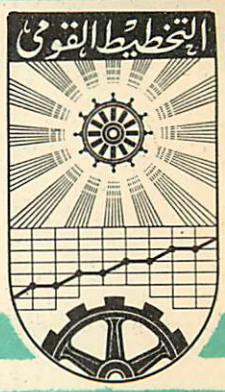
UNITED ARAB REPUBLIC

THE INSTITUTE OF NATIONAL PLANNING



Memo. No. 567

Advanced Compiler Techniques
(Lecture Notes)

Lionello A. Lombardi

May 1965

Operations Research Center

Table of Contents

Compiler Techniques

(Extensions from FORTRAN:
a) Mixed mode expressions
b) Variables with any number of dimensions)

1. Polish Notation (prefixes)

The operators (e.g., +, -, etc.) precede the operands.

Examples:

1) + 3. A instead of 3. + A
2) + 4. * SQRTF A 2. instead of 4. + SQRTF (A) * 2.

2. Order of operators number of operands.

Examples:

1) SQRTF has order 1.
2) + has order 2.

3. Parentheses in polish notation are necessary only if the order of operators varies.

4. A push down list (pd l) consists of :

a) A top pointer
b) an indication of its bottom
c) Same memory space

It can be organized as association List for the purpose of saving space when several pdl coexist.

(Clue : NEVER tie up index registers as top pointers of a pdl)

5. Translation FORTRAN into Polish needs one push down (pdl) list for infix operators, named infix pdl, which also contains closed parontheses.

6. Translation from Polish into machine code needs one pdl for operands, named operand pdl. It only contains addressess of operands and intermediate variables, along with indication of whether they are declared variables or constants or intermediate variables. The operands pdl can be augmented by associating to each entry the following information:

mode (e.g. floating, Boolean, etc.)

7. One pass formula translation

Both infix pdl and operand pdl coexist at compilation time. The Polish stage is skipped.

8. Treatment of intermediate variables (iv)

IV are the results of applying operators to operands within a formula.

Example:

In 4 * (3 + N), the number 3 + N is an intermediate variable.

They are actually only computed at execution time, but space for them is allocated at compilation time in the ghost pdl. The ghost pdl at compilation time simply consists of a top pointer (without space) and a register to remember its maximum length. It has space but no top pointer at excution time.

9. Organization of the compiled program (COMMON excluded)

(This is a very simple one, more efficient ones are desirable)

1) Transfer vector (list of names of subroutines and functions)
2) Excutable code followed by constants
3) Areas for declared variables and constants.
4) Space for ghost pdl.

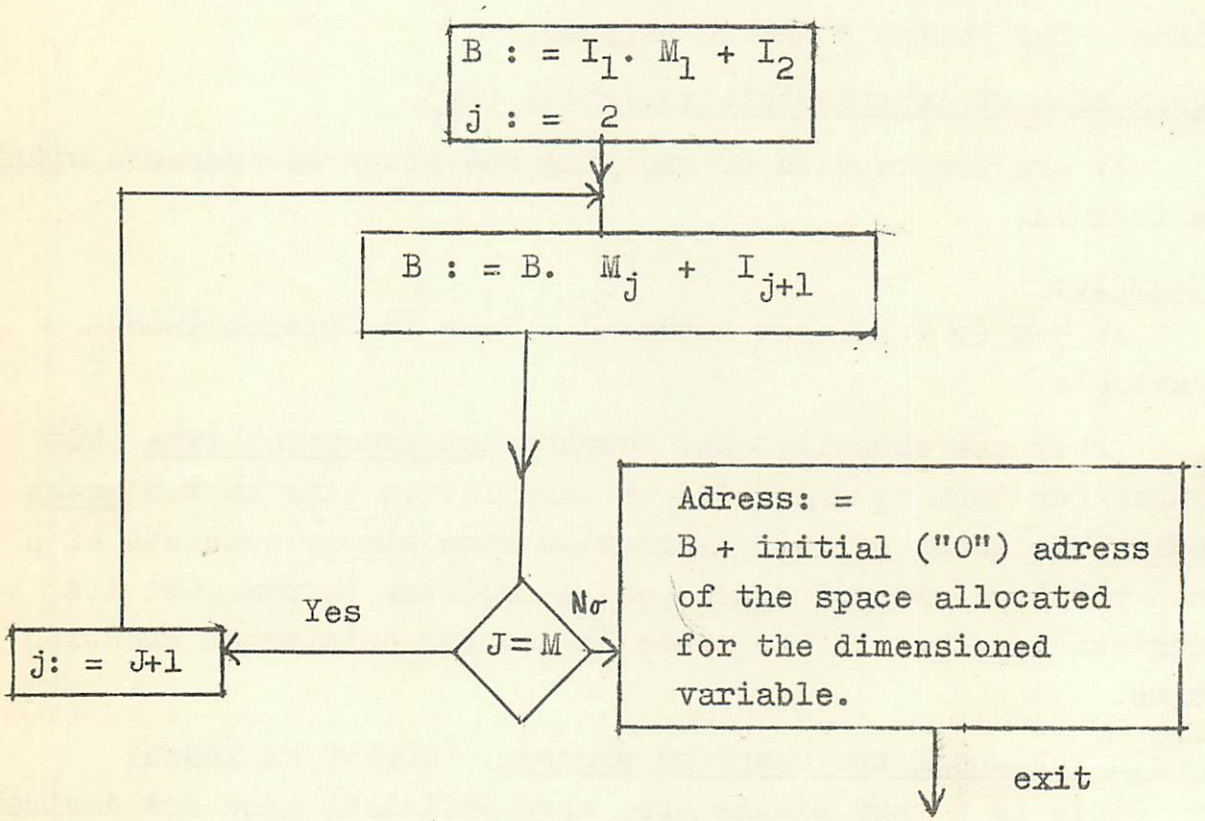
During compilation it is necessary to build the transfer vector and to keep track of the length of items 1,2,3 and 4 above. At the end of compilation or at leading time item 2 above need to be relocated.

