EXPERIENCES WITH A SIMPLE STRUCTURED PROGRAMMING LANGUAGE

Victor R. Basili
Albert J. Turner
Computer Science Department
University of Maryland

1. Introduction

A great deal of interest has developed in structured programming [Dahl, Dijkstra, and Hoare, 1972] during the past few years. This paper is concerned with some experiences obtained in the use of a structured programming language in the computer science curriculum at the University of Maryland. The language used was SIMPL-X [Basili, 1973], a language designed and implemented at the University of Maryland.

SIMPL-X was designed to be a transportable, extendable, compiler-writing language that was to be the base language for a family of programming languages. It is, in fact, being used for that purpose as the SIMPL-X compiler [Basili and Turner, 1973] is written in SIMPL-X, and a compiler for the graph algorithmic language GRAAL [Rheinboldt, Basili, and Mesztenyi, 1972] is presently being designed as an extension of the SIMPL-X compiler.

However, some of the design criteria for SIMPL-X have made it a reasonable language for use in programming courses at all levels. These criteria include the requirements that the language

1) have a "simple" control structure and require only a "simple" run time environment.

2) conform to the standards of structured programming and modular program design.

3) support and encourage the writing of readable, well-commented programs.

4) be translatable into efficient object code for most machines.

This paper summarizes the SIMPL-X language and some of the experiences resulting from its use at the University of Maryland. Also included are some opinions on the use of a structured programming language in a computer science curriculum.

2. The SIMPL-X Language

Most of the features of SIMPL-X were derived from components that exist in other programming languages. As an overview of SIMPL-X, some of its main features are

1) the main statement constructions are the assignment, while, if-then-else, case, and call statements. There is no goto statement.

2) a program contains a sequence of procedures and functions, each of which can access a set of global variables, and a set of parameters and local variables.

3) there are compound statement constructions but there is no block structure other than that provided by procedures.

4) facilities for declaring external references and entry points are included.

5) procedures and functions can be recursive if so declared.

6) an extensive set of operators may be used in an expression. These include arithmetic, relational, logical, bit manipulation, shift, and partword operators.

These features are briefly discussed below.

A SIMPL-X program consists of a set of global declarations and a set of segments. Each segment is a procedure or function that consists of a parameter specification, a set of local declarations, and a statement list to be executed whenever the segment is invoked. Execution begins with the procedure designated as the start procedure.

In keeping with the design objective of simplicity, segments may be neither declared as locals (that is, there are no internal procedures or functions) nor passed as parameters. Thus an identifier is either local to a particular segment or global to all segments.

There are five basic statement types in SIMPL-X: assignment statement, CALL statement, IF statement, WHILE statement, and CASE statement. The assignment and CALL statements are similar to the corresponding statements in ALGOL, FORTRAN, or PL/I.

The structure of the IF statement is given by

```
IF <expression>
   THEN <statement list1>
  {ELSE <statement list2>}
   END
```

This causes the execution of <statement list$_1$> if <expression> has a nonzero value and <statement list$_2$> otherwise. (There is no boolean type in SIMPL-X.) The braces ({}) indicate that the ELSE part is optional.

A WHILE statement is used to cause the repeated execution of a list of statements. Its syntax is given by

```
WHILE <expression>
   DO <statement list> END
```

This causes the <statement list> to be executed repeatedly as long as <expression> has a nonzero value.

The CASE statement is essentially an extended IF statement. The syntax is

```
CASE <expression> OF
 \n1\ <statement list1>
 \n2\ <statement list2>
    .
    .
    .
 \nk\ <statement listk>
{ELSE <statement listk+1>}
END
```

where $n_1,n_2,\ldots,n_k$ are constants. When this statement is executed, the value of <expression> determines the statement list to be executed. If <expression> = n_i for some i=1,...,k, then <statement list$_i$> is executed. Otherwise, the ELSE part, <statement list$_{k+1}$>, is executed (if included).

