MODULAR SYSTEM

PROGRAMMING WITH C++

Osman AY
Muhammed Akif HORASANLI

Zambak PUBLISHING

http://book.zambak.com

Digital Assembly
Zambak Typesetting & Design

Editor
Osman AY

Proofreader
Andy MARTIN

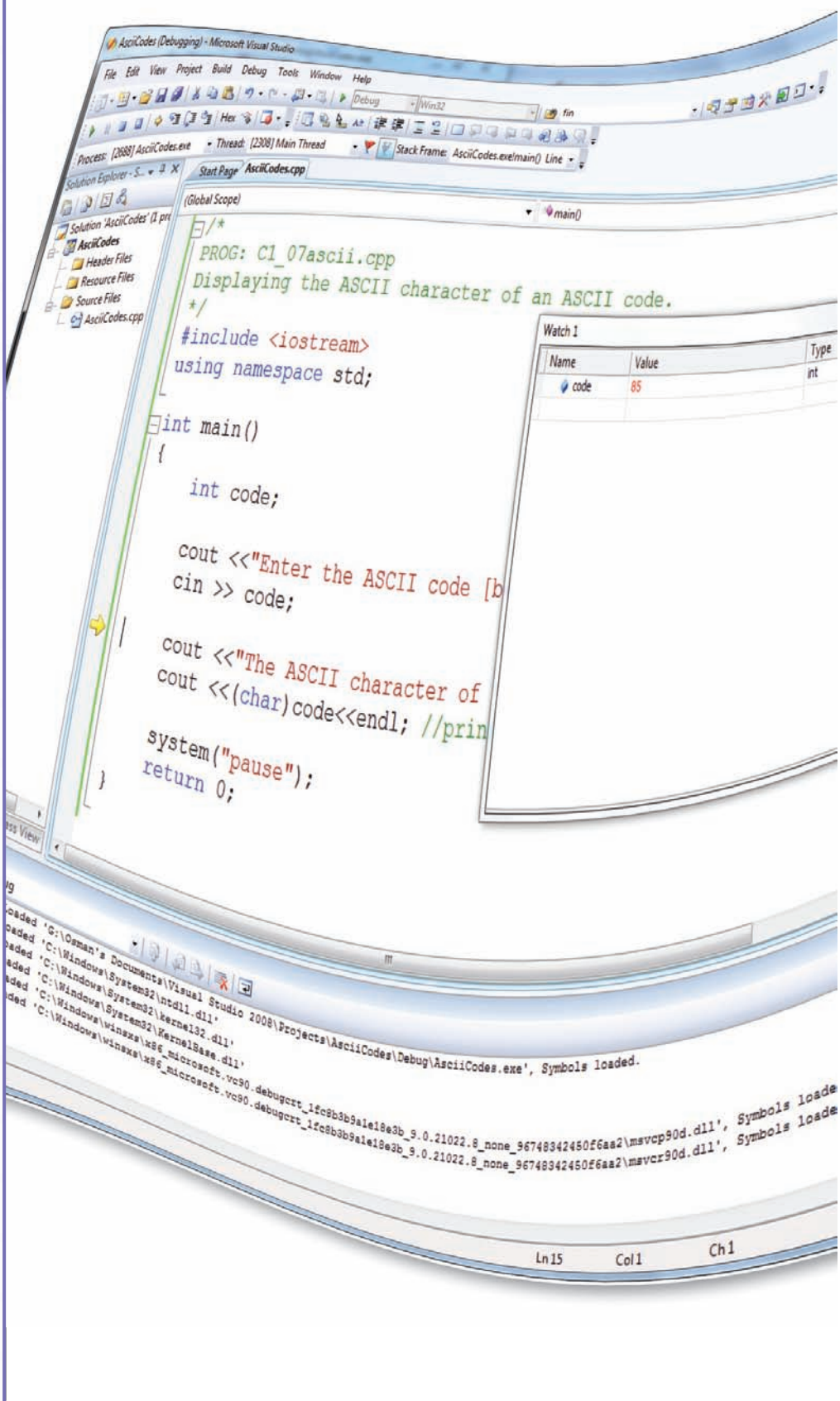
Page Design
Edip TÜRK

Publisher
Zambak Basım Yayın Eğitim ve Turizm
İşletmeleri Sanayi Ticaret A.Ş.

Printed by
Çağlayan A.Ş. Sarnıç Yolu Üzeri No:7
Gaziemir / Izmir, May 2010
Tel: +90-0-232-252 22 85
+90-0-232-522-20-96-97
ISBN: 978-975-266-245-2
Printed in Turkey

DISTRIBUTION
ZAMBAK YAYINLARI
Bulgurlu Mah. Haminne Çeşmesi Sok.
No. 20 34696 Üsküdar / Istanbul

Tel.: +90-216 522 09 00 (pbx)
Fax: +90-216 443 98 39
http://book.zambak.com



CONTENTS

CHAPTER 1

INTRODUCTION TO PROGRAMMING

- Understanding the Programming
- Basic Input/Output
- Operators
- Variables
- ASCII Code
- Using Text Files

Understanding the Programming

Ibn Musa **al-Khwarizmi** (Algorizm) (770 - 840 AD) was born in Uzbekistan. His parents migrated to Baghdad when he was a child. He is best known for introducing the mathematical concept Algorithm, which is so named after his last name.

Al-Khwarizmi was one of the greatest mathematicians who ever lived. He was the founder of several branches and basic concepts of mathematics. He is also famous as an astronomer and geographer. He is recognized as the founder of Algebra, as he not only initiated the subject in a systematic form but also developed it to the extent of giving analytical solutions of linear and quadratic equations. The name Algebra is derived from his famous book Al-Jabr wa-al-Muqabilah. He developed in detail trigonometric tables containing the sine functions. Al-Khwarizmi also developed the calculus of two errors, which led him to the concept of differentiation. He also refined the geometric representation of conic sections.

Programming is instructing a computer to perform a task for you with the help of a programming language. The instructing part requires a step by step solution to the task. This step by step solution is called an algorithm after the name of AlKharizmi.

People who make computer programs are called programmers. There are usually two difficulties for computer programmers; Finding a feasible algorithm (algorithm design) and writing the program (implementation). People who use the programs are called end-users.

A computer program (software) contains a sequence of instructions for a computer. One program is usually composed of three parts:

- Input part gets the data from an input device. Our programs, in the book, will get the data from keyboard or from a text file.
- Process part is the hardest working part of the program. It carries out the algorithm and finds out the desired result.
- Output part gives the result of the program. Our programs will display the result on the screen or print it into a text file.

The First C++ Program

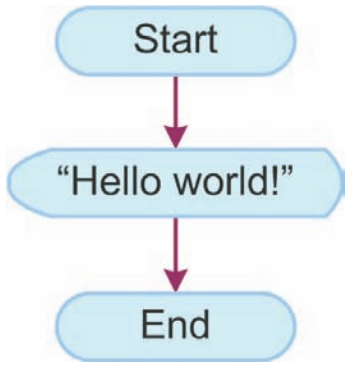
It is time to type our first C++ program. This program is going to prompt a line of text that says "Hello World!". This program has no input and no process but only output which says "Hello world!".

All the programs in this book have been compiled with Microsoft Visual Studio 2005. You can also compile them with GNU C++ (g++) in Linux environment, or use free Windows IDEs like Dev-C++ (http://www.bloodshed.net) and CodeBlocks (http://www.codeblocks.org).

```
/*
PROG: C1_01hello.cpp
Understanding structure of a C++ program.
Printing a line of text.
Using comments.
*/

#include <iostream>
//includes the declarations of the basic standard input-output
//library in C++, and its functionality is going to be used later
//in the program.

using namespace std;
//Namespaces are containers that contain the declarations of all
//the elements of the standard C++ library

int main()                  //the only function in this program.
{
  cout <<"Hello world!";    //print "Hello world!". cout is
                            //declared in the iostream standard
                            //file within the std namespace


  system("pause");          //Wait until user hits a key and
```

```
                                  //displays a message
  return 0;                       //the main function ends properly
}
```

```
Hello world!Press any key to continue . . .
```



Flowchart of the Program "01hello".

C++ programs consist of one or more modules. Each module performs a specific task. These modules are called functions. The "Hello World!" program has only one module that is the main function. Any C++ program is been started to execute from the main function so each program must have this function.

Before the functions, the program has an "**include**" and "**using namespace**" part. This part declares the libraries where the C++ commands we are going to use in the program are defined.

Like each sentence ends with a period ('.'), each C++ statement ends with a semicolon character (';').

Besides the program codes, the program has some comments. C++ has two ways to insert comments into source code: Single line comment and multiple line comment. Single line comments are written behind the double slash characters ("**//**") and multiple line comments are enclosed between slash and asterisk ("**/***") and asterisk and slash ("***/**") characters. Comments are ignored by the compiler.

A **flowchart** is a visual representation of the algorithms. Is is made up of a few symbols: terminal, input, process, decision, output, and connector.

Breaking a Text into Multiple Lines

Use end of line "**endl**" or new line '**n/'** characters with in a cout statement to make a new line. '**n/**' characters are inherited from C programming. We prefer to use "**endl**" notation.

Include precise comments in your program to make it self-documentary and easy to read. Usually the reading time for programs is much more than the writing time.

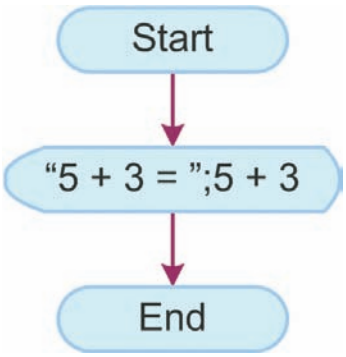
```
/*
PROG: C1_02hello.cpp
Using endl.
*/
#include <iostream>
using namespace std;

int main()
{
  cout <<"Hello world!"<<endl;    //move the cursor to the
                                  //beginning of the next line.
  cout <<"This is my C++ program."<<endl<<endl;
  system("pause");
  return 0;
}
```

`cout<<"Hello "<<endl;` and `cout<<"Hello \n";` statemens print the same output.

```
Hello world!
This is my C++ program.

Press any key to continue . . .
```

Basic Arithmetic

Any statement enclosed with double quotes (" ") in a cout statement is displayed directly and any arithmetical or logical expression is evaluated and then the result is displayed. The program below shows the result of the expression 3 + 5.



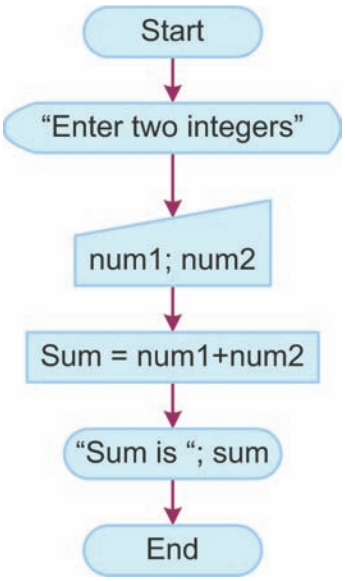
Flowchart of the Program “03sum”

```
/*
PROG: C1_03sum.cpp
*/
#include <iostream>
using namespace std;

int main()
{
  cout <<"5 + 3 = "<<5+3<<endl;  //calculate and print the sum
  system("pause");
  return 0;
}
```

```
5 + 3 = 8
Press any key to continue . . .
```

Getting Data from the User (Input)

Programs usually read the input data from the standard input (keyboard) or from an input file. "**cin**" command is used to read data from the standard input. The following program reads two integers, calculates their sum and then outputs the result.



Flowchart of the Program “04sum”

int num1, num2, sum; declares three variables. The names of the variables are num1, num2 and sum. A **variable** is a named storage location that can contain data that can be modified during program execution. This declaration specifies that those variables can contain integer values (-45, 0, 11, 37, etc.). "num1" and "**num2**" will be used to store the input data, and "**sum**" will be used to keep the sum of input values in the program.

```
/*
PROG: C1_04sum.cpp
Getting data from keyboard, making sum of two integers,
understanding variables, and using assignment operator.
*/
#include <iostream>
using namespace std;

int main()
{
     int num1, num2, sum;          //num1, num2 and sum are three
                                   //variables type of integer.
     cout<<"Enter two integers:"<<endl;
     cin >> num1 >> num2;  //cin reads two values for
                           //num1 and num2.
     sum = num1 + num2;    //sum gets the value of num1+num2.
```

```
    cout <<"Sum is "<<sum<<endl;
    system("PAUSE");
    return 0;
}
```

```
Enter two integers:
5 7
Sum is 12
Press any key to continue . . .
```

sum = num1 + num2; first computes the sum of the values of num1 and num2 then assigns the result to the variable sum. We have used an arithmetic operator (+) and assignment operator (=) in this statement.

C++ provides **assigment operator** and **compound assignment operators**(`+=, -=, *=, /=, %=, >>=, <<=, &=, ^=, |=`). For example `sum=sum+5` can be written as `sum+=5`. The assigment operator can be used for **chain assignment** processes. For example `a=3;` and `b=3;` can be written as `a=b=3;`

Arithmetic Operators

Operation	Operator	Example
Addition	+	5 + 4 = 9
Subtraction	-	5 - 4 = 1 and 4 - 5 = -1
Multiplication	*	5 * 4 = 9
Division (integer)	/	15 / 3 = 5 and 12 / 5 = 2
Modulus	%	12 % 5 = 2, 15 % 3 = 0, and 3 % 5 = 3

Beside the assignment and the arithmetic operators C++ has many others. The most common **C++ operators** are:

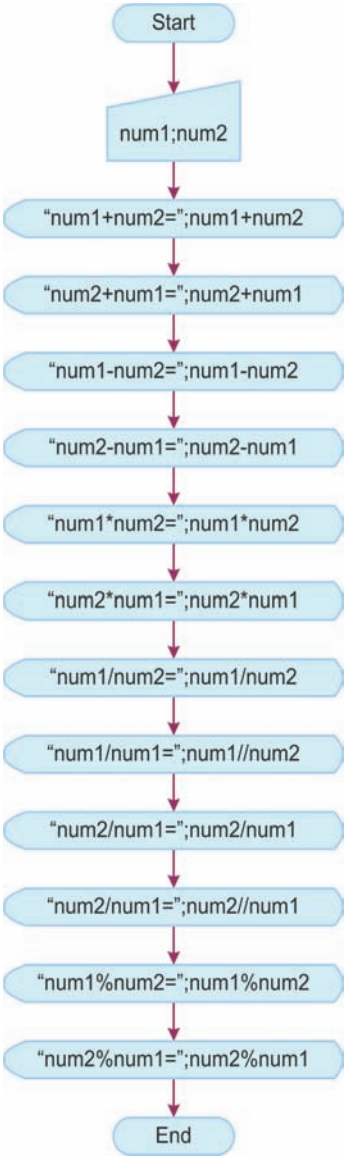
- assignment
- arithmetic
- increment and decrement
- string concatenation
- relational
- logical
- conditional
- bitwise

PROG: C1_05calculator.cpp

```
/*
Demonstrating arithmetic operators. Calculating sum, difference,
product, quotient, and remainder. Using (float) casting to get
floating-point quotient.
*/
#include <iostream>
 using namespace std;

int main()
{
    int num1, num2;

    cout <<"Enter two integers:";
    cin >> num1 >> num2;

    cout <<num1 <<"+"<<num2<<"="<<num1+num2<<endl;
    cout <<num2 <<"+"<<num1<<"="<<num2+num1<<endl<<endl;

    cout <<num1 <<"-"<<num2<<"="<<num1-num2<<endl;
    cout <<num2 <<"-"<<num1<<"="<<num2-num1<<endl<<endl;
```

The **bitwise shift operators** shift their first operand left (<<) or right (>>) by the number of positions the second operand specifies. Shifting one position left is equivalent to multiply the number by 2, and shifting one position right is equivalent to divide the number by 2.

(4<<3) yields 32 and (12>>2) yields 3.



Flowchart of the Program "05calculator"

```
    cout <<num1 <<"*"<<num2<<"="<<num1*num2<<endl;
    cout <<num2 <<"*"<<num1<<"="<<num2*num1<<endl<<endl;

    cout <<num1 <<"/"<<num2<<"="<<num1/num2<<endl;
    cout <<num1 <<"/"<<num2<<"="<<(float)num1/num2<<endl;
    cout <<num2 <<"/"<<num1<<"="<<num2/num1<<endl;
    cout <<num2 <<"/"<<num1<<"="<<(float)num2/num1<<endl<<endl;

    cout <<num1 <<"%"<<num2<<"="<<num1%num2<<endl;
    cout <<num2 <<"%"<<num1<<"="<<num2%num1<<endl<<endl;

    system("PAUSE");     return 0;
}
```

```
Enter two integers:7 3
7+3=10
3+7=10

7-3=4
3-7=-4

7*3=21
3*7=21

7/3=2
7/3=2.33333
3/7=0
3/7=0.428571

7%3=1
3%7=3

Press any key to continue . . .
```

Precedence of Arithmetic Operators

- Parentheses ("( )") are evaluated first. The expression in the innermost parentheses is evaluated first if the parentheses are nested.
- After parentheses multiplication (*), division (/), and modulus (%) operators are evaluated.
- Addition (+) and subtraction (-) are evaluated last.
- The operators with the same precedence are evaluated left to right.

```
3 * 5 + 2 = 17
3 * (5 + 2) = 21
5 + 3*4 - 2 = 15
6*8/4 = 12
6*(8/4) = 12
```

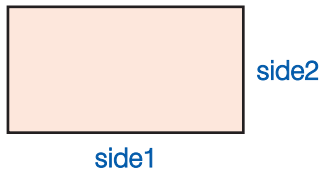
Exercise: Rectangle

Make a program to calculate area and perimeter of a rectangle.

Input: length of side1 and length of side2.

Process: area = side1*side2
Perimeter = 2*(side1+side2)

Output: area and perimeter



Fundamental C++ Variables

A variable is a memory place in which you can store a value and from which you can later retrieve that value. Notice that this temporary storage is used only during the execution of the program.

The following table summarizes the fundamental C++ variables.

	Type	Size in Bytes	Values
integer variables	unsigned short int	2	0 to 65,535
	short int	2	-32,768 to 32,767
	unsigned int	4	0 to 4,294,967,295
	int	4	-2,147,483,648 to 2,147,483,647
	unsigned long int	4	0 to 4,294,967,295
	long int	4	-2,147,483,648 to 2,147,483,647
	long long int	8	9.223.372.036.854 to 9.223.372.036.853
floating-point variables	float	4	1.2e-38 to 3.4e38
	double	8	2.2e-308 to 1.8e308
logical variable	bool	1	true or false
character variable	char	1	256 character values

Always name your variables with a great care, and explain them thoroughly.

- **Integer variables** store whole numbers (-4, 3, 51, etc). Unsigned integer type variables cannot have negative values, whereas other integer type variables (signed integers) may have negative and positive values.
- **Floating-point variables** store decimal numbers (3.5, -5,123, 4.0, etc).

- **Logical variables** store the result of logical expressions and get only the values true and false. False is represented with 0 and true is represented with a positive value (usually 1) in C++. Logical expressions are usually used in decision and repetition structures to control the flow of the program.
- **Character variables** are used to store characters (letters, numbers, punctuation characters, etc). Characters are enclosed with a pair of single quotes in C++, like 'a', 'B', '7', '+', etc.

The **sizes of variables** might be different from those shown in the table, depending on the compiler and the computer you are using. Use **sizeof()** operator to measure the sizes of variable types in your system. The **sizeof()** operator returns the number of bytes in variable or type.

```
/*
PROG: C1_06sizeof.cpp
C++ variable types and their sizes in bytes
*/
#include <iostream>
using namespace std;
int main()
{
  cout <<"int = "<<sizeof(int)<<endl;
  cout <<"short int = "<<sizeof(short int)<<endl;
  cout <<"long int = " <<sizeof(long int)<<endl;
  cout << "long long int =" <<sizeof(long long int)<<endl;

  cout <<"float = "<<sizeof(float)<<endl;
  cout <<"double ="<<sizeof(double)<<endl;
  cout <<"long double ="<<sizeof(double)<<endl;

  cout <<"bool ="<<sizeof(bool)<<endl;

  cout <<"char ="<<sizeof(char)<<endl;

  system("pause");  return 0;
}
```

```
int = 4
short int = 2
long int = 4
long long int =8
float = 4
double =8
long double =8
bool =1
char =1
Press any key to continue . . .
```

Exercise: Circle

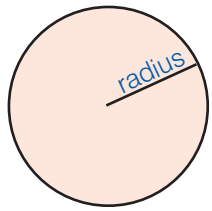
Make a program to calculate area and circumference of a circle.

Input: radius of circle.

Process: area = PI*radius*radius

circumference = 2*PI*radius

Output: area and circumference



PI is an arithmetical constant whose approximated value is 3.14159. Add the following line at the beginning of your program to use this constant in your program:

```
#define PI 3.14159
```

Like variables, **constants** are data storage locations. Unlike variables, constants don't change. You must initialize a constant when you create it, and you cannot assign a new value later.

Most of the common constants have already been defined in C++. For the user defined constants, there are two main techniques used to create constant values:

a. using the **define** keyword

```
#include <iostream>
#define PI 3.14
.
.
```

b. using the **const** keyword

```
.
.
int main()
const float PI = 3.14;
.
.
```

Exercise: 1 or 0

Make a program that gets two numbers (let's say A and B) and then displays 1 if the first number is bigger then second one, otherwise displays 0.

Input: Two numbers A and B.

Process: Comparing A and B. Use bigger than operator in the comparison (A>B).

Output: 1 if the first number is bigger, 0 if the first number is not bigger.

Visit the site http://www.asciitable.com to see the ASCII table.

ASCII Codes

There are only 1s and 0s in a computer system. Computers work with binary numbers. We can convert binary numbers into their equivalent decimal numbers that are the numbers we use in our daily life. But what about other characters, letters, punctuation marks, and special characters. How can we present them to the computers?

Character encoding (character set) tables are used to represent the characters with numbers. **ASCII** (1963) and **EBCDIC** (1964) are two international standard character sets.

ASCII (Pronounced ask-ee) is an acronym for American Standard Code for Information Interchange. In ASCII, every letter, number, and punctuation symbol has

EBCDIC (Extended Binary Coded Decimal Interchange Code) is an 8-bit character encoding (code page) used on IBM mainframe operating systems as well as IBM minicomputer operating systems. It is also employed on various non-IBM platforms such as Fujitsu-Siemens and Unisys MCP. Although it is widely used on large IBM computers, most other computers, including PCs and Macintoshes, use ASCII codes.

a corresponding number, or ASCII code. For example, the character for the number 1 has the code 49, capital letter A has the code 65, and a blank space has the code 32. This encoding system not only lets a computer store a document as a series of numbers, but also makes it possible to transfer data from one computer to another.

In an ASCII file, each alphabetic, numeric, or special character is represented with a 7-bit binary number (a string of seven 0s or 1s). 128 possible characters are defined. There are also ASCII extensions in use which utilize 8 bit codes to represent international characters in addition to the standard ASCII scheme.

cout prints the ASCII character of an ASCII code with "char" casting. The following program reads an ASCII code (an integer) and prints its character.

```
/*
PROG: C1_07ascii.cpp
Displaying the ASCII character of an ASCII code.
*/
#include <iostream>
using namespace std;

int main()
{
  int code;
  cout <<"Enter the ASCII code [between 0 and 127] ";
  cin >> code;

  cout <<"The ASCII character of "<<code<<" is "
       <<(char)code<<endl; //print the character not number

  system("pause");
  return 0;
}
```

```
Enter the ASCII code [between 0 and 127] 75
The ASCII character of 75 is K
Press any key to continue . . .
```

Exercise: ASCII Character

Make a program to print code of an ASCII character.

Input: an ASCII character.

Output: ASCII code of the input character.

Sample Output:

```
Enter an ASCII character A

The ASCII code of A is 65
Devam etmek için bir tusa basin . . .
```

Reading and Printing Strings

In programming, a string is an ordered series of characters. You can consider them words and sentences. They are represented with enclosed double quotes in a C++ program code, like "Hello!" or "I am learning C++".

C++ provides string class for string operations. S string class is defined in the <string> standard library. The instances of variable types are called variables, whereas the instances of classes are called objects. We are going to study classes and objects in the later chapters of this book.

Class and object are two main terms of Object Oriented Programming. Objects are the real elements in the program and classes are like a blueprint of the objects. Whenever we need an object we can create it from its class.

cin reads only the first word of a string, and cout prints all characters of a string. The string class has its own methods (functions) for string manipulation. We are going to study string class later in this book.