10. Formula for dimensioned variables

The adress of A (I_1, I_2, ... , I_n),
where I_i is an integer formula, preceeded by DIMENSION

A (M_1 , M_2 , ... , M_n)

where M_j is an integer constant,
is computed at excution time by the following routine:



An Interactive, Algebraic Compiler

Introduction

A computer time-sharing system, such as that under development at MIT's Computation Center and at Project MAC provides one with the unusual ability, in this age of ultra high-speed and ultra high cost computer equipment, of enjoying "hands-on" operation of the machine. The programmer need not wait long periods of time to locate his program bugs; he may instead debug his program on'line, with direct supervision of the machine's operation.

Seated at a remote console, usually consisting of a typewriter and printer, the user may "interact" with the computer. In the MIT Compatible Time Sharing System (CTSS), for example, the user enters his system commands, such as INPUT, LOAD, START, EDIT, and compilation commands in response to observed action by the computer, as reflected by the printer portion of the console. Thus, a normal sequance of commands might be

INPUT	Enables the user to enter his source statements
FILE	Causes list of source statements to be retain on the disc file.
FAP	Causes the source program to be assembled in the FAP language
LOAD	Loads the assembled object code into the computer.
START	Causes execution of the previously loaded object code.

In each instance, the user enters a command after he receives a confirmation that his last command has been successfully executed. For example, if the FAP assembly, in the above illustration, was not successeful, the user will be able to learn this and take steps to correct his source program, rather than to load the hoped-for object code. The important point here is that the user does, in fact, interact with the computer. His actions are generally based upon information received from the computer, and the computer's actions are generally based upon instructions received

from the programmer. Both the user and the machine make decisions, each upon some action taken by the other.

If the user makes a mistake, for example, trying to load the object code of the FAP program that did not assemble, the computer will inform him of his error immediately. Similarly, if the user enters a command that is misspelled or that just does not exist, the computer will inform him of this immediately. The user is thus given an opportunity to take appropriate measures.

Let us go into a bit more detail about the means of entering a source program. In particular, let us suppose that a hypothetical user wishes to enter a program written in FORTRAN, the most widely used algebraic programming system. Under the present CTSS development, the computer user follows the following procedures. He enters an INPUT command, which causes the computer to type out a line number, starting with 00010. After the line number appears, the programmer enters his first source program statement. He enters a carriage return to denote the end of the line, and a new line number is given automatically by the computer. The line number progress in jumps of 10, thus, the second line number is 00020, the third is 00030, etc. The programmer continues to enter his source statements until his program has been completely entered. If he makes an error in any statement, he may use the regular CTSS error correcting procedures ("to delete the previous character, to delete the entire line) or he may reenter the entire statement by entering the line number followed by the corrected text of the statement. A new statement may be inserted by first entering a line number between the two statements that border the inserted one, such as 15 between lines 10 and 20, and then following this number the text of the new statement. A statement may be deleted by using the DELETE command. Finally, the corrected list of statements is given a symbolic name and is file. It is here that the computer is informed of the particular language in which the file has been written. Thus, the statement to file an input list named, say, TEST, would be

FILE TEST FORTRAN

This means "Store the previously created input file in a disc file named TEST in the FORTRAN language." To compile this program, the user enters the command

FORTRAN TEST

which tells the computer to compile the source program file called TEST in the FORTRAN mode using the FORTRAN compiler associated with the CTSS monitor. After the program has been compiled, the user must load it, and then start it, by means of the LOAD and START sequence, or the newer LOADGO command. If an error occurred during compilation, due to either a typographical error in the source program, or possibly some more substantive mistake, such as branching to a nonexistent statement number, the user must correct the source program, by using the EDIT command to reinstate the program in the INPUT mode, make his correction (s), file the program, and recompile it. Basically, he does not learn about errors until he commences the compilation process, or until he executes the compiled program. And he cannot correct these errors except by reinstating his source program in the input mode and then recompiling it from scratch.

