INTRODUCTION TO THE SIMSCRIPT II PROGRAMMING LANGUAGE

Philip J. Kiviat

February 1966

P-3314

Introduction to the SIMSCRIPT II Programming Language

Philip J. Kiviat*

The RAND Corporation, Santa Monica, California

This is an introduction to a programming language. It is a glimpse into the minds of the designers and implementors, a documentary of their views of what a programming language should look like, and a peek at the details of the language called SIMSCRIPT II. It is not a descriptive report on the language, a programmers manual or a coders guide.

The value of this paper lies in its presentation of the authors'** design philosophy. Students of programming languages, and particularly of simulation programming languages, will note the way in which SIMSCRIPT II has evolved. It looks somewhat like SIMSCRIPT, a little like FORTRAN, a little like ALGOL; yet it has a form of its own. It has been the authors' intent to incorporate their accumulated experience into SIMSCRIPT II. The language therefore is reminiscent of its predecessors in many ways. But it is uniquely different in a number of important ways. If it were not, there would be little rationale for its existence. It cannot do everything that every other programming language can do, but it can do a great number of things. And it can be expanded to reproduce features of other programming languages, and to introduce new user-oriented facilities.

The following sections trace the authors thoughts on the design, implementation and structure of SIMSCRIPT II. These sections are not exhaustive; they do reflect the major considerations of the language design.

---

*Any views expressed in this paper are those of the author. They should not be interpreted as reflecting the views of The RAND Corporation or the official opinion or policy of any of its governmental or private research sponsors. Papers are reproduced by The RAND Corporation as a courtesy to members of its staff.

**The SIMSCRIPT II design team consists of B. Hausner, P. J. Kiviat and H. M. Markowitz. Markowitz is the team leader and Hausner is responsible for the programming. Kiviat is preparing the documentation.

Design Considerations

The basic design rule that has influenced SIMSCRIPT II is "make the language easy to use." Since use, in a practical sense, extends from problem analysis and coding to debugging and documentation this rule has been foremost at all times. It has prompted the language to evolve into a free-form, English-like sentence structure that is easy to write and easy to read. This structure has made SIMSCRIPT II very much self-documenting, especially with its widespread use of synonyms and alternate sentence forms. Statements can be written so that they convey, through their wording as well as their structure, the intent of the tasks they perform.

A language must also be easy to use in the sense that programs have a high probability of running each time they are submitted. The SIMSCRIPT II language structure, with its widespread use of synonyms for those who cannot remember the correct words and minimal coding rules assists in this task. In addition, the compiler detects and corrects a large number of incorrectly written statements; these corrections may not always be right, but they always permit a compiled program to execute up to the point of the correction (or past it if this can happen).

Our choice of statement forms and selection of words in statements has essentially been guided by the principle "make every statement easy to remember, easy to write, self-descriptive and necessary." This last factor has been important in keeping the number of statements in the language down to a minimum. Our rule has been to reject the addition of a new statement if its actions can easily be accomplished by two or three existing statements, or if it can easily be added to the language by a user.

Implementation Considerations

Our implementation guidelines have been few, but permissive. The structure of the SIMSCRIPT II compiler was determined by the need for the language to be extendable and implementable on many computers. The SIMSCRIPT II compiler is written in SIMSCRIPT II itself, making the translator self-documenting to a SIMSCRIPT II programmer and having minimal dependence on machine language instructions. Such schemes have been used before, notably in the NELLIAC compilers.

The language can be expanded by writing new commands in terms of existing commands (what we call "scripting") or by adding to the recognition and compilation parts of the compiler. The result of compiling this new compiler is a SIMSCRIPT II compiler that is augmented by the new commands. The language is thus open-ended in the sense that it can be expanded or modified at any time.

Implementation on different computers is carried out by changing the basic machine language generators of the compiler and going through several bootstraping compilations and assemblies. While this entails more than a trivial amount of work, it is far less costly and time-consuming than writing an entire new translator.

Two other considerations were important in programming the compiler: dynamic storage allocation and generation of efficient object code. These are two somewhat contradictory requirements and todays compilation scheme and memory structure are the result of several design iterations. The code that is generated is generally as good as that produced by a good programmer although there is no attempt made at index register optimization or subexpression reduction. Storage of all arrays and of local variables is fully dynamic and subroutines can be swapped with one other and with data arrays to provide an automatic program overlay feature.

Language Considerations

The two preceding sections have described our guidelines for determining what we wanted SIMSCRIPT II to do. This section, and the one following, describe how we decided what the language should be able to do, what it contains and how it is presented.

As implemented at RAND, SIMSCRIPT is a source language to FORTRAN translator. The decision to produce this type of SIMSCRIPT compiler speeded production of the language but restricted it to the capabilities of FORTRAN. One of the first SIMSCRIPT II decisions was that the new language would not go through another programming language, but would compile directly from source language to machine code. (Assembly language now and hopefully binary in the future.) As such, the language had to contain all the elements of an algebraic compiler.

