How ML/I works

Revised First Edition

April 2004

P.J. Brown, R.C. Saunders, R.D. Eager

Table of Contents

Preface

These notes have been prepared to help implementors of ML/I understand how the logic works. Within these notes references are sometimes made to line numbers on a listing of the MI-logic of ML/I. These refer to version AIB dated December 1970. Listings are divided into two parts: declarations, which occupy about 270 lines, and code, which occupy about 2170 lines. Line numbers start again at one in the code part.

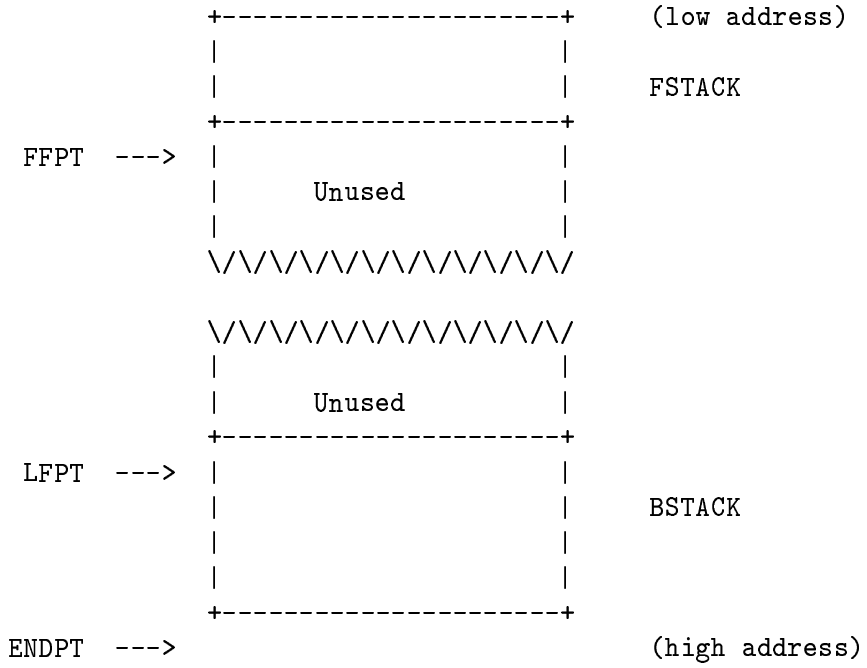
Periodically minor changes are made to the MI-logic, so line numbers are subject to slight change. However, no major changes have been made for the last three years.

Preface to the Revised First Edition

This Edition has been rewritten in Texinfo, so that it can be published in both printed and machine readable form; this has necessitated some re-wording and re-ordering of the text. Some minor corrections and clarifications have also been made, but the text still describes version AIB. The references have been updated to reflect the most up to date versions of relevant documents.

1 Introduction — the stacks

The two stacks occupy all available free storage. The forwards stack (FSTACK) and the backwards stack (BSTACK) are organised as follows:



FFPT points at the first free location on FSTACK, and LFPT points at the last used location on BSTACK. If the stacks meet, the process is aborted. In general, FSTACK contains permanent information and, at the top, ephemeral text. BSTACK contains temporary information and operates as a pushdown list (the word "top" in relation to FSTACK and BSTACK means the movable end, i.e. the one described by FFPT or LFPT respectively).

1.1 BSTACK

Information is added to BSTACK under the following circumstances:

a. When a macro call or insert is performed, a block of information (see Chapter 3 [Information stacked at calls or inserts], page 9) is stacked. When the evaluation of the call or insert is complete, the stack is reset to its state on entry.

b. Each of the local macros MCDEF, MCWARN, MCINS and MCSKIP causes a definition to be placed on BSTACK. If a local NEC macro is the first such to occur in the current text, then a new hash table, local to the current text, is reserved at the top of BSTACK.

c. Each placing of a label, other than in the source text, causes an entry to be made at the top of BSTACK. These entries have form:

```
+----------------------+
| pointer              |
+----------------------+
| number 1             |
+----------------------+
| number 2             |
+----------------------+
| number 3             |
+----------------------+
```

where:

`pointer` is a pointer to the next item in the chain of labels local to the current text. `NULLPT` means end of chain.

`number 1` is the label number.

`number 2` is the offset of the position of the label from the *end* of the text (i.e. from `STOPPT`).

`number 3` is the line number in which the label is placed.

d. During the scanning of a nested construction, a block of information describing the containing text is stacked.

e. The top of `BSTACK` is used as working storage during the setting up of construction definitions.

Note that:

- Entries of type c) and d) may be interspersed; they are automatically deleted at the end of a piece of text.
- Entries of type d) or e) only occur at the top of `BSTACK`.

