST
- struct KEYWORD
- struct KEYWORDV
- struct NaMeNode
- struct NaMeTree
- struct tree
- struct MiNmAx
- struct BrAcKeTiNdEx
- struct BrAcKeTiNfO
- struct TaBlEs
- struct ExPrEsSiOn
- struct SyMbOl
- struct InDeX
- struct VeCtOr
- struct FuNcTiOn
- struct SeTs
- struct DuBiOuS
- struct FaCdOlLaR
- struct DoLlArS
- struct MoDoPtDoLlArS
- struct fixedset
- struct TaBlEbAsEsUbInDeX
- struct TaBlEbAsE
- struct FUN_INFO
- struct FiLe
- struct StreaM
- struct SpecTatoR
- struct TrAcEs
- struct TrAcEn
- struct pReVaR
- struct INSIDEINFO
- struct PRELOAD
- struct PROCEDURE
- struct DoLoOp
- struct bit_field
- struct HANDLERS

- struct CbUf
- struct ChAnNeL
- struct SETUPPARAMETERS
- struct NeStInG
- struct StOrEcAcHe
- struct PeRmUtE
- struct PeRmUtEp
- struct DiStRiBuTe
- struct PaRtI
- struct sOrT
- struct POLYMOD
- struct SHvariables
- struct COST
- struct MODNUM
- struct OPTIMIZE
- struct OPTIMIZERESULT
- struct DICTIONARY_ELEMENT
- struct DICTIONARY
- struct SWITCHTABLE
- struct SWITCH
- struct M_const
- struct P_const
- struct C_const
- struct S_const
- struct R_const
- struct T_const
- struct N_const
- struct O_const
- struct X_const
- struct AllGlobals
- struct FixedGlobals

Macros

- #define INFILEINDEX ((512-2∗sizeof(POSITION))/sizeof(INDEXENTRY))
- #define EMPTYININDEX (512-2∗sizeof(POSITION)-INFILEINDEX∗sizeof(INDEXENTRY))
- #define **PHEAD**
- #define **PHEAD0** VOID
- #define **BHEAD**
- #define **BHEAD0**

Typedefs

- typedef struct PoSiTiOn **POSITION**
- typedef struct InDeXeNtRy INDEXENTRY
- typedef struct FiLeInDeX FILEINDEX
- typedef struct FiLeDaTa **FILEDATA**
- typedef struct VaRrEnUm VARRENUM
- typedef struct ReNuMbEr ∗ RENUMBER
- typedef struct NaMeNode NAMENODE
- typedef struct NaMeTree NAMETREE
- typedef struct tree COMPTREE
- typedef struct MiNmAx **MINMAX**

- typedef struct BrAcKeTiNdEx **BRACKETINDEX**
- typedef struct BrAcKeTiNfO **BRACKETINFO**
- typedef struct TaBlEs * TABLES
- typedef struct ExPrEsSiOn * **EXPRESSIONS**
- typedef struct SyMbOl * **SYMBOLS**
- typedef struct InDeX * **INDICES**
- typedef struct VeCtOr * **VECTORS**
- typedef struct FuNcTiOn * FUNCTIONS
- typedef struct SeTs * **SETS**
- typedef struct DuBiOuS * **DUBIOUSV**
- typedef struct FaCdOlLaR **FACDOLLAR**
- typedef struct DoLlArS * **DOLLARS**
- typedef struct MoDoPtDoLlArS **MODOPTDOLLAR**
- typedef struct fixedset **FIXEDSET**
- typedef struct TaBlEbAsEsUbInDeX **TABLEBASESUBINDEX**
- typedef struct TaBlEbAsE **TABLEBASE**
- typedef struct FiLe FILEHANDLE
- typedef struct StreaM STREAM
- typedef struct SpecTatoR **SPECTATOR**
- typedef struct TrAcEs TRACES
- typedef struct TrAcEn * TRACEN
- typedef struct pReVaR PREVAR
- typedef struct DoLoOp DOLOOP
- typedef struct bit_field set_of_char[32]
- typedef struct bit_field * one_byte
- typedef WORD(* **FINISHUFFLE**) (WORD *)
- typedef WORD(* **DO_UFFLE**) (WORD *, WORD, WORD, WORD)
- typedef WORD(* **COMPARE**) (WORD *, WORD *, WORD)
- typedef struct CbUf CBUF
- typedef struct ChAnNeL CHANNEL
- typedef struct NeStInG * NESTING
- typedef struct StOrEcAcHe * STORECACHE
- typedef struct PeRmUtE PERM
- typedef struct PeRmUtEp PERMP
- typedef struct DiStRiBuTe DISTRIBUTE
- typedef struct PaRtI PARTI
- typedef struct sOrT SORTING
- typedef struct AllGlobals ALLGLOBALS
- typedef WORD(* **WCN**) ()
- typedef WORD(* **WCN2**) ()
- typedef struct FixedGlobals FIXEDGLOBALS

Functions

- **STATIC_ASSERT** (sizeof(STOREHEADER)==512)
- **STATIC_ASSERT** (sizeof(FILEINDEX)==512)

4.57.1 Detailed Description

Contains definitions for global structs.

!!!CAUTION!!! Changes in this file will most likely have consequences for the recovery mechanism (see checkpoint.c). You need to care for the code in checkpoint.c as well and modify the code there accordingly!

The marker [D] is used in comments in this file to mark pointers to which dynamically allocated memory is assigned by a call to malloc() during runtime (in contrast to pointers that point into already allocated memory). This information is especially helpful if one needs to know which pointers need to be freed (cf. checkpoint.c).

4.57.2 Macro Definition Documentation

4.57.2.1 INFILEINDEX

```
#define INFILEINDEX ((512-2*sizeof(POSITION))/sizeof(INDEXENTRY))
```

Maximum number of entries (struct InDeXeNtRy) in a file index (struct FiLeInDeX). Number is calculated such that the size of a file index is no more than 512 bytes.

Definition at line 118 of file structs.h.

4.57.2.2 EMPTYININDEX

```
#define EMPTYININDEX (512-2*sizeof(POSITION)-INFILEINDEX*sizeof(INDEXENTRY))
```

Number of empty filling bytes for a file index (struct FiLeInDeX). It is calculated such that the size of a file index is always 512 bytes.

