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 DEFESNE
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 <u>not</u> 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
                'PROCEDURE' SKETCH.;

                'INCLUDE' "DISPLAYDEF"

                'DECLARE' 'NORMAL MODE' 'INTEGER';
                'DECLARE' COMMAND 'CHARACTER'(4);
                'DECLARE' DISPLAY 'FIXED ARRAY'(50,50) 'CHARACTER'(1);
                'DECLARE' (POINT,P1,P2,P3) 'POINT';
                'DECLARE' LINE 'LINE';
                'DECLARE' PICTURE 'FIXED ARRAY'(100) 'PICTURE';
                'DECLARE' QQ1 'POINTER';
                'DECLARE' QQ2 'POINTER';
                'DECLARE' (M,M1,M2) 'FLOATING';

                'PRESET' PICTUREN := 0;


       SKETCH:  'WRITE' , "'0ENTER A COMMAND PLEASE.'*";
                'READ' , "C4.4*" , COMMAND;

                'IF' COMMAND = "POIN";
                     'WRITE' , "' ENTER X AND Y COORDINATES:'*";
                     'READ' , "2I*", X,Y;
                     'IF' 1 <= X & 50 >= X & 1 <= Y & 50 >= Y;
                          'CREATE POINT' POINT,X,Y;
                          'WRITE' , "' ASSIGNED DISPLAY NUMBER',HI4*", .DISPN. POINT;
                     'ELSE';
                          'WRITE' , "' POINT IS OUTSIDE RASTER RANGE.'*";
                     'END';

                'OR IF' COMMAND = "LINE";
                     'WRITE' , "' ENTER DISPLAY NUMBERS FOR END-POINTS:'*";
                     'READ' , "2I*", X,Y;
                     P1 .EVAL. .ADROF. X; P2 .EVAL. .ADROF. Y;
                     'IF' .POINT. P1 & .POINT. P2;
                          'CREATE LINE' LINE,P1,P2;
                     'ELSE';
       NOTPOINT:          'WRITE' , "' THOSE ARE NOT POINTS.'*";
                     'END';

                'OR IF' COMMAND = "PICT";
                     'WRITE' , "' ENTER DISPLAY NUMBERS FOR ALL POINTS.'*";
                     PICTUREN := PICTUREN+1;
                     'IF' PICTUREN > 100;
                          'WRITE' , "' TOO MANY PICTURES.'*";
                     'ELSE';
                          PICTURE(PICTUREN) := 0 .AS. ('POINTER');
       PICTA:             'READ' , "I*", X;
                          'IF' X¬= 0;
                               P1 .EVAL. .ADROF. X;
                               'IF' ¬.POINT. P1, 'GO TO' NOTPOINT;
                               QQ2 := PICTURE(PICTUREN);
                               'CONNECT' P1 'TO' QQ2;
                               PICTURE(PICTUREN) := QQ2;
                               'GO TO' PICTA;
                          'END'
                     'END';
```

```
          'OR IF' COMMAND = "MOVE";
               'WRITE' , "' ENTER DISPLAY NUMBER OF POINT AND NEW X,Y'*";
               'READ' , "3I*", DIS,X,Y;
               P1 .EVAL. .ADROF. DIS;
               'IF' ¬.POINT. P1, 'GOTO' NOTPOINT;
               DX := X-.XOF. P1; DY := Y-.YOF. P1;
               (.XOF. P1) := .XOF. P1+DX; (.YOF. P1) := .YOF. P1+DY;
               QQ1 := P1(2);
               'IF' QQ1 .AS. ('INTEGER') ¬= 0;
                    P2 .EVAL. .NEXT. P1;
MOVEA:              'IF' .DISPN. P1 ¬= .DISPN. P2;
                         (.XOF. P2) := .XOF. P2+DX; (.YOF. P2) := .YOF. P2
                                                   +DY;
                         P2 .EVAL. .NEXT. P2;
                         'GO TO' MOVEA;
                    'END'
               'END';

          'OR IF' COMMAND = "DISP";
               'IF' HEAD .AS. ('INTEGER') = 0;
                    'WRITE' , "' NOTHING TO DISPLAY.'*";
                    'GO TO' SKETCH;
               'END';
               'FOR' I := 1,1,I>50, 'FOR' J := 1,1,J>50, DISPLAY(I,J) := " ";
               P1 .EVAL. HEAD;
DISPA:         ;
               'IF' .POINT. P1;
                    Q1 := .XOF. P1; Q2 := .YOF. P1; DISPLAY(Q2,Q1) := "*";
               'ELSE';
                    LINE .EVAL. (.PT. P1);
                    QQ1 := .ENDA. LINE; P2 .EVAL. QQ1;
                    QQ1 := .ENDB. LINE; P3 .EVAL. QQ1;
                    X1 := MIN.(.XOF. P2,.XOF. P3);
                    X2 := MAX.(.XOF. P2,.XOF. P3);
                    'IF' X1 = X2;
                         Y1 := MIN.(.YOF. P2,.YOF. P3);
                         Y2 := MAX.(.YOF. P2,.YOF. P3);
                         'FOR' Y := Y1,1, Y > Y2, DISPLAY(Y,X1) := "*";
                    'ELSE';
                         M1 := .YOF. P3 - .YOF. P2;
                         M2 := .XOF. P3 - .XOF. P2;
                         M := M1/M2;
                         'FOR' X := X1,1,X > X2;
                              Q2 := M*(X - .XOF. P2) + .YOF. P2;
                              DISPLAY(Q2,X) := "*";
                         'END';
                    'END';
               'END';
               QQ1 := .HEAD. P1;
               'IF' QQ1 .AS. ('INTEGER') ¬= 0;
                    P1 .EVAL. .HEAD. P1;
                    'GO TO' DISPA;
               'END';
               'WRITE' , "'1    .',10('     .')*";
               'FOR' I := 50,-1, I < 1,
                    'WRITE' , "I3,52C1.1*", I," ",".",
                                        DISPLAY(I,1)...DISPLAY(I,50);
               'WRITE' , "'     .',10('     .')*";

          'ELSE';
```

```
                    'WRITE' , "' ILLEGAL COMMAND'*";
                'END';

                'GO TO' SKETCH;

                'PROCEDURE' MIN.(X,Y);
                'INTEGER SHORT' (X,Y);
        MIN:    'IF' X <= Y, 'RETURN' X;
                'RETURN' Y;
                'END';

                'PROCEDURE' MAX.(A,B);
                'INTEGER SHORT' (A,B);
        MAX:    'IF' A >= B, 'RETURN' A;
                'RETURN' B;
                'END';

                'END'
END OF FILE
```

Appendix B. Contents of File DISPLAYDEF

```
IST DISPLAYDEF
        <<
                        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 @ CONSTRUCTION.  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 SUBSTITUITION.
             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
```

```

                     'CREATE POINT' POINT,X,Y

               WHICH HAS THE EFFECT OF THE LIST OF STATEMENTS SHOWN
       >>
               'DEFINE STATEMENT' 'CREATE POINT' POINT,X,Y;
                    'ALLOCATE' POINT;
                    POINT(1) := HEAD;
                    HEAD := .PT. POINT;
                    POINT(2) := (0 .AS. ('POINTER'));
                    DISPLAYN := DISPLAYN+1;
                    POINT(3) := DISPLAYN;
                    POINT(4) := 1;
                    POINT(5) := X;
                    POINT(6) := Y;
               'END STATEMENT';
       <<
               THE FOLLOWING DEFINES THE STATEMENT

                          'CREATE LINE' LINE,P1,P2

               WHICH HAS THE SAME EFFECT AS THE LIST OF STATEMENTS SHOWN
       >>
               'DEFINE STATEMENT' 'CREATE LINE' LINE,P1,P2;
                    'ALLOCATE' LINE;
                    LINE(1) := HEAD;
                    HEAD := .PT. LINE;
                    LINE(2) := (0 .AS. ('POINTER'));
                    DISPLAYN := DISPLAYN+1;
                    LINE(3) := DISPLAYN;
                    LINE(4) := 2;
                    LINE(5) := .PT. P1;
                    LINE(6) := .PT. P2;
               'END STATEMENT';
       <<
               THE FOLLOWING DEFINES THE STATEMENT

                          'CONNECT' POINT 'TO' PICTURE

               WHICH HAS THE SAME EFFECT AS THE LIST OF STATEMENT SHOWN
       >>
               'DECLARE SYNTACTIC CLASS' 'TO' 'SAME AS' , ;
               'DEFINE STATEMENT' 'CONNECT' POINT 'TO' PICTURE;
                    QQ1 := PICTURE;
                    'IF' (QQ1 .AS. ('INTEGER')) = 0;
                         PICTURE := .PT. POINT;
                         POINT(2) := .PT. POINT;
                    'ELSE';
                         QQSV .EVAL. PICTURE;
                         POINT(2) := QQSV(2);
                         QQSV(2) := .PT. POINT;
                    'END';
               'END STATEMENT';
       <<
               THE FOLLOWING MACRO DEFINES THE .POINT. OPERATOR AS A PASS ONE MACRO.
               IT CAUSES .POINT. A TO BE TREATED AS A(4) = 1
       >>
               'DECLARE SYNTACTIC CLASS' .POINT. 'SAME AS' .ABS.;
               'DEFINE';
               MACRO,.POINT.,T,B;
```