1.1.1 Implementation of `MCNODEF`, etc.

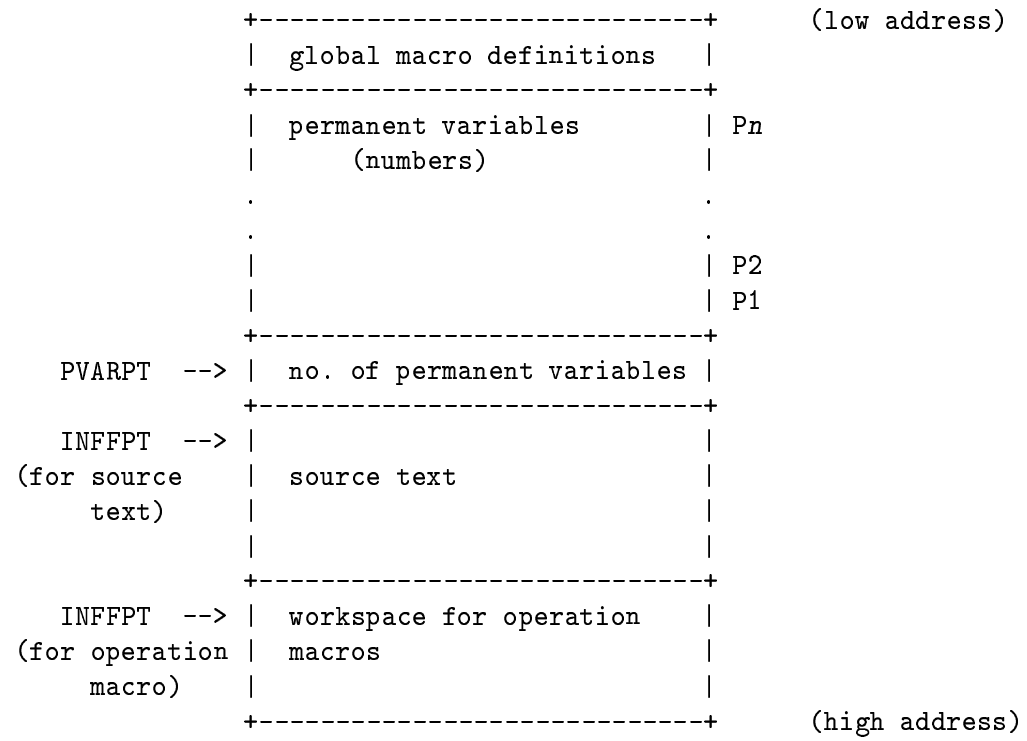
The hash table contains a slot which contains the maximum `BSTACK` address at which a local macro definition may occur. Similar slots exist for other constructions (skips, inserts, warning markers).

Initially the slot points at the end of `BSTACK`, and hence all definitions are in use. At a `MCNODEF` the slot is reset to contain the current value of `LFPT`, thus temporarily invalidating all the local macros on `BSTACK`. This happens in a similar way, using the appropriate slot, for `MCNOSKIP`, `MCNOINS` and `MCNOWARN`.

The `CKVALY` subroutine checks whether a construction is valid.

1.2 `FSTACK`

`FSTACK` has the following format:



New global definitions are added below the permanent variables, and the rest of the stack is shifted up to make room for them.

When an operation macro is entered, `FSTACK` is used to set up the values of its arguments. It is also used for workspace in setting up delimiter structures. When an exit is made from an operation macro, all the workspace is cleared.

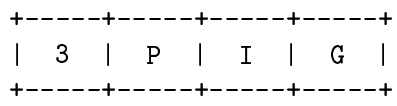
When the source text has been processed, it is cleared from `FSTACK`. However, in the case of a call (or insert), the entire call has to be read in and processed before it can be deleted. Outside of calls, the `GTATOM` subroutine clears all source text from `FSTACK` at every possible opportunity.

2 Formats of stack entries for constructions

When a new construction is defined it is set up on FSTACK or BSTACK, depending on whether it is global or local respectively.

Names of constructions are linked in hash chains for speedy recognition. There are separate chains for locals and globals.

The name of an atom of a delimiter is represented by a LID (see [2], section 6.1.2), which consists of the length of the atom, represented as a number of characters, followed by the character string representing it. For example:



The name of a delimiter is represented by the concatenation of the atoms which make it up. The atoms are separated by one of the following markers, which are numbers:

WITHMK this means spaces are *not* allowed between the atoms.

WTHSMK this means spaces *are* allowed between the atoms.

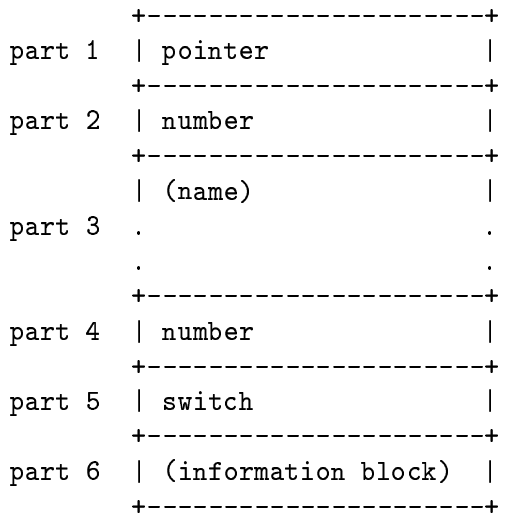
The keyword SPACES is implemented thus:

a. SPACES WITH and SPACES WITHS are both treated as SPACE WITHS.

b. An occurrence of SPACES at the end of a delimiter name is represented as a space with the marker SPCSMK following. Hence SPCSMK can only occur at the end of a delimiter name.

2.1 Format of Construction Names

The names of constructions take the following format:



where:

part 1 *Hash chain.* Absolute pointer. The marker NULLPT indicates the end of a chain.

`part 2` *Orlink.* Relative pointer to alternative names. The marker `ENDCHN` indicates the end of a chain.

`part 3` *Name.* Sequence of LIDs.

`part 4` *Nextlink.* Relative pointer to chain of successors. `ENDCHN` indicates the end of a chain, i.e. a closing delimiter. `EXCLMK` indicates an exclusive delimiter (the values of `WITHMK` and `WTHSMK` must be chosen so that they are not identical with possible values of part 4).

`part 5` *Type.* Switch indicating type of construction. 1 for macro, 2 for warning marker, 3 for insert, 4 for skip.

`part 6` *Information block.* The format depends on the type of the construction (see Section 2.1.1 [Information Block Format], page 6). If a construction has several names, each has its own information block.

