LINUS: A STRUCTURED LANGUAGE FOR INSTRUCTIONAL USE

John D. Woolley and Leland R. Miller
Department of Computer Science
Bowling Green State University

1. INTRODUCTION

One of the crucial decisions in organizing a first course in computer science is the choice of a programming language. Although there is considerable variance of opinion as to what the ideal language should be, two main approaches can be delineated. The first approach stresses the necessity of learning the dominant scientific language, which in the Americas amounts to a vote for Fortran (2). The practicality of this choice is as indisputable as the awkwardness of the syntax of that language. The alternative view stresses the importance of the program structure in developing a sound sense of "algorithmic thinking". Proponents of this view would suggest Algol W (4) or perhaps Pascal (5). We contend that both approaches have important advantages. This paper explores an approach which attempts to maximize the benefits of both.

For pragmatic reasons, the student of a first course should obtain a knowledge of the fundamentals of Fortran. A number of structural problems (especially with ANS Fortran) present obstacles to the student in learning algorithmic thinking. These include inadequate control structures, lack of data type character, no free format input/output, and use of default data typing and conversions to trap the unwary. By regarding ANS Fortran as a machine, to which programs in a higher level language are translated, the advantages of learning Fortran are still realized. Initially, the student concentrates on learning a language which is free of the above restrictions. Later, in the same way an assembler programmer acquires knowledge of his machine, the student acquires knowledge of Fortran.

The solution we have adopted is to implement a language called Linus (Language for instructional use). This language is preprocessed to ANS Fortran, but has more the appearance of PL/I or Algol 68 (3), facilitating learning corresponding features of those languages. The majority of the language has been implemented and is presently undergoing testing.

The remainder of the paper is divided into the following sections:

Section 2--a description of the language
Section 3--sample programs
Section 4--options and debugging aids
Section 5--the impact of the language on the curriculum
Section 6--future plans

2. LANGUAGE DESCRIPTION

Linus is a structured, goto-less language in the sense of (6). Those who prefer to program in the goto style may do so by means of a FORTRAN block (see Section 2.3).

The language is free format and statements may be combined to form blocks. A block corresponds to an Algol 68 serial clause (3). Each block has its own opening (such as FOR) and its own closing (END_FOR). This aids the student in correctly structuring programs, as the sample program segment below illustrates.

```
PROGRAM  ¢SKELETON MAIN PROGRAM¢
     DECLARE
          ¢DECLARE VARIABLES¢
     END_DECLARE;
     FOR I := E DOWN_BY 2 TO 0 LOOP
          IF I*3 > J/2 THEN OUTPUT := I;
                       ELSE OUTPUT := J;
          END_IF;
     END_FOR;
END_PROGRAM;
```

2.1 DATA TYPES AND OPERATORS

Linus has six scalar data types: integer, decimal, Boolean, string, format, and set. Table 2.1 illustrates the six types of constants and the operators and functions associated with each data type. Where applicable the equivalent Fortran symbol is given in parentheses.

data type	sample constant	operators and functions
integer	126	+(+), -(-), *(*), %(/), @(**)
decimal	32.536	+(+), -(-), *(*), /(/), @(**)
Boolean	TRUE	&(.AND.), |(.OR.), ¬(.NOT.)
string	'HOW NOW'	LENGTH, SEGMENT, JOIN, INDEX
format	"S5,I3,D8F3"	
set	(/CAT,3.12,5/)	UNION, INTERSECTION, SUBSET, DIFFERENCE, ELEMENT, LENGTH

Table 2.1 Linus Data Types

The string functions LENGTH, SEGMENT, and INDEX correspond respectively to the PL/I functions

LENGTH, SUBSTR, and INDEX. JOIN is similar to the PL/I operator ||.

Format constants are used in connection with input/output statements. The format constant "S5,13,D8F3", is the same as the Fortran format (5X,13,F8.3). Initially students are taught the long form of a format which for this example would be:

"SPACE WIDTH 5, INTEGER WIDTH 5, DECIMAL WIDTH 8 FRACTION 3"

The elements of a set may be any string of characters except the comma. Sets may be embedded within sets to any arbitrary degree.

Mixed modes are not allowed within an expression. However, there exist data type convertors. Figure 2.1 illustrates the Linus data type converters where each arrow represents a converter. An example of a converter is INTEGER_OF_STRING(e)

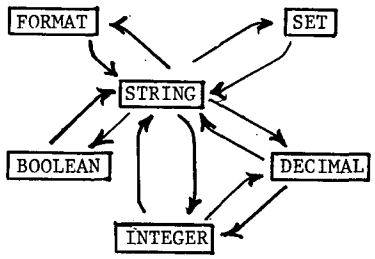

Figure 2.1 data type converters

which converts the string e into an integer.