```
                LOCAL,U;
                TEMPORARY,U;
                .TAG.,U,B,'LOCALLITERAL' 4;
                LN=,T,U,'LOCALLITERAL' 1;
                MEND,.POINT.;
                END;
        <<
                THE FOLLOWING MACRO DEFINES .XOF. AS A PASS ONE OPERATOR.
                IT CAUSES .XOF. A TO BE TREATED AS A(5)
        >>
                'DECLARE SYNTACTIC CLASS' .XOF. 'SAME AS' .ABS.;
                'DEFINE';
                MACRO,.XOF.,T,A;
                .TAG.,T,A,'LOCALLITERAL' 5;
                MEND,.XOF.;
                END;
        <<
                THE FOLLOWING CAUSES .YOF. TO BE DEFINED AS A PASS ONE OPERATOR
                .YOF. A HAS THE SAME EFFECT AS A(6)
        >>
                'DECLARE SYNTACTIC CLASS' .YOF. 'SAME AS' .ABS.;
                'DEFINE';
                MACRO,.YOF.,T,A;
                .TAG.,T,A,'LOCALLITERAL' 6;
                MEND,.YOF.;
                END;
        <<
                THE FOLLOWING DEFINES .NEXT. A TO BE THE SAME AS A(2)
        >>
                'DECLARE SYNTACTIC CLASS' .NEXT. 'SAME AS' .ABS.;
                'DEFINE';
                MACRO,.NEXT.,T,A;
                .TAG.,T,A,'LOCALLITERAL' 2;
                MEND,.NEXT.;
                END;
        <<
                THE FOLLOWING DEFINES .DISPN. A TO BE THE SAME AS A(3)
        >>
                'DECLARE SYNTACTIC CLASS' .DISPN. 'SAME AS' .ABS.;
                'DEFINE';
                MACRO,.DISPN.,T,A;
                .TAG.,T,A,'LOCALLITERAL' 3;
                MEND,.DISPN.;
                END;
                'DECLARE SYNTACTIC CLASS' .ENDA. 'SAME AS' .ABS.;
                'DEFINE';
                MACRO,.ENDA.,T,A;
                .TAG.,T,A,'LOCALLITERAL' 5;
                MEND,.ENDA.;
                END;
        <<
                THE FOLLOWING DEFINES .ENDB. A TO BE THE SAME AS A(6)
        >>
                'DECLARE SYNTACTIC CLASS' .ENDB. 'SAME AS' .ABS.;
                'DEFINE';
                MACRO,.ENDB.,T,A;
                .TAG.,T,A,'LOCALLITERAL' 6;
                MEND,.ENDB.;
                END;
        <<
```

```
                THE FOLLOWING DEFINES .HEAD. A TO BE THE SAME AS A(1)
      >>
                'DECLARE SYNTACTIC CLASS' .HEAD. 'SAME AS' .ABS.;
                'DEFINE';
                MACRO,.HEAD.,T,A;
                .TAG.,T,A,'LOCALLITERAL' 1;
                MEND,.HEAD.;
                END;
      <<
                THE FOLLOWING DEFINES .ADROF. A TO BE THE SAME AS THE SEQUENCE OF
                STATEMENTS
                     QQSV .EVAL. HEAD
                B1:  'IF' .DISPN. QQSV = A, 'GO TO' B2
                     QQSV .EVAL. (.HEAD. QQSV)
                     'GO TO' B1
                B2:  'RETURN' .PT. QQSV
      >>
                'DECLARE SYNTACTIC CLASS' .ADROF. 'SAME AS' .ABS.;
                'DEFINE';
                MACRO,.ADROF.,T,A;
                LOCAL,B1,B2;
                LOCAL,T1,T2,T3,T4,T5;
                TEMPORARY,T1,T2,T3,T4,T5;
                FLAD,B1,B2;
                .EVAL.,T1,%QQSV,%HEAD;
                DES,B1;
                .DISPN.,T2,%QQSV;
                LN=,T3,T2,A;
                TNZ,B2,T3;
                .HEAD.,T4,%QQSV;
                .EVAL.,T5,%QQSV,T4;
                GOTO,B1;
                DES,B2;
                .PT.,T,%QQSV;
                MEND,.ADROF.;
                END;
      <<
                          END OF DEFINITIONAL PACKAGE
      >>
END OF FILE
```

Appendix C. Contents of File DEFFACILITY

```
LIST DEFFACILITY
        <<
                THE FOLLOWING DEFINES THE NEW SIMPLE STATEMENT

                         'DECLARE SYNTACTIC CLASS' A 'SAME AS' B

                WHICH GIVES THE SYMBOL A EXACTLY THE SAME SYNTACTIC DEFINITION
                (CLASS,SYNTACTIC CLASS,LEFT PRECEDENCE,RIGHT PRECEDENCE) AS B.
                A WILL NOW BE TREATED EXACTLY AS B BY THE SYNTACTIC SCANNER.
        >>
                'DEFINE';                         <<ESCAPE INTO THE MACRO LANGUAGE>>
                SETUP,'DECLARE SYNTACTIC CLASS',14,3,4;      << SIMPLE KEYWORD>>
                MACRO,'DECLARE SYNTACTIC CLASS';         <<START THE MACRO DEF>>
                LOCAL,A;                          << A IS SAVED IN THIS LOCAL>>
                GTDSX;                            << THIS READS A>>
                SET,A,0,DSX;                      <<SAVE DSX OF A IN LOCAL SYMBOL>>
                GTDSX;                            <<NOW READ 'SAME AS'>>
                JMP,LOC+3,DSX = 'SAMEAS';         <<BE SURE IT REALLY IS 'SAMEAS'>>
                MNOTE,"****  ONLY 'SAME AS' IS ALLOWED";
                MEXIT;                            <<LEAVE HIM AT THE MERCY OF JSCAN>>
                GTDSX;                            <<READ B>>
                SET,IND.(A),1,DSX(1);             <<SET THE CLASS CODE OF A>>
                SET,IND.(A),2,DSX(2);             <<SET THE SYNTACTIC CLASS OF A>>
                SET,IND.(A),3,DSX(3);             <<SET THE LEFT PRECEDENCE OF A>>
                SET,IND.(A),4,DSX(4);             <<SET THE RIGHT PRECEDENCE OF A>>
                GTDSX;                            <<READ THE STATEMENT TERMINATOR>>
                MEND,'DECLARE SYNTACTIC CLASS';          <<ALL DONE>>
                END;                              <<BACK INTO MAD/I>>
        <<
                THE FOLLOWING DEFINES A SPECIAL PURPOSE DEFINITIONAL FACILITY
                WHICH ALLOWS THE DEFINITION OF A SIMPLE STATEMENT.  THE DEFINITIONAL
                STATEMENT IS OF THE FORM

                     'DEFINE STATEMENT' KEYWORD LIST OF VARIABLES
                          LIST OF STATEMENTS
                     'END STATEMENT'

                THE NEWLY DEFINED STATEMENT HAS THE FORM

                     KEYWORD  LIST OF EXPRESSIONS

                EACH EXPRESSION WHICH WILL OCCUR IN AN INSTANCE OF THE DEFINED
                STATEMENT IS REPRESENTED BY A VARIABLE IN THE PROTOTYPE OF THE DEFINE
                STATEMENT.  IN THE CODE GENERATED FOR THE STATEMENT THE CODE FOR
                THE EXPRESSIONS WILL BE GENERATED FIRST, FOLLOWED BY THE CODE FOR THE
                MAD/I STATEMENTS SPECIFIED IN THE DEFINITION.  ALL OCCURENCES OF THE
                VARIABLES CORRESPONDING TO THE EXPRESSIONS ARE REPLACED BY THE
                RESULT OF THE CORRESPONDING EXPRESSION.
        >>
                'DECLARE SYNTACTIC CLASS' 'DEFINE STATEMENT' 'SAME AS' 'DEFINE';
                'DEFINE';                         <<ESCAPE INTO THE METALANGUAGE>>
                CREATE,DEFSTATLAB,DFSTLB0000,PERCLS(@VAL);    <<SETS UP CRS>>
                MACRO,'DEFINE STATEMENT';
                LOCAL,KEYWD,VLIST,SLIST,NVLIST,I,Q;
                GTDSX;                            << GET THE KEYWORD>>
                SETUP,DSX,14,3,4;                 <<MAKE IT SIMPLE STATEMENT >>
                SET,KEYWD,0,DSX;                  <<KEYWD POINTS TO THE KEYWORD>>
                SET,KEYWD,1,INDCLS(@VAL);         <<AND IT IS INDIRECT>>
        A:      GTDSX;                            << GET A VARIABLE NAME>>
```