2.1.1 Format of Information Blocks

The first number in an information block indicates whether it belongs to a straight scan or normal scan substitution macro, to an insert or to an operation macro. Other constructions are treated separately.

2.1.1.1 Warning Marker

The information block is null.

2.1.1.2 Skip

The information block for a skip is as follows:

```
          +----------------------+
 part 1   | switch               |
          +----------------------+
```

where:

`part 1` *Attributes.* Switch indicating the skip's attributes. Only the least significant three bits are used. Counting from the least significant end of the storage unit, starting at zero, the bits are assigned as follows. A value of 0 means the option is not set, and a value of 1 means the option is set.

`bit 0` Delimiter option.

`bit 1` Text option.

`bit 2` Matched option.

2.1.1.3 Substitution macro

The information block for a substitution macro is as follows:

```
          +---------------------+
part 1    | number              |
          +---------------------+
part 2    | number              |
          +---------------------+
part 3    | number              |
          +---------------------+
part 4    | number              |
          +---------------------+
```

where:

`part 1` This part is optional, and may be omitted completely. If present, it must have the value `STRMK`, and indicates that the macro is straight-scan. If absent, the macro is normal-scan.

`part 2` Relative offset of end of replacement text (more exactly, the character beyond the end of the replacement text).

`part 3` Relative offset of start of replacement text.

`part 4` Number of temporary variables (the *capacity* of the macro).

Example

```
MCDEF ABC AS XYZ
```

gives rise to:

```
                +----------------------------------+
                |                                  |
                V                                  |
+-+-+-+------------+--------+-+-+-+-+--------+-+--+---+-+
|X|Y|Z| hash chain | ENDCHN |3|A|B|C| ENDCHN |1|-8|-12|3|
+-+-+-+------------+--------+-+-+-+-+--------+-+--+---+-+
 ^                                                  |
 |                                                  |
 +--------------------------------------------------+
```

assuming that each "box" occupies one unit of storage of the correct kind.

Note that parts 2 and 3 are negative and part 1 positive, so the presence or absence of part 1 can be found by the sign of the first number in the information block.

2.1.1.4 Insert

The information block for an insert is as follows:

```
          +---------------------+
part 1    | number              |
          +---------------------+
```

where:

`part 1` *Type.* Indicates the type of insert:

- `value 3` Indicates unprotected insert.
- `value 2` Indicates protected insert.

2.1.1.5 Operation macro

The information block for an operation macro is as follows:

```
          +----------------------+
part 1    | number               |
          +----------------------+
part 2    | number               |
          +----------------------+
```

where:

`part 1` *Type.* Index for `TEST` statement in MI-logic (circa line 557) which selects the code to deal with an operation macro.

`part 2` *Class.* Indicates class of operation macro, as follows:

`LOCMK` (value 1)
: indicates local NEC macro.

`OPMK` (value 0)
: indicates other operation macro.

2.1.2 Format of Secondary Delimiter

Information concerning secondary delimiters is stored as follows:

```
          +----------------------+
part 1    | number               |
          +----------------------+
          | (name)               |
part 2    .                      .
          .                      .
          +----------------------+
part 3    | number               |
          +----------------------+
```

where:

`part 1` *Orlink.* Relative pointer to alternative names. The marker `ENDCHN` indicates the end of a chain.

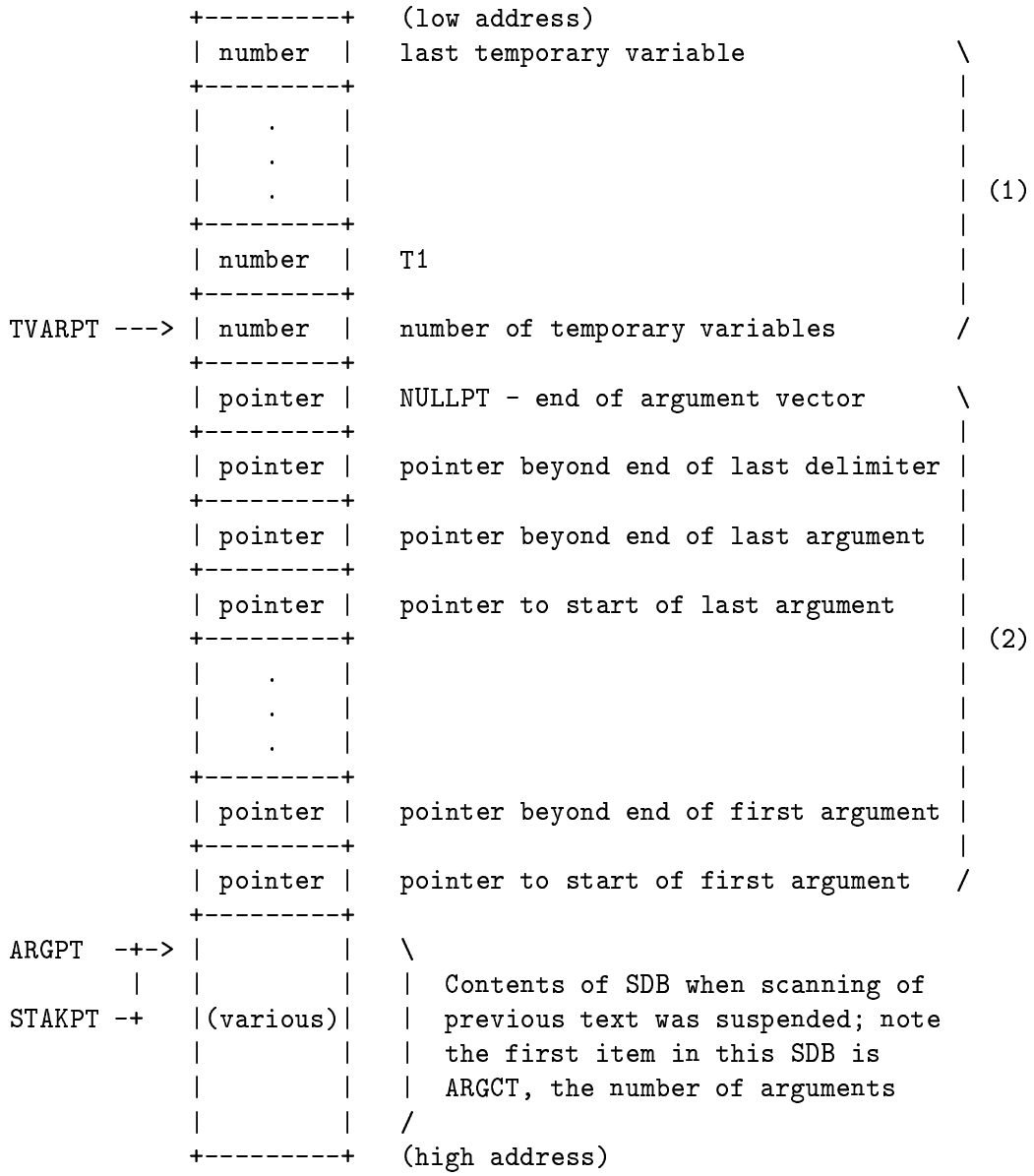
`part 2` *Name.* Sequence of LIDs.

