

T H E U N I V E R S I T Y O F M I C H I G A N

Memorandum 32

AN EXAMPLE DEFINITIONAL FACILITY IN MAD/I

Ronald J. Srodawa

CONCOMP: Research in Conversational Use of Computers  
ORA Project 07449  
F.H. Westervelt, Director

supported by:

DEPARTMENT OF DEFENSE  
ADVANCED RESEARCH PROJECTS AGENCY  
WASHINGTON, D.C.

CONTRACT NO. DA-49-083 OSA-3050  
ARPA ORDER NO. 716

administered through:

OFFICE OF RESEARCH ADMINISTRATION ANN ARBOR

August 1970



## Abstract

The MAD/I language is a procedure-oriented algebraic language which is a descendant of ALGOL 60 and 7090 MAD, similar in power and scope to PL/I. The MAD/I compiler is implemented using the MAD/I facility, a flexible translator-building system whose dynamic nature allows compilers to be extended during the compilation process. This paper demonstrates the extension of MAD/I to include several graphics-oriented statements and operators through a lucid example.



## 1. INTRODUCTION

The MAD/I project of the University of Michigan has designed and implemented a flexible translator-building system called the MAD/I facility. The facility provides services to aid in the lexical and syntactic scanning [3] of the program, storage allocation, and object-code generation. A compiler is written in the facility as a set of procedures, called a macro, to which is control passed at various times by the syntactic scanner and by the contents of the intermediate storage of the partially compiled programs. New macros can be redefined while a compiler is executing, thus making extensions to the compiler (and hence to the language) possible.

A compiler, called the MAD/I compiler, has been implemented using the MAD/I facility. The language accepted by the MAD/I compiler is called the MAD/I language [1]. The MAD/I language is a procedure-oriented algebraic language which is a descendant of ALGOL 60 and 7090 MAD, similar in power and scope to PL/I. Because the MAD/I compiler is written in the MAD/I facility, there is a great potential for extensibility features within the MAD/I language. To date, no extension facilities have been designed for the MAD/I language; that is properly a goal of further research.

This report presents an example definitional facility in the MAD/I language. A simple list-processing program is written in the MAD/I language as extended to include three

new modes, three new statements, and eight new operators. These extensions are written using the macro language of the MAD/I facility and two experimental definitional statements. These definitional statements, or similar ones determined to be more appropriate, could easily be incorporated as a part of the MAD/I language. For the moment, they also are defined at compile-time.

The remainder of this report explains in detail the simple program and the code necessary to define the language extensions. This explanation references the computer output which appears at the end of the report. This output consists of six parts:

- (1) a listing of the contents of the file SKETCH which is the sample MAD/I program,
- (2) a listing of the contents of the DISPLAYDEF which defines the extensions fo the MAD/I language,
- (3) a listing of the contents of the file DEFFACILITY which defines the two experimental statements,
- (4) a listing of the contents of the file -DATA which contains the data used in the run of Step (6),
- (5) the compilation of the MAD/I program, and
- (6) the run of the generated object program using the data of Step (4).

The object-code listing of the compilation has been removed to reduce the bulk of the report.

## 2. SKETCH

The file SKETCH contains the sample MAD/I program. This program maintains a simple list structure representing points, lines, and collections of points called pictures. The list structure can be manipulated or printed through several commands which are recognized by the program.

These commands are:

- POINT        which adds a point to the list structure. The user is prompted for the x and y coordinates of the point. The point is assigned an internal display number which is used to reference the point in other commands.
- LINE         which adds a line to the list structure. The user is prompted for the internal display numbers of the two endpoints of the line. The line is defined in terms of its endpoints and will be moved appropriately if its endpoints are moved.
- PICTURE     which groups several points together into a collection called a picture. The user is prompted for the internal display numbers of the points in the picture. Whenever one point of a picture is translated, all the points in the picture are translated.
- MOVE        which moves a point to new x and y coordinates. The user is prompted for the display number of

the point and its new x and y coordinates. If the point is in a picture, all other points in the picture are also translated by the same amount.

DISPLAY which prints a display of the current list structure.

This program is oriented to standard typewriter terminals, such as teletypes. It could easily be modified to interface to a remote graphics terminal using the display subroutines developed as a part of the CONCOMP Project [2].

Line 1 of SKETCH simply begins a procedure named SKETCH.

Line 3 of SKETCH includes the contents of the file

DISPLAYDEF which defines the new statements, modes, and operators which will be used in this program.

The contents of DISPLAYDEF will be described in the next section.

Lines 5 through 13 declare the modes of variables

used in the program. Note that 'POINT', 'LINE', and 'PICTURE' are used as modes in declarations.

These have been defined as described for line 3

above. Line 5 causes all variables which are not explicitly declared to receive the default mode

'INTEGER'.

Line 15 presets the number of pictures to zero.



Lines 18 and 19 prompt the user for the next command from his terminal. The first four characters of the command are stored in the variable COMMAND.

Lines 21 through 120 form a compound 'IF' statement.

The subsection of this statement which corresponds to the command entered is given control.

Lines 22 through 29 are invoked by the POINT command.

Note that line 25 uses the newly defined statement 'CREATE POINT' to create a point having the values of X and Y as its coordinates. X and Y, although shown here as simple variables, can be general expressions. The operator .DISPN., which accesses the internal display number from a point, is used in line 26 to print the display number to the user.

Lines 32 through 39 are invoked by the LINE command.

The operator .ADROF. used on line 34 converts an internal display number to a 'POINTER' to the corresponding list structure item. The operator .EVAL. also used on line 34 sets the storage allocation of a based variable to the value of a variable of 'POINTER' mode. In this case P1 and P2 are allocated to the list structure items corresponding to the two endpoints. .EVAL. is a built-in MAD/I operator whose name has since been changed to .ALLOC.. The operator .POINT. used

in line 35 returns a value of 'TRUE' if its operand is a list structure item corresponding to a point; 'FALSE' otherwise. Note that line 36 uses the new statement 'CREATE LINE' to create the line whose endpoints are P1 and P2.

Lines 42 through 57 are invoked by the PICTURE command.

PICTURE is an array of up to 100 pictures, each element being the head of a linked ring of points in the picture. PICTUREN is the number of pictures allocated thus far. Lines 43 through 47 increment the number of pictures thus far, test to see that less than 100 pictures have been formed, and initialize the current picture to the empty set. The O.AS. ('POINTER') of line 47 is the empty picture constant and would better have been written O.AS.('PICTURE'). As we will see later, 'PICTURE' has been defined as a synonym for 'POINTER' which explains why the former case works. Lines 48 through 55 are executed once for each point in the picture. Lines 50 and 51 access the list structure item corresponding to the next point to be coded to the picture and test that is a point. Lines 52 through 54 insert the point into the picture using the 'CONNECT' statement. A restriction in the implementation of our experimental define statement facility prevents us from rewriting these three statements as the one statement

```
'CONNECT' P1 'TO' PICTURE(PICTUREN)
```

Lines 60 through 75 are invoked by the MOVE command.

Line 64 computes the difference between the new coordinates of the point and the old coordinates of the point. The two operators .XOF. and .YOF. access the x and y coordinates respectively of a point. Line 65 modifies the coordinates of the point to their new values. Note that .XOF. and .YOF. can be used on the left-hand side of an assignment as they return a reference. Lines 66 and 67 test if the point is a member of a picture. If the point is in a picture, line 68 accesses the next point in the picture, and lines 69 through 73 are executed for each point in the picture until we return to the original point. The .NEXT. operator returns a 'POINTER' result which points to the list structure item representing the next point in the same picture as its operand.

Lines 77 through 116 are invoked by the DISPLAY command.

This code runs through the entire list structure and generates points and lines as sets of asterisks in the array DISPLAY. This array is then printed to give a visual depiction of the display on a typewriter-like terminal. Note that the variable HEAD referenced in line 83 is a 'POINTER' to the first item in the list-structure. The operator .HEAD. referenced in line 109 returns a 'POINTER' to the

list structure element which follows its operand.

This operator is used to traverse the list structure.

Line 119 is invoked if the command was not recognizable based on its first four characters.

Line 122 transfers control back to line 18 where the user is prompted for the next command.

The remainder of the program consists of two small procedures for computing the minimum and maximum of two values.

### 3. DISPLAYDEF

The file DISPLAYDEF contains the definitions for the extensions to the MAD/I language used in the preceding program. In actual practice, packages of definitions such as this would be written and used in programs much as subroutines are written and used in programs at present. Generally useful definitional packages would be provided by system programmers for general use just as subroutine libraries are now provided.

Lines 18 through 20 define the mode 'POINT' which is simply a synonym for a based component structure. The components are used as described in lines 11 through 16.

Lines 35 through 37 define the mode 'LINE' which is simply a synonym for a based component structure. The components are used as described in lines 28 through 33.

Line 43 defines the new mode 'PICTURE' which is simply a synonym for 'POINTER'.

Lines 48 through 52 declare and preset the variables which are used by the various statements and operators of the definitional package.

Line 56 includes the contents of the file DEFFACILITY which defines the two experimental definitional statements which are used below. The contents of DEFFACILITY will be described in the next section.

Lines 64 through 74 define the statement 'CREATE POINT' using the experimental definitional statement 'DEFINE STATEMENT'. The 'DEFINE STATEMENT' facility allows a new statement to be defined in terms of other MAD/I statements. The 'CREATE POINT' statement consists of the keyword 'CREATE POINT' followed by three expressions which correspond to the identifiers POINT, X, and Y. These three expressions will be evaluated. Then the MAD/I statements in the definition will be executed, with the results of the three expressions being substituted for each occurrence of POINT, X, and Y. Line 65 allocates a block of storage to the expression corresponding to POINT, which must be a reference to a variable of 'POINT' mode. Line 66 and 68 insert this new point into the chain of all items in the list structure. Line 68 initializes the point as not being an element of any picture. Lines 69 and 70 assign the next internal display number to this point. In an application using a remote display this would be an identification number for the element in the remote display program so that light-pen detects could be mapped back to the data structure in the machine in which this program is running. Line 71

sets the display item type to 1, indicating that this is a point. Lines 72 and 73 set the x and y coordinates of the point.

Lines 82 through 92 define the statement 'CREATE LINE'. This definition is similar to that used to define the 'CREATE POINT' statement above and won't be discussed further.

Line 100 defines the keyword 'TO' to be syntactically equivalent to the comma (,). This will allow us to write 'CONNECT' A 'TO' B rather than 'CONNECT' A,B. Note that this definition is done using the experimental 'DECLARE SYNTACTIC CLASS' statement.

Lines 101 through 111 define the 'CONNECT' statement. This definition is also made using the experimental definitional statement 'DEFINE STATEMENT'. Lines 104 and 105 are executed if the second point is already a member of a picture. In this case the new point is inserted into the existing ring.

Lines 116 through 124 define the operator .POINT. which returns 'TRUE' if its operand is a point, 'FALSE' otherwise. This definition is written using the macro language of the MAD/I facility and requires some explanation. If .POINT. A is written we really want to transform that into A(4)=1, a test of whether the type component of A is equal to 1. Now .POINT. A will be converted by the

syntactic scanner into the triple:

```
.POINT.,%TMP,A
```

where %TMP is an internally generated temporary symbol which represents the result of the operation. Now  $A(4)=1$  would be converted by the syntactic scanner into the two triples:

```
.TAG.,%TMP1,A,4
```

```
=%TMP2,%TMP1,1
```

where %TMP2 is the result of the expression. Now, if we define a macro whose name is .POINT., it will be called by the syntactic scanner with two operands, the temporary assigned and the operand. This macro can in turn generate the two triples that  $A(4)=1$  would have generated. Line 116 declares .POINT. to be syntactically equivalent to .ABS.; that is, a unary operator with the same left and right precedence values as .ABS.. Line 117 declares to the compiler that what follows are to be considered as statements directed to the MAD/I facility. Lines 118 through 123 define the macro whose name is .POINT. and whose two operands (parameters) are given the names T and B. All identifiers in a macro definition, unless preceded by a %, are different identifiers than those of the same name in the MAD/I program. Likewise, all constants referenced in a macro definition, unless



preceded by 'LOCAL LITERAL', are self-defining constants rather than literal constants within the MAD/I program. Line 119 declares U to be a local symbol within this macro. This is roughly equivalent to automatic variables in higher-level languages. Line 120 calls the macro TEMPORARY which assigns a temporary symbol and causes U to become a synonym for the temporary. This temporary will be used as the result of the = operator. The macro TEMPORARY is defined in the next section. Line 121 generates the triple

```
.TAG., U,B,4
```

where U is the temporary result and B is the operand of the .POINT. operator. The 'LOCAL LITERAL' keyword is required so that the symbol 4 represents the MAD/I constant value 4 rather than a self-defining term 4 in the MAD/I facility. Likewise line 122 generates the triple

```
=,T,U,1
```

where T is the temporary result of the .POINT. operator which has been passed as a parameter and U is the result of the .TAG. operation generated by the preceding line. The LN is necessary preceding the "=" to indicate that this is the MAD/I operator "=" rather than the MAD/I facility

operator "=". These two triples generated are the two we have previously discussed as being equivalent to  $A(4)=1$ . Line 124 exits from the MAD/I facility. Further lines are interpreted as being a part of the MAD/I program being compiled.

Lines 129 through 186 define the operators `.XOF.`, `.YOF.`, `.NEXT.`, `.DISPN.`, `.ENDA.`, `.ENDB.`, and `.HEAD.` in a manner similar to the definition of `.POINT.` discussed above. In each case the expression involving the operator, say `.XOF.A`, is to be mapped into an instance of subscription such as  $A(5)$ . The operators differ only in the value of the subscript used. In each case the triple resulting from the syntactic scanning of the former case,

```
.XOF. ,%TMP,A
```

is translated into the triple which would result from the syntactic scanning of the latter case,

```
.TAG. ,%TMP,A,5
```

In each case the operator is declared to be syntactically equivalent to `.ABS.` through the 'DECLARE SYNTACTIC CLASS' statement and is semantically defined through a very simple macro which generates the corresponding `.TAG.` triple.

