SOFTWARE—PRACTICE AND EXPERIENCE, VOL. 7, 625–630 (1977)

RATSNO—An Experiment in Software Adaptability

DAVID R. HANSON
Department of Computer Science, Yale University
10 Hillhouse Avenue, New Haven, Conn. 06520, U.S.A.

SUMMARY

Adaptable programs are one of the benefits of structured programming. The adaptability of a program is the degree to which it can be transformed into another program that performs a similar, but slightly different, function. While it is clear that the aims of structured programming are best satisfied by the use of modern programming languages, a great number of programmers must use languages such as Fortran. To alleviate this situation, a number of preprocessors have been introduced that give Fortran a more structured facade. This paper describes an experiment performed to test the adaptability of programs written in RATFOR, one of these preprocessors. Judging from the results, the use of a good preprocessor can significantly increase the adaptability of Fortran programs.

KEY WORDS Preprocessor Software adaptability Software engineering Fortran RATFOR SNOBOL4

INTRODUCTION

The increasing demand for reliable software has been a major motivation behind the great amount of effort expended in the development of structured programming. It is widely accepted that the use of structured programming and its associated programming methodologies can lead to programs that are more reliable, easier to understand and easier to maintain.[1] Programming languages have played a central role in the development of structured programming and are the principal tool with which to construct reliable software.

The structure of a programming language has an enormous influence on the thought processes used in programming and hence on the structure of the programs written in that language. For this reason, numerous programming languages have emerged in recent years, most of which contain features designed to facilitate structured programming. Inspired by the success of Pascal[2] and Simula 67,[3] many of these emerging languages provide features for the definition of abstract datatypes, structured control flow, information hiding,[4] modularization, program verification and top-down design. In short, research in structured programming has resulted in programming languages with better abstraction mechanisms for dealing with the complexity that is inherent in the programming process.

Besides resulting in more reliable software, these languages also result in programs with a great deal more *adaptability*. The adaptability of a program is the degree to which it can be transformed into another program that performs a similar, but slightly different, function. Judging from the tendency of programmers to re-invent programs for each seemingly new application, the adaptability of programs written in languages such as Fortran is very small. The only way to make substantial headway in any engineering field is to build on previous work. Thus, adaptability is an important aspect of programming languages and programming methodologies.

Received 5 April 1977

© 1977 by John Wiley & Sons, Ltd.

Realistically, however, most practising programmers do not have, or are not permitted, access to the latest programming languages. Indeed, the very nature of most programming tasks demands the use of an established language, which typically includes a run-time system and a production-quality compiler. For the majority of programmers, Fortran is the only choice.

Faced with this situation, numerous Fortran preprocessors, which supposedly permit the programmer to work in terms of 'structured' Fortran, have been implemented. This approach is quite popular; a recent compilation lists over 50 such preprocessors.[5] While a preprocessor cannot provide the advanced data-structuring facilities of the newer languages, it can ameliorate the unpleasant control structures and improve on the cosmetic deficiencies of Fortran.

Although it appears that the use of a preprocessor is a definite advantage in the development of *new* programs, it remains to be seen if the adaptability of these programs is improved. That is, does the use of a preprocessor increase the adaptability of Fortran programs? A negative response to this question would suggest that preprocessors are of limited utility and are little more than a stopgap measure until the use of the newer languages becomes more widespread.

The remainder of this paper describes a simple experiment designed to test the adaptability of a program written in RATFOR,[6] one of the more recent Fortran preprocessors. RATFOR was chosen because it appears to be one of the better of the existing preprocessors, is well documented, has a growing user community and has been used for numerous published programs.[7] More importantly, the RATFOR preprocessor is itself a medium-sized (1,500 lines) RATFOR program, providing an existing program for the experiment.

THE EXPERIMENT

One of the design principles used in RATFOR was that 'RATFOR does not know any Fortran'.[6] As such, it should be possible to modify the RATFOR preprocessor to obtain a preprocessor for another language, thereby testing the adaptability of RATFOR. A preprocessor for another language meets the criteria of a transformation to a program having a similar, but slightly different, function. The experiment consisted of transforming RATFOR from a preprocessor for Fortran into RATSNO, a preprocessor for SNOBOL4.[8]

SNOBOL4 was chosen because it is sufficiently different from Fortran to require more than trivial modifications. In addition, the result may prove useful in its own right considering that SNOBOL4 has been criticized for its lack of decent control structures.[9] The idea of a preprocessor for SNOBOL4 is not new, of course; other SNOBOL4 preprocessors exist.[10] A summary of RATSNO syntax and a sample program are given in the Appendix.

