PHYSNET

MISN-0-370

COMPUTER PROGRAMMING TECHNIQUES: A REVIEW

Project PHYSNET Physics Bldg. Michigan State University East Lansing, MI

COMPUTER PROGRAMMING TECHNIQUES: A REVIEW

by

Robert Ehrlich, George Mason University

Title: **Computer Programming Techniques: A Review**

Author: R. Ehrlich, Physics Dept., George Mason Univ., Fairfax, VA

Version: 2/1/2000 Evaluation: Stage 0

Length: 1 hr; 16 pages

Input Skills:

1. Vocabulary: function (in Glossary).

Output Skills (Knowledge):

K1. Vocabulary: array, program, algorithm, flowchart, source program, machine language, compilation, floating point, double precision, triple precision, round-off error, word.

K2. List the steps usually followed in constructing a computer program.

K3. State the meaning of the five standard symbols used in constructing flowcharts.

Output Skills (Rule Application):

R1. Construct a flowchart for a simple set of operations to be programmed as computer code.

Output Skills (Problem Solving):

S1. Given an algorithm or a flowchart representing a series of operations, recognize and correct each of these common programming errors: use of an array index outside specified range, infinite loops, failure to bypass the unselected branch of a decision, division by zero, argument of a function outside the allowed range, data not read properly.

Post-Options:

1. "Review of Elementary FORTRAN" (MISN-0-346).

COMPUTER PROGRAMMING TECHNIQUES: A REVIEW

by

Robert Ehrlich, George Mason University

1. Introduction

1a. Definition of a Computer Program. A computer program is a coded list of instructions that "tell" a computer how to perform a set of calculations or operations. These instructions are fed into the computer from a magnetic disk, a keyboard or some other medium. Usually, instead of carrying out or executing each separate instruction as it comes in, the computer temporarily stores all the instructions in memory. At a later time it retrieves the instructions from its memory one at a time and executes them. All the operations that a computer performs for you are specified by a computer program, and there are a number of steps involved in going from the problem statement to the program.

1b. How To Construct a Computer Program. In order to create a computer program one normally follows these steps:

1. Define the Problem: State in the clearest possible terms the problem you wish to solve. It is impossible to write a computer program to solve a problem that has been ambiguously or imprecisely stated.

2. Devise an Algorithm: An algorithm is a step-by-step procedure for solving the problem. Each of the steps must be a simple operation which the computer is capable of doing. A universally-used representation of an algorithm is a "flowchart" or "flow diagram," in which boxes representing procedural steps are connected by arrows indicating the proper sequence of the steps. In many problems you will need to define a mathematical procedure, expressed in strictly numerical terms since the use of computers to do higher level analytic processes such as solving algebraic equations or doing integrals in a non-numerical fashion is relatively limited.

3. Code the Program: The steps in an algorithm, translated into a series of instructions to the computer, comprise the computer program. There are many languages in which computer programs can be coded, each with its own syntax, vocabulary, and special features.

One of the most widely used languages for scientific applications is FORTRAN. Other languages such as BASIC and PASCAL are also available on many computer systems.

4. Debug the Program: Most programs of any length don't work properly the first time they are run and must therefore be "debugged." Often, during the debugging phase, errors and ambiguities in the original statement of the problem reveal themselves, calling for basic revisions in the solution algorithm.

5. Run the Program: After the program has been fully debugged you run it, possibly using many sets of input data. This step may take anywhere from a few seconds to many hours depending on the complexity of the problem and the speed of the computer.

6. Analyze the Results: Often the output from a computer program requires considerable further analysis. In some cases, even though the program worked perfectly, you may find that you solved the "wrong" problem. There is an acronym well known to computer users: GIGO, which stands for "garbage in, garbage out."

1c. Computers and Problem-Solving. Clearly, the use of a computer does not in any way diminish our need for a thorough understanding of the nature of the problem we wish to solve or the underlying mathematical analysis. In fact, from the preceding discussion of the steps necessary to create a computer program, it should be clear that in order to write the program we must actually know how to solve the problem without a computer! That is, once we devise an algorithm, we could in principle perform each operation in sequence by hand and arrive at the solution. However, in practice many algorithms have so many steps that only a computer, which can do perhaps millions of arithmetic operations per second, can perform all the operations without error and in a reasonable amount of time.

1d. Practical Experience Is Necessary. In order to become proficient at writing programs and making them work, practical experience is very important for the beginner - even before understanding the finer points. Beginning computer programmers can make the mistake of wanting to understand everything about a computer language before attempting to write and run a program. That point of view can be an obstacle to achieving the better understanding obtained through experience.