```
        JMP,B,DSX = LN; ;              <<SEMICOLON MARKS THE END>>
        SETLST,VLIST,,DSX;             << PUT THE VARIABLE INTO VLIST>>
        GTDSX;                         << GET THE COMMA OR SEMICOLON>>
        JMP,A,DSX ¬= LN; ;             <<THERE IS ANOTHER VARIABLE>>
B:      CRS,DEFSTATLAB,NOTIND.(DEFSTATLAB);      << A NEW LIST TO HOLD STM>>
        SET,SLIST,0,DEFSTATLAB;        <<SAVE THE LIST NAME>>
        SET,SLIST,1,INDCLS(@VAL);      << MAKE IT INDIRECT>>
        CRS,DEFSTATLAB,NOTIND.(DEFSTATLAB);      <<UNIQUE SYM FOR HIS VAR>>
        SETLST,NVLIST,,DEFSTATLAB;     <<BUILD A LIST OF THEM>>
        JMP,LOC-2,NVLIST(0) < VLIST(0);     <<WANT ONE FOR EACH VARIABLE>>
C:      GTDSX;                   <<READ A STATEMENT DSX>>
        JMP,D,DSX = 'END STATEMENT';   <<CHECK FOR END OF DEFINITION>>
        SET,I,0,1;                     <<NOW CHECK FOR IT IN THE VARIABLES>>
        JMP,E,VLIST(I(0)) = DSX;       <<JUMP IF FOUND IT>>
        SET,I,0,I(0)+1;                <<UP THE ANTE>>
        JMP,LOC-2,I(0) <= VLIST(0);    <<CONTINUE IF STILL MORE VARIABLES>>
        SETLST,SLIST,,DSX;             << ADD HIS DSX TO THE STREAM>>
        JMP,C;                         << PROCESS THE NEXT DSX>>
E:      SET,NOTIND.(DSX),0,NVLIST(I(0));    <<PERFORM THE SUBSTITUTION>>
        JMP,LOC-3;                     << AND NOW DO IT TO THIS>>
D:      SETLST,SLIST,,'END';           <<PUT A 'END' ON THE END >>
        SETLST,SLIST,,LN;;             << AND FINISH UP WITH A SEMICOLON >>
        MACRO,KEYWD;                   <<NOW DEFINE THE MACRO FOR THE KEYWD>>
        DESIST;
        SET,I,0,1;               <<GENERATE AN EXPRESSION SCAN FOR EACH VAR>>
        JMP,LOC+10,I(0) > VLIST(0);    << STOP IF END OF LIST>>
        SET,NOTIND.(Q),0,NVLIST(I(0));      <<Q WILL BE THE VARIABLE NAME>>
        SET,NOTIND.(Q),1,INDCLS(@VAL);
        RESUME;
        JSCAN,.TAG.,LIST;        <<SCAN OFF AN EXPRESSION>>
        SET,Q,0,EXP.(1);               <<MAKE THE VARIABLE THE RESULT>>
        SET,Q,1,INDCLS(@VAL);
        DESIST;
        SET,I,0,I(0)+1;                <<UP TO THE NEXT VARIABLE>>
        JMP,LOC-9;                     << AROUND AND AROUND>>
        RESUME;
        SET,GTDSXLIST,0,SLIST;         << REDEFINE GTDSX FOR AWHILE >>
        BLOCK;                         << THESE LOOK LIKE SCOPE OF STM>>
        MEND,KEYWD;                    <<END THE GENERATED MACRO>>
        GTDSX;                         << GET OUR SEMICOLON>>
        MEND,'DEFINE STATEMENT';
        END;                           <<BACK INTO MAD/I>>
<<
        THE FOLLOWING REDEFINES THE PSEUDO OP FOR GTDSX.  THE NEW GTDSX ACTS
        EXACTLY LIKE THE OLD DSX EXCEPT THAT IT WILL INSERT THE CONTENTS
        OF THE LIST POINTED TO BY GTDSXLIST INTO THE INPUT STREAM.  THIS
        ALLOWS A SEQUENCE OF DESCRIPTORS TO BE RETURNED TO JSCAN AS IF THEY
        CAME FROM THE INPUT STREAM
>>
        'DEFINE';
        SET,PREVGTDSX,0,GTDSX(0);      << THIS IS NOW THE REAL GTDSX >>
        SET,PREVGTDSX,1,GTDSX(1);
        SET,GTDSXLIST,@SIZE,0;         << INITIALIZE GTDSX CHEAT LIST >>
        SET,GTDSXLIST,0,0;             << USE REAL GTDSX FOR THE TIME BEING >>
        MACRO,GTDSX;                   << NOW REDEFINE GTDSX >>
        JMP,LOC+3,GTDSXLIST(0) ¬= 0;   << JUMP IF INSERTIONS TO BE MADE >>
        PREVGTDSX;                     << USE THE OLD-FASHION GTDSX >>
        MEXIT;
        SET,GTDSXLIST,@SIZE,GTDSXLIST(@SIZE)+1; << UP THE LIST INDEX >>
        JMP,LOC+5,GTDSXLIST(@SIZE) <= (IND.(GTDSXLIST(0)))(0);
```

```
            SET,GTDSXLIST,@SIZE,0;          << LIST PROCESSED SO RESET >>
            SET,GTDSXLIST,0,0;
            GTDSX;                          << NOW TRY READING AGAIN >>
            MEXIT;
            SET,NOTIND.(DSX),0,DSXOF.((IND.(GTDSXLIST(0)))(GTDSXLIST(@SIZE)));
            SET,NOTIND.(DSX),1,INDCLS(0);       << SET UP DSX >>
            MEND,GTDSX;
            END;
    <<
            THE FOLLOWING MACRO CREATES A FLOATING ADDRESS CORRESPONDING TO
            EACH PARAMETER
    >>
            'DEFINE';
            MACRO,FLAD;
            LOCAL,I;
            SET,I,0,1;
            JMP,LOC+2,PAR.(0) >= I(0);
            MEXIT;
            CRS,FLD,NOTIND.(FLD);
            SET,FLD,1,NOTIND.(FLD)(@MOD);
            SET,PAR.(I(0)),0,FLD;
            SET,PAR.(I(0)),1,INDCLS(0);
            SET,I,0,I(0)+1;
            JMP,LOC-7;
            MEND,FLAD;
            END;
    <<
            THE FOLLOWING MACRO CREATES A TEMPORARY CORRESPONDING TO EACH PARAMETER
    >>
            'DEFINE';
            MACRO,TEMPORARY;
            LOCAL,I;
            SET,I,0,1;
            JMP,LOC+2,PAR.(0) >= I(0);
            MEXIT;
            CRS,TMP,NOTIND.(TMP);
            SET,TMP,1,NOTIND.(TMP)(@MOD);
            SET,PAR.(I(0)),0,TMP;
            SET,PAR.(I(0)),1,INDCLS(C);
            SET,I,0,I(0)+1;
            JMP,LOC-7;
            MEND,TEMPORARY;
            END;
    <<
            THE FOLLOWING REDEFINES THE GETTEMP AND FREETEMP MACROS TO
            GET AROUND THE PROBLEM OF THE REASSIGNMENT OF THE STORAGE
            ALLOCATION OF A TEMPORARY.
    >>
            'DEFINE';
            MACEXCTYPE,7;
            MACDEFTYPE,2;
            POPMACRO,GETTEMP,FREETEMP;
            ATR,@TEMP1,EXTENDED;
            ATR,@TEMP2,LOCAL,20,4;
            SET,TEMPLST,@VAL,0;

            MACRO,GETTEMP,S,MODE,LEN;
            LOCAL,I,J;
            JMP,LOC+4,LEN <= 16;
    ERR:    MNOTE,"****  GETTEMP CALLED FOR MORE THAN SIXTEEN BYTES.";
```

```
                DUMPDSX,S,MODE,LEN;
                MEXIT;
                CLEAR,S;
                SET,I,@VAL,5;
                SET,J,@VAL,0;
     TEST:      JMP,DONE,I > TEMPLST;
                JMP,LOC+3,TEMPLST(I)(@TEMP2) ¬= 0;
                SET,J,@VAL,I;
                JMP,SEARCH;
                JMP,SEARCH,TEMPLST(I)(@TEMP2) ¬= DSXOF.(S);
     GOOD:      SET,TEMPLST(I),@TEMP2,DSXOF.(S);
                SET,S,@XA,1;
                SET,S,@MODE,MODE(@MODE);
                SET,S,@LEN,LEN;
                SET,S,@VAL,TEMPLST(I);
                MEXIT;
     SEARCH:    SET,I,@VAL,I+1;
                JMP,TEST;
     DONE:      JMP,NO,J = 0;
                SET,I,@VAL,J;
                JMP,GOOD;
     NO:        APND,TEMPLST,1,I;
                SET,TEMPLST,@VAL,I;
                SPACE,TEMPLST(I),16,8;
                JMP,GOOD;
                MEND,GETTEMP;

                MACRO,FREETEMP,S;
                LOCAL,I;
                RELSYMB,S;
                SET,I,@VAL,5;
                JMP,NO,I > TEMPLST;
                JMP,GOTIT,TEMPLST(I)(@TEMP2) = DSXOF.(S);
                SET,I,@VAL,I+1;
                JMP,LOC-3;
     GOTIT:     SET,TEMPLST(I),@TEMP2,0;
     NO:        SET,S,@VAL,0;
                SET,S,@XA,1;
                SET,S,@SIZE,0;
                SET,S,@CLS,NOTIND.(TMP)(@MOD);
                MEND,FREETEMP;

                MACDEFTYPE,1;
                MACEXCTYPE,1;
                END;
END OF FILE
```

Appendix D. Contents of File -DATA

```
$LIST -DATA
DISPLAY
POINT
10 10
POINT
10 40
POINT
40 10
POINT
40 40
DISPLAY
LINE
1 2
LINE
1 3
LINE
2 4
LINE
3 4
DISPLAY
MOVE
1 15 20
DISPLAY
PICTURE
1
2
3
4
0
MOVE
1 20 20
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),COMPILE
NOMAP,NOXREF,ATR,OPLIST,USER,NOADDENDA
```

MAD/I COMPILER VERSION AN049-134322.

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

```
           'PROCEDURE' SKETCH.;

           'INCLUDE' "DISPLAYDEF"
<<
                          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 @ CONSTRUCTION.  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 SUBSTITUITION.
           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 NEW SIMPLE STATEMENT

                'DECLARE SYNTACTIC CLASS' A 'SAME AS' B

        WHICH GIVES THE SYMBOL A EXACTLY THE SAME SYNTACTIC DEFINITION
        (CLASS,SYNTACTIC CLASS,LEFT PRECEDENCE,RIGHT PRECEDENCE) AS B.
        A WILL NOW BE TREATED EXACTLY AS B BY THE SYNTACTIC SCANNER.
>>
        'DEFINE';                         <<ESCAPE INTO THE MACRO LANGUAGE>>
        SETUP,'DECLARE SYNTACTIC CLASS',14,3,4;       << SIMPLE KEYWORD>>
        MACRO,'DECLARE SYNTACTIC CLASS';          <<START THE MACRO DEF>>
        LOCAL,A;                          << A IS SAVED IN THIS LOCAL>>
        GTDSX;                            << THIS READS A>>
        SET,A,0,DSX;                      <<SAVE DSX OF A IN LOCAL SYMBOL>>
        GTDSX;                            <<NOW READ 'SAME AS'>>
        JMP,LOC+3,DSX = 'SAMEAS';         <<BE SURE IT REALLY IS 'SAMEAS'>>
        MNOTE,"****  ONLY 'SAME AS' IS ALLOWED";
        MEXIT;                            <<LEAVE HIM AT THE MERCY OF JSCAN>>
        GTDSX;                            <<READ B>>
        SET,IND.(A),1,DSX(1);             <<SET THE CLASS CODE OF A>>
        SET,IND.(A),2,DSX(2);             <<SET THE SYNTACTIC CLASS OF A>>
        SET,IND.(A),3,DSX(3);             <<SET THE LEFT PRECEDENCE OF A>>
        SET,IND.(A),4,DSX(4);             <<SET THE RIGHT PRECEDENCE OF A>>
        GTDSX;                            <<READ THE STATEMENT TERMINATOR>>
        MEND,'DECLARE SYNTACTIC CLASS';           <<ALL DONE>>
        END;                              <<BACK INTO MAD/I>>
<<
        THE FOLLOWING DEFINES A SPECIAL PURPOSE DEFINITIONAL FACILITY
        WHICH ALLOWS THE DEFINITION OF A SIMPLE STATEMENT.  THE DEFINITIONAL
        STATEMENT IS OF THE FORM