Definition at line 123 of file structs.h.

4.57.3 Typedef Documentation

4.57.3.1 INDEXENTRY

```
typedef struct InDeXeNtRy INDEXENTRY
```

Defines the structure of an entry in a file index (see struct FiLeInDeX).

It represents one expression in the file.

4.57.3.2 FILEINDEX

```
typedef struct FiLeInDeX FILEINDEX
```

Defines the structure of a file index in store-files and save-files.

It contains several entries (see struct InDeXeNtRy) up to a maximum of INFILEINDEX.

The variable number has been made of type POSITION to avoid padding problems with some types of computers/↩ OS and keep system independence of the .sav files.

This struct is always 512 bytes long.

4.57.3.3 VARRENUM

```
typedef struct VaRrEnUm VARRENUM
```

Contains the pointers to an array in which a binary search will be performed.

4.57.3.4 RENUMBER

```
typedef struct ReNuMbEr * RENUMBER
```

Only symb.lo gets dynamically allocated. All other pointers points into this memory.

4.57.3.5 NAMENODE

```
typedef struct NaMeNode NAMENODE
```

The names of variables are kept in an array. Elements of type NAMENODE define a tree (that is kept balanced) that make it easy and fast to look for variables. See also NAMETREE.

4.57.3.6 NAMETREE

```
typedef struct NaMeTree NAMETREE
```

A struct of type NAMETREE controls a complete (balanced) tree of names for the compiler. The compiler maintains several of such trees and the system has been set up in such a way that one could define more of them if we ever want to work with local name spaces.

4.57.3.7 COMPTREE

```
typedef struct tree COMPTREE
```

The subexpressions in the compiler are kept track of in a (balanced) tree to reduce the need for subexpressions and hence save much space in large rhs expressions (like when we have xxxxxxx occurrences of objects like f(x+1,x+1) in which each x+1 becomes a subexpression. The struct that controls this tree is COMPTREE.

4.57.3.8 TABLES

```
typedef struct TaBlEs * TABLES
```

buffers, mm, flags, and prototype are always dynamically allocated, tablepointers only if needed (=0 if unallocated), boomlijst and argtail only for sparse tables.

Allocation is done for both the normal and the stub instance (spare), except for prototype and argtail which share memory.

4.57.3.9 FUNCTIONS

```
typedef struct FuNcTiOn * FUNCTIONS
```

Contains all information about a function. Also used for tables. It is used in the LIST elements of #AC.

4.57.3.10 FILEHANDLE

```
typedef struct FiLe FILEHANDLE
```

The type FILEHANDLE is the struct that controls all relevant information of a file, whether it is open or not. The file may even not yet exist. There is a system of caches (POBuffer) and as long as the information to be written still fits inside the cache the file may never be created. There are variables that can store information about different types of files, like scratch files or sort files. Depending on what is available in the system we may also have information about gzip compression (currently sort file only) or locks (TFORM).

4.57.3.11 STREAM

```
typedef struct StreaM STREAM
```

Input is read from 'streams' which are represented by objects of type STREAM. A stream can be a file, a do-loop, a procedure, the string value of a preprocessor variable ..... When a new stream is opened we have to keep information about where to fall back in the parent stream to allow this to happen even in the middle of reading names etc as would be the case with a`i'b

4.57.3.12 TRACES

```
typedef struct TrAcEs TRACES
```

The struct TRACES keeps track of the progress during the expansion of a 4-dimensional trace. Each time a term gets generated the expansion tree continues in the next statement. When it returns it has to know where to continue. The 4-dimensional traces are more complicated than the n-dimensional traces (see TRACEN) because of the extra tricks that can be used. They are responsible for the shorter final expressions.

4.57.3.13 TRACEN

```
typedef struct TrAcEn * TRACEN
```

The struct TRACEN keeps track of the progress during the expansion of a 4-dimensional trace. Each time a term gets generated the expansion tree continues in the next statement. When it returns it has to know where to continue.

4.57.3.14 PREVAR

```
typedef struct pReVaR PREVAR
```

An element of the type PREVAR is needed for each preprocessor variable.

4.57.3.15 DOLOOP

```
typedef struct DoLoOp DOLOOP
```

Each preprocessor do loop has a struct of type DOLOOP to keep track of all relevant parameters like where the beginning of the loop is, what the boundaries, increment and value of the loop parameter are, etc. Also we keep the whole loop inside a buffer of type PRELOAD

4.57.3.16 set_of_char

```
typedef struct bit_field set_of_char[32]
```

Used in set_in, set_set, set_del and set_sub.

Definition at line 144 of file structs.h.

4.57.3.17 one_byte

```
typedef struct bit_field* one_byte
```

Used in set_in, set_set, set_del and set_sub.

Definition at line 909 of file structs.h.

4.57.3.18 CBUF

```
typedef struct CbUf CBUF
```

The CBUF struct is used by the compiler. It is a compiler buffer of which since version 3.0 there can be many.

4.57.3.19 CHANNEL

```
typedef struct ChAnNeL CHANNEL
```

When we read input from text files we have to remember not only their handle but also their name. This is needed for error messages. Hence we call such a file a channel and reserve a struct of type CHANNEL to allow to lay this link.

4.57.3.20 NESTING

```
typedef struct NeStInG * NESTING
```

The NESTING struct is used when we enter the argument of functions and there is the possibility that we have to change something there. Because functions can be nested we have to keep track of all levels of functions in case we have to move the outer layers to make room for a larger function argument.

4.57.3.21 STORECACHE

```
typedef struct StOrEcAcHe * STORECACHE
```

The struct of type STORECACHE is used by a caching system for reading terms from stored expressions. Each thread should have its own system of caches.

4.57.3.22 PERM

```
typedef struct PeRmUtE PERM
```

The struct PERM is used to generate all permutations when the pattern matcher has to try to match (anti)symmetric functions.

4.57.3.23 PERMP

```
typedef struct PeRmUtEp PERMP
```

Like struct PERM but works with pointers.

4.57.3.24 DISTRIBUTE

```
typedef struct DiStRiBuTe DISTRIBUTE
```

The struct DISTRIBUTE is used to help the pattern matcher when matching antisymmetric tensors.