2. Flowcharts

2a. Introduction. Flowcharts or flow diagrams are important tools in writing a computer program. A flowchart allows you to plan the sequence of steps in a program before writing it. The flowchart serves as a visual representation which many programmers find indispensable in planning any program of at least moderate complexity.

2b. Elements of a Flowchart. A flowchart consists of a set of boxes, the shapes of which indicate specific operations. The separate boxes are connected with arrows to show the sequences in which the various operations are performed. We use these standard symbols:

rectangle:	Any processing operation except a decision.
diamond:	A decision operation.
parallelogram:	An input or output operation.
oval:	The beginning or ending point of the program.
arrow:	The direction of flow from one operation to the next. Every operation box must have at least one incoming or outgoing arrow. Any arrow leaving a decision box must be labeled with the decision result which will cause that path to be followed.
small circle:	A connection between two points in a flowchart, where a connecting arrow would be too clumsy. A reference point or numbered statement.

2c. Sample Flowchart. An example of a flowchart is shown, in Fig. 1, for a program which uses the quadratic formula to compute the two roots of a quadratic equation. For a program as simple as this one, most experienced programmers would not draw a flowchart.

3. Running a Program on a Computer

3a. Interactive Processing. The basic computer system is interactive use from a terminal. With interactive processing the user enters input and receives output directly at a computer terminal. It is necessary to become familiar with a number of system commands which instruct the

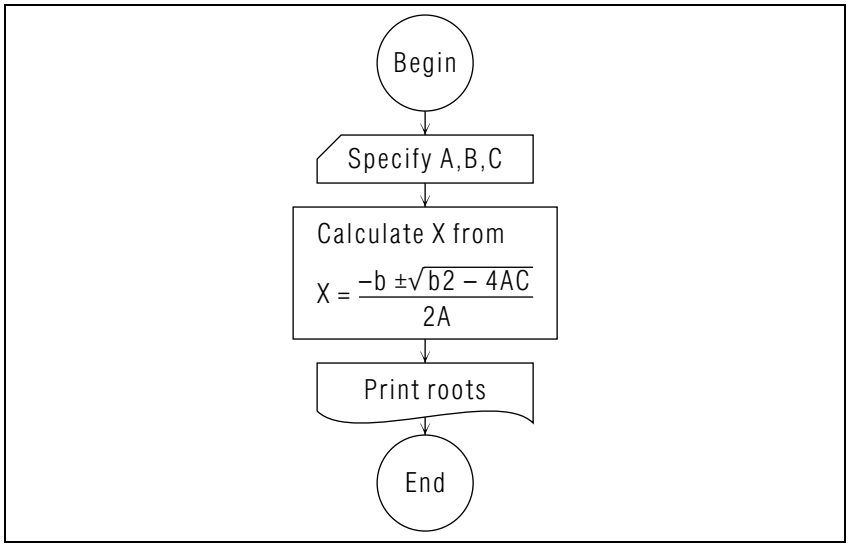


Figure 1. A flowchart for a program to compute the two roots of a quadratic equation.

computer about tasks the user wishes to have performed.

3b. System Commands. System commands, somewhat like the statements of a computer program, are instructions to the computer. However, they are not part of a programming language and they vary from one computer to another. Typical examples of system commands include instructions to load a specific program into memory, to start execution of the program, to print a file on a printer, and to sign-on or sign-off a terminal.

3c. Compiling or Interpreting a Program. Before the computer can execute a program written in a high-level language such as FORTRAN or BASIC, the program must first be "compiled" or "interpreted," a process wherein the computer converts the FORTRAN or BASIC statements of the program into a set of "machine language" instructions. Each program statement is thereby broken up into many elementary operations by means of a utility program called a "FORTRAN compiler" or a "BASIC interpreter." For example, the compiler would reduce the FORTRAN statement

$$\text{X} = 2.0*\text{A}**2/\text{B}$$

or the BASIC statement

$$\text{X} = 2.0*\text{A}\hat{}2/\text{B}$$

into a series of elementary operations: find the values of A and B, multiply A by itself, multiply the result by 2.0, divide that result by B, store that result in X.

3d. Compiler or Interpreter Errors. When the computer converts a program into machine language, it may uncover errors in some of the program statements and print out a message to this effect. If the computer converts a program without finding any errors, the machine language version of the program is loaded into memory (automatically on many computers), and the computer attempts to execute the program. It sometimes happens that due to some error in the program that was not detected during the conversion process, the computer is unable to execute a program. In this case, the computer may report an error message and abort the execution, or it may simply "hang" and not be able to proceed. Even if the computer does execute the program and produce results as expected, there is no guarantee that the results are correct. There are many types of errors that you may make in writing a program or entering it into the computer that will not result in any errors that the computer can detect.

