THE DESIGN AND IMPLEMENTATION OF A STRUCTURED PROGRAMMING LANGUAGE WITH FEW ARBITRARY SYNTACTIC RESTRICTIONS - THE INTERPRETIVE PHASE

A dissertation presented by

P.B. Gibbons

in partial fulfilment of the requirements

for the degree of

Master of Science in Computer Science

at

Massey University

August 1972

ACKNOWLEDGEMENTS

There are a number of people I wish to thank for the parts they have played, either directly or indirectly, in the production of this thesis.

Firstly, I would like to sincerely thank my supervisors, Professor G. Tate and Mr. R.W. Doran for all the guidance and encouragement they have given me during the preparation of this work. I would particularly like to thank Bob Doran for providing the original syntax of MUSSEL.

I am also extremely grateful to Miss Nola Gordon for the many long hours spent with her discussing and designing the implementation of MUSSEL; to Mr. P.J. Herbert for his co-operation in printing the structure diagrams; to the Massey University Computer Unit for absorbing much of the cost of the project; to the Mathematics Department for the use of its electric typewriter; and last, but by no means least, to the IBM 1620 II Computer at Massey, with profound apologies for the considerable amount of abuse it has quietly suffered during the development of the Interpreter.

I would also like to take this opportunity to thank my parents for their unfailing generosity and tolerance during my years of study at Massey, particularly while this thesis was being written.

Finally, I would like to say a very special 'thank you' to Noela for her constant patience, understanding, and untold assistance during the preparation of this work. For these qualities, alone, I shall always be grateful.

Massey University
August 1972

P.B. Gibbons

TABLE OF CONTENTS

CHAPTER 0

<u>INTRODUCTION</u>

<u>THE CASE FOR A NEW LANGUAGE</u>

The first and most important question that must be answered is, "Why in fact do we need a new programming language?". In order to answer this, we must really go back and try to answer the question, "What is programming?".

In designing MUSSEL, we have been very much influenced by the ideas of E.W. Dijkstra [35] and N. Wirth [7], [8], and [34]. Wirth, in particular, provides some strong criticism of present day programming courses, and in doing so, formulates some well-reasoned and constructive answers to the question, "What is programming?", or rather, "What should a programming course be?". His views, in fact, are representative of a growing dissatisfaction among many Computer Scientists with the conventional methods being taught to students as an aid to writing programs.

The process of writing a program to solve a particular problem on the computer may be divided into 2 fundamental steps. The first of these is the construction of a well-defined and efficient algorithm to solve the problem. The second is the translation of this algorithm to a well-structured, effective, and reliable program for the computer.

$$\begin{tikzcd}\boxed{\begin{array}{c}\text{THE}\\ \underline{\text{PROBLEM}}\end{array}} \arrow[r, "\text{(Step 1)}"] & \boxed{\begin{array}{c}\text{THE}\\ \underline{\text{ALGORITHM}}\end{array}} \arrow[r, "\text{(Step 2)}"] & \boxed{\begin{array}{c}\text{THE}\\ \underline{\text{PROGRAM}}\end{array}}\end{tikzcd}$$

Wirth's main criticism is that too many present day courses concentrate on step 2 of this process, and therefore tend to ignore the fundamentals of step 1. In other words, while language details are being taught and examined at great length, the actual heart of the matter, the construction of algorithms, is largely left to the intuition of the student. Experience has shown that languages are not merely tools to communicate with the computer, but are the framework in whose terms the programmer thinks and designs. And as many of the languages used are extremely awkward to express algorithms in, it is not therefore surprising that the end product of such courses is more the knowledge of the details and idiosyncrasies of the languages used, rather than an appreciation of the disciplined reasoning needed to develop algorithms. In fact, as Wirth puts it, the course could almost be described as a 'computer disappreciation' course! This is particularly true of a language such as FORTRAN.

More emphasis, then, should be placed on step 1 of the programming process i.e. on teaching the student disciplined methods of constructing algorithms. Wirth believes that this discipline can be achieved only by emphasising the essential principles of program structuring from the very beginning. He believes that programming should be considered as an activity consisting of a succession of steps, each one breaking up a given task into a number of subtasks, starting with the original problem, and ending when all subtasks can be expressed by elementary statements of the underlying programming language. This process of successive refinement of tasks was originally proposed by Dijkstra, and has now come to be called "structured programming".

A good way of picturing the process is by the use of the "structure diagram" (R.W. Doran [3], [4], & [5]). In such a diagram, refinement of an algorithm is represented pictorially by a 'tree', the nodes of which are boxes essentially containing statements of subalgorithms. (More strictly, such a diagram represents a List, rather than a tree, since the algorithms represented may be resursive.)

Having assumed, then, that the use of structure diagrams was a good method of formulating algorithms, we were then faced with the problem of choosing a suitable language which was compatible with these ideas, and which would therefore make step 2 as natural as possible for the student. FORTRAN, besides being unstructured, has a number of 'bad' features (see Chapter 1, and also Appendix E), and is therefore not particularly satisfactory. ALGOL, on the other hand, is indeed structured. Some of the concepts of the language, however, are probably a little sophisticated for instruction in a first year course. And among other things, we had no PL/I compiler.

There were, then, 2 options open to us. The first would be to modify an existing language so as to cover up or remove some of its 'bad' features. While this would at least have the advantage of teaching the student a language which is in wide use, it would not be entirely satisfactory, as it is unlikely that <u>all</u> of the bad features could be easily removed, if at all.

The other option, the one we chose, was to create a new language, entirely compatible with our ideas on structured programming, and yet simple enough for the average student to be able to grasp quickly and naturally. Although this approach has the disadvantage of teaching the student a language which is not in wide use, we believe that use of the language, in conjunction with a course on structured programming, would so train the student in the basic fundamentals of programming, as we have defined them, that he should have no difficulty in adapting to other, more widely used languages in later, more advanced courses.

The language we have designed has been called MUSSEL (Massey University Structured Student Language). The language has 2 main features:

1. It is structured.

2. The syntax has been made as natural as possible for the student.

In fact, both the structure and syntax of MUSSEL resemble very closely the form of the structure diagram. In this way, we believe that the student will be able to translate his structure diagram quite easily to program form.

To summarise, then, we have created a language to fit in with what we believe is a good strategy for formulating and expressing algorithms. In doing so, we have essentially endeavoured to shift the emphasis in a programming course from the learning of a language, to the construction of algorithms.