4.57.3.25 PARTI

```
typedef struct PaRtI PARTI
```

The struct PARTI is used to help determining whether a partition_ function can be replaced.

4.57.3.26 SORTING

```
typedef struct sOrT SORTING
```

The struct SORTING is used to control a sort operation. It includes a small and a large buffer and arrays for keeping track of various stages of the (merge) sorts. Each sort level has its own struct and different levels can have different sizes for its arrays. Also different threads have their own set of SORTING structs.

4.57.3.27 ALLGLOBALS

```
typedef struct AllGlobals ALLGLOBALS
```

Without pthreads (FORM) the ALLGLOBALS struct has all the global variables

4.57.3.28 FIXEDGLOBALS

```
typedef struct FixedGlobals FIXEDGLOBALS
```

The FIXEDGLOBALS struct is an anachronism. It started as the struct with global variables that needed initialization. It contains the elements Operation and OperaFind which define a very early way of automatically jumping to the proper operation. We find the results of it in parts of the file opera.c Later operations were treated differently in a more transparent way. We never changed the existing code. The most important part is currently the cTable which is used intensively in the compiler.

4.58 symmetr.c File Reference

```
#include "form3.h"
```

Functions

- WORD **MatchE** (PHEAD WORD ∗pattern, WORD ∗fun, WORD ∗inter, WORD par)
- WORD **Permute** (PERM ∗perm, WORD first)
- WORD **PermuteP** (PERMP ∗perm, WORD first)
- WORD **Distribute** (DISTRIBUTE ∗d, WORD first)
- int **MatchCy** (PHEAD WORD ∗pattern, WORD ∗fun, WORD ∗inter, WORD par)
- int **FunMatchCy** (PHEAD WORD ∗pattern, WORD ∗fun, WORD ∗inter, WORD par)
- int **FunMatchSy** (PHEAD WORD ∗pattern, WORD ∗fun, WORD ∗inter, WORD par)
- int **MatchArgument** (PHEAD WORD ∗arg, WORD ∗pat)

4.58.1 Detailed Description

The routines that deal with the pattern matching of functions with symmetric properties.

4.59 tables.c File Reference

```
#include "form3.h"
#include "minos.h"
```

Functions

- void **ClearTableTree** (TABLES T)
- int **InsTableTree** (TABLES T, WORD *tp)
- void **RedoTableTree** (TABLES T, int newsize)
- int **FindTableTree** (TABLES T, WORD *tp, int inc)
- WORD **DoTableExpansion** (WORD *term, WORD level)
- int **CoTableBase** (UBYTE *s)
- int **FlipTable** (FUNCTIONS f, int type)
- int **SpareTable** (TABLES TT)
- DBASE * **FindTB** (UBYTE *name)
- int **CoTBcreate** (UBYTE *s)
- int **CoTBopen** (UBYTE *s)
- int **CoTBaddto** (UBYTE *s)
- int **CoTBenter** (UBYTE *s)
- int **CoTestUse** (UBYTE *s)
- int **CheckTableDeclarations** (DBASE *d)
- int **CoTBload** (UBYTE *ss)
- WORD **TestUse** (WORD *term, WORD level)
- int **CoTBaudit** (UBYTE *s)
- int **CoTBon** (UBYTE *s)
- int **CoTBoff** (UBYTE *s)
- int **CoTBcleanup** (UBYTE *s)
- int **CoTBreplace** (UBYTE *s)
- int **CoTBuse** (UBYTE *s)
- int **CoApply** (UBYTE *s)
- int **CoTBhelp** (UBYTE *s)
- VOID **ReWorkT** (WORD *term, WORD *funs, WORD numfuns)
- WORD **Apply** (WORD *term, WORD level)
- int **ApplyExec** (WORD *term, int maxtogo, WORD level)
- WORD **ApplyReset** (WORD level)
- WORD **TableReset** ()
- int **ReleaseTB** ()

Variables

- char * **helptb** [ ]

4.59.1 Detailed Description

Contains all functions that deal with the table bases on the 'FORM level' The low level databse routines are in minos.c

4.60 threads.c File Reference

4.60.1 Detailed Description

Routines for the interface of FORM with the pthreads library

This is the main part of the parallelization of TFORM. It is important to also look in the files structs.h and variable.h because the treatment of the A and B structs is essential (these structs are used by means of the macros AM, AP, AC, AS, AR, AT, AN, AO and AX). Also the definitions and use of the macros BHEAD and PHEAD should be looked up.

The sources are set up in such a way that if WITHPTHREADS isn't defined there is no trace of pthread parallelization. The reason is that TFORM is far more memory hungry than sequential FORM.

Special attention should also go to the locks. The proper use of the locks is essential and determines whether TFORM can work at all. We use the LOCK/UNLOCK macros which are empty in the case of sequential FORM These locks are at many places in the source files when workers can interfere with each others data or with the data of the master.

4.61 token.c File Reference

```
#include "form3.h"
```

Macros

- #define **CHECKPOLY** {if(polyflag)MesPrint("&Illegal use of polynomial function"); polyflag = 0; }

Functions

- int **tokenize** (UBYTE ∗in, WORD leftright)
- void **WriteTokens** (SBYTE ∗in)
- int **simp1token** (SBYTE ∗s)
- int **simpwtoken** (SBYTE ∗s)
- int **simp2token** (SBYTE ∗s)
- int **simp3atoken** (SBYTE ∗s, int mode)
- int **simp3btoken** (SBYTE ∗s, int mode)
- int **simp4token** (SBYTE ∗s)
- int **simp5token** (SBYTE ∗s, int mode)
- int **simp6token** (SBYTE ∗tokens, int mode)

Variables

- char ∗ **ttypes** [ ]

4.61.1 Detailed Description

The tokenizer. This is a part of the compiler that does an intermediate type of translation. It does look up the names etc and can do a number of optimizations. The resulting output is a stream of bytes which can be processed by the code generator (in the file compiler.c)

4.61.2 Variable Documentation

4.61.2.1 ttypes

```
char* ttypes[]
```

Initial value:

```
= { "\n", "S", "I", "V", "F", "set", "E", "dotp", "#",
   "sub", "d_", "$", "dub", "(", ")", "?", "??", ".", "[", "]",
   ",", "((", "))", "*", "/", "^", "+", "-", "!", "end", "{{", "}}",
   "N_?", "conj", "()", "#d", "^d", "_", "snum"  }
```

Definition at line 587 of file token.c.

