SESSION 12: Invited Papers

WPM 12.2: History of Writing Compilers

D. E. Knuth

California Institute of Technology

Pasadena, Calif.

THIS PAPER will discuss the evolution of techniques used in writing compilers, from *IT* and *FORTRAN* to the present day. Five years ago it was very difficult to explain the internal mechanisms of a compiler, since the various phases of translation were jumbled together into a huge, sprawling algorithm. The passing of time has shown how to distinguish the various components of this process, revealing a basic simplicity.

Decomposing Formulas

The first (and in some ways the most interesting) problem that faces the author of a compiler is that of taking an arithmetic formula and finding a way to evaluate it step by step. The first solution, used in the *IT* compiler, was based on the concept of parenthesis levels. (The parenthesis level at any point in a formula is the number of parenthesis pairs whose scope contains that point.) In *IT* language a maximum depth of ten nested parentheses was allowed, and then the formulas were broken into ten levels. In this compiler no precedence of operators was recognized; i.e., A*B+C was the same as A*(B+C). Therefore a simple algorithm something like the following could be used:
(If "op" is a binary operator, then "anti-op" is such that A op B = B anti-op A. Also, LOAD and NO-OP are anti-ops of each other in this algorithm.)

K:= 0; OP(0) := NO-OP; scan: S := next character right-to-left; if S = operand then produce code "anti-op (OP(K)) S"
  else if S = binary operator then OP(K) := S
  else if S = '(' then begin K := K—1; produce code "OP(K) TEMP(K)" end
  else if S = ')' then begin if OP(K) ≠ NO-OP then produce code "STORE TEMP(K)"; K := K+1; OP(K) := NO-OP;
go to scan;

It is a straightforward matter to verify that the preceding algorithm will produce, from the expression ((Y*Z) + (W*V))+X the code

```
LOAD X
STORE TEMP (0)
LOAD V
MULT W
STORE TEMP (1)
LOAD Z
MULT Y
ADD TEMP (1)
ADD TEMP (0).
```

The OP table is the ancestor of the now famous *push-down list* or *stack*.

Researchers then sought for a way to treat the precedence of operators correctly; the IT scheme didn't adapt readily to this purpose. The *FORTRAN* group had the ingenious idea of replacing + by)))+(((, replacing * by))*((, and replacing ** by)**(. Believe it or not, this leads to a properly parenthesized expression, if an additional three pairs of parentheses are placed around the formula. For example, the foregoing expression would be changed to:

(((((Y))*((Z))))+((((W))*((V)))))+(((X))).

A parenthesis-level method can now be used to translate the resulting formula.

Closer inspection revealed that the parentheses need not actually be inserted at all, and this led eventually to a new way of looking at the translation process. The technique is to start reading in the statement, but hold off compilation until we get to something we know can be computed regardless of what follows. Then output the code for that and continue reading across the statement. This may be thought of as an *inside-out* method compared to the previous *outside-in* approach. There are three principal significant ideas involved in the final, polished form of the newer algorithm, namely:

(1)—Merely testing the priority of each operator with that of the preceding is sufficient to decide in what order to compute the statement.

(2)—Left and right parentheses can actually be treated as operators with priorities.

(3)—The information saved may be kept in a push-down list, since the only information required at any time is the *youngest* item in the list.

This algorithm, using an informal form of *Wegstein* and *Youden's* String *ALGOL* language, is:

string INPUT(infinity), STACK(infinity), X(1), Y(1), Z(1), S(1); I := 1; S := '@';
test: if S = binary operator, semicolon, or right parenthesis and if priority(Y) ≧ priority(S) then go to generator (Y)
else shift STACK & X & Y & Z & S left 1;
scan: shift S & INPUT left 1; go to test;
generator for binary operator: emit code for T(1) := X Y Z; shift STACK&X&Y right 2; result: Z := 'T(I)'; I := I + 1; go to test;
generator for unary operator: emit code for T(I) := Y Z; shift STACK&X&Y right 1; go to result;
generator for (: shift STACK&X&Y right 1; go to scan;
generator for ':=' : emit code for X := Z; shift STACK &X&Y right 2; go to test;
generator for @: we are done with the statement.

While the algorithm given earlier handled only arithmetic expressions with binary operators and parentheses, this handles priority of operators, unary operators and multiple assignment statements as well. The priorities are zero for @; := (and), one for + and —, two for * and /, three for exponentiation, and four for unary operators like ABS, SIN, SQRT, etc. The name of an array which is singly subscripted can be thought of as a unary operator.