`part 3` *Nextlink.* Relative pointer to chain of successors. `ENDCHN` indicates the end of a chain, i.e. a closing delimiter. `EXCLMK` indicates an exclusive delimiter (the values of `WITHMK` and `WTHSMK` must be chosen so that they are not identical with possible values of part 3).

3 Information stacked at calls or inserts

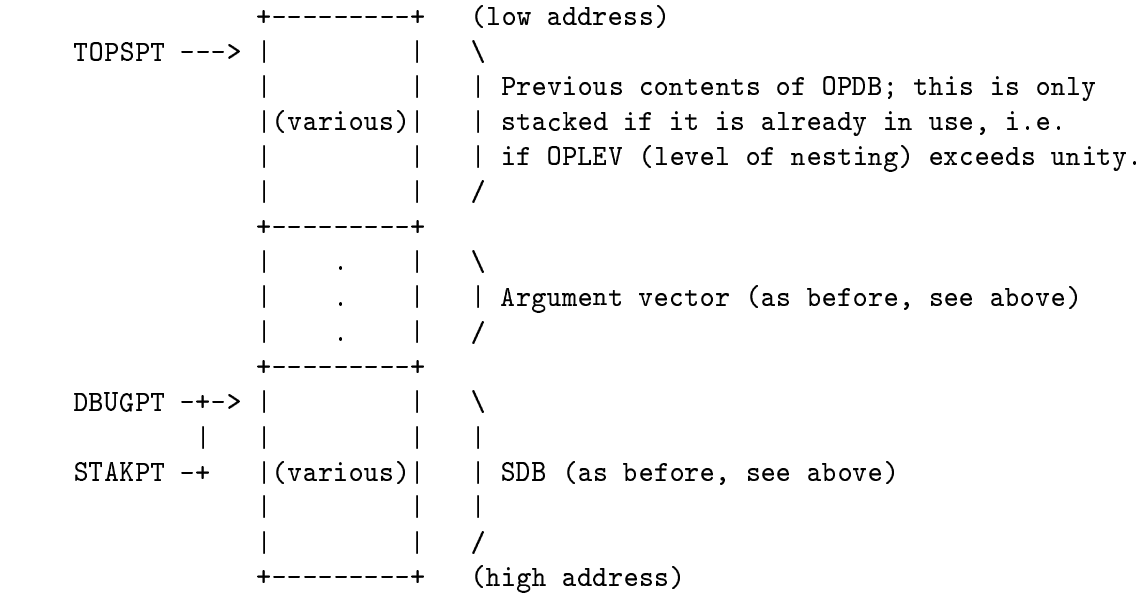
A *block* of variables is a group of variables that describe in some way the state of ML/I. When ML/I encounters a situation involving nesting, these blocks often need to be stacked so that they can be restored to their original state when the nested activity is complete. Blocks are fully described in Chapter 5 of [2]. Each block has a name. The most important block is the one that describes the current state of scan; this is called the `SDB` (scanning description block). Another important block is the `OPDB`, which describes the current state during processing of an insert or an operation macro.

The following information is stacked on `BSTACK` when a substitution macro is called:



In the above, (1) is the *temporary variable vector*, and (2) is the *argument vector*.