The '+' is called string concatenation operator with strings and is used to concatenate two string objects. Do not confuse the string concatenation and arithmetic addition operator. Examine the following two operations.

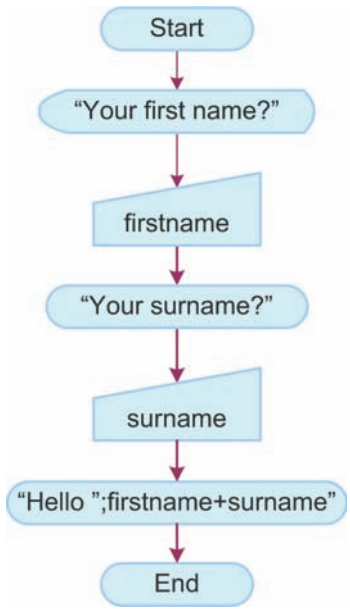
57 + 33 is 90 Here '+' is addition operator.

"57" + "33" is "5733" Here '+' is string concatenation operator.

The following example reads name and surname of the user, and then prints a hello message.

```
/*
PROG: c1_08string.cpp
Reading a word. Using the string data type and the string
concatenation operator (+).
*/
#include <iostream>
#include <string>                  //string library
using namespace std;
int main()
{
  string firstname, surname;       //two objects of string class

  cout << "Enter your firstname."<<endl;
  cin >> firstname;
  cout << "Enter your surname."<<endl;
  cin >> surname;
  //string concatenation
  cout <<"Hello "<<firstname + " " + surname<<endl;
  system("pause");  return 0;
}
```

```
Enter your firstname.
Alan
Enter your surname.
Smith
Hello Alan Smith
Press any key to continue . . .
```



Flowchart of the Program "08string"

Initialization of Variables

You can give the initial values to the variables during the declaration in C++. The following program demonstrates how to declare different type of variables. Notice that string objects can be initialized in two ways. You may use either of them in your programs.

```
/*
PROG: c1_09init.cpp
Initializtion of variables.
*/
#include <iostream>
#include <string>
using namespace std;
int main()
{
  string st1("I am learning ");   //st1 is been initialized to
                                  //"I am learging "
  string st2 = "C++";      //st2 is been initialized to "C++"
  int int1 = 8, int2 = 5; //int1 is been initialized to 8, and
                          //int2 to 5
  float flt1 = 7.5, flt2 = 3.9;  //flt1 is been initialized to
                                 //7.5, and flt2 to 3.9
  char ch1 = 'A';                //ch1 is been initialized to 'A'
  bool bl1 = true, bl2 = false;  //bl1 is been initialized to
                                 //true, and bl2 to false

  cout<<"st1 + st2 is "<<st1 + st2<<endl;
  cout<<"int1 + int2 is "<<int1 + int2<<endl;
  cout<<"flt1 + flt2 is "<<flt1 + flt2<<endl;
  cout<<"ch1 is "<<ch1<<endl;
  cout<<"bl1 is "<<bl1<<endl;
  cout<<"bl2 is "<<bl2<<endl;
  system("pause");  return 0;
}
```

```
st1 + st2 is I am learning C++
int1 + int2 is 13
flt1 + flt2 is 11.4
ch1 is A
bl1 is 1
bl2 is 0
Press any key to continue . . .
```

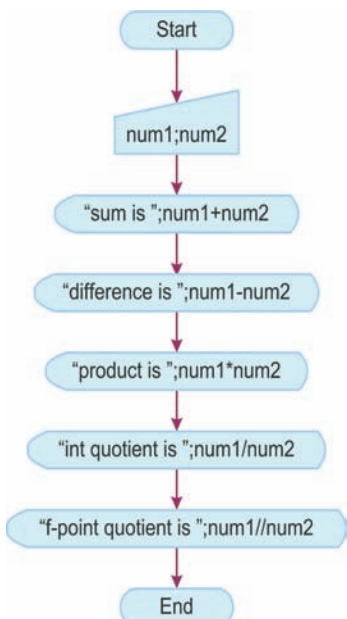
Using Text Files as Input and Output

C++ provides two functions to read from a text file and to write into a text file. Those functions are **ifstream()** and **ofstream()**. Both functions are declared in the <fstream> header. **ifstream** opens an existing input file whereas, **ofstream** creates or recreates and opens the output file.

Data input and output operations on text files are performed in the same way we operated with “cin” and “cout”. ifstream() function defines an identifier to read from a file, and the name and location (path) of that file. In the same way, ofstream() function defines an identifier to write into a file, and the name and location (path) of the file. I prefer to use "fin" and "fout" identifiers in my programs fin takes the role of cin and fout takes the role of cout. After you have finished with the input and output files you should close them so that their resources become available again for the system. close() function is used to close the open files.

The following program read two integers (num1 and num2) from the file numbers.in. Computes sum, difference, product and quotient of those two numbers and then writes the result into the file numbers.out.

```
/*
PROG: c1_10file.cpp
Using input and output files.
*/
#include <fstream>
using namespace std;

int main()
{
     ifstream fin("numbers.in");  //open input file
     ofstream fout("numbers.out");//create and open output file

     int num1, num2;
     fin >>num1 >>num2;     //read two integers from the input file

     //Make arithmetical calculations and write the result into
     //the output file
     fout <<"sum is "<<num1+num2<<endl;
     fout <<"difference is "<<num1-num2<<endl;
     fout <<"product is "<<num1*num2<<endl;
     fout <<"integer quotient is "<<num1/num2<<endl;
     fout <<"floating-point quotient is "<<(float)num1/num2<<endl;

     fin.close();           //close the input file
     fout.close();          //close the output file

     system("PAUSE");
     return 0;
}
```



Flowchart of the Program “10file”

numbers.in

```
5 3
```

numbers.out

```
sum is 8
difference is 2
product is 15
integer quotient is 1
floating-point quotient is 1.66667
```

SUMMARY

An **algorithm** is a set of ordered steps for solving a particular problem. A **computer program** is series of instructions or statements, in a form acceptable to a computer, to cary out an algorithm. A **programming language** is a human-created language that translates instructions from programmers to computers.

C++ is an object-oriented programming (OOP) language and the primary programming languages for computers of today. Any C++ program consists of modules that are called functions. The primary function of a C++ program is "**main**". C++ uses **cout** and **cin** for printing and getting the data.

Operators are symbols (such as +, *, /) used to perform arithmetic, relational, logical, assignment, string, bitwise etc. operations.

A **variable** is a named item used to represent data that can be changed while the program is running. Like a variable, a **constant** is a named item but it has a fixed value that does not change.

ASCII (American Standard Code of Information Interchange) is an international code standard for representation of characters, numbers, symbols and control characters, for use in data communication and data storage. ASCII text does not include special formatting features and therefore can be exchanged and read by most computer systems.

A **string** is a series of alphanumeric characters of any length. Strings are enclosed by double quotes in C++.

File processing consists of creating a file, storing data into a file, and retrieving data from a file. C++ performs file processing with **ifstream()** and **ofstream()** functions that are defined in the **<string>** header file.

REVIEW QUESTIONS

1. What is the output of the following program?

```
#include <iostream>
using namespace std;
int main()
{
     cout <<"I am ";
     cout <<"learning "<<endl<<"C++";
     return 0;
}
```

2. What is the output of the following program?

```
#include <iostream>
using namespace std;
int main()
{
     int a=1, b=2;
     cout <<a<<" "<<b;
     a = a+b;
     b = a+b;
     cout <<b<<" "<<a;
     return 0;
}
```

3. Calculate the values of variables in the given lines and complete the table.

```
#include <iostream>
using namespace std;
int main()
{
     int a=1, b=2, c=3;
     a =b + c;     b =a - b*c;
     c =(a + c)/(c - b); a=a%b/c; b=-b;
     return 0;
}
```

Line	a	b	c
İnt a=1, b=2, c=3;	1	2	3
a = b + c; b = a - b*c;			
c = (a + c)/(c - b); a = a%b/c; b = -b;			

4. What are the most suitable variable types for the given data?

Data	Variable Type
Age of a person	unsigned short
Name of a person	
Gender of a person ('F' or 'M')	
State of a electrical switch (ON or OFF)	
Number of the passengers in an airplane	
Area of a circle	
Temperature	

PROGRAMMING PROBLEMS

1. **(Sum of digits)** Make a program that reads a three-digit integer from the file "number.in" and then calculates sum of the digits and writes the result into the file "sum.out". Use the format of the sample output in the file "sum.out". (Hint: Use the modulus (%) and integer division (/) operators.)

number.in	sum.out
423	Sum of the digits is 4+2+3 = 9

2. **(Swapping)** You are given two integer variables, let's say a and b. How can you interchange the values of those two variables? In other words, **a** should get the value of **b**, and **b** should get the value of **a**.

   a. Use a temporarily third variable

   b. Do not use any additional variable

   c. Use the **swap** function (`swap(a, b);`).

3. **(To Upper)** Make a program that reads a letter from the keyboard and then converts the letter to upper case, if the letter is lower case.

   a. Change the ASCII code of the letter.

   b. Use the **toupper** function (`a = toupper(a);`).

FLOWCHART PROGRAMMING (OPTIONAL)

Understanding Flowchart Programming

Flowchart is a symbolic language to express algorithms. It visually presents the solution of a specific task.You have already seen flowcharts of some sample programs in this book and you became familiar with some of the flowchart symbols.

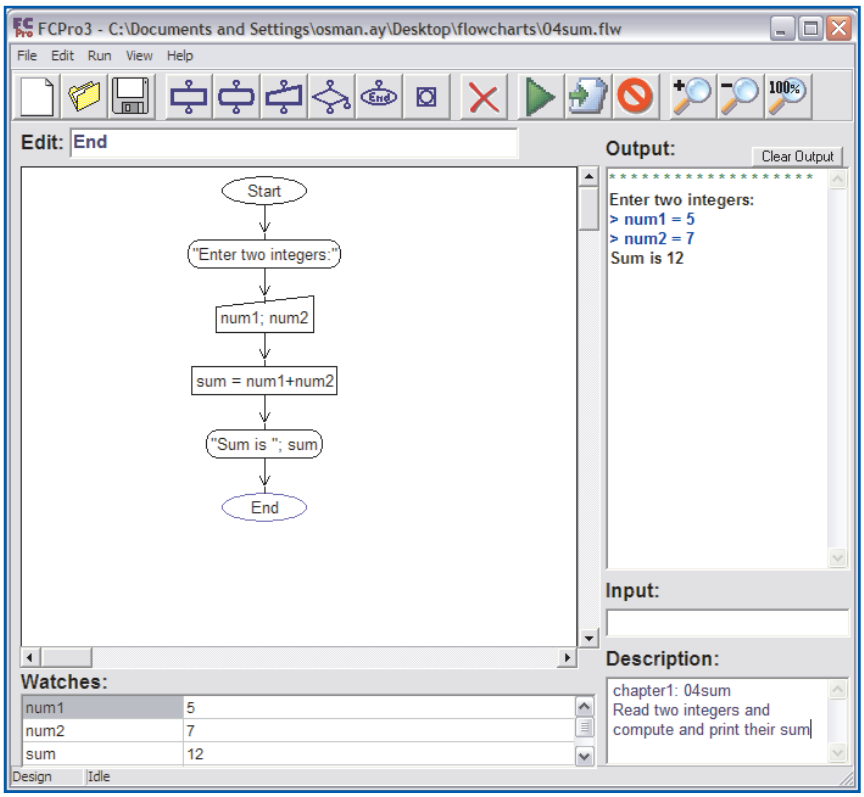
Traditionally people begin with flowchart programming, learn the fundamental techniques of programming (such as input, output, making calculations and comparisons, decisions, repetition) and then start to write their programs with a programming language. When starting the programming directly with a programming language will face you two difficulties: learning the syntax of the programming language and learning the programming techniques.

We have started the programming directly with a programming language (C++) in this book, because the syntax of C++ is self-explanatory and easy to understand for English speaking people.

You will find an IDE (Integrated Development Environment) flowchart programming (FCPRO) in the Web site of the book (http://book.zambak.com) where you can draw, run, trace the variables, and debug your flowcharts. FCPRO was made in C++ by Cosmin Raianu who was one of my best students from Romania.

By Using FCPRO, You Can Do The Following:

- Design your own flowcharts
- Save your flowcharts to disk and open them whenever you need to.
- Export your flowcharts as BMP files.
- Run your flowcharts, reading the input from the user and displaying the output.
- Debug your flowcharts, by stepping through them and inspecting the values of variables and expressions by using the Watches.
- Compile your flowcharts into standalone Win32 Console applications.
- Edit, save and open programs written in a special pseudocode.
- Convert flowcharts to their corresponding pseudocode programs and vice versa.
- Learn all about flowcharts, pseudocode and the way FCPro works by using the user-friendly integrated help system.

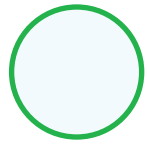


Refer to the FCPRO built-in help to get detailed information about flowchart programming and to learn how to use FCPRO.

Flowchart Symbols

Terminal (Start, End)

The terminal symbol marks where the flowchar starts and where it ends. Usually a starting terminal contains the word “START” or “BEGIN”, and an ending terminal contains the word “END” or “FINISH”.



Input (Get, Read)

The input symbols are used to get data from the user. The data is stored in variables. You don’t have to declare the type of the variables in FCPro. FCPro determines the type of the variables according to the first value you assign them.

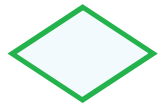
The shapes of the flowchart symbols can be slightly different in some other sources.

Process (Do)

A process symbol denotes that a process (arithmetic, logic, or assighment) will be carried out.

Decision (If)

It is a decision making and branching symbol. You can control the flow of the program by means of the decision symbol. That is, you can set loops and execute either part of the program depending of the result of the logical expressin in the decision symbol.



Output (Print, Display)

An output symbol is used to print a message, values of variables and results of expressions.

Flow Line (Arrow)

Lines indicate the sequence of steps and the direction of flow.

Connector (Joining)

The connector symbol connects two parts of a flowchart. It is usually used to connect two pages of a flowchart when you are manually drawing a flowchart on paper. In FCPro, It is used to prevent an arrow intersect with a flowchart symbol for a better view.

FCPro **Arithmetic Operators**:

- `+` : Addition
- `-` : Subtraction
- `*` : Multiplication
- `/` : Integer Division
- `//` : Fractional Division
- `%` : Modulus

Exercise: Flowcharts

Draw and run all the flowchart programs (01hello, 03sum, 04sum, 05calculator, 07string, 08file) that are given in this chapter in FCPRO.

CHAPTER 2

DECISION STRUCTURES

- The if Structure
- The if/else Structure
- Logical Operators
- The Conditional Operator
- The Switch Structure

C++

Introduction

In our daily life, we often encounter many options and we have to select one. Our selections decide the way of our life. In every separation point we make our decision and go on our way. While programming, we usually need to decide the path of the program flow according to the parameters and conditions. Actually the ability of making decision is one of the key points of intelligent programming. Thanks to control structures (decision & repetition) we are able to evaluate the condition and select the flow path. We have 4 types of decision structures in C++:

- If
- If/Else
- Conditional Operator (?)
- Switch

The If/else statement chooses between two alternatives. This statement can be used without the else, as a simple **if** statement. Another decision statement, **switch**, creates branches for multiple alternative sections of code, depending on the value of a single variable. Finally, the **conditional operator** is used in specialized situations.

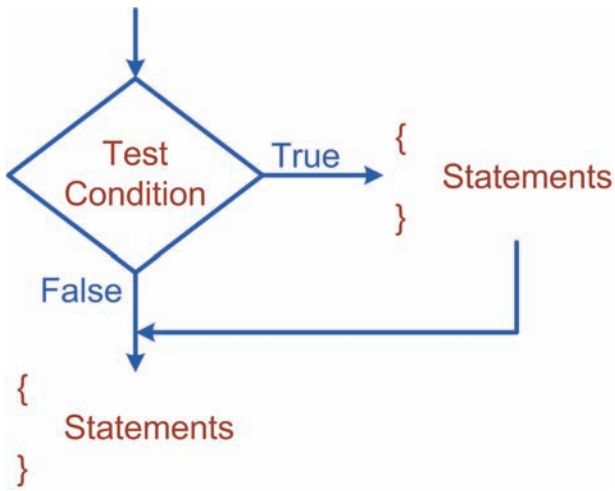
Relational and Equality Operators tests some kind of relation between two entities. These include numerical equality and inequalities. These operators usually return true or false, depending on whether the conditional relationship between the two operands holds or not. An expression created using a relational operator forms what is known as a **condition**. Decision structures evaluate the conditions.

C++ Relational and Equality Operators are:

>	:	Greater
>=	:	Greater or Equal
<	:	Smaller
<=	:	Smaller or Equal
==	:	Equal
!=	:	Not Equal

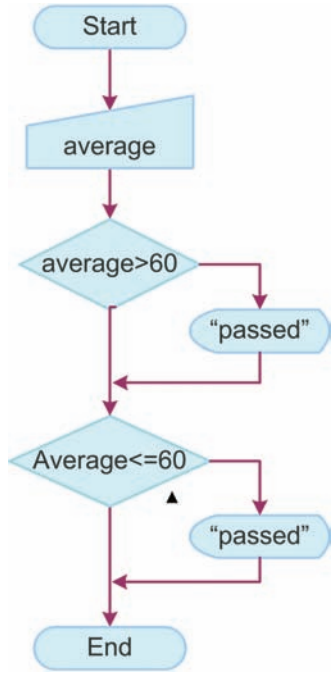
The if structure

The "if structure" is used to execute statement(s) only if the given condition is satisfied. The Computer evaluates the condition first, if it is true, statement is executed if not the statement is skipped and the program continues right after this conditional structure. This can be illustrated as:



The following code prints "passed" or "failed" depending on the average of the student by using two separate if structures.

```
if (average>60)
{
  cout<<"passed";        //if the  average is greater than 60
                         //print passed.
}

if (average<=60)
{
  cout<<"failed";        //if the  average is less than or equal
                         //to 60 print failed.
}
```



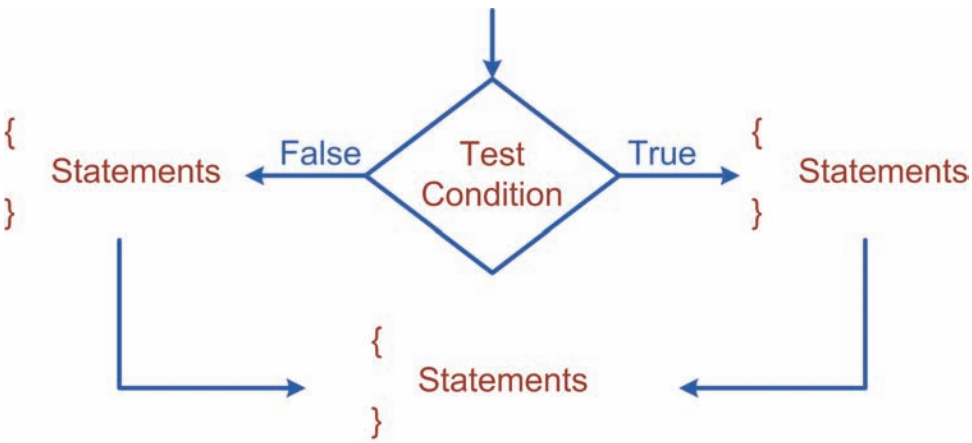
Using "if" in a Flowchart

The first **if** structure checks whether the average is greater than 60, if so writes "passed" to the output. The second **if** structure checks whether the average is less than or equal to 60 if so writes "failed" to the output.

If we have a single statement in a "if" then it can be written without braces; but if we have a compound statement to be executed we need to enclose them between braces ({ }).

The if/else structure

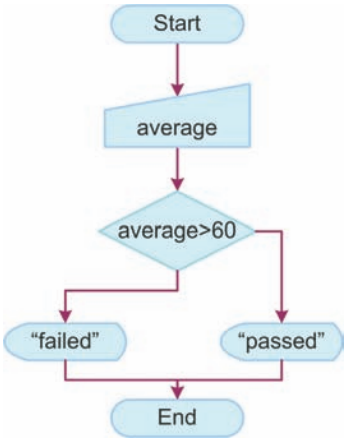
We can additionally specify what we want to do if the condition does not hold with **else**. While the **if** statement lets a program decide whether a statement or block is executed, the **if/else** statement lets a program decide which of two statements or blocks is executed. This can be illustrated as :



If any student passed, namely the average is bigger than 60, it is unnecessary to check if he failed. If he passed it is obvious that he didn't fail, and in the same manner if he didn't pass it is obvious that he failed.

Truth in C++

C++ has a very simple logic to cope with truth. Any value other than zero is regarded as true. Thus 1 and -2 are both true just like 0.0001.Since they are all different from zero. In the same logic 0, 0.0 or +0.0 and -0 are all accepted as false.



Using "if/else" in a Flowchart

```
if (average>60)
  cout<<"passed";          //if the  average is greater than 60
                           //prints passed.
else
  cout<<"failed";          //if condition does not hold then
                           //prints failed.
```

Exercise: Odd or Even

Write a program that decides if a number is odd or even.

Input : An integer.

Output : "Odd" or "Even".

Hint : Even numbers are divisible by 2. That is to say when an even number is divided by 2, the remainder is 0. Use modulus operator (%) to obtain the remainder.

Let's improve our average example. What if we have an average value less than 0? As a programmer we should always keep in mind the unexpected cases. We would be able to respond to the user that any value less than 0 is not valid.

```
if (average>60)
{
  cout<<"Passed";
}
else if (average<0)
{
  cout<<"Wrong Input";
}
else
{
  cout<<"Failed";
}
```

The **if/else** structure above checks the first condition, if it is satisfied, prints "Passed" and skips the rest of the structure. If the first condition is not satisfied, the second condition is checked, and so on. If none of the conditions is held, the last statement is executed.

Exercise: Sign

Write a program that decides the sign of a number.

Input: An integer.

Output: Positive, Negative or Zero.

Logical Operators

Logical operators simplify nested **if** and **if/else** structures. They combine multiple logical expressions and return a single result (True or False). There are three logical operators in C++:

- ! (Not)
- && (And)
- || (Or)

! (Logical Not) operator has only one operand (unary operator) and returns the opposite of it. **Not Operator** gives true if the operand is false, and gives false if the operand is true. For example:

```
!(5 > 7)          //evaluates to true.
!true             //evaluates to false.
```

X	!X
0	1
1	0

The Truth Table of the NOT Operator (!)

&& (Logical And) operator has two operands (binary operator). It returns true only if both operands are true, and returns false otherwise. So we may need this operator when we have to perform a task if two conditions are fulfilled at the same time. For example we want to determine if a given integer (num) is divisible by 3 and 7.

```
if ((num % 3 == 0) && (num % 7 == 0))
  cout<<"It is divisible by 3 and 7";
```

X	Y	X && Y
0	0	0
0	1	0
1	0	0
1	1	1

The Truth Table of the AND Operator (&&)

|| (Logical Or) operator has two operands. It returns false if both operands are false, and returns true otherwise. It can be used in a case if at least one of two conditions has to be true to perform a task. For example we want to check if a number is divisible by 3 or 7.

```
if ((num % 3 == 0) || (num % 7 == 0))
  cout<<"It is divisible by 3 or 7";
```

X	Y	X && Y
0	0	0
0	1	1
1	0	1
1	1	1

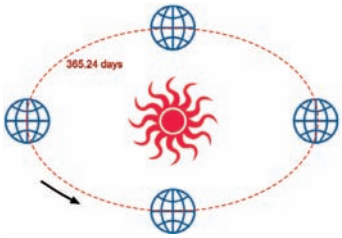
The Truth Table of the OR Operator (||)

Exercise: Letter

Write a program that checks if an entered character is between 'a' and 'z' or 'A' and 'Z'. Namely check if it is a letter or not.

Input: One character.

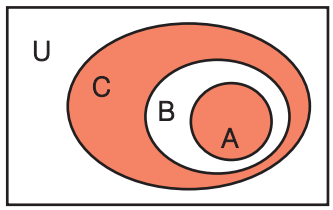
Output: "IT IS A LETTER" or "IT IS NOT A LETTER".



Leap years are needed so that the calendar is in alignment with the earth's motion around the sun.

Let's have a look how to use logical operators to determine whether a given year is a **leap year**. Leap years are years with an extra day (February 29); this happens almost every four years. Generally, leap years are divisible by four, but century years are special, they must also be divisible by 400. Given a year decide whether it is a leap year or not.

We know that the set of numbers divisible by 400 is a subset of the set of numbers divisible by 100. And the set of numbers divisible by 100 is a subset of numbers divisible by 4.