4.62 tools.c File Reference

```
#include "form3.h"
#include <sys/timeb.h>
```

Macros

- #define **FILLVALUE** 0
- #define **COPYFILEBUFSIZE** 40960L
- #define TERMMEMSTARTNUM 16
- #define **TERMEXTRAWORDS** 10
- #define NUMBERMEMSTARTNUM 16
- #define **NUMBEREXTRAWORDS** 10L
- #define **DODOUBLE**(x)
- #define **DOEXPAND**(x)

Functions

- UBYTE * **LoadInputFile** (UBYTE *filename, int type)
- UBYTE **ReadFromStream** (STREAM *stream)
- UBYTE **GetFromStream** (STREAM *stream)
- UBYTE **LookInStream** (STREAM *stream)
- STREAM * **OpenStream** (UBYTE *name, int type, int prevarmode, int raiselow)
- int **LocateFile** (UBYTE **name, int type)
- STREAM * **CloseStream** (STREAM *stream)
- STREAM * **CreateStream** (UBYTE *where)
- LONG **GetStreamPosition** (STREAM *stream)
- VOID **PositionStream** (STREAM *stream, LONG position)
- int **ReverseStatements** (STREAM *stream)
- VOID **StartFiles** ()
- int **OpenFile** (char *name)
- int **OpenAddFile** (char *name)
- int **ReOpenFile** (char *name)
- int **CreateFile** (char *name)

- int **CreateLogFile** (char ∗name)
- VOID **CloseFile** (int handle)
- int CopyFile (char ∗source, char ∗dest)
- int **CreateHandle** ()
- LONG **ReadFile** (int handle, UBYTE ∗buffer, LONG size)
- LONG **ReadPosFile** (PHEAD FILEHANDLE ∗fi, UBYTE ∗buffer, LONG size, POSITION ∗pos)
- LONG **WriteFileToFile** (int handle, UBYTE ∗buffer, LONG size)
- VOID **SeekFile** (int handle, POSITION ∗offset, int origin)
- LONG **TellFile** (int handle)
- VOID **TELLFILE** (int handle, POSITION ∗position)
- void **FlushFile** (int handle)
- int **GetPosFile** (int handle, fpos_t ∗pospointer)
- int **SetPosFile** (int handle, fpos_t ∗pospointer)
- VOID **SynchFile** (int handle)
- VOID **TruncateFile** (int handle)
- int **GetChannel** (char ∗name, int mode)
- int **GetAppendChannel** (char ∗name)
- int **CloseChannel** (char ∗name)
- void **UpdateMaxSize** ()
- int **StrCmp** (UBYTE ∗s1, UBYTE ∗s2)
- int **StrICmp** (UBYTE ∗s1, UBYTE ∗s2)
- int **StrHICmp** (UBYTE ∗s1, UBYTE ∗s2)
- int **StrICont** (UBYTE ∗s1, UBYTE ∗s2)
- int **CmpArray** (WORD ∗t1, WORD ∗t2, WORD n)
- int **ConWord** (UBYTE ∗s1, UBYTE ∗s2)
- int **StrLen** (UBYTE ∗s)
- VOID **NumToStr** (UBYTE ∗s, LONG x)
- VOID **WriteString** (int type, UBYTE ∗str, int num)
- VOID **WriteUnfinString** (int type, UBYTE ∗str, int num)
- UBYTE ∗ **AddToString** (UBYTE ∗outstring, UBYTE ∗extrastring, int par)
- UBYTE ∗ **strDup1** (UBYTE ∗instring, char ∗ifwrong)
- UBYTE ∗ **EndOfToken** (UBYTE ∗s)
- UBYTE ∗ **ToToken** (UBYTE ∗s)
- UBYTE ∗ **SkipField** (UBYTE ∗s, int level)
- WORD **ReadSnum** (UBYTE ∗∗p)
- UBYTE ∗ **NumCopy** (WORD y, UBYTE ∗to)
- char ∗ **LongCopy** (LONG y, char ∗to)
- char ∗ **LongLongCopy** (off_t ∗y, char ∗to)
- UBYTE ∗ **MakeDate** ()
- int **set_in** (UBYTE ch, set_of_char set)
- one_byte **set_set** (UBYTE ch, set_of_char set)
- one_byte **set_del** (UBYTE ch, set_of_char set)
- one_byte **set_sub** (set_of_char set, set_of_char set1, set_of_char set2)
- VOID **iniTools** (VOID)
- VOID ∗ **Malloc** (LONG size)
- VOID ∗ **Malloc1** (LONG size, const char ∗messageifwrong)
- void **M_free** (VOID ∗x, const char ∗where)
- void **M_check1** ()
- void **M_print** ()
- VOID **TermMallocAddMemory** (PHEAD0)
- VOID **NumberMallocAddMemory** (PHEAD0)
- VOID **CacheNumberMallocAddMemory** (PHEAD0)
- VOID ∗ **FromList** (LIST ∗L)
- VOID ∗ **From0List** (LIST ∗L)
- VOID ∗ **FromVarList** (LIST ∗L)

- int **DoubleList** (VOID ∗∗∗lijst, int ∗oldsize, int objectsize, char ∗nameoftype)
- int **DoubleLList** (VOID ∗∗∗lijst, LONG ∗oldsize, int objectsize, char ∗nameoftype)
- void **DoubleBuffer** (void ∗∗start, void ∗∗stop, int size, char ∗text)
- void **ExpandBuffer** (void ∗∗buffer, LONG ∗oldsize, int type)
- LONG **iexp** (LONG x, int p)
- void **ToGeneral** (WORD ∗r, WORD ∗m, WORD par)
- int **ToFast** (WORD ∗r, WORD ∗m)
- WORD **ToPolyFunGeneral** (PHEAD WORD ∗term)
- int **IsLikeVector** (WORD ∗arg)
- int **AreArgsEqual** (WORD ∗arg1, WORD ∗arg2)
- int **CompareArgs** (WORD ∗arg1, WORD ∗arg2)
- int **CompArg** (WORD ∗s1, WORD ∗s2)
- LONG TimeWallClock (WORD par)
- LONG **TimeChildren** (WORD par)
- LONG TimeCPU (WORD par)
- int **Crash** ()
- int TestTerm (WORD ∗term)

Variables

- FILES ∗∗ **filelist**
- int **numinfilelist** = 0
- int **filelistsize** = 0
- WRITEFILE **WriteFile** = &WriteFileToFile

4.62.1 Detailed Description

Low level routines for many types of task. There are routines for manipulating the input system (streams and files) routines for string manipulation, the memory allocation interface, and the clock. The last is the most sensitive to ports. In the past nearly every port to another OS or computer gave trouble. Nowadays it is slightly better but the poor POSIX compliance of LINUX again gave problems for the multithreaded version.