When an operation macro or insert is called, the following information is stacked:

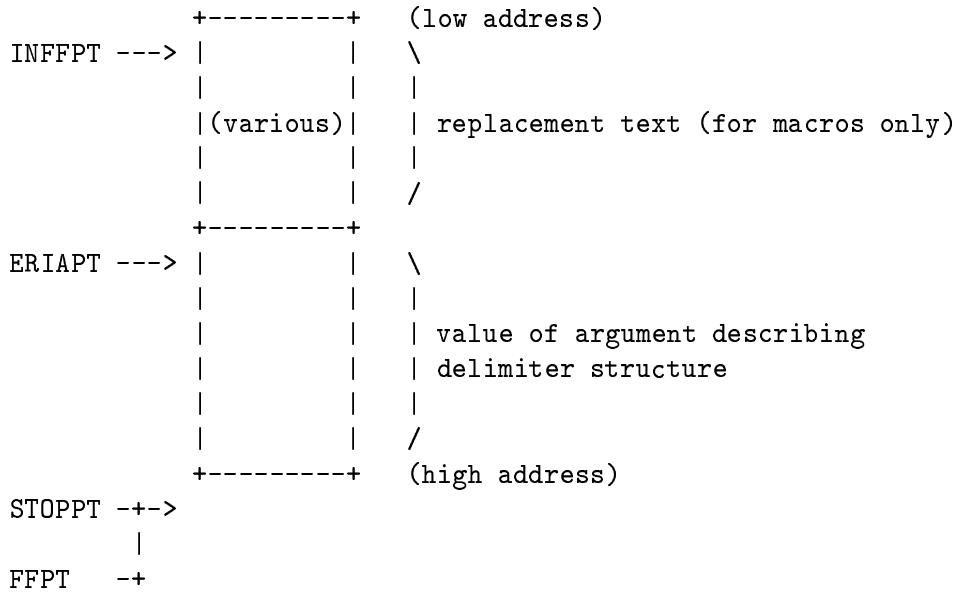


When the argument of an insert or operation macro is evaluated, no further stacking is necessary.

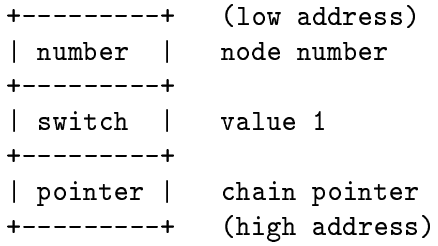
If an insert causes an argument or delimiter to be inserted, then the stack is partially collapsed by setting LFPT as STAKPT (see above diagram) before evaluating the inserted text. When the inserted text has been evaluated, the stack is, of course, reset to its state before the insert.

4 Setting up new constructions

A relatively large and complicated section of the logic is devoted to setting up definitions of constructions. The same code, EVTREE, is used to set up the delimiter structures of all constructions. When EVTREE is entered, FSTACK is thus:

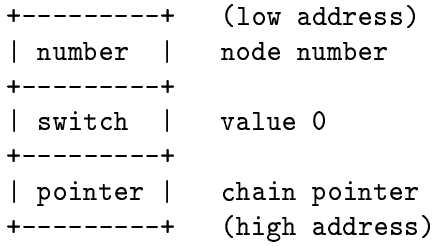


The code labelled EVTREE and subroutine GETDEL build up the new delimiter structure at the top of FSTACK. There is a block of information, the ALL block, associated with OPT-ALL brackets and this block is stacked on BSTACK at each OPT and restored at each ALL. BSTACK also contains information about nodes. The entry for a node depends on whether it has been placed yet. When a node has been placed, its stack entry is:

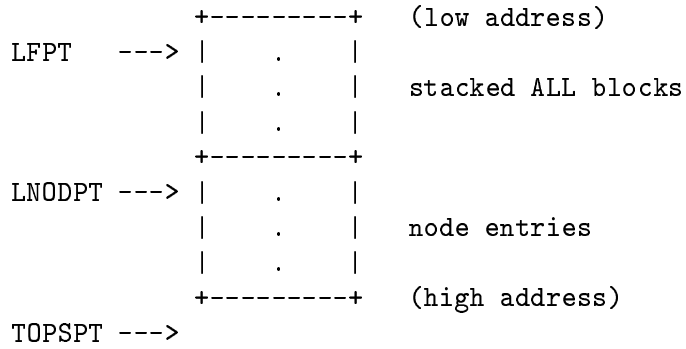


The chain pointer points to the orlink of the delimiter (or head of chain of delimiters) designated by the node.

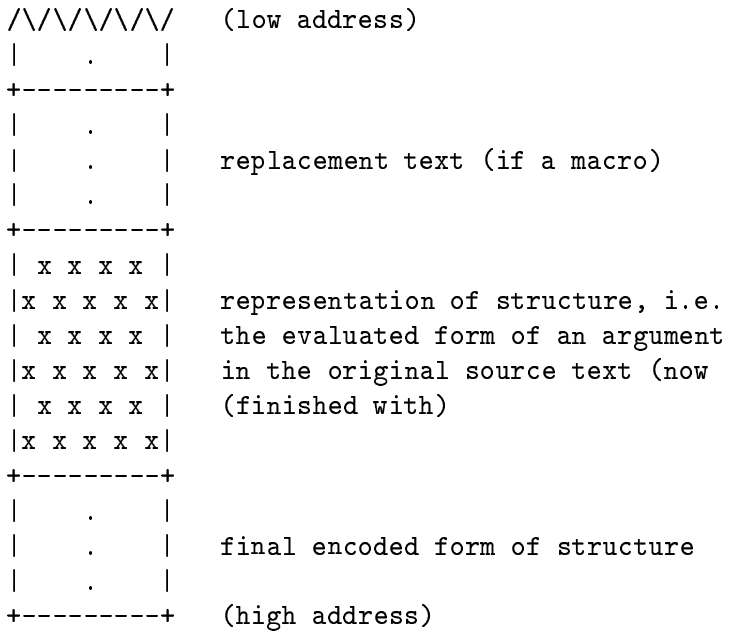
For a node that is gone to before being placed, the stack entry is (until the placing occurs):



