A STRUCTURED LANGUAGE FOR INTERACTIVE GRAPHICS

by

DOUGLAS WILLIAM LLOYD TEEPLE

B.A.Sc., University of Waterloo, 1975

A THESIS SUBMITTED IN PARTIAL FULFILMENT OF
THE REQUIREMENTS FOR THE DEGREE OF
MASTER OF APPLIED SCIENCE

in

Interdisciplinary Studies
(Electrical Engineering
and
Computer Science)

We accept this thesis as conforming to the
required standards

THE UNIVERSITY OF BRITISH COLUMBIA

March, 1977

© D.W.L. Teeple, 1977
In presenting this thesis in partial fulfilment of the requirements for an advanced degree at the University of British Columbia, I agree that the Library shall make it freely available for reference and study. I further agree that permission for extensive copying of this thesis for scholarly purposes may be granted by the Head of my Department or by his representatives. It is understood that copying or publication of this thesis for financial gain shall not be allowed without my written permission.

Department of **INTERDISCIPLINARY STUDIES (ELECTRICAL ENGINEERING, COMPUTER SCIENCE)**

The University of British Columbia
2075 Wesbrook Place
Vancouver, Canada
V6T 1W5

Date **MARCH 17, 1977**
This thesis discusses the design and implementation of the computer language SLING (Structured Language for Interactive Graphics). SLING by design is a high-level structured procedural language. Constructs promoting user interaction have been added to enhance graphics capabilities. The implementation of SLING is core and execution-time efficient relative to current computer graphics languages. It is designed to run on a minicomputer with minimal core requirements.
Table of Contents

0) Introduction ......................................................... 1
   A) Background ....................................................... 1
   B) The Language ..................................................... 3

1) Predecessors ......................................................... 8
   A) LIG ............................................................... 8
   B) IMAGE ........................................................... 10
   C) GRIP ............................................................. 10
   D) ESP3 ............................................................. 11
   E) BGRAF2 .......................................................... 11
   F) DALI ............................................................. 12
   G) SUGAR ............................................................ 12
   H) SKETCHPAD ....................................................... 13

2) Objectives ............................................................ 14

3) Facilities of the UBC Implementation ............................... 16
   A) General Interactive Graphic Facilities ....................... 16
   B) General Control Structures ..................................... 19
      i) WITH ......................................................... 20
      ii) FOR ....................................................... 21
      iii) AS ......................................................... 21
      iv) WHILE ..................................................... 22
      v) CASE ....................................................... 22
      vi) IF .......................................................... 23
Table of Contents

vii) ON ................................................................. 24
C) Transput Structures ........................................... 25
D) The Graphical Function ....................................... 26
E) Semantics of Operators and Assignments .................. 27
F) Type Coercions ................................................ 30
4) Shortcomings .................................................. 32
5) Suggestions for Improvement .................................. 34
6) Summary ........................................................ 35
7) Future Work ..................................................... 36
Appendices .......................................................... 41
   A) Syntax charts ................................................. 41
   B) Backus - Naur Form Syntax .................................. 54
   C) Graphical Data Structure .................................... 58
   D) The Translator ............................................... 63
   E) Reserved Words and Symbols ................................ 65
   F) Running Instructions .......................................... 68
   G) Sample Program Listing ....................................... 70
   H) Sample FORTRAN Object Code ............................... 73
   I) A Review of Language Statements ............................ 77
Index ................................................................. 82
List of Figures

I) INKING Sample Output ........................................ 7
II) Graphical Datum Table ........................................ 60
III) Homogeneous Coordinate Matrices .......................... 62
List of Tables

I) Core Usage Comparison ........................................ 61
Acknowledgement

I would like in particular to thank Dr. G.F. Schrack for his continued encouragement in the thesis work. I would also like to thank Dr. J.E.L. Peck, Dr. B. Pollack, and Dr. M.R. Ito for their constructive criticism in reading early versions of the thesis. Finally I would like to thank various students in the UBC Department of Computer Science for a number of critical discussions of the work and for their suggestions in improving the quality of the report.
0) Introduction

A) Background

In reading the literature on present graphics languages it became apparent that there is still much controversy over the forms that such a language might take. General purpose languages are developing now based on the work of such people as Hoare [1973ab], Dijkstra [1971, 1972], Wirth [1973, 1976] and others. The trend is towards highly structured procedural languages.

Cumulative experience in the use of languages has shown that some control constructs are notably conducive to the writing of code that is difficult to understand, time consuming to debug and difficult to communicate to other programmers. An active research area is concerned with replacing such constructs with control structures that promote understandable expression of a computer algorithm. Hoare [1973b] stated that "a good programming language should give assistance in expressing not only how the program is run, but what it is intended to accomplish; and it should enable this to be expressed at various levels, from the overall strategy to the details of coding and data representation. ... A good programming language will
encourage and assist the programmer to write clear self-documenting code and even perhaps to develop and display a pleasant style of writing".

A number of books have recently been published on programming style. Dijkstra et al. [1972] stress that programming style is of great importance in producing correct programs. His ideal language was to be used as "a communication language" say, not for the communication of algorithms, but for the communication of ways of thinking, as a vehicle of programming style". The efficiency of a programming system is in part dependent on the facility with which a programmer can express an algorithm in a given language. The facility of expression is dependent on a number of factors, notably the "human-engineering" aspects of a language as cited in recent literature. Weinberg [1971] has made a significant contribution in his book The Psychology of Computer Programming. In this book Weinberg sets a number of goals for improving the design and management of programs. Notably he stresses improvements in the readability and writeability of code, and proposes "egoless" and group programming techniques.

Graphical language designers have adopted some of the recent language constructs and ideas creating some reasonably advanced control and data structures for particular use in computer graphics. The problem is somewhat different than that

Introduction
of conventional languages because the designer is dealing with a number of different graphical input devices (light pen, cursor, etc). An attempt was made in the language developed for this thesis to manage this interaction in a natural way, and to promote the writing of structured code.

B) The Language

The reader should consult Appendix G for a sample program listing and Appendix I for a review of the language statements to make the following discussion more concrete.

A graphical datum can be thought of as being a member of a data type. Variables of this data type (named GRAPHICAL in SLING) have some properties similar to the properties of more common data types such as REAL or INTEGER in that a value is returned in an expression and this value can be assigned. However the graphical data type is distinguished from other data types in that the graphical datum is a recursive record structure. A graphical variable is recursive as it is composed of graphical variable(s). A graphical variable is a record because references to the properties of a graphical variable return values of type REAL. Graphical variables must resolve to primitive display values of line or blank.

A data type called CHARACTER is available in the language
having the properties of a graphical variable except that it must resolve to a string of characters. From this discussion it is clear that a program written in the language can only output titled line drawings.

A program consists of any number of nested, named procedures and/or functions. The outermost level must be a graphical procedure. A procedure is any number of executable statements in the set of permissible statements preceded by data declarations that declare every variable, array, label, literal, external function and procedure used in the program. The Backus–Naur Form syntax defining the permissible statement set is found in Appendix B.

Procedures may be nested to any depth. Any procedure or function called or referenced in a program must be preceded by the body of code constituting that procedure or function. All external procedures, functions, or subroutines must be declared as external in the data declaration. External FORTRAN subroutines may be called from a program.

Graphical, real, integer and Boolean data are not known outside the scope of the procedure in which they are declared except when passed as parameters. Parameter passing follows the conventions of FORTRAN subroutine parameter passing with the exception of data of the type graphical which are passed by
reference only. It is possible to pass reference to a graphical function in SLING. This property is useful in dealing with external procedures.

Graphical arrays may have only one dimension. Subscripting starts from zero.

Character variables must be declared with a maximum length. This length is used to set the length of the string vector where the characters are stored. It is possible for a string to exceed its declared length though the total declared length may not be exceeded.

Graphical functions are defined. A graphical function is a piece of code which returns a graphical value. This value has associated with it all the properties of graphical variables. The functions use space in the symbol table only during execution of the function for is core efficiency. The graphical function takes parameters of any type. Functions may be compiled externally, stored in library files and loaded selectively.

Control structures available in SLING are explained in detail later in the report.

Only free format transput is defined. Graphical and Introduction
character variables can be retrieved and stored in files under programmer control.

The language benefitted from experience derived from LIG (Language for Interactive Graphics) which was developed by Pieke [1973] and Schrack [1976b].

The domain of SLING is two-dimensional line drawings. A storage tube display is used for transput in the UBC implementation. Input devices are a joystick cursor and a keyboard. The display is driven by a NOVA 840, 32K minicomputer running under RDOS 5. The translator, written in XPL resides on an IBM 370 machine running under MTS, the Michigan Terminal System. The FORTRAN object produced by the translator is compiled on the minicomputer.

The following section introduces a number of graphical programming languages. The following discussion serves only to introduce the reader to the languages, no attempt is made for a complete description. Readers should consult the references given for a more complete discussion.
Figure I

Sample Output

Sample output from the "INKING" program used in system debugging. See appendix G for the source program.
1) PREDECESSORS

A) LIG (Language for Interactive Graphics)

LIG was developed at the University of British Columbia by Pieke [1973] and Schrack[1976b]. It is a high-level user-oriented language for interactive graphics.