U = All of the years

C= Years divisible by 4

B= Years divisible by 100

A= Years divisible by 400

From the description it is clear that if the number is in the red area than it is a leap year otherwise it is not. We can describe the red area as (A or (C and (Not B)))

```
if ((year%400==0)||((year%4==0)&&(!(year%100==0))))
   cout << "Leap Year";
else
   cout << "Not a Leap Year";
```

Exercise: Leap Year

Implement the leap year problem without logical operators (by nested **if/else**)

Input: An integer for year value.

Output: "IT IS A LEAP YEAR" or "IT IS NOT A LEAP YEAR".

The Conditional Operator (?)

C++ provides another selective operator that can be an alternative of simple **if/else**. If we are choosing from two options based on a condition we can simply implement it with a conditional operator. Let's implement our "pass" "fail" example with "?".

With if/else	With ?

`if (average>60)`
`  cout<<"passed";`
`else`
`  cout<<"failed";`
	`cout<<((average>60) ? "passed" : "failed");`

Here condition (average > 60) is evaluated. If it is true, "passed" is written; If not, "failed' is printed.

Our leap year problem can be written as:

```
february=((year%400==0)||((year%100!=0)&&(year%4==0))) ? 29 : 28;
```

If the condition is true it is a leap year, if false it is not.

The Switch Structure

Switch allows us to select from multiple choices based on constant values. If we are to use several "if" and "else if" instructions to check constant values it is best to implement it by using switch. For example we are getting the month number from the user and printing the month name on the screen

```
/*
PROG: c2_01switch.cpp
Read the order of a month and prints its name.
*/
int main()
{
     int month;
     cout<<"enter the month number";
     cin>>month;
     switch (month) {
            case 1 :
                  cout<<"it is january";
                  break;
            case 2 :
                  cout<<"it is february";
                  break;
            case 3 :
                  cout<<"it is march";
                  break;
            case 4 :
                  cout<<"it is april";
                  break;
            case 5 :
                  cout<<"it is may";
                  break;
            case 6 :
                  cout<<"it is june";
                  break;
            case 7 :
                  cout<<"it is july";
                  break;
            case 8 :
                  cout<<"it is august";
                  break;
            case 9 :
```

```
            cout<<"it is september";
            break;
      case 10 :
            cout<<"it is october";
            break;
      case 11 :
            cout<<"it is november";
            break;
      case 12 :
            cout<<"it is december";
            break;
      default :
            cout<<"wrong input";
   }
```

Switch evaluates the value of expression and performs all of the instructions starting from the true case of the expression's value till the ending brace.

If none of the cases has the value of expression then instructions under default part are executed. You can have only one default statement in a switch statement block.

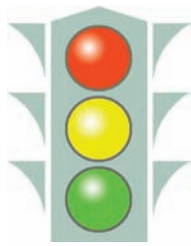
The break statement at the end of the case statement tells C++ to exit the switch statement. C++ does not generate an error message if you omit a break statement. However, if you omit it, C++ executes all the statements in the following case statement, even if that case is false. In nearly all circumstances, this is not what you want to do.

Sometimes we need to group the cases since we have the same operation for some values. This can be understood better by an example. Again we are getting the month number from the user but this time printing how many days this month has to the screen.

```
/*
PROG: c2_02months.cpp
Read the order of a month and print how many days it has.
*/
  switch (month)
  {
       case 2 :     cout<<"it has 28 days"; break;
       case 4 : case 6 : case 9 : case 11:
                    cout<<"it has 30 days"; break;
       default :    cout<<"it is 31 days";  break;
  }
```

Exercise: Traffic Lights

Implement the program which reads a character from the user as the indication of color of the traffic light, and send a comment to the user.



Input : A character indicating color of light ('r','y','g').

Output : "WAIT", "GET READY", "GO" or "WRONG INPUT".

SUMMARY

We need **decision structures** in order to give direction to the program flow.

We execute or skip a statement block by using simple **if** statement. If the condition holds then execute, if not skip the statement block.

We choose one of two options by using an **if/else** structure. If the condition holds then execute statement block, if not execute statement block 2.

C++ provides us with three kinds of **logical operators** in order to combine simple logical expressions and construct compound complex. The C++ logical operators:

Not Operator (!) is used to get opposite of one expression.

And Operator (&&) is used to get intersection of two expressions.

Or Operator (||) is used to get union area of two expressions.

Conditional Operator (? :) can be used as an alternative of simple if /else.

Several **if**, **if/ else** structures can be expressed by a **switch** structure. If we have the case that an expression may have constant values, it is best to check them by a **switch** instead of messy **if/else** components.

REVIEW QUESTIONS

1. What are the possible four outputs of the following program segment depending on the values of c1 and c2 ?

```
if (c1)
  if (c2)
    cout <<"if"<<endl;
  else
    cout <<"if/else"<<endl;
else if (!c2)
    cout <<"?"<<endl;
else
    cout <<"switch"<<endl;
```

c1	c2	Output
True	True	if
True	False	
False	True	
False	False	

2. The given code segment copies the bigger of two variables to the third one. Rewrite it by using if/else.

With '?'	With 'if/else'
`c = (a>b) ? a : b;`	

3. What is the final value of x?

```
int x=1;
if (x >= 0)       x += 5;
if (x >=5)        x += 2;
else x*=3;
```

4. We have the following code segment:

```
if ( weight+110 < height )
   cout << "normal weight" << endl;
else
   cout << "over weight" << endl;
```

Which of the following possible values for weight and height cause the message "over weight" to be printed?

I. weight = 50, height = 155
II. weight = 80, height = 192
III. weight = 25, height = 125

a) I and II
b) II and III
c) I and III
d) I, II and III

5. Which results do logical expressions evaluate with given values.

a	b	c	Expression	Result
5	3	1	c && (a>b)	true
0	4	4	(b==c) || a	
0	5	4	((!a) && (b!=c)) && (b <= c)	
4	2	3	(a > b) && (c < a)	

6. The ranges of marks in a school as follows :

0	20	**F**
21	40	**D**
41	60	**C**
61	80	**B**
81	100	**A**

Does the following program segment automate the assessment of marks? If not, correct the program.

```
if (avg > 0)
   mark = F;
else if (avg > 20)
   mark = D;
else if (avg > 40)
   mark = C;
else if (avg > 60)
   mark = B;
else if (avg > 80)
   mark = A;
```

PROGRAMMING PROBLEMS

1. (2nd Degree Equation) Make a program that reads three integers (a, b, c) from a user as coefficients of a quadratic equation and calculate the solution set (x1, x2) of the equation.

   For $ax^2+bx+c=0$

   delta = b^2-4ac

   x1= (- b + sqrt (delta)) /2a

   x2= (- b - sqrt (delta)) /2a)

   C++ provides us with **sqrt()** function to perform square root operations. It is defined under <**math.h**> library. e.g. `x=sqrt(y)` x becomes square root of y. **Sqrt** funtion returns a value type of **double**.

sample inputs	sample outputs
1 3 -10	x1=2 and x2= -5
1 -4 4	x1=2 and x2=2
1 -2 4	No real roots

2. (Mini Calculator) Write a program which reads two integers as operands and a character as an operator between them, and prints the result.

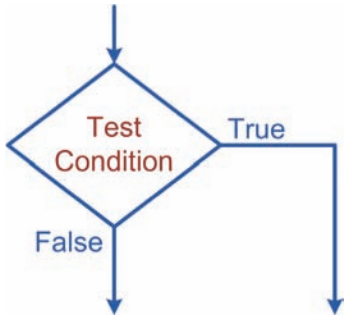
sample inputs	sample outputs
3 * 7	21
5 - 12	-7
32767 / 1024	31

3. (Character Recognizer) Write a program that reads a character from the user and tells if it is a letter ('a'..'.z', 'A'…'Z'), digit ('1'…'9') punctuation mark ('.','?',',','!' etc.) or special character (other than these).

sample inputs	sample outputs
a	letter
5	digit
?	punctuation mark
/	special character

FLOWCHART PROGRAMMING (OPTIONAL)

Making A Decision

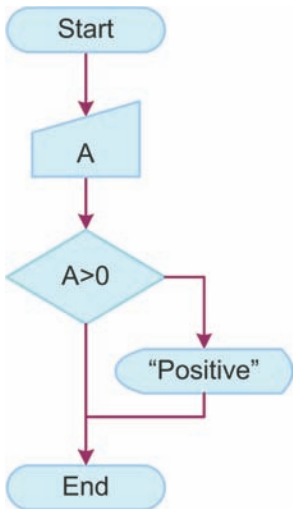


The diamond-shaped symbol is used to make a decision in flowchart programming. This symbol is unique in that it has two arrows coming out of it, from the right point and bottom point, one corresponding to **Yes** or **True**, and one corresponding to **No** or **False**.

Example: Positive

Make a flowchart that gets an integer and then prints "Positive" if the integer is bigger than zero.

This is a **single selection** problem. That is how we should determine whether to print "Positive". If the test condition A>0 is satisfied the output symbol is executed, otherwise the program simply finishes.



FCPro Relational and Equality Operators:

>	:	Greater
>=	:	Greater or Equal
<	:	Smaller
<=	:	Smaller or Equal
==	:	Equal
!=	:	Not Equal

Exercise: Sum

Make a flowchart that gets two numbers (A and B), and then prints the sum of these numbers (A+B) if the first number is bigger than the second number (A>B).

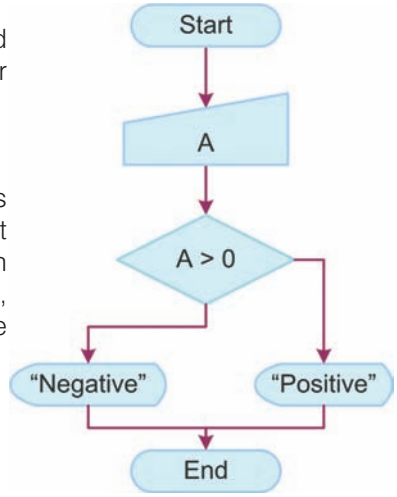
input : Two numbers: A and B

output : A + B if A>B

Example: Negative or Positive

Make a flowchart that gets an integer and then prints "Positive" if the integer is bigger than zero, and prints "Negative" otherwise.

This is a **double selection** problem. That is we should determine whether to print "Positive" or "Negative". If the test condition A>0 is satisfied "Positive" will be printed, otherwise "Negative" will be printed and the program finishes.



Exercise: Positive, Negative or Zero

Make a flowchart that gets an integer and then prints "Positive" if the integer is bigger than zero, and prints "Negative" if the number is smaller than zero, or prints "Zero" otherwise.

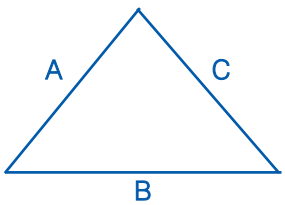
input : A number : A

output : "Positive" if A is bigger than zero
"Negative" if A is less than zero
"Zero" if A is equal to zero

FCPro Logical Operators:

! : Not
&& : And
|| : Or

Exercise: Triangle

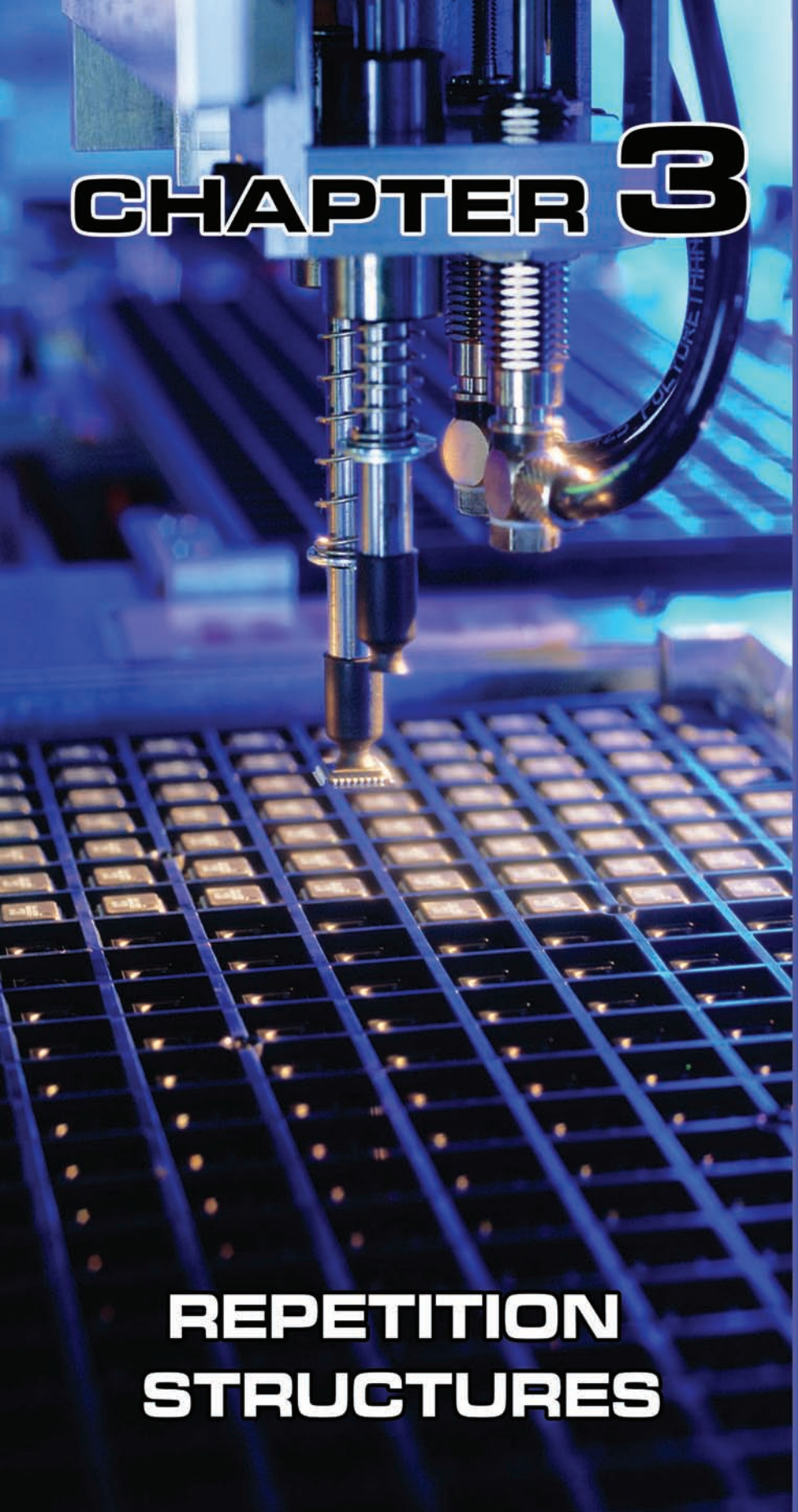


In a triangle, sum of any two sides is always larger than the third side.

Make a flowchart that gets three numbers and then determines if these three numbers represent the length of all sides of a triangle.

input : Three numbers : A, B and C

output : "Triangle" if A, B and C can represent sides of a triangle.
"Not Triangle" if A, B and C cannot represent sides of a triangle.



CHAPTER 3

REPETITION STRUCTURES

- The While Loop
- Increment and Decrement Operators
- Counter-Controlled and Sentinel-Controlled Loops
- The Do/While Loop
- The For Loop
- Break and continue Statements
- Nested Loops

C++

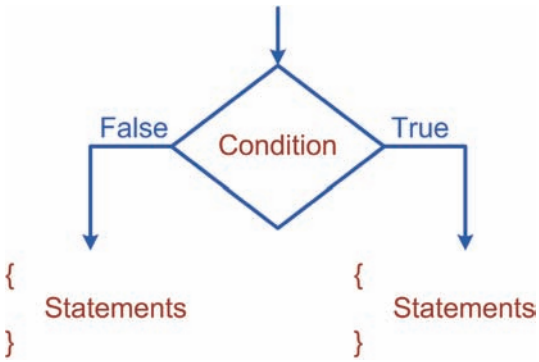
Repetition Structures (Loops)

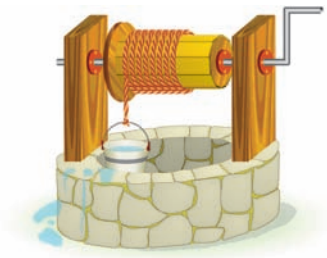
Any computer program is executed sequentially if the programmer doesn't change the flow of the program. The sequential programs start from the first line of the program code and execute all the statements one by one until the end of the program.

Making decisions is a feature granted to human beings.

There are two kinds of program flow controls: **decision structures** and **repetition structures**.

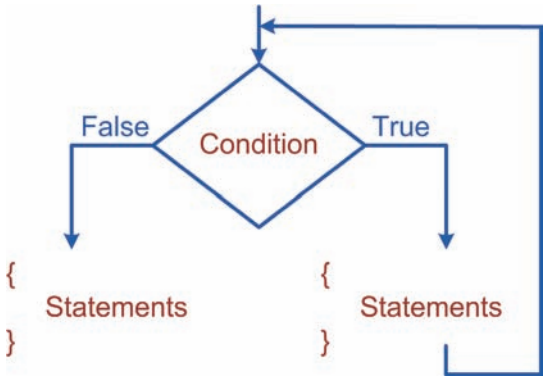
Decision structures are used to alter the normal flow of program execution based on an evaluation of one or more logical expressions (condition). We have already studied decision structures in the previous chapter.





Loops are everywhere.

Repetition structures are used to repeat a block of code either a specific number of times or while some condition remains true. For example, "Read 50 items from the input file" or "While there are more items in the input file, continue reading the input."



Executing a block of code is one of the most basic but useful tasks in programming. Many programs or Web sites that produce complex output are really only executing a single task many times.

There are three kinds of repetition structures in C++:

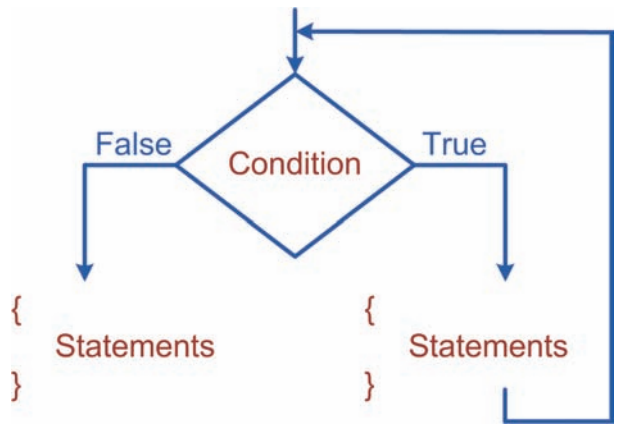
- **While**
- **Do While**
- **For**

The "while" Loop

The **"while loop"** is the easiest repetition structure to understand for beginners. Its structure suits the most to the description stated above.

The structure is made up of a logical condition and the block of code to be repeated. First, the condition is evaluated, if it is true the statements in the block of the code are executed, and the condition is evaluated one more time. This process continues until the condition becomes false. The condition must become false at the end; otherwise we can fall into an infinitive loop. The **while loop** first checks the condition and then decides whether executes the statements therefore, it is called a **pre-conditional** repetition structure.

Be careful with **infinitive loops**. An infinitive loop executes forever without terminating. Usually incorrectly setting the termination condition or incorrectly increasing or decreasing the loop-control variables (counters) causes an infinitive loop.



Let's make a basic program to understand the **while loop**. The following program prints the numbers from 1 to 10. It prints the value of a variable (that is a counter) one at a time, and repeats this process ten times. To print the numbers from 1 to 10, the variable counter is initialized to one, and its value is increased by 1 in each iteration. On the other hand, the condition "`counter <= 10`" checks if the counter exceeds its final value.

The key concepts of looping are, the **counter**, the **initial value** of the counter, the **final value** of the counter, **increment or decrement factor** of the counter, the **condition** and the **statements** to be executed.

The variable "`counter`" is our counter in this program. Since it gets only the whole numbers from 1 to 10, its type is integer. The initial value of counter is 1, the final value of counter is 10, and the increment factor of the counter is 1.

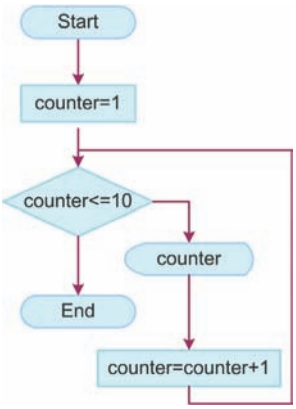
"`counter <=10`" is the condition of the loop. The **while loop** checks the condition at the header of the structure. This condition states that the statement will be executed while the counter is not bigger than 10.

"`cout <<counter<<" ";`" is the statement that is executed each time in the looping process. The purpose of the program is to print the consecutive numbers from 1 to 10 thus the value of the counter must get those values one by one in each turn.

The following three statements are all the same:

```
c++;
c = c+1;
c += 1;
```

"**counter++**" increases the value of counter by one. This statement can be written as "**counter = counter + 1**". C++ used this syntax at the first time. Usually programmers use only 'c' instead of "counter" as a variable name, so "**c++**" means increase the value of a counter "c" by one. I think, this is where the name of this programming language is comes from.



Flowchart of the Program "while"

```
/*
PROG: C3_01while.cpp
Printing the numbers from 1 to 10 increasing by 1.
*/
#include <iostream>
using namespace std;
int main()
{
    int counter = 1;            //initialize the counter to its
                                //starting value
    while (counter <= 10)       //continue looping until counter
                                //is less than its final value
    {
        cout <<counter<<" ";    //print the current value of the
                                //counter
        counter++;              //counter gets its next value
    }
    system("PAUSE"); return 0;
}
```

1 2 3 4 5 6 7 8 9 10 Press any key to continue . . .

Increment and Decrement Operators

C++ provides unary **increment** (++) and **decrement** (--) operators. Both operators can be used after (a++, post-increment) or before (++a, pre-increment) their operands. If the variable 'a' needs to be incremented by 1, "a++", "a=a+1", or "a+=1" can be used. The following program demonstrates the use of **increment** and **decrement** operators.

```
/*
PROG: C3_02inc_dec.cpp
Demonstrating increment and decrement operators
*/
#include <iostream>
using namespace std;

int main()
{
  int a=5;
  cout <<"initial a is "<<a<<endl;
  cout <<"after a++ is "<<a++<<endl;
  cout <<"current value of a is "<<a<<endl;
  cout <<"after ++a is "<<++a<<endl;
```

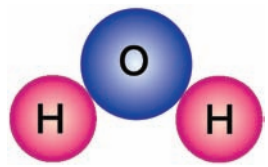
```
  cout <<"current value of a is "<<a<<endl;
  cout <<"after --a is "<<--a<<endl;

  system("pause");  return 0;
}
```

```
initial a is 5
after a++ is 5
current value of a is 6
after ++a is 7
current value of a is 7
after --a is 6
Press any key to continue . . .
```

Example: Molecules

Calculate the mass of a molecule. A molecule is a chemical structure that is made up of more then one atom. H_2O is the molecule of water. It contains two hydrogen and one oxygen atoms. The weight of hydrogen is 1, and the weight of oxygen is 16. Thus the weight of water molecule is 2*1 + 16*1 = 18.

The first line of the input contains an integer (N) that denotes the number of the atoms in the molecule. Each of the following N lines denotes an atom of the molecule with two integers. The first integer is the mass of the atom, and the second integer is the quantity of the atom.



A Water (H_2O) Molecule

chem.in

```
2
1 2
16 1
```

chem.out

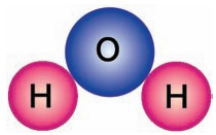
```
18
```

```
/*
PROG: c3_03molecules.cpp
*/
#include <fstream>
using namespace std;

int main()
{
  int n, weight, quantity;
  int sum=0, counter=1;

  ifstream fin("chem.in");
  ofstream fout("chem.out");
```

```
fin >> n;          //How many kinds of atoms?

while (counter <= n)
{
  fin >> weight >> quantity;
  sum += weight*quantity;       //add the weight for the new
                                //kind of atom.
  counter++;
}

fout << sum <<endl;

fin.close();
fout.close();

system("pause");
return 0;
```

Exercise: Molecules

Modify the program “Molecules” so that it calculates the weight of a molecule without using any variable as counter.



Exercise: Train

A train leaves the first station with N passengers and visits K stations before it arrives at the last station. In each station some passengers got off the train and some passengers got on the train. Everybody in the train got off at the last station. Make a program that calculates how many passengers got off the train at the last station.

The first line of the input has two integers N and K. Each of the following K lines contains two integers, the first one denotes the number of passengers who got off the train at that station, and the second one denotes the number of passengers got on the train at that station. The output should hava a single integer that is number of passengers who got off the train at the last station.

