INL/CON-05-00719
PREPRINT

Restructuring of RELAP5-3D

International RELAP5 Users Seminar

George L. Mesina
Joshua Hykes

September 2005

The INL is a
U.S. Department of Energy
National Laboratory
operated by
Battelle Energy Alliance

Restructuring of RELAP5-3D

Dr. George L. Mesina and Joshua Hykes
Idaho National Laboratory, P.O. Box 1625, Idaho Falls, ID, 83415-3890, George.Mesina@inl.gov
Penn State University, 0355 ATHERTON HALL, UNIVERSITY PARK, PA 16802, jmh539@psu.edu

1.0 INTRODUCTION

Converting a hierarchical language computer program to structured code is a standard method to improve it in many ways. Structured code is comprised of a sequence of blocks of code that each has only one entry point, one exit point, and is itself comprised of individual lines of code or sub-blocks. Structured code is easier to read as the logic paths are clearer. This results in reduced development and maintenance costs. It also leads to greater robustness and increased longevity of the code.

FOR_STRUCT [1, 7] is a commercial restructuring tool that guarantees to reengineer unstructured FORTRAN programs to create structured programs that always produce exactly the same calculations. FOR_STRUCT was applied to RELAP5-3D [2]. This was an involved process due to inherent limitations of FOR_STRUCT and the complexity of RELAP5-3D. The process for restructuring RELAP5-3D and measurements of the improvements are reported.

2.0 BACKGROUND

2.1 Structured Programming

Programs are written from algorithms generated to solve specific problems. A program for implementing the algorithm can be written in numerous ways. Many of these ways can be difficult to read and understand. Others are easy to read, understand, and modify. The latter is preferable to the former because the time and cost for development and maintenance is less.

However, long experience of the computer industry indicates that the former is too often produced. In fact, so many of these kinds of programs were written in the early decades of computer programming that means to alleviate the problem were sought. One solution was to develop a language that strictly controlled the ways an algorithm could be implemented; the language was called ADA [3]. Another solution was to develop paradigms for writing code that, if adhered to, produced programs that were easier to read, understand and maintain. The best known are structured and object oriented programming. So successful was structured programming for procedural programs, that by the mid 1970s, college texts on structured programming in FORTRAN [4] were in use.

Procedural programs that are not structured can be characterized as having interwoven logic paths. The colloquial term for this is spaghetti code. Two otherwise separated logic paths of sequentially executed statements can be intermingled by a GOTO that transfers execution from the first path to the interior of the second. Backward GOTO statements have a greater potential to interweave than forward GOTO statements because they can cause portions of the same or a different logic path to be repeated. The GOTO statement has such potential to lead to unstructured code, that it was considered harmful by at least one of the greatest computer scientists, Edsger Dijkstra [5].

According to Federal Standard 1037 [6], structured programming is a technique for organizing and coding computer programs in which a hierarchy of modules is used, each having a single entry and a single exit point, and in which control is passed downward through the structure with no unconditional branches to higher levels of the structure. There are three types of flow control: sequential; test (if and case); and iteration (loop). We use the term "block of code" or simply block in place of module for languages with module constructs, such as FORTRAN 90.

The value of structured programming is manifold. Structured programs are easier to read and understand than unstructured programs. This most always leads to reduced time and cost for maintenance and development. Further, with structured coding it is easier to extract and reuse a portion of the code in future computer programs. Structured code tends to be more robust, having fewer or no program errors in the implementation of the underlying algorithm. Structured programs tend to have a much greater longevity; some are still in use today in the form of libraries such as IMSL and LINPACK. Finally, it takes less time for new developers to learn the program and be effective working on structured code.

2.2 Code Restructuring

Most computer programs are written by scientists and engineers who have little or no training in structured programming. Very few programs start out as structured programming. Moreover, subsequent development and maintenance work can lead to loss of structured coding as new features and patches are added.

Fortunately, it is possible to re-engineer an existing program into a structured program. There are commercial software packages available that do this.

The FOR_STRUCT software tool was selected for restructuring RELAP5-3D code. It reengineers the logic paths within subroutines to produce structured code with block-oriented, Fortran 90 constructs. The vendor guarantees that FOR_STRUCT code restructuring has no impact on the calculated results. FOR_STRUCT has the

added advantage of applying consistent style rules, such as indentation and blanks around keywords and operators.

3.0 RELAP5-3D RESTRUCTURING

Code restructuring of RELAP5-3D is complicated by the extreme complexity of the coding and the limitations of FOR_STRUCT. Three limitations of FOR_STRUCT are relevant to restructuring RELAP5-3D: inability to produce completely structured code for a very long and intricately interwoven subroutine; inability to restructure FORTRAN 90 code; inability to handle pre-compiler directives. Means to overcome all three limitations are reported in this section.

3.1 Overcoming FOR_STRUCT Limitations

The first complicating factor to be dealt with is the length and complex interwoven logic paths of some RELAP5-3D subroutines. Applying FOR_STRUCT to these produced code with fewer GO TO statements and was closer to being structured programming, but that was not yet fully structured. In such cases, reapplying FOR-STRUCT to its own output produced code with even fewer GO TO statements that was either fully structured or much closer. It was found that, in general, little improvement was made beyond the third application of FOR_STRUCT; therefore, three iterative applications were used for all subroutines.

The second complicating factor was FORTRAN 90. FOR_STRUCT was written to convert older FORTRAN coding to FORTRAN 90, but it does not recognize most of the post-FORTRAN 77 constructs of FORTRAN 90 and therefore cannot be used to reengineer FORTRAN 90 code. There are several ways to handle this. The method developed for RELAP5-3D was to pre-process the source code. All references to derived type arrays, for example, were replaced with legal FORTRAN 77 variable names. The derived types were restored after FOR_STRUCT had been applied.