Because LIG is a direct predecessor of SLING, the bulk of the discussion on other graphics programming languages concerns LIG. It is oriented towards running graphical programs on a 32K core minicomputer. It was developed for use by the Department of Electrical Engineering at the University of British Columbia for the development of circuit analysis, critical path, and similar programs. This discussion of LIG presupposes knowledge of the language. See Pieke [1973] and Schrack [1976b] for a discussion in greater depth.

LIG is a superset of the host language FORTRAN. The user who knows FORTRAN has only to learn a few more graphical extensions to learn LIG. All of the familiar FORTRAN control structures are available in LIG. While learning time can be reduced by such an approach and the implementation is faster and easier than the implementation of a full graphics language,
there are some disadvantages. In particular, the user has available only the control structures of FORTRAN. That means that none of the newer language constructs of structured languages are possible. SLING attempts to alleviate this weakness by discarding the host language concept in favour of a complete language design and implementation.

The LIG subroutine system is large, over thirty routines are needed in the supporting system. As a result, the size of the user program is restricted. The user must resort to complicated and time consuming overlay techniques to run a moderate sized program.

The graphical database used by LIG is very large. It takes 3.6K words of core space to store the graphical data structure. This amount of core is used whether the source program uses no graphical variables or the maximum of thirty nine. If a user wants to use fewer graphical variables presumably he wants to use more variables of another type (e.g. REAL or INTEGER). However the large core overhead of the graphical data structure must be paid.

LIG incorporates six graphical primitives - LINE, BLANK, CIRCLE, SQUARE, TRIANGLE and SEMICIRCLE. These are sometimes useful in that the user is saved from forming them himself. However the user pays for the primitives in core space whether

Predecessors
or not they are used in the program. A different approach has been taken in the current design in which function calls accomplish the benefits of a large set of primitives without the disadvantage mentioned.

B) IMAGE

IMAGE (O'Brien and Bown[1975]) was designed as a high-level, portable, interactive, structured language. It is particularly suited to the handling of interrupts from the user. The language is structured around OBJECT and ACTION blocks relating a particular set of actions to a block of object code. The control structures are Algol W-like. The syntax is not well human-engineered. The language uses some constructs like data initialization from FORTRAN and modifiers somewhat like those of LIG.

C) GRIP

GRIP, Graphical Procedures for Instruction Purposes (Giloi [1975]) uses data structures suitable for implementation in a number of languages, either as a subroutine system, or in conjunction with a preprocessor. The author discusses the data structures mainly, giving little attention to implementation.
The language is quite large and not suitable for minicomputer applications. The syntax is very poorly human-engineered although little source code is needed to generate pictures.

D) ESP³

ESP³ (Shapiro [1975]) is an extension to the language SNOBOL (Extended SNOBOL Picture Pattern Processor). It was designed to "provide simple natural and efficient manipulation of line drawings". Simple and efficient are defined in large machine terms; the programs are too large for minicomputer applications. The language uses predicates to retrieve attribute information on the graphical data structure. Assignment and union operators as well as scaling and transformation functions are available. Very little control structure is noted in the paper. The language handles matching of picture patterns very well. It is oriented towards Artificial Intelligence applications.

E) BGRAF2

BGRAF2 (Bergman and Kaufman [1976]) uses graphical procedures, but not graphical functions. The language is very terse, making it difficult to understand programs. One strong advantage of BGRAF2 is its recursive feature, a feature that would be valuable in SLING, but difficult to implement because
of the object language chosen.

F) DALI

DALI (Pfister [1976]) is oriented to providing the motion attribute for graphical data in a natural way. Much of the discussion is taken up by the scheduling algorithms. A graphical function is defined, it returns a value (non-graphical) that is ignored. DALI can be embedded in FORTRAN, PL/I, ALGOL, and LISP. The author chose MUDDLE, a LISP-like language available on the PDP10.

G) SUGAR

SUGAR (Kriger [1976]) is intended as a high-level language for geographical analysis and mapping. It is a non-procedural language using "English-like" sentences as program input. The BNF syntax is defined using conventional simple English formats. Verbs, nouns, and prepositional phrases are handled. The particular set of words used is kept on file, so that particular dialects of the language can develop. While the system is useful for applications type work (i.e., for use by non-computer-trained scientists in this case), its non-procedural approach is a setback for general graphics programming.

Predecessors
H) SKETCHPAD

SKETCHPAD (Sutherland [1967]) is one of the original graphics interaction packages. It was developed by Sutherland at MIT. SKETCHPAD promised to "open up a new area of man-machine communication". It is an interactive non-programming graphics package in which structures are created during the interactive session. Sutherland's work initiated research on graphical databases.
2) OBJECTIVES

The primary objective in designing SLING is to create a structured procedural graphics language small enough to run on a 32K core minicomputer. More recent control constructs such as CASE, WHILE, FOR, WITH, and IF-THEN-ELSE are implemented. Every attempt is made to make the control structures conceptually as simple and straightforward as possible. This objective is based upon Hoare's [1973] assertion that any language must be simple to understand and use. The designer benefits because implementation is simpler, thus the language is more likely to work correctly. The user benefits because the language is easy to learn and program debugging is faster.

In keeping with the size considerations and reasonable run time speed operations on graphical data have been made as fast and core efficient as possible.

The operation of the superposition and deletion operators has been simplified conceptually from the operators as supported by LIG. The similarity between graphical data superposition and set union is exploited. Several new control structures based on Objectives
set operations are available to the programmer. These will be discussed in detail in later sections.

The language has a simple transput structure thus reducing the size of the support routines required leaving the user with more available core space. The user has the capability to define a library of functions and procedures. FORTRAN routines can be called from a program, thus a large number of support routines are available.

Disk facilities are provided to increase the available storage area. Disk access is programmer-controlled in this implementation. It is anticipated that subsequent implementations may support automatic disking, or a virtual memory system.

Graphical functions returning graphical values are an important feature. The graphical function can be manipulated as any other graphical variable. This means that the user can have a function called CIRCLE, which returns the graphical value of a circle, thus obviating built-in primitives of a complexity greater than BLANK or LINE. The function is useful as a core saving device. Only during the display of that function need the graphical variables used in that function use storage in the database. See the sample program in appendix G for an example of the use of graphical functions.

Objectives
3) FACILITIES OF THE UBC IMPLEMENTATION

A) GENERAL INTERACTIVE GRAPHIC FACILITIES

The language supports graphical functions, procedures, and variables. A graphical variable is a data structure that has associated with it a modifiable graphical value. The display value of a graphical variable resolves to a line or a series of lines, or a blank or a series of blanks. A graphical variable has associated with it the five attributes of xlocation, ylocation, xscale, yscale and angle of rotation. The values of the attributes are of type REAL. The screen representation is called the areal external representation as developed by Schrack [1976a]. Graphical objects are defined as appearing on a large but finite surface. By default a graphical variable appears centered on the screen, with default attributes as given:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>XLOCATION</td>
<td>0.5 UNITS</td>
</tr>
<tr>
<td>YLOCATION</td>
<td>0.5 UNITS</td>
</tr>
<tr>
<td>XSCALE</td>
<td>1.0 UNIT</td>
</tr>
<tr>
<td>YSCALE</td>
<td>1.0 UNIT</td>
</tr>
<tr>
<td>ANGLE</td>
<td>0.0 RADIANS</td>
</tr>
</tbody>
</table>

The default values are defined on a screen that is one square unit in size. The lower left hand corner has coordinates (0.0,0.0) and the upper right (1.0,1.0). All graphical
variables have an area associated with them called the identification area. The location of a variable is the center of that area which defaults to the unit screen as explained. The scale of a variable determines the extent of the area representing the area of the graphical variable. Scaling is used to change the screen size and the identification area of the graphical variable. It is used for the "ON" logical test in graphical interaction. A variable's attributes can be accessed by using the conventional notation for accessing a record structure. In this notation the graphical variable name appears followed by a dot "." followed by the name of the attribute.

EXAMPLE:
\[ X := \text{TRUCK}.\text{XSCALE}; \]

Note that the attributes location and scale represent two values of type REAL.

EXAMPLE:
\[ \text{TRUCK}.\text{LOCATION} := X,Y; \]

The only operators valid for use in graphical assignment statements are the superposition and deletion operators, "+" and "-" respectively. The semantics of these operators are quite different from that of the corresponding real and integer operators. The reason for the difference is the set-oriented implementation of the operations. The problems will be examined in detail in section 3-E.

---

1 Throughout the report syntactic keywords are underlined.
A new logical operator "IN" is available in SLING. This operator is a set testing operator returning a logical result. It essentially gives the programmer the capability to determine whether a graphical datum is an element of another graphical datum.