Input file: train.in

output file: train.out

Sample input	Sample output
5 20 6 15 5 30 20 12 15 8 6 7	40

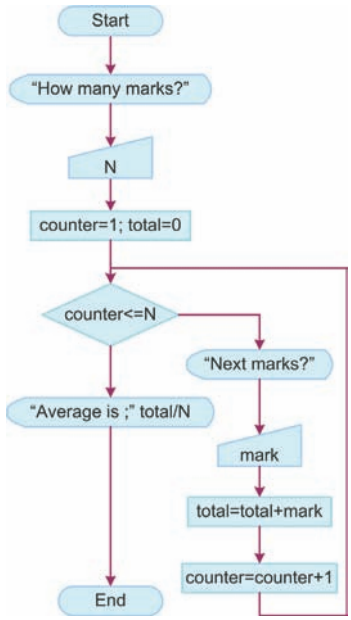
Counter-Controlled and Sentinel-Controlled Repetitions

Counter-Controlled Repetition

Counter-controlled repetition uses a variable called a "counter" to control the number of times the statements will be executed. The variable counter must be initialized before the repetition and must be incremented or decremented during the repetition process. **Counter-controlled repetition** is called "**definite repetition**" because the number of repetitions is known before the loop begins executing.

The following program calculates a class average with a counter-controlled loop.

```
/*
PROG: c3_04countercont.cpp
Class average with a counter-controlled loop. Make a program to
calculate average of a class after an exam. There are N students in
the classroom and the marks are between 1 and 5.
*/
#include <iostream>
using namespace std;

int main()
{
  int N, total, mark, counter;

  //Get number of the marks.
  cout<<"How many marks?";
  cin >> N;
  //Initialize the counter and total.
  counter = 1;
  total = 0;

  //Get the mark one by one and add to total.
  while (counter <= N)
  {
    //Get the next mark.
    cout <<"Enter the next mark [1..5]: ";
    cin >> mark;
    total+=mark;          //Add the new mark to total.
```



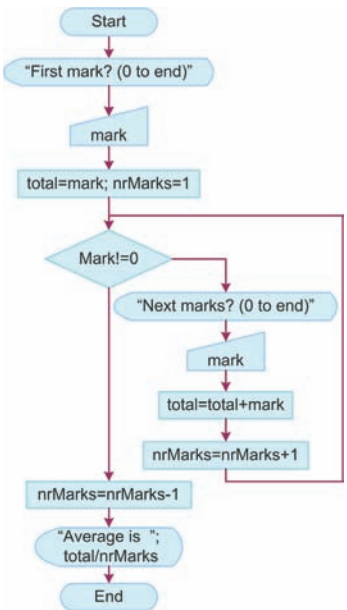
Flowchart of the Program "countercont"

```
    counter++;                          //Increment the counter.
  }

  float average = (float)total/N;  //Calculate the average.

  cout<<"The average is "<<average<<endl;

  system ("pause");
  return 0;
}
```

```
How many marks?5
Enter the next mark [1..5]: 4
Enter the next mark [1..5]: 5
Enter the next mark [1..5]: 3
Enter the next mark [1..5]: 4
Enter the next mark [1..5]: 5
The average is 4.2
Press any key to continue . . .
```

Sentinel-Controlled Repetition

In the case when users don't know the number of the repetitions in advance, a **sentinel-controlled repetition** can be used instead of a counter-controlled repetition. The idea of a **sentinel controlled repetition** is that there is a special value (the "**sentinel**") that is used to say when the loop is done. The sentinel value must be chosen carefully so that it cannot be confused with a legitimate value. **Sentinel-controlled repetition** is called "**indefinite repetition**" because the number of repetitions is not known in advance.



Flowchart of the Program "sentinelcont"

```
/*
PROG: c3_05sentinelcont.cpp
Class average with a sentinel-controlled loop. Make a program to
calculate average of a class after an exam. Your program will
process an arbitrary number of the marks. The marks are between 1
and 5. Use 0 as sentinel value to end the program execution.
*/
#include <iostream>
using namespace std;

int main()
{
  int total, mark, nrMarks;
  //Get the first mark and set the nrMarks to 1.
  cout<<"Enter the first mark: ";
  cin >> mark;
  total = mark;
  nrMarks = 1;      //We have got one mark so far.

  //Get the rest of the marks one by one and add to total.
  while (mark != 0) //Continue if mark is not sentinel
```

```
  {
    //Get the next mark.
    cout <<"Enter the next mark [1..5] to finish enter 0: ";
    cin >> mark;
    total+=mark;          //Add the new mark to total.
    nrMarks++;            //Increment the counter.
  }

  nrMarks--;              //Decrease for the sentinel.
  float average = (float)total/nrMarks; //Calculate the average.
  cout<<"The average is "<<average<<endl;
  system ("pause");
  return 0;
}
```

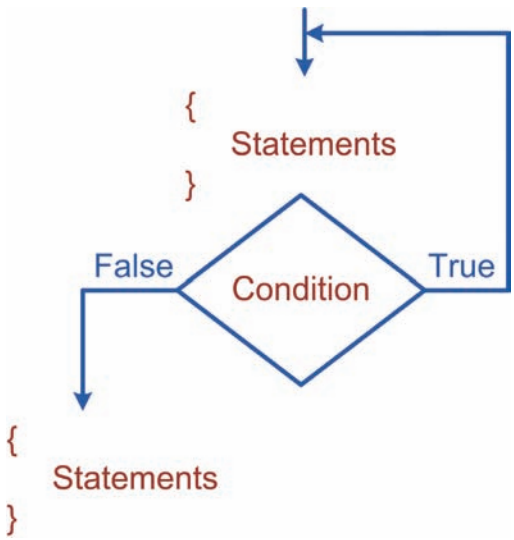
```
Enter the first mark [1..5] to finish enter 0: 5
Enter the next mark [1..5] to finish enter 0: 3
Enter the next mark [1..5] to finish enter 0: 4
Enter the next mark [1..5] to finish enter 0: 5
Enter the next mark [1..5] to finish enter 0: 3
Enter the next mark [1..5] to finish enter 0: 5
Enter the next mark [1..5] to finish enter 0: 0
The average is 4.16667
Press any key to continue . . .
```

Use **setprecision** function to set the precision for floating-point values. Setprecision function is defined in the iomanip library.

```
#include <iomanip>
.
.
.
cout<<"The average is"
<<setprecision(3)
<<average<<endl;
.
.
.
```

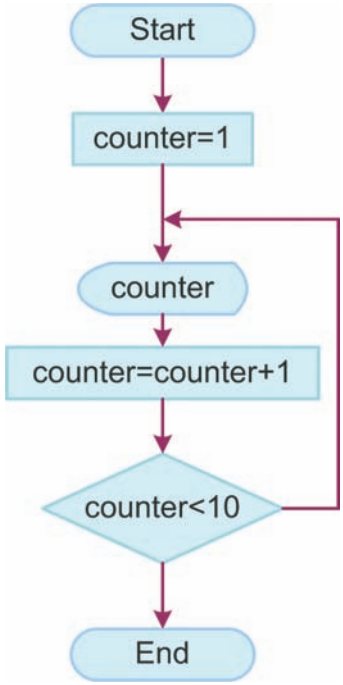
The "do/while" Loop

The "do/while" statement is similar to the "while" statement with an important difference: the "do/while" statement performs a test after each execution of the loop body. Therefore, the loop body is executed at least once. The "do/while" statement is usually preferred to design menu-driven programs.



A **menu-driven program** has an interface that offers the user a simple menu from which to choose an option. The opposite of menu-driven is the **command-driven.** There usually exists one option to exit in the menu-driven programs.

```
Make your choice:
1. New record
2. Modify
3. Delete
4. Print
5. Exit
```

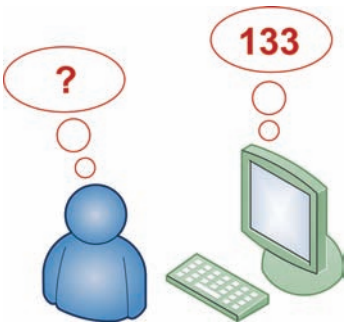


Flowchart of the Program "dowhile"

```
/*
PROG: c3_06dowhile.cpp
Printing the numbers from 1 to 10 increasing by 1.
*/

#include <iostream>
using namespace std;
int main()
{
     int counter = 1;        //initialize the counter to its
                             //starting value

  do
    {
          cout <<counter<<" ";  //print the current value of the
                                //counter
          counter++;            //counter gets its next value
    }
    while (counter <= 10);      //continue looping until counter
                                //is bigger than the final
                                //value.
    system("PAUSE");
    return 0;
}
```

1 2 3 4 5 6 7 8 9 10 Press any key to continue . . .

Exercise: Guess the number



The **rand** function returns a pseudorandom integer in the range 0 to RAND_MAX (32767).

The game "guess the number" is played between two players. The first player selects an integer at random in the range 1 to 1000. The second player makes his first guess. The first player responds "Too low", "Too high", or "You guessed the number.". The game goes on until the second player guesses the number correctly.

Make a program that plays the game "guess the number" as the first player. Add the following code to your program to generate a random integer between 1 and 1000.

```
#include <time.h>
int main()
{
    srand( (unsigned)time( NULL ) );
    int randomNumber = rand()%1001 + 1;
.
.
.
```

The "for" Loop

Like "while" and "do/while" the "for" loop enables you to evaluate a sequence of expressions multiple numbers of times. For loops are best when you know the number of times that the expressions are needed to be evaluated in advance (counter-controlled repetition).

Syntax of “for” statement

```
for( initializations; condition; increment or decrement)
{
  Statements;
}
```

for: keyword

Initializations: You may initialize the counter and other variables here. This part is executed only once at the beginning.

Condition: Set the continuation-condition here to continue or end the loop. The condition is tested each time before the statements are executed.

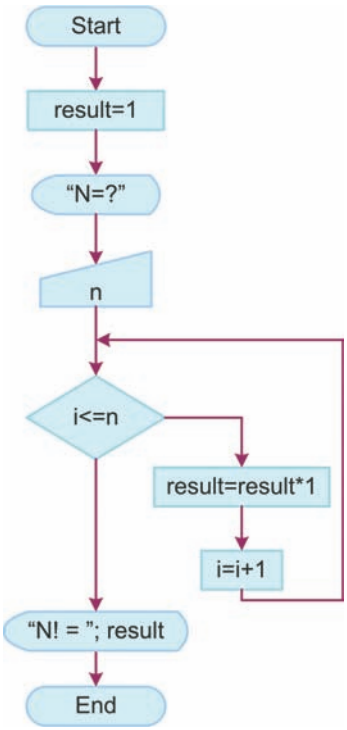
Increment or decrement: You may increment or decrement the counter here. This part is executed each time after the statements have been executed.

Statements: The statements are executed each time in the loop.

```
/*
PROG: c3_07for.cpp
Printing the numbers from 1 to 10 increasing by 1.
*/

#include <iostream>
using namespace std;
int main()
{
    int counter;
    for (counter=1; counter<=10; counter++)
    {
      cout<<counter<<" ";
    }

    system("PAUSE");
    return 0;
}
```

1 2 3 4 5 6 7 8 9 10 Press any key to continue . . .

Example: Factorial



Flowchart of the Program "factorial"

The **factorial** of a non-negative integer n is the product of the positive integers less than or equal to n. This is written as "n!", and pronounced "n factorial". 0!=1, 1!=1, 2!=2, 3!=6, 4!=24, 5!=120. The following program reads a non-negative integer from the standard input and computes its **factorial.**

```
/*
PROG: c3_08factorial.cpp
*/
#include<iostream>
using namespace std;

int main()
{
  int i, n;
  long long result=1;      //Initialize result for n=0 and n=1
                           //Factorial of numbers rapidly grow
                           //very large.
  cout<<"N=?";
  cin>>n;

  for(i=2; i<=n; i++)
    result *= i;           //result = result*i

  cout<<"N! = "<<result<<endl;

  system("pause");  return 0;
}
```

```
N=?11
N! = 39916800
Press any key to continue . . .
```

Example: X to the power of Y

The following program computes X raised to the power Y (X^Y) where X and Y are both integer numbers and Y is non-negative. X is the base (mantissa) and Y is the exponent value.

To get the result we have to multiply X, Y times. For example for `X=5` and `Y=3`, the result will be 5*5*5. In fact, C++ provides a built-in math "power" function (**double pow(double x, double y)**) that calculates power of integer or floating point numbers. We are going to use it in the chapter "Functions" in this book.

```
 /*
PROG: c3_09power.cpp
*/
#include <iostream>
using namespace std;

int main()
{
  int base, power;
  long long result = 1;    //Initial value for power = 0;

  cout <<"Enter the base and the power?";
  cin >> base >> power;    //Get the input

  //Calculate the result. Multiply base itself power times.
  for (int i=1; i<=power; i++)
    result *= base;

  //Print the result
  cout <<"Result is "<<result<<endl;

  system("pause");  return 0;
}
```

```
Enter the base and the power?4 3
Result is 64
Press any key to continue . . .
```

Example: Fibonacci Series

The Fibonacci series begins with 0 and 1 and has the property that each subsequent Fibonacci number is the sum of the previous two Fibonacci numbers. Some of the first Fibonacci numbers are 0, 1, 1, 2, 3, 5, 8, 13, …

Make a program to print the first N Fibonacci numbers where N is a positive integer.

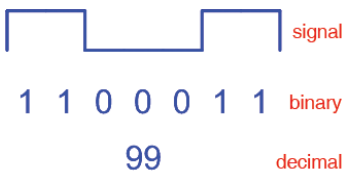
Leonardo Fibonacci (1170 - 1250) introduced to Europe and popularized the Hindu-Arabic number system (also called the decimal system). He contributed greatly to number theory.

```
/*
PROG: c3_10fibonacci.cpp
*/
#include <iostream>
using namespace std;

int main()
{
  int N;
  int currentNum=1;        //The first Fibonacci number
```

```
  int previousNum=0;

  cout <<"N =? ";
  cin >> N;

  cout<<previousNum<<" "; //Print the first Fibonacci number

  /*Set a loop to print the rest of the first N Fibonacci
  numbers*/
  for (int i=1; i<N; i++)
  {
    cout<<currentNum<<" ";       //Print the current Fibonacci
                                 //number
    currentNum += previousNum;   //CurrentNum gets the value
                                 //of the next Fib. number
    //previousNum gets the old value of currentNum.
    previousNum = currentNum - previousNum;
  }

  system("pause");
  return 0;
}
```

```
N =? 10
0 1 1 2 3 5 8 13 21 34 Press any key to continue . . .
```

Example: Binary to Decimal



Binary is a number system used by digital devices like computers. The computer represents values using two voltage levels (usually 0V and +5V). With two levels we can represent exactly two different values. These could be any two different values, but by convention we use the values zero (0) and one (1).

To convert **binary** into **decimal** is very simple. Just like the decimal system, we multiply each digit by its weighted position, and add each of the weighted values together. For example, the binary value 10011101 represents:

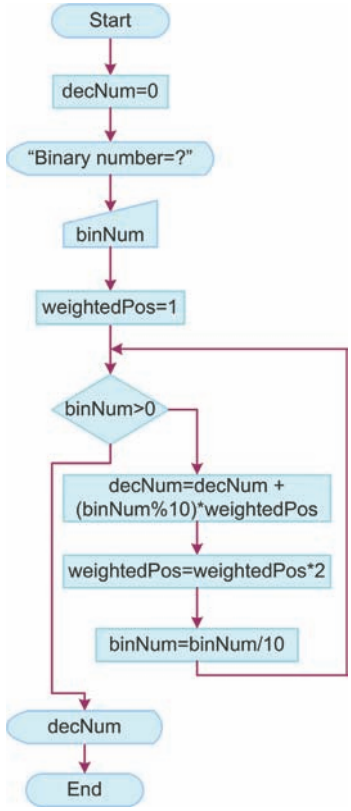
$1*2^7 + 0*2^6 + 0*2^5 + 1*2^4 + 1*2^3 + 1*2^2 + 0*2^1 + 1*2^0 = 157.$

Make a program that gets a binary number and prints its decimal equivalent as output.

```
/*
PROG: c3_11bintodec.cpp
*/
#include <iostream>
using namespace std;
```

```
int main()
{
  long long binNumber, decNumber = 0;

  cout<<"Enter the binary number:";
  cin >> binNumber;

  int weightedPos = 1;     //The weighted position of the
                           //right-most digit
  while (binNumber > 0)    //We have some more digits to
                           //process
  {
    /*Multiply the right-most digit by its weighted position
    and add the product to decNumber.*/
    decNumber += (binNumber%10)*weightedPos;
    weightedPos *= 2;      //The weighted position of the next
                           //digit
    binNumber /= 10;       //Cut out the right-most digit of
                           //binNumber.
  }

  cout<<"Decimal equivalent is "<<decNumber<<endl;
  system("pause");
  return 0;
}
```

```
Enter the binary number:1110011
Decimal equivalent is 115
Press any key to continue . . .
```



Flowchart of the Program "bintodec"

Exercise: Decimal to Binary

Make a program that gets a **decimal** number and prints its **binary** equivalent as output.

To convert a **decimal** number to its **binary** equivalent you can use the "**short division by 2 remainder**" method. This method relies only on division by two.

For this example, let's convert the decimal number 156 to binary. Write the decimal number as the dividend above the "long division" symbol.

156

Divide the dividend by 2 and write the integer answer (quotient) under the long division symbol, and write the remainder (0 or 1) to the right of the dividend.

156 | 0
78

Continue downward, dividing each new quotient by two and writing the remainders to the right of each dividend. Stop when the quotient is 0.

156	0
78	0
39	1
19	1
9	1
4	0
2	0
1	1
0	

Starting with the bottom-most remainder 1, read the sequence of 1's and 0's upwards to the top. You should have 10011100. This is the binary equivalent of the decimal number 156.

Example: Maximum

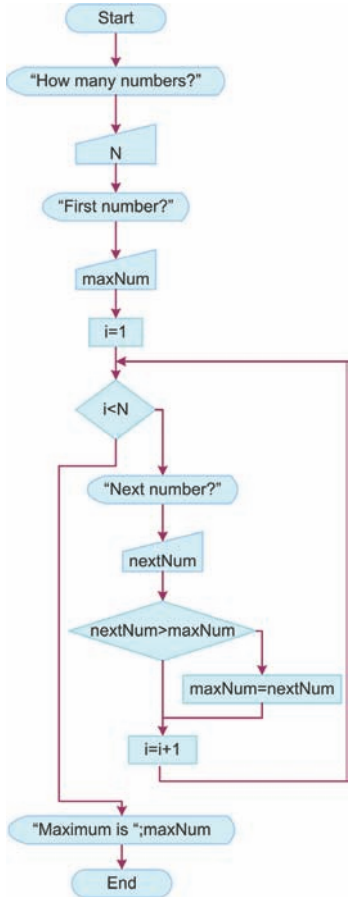
You are given the list of N integers. Make a program to find the **maximum** item in the list. Finding the **maximum** and **minimum** items is a fundamental algorithm in programming and has many applications. The algorithm is as follow:

Keep the first item as the **maximum**, and compare each item with the **maximum**. If the new value is bigger than the **maximum**, **maximum** gets the value of the new item.

```
/*
PROG: c3_12max.cpp
*/
#include <iostream>
using namespace std;

int main()
{
  int N, maxNum, nextNum;
  cout<<"How many numbers?";
  cin >> N;

  cout <<"Enter the first number:";
  cin >> maxNum;    //Suppose the first number is maximum.
```

```
  for (int i=1; i<N; i++)
  {
       cout <<"Enter the next number:";
       cin >> nextNum;
       if (nextNum > maxNum)      //Compare the new number with
                                  //maxNum
              maxNum = nextNum;   //newNumber is bigger
  }

  cout << "Maximum number is "<<maxNum<<endl;
  system("pause");
  return 0;
}
```

```
How many numbers?5
Enter the next number:8
Enter the next number:-15
Enter the next number:13
Enter the next number:7
Enter the next number:9
Maximum number is 13
Press any key to continue . . .
```

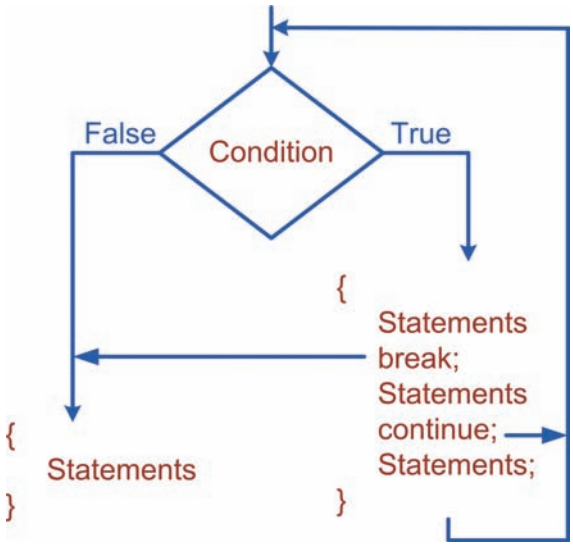


Flowchart of the Program "max"

The "break" and "continue" Statements

The **break** statement stops the loop in which it resides. The program continues with the next statement immediately following the loop.

The **continue** statement causes the program to skip the rest of the loop in the current iteration. The program continues with the start of the following iteration.



The following program demonstrates the **break** and **continue** statements. The continue statement causes the program skip printing 5, and the break statement causes the program to break the loop when the counter is 10.

```
/*
PROG: c3_13breakcon.cpp
*/

#include <iostream>
using namespace std;

int main()
{
  for(int counter=1; counter<100; counter++)
  {
    if (counter == 5)
       continue;            //skip 5
    if (counter == 10)
       break;               //stop the looping process
    cout<<counter<<" ";
  }
  system("pause");
  return 0;
}
```

```
1 2 3 4 6 7 8 9 Press any key to continue . . .
```

Which loop should I use?

Loop Type	Description
while	This is a good, solid looping process with applications to numerous situations.
do/while	This looping process is a good choice when you are asking a question, whose answer will determine if the loop is repeated.
for	This loop is a good choice when the number of repetitions is known, or can be supplied by the user.

Nested Loops

The number of digits in the web page counter determines the number of nested loops needed to imitate the process. How many loops do you need for the following counter?

0 1 2 3 4 5 6 7 8 9

Loops can be nested inside other loops. A **nested loop** is a loop within a loop, an inner loop within the body of an outer one.

The way a **nested loop** works is that the first iteration of the outer loop triggers the inner loop, which executes to completion. Then the second iteration of the outer loop triggers the inner loop again. This repeats until the outer loop finishes.

The following program prints a number triangle to demonstrate a **nested loop**. The

outer loop determines the number of the lines in the triangle and the last number in the current line, and the inner loop prints a line.

```
/*
PROG: c3_14nested.cpp
Printing a number triangle
*/
#include <iostream>
using namespace std;

int main()
{
  int N;

  cout <<"How many lines in the triangle?";
  cin >> N;

  for (int i=1; i<=N; i++)        //Outer loop
  {
    for (int j=1; j<=i; j++)      //Inner loop
    {
      cout<<j<<" ";//Print the current number
    }
    cout<<endl;                   //Go to the next line
  }
  system("pause");
  return 0;
}
```

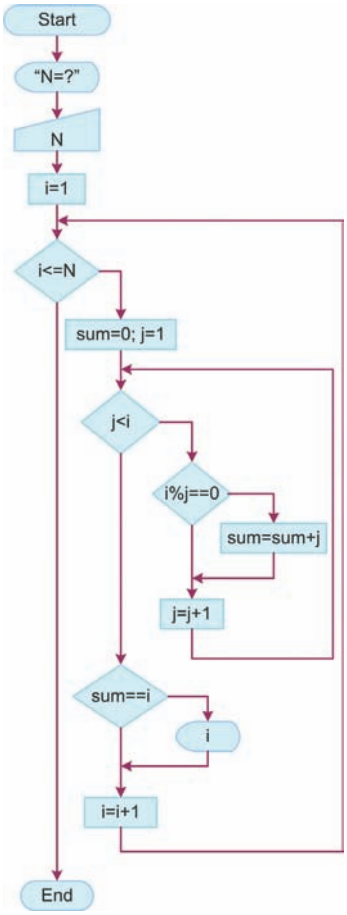
Be careful choosing the counters in a nested loop. The Outer and inner loops must have different counters in such a structure.

```
How many lines in the triangle?5
1
1 2
1 2 3
1 2 3 4
1 2 3 4 5
Press any key to continue . . .
```

Exercise: Perfect Numbers

A **perfect number** is a whole number, an integer greater than zero and when you add up all of the factors less than that number, you get that number. The first two **perfect numbers** are 6 and 28. 1+2+3 = 6, and 1+2+4+7+14 = 28. There are 37 known **perfect numbers**.