Although the time sharing concept provides for the relatively short delay time in learning about and correcting program errors, much of the potential of the interactive features of the time sharing system are not present in the existing approach to program writing. It is for these reasons that work is being done towards the development of an algebraic programming system that will, in fact, be based upon its inclusion in the repertoire of a time sharing operation, such as CTSS. This system will be based, in general upon the existing FORTRAN language, since it has been found to be the most widely used algebraic language in existence today. Hopefully, the new system will be fully compatible with the existing FORTRAN systems, such that any program or subprogram written in languages such as FORTRAN II or FORTRAN IV may be included in systems using the new interactive language. A questionnaire was recently circulated to computer users in the MIT community for the purpose of determining their preferences and requirements for such

an interactive algebraic programming system. Detailed results of this study are given in a later section of this paper.

The interactive language will provide its users with the many features that will best enable them to take full advantage of the time sharing hardward at their disposal. For example, the entire structure of the input-output aspects of the FORTRAN system are being redesigned so as to provide for the requirements of the time sharing console. A description of this aspect of the work is also included in a later section of this paper.

But most important, the new system will give the programmer significant advantages in his ability to effectively take advantage of the time sharing system in the process of debugging. First of all, he will be able to make changes in the source program during execution and without total recompilation. He will be able to keep track of the progres of program's execution by means of some powerful tracing features which may be selectively controlled at the remote console. The programmer will be provided with some powerful interactive features, enabling him to enter a part of his program, in the form of regular FORTRAN source statements, during execution of the program. Finally, the programmer will be able to dump and reload his completed object code in a manner that will provide the most efficient operation of the program during its regular execution.

The programming system under discussion here is not of the interpretive form. Rather, it will generate actual machine code in response to each FORTRAN source statement, without the use of an intermediate language. (An interpretive system, incorporating some of the features described here, has been developed by the IBM Applied Programming group of the Data Systems Division in New York. However, because it is an interpretive system, it produces an object code that is rather slow, and hence cannot be used to compile programs intended for production runs.)

The purpose of this paper is to describe the various component parts of this system, with a view toward specifying the means by which it should be constructed. Thus, the paper includes sections

on the language specifications, the input-output operations, the construction of the compiler and the compiled executive routine, and finally the results of the aforementioned questionnaire.

Specifications of the Language

In general, the interactive system will include all of the language features now present in FORTRAN II, and probably will include those present in FORTRAN IV. The latter system includes the ability or using recursive subscripts [e.g., A(I(J(K))) ,] Boolean statements, [e.g., IF (A .LE. B .OR. (C+2*D) .GE. 50) GO TO 25], and the ability to use an expression anywhere in the program. In addition, FORTRAN IV provides for the legality of mixed mode expressions, and also permits the programmer to define labels as being either fixed or floating, so that he need not adhere to the I ... N rules in FORTRAN II. There will be some changes in the input-output statements, and there will be a more complete discussion of these in the next section.

In addition to the regular FORTRAN statements, the interactive system will provide a number of new features that aid in writing and debugging a program. We may call these pseudo instructions, since they direct the operation of the compiler, but do not have any direct affect upon the generated object program.

The Alteration Statements. These statements permit the programmer to change the nature of his program either before or after compilation.

DELETE s_1 + s_2 LINES	where s_1 is a statement number, and s_2 is the number of additional lines that define the statement to be deleted. If s_2 is 0, the "+ s_2 LINES" may be omitted. Causes deletion of the defined statement from the source program.
DELETE s_1 + s_2 LINES THROUGH s_3 + s_4 LINES	Causes deletion of a string of statements inclusive of the beginning and end statement as defined by s_1, s_2, s_3. and s_4.