**EXAMPLE:**

```plaintext
A := LINE;
B := LINE;
B.ANGLE := 3.14159/2.0;
CROSS += A + B;
IF A IN CROSS THEN PUT CROSS;
```

In this case the value of the logical expression is TRUE. A has been assigned to CROSS and has not been deleted from CROSS before the test so CROSS is displayed on the screen.

The "ON" statement tests whether the cursor was within the identification area of the graphical variable when the user hit the keyboard interrupt.

**EXAMPLE:**

```plaintext
ON A THEN PUT 'YOU HIT A';
```

The cursor is a crosshair on the display screen which the user can manipulate. If the cursor is over a graphical object identification area when the user presses a key then the ON statement will be TRUE for that object. In this example if the cursor was over the identification area of A then "YOU HIT A" would be printed on the screen. The statement will also be TRUE for any other graphical object identification area under the facilities of the UBC Implementation.
cursor. A graphical object identification area is defined by the user through the use of the SCALE attributes. The default for the identification area is a square covering the whole screen. A scale factor of $(0.5,0.5)$ will cause the identification area to shrink to half its size along each axis. The identification area is centered on the graphical object which is in turn centered on the default x and y locations $(0.5,0.5)$. If the graphical datum is moved to $(0.4,0.1)$ then these coordinates become the center of the identification area.

The user may define character variables of varying lengths although a maximum must be specified. The character variable has the properties of a graphical variable including four of the five attributes of x and y, scale and location. Rotation is not supported as an attribute of a character variable as the existing hardware will not rotate characters. The logical ON operation can be used in the selection of menu items in conjunction with character variables. The operations of superposition, concatenation, substring and length of string are available in SLING as library functions. Section 3-E contains a discussion in greater depth.

B) GENERAL CONTROL STRUCTURES

Seven basic control structures are available:
In every construct the control statement is completed by a full statement. The \texttt{BEGIN ... END;} statement is a valid statement which brackets a series of statements. The full syntax of the control structures can be found in Appendix A; a discussion of the semantics of each, illustrated by examples, follows.

i) \texttt{WITH}

The \texttt{WITH} statement opens up the named graphical variable or the named attribute of a graphical variable for attribute assignment.

\textbf{EXAMPLE:}
\begin{verbatim}
WITH TRUCK DO LOCATION := 0.5,0.4;
\end{verbatim}

\textbf{EXAMPLE:}
\begin{verbatim}
WITH LOCATION DO
BEGIN
  TRUCK := 0.5,0.4;
  BUS := 0.3,0.1;
END;
\end{verbatim}

The \texttt{WITH} construct is useful in reducing the length of source code required to set the attributes of a variable, or to modify the common attributes of a number of graphical or character variables.

Facilities of the UBC Implementation
ii) FOR

The FOR construct allows the programmer to access the members of the set of a graphical or character variable. The FOR statement is iterative, in that the object statement is executed once for each member of the argument.

EXAMPLE:

```
A += B + C;
I := 0;
FOR A DO
  BEGIN
    I := I + 1;
    SCAL(I) := XSCALE;
  END;
```

In this example the vector SCAL is assigned the XSCALE values of each of the top-level members of A (i.e. B,C). The value of I is two on exit from the loop.

iii) AS

The AS statement is an iterative statement which executes the object statement a specified number of times.

EXAMPLE:

```
AS I := 15 TO 1 BY -3 DO
  K := K + I;
```

The object statement is not executed if the loop conditions do not hold at the entrance to the loop. The "BY" clause is optional, defaulting to the value "one" if not specified.
of the iteration variables (1, 15, and -3 in this example) may be full expressions that resolve to possibly negative integer values. The values of these expressions should not change within the object statement as undefined results will occur.

iv) WHILE

The WHILE statement executes the object statement while the argument of the statement is TRUE.

EXAMPLE:

\[
\text{WHILE } X \leq Y \text{ DO} \\
\text{BEGIN} \\
X := X + 1; \\
F(X) := X \times 2; \\
\text{END;}
\]

The test of the truth value occurs at the entrance to the WHILE statement. The object statement will not be executed if the condition is FALSE upon entry to the construct.

v) CASE

The case statement is a simple construct which executes the "nth" statement following the case statement within the bounds of that statement, where "n" is an expression that resolves to an integer value. The ELSE clause is executed if the argument is not inside the range of cases.
EXAMPLE:

```
CASE N DO
  BEGIN
    A := ...;
    B := ...;
    C := ...;
  BEGIN
    I := ...;
    J := ...;
  END;
END;
ELSE PUT 'ERROR';
```

There are four cases in this example. In cases one, two and three assignments are made to A, B, and C respectively. In case four both I and J are assigned values. The object statement of the ELSE clause is the default if "N" is not equal to any of one, two, three, or four. The ELSE clause is not optional.

vi) IF

The IF statement executes the object code conditionally, depending on the logical value of the argument.

EXAMPLE:

```
IF X = Y THEN GO TO LABEL1;
```

If the condition $X = Y$ is TRUE then control will transfer to the statement indicated by LABEL1, otherwise execution continues with the following statement. An ELSE clause can be used in conjunction with the IF statement. The object statement of the ELSE clause is executed if the argument of the IF
A Structured Language for INteractive Graphics

statement is FALSE.

EXAMPLE:

\[
\text{IF } X = Y \text{ THEN } I := 3 \\
\text{ELSE } I := 4;
\]

vii) ON

The ON statement previously discussed can also take an ELSE clause.

EXAMPLE:

\[
\text{ON TRUCK THEN } I := 3 \\
\text{ELSE } I := 4;
\]

Logical statements (ON, IF) are precluded from being objects of the THEN clause of a logical statement if an ELSE clause follows. This approach is taken to resolve the ambiguity associated with compound IF(ON) statements.
C) TRANSPUT STRUCTURES

The transput structures are straightforward. The programmer can PUT variables, constants, or expressions on the screen, with the same syntax regardless of type. Similarly the programmer can GET variables from the keyboard or cursor. The syntax of the GET statement is not as simple as that of PUT. See the Backus - Naur Form syntax description in Appendix B.

Files can be accessed to store and retrieve graphical and character variables. This capability permits the user to develop large databases by maintaining data on disk.

EXAMPLE:

```
GRAPHICAL VARIABLE TRUCK,BUS;
REAL VARIABLE X,Y;
GET TRUCK FROM FILE TRUCKFILE;
PUT TRUCK;
PUT 'TRUCK';
GET X,Y FROM CURSOR;
GET BUS FROM FILE BUSFILE;
BUS,LOCATION := X,Y;
PUT BUS;
PUT 'BUS';
```
D) THE GRAPHICAL FUNCTION

The graphical function has analogs in the more common data types. In FORTRAN for instance the integer function is a piece of code that returns an integer value. Similarly a graphical function is a piece of code which returns a graphical value. The graphical storage used by the function is used only during execution of the function. A graphical function can take arguments of any type. The functions are selectively loaded so that only the called functions reside in core. The graphical function has all the attributes of a graphical variable, namely x and y scale, and location attributes as well as a rotation angle and identification area. The function is a major contribution both to the minimal size of programs and to the power of the programming techniques available.

Graphical functions may be passed as parameters. This property is useful for handling external graphical functions. The graphical function returns a single graphical value. All graphical variables and external graphical functions defined inside the function during execution are undefined outside that function. Consequently complex graphical values and graphical
functions must be passed as parameters. For a contrasting implementation of graphical functions see Newman and Sproull [1973], and Newman [1971].

E) SEMANTICS OF OPERATORS AND ASSIGNMENTS

The superposition and deletion operators of SLING ("+" and "-" respectively) have somewhat different interpretations than similar operators used in integer and real arithmetic. Firstly, the operators are strictly dyadic in nature meaning that each operator takes two operands. Secondly, semantically "+" means set union and "-" means deletion from a set. The set implementation is core efficient, is consistent with the recursive nature of the graphical datum and is an elegant representation of graphical datum manipulations.

EXAMPLE:

```
A += LINE;
B += LINE;
B.ANGLE := 3.14159/2.0;
CROSS += A + B;
```

In this example A is by default a horizontal line, and B is a line whose angle is 90 degrees. CROSS is not a primitive of the language as are A and B, but is a set containing elements that are primitives. Every graphical variable must resolve to primitives at some level to be displayed. CROSS is linked to A and to B. When CROSS is displayed, both A and B are displayed.
If CROSS.LLOCATION were changed to (0.1,0.2) for instance, then the entire cross would be translated. But note that if the location of A were changed then the shape of CROSS would change. If A was moved up to coincide with the top of B then CROSS would display as a "T". Thus it is important that the user know that the semantics of superposition are not the semantics of the addition operation with which he is probably more familiar. If the user thinks of superposition as set union the language will be easier to learn.

EXAMPLE:
\[ C += C + A; \]

In this case A becomes a member of C and no elements are deleted from C. Note that in the case of set union where C is not united to itself, the current set of C is emptied before the new element is added to C.