             'DEFINE STATEMENT' KEYWORD LIST OF VARIABLES
                 LIST OF STATEMENTS
             'END STATEMENT'

        THE NEWLY DEFINED STATEMENT HAS THE FORM

             KEYWORD  LIST OF EXPRESSIONS

        EACH EXPRESSION WHICH WILL OCCUR IN AN INSTANCE OF THE DEFINED
        STATEMENT IS REPRESENTED BY A VARIABLE IN THE PROTOTYPE OF THE DEFINE
        STATEMENT.  IN THE CODE GENERATED FOR THE STATEMENT THE CODE FOR
        THE EXPRESSIONS WILL BE GENERATED FIRST, FOLLOWED BY THE CODE FOR THE
        MAD/I STATEMENTS SPECIFIED IN THE DEFINITION.  ALL OCCURENCES OF THE
        VARIABLES CORRESPONDING TO THE EXPRESSIONS ARE REPLACED BY THE
        RESULT OF THE CORRESPONDING EXPRESSION.
>>
        'DECLARE SYNTACTIC CLASS' 'DEFINE STATEMENT' 'SAME AS' 'DEFINE';
        'DEFINE';                         <<ESCAPE INTO THE METALANGUAGE>>
```

```
        CREATE,DEFSTATLAB,DFSTLB0000,PERCLS(@VAL);   <<SETS UP CRS>>
        MACRO,'DEFINE STATEMENT';
        LOCAL,KEYWD,VLIST,SLIST,NVLIST,I,Q;
        GTDSX;                           << GET THE KEYWORD>>
        SETUP,DSX,14,3,4;                <<MAKE IT SIMPLE STATEMENT >>
        SET,KEYWD,0,DSX;                 <<KEYWD POINTS TO THE KEYWORD>>
        SET,KEYWD,1,INDCLS(@VAL);        <<AND IT IS INDIRECT>>
A:      GTDSX;                           << GET A VARIABLE NAME>>
        JMP,B,DSX = LN; ;                <<SEMICOLON MARKS THE END>>
        SETLST,VLIST,,DSX;               << PUT THE VARIABLE INTO VLIST>>
        GTDSX;                           << GET THE COMMA OR SEMICOLON>>
        JMP,A,DSX ¬= LN; ;               <<THERE IS ANOTHER VARIABLE>>
B:      CRS,DEFSTATLAB,NOTIND.(DEFSTATLAB);      << A NEW LIST TO HOLD STM>>
        SET,SLIST,0,DEFSTATLAB;          <<SAVE THE LIST NAME>>
        SET,SLIST,1,INDCLS(@VAL);        << MAKE IT INDIRECT>>
        CRS,DEFSTATLAB,NOTIND.(DEFSTATLAB);      <<UNIQUE SYM FOR HIS VAR>>
        SETLST,NVLIST,,DEFSTATLAB;       <<BUILD A LIST OF THEM>>
        JMP,LOC-2,NVLIST(0) < VLIST(0);      <<WANT ONE FOR EACH VARIABLE>>
C:      GTDSX;                  <<READ A STATEMENT DSX>>
        JMP,D,DSX = 'END STATEMENT';     <<CHECK FOR END OF DEFINITION>>
        SET,I,0,1;                       <<NOW CHECK FOR IT IN THE VARIABLES>>
        JMP,E,VLIST(I(0)) = DSX;         <<JUMP IF FOUND IT>>
        SET,I,0,I(0)+1;                  <<UP THE ANTE>>
        JMP,LOC-2,I(0) <= VLIST(0);      <<CONTINUE IF STILL MORE VARIABLES>>
        SETLST,SLIST,,DSX;               << ADD HIS DSX TO THE STREAM>>
        JMP,C;                           << PROCESS THE NEXT DSX>>
E:      SET,NOTIND.(DSX),0,NVLIST(I(0));     <<PERFORM THE SUBSTITUTION>>
        JMP,LOC-3;                       << AND NOW DO IT TO THIS>>
D:      SETLST,SLIST,,'END';             <<PUT A 'END' ON THE END >>
        SETLST,SLIST,,LN;;               << AND FINISH UP WITH A SEMICOLON >>
        MACRO,KEYWD;                     <<NOW DEFINE THE MACRO FOR THE KEYWD>>
        DESIST;
        SET,I,0,1;                <<GENERATE AN EXPRESSION SCAN FOR EACH VAR>>
        JMP,LOC+10,I(0) > VLIST(0);      << STOP IF END OF LIST>>
        SET,NOTIND.(Q),0,NVLIST(I(0));       <<Q WILL BE THE VARIABLE NAME>>
        SET,NOTIND.(Q),1,INDCLS(@VAL);
        RESUME;
        JSCAN,.TAG.,LIST;         <<SCAN OFF AN EXPRESSION>>
        SET,Q,0,EXP.(1);                 <<MAKE THE VARIABLE THE RESULT>>
        SET,Q,1,INDCLS(@VAL);
        DESIST;
        SET,I,0,I(0)+1;                  <<UP TO THE NEXT VARIABLE>>
        JMP,LOC-9;                       << AROUND AND AROUND>>
        RESUME;
        SET,GTDSXLIST,0,SLIST;           << REDEFINE GTDSX FOR AWHILE >>
        BLOCK;                           << THESE LOOK LIKE SCOPE OF STM>>
        MEND,KEYWD;                      <<END THE GENERATED MACRO>>
        GTDSX;                           << GET OUR SEMICOLON>>
        MEND,'DEFINE STATEMENT';
        END;                             <<BACK INTO MAD/I>>
<<
        THE FOLLOWING REDEFINES THE PSEUDO OP FOR GTDSX.  THE NEW GTDSX ACTS
        EXACTLY LIKE THE OLD DSX EXCEPT THAT IT WILL INSERT THE CONTENTS
        OF THE LIST POINTED TO BY GTDSXLIST INTO THE INPUT STREAM.  THIS
        ALLOWS A SEQUENCE OF DESCRIPTORS TO BE RETURNED TO JSCAN AS IF THEY
        CAME FROM THE INPUT STREAM
>>
        'DEFINE';
        SET,PREVGTDSX,0,GTDSX(0);        << THIS IS NOW THE REAL GTDSX >>
        SET,PREVGTDSX,1,GTDSX(1);
```

```
SET,GTDSXLIST,@SIZE,0;          << INITIALIZE GTDSX CHEAT LIST >>
SET,GTDSXLIST,0,0;              << USE REAL GTDSX FOR THE TIME BEING >>
MACRO,GTDSX;                    << NOW REDEFINE GTDSX >>
JMP,LOC+3,GTDSXLIST(0) ¬= 0;    << JUMP IF INSERTIONS TO BE MADE >>
PREVGTDSX;                      << USE THE OLD-FASHION GTDSX >>
MEXIT;
SET,GTDSXLIST,@SIZE,GTDSXLIST(@SIZE)+1; << UP THE LIST INDEX >>
JMP,LOC+5,GTDSXLIST(@SIZE) <= (IND.(GTDSXLIST(0)))(0);
SET,GTDSXLIST,@SIZE,0;          << LIST PROCESSED SO RESET >>
SET,GTDSXLIST,0,0;
GTDSX;                          << NOW TRY READING AGAIN >>
MEXIT;
SET,NOTIND.(DSX),0,DSXOF.((IND.(GTDSXLIST(0)))(GTDSXLIST(@SIZE)));
SET,NOTIND.(DSX),1,INDCLS(0);        << SET UP DSX >>
MEND,GTDSX;
END;
<<
THE FOLLOWING MACRO CREATES A FLOATING ADDRESS CORRESPONDING TO
EACH PARAMETER
>>
'DEFINE';
MACRO,FLAD;
LOCAL,I;
SET,I,0,1;
JMP,LOC+2,PAR.(0) >= I(0);
MEXIT;
CRS,FLD,NOTIND.(FLD);
SET,FLD,1,NOTIND.(FLD)(@MOD);
SET,PAR.(I(0)),0,FLD;
SET,PAR.(I(0)),1,INDCLS(0);
SET,I,0,I(0)+1;
JMP,LOC-7;
MEND,FLAD;
END;
<<
THE FOLLOWING MACRO CREATES A TEMPORARY CORRESPONDING TO EACH PARAMETER
>>
'DEFINE';
MACRO,TEMPORARY;
LOCAL,I;
SET,I,0,1;
JMP,LOC+2,PAR.(0) >= I(0);
MEXIT;
CRS,TMP,NOTIND.(TMP);
SET,TMP,1,NOTIND.(TMP)(@MOD);
SET,PAR.(I(C)),0,TMP;
SET,PAR.(I(0)),1,INDCLS(0);
SET,I,C,I(0)+1;
JMP,LOC-7;
MEND,TEMPORARY;
END;
<<
THE FOLLOWING REDEFINES THE GETTEMP AND FREETEMP MACROS TO
GET AROUND THE PROBLEM OF THE REASSIGNMENT OF THE STORAGE
ALLOCATION OF A TEMPORARY.
>>
'DEFINE';
MACEXCTYPE,7;
MACDEFTYPE,2;
POPMACRO,GETTEMP,FREETEMP;
```