4. Debugging a Program

4a. Introduction. Most programmers find that their programs do not work properly the first time they are run. In fact, it is not at all unusual for the debugging of a program to require considerably more time and effort than was required to write the program in the first place. Sometimes the final correct version of the program bears little resemblance to the original version! This is largely due to the many different kinds of errors that can be made in writing a program.

4b. Program Bugs May Be Subtle. While the computer can detect some types of errors in your program and alert you to them, there are other kinds of errors (the most difficult to find), which do not give any obvious indication of anything wrong - except that the answer is not correct. For this reason it is important to take a very skeptical attitude toward the results of a computer calculation, checking it whenever possible by a hand calculation. In cases where this is impractical, check for:

1. Internal consistency - Often calculations can be done more than one way. Do the answers obtained using different methods agree?

2. Reasonableness - Often by making simplifying assumptions, an estimate to the answer can be found. How does the computed result compare with the estimate?

3. Limiting cases - When a calculation is done involving parameters that must be assigned numerical values, sometimes the result is obvious for certain numerical values of certain parameters. Are the computed results for these limiting cases correct?

4c. Trace Your Program's Flow. It cannot be overemphasized that you should not grant automatic validity to the results of a computer calculation - always check them in some way or other. If you suspect that an error is present in a program, the general approach in trying to locate the error is to go through the program step by step to see how the computer obtained its results. The idea is to assume nothing, but simply follow each instruction mechanically. In some cases the error may be due to something trivial, like a typo (perhaps you accidentally typed the letter "O" when you intended to type the number "0"), or possibly you inadvertently left out a line.

5. Common Programming Errors

5a. Use of an Array Index Outside the Specified Range. Whenever an integer variable is evaluated from some mathematical expression and then is used as an array index, we must be very careful that the form of the expression never causes the index to go outside its permitted range. This very common mistake can occur in a number of ways, one of which is illustrated here:

$$J = K^2 \sin(Y/Z), \text{ and } F(J) = 1.0$$

In the present example, we are certainly in trouble if Y and Z assume values that make $\sin(Y/Z)$ negative or zero.

5b. Infinite Loops. An infinitely repeating loop will occur any time the exit from a loop depends on some condition which is never satisfied. An example of an infinite loop is illustrated in the program in Fig. 2. This program is supposed to calculate and print a table of values of the function e^x for $x = 0, 1, 2, ..., 10$. When the variable X is incremented

```
      X=0.
10    E=EXP(X)
      WRITE(3,30)X,E
      X=X+1.0
      IF(X-10.0)10,20,10
20    CALL EXIT
30    FORMAT(2F10.5)
      END
```

Figure 2. An example of an infinite loop.

by 1.0 each time through the loop, the result is only accurate to some specific number of significant figures. Thus, on the tenth time through the loop the binary-computed value of X might be 9.9999999351... instead of exactly 10.0, so that the decision operation never causes the program to branch out of the loop to the end of the program. One solution is to test whether the quantity

$$\text{ABS(X}-10.0)-0.0001$$

is positive, negative or zero. The program would branch out of the loop when X differs from 10.0 by no more than 0.0001.

5c. Failure to Bypass the Unselected Branch. As an example of this error let us assume that we wish to calculate the value of C using a different formula depending on the value of the variable A. If the value of A is negative then the program branches one way and calculates C according to the intended formula. If the value of A is positive or zero the program branches another way and calculates C according to a different formula. However, the program then immediately recomputes C according to the wrong formula which we really intended to bypass for non-negative values of C.

5d. Division by Zero. If this mathematically improper operation is attempted in a program, the result depends on the particular computer: some computers cause an error message to be printed but many do not. If no error message is printed, the computer may take the value to be

```
      IF(A)1,2,2
1     C=D+2
2     C=D+2+E
3     CONTINUE
```

Figure 3. An example of failure to bypass the unselected branch in a decision.

some arbitrary number (possibly zero, one, or the largest number that can be defined on the particular computer) in order not to terminate the program. One way to avoid a division by zero is to test the divisor before performing the operation. If the divisor is found to be zero you may wish to either stop the program, or alternatively print out a message to that effect and simply bypass that particular step. An alternative procedure is to modify the divisor in some way so that it is never zero.

5e. Argument of a Function Outside the Allowed Range. The most obvious example of this problem is an attempt to use the square root function with a negative argument. Other functions, such as the exponential, also have limitations on the range of values the argument may assume. (In the case of the exponential function, the range is dictated by the largest (or smallest) number that can be represented on the particular computer). As in the case of a division by zero, many computers do not print an error message if the argument of a function is outside the allowed range.

