# AN IMPLEMENTATION OF A FORTRAN SOURCE CODE REARRANGER AND DOCUMENTATION GENERATOR PROGRAM

by

ARTHUR SCOTT MYERS

B.S., Lehigh University, 1969

---

A MASTER'S REPORT

submitted in partial fulfillment of the

requirements of the degree

MASTER OF SCIENCE

Department of Computer Science

KANSAS STATE UNIVERSITY
Manhattan, Kansas

1983

Approved by:

Major Professor

TABLE OF CONTENTS

# I. INTRODUCTION

One of the most important, and unfortunately most neglected needs in the software development process today is documentation, especially documentation that aids the programmer or systems analyst in readily understanding computer programs' source code. The intent of this implementation project is to address that need by developing a software tool that will provide documentation showing a program's hierarchical top-down order and alphabetical listing of its modules; it will also rearrange the program's source code into this top-down order of modules.

This documentation should have a very high utility to the programmer because it will concisely show, typically on one or two pages, a complete program's overall structure and detailed inter-module relationships via the report that shows the top-down order of modules. This could be viewed as a table of contents for the program's source code, because it will show each module's location within the source code file. It should be even more useful since the same report will also contain structure, size and inter-module relationship information for each module. The report displaying the alphabetical listing of modules can be thought of as an index to the program, but it too will contain information that goes beyond that function. Furthermore, since the source code file will actually be rearranged into its hierarchical top-down order, an implicit documentation feature will come about because when one reads the source

code it will be in logically related order. Thus, if the reader wants to look at the code of a module called by the module he or she is currently studying, it will usually be a matter of flipping to the next page or two, in most cases, instead of having to make a time consuming search through the entire source code listing.

Because of the documentation and organization capabilities that this software tool will provide, it should prove to be quite valuable to the programmer enabling he or she to improve both productivity and quality with the following benefits:

(1) ordering of source code in top-down order

(2) program documentation shown in top-down and alphabetical order

(3) understanding of program structure

(4) rudimentary measure of software quality - e.g. program unit sizes, percent of comments, etc.

(5) improved productivity in developing new programs

(6) greater ease of maintaining and enhancing existing programs

While using this tool in developing and maintaining one's own software should be of considerable worth, it should be even more useful when one has to work with programs developed by someone else. Furthermore, the rudimentary measures of software quality provided should be of value to the software manager.

Previous work that has been done in this area can be

classified into two approaches: (1) analyzing module relationships of FORTRAN programs which centers around intermodule communications and how the data shared between the modules are affected and (2) identifying basic blocks of FORTRAN programs. The first approach constructs the call graph of a FORTRAN program [2] in order to examine interprocedural communications. This call graph serves as a data structure for providing static representation of interprocedural relations for nonrecursive programs. The algorithm for constructing the call graph is used in the PFORT Verifier [3], a program which checks FORTRAN programs for compliance with a portable subset of FORTRAN. The uses of the call graph in the Verifier are:

(1) checking the legality of module linkage (i.e. the matching of parameter number, usage, type, and structure)

(2) checking for unsafe references to interprocedural communications which may have implementation dependent results

(3) checking the sharing of global data areas

(4) outputting a depiction of the call graph for use as documentation and as a debugging aid

The construction of a program's call graph differs from this project because it does not actually rearrange the source code, nor does it create the tabular documentation showing the program's top-down order (table of contents) and its alphabetical listing of modules (index). On the other hand,