```
                ATR,@TEMP1,EXTENDED;
                ATR,@TEMP2,LOCAL,20,4;
                SET,TEMPLST,@VAL,0;

                MACRO,GETTEMP,S,MODE,LEN;
                LOCAL,I,J;
                JMP,LOC+4,LEN <= 16;
ERR:            MNOTE,"****  GETTEMP CALLED FOR MORE THAN SIXTEEN BYTES.";
                DUMPDSX,S,MODE,LEN;
                MEXIT;
                CLEAR,S;
                SET,I,@VAL,5;
                SET,J,@VAL,0;
TEST:           JMP,DONE,I > TEMPLST;
                JMP,LOC+3,TEMPLST(I)(@TEMP2) ¬= 0;
                SET,J,@VAL,I;
                JMP,SEARCH;
                JMP,SEARCH,TEMPLST(I)(@TEMP2) ¬= DSXOF.(S);
GOOD:           SET,TEMPLST(I),@TEMP2,DSXOF.(S);
                SET,S,@XA,1;
                SET,S,@MODE,MODE(@MODE);
                SET,S,@LEN,LEN;
                SET,S,@VAL,TEMPLST(I);
                MEXIT;
SEARCH:         SET,I,@VAL,I+1;
                JMP,TEST;
DONE:           JMP,NO,J = 0;
                SET,I,@VAL,J;
                JMP,GOOD;
NO:             APND,TEMPLST,1,I;
                SET,TEMPLST,@VAL,I;
                SPACE,TEMPLST(I),16,8;
                JMP,GOOD;
                MEND,GETTEMP;

                MACRO,FREETEMP,S;
                LOCAL,I;
                RELSYMB,S;
                SET,I,@VAL,5;
                JMP,NO,I > TEMPLST;
                JMP,GOTIT,TEMPLST(I)(@TEMP2) = DSXOF.(S);
                SET,I,@VAL,I+1;
                JMP,LOC-3;
GOTIT:          SET,TEMPLST(I),@TEMP2,0;
NO:             SET,S,@VAL,0;
                SET,S,@XA,1;
                SET,S,@SIZE,0;
                SET,S,@CLS,NOTIND.(TMP)(@MOD);
                MEND,FREETEMP;

                MACDEFTYPE,1;
                MACEXCTYPE,1;
                END;
<<
                THE FOLLOWING DEFINES THE STATEMENT

                     'CREATE POINT' POINT,X,Y

                WHICH HAS THE EFFECT OF THE LIST OF STATEMENTS SHOWN
>>
```

```
'DEFINE STATEMENT' 'CREATE POINT' POINT,X,Y;
     'ALLOCATE' POINT;
     POINT(1) := HEAD;
     HEAD := .PT. POINT;
     POINT(2) := (0 .AS. ('POINTER'));
     DISPLAYN := DISPLAYN+1;
     POINT(3) := DISPLAYN;
     POINT(4) := 1;
     POINT(5) := X;
     POINT(6) := Y;
'END STATEMENT';
<<
THE FOLLOWING DEFINES THE STATEMENT

          'CREATE LINE' LINE,P1,P2

WHICH HAS THE SAME EFFECT AS THE LIST OF STATEMENTS SHOWN
>>
'DEFINE STATEMENT' 'CREATE LINE' LINE,P1,P2;
     'ALLOCATE' LINE;
     LINE(1) := HEAD;
     HEAD := .PT. LINE;
     LINE(2) := (0 .AS. ('POINTER'));
     DISPLAYN := DISPLAYN+1;
     LINE(3) := DISPLAYN;
     LINE(4) := 2;
     LINE(5) := .PT. P1;
     LINE(6) := .PT. P2;
'END STATEMENT';
<<
THE FOLLOWING DEFINES THE STATEMENT

          'CONNECT' POINT 'TO' PICTURE

WHICH HAS THE SAME EFFECT AS THE LIST OF STATEMENT SHOWN
>>
'DECLARE SYNTACTIC CLASS' 'TO' 'SAME AS' , ;
'DEFINE STATEMENT' 'CONNECT' POINT 'TO' PICTURE;
     QQ1 := PICTURE;
     'IF' (QQ1 .AS. ('INTEGER')) = 0;
          PICTURE := .PT. POINT;
          POINT(2) := .PT. POINT;
     'ELSE';
          QQSV .EVAL. PICTURE;
          POINT(2) := QQSV(2);
          QQSV(2) := .PT. POINT;
     'END';
'END STATEMENT';
<<
THE FOLLOWING MACRO DEFINES THE .POINT. OPERATOR AS A PASS ONE MACRO.
IT CAUSES .POINT. A TO BE TREATED AS A(4) = 1
>>
'DECLARE SYNTACTIC CLASS' .POINT. 'SAME AS' .ABS.;
'DEFINE';
MACRO,.POINT.,T,B;
LOCAL,U;
TEMPORARY,U;
.TAG.,U,B,'LOCALLITERAL' 4;
LN=,T,U,'LOCALLITERAL' 1;
MEND,.POINT.;
```

```
            END;
      <<
            THE FOLLOWING MACRO DEFINES .XOF. AS A PASS ONE OPERATOR.
            IT CAUSES .XOF. A TO BE TREATED AS A(5)
      >>
            'DECLARE SYNTACTIC CLASS' .XOF. 'SAME AS' .ABS.;
            'DEFINE';
            MACRO,.XOF.,T,A;
            .TAG.,T,A,'LOCALLITERAL' 5;
            MEND,.XOF.;
            END;
      <<
            THE FOLLOWING CAUSES .YOF. TO BE DEFINED AS A PASS ONE OPERATOR
            .YOF. A HAS THE SAME EFFECT AS A(6)
      >>
            'DECLARE SYNTACTIC CLASS' .YOF. 'SAME AS' .ABS.;
            'DEFINE';
            MACRO,.YOF.,T,A;
            .TAG.,T,A,'LOCALLITERAL' 6;
            MEND,.YOF.;
            END;
      <<
            THE FOLLOWING DEFINES .NEXT. A TO BE THE SAME AS A(2)
      >>
            'DECLARE SYNTACTIC CLASS' .NEXT. 'SAME AS' .ABS.;
            'DEFINE';
            MACRO,.NEXT.,T,A;
            .TAG.,T,A,'LOCALLITERAL' 2;
            MEND,.NEXT.;
            END;
      <<
            THE FOLLOWING DEFINES .DISPN. A TO BE THE SAME AS A(3)
      >>
            'DECLARE SYNTACTIC CLASS' .DISPN. 'SAME AS' .ABS.;
            'DEFINE';
            MACRO,.DISPN.,T,A;
            .TAG.,T,A,'LOCALLITERAL' 3;
            MEND,.DISPN.;
            END;
            'DECLARE SYNTACTIC CLASS' .ENDA. 'SAME AS' .ABS.;
            'DEFINE';
            MACRO,.ENDA.,T,A;
            .TAG.,T,A,'LOCALLITERAL' 5;
            MEND,.ENDA.;
            END;
      <<
            THE FOLLOWING DEFINES .ENDB. A TO BE THE SAME AS A(6)
      >>
            'DECLARE SYNTACTIC CLASS' .ENDB. 'SAME AS' .ABS.;
            'DEFINE';
            MACRO,.ENDB.,T,A;
            .TAG.,T,A,'LOCALLITERAL' 6;
            MEND,.ENDB.;
            END;
      <<
            THE FOLLOWING DEFINES .HEAD. A TO BE THE SAME AS A(1)
      >>
            'DECLARE SYNTACTIC CLASS' .HEAD. 'SAME AS' .ABS.;
            'DEFINE';
            MACRO,.HEAD.,T,A;
```

```
            .TAG.,T,A,'LOCALLITERAL' 1;
            MEND,.HEAD.;
            END;
<<
            THE FOLLOWING DEFINES .ADROF. A TO BE THE SAME AS THE SEQUENCE OF
            STATEMENTS
                  QQSV .EVAL. HEAD
            B1:   'IF' .DISPN. QQSV = A, 'GO TO' B2
                  QQSV .EVAL. (.HEAD. QQSV)
                  'GO TO' B1
            B2:   'RETURN' .PT. QQSV
>>
            'DECLARE SYNTACTIC CLASS' .ADROF. 'SAME AS' .ABS.;
            'DEFINE';
            MACRO,.ADROF.,T,A;
            LOCAL,B1,B2;
            LOCAL,T1,T2,T3,T4,T5;
            TEMPORARY,T1,T2,T3,T4,T5;
            FLAD,B1,B2;
            .EVAL.,T1,%QQSV,%HEAD;
            DES,B1;
            .DISPN.,T2,%QQSV;
            LN=,T3,T2,A;
            TNZ,B2,T3;
            .HEAD.,T4,%QQSV;
            .EVAL.,T5,%QQSV,T4;
            GOTO,B1;
            DES,B2;
            .PT.,T,%QQSV;
            MEND,.ADROF.;
            END;
<<
                      END OF DEFINITIONAL PACKAGE
>>

            'DECLARE' 'NORMAL MODE' 'INTEGER';
            'DECLARE' COMMAND 'CHARACTER'(4);
            'DECLARE' DISPLAY 'FIXED ARRAY'(50,50) 'CHARACTER'(1);
            'DECLARE' (POINT,P1,P2,P3) 'POINT';
            'DECLARE' LINE 'LINE';
            'DECLARE' PICTURE 'FIXED ARRAY'(100) 'PICTURE';
            'DECLARE' QQ1 'POINTER';
            'DECLARE' QQ2 'POINTER';
            'DECLARE' (M,M1,M2) 'FLOATING';

            'PRESET' PICTUREN := 0;

SKETCH:     'WRITE' , "'0ENTER A COMMAND PLEASE.'*";
            'READ' , "C4.4*" , COMMAND;