Make a program that finds and prints all the prime numbers less than or equal to a given number N.



Flowchart of the Program "perfect"

```
 /*
PROG: c3_15perfect.cpp
Print all perfect numbers in the range [1..N]
*/
#include <iostream>
using namespace std;

int main()
{
  int N;
  cout <<"N=?";
  cin >> N;

  for (int i=1; i<=N; i++)        //i is the next number to be
                                  //checked
  {
    int sum = 0;              //Accumulate the factors in sum
    for (int j=1; j<i; j++)
    {
      if (i%j == 0)           //j is a factor of i
            sum += j;
    }
    if (sum == i)             //We have a perfect number.
      cout << i <<" ";
  }
  system("pause");
  return 0;
}
```

```
N=?10000
6 28 496 8128 Press any key to continue . . .
```

SUMMARY

Repetition structures execute a series of statements multiple times until a certain condition is met.C++ provides three sorts of repetition structures: "**while**", "**do/while**", and "**for**".

"while" and **"for"** are **pre-conditional** loops, whereas **"do/while"** is a **post-conditional** loop. The test condition determines if the loop process continues. Repetition structures can be **counter-controlled** or **sentinel-controlled** depending on the test condition.

It is often necessary to handle exception conditions within loops. The statements **break** and **continue** are used for such cases. The **break** statement is used to exit the current loop before its normal ending. The **continue** statement causes the current iteration to be skipped.

The placing of one loop inside the body of another loop is called **nesting**. In a **nested loop**, the **outer loop** takes control of the number of complete repetitions of the **inner loop**.

REVIEW QUESTIONS

1. Which structure repeats a block of code?
   a) decision
   b) loop
   c) cycle
   d) continue

2. Which of the following is not a repetition structure?
   a) while
   b) do/while
   c) for
   d) switch

3. Which statement causes the program execution to skip an iteration of a loop and continue with the next iteration?
   a) continue
   b) break
   c) if
   d) return

4. In which of the following structure canyou not use a **break** statement to continue the execution of the program after the structure?
   a) switch
   b) while
   c) if/else
   d) for

5. How are the decision and repetition structures called in programming?
   a) loop
   b) sequential
   c) control
   d) basic

6. What is the output of the following program?

```
#include <iostream>
using namespace std;
int main()
{
    for (char ch1='A'; ch1<='D'; ch1++)
    {
      for(char ch2='D';ch2>=ch1; ch2--)
          cout <<ch2<<" ";
      cout<<endl;
    }
    return 0;
}
```

7. What is the output of the following program?

```
#include <iostream>
using namespace std;

int main()
{
    int result,a,b;
    result = 0;
    a = 10;
    while (a>0)
    {
          b=1;
          do
          {
                a -= b;
                result += b;
                b *= 2;
          }
          while(b<a);
          result += a;
    }
    cout <<a<<" "<<b<<" "<<result<<endl;
    system("pause");
```

PROGRAMMING PROBLEMS

1. **(Wonder Primes)** A **wonder prime** is a number that can be partitioned into two prime numbers, each of which has at least D digits and, of course, doesn't start with 0. When D=2, the number 11329 is a **wonder prime** (since it connects 113 and 29, both of which are prime). Your job is to find the first **wonder prime** greater than or equal to a supplied integer N when you are given D.

   Sample Input : 2 11328

   Sample output : 11329

2. **(Round Numbers)** A positive integer N is said to be a **"round number"** if the binary representation of N has as many or more zeroes as it has ones. For example, the integer 9, when written in binary form, is 1001. 1001 has two zeroes and two ones; thus, 9 is a **round number**. The integer 26 is 11010 in binary; since it has two zeroes and three ones, it is not a **round number**.

   Write a program that tells how many **round numbers** appear in the inclusive range given by the input.

   Sample input : 2 12

   Sample output : 6

   Output Details:

2	10	1x0 + 1x1	round
3	11	0x0 + 2x1	not round
4	100	2x0 + 1x1	round
5	101	1x0 + 2x1	not round
6	110	1x0 + 2x1	not round
7	111	0x0 + 3x1	not round
8	1000	3x0 + 1x1	round
9	1001	2x0 + 2x1	round
10	1010	2x0 + 2x1	round
11	1011	1x0 + 3x1	not round
12	1100	2x0 + 2x1	round

3. **(Tower of Happiness)** The **tower of happiness** is located in the garden of effort. Happiness is waiting for you in the top chamber of the tower. The stairs that lead you to the chamber has N steps and you are allowed to climb up one or two steps from your current location. The door of the chamber will be open only if you could climb the steps in a secret combination. Of course, finding this secret combination is a matter of chance. For instance, you may choose one of the three ways to climb a three-step stairs: 0 1 2 3, or 0 1 3, or 0 2 3.

   Make a program to determine how many different ways you can climb up the **tower of happiness** when you are given the number of steps. You must start from the ground which is step 0. The higher the tower is, the smaller the chance of catching the happiness gets.

   Sample input : 3

   Sample output : 3

4. **(Consanguineous)** Each person's blood has two markers called ABO alleles. Each of these markers is represented by a character chosen from the set {A, B, O}. This gives six possible combinations: AA, AB, AO, BB, BO, and OO. The ABO blood type for people with these combinations of alleles are A, AB, A, B, B, and O, respectively.

   Blood types are inherited, and each biological parent donates one ABO allele to their child. For example, a child of two parents each having blood type A could have either type A or type O blood. A child of parents with blood types A and B could have any blood type.

   In this problem you are given list of the blood type of many couples. You should determine the set of blood types that might characterize their children.

   Input

   The first line of the input contains an integer N that denotes the number of the couples. Each of the following N lines contains two alleles. Those are the blood types of a couple.

   Output

   The output has N lines, each contains possible blood types of a child.

alleles.in

```
3
A B
AB AB
0 0
```

alleles.out

```
A B AB 0
A AB B
0
```

5. (Nim) *"Take five, or better, take one, or better, always already take none, ah!"*

Nim, also known as the Marienbad game, is a kind of logic game which is played between two players.

There are many variants of Nim. All of them start with N items (e.g. matches) and the purpose of the game is to take the last item. One of the easiest variant of Nim is played as follows:

Initially there are N items on the table between two players. The first player may take at least 1, at most M items. Then the second player may take at least 1,at most M items. They continue in this way in turn until there are no items to take any more. The one who takes the last item (or items) is the winner.

You have to write a program to play this game. Your program must calculate the number of the items it takes to win the game if it is possible in the first turn when you are the first player. Consider that your opponent plays the game perfectly (I think he has already made a program to play the game).

Input

The input file has only one line containing two integers N and M (1<=N, M<=200) where N is the number of all the items and M is the number of the items that can be taken at a time.

Output

Your program must write "You will lose", or "You will win. Start with X" where X is the number of the items you must take to win the game.

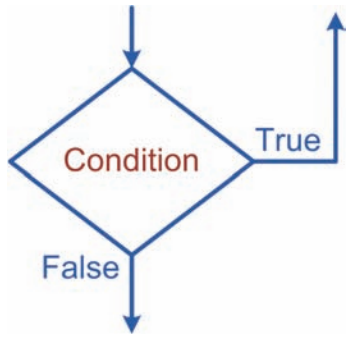
nim.in	nim.out
5 3	You will win. Start with 1

FLOWCHART PROGRAMMING (OPTIONAL)

Making Loops

Decision and repetition structures are called control structures in programming. Because programmers change the flow of the execution of the program with the help of these structures.

You have already learned how to use the diamond-shaped symbol to make decisions in flowchart programming. The same diamond-shaped symbol is also used to make loops.



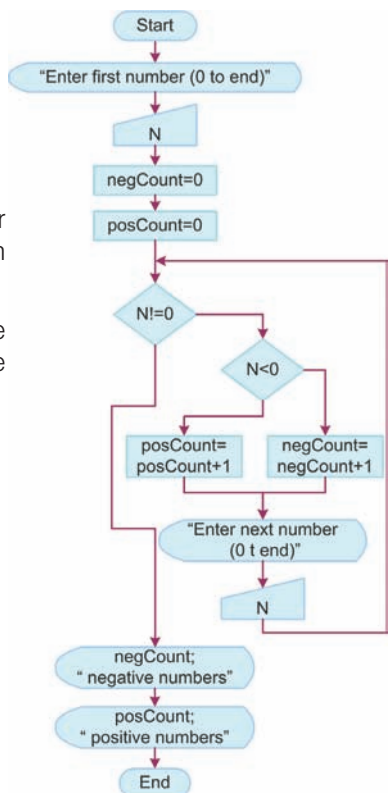
Pre-conditional Loops

In such a loop, first, the condition is evaluated. If the condition is true, statements in the loop body are executed and condition is evaluated again. If the condition is false, the program continues with statements after the loop.

Example: Pre-conditional loop

Make a flowchart that reads several integer numbers. The data entry will terminate when the user enters 0.

Your flowchart should print the number of the negative and the number of the positive numbers which have been read.



Since the test condition has been evaluated at the beginning of the loop structure, the **while** is a **pre-conditional loop** in C++.

Post-conditional Loops

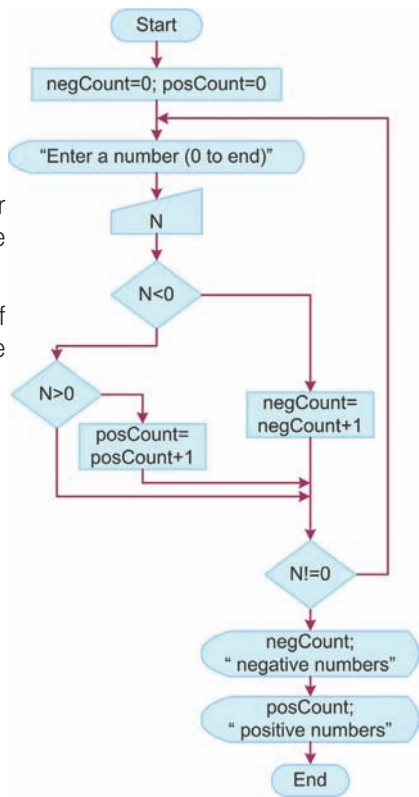
In such a loop, first, statements in the loop body are executed and then the condition is evaluated. If the condition is true, statements in the loop body are executed and condition is evaluated again. If the condition is false, the program continues with statements after the loop.

Example: Post-conditional loop

Since the test condition is been evaluated at the end of the loop structure, the **do/while** is a **pre-conditional loop** in C++.

Make a flowchart that reads several integer numbers. The data entry will terminate when the user enters 0.

Your flowchart should print the number of the negative and the number of the positive numbers which have been read.



Exercise: Class Average

You are asked to make a flowchart to calculate the average of a class from a school subject such as Computer Science.

There are N number of students in the classroom. Some students have only one mark, some students have more than one mark, and some students have no mark at all. The marks are between 1 and 10.

CHAPTER 4

FUNCTIONS

- Understanding Functions
- Pre-defined C++ Functions
- Call by Value and Reference
- Local and Global Variables
- Overloading Functions

C++

Introduction

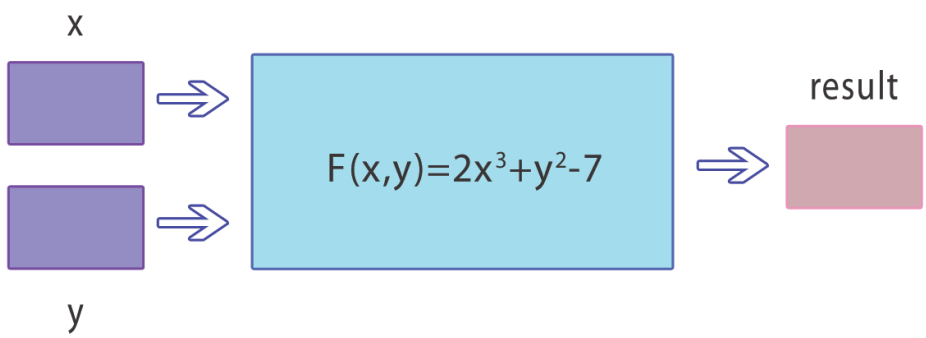
A **function** is a group of instructions that perform a specific task and executed when it is called from any point of the program code.

So far, our programs have had only one function (**main**). Most of the contemporary programs which solve real life problems are much larger than the programs that we have studied. The larger the program gets; the harder it becomes to cope with it. "**Function**" is the solution to this problem. Dividing the problem into simpler and more specific tasks and solving each with a function makes it easier to solve.

Another advantage of functions is preventing unnecessary repetition of the code segments. Forming the code as functions allows executing the same segment several times by calling from different points of the program.

Software re-usability can be counted as another benefit of using functions. After implementing a function for a specific task (say finding max of many numbers) it can be copied and used in future programs also.

Most probably you are already familiar with functions from math studies. There are polynomial, logarithmic, exponential and many other functions. Common properties of functions are: they have some input values as parameters, they do some operation with them and produce a result value as output. Functions in programming languages have the same process.



How a Function Operates

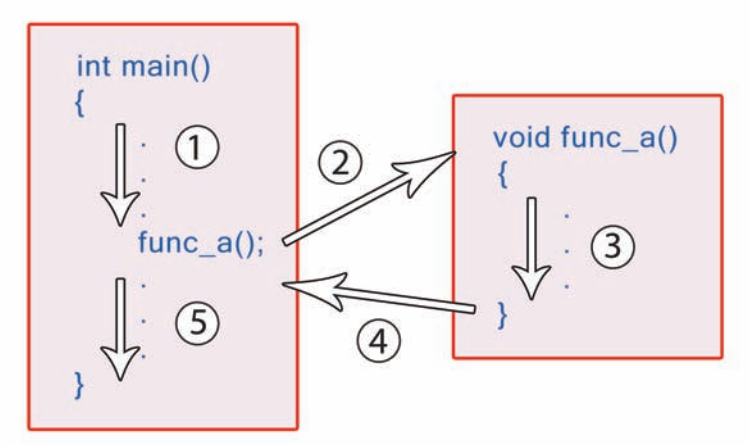
A function takes some input values, after a series of operations and produces a result. Function will produce different results for varying input values.

The Program Flow

The evolution of programming started in a sequential structure. The program execution was starting from top going step by step downward on a straight line. Thanks to functions modular programming was developed. This made program flow jump between functions.

Program flow always starts from the main function. As any function is called all local

variables are stacked, (to be able to be remembered after the execution of a function) and program flow runs the codes of the function body. The following figure describes how program flow changes:



Calling a Function From the Main Function

Some Pre-defined C++ Functions

C++ provides a wide range of predefined functions. For example "**cmath**" library contains plenty of different math functions. Here are some of them.

Function	Description	Example	Output
abs(x)	Returns the absolute value of x.	cout<<abs(-5); cout<<abs(10);	5 10
ceil(x)	Returns the smallest integer bigger than or equal to x.	cout<<ceil(15.8);	16
floor(x)	Returns the biggest integer less than or equal to x.	cout<<floor(16.3);	16
exp(x)	Returns Euler's number **e** raised to the power of x.	cout<<exp(1);	2.71828
log(x)	Returns natural logarithm of x.	cout<<log(2.72);	1.00063
log10(x)	Returns logarithm (base 10) of x.	cout<<log10(100);	2
pow(x,y)	Returns x to the power y.	cout<<pow(10.0 , 3.0);	1000
sqrt(x)	Returns the square root of x.	cout<<sqrt(9);	3
rand()	Returns a pseudorandom integer in [0 .. RAND_MAX (32767)].	cout<<rand() % 2;	0 or 1

Example: Area of a Triangle

Heron or **Hero of Alexandria** was born in 75 AD. He was an important geometer and worker in mechanics who invented many machines including a steam turbine. His best known mathematical work is the formula for the area of a triangle in terms of the lengths of its sides.

Let's see the usage of mathematical functions in an example. The following program takes the coordinates of three points of a triangle and calculates the area by using **Heron's formula**. Heron's formula states that the area (A) of a triangle whose sides have lengths **a**, **b**, and **c**

$$\sqrt{s(s-a)(s-b)(s-c)}$$

where **s** is half of the perimeter of the triangle:

$$s = \frac{a+b+c}{2}$$

Given the coordinates first we calculate the length of the sides by using **sqrt()** and **pow()** functions defined in the <math> header file. Then calculate **s** and calculate the area again with **sqrt()** function.

```
//PROG: c4_01math.cpp
#include <iostream>
#include <cmath>
using namespace std;
int main()
{
     int x1,x2,x3,y1,y2,y3;
     float a,b,c,s,area;

     cout<<"Enter the coordinates of first point :";
     cin>>x1>>y1;
     cout<<"Enter the coordinates of second point :";
     cin>>x2>>y2;
     cout<<"Enter the coordinates of third point :";
     cin>>x3>>y3;

     //calculate one side of triangle
     a=sqrt( pow(x1-x2,2.0)+pow(y1-y2,2.0) );
     //calculate one side of triangle
     b=sqrt( pow(x2-x3,2.0)+pow(y2-y3,2.0) );
     //calculate one side of triangle
     c=sqrt( pow(x3-x1,2.0)+pow(y3-y1,2.0) );

     s =(a+b+c) /2;
     area = sqrt( s * (s-a) * (s-b) * (s-c) );

     cout<<"\nArea is :"<<area<<endl;
     system("PAUSE");
     return 0;
}
```

As you may have noticed **main** has the same format of a function. In fact it is a special function that every program must have.

int main()

This means **main** has an integer type return value. The input parameters of a function are declared after the function name enclosed by parenthesis. Here **main** has an empty parameter list.

return 0;

return 0 is used to indicate a successful termination of the program.

The Structure of a Function

Every function consists of two parts: Function header and function body.

A function header contains

- **The return value type:** Gives the type of data returned by function
- **The name of the function:** The name that will be used to call the function. The same rules are applied as those applied for a variable name.
- **The input parameters:** Can be declared as many as needed. Separated by comas, each parameter has a variable type and name just as other variable declarations.

Function body part contains a group of statements enclosed by curly brackets.

```
return-value-type function-name (parameter-list) //function Header
{
  statements //function Body
}
```

Exercise: Dice

Make a program that simulates rolling a pair of regular six-faced dice.

Use

```
srand(time( NULL));
```

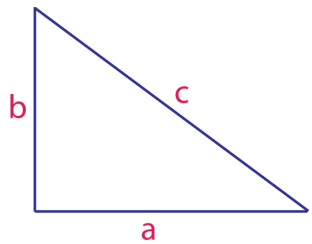
function to seed the random-number generator with current time so that the numbers will be different every time we run.

```
#include<time.h>
.
.
.
int main()
{
  srand(time(NULL));
  .
  .
  .
```

Exercise: Hypotenuse

Pythagoras was a Greek philosopher who made important developments in mathematics, astronomy, and the theory of music. The theorem now known as Pythagoras's theorem was known to the Babylonians 1000 years earlier but he may have been the first to prove it.

Write a program which takes two parameters as legs of a right triangle (a and b) and calculate the hypotenuse (c) of the triangle. Use pow(x,y), and sqrt(x) functions to accomplish your task.



Input : 3 4
Output : 5

Use Pythagorean theorem to calculate the hypotenuse. The square on the hypotenuse is equal to the sum of the squares on the other two sides.

$$a^2 + b^2 = c^2$$

Using Functions

Let's see the usage of functions ion an example. The following example shows the implementation of a simple **makeSum** program with function and without function.

The first program takes two integers (**a** and **b**) and prints to the screen the total of them (**sum**). The second program does the same job in a function. It can be called from anywhere in the program.

Each function should do only one task, but do it well. Name your functions according to their tasks. If you have a difficultly naming your function, probably it performs more than one task.

```
/*
PROG: c4_02sum.cpp
*/
#include<iostream>
using namespace std;

int main()
{
     int a,b,sum;

     cout<<"Enter two integers :";
     cin>>a>>b;
     sum=a+b;
     cout<<"Sum is :"<<sum<<endl;
     system("PAUSE");
     return 0;
}
```

```
Enter two integers :3 5
Sum is :8
Press any key to continue . . .
```

```
/*
PROG: c4_03void.cpp
Sum of two integers with a void function without any parameters.
*/
#include<iostream>
using namespace std;

void makeSum();      //function prototype, no parameters, no return
int main()
{
     makeSum();      //function call statement
     system("PAUSE");
     return 0;
}
void makeSum()       //gets two integers and prints their sum
{
     int a,b,sum;
     cout<<"Enter two integers :";
     cin>>a>>b;
     sum=a+b;
     cout<<"Sum is :"<<sum<<endl;
}
```

A **function prototype** declares the function name, its parameters, and its return type to the rest of the program prior to the function's actual declaration.

```
Enter two integers :3 5
Sum is :8
Press any key to continue . . .
```

The Return Statement

Although a function can be a **void** type (not returning any result), they are generally used with a return value. In our previous example, function makes an operation and prints the result to the screen. A function which returns the result into main function would seem smarter. See the following program:

```
/*
PROG: c4_04return.cpp
Sum of two integers with a function that reads two numbers and
return their sum.
*/
#include<iostream>
using namespace std;
int makeSum();       //function prototype, no parameters, return an
                     //integer value.
int main()
{
  int sum = makeSum();     //call the makeSum function
  cout<<sum<<endl;
  system("Pause");
  return 0;
}
```

```
int makeSum()
{
     int a,b;
     cout<<"Enter two integers :";
     cin>>a>>b;
     return a+b;     //return a+b to the caller function(main)
}
```

The function does not write the result but it returns the sum as the result of function to "sum" variable defined in main.

The return statement has two important roles:

- Carries the result of the function. It can return anything including variables, constants even function calls provided that it evaluates to a value of the type declared in the function header.
- Immediately terminates execution of function and returns to its caller.

A function may have more than one return statement. Function will terminate with any of them. See how the following function will quit with one of the return statements.

```
int max()
{
     int a,b;
     cin>>a>>b;
     if (a>b)
           return a;
     return b
}
```

The max function with conditional operator.

```
int max()
{
  int a,b;
  cin>>a>>b;
  return (a>b ? a: b);
}
```

Passing Arguments to the Functions

As we have mentioned before, functions generally take some values as input and these are called the parameters of the function. They behave like other local variables declared in the function and are created at the beginning of the function execution and destroyed on exit. So far our examples do not take any parameters. Let's improve our example so that it takes two integers as parameter and returns their sum.

```
/*
PROG: c4_05paramter.cpp
Sum of two function with a function that has two integers as input
parameters and returns their sum as a return value.
*/

#include<iostream>
using namespace std;
int makeSum(int, int);            //function prototype
int main()
{
```

```
    int a,b,sum;
    cout<<"Enter two integers :";
    cin>>a>>b;

    //call makeSum function
    sum = makeSum(a,b);          //a and b are actual parameters

    cout<<"Sum is :"<<sum<<endl;
    system("PAUSE");
    return 0;
}
int makeSum(int x, int y)        //x and y are formal parameters
{
    return x+y;
}
```

Parameters (arguments) are used to pass information back and forth between the calling and caller functions. The parameters in the function call statement is called **actual parameters**, and the parameters in the header of the function are called **formal parameters**.

Now it is more similar to our first function definition, which takes some input values, does some calculations and returns a result value. This makeSum function can be called from anywhere in the code with any values and will return sum of the inputs.

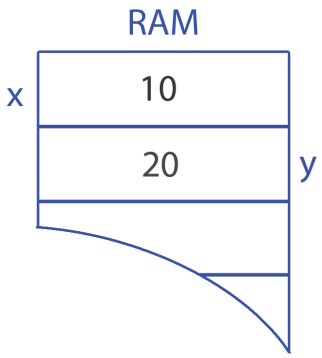
We have an important consideration while passing arguments to the function. That is the passed arguments change their values as the variables are passed. In other word, if you want the actual parameters to remain unchanged you should prefer the pass by value; on the other hand, if you want the change to affect actual parameters, prefer pass by reference.

Pass by Value

By default C++ passes the arguments by value. In the previous example, the makeSum function passed its parameters by value. More precisely, when a variable is used to call a function, a copy of that variable is created and used during the execution of the function. All the changes are done on this copy. In return the operations done inside a function do not alter the variables that are used to call the function. Examine the following example:

```
/*
PROG: c4_06byvalue.cpp
Passing an integer to a function as a value.
*/
#include<iostream>
using namespace std;
void doubleIt(int);

int main()
{
    int x=10;
    cout<<"Before calling the function dobuleIt, x is "<<x<<endl;
    doubleIt(x);
    cout<<"After calling the function dobuleIt, x is "<<x<<endl;
    system("PAUSE");
```

When calling a function with the **pass by value** method, the formal parameters take the values of the actual parameters. Actual and formal parameters reside in different locations of the memory.