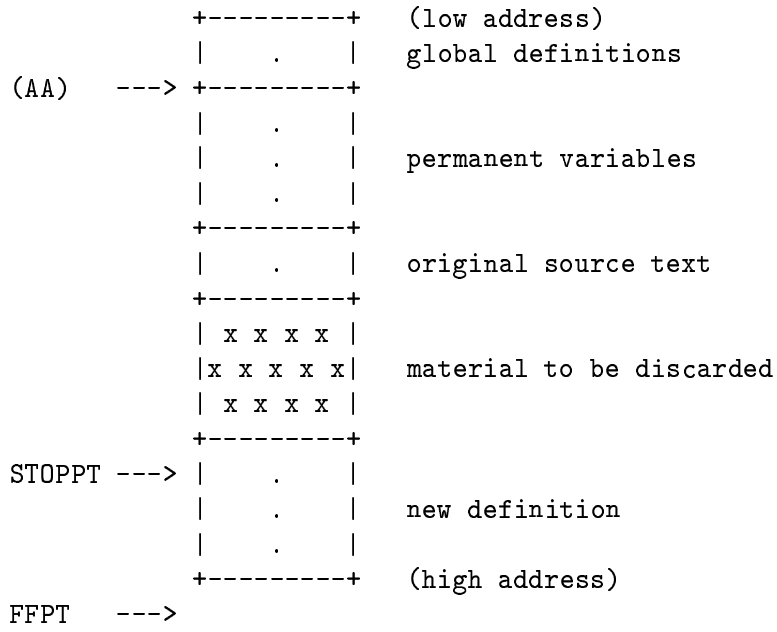
The chain pointer points to the chain of nextlinks. Each nextlink is to be filled with the node address when the node is finally placed. Entries are arranged on BSTACK as follows:



Note that when a new node entry is made, all the ALL blocks are shifted up to make room. After EVTREE has finished, the top of FSTACK is:

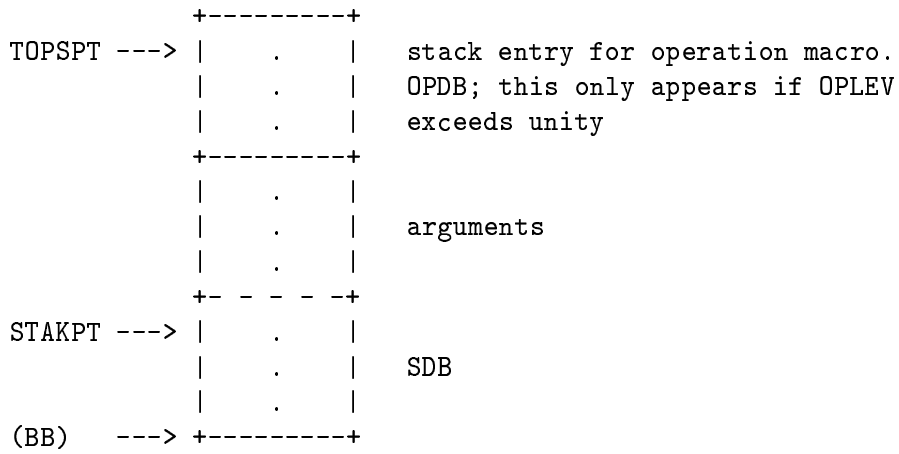


In the case of a macro, the delimiter structure is shifted up to fill in the dead space. After this has been done, the stacks have the following form in all cases. First, FSTACK:



OFFSET gives the offset from STOPPT of the start of the chain of names of the new definition.

Now BSTACK:



In the global case, the new definitions must be placed at (AA); in the local case, at (BB) and upwards. The latter case is quite simple. The SDB is, of course, unstacked before the new definition is moved on top of it.

In the global case, the MKROOM subroutine makes a "hole" of the required size at (AA). The new definition is then moved into the hole.

4.1 Details of EVTREE code

A number of switches are used in checking the syntax of the structure representation. These are as follows:

NODESW on if a node is legal in next position.

KEYSW on if a keyword is legal in next position.

`CONSW` used in checking correctedness. See table below.

`EXITSW` true only if an exclusive delimiter has been found.

Note that correctedness checking is inadequate; the following case is not detected:

```
OPT X N1 ALL N2 XX N1 XXX
```

Here, `XX` is not connected, but this is not detected as it is preceded by a node. The tables below indicate how the checking is performed; it is all carried out inside the `GETDEL` subroutine.

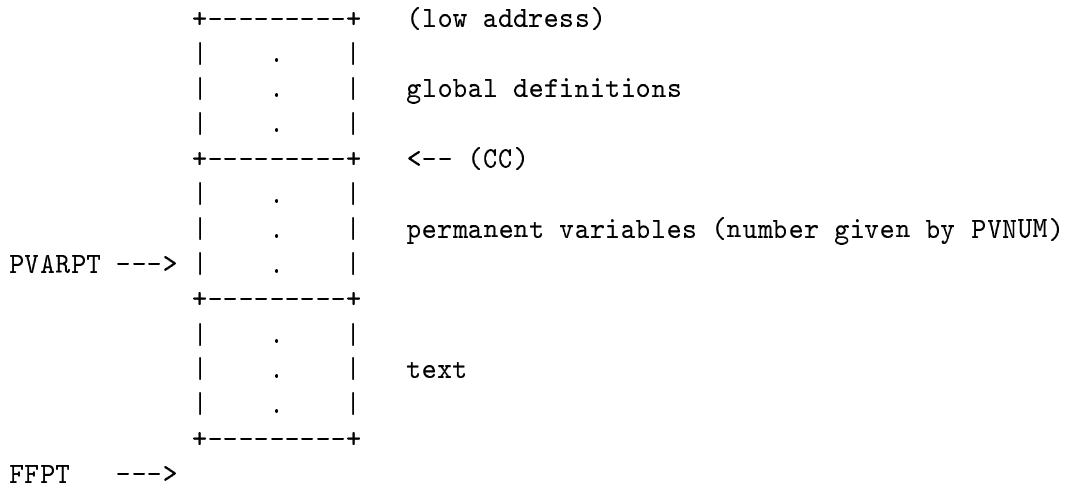
	at start	**after del**	**after node**	**after OPT**	**after OR**	**after ALL**	**after WITH**
`NODESW`	T	T	F	F	T(U)	T(U)	F
`KEYSW`	T	T	U	F	F	T	F
`CONSW`	T	T(U)	T	U	?	F	I