            'IF' COMMAND = "POIN";
                  'WRITE' , "' ENTER X AND Y COORDINATES:'*";
                  'READ' , "2I*", X,Y;
                  'IF' 1 <= X & 50 >= X & 1 <= Y & 50 >= Y;
                        'CREATE POINT' POINT,X,Y;
                        'WRITE' , "' ASSIGNED DISPLAY NUMBER',HI4*", .DISPN. POINT;
                  'ELSE';
                        'WRITE' , "' POINT IS OUTSIDE RASTER RANGE.'*";
                  'END';
```

```

         'OR IF' CCMMAND = "LINE";
              'WRITE' , "' ENTER DISPLAY NUMBERS FOR END-POINTS:'*";
              'READ' , "2I*", X,Y;
              P1 .EVAL. .ADROF. X; P2 .EVAL. .ADROF. Y;
              'IF' .POINT. P1 & .POINT. P2;
                   'CREATE LINE' LINE,P1,P2;
              'ELSE';
NOTPOINT:          'WRITE' , "' THOSE ARE NOT POINTS.'*";
              'END';

         'OR IF' CCMMAND = "PICT";
              'WRITE' , "' ENTER DISPLAY NUMBERS FOR ALL POINTS.'*";
              PICTUREN := PICTUREN+1;
              'IF' PICTUREN > 100;
                   'WRITE' , "' TOC MANY PICTURES.'*";
              'ELSE';
                   PICTURE(PICTUREN) := 0 .AS. ('POINTER');
PICTA:             'READ' , "I*", X;
                   'IF' X¬= 0;
                        P1 .EVAL. .ACROF. X;
                        'IF' ¬.POINT. P1, 'GO TO' NOTPOINT;
                        QQ2 := PICTURE(PICTUREN);
                        'CONNECT' P1 'TO' QQ2;
                        PICTURE(PICTUREN) := QQ2;
                        'GO TO' PICTA;
                   'END'
              'END';

         'OR IF' CCMMAND = "MOVE";
              'WRITE' , "' ENTER DISPLAY NUMBER OF POINT AND NEW X,Y'*";
              'READ' , "3I*", DIS,X,Y;
              P1 .EVAL. .ADROF. DIS;
              'IF' ¬.POINT. P1, 'GCTC' NOTPOINT;
              DX := X-.XOF. P1; DY := Y-.YOF. P1;
              (.XOF. P1) := .XOF. P1+DX; (.YOF. P1) := .YOF. P1+DY;
              QQ1 := P1(2);
              'IF' QQ1 .AS. ('INTEGER') ¬= 0;
                   P2 .EVAL. .NEXT. P1;
MOVEA:             'IF' .DISPN. P1 ¬= .DISPN. P2;
                        (.XOF. P2) := .XOF. P2+DX; (.YOF. P2) := .YOF. P2
                                                   +DY;
                        P2 .EVAL. .NEXT. P2;
                        'GO TO' MOVEA;
                   'END'
              'END';

         'OR IF' CCMMAND = "DISP";
              'IF' HEAD .AS. ('INTEGER') = 0;
                   'WRITE' , "' NOTHING TO DISPLAY.'*";
                   'GO TO' SKETCH;
              'END';
              'FOR' I := 1,1,I>50, 'FOR' J := 1,1,J>50, DISPLAY(I,J) := " ";
              P1 .EVAL. HEAD;
DISPA:        ;
              'IF' .POINT. P1;
                   Q1 := .XOF. P1; Q2 := .YOF. P1; DISPLAY(Q2,Q1) := "*";
              'ELSE';
                   LINE .EVAL. (.PT. P1);
                   QQ1 := .ENDA. LINE; P2 .EVAL. QQ1;
```

```
                   QQ1 := .ENDB. LINE; P3 .EVAL. QQ1;
                   X1 := MIN.(.XOF. P2,.XOF. P3);
                   X2 := MAX.(.XOF. P2,.XOF. P3);
                   'IF' X1 = X2;
                        Y1 := MIN.(.YOF. P2,.YOF. P3);
                        Y2 := MAX.(.YOF. P2,.YOF. P3);
                        'FOR' Y := Y1,1, Y > Y2, DISPLAY(Y,X1) := "*";
                   'ELSE';
                        M1 := .YOF. P3 - .YOF. P2;
                        M2 := .XOF. P3 - .XOF. P2;
                        M := M1/M2;
                        'FOR' X := X1,1,X > X2;
                             Q2 := M*(X - .XOF. P2) + .YOF. P2;
                             DISPLAY(Q2,X) := "*";
                        'END';
                   'END';
              'END';
              QQ1 := .HEAD. P1;
              'IF' QQ1 .AS. ('INTEGER') ¬= 0;
                   P1 .EVAL. .HEAD. P1;
                   'GO TO' DISPA;
              'END';
              'WRITE' , "'1   .',10('    .')*";
              'FOR' I := 50,-1, I < 1,
                   'WRITE' , "I3,52C1.1*", I," ",".",
                                        DISPLAY(I,1)...DISPLAY(I,50);
              'WRITE' , "'    .',10('    .')*";

         'ELSE';
              'WRITE' , "' ILLEGAL COMMAND'*";
         'END';

         'GO TO' SKETCH;

         'PROCEDURE' MIN.(X,Y);
         'INTEGER SHORT' (X,Y);
MIN:     'IF' X <= Y, 'RETURN' X;
         'RETURN' Y;
         'END';

         'PROCEDURE' MAX.(A,B);
         'INTEGER SHORT' (A,B);
MAX:     'IF' A >= B, 'RETURN' A;
         'RETURN' B;
         'END';

         'END'
```

```
000000                             ##SKETCH CSECT
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
000084 00000000                +          CCNST 0
0000B4 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 7D40E3D6D640D4C1        +          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 NCT 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 C3F44BF45C              +          CCNST "C4.4*"
000432 7DF0C5D5E3C5D940        +          CCNST "'0ENTER 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 0000 00000004 Y1
01 01 0000 00000008 MAX
01 01 0000 00000010 X2
01 01 0000 00000014 MIN
01 01 0000 0000001C X1
01 01 0000 00000020 Q2
01 01 0000 00000024 Q1
01 01 0000 00000028 DISPA
01 01 0000 00000030 J
01 01 0000 00000034 I
01 01 0000 00000038 MOVEA
01 01 0000 00000040 DY
01 01 0000 00000044 DX
01 01 0000 00000048 DIS
01 01 0000 0000004C PICTA
01 01 0000 00000054 NOTPOINT
01 01 0000 0000005C IOP
01 01 0000 00000064 MADREAD
01 01 0000 0000006C ENDIOP
01 01 0000 00000074 FORMAT
01 01 0000 0000007C MADWRITE
01 01 0000 00000084 PICTUREN
01 01 0000 00000088 M2
01 01 0000 0000008C M1
01 01 0000 00000090 M
01 01 0000 00000094 QQ2
01 01 0000 00000098 COMMAND
01 01 0000 0000009C QQ1
01 01 0000 000000A0 'ENDSTATEMENT'
01 01 0000 000000A4 Y
01 01 0000 000000A8 X
01 01 0000 000000AC 'SAMEAS'
01 01 0000 000000B0 HEAD
01 01 0000 000000B4 DISPLAYN
01 01 0000 000000B8 SKETCH
01 01 0000 0000029C "' ILLEGAL COMMAND'*"
01 01 0000 000002AF "'     .',10('     .')*"
01 01 0000 000002C3 "."
01 01 0000 000002C4 "13,52C1.1*"
01 01 0000 000002CE "'1    .',10('     .')*"
01 01 0000 000002E2 "*"
01 01 0000 000002E3 " "
01 01 0000 000002E4 "' NOTHING TO DISPLAY.'*"
01 01 0000 000002FB "DISP"
01 01 0000 000002FF "3I*"
01 01 0000 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 "' THESE 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 00000404 50
01 01 0000 00000408 "2I*"
01 01 0000 0000040B "' ENTER X AND Y COORDINATES:'*"
01 01 0000 00000429 "POIN"
01 01 0000 0000042D "C4.4*"
01 01 0000 00000432 "'0ENTER 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 %DIM0002
01 01 0000 00000480 %DIM0001
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
ENDIOP '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'  0007001C 0000000C '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'  01C10C00 00000C84
POINT 'COMPONENTSTRUCTURE'  0007002C 00000C00 'BASED'
     SIZE=16
   COMPONENT= 'PCINTER'
   COMPONENT= 'POINTER'
   COMPONENT= 'INTEGERSHORT'
   COMPONENT= 'INTEGERSHORT'
   COMPONENT= 'INTEGERSHORT'
   COMPONENT= 'INTEGERSHORT'
P1 'COMPONENTSTRUCTURE'  0C070018 CCCC000C 'BASED'
     SIZE=16
   COMPONENT= 'PCINTER'
   COMPONENT= 'POINTER'
   COMPONENT= 'INTEGERSHORT'
   COMPONENT= 'INTEGERSHORT'
   COMPONENT= 'INTEGERSHORT'
   COMPONENT= 'INTEGERSHORT'
P2 'COMPONENTSTRUCTURE'  00070014 00000000 'BASED'
     SIZE=16
   COMPONENT= 'PCINTER'
   COMPONENT= 'POINTER'
   COMPONENT= 'INTEGERSHORT'
   COMPONENT= 'INTEGERSHORT'
   COMPONENT= 'INTEGERSHORT'
   COMPONENT= 'INTEGERSHORT'
P3 'COMPONENTSTRUCTURE'  0C070010 CCCC0000 'BASED'
     SIZE=16
   COMPONENT= 'PCINTER'
   COMPONENT= 'PCINTER'
   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 C000009C
QQ2 'POINTER'  01010000 CCC00094
Q1 'INTEGER'  0101000C CCC00024
Q2 'INTEGER'  01C10000 00000020
SKETCH 'ENTRYNAME'  010100C0 000000B8 'ACCESSIBLE'
   RESULT= 'INTEGER'
X 'INTEGER'  01C1C000 000000A8
X 'INTEGERSHORT'  00030C2C 0000000C (FORMAL PAR)
X1 'INTEGER'  01C10000 CC00001C
X2 'INTEGER'  01C1000C CCCC0010
Y 'INTEGER'  01010000 CCC0C0A4
Y 'INTEGERSHORT'  0003C028 00000000 (FORMAL PAR)
Y1 'INTEGER'  01C1000C C0000004
Y2 'INTEGER'  01C100CC CCCC0000
" " 'CHARACTER'  010100CC 000002E3
     LENGTH=1
"." 'CHARACTER'  C1C100C0 C000C2C3
```