Object Code Efficiency

The foregoing *IT* translation scheme produced the instructions LOAD X, STORE TEMP (0) which were quite unnecessary. An early attempt to prevent these was used in RUNCIBLE, where the unnecessary instructions were compiled as usual, but later when they were found to be unnecessary they were scratched out. The *modern scanner* described does not produce the unnecessary instructions at all. The expression A+B*C is treated just as if it were B*C+A. There is yet a further refinement possible, however. If we have a machine with more than one arithmetic register, we would like to go a step further and treat A*B+(C*D+E*F) as (C*D+E*F)+A*B, because the latter version requires no reference to temp storage. The appropriate generalization for a machine with n registers is: Read in code until an expression occurs which requires exactly n registers for the computation, then generate the code for it. When n equals 1, this is precisely the above algorithm, but when n is greater

[Continued on page 126]

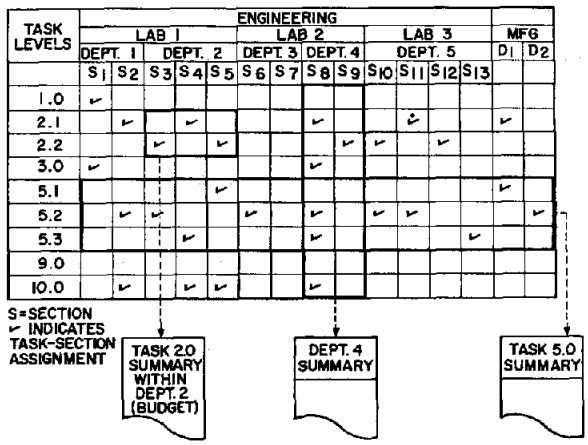
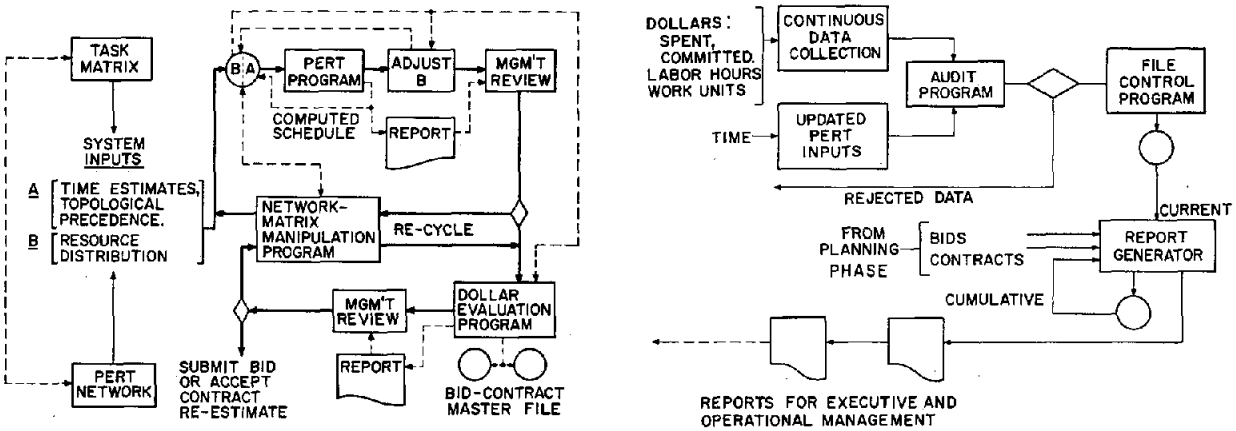


Figure 3 — A task matrix. A mapping of the contract identifying both organizational interface and functional responsibility through task-section assignments.



Figures 4*a* (left) and *b* (right)—The *ICON* system *(planning phase)* is illustrated at left. The scheduling loop is generally performed prior to dollar evaluation. Recycling through both loops will produce sets of alternatives expressed in dollars and time. Diagram at right shows the *report phase.* Actual costs incurred through contract operation are collected and processed against task progress and the budgetary information established during the planning phase.

WPM 12.2: History of Writing Compilers

[Continued from page 43]

a more elaborate scheme which needs more than a simple push-down list is required. There is even a generalization for n = 0, and on a zero-register machine the *IT* scheme is as good as any.

Many other improvements in object code efficiency are easily incorporated into the above algorithm, such as carrying a sign and absolute value tag with all operands, including temp storages.

Organization

The real differences between compilers is the way the basic components are organized into the program. The components themselves are essentially equivalent. The chief types of organization have been:

(1)—Symbol pair, used in *IT* and early compilers, in which each adjacent pair of symbols in a formula indicated what generator is to be used.

(2)—Operator pair, used in *NELIAC*, similar to the symbol-pair, but more economical since operands are treated more generally as a class by themselves.

(3)—Simple scan, similar to the algorithm given above, which generalizes operator-pair methods by putting operators into classes; they usually use several stacks instead of just one.

(4)—Recursive organization, where each generator can call on any other generator including itself, and delimiters are used to initiate and terminate control of each generator, as in the Carnegie Tech and the Burroughs B5000 *ALGOL* compilers.

(5)—Syntax-directed, analogous to recursively organized, except control is handled by one master generator which references syntax and semantics tables.

(6)—Various multiple pass compilers which have so complex an organization it cannot be summarized in less than several pages.