There is a library of standard functions such as SIN(X) and LOG_E(X).

Multiple scalar values may be formed by declaring arrays and structures. One, two, and three dimensional arrays are allowed. Subscripts may be positive, zero, or negative integers. Structures may be defined over scalars, arrays, and other structures (nonrecursive).

2.2 PROCEDURES, FUNCTIONS, AND MACROS

Procedures correspond to Fortran subroutines. The following example declares a procedure to complement a set with respect to universe, UNIVERSE, using the intrinsic set function DIFFERENCE:

```
PROCEDURE COMPLEMENT
     (SET UNIVERSE ALWAYS CONSTANT/
      SET A ALWAYS CONSTANT/SET B)
      B:=DIFFERENCE(UNIVERSE,A);
      RETURN;
END_PROCEDURE;
```

This procedure would be invoked by:

```
DECLARE SET UNIVERSE, X, Y;
        PROCEDURE COMPLEMENT(SET/SET/SET);
END_DECLARE;
UNIVERSE:=INPUT; X:=INPUT;
CALL COMPLEMENT(UNIVERSE,X,Y);
```

Note that procedures must be declared in the calling program so that the precompiler may verify that all calls are in the intended format. Also, inside the procedure body, the parameters "UNIVERSE" and "A" are "storage-protected" that is, there values may not be altered.

The same procedure declared as a function would be:

```
FUNCTION COMPLEMENT
    (SET UNIVERSE ALWAYS CONSTANT/
     SET A ALWAYS CONSTANT) RETURN SET
     RETURN DIFFERENCE (UNIVERSE,A);
END_FUNCTION;
```

Procedures and functions are independently compilable as with Fortran. In contrast to Fortran they may be recursive.

A facility to allow macros both with and without parameters is also being added to the language.

2.3 BLOCKS

Declare Block

Every identifier (scaler variable, array, structure, procedure and function) must be specified in a DECLARE block. There are no default rules such as the I-N default in Fortran. Scalar, array, and structure variables may be initialized within a DECLARE block. The keyword ALWAYS declares a symbolic constant whose value cannot change within the program unit. An example of a DECLARE block including scalar, array, and procedure declaration statements is:

```
DECLARE INTEGER ARRAY G(-3:25) INITIALLY 29*0;
     DECIMAL MIL INITIALLY 21.4, PI ALWAYS 3.14159;
     PROCEDURE STOR (INTEGER/DECIMAL);
END_DECLARE;
```

An identifier contains one or more alphabetic (A,B,...,Z,_) or numeric (0,1,2,...,9) characters, the first of which is alphabetic. There is no restriction on the length of an identifier.

Control Blocks

Linus has six control blocks which may be used to alter the normal execution sequence of statements.

The FOR block is similar to the FOR statement in Algol 68 in terms of sequence of execution, and allowing arbitrary integer or decimal expressions as loop parameters. The step size is positive or negative depending on whether the keywords UP_BY or DOWN_BY are used. In the example

```
FOR I:=10 DOWN_BY 2 TO -10 LOOP
    .
    .
    .
END_FOR;
```

the step size is -2.

Other Linus control blocks which generate a loop are the UNTIL and WHILE blocks. An example of a while block is

```
WHILE 3.*X < 14.1 LOOP
    .
    .
    .
END_WHILE;
```

The Linus CASE block is similar to the Fortran computed GO TO statement.

The IF block corresponds to the PL/I IF... THEN...ELSE statement. The WHEN block corresponds to the IF block without the ELSE.

Fortran Block

The programmer may write any portion of his program in Fortran, simply by placing the Fortran statements in a FORTRAN block. With this facility the programmer can move back and forth from Linus to Fortran.

2.4 STATEMENTS

The Linus assignment statement has the form:

```
v := e;
```

where v is either a scalar, a subscripted array, or a qualified structure identifier and e is an expression of the same data type. A special case of the assignment statement is the simple input/output statements which have the following forms:

```
v := INPUT;
OUTPUT := e;
```

The formatted input/output statements use the format data type. The general form is:

```
READ(n,y) v1,v2,...,vk;
WRITE(m,x) e1,e2,...,ek;
```

where y and x are format expressions, n and m are device reference numbers, vi is a variable and ei is an expression.