```
    LENGTH=1
"*" 'CHARACTER'  C1010000 C00002E2
    LENGTH=1
"'    .',10('    .')*" 'CHARACTER'  01010000 000002AF
    LENGTH=20
"' ASSIGNED DISPLAY NUMBER',HI4*" 'CHARACTER'  01C10000 COOC03E5
    LENGTH=31
"' ENTER DISPLAY NUMBER CF POINT AND NEW X,Y'*" 'CHARACTER'  01010000 00000302
    LENGTH=45
"' ENTER DISPLAY NUMBERS FCR ALL PCINTS.'*" 'CHARACTER'  01C10000 00000350
    LENGTH=41
"' ENTER DISPLAY NUMBERS FCR ENC-PCINTS:'*" 'CHARACTER'  01C10000 00000396
    LENGTH=41
"' ENTER X AND Y COORDINATES:'*" 'CHARACTER'  01C1000C C000C408
    LENGTH=30
"' ILLEGAL COMMAND'*" 'CHARACTER'  C1C100CC 00C0029C
    LENGTH=19
"' NOTHING TO DISPLAY.'*" 'CHARACTER'  01C10000 000002E4
    LENGTH=23
"' POINT IS OUTSIDE RASTER RANGE.'*" 'CHARACTER'  01010000 000003C3
    LENGTH=34
"' THOSE ARE NOT POINTS.'*" 'CHARACTER'  0101000C C000037D
    LENGTH=25
"' TCC MANY PICTURES.'*" 'CHARACTER'  01C1000C 00000335
    LENGTH=22
"'0ENTER A COMMAND PLEASE.'*" 'CHARACTER'  C1010000 00000432
    LENGTH=27
"'1   .',10('    .')*" 'CHARACTER'  010100CC CC0002CE
    LENGTH=20
"C4.4*" 'CHARACTER'  0101CC00 0000042D
    LENGTH=5
"DISP" 'CHARACTER'  01C10000 000002FB
    LENGTH=4
"I*" 'CHARACTER'  010100CC 00000333
    LENGTH=2
"I3,52C1.1*" 'CHARACTER'  C1010000 C00002C4
    LENGTH=10
"LINE" 'CHARACTER'  01010000 CCC003BF
    LENGTH=4
"MCVE" 'CHARACTER'  C1C10000 0000032F
    LENGTH=4
"PICT" 'CHARACTER'  01010000 00C00379
    LENGTH=4
"POIN" 'CHARACTER'  01010000 00000429
    LENGTH=4
"2I*" 'CHARACTER'  01010CCC 00000408
    LENGTH=3
"3I*" 'CHARACTER'  0101C00C 000002FF
    LENGTH=3
0 'INTEGER'  01010000 COC00460
1 'INTEGER'  01C10000 COC00468
100 'INTEGER'  C1C1000C C000034C
2 'INTEGER'  0101C000 CC000464
3 'INTEGER'  C1C10000 0000045C
4 'INTEGER'  01010000 C0C00458
5 'INTEGER'  01010000 C0000454
5C 'INTEGER'  01C1000C CC000404
6 'INTEGER'  01C10000 0C000450
```

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

```
##SKETCH PD 01 000000 00124C
#SKETCH  SD 02 000000 0016BA
MADSTACK ER 03
SKETCH   LD    000000 000002
MACWRITE ER 04
FORMAT   ER 05
ENDIOP   ER 06
MACREAD  ER 07
IOP      ER 08
GETSPACE ER 09
LINSUB   ER 0A
```

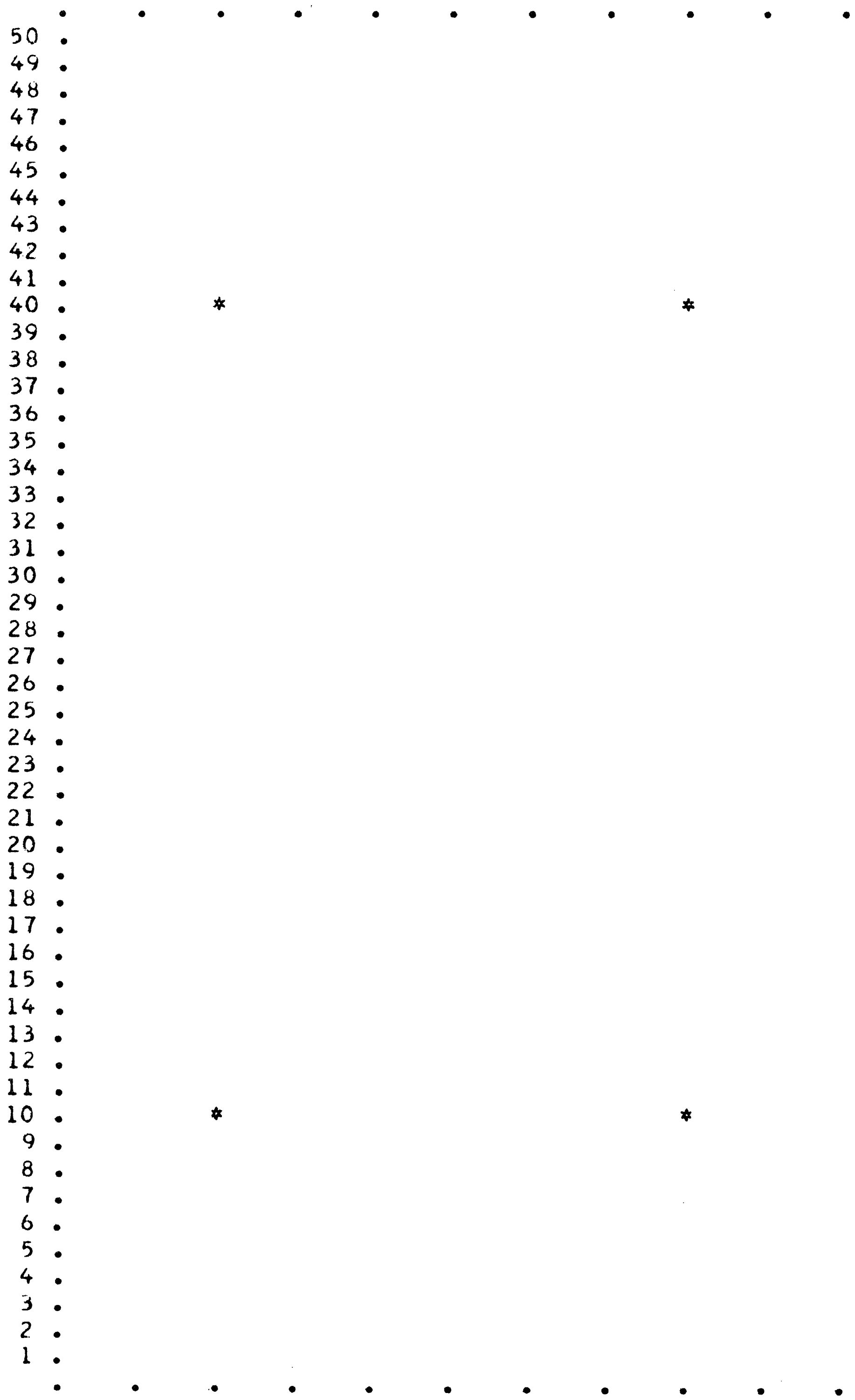
RELOCATION DICTIONARY (P.ID,R.ID,FLAGS,ADDRESS)

01	02	0C	0000B8
01	04	1C	00007C
01	05	1C	000074
01	01	0C	000294
01	06	1C	00006C
01	07	1C	000064
01	01	0C	000288
01	08	1C	00005C
01	01	0C	00027C
01	01	0C	000270
01	01	0C	000264
01	01	0C	000258
01	01	0C	00024C
01	09	1C	001060
01	01	0C	000240
01	01	0C	000228
01	01	0C	00021C
01	01	0C	000210
01	01	0C	000204
01	01	0C	0001F8
01	02	0C	000054
01	01	0C	0001EC
01	01	0C	0001E0
01	01	0C	0001D4
01	0A	1C	0010A4
01	01	0C	001094
01	01	0C	001098
01	02	0C	00004C
01	01	0C	0001C8
01	01	0C	0001BC
01	01	0C	0010B8
01	01	0C	0010BC
01	01	0C	0010D4
01	01	0C	0010D8
01	01	0C	0001B0
01	01	0C	0001A4
01	01	0C	000198
01	01	0C	00018C
01	01	0C	000180
01	02	0C	000038
01	01	0C	000174
01	01	0C	00110C
01	01	0C	001110
01	01	0C	001114
01	02	0C	000028
01	01	0C	001130
01	01	0C	001134
01	01	0C	001138
01	01	0C	00116C
01	01	0C	001164
01	01	0C	001168
01	01	0C	00119C
01	01	0C	0011A0
01	01	0C	0011A4
01	01	0C	000118
01	01	0C	00010C
01	01	0C	000100
01	01	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   0015B4
 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 000000                                        END   SKETCH

      MAD/I COMPILER TIMINGS:                 171.625 CPU SECONDS.
                                              326.063 ELAPSED SECONDS.
EXECUTION TERMINATED
```