Lines 196 through 214 define the operator `.ADROF.` which returns the `.POINTER'` to the list-structure element

which has been assigned the value of its operand as its internal display number. The .ADROF. operator could be represented in MAD/I by lines 190 through 194, However, we have not yet implemented a statement which allows operators to be defined in terms of MAD/I statements. Instead, we have implemented the .ADROF. operator as a macro which generates the same triples as would be generated by the MAD/I statements shown. Line 199 defines B1 and B2 to be local symbols. These will be used for the labels required. Lines 200 and 201 assign temporaries to T1 through T5. Line 202 calls the FLAD macro which assigns two floating addresses and makes B1 and B2 synonyms for these two floating addresses. The macro is defined in the next section. Line 203 is equivalent to the MAD/I statement of line 190. Lines 204 through 207 are equivalent to the MAD/I statement of line 191. Line 204 allocates the floating address B1 to the current value of the instruction location counter. Lines 205 and 206 compute the Boolean expression

```
.DISPN. QQSV = A
```

while line 207 transfers to B2 if the expression result is 'TRUE'. Lines 208 and 209 are equivalent to the MAD/I statement of line 192. Line 210 is equivalent to the MAD/I statement of line 193. Lines 211 and 212 are equivalent to the intended effect of the MAD/I statement of line 194 which is to return a pointer to QQSV as the result of the operator. Line 211 allocates the floating

address B2 to the current value of the instruction location counter while line 212 computes the pointer to QQSV assigning the result to T, the temporary assigned by the syntactic scanner as the result of .ADROF..

## 4. DEFFACILITY

The file DEFFACILITY contains the definitions of the 'DECLARE SYNTACTIC CLASS' and 'DEFINE STATEMENT' statements and the FLAD and TEMPORARY macros. Other macros have also been defined or redefined as required to implement the above. The macros used to define these statements are much more complicated than the macros used in the preceding section and require a detailed knowledge of the MAD/I facility and MAD/I compiler in order to implement them successfully. We stress that users of MAD/I will not be required to learn this detail, as appropriate higher-level definitional statements such as 'DECLARE STATEMENT' will be provided for them; only the system programmer assigned to MAD/I need to know these details.

Lines 10 through 27 define the 'DECLARE SYNTACTIC CLASS' statement. Line 11 causes 'DECLARE SYNTACTIC CLASS' to be considered syntactically a keyword which begins a simple statement. The macro named 'DECLARE SYNTACTIC CLASS' will be called by the syntactic scanner whenever the keyword is encountered in the MAD/I program. Lines 12 through 26 define the macro 'DECLARE SYNTACTIC CLASS'. Lines 14 and 15 scan off the next symbol and insert a pointer to its symbol table entry into the local symbol A. Lines 16 through 19 scan off the next symbol and

verify that it is 'SAME AS'. Line 20 scans off the next symbol which is the one having the desired syntactic qualities. Lines 21 through 24 set the syntactic attributes in the symbol table entry of the symbol being declared to the same values as on the symbol already having the desired syntactic class. Line 25 scans off the statement terminator so that we are ready to return to the syntactic scanner.

Lines 49 through 100 define the statement 'DEFINE STATEMENT'. Lines 52 through 99 define the macro 'DEFINE STATEMENT' which is called by the syntactic scanner whenever the keyword 'DEFINE STATEMENT' is encountered. This macro scans the entire 'DEFINE STATEMENT' statement scope, saving its contents as a list of symbol table pointers. It then creates a macro (lines 81 through 97) which has as its name the keyword identifying the statement being defined. This new macro is called by the syntactic scanner whenever its namesake keyword is found in the input stream. It then will call the syntactic scanner once for each expression which is a part of the statement, modify the lexical scanner (GETDSK) to return the symbol table pointers on the list formed above before continuing with

the standard input text, and call the syntactic scanner asking it to scan the scope of a compound statement. This scope, of course, consists of the statements which define the new statement saved on the list given to the lexical scanner. Lines 108 through 126 define a macro named GETDSX. This macro will be called instead of the pseudo-operation GETDSX, which itself is the entry-point to the lexical scanner. This new macro will normally simply call the pseudo-operation GETDSX to lexically scan for the next symbol in the input stream. However, if it is passed a list of symbol table pointers via the global symbol GTDSXLIST from a macro defining a new statement, it will return the symbols from the list until the list is exhausted.

Lines 131 through 144 define the macro FLAD which generates new floating address symbols. This macro is referenced by the .ADROF. macro from the preceding section.

Lines 148 through 161 define the macro TEMPORARY which assigns new temporary result symbols. This macro is referenced by several macros in the preceding section.

Lines 167 through 223 re-define the macros GETTEMP and FREETEMP to remove deficiencies in their original

implementation in the MAD/I compiler. These macros have since been changed in the compiler so that this update is no longer necessary.



## 5. -DATA

The file -DATA contains the data presented to the MAD/I program in the run of Section 7. The program is intended to be used from a terminal on a conversational manner. Running the program in batch has required us to anticipate the requests for input and the assignment of internal display numbers. The reader will find it helpful to look at the printed output from the run with this data (Section 7) while reading Section 5.

Line 1 requests a display of the current contents of the list-structure. Since the list-structure is empty the comment "NOTHING TO DISPLAY." is printed.

Lines 2 through 9 define four points having coordinates (10,10), (10,40), (40,10) and (40,40). These points are assigned internal display numbers 1,2,3, and 4, respectively. Line 10 requests the current list-structure to be displayed, resulting in the first graph showing the four points.

Lines 11 through 18 connect the four points with lines forming a square. Line 19 requests the current list-structure to be displayed, resulting in the second graph showing the square. (This looks like a rectangle because the horizontal scale is 10 characters/inch while the vertical scale

is 6 characters/inch (before reduction)).

Lines 20 and 21 move the first point from its original position of (10,10) to the new position (15,20). Since this point is not a member of any picture, it is the only point moved. Line 22 displays the third graph showing the point moved to its new location.

Lines 23 through 28 mark the four points as members of the same picture. Lines 29 and 30 move the first point from its current position of (15,20) to the new position (20,20). The second, third, and fourth points are also translated horizontally by five raster units because they are members of the same picture as the first point. Line 31 causes the display of the fourth graph which shows the results of this translation.

## 6. COMPILATION OF THE MAD/I PROGRAM

The fifth listing is the printed output resulting from the compilation of the MAD/I program. This output begins with a listing of the source program. Notice that the contents of the files DISPLAYDEF and DEFFACILITY are included at the points where the 'INCLUDE' statements are encountered. Following the source program listing is the output of the storage allocation phase, giving the storage allocated to each variable and constant in the program. Following that is a dictionary giving the attributes of each variable and constant. Following that are the external symbol dictionary, relocation dictionary, and statistics for the compilation. The object listing has been left out because of its size (about 40 printed pages).

## 7. RUN OF THE MAD/I PROGRAM

The last listing is the printed output resulting from a run of the generated object module. This listing consists of the loading map followed by the printed output from the program. See Section 5 for an annotation of the output from the run.

## BIBLIOGRAPHY

1. Bolas, B.J., Springer, A.L., and Srodawa, R.J., The MAD/I Manual, Technical Report 32, Concomp Project, University of Michigan, Ann Arbor, August 1970.
2. Cocanower, A.B., The DF Routines User's Guide, Memorandum 23, Concomp Project, University of Michigan, Ann Arbor, May 1969.
3. Mills, D.L., The Syntactic Structure of MAD/I, Technical Report 7, Concomp Project, University of Michigan, Ann Arbor, June 1968.

Appendix A. Contents of File SKETCH

\$LIST SKETCH

```

1      'PROCEDURE' SKETCH.;
2
3      'INCLUDE' "DISPLAYDEF"
4
5      'DECLARE' 'NORMAL MODE' 'INTEGER';
6      'DECLARE' COMMAND 'CHARACTER'(4);
7      'DECLARE' DISPLAY 'FIXED ARRAY'(50,50) 'CHARACTER'(1);
8      'DECLARE' (POINT,P1,P2,P3) 'POINT';
9      'DECLARE' LINE 'LINE';
10     'DECLARE' PICTURE 'FIXED ARRAY'(100) 'PICTURE';
11     'DECLARE' QQ1 'POINTER';
12     'DECLARE' QQ2 'POINTER';
13     'DECLARE' (M,M1,M2) 'FLOATING';
14
15     'PRESET' PICTUREN := 0;
16
17
18     SKETCH: 'WRITE' , "'ENTER A COMMAND PLEASE.'*";
19     'READ' , "C4.4*" , COMMAND;
20
21     'IF' COMMAND = "POIN";
22         'WRITE' , "' ENTER X AND Y COORDINATES:'*";
23         'READ' , "2I*", X,Y;
24         'IF' 1 <= X & 50 >= X & 1 <= Y & 50 >= Y;
25             'CREATE POINT' POINT,X,Y;
26             'WRITE' , "' ASSIGNED DISPLAY NUMBER',HI4*", .DISPN. POINT;
27         'ELSE';
28             'WRITE' , "' POINT IS OUTSIDE RASTER RANGE.'*";
29         'END';
30
31     'OR IF' COMMAND = "LINE";
32         'WRITE' , "' ENTER DISPLAY NUMBERS FOR END-POINTS:'*";
33         'READ' , "2I*", X,Y;
34         P1 .EVAL. .ADROF. X; P2 .EVAL. .ADROF. Y;
35         'IF' .POINT. P1 & .POINT. P2;
36             'CREATE LINE' LINE,P1,P2;
37         'ELSE';
38             'WRITE' , "' THOSE ARE NOT POINTS.'*";
39         'END';
40
41     'OR IF' COMMAND = "PICT";
42         'WRITE' , "' ENTER DISPLAY NUMBERS FOR ALL POINTS.'*";
43         PICTUREN := PICTUREN+1;
44         'IF' PICTUREN > 100;
45             'WRITE' , "' TOO MANY PICTURES.'*";
46         'ELSE';
47             PICTURE(PICTUREN) := 0 .AS. ('POINTER');
48             'READ' , "I*", X;
49             'IF' X = 0;
50                 P1 .EVAL. .ADROF. X;
51                 'IF' ~.POINT. P1, 'GO TO' NOTPOINT;
52                 QQ2 := PICTURE(PICTUREN);
53                 'CONNECT' P1 'TO' QQ2;
54                 PICTURE(PICTUREN) := QQ2;
55                 'GO TO' PICTA;
56             'END'
57         'END';
58

```

```

59      'OR IF' COMMAND = "MOVE";
60          'WRITE' , " ENTER DISPLAY NUMBER OF POINT AND NEW X,Y";
61          'READ' , "3I*", DIS,X,Y;
62          P1 .EVAL. .ADROF. DIS;
63          'IF' ~.POINT. P1, 'GOTO' NOTPOINT;
64          DX := X-.XOF. P1; DY := Y-.YOF. P1;
65          (.XOF. P1) := .XOF. P1+DX; (.YOF. P1) := .YOF. P1+DY;
66          QQ1 := P1(2);
67          'IF' QQ1 .AS. ('INTEGER') ~ = 0;
68              P2 .EVAL. .NEXT. P1;
69 MOVEA:      'IF' .DISPN. P1 ~ = .DISPN. P2;
70              (.XOF. P2) := .XOF. P2+DX; (.YOF. P2) := .YOF. P2
71              +DY;
72              P2 .EVAL. .NEXT. P2;
73              'GO TO' MOVEA;
74          'END';
75      'END';
76
77      'OR IF' COMMAND = "DISP";
78          'IF' HEAD .AS. ('INTEGER') = 0;
79              'WRITE' , " NCTHING TO DISPLAY."";
80              'GO TO' SKETCH;
81          'END';
82          'FOR' I := 1,1,I>50, 'FOR' J := 1,1,J>50, DISPLAY(I,J) := " ";
83          P1 .EVAL. HEAD;
84 DISPA:      ;
85          'IF' .POINT. P1;
86              Q1 := .XOF. P1; Q2 := .YOF. P1; DISPLAY(Q2,Q1) := " ";
87          'ELSE';
88              LINE .EVAL. (.PT. P1);
89              QQ1 := .ENDA. LINE; P2 .EVAL. QQ1;
90              QQ1 := .ENDB. LINE; P3 .EVAL. QQ1;
91              X1 := MIN(.XOF. P2,.XOF. P3);
92              X2 := MAX(.XOF. P2,.XOF. P3);
93              'IF' X1 = X2;
94                  Y1 := MIN(.YOF. P2,.YOF. P3);
95                  Y2 := MAX(.YOF. P2,.YOF. P3);
96                  'FOR' Y := Y1,1, Y > Y2, DISPLAY(Y,X1) := " ";
97              'ELSE';
98                  M1 := .YOF. P3 - .YOF. P2;
99                  M2 := .XOF. P3 - .XOF. P2;
100                 M := M1/M2;
101                 'FOR' X := X1,1,X > X2;
102                     Q2 := M*(X - .XOF. P2) + .YOF. P2;
103                     DISPLAY(Q2,X) := " ";
104                 'END';
105             'END';
106         'END';
107         QQ1 := .HEAD. P1;
108         'IF' QQ1 .AS. ('INTEGER') ~ = 0;
109             P1 .EVAL. .HEAD. P1;
110             'GO TO' DISPA;
111         'END';
112         'WRITE' , "1 .',10(' .')*";
113         'FOR' I := 50,-1, I < 1,
114             'WRITE' , "I3,52C1.1*", I," ",",",
115                 DISPLAY(I,1)...DISPLAY(I,50);
116         'WRITE' , " .',10(' .')*";
117
118     'ELSE';

```

```
119         'WRITE' , "" ILLEGAL COMMAND*";
120     'END';
121
122     'GO TO' SKETCH;
123
124     'PROCEDURE' MIN.(X,Y);
125     'INTEGER SHORT' (X,Y);
126     MIN: 'IF' X <= Y, 'RETURN' X;
127         'RETURN' Y;
128         'END';
129
130     'PROCEDURE' MAX.(A,B);
131     'INTEGER SHORT' (A,B);
132     MAX: 'IF' A >= B, 'RETURN' A;
133         'RETURN' B;
134         'END';
135
136     'END'
```

END OF FILE



Appendix B. Contents of File DISPLAYDEF

IST DISPLAYDEF

1 <<  
2  
3  
4  
5  
6  
7  
8  
9  
10  
11  
12  
13  
14  
15  
16  
17 >>  
18  
19  
20  
21 <<  
22  
23  
24  
25  
26  
27  
28  
29  
30  
31  
32  
33  
34 >>  
35  
36  
37  
38 <<  
39  
40  
41  
42 >>  
43  
44 <<  
45  
46  
47 >>  
48  
49  
50  
51  
52  
53 <<  
54  
55 >>  
56  
57 <<  
58

DISPLAY DEFINITIONAL PACKAGE

THE FOLLOWING PARAMETER STATEMENT DEFINES THE COMPONENT STRUCTURE FOR A POINT. THE USER SIMPLY USES 'POINT' AS A MODE NAME IN EITHER A 'DECLARE' STATEMENT OR AN @ CCSTRUCTION. NOTE THAT THE STATEMENT