Variables in the Memory

```
    return 0;
}
void doubleIt(int y)
{
    y=2*y;
    cout<<"After doubled in the function, y is "<<y<<endl;
}
```

```
Before calling the function dobuleIt, x is 10
After doubled in the function, y is 20
After calling the function dobuleIt, x is 10
Press any key to continue . . .
```

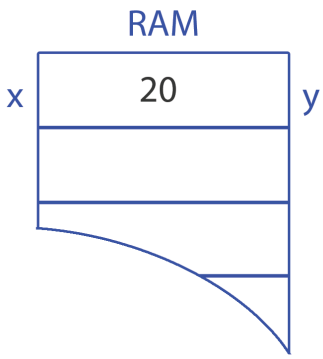
Here the **doubleIt()** function does not change the value of the variable used to call the function. Variable **y** is created as a copy of **x** but change on **y** is not applied to **x**.

Pass by Reference

The **address operator (&)** is a unary operator that can be used to obtain the memory address of any variable or object.

As we have seen, values of variables passed to the function remain unchanged; however you may intend to alter the values of actual parameters as the formal parameters change. You can accomplish this by passing the variables by reference. Technically, the difference is adding an **address operator (&)** in front of the passed variable. Theoretically, this means: pass the actual memory address of the variable instead of a copy of it. This makes the function do all operations on the actual memory address of the variable. As a result what applied in the function automatically affects the variable passed to the function.

Now let's implement the previous example with the pass by reference.



Variables in the Memory

```
/*
PROG: c4_07byreference.cpp
Passing address of an integer to a function as a value.
*/
#include<iostream>
using namespace std;

void doubleIt(int &);      //doubleIt will pass an address of an
                           //integer.
int main()
{
    int x=10;

    cout<<"Before calling the function dobuleIt, x is "<<x<<endl;
    doubleIt(x);
    cout<<"After calling the function dobuleIt, x is "<<x<<endl;

    system("PAUSE");
    return 0;
}
```

```
void doubleIt(int &y)
{
     y=2*y;
     cout<<"After doubled in the function, y is "<<y<<endl;
}
```

When calling a function **pass by reference**, the formal parameters take the addresses of actual parameters. Actual and formal parameters reside in the same locations of the memory.

```
Before calling the function dobulelt, x is 10
After doubled in the function, y is 20
After calling the function dobulelt, x is 20
Press any key to continue . . .
```

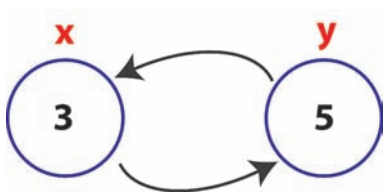
Exercise: makeSum

Remake the makeSum function so that it takes three parameters: the value of a, the value of b, and the address of sum. Then, calculates sum of a and b, assigns the result to sum. The main function of the program might be like the following:

```
int main()
{
  int a,b,sum;
  cout<<"Enter two integers:";
  cin >> a >> b;
  makeSum(a, b, sum);
  cout<<"Sum of your numbers is "<<sum<<endl;
  system("pause");
  return 0;
}
```

Exercise: Swap Function

Make your own version of swap function and test it in a program. To implement a swap function, what you would like to do is pass in two variables and have the function swap their values.



Scope and Lifetime

The scope of a variable is the program segment from which it can be referenced. In other words, scope rules manage when to allocate memory space for a variable and when to destroy it. A variable can be either of type local, global or static local. Local variables have function or block scope; while global variables have file scope.

Local Variables

All **variables** that are declared in a function are **local** to the function. They can not be referenced from outside the function (local scope). Their **lifetime** is the same as the activation of the function.

Parameters are like local variables (local scope, function lifetime).

Local variables have no initial values whereas parameters are assigned an initial value from the corresponding expression in the function call.

A variable declared within a block is local to that block. Such a variable can be accessed only in that block or in blocks nested within that block. And it is known from the point it is declared. This means that the variable is not created until the declaration statement is executed and it is destroyed with the ending curly brackets of that block. This is important to remember a local variable will not hold its value between activations of its block.

When a local variable declared with the same name of a variable in outer block, the variable in the outer block is hidden until the inner block is terminated. That is to say, while executing in the inner block, compiler sees the value of innermost local variable, not the value of the identically named variable of outer block. We will analyze this on the example after studying global variables.

The same rules are applied to the function arguments as those applied to local variables. That is function argument's scope is also local to the function and they are also destroyed on exit of function.

Global Variables

A **global variable** is accessible in every scope unless there is another local variable or parameter which has the same name as the global variable.

A variable declared outside of any function is a global variable and it can be accessed throughout the entire program. (Starting from the point it is declared). In other words global variables and function definitions have file scope.

Since the lifetime of global variables, starts from the point they are declared, it is best to declare them near the top of the program. However, theoretically they must be declared before they are first referenced.

Remember that if a local variable is defined with the same name, it is used in that local area not the global. Thus, although global variable can be referenced from any piece of code in the program, this requires that no local variable has the same name.

The following example illustrates the usage of local and global variables.

Keep your connections simple. Avoid global variables whenever possible. If you must use global variables as input, document them thoroughly.

```
/*
PROG: c4_08global.cpp
using local and global variables.
*/
#include<iosteam>
using namespace std;

void first();
void second();
```

```
int x = 1;                  //global x
int main()
{
       int x = 5;          //local x of function main()
       cout<<"local x in outer scope of main is "<<x<<endl;
       {
            int x = 7;      //local x special to this block
            cout<<"local x in inner scope of main is "<<x<<endl;
       }
       cout<<"local x in outer scope of main is "<<x<<endl;

       first();
       second();
       cout<<"local x in main is "<<x<<endl;
       system("PAUSE");
       return 0;
}

void first()
{
      int x = 25;    //local x of function first()
      cout<<"local x in the function first is "<<x<<endl;
}
void second()
{
     //since second( ) does not have any local x
     cout<<"global x in the function second is "<<x<<endl;
     x=10;           //change the value of global x.
}
```

```
local x in outer scope of main is 5
local x in inner scope of main is 7
local x in outer scope of main is 5
local x in the function first is 25
global x in the function second is 1
local x in main is 5
Press any key to continue . . .
```

Exercise: Seconds

Write a function that takes three integer parameters; number of hours, number of minutes, and number of seconds and returns the total number of seconds.

Static Local Variables

Static local variables are a mixture of local and global variables. They are defined like local variables, can not be accessed from out of the block. However, they are not destroyed at the end of the block, their storage duration extends the entire program. This means a static local variable will hold its value between activations of its block. Let's see the effect in the following example:

```
/*
PROG: c4_09static.cpp
Using static local variables.
*/
#include<iostream>
using namespace std;

void first()
{
      int x = 25;         //local x of function first()
      cout<<"local x in the function first is "<<x<<endl;
      ++x;
}

void second()
{
      static int x = 1; //x is initialized to 1 only on first run
      cout<<"This function is called "<<x<<" times "<<endl;
      ++x;
}

int main()
{
       first();
       second();
       cout<<"Calling functions once more"<<endl;
       first();
       second();
       system("PAUSE");
       return 0;
}
```

```
local x in the function first is 25
This function is called 1 times
Calling functions once more
local x in the function first is 25
This function is called 2 times
Press any key to continue . . .
```

This example shows the difference of local and static local variables. Compiler lost the value of local variable x and initialized to the same number when called for the second time (in function first()). However the static local variable is initialized only on the first run and it is remembered in the second run (in function second()).

Overloading Functions

Function overloading enables you to use the same name for different functions by differentiating the parameter list. The compiler chooses the right function according to the right parameter list. These functions with the same name, but are differentiated by parameter lists, are said to be overloaded functions.

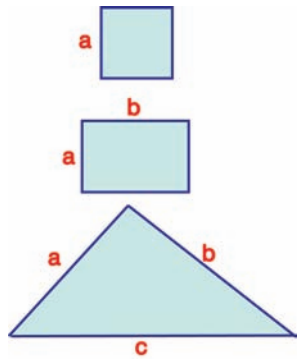
In the following example we have a function "max()" which returns the maximum of 2 numbers. Another instance of this function is written with 3 inputs. That is max function with two parameters returns the max of two numbers, whereas max function with three parameters returns the max of three.

```
/*
PROG: c4_10overload.cpp
Function overloading
*/
#include<iostream>
using namespace std;

int max(int, int);
int max(int, int, int);

int main()
{
     int x,y,z;
     cout<<"Please enter three numbers :";
     cin>>x>>y>>z;
     cout << "max of " << x<<" "<< y<<" "<< z<< " is:";
     cout << max(x,y,z)<< endl;

     system("PAUSE");
     return 0;
}

int max(int a, int b)    //max function with two parameters
{
  return (a>b ? a : b);
}

int max(int a, int b, int c)  //max function with three
parameters
{
  return max(max(a,b),c);
}
```

```
Please enter three numbers :3 5 2
max of 3 5 2 is:5
Press any key to continue . . .
```

Exercise: Area

Make a program to calculate the area of a square, rectangle, and triangle. Make three different functions with the same name to calculate the areas.



The area function to calculate area of a square should take single integer parameter, and return an integer value. The area function to calculate area of a rectangle should take two integer parameters and return an integer value. The area function to calculate area of a triangle should take three integer parameters and return a double value.

The main function of your program and a single run might be as follows:

```
  .
  .
  .
int main()
{
  int a,b,c;

  cout<<"side of a square=? ";
  cin >>a;
  cout<<"Area of the square is "<<area(a)<<endl;

  cout<<"sides of a rectangle=? ";
  cin >>a>>b;
  cout<<"Area of the rectangle is "<<area(a,b)<<endl;
  cout<<"sides of a triangle=? ";

  cin >>a>>b>>c;
  cout<<"Area of the rectangle is "<<area(a,b,c)<<endl;

  system("pause");
  return 0;
}
```

```
side of a square=? 4
Area of the square is 16
sides of a rectangle=? 3 5
Area of the rectangle is 15
sides of a triangle=? 5 6 8
Area of the rectangle is 14.9812
Press any key to continue . . .
```

SUMMARY

A **function** is a group of statements which performs a specific task. They are beneficial in terms of dividing the problem into simpler tasks, decreasing the code repetition and software re-usability. C++, provides the programmer with a large number of predefined functions.

Related to functions we have two important subjects: the method of *passing arguments* into function and types of variables according to **scope and lifetime**.

Speaking about scope and lifetime there are three different variable types which are: **local**, **global** and **static local variables**. Local variables are known only in the block they are defined. Global variables can be accessed from any part of the program unless a local variable is defined with the same name. Static local variables can be accessed from their own block but they exist till the end of program. This means that each function call compiler remembers the value of static local variables from the last call.

There are two ways to pass arguments to a function namely **pass by value** and **pass by reference**. By default, C++ uses the pass by value method for passing arguments to functions. "Pass by value" method creates a copy of the argument and does not influence the variable used to call the function. On the other hand "pass by reference" method passes the address of the variable, thus it enables altering the passed arguments value in the function.

Finally, a function can be rewritten with the same name but different parameter list. This is advantageous if the same operation with different types or amount of data is needed. This rewriting of a function is called **function overloading**.

REVIEW QUESTIONS

1. What is the name of the function each C++ function must have?
   a) start
   b) main
   c) c++
   d) begin

2. A _____ tells the compiler the name of the function, the type of data returned by the function, the number of parameters the function expects to receive, the types of the parameters, and the order in which these parameters are expected. The compiler use them to validate function calls.
   a) function prototype
   b) function call
   c) variable
   d) parameter

3. _____ function returns no value.
   a) int
   b) static
   c) void
   d) main

4. What is the output of the following program?

```
#include<iostream>
using namespace std;

int a,b;
void function()
{
    a = a+b;
    b = a*2;
}
void main()
{
    a=1; b=3;
    cout <<a<<" "<<b<<endl;
    function();
    cout <<a<<" "<<b<<endl;
}
```

5. What is the output of the following program?

```
#include<iostream>
using namespace std;

int a,b;
void function()
{
    int a=3, b=2;
    cout <<a<<" "<<b<<endl;
}
void main()
{
    a=1; b=3;
    cout <<a<<" "<<b<<endl;
    function();
    cout <<a<<" "<<b<<endl;
}
```

6. What is the output of the following program?

```
#include<iostream>
using namespace std;

int function(int x, int y)
{
    return x+y;
}
int main()
{
    int a=2, b=3;
    a = function(a, b);
    b = function(function(a, b), b);
    cout <<a<<" "<<b<<endl;
    return 0;
}
```

7. What is the output of the following program?

```
#include<iostream>
using namespace std;

int f1(int, int);
int f2(int, int);
int main()
{
    int a=2, b=1;
    a = f1(a, a);
    b = f2(b, a);
    cout <<a<<" "<<b<<endl;
    return 0;
}
int f1(int x, int y)
{
    return x+y;
}
int f2(int x, int y)
{
    return x*f1(x,y);
}
```

8. What does the following program do?

```
#include<iostream>
using namespace std;

void go(int &, int &);
void go(int &, int &, int &);
int main()
{
  int a, b, c;
  cout <<"Enter three integers: ";
  cin >>a>>b>>c;
  go(a,b,c);
  cout <<a<<" "<<b<<" "<<c<<endl;
  return 0;
}
```

```
void go(int &x, int &y)
{
  int temp = x;
  x = y;
  y = temp;
}
void go(int &x, int &y, int &z)
{
  if (x > y)
    go(x,y);
  if (x > z)
    go(x,z);
  if (y > z)
    go(y, z);
}
```

9. What does the following program do?

```
#include<iostream>
using namespace std;

int xyz(int &);
int main()
{
    int num;
    cout <<"Enter a positive number: ";
    cin >>num;
    while (num != 0)
          cout<<xyz(num)<<" ";
    return 0;
}
int xyz(int &n)
{
    int d = n%10;
    n /= 10;
    return d;
}
```

10. What does the following program do?

```
#include<iostream>
using namespace std;

int mystery(int a);
int main()
{
    int N,result=2;
    cout <<"N=? ";
    cin >>N;
    for (int i=3; i<=N; i++)
          result += mystery(i);
    cout <<result<<endl;
    return 0;

}
int mystery(int x)
{
    for (int i=2; i*i<=x; i++)
          if (x%i == 0)
                return 0;
    return x;
}
```

11. What does the following program do? (In the program, the what function calls itself. Such functions are known as recursive functions.)

```
#include<iostream>
using namespace std;
int what(int, int);
int main()
{   int b, p;
    cout <<"Enter two integers: ";
    cin >>b>>p;
    cout << what(b,p) <<endl;
    return 0;
}
int what(int x, int y)
{
    return (y>0? x*what(x,y-1) : 1);
}
```

12. The following program has been made to read a positive integer N, and then print the first N Fibonacci numbers. What is wrong with this program? (Fibonacci numbers starts with 0 and 1, and each successive number is the sum of previous two numbers. The first Fibonacci numbers are 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89.)

```
#include<iostream>
using namespace std;
void fib(int);
int main()
{
    int n;
    cout <<"N=? ";
    cin >> n;
```

```
    cout <<"The first "<<n<<" Fibonacci
    numbers are: "<<endl;
    fib(n);
    return 0;
}
void fib(int n)
{
    int prev=0, next=1;
    cout<<0<<" ";
    while(next <= n)
    {
          cout<<next<<" ";
          int temp = next;
          next += prev;
          prev = temp;
    }
}
```

13. The following program has been made to read a positive integer N, and then print the sum of the digits of N. What is wrong with this program?

```
#include<iostream>
using namespace std;
int go(int &);
int main()
{
    int n, sum=0;
    cout <<"N=? ";
    cin >> n;
    while (n>0)
    //add the next digit to the sum
          sum += go(n);
    cout<<sum<<endl;
    return 0;
}
int go(int &x)
{
    //return the right-most digit
    return x%10;
    //cut the right-most digit
    x /= 10;
}
```

PROGRAMMING PROBLEMS

1. **(Geometric Means)** Write a program that repeatedly asks the user to enter pairs of numbers until one of the pair is zero. For each pair, the program should use a function to calculate the geometric mean of the numbers. The function should return the answer to main(), which reports the result.The geometric mean of 2 numbers is the square root of the product of numbers, can be formulated as:

   Geometric Mean = square root of (X1 *(X2)

   Sample input : 2 32
   4 36
   14 0

   Sample output : 8
   12

2. **(Pascal Triangle)** Write a complete program which will display the Pascal's triangle. Program should read an integer (N) that is the number of the rows in the triangle. Write a function **factorial** that takes an integer n and returns its factorial (n!). Write a second function **combination** which takes two integers n and r and return the c(n,r). Remember c(n,m) = n! / ((n-r)! r!)

   Sample input : 5

   Sample output : 1
   1 1
   1 2 1
   1 3 3 1
   1 4 6 4 1

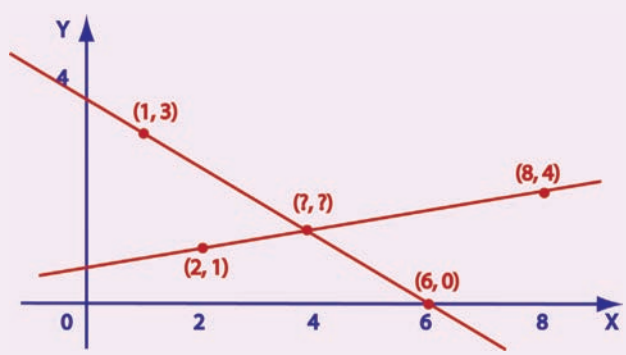
3. **(Line Intersection)** Make a program that gets two lines and finds their intersection point. Your program will reads two points of each line.

   The line is passing through the points p1(x1, y1) and the point (x2, y2) can be formulized as

$$\frac{(x1 - x2)}{(x - x1)} = \frac{(y1 - y2)}{(y - y1)}$$

   Sample input : 1 3 6 0
   2 1 1 3

   Sample output : 3.83 1.30

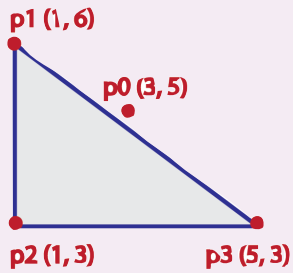


4. **(Mined Area)** You need to land a helicopter in a place where the war is going on. But you have been informed that a triangular area had been mined by the enemy forces. You are given coordinates of corner points (p1(x1, y1), p2(x2, y2), p3(x3, y3)) of the triangular area and coordinates of the landing point (p0(x0, y0)). Write a program that determines whether the landing point is safe.

The input consists of four lines. The first three lines contain the coordinates of the corner points, and the fourth line contains the coordinates of the landing point. The output should be "SAFE" if the landing point is out of the mined area. If the landing point is in the mined area, including borders, the output should be "DANGEROUS"

Sample input : 1 6
1 3
5 3
3 5

Sample output : SAFE



5. **(Highway)** Your are hired to calculate the total length of illuminated part of a highway. The length of the highway is L km and there are currently N number of lamps which are used on the highway. The lamps are not identical, thus, the length of the interval each lamp illuminates may vary. You are given the intervals of lamps. Make a program that calculates total length of the interval. The overlapped parts of the intervals will be calculated only once.

The input file has several lines. In the first line there are two integers L (1<= L <= 100) that denote the length of the highway in meters, and N (1<=N<=1000) that denotes the number of the intervals. Each of the following N lines contains two integers that denote the interval illuminating by a lamp. All the intervals are given in order by their starting position.

highway.in

```
15 6
0 1
2 4
6 9
6 10
11 14
12 13
```

highway.out

```
10
```

CHAPTER 5

ARRAYS & STRINGS

- Static Arrays & Vector Class
- Vector Manipulation
- Multidimensioanal Arrays
- Arrays as Parameters
- Searching Arrays
- Sorting Arrays
- String Class

C++

Arrays and Vector Class

A **data structure** is a way of storing data in a computer so that it can be used efficiently. Often a carefully chosen data structure will allow a more efficient algorithm to be used.

We have covered how to declare and use variables of various types, which can hold only one value at a time except string. We know how to declare a single char, integer, long and float values. These are big enough to implement the problems that can be handled with a small amount of memory. However we often encounter problems which need to keep a series of data. For example all marks of a student, or even all marks of all students etc. For such cases, we need a **data structure** that is capable to store a series of variables that are of the same type and size.

An **array** is a simple data structure that allows us to declare and use a series of same type of data.

Array Declaration and Accessing Array Elements

Any **array element** (item) can be accessed by giving the name of the array followed by the position number of the particular element in square brackets([]). The first element in an array always has the index zero, thus, **a[0]** refers to the first element of the array **a**. The second element of the array a is **a[1]**, the third element is **a[2]**, and so on. That is, the index of the i^{th} element of an array is **i-1**.

C++ supports two sorts of array: **static arrays** and **dynamic arrays (vector)**.

The declaration of an static array slightly differs from the declaration of a single variable. It is the size of the array which is specified within square brackets following the variable name. For example, an array (a) of 10 integers can be declared as:

```
int a[10];
```

Vector class is a container class that represent arrays in C++. Vector class is defined in the **<vector>** header file. Vector is a dynamic data structure in that, its size can increase and shrink at the time. The following program declares an array of integer (a), reserves memory for 5 elements, and then assigns some numbers to the array.

//PROG: c5_01vector.cpp

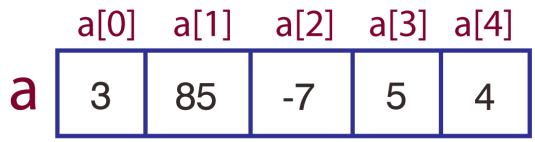
```
#include<vector>
using namepace std;
int main()
{
  vector<int> a;    //declare a vector of integer
  a.resize(5);      //allocate memory for five elements
  a[0]=3; a[1]=85; a[2]=-7; a[3]=5; a[4]=4;
  cout<<"a[1] is "<<a[1]<<endl;
  cout<<"a[a[0]] is "<<a[a[0]]<<endl;
  system("Pause");
  return 0;
}
```

The static arrays (C-like arrays) can be initialized during the declaration.

```
int a[3]={1,2,3};
int b[5]={1,2};
char c[] = "abc";
```

a gets (1, 2, 3), **b** gets (1, 2, 0,0, 0) and **c** gets "abc".

```
a[1] is 85
a[a[0]] is 5
Press any key to continue...
```

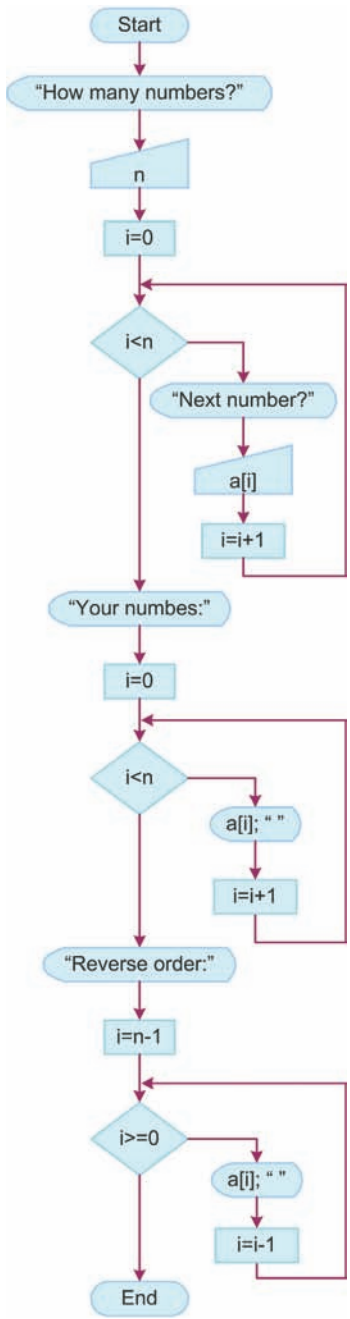


Example: Reading and Printing Vectors

Make a program that reads N numbers from the user and then prints them in reading order and in reverse order.

```
/*
PROG: c5_02reverse.cpp
*/
#include<iostream>
#include<vector>    //for vector class
using namespace std;

int main()
{
  int n;
  vector<int> numbers;
  cout<<"How many numbers? ";
  cin >> n;
  numbers.resize(n);       //allocate memory for n items.
  for(int i=0; i<n; i++)          //read the numbers
  {
    cout<<"Enter the next number: ";
    cin >> numbers[i];
  }
  cout<<"Your numbers in reading order: ";
  for(int i=0; i<n; i++)          //print in reading order
    cout<<numbers[i]<<" ";
  cout<<endl;
  cout<<"Your numbers in reverse order: ";
  for(int i=n-1; i>=0; i--)       //print in reverse order
    cout<<numbers[i]<<" ";
  cout<<endl;
  system("Pause");
  return 0;
}
```

```
How many numbers? 5
Enter the next number: 3
Enter the next number: 4
Enter the next number: 5
Enter the next number: 2
Enter the next number: 6
Your numbers in reading order: 3 4 5 2 6
Your numbers in reverse order: 6 2 5 4 3
Press any key to continue . . .
```



Flowchart of the Program "reverse"

Exercise: Collecting Coins

Your are given N coins in a line. The value of a coin can be either 1, 2, 5, 10, 25 or 50 . Make a program that gets the number of the coins (N), a selection factor (K), and the values of coins. Your program must calculate the sum of the coins in the positions i*K where i starts with 0 and increases by 1.

coins.in

```
8 3
2 5 25 10 1 2 50 5
```

coins.out

```
62
```

Vector Manipulation

A vector can be assigned to another vector by using the assignment operator (=), and relational operators (==, <, <=, >, >=, !=) can be used to compare two vectors.