4.62.2 Macro Definition Documentation

4.62.2.1 TERMMEMSTARTNUM

```
#define TERMMEMSTARTNUM 16
```

Provides memory for one term (or one small polynomial) This means that the memory is limited to a buffer of size AM.MaxTer plus a few extra words. In parallel versions, each worker has its own memory pool.

The way we use the memory is by: term = TermMalloc(BHEAD0); and later we free it by TermFree(BHEAD term);

Layout: We have a list of available pointers to buffers: AT.TermMemHeap Its size is AT.TermMemMax We take from the top (indicated by AT.TermMemTop). When we run out of buffers we assign new ones (doubling the amount) and we have to extend the AT.TermMemHeap array. Important: There is no checking that the returned memory is legal, ie is memory that was handed out earlier.

Definition at line 2474 of file tools.c.

4.62.2.2 NUMBERMEMSTARTNUM

```
#define NUMBERMEMSTARTNUM 16
```

Provides memory for one Long number This means that the memory is limited to a buffer of size AM.MaxTal In parallel versions, each worker has its own memory pool.

The way we use the memory is by: num = NumberMalloc(BHEAD0); Number = AT.NumberMemHeap[num]; and later we free it by NumberFree(BHEAD num);

Layout: We have a list of available pointers to buffers: AT.NumberMemHeap Its size is AT.NumberMemMax We take from the top (indicated by AT.NumberMemTop). When we run out of buffers we assign new ones (doubling the amount) and we have to extend the AT.NumberMemHeap array. Important: There is no checking on the returned memory!!!!

Definition at line 2574 of file tools.c.

4.62.2.3 DODOUBLE

```
#define DODOUBLE(
              x )
```

Value:

```
{ x *s, *t, *u; if ( *start ) { \
oldsize = *(x **)stop - *(x **)start; newsize = 2*oldsize; \
t = u = (x *)Malloc1(newsize*sizeof(x),text); s = *(x **)start; \
for ( i = 0; i < oldsize; i++ ) *t++ = *s++; M_free(*start,"double"); } \
else { newsize = 100; u = (x *)Malloc1(newsize*sizeof(x),text); } \
*start = (void *)u; *stop = (void *)(u+newsize); }
```

Definition at line 2897 of file tools.c.

4.62.2.4 DOEXPAND

```
#define DOEXPAND(
              x )
```

Value:

```
{ x *newbuffer, *t, *m;                                 \
t = newbuffer = (x *)Malloc1((newsize+2)*type,"ExpandBuffer");      \
if ( *buffer ) { m = (x *)*buffer; i = *oldsize;                    \
    while ( --i >= 0 ) *t++ = *m++; M_free(*buffer,"ExpandBuffer"); \
} *buffer = newbuffer; *oldsize = newsize; }
```

Definition at line 2923 of file tools.c.

4.62.3 Function Documentation

4.62.3.1 CopyFile()

```
int CopyFile (
             char * source,
             char * dest )
```

Copies a file with name ∗source to a file named ∗dest. The involved files must not be open. Returns non-zero if an error occurred. Uses if possible the combined large and small sorting buffers as cache.

Definition at line 1029 of file tools.c.

4.62.3.2 TimeWallClock()

```
LONG TimeWallClock (
             WORD par )
```

Returns the wall-clock time.

Parameters

par	If zero, the wall-clock time will be reset to 0.

Returns

The wall-clock time in centiseconds.

Definition at line 3404 of file tools.c.

Referenced by DoCheckpoint().

4.62.3.3 TimeCPU()

```
LONG TimeCPU (
             WORD par )
```

Returns the CPU time.

Parameters

par	If zero, the CPU time will be reset to 0.

Returns

The CPU time in milliseconds.

Definition at line 3478 of file tools.c.

4.62.3.4 TestTerm()

```
int TestTerm (
            WORD * term )
```

Tests the consistency of the term. Returns 0 when the term is OK. Any nonzero value is trouble. In the current version the testing isn't 100% complete. For instance, we don't check the validity of the symbols nor do we check the range of their powers. Etc. This should be extended when the need is there.

Parameters

term	the term to be tested

Definition at line 3789 of file tools.c.

4.63 topowrap.cc File Reference

```
#include "form3.h"
#include "gentopo.h"
```

Data Structures

- struct ToPoTyPe

Macros

- #define **MAXPOINTS** 120

Typedefs

- typedef struct ToPoTyPe **TOPOTYPE**

Functions

- int **GenerateVertices** (TOPOTYPE ∗TopoInf, int pointsremaining, int level)
- WORD **GenerateTopologies** (PHEAD WORD nloops, WORD nlegs, WORD setvert, WORD setmax)
- void **toForm** (EGraph ∗egraph)