'POINT' (VARIABLE LIST)  
WILL NOT WORK BECAUSE THIS IS SIMPLY A PARAMETRIC SUBSTITUTION. THE STRUCTURE OF A POINT IS

- |   |               |                                      |
|---|---------------|--------------------------------------|
| 1 | POINTER       | POINTER TO NEXT LIST ITEM            |
| 2 | POINTER       | POINTER TO NEXT ITEM IN SAME PICTURE |
| 3 | INTEGER SHORT | DISPLAY ITEM NUMBER                  |
| 4 | INTEGER SHORT | DISPLAY ITEM TYPE (1 FOR A POINT)    |
| 5 | INTEGER SHORT | X COORDINATE OF THE POINT            |
| 6 | INTEGER SHORT | Y COORDINATE OF THE POINT            |

'PARAMETER' 'POINT' 'BASED' 'COMPONENT STRUCTURE' ('POINTER', 'POINTER', 'INTEGER SHORT', 'INTEGER SHORT', 'INTEGER SHORT', 'INTEGER SHORT') 'ENDP'

THE FOLLOWING PARAMETER STATEMENT DEFINES THE COMPONENT STRUCTURE FOR A LINE. THE USER SIMPLY USES 'LINE' AS A MODE NAME IN EITHER A 'DECLARE' STATEMENT OR AN @ CONSTRUCTION. NOTE THAT THE STATEMENT 'LINE' (VARIABLE LIST)

WILL NOT WORK BECAUSE THIS IS SIMPLY A PARAMETRIC SUBSTITUTION. THE STRUCTURE OF A LINE IS

- |   |               |   |
|---|---------------|---|
| 1 | POINTER       | POINTER TO THE NEXT LIST ITEM           |
| 2 | POINTER       | POINTER TO THE NEXT ITEM IN THE PICTURE |
| 3 | INTEGER SHORT | DISPLAY ITEM NUMBER                     |
| 4 | INTEGER SHORT | DISPLAY ITEM TYPE (2 FOR A LINE)        |
| 5 | POINTER       | POINTER TO FIRST END-POINT              |
| 6 | POINTER       | POINTER TO SECOND END-POINT             |

'PARAMETER' 'LINE' 'BASED' 'COMPONENT STRUCTURE' ('POINTER', 'POINTER', 'INTEGER SHORT', 'INTEGER SHORT', 'POINTER', 'POINTER') 'ENDP'

THE FOLLOWING PARAMETER STATEMENT DEFINES THE MODE OF A PICTURE. A PICTURE IS A POINTER TO ONE OF THE ITEMS IN THE PICTURE. HENCE IT IS SIMPLY PARAMETERIZED TO 'POINTER' MODE.

'PARAMETER' 'PICTURE' 'POINTER' 'ENDP'

THE FOLLOWING STATEMENTS DEFINE THE GLOBAL VARIABLES USED BY THE DEFINITION PACKAGE.

'DECLARE' DISPLAYN 'INTEGER SHORT';  
'PRESET' DISPLAYN := 0;  
'DECLARE' QQSV 'POINT';  
'DECLARE' HEAD 'POINTER';  
'PRESET' HEAD := 0;

THE FOLLOWING BRINGS IN THE DEFINE FACILITY FROM THE FILE DEFFACILITY

'INCLUDE' "DEFFACILITY"

THE FOLLOWING DEFINES THE STATEMENT

```

59
60         'CREATE POINT' POINT,X,Y
61
62 WHICH HAS THE EFFECT OF THE LIST OF STATEMENTS SHOWN
63 >>
64 'DEFINE STATEMENT' 'CREATE POINT' POINT,X,Y;
65     'ALLOCATE' POINT;
66     POINT(1) := HEAD;
67     HEAD := .PT. POINT;
68     POINT(2) := (0 .AS. ('POINTER'));
69     DISPLAYN := DISPLAYN+1;
70     POINT(3) := DISPLAYN;
71     POINT(4) := 1;
72     POINT(5) := X;
73     POINT(6) := Y;
74 'END STATEMENT';
75 <<
76 THE FOLLOWING DEFINES THE STATEMENT
77
78         'CREATE LINE' LINE,P1,P2
79
80 WHICH HAS THE SAME EFFECT AS THE LIST OF STATEMENTS SHOWN
81 >>
82 'DEFINE STATEMENT' 'CREATE LINE' LINE,P1,P2;
83     'ALLOCATE' LINE;
84     LINE(1) := HEAD;
85     HEAD := .PT. LINE;
86     LINE(2) := (0 .AS. ('POINTER'));
87     DISPLAYN := DISPLAYN+1;
88     LINE(3) := DISPLAYN;
89     LINE(4) := 2;
90     LINE(5) := .PT. P1;
91     LINE(6) := .PT. P2;
92 'END STATEMENT';
93 <<
94 THE FOLLOWING DEFINES THE STATEMENT
95
96         'CONNECT' POINT 'TO' PICTURE
97
98 WHICH HAS THE SAME EFFECT AS THE LIST OF STATEMENT SHOWN
99 >>
100 'DECLARE SYNTACTIC CLASS' 'TO' 'SAME AS' , ;
101 'DEFINE STATEMENT' 'CONNECT' POINT 'TO' PICTURE;
102     QQ1 := PICTURE;
103     'IF' (QQ1 .AS. ('INTEGER')) = 0;
104         PICTURE := .PT. POINT;
105         POINT(2) := .PT. POINT;
106     'ELSE';
107         QQSV .EVAL. PICTURE;
108         POINT(2) := QQSV(2);
109         QQSV(2) := .PT. POINT;
110     'END';
111 'END STATEMENT';
112 <<
113 THE FOLLOWING MACRO DEFINES THE .POINT. OPERATOR AS A PASS ONE MACRO.
114 IT CAUSES .POINT. A TO BE TREATED AS A(4) = 1
115 >>
116 'DECLARE SYNTACTIC CLASS' .POINT. 'SAME AS' .ABS.;
117 'DEFINE';
118 MACRO,.POINT.,T,B;

```

```

119     LOCAL,U;
120     TEMPORARY,U;
121     .TAG.,U,B,'LOCALLITERAL' 4;
122     LN=,T,U,'LOCALLITERAL' 1;
123     MEND,.POINT.;
124     END;
125 <<
126     THE FOLLOWING MACRO DEFINES .XOF. AS A PASS ONE OPERATOR.
127     IT CAUSES .XOF. A TO BE TREATED AS A(5)
128 >>
129     'DECLARE SYNTACTIC CLASS' .XOF. 'SAME AS' .ABS.;
130     'DEFINE';
131     MACRO,.XOF.,T,A;
132     .TAG.,T,A,'LOCALLITERAL' 5;
133     MEND,.XOF.;
134     END;
135 <<
136     THE FOLLOWING CAUSES .YOF. TO BE DEFINED AS A PASS ONE OPERATOR
137     .YOF. A HAS THE SAME EFFECT AS A(6)
138 >>
139     'DECLARE SYNTACTIC CLASS' .YOF. 'SAME AS' .ABS.;
140     'DEFINE';
141     MACRO,.YOF.,T,A;
142     .TAG.,T,A,'LOCALLITERAL' 6;
143     MEND,.YOF.;
144     END;
145 <<
146     THE FOLLOWING DEFINES .NEXT. A TO BE THE SAME AS A(2)
147 >>
148     'DECLARE SYNTACTIC CLASS' .NEXT. 'SAME AS' .ABS.;
149     'DEFINE';
150     MACRO,.NEXT.,T,A;
151     .TAG.,T,A,'LOCALLITERAL' 2;
152     MEND,.NEXT.;
153     END;
154 <<
155     THE FOLLOWING DEFINES .DISPN. A TO BE THE SAME AS A(3)
156 >>
157     'DECLARE SYNTACTIC CLASS' .DISPN. 'SAME AS' .ABS.;
158     'DEFINE';
159     MACRO,.DISPN.,T,A;
160     .TAG.,T,A,'LOCALLITERAL' 3;
161     MEND,.DISPN.;
162     END;
163     'DECLARE SYNTACTIC CLASS' .ENDA. 'SAME AS' .ABS.;
164     'DEFINE';
165     MACRO,.ENDA.,T,A;
166     .TAG.,T,A,'LOCALLITERAL' 5;
167     MEND,.ENDA.;
168     END;
169 <<
170     THE FOLLOWING DEFINES .ENDB. A TO BE THE SAME AS A(6)
171 >>
172     'DECLARE SYNTACTIC CLASS' .ENDB. 'SAME AS' .ABS.;
173     'DEFINE';
174     MACRO,.ENDB.,T,A;
175     .TAG.,T,A,'LOCALLITERAL' 6;
176     MEND,.ENDB.;
177     END;
178 <<

```

```

179 THE FOLLOWING DEFINES .HEAD. A TO BE THE SAME AS A(1)
180 >>
181 'DECLARE SYNTACTIC CLASS' .HEAD. 'SAME AS' .ABS.;
182 'DEFINE';
183 MACRO,.HEAD.,T,A;
184 .TAG.,T,A,'LOCALLITERAL' 1;
185 MEND,.HEAD.;
186 END;
187 <<
188 THE FOLLOWING DEFINES .ADROF. A TO BE THE SAME AS THE SEQUENCE OF
189 STATEMENTS
190 QQSV .EVAL. HEAD
191 B1: 'IF' .DISPN. QQSV = A, 'GO TO' B2
192 QQSV .EVAL. (.HEAD. QQSV)
193 'GO TO' B1
194 B2: 'RETURN' .PT. QQSV
195 >>
196 'DECLARE SYNTACTIC CLASS' .ADROF. 'SAME AS' .ABS.;
197 'DEFINE';
198 MACRO,.ADROF.,T,A;
199 LOCAL,B1,B2;
200 LOCAL,T1,T2,T3,T4,T5;
201 TEMPORARY,T1,T2,T3,T4,T5;
202 FLAD,B1,B2;
203 .EVAL.,T1,%QQSV,%HEAD;
204 DES,B1;
205 .DISPN.,T2,%QQSV;
206 LN=,T3,T2,A;
207 TNZ,B2,T3;
208 .HEAD.,T4,%QQSV;
209 .EVAL.,T5,%QQSV,T4;
210 GUTU,B1;
211 DES,B2;
212 .PT.,T,%QQSV;
213 MEND,.ADROF.;
214 END;
215 <<
216 END OF DEFINITIONAL PACKAGE
217 >>
END OF FILE

```

Appendix C. Contents of File DEFFACILITY

LIST DEFFACILITY

```

1  <<
2  THE FOLLOWING DEFINES THE NEW SIMPLE STATEMENT
3
4      'DECLARE SYNTACTIC CLASS' A 'SAME AS' B
5
6  WHICH GIVES THE SYMBOL A EXACTLY THE SAME SYNTACTIC DEFINITION
7  (CLASS,SYNTACTIC CLASS,LEFT PRECEDENCE,RIGHT PRECEDENCE) AS B.
8  A WILL NOW BE TREATED EXACTLY AS B BY THE SYNTACTIC SCANNER.
9  >>
10 'DEFINE';          <<ESCAPE INTO THE MACRO LANGUAGE>>
11 SETUP,'DECLARE SYNTACTIC CLASS',14,3,4;    << SIMPLE KEYWORD>>
12 MACRO,'DECLARE SYNTACTIC CLASS';          <<START THE MACRO DEF>>
13 LOCAL,A;          << A IS SAVED IN THIS LOCAL>>
14 GTDSX;           << THIS READS A>>
15 SET,A,0,DSX;     <<SAVE DSX OF A IN LOCAL SYMBOL>>
16 GTDSX;           <<NOW READ 'SAME AS'>>
17 JMP,LOC+3,DSX = 'SAMEAS'; <<BE SURE IT REALLY IS 'SAMEAS'>>
18 MNOTE,"**** ONLY 'SAME AS' IS ALLOWED";
19 MEXIT;           <<LEAVE HIM AT THE MERCY OF JSCAN>>
20 GTDSX;           <<READ B>>
21 SET,IND.(A),1,DSX(1); <<SET THE CLASS CODE OF A>>
22 SET,IND.(A),2,DSX(2); <<SET THE SYNTACTIC CLASS OF A>>
23 SET,IND.(A),3,DSX(3); <<SET THE LEFT PRECEDENCE OF A>>
24 SET,IND.(A),4,DSX(4); <<SET THE RIGHT PRECEDENCE OF A>>
25 GTDSX;           <<READ THE STATEMENT TERMINATOR>>
26 MEND,'DECLARE SYNTACTIC CLASS';          <<ALL DONE>>
27 END;             <<BACK INTO MAD/I>>
28 <<
29 THE FOLLOWING DEFINES A SPECIAL PURPOSE DEFINITIONAL FACILITY
30 WHICH ALLOWS THE DEFINITION OF A SIMPLE STATEMENT. THE DEFINITIONAL
31 STATEMENT IS OF THE FORM
32
33     'DEFINE STATEMENT' KEYWORD LIST OF VARIABLES
34     LIST OF STATEMENTS
35     'END STATEMENT'
36
37 THE NEWLY DEFINED STATEMENT HAS THE FORM
38
39     KEYWORD LIST OF EXPRESSIONS
40
41 EACH EXPRESSION WHICH WILL OCCUR IN AN INSTANCE OF THE DEFINED
42 STATEMENT IS REPRESENTED BY A VARIABLE IN THE PROTOTYPE OF THE DEFINE
43 STATEMENT. IN THE CODE GENERATED FOR THE STATEMENT THE CODE FOR
44 THE EXPRESSIONS WILL BE GENERATED FIRST, FOLLOWED BY THE CODE FOR THE
45 MAD/I STATEMENTS SPECIFIED IN THE DEFINITION. ALL OCCURENCES OF THE
46 VARIABLES CORRESPONDING TO THE EXPRESSIONS ARE REPLACED BY THE
47 RESULT OF THE CORRESPONDING EXPRESSION.
48 >>
49 'DECLARE SYNTACTIC CLASS' 'DEFINE STATEMENT' 'SAME AS' 'DEFINE';
50 'DEFINE';          <<ESCAPE INTO THE METALANGUAGE>>
51 CREATE,DEFSTATLAB,DFSTLBC000,PERCLS(@VAL); <<SETS UP CRS>>
52 MACRO,'DEFINE STATEMENT';
53 LOCAL,KEYWD,VLIST,SLIST,NVLIST,I,Q;
54 GTDSX;           << GET THE KEYWORD>>
55 SETUP,DSX,14,3,4; <<MAKE IT SIMPLE STATEMENT >>
56 SET,KEYWD,0,DSX; <<KEYWD POINTS TO THE KEYWORD>>
57 SET,KEYWD,1,INDCLS(@VAL); <<AND IT IS INDIRECT>>
58 A: GTDSX;        << GET A VARIABLE NAME>>