Vector Class provides some methods for handling some vector operations. Some of the Vector Class methods are listed as follows:

Method	Description
size	Returns the number of elements in the vector.
resize	Specifies a new size for a vector.
push_back	Add an element to the end of the vector.
begin	Returns an iterator to the first element in the vector.
end	Returns an iterator that points just beyond the end of the vector.
assign	Erases a vector and copies the specified elements to the empty vector.
reverse	Reserves a minimum length of storage for a vector object.
insert	Inserts an element or a number of elements into the vector at a specified position.
at	Returns a reference to the item at a specified location in the vector.

Iterator is a pointer that is used to refer a single item of a vector or another STL container. In the following program segment, **intArray** is an integer vector and **intIt** is an iterator of an integer vector.

```
vector <int> intArray;

vector<int>::iterator intIt;
```

The **Standard Template Library (STL)**, is a C++ library of container classes (vector, list, set, map, etc), algorithms (sort, find, etc), and iterators; it provides many of the basic algorithms and data structures of computer science.

Example: Merging Vectors

Make a program that merges two sorted lists of numbers. Your program should read the list from a file and print the merged list into another file.

merge.in

```
5
3 5 5 9 11
8
1 3 5 8 9 14 15 20
```

merge.out

```
1 3 3 5 5 5 8 9 9 11 14 15 20
```

//PROG: c5_03merge.cpp

```
#include<fstream>
#include<vector>
using namespace std;
int main()
{
     vector<int> a,b,c;
     int i,j,n;
     ifstream fin("merge.in");
     ofstream fout("merge.out");
     fin>>n;  a.resize(n);
     for (i=0;i<n;i++)
          fin>>a[i];
     fin>>n;  b.resize(n);
     for (i=0;i<n;i++)
          fin>>b[i];

     i=j=0;
     while ((i<a.size()) && (j<b.size()))
       if (a[i] < b[j])
            c.push_back(a[i++]);
       else
            c.push_back(b[j++]);

     while (i<a.size())
       c.push_back(a[i++]);
     while (j<b.size())
       c.push_back(b[j++]);

     for (i=0;i<c.size();i++)
          fout<<c[i]<<" ";

     return 0;
}
```

The process of merging two sorted lists of items can be used for a variety of applications. For example, the well-known merge sort algorithm is based on dividing and merging the lists.

Multidimensional Arrays

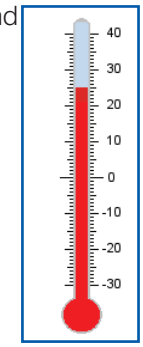
Arrays can have more than one dimension which are called **multidimensional arrays**. Suppose that we want to keep temperature statistics of a month. Each entry keeps the temperature value of one day. Assuming a month with 4 weeks of 7 days it can be illustrated as:

	0	1	2	3	4	5	6
0	25	27	27	28	25	25	23
1	22	22	20	21	21	23	24
2	24	25	25	29	30	33	35
3	38	38	35	33	33	30	27

Vector refers to a one dimensional array, and **matrix** refers to a two dimensional array in computer science

Exercise: Reading and Printing a Matrix

Make a program that gets daily temperatures of a month and prints them in a tabular format.




```
//PROG: c5_04matrix.cpp
#include<iostream>
#include<vector>
using namespace std;

int main()
{
  //declare a vector of vector with 4 rows
  vector< vector<int> > month(4);
  int i,j;

  for(i=0; i<4; i++)
  month[i].resize(7);              //resize each row to 7

  //Read the temperature of each day
  for (i=0; i<4; i++)
    for (j=0; j<7; j++)
    {
       cout<<"Temperature of week "<<i<<" and day "<<j<<" :";
       cin >> month[i][j];
    }
  cout<<endl;
```

The **month** could be declared as a static matrix like below:

```
int month[4][7];
```

```
  //Print the matrix in a tabular format
  for (i=0; i<4; i++)
  {
    for (j=0; j<7; j++)
       cout<<month[i][j]<<" ";
    cout<<endl;
  }

  system("Pause");
  return 0;
}
```

```
Temperature of week 0 and day 0 :25
Temperature of week 0 and day 1 :27
Temperature of week 0 and day 2 :27
Temperature of week 0 and day 3 :28
Temperature of week 0 and day 4 :25
Temperature of week 0 and day 5 :25
Temperature of week 0 and day 6 :23
Temperature of week 1 and day 0 :22
Temperature of week 1 and day 1 :22
Temperature of week 1 and day 2 :20
Temperature of week 1 and day 3 :21
Temperature of week 1 and day 4 :21
Temperature of week 1 and day 5 :23
Temperature of week 1 and day 6 :24
Temperature of week 2 and day 0 :24
Temperature of week 2 and day 1 :25
Temperature of week 2 and day 2 :25
Temperature of week 2 and day 3 :29
Temperature of week 2 and day 4 :30
Temperature of week 2 and day 5 :33
Temperature of week 2 and day 6 :35
Temperature of week 3 and day 0 :38
Temperature of week 3 and day 1 :38
Temperature of week 3 and day 2 :35
Temperature of week 3 and day 3 :33
Temperature of week 3 and day 4 :33
Temperature of week 3 and day 5 :30
Temperature of week 3 and day 6 :27

25 27 27 28 25 25 23
22 22 20 21 21 23 24
24 25 25 29 30 33 35
38 38 35 33 33 30 27
Press any key to continue . . .
```

Exercise: Transpose Matrix

The **transpose** of a matrix is the reverse orientation of the original matrix, so that the values across the rows become the values down the columns, and the values of the columns become the values across the rows.

Make a program to transpose a matrix.

Sample run:

```
Enter the number of rows 2
Enter the number of columns 4
Enter the values of the matrix
3 6 9 1
5 2 7 4
Transpose of the matrix is:
3 5
6 2
9 7
1 4
Press any key to continue . . .
```

Exercise: The Biggest Subtotal

You have an NxM matrix filled with numbers between -100 and 100. Find the KxK submatrix which has the biggest sum, where (K<=N and K<=M).

The input file has several lines. The first line contains three integers: N, M, and K respectively. Each of the following N lines represent a row of the matrix with M integers.

The output should display the position of the left-upper element of the submatrix in the first line, and the sum in the second line.

subtotal.in

```
4 5 3
2 6 7 -2 4
4 8 -3 9 3
7 6 -1 5 2
-5 1 -11 3 -8
```

subtotal.out

```
0 1
35
```

Passing Arrays to Functions

There is no difference passing individual array elements to a function. They can pass as value or reference.

Static arrays and vectors behave differently while passing them to functions. Static arrays are always passed by reference, where as, vectors can be passed to functions by value or by reference. The name of a static array is the address of its first element, thus, passing it to a function, it implicitly takes the address.

The following program demonstrates passing arrays to functions.

```
//PROG: 05_05pass.cpp
#include<iostream>
#include<vector>
using namespace std;

typedef vector<int> TIntVec;      //user defined type definition
void f1(int[], int);              //call by reference
void f2(TIntVec);                 //call by value
void f3(TIntVec &);               //call by reference
void print(int[], int);           //call by reference
void print(TIntVec);              //call by value
//-------------------------------------------------
int main()
{
  int a[5] = {1,2,3,4,5}; //declare and initialize a
  vector <int> v(a, a+5); //declare v and initialize it to a

  print(a,5);
  print(v);
  f1(a, 5);
  f2(v);
  print(a, 5);
  print(v);
  f3(v);
  print(v);
  system("pause");
  return 0;
}
//-------------------------------------------------
void f1(int a[], int n)
{
  int temp = a[0];
  for(int i=1; i<n; i++)
    a[i-1] = a[i];
  a[n-1] = temp;
}
//-------------------------------------------------
void f2(TIntVec a)
{
  int temp = a[0];
  for(int i=1; i<a.size(); i++)
```

Every variable must have a data type in C++. The **typedef** keyword is used to define new data type names to make a program more readable.

When passing a static vector to a function, you may omit the length of the vector. It can be passed as a second parameter:

```
void f(int[], int);
```

When passing a static multidimensional array to a function, you must denote the length of all dimensionals except the first one. The length of the first dimension may be passed as a second parameter

```
void f(int[][5], int);
```

```
    a[i-1] = a[i];
  a[a.size()-1] = temp;
}
//-----------------------------------------------
void f3(TIntVec &a)
{
  int temp = a[0];
  for(int i=1; i<a.size(); i++)
    a[i-1] = a[i];
  a[a.size()-1] = temp;
}
//-----------------------------------------------
void print(int a[], int n)
{
  for (int i=0; i<n; i++)
    cout<<a[i]<<" ";
  cout<<endl;
}
//-----------------------------------------------
void print(TIntVec v)
{
  for (int i=0; i<v.size(); i++)
    cout<<v[i]<<" ";
  cout<<endl;
}
```

```
1 2 3 4 5
1 2 3 4 5
2 3 4 5 1
1 2 3 4 5
2 3 4 5 1
Press any key to continue . . .
```

Exercise: Sorted

The following function returns true if the vector v is sorted in nondecreasing order, and returns false otherwise. Complete the function.

```
bool isSorted (vector<int> v)
{
  .
  .
  .
}
```

Searching Arrays

Searching arrays for a particular value is a common activity that any programmer should know how to do. Searching is especially useful with arrays. Searching is used daily on the Internet, with surfers using search engines.

C++ provides two kinds of searching: **sequential searching** and **binary searching**.

Sequential (Linear) Searching

The **sequential search** is best used if the array you are searching is unsorted. It compares the target value with all the elements of the array one by one starting from the first element until it finds the target value or there is no more element to compare.

The **find** method in the **<algorithm>** header file performs sequential searching. It locates the position of the first occurrence of the target in a range that has a specified value.

The find method returns an iterator addressing the first occurrence of the target in a range. If no such value exists in the range, the iterator returns the address of the position that is one past the final element.

The following program demonstrates the **find** method.

```
//PROG: 05_06linsearch.cpp
#include<iostream>
#include<vector>
#include<algorithm>
using namespace std;

int main()
{
  vector<int> v;
  vector<int>::iterator it; //it is an iterator to an int vector.
  v.push_back(1);
  v.push_back(13);
  v.push_back(4);
  v.push_back(7);

  it = find(v.begin(), v.end(), 4);     //searching for 4
  if (it != v.end())
    cout<<"Found in the position "<<it-v.begin()<<endl;
  else
    cout<<"Not found"<<endl;

  system("pause");
  return 0;
}
```

the **pointer dereferencing operator (*)** is used to get the value which is pointed by an iterator. The statement below prints the value that is pointed by the iterator **it**:

```
cout<<(*it)<<endl;
```

```
Found in the position 2
Press any key to continue . . .
```

Exercise: Linear Searching

Make your own version of the linear search function. Your function should take a vector and a target value as input parameters, and returns the position of the first occurrence of the target in the vector. If the target doesn't exist in the vector, your function returns -1.

```
int linearSearch(vector<int> v, int target)
{
   .
   .
   .
}
```

Binary Searching

Sometimes we need to search for a specific value in a sorted list. For example, looking up a word in a dictionary or finding a name in a phone book.. Having the information of list being sorted decreases the search algorithms complexity logarithmically. That is to say the number of comparison operations decreases to $\log_2(N)$ instead of N. When performing a search in a list of 1000 items, a linear search requires 1000 comparisons in the worst case, on the other hand, a binary search requires 10 comparisons at most.

The **binary_search** method in the <algorithm> header file performs binary searching. It tests whether there is an element in a sorted range that is equal to a target value. If the target value is in the range **binary_search** method returns **true**, otherwise it returns **false**.

The following program demonstrates how to use tthe **binary_search** method.

```
//PROG: c5_07binsearch.cpp
#include<iostream>
#include<vector>
#include<algorithm>
using namespace std;

int main()
{
  vector<int> v;
  v.push_back(1);
  v.push_back(13);
  v.push_back(4);
  v.push_back(7);

  sort(v.begin(), v.end());
```

```
  if(binary_search(v.begin(), v.end(), 4))
    cout<<"found."<<endl;
  else
    cout<<"not found."<<endl;

  system("pause");
  return 0;
}
```

found.

Exercise: Binary Search

The binary_search method returns true if the target value exists in the given range. In the case you need any position of the target in the range, you must make your own version of a binary search function.

The binary search begins by comparing the target value to the value in the middle of the vector; because the values are sorted, it is clear that whether the target value would belong before or after that middle value, and the search then continues through the correct half in the same way.

Complete the following user-made binary search function, so that it returns the position of the target value or -1 if the target is not in the vector.

```
int binarySearch(vector<int> v, int target)
{
  int first=0, last=v.size()-1;
  while(first<=last)
  {
    int middle = (_____________)/2;
    if (v[middle] == target)
      return middle;
    if (target < v[middle])
      last = middle-1;
    else
      first = _____________;
  }
  return _____________;
}
```

Exercise: Vectors

Write a function which takes two vectors as parameters and checks if the second vector contains all of the elements of the first vector. If yes returns true, otherwise returns false.

```
bool test(TIntVec v1, TIntVec v2)
{
   .
   .
   .
}
```

input : v1 = 3 6 7 3 8 7
v2 = 2 9 4 8 5 6 7 3 1

output : yes

Sorting Arrays

Sorting data (arranging items in order) is one of the most important and fundamental applications. As we have seen in binary search, having sorted data facilitates the processing of information. All the words in a dictionary or the names in a phone book are sorted in alphabetical order. You can sort the files in a folder in your computer by their names, dates, types or sizes.

C++ provides the **sort** method to sort the vectors and other containers. The **sort** method is defined in the <**algorithm**> header file. The following statement arranges the elements of the vector into ascending order :

```
sort(v.begin(), v.end());
```

You should add a comparison criterion as a third parameter of the **sort** method to arrange the items in descending order:

```
sort(v.begin(), v.end(), greater<int>());
```

You must add <**functional**> header file for **greater<int>()**.

The following program demonstrates using the **sort** method.

```
//PROG: c5_08sort.cpp
#include<iostream>
#include<vector>
#include<functional>        //for greater<int>()
#include<algorithm>
using namespace std;
```

```
typedef vector<int> TIntVector;
void print(TIntVector);

int main()
{
  vector<int> v;
  v.push_back(1);
  v.push_back(13);
  v.push_back(4);
  v.push_back(7);

  cout<<"The original vector: ";
  print(v);
  sort(v.begin(), v.end());                    //increasing order
  cout<<"After sorting in ascending order: ";
  print(v);
  sort(v.begin(), v.end(), greater<int>());    //decreasing order
  cout<<"After sorting in descending order: ";
  print(v);

  system("pause");
  return 0;
}

void print(TIntVector v)
{
  for(int i=0; i<v.size(); i++)
    cout<<v[i]<<" ";
  cout<<endl;
}
```

Once you get a vector in ascending order, you may use the **reverse** method to get the vector sorted in descending order.

```
reverse(v.begin(), v.end());
```

```
The original vector: 1 13 4 7
After sorting in ascending order: 1 4 7 13
After sorting in descending order: 13 7 4 1
Press any key to continue . . .
```

Exercise: Selection Sort

There exist many sorting algorithms in programming. The selection sort algorithm is the most intuitive one. The idea of **selection sort** is rather simple: you repeatedly find the next smallest element in the array and move it to its final position in the sorted array.

Step 1: Find the minimum value in the list

Step 2: Swap it with the value in the first position

Step 3: Repeat the steps above for remainder of the list (starting at the second position)

Complete the following **selectionSort** function:

```
void selectionSort(TIntVector &v)
{
   .
   .
   .
}
```

Research: Sorting Algorithms

Study the sorting algorithms listed below, from the Internet and other resources.

- Bubble Sort
- Insertion Sort
- Counting Sort (Pigeon Hole Sort)
- Quick Sort
- Merge Sort

String Class

Strings are written in double quotation marks, and characters are written in single quotation marks in C++.

```
int main()
{
   char ch = 'a';
   string s = "Zambak";
   .
   .
   .
}
```

Remember that a **character** is any symbol that can be read from the keyboard. A **string** is a series of characters which is stored in consecutive bytes in memory. So that any input sequence from keyboard can be expressed as a string.

C++ has a **string class** that is defined in the **<string>** header file to manipulate the strings. You have already learned how to read and print the strings with **cin** and **cout** where **cout** prints all the string, whereas, **cin** reads a one-word string. Use the **getline()** method to read the whole string, instead of **cin**. **getline() gets a line of text from the input stream.**

Reading and Printing Strings

The following program demonstrates how to declare, initialize, read and print the characters and strings.

```
//PROG: c5_09string.cpp
#include<iostream>
#include<string>
using namespace std;
```

```
int main()
{
  string s1= "Hello Word", s2, s3;
  cout<<"Enter a string: ";
  getline(cin, s2);        //reads a line of text.
  cout<<"Enter the same string: ";
  cin >> s3;               //reads only the first word.

  cout<<"s1 = "<<s1<<endl;
  cout<<"s2 = "<<s2<<endl;
  cout<<"s3 = "<<s3<<endl;

  system ("pause");
  return 0;
}
```

The **getline()** function can be used with a **delimiter** to read the first part of the input stream until the delimiter. The following statement reads only the first word of the input text.

```
getline(cin, s, ' ');
```

```
Enter a string: C++ is an OOP language.
Enter the same string: C++ is an OOP language.
s1 = Hello Word
s2 = C++ is an OOP language.
s3 = C++
Press any key to continue . . .
```

String Manipulation

Strings are actually the arrays of characters. Most of the **vector class methods** and **string class methods** are common. Assignment (=) and relational (==, <, <=, >, >=, !=) operators are used with strings. Strings can be passed to a function as a value or reference, and any function can return a string value.

String class has a **find()** and a **length()** method that are not available for vectors. The member function **find()** searches a string in a forward direction for the first occurrence of a substring that matches a specified sequence of characters. The **length()** member function is the same as **size()**. In addition the **find()** method, **find_first_not_of()**, **find_first_of()**, **find_last_not_of()**, and **find_last_of()** methods are available with the string class.

The following program demonstrates how to handle strings in C++.

```
//PROG: c5_10string.cpp
#include<iostream>
#include<string>
#include<algorithm>
using namespace std;

string spaces(string);
int main()
{
  string name;
```

```
  cout<<"Your name and surname?";
  getline(cin, name);

  cout<<"Your name with spaces: ";
  cout<<spaces(name)<<endl;

  int pos = name.find("ARM");
  if (pos == -1)
    cout<<"ARM is not in your name"<<endl;
  else
    cout<<"ARM was found at the position "<<pos<<endl;

  cout<<"Your name in reverse order: ";
  reverse(name.begin(), name.end());
  cout<<name<<endl;

  system ("pause");
  return 0;
}
//-----------------------------------------------
//insert a space between the characters
string spaces(string s)
{
  string s2;
  for (int i=0; i<s.length(); i++)
  {
    s2.push_back(s[i]);
    s2.push_back(' ');
  }
  return s2;
}
```

```
Your name and surname?ALAN FARMER
Your name with spaces: A L A N   F A R M E R
ARM was found at the position 6
Your name in reverse order: REMRAF NALA
Press any key to continue . . .
```

Example: Palindromes

A **palindrome** is a word, phrase, number or any other sequence of units which has the property of reading the same in either direction. “MADAM” is a palindrome but “ADAM” is not.

Make a program that reads a text from an input file and prints the lines that are palindrome into another file.

```
//PROG: c5_11palindrome.cpp
#include<fstream>
#include<string>
#include<algorithm>
using namespace std;

ifstream fin("palin.in");
ofstream fout("palin.out");

int main()
{
  string s1, s2;
  while(!fin.eof()) //continue if not end of file.
  {
    getline(fin,s1);
    s2 = s1;
    reverse(s2.begin(), s2.end());
    if (s1==s2) //reverse of a palindrome is itself
       fout<<s1<<endl;
  }
  return 0;
}
```

palin.in

```
MADAM
ADAM
EY EDIP ADANADA PIDE YE
THIS IS A PALINDROME
MALAYALAM
PALINDROME
A GUNG A GNUG A
```

palin.out

```
MADAM
EY EDIP ADANADA PIDE YE
MALAYALAM
A GUNG A GNUG A
```

Exercise: Palindrome

Remake the palindrome program above. Your program should test a line of text if it is a palindrome without using the **reverse** method. Compare the characters of the line yourself.

Exercise: Names

Make a program that reads the names of the students in your classroom, and then sorts the names in alphabetical order.

names.in

```
George VICTORIA
Lara NIKOLAEVICH
Naif SHERIF
Baden JANNINGS
Vanna NAYOR
Abiba JUMBA
Macarena ALFONSO
```

names.out

```
Abiba JUMBA
Baden JANNINGS
George VICTORIA
Lara NIKOLAEVICH
Macarena ALFONSO
Naif SHERIF
Vanna NAYOR
```

SUMMARY

An **array** is a container object that holds a fixed number of values of a single type. Arrays can be with one **(vector)**, two (**matrix**), or more dimensionals. The length of a **static array** is established when the array is created. After creation, its length is fixed. The C++ **vector class** lets us create and manipulate **dynamic arrays**. **Dynamic array** is a data structure that can be resized and allows elements to be added or removed.

C++ supports C-style strings that are the static array of characters. However, in the C++ programming language, the **string class** is a standard representation for a string of text.

Vector class, string class, and **STL** algorithms provides plenty of vector and string manipulation methods. Some of these methods are **size**, **resize**, **push_back**, **begin**, **end**, **assign**, **reverse**, **insert**, **at**, **sort**, and **find**.

Instances of vector class and string class can be passed to a function in the same manner the standard variables are passed. That is, they can pass by value or they can pass by reference. On the other hand, keep in mind that, static arrays are always passed to a function by reference.

The **Standard Template Library (STL)** is a software library available with C++. It provides some common data structures as **containers**, and the methods to handle the containers as **algorithms**.

In computer science, **sorting** is the process of putting elements of a list in a certain order. Efficient sorting is important to optimize the use of other algorithms that require sorted lists to work correctly.

The act or process of finding a particular item of an array is called **searching**. Linear search and binary search are two well-known searching algorithms: **Linear search** operates by checking every element of a list one at a time in sequence until a match is found. A faster way to search a sorted array is to use a **binary search**. The idea is to look at the element in the middle. If the key is equal to that, the search is finished. If the key is less than the middle element, do a binary search on the first half. If it's greater, do a binary search of the second half.