Pre-compiler directives that are used throughout RELAP5-3D are the third and most difficult complicating factor. FOR_STRUCT does not handle conditional code. To overcome this, a method of preprocessing and post-processing the files was devised. It is described in Subsection 3.1.1

3.1.1 Handling pre-compiler directives

For RELAP5-3D subroutines with one or more directives, the file must be pre-processed to eliminate the directives before applying FOR_STRUCT. First, a define file that activates directives is created and pre-pended to the file. The pre-compiler processes the resulting file and then FOR_STRUCT can be applied.

For a file with zero or one directive, this is a simple process. A duplicate of each pre-compiler directive is created immediately below it the original in the source code, then the duplicate is made into a comment. After pre-processing, FOR_STRUCT is applied. During post-processing, the commented ENDIF-directives that are often misplaced by FOR_STRUCT are found by visual inspection and moved manually to the correct position.

Note that pre-processing expands the included COMDECKS and this must be undone after restructuring, although this can be automated.

For files with 2 or more directives, the process is more involved. If the pre-compiler directives are nested or are mutually exclusive, no define file suffices to build a single source code file that covers all possibilities for conditional code inclusion. In these cases, a minimal set of define files that fully covers all such possibilities is constructed. With each define file, the source file is handled as explained for the case of zero or one directive. After the source file is processed with each define file, the resulting output files are combined manually to construct the restructured subroutine.

3.2 Controlling complexity

With these operations in mind, the subroutines were ranked according to their complexity. Smaller routines are simpler to convert than larger ones and code with more pre-compiler directives is generally more complex than code with less. The subroutines were grouped according to the number of unique pre-compiler directives they had. See Table 1.

# Directives	# Files
0	22
1	116
2	173
3	106
4	53
5	37
6	9
7	7
8	4
9	3
10	7
11	7
12	2
14	4
15	1
17	1
22	1
58	1

Table 1 Directive Groupings.

Within each directive group, the subroutines were sorted from smallest to largest. The subroutines were then restructured according to this order. As each new difficulty arose, means to handle it were developed as was summarized in Section 3.1.

4.0 TESTING

With all the hand manipulation and pre- and post-processing operations that must be performed, testing is absolutely essential to ensure against introduction of code bugs.

Each modified subprogram is tested by recreating the RELAP5-3D executable to include it and then running a small set of test cases. After a small group of about 5 subprograms is converted, all normal test cases are run. Conversion is deemed successful only when output from the modified code is identical to the output of the unconverted code for all test cases.

At the conclusion of the restructuring task, the entire code was compared to the non-restructured code. The reengineered code produced exactly the same output, to the last character printed, as the original for all test cases.

5.0 RESULTS

There are 554 source files in the relap directory. Of these, there were 60 files that needed no restructuring because they were already written with the structured programming paradigm. 447 files comprising some 80,000 lines of FORTRAN code were restructured. The remaining files have not yet been converted.

One important result is that all the restructured files have also been reformatted with a consistent indentation and spacing rules. Many or the format statements that have text strings out to column 72 and wrap around to column 7 have been rewritten; they now break at the end of each line with an ending quote mark. The indentation rules now apply to format statements.

One measure of improvement would be a reduction in code run time; however, there was no measurable change.

The restructured files showed significant reduction in the number of logic jumps they contain. This is measured by the reduction in number of GOTO statements and line labels. The average number of GOTO statements per subroutine dropped from 8.8 before restructuring to 5.3 afterwards, a reduction of 40%. The maximum number of GOTO statements in any subroutine dropped from 213 to 99, a factor of 2.1. Finally, the maximum number of statement labels dropped from 210 to 43, a factor of nearly 5. This is summarized in Table 2.

Measure	Before	After	Ratio
Ave GOTO	8.8	5.3	1.66
Max GOTO	213	99	2.15
Ave Labels	22.0	10.2	2.16
Max Labels	210	43	4.88

Table 2 Measurements of restructuring improvement

While some blocks of code remain unstructured, a much greater fraction of the code is now structured. These measurements indicate a significant reduction in the degree of interwoven logic paths and corresponding increase in the degree of readability of the code.

REFERENCES

1. COBALT BLUE, INC., “FOR_STRUCT, Your FORTRAN Structuring Solution,” 11585 Jones Bridge Rd, Suite #420-306, Alpharetta, GA, 30005, USA, (1997).

2. RELAP5-3D CODE DEVELOPMENT TEAM, “RELAP5-3D Code Manual,” INEEL-EXT-98-00834, Revision 2.3, Idaho National Laboratory, Idaho Falls, ID, USA (2005).

3. F. L. Friedman, E. B. Elliot, Problem Solving and Structured Programming in FORTRAN, ISBN 0-201-01967-1 BCDEFGHIJ-MA-7987, Library of Congress Catalog Number 76-45154, Addison-Wesley Publishing Company Inc., 1977.

4. D. A. Fisher, "DoD's common programming language effort," *IEEE Computer*, volume 11, number 3, pages 24-33, March 1978.

5. E. W. Dijkstra, “Go To Statement Considered Harmful,” Communications of the Association for Computing Machinery, Inc, Vol. 11, No. 3, pp. 147-148, March 1968.

6. “The Definition of Structured Programming,” General Services Administration, Federal Standard 1037C, Telecom Glossary, 2000.

7. PRODUCT DISCLAIMER