```

```

59      JMP,B,DSX = LN; ;                <<SEMICOLON MARKS THE END>>
60      SETLST,VLIST,,DSX;              << PUT THE VARIABLE INTO VLIST>>
61      GTDSX;                          << GET THE COMMA OR SEMICOLON>>
62      JMP,A,DSX = LN; ;                <<THERE IS ANOTHER VARIABLE>>
63      B:  CRS,DEFSTATLAB,NOTIND.(DEFSTATLAB); << A NEW LIST TO HOLD STM>>
64      SET,SLIST,0,DEFSTATLAB;         <<SAVE THE LIST NAME>>
65      SET,SLIST,1,INDCLS(@VAL);       << MAKE IT INDIRECT>>
66      CRS,DEFSTATLAB,NOTIND.(DEFSTATLAB); <<UNIQUE SYM FOR HIS VAR>>
67      SETLST,NVLIST,,DEFSTATLAB;      <<BUILD A LIST OF THEM>>
68      JMP,LOC-2,NVLIST(0) < VLIST(0); <<WANT ONE FOR EACH VARIABLE>>
69      C:  GTDSX;                       <<READ A STATEMENT DSX>>
70      JMP,D,DSX = 'END STATEMENT';    <<CHECK FOR END OF DEFINITION>>
71      SET,I,0,1;                      <<NOW CHECK FOR IT IN THE VARIABLES>>
72      JMP,E,VLIST(I(0)) = DSX;        <<JUMP IF FOUND IT>>
73      SET,I,0,I(0)+1;                <<UP THE ANTE>>
74      JMP,LOC-2,I(0) <= VLIST(0);    <<CONTINUE IF STILL MORE VARIABLES>>
75      SETLST,SLIST,,DSX;             << ADD HIS DSX TO THE STREAM>>
76      JMP,C;                          << PROCESS THE NEXT DSX>>
77      E:  SET,NOTIND.(DSX),0,NVLIST(I(0)); <<PERFORM THE SUBSTITUTION>>
78      JMP,LOC-3;                      << AND NOW DO IT TO THIS>>
79      D:  SETLST,SLIST,, 'END';       <<PUT A 'END' ON THE END >>
80      SETLST,SLIST,,LN;;             << AND FINISH UP WITH A SEMICOLON >>
81      MACRO,KEYWD;                   <<NOW DEFINE THE MACRO FOR THE KEYWD>>
82      DESIST;
83      SET,I,0,1;                      <<GENERATE AN EXPRESSION SCAN FOR EACH VAR>>
84      JMP,LOC+10,I(0) > VLIST(0);    << STOP IF END OF LIST>>
85      SET,NOTIND.(Q),C,NVLIST(I(0)); <<Q WILL BE THE VARIABLE NAME>>
86      SET,NOTIND.(Q),1,INDCLS(@VAL);
87      RESUME;
88      JSCAN,.TAG.,LIST;               <<SCAN OFF AN EXPRESSION>>
89      SET,Q,0,EXP.(1);               <<MAKE THE VARIABLE THE RESULT>>
90      SET,Q,1,INDCLS(@VAL);
91      DESIST;
92      SET,I,0,I(0)+1;                <<UP TO THE NEXT VARIABLE>>
93      JMP,LCC-9;                     << AROUND AND AROUND>>
94      RESUME;
95      SET,GTDSXLIST,0,SLIST;         << REDEFINE GTDSX FOR AWHILE >>
96      BLOCK;                         << THESE LOOK LIKE SCOPE OF STM>>
97      MEND,KEYWD;                    <<END THE GENERATED MACRO>>
98      GTDSX;                         << GET OUR SEMICOLON>>
99      MEND,'DEFINE STATEMENT';
100     END;                            <<BACK INTO MAD/I>>
101     <<
102     THE FOLLOWING REDEFINES THE PSEUDO OP FOR GTDSX.  THE NEW GTDSX ACTS
103     EXACTLY LIKE THE OLD DSX EXCEPT THAT IT WILL INSERT THE CONTENTS
104     OF THE LIST POINTED TO BY GTDSXLIST INTO THE INPUT STREAM.  THIS
105     ALLOWS A SEQUENCE OF DESCRIPTORS TO BE RETURNED TO JSCAN AS IF THEY
106     CAME FROM THE INPUT STREAM
107     >>
108     'DEFINE';
109     SET,PREVGTDSX,0,GTDSX(0);        << THIS IS NOW THE REAL GTDSX >>
110     SET,PREVGTDSX,1,GTDSX(1);
111     SET,GTDSXLIST,@SIZE,0;          << INITIALIZE GTDSX CHEAT LIST >>
112     SET,GTDSXLIST,C,0;              << USE REAL GTDSX FOR THE TIME BEING >>
113     MACRO,GTDSX;                    << NOW REDEFINE GTDSX >>
114     JMP,LCC+3,GTDSXLIST(0) = 0;    << JUMP IF INSERTIONS TO BE MADE >>
115     PREVGTDSX;                      << USE THE OLD-FASHION GTDSX >>
116     MEXIT;
117     SET,GTDSXLIST,@SIZE,GTDSXLIST(@SIZE)+1; << UP THE LIST INDEX >>
118     JMP,LOC+5,GTDSXLIST(@SIZE) <= (IND.(GTDSXLIST(0)))(0);

```

```

119      SET,GTDSXLIST,@SIZE,0;          << LIST PROCESSED SO RESET >>
120      SET,GTDSXLIST,0,0;
121      GTDSX;                          << NOW TRY READING AGAIN >>
122      MEXIT;
123      SET,NOTIND.(DSX),0,DSXCF.(((IND.(GTDSXLIST(0)))(GTDSXLIST(@SIZE))));
124      SET,NOTIND.(DSX),1,INDCLS(0);    << SET UP DSX >>
125      MEND,GTDSX;
126      END;
127  <<
128      THE FOLLOWING MACRO CREATES A FLOATING ADDRESS CORRESPONDING TO
129      EACH PARAMETER
130  >>
131      *DEFINE*;
132      MACRO,FLAD;
133      LOCAL,I;
134      SET,I,0,1;
135      JMP,LOC+2,PAR.(0) >= I(0);
136      MEXIT;
137      CRS,FLD,NOTIND.(FLD);
138      SET,FLD,1,NOTIND.(FLD)(@MOD);
139      SET,PAR.(I(0)),0,FLD;
140      SET,PAR.(I(0)),1,INDCLS(0);
141      SET,I,0,I(0)+1;
142      JMP,LOC-7;
143      MEND,FLAD;
144      END;
145  <<
146      THE FOLLOWING MACRO CREATES A TEMPORARY CORRESPONDING TO EACH PARAMETER
147  >>
148      *DEFINE*;
149      MACRO,TEMPORARY;
150      LOCAL,I;
151      SET,I,0,1;
152      JMP,LOC+2,PAR.(0) >= I(0);
153      MEXIT;
154      CRS,TMP,NOTIND.(TMP);
155      SET,TMP,1,NOTIND.(TMP)(@MOD);
156      SET,PAR.(I(0)),0,TMP;
157      SET,PAR.(I(0)),1,INDCLS(0);
158      SET,I,0,I(0)+1;
159      JMP,LOC-7;
160      MEND,TEMPORARY;
161      END;
162  <<
163      THE FOLLOWING REDEFINES THE GETTEMP AND FREETEMP MACROS TO
164      GET AROUND THE PROBLEM OF THE REASSIGNMENT OF THE STORAGE
165      ALLOCATION OF A TEMPORARY.
166  >>
167      *DEFINE*;
168      MACEXCTYPE,7;
169      MACDEFTYPE,2;
170      POPMACRO,GETTEMP,FREETEMP;
171      ATR,@TEMP1,EXTENDED;
172      ATR,@TEMP2,LOCAL,20,4;
173      SET,TEMPLST,@VAL,0;
174
175      MACRO,GETTEMP,S,MODE,LEN;
176      LOCAL,I,J;
177      JMP,LOC+4,LEN <= 16;
178  ERR: MNOTE,"**** GETTEMP CALLED FOR MORE THAN SIXTEEN BYTES.";

```

```

179          DUMPDSX,S,MODE,LEN;
180          MEXIT;
181          CLEAR,S;
182          SET,I,@VAL,5;
183          SET,J,@VAL,0;
184 TEST:     JMP,DCNE,I > TEMPLST;
185          JMP,LCC+3,TEMPLST(I)(@TEMP2) = 0;
186          SET,J,@VAL,I;
187          JMP,SEARCH;
188          JMP,SEARCH,TEMPLST(I)(@TEMP2) = DSXOF.(S);
189 GOOD:     SET,TEMPLST(I),@TEMP2,DSXOF.(S);
190          SET,S,@XA,1;
191          SET,S,@MODE,MODE(@MODE);
192          SET,S,@LEN,LEN;
193          SET,S,@VAL,TEMPLST(I);
194          MEXIT;
195 SEARCH:  SET,I,@VAL,I+1;
196          JMP,TEST;
197 DONE:    JMP,NO,J = 0;
198          SET,I,@VAL,J;
199          JMP,GOOD;
200 NO:      APND,TEMPLST,1,I;
201          SET,TEMPLST,@VAL,I;
202          SPACE,TEMPLST(I),16,8;
203          JMP,GOOD;
204          MEND,GETTEMP;
205
206          MACRO,FREETEMP,S;
207          LOCAL,I;
208          RELSYMB,S;
209          SET,I,@VAL,5;
210          JMP,NC,I > TEMPLST;
211          JMP,GCTIT,TEMPLST(I)(@TEMP2) = DSXOF.(S);
212          SET,I,@VAL,I+1;
213          JMP,LCC-3;
214 GOTIT:   SET,TEMPLST(I),@TEMP2,0;
215 NU:     SET,S,@VAL,0;
216          SET,S,@XA,1;
217          SET,S,@SIZE,0;
218          SET,S,@CLS,NOT IND.(TMP)(@MOD);
219          MEND,FREETEMP;
220
221          MACDEFTYPE,1;
222          MACEXCTYPE,1;
223          END;
END OF FILE

```



Appendix D. Contents of File -DATA

```
$LIST -DATA
 1 DISPLAY
 2 POINT
 3 10 10
 4 POINT
 5 10 40
 6 POINT
 7 40 10
 8 POINT
 9 40 40
10 DISPLAY
11 LINE
12 1 2
13 LINE
14 1 3
15 LINE
16 2 4
17 LINE
18 3 4
19 DISPLAY
20 MOVE
21 1 15 20
22 DISPLAY
23 PICTURE
24 1
25 2
26 3
27 4
28 0
29 MOVE
30 1 20 20
31 DISPLAY
END OF FILE
```

Appendix E. Compilation of the MAD/I Program

```
$CREATE SKETCHOBJ  
FILE ALREADY EXISTS  
$EMPTY SKETCHOBJ  
DONE.  
$RUN *MAD1 SCARDS=SKETCH SPUNCH=SKETCHOBJ PAR=L  
EXECUTION BEGINS
```

MAD/I COMPILER OPTION ASSIGNMENTS:

SOURCE, DECK, LIST, SORMGIN=(001,256), FREEFORM, CONTCHAR=+  
SOURCETAB=006, SIZE=(0003,0255), COMPILER  
NOMAP, NOXREF, ATR, OPLIST, USER, NOADDENDA

MAD/I COMPILER VERSION AN049-134322.