3. SAMPLE PROGRAMS

```
PROGRAM
¢ BUBBLE SORT PROGRAM ¢
     DECLARE INTEGER I,N;
          DECIMAL Z,ARRAY X(1:100);
     END_DECLARE;
     N:=INPUT;
     FOR I:=1 UP_BY 1 TO N LOOP X(1):=INPUT; END_
                                                FOR;
     I:=1;
     WHILE I<N LOOP
          IF X(1)<X(I+1)
              THEN I:=I+1;
               ELSE Z:=X(I+1); X(I+1):=X(I);
                    X(I):=Z; I:=1;
          END_IF;
     END_WHILE;
     FOR I:=1 UP_BY 1 TO N LOOP OUTPUT:=X(I);
     END_FOR;
     STOP;
END PROGRAM;
*DATA
   4
  12
  16
  21
 -31
END_*DATA;

PROGRAM
¢FIBONACCI NUMBERS ¢
 DECLARE INTEGER FIRST, SECOND, THIRD, I;
 END_DECLARE;
     FIRST := 0; SECOND := 1;
     OUTPUT := FIRST; OUTPUT := SECOND;
     FOR I:= 3 UP_BY 1 TO 1000 LOOP
          THIRD := FIRST + SECOND; OUTPUT := THIRD;
          FIRST := SECOND; SECOND := THIRD;
     END_FOR;
     STOP;
 END_PROGRAM;

PROGRAM
¢ PRINTING OUT ELEMENTS OF A SET ¢
     DECLARE SET S; INTEGER I; END_DECLARE;
     S :=INPUT;
     FOR I:=1 UP_BY 1 TO LENGTH(S) LOOP
          OUTPUT:=ELEMENTS(S,I);
     END_FOR;
     STOP;
END_PROGRAM;
*DATA
(/(/A,B/),(/A/),(/B/),(//)/)
END_*DATA;
```

4. OPTIONS AND DEBUGGING AIDS

The user has several options to aid in debugging a program. The TRACE statement may be used to have values of variables printed whenever their values are changed. The NO_TRACE statement deactivates the trace.

The HISTORY statement causes a source line number (as generated by the preprocessor) to be printed whenever the statement is executed. The NO_HISTORY statement deactivates the history.

A DUMP may also be specified anywhere within the program. This will give a list of all variables used in the program and their present values.

Thorough diagnostics are given for all syntax errors. An attempt is made to execute a program regardless of the severity of syntax errors.

There are 20 options available to the user, some of which are indentation of source code, cross reference, environment map (installation dependent information such as magnitude of integers and decimals), and interlisting of Fortran code with Linus code.

5. IMPACT ON CURRICULUM

At Bowling Green State University, course B1 of curriculum 68 (1) is divided into a two quarter sequence. Currently, in this sequence the student is taught a thorough knowledge of Fortran IV and a comfortable knowledge of Snobol4. Using Linus the student learns algorithmic thinking with a language that is structurally simpler and more consistent. As he studies the Fortran programs produced by the preprocessor, he begins a gradual transition into Fortran. By using the FORTRAN block, an increasing proportion of the program can be written in Fortran. At the end of the first quarter the student will be using Fortran, but now he will be more disciplined in his programming. For example, even though there is no WHILE construct in Fortran he will tend to program its Fortran equivalent, having learned it from the preprocessor.

If the Linus experiment proves successful in the first two courses of the curriculum its uses in other courses will be explored. For example, it could be a useful language in teaching Data Structures (I1) or Discrete Structures (B3).

6. FUTURE PLANS

When the full Linus language has been implemented, its effectiveness in teaching will be evaluated. To do this, the beginning students in computer science will be divided into two control groups. One group will be taught Fortran, and the other group Linus. Tests will be given to each control group to determine the effectiveness of Linus.

Thought is also being given to a Snobol4-based Linus and a PL/I-based Linus. These would be useful for a student who already knew Linus and wished to rapidly acquire a working knowledge of Snobol4

or PL/I. A Snobol block and a PL/I block would also be provided to allow the student to move into the desired language.

Acknowledgments

The authors wish to express their appreciation to the students who have participated in the implementation of the Linus language. Special thanks goes to Mr. Charles M. Bernstein for his efforts in coordinating the implementation of the project.

REFERENCES

1. Atchison, W. F., et al, "Curriculum 68 - Recommendations for Academic Programs in Computer Science," CACM, 11, 1968, pp. 151-197.

2. Ralston, A., "Fortran and the First Course in Computer Science," SIGCSE Bulletin, Vol. 3, No. 4, December 1971.

3. Van Wijngaarden, A., Mailloux, B. J., Peck, J. E. L., and Koster, C. H. A., Report on the Algorithmic Language Algol 68, MR 101, Mathematisch Centrum, Amsterdam, October 1969.

4. Wirth, N., Hoare, C., "A Contribution to the Development of ALGOL," CACM, 9, 1966, pp. 413-432.

5. Wirth, N., "The Programming Language PASCAL," Acta Informatica, 1, 1971, pp. 35-64.

6. Wulf, William A., "A Case Against the GOTO," SIGPLAN Notices, Vol. 7, No. 11, November 1972.