In the above, T means true, F means false, U means unchanged, I means immaterial and ? means undefined. Additional rules are:

if node is placed:	`MEVAL` must be nonzero	`NODEPT` *can* be null
if node is gone to:		`NODEPT` *must not* be null
at delimiter:		`NODEPT` can only be null if `CONSW` is true.

4.2 Details of `MKROOM` routine

This routine makes room for a new global definition or extra permanent variables. `FSTACK` is as follows:



`MKROOM` makes space for a block of length `IDLEN` to be inserted at `(CC)`. All pointers to the text that have been stacked on `BSTACK` are adjusted by `IDLEN`.

5 Scanning of constructions

The scanning of calls and skips may involve nesting. When a nested construction is encountered, the following information is stacked on BSTACK:

DELPT head of chain of delimiters being searched for.

MTCHPT pointer at orlink of name of construction.

MCHLIN line count at start of construction.

CLLFPT points to the latest block of information stacked (CLLFPT is used by the error routine that is invoked if stack overflow occurs).

5.1 Use of SDB

The SDB describes the current state of the scan, and is always stacked when a nested construction is encountered. Part of the SDB, the EDB, is for error message production. Some variables in the SDB have different uses depending on the state of processing. The table that follows shows the meanings of the variables in the SDB in each of the possible states.

"Calling value" is the value that was in force when the current insert or macro was called. "Containing text value" applies to the insertion of arguments and delimiters, and means the value in force when the text containing the argument or delimiter was being evaluated.

The following pages indicate the uses of variables declared in the SDB.

name of variable	1. scanning source text	2. scanning replacement text	3. scanning argument or delimiter	4. evaluating operation macro or first insert
ARGCT	counts number of arguments when nested construction is scanned			not used
STAKPT	NULLPT	points at latest SDB on stack		
ARGPT	NULLPT	points at arguments of macro	containing text value	calling value
DBUGPT	not used	points at orlink preceding macro name	points at argument vector in which argument or delimiter is included	points at argument vector
MCHLIN	set to current value of LINECT when a nested construction is encountered			not used
LINECT	line count of current text			not used
ARGNO	not used	not used	number of argument or delimiter	number of argument currently being processed
DBUGSW	0	1	2 for operation macro argument 4 for substitution macro argument 6 for delimiter	5
HASHPT	points at local hash table		U insert: calling value P insert: containing text value	calling value
TVARPT	NULLPT	points at temporary variables	containing text value	calling value
MTCHPT	when a nested construction is encountered, this is set to point at the orlink preceding the name of the construction			used as workspace
SPT	points at last scanned character			used in scanning values of arguments
STOPPT	NULLPT	points beyond last character of current text		points beyond end of latest argument
LABPT	NULLPT	head of chain of labels		not used
INFFPT	points at start of source text on FSTACK	not used	for operation macro argument, as 4., otherwise undefined	value of FFPT when operation macro was called

name of variable	1. scanning source text	2. scanning replacement text	3. scanning argument or delimiter	4. evaluating operation macro or first insert
SKVAL	if not scanning call or insert then number of designated label for forward GOTO; zero otherwise. if scanning call or insert then set to (-1 - [above value])			not used
OHSW	true if text has its own local hash table. false if hash table of another piece of text is being used			not used
SKLIN	if in forward GOTO then line number in which MCGO occurred			not used

Descriptions of further variables used in the logic of ML/I are given in Appendix A. In the future these comments will be incorporated into listings of the logic.

References

1. Brown, P.J. and Eager, R.D., *ML/I User's Manual*, Sixth Edition.
2. Brown, P.J. and Eager, R.D., *Implementing software using the L language*.

Appendix A Description of Uses of Variables

The following summarises the uses of important variables in the MI-logic of ML/I.

Variable	Declared in line	Meaning
ALLPT	61	head of chain of nextlinks to be attached to delimiter following ALL
ARGCT	13	in SDB
ARGLEN	49	length of value of current argument
ARGNO	24	in SDB
ARGPT	19	in SDB
BESLIN	123	best-so-far value of LINECT
BESPT	96	best-so-far value of SPT
BESTPL	120	switch value used in basic scan routine
BFNDPT	87	best-so-far value of FNDPT
BINDIC	111	best-so-far value of INDIC
BINFPT	89	best-so-far value of INFOPT
CALTYP	124	first number in information block
CHANPT	94	for chaining
CHLINK	116	chain link
CINFPT	103	points at information block
CLLFPT	101	points to top entry after scanning information is stacked
CONSW	150	for syntax checking
COPDSW	131	for skips. Reflects setting of delimiter option.
COPTSW	130	for skips. Reflects setting of text option.
DBUGPT	20	in SDB
DBUGSW	25	in SDB
DELCT	145	count of delimiters
DELPT	104	head of chain of delimiters
ENDPT	71	points at end of BSTACK
ERIAPT	95	points at value of operation macro argument
ETEMPT	84	points at error block (temporary storage for EDB)
EXIDPT	140	
EXITSW	151	for syntax checking
FFPT	90	points to first free location on FSTACK
FLAGPT	158	points at flag
FNDPT	86	points at LID when a name is found
GHSHPT	845	points at global hash table
GLBWSW	705	value 6 if there is a global warning; value 7 otherwise
HASHPT	30	in SDB
HTABPT	97	points at current hash table
IDLEN	119	length of current identifier
IDPT	98	points at current identifier
INDIC	115	contents of nextlink
INFFPT	34	in SDB
INFOPT	88	points beyond currently matched LID

