PART-COMPILATION IN HIGH-LEVEL LANGUAGES

JACOB PALME

Abstract.

Many programming languages include the ability to divide large programs into smaller segments, which are compiled separately. When a small modification is made to a large program, then the affected segment only has to be re-compiled.

This paper discusses how high-level languages like Algol 68, Algol W or Simula 67 can incorporate part-compilation in a usable, secure and efficient way.

Key words and phrases: Part-compilation, Separate compilation, Programming language, High-level language, Algol 68, Algol W, Simula 67, Compiler, External procedures.


0. Introduction.

One of the main reasons for the popularity of Fortran is probably the concept of part-compilation. When a change is made in a large Fortran program, then only those subroutines must be re-compiled which are affected by the change.

The advantage with this is that the computer cost for compiling is lowered. This also means that the turn-around time is lowered during program development and debugging. Some of the reduced compilation cost can be used to make the compiler more optimizing. The extra cost for a more optimizing compiler is not so frightening if you usually only have to re-compile a small part of your program.

On the other hand, part-compilation will raise the cost of re-compiling the whole program, since:

i) Some effort must be spent once for each separately compiled segment, which had to be done only once for a non-segmented program.

ii) The mechanism for part-compilation will increase the complexity of the system.

iii) If the program is not segmented into part-compilable parts, then sometimes a simpler, faster loader or no loader at all need be used.

This means that the value of part-compilation depends on the job mix. With many small student jobs, part-compilation may increase the

Received May 26, 1972.
cost substantially compared to fast student job compilers which produce directly executable programs. However, such student job compilers do not permit assembler language procedures.

In a job mix with many large, complex programs, part-compilation on the other hand will give a substantial decrease in cost. If a conventional loader is used both with and without part-compilation, then part-compilation will surely lower the cost of recompiling a segment in a 1000-card program to less than a fourth.

If the conventional loader can be avoided without part-compilation, then (based on IBM 360 Fortran G data) part-compilation will still lower the cost for recompiling a segment in a 1000-card program to less than half, and for a segment in a 2000-card program to less than a third of the cost without part-compilation.

If high-level languages like Algol 68, Algol W or Simula 67 are to compete with Fortran not only for simple student jobs, but also for more complex programs, then they ought to have a system for part-compilation which is equally good or better than the Fortran system.

The basic requirements on a good system for part-compilation are:

i) Program structuring.

ii) Data structuring.

iii) Security, by which I mean the ability for the programming system to discover as many programming errors as possible as early as possible.

iv) Efficiency, since reduction of computer cost was the reason to introduce part-compilation in the first place.

I will discuss these requirements one at a time.

1. Program structuring.

Program structuring is accomplished in Fortran by dividing the program into separately compiled subroutines. This is a good and simple way of program structuring. However, Fortran usually demands that each subroutine is a separately compiled unit. It is valuable to have small, local procedures which are not separately compiled.

Calls to such procedures can be made more efficient by an optimizing compiler. Also, local procedures have easy access to global data inside the separately compiled program unit. Finally, \texttt{go to}-statements are possible out of local procedures but not out of separately compiled procedures.

Many Fortran systems provide the possibility of several entry points to separately compiled subroutines. The simplest way to introduce this