4.63.1 Detailed Description

routines for conversion of topology and diagram output to FORM notation

4.64 transform.c File Reference

```
#include "form3.h"
```

Functions

- int **CoTransform** (UBYTE ∗in)
- WORD **RunTransform** (PHEAD WORD ∗term, WORD ∗params)
- WORD **RunEncode** (PHEAD WORD ∗fun, WORD ∗args, WORD ∗info)
- WORD **RunDecode** (PHEAD WORD ∗fun, WORD ∗args, WORD ∗info)
- WORD **RunReplace** (PHEAD WORD ∗fun, WORD ∗args, WORD ∗info)
- WORD **RunImplode** (WORD ∗fun, WORD ∗args)
- WORD **RunExplode** (PHEAD WORD ∗fun, WORD ∗args)
- WORD **RunPermute** (PHEAD WORD ∗fun, WORD ∗args, WORD ∗info)
- WORD **RunReverse** (PHEAD WORD ∗fun, WORD ∗args)
- WORD **RunDedup** (PHEAD WORD ∗fun, WORD ∗args)
- WORD **RunCycle** (PHEAD WORD ∗fun, WORD ∗args, WORD ∗info)
- WORD **RunAddArg** (PHEAD WORD ∗fun, WORD ∗args)
- WORD **RunMulArg** (PHEAD WORD ∗fun, WORD ∗args)
- WORD **RunIsLyndon** (PHEAD WORD ∗fun, WORD ∗args, int par)
- WORD **RunToLyndon** (PHEAD WORD ∗fun, WORD ∗args, int par)
- WORD **RunDropArg** (PHEAD WORD ∗fun, WORD ∗args)
- WORD **RunSelectArg** (PHEAD WORD ∗fun, WORD ∗args)
- int **TestArgNum** (int n, int totarg, WORD ∗args)
- WORD **PutArgInScratch** (WORD ∗arg, UWORD ∗scrat)
- UBYTE ∗ **ReadRange** (UBYTE ∗s, WORD ∗out, int par)
- int **FindRange** (PHEAD WORD ∗args, WORD ∗arg1, WORD ∗arg2, WORD totarg)

4.64.1 Detailed Description

Routines that deal with the transform statement and its varieties.

4.65 unix.h File Reference

Macros

- #define **LINEFEED** '\n'
- #define **CARRIAGERETURN** 0x0D
- #define **WITHPIPE**
- #define **WITHSYSTEM**
- #define **WITHEXTERNALCHANNEL**
- #define **TRAPSIGNALS**
- #define **P_term**(code) exit((int)(code<0?-code:code))
- #define **SEPARATOR** '/'
- #define **ALTSEPARATOR** '/'
- #define **PATHSEPARATOR** ':'
- #define **WITH_ENV**

4.65.1 Detailed Description

Settings for Unix-like systems.

4.66 unixfile.c File Reference

```
#include "form3.h"
```

4.66.1 Detailed Description

The interface to a fast variety of file routines in the unix system.

4.67 variable.h File Reference

Macros

- #define **chartype** FG.cTable
- #define **Procedures** ((PROCEDURE ∗)(AP.ProcList.lijst))
- #define **NumProcedures** AP.ProcList.num
- #define **MaxProcedures** AP.ProcList.maxnum
- #define **DoLoops** ((DOLOOP ∗)(AP.LoopList.lijst))
- #define **NumDoLoops** AP.LoopList.num
- #define **MaxDoLoops** AP.LoopList.maxnum
- #define **PreVar** ((PREVAR ∗)(AP.PreVarList.lijst))
- #define **NumPre** AP.PreVarList.num
- #define **MaxNumPre** AP.PreVarList.maxnum
- #define **SetElements** ((WORD ∗)(AC.SetElementList.lijst))
- #define **Sets** ((SETS)(AC.SetList.lijst))
- #define **functions** ((FUNCTIONS)(AC.FunctionList.lijst))
- #define **indices** ((INDICES)(AC.IndexList.lijst))
- #define **symbols** ((SYMBOLS)(AC.SymbolList.lijst))
- #define **vectors** ((VECTORS)(AC.VectorList.lijst))
- #define **tablebases** ((DBASE ∗)(AC.TableBaseList.lijst))
- #define **NumFunctions** AC.FunctionList.num
- #define **NumIndices** AC.IndexList.num
- #define **NumSymbols** AC.SymbolList.num
- #define **NumVectors** AC.VectorList.num
- #define **NumSets** AC.SetList.num
- #define **NumSetElements** AC.SetElementList.num
- #define **NumTableBases** AC.TableBaseList.num
- #define **GlobalFunctions** AC.FunctionList.numglobal
- #define **GlobalIndices** AC.IndexList.numglobal
- #define **GlobalSymbols** AC.SymbolList.numglobal
- #define **GlobalVectors** AC.VectorList.numglobal
- #define **GlobalSets** AC.SetList.numglobal
- #define **GlobalSetElements** AC.SetElementList.numglobal
- #define **cbuf** ((CBUF ∗)(AC.cbufList.lijst))

- #define **channels** ((CHANNEL ∗)(AC.ChannelList.lijst))
- #define **NumOutputChannels** AC.ChannelList.num
- #define **Dollars** ((DOLLARS)(AP.DollarList.lijst))
- #define **NumDollars** AP.DollarList.num
- #define **Dubious** ((DUBIOUSV)(AC.DubiousList.lijst))
- #define **NumDubious** AC.DubiousList.num
- #define **Expressions** ((EXPRESSIONS)(AC.ExpressionList.lijst))
- #define **NumExpressions** AC.ExpressionList.num
- #define **autofunctions** ((FUNCTIONS)(AC.AutoFunctionList.lijst))
- #define **autoindices** ((INDICES)(AC.AutoIndexList.lijst))
- #define **autosymbols** ((SYMBOLS)(AC.AutoSymbolList.lijst))
- #define **autovectors** ((VECTORS)(AC.AutoVectorList.lijst))
- #define **xsymbol** (cbuf[AM.sbufnum].rhs)
- #define **numxsymbol** (cbuf[AM.sbufnum].numrhs)
- #define **PotModdollars** ((WORD ∗)(AC.PotModDolList.lijst))
- #define **NumPotModdollars** AC.PotModDolList.num
- #define **ModOptdollars** ((MODOPTDOLLAR ∗)(AC.ModOptDolList.lijst))
- #define **NumModOptdollars** AC.ModOptDolList.num
- #define **BUG** A.bug;
- #define **AC** A.C
- #define **AM** A.M
- #define **AN** A.N
- #define **AO** A.O
- #define **AP** A.P
- #define **AR** A.R
- #define **AS** A.S
- #define **AT** A.T
- #define **AX** A.X

Variables

- WRITEBUFTOEXTCHANNEL **writeBufToExtChannel**
- GETCFROMEXTCHANNEL **getcFromExtChannel**
- SETTERMINATORFOREXTERNALCHANNEL **setTerminatorForExternalChannel**
- SETKILLMODEFOREXTERNALCHANNEL **setKillModeForExternalChannel**
- WRITEFILE **WriteFile**
- ALLGLOBALS **A**
- FIXEDGLOBALS **FG**
- FIXEDSET **fixedsets** [ ]
- char ∗ **setupfilename**

4.67.1 Detailed Description

Contains a number of defines to make the coding easier. Especially the defines for the use of the lists are very nice. And of course the AC for A.C and AT for either A.T of B->T are indispensible to keep FORM and TFORM in one set of sources.

