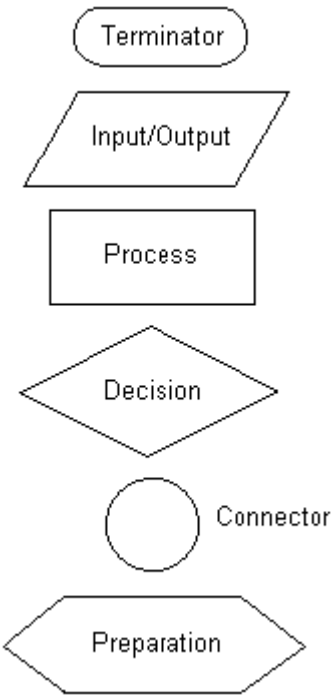
Introduction to Programming in C++: Algorithms, Flowcharts and Pseudocode

by Brent Daviduck

> **The following material was developed by Brent Daviduck of the Computer Systems Technology program at Red Deer College in Alberta, Canada. For more information on the program and for other material, see: http://cst.rdc.ab.ca/.**

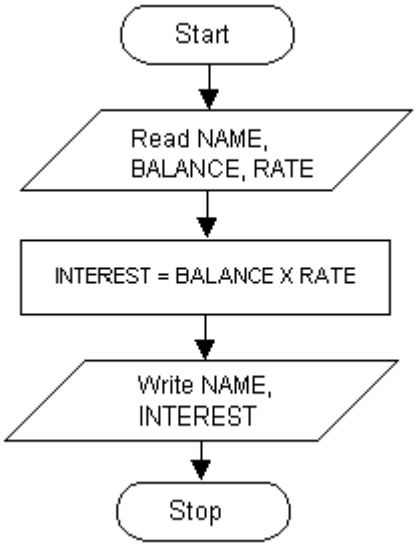
A sequence of instructions is called an **algorithm**. Algorithms are a fundamental part of computing. If you study computing for many years you will study algorithms of frequently used processes. Books have been written on algorithms for such common activities as storing and ordering data. As most problems you get are unique, you will develop your own algorithms. However, you may find standard algorithms for those parts of your programs that do common activities.

There are two commonly used tools to help to document program logic (the algorithm). These are **flowcharts** and **Pseudocode**. We will use both methods here. Generally, flowcharts work well for small problems but Pseudocode is used for larger problems. Some of the common symbols used in flowcharts are shown below:



With flowcharting, essential steps of an algorithm are shown using the shapes above. The flow of data between steps is indicated by arrows, or flowlines. For example, a flowchart (and equivalent Pseudocode) to compute the interest on a loan is shown below:

Flowchart	Pseudocode



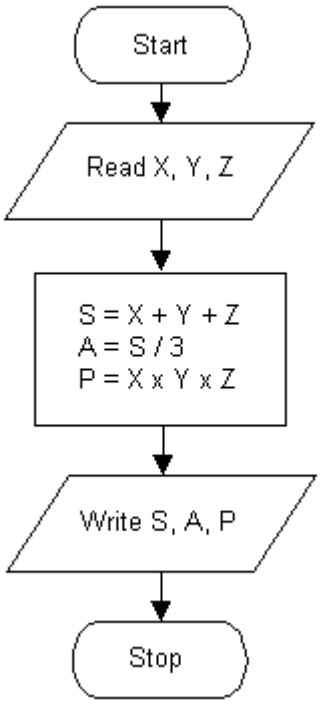
Read NAME, BALANCE, RATE

Compute INTEREST as BALANCE x RATE

Write (Display) NAME and INTEREST

Note that the Pseudocode also describes the essential steps to be taken, but without the graphical enhancements. Another example of a flowchart and the equivalent Pseudocode is shown below. In this case, the program computes the sum, average and product of three numbers:

Flowchart	Pseudocode



Read X, Y, Z

Compute Sum (S) as X + Y + Z

Compute Average (A) as S / 3

Compute Product (P) as X x Y x Z

Write (Display) the Sum, Average and Product

Decisions (Switching logic)