Note that the user can form the structure:

\[
\begin{align*}
    &A += \text{LINE}; \\
    &B += \text{LINE}; \\
    &B.\text{ANGLE} := 3.141593/2.0; \\
    &A += B; \\
    &B += A; \\
    &\text{PUT A;} \\
\end{align*}
\]

In this case the recursive nature of the datum is exploited. The program will display A in infinite recursion.

The superposition operation is also defined for variables
of the type CHARACTER. Superposition is different than the operation of concatenation. Concatenation means that the two strings are appended to form one string.

EXAMPLE:

\[
S1 := 'III';
S2 := '000';
S3 := S1 || S2;
S4 += S1 + S2;
PUT S3;
PUT S4;
\]

In this example S3 will appear on the screen as "III000", while S4 will appear as "000", if all the variables retain their default attributes. Of course if the attributes of S1 or S2 are changed then S4 will change. For instance if S1.YLOCATION is modified to be lower than that of S2 then when S4 is displayed S1 will appear below S2 on the screen.

The assignment operator "\(:=\)" is used in an assignment statement to copy the value of a graphical variable to the assigned graphical variable. In contrast the set union operator "\(+=\)" simply enters the assigned variable entry point into the set of the assignee. The assignment operation is useful when several instances of a graphical variable are required.

EXAMPLE:

\[
\text{TRUCK.LOCATION := 0.5,0.5;}
\text{TRUCKA += TRUCK;}
\text{TRUCKB := TRUCK;}
\text{TRUCK.LOCATION := 0.2,0.5;}
\text{PUT TRUCKA,TRUCKB;}
\]

In this example TRUCKB is a copy of TRUCK while TRUCK is a
member of the set TRUCKA. When TRUCKA and TRUCKB are displayed
TRUCKA will appear centered on the location (0.2,0.5) and TRUCKB
will appear centered on (0.5,0.5).

The semantics of all other operators (i.e. *
*,/,+,-,**,MOD,&,||,\-,>,<,=) are similar to equivalent operators
in modern programming languages. See Appendix E for available
built-in functions.

P) TYPE COERCIONS

As the data types graphical and character are the only
structures available they cannot be coerced to any other type.
This fact imposes some restrictions on the use of character and
graphical variables. Graphical and character variables are the
only type of variable that can be put in a FILE using the file
statement. Due to implementation restrictions neither graphical
nor character variables may be used in the GET statement. Mixed
mode character and graphical variables may appear in a graphical
expression, as coercion between these types is defined though no
other type may be mixed with graphical data.

Character string constants, as distinct from character
variables may appear as the object of a PUT statement.
Character string constants are undeclared strings of arbitrary

Facilities of the UBC Implementation
length enclosed in single quotes.

Coercion takes place between integer and real variables. Real values are truncated if assigned to an integer variable.

The attributes of a graphical or character variable are of type REAL, and lie within the ranges:

```
XLOCATION, YLOCATION 0.0 - 1.0
XSCALE, YSCALE 0.0 - INFINITY
ROTATION ANGLE ±Pi
```

Note that if coercion of X or Y location values from real to integer takes place the value will always be either one or zero.

Logical variables are not coerced to any other type. A logical expression may contain real and integer values, and in the special case of the "IN" operator may contain graphical and character variables.

**EXAMPLE:**

```
LOGICAL VARIABLE L1, L2;
GRAPHICAL VARIABLE G1, G2;
INTEGER VARIABLE I1, I2;
REAL VARIABLE R1, R2;

IF G1 IN G2 THEN...
IF I1 >= I2 THEN...
IF L1 THEN...
IF L1 AND L2 THEN...
IF (G1 IN G2) OR (I1 = R2) THEN...
```
4) SHORTCOMINGS

The UBC implementation of SLING is not without its shortcomings. These are due for the most part to the approach taken in generating the language.

The XPL based Translator Writing System (TWS) of McKeeman, Horning and Wortman [1970] was used to translate SLING to FORTRAN. XPL runs on the IBM 370 but not on the minicomputer. This means that the user must translate a program on the IBM 370 and transfer the object to the minicomputer. The language is therefore not fully transportable. A better approach would have been that of creating a self-compiling language including as one of its capabilities the creation of graphical data handlers.

The FORTRAN object code produced is of course reasonably transportable. The only system dependent routines are the plotting and bit handling routines. The semantics and data structure handling routines can be written in SLING so that they are as transportable as any other part of the system. While FORTRAN is reasonably common and widely understood its control structures and data structures are weak. Thus the object code of a program is somewhat difficult to understand. The weak data
structuring in particular makes more powerful data structures which the user might want difficult to implement. SLING has no data structuring capabilities over and above the graphical data structure.

FORTRAN does not support recursive subroutine or function calls. The language would be more useful if the object language had these recursive capabilities.

The language is not extensible. The programmer must use the strict syntax of SLING as given by the BNF syntax. The programmer can redefine single keywords such as YLOCATION, but the literal capabilities of the language are weak. See Appendix E for the full list of keywords and symbols.
5) SUGGESTIONS FOR IMPROVEMENT

The shortcomings and restrictions of SLING have been documented throughout the thesis. Suggestions for improvement are gathered here in a concise form.

i) Change the object language from FORTRAN to a language that supports more advanced data structures, has efficient parameter passing, supports recursion, and has more modern control structures.

ii) Change the orientation of the language from an interactive graphics language to a more general small computer structured procedural language. Graphics capabilities can then be added to the data structure as a prelude and to the supporting subroutine system.

iii) Write a compiler for the language in the language so that the XPL system is no longer needed. This and the previous suggestion will improve the portability of the language.

iv) Work more on the disk facilities, a possibility being a virtual memory store with automatic disking of data.

v) Expand the transput facilities to handle interrupts and transput from a number of different devices.

Suggestions for Improvement
6) SUMMARY

A high-level graphics language has been implemented that permits the manipulation of variables of the data type GRAPHICAL. The system is compact and efficient in its use of resources. The similarity between it and a number of modern programming languages indicates that most programmers will have little trouble in learning the language. The language's weaknesses stem to a large extent from the target language chosen.

The language is not self-compiling, thus its portability is limited. The analyser of McKeeman, Horning, and Wortman is written in XPL, a language that does not presently run on minicomputers. A more profitable approach to the design of a graphical language is to write a language capable of handling data structures that support the transput facilities through utility routines.
7) Future Work

Implementation of a second version has begun. The language translates to "minicode", an assembler-like language used as a common target language for reasons of portability. The data structure as given in Hoare [1973b] is used. The data type GRAPHICAL can be defined as a prelude in a language supporting such structures. The homogeneous coordinate system used in the previous version will be dropped in favour of simple storage of properties, as the homogeneous coordinate system is prone to storage overflow problems. A BCPL-like approach to transput is being taken. The graphics output routines can be stored as functions written in the language and stored in a library file.
Bibliography and References

Bergman, S. and A. Kaufman
"BGRAF2: A REAL TIME GRAPHICS LANGUAGE WITH MODULAR OBJECTS AND IMPLICIT DYNAMICS"
Computer Graphics, (SIGGRAPH - ACM)
Vol. 10 No. 2
Summer 1976

Chan, B.B.
MATHEMATICAL ASPECTS OF A GRAPHICAL PROGRAMMING LANGUAGE AND ITS IMPLEMENTATION
Master's Thesis, Department Electrical Engineering
University of British Columbia
July 1976

Cleavland, J.C. and R.C. Uzgalis
GRAMMARS FOR PROGRAMMING LANGUAGES
WHAT EVERY PROGRAMMER SHOULD KNOW ABOUT GRAMMARS
University of California, Los Angeles
1974

Dijkstra, E.W.
A SHORT INTRODUCTION TO THE ART OF PROGRAMMING
monograph
August 1971

Dijkstra, E.W., O.J. Dahl AND C.A.R. Hoare
STRUCTURED PROGRAMMING
Academic Press, New York
1972

Giloi, W.K.
"ON HIGH-LEVEL PROGRAMMING SYSTEMS FOR STRUCTURED DISPLAY PROGRAMMING"
Computer Graphics, (SIGGRAPH - ACM)
Vol. 9 No. 1
Spring 1975

Harrison, M. C.
DATA STRUCTURES AND PROGRAMMING
Scott Foresman and Company, Illinois
1973
Bibliography and References
O'Brien, C.D. and H.G. Bown
"IMAGE: A LANGUAGE FOR THE INTERACTIVE MANIPULATION
OF A GRAPHICS ENVIRONMENT"
Computer Graphics, (SIGGRAPH - ACM)
Vol. 9 No. 1
Spring 1975

Pfister, G.F.
"A HIGH LEVEL LANGUAGE EXTENSION FOR CREATING
AND CONTROLLING DYNAMIC PICTURES"
Computer Graphics, (SIGGRAPH - SIGPLAN - ACM)
Vol. 11 No. 6
June 1976

Pieke, B.
DESIGN AND IMPLEMENTATION OF A HIGH-LEVEL LANGUAGE
FOR INTERACTIVE GRAPHICS
Master's Thesis, Department of Electrical Engineering
University of British Columbia
July 1973