MAD/I COMPILER SOURCE PROGRAM LISTING ... ..

```
*0001      'PROCEDURE' SKETCH.;
*0002
*0003      'INCLUDE' 'DISPLAYDEF'
*0004 <<
*0005          DISPLAY DEFINITIONAL PACKAGE
*0006
*0007      THE FOLLOWING PARAMETER STATEMENT DEFINES THE COMPONENT STRUCTURE
*0008      FOR A POINT.  THE USER SIMPLY USES 'POINT' AS A MODE NAME IN
*0009      EITHER A 'DECLARE' STATEMENT OR AN @ CONSTRUCTION.  NOTE THAT
*0010      THE STATEMENT
*0011          'POINT' (VARIABLE LIST)
*0012      WILL NOT WORK BECAUSE THIS IS SIMPLY A PARAMETRIC SUBSTITUTION.
*0013      THE STRUCTURE OF A POINT IS
*0014          1    POINTER          POINTER TO NEXT LIST ITEM
*0015          2    PCINTER         POINTER TO NEXT ITEM IN SAME PICTURE
*0016          3    INTEGER SHORT   DISPLAY ITEM NUMBER
*0017          4    INTEGER SHORT   DISPLAY ITEM TYPE (1 FOR A POINT)
*0018          5    INTEGER SHORT   X COORDINATE OF THE POINT
*0019          6    INTEGER SHORT   Y COORDINATE OF THE POINT
*0020 >>
*0021      'PARAMETER' 'POINT' 'BASED' 'COMPONENT STRUCTURE' ('POINTER',
*0022          'POINTER', 'INTEGER SHORT', 'INTEGER SHORT', 'INTEGER SHORT',
*0023          'INTEGER SHORT') 'ENDP'
*0024 <<
*0025      THE FOLLOWING PARAMETER STATEMENT DEFINES THE COMPONENT STRUCTURE
*0026      FOR A LINE.  THE USER SIMPLY USES 'LINE' AS A MODE NAME IN EITHER A
*0027      'DECLARE' STATEMENT OR AN @ CONSTRUCTION.  NOTE THAT THE STATEMENT
*0028          'LINE' (VARIABLE LIST)
*0029      WILL NOT WORK BECAUSE THIS IS SIMPLY A PARAMETRIC SUBSTITUTION.
*0030      THE STRUCTURE OF A LINE IS
*0031          1    POINTER          POINTER TO THE NEXT LIST ITEM
*0032          2    POINTER          POINTER TO THE NEXT ITEM IN THE PICTURE
*0033          3    INTEGER SHORT   DISPLAY ITEM NUMBER
*0034          4    INTEGER SHORT   DISPLAY ITEM TYPE (2 FOR A LINE)
*0035          5    POINTER          POINTER TO FIRST END-POINT
*0036          6    POINTER          POINTER TO SECOND END-POINT
*0037 >>
*0038      'PARAMETER' 'LINE' 'BASED' 'COMPONENT STRUCTURE' ('POINTER',
*0039          'POINTER', 'INTEGER SHORT', 'INTEGER SHORT', 'POINTER',
*0040          'POINTER') 'ENDP'
*0041 <<
*0042      THE FOLLOWING PARAMETER STATEMENT DEFINES THE MODE OF A PICTURE.
*0043      A PICTURE IS A POINTER TO ONE OF THE ITEMS IN THE PICTURE.  HENCE
*0044      IT IS SIMPLY PARAMETERIZED TO 'POINTER' MODE.
*0045 >>
*0046      'PARAMETER' 'PICTURE' 'POINTER' 'ENDP'
*0047 <<
*0048      THE FOLLOWING STATEMENTS DEFINE THE GLOBAL VARIABLES USED BY
*0049      THE DEFINITION PACKAGE.
```

```

*0050 >>
*0051 'DECLARE' DISPLAYN 'INTEGER SHORT';
*0052 'PRESET' DISPLAYN := 0;
*0053 'DECLARE' QQSV 'POINT';
*0054 'DECLARE' HEAD 'POINTER';
*0055 'PRESET' HEAD := 0;
*0056 <<
*0057 THE FOLLOWING BRINGS IN THE DEFINE FACILITY FROM THE FILE DEFFACILITY
*0058 >>
*0059 'INCLUDE' "DEFFACILITY"
*0060 <<
*0061 THE FOLLOWING DEFINES THE NEW SIMPLE STATEMENT
*0062
*0063 'DECLARE SYNTACTIC CLASS' A 'SAME AS' B
*0064
*0065 WHICH GIVES THE SYMBOL A EXACTLY THE SAME SYNTACTIC DEFINITION
*0066 (CLASS,SYNTACTIC CLASS,LEFT PRECEDENCE,RIGHT PRECEDENCE) AS B.
*0067 A WILL NOW BE TREATED EXACTLY AS B BY THE SYNTACTIC SCANNER.
*0068 >>
*0069 'DEFINE'; <<ESCAPE INTO THE MACRO LANGUAGE>>
*0070 SETUP,'DECLARE SYNTACTIC CLASS',14,3,4; << SIMPLE KEYWORD>>
*0071 MACRO,'DECLARE SYNTACTIC CLASS'; <<START THE MACRO DEF>>
*0072 LOCAL,A; << A IS SAVED IN THIS LOCAL>>
*0073 GTDSX; << THIS READS A>>
*0074 SET,A,0,DSX; <<SAVE DSX OF A IN LOCAL SYMBOL>>
*0075 GTDSX; <<NOW READ 'SAME AS'>>
*0076 JMP,LOC+3,DSX = 'SAMEAS'; <<BE SURE IT REALLY IS 'SAMEAS'>>
*0077 MNOTE,'***** ONLY 'SAME AS' IS ALLOWED';
*0078 MEXIT; <<LEAVE HIM AT THE MERCY OF JSCAN>>
*0079 GTDSX; <<READ B>>
*0080 SET,IND.(A),1,DSX(1); <<SET THE CLASS CODE OF A>>
*0081 SET,IND.(A),2,DSX(2); <<SET THE SYNTACTIC CLASS OF A>>
*0082 SET,IND.(A),3,DSX(3); <<SET THE LEFT PRECEDENCE OF A>>
*0083 SET,IND.(A),4,DSX(4); <<SET THE RIGHT PRECEDENCE OF A>>
*0084 GTDSX; <<READ THE STATEMENT TERMINATOR>>
*0085 MEND,'DECLARE SYNTACTIC CLASS'; <<ALL DONE>>
*0086 END; <<BACK INTO MAD/I>>
*0087 <<
*0088 THE FOLLOWING DEFINES A SPECIAL PURPOSE DEFINITIONAL FACILITY
*0089 WHICH ALLOWS THE DEFINITION OF A SIMPLE STATEMENT. THE DEFINITIONAL
*0090 STATEMENT IS OF THE FORM
*0091
*0092 'DEFINE STATEMENT' KEYWORD LIST OF VARIABLES
*0093 LIST OF STATEMENTS
*0094 'END STATEMENT'
*0095
*0096 THE NEWLY DEFINED STATEMENT HAS THE FORM
*0097
*0098 KEYWORD LIST OF EXPRESSIONS
*0099
*0100 EACH EXPRESSION WHICH WILL OCCUR IN AN INSTANCE OF THE DEFINED
*0101 STATEMENT IS REPRESENTED BY A VARIABLE IN THE PROTOTYPE OF THE DEFINE
*0102 STATEMENT. IN THE CODE GENERATED FOR THE STATEMENT THE CODE FOR
*0103 THE EXPRESSIONS WILL BE GENERATED FIRST, FOLLOWED BY THE CODE FOR THE
*0104 MAD/I STATEMENTS SPECIFIED IN THE DEFINITION. ALL OCCURENCES OF THE
*0105 VARIABLES CORRESPONDING TO THE EXPRESSIONS ARE REPLACED BY THE
*0106 RESULT OF THE CORRESPONDING EXPRESSION.
*0107 >>
*0108 'DECLARE SYNTACTIC CLASS' 'DEFINE STATEMENT' 'SAME AS' 'DEFINE';
*0109 'DEFINE'; <<ESCAPE INTO THE METALANGUAGE>>

```

```

*0110 CREATE,DEFSTATLAB,DFSTLBOOOO,PERCLS(@VAL); <<SETS UP CRS>>
*0111 MACRO,'DEFINE STATEMENT';
*0112 LOCAL,KEYWD,VLIST,SLIST,NVLIST,I,Q;
*0113 GTDSX; << GET THE KEYWORD>>
*0114 SETUP,DSX,14,3,4; <<MAKE IT SIMPLE STATEMENT >>
*0115 SET,KEYWD,0,DSX; <<KEYWD POINTS TO THE KEYWORD>>
*0116 SET,KEYWD,1,INDCLS(@VAL); <<AND IT IS INDIRECT>>
*0117 A: GTDSX; << GET A VARIABLE NAME>>
*0118 JMP,B,DSX = LN; ; <<SEMICOLON MARKS THE END>>
*0119 SETLST,VLIST,,DSX; << PUT THE VARIABLE INTO VLIST>>
*0120 GTDSX; << GET THE COMMA OR SEMICOLON>>
*0121 JMP,A,DSX ^= LN; ; <<THERE IS ANOTHER VARIABLE>>
*0122 B: CRS,DEFSTATLAB,NOTIND.(DEFSTATLAB); << A NEW LIST TO HOLD STM>>
*0123 SET,SLIST,0,DEFSTATLAB; <<SAVE THE LIST NAME>>
*0124 SET,SLIST,1,INDCLS(@VAL); << MAKE IT INDIRECT>>
*0125 CRS,DEFSTATLAB,NOTIND.(DEFSTATLAB); <<UNIQUE SYM FOR HIS VAR>>
*0126 SETLST,NVLIST,,DEFSTATLAB; <<BUILD A LIST OF THEM>>
*0127 JMP,LOC-2,NVLIST(0) < VLIST(0); <<WANT ONE FOR EACH VARIABLE>>
*0128 C: GTDSX; <<READ A STATEMENT DSX>>
*0129 JMP,D,DSX = 'END STATEMENT'; <<CHECK FOR END OF DEFINITION>>
*0130 SET,I,0,1; <<NOW CHECK FOR IT IN THE VARIABLES>>
*0131 JMP,E,VLIST(I(0)) = DSX; <<JUMP IF FOUND IT>>
*0132 SET,I,0,I(0)+1; <<UP THE ANTE>>
*0133 JMP,LOC-2,I(0) <= VLIST(0); <<CONTINUE IF STILL MORE VARIABLES>>
*0134 SETLST,SLIST,,DSX; << ADD HIS DSX TO THE STREAM>>
*0135 JMP,C; << PROCESS THE NEXT DSX>>
*0136 E: SET,NOTIND.(DSX),0,NVLIST(I(0)); <<PERFORM THE SUBSTITUTION>>
*0137 JMP,LOC-3; << AND NOW DO IT TO THIS>>
*0138 D: SETLST,SLIST,, 'END'; <<PUT A 'END' ON THE END >>
*0139 SETLST,SLIST,,LN; << AND FINISH UP WITH A SEMICOLON >>
*0140 MACRO,KEYWD; <<NOW DEFINE THE MACRO FOR THE KEYWD>>
*0141 DESIST;
*0142 SET,I,0,1; <<GENERATE AN EXPRESSION SCAN FOR EACH VAR>>
*0143 JMP,LOC+10,I(0) > VLIST(0); << STOP IF END OF LIST>>
*0144 SET,NOTIND.(Q),0,NVLIST(I(0)); <<Q WILL BE THE VARIABLE NAME>>
*0145 SET,NOTIND.(Q),1,INDCLS(@VAL);
*0146 RESUME;
*0147 JSCAN,.TAG.,LIST; <<SCAN OFF AN EXPRESSION>>
*0148 SET,Q,0,EXP.(1); <<MAKE THE VARIABLE THE RESULT>>
*0149 SET,Q,1,INDCLS(@VAL);
*0150 DESIST;
*0151 SET,I,0,I(0)+1; <<UP TO THE NEXT VARIABLE>>
*0152 JMP,LOC-9; << AROUND AND AROUND>>
*0153 RESUME;
*0154 SET,GTDSXLIST,0,SLIST; << REDEFINE GTDSX FOR AWHILE >>
*0155 BLOCK; << THESE LOOK LIKE SCOPE OF STM>>
*0156 MEND,KEYWD; <<END THE GENERATED MACRO>>
*0157 GTDSX; << GET OUR SEMICOLON>>
*0158 MEND,'DEFINE STATEMENT';
*0159 END; <<BACK INTO MAD/I>>
*0160 <<
*0161 THE FOLLOWING REDEFINES THE PSEUDO OP FOR GTDSX. THE NEW GTDSX ACTS
*0162 EXACTLY LIKE THE OLD DSX EXCEPT THAT IT WILL INSERT THE CONTENTS
*0163 OF THE LIST POINTED TO BY GTDSXLIST INTO THE INPUT STREAM. THIS
*0164 ALLOWS A SEQUENCE OF DESCRIPTORS TO BE RETURNED TO JSCAN AS IF THEY
*0165 CAME FROM THE INPUT STREAM
*0166 >>
*0167 'DEFINE';
*0168 SET,PREVGTDSX,0,GTDSX(0); << THIS IS NOW THE REAL GTDSX >>
*0169 SET,PREVGTDSX,1,GTDSX(1);

```

```

*0170 SET,GTDSXLIST,@SIZE,0; << INITIALIZE GTDSX CHEAT LIST >>
*0171 SET,GTDSXLIST,0,0; << USE REAL GTDSX FOR THE TIME BEING >>
*0172 MACRO,GTDSX; << NOW REDEFINE GTDSX >>
*0173 JMP,LOC+3,GTDSXLIST(0) <=> 0; << JUMP IF INSERTIONS TO BE MADE >>
*0174 PREVGTDSX; << USE THE OLD-FASHION GTDSX >>
*0175 MEXIT;
*0176 SET,GTDSXLIST,@SIZE,GTDSXLIST(@SIZE)+1; << UP THE LIST INDEX >>
*0177 JMP,LOC+5,GTDSXLIST(@SIZE) <=> (IND.(GTDSXLIST(0)))(0);
*0178 SET,GTDSXLIST,@SIZE,0; << LIST PROCESSED SO RESET >>
*0179 SET,GTDSXLIST,0,0;
*0180 GTDSX; << NOW TRY READING AGAIN >>
*0181 MEXIT;
*0182 SET,NOTIND.(DSX),0,DSXOF.(((IND.(GTDSXLIST(0)))(GTDSXLIST(@SIZE))));
*0183 SET,NOTIND.(DSX),1,INDCLS(0); << SET UP DSX >>
*0184 MEND,GTDSX;
*0185 END;
*0186 <<
*0187 THE FOLLOWING MACRO CREATES A FLOATING ADDRESS CORRESPONDING TO
*0188 EACH PARAMETER
*0189 >>
*0190 'DEFINE';
*0191 MACRO,FLAD;
*0192 LOCAL,I;
*0193 SET,I,0,1;
*0194 JMP,LOC+2,PAR.(0) >= I(0);
*0195 MEXIT;
*0196 CRS,FLD,NOTIND.(FLD);
*0197 SET,FLD,1,NOTIND.(FLD)(@MOD);
*0198 SET,PAR.(I(0)),0,FLD;
*0199 SET,PAR.(I(0)),1,INDCLS(0);
*0200 SET,I,0,I(0)+1;
*0201 JMP,LOC-7;
*0202 MEND,FLAD;
*0203 END;
*0204 <<
*0205 THE FOLLOWING MACRO CREATES A TEMPORARY CORRESPONDING TO EACH PARAMETER
*0206 >>
*0207 'DEFINE';
*0208 MACRO,TEMPCRARY;
*0209 LOCAL,I;
*0210 SET,I,0,1;
*0211 JMP,LOC+2,PAR.(0) >= I(0);
*0212 MEXIT;
*0213 CRS,TMP,NOTIND.(TMP);
*0214 SET,TMP,1,NOTIND.(TMP)(@MOD);
*0215 SET,PAR.(I(0)),0,TMP;
*0216 SET,PAR.(I(0)),1,INDCLS(0);
*0217 SET,I,0,I(0)+1;
*0218 JMP,LOC-7;
*0219 MEND,TEMPORARY;
*0220 END;
*0221 <<
*0222 THE FOLLOWING REDEFINES THE GETTEMP AND FREETEMP MACROS TO
*0223 GET AROUND THE PROBLEM OF THE REASSIGNMENT OF THE STORAGE
*0224 ALLOCATION OF A TEMPORARY.
*0225 >>
*0226 'DEFINE';
*0227 MACEXCTYPE,7;
*0228 MACDEFTYPE,2;
*0229 POPMACRO,GETTEMP,FREETEMP;

