Copyright is owned by the Author of the thesis. Permission is given for a copy to be downloaded by an individual for the purpose of research and private study only. The thesis may not be reproduced elsewhere without the permission of the Author.
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 cooperation 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

P.B. Gibbons

August 1972

- ii -
# TABLE OF CONTENTS

Chapter 0  INTRODUCTION - THE CASE FOR A NEW LANGUAGE . . 1

Chapter 1  A DESCRIPTION OF THE LANGUAGE . . . . . . . . . . . . . 5

1.1. Admissible characters 5
1.2. Constants 5
1.3. Operators 6
1.4. Names 7
1.5. Variables 8
1.6. Arrays and subscripted variables 8
1.7. Assignment instructions 9
1.8. Program structuring 10
1.9. Conditional groups 10
1.10. Loops 13
1.11. Procedures 16
1.12. Recursion 17
1.13. Dynamic arrays 18
1.14.1. Input 19
1.14.2. Output 20
1.15. An example 26
1.15.1. The structure diagram 27
1.15.2. The program 28

Chapter 2  AN OVERVIEW OF THE IMPLEMENTATION . . . . . . . . 29
3.1. The Interpreter Stack 32
3.2. The Symbol Table 34
3.3. The basic structure of the Interpreter 36
3.4. Expression evaluation 39
3.5. Assignment instructions 44
3.6. Main arrays & subscripted variables 46
3.7. Number representation & arithmetic 52
3.8. Logical operations 53
3.9. String storage & manipulation 54
3.10. The Line Number operator 63
3.11. General choice-groups, conditional groups, and if-groups 63
3.12. Repeat-group instructions 66
3.12.1. Simple REPEAT without control 67
3.12.2. REPEAT-while instruction 68
3.12.3. REPEAT-until instruction 70
3.12.4. REPEAT-times instruction 71
3.12.5. REPEAT-for instr. grp. (with test) 75
3.12.6. " " " (without test) 78
3.12.7. REPEAT-set instruction group 80
3.13. CASE-group instruction 83
3.14. Functions and procedures 88
3.14.1. Overview of their implementation 88
3.14.2. The link data 90
3.14.3. Fn. & proc. calls & declarations 91
3.14.4. The use of local variables in expressions & assignments 99
3.14.5. Dynamic arrays 100
3.15. Input/Output 105
3.15.1. Input 105
3.15.2. Output 110
3.15.2.1. Unformatted output 111
3.15.2.2. PIC controlled O/P 115
Chapter 4 POSSIBLE DEVELOPMENTS OF THE SYSTEM . . . . . . 124

APPENDICES AND REFERENCES . . . . . . . . . . . . . . . . . . . . 128

APPENDIX A The Syntax of MUSSEL in BNF
APPENDIX B A Summary of the Intermediate Language
APPENDIX C The Interpreter Structure Diagrams
APPENDIX D The Interpreter SPS Program
APPENDIX E Error Analysis of First Year Student FORTRAN Programs at Massey University
APPENDIX F 2 Sample MUSSEL Programs (with corresponding object code)

REFERENCES
CHAPTER 0

INTRODUCTION

THE CASE FOR A NEW LANGUAGE

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.
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 recursive.)

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 all 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.