Shapiro, L.G.
"ESP³: A HIGH LEVEL GRAPHICS LANGUAGE"
Computer Graphics, (SIGGRAPH - ACM)
Vol. 9 No. 1
Spring 1975

Schrack, G.F.
"ON THE SEMANTICS OF THE ASSIGNMENT STATEMENT
OF HIGH LEVEL GRAPHICAL LANGUAGES"
Computer Graphics, (SIGGRAPH - ACM)
Vol. 10 No. 2
Summer 1976

Schrack, G.F.
"DESIGN, IMPLEMENTATION AND EXPERIENCES WITH A
HIGHER-LEVEL GRAPHICS LANGUAGE FOR INTERACTIVE
COMPUTER-AIDED DESIGN PURPOSES"
SIGPLAN Notices, SIGGRAPH - SIGPLAN - ACM)
Vol. 11 No. 6
June 1976

Sutherland, I.E.
"SKETCHPAD: A MAN-MACHINE GRAPHICAL
COMMUNICATION SYSTEM"
ACM Professional Development Seminar
Readings for Computer Graphics
1967

Bibliography and References
Van Dam, A.
"DATA AND STORAGE STRUCTURES FOR INTERACTIVE GRAPHICS"
SIGPLAN Notices, (SIGGRAPH - ACM)
Vol. 6 No. 2
1971

Weinberg, G.M.
THE PSYCHOLOGY OF COMPUTER PROGRAMMING
Van Nostrand Reinhold, New York
1971

Wegner, P.
"DATA STRUCTURES IN PROGRAMMING LANGUAGES SEMINAR"
SIGPLAN Notices, (ACM)
Vol. 6 No. 2
February, 1971

Wirth, N.
SYSTEMATIC PROGRAMMING: AN INTRODUCTION
Prentice Hall, New Jersey
1973

Wirth, N.
ALGORITHMS + DATA STRUCTURES = PROGRAMS
Prentice Hall, New Jersey
1976
Appendix A

Syntax Charts

- Words in square cornered boxes are defined after their use in the charts.
- Words in round cornered boxes are syntactic keywords.
- Those word sequences ending in "identifier" are programmer declared names following PL/I identifier conventions.
PROGRAM

algorithm → eof

ALGORITHM

algorithm head → data declaration → body → end

ALGORITHM HEAD

real
integer
function
logical
graphical

procedure
identifier

(data declaration)

;
LOGICAL STATEMENT

if logical expression

then

statement

on variable


if logical expression

then

basic statement

on variable

else

statement
INPUT STATEMENT

get variable identifier

from cursor

OUTPUT STATEMENT

put expression

in filename identifier

ERASE STATEMENT

erase variable identifier
CASE STATEMENT

begin
statement

ASSIGNMENT STATEMENT

variable

expression

logical expression

EXPRESSION

term
LOGICAL EXPRESSION

LOGICAL TERM

LOGICAL FACTOR
COMPARATOR

VARIABLE (OR FUNCTION CALL)

identifier
(expression)


APPENDIX B

Backus - Naur Form Syntax

- Words and symbols in angle brackets are syntactic variables which are defined subsequently to use in the syntax.
- The operator "::=" means "is replaced by".
- The operator "|" means "or"
- Words and symbols not in angle brackets are syntactic keywords and symbols respectively.
<PROGRAM> ::= <ALGORITHM> EOF

<ALGORITHM> ::= <ALGORITHM FORMAT> <STATEMENT LIST> END
<ALGORITHM NAME>

<ALGORITHM FORMAT> ::= <ALGORITHM HEAD> ; <DATA DECLARATIONS>

<ALGORITHM HEAD> ::= <DATA TYPE> <ALGORITHM TYPE> <IDENTIFIER>
| <ALGORITHM HEAD> ( <DECLARATIONS> )

<ALGORITHM TYPE> ::= FUNCTION
| PROCEDURE

<DECLARATIONS> ::= <DATA DECLARATION>
| <ALGORITHM>
| <DECLARATIONS> ; <DATA DECLARATION>

(DATA DECLARATION) ::= <DATA HEAD> <EXPRESSION>
| <DATA DECLARATION> , <EXPRESSION>

<STATEMENT> ::= <LOGICAL STATEMENT>
| <CONTROL STATEMENT>
| <BASIC STATEMENT> ;
| <LABEL> : <STATEMENT>
| ;

<BASIC STATEMENT> ::= <INPUT STATEMENT>
| <OUTPUT STATEMENT>
| <ERASE STATEMENT>
| <ASSIGNMENT STATEMENT>
| <ALGORITHM CALL>
| <RETURN STATEMENT>
| <GO TO STATEMENT>
| <BEGIN STATEMENT>

<Data HEAD> ::= <DATA TYPE> <VARIABLE TYPE>
| LITERAL <IDENTIFIER> IS
| LABEL

<Data TYPE> ::= GRAPHICAL
| REAL
| INTEGER
| LOGICAL

<VARIABLE TYPE> ::= VARIABLE
| ARRAY
| EXTERNAL

<Input statement> ::= GET <EXPRESSION LIST>
| GET <EXPRESSION LIST> FROM <DEVICE>

APPENDICES
<OUTPUT STATEMENT> ::= PUT <EXPRESSION LIST>
   | PUT <EXPRESSION LIST> IN <FILE NAME>

<ERASE STATEMENT> ::= ERASE
   | ERASE <VARIABLE>

<LOGICAL CLAUSE> ::= IF <LOGICAL EXPRESSION> THEN
   | ON <VARIABLE> THEN

<LOGICAL STATEMENT> ::= <LOGICAL CLAUSE> <STATEMENT>
   | <LOGICAL CLAUSE> <BASIC STATEMENT>
   | ELSE <STATEMENT>

<CONTROL CLAUSE> ::= FOR <VARIABLE>
   | WITH <VARIABLE>
   | AS <ITERATION>
   | CASE <EXPRESSION>
   | WHILE <LOGICAL EXPRESSION>

<CONTROL STATEMENT> ::= <CONTROL CLAUSE> DO <STATEMENT>

<ITERATION> ::= <ASSIGNMENT> TO <ARGUMENT>
   | <ASSIGNMENT> TO <ARGUMENT> BY <ARGUMENT>

<BEGIN STATEMENT> ::= BEGIN <STATEMENT LIST> END

<ALGORITHM CALL> ::= CALL <VARIABLE>

<ASSIGNMENT STATEMENT> ::= <VARIABLE> := <EXPRESSION>
   | <VARIABLE> := <LOGICAL EXPRESSION>
   | <ASSIGNMENT> , <EXPRESSION>

<RETURN STATEMENT> ::= RETURN

<GO TO STATEMENT> ::= GO TO <LABEL>

<EXPRESSION> ::= <EXPRESSION> + <TERM>
   | <EXPRESSION> - <TERM>
   | <TERM>
   | <TERM> * <FACTOR>
   | <TERM> / <FACTOR>
   | <TERM> ** <FACTOR>
   | <TERM> MOD <FACTOR>
   | <TERM> | <FACTOR>
   | <FACTOR>

<FACTOR> ::= ( <EXPRESSION> )
   | <ALGORITHM TAIL>

APPENDICES
A Structured Language for INteractive Graphics

| <ARGUMENT> |
| <STRING> |

<LOGICAL EXPRESSION> ::= <LOGICAL EXPRESSION> | <LOGICAL TERM> |
|<LOGICAL TERM> |

<LOGICAL TERM> ::= <LOGICAL TERM> & <LOGICAL FACTOR> |
|<LOGICAL FACTOR> |

<LOGICAL FACTOR> ::= ( <LOGICAL EXPRESSION> ) |
| <EXPRESSION> <COMPARATOR> <EXPRESSION> |
| TRUE |
| FALSE |

<COMPARATOR> ::= < |
| > |
| < |
| = |
| > = |
| = |
| ¬ = |
| IN |

<VARIABLE> ::= <IDENTIFIER> |
| <VARIABLE> ( <EXPRESSION LIST> ) |
| <VARIABLE> . <IDENTIFIER> |

<DEVICE> ::= CURSOR |
| KEY |
| <FILE NAME> |

<FILE NAME> ::= <IDENTIFIER> |

<IDENTIFIER> ::= <ALPHABETIC CHARACTER> |
| <IDENTIFIER> <ALPHANUMERIC CHARACTER> |

<number> ::= <DIGIT SEQUENCE> |
| <DIGIT SEQUENCE> . <DIGIT SEQUENCE> |

<ALPHABETIC CHARACTER> ::= A|B|C|D|E|F|G|H|I|J|K|L|M |
|N|O|P|Q|R|S|T|U|V|W|X|Y|Z |

<ALPHANUMERIC CHARACTER> ::= <ALPHABETIC CHARACTER> |
| <DIGIT> |

<DIGIT SEQUENCE> ::= <DIGIT> |
| <DIGIT SEQUENCE> <DIGIT> |

<DIGIT> ::= 1|2|3|4|5|6|7|8|9|0

APPENDICES
Appendix C

Graphical Data Structure

The graphical variable attributes, display value, and set union information are stored in the "Graphical Datum Table". The size of the table is calculated by the translator as (number of graphical variables and functions + size of subscripted variables) * the core factor. The core factor increases the table size by a default value of 120%. The homogeneous transformation matrices are stored in six locations of the variable's entry in the Graphical Datum Table. A "control" word is stored for each graphical variable. Bit 3 indicates whether a graphical variable is currently being displayed ("1" = yes). Bits zero through two indicate the display value of the variable ("0" = blank, "1" = line, "2" = string). A minimum of two words are used to store set union information for each graphical variable. Each bit corresponds to a position in the Graphical Datum Table. If the bit is set then the corresponding graphical variable is an element of that graphical variable set. A minimum of two words is used because graphical variables of the type character use the words as pointers into the string vector,
rather than as set union indicators. The number of words in the set union indicators is dependent on the number of Graphical Variable Table entries. There is one bit allocated for each entry. Thus the two words allocated initially will be enough for up to thirty two graphical variables. The translator allocates all needed storage and manages the variable list and entry points into the graphical variable table. Allocation in the table is made on a simple sequential basis from the user data declaration.