```

```

*0230      ATR,@TEMP1,EXTENDED;
*0231      ATR,@TEMP2,LOCAL,20,4;
*0232      SET,TEMPLST,@VAL,0;
*0233
*0234      MACRO,GETTEMP,S,MODE,LEN;
*0235      LOCAL,I,J;
*0236      JMP,LOC+4,LEN <= 16;
*0237 ERR:  MNOTE,"**** GETTEMP CALLED FOR MORE THAN SIXTEEN BYTES.";
*0238      DUMPDSX,S,MODE,LEN;
*0239      MEXIT;
*0240      CLEAR,S;
*0241      SET,I,@VAL,5;
*0242      SET,J,@VAL,0;
*0243 TEST:  JMP,DONE,I > TEMPLST;
*0244      JMP,LOC+3,TEMPLST(I)(@TEMP2) = 0;
*0245      SET,J,@VAL,I;
*0246      JMP,SEARCH;
*0247      JMP,SEARCH,TEMPLST(I)(@TEMP2) = DSXOF.(S);
*0248 GOOD: SET,TEMPLST(I),@TEMP2,DSXOF.(S);
*0249      SET,S,@XA,1;
*0250      SET,S,@MODE,MODE(@MODE);
*0251      SET,S,@LEN,LEN;
*0252      SET,S,@VAL,TEMPLST(I);
*0253      MEXIT;
*0254 SEARCH: SET,I,@VAL,I+1;
*0255      JMP,TEST;
*0256 DONE:  JMP,NO,J = 0;
*0257      SET,I,@VAL,J;
*0258      JMP,GOOD;
*0259 NU:    APND,TEMPLST,1,I;
*0260      SET,TEMPLST,@VAL,I;
*0261      SPACE,TEMPLST(I),16,8;
*0262      JMP,GOOD;
*0263      MEND,GETTEMP;
*0264
*0265      MACRO,FREETEMP,S;
*0266      LOCAL,I;
*0267      RELSYMB,S;
*0268      SET,I,@VAL,5;
*0269      JMP,NO,I > TEMPLST;
*0270      JMP,GUTIT,TEMPLST(I)(@TEMP2) = DSXOF.(S);
*0271      SET,I,@VAL,I+1;
*0272      JMP,LOC-3;
*0273 GUTIT: SET,TEMPLST(I),@TEMP2,0;
*0274 NO:    SET,S,@VAL,0;
*0275      SET,S,@XA,1;
*0276      SET,S,@SIZE,0;
*0277      SET,S,@CLS,NOTIND.(TMP)(@MOD);
*0278      MEND,FREETEMP;
*0279
*0280      MACDEFTYPE,1;
*0281      MACEXCTYPE,1;
*0282      END;
*0283 <<
*0284      THE FOLLOWING DEFINES THE STATEMENT
*0285
*0286          'CREATE POINT' POINT,X,Y
*0287
*0288      WHICH HAS THE EFFECT OF THE LIST OF STATEMENTS SHOWN
*0289 >>

```

```

*0290      'DEFINE STATEMENT' 'CREATE POINT' POINT,X,Y;
*0291          'ALLOCATE' POINT;
*0292          POINT(1) := HEAD;
*0293          HEAD := .PT. POINT;
*0294          POINT(2) := (0 .AS. ('PCINTER'));
*0295          DISPLAYN := DISPLAYN+1;
*0296          POINT(3) := DISPLAYN;
*0297          POINT(4) := 1;
*0298          POINT(5) := X;
*0299          POINT(6) := Y;
*0300      'END STATEMENT';
*0301  <<
*0302      THE FOLLOWING DEFINES THE STATEMENT
*0303
*0304          'CREATE LINE' LINE,P1,P2
*0305
*0306      WHICH HAS THE SAME EFFECT AS THE LIST OF STATEMENTS SHOWN
*0307  >>
*0308      'DEFINE STATEMENT' 'CREATE LINE' LINE,P1,P2;
*0309          'ALLOCATE' LINE;
*0310          LINE(1) := HEAD;
*0311          HEAD := .PT. LINE;
*0312          LINE(2) := (0 .AS. ('PCINTER'));
*0313          DISPLAYN := DISPLAYN+1;
*0314          LINE(3) := DISPLAYN;
*0315          LINE(4) := 2;
*0316          LINE(5) := .PT. P1;
*0317          LINE(6) := .PT. P2;
*0318      'END STATEMENT';
*0319  <<
*0320      THE FOLLOWING DEFINES THE STATEMENT
*0321
*0322          'CONNECT' POINT 'TC' PICTURE
*0323
*0324      WHICH HAS THE SAME EFFECT AS THE LIST OF STATEMENT SHOWN
*0325  >>
*0326      'DECLARE SYNTACTIC CLASS' 'TO' 'SAME AS' , ;
*0327      'DEFINE STATEMENT' 'CONNECT' POINT 'TO' PICTURE;
*0328          QQ1 := PICTURE;
*0329          'IF' (QQ1 .AS. ('INTEGER')) = 0;
*0330              PICTURE := .PT. POINT;
*0331              POINT(2) := .PT. POINT;
*0332          'ELSE';
*0333              QQSV .EVAL. PICTURE;
*0334              POINT(2) := QQSV(2);
*0335              QQSV(2) := .PT. POINT;
*0336          'END';
*0337      'END STATEMENT';
*0338  <<
*0339      THE FOLLOWING MACRO DEFINES THE .POINT. OPERATOR AS A PASS ONE MACRO.
*0340      IT CAUSES .POINT. A TO BE TREATED AS A(4) = 1
*0341  >>
*0342      'DECLARE SYNTACTIC CLASS' .POINT. 'SAME AS' .ABS.;
*0343      'DEFINE';
*0344      MACRO, .POINT., T, B;
*0345      LOCAL, U;
*0346      TEMPORARY, U;
*0347      .TAG., U, B, 'LOCALLITERAL' 4;
*0348      LN=, T, U, 'LOCALLITERAL' 1;
*0349      MEND, .POINT.;

```



```

*0350      END;
*0351 <<
*0352      THE FOLLOWING MACRO DEFINES .XOF. AS A PASS ONE OPERATOR.
*0353      IT CAUSES .XOF. A TO BE TREATED AS A(5)
*0354 >>
*0355      'DECLARE SYNTACTIC CLASS' .XOF. 'SAME AS' .ABS.;
*0356      'DEFINE';
*0357      MACRO,.XOF.,T,A;
*0358      .TAG.,T,A,'LOCALLITERAL' 5;
*0359      MEND,.XOF.;
*0360      END;
*0361 <<
*0362      THE FOLLOWING CAUSES .YOF. TO BE DEFINED AS A PASS ONE OPERATOR
*0363      .YOF. A HAS THE SAME EFFECT AS A(6)
*0364 >>
*0365      'DECLARE SYNTACTIC CLASS' .YCF. 'SAME AS' .ABS.;
*0366      'DEFINE';
*0367      MACRO,.YOF.,T,A;
*0368      .TAG.,T,A,'LOCALLITERAL' 6;
*0369      MEND,.YOF.;
*0370      END;
*0371 <<
*0372      THE FOLLOWING DEFINES .NEXT. A TO BE THE SAME AS A(2)
*0373 >>
*0374      'DECLARE SYNTACTIC CLASS' .NEXT. 'SAME AS' .ABS.;
*0375      'DEFINE';
*0376      MACRO,.NEXT.,T,A;
*0377      .TAG.,T,A,'LOCALLITERAL' 2;
*0378      MEND,.NEXT.;
*0379      END;
*0380 <<
*0381      THE FOLLOWING DEFINES .DISPN. A TO BE THE SAME AS A(3)
*0382 >>
*0383      'DECLARE SYNTACTIC CLASS' .DISPN. 'SAME AS' .ABS.;
*0384      'DEFINE';
*0385      MACRO,.DISPN.,T,A;
*0386      .TAG.,T,A,'LOCALLITERAL' 3;
*0387      MEND,.DISPN.;
*0388      END;
*0389      'DECLARE SYNTACTIC CLASS' .ENCA. 'SAME AS' .ABS.;
*0390      'DEFINE';
*0391      MACRO,.ENCA.,T,A;
*0392      .TAG.,T,A,'LOCALLITERAL' 5;
*0393      MEND,.ENCA.;
*0394      END;
*0395 <<
*0396      THE FOLLOWING DEFINES .ENCB. A TO BE THE SAME AS A(6)
*0397 >>
*0398      'DECLARE SYNTACTIC CLASS' .ENCB. 'SAME AS' .ABS.;
*0399      'DEFINE';
*0400      MACRO,.ENCB.,T,A;
*0401      .TAG.,T,A,'LOCALLITERAL' 6;
*0402      MEND,.ENCB.;
*0403      END;
*0404 <<
*0405      THE FOLLOWING DEFINES .HEAD. A TO BE THE SAME AS A(1)
*0406 >>
*0407      'DECLARE SYNTACTIC CLASS' .HEAD. 'SAME AS' .ABS.;
*0408      'DEFINE';
*0409      MACRO,.HEAD.,T,A;

```

```

*0410      .TAG.,T,A,'LOCALLITERAL' 1;
*0411      MEND,.HEAD.;
*0412      END;
*0413 <<
*0414      THE FOLLOWING DEFINES .ADROF. A TO BE THE SAME AS THE SEQUENCE OF
*0415      STATEMENTS
*0416          QQSV .EVAL. HEAD
*0417      B1:  'IF' .DISPN. QQSV = A, 'GO TO' B2
*0418          QQSV .EVAL. (.HEAD. QQSV)
*0419          'GO TO' B1
*0420      B2:  'RETURN' .PT. QQSV
*0421 >>
*0422      'DECLARE SYNTACTIC CLASS' .ADROF. 'SAME AS' .ABS.;
*0423      'DEFINE';
*0424      MACRO,.ADROF.,T,A;
*0425      LOCAL,B1,B2;
*0426      LOCAL,T1,T2,T3,T4,T5;
*0427      TEMPORARY,T1,T2,T3,T4,T5;
*0428      FLAD,B1,B2;
*0429      .EVAL.,T1,%QQSV,%HEAD;
*0430      DES,B1;
*0431      .DISPN.,T2,%QQSV;
*0432      LN=,T3,T2,A;
*0433      TNZ,B2,T3;
*0434      .HEAD.,T4,%QQSV;
*0435      .EVAL.,T5,%QQSV,T4;
*0436      GOTO,B1;
*0437      DES,B2;
*0438      .PT.,T,%QQSV;
*0439      MEND,.ADROF.;
*0440      END;
*0441 <<
*0442          END OF DEFINITIONAL PACKAGE
*0443 >>
*0444
*0445      'DECLARE' 'NORMAL MODE' 'INTEGER';
*0446      'DECLARE' COMMAND 'CHARACTER'(4);
*0447      'DECLARE' DISPLAY 'FIXED ARRAY'(50,50) 'CHARACTER'(1);
*0448      'DECLARE' (POINT,P1,P2,P3) 'POINT';
*0449      'DECLARE' LINE 'LINE';
*0450      'DECLARE' PICTURE 'FIXED ARRAY'(100) 'PICTURE';
*0451      'DECLARE' QQ1 'PCINTER';
*0452      'DECLARE' QQ2 'PCINTER';
*0453      'DECLARE' (M,M1,M2) 'FLOATING';
*0454
*0455      'PRESET' PICTUREN := 0;
*0456
*0457      SKETCH: 'WRITE' , "'ENTER A COMMAND PLEASE.'*";
*0458      'READ' , "C4.4*", COMMAND;
*0459
*0460
*0461      'IF' COMMAND = "POIN";
*0462          'WRITE' , "' ENTER X AND Y COORDINATES:'*";
*0463          'READ' , "2I*", X,Y;
*0464          'IF' 1 <= X & 50 >= X & 1 <= Y & 50 >= Y;
*0465              'CREATE POINT' POINT,X,Y;
*0466              'WRITE' , "' ASSIGNED DISPLAY NUMBER',HI4*", .DISPN. POINT;
*0467          'ELSE';
*0468              'WRITE' , "' POINT IS OUTSIDE RASTER RANGE.'*";
*0469          'END';

```

```

*0470
*0471      'OR IF' CCOMMAND = "LINE";
*0472          'WRITE' , "" ENTER DISPLAY NUMBERS FOR END-POINTS:"";
*0473          'READ' , "2I*", X,Y;
*0474          P1 .EVAL. .ADROF. X; P2 .EVAL. .ADROF. Y;
*0475          'IF' .POINT. P1 & .POINT. P2;
*0476              'CREATE LINE' LINE,P1,P2;
*0477          'ELSE';
*0478 NOTPOINT:      'WRITE' , "" THOSE ARE NOT POINTS."";
*0479          'END';
*0480
*0481      'OR IF' CCOMMAND = "PICT";
*0482          'WRITE' , "" ENTER DISPLAY NUMBERS FOR ALL POINTS:"";
*0483          PICTUREN := PICTUREN+1;
*0484          'IF' PICTUREN > 100;
*0485              'WRITE' , "" TOO MANY PICTURES:"";
*0486          'ELSE';
*0487              PICTURE(PICTUREN) := 0 .AS. ('POINTER');
*0488 PICTA:          'READ' , "I*", X;
*0489              'IF' X /= 0;
*0490                  P1 .EVAL. .ADROF. X;
*0491                  'IF' -.POINT. P1, 'GO TO' NOTPOINT;
*0492                  QQ2 := PICTURE(PICTUREN);
*0493                  'CONNECT' P1 'TO' QQ2;
*0494                  PICTURE(PICTUREN) := QQ2;
*0495                  'GO TO' PICTA;
*0496              'END';
*0497          'END';
*0498
*0499      'OR IF' CCOMMAND = "MOVE";
*0500          'WRITE' , "" ENTER DISPLAY NUMBER OF POINT AND NEW X,Y:"";
*0501          'READ' , "3I*", DIS,X,Y;
*0502          P1 .EVAL. .ADROF. DIS;
*0503          'IF' -.POINT. P1, 'GCTC' NOTPOINT;
*0504          DX := X-.XOF. P1; DY := Y-.YOF. P1;
*0505          (.XOF. P1) := .XOF. P1+DX; (.YOF. P1) := .YOF. P1+DY;
*0506          QQ1 := P1(2);
*0507          'IF' QQ1 .AS. ('INTEGER') /= 0;
*0508              P2 .EVAL. .NEXT. P1;
*0509 MOVEA:          'IF' .DISPN. P1 /= .DISPN. P2;
*0510                  (.XOF. P2) := .XOF. P2+DX; (.YOF. P2) := .YOF. P2
*0511                      +DY;
*0512                  P2 .EVAL. .NEXT. P2;
*0513                  'GO TO' MOVEA;
*0514              'END';
*0515          'END';
*0516
*0517      'OR IF' CCOMMAND = "DISP";
*0518          'IF' HEAD .AS. ('INTEGER') = 0;
*0519              'WRITE' , "" NOTHING TO DISPLAY:"";
*0520              'GO TO' SKETCH;
*0521          'END';
*0522          'FOR' I := 1,1,I>50, 'FOR' J := 1,1,J>50, DISPLAY(I,J) := " ";
*0523          P1 .EVAL. HEAD;
*0524 DISPA:          ;
*0525              'IF' .POINT. P1;
*0526                  Q1 := .XOF. P1; Q2 := .YOF. P1; DISPLAY(Q2,Q1) := "";
*0527          'ELSE';
*0528              LINE .EVAL. (.PT. P1);
*0529              QQ1 := .ENDA. LINE; P2 .EVAL. QQ1;