The transformation from RATFOR to RATSNO was performed in a step-by-step fashion. That is, instead of attempting the transformation by one massive modification, RATSNO emerged gradually as a series of small successive transformations were made. Each transformation contributed to a small part of the total conversion. This methodology was used in order to discover any interconnection or data dependency problems that might exist between subprograms, since these kinds of problems can have a substantial negative impact on program adaptability. The amount of time required for each step was recorded, along with an indication of the RATFOR subprograms requiring modification and the complexity of the modification.

For the most part, the transformation proceeded smoothly without any one step taking a disproportionate amount of time. The complete transformation required just under 8 man-hours. Out of the 56 subprograms comprising the RATFOR preprocessor, 5 required

only trivial modifications, 16 required substantial but straightforward modifications, and 3 were deleted. In addition, 4 new subprograms were added. None of the changes required extensive changes to the algorithms used in RATFOR, and the performance (size and speed) of RATSNO is comparable to that of RATFOR.

PROBLEMS

Examination of the output from RATFOR indicated that the modifications necessary to produce RATSNO would be concentrated in the areas of lexical analysis, code generation and the addition of several new statements. While code generation might seem to be the most likely source of adaptability problems, it turned out that lexical analysis generated the most extensive modifications.

RATFOR 'knew' enough Fortran to ignore blanks during lexical analysis. In SNOBOL4, however, blanks are used to separate binary operators and cannot be ignored. This seemingly simple problem required the greatest amount of time and was responsible for the majority of the modifications. Many of the modifications were trivial, but nonetheless necessary.

Another problem concerned the use of the RATFOR *define* statement, which permits the definition of symbolic parameters. Unfortunately, DEFINE is the built-in SNOBOL4 function that defines programmer-defined functions. This problem in itself is not major; a simple solution is to change the RATFOR reserved word *define* to something else. But the use of common files containing widely used definitions is one of the advantages of having a *define* facility in similar preprocessors for dissimilar languages. For this reason, it was decided to retain the *define* facility in its RATFOR form. Rather than introducing another statement into RATSNO for defining functions, the OPSYN facility of SNOBOL4 was used to give the built-in function DEFINE another name. Thus, the first line of output from RATSNO is

```
OPSYN ("DEF", "DEFINE", 2)
```

and RATSNO functions are defined using DEF.

While much more elaborate schemes could be (and were) devised, it seemed best to choose a simple solution to what really was a simple, albeit thorny, problem. Because RATFOR is highly adaptable, more elegant solutions can be implemented once RATSNO has proved its utility.

CONCLUSIONS

Judging from the results of this experiment, the use of a preprocessor such as RATFOR can significantly increase the adaptability of Fortran programs. The most important aspect of RATFOR programs contributing to their adaptability is that they can be *read*. The structure of the program can be understood from the program itself. This is demonstrated by the paucity of comments in the RATFOR preprocessor; in over 1,500 lines there are only a handful of comments in addition to the one-line explanation that precedes each subprogram. Nevertheless, the structure of the program can be understood by reading the code.

The heavy use of subprograms also contributed to the adaptability of the preprocessor. While it is difficult to ascertain the effect of RATFOR on modularity, it does appear that the presence of modern control structures encourages the use of subroutines and functions. The important point is that when modern control structures are provided, programmers tend to *think* differently about programming; they tend to think in terms of modularity and separation of function.

The RATFOR statements *define* and *include* proved to be invaluable aids to adaptability. The *define* statement permits the definition of symbolic parameters, and the *include* statement permits the inclusion of a file in place of the *include* statement (this is similar to %INCLUDE in PL/I). The use of these statements greatly contributed to the readability of the program. The *include* statement is a simple, yet effective, means of accomplishing information hiding, and was used to ensure the uniformity of common blocks. Since a preprocessor consisting of only these two statements is quite easy to implement,[7] there is no reason for not including them as a part of any preprocessor or programming language.

On the negative side, RATFOR does not ameliorate the data-structuring facilities of Fortran. It can, however, make data handling significantly simpler and more understandable. By adhering to simple programming conventions, such as the use of functions to access complex data structures, reasonably adaptable programs can be written in RATFOR.