Switching logic consists of two components - a condition and a *goto* command depending on the result of the condition test. The computer can determine the *truth value* of a statement involving one of six mathematical relations symbolized in the table below:

Symbol	Meaning
==	Equals
!=	Not Equal
<	Less than
<=	Less than or equal to
>	Greater than
>=	Greater than or equal to

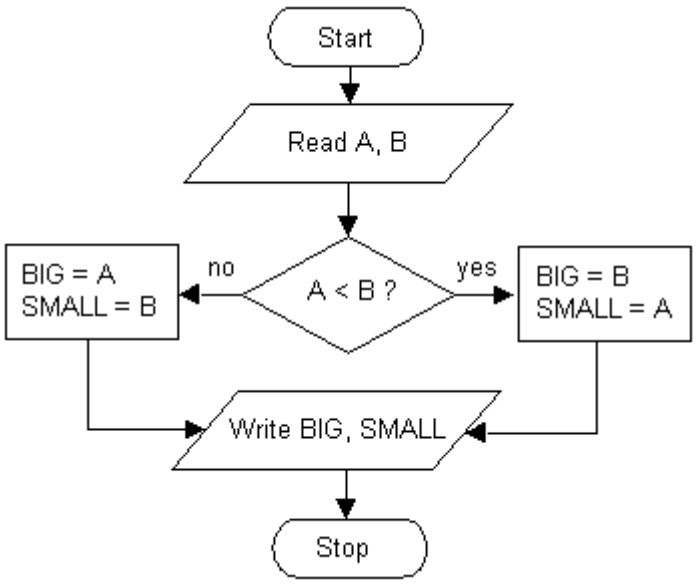
In practice, the computer is presented not with a true/false statement, but with a question having a "Yes" or "No" answer, for example if A = 10, B = 20, K = 5, and SALES = 10000, then:

Condition (Question)	"Answer"
Is A == B?	No
Is B > A?	Yes
Is K <= 25?	Yes
Is SALES >= $5000.00?	Yes

With each question, the computer can be programmed to take a different course of action depending on the answer. **A step in an algorithm that leads to more than one possible continuation is called a decision**.

In flowcharting, the diamond-shaped symbol is used to indicate a decision. The question is placed inside the symbol, and each alternative answer to the question is used to label the exit arrow which leads to the appropriate next step of the algorithm. The decision symbol is the only symbol that may have more than one exit.

The example below shows the flowchart for a program that reads two numbers and displays the numbers read in decreasing order:



The equivalent Pseudocode is shown below. Note that with Pseudocode, indentation is used to show the various steps that apply to a decision:

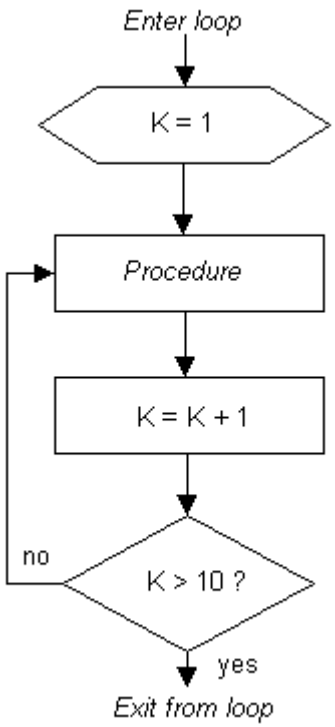
```
Read A, B

If A is less than B
   BIG = B
   SMALL = A
else
   BIG = A
   SMALL = B

Write (Display) BIG, SMALL
```

Loops

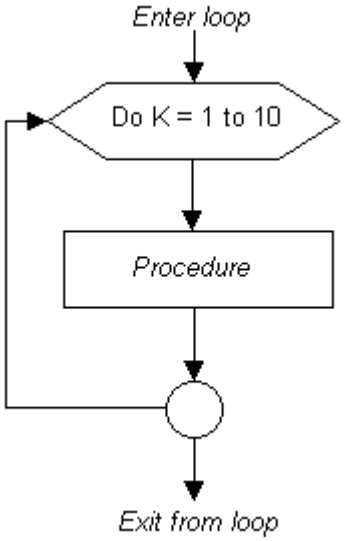
Most programs involve repeating a series of instructions over and over until some event occurs. For example, if we wish to read ten numbers and compute the average, we need a loop to count the number of numbers we have read.