4.68 vector.h File Reference

```
#include <stddef.h>
#include <string.h>
#include "declare.h"
```

Macros

- #define VectorStruct(T)
- #define Vector(T, X) VectorStruct(T) X = { NULL, 0, 0 }
- #define DeclareVector(T, X) VectorStruct(T) X
- #define VectorInit(X)
- #define VectorFree(X)
- #define VectorPtr(X) ((X).ptr)
- #define VectorFront(X) ((X).ptr[0])
- #define VectorBack(X) ((X).ptr[(X).size - 1])
- #define VectorSize(X) ((X).size)
- #define VectorCapacity(X) ((X).capacity)
- #define VectorEmpty(X) ((X).size == 0)
- #define VectorClear(X) do { (X).size = 0; } while (0)
- #define VectorReserve(X, newcapacity)
- #define VectorPushBack(X, x)
- #define VectorPushBacks(X, src, n)
- #define VectorPopBack(X) do { (X).size --; } while (0)
- #define VectorInsert(X, index, x)
- #define VectorInserts(X, index, src, n)
- #define VectorErase(X, index)
- #define VectorErases(X, index, n)

4.68.1 Detailed Description

An implementation of dynamic array.

Example:

```
size_t i;
Vector(int, vec);
VectorPushBack(vec, 1);
VectorPushBack(vec, 2);
VectorPushBack(vec, 3);
for ( i = 0; i < VectorSize(vec); i++ )
  printf("%d\n", VectorPtr(vec)[i]);
VectorFree(vec);
```

4.68.2 Macro Definition Documentation

4.68.2.1 VectorStruct

```
#define VectorStruct(
              T )
```

Value:

```
    struct { \
        T *ptr; \
        size_t size; \
        size_t capacity; \
    }
```

A struct for vector objects.

Parameters

T	the type of elements.

Definition at line 65 of file vector.h.

4.68.2.2 Vector

```
#define Vector(
              T,
              X )  VectorStruct(T) X = { NULL, 0, 0 }
```

Defines and initialises a vector X of the type T. The user must call VectorFree(X) after the use of X.

Parameters

T	the type of elements.
X	the name of vector object

Definition at line 84 of file vector.h.

4.68.2.3 DeclareVector

```
#define DeclareVector(
              T,
              X )  VectorStruct(T) X
```

Declares a vector X of the type T. The user must call VectorInit(X) before the use of X.

Parameters

T	the type of elements.
X	the name of vector object

Definition at line 99 of file vector.h.

4.68.2.4 VectorInit

```
#define VectorInit(
              X )
```

Value:

```
do { \
     (X).ptr = NULL; \
     (X).size = 0; \
     (X).capacity = 0; \
} while (0)
```

Initialises a vector X of the type T. The user must call VectorFree(X) after the use of X.

Parameters

X	the vector object.

Definition at line 113 of file vector.h.

4.68.2.5 VectorFree

```
#define VectorFree(
               X )
```

Value:

```
do { \
     M_free((X).ptr, "VectorFree:" #X); \
     (X).ptr = NULL; \
     (X).size = 0; \
     (X).capacity = 0; \
} while (0)
```

Frees the buffer allocated by the vector X.

Parameters

X	the vector object.

Definition at line 130 of file vector.h.

4.68.2.6 VectorPtr

```
#define VectorPtr(
               X )   ((X).ptr)
```

Returns the pointer to the buffer for the vector X. NULL when VectorCapacity(X) == 0.

Parameters

X	the vector object.

Returns

the pointer to the allocated buffer for the vector.

Definition at line 150 of file vector.h.

4.68.2.7 VectorFront

```
#define VectorFront(
              X )  ((X).ptr[0])
```

Returns the first element of the vector X. Undefined when VectorSize(X) == 0.

Parameters

X	the vector object.

Returns

the first element of the vector.

Definition at line 165 of file vector.h.

4.68.2.8 VectorBack

```
#define VectorBack(
              X )  ((X).ptr[(X).size - 1])
```

Returns the last element of the vector X. Undefined when VectorSize(X) == 0.

Parameters

X	the vector object.

Returns

the last element of the vector.

Definition at line 180 of file vector.h.

4.68.2.9 VectorSize

```
#define VectorSize(
              X )  ((X).size)
```

Returns the size of the vector X.

Parameters

X	the vector object.

Returns

the size of the vetor.

Definition at line 194 of file vector.h.

4.68.2.10 VectorCapacity

```
#define VectorCapacity(
              X )  ((X).capacity)
```

Returns the capacity (the number of the already allocated elements) of the vector X.

Parameters

X	the vector object.

Returns

the capacity of the vetor.

Definition at line 208 of file vector.h.

4.68.2.11 VectorEmpty

```
#define VectorEmpty(
              X )  ((X).size == 0)
```

Returns true the size of the vector X is zero.

Parameters

X	the vector object.

Returns

true if the vector has no elements, false otherwise.

Definition at line 222 of file vector.h.

4.68.2.12 VectorClear

```
#define VectorClear(
              X )  do { (X).size = 0; } while (0)
```

Sets the size of the vector X to zero.

Parameters

X	the vector object.

Definition at line 235 of file vector.h.