INSW	132	to set DBUGSW, and as implied parameter to SETPTS
INVOCT	113	count of macro calls
KEYSW	149	for syntax checking
KNPT	83	
LABPT	33	in SDB
LEVEL	112	level of macros and inserts
LFPT	73	points at top of BSTACK (last used location)
LINECT	23	in SDB
LINKPT	48	link for STKARG
LNODPT	139	points at topmost node entry on BSTACK
MASKSW	129	mask used to indicate which types of construction are recognised
MCHLIN	22	in SDB
MEVAL	122	miscellaneous. Used for numerical values calculated at macro time
MHSHPT	46	value of HASHPT when macro was called
MTCHPT	31	in SDB
MTYPE	175	type of items to be printed
NARGPT	102	points at argument vector
NDEFPT	143	destination of new definition
NEGVAL	162	indicates if result is to be negative
NESTLV	109	nesting level of calls and skips during scanning
NNODPT	138	points at current node entry on BSTACK
NODEPT	137	head of chain of links to be attached to next delimiter
NODESW	148	for syntax checking
NTYPSW	53	type of construction being defined
OFFSET	121	offset in hash table
OHSW	37	in SDB
OLDSPT	142	previous value of SPT
OLIDPT	157	temporary storage for IDPT
OLLFPT	141	previous value of LFPT
OP1	160	LHS operand
OPLEV	110	level of operation macros and inserts
OPSW	164	"operation expected" switch
OPTHPT	663	head of orlink chain
OPTLEV	146	level of OPT-ALL brackets
OPTPT	62	last entry on orlink chain
OPTYP	50	type, e.g. global, local, insert. Also other uses
PARNM	117	parameter of type number
PARPT	91	parameter of type pointer
PARSW	128	parameter of type switch
PRSTPT	172	start of current hash table
PRT1PT	173	counts down hash table
PRT2PT	174	follows down hash chains
PVARPT	72	points at permanent variables
PVNUM	74	number of permanent variables
SKIPLV	108	

SKLIN	36	in SDB
SKVAL	35	in SDB
SPT	21	in SDB
SQNUM	51	safe variable, miscellaneous uses
SQPT	47	safe variable, miscellaneous uses
SQSW	52	safe variable, miscellaneous uses
STAKPT	18	in SDB
STOPPT	32	in SDB
SUM	161	running total
TEM1PT	93	
TEMP	114	temporary number
TEMPSW	127	temporary switch
TEMPT	92	temporary pointer
TIDPT	99	temporary storage for IDPT
TLINCT	125	temporary storage for LINECT
TOPSPT	45	points at latest OPDB on BSTACK
TSPT	100	temporary storage for SPT
TVARPT	29	in SDB
TYPE	118	type of construction name
VARPT	156	points at vector of variables
VARSW	165	"variable expected" switch
WNIDPT	106	temporary storage for IDPT
WNSPT	105	temporary storage for SPT
WSW	166	written argument or delimiter

Appendix B List of subroutines and code sections in L

The following lists all of the sections, subroutines and other important areas of code in the L source.

Name	Section	Type	Declared in line
ADVNCE	MAINSUBS	Subroutine	383
BUMPFF	MAINSUBS	Subroutine	395
CHATOM	MAINSUBS	Subroutine	406
CHEKID	MAINSUBS	Subroutine	418
CKVALY	MAINSUBS	Subroutine	430
CMPARE	MAINSUBS	Subroutine	442
CORECT	MAINSUBS	Subroutine	460
DECALV	MAINSUBS	Subroutine	471
DECLF	MAINSUBS	Subroutine	484
DEFSUBS		Section	1448
ENCALL	MAINSUBS	Subroutine	495
ENVPR		Section	2116
ER1TST	MAINSUBS	Subroutine	511
ERMTST	ERR	Subroutine	1773
ERR		Section	1712
ERSIC	ERR	Subroutine	1743
ERSNW	ERR	Subroutine	1759
ERTEST	DEFSUBS	Subroutine	1450
GARGCH	MAINSUBS	Subroutine	523
GETDEL	DEFSUBS	Subroutine	1460
GETEXP	MAINSUBS	Subroutine	535
GMEADD	MAINSUBS	Subroutine	584
GSATOM	MAINSUBS	Subroutine	610
GSRATM	DEFSUBS	Subroutine	1610
GTATOM	MAINSUBS	Subroutine	626
INVALS		Section	8
JOINCH	DEFSUBS	Subroutine	1622
KEYSRC	DEFSUBS	Subroutine	1639
LUDEL	MAINSUBS	Subroutine	652
LULAYK	MAINSUBS	Subroutine	665
MAIN		Section	26
MAINSUBS		Section	380
MCALTERMAC	OPMACS	Code for	1024
MCDEFMAC	OPMACS	Code for	1238
MCGOMAC	OPMACS	Code for	1056
MCINSMAC	OPMACS	Code for	1260
MCLENGMAC	OPMACS	Code for	1135
MCNO---	OPMACS	Code for	1216
MCNOTEMAC	OPMACS	Code for	1148
MCPVARMAC	OPMACS	Code for	1161

Concept Index