REVIEW QUESTIONS

1. Which of the array declaration is wrong?
   a) int a[n];
   b) int a[5];
   c) vector<int> a;
   d) vector<int> a(5);

2. Which of the following statements, initializes the array a to ten zeros?
   a) int a[10];
   b) vector<int> a(10);
   c) int a[10] = {0}
   d) int a[] = {0};

3. What is the output of the following program?

```
#include<iostream>
#include<vector>
using namespace std;

int main()
{
    vector<int> v;
    v.push_back(0);
    for (int i=1; i<5; i++)
          v.push_back(i+v[i-1]);
    int sum=0;
    for(int i=0; i<v.size(); i++)
          sum += v[i];
    cout<<sum;
    return 0;
}
```

4. What is the output of the following program?

```
#include<iostream>
#include<vector>
#include<algorithm>
using namespace std;

int main()
{
    vector<int> v;
    v.push_back(7);
    v.push_back(4);
    v.push_back(5);
    v.push_back(1);
    sort(v.begin(), v.end());
    cout<<v[2];
    return 0;
}
```

5. What is the output of the following program?

```
#include<iostream>
#include<vector>
#include<algorithm>
using namespace std;

int main()
{
    vector<int> v;
    v.push_back(7);
    v.push_back(4);
    v.push_back(5);
    v.push_back(1);
    sort(v.begin(), v.end());
    cout<<*(find(v.begin(),v.end(),4));
    return 0;
}
```

6. What is the output of the following program?

```
#include<iostream>
#include<vector>
using namespace std;

typedef vector<int> TIntVec;
void function1(TIntVec &);
int function2(TIntVec, int);
int main()
{
    TIntVec v;
    function1(v);
    cout<<function2(v,2)<<endl;
    cout<<function2(v,3)<<endl;
    return 0;
}
```

```
void function1(TIntVec &v)
{
    v.push_back(5);
    v.push_back(8);
    v.push_back(9);
    v.push_back(1);
    v.push_back(6);
}
int function2(TIntVec v, int k)
{
    int res=0;
    for(int i=0; i<v.size(); i++)
          if (v[i]%k == 0)
                res += v[i];
    return res;
}
```

7. What does the following program do?

```
#include<iostream>
#include<vector>
#include<time.h>
using namespace std;

typedef vector<int> TIntVec;
void abc(TIntVec &,int);
int main()
{
    srand(time(NULL));
    TIntVec v1, v2;
    int counter = 0, k;
    cout<<"size =?";
    cin >> k;
    do
    {     abc(v1, k);
          abc(v2, k);
          counter++;
    }while(v1 != v2);
    cout<<"After "<<counter<<" tries.";
    return 0;
}
void abc(TIntVec &v, int k)
{
    v.resize(k);
    for(int i=0; i<k; i++)
          v[i] = rand()%2;
}
```

8. The islower and isupper functions determine if a particular character is in lower case, or upper case. The tolower and toupper functions convert a character to lower case, or uppercase. What does the following program do?

```
#include<iostream>
#include<string>
#include<time.h>
using namespace std;
int main()
{
    string s;
    cout<<"Enter a line of text:";
    getline(cin, s);
    for (int i=0; i<s.length(); i++)
    {
          if (islower(s[i]))
                s[i] = toupper(s[i]);
          else if (isupper(s[i]))
                s[i] = tolower(s[i]);
    }
    cout<<s<<endl;      return 0;
}
```

PROGRAMMING PROBLEMS

1. **(Data Compressor)** Data compression is the reduction in size of data in order to save space or transmission time. There are many data compression algorithms and programs. The files with **.zip** extension are compressed files, and the **.jpeg** format files are compressed pictures.

   You are asked to make a program to compress a line of text. Your program compresses only consecutive repeating characters. If any character is consecutively repeating, your program should replace those repeating part with the character itself and a number denoting the occurrence of the letter. Spaces will not be compressed.

   Sample Input : aaccccabbbccaaaaa xxxxyyyyxxxyyyyyyyyy

   Sample output : a2c4ab3c2a5 x4y3x3y10

2. **(Magic Square)** A magic square is a subject of recreational mathematics. It is an NxN matrix of the integers 1 to N^2 such that the sum of every row, column and diagonal is the same. In the figure there is an example magic square for the case N=5. In this example the common sum is 65.

	1	2	3	4	5
1	15	8	1	24	17
2	16	14	7	5	23
3	22	20	13	6	4
4	3	21	19	12	10
5	9	2	25	18	11

   When N is odd, H. Coxeter has given a simple rule for generating a magic square: Start with 1 in the middle of the top row; then go up and left assigning numbers in increasing order to empty squares; if you fall off the square imagine the same square as tiling the plane and continue; if a square is occupied, move down instead and continue.

   Make a program that gets N from the file "magic.in" and creates and writes the magic square into the file "magic.out" in a tabular format like in the sample output.

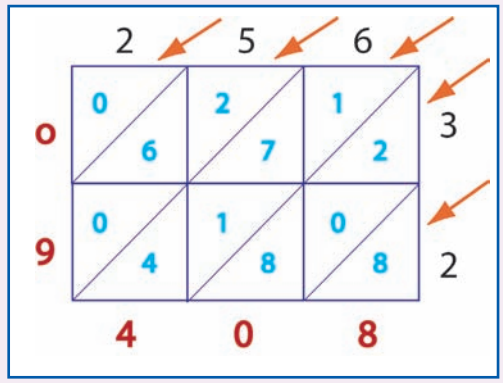
magic.in

```
3
```

magic.out

```
6 1 8
7 5 3
2 9 4
```

3. **(Lattice Multiplication)** Lattice multiplication is algorithmically equivalent to **long multiplication**. It requires the preparation of a lattice (a grid drawn on paper) which guides the calculation and separates all the multiplications from the additions.



As shown in the example, the multiplicand (256) and multiplier (32) are written above and to the right of a lattice. During the multiplication phase, the lattice is filled in with two-digit products of the corresponding digits labeling each row and column: the tens digit goes in the top-left corner. During the addition phase, the lattice is summed on the diagonals. Finally, if a carry phase is necessary, the answer as shown along the left and bottom sides of the lattice is converted to the normal form by carrying ten's digits as in long addition or multiplication.

Make a program that gets two long integers (up to 50 digits) and multiplies them and prints out the result.

Sample Input : 123456789 987654321

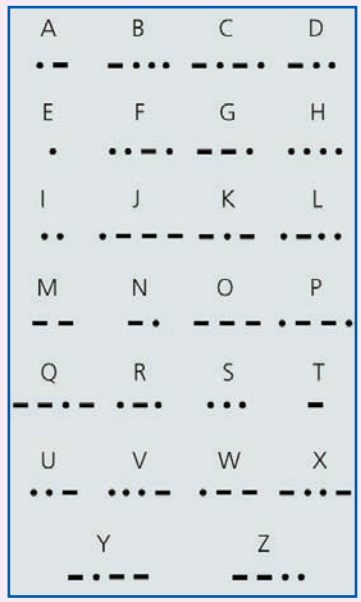
Sample output : 121932631112635269

4. **(Football Tournament)** A football tournament is organized among n teams. Teams are numbered from 1 to n. Each team will play with all others once. If n is an even number, there will be n tours, if it is an odd number, there will be n-1 tours. You are asked to arrange the fixture of the tournament. Your program will read N as the number of teams.

Your program should display a table of n lines. j^{th} number in i^{th} line representing the team that will play with i^{th} team in j^{th} week. Apparently if i^{th} team is playing with k in j^{th} week, team k must be playing with with i^{th} team in j^{th} week. If i^{th} team is not playing in j^{th} week then j^{th} number of i^{th} line must be 0.

football.in	football.out
3	2 3 0 1 0 3 0 1 2

5. **(Morse Code)** Morse code is a system of representing letters, numbers and punctuation marks by means of a code signal (dots and dashes) sent intermittently. It was developed by **Samuel Morse** and **Alfred Vail** in 1835. It is used as a standard code in telegraphs.



Make a program that reads a line of text containing only the upper case letters of the English alphabet, and encodes the text into Morse code. Leave a space between each morse-coded letter and three spaces each morse-coded word.

Also write a verification program that converts the morse code back into the original text.

Sample Input : MORSE CODE

Sample output : -- --- .-. ... . -.-. --- -.. .

6. **(Text Justifying)** Aligning text to both left and right sides and adding extra space between words as necessary is a common feature for advanced word processor programs. You are going to format a text document in this problem. Your program should justify the text on both sides, and minimize the biggest space between any two consecutive words in a line of the document.

The first line of the input file (unformatted document) has an integer number **N** (1<N<=100) indicating the length of the lines in the output file (formatted document). Each line must start with a word and end with a word in the formatted document (if there is only one word in the line, you must left justify it). Skip the blank lines, don't transfer the words from one line to another line, don't change the order of the words in a line, and don't concatenate the words.

You must calculate the quality point of your task after you finish it. At the beginning, the quality point is zero. If there is only one space between two consecutive words in a line, you must not change the quality point. If there are **S** (S>1) spaces between any two consecutive words in a line, you must increase your quality point by S*S. If there is only one word in a line, do not change the quality point. For the empty lines don't change the quality mark. You must write the quality point at the end of the file as a separate line. The smaller the quality point is, the better it is.

text.in

```
20
HELLO!
MY DEAR STUDENTS.

I HOPE YOU
LIKE THE QUESTION.
```

text.out

```
HELLO!
MY   DEAR   STUDENTS.
I          HOPE          YOU
LIKE   THE   QUESTION.
93
```

FLOWCHART PROGRAMMING (OPTIONAL)

Arrays and Strings

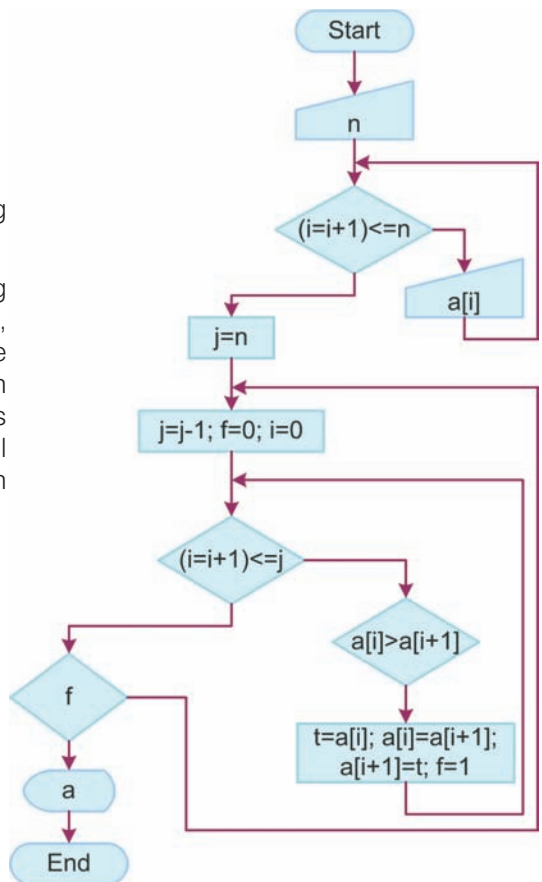
Arrays contain a sequence of other variables. Each variable can be accessed through an index, following the syntax: ArrayName [VariableIndex]. The indices are natural numbers (numbered from 0). FCPRO supports only the static arrays. Arrays may have one dimension as well as two or more dimensions. Strings contain a sequence of characters. FCPRO lets you read and print strings. But doesn't let you access the individual string elements.

Example: Bubble Sort

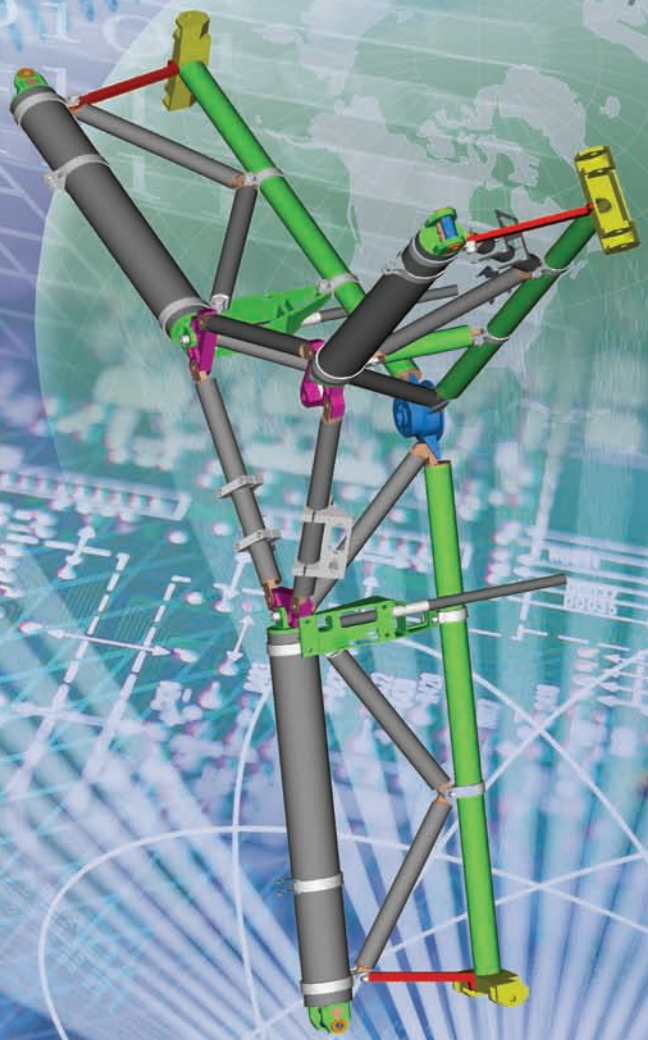
The **bubble sort** algorithm gets its name from the way smaller elements gradually "bubble" their way upward to the top of the array like air bubbles rising in water, while the larger elements sink to the bottom of the array.

Bubble sort is a simple sorting algorithm.

It works by repeatedly stepping through the list to be sorted, comparing two items at a time and swapping them if they are in the wrong order. The pass through the list is repeated until no swaps are needed, which means the list is sorted.



CHAPTER 6



- Understanding Structs
- Hierarchical Structs
- Array of Structs

C++

STRUCTS

Introduction

Struct (short for structure) is an aggregate data type for grouping related variables together. Structs may contain any kind of data type including other structs but not files. For example the following struct contains the related information for a person.

```
struct TPerson
{
     string  name;
     string surname;
     int age;
     string phoneNumber;
     string address;
     char gender;        //'M' or 'F'
};
```

Mergen
Micheal
25
987 378971
H-14, Green Park Extension Delhi - 110016
M

*A Sample **TPerson***

The keyword struct denotes the structure definition. Struct declares a new data-type. The identifier TPerson is the name of the structure and is used to declare the variables type of TPerson. The variables declared between the braces are the structures' members or fields.

Unlike with an array, the data items in a struct can be of different types. Also, with a struct the data items are identified by field name, whereas with an array the data items are identified through the use of an index number.

Declaring Structs and Accessing Members

The dot operator (.) is used to access members of structures. Each member of a structure can be used just like a normal variable, but its name will be a bit longer; name of the structure variable + '.' + name of the member. Here the dot is an operator which selects a member from a structure.

Reading and Printing Structures

The following example defines a structure that is TPerson and declares a variable person type of TPerson. Then the function getData reads the members of person, and the function print displays the members of person on the screen.

```
/*
PROG: c6_01person.cpp
person structure
*/
#include <iostream>
#include <string>

using namespace std;
```

```
struct TPerson
{
  string name;
  string phoneNumber;
  string address;
  char gender;              //'F' or 'M'
  int age;
  long ID;
};

void getData(TPerson &);   //Call by reference
void print(TPerson);       //Call by value

int main()
{
  TPerson person;          //Person is an variable type of TPerson
  getData(person);         //Read the person
  print(person);           //Print the person

  system ("pause");
  return 0;
}
//-----------------------------------------
void getData(TPerson &person)
{
  cout<<"Name = ?";
  getline(cin, person.name);
  cout<<"PhoneNumber = ?";
  getline(cin, person.phoneNumber);
  cout<<"Address = ?";
  getline(cin, person.address);
  cout<<"Gender [M or F] = ?";
  cin >> person.gender;
  cout<<"Age = ?";
  cin >> person.age;
  cout<<"ID = ?";
  cin >> person.ID;
}
//-----------------------------------------
void print(TPerson person)
{
  cout<<endl<<"Name is "<<person.name<<endl;
  cout<<"PhoneNumber is "<<person.phoneNumber<<endl;
  cout<<"Address is "<<person.address<<endl;
  cout<<"Age is "<<person.age<<endl;
  if (person.gender == 'M')
    cout<<"Gender is Male"<<endl;
  else cout<<"Gender isFemale"<<endl;
  cout<<"ID is "<<person.ID<<endl;
}
```

You can initialize structures while the declaration.

```
struct TPoint
{
        int x, y;
};
int main()
{
     TPoint p = {1, 2};
     cout<<p.x<<
     " "<<p.y<<endl;

     system("PAUSE");
     return 0;
}
```

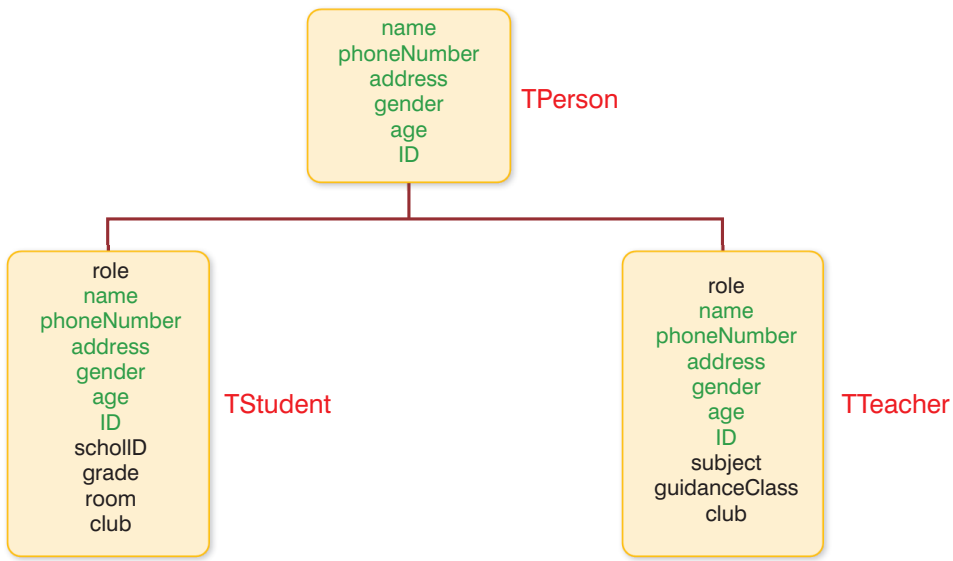
1 2

The **getline()** function reads a line of text and stores it into a string variable. **cin** reads only a word. If you need to read more than a single word, use the **getline()** function.

```
Name = ?Alan Black
PhoneNumber = ?505 4789822
Address = ?New Avenue Park Str. No:44
Gender [M or F] = ?M
Age = ?15
ID = ?30958070351

Name is Alan Black
PhoneNumber is 505 4789822
Address is New Avenue Park Str. No:44
Age is 15
Gender is Male
ID is -858993460
Press any key to continue . . .
```

Hierarchical Structures

Structures can be member of other structures. This kind of usage creates a **structures tree**. There are **parent** and **child** structures in a structures tree. A child structure is a member of its parent structure. **TPerson** is the parent structure of **TStudent** and **TTeacher** structures in the following example. Any child structure has all members of its parent and, in addition, usually has its own members as well.



Example: Struct of Structs

This example demonstrates how to declare a **struct** that has another **struct** as a member. **TStudent** and **TTeacher** structures have common members (fields) and different members. The common members are the personal information. Those common members are grouped within a new structure, that is **TPerson**, and included in both **TStudent** and **TTeacher** structs as members. Thus, **TPerson** is the parent of both **TStudent** and **TTeacher**.

The following program prompts a menu to the user to add a new student, to add a new teacher or to exit the program. It writes all the input to the file **"school.txt"** for later use.

```
/*
PROG: c6_02structofstruct.cpp
*/
#include <iostream>
#include <string>
#include <fstream>
using namespace std;

struct TPerson
{
  string name;
  string phoneNumber;
  string address;
  char gender;              //'F' or 'M'
  int age;
  long ID;
};

struct TStudent
{
  string role;
  TPerson person;
  long schoolID;
  string grade;
  string room;
  string club;
};

struct TTeacher
{
  string role;
  TPerson person;
  string subject;
  string guidanceClass;
  string club;
};
```

```
 int menu();
void newStudent();
void newTeacher();
TStudent getStudent();
TTeacher getTeacher();
void printStudent(TStudent);
void printTeacher (TTeacher);

ofstream fout("school.txt");

int main()
{
  int choice;
  do
  {
    choice = menu();
    switch(choice)
    {
    case 1: newStudent(); break;
    case 2: newTeacher(); break;
    };
  }
  while(choice!=3);

  fout.close();
  return 0;
}
//-----------------------------------------
int menu()
{
  int choice;
  system("cls");
  cout<<"1: New Student"<<endl;
  cout<<"2: New Teacher"<<endl;
  cout<<"3: Quit"<<endl<<endl;
  cout<<"Your choice? ";
  cin >> choice;
  return choice;
}
//-----------------------------------------
void newStudent()
{
  TStudent student;
  system("cls");
  student = getStudent();
  printStudent(student);
}
//-----------------------------------------
void newTeacher()
{
```

system("cls") is a Windows system function that clears the screen.

```
  TTeacher teacher;
  system("cls");
  teacher = getTeacher();
  printTeacher(teacher);
}
//-----------------------------------------
TStudent getStudent()
{
  TStudent std;
  std.role = "Student";
  getline(cin, std.person.name);
  cout << "Name and Surname= ?";
  getline(cin, std.person.name);
  cout << "PhoneNumber = ?";
  getline(cin, std.person.phoneNumber);
  cout << "Address = ?";
  getline(cin, std.person.address);
  cout << "Gender [M or F] = ?";
  cin >> std.person.gender;
  cout << "Age = ?";
  cin >> std.person.age;
  cout << "ID = ?";
  cin >> std.person.ID;
  cout << "schoolID = ?";
  cin >> std.schoolID;
  cout << "Grade = ?";
  cin >> std.grade;
  cout << "Room = ?";
  cin >> std.room;
  cout << "Club = ?";
  cin >> std.club;

  return std;
}
//-----------------------------------------
TTeacher getTeacher()
{
  TTeacher teacher;
  teacher.role = "Teacher";
  getline(cin, teacher.person.name);
  cout << "Name and Surname= ?";
  getline(cin, teacher.person.name);
  cout << "PhoneNumber = ?";
  getline(cin, teacher.person.phoneNumber);
  cout << "Address = ?";
  getline(cin, teacher.person.address);
  cout << "Gender [M or F] = ?";
  cin >> teacher.person.gender;
  cout << "Age = ?";
  cin >> teacher.person.age;
```

```
  cout << "ID = ?";
  cin >> teacher.person.ID;
  cout <<"Subject = ?";
  cin >> teacher.Subject;
  cout <<"guidanceClass = ?";
  cin >> teacher.guidanceClass;
  cout << "Club = ?";
  cin >> teacher.club;

  return teacher;
}
//-----------------------------------------
void printStudent(TStudent std)
{
  fout << std.role<<endl;
  fout << std.person.name<<endl;
  fout << std.person.phoneNumber<<endl;
  fout << std.person.address<<endl;
  fout << std.person.gender<<endl;
  fout << std.person.age<<endl;
  fout << std.person.ID<<endl;
  fout << std.schoolID<<endl;
  fout << std.grade<<endl;
  fout << std.room<<endl;
  fout << std.club<<endl<<endl;
}
//-----------------------------------------
void printTeacher(TTeacher teacher)
{
  fout << teacher.role<<endl;
  fout << teacher.person.name<<endl;
  fout << teacher.person.phoneNumber<<endl;
  fout << teacher.person.address<<endl;
  fout << teacher.person.gender<<endl;
  fout << teacher.person.age<<endl;
  fout << teacher.person.ID<<endl;
  fout << teacher.Subject<<endl;
  fout << teacher.guidanceClass<<endl;
  fout << teacher.club<<endl<<endl;
}
```

```
1: New Student
2: New Teacher
3: Quit

Your choice? 1
```

A sample "school.txt" file with two records:

```
Student
Maria Flora
389123894
Center Boulevard, A-1 Block, No:59
F
16
3082889944
2007135
10
A
Origami

Teacher
Alan Black
505 3398934
New Avenue, Park Str. No:44
M
45
309889994
Computer
10-A
Programming
```

Exercise: Struct of Structs

Modify the program above so that each student and teacher have a birthday instead of age. Use a separate struct for birthday and address like below:

```
struct TBirthday
{
     short year, month, day;
};

struct TAddress
{
     string city, state;
     string streetAddress;
     int zipCode;
};
```

Exercise: Points

Make a program that reads coordinates (x, y) of some points on the plane and prints the point that is closest to the origin (0, 0). Use a struct for points.

```
struct Tpoint
{
     int x, y;
};
```

Array of Structs

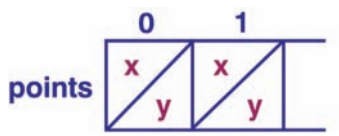
An array of structures is useful for temporarily storing a sequence of structures. For example, the following program reads many points as **structs** and keep them temporarily in an array to calculate the maximum distance between any two points.

Example: Distance between Two Furthest Points

The following program reads **N** points from the file **"points.txt"** and then calculates and prints the distance between two furthest points.

```
/*
PROG: c6_03distance.cpp
The longest distance between two points. Read N points from the