```

```

*0530      QQ1 := .END8. LINE; P3 .EVAL. QQ1;
*0531      X1 := MIN.(.XOF. P2,.XOF. P3);
*0532      X2 := MAX.(.XOF. P2,.XOF. P3);
*0533      'IF' X1 = X2;
*0534          Y1 := MIN.(.YOF. P2,.YOF. P3);
*0535          Y2 := MAX.(.YOF. P2,.YOF. P3);
*0536          'FOR' Y := Y1,1, Y > Y2, DISPLAY(Y,X1) := "*";
*0537      'ELSE';
*0538          M1 := .YOF. P3 - .YOF. P2;
*0539          M2 := .XOF. P3 - .XOF. P2;
*0540          M := M1/M2;
*0541          'FOR' X := X1,1,X > X2;
*0542              Q2 := M*(X - .XOF. P2) + .YOF. P2;
*0543              DISPLAY(Q2,X) := "*";
*0544          'END';
*0545      'END';
*0546  'END';
*0547  QQ1 := .HEAD. P1;
*0548  'IF' QQ1 .AS. ('INTEGER') /= 0;
*0549      P1 .EVAL. .HEAD. P1;
*0550      'GO TO' DISPA;
*0551  'END';
*0552  'WRITE' , "1  ",10("  ")#";
*0553  'FOR' I := 50,-1, I < 1,
*0554      'WRITE' , "13,52C1.1*", I," ",",",
*0555      DISPLAY(I,1)...DISPLAY(I,50);
*0556  'WRITE' , "  ",10("  ")#";
*0557
*0558  'ELSE';
*0559      'WRITE' , " ILLEGAL CCOMMAND*";
*0560  'END';
*0561
*0562  'GO TO' SKETCH;
*0563
*0564  'PROCEDURE' MIN.(X,Y);
*0565  'INTEGER SHORT' (X,Y);
*0566  MIN: 'IF' X <= Y, 'RETURN' X;
*0567      'RETURN' Y;
*0568  'END';
*0569
*0570  'PROCEDURE' MAX.(A,B);
*0571  'INTEGER SHORT' (A,B);
*0572  MAX: 'IF' A >= B, 'RETURN' A;
*0573      'RETURN' B;
*0574  'END';
*0575
*0576  'END'

```

ORAGE ALLOCATION

ADDRESS	HEX	HEX	##SKETCH	CSECT
000000				
000470	00000001	+	CCNST	1
000474	00000001	+	CCNST	1
000478	00000064	+	CCNST	100
000480	00000002	+	CCNST	2
000484	00000001	+	CCNST	1
000488	00000032	+	CCNST	50
00048C	00000001	+	CCNST	1
000490	00000032	+	CCNST	50
0000B0	00000000	+	CCNST	0
0000B4	00000000	+	CCNST	0
0000B8	00000000	+	CCNST	0
00029C	7D40C9D3D3C5C7C1	+	CCNST	" ILLEGAL COMMAND.*"
0002AF	7D404040404B7D6B	+	CCNST	" .',10(' .')*"
0002C3	4B	+	CCNST	"."
0002C4	C9F36BF5F2C3F14B	+	CCNST	"I3,52C1.1*"
0002CE	7DF14040404B7D6B	+	CCNST	"1 .',10(' .')*"
0002E2	5C	+	CCNST	"*"
0002E3	40	+	CCNST	" "
0002E4	7D40D5D6E3C8C9D5	+	CCNST	" NOTHING TO DISPLAY.*"
0002FB	C4C9E2D7	+	CCNST	"DISP"
0002FF	F3C95C	+	CCNST	"3I*"
000302	7D40C5D5E3C5D940	+	CCNST	" ENTER DISPLAY NUMBER OF POINT AND NEW X,Y*"
00032F	D4D6E5C5	+	CCNST	"MOVE"
000333	C95C	+	CCNST	"I*"
000335	7D40E3D6D64DD4C1	+	CCNST	" TOO MANY PICTURES.*"
00034C	00000064	+	CCNST	100
000350	7D40C5D5E3C5D940	+	CCNST	" ENTER DISPLAY NUMBERS FOR ALL POINTS.*"
000379	D7C9C3E3	+	CCNST	"PICT"
00037D	7D40E3C8D6E2C540	+	CCNST	" THOSE ARE NOT POINTS.*"
000396	7D40C5D5E3C5D940	+	CCNST	" ENTER DISPLAY NUMBERS FOR END-POINTS.*"
0003BF	D3C9D5C5	+	CCNST	"LINE"
0003C3	7D40D7D6C9D5E340	+	CCNST	" POINT IS OUTSIDE RASTER RANGE.*"
0003E5	7D40C1E2E2C9C7D5	+	CCNST	" ASSIGNED DISPLAY NUMBER',HI4*"
000404	00000032	+	CCNST	50
000408	F2C95C	+	CCNST	"2I*"
00040B	7D40C5D5E3C5D940	+	CCNST	" ENTER X AND Y COORDINATES.*"
000429	D7D6C9D5	+	CCNST	"POIN"
00042D	C3F448F45C	+	CCNST	"C4.4*"
000432	7DF0C5D5E3C5D940	+	CCNST	" CENTER A COMMAND PLEASE.*"
000450	00000006	+	CCNST	6
000454	00000005	+	CCNST	5
000458	00000004	+	CCNST	4
00045C	00000003	+	CCNST	3
000460	00000000	+	CCNST	0
000464	00000002	+	CCNST	2
000468	00000001	+	CCNST	1

STORAGE MAP

```

00 03 0028 00000000 Y
00 03 002C 00000000 X
00 03 0030 00000000 B
00 03 0034 00000000 A
00 07 0010 00000000 P3
00 07 0014 00000000 P2
00 07 0018 00000000 P1
00 07 001C 00000000 LINE
00 07 0020 00000000 POINT
00 07 0024 00000000 QQSV
01 01 0000 00000000 Y2
01 01 0004 00000000 Y1
01 01 0008 00000000 MAX
01 01 000C 00000010 X2
01 01 0010 00000014 MIN
01 01 0014 0000001C X1
01 01 0018 00000020 Q2
01 01 001C 00000024 Q1
01 01 0020 00000028 DISPA
01 01 0024 00000030 J
01 01 0028 00000034 I
01 01 002C 00000038 MOVEA
01 01 0030 00000040 DY
01 01 0034 00000044 DX
01 01 0038 00000048 DIS
01 01 003C 0000004C PICTA
01 01 0040 00000054 NOTPOINT
01 01 0044 0000005C IUP
01 01 0048 00000064 MADREAD
01 01 004C 0000006C ENDIUP
01 01 0050 00000074 FORMAT
01 01 0054 0000007C MADWRITE
01 01 0058 00000084 PICTUREN
01 01 005C 00000088 M2
01 01 0060 0000008C M1
01 01 0064 00000090 M
01 01 0068 00000094 QQ2
01 01 006C 00000098 COMMAND
01 01 0070 0000009C QQ1
01 01 0074 000000A0 'ENDSTATEMENT'
01 01 0078 000000A4 Y
01 01 007C 000000A8 X
01 01 0080 000000AC 'SAMEAS'
01 01 0084 000000B0 HEAD
01 01 0088 000000B4 DISPLAYN
01 01 008C 000000B8 SKETCH
01 01 0090 0000029C "' ILLEGAL COMMAND'*"
01 01 0094 000002AF "' .',10(' .')*"
01 01 0098 000002C3 ". "
01 01 009C 000002C4 "13,5201.1*"
01 01 00A0 000002CE "'1 .',10(' .')*"
01 01 00A4 000002E2 "* "
01 01 00A8 000002E3 " "
01 01 00AC 000002E4 "' NOTHING TO DISPLAY.*"
01 01 00B0 000002FB "DISP"
01 01 00B4 000002FF "31*"
01 01 00B8 00000302 "' ENTER DISPLAY NUMBER OF POINT AND NEW X,Y'*"

```

```

01 01 0000 0000032F "MOVE"
01 01 0000 00000333 "I*"
01 01 0000 00000335 "' TOO MANY PICTURES.'*"
01 01 0000 0000034C 100
01 01 0000 00000350 "' ENTER DISPLAY NUMBERS FOR ALL POINTS.'*"
01 01 0000 00000379 "PICT"
01 01 0000 0000037D "' THOSE ARE NOT POINTS.'*"
01 01 0000 00000396 "' ENTER DISPLAY NUMBERS FOR END-POINTS.'*"
01 01 0000 000003BF "LINE"
01 01 0000 000003C3 "' POINT IS OUTSIDE RASTER RANGE.'*"
01 01 0000 000003E5 "' ASSIGNED DISPLAY NUMBER',HI4*"
01 01 0000 000004C4 5C
01 01 0000 000004C8 "2I*"
01 01 0000 0000040B "' ENTER X AND Y COORDINATES.'*"
01 01 0000 00000429 "PUIN"
01 01 0000 0000042D "C4.4*"
01 01 0000 00000432 "'ENTER A COMMAND PLEASE.'*"
01 01 0000 00000450 6
01 01 0000 00000454 5
01 01 0000 00000458 4
01 01 0000 0000045C 3
01 01 0000 00000460 0
01 01 0000 00000464 2
01 01 0000 00000468 1
01 01 0000 00000470 %DIM002
01 01 0000 00000480 %DIM001
01 01 0000 00000498 PICTURE
01 01 0000 00000628 DISPLAY

```

DICTIONARY

```

%DIM0001 'FIXEDARRAY' 01010000 00000480
  COMP.SIZE=4
  COMPONENT= 'INTEGER'
%DIM0002 'FIXEDARRAY' 01010000 00000470
  COMP.SIZE=4
  COMPONENT= 'INTEGER'
'ENDSTATEMENT' 'INTEGER' 01010000 000000AC
'SAMEAS' 'INTEGER' 01010000 000000AC
A 'INTEGERSHORT' 00030034 00000000 (FORMAL PAR)
B 'INTEGERSHORT' 00030030 00000000 (FORMAL PAR)
COMMAND 'CHARACTER' 01010000 00000098
  LENGTH=4
DIS 'INTEGER' 01010000 00000048
DISPA 'ENTRYNAME' 01010000 00000028
  RESULT= 'INTEGER'
DISPLAY 'FIXEDARRAY' 01010000 00000628
  COMP.SIZE=1 DIM.VEC.=%DIM0001
  COMPONENT= 'CHARACTER'
  LENGTH=1
DISPLAYN 'INTEGERSHORT' 01010000 000000B4
DX 'INTEGER' 01010000 00000044
DY 'INTEGER' 01010000 00000040
ENDICP 'ENTRYNAME' 01010000 0000006C 'EXTERNAL'
  RESULT= 'INTEGER'
FORMAT 'ENTRYNAME' 01010000 00000074 'EXTERNAL'
  RESULT= 'INTEGER'
HEAD 'POINTER' 01010000 00000080
I 'INTEGER' 01010000 00000034
IOP 'ENTRYNAME' 01010000 0000005C 'EXTERNAL'
  RESULT= 'INTEGER'
J 'INTEGER' 01010000 00000030
LINE 'COMPONENTSTRUCTURE' 00070010 00000000 'BASED'
  SIZE=20
  COMPONENT= 'POINTER'
  COMPONENT= 'POINTER'
  COMPONENT= 'INTEGERSHORT'
  COMPONENT= 'INTEGERSHORT'
  COMPONENT= 'POINTER'
  COMPONENT= 'POINTER'
M 'FLOATING' 01010000 00000090
MADREAD 'ENTRYNAME' 01010000 00000064 'EXTERNAL'
  RESULT= 'INTEGER'
MADWRITE 'ENTRYNAME' 01010000 0000007C 'EXTERNAL'
  RESULT= 'INTEGER'
MAX 'ENTRYNAME' 01010000 00000008
  RESULT= 'INTEGER'
MIN 'ENTRYNAME' 01010000 00000014
  RESULT= 'INTEGER'
MOVEA 'ENTRYNAME' 01010000 00000038
  RESULT= 'INTEGER'
M1 'FLOATING' 01010000 0000008C
M2 'FLOATING' 01010000 00000088
NOTPOINT 'ENTRYNAME' 01010000 00000054
  RESULT= 'INTEGER'
PICTA 'ENTRYNAME' 01010000 0000004C
  RESULT= 'INTEGER'
PICTURE 'FIXEDARRAY' 01010000 00000498

```



```

      COMP.SIZE=4 DIM.VEC.=%DIM0002
      COMPONENT= 'POINTER'
PICTUREN 'INTEGER' 01C10000 00000084
POINT 'COMPONENTSTRUCTURE' 0007002C 00000000 'BASED'
      SIZE=16
      COMPONENT= 'PCINTER'
      COMPONENT= 'POINTER'
      COMPONENT= 'INTEGERSHORT'
      COMPONENT= 'INTEGERSHORT'
      COMPONENT= 'INTEGERSHORT'
      COMPONENT= 'INTEGERSHORT'
P1 'COMPONENTSTRUCTURE' 00070018 00000000 'BASED'
      SIZE=16
      COMPONENT= 'POINTER'
      COMPONENT= 'POINTER'
      COMPONENT= 'INTEGERSHORT'
      COMPONENT= 'INTEGERSHORT'
      COMPONENT= 'INTEGERSHORT'
      COMPONENT= 'INTEGERSHORT'
P2 'COMPONENTSTRUCTURE' 00070014 00000000 'BASED'
      SIZE=16
      COMPONENT= 'POINTER'
      COMPONENT= 'POINTER'
      COMPONENT= 'INTEGERSHORT'
      COMPONENT= 'INTEGERSHORT'
      COMPONENT= 'INTEGERSHORT'
      COMPONENT= 'INTEGERSHORT'
P3 'COMPONENTSTRUCTURE' 00070010 00000000 'BASED'
      SIZE=16
      COMPONENT= 'POINTER'
      COMPONENT= 'POINTER'
      COMPONENT= 'INTEGERSHORT'
      COMPONENT= 'INTEGERSHORT'
      COMPONENT= 'INTEGERSHORT'
      COMPONENT= 'INTEGERSHORT'
QQSV 'COMPONENTSTRUCTURE' 00070024 00000000 'BASED'
      SIZE=16
      COMPONENT= 'POINTER'
      COMPONENT= 'PCINTER'
      COMPONENT= 'INTEGERSHORT'
      COMPONENT= 'INTEGERSHORT'
      COMPONENT= 'INTEGERSHORT'
      COMPONENT= 'INTEGERSHORT'
QQ1 'POINTER' 01C10000 0000009C
QQ2 'POINTER' 01010000 00000094
Q1 'INTEGER' 01010000 00000024
Q2 'INTEGER' 01010000 00000020
SKETCH 'ENTRYNAME' 01010000 000000B8 'ACCESSIBLE'
      RESULT= 'INTEGER'
X 'INTEGER' 01010000 000000A8
X 'INTEGERSHORT' 0003002C 00000000 (FORMAL PAR)
X1 'INTEGER' 01010000 0000001C
X2 'INTEGER' 01010000 00000010
Y 'INTEGER' 01010000 000000A4
Y 'INTEGERSHORT' 0003002E 00000000 (FORMAL PAR)
Y1 'INTEGER' 01010000 00000004
Y2 'INTEGER' 01010000 00000000
" " 'CHARACTER' 01010000 000002E3
      LENGTH=1
"." 'CHARACTER' 01010000 000002C3