Count loops are loops where the program must count the number of times operations are completed. The flowchart below illustrates a loop that counts from 1 to 10:

Count loop flowchart



The flowchart shown on the left may be simplified to the form shown on the right.



While count loops work the exact number of times needed, in many cases we do not know how many times we want to do something. It is often dependent on the data provided to the program. Imagine we change our problem to read and compute the average of a number of numbers. We won't know how many numbers there are but will read numbers until there are no more.

Two alternative solutions (using Pseudocode) are shown below:

pre-test loop:

```
set average to zero
```

post-test loop:

```
set average to zero
```

```
set count to zero
set total to zero

read number
while ( not end-of-data )
   increment count by 1
   total = total + number
   read number

if ( count > 0 ) then
 average = total / count

display average
```

```
set count to zero
set total to zero

do
   read a number
   increment count by 1
   total = total + number
while ( not end-of-data )

if ( count > 0 ) then
 average = total / count

display average
```

Both these assume that the computer will tell the program when there is no more numbers. This is called an **end-of-data** or **end-of-file** test.

There is an important difference between the pre-test and post-test loops. The pre-test version will work even if there are no numbers, the post-test version assumes the body of the code will be obeyed at least one time. Both forms of loops are appropriate in different circumstances.

Looping with switching and goto's

The looping and switching logic above follow well defined rules. In fact, we can implement any of these constructs with a condition and a goto (unconditional branch) instruction. An example of this logic is was illustrated in the loop flowchart shown previously. Early programs were written this way. As the problems became more complex it became impossible to follow the logic when things go wrong.

Imagine trying to sort out code like this

```
step 01: do something
step 02: go to step 16
step 03: do something
step 04: if some event has occurred go to step 19
…
step 16: if some event has occurs go back to step 4
step 17: go to step 1
step 18: do something
step 19: if something is greater than 10 goto step 1
step 20: go to step 1
```

Spaghetti code was born! The rule is simple, you avoid goto statements but use the higher level constructs we have introduced here for switching logic and looping logic. Even with these, sorting out the logic in a program can be quite excruciating. However, it is always good fun when you find eventually the error!.

You notice that the logic is nested. Within a loop we may have switching logic and within that switching logic we may again have a loop, and so it goes. Program algorithms are usually hierarchical in nature. The style of programming we are using is called procedural, because we define the solution in the firm of a procedure indicating what to do. Most problems require this approach. There are programming methods (or paradigms) where you express the problem algebraically, or as formal logic. A compiler like program will then solve the problem for you. A skilled programmer will use the appropriate method for the nature of the problem being solved.

As logic gets deeply nested it becomes difficult to follow what is happening. As a general rule, logic should not be nested more than 3 or 4 levels deep. The way to avoid these problems is with the use of functions.

Functions

Functions (or subroutines, or sub programs) are named chunks of logic that do something. It is the bundle of sticks issue again. We break our big problem into many smaller problems that can be easily solved. These become our functions. We then use these functions to solve the big problem. You may have noticed that we can more easily test our solutions this way. We can test the functions separately, and then use the working functions to build a larger program. This is far easier than trying to find out what has gone wrong when you have many hundreds of instructions interacting, and all you know is that something unexpected has happened.

We have covered an immense amount of ground. The way it will become real is in use. You can read as many books about repairing cars as you like, but until you actually do it, you won't see the value of the written advice. Practice and enjoy, and be kind to yourself, it can be very frustrating.

Additional Reference:

Schaum's Outline of Theory and Problems of Essential Computer Mathematics, by Dr. Seymour Lipshutz, McGraw-Hill, Inc., ISBN: 0-07-037990-4. Chapter 5 - Algorithms, Flowcharts, Pseudocode programs.