Two additional statements, the EXIT and RETURN statements, are also available to facilitate the abnormal termination of a WHILE loop and the termination of a procedure execution, respectively. The RETURN statement is also used to specify the value that is the result of a function call.

The data types available in SIMPL-X are integer, character, and (character) string. The only structure in the language is a one-dimensional array whose elements must all be of the same type (integer, character, or string). Strong typing is observed as no implicit type conversion is permitted.

The operators available in SIMPL-X include those found in most general-purpose languages and will not be discussed extensively here. However, it does seem desirable to comment on the relational and boolean operators since there is no boolean type in SIMPL-X.

Relational operators (=,<,≠,etc.) are binary operators whose operands must be of the same type. The result of a relational operation is zero if the relation is false, and one if the relation is true. Boolean operators (.AND.,.OR.,.NOT.) apply only to integer operands and also result in one or zero. The expression X .AND. Y has value 1 if both X and Y are nonzero. The .OR. and .NOT. operators function in a similar manner.

The syntax of SIMPL-X is free format, but statement separators, such as semicolons, are not used. This lack of redundancy enhances the simplicity of the language and removes a stumbling block for students, who seem to have trouble learning where to put the semicolons in languages such as ALGOL or PL/I. Comments are delimited by /* and */ and may be inserted wherever blanks may occur.

3. Experience with SIMPL-X

SIMPL-X was first used in a course during the Fall Semester 1972 in an upper-division compiler writing course. A typical programming effort for this course was the writing of a small compiler. Previously both ALGOL and FORTRAN had proved to be unsatisfactory for use in the course, although for different reasons. ALGOL was undesirable for the writing of large programs due in part to the lack of facilities in the language for separately compiled program modules, and in part to the deficiency of the UNIVAC 1108 implementations in handling large programs. FORTRAN, while adequately supporting large programs, did not have good compiler-writing facilities nor did it have desirable structuring.

The second use of SIMPL-X was in the Spring Semester 1973 in a second-semester programming course and in an upper-division systems programming course. The programming course involved the teaching of a second semester of algorithmic problem-solving and the basics of programming. All of the students had been programming in FORTRAN for at least a semester and a half, but its inadequate statement structure made its continued use undesirable. Only two sessions of one and one-half hours each were required to teach them the SIMPL-X language.

In the systems programming course, most of the work was done for the PDP-11 using a SIMPL-X cross-compiler on the UNIVAC 1108. The projects included the design and programming of a loader, a segmentation program, and other operating system routines.

In some respects SIMPL-X was more of a problem to use for the more advanced students in the compiler writing course than for the students in the early course. Many of the more experienced students were hindered by bad habits that they had learned previously and were unwilling to learn to do the kind of thinking and organizing required by the gotoless nature of SIMPL-X. On the other hand, most students in the earlier course enjoyed the structured approach to programming and seemed to have relatively little trouble in learning to program in SIMPL-X.

Perhaps surprisingly, there was little objection by the students in the programming course to the lack of a GOTO statement. In fact, several commented that they especially liked this feature of SIMPL-X.

Additional experience with SIMPL-X has been obtained through its use by some students in special projects. Typical of their generally favorable comments has been a statement to the effect that the syntax and structure of the language are such that fewer errors are made when programming in SIMPL-X as opposed to languages previously used.

As a result of these favorable early experiences, SIMPL-X has been adopted for usage in courses by several faculty members. These courses include the second-semester programming course, graduate and undergraduate compiler-writing courses, graduate and undergraduate systems programming courses, and a course in the structure of programming languages. Additionally, some higher-level courses use a structured SIMPL-X-like language for expressing algorithms and program-type function definitions. Examples of these courses are a course in the certification of programs and courses in semantic models for programming languages.

The following are results from a questionnaire submitted to two classes whose students used SIMPL-X after programming in FORTRAN for at least a semester and a half. These results were not obtained by using statistically valid evaluation procedures. However, they were obtained without any attempt to bias the students in favor of SIMPL-X and have been considered worthy of note by many faculty members.