The total storage area for a non-character graphical variable is (4+2+1+2)=9 words. Note that if more than thirty two graphical variables are used then more storage is needed. A count of 33 to 48 graphical variables requires 10 words per variable.

The length of the string storage vector is calculated from the string declaration in the source program. One word is allocated for every two characters. The set information words of the entry of a character variable point to the beginning and the end of the associated string. Length, substring and concatenation operations modify these pointers. No garbage collection scheme exists as yet for the string storage area.

APPENDICES
### Figure II

Graphical Datum Table

"A" = A Matrix  
"B" = B Matrix  
"C" = Control word  
"SET" = set union indicator words
The homogeneous transformation matrices are those used in LIG. Use of the matrices is advantageous in that calculation a series of transformations is quite simple. However overflow of matrix entries occurs if the scale factor is greater than three.
A Structured Language for INteractive Graphics

$$A = \begin{bmatrix} XSCALE \cdot \cos \phi & -YSCALE \cdot \sin \phi \\ XSCALE \cdot \sin \phi & YSCALE \cdot \cos \phi \end{bmatrix}$$

$$B = \begin{bmatrix} XLOC + \frac{(YSCALE \cdot \sin \phi - XSCALE \cdot \cos \phi)}{2} \\ YLOC - \frac{(XSCALE \cdot \sin \phi + YSCALE \cdot \cos \phi)}{2} \end{bmatrix}$$

Figure III

Homogeneous Coordinate Matrices

XLOC = the horizontal axis x location
YLOC = the vertical axis y location
XSCALE = the horizontal axis scale factor
YSCALE = the vertical axis scale factor
$\phi$ = the rotation angle in Radians
Appendix D

The Translator

The main data structure in the translator is a binary tree used to access the names and attributes of graphical and character variables. As each graphical or character declaration is read, the name is put into the tree in alphabetical order. The level of nesting at which the variable was declared is appended to the name, the variable type (character or graphical, variable, function or parameter) is entered and the execution symbol table entry is recorded. Execution table entries (the graphical variable table of Appendix C, Figure II) are allocated sequentially as the variables are read. Note that real, integer and logical variables are not entered into the table. Expressions using such variables are not processed by the translator other than by conversion to FORTRAN formatting. Predefined functions and keywords are entered into the tree at initialization time. The processing of graphical statements
creates a series of calls to FORTRAN run-time routines in the supporting subroutine system. Access to the graphical variable table is handled by the translator. The entry point into the table is passed to the run-time routines to effect access to the stored attributes of that variable. Graphical parameters can be passed to graphical functions and procedures. Graphical parameter passing is done by reference. A variable is put in the formal parameter list which is replaced by the actual parameter entry point at time of call. External functions access the table via an offset passed in the system common area. The table size is determined by counting the number of declared variables, arrays and functions. This number is then increased by the core factor (default 120%). The difference between the amount actually needed and that allocated can be used by external functions. If overflow occurs the user must recompile his program with a larger core factor.
Appendix E

RESERVED WORDS AND SYMBOLS

i) Those modifiable by the LITERAL statement:

XLOCATION
YLOCATION
LOCATION
XSCALE
YSCALE
SCALE
ANGLE

ii) Syntactical keywords:

BEGIN END
FUNCTION PROCEDURE
GRAPHICAL REAL INTEGER LOGICAL
VARIABLE ARRAY
LITERAL IS
LABEL
GET FROM KEY CURSOR
PUT IN FILE
ERASE
IF ON THEN ELSE
FOR AS TO BY CASE WHILE WITH DO
CALL RETURN
GO TO
TRUE FALSE
::

iii) Operators:

:= assignment
+= set union
+ addition, superposition
- subtraction, deletion
* multiplication
/ division
** power of
MOD modulus
|| string concatenation
& logical AND
| logical OR
¬ logical NOT
() brackets

iv) Comparators:
< less than
> greater than
<= less than or equal to
>= greater than or equal to
equal to
~= not equal to
IN element of (graphical only)

vi) Graphical Primitives:
LINE
BLANK

vii) System Routines:
GATR CAT
COPY DISPLAY
ERROR HITH
LENGTH PATR
PLOP PUSH
POP RESTORE
STORE STRING
SUBSTR STKPLT*
VERASE

viii) Built-in Functions:
ABS FLOAT
ALOG IABS
ARCOS IFIX
ARSIN INT
ATAN SIGN
ATAN2 SIN

APPENDICES
COS  SINH
COSH  SQRT
EXP  TAN

* machine dependent routine
Appendix F

RUNNING INSTRUCTIONS

To translate a SLING program:

$RUN ELEC:SLING [SCARDS=...] [SPUNCH=...] [SPRINT=...]
[PAR=CORE=<CORE FACTOR NUMBER>]

Defaults:

SCARDS *SOURCE*
SPUNCH -LOAD
SPRINT *SINK*
CORE 120

To run a SLING program on the NOVA:

1) copy the files to the NOVA disk
2) compile the file contents:
   FORTRAN <MAIN PROCEDURE NAME>
   {FORTRAN <SUB PROCEDURE NAME>}
3) load the procedures and libraries:
   RLDR <MAIN PROCEDURE NAME> [<SUB PROCEDURE NAME>]
   [<USER.LB>] SLING.LB IO.LB MATH.LB TSK.LB
4) run the program:

<MAIN PROCEDURE NAME>

Legend:

<> replace with actual name

[ ] optional

{} repeat 0 or more times
Appendix G

Sample Program Listing
THE STRUCTURED LANGUAGE FOR INTERACTIVE GRAPHICS
VERSION OF DECEMBER 1976
01-24-77 19:32:30

GRAPHICAL PROCEDURE INKING;
/* this procedure calls INK and displays it
at 8 different locations chosen by the user */
INTEGER VARIABLE I, NUMBER, ROTAT;
REAL VARIABLE X, Y, SCALX, SCALY;
GRAPHICAL ARRAY LINES(30);
GRAPHICAL VARIABLE FIGURE;
GRAPHICAL EXTERNAL INK;
PUT 'PLEASE ENTER THE NUMBER OF LINES IN THE FIGURE';
GET NUMBER;
PUT 'PLEASE ENTER THE ANGLE OF ROTATION IN DEGREES';
GET ROTAT;
PUT 'PLEASE ENTER THE XY SCALES (0 - 1)';
GET SCALX; GET SCALY;
ERASE;
FIGURE += INK(LINES, NUMBER);
FIGURE.SCALE := SCALX, SCALY;
AS I := 1 TO 8 DO
BEGIN
GET X, Y FROM CURSOR;
FIGURE.LOCATION := X, Y;
PUT FIGURE;
FIGURE.ANGLE := FIGURE.ANGLE + FLOAT(ROTAT)/180.*3.14159;
END;
END INKING;

OBJECT FILE WRITTEN
CORE REQUIREMENTS:
GRAPHICAL DATA STRUCTURE: 39 * 10 WORDS
STRING STORAGE: 0 WORDS
CORE FACTOR: 120%

COMPILATION TIME: 0.18 SECONDS
END OF COMPILATION
NO ERRORS WERE DETECTED.
THE STRUCTURED LANGUAGE FOR INTERACTIVE GRAPHICS
VERSION OF DECEMBER 1976
01-25-77 19:53:57

GRAPHICAL FUNCTION INK(GRAPHICAL ARRAY LINES;
INTEGER VARIABLE NUMBER);
GRAPHICAL EXTERNAL LINEAT;
/* this procedure allows the user to define
a graphical object of his own choosing */
REAL VARIABLE A,B,C,D;
INTEGER VARIABLE I;
GET A,B FROM CURSOR;
AS I := 1 TO NUMBER DO
BEGIN
GET C,D FROM CURSOR;
LINES(I) := LINEAT(A,B,C,D);
PUT LINES(I);
INK += INK + LINES(I);
A := C;
B := D;
END;
END INK;