```

```

    LENGTH=1
"*" 'CHARACTER' 01010000 000002E2
    LENGTH=1
"1",10("1")*" 'CHARACTER' 01010000 000002AF
    LENGTH=20
"1 ASSIGNED DISPLAY NUMBER",HI4*" 'CHARACTER' 01010000 000003E5
    LENGTH=31
"1 ENTER DISPLAY NUMBER OF POINT AND NEW X,Y*" 'CHARACTER' 01010000 00000302
    LENGTH=45
"1 ENTER DISPLAY NUMBERS FOR ALL POINTS.*" 'CHARACTER' 01010000 00000350
    LENGTH=41
"1 ENTER DISPLAY NUMBERS FOR END-POINTS.*" 'CHARACTER' 01010000 00000396
    LENGTH=41
"1 ENTER X AND Y COORDINATES.*" 'CHARACTER' 01010000 0000040B
    LENGTH=30
"1 ILLEGAL COMMAND*" 'CHARACTER' 01010000 0000029C
    LENGTH=19
"1 NOTHING TO DISPLAY.*" 'CHARACTER' 01010000 000002F4
    LENGTH=23
"1 POINT IS OUTSIDE RASTER RANGE.*" 'CHARACTER' 01010000 00000303
    LENGTH=34
"1 THOSE ARE NOT POINTS.*" 'CHARACTER' 01010000 0000037D
    LENGTH=25
"1 TOO MANY PICTURES.*" 'CHARACTER' 01010000 00000335
    LENGTH=22
"1 ENTER A COMMAND PLEASE.*" 'CHARACTER' 01010000 00000432
    LENGTH=27
"1",10("1")*" 'CHARACTER' 01010000 000002CE
    LENGTH=20
"C4.4*" 'CHARACTER' 01010000 0000042D
    LENGTH=5
"DISP" 'CHARACTER' 01010000 000002FB
    LENGTH=4
"I*" 'CHARACTER' 01010000 00000333
    LENGTH=2
"13,52C1.1*" 'CHARACTER' 01010000 000002C4
    LENGTH=10
"LINE" 'CHARACTER' 01010000 000003BF
    LENGTH=4
"MOVE" 'CHARACTER' 01010000 0000032F
    LENGTH=4
"PICT" 'CHARACTER' 01010000 00000379
    LENGTH=4
"POIN" 'CHARACTER' 01010000 00000429
    LENGTH=4
"2I*" 'CHARACTER' 01010000 00000408
    LENGTH=3
"3I*" 'CHARACTER' 01010000 000002FF
    LENGTH=3
0 'INTEGER' 01010000 00000460
1 'INTEGER' 01010000 00000468
100 'INTEGER' 01010000 0000034C
2 'INTEGER' 01010000 00000464
3 'INTEGER' 01010000 0000045C
4 'INTEGER' 01010000 00000458
5 'INTEGER' 01010000 00000454
50 'INTEGER' 01010000 00000404
6 'INTEGER' 01010000 00000450

```

EXTERNAL SYMBOL DICTIONARY (SYMBOL,TYPE,LD,ADDR,LENGTH/LDID)

##SKETCH	PD	01	0C0000	0C124C
#SKETCH	SD	02	000C00	0C168A
MADSTACK	ER	03		
SKETCH	LD		0C0000	0C0002
MACWRITE	ER	04		
FORMAT	ER	05		
ENDIOP	ER	06		
MACREAD	ER	07		
IOP	ER	08		
GETSPACE	ER	09		
LINSUB	ER	0A		

RELOCATION DICTIONARY (P.ID,R.ID,FLAGS,ADDRESS)

01	C2	0C	0000B8
01	04	1C	00007C
01	C5	1C	000074
01	01	0C	000294
01	C6	1C	00006C
01	07	1C	000064
01	01	0C	000288
01	C8	1C	00005C
01	C1	0C	00027C
01	C1	0C	000270
01	C1	0C	000264
01	C1	0C	000258
01	C1	0C	00024C
01	C9	1C	001060
01	C1	0C	000240
01	C1	0C	000228
01	C1	0C	00021C
01	C1	0C	000210
01	C1	0C	000204
01	C1	0C	0001F8
01	C2	0C	000054
01	C1	0C	0001EC
01	C1	0C	0001E0
01	C1	0C	0001D4
01	CA	1C	0010A4
01	C1	0C	001094
01	01	0C	001098
01	C2	0C	00004C
01	C1	0C	0001C8
01	01	0C	0001BC
01	C1	0C	001088
01	C1	0C	00108C
01	C1	0C	0010D4
01	C1	0C	0010D8
01	C1	0C	0001B0
01	C1	0C	0001A4
01	C1	0C	000198
01	C1	0C	00018C
01	C1	0C	000180
01	C2	0C	000038
01	C1	0C	000174
01	C1	0C	00110C
01	C1	0C	001110
01	C1	0C	001114
01	C2	0C	000028
01	C1	0C	001130
01	C1	0C	001134
01	C1	0C	001138
01	C1	0C	00116C
01	01	0C	001164
01	C1	0C	001168
01	C1	0C	00119C
01	C1	0C	0011A0
01	C1	0C	0011A4
01	C1	0C	000118
01	C1	0C	00010C
01	C1	0C	000100
01	C1	0C	0000F4

01 C1 0C 0000E8  
01 C1 0C 0011BC  
01 C1 0C 0011C0  
01 C1 0C 0011C4  
01 C1 0C 0011D8  
01 C1 0C 0011DC  
01 C1 0C 0011E0  
01 C1 0C 0000D0  
01 C1 0C 0000C4  
01 C2 0C 000014  
01 C2 0C 0000C8  
02 C1 0C 001648  
02 C1 0C 001584  
02 C1 0C 000020  
01 C1 0C 0011F0  
01 C1 0C 0011F4  
01 C2 0C 0011F8  
01 C3 0C 0011FC  
01 C4 0C 001228  
01 C5 0C 00122C  
01 C6 0C 001230  
01 C7 0C 001234  
01 C8 0C 001238  
01 C1 0C 00123C  
01 C9 0C 001240  
01 CA 0C 001244  
01 C2 0C 001248  
02 00C000

MAD/I COMPILER TIMINGS:  
EXECUTION TERMINATED

END SKETCH

171.625 CPU SECCNDS.  
326.663 ELAPSED SECONDS.

Appendix F. Run of the MAD/I Program

\$SET ERRORDUMP=UN  
\$RUN SKETCHOBJ MAP SCARDS=-DATA

.....

ENTRY = 10C000 SIZE = 00A6B3

NAME	VALUE	T	RF	NAME	VALUE	T	RF	NAME	VALUE	T	RF
GETSPACE	011610	*		FREESPAC	0118A2	*		SYSTEM	0167A4	*	
ERROR	0167CE	*		PGNTTRP	018E8C	*		LOAD	018EF2	*	
GETFD	0197E0	*		SCARDS	019A80	*		SPRINT	019A92	*	
SPUNCH	019AA4	*		SERCOM	019AB6	*		READ	019B34	*	
WRITE	019B50	*		LCSYMBOL	01A5E8	*		GLAP	1030C8		OFFA60
LINSUB	103B48		103B48	MADIO	103C70		103C70	MADREAD	103C7C		
MADWRITE	103C9E			FORMAT	103CBA			IOP	103DFC		
ENDIOP	103E42			#SKETCH	10A00C		10A000	IOPKG	10B250		10B250
ROPEN	10B31E			RCLOSE	10B398			POPEN	10B3C4		
PCLOSE	10B410			MDIOPSC	10B500		10B9E8	SPIE	10BDE8		10BDE8
#SKETCH	10C000		10C000	IOH360	111000		111000	IOHIN	1110F0		
IOHOUT	111114			ICHETC	11183C			ONE@ATIM	11192C		
IOHERP	115000		110F18	MADSTACK	117000		117C00				

.....

EXECUTION BEGINS

ENTER A COMMAND PLEASE.  
NOTHING TO DISPLAY.

ENTER A COMMAND PLEASE.  
ENTER X AND Y COORDINATES:  
ASSIGNED DISPLAY NUMBER 1

ENTER A COMMAND PLEASE.  
ENTER X AND Y COORDINATES:  
ASSIGNED DISPLAY NUMBER 2

ENTER A COMMAND PLEASE.  
ENTER X AND Y COORDINATES:  
ASSIGNED DISPLAY NUMBER 3

ENTER A COMMAND PLEASE.  
ENTER X AND Y COORDINATES:  
ASSIGNED DISPLAY NUMBER 4

ENTER A COMMAND PLEASE.

```

. . . . .
50 .
49 .
48 .
47 .
46 .
45 .
44 .
43 .
42 .
41 .
40 .      *
39 .
38 .
37 .
36 .
35 .
34 .
33 .
32 .
31 .
30 .
29 .
28 .
27 .
26 .
25 .
24 .
23 .
22 .
21 .
20 .
19 .
18 .
17 .
16 .
15 .
14 .
13 .
12 .
11 .
10 .      *
9 .
8 .
7 .
6 .
5 .
4 .
3 .
2 .
1 .
. . . . .

```

ENTER A COMMAND PLEASE.  
ENTER DISPLAY NUMBERS FOR END-POINTS:

ENTER A COMMAND PLEASE.  
ENTER DISPLAY NUMBERS FOR END-POINTS:

ENTER A COMMAND PLEASE.

ENTER DISPLAY NUMBERS FOR END-POINTS:

ENTER A COMMAND PLEASE.

ENTER DISPLAY NUMBERS FOR END-POINTS:

ENTER A COMMAND PLEASE.



```

. . . . .
50 .
49 .
48 .
47 .
46 .
45 .
44 .
43 .
42 .
41 .
40 . *****
39 . *
38 . *
37 . *
36 . *
35 . *
34 . *
33 . *
32 . *
31 . *
30 . *
29 . *
28 . *
27 . *
26 . *
25 . *
24 . *
23 . *
22 . *
21 . *
20 . *
19 . *
18 . *
17 . *
16 . *
15 . *
14 . *
13 . *
12 . *
11 . *
10 . *****
9 .
8 .
7 .
6 .
5 .
4 .
3 .
2 .
1 .
. . . . .

```

ENTER A COMMAND PLEASE.  
ENTER DISPLAY NUMBER OF POINT AND NEW X,Y  
ENTER A COMMAND PLEASE.

```

50 .
49 .
48 .
47 .
46 .
45 .
44 .
43 .
42 .
41 .
40 . *****
39 . *
38 . *
37 . *
36 . *
35 . *
34 . *
33 . *
32 . *
31 . *
30 . *
29 . *
28 . *
27 . *
26 . *
25 . *
24 . *
23 . *
22 . *
21 . *
20 . *
19 . **
18 . ***
17 . **
16 . ***
15 . **
14 . ***
13 . **
12 . ***
11 . **
10 . ***
9 .
8 .
7 .
6 .
5 .
4 .
3 .
2 .
1 .

```

ENTER A COMMAND PLEASE.  
 ENTER DISPLAY NUMBERS FOR ALL POINTS.

ENTER A COMMAND PLEASE.  
 ENTER DISPLAY NUMBER OF POINT AND NEW X,Y

ENTER A COMMAND PLEASE.

```

. . . . .
50 .
49 .
48 .
47 .
46 .
45 .
44 .
43 .
42 .
41 .
40 . *****
39 . *
38 . *
37 . *
36 . *
35 . *
34 . *
33 . *
32 . *
31 . *
30 . *
29 . *
28 . *
27 . *
26 . *
25 . *
24 . *
23 . *
22 . *
21 . *
20 . *
19 . **
18 . ***
17 . **
16 . ***
15 . **
14 . ***
13 . **
12 . ***
11 . **
10 . ***
9 .
8 .
7 .
6 .
5 .
4 .
3 .
2 .
1 .
. . . . .

```

ENTER A COMMAND PLEASE.

\*\*\*\* ALL INPUT DATA HAS BEEN PROCESSED - AT LOCATION 1C3DE8  
EXECUTION TERMINATED



Security Classification

## DOCUMENT CONTROL DATA - R &amp; D

*(Security classification of title, body of abstract and indexing annotation must be entered when the overall report is classified)*

1. ORIGINATING ACTIVITY <i>(Corporate author)</i>		2a. REPORT SECURITY CLASSIFICATION	
UNIVERSITY OF MICHIGAN CONCOMP PROJECT		Unclassified	
		2b. GROUP	
3. REPORT TITLE			
AN EXAMPLE DEFINITIONAL FACILITY IN MAD/I			
4. DESCRIPTIVE NOTES <i>(Type of report and inclusive dates)</i>			
Memorandum			
5. AUTHOR(S) <i>(First name, middle initial, last name)</i>			
Ronald J. Srodawa			
6. REPORT DATE		7a. TOTAL NO. OF PAGES	7b. NO. OF REFS
August 1970		25	3
8a. CONTRACT OR GRANT NO.		8a. ORIGINATOR'S REPORT NUMBER(S)	
DA-49-083 OSA-3050		Memorandum 32	
b. PROJECT NO.			
c.		9b. OTHER REPORT NO(S) <i>(Any other numbers that may be assigned this report)</i>	
d.			
10. DISTRIBUTION STATEMENT			
Qualified requesters may obtain copies of this report from DDC.			
11. SUPPLEMENTARY NOTES		12. SPONSORING MILITARY ACTIVITY	
		Advanced Research Projects Agency	
13. ABSTRACT			
<p>The MAD/I language is a procedure-oriented algebraic language which is a descendant of ALGOL 60 and 7090 MAD, similar in power and scope to PL/I. The MAD/I compiler is implemented using the MAD/I facility, a flexible translator-building system whose dynamic nature allows compilers to be extended during the compilation process. This paper demonstrates the extension of MAD/I to include several graphics-oriented statements and operators through a lucid example.</p>			

DD FORM 1473  
NOV 65Unclassified  
Security Classification

14. KEY WORDS	LINK A		LINK B		LINK C	
	ROLE	WT	ROLE	WT	ROLE	WT
MAD MAD/I Programming Language Extensible Language Statement Definitions Operator Definitions						