5f. Data Not Read Properly. Given the complexities of the rules concerning input statements, it sometimes happens that the format of numbers on a data card or line does not exactly agree with that specified in the relevant input statement. In such a case, the computer may read in incorrect values for certain quantities. The best way to guard against this possibility is to use an "echo check" in a program, which means: print out the input quantities right after they are read in, before any calculations are done using them.

6. Precision of Computations

6a. Finite Word Size Limits Precision. Numbers are represented in the computer by binary "bits" grouped into "words" (a fixed number of bits). Due to its finite word size, the computer cannot perform "floating point" (non-integer) arithmetic operations and obtain exact results. For example, if the computer has a word size equivalent to eight significant figures, then the value computed for the quotient 7.0/2.0 is only correct to eight figures. Note that even though the exact answer 3.50000... has all zeros after the second figure, an inaccuracy arises, since, on most computers, numbers are represented internally using the binary rather than the decimal system.

6b. Internal Round-Off Error. Most decimal numbers (base 10) suffer a loss of precision ("round-off error") when the computer converts

them to binary numbers (base 2) even before the numbers are used in arithmetic operations. In most computations, the loss of precision means that the final result may be incorrect after the seventh or eighth figure, which may be considered only a minor annoyance. Random accumulation of round-off error due to a large number of arithmetic operations can cause the final result to be inaccurate by a significant amount. Even a single arithmetic operation can cause a large loss of precision, if, for example, two numbers of nearly equal size are subtracted from one another.

6c. Double and Triple Precision Calculations. On most computers it is possible to specify that the number of significant digits for each arithmetic operation be increased from its normal value. To accomplish this, the computer stores each quantity in two or possibly three words of memory, permitting it to double or triple the number of significant digits ("double precision" or "triple precision"). If you suspect that the results of a program are inaccurate due to round-off error, then it is advisable to run the program again using double or triple precision.[1] If you find that the numerical results differ substantially between the original single precision run and the later double precision one, then in fact round-off error is present. While the double precision result is generally more trustworthy than the single precision one, it is possible for it to also be in error if the two results are substantially different.

6d. Round-Off In Program Output. The number of digits in a printed result is determined by the appropriate FORMAT statement in FORTRAN, or PRINT USING statement in BASIC, and this can also have "round off." On a computer having a word size equivalent to eight digits, the single precision result of the quotient 7.0/2.0, printed out to four, eight, and twelve digits might be, respectively,

3.500 3.4999999 3.49999992174

In this example the digits after the eighth are meaningless, and, when fewer than eight digits are printed, the least significant digit is rounded off according to the value of the next (non-printed) digit.

Acknowledgments

Preparation of this module was supported in part by the National Science Foundation, Division of Science Education Development and Research, through Grant #SED 74-20088 to Michigan State University.

[1] To see how to use double or triple precision in FORTRAN, see "Advanced Features of FORTRAN," (MISN-0-347).

MODEL EXAM

1. Construct a flowchart for the simple computer program below using standard flowchart symbols.

```
PROGRAM MULTIPLY 100 READ('Value for N?',N) M = N
* 5 WRITE('N x 5 =',M) IF (M - 5) 200,100,100 200
WRITE('DONE') END
```

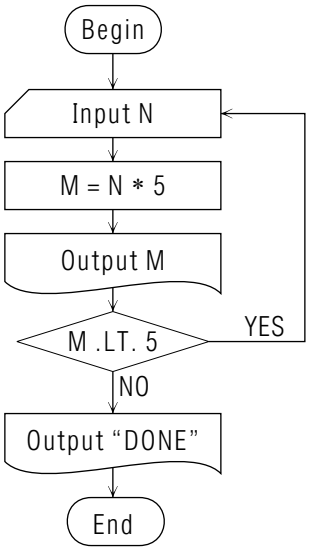
2. (a) List the steps usually followed in constructing a sizeable computer program. (b) Indicate which step is often the most time consuming, and two reasons why this is so.

3. Identify the programming error(s) in the following code and correct the program in a simple manner.

```
PROGRAM EVALUATE I = 0 100 I = I + 1 WRITE('FIRST VALUE
',I) J = 1 / (I - 10) WRITE('PRODUCES ',J) IF (I - 10)
100,110,110 110 STOP END
```


Brief Answers:

1.



2. a. Programming Steps:

   Define the problem

   Devise an algorithm

   Code the program

   Debug the program

   Run the program

   Analyze the results

   b. Debugging a large program often is the most time consuming step. Errors in coding (typos) may be hard to track down. Ambiguities or errors in the definition of the problem may also lead to erroneous results and require a basic revision in the algorithm.

3. Division by zero will occur when I = 10. Insert a statement such as

   `IF I = 10 GOTO 110`

   before performing the division.