OBJECT FILE WRITTEN
CORE REQUIREMENTS:
GRAPHICAL DATA STRUCTURE: 2 * 9 WORDS
STRING STORAGE: 0 WORDS
CORE FACTOR: 120%

COMPILATION TIME: 0.25 SECONDS

END OF COMPILATION
NO ERRORS WERE DETECTED.
Appendix H

Sample FORTRAN Object Code

- This FORTRAN code is a sample of the code produced by the translator for the inking program of Appendix G.
This routine is the FORTRAN target code generated by the SLING routine INKING in the previous Appendix. It drives the INK graphical function, supplying the size (number of lines) of the graphical object to be drawn. The routine also inputs parameters used to manipulate the graphical object.

```
REAL SCALY
REAL SCALX
REAL Y
REAL X
INTEGER ROTAT
INTEGER NUMBER
INTEGER I
INTEGER AMATRIX(2,2,39), BMATRIX(2,39)
INTEGER CONTROL(39), SET(39,3)
INTEGER STRING(1)
REAL GATR
INTEGER TABLETOP, SHEAD, STACKP, ERRCO
LOGICAL HIT, ITEST
EXTERNAL HIT, GATR, ITEST
DATA MAXSIZE/39/, TABLETOP/33/, ISLEN/1/, ISETW/3/
DATA (((AMATRIX(I,J,K), K=1,39), J=1,2), I=1,2) /39*10000, 39*0, 39*0, 39*10000/,
DATA BMATRIX/78*0/
DATA COMMON MAXSIZE, TABLETOP, ISETW, ISLEN, SHEAD, STACKP, ERRCO
WRITE FREE(10) 'ENTER THE NUMBER OF LINES IN THE FIGURE'
READ FREE(11) NUMBER
WRITE FREE(10) 'ENTER THE ANGLE OF ROTATION IN DEGREES'
READ FREE(11) ROTAT
WRITE FREE(10) 'ENTER THE XY SCALES (0 - 1)'
READ FREE(11) SCALX
READ FREE(11)SCALY
CALL ERASE
SET(32,1)=0
CALL ISET SET(32, (33-1)/16+1), MOD(33-1,16)
CALL INK(1, NUMBER, 33, AMATRIX, BMATRIX, CONTROL, SET, STRING)
CALL PATR(32, 'XS', SCALX, AMATRIX, BMATRIX)
CALL PATR(32, 'YS', SCALY, AMATRIX, BMATRIX)
I=1
10001 IF((1.LT.8) AND. (I.LT.1.OR.I.GT.8)) GO TO 10000
IF((1.GT.8) AND. (I.GT.1.OR.I.LT.8)) GO TO 10000
CALL CURSOR (MDUM, X, Y)
CALL PATR(32, 'XL', X, AMATRIX, BMATRIX)
CALL PATR(32, 'YL', Y, AMATRIX, BMATRIX)
CALL DISPLAY(32, AMATRIX, BMATRIX, CONTROL, SET, STRING)
CALL PATR(32, 'AN', GATR(32, 'AN', AMATRIX, BMATRIX)
```

APPENDICES
*FLOAT(ROTAT)/180.*3.14159,A MATRIX,B MATRIX)
 I=I+(1)
 GO TO 10001
10000 CONTINUE
 STOP
 END
This routine appears in the user function library. It is used to return a graphical value representing a series of lines input by the user.

```
SUBROUTINE INK(IN1,NUMBER,IN0,AMATRIX,BMATRIX,
*CONTROL,SET,STRING)
INTEGER I
REAL D
REAL C
REAL B
REAL A
INTEGER NUMBER
INTEGER AMATRIX(2,2,MAXSIZE), BMATRIX(2,MAXSIZE)
INTEGER CONTROL(MAXSIZE), SET(MAXSIZE, ISETB)
INTEGER STRING(ISLEN)
REAL GATR
INTEGER TABLETOP,SHEAD,STACKP,ECO
LOGICAL HIT,ITEST
EXTERNAL HIT,GATR,ITEST
COMMON MAXSIZE,TABLETOP,ISETH,ISLEN,SHEAD,STACKP,SSIZE
CALL CURSOR(MDUM,A,B)
I=1
10001 IF((1.LT.NUMBER).AND.(I.LT.1.OR.I.GT.NUMBER)) GO TO 10000
IF((1.GT.NUMBER).AND.(I.GT.1.OR.I.LT.NUMBER)) GO TO 10000
CALL CURSOR(MDUM,C,D)
SET(IN1+I,1)=0
CALL ISET(SET(IN1+I,(TABLETOP+2)/16+1),
*MOD(TABLETOP+2-1,15))
CALL LINEAT(A,B,C,D,TABLETOP+2,AMATRIX,
*BMATRIX,CONTROL,SET,STRING)
CALL COPY(IN1+I,TABLETOP+2,AMATRIX,BMATRIX,CONTROL,SET)
CALL DISPLAY(IN1+I,AMATRIX,BMATRIX,CONTROL,SET,STRING)
CALL ISET(SET(IN0,(IN1+I)/16+1),MOD(IN1+I-1,15))
A=C
B=D
I=I+(1)
GO TO 10001
10000 CONTINUE
RETURN
END
```
Appendix I

A Review of Language Statements

i) Transput

**PUT A,B;**
- The PUT statement will put a variable or constant of any type on the screen.
- If a blank is put on the screen then the next non-graphical or non-character variable will be put over that blank.

**PUT TRUCK IN FILE TRUCKFILE;**
- The graphical variable TRUCK is written into a file called "TRUCKFILE". The file is automatically created.

**GET X,Y;**
- The GET statement gets values from the keyboard. In this case the values must be REAL or INTEGER.

**GET TRUCK FROM FILE TRUCKFILE;**
- In this variant a graphical variable is returned from a file.

**GET X,Y FROM CURSOR;**
- The x,y cursor coordinates are returned when the user hits any key on the keyboard. The values are from (0.0 - 1.0).

**GET CHAR FROM KEY;**
- The key that the user hit is returned after the user keyboard interrupt. Both this and the above variant stop execution of the program and turn on the cursor until the user responds.
ii) Begin Statement

BEGIN:
  A := 3;
  B := 4;
END;

- The BEGIN statement is used to bracket a series of statements. It is a valid object statement of a control construct.

iii) Control

WITH LOCATION DO BUS := 0.5,0.5;

- The attribute LOCATION of the graphical or character variable BUS is set to the given values.

WITH BUS DO LOCATION := 0.5,0.5;

- Again the attributes are set to the given values. There is no difference in the given examples, they show that either a variable or an attribute may be modified using the WITH statement.

FOR TRUCK DO I := I + 1;

- This statement iterates through the members of the set of TRUCK (i.e., its components). In this case "I" counts the number of elements of the TRUCK. The named variable (TRUCK) must be graphical or character.

AS I := J TO K BY -1 DO R := R + 1.0;

- This statement is similar to the DO loop of PL/I. The object statement is not executed if the loop conditions do not hold on entrance to the loop. Testing is done at the top of the loop.

WHILE I <= 10 DO R := R + 1.0;

This statement is similar to the WHILE statement of PL/I. The test of the logical value is made at the entrance to the loop.

CASE I DO

BEGIN
  J := J + 1;
  K := K + 1;
END;

APPENDICES
ELSE \( L := L + 1; \)

- "I" must resolve to an integer value. The "I"th statement in the \( \text{BEGIN} \) clause is executed. If "I" does not resolve to a number in the range of statements in the \( \text{BEGIN} \) clause then the ELSE clause is executed.

\[
\begin{align*}
\text{IF } A &< B \text{ THEN GO TO ALPHA;} \\
\text{ELSE GO TO BETA;} \\
\end{align*}
\]

- If the logical value of the argument is true then the THEN clause is executed, otherwise the ELSE clause is executed.

\[
\begin{align*}
\text{ON TRUCK THEN } I &:= 3; \\
\text{ELSE } I &:= 4; \\
\end{align*}
\]

- The cursor is turned on and execution halts until the user presses a key. If the cursor was over the identification area of the argument (TRUCK) the THEN clause is executed, otherwise the ELSE clause is executed. The argument must be a graphical or a character variable.

iv) Assignment

\[
A := A + 1;
\]

The assignment statement places the value of the expression on the right hand side of the assignment operator into the location addressed by the name of the variable on the left hand side. The graphical set union operator "+=" discussed in 3-E has a different meaning than the assignment operator:

\[
\text{TRUCK } += \text{ TRUCKB;}
\]

- The set union statement has the effect of putting TRUCKB into the set of TRUCKA where assignment moves the 9 data words associated with a graphical variable. The assignment statement should be used only if it is necessary that the set of attributes be stored temporarily.

v) Declarations

\[
\begin{align*}
\text{REAL VARIABLE } &X, Y; \\
\text{INTEGER VARIABLE } &I1, K2; \\
\text{LOGICAL ARRAY } &A(10), B(20); \\
\text{GRAPHICAL VARIABLE } &\text{TRUCK, BUS;}
\end{align*}
\]
GRAPHICAL ARRAY MEMBER(10);
CHARACTER(10) CHAR(10);

- Real, integer and logical declarations are self explanatory
- TRUCK and BUS are declared as graphical variables.
MEMBER is an array of 11 elements, each of which is a graphical datum.
- Graphical array bounds are indexed from zero, no lower bound specification is possible. Multidimensional arrays of graphical and character variables are not permitted.
- The character array CHAR has 11 elements each of which contain a maximum of ten characters.