Good Fortran preprocessors, such as RATFOR, are not substitutes for better languages, when better languages are available. In the absence of such languages, however, good preprocessors are substantial improvements over Fortran. Such preprocessors are more than a stopgap measure, and can be considered as practical tools for the construction of reliable software.

ACKNOWLEDGEMENT

The version of RATFOR used for this experiment was provided by Brian W. Kernighan.

APPENDIX

The syntax of RATSNO is essentially that of RATFOR. The syntax can be specified as follows.

```
prog : stat
     | prog stat
stat : if (stmt) stat
     | if (stmt) stat else stat
     | while (stmt) stat
     | for (stmt; stmt; stmt) stat
     | repeat stat
     | repeat stat until (stmt)
     | return
     | freturn
     | nreturn
     | stop
     | break
     | next
     | label: stat
     | { prog }
     | anything unrecognizable
stmt : any SNOBOL4 statement sans gotos
```

The *freturn*, *nreturn* and *stop* statements were added for RATSNO. Labels are indicated by an identifier followed by a colon.

One difference between RATSNO and RATFOR is that the condition in the *while*, *if*, *repeat* and *for* statements is an arbitrary SNOBOL4 statement rather than an expression. This is indicated by stmt in the above grammar. The success or failure of the condition is used to control program flow. For example, a RATSNO statement of the form

```
for (S1; S2; S3)
  S4
```

results in the SNOBOL4 code

```
      S1
L1    S2   :F (L3)
      S4
L2    S3   : (L1)
L3
```

The following program, which is a simple program for producing a cross-reference list, illustrates the use of RATSNO.

```
# xref - cross reference program
    define MAXWORD 15   # maximum length of a word
    define NSIZE 3      # length of entry in line number list
    uc = "ABCDEFGHIJKLMNOPQRSTUVWXYZ"
    lc = "abcdefghijklmnopqrstuvwxyz"
    digits = "0123456789"
    word = BREAK (lc) SPAN (lc digits) . w
    t = TABLE ()
    DEF ("add (s, n)")
    dejavu = RTAB (NSIZE) (SPAN(" ") | null) *lineno RPOS (0)
    for (lineno = 1; line = REPLACE (INPUT, uc, lc); lineno = lineno + 1)
      while (line word =) {   # break out words
        w = SUBSTR (w, 1, MAXWORD)        # truncate long words
        maxw = GT (SIZE (w), maxw) SIZE (w)
        t[w] = add (t [w], lineno)        # avoid duplicate references
        }
    if (t = SORT (t)) {                   # sort table and convert to array
      maxw = maxw + 2
      for (i = 1; OUTPUT = RPAD (t<i, 1>, maxw) t<i, 2>; i = i + 1)
        ;
      }
    stop
# add (s, n) - add line number n to s only if not already there.
add:  if (s dejavu)
        freturn
      add = s LPAD (n, NSIZE)   # new entry
      return
    end
```

REFERENCES

1. F. P. Brooks, Jr., *The Mythical Man-Month*, Addison–Wesley, Reading, Mass., 1975.
2. K. Jensen and N. Wirth, *PASCAL—User Manual and Report*, Springer–Verlag, New York, 1975.
3. G. M. Birthwhistle, O. J. Dahl, B. Myhrhaug and K. Nygaard, SIMULA BEGIN, Student Literatur, Auerbach, 1973.

4. D. L. Parnas, 'Information distribution aspects of design methodology', *IFIPS Congress 71*, 26–30 (1971).
5. *For-word: Fortran Development Newsletter*, August 1975.
6. B. W. Kernighan, 'RATFOR—a preprocessor for a rational Fortran', *Software—Practice and Experience*, **5**, 395–406 (1975).
7. B. W. Kernighan and P. L. Plauger, *Software Tools*, Addison–Wesley, Reading, Mass., 1976.
8. R. E. Griswold, J. F. Poage and I. P. Polonsky, *The SNOBOL4 Programming Language*, 2nd ed., Prentice–Hall Inc., Englewood Cliffs, N.J., 1971.
9. R. E. Griswold, 'Suggested revisions and additions to the syntax and control mechanisms of SNOBOL4', *SIGPLAN Notices*, **9**, 7–23 (1974).
10. D. L. Croff, 'SNOFLEX Handbook', *Technical Report*, Department of Computer Science, University of Oregon, Eugene, Oregon, 1974.