COROUTINES: A PROGRAMMING METHODOLOGY, A LANGUAGE DESIGN, AND AN IMPLEMENTATION

by

Christopher David Marlin, B.Sc. (Hons.)

Department of Computing Science
The University of Adelaide

A thesis submitted for the degree of Doctor of Philosophy

16th November 1979

TABLE OF CONTENTS

Chapter Page

SUMMARY

Coroutines, although known and discussed for some years, are often regarded as giving rise to poor program structure and inefficient program execution, and are not widely available in programming languages. This thesis describes a top-down methodology for programming with coroutines, which is based on the extension of the notion of hierarchical program structure to cover programs containing coroutines. The support and promotion of this methodology formed part of the motivation for the design of a programming language with coroutines; this design was also motivated by a desire to experiment with the provision of a coroutine facility in the framework of a modern, general-purpose programming language. Among the goals of the language design was the encouragement of a close correspondence between the static, textual structure of a program written in the language, and its structure at run-time.

The language with coroutines, called ACL, was designed using a novel approach to language design which firstly involves the division of the design task into the design of the semantics of the language, followed by the design of its syntax. The first of these is further subdivided into three parts, one for each of three aspects of the language being designed: data structures, sequence control and data control. For each aspect, a semantic model was devised which is tailored both to the aspect, and to the needs of this particular language design. These models (which vary in the extent to which they can be described as formal) were used to compare this aspect of existing language features, and to make similar comparisons among alternatives being considered for inclusion in this aspect of ACL; these comparisons were very much facilitated by the models, with both similarities and differences being made clear. In this way, descriptions of the semantics of each of the three apsects of ACL were obtained; these were then given a syntactic form in the Pascal idiom.

The programming language ACL has been implemented, and some aspects of this implementation are also briefly discussed.