LITERAL XL IS XLOCATION;

- The literal statement allows the programmer to shorten certain keywords. The list of keywords that can be redefined in the literal statement is given in Appendix E.

GRAPHICAL EXTERNAL CIRCLE, SQUARE;
INTEGER EXTERNAL NTHROOT;

- All external functions not mentioned as being built-in in Appendix E must be declared. The declaration must specify the TYPE of the function.

LABEL L1, L2;

- All labels must be declared. If a label is not declared then it will be assumed by context, and the translator will give a warning.

vi) Algorithm Head

GRAPHICAL PROCEDURE ALPHA;
INTEGER FUNCTION BETA(K);
GRAPHICAL FUNCTION TRUCK;

- Procedures and functions may be REAL, INTEGER, LOGICAL or GRAPHICAL. A function returns a value of the associated type. The outer procedure must be graphical if nested procedures are graphical. Real, integer, and logical procedures are more efficient than graphical procedures, however these procedures may contain no graphical or character variables.

vii) Return Statement

APPENDICES
RETURN;

- The return statement returns execution to the calling program. It must appear once in a function, but can appear any number of times.

viii) Go to Statement

GO TO ALPHA;

- ALPHA must be a declared label attached to some statement in the procedure body.

ix) Erase Statement

ERASE CAR;
ERASE;

- The erase statement can erase a single variable, as shown in the first example or it can be used to erase the whole screen as in the second example.

x) Procedure Call

CALL S1(A,B,C)

- Starts execution of the named procedure at the point of call. On return from the procedure execution begins again at the statement after the call. Parameters are passed by the normal FORTRAN calling conventions except graphicals which are passed by reference only.
Index

angle ........................................ 16, 65
array ........................................ 4, 80
as .............................................. 20, 21, 78
attribute ..................................... 16, 17, 19, 20, 31, 58
begin statement ............................ 20, 78
bibliography ................................ 37
bnf syntax .................................... 55
built-in function ............................ 66
case .......................................... 14, 20, 22, 79
character string constant .................. 30
character variable .......................... 5, 19, 28, 30, 80
coercion ..................................... 30
comparator ................................... 66
compile ....................................... 68
concatenation ................................ 19, 29, 59
core structure ................................ 32, 78
core factor ................................... 58, 68
core usage .................................... 5, 9, 15, 26, 27, 58
cursor ........................................ 6, 18, 25, 77
data declaration ............................ 4
data structure ................................ 10, 16, 32, 58
database ....................................... 9, 25
declaration .................................... 79
default ......................................... 16, 19, 27, 68
deletion ....................................... 14, 17, 19, 27
disk ........................................... 14, 15, 25, 34
display ........................................ 6
extensible ..................................... 4, 33
external ....................................... 4, 80
file ............................................ 15, 25, 30, 77
for ............................................. 14, 20, 21, 78
garbage collection ........................... 59
get ............................................. 25, 30, 77
graphical data structure ................... 58
graphical datum table ........................ 60
graphical function ........................... 5, 15, 16, 26
graphical procedure ........................... 11, 16
graphical variable ............................ 16, 30, 58
homogeneous coordinate matrices .......... 58, 61
host language ................................ 8
human engineering ........................... 10
identification area .......................... 18, 19
if .............................................. 14, 20, 23, 79
interrupt ..................................... 10, 18, 34
iterative ...................................... 21
keyword ........................................ 65
label ........................................... 80
<table>
<thead>
<tr>
<th>Term</th>
<th>Page Numbers</th>
</tr>
</thead>
<tbody>
<tr>
<td>length</td>
<td>19,59</td>
</tr>
<tr>
<td>library</td>
<td>5,15,19,26,68</td>
</tr>
<tr>
<td>literal</td>
<td>33,65,80</td>
</tr>
<tr>
<td>location</td>
<td>65</td>
</tr>
<tr>
<td>logical expression</td>
<td>18,22,23,31</td>
</tr>
<tr>
<td>logical operator</td>
<td>17</td>
</tr>
<tr>
<td>minicomputer</td>
<td>6,32</td>
</tr>
<tr>
<td>nesting</td>
<td>4</td>
</tr>
<tr>
<td>objectives</td>
<td>14</td>
</tr>
<tr>
<td>on</td>
<td>18,20,24,79</td>
</tr>
<tr>
<td>operator</td>
<td>27,65</td>
</tr>
<tr>
<td>overlay</td>
<td>9</td>
</tr>
<tr>
<td>portable</td>
<td>34</td>
</tr>
<tr>
<td>primitive</td>
<td>9,15,66</td>
</tr>
<tr>
<td>procedural language</td>
<td>14</td>
</tr>
<tr>
<td>program</td>
<td>4</td>
</tr>
<tr>
<td>put</td>
<td>25,30,77</td>
</tr>
<tr>
<td>recursive</td>
<td>11</td>
</tr>
<tr>
<td>references</td>
<td>37</td>
</tr>
<tr>
<td>reserved symbol</td>
<td>65</td>
</tr>
<tr>
<td>reserved word</td>
<td>65</td>
</tr>
<tr>
<td>review</td>
<td>77</td>
</tr>
<tr>
<td>run</td>
<td>68</td>
</tr>
<tr>
<td>scale</td>
<td>65</td>
</tr>
<tr>
<td>scards</td>
<td>68</td>
</tr>
<tr>
<td>scope</td>
<td>4</td>
</tr>
<tr>
<td>screen</td>
<td>16</td>
</tr>
<tr>
<td>self-compiling</td>
<td>32,35</td>
</tr>
<tr>
<td>set</td>
<td>15,17,27,58</td>
</tr>
<tr>
<td>shortcomings</td>
<td>32</td>
</tr>
<tr>
<td>sprint</td>
<td>32</td>
</tr>
<tr>
<td>spunch</td>
<td>68</td>
</tr>
<tr>
<td>statement review</td>
<td>77</td>
</tr>
<tr>
<td>string vector</td>
<td>59</td>
</tr>
<tr>
<td>structured language</td>
<td>9,10,14</td>
</tr>
<tr>
<td>substring</td>
<td>19,59</td>
</tr>
<tr>
<td>superposition</td>
<td>14,17,19,27</td>
</tr>
<tr>
<td>symbol table</td>
<td>5,58</td>
</tr>
<tr>
<td>syntax</td>
<td>33,41,55</td>
</tr>
<tr>
<td>syntax charts</td>
<td>41</td>
</tr>
<tr>
<td>system routine</td>
<td>66</td>
</tr>
<tr>
<td>translator</td>
<td>63,64</td>
</tr>
<tr>
<td>translator writing system</td>
<td>32,35</td>
</tr>
<tr>
<td>transportable</td>
<td>32</td>
</tr>
<tr>
<td>transput</td>
<td>5,6,15,25,34,77</td>
</tr>
<tr>
<td>type</td>
<td>25,30</td>
</tr>
<tr>
<td>use</td>
<td>68</td>
</tr>
<tr>
<td>while</td>
<td>14,20,22,78</td>
</tr>
<tr>
<td>with</td>
<td>14,20,78</td>
</tr>
<tr>
<td>xlocation</td>
<td>16,65</td>
</tr>
<tr>
<td>xscale</td>
<td>16,65</td>
</tr>
<tr>
<td>ylocation</td>
<td>16,65</td>
</tr>
</tbody>
</table>
Index

<table>
<thead>
<tr>
<th>Name</th>
<th>Pages</th>
</tr>
</thead>
<tbody>
<tr>
<td>yscale</td>
<td>16, 65</td>
</tr>
<tr>
<td>BCPL</td>
<td>36</td>
</tr>
<tr>
<td>BGRAF2</td>
<td>11</td>
</tr>
<tr>
<td>DALI</td>
<td>12</td>
</tr>
<tr>
<td>ESP</td>
<td>11</td>
</tr>
<tr>
<td>FORTRAN</td>
<td>4, 6, 8, 15, 73</td>
</tr>
<tr>
<td>GRIP</td>
<td>10</td>
</tr>
<tr>
<td>IBM 370</td>
<td>6, 32</td>
</tr>
<tr>
<td>IMAGE</td>
<td>10</td>
</tr>
<tr>
<td>LIG</td>
<td>6, 8</td>
</tr>
<tr>
<td>MTS</td>
<td>6</td>
</tr>
<tr>
<td>NOVA</td>
<td>6</td>
</tr>
<tr>
<td>SKETCHPAD</td>
<td>13</td>
</tr>
<tr>
<td>SUGAR</td>
<td>12</td>
</tr>
<tr>
<td>XPL</td>
<td>6, 35</td>
</tr>
</tbody>
</table>