```
************************************************
************************************************
***                                          ***
***  Name:                                   ***
***                                          ***
***  Project:    1       Programmer:  MWK    ***
***                                          ***
***  File Name: MIN[ 1,MWK]                  ***
***                                          ***
***  File Last Written:  19:10 15 Jul 1973   ***
***                                          ***
***  Time: 19:10         Date: 15 Jul 1973   ***
***                                          ***
***           Stanford University            ***
***     Artificial Intelligence Project      ***
***       Computer Science Department        ***
***          Stanford,  California           ***
***                                          ***
************************************************
************************************************
```

COMMANDS

(Typing CTRL-ALT at any time causes control to be passed immediately to the basic command loop, terminating whatever other command happens to be progress at the time.)

w (write world)
FORM: w [<file_name>]
Writes the current state of the model to the disk file specified. If none is specified, write is to the file most recently specified in either a "w" or "r" command, or to WORLD.NEW in case no file has ever been mentioned. The current state of the model is unaffected. Normally, a "w" command should be issued at the end of a session to record the current state.

r (read world)
FORM: r [<file_name>]
Similar to the "w" command. Reads the file into memory as the current state of the model, destroying the existing state. An "r" command must be issued at the beginning of each session before anything else. The file read is unaffected.

e (end session)
FORM: e
Terminates the MIND program without preserving the current state of the world. Closes any output recording file (see "out", "fout").

we (write world and end session)
FORM: we [<file_name>]
Equivalent to a "w" command followed by an "e" command.

l (lookup dictionary word)
FORM: l <word>
Not a very useful command, but it tells you whether a word exists in the dictionary.

i (insert a dictionary word)
FORM: i <word>
Also not a very useful command. Enters the word in the dictionary if it isn't already there. Creates a null association block for the word.

a (form associations with a word)
FORM: a {<word> | <graph_number>}
where <graph_number>= ##<digits>
Creates a dictionary entry and null association block for the word if none exist. Enters "associate" mode which can be exited by typing ALT at the start of the line.
In associate mode, associations to other words or graphs may be established by typing <association_type> {<word> | <graph_number>} for each association. Entries in the dictionary (for words only) and association blocks are created for associated items when they are not already present. See the appendix for association types.
Associations may be deleted in associate mode by typing the association type and word or graph number of the association to be deleted, terminating the line with an ALT.

dv (display the dictionary over specified range)
FORM: dv [<word> [<word>]]
Lists the dictionary over the specified range.

dva (display the dictionary, with associations, over specified range)
FORM: dva [<word> [<word>]]
Similar to the "dv" command, except that the association block for each word is also displayed.

rw (remove a word from the model)
FORM: rw <word>
Deletes the dictionary entry and association block for a word if possible. The action is "impossible" if the word is referenced in a graph or node.

da (display association block for a word)
FORM: da <word>
Displays all associations for a word.

g (build a graph)
FORM: g
Enters graph-building sub-command mode, which can be prematurely exited at any time by CTRL-ALT with no ill effects. Graph-mode consists of two parts: token naming and link building. In token naming, the tokens of the graph are merely listed, as many to the line as possible. Transition to link building mode is made by typing ALT. At that time the system names the tokens by letters of the alphabet. These letters are used in link building mode to reference the tokens. A link is built by the format
<token_letter> <link> [<link_modifier>] <token_letter>
one link per line. The graph is completed and "fixed" into memory by typing ALT in link building mode. This exits to the outer command loop. See the appendix for links and link modifiers.

dg (display graphs)
FORM: dg <graph_number_range> | dg <token-list>

```
      where <graph_number_range>= <graph_number> [<graph_number>]
      and   <token_list>= <word> ... <word>
      and   <graph_number>= ##<digits>
```

In the first form, graphs are listed over a specified range. Graph numbers are "floating", and are not guaranteed to remain the same as the model is changed. In the second form, graphs are displayed by their contents: all graphs which reference all tokens of the token list are displayed.

dga (display graphs and their association blocks)
FORM: dga (same as "dg" command)
Similar to the "dg" command, except that the association block of each graph is also displayed.

rg (remove a graph from the model)
FORM: rg <graph_number>
Deletes the specified graph from the system if possible. The operation is "impossible" when the graph is referenced by another graph or by a node. Note that the removal of a graph can alter the "floating" reference numbers of other graphs.

s (display model statistics)
FORM: s
Displays vital statistics of the model such as sizes of memory areas.

out (specify an output listing file)
FORM: out <file_name>
Sets the name of an output listing file. If another listing file is already open, it is closed first. The listing file simply provides a means of creating a hard copy of the model's output responses to commands. Responses from a command are recorded on this file only when the command is immediately preceeded by the character "!". The "out" command need never be issued before using "!". In this case, output is recorded on the file WORLD.LST.

fout (finish output listing file)
FORM: fout
The current output listing file is closed. The name of the file remains set, however, so that any subsequent uses of "!" before another "out" would re-write the file. "fout" is implicitly called by the "e" and "we" commands before the model terminates.