4.68.2.13 VectorReserve

```
#define VectorReserve(
              X,
              newcapacity )
```

Value:

```
    do { \
        size_t v_tmp_newcapacity_ = (newcapacity); \
        if ( (X).capacity < v_tmp_newcapacity_ ) { \
            void *v_tmp_newptr_; \
            v_tmp_newcapacity_ = (v_tmp_newcapacity_ * 3) / 2; \
            if ( v_tmp_newcapacity_ < 4 ) v_tmp_newcapacity_ = 4; \
            v_tmp_newptr_ = Malloc1(sizeof((X).ptr[0]) * v_tmp_newcapacity_, "VectorReserve:" #X); \
            if ( (X).ptr != NULL ) { \
                memcpy(v_tmp_newptr_, (X).ptr, (X).size * sizeof((X).ptr[0])); \
                M_free((X).ptr, "VectorReserve:" #X); \
            } \
            (X).ptr = v_tmp_newptr_; \
            (X).capacity = v_tmp_newcapacity_; \
        } \
    } while (0)
```

Requires that the capacity of the vector X is equal to or lager than newcapacity.

Parameters

X	the vector object.
newcapacity	the capacity to be reserved.

Definition at line 249 of file vector.h.

4.68.2.14 VectorPushBack

```
#define VectorPushBack(
              X,
              x )
```

Value:

```
    do { \
        VectorReserve((X), (X).size + 1); \
        (X).ptr[(X).size++] = (x); \
    } while (0)
```

Adds an element x at the end of the vector X.

Parameters

X	the vector object.
x	the element to be added.

Definition at line 277 of file vector.h.

4.68.2.15 VectorPushBacks

```
#define VectorPushBacks(
                X,
                src,
                n )
```

Value:

```
    do { \
        size_t v_tmp_n_ = (n); \
        VectorReserve((X), (X).size + v_tmp_n_); \
        memcpy((X).ptr + (X).size, (src), v_tmp_n_ * sizeof((X).ptr[0])); \
        (X).size += v_tmp_n_; \
    } while (0)
```

Adds an n elements of src at the end of the vector X.

Parameters

X	the vector object.
src	the starting address of the buffer storing elements to be added.
n	the number of elements to be added.

Definition at line 295 of file vector.h.

4.68.2.16 VectorPopBack

```
#define VectorPopBack(
                X )   do { (X).size --; } while (0)
```

Removes the last element of the vector X. VectorSize(X) must be > 0.

Parameters

X	the vector object.

Definition at line 314 of file vector.h.

4.68.2.17 VectorInsert

```
#define VectorInsert(
               X,
               index,
               x )
```

Value:

```
    do { \
        size_t v_tmp_index_ = (index); \
        VectorReserve((X), (X).size + 1); \
        memmove((X).ptr + v_tmp_index_ + 1, (X).ptr + v_tmp_index_, ((X).size - v_tmp_index_) *
       sizeof((X).ptr[0])); \
        (X).ptr[v_tmp_index_] = (x); \
        (X).size++; \
    } while (0)
```

Inserts an element x at the specified index of the vector X. The index must be 0 <= index < VectorSize(X).

Parameters

X	the vector object.
index	the position at which the element will be inserted.
x	the element to be inserted.

Definition at line 330 of file vector.h.

4.68.2.18 VectorInserts

```
#define VectorInserts(
               X,
               index,
               src,
               n )
```

Value:

```
    do { \
        size_t v_tmp_index_ = (index), v_tmp_n_ = (n); \
        VectorReserve((X), (X).size + v_tmp_n_); \
        memmove((X).ptr + v_tmp_index_ + v_tmp_n_, (X).ptr + v_tmp_index_, ((X).size - v_tmp_index_) *
       sizeof((X).ptr[0])); \
        memcpy((X).ptr + v_tmp_index_, (src), v_tmp_n_ * sizeof((X).ptr[0])); \
        (X).size += v_tmp_n_; \
    } while (0)
```

Inserts an n elements of src at the specified index of the vector X. The index must be 0 <= index < VectorSize(X).

Parameters

X	the vector object.
index	the position at which the elements will be inserted.
src	the starting address of the buffer storing elements to be inserted.
n	the number of elements to be inserted.

Definition at line 353 of file vector.h.

4.68.2.19 VectorErase

```
#define VectorErase(
                X,
                index )
```

Value:

```
    do { \
         size_t v_tmp_index_ = (index); \
         memmove((X).ptr + v_tmp_index_, (X).ptr + v_tmp_index_ + 1, ((X).size - v_tmp_index_ - 1) *
        sizeof((X).ptr[0])); \
         (X).size--; \
    } while (0)
```

Removes an element at the specified index of the vector X. The index must be 0 <= index < VectorSize(X).

Parameters

X	the vector object.
index	the position of the element to be removed.

Definition at line 374 of file vector.h.

4.68.2.20 VectorErases

```
#define VectorErases(
                X,
                index,
                n )
```

Value:

```
    do { \
         size_t v_tmp_index_ = (index), v_tmp_n_ = (n); \
         memmove((X).ptr + v_tmp_index_, (X).ptr + v_tmp_index_ + v_tmp_n_, ((X).size - v_tmp_index_ - 1) *
        sizeof((X).ptr[0])); \
         (X).size -= v_tmp_n_; \
    } while (0)
```

Removes an n elements at the specified index of the vector X. The index must be 0 <= index < VectorSize(X) - n + 1.

Parameters

X	the vector object.
index	the starting position of the elements to be removed.
n	the number of elements to be removed.

Definition at line 394 of file vector.h.

4.69 wildcard.c File Reference

```
#include "form3.h"
```

Macros

- #define **DEBUG**(x)

Functions

- WORD **WildFill** (PHEAD WORD ∗to, WORD ∗from, WORD ∗sub)
- WORD **ResolveSet** (PHEAD WORD ∗from, WORD ∗to, WORD ∗subs)
- VOID **ClearWild** (PHEAD0)
- WORD **AddWild** (PHEAD WORD oldnumber, WORD type, WORD newnumber)
- WORD **CheckWild** (PHEAD WORD oldnumber, WORD type, WORD newnumber, WORD ∗newval)
- int **DenToFunction** (WORD ∗term, WORD numfun)

4.69.1 Detailed Description

Contains the functions that deal with the wildcards. During the pattern matching there are two steps: 1: check that a wildcard substitution is correct (if there was already an assignment for this variable, it is the same; it is part of the proper set; it is the proper type of variables, etc.) 2: make the assignment In addition we have to be able to clear assignments. During execution we have to make the actual replacements (WildFill)

Index