	Question	Yes	No
1.	Was SIMPL-X easy to learn?	33	5
2.	Is SIMPL-X easy to program in?	29	8
3.	Did SIMPL-X contribute to your understanding of programming and algorithms?	35	2
4.	Do you think SIMPL-X would be a good first programming language?	26	13

4. Conclusion

Simplicity is an important attribute of any programming language [Hoare, 1973]. Attempts have been made to achieve simplicity by using a subset of an existing language [Holt, 1973]. Although using a language subset may achieve the desired simplicity, it can also cause confusion if, for example, a construction occurs that is valid in the superset but not in the subset [Hoare, 1973]. Thus the use of a complete, simple language is preferable to the use of a subset of an existing language.

The simplicity and lack of syntactic redundancy in SIMPL-X seem to be important factors that contribute to the writing of more error-free programs by both experienced and inexperienced programmers. Additionally, the simplicity of SIMPL-X aids its compiler in error analysis, even though the lack of redundancy in the syntax can cause problems in error recovery. The fact that SIMPL-X is a simple, yet reasonably powerful, language that encourages the use of structured programming techniques makes it a desirable language to use at all levels in a computer science curriculum.

There is some disagreement regarding the stage at which a language like SIMPL-X should be introduced into the curriculum. One view is that such a language should be used in a first course in computer science. It is argued that too many programmers have already been at least partially hindered by having been taught to program using poor programming principles primarily due to inherent deficiencies in the language used. The other view maintains that the entrenchment of FORTRAN and similar languages, plus the scarcity of reasonable alternative languages in the "outside world", rule against the use of languages such as SIMPL-X. Those holding this latter opinion feel that it is important to provide a student with a programming tool that is likely to be available wherever he may go.

These opposing points of view, coupled with the common problem of what to teach a student who will take only one programming course, have resulted in making the structured programming course the second programming course at the University of Maryland. It is generally felt that one semester of FORTRAN will not do too much damage to a student since few, if any, habits (good or bad) are learned in one semester of an introductory course. Thus SIMPL-X is currently being used in the second programming course and is not used in the introductory course. However, future developments, such as a more definite delineation of courses, could well result in the use of SIMPL-X in an introductory course for majors.

We feel strongly that the art of programming is developing more and more into a science, and that the use of methods such as structured programming should be taught at an early stage in the development of programmers and computer scientists. It is time to do away with practices that encourage students to use "micro tricks to save micro seconds" [Holt, 1973]. Our experiences with SIMPL-X have led us to believe that not only do students prefer programming in a structured language but they also make fewer errors and write "better" programs when using structured programming techniques. We also feel that the use of a structured programming language like SIMPL-X gives students a better understanding of programing and more respect for programming languages as a powerful tool for problem solving.

5. Acknowledgments

The development of the SIMPL-X language and its compiler was supported in part by the Office of Naval Research under Grant N00014-67-A-0239-0021 (NR-044-431) and in part by the Computer Science Center of the University of Maryland.

6. References

Basili, V.R., SIMPL-X. A Language for Writing Structured Programs, TR-223, University of Maryland, Computer Science Center, January 1973.

Basili, V.R., and Turner, A.J., A Transportable Extendable Compiler, TR-269, University of Maryland, Computer Science Center, October 1973.

Dahl, O.J., Dijkstra, E.W., and Hoare, C.A.R., Structured Programming, Academic Press, 1972.

Hoare, C.A.R., Hints on Programming Language Design, Address at SIGACT/SIGPLAN Symposium on Principles of Programming Languages, October 1973.

Holt, R.C., Teaching the Fatal Disease (or) Introductory Computer Programming Using PL/I, SIGPLAN Notices 8, 5, May 1973.

Rheinboldt, W.C., Basili, V.R., and Mesztenyi, C.K., On a Programming Language for Graph Algorithms, BIT 12, 2, 1972.