INSERT $s_1 + s_2$ LINES	Causes the compiler to accept and sequentially compile into the overall object code a string of statements starting with the first to be located in the position defined by s_1 and s_2.
INSERT $s_1 + s_2$ IN $s_3 + s_4$	Causes the statement located at s_1+s_2 lines to be removed from that spot and be inserted at the point defined by s_3 and s_4. The statement previously at s_3+s_4 lines is pushed up one line.
ALTER $s_1 + s_2$ LINES AT c	where c is the column number of the card image where the alteration is to commence. Causes the compiler to accept changes in the statement number defined by s_1 and s_2 starting at position given by c.
ENDCHANGE	Causes the end of the insert mode.
TRACE s_1+s_2 LINES THROUGH $s_3 + s_4$ LINES	Results in a logical trace on all statements of the source program that lie between the two statements defined by the parameters of the TRACE statement. The tracing includes the statement and line number, and the value of the statement if an arithmetic expression, the value of the index of a DO if a DO statement, or the type of statement if some other kind is being executed.
TRACE ON $v_1 \ldots v_n$	where v_i is the symbolic name of a variable to be traced. Causes tracing of all statements where the variables v_i are affected or appear. In addition to the tracing information described above, the symbolic name of the variable is printed out for each statement traced.

Definition of an operation at object time. Present FORTRAN systems, which provide for an alphammeric FORMAT specification, also provide for a so-called "variable FORMAT statement," which is read into the program at object time. This is made possible in the present FORTRAN systems because the execution of input-output statements with FORMAT specifications is accomplished by means of an interpretive process. Thus, the FORMAT statement is never compiled; rather, it is stored in memory in basically the same form as it was written in the source program. Thus, a FORMAT specification may be quite readily read in at object time and used in the same manner as any other type of FORMAT statement. With a remote access time sharing system, it is possible for the programmer to thus enter a FORMAT statement at the console, followed by, say, input data, which will be read in according to this specification.

It was suggested, by Professor L. A. Lombardi and S. S. Alexander, that an analogous feature be extended to arithmetic expressions that may occur in virtually any source program statement. This feature would work as follows: The programmer will insert a special symbol in any particular position where he wishes to enter an expression at object time. This symbol will consist of a dollar sign followed by an alphanumeric label of from one to five characters in length. Thus, such a label might be $PRICE, or $X. During execution, when a statement containing such a label is encountered, the program comes to a pause, and the location (in the source program) and the name of the insertion variable are printed out at the console. The programmer then enters an expression, in the source program language (FORTRAN) that is to be substituted for the insertion symbol. The expression is compiled, and execution continues.

When the expression is entered, the programmer may attach, at the beginning, a special character, such as another dollar sign, to indicate that this inserted expression should be used throughout the current execution of the program. Without this special

symbol, the system will call for the insertion of a new expression each time the particular insertion symbol is encountered during execution.

The programmer may, if he wishes, include informational printouts of specific instructions governing the particular insertion. This would be done in the conventional manner, as will be more fully discussed in the section on input-output statements.

The insertion feature provides for great flexibility in the use of the remote console. The same program may thus be used to handle similar problems with varying parameters, decision rules, etc.

Finally, it should be noted that the insertion feature need not be limited to the insertion of expressions; it would be quite feasible to permit the insertion of one or more statements in basically the same manner. However, such inserted statements will, collectively, take the form of a single-valued FUNCTION subprogram.

Tracing. The interactive algebraic programming system will incorporate a number of powerful program tracing features to facilitate the debugging process. The trace is selective, i.e., it is defined and requested by the programmer, who also starts it and stops it as he pleases.

An example of some of the trace features to be incorporated in this new system may be found in the FORGO system developed at the University of Wisconsin for use with the IBM 1620. FORGO is a load-and-go version of FORTRAN II, with some language restrictions and, in the case of the 1620 FORTRAN II system, a few extentions. FORGO is an interpretive language, i.e., it translates the source language into an intermediate language, and translates this into machine code for the purposes of execution. FORGO was recently converted such that it may be used interactively on the 1620, making it an excellent demonstration of the type of system under discussion here.

Since FORGO, like the interactive system under development, maintains an image of each source program statement, logical tracing, by source statement, is made possible. FORGO includes a number of trace options. They include a complete trace on all statements between two given limits, a complete trace on all branches in the program, and other features that permit the programmer to limit the length of the program, if it appears to be excessively long, and to dump the object program.

The new system will incorporate these same features, but will, in addition, provide even greater tracing ability. Most important will be the ability to trace selectively on a single variable, or on a group of several given variables. The trace will contain the source program name of the variable, the location in the source program of the statement being traced, and the net result of the statement on the variable in question. This one features is, in particular, a very significant improvement in the repertoire of debugging aids.

In addition to the trace features for logical tracing at execution time, the interactive algebraic system will include some of this same ability for use by the programmer during compilation. In particular, the computer will inform the user, either voluntarily or on request, the means by which his program may get to a numbered statement as it is entered. Thus, when the programmer enters, say, statement 47, the computer will inform him how he might get to this statement from the various other parts of the program. This feature would be voluntary, i.e., it would be requested by the programmer, on the initial input of the source program; however, it would be mandatory, i.e., generated automatically, whenever the programmer inserts, alters, or deletes a numbered statement.