Appendix F. Run of the MAD/I Program

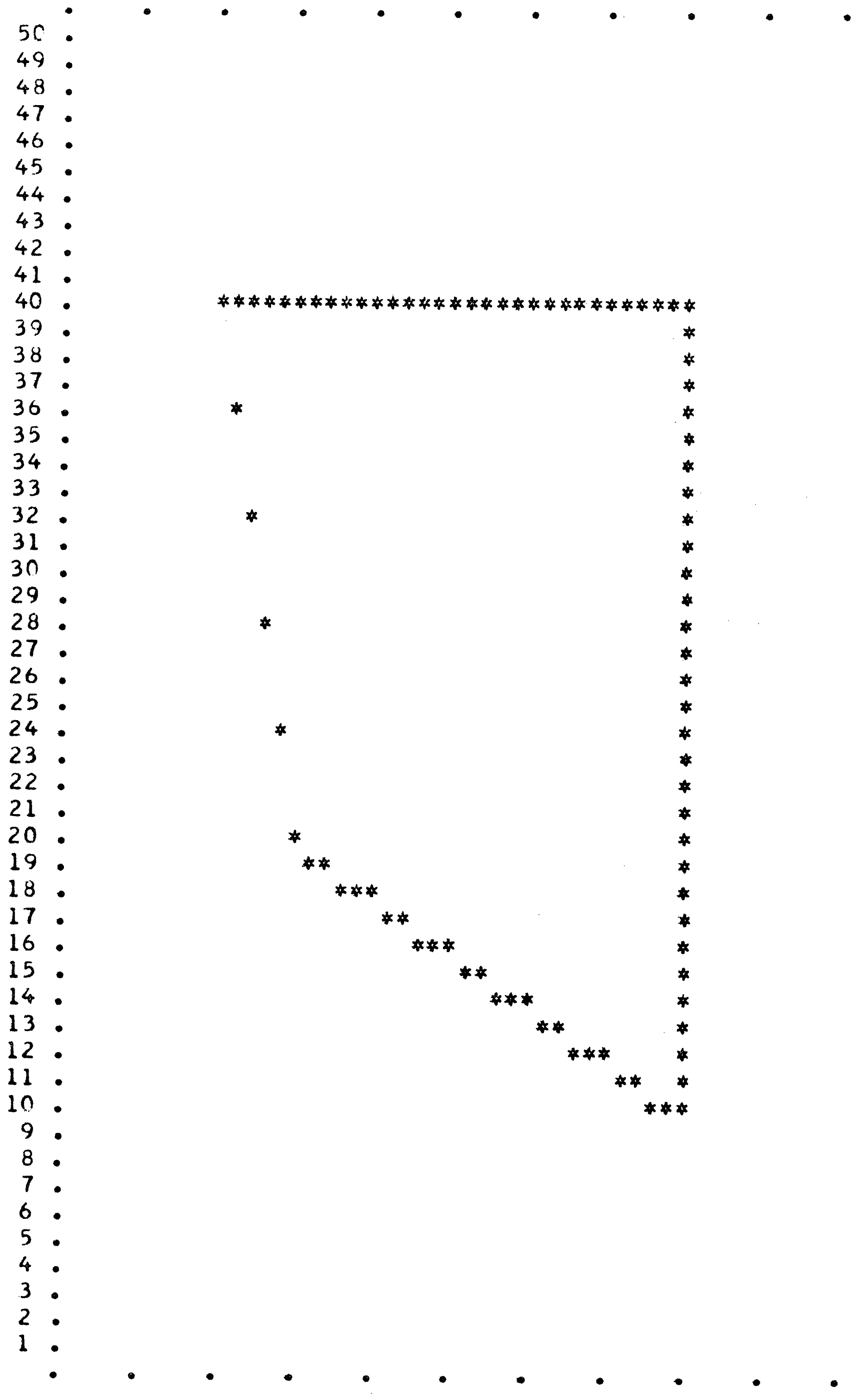
```
$SET ERRORDUMP=ON
$RUN SKETCHOBJ MAP SCARDS=-DATA

 ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ...

 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   0FFA60
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          MDIOPSCT  10BD50   10B9E8 SPIE      10BDE8   10BDE8
#SKETCH   10C000   10C000 IOH360    111000   11100C 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.
```




```
ENTER A COMMAND PLEASE.
ENTER DISPLAY NUMBERS FOR END-PCINTS:

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

ENTER A COMMAND PLEASE.
```

```
ENTER DISPLAY NUMBERS FOR END-POINTS:

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

ENTER A COMMAND PLEASE.
```

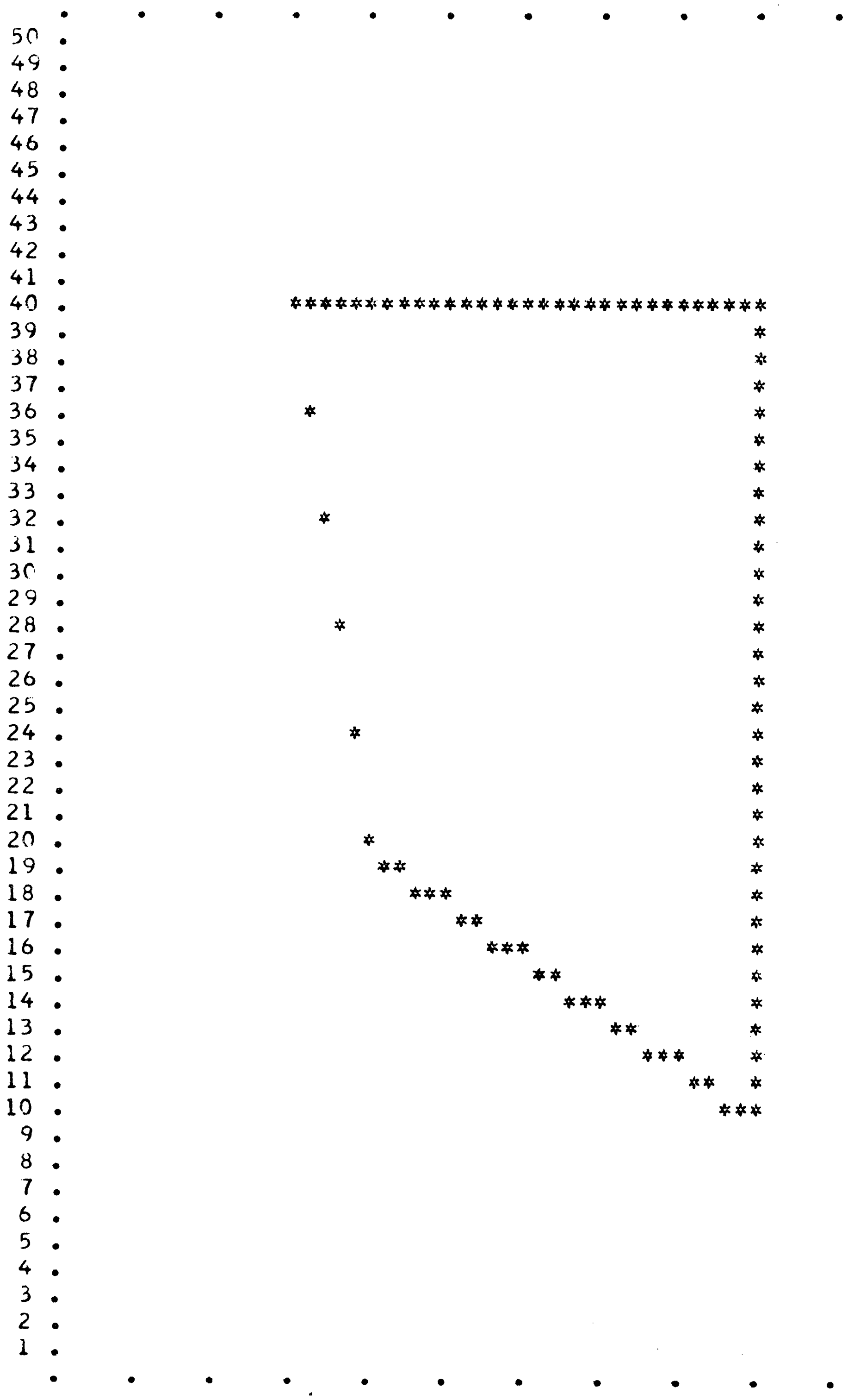
```
   .    .    .    .    .    .    .    .    .    .    .
50 .
49 .
48 .
47 .
46 .
45 .
44 .
43 .
42 .
41 .
40 .         *******************************
39 .         *                             *
38 .         *                             *
37 .         *                             *
36 .         *                             *
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.
```



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

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

ENTER A COMMAND PLEASE.



```
ENTER A COMMAND PLEASE.

**** ALL INPUT DATA HAS BEEN PROCESSEC - AT LOCATION 1C3DE8
EXECUTION TERMINATED
```

UNCLASSIFIED
Security Classification

DOCUMENT CONTROL DATA - R & D

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

1. ORIGINATING ACTIVITY *(Corporate author)* UNIVERSITY OF MICHIGAN CONCOMP PROJECT	2a. REPORT SECURITY CLASSIFICATION Unclassified 2b. GROUP

3. REPORT TITLE

AN EXAMPLE DEFINITIONAL FACILITY IN MAD/I

4. DESCRIPTIVE NOTES *(Type of report and inclusive dates)*
Memorandum

5. AUTHOR(S) *(First name, middle initial, last name)*

Ronald J. Srodawa

6. REPORT DATE	7a. TOTAL NO. OF PAGES	7b. NO. OF REFS
August 1970	25	3

8a. CONTRACT OR GRANT NO. DA-49-083 OSA-3050 b. PROJECT NO. c. d.	9a. ORIGINATOR'S REPORT NUMBER(S) Memorandum 32 9b. OTHER REPORT NO(S) *(Any other numbers that may be assigned this report)*

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

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.

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						

Security Classification