SIMSCRIPT II also had to be compatible with SIMSCRIPT in so far as it would be able to accept (after a simple SIFT-ing process) SIMSCRIPT programs. The goals of free-form inputs and less rigid rules drew SIMSCRIPT II away from the format of SIMSCRIPT, but the commands are similar. SIMSCRIPT II increases the power of the SIMSCRIPT commands by extending their scope to fit a wider range of uses. The transition has been natural in the sense that if you were to ask the question "How could SIMSCRIPT be improved"? you would arrive at the SIMSCRIPT statements of SIMSCRIPT II.

SIMSCRIPT II also had to contain some important features of its own. Some of these are the new format free input-output commands, diagnostic commands and data bank concept. This concept raises the language to the level of information storage and retrieval language. There are other new features but we will not mention them here; numerous small features cannot be named but must be used to be appreciated, and several major features cannot be described now for proprietary reasons.

Finally, SIMSCRIPT II was designed so that it forms a natural hierarchy of languages. Computer users come from a spectrum of individuals with different levels of knowledge and different kinds of problems. We didn't want to produce a language that would throw a programming novice into the sophisticated world of symbol manipulation and recursive list processing or force a business programmer to read about simulation commands and random variate generation. SIMSCRIPT II had to be easy to learn, easy to use, and yet still be applicable to a wide range of problems.

To solve the teaching problem, the problem of merging the many ideas of SIMSCRIPT II into a coherent whole, and to show how the language has uses that range from a simple teaching language to a compiler-compiler, we present SIMSCRIPT II as a series of seven languages. Each language or level contains the preceding levels as a compatible subset and enough new commands and features to make it distinctly more useful than the preceding level. A programmer with a particular requirement need read only those levels that contain the concepts and statements needed to solve his problem. And a novice can be led gradually from the fundamentals of programming to the subtleties of the Level VII language.

SIMSCRIPT II is thus a balance of design, implementation and language considerations. Compilation and execution times have been balanced against language extension and availability; mathematical terseness has been balanced against descriptive power. The correctness of our choices will be judged by the users of the language and by the amount by which SIMSCRIPT II reduces the total time it takes a person to solve a problem.

Language Structure

SIMSCRIPT II contains approximately 50 statements. Some of these are definitional, most are executable. Almost without exception these statements have both long and short forms, contain synonyms to improve their readability, and allow options in their use. When options are permitted but not taken, normal or default conditions are implied. The statements have been designed for maximum flexibility, ability to combine with other statements and in some instances, multiplicity of purpose. The overriding influence in all statements has been ease of use; ease with regard to programming, debugging and documenting. The language could have been made more succinct, but then it would not be as good a communication medium. It could have been made less redundant, but then it would require artifice to do what can now be done directly. We believe the form chosen will be agreeable to analysts, programmers and managers and that it will enhance communication between people as well as between programmers and computers.

Without going into the individual statements these are the characteristics of the seven levels of SIMSCRIPT II.

Level I. Level I introduces the basic concepts of programming through a simple teaching language (which we fondly call SIMPLESCRIPT). This language contains only 11 statements and few programming subtleties. It is easy to learn, and serves the purposes of introducing a novice to the concepts of programming, the structure of SIMSCRIPT II programs and the use of computers to solve problems.

Level II. Level II extends the simple concepts of SIMPLESCRIPT to a language of FORTRAN-like capability. The 12 new statements of this level introduce looping, subprograms and subscripted variables and make it a useful programming language; of course its use is limited to

scientific and business applications that do not require more power than an algebraic compiler can provide.

Level III. The 6 statements of Level III raise the language to a near ALGOL status, for beside these statements a host of variations in the use of previous statements are introduced. In this level the language improves on its decision-making and searching capability; it also introduces a storage allocation statement that extends the control a programmer has over dynamically stored variables. A report generator rounds out the level.

Level IV. In Level IV the algebraic language structure of Levels I through III is extended by introducing the entity-attribute-set concepts of SIMSCRIPT and the statements that support them. Addition of commands for defining and manipulating non-numeric data make this level a true symbolic list-processing language. SIMSCRIPT users will appreciate this level for its replacement of the SIMSCRIPT definition form by a series of English-like descriptive sentences, expansion of the entity concept and more extensive set manipulation statements.

Level V. Events, simulation commands, entity control statements and automatic data collection commands constitute the additions of Level V. This level completes the sequence of languages which lead from algebraic processor to list processor to simulation language. Several new commands permit much greater control over the occurrance of events than was possible in SIMSCRIPT. These same commands make it possible to trace or record the progress of individual entities or types of entities without actually writing trace and analysis statements throughout a program. Simulation programs can be written more concisely, and can be debugged with less difficulty.

Level VI. Level VI is somewhat hardware oriented. Its commands allow a programmer to create banks of data in secondary storage and reference these data in programs without regard to where or how they are stored. It can be viewed as permitting SIMSCRIPT II to operate as a general purpose information retrieval language. This level is a major innovation in SIMSCRIPT II.

Level VII. SIMSCRIPT II in SIMSCRIPT II forms Level VII. In this level a user learns how to add commands to the SIMSCRIPT II language and how to modify existing commands to suit his specific needs. It is this level that makes SIMSCRIPT II an open-ended language and a language that will probably be used for constructing other programming languages. While this level can be used by individual programmers it probably will not; it will most likely be used by systems programmers to create applications - oriented additions to the basic language.