Data area format.
=================

```
+------+-----+-----+-----+-----+-----+ ←←←Q.FF
|             dictionary             |
+------+-----+-----+-----+-----+-----+ ←←←Q.D
|        dictionary overflow         |
+------+-----+-----+-----+-----+-----+ ←←←Q.G
|        graph pointer table         |
+------+-----+-----+-----+-----+-----+ ←←←Q.N
| .       node pointer table         |
+------+-----+-----+-----+-----+-----+ ←←←Q.M
|  word, graph, node memory area     |
+------+-----+-----+-----+-----+-----+ ←←←Q.S
|             stack area             | ←←←Q.SC,Q.SL
+------+-----+-----+-----+-----+-----+
                                       ←←←Q.F
```

Dictionary.
===========

2 words are allocated to each dictionary word. Their format is

```
+------+-----+-----+-----+-----+-----+
|xc ch1 ⊃c ch2 ⊃c ch3 ⊃c ch4 ⊃c ch5 ⊃|
+------+-----+-----+-----+-----+-----+
|xc ch6 ⊃c ch7 ⊃xxxc    pointer     ⊃|
+------+-----+-----+-----+-----+-----+
```

The pointer points to the first overflow word for the entry if any exists, or to the first word of the word's association block in the memory area otherwise. The first pair of words of the dictionary area are null dummy words.

Dictionary overflow area.
=========================

As many overflow words as are needed for an entry are present. All but the last have their first bit set. The first overflow word is pointed to by the primary dictionary entry, and the pointer in the last word of the overflow points to the first word of the word's association block in the memory area.

```
+-------+------+------+------+------+------+
|a⊂ ch  ⊃⊂ ch  ⊃⊂ ch  ⊃⊂ ch  ⊃⊂ ch  ⊃|
+-------+------+------+------+------+------+
a=1 if not last ovlf word, 0 if last
```

Graph pointer table.
====================

This table provides a way of referencing graphs in memory by their "floating" ordinal at any given time. The first word of this area is set to all ones, and serves only as a boundary word. Words in this area have the format:

```
+-------+------+------+------+------+------+
|0xxxxxxxxxxxxxxxxxx⊂    pointer     ⊃|
+-------+------+------+------+------+------+
```

where pointer points to the graph block first word.

Node table.
===========

The node table is of the same form as the graph table, except that pointer points to the first word of its node.

Word, graph, node memory area.
==============================

This area is the heart of the system. It contains three types of structural units which can be interspersed within the area in any order. The three unit types are the ASSOCIATION BLOCK for a word, the GRAPH BLOCK and the NODE BLOCK. The format of an ASSOCIATION BLOCK is:

```
+-------+------+------+------+------+------+
|⊂ T.ABLK⊃⊂ L.DICT⊃⊂  dict. pointer ⊃|
+-------+------+------+------+------+------+
|⊂ T.NULL⊃⊂ L.STAK⊃⊂ stack ptr or 0 ⊃|
+-------+------+------+------+------+------+
|⊂ T.NULL⊃⊂ l-type⊃⊂  link pointer  ⊃|
+-------+------+------+------+------+------+
|                (etc.)                |
+------+-----+------+-----+-----+------+
```

The format of a GRAPH BLOCK is:

```
+------+-----+-----+-----+-----+-----+
|⊂ T.GRPH⊃⊂ g-len ⊃⊂ g-table pointer⊃|    Q
+------+-----+-----+-----+-----+-----+
|⊂ T.NULL⊃⊂ L.STAK⊃⊂ stack ptr or Ø ⊃|
+------+-----+-----+-----+-----+-----+
|                                    |
|             graph body             |
|                                    |
+------+-----+-----+-----+-----+-----+
|⊂      -(distance from here to Q)  ⊃|
+------+-----+-----+-----+-----+-----+
|⊂ T.NULL⊃⊂ l-type⊃⊂  link pointer  ⊃|
+------+-----+-----+-----+-----+-----+
|                (etc.)              |
+------+-----+-----+-----+-----+-----+
```

The region below the graph body corresponds to the association block for the graph, and need not be present if no associations exist. A graph body word has the format:

```
+------+-----+-----+-----+-----+-----+
|Øa⊂b⊃⊂  t  ⊃⊂  l ⊃⊂ token ptr or Ø ⊃|
+------+-----+-----+-----+-----+-----+
```

where a existentially quantifies the token if it is set
      b is the variable number of the token, or Ø if it is a constant
      t is the CD link type of the inter-graph link, l
      l is the inter graph CD link pointer

If the token pointer field is zero, the graph body word is a continuation of the last token (last graph body word whose token pointer field is non-zero. All links are two-way, with the convention that the reverse link type is the forward link type + 1ØØ octal.

The format of a NODE BLOCK is:

```
+------+-----+-----+-----+-----+------+
|⊂ T.NODE⊃⊂ n-type⊃⊂ node table ptr ⊃|
+------+-----+-----+-----+-----+------+
|⊂ T.NULL⊃⊂ L.STAK⊃⊂ stack ptr or Ø ⊃|
+------+-----+-----+-----+-----+------|
|                                     |
|             antecedents             |
|                                     |
+------+-----+-----+-----+-----+------+
|                                     |
|             consequents             |
|                                     |
+------+-----+-----+-----+-----+------+
```

where n-type is the type of the node, and is either ANDNOD or ORNOD.

Antecedents and consequents have the following form:

```
+------+-----+-----+-----+-----+-----+
|⊂ type  ⊃⊂ cert  ⊃⊂ constituent ptr⊃|
+------+-----+-----+-----+-----+-----+
|1⊂ s1⊃⊂ s2⊃⊂ s3⊃⊂ s4⊃⊂ s5⊃⊂ s6⊃⊂ s7⊃|
+------+-----+-----+-----+-----+-----+
|                (etc.)              |
+------+-----+-----+-----+-----+-----+
```

```
where type is either L.ANTE or L.CONS,
      cert is the level of certainty required for an antecedent or
           the propagated certainty of a consequent
and sn is the nth binding slot.
```

The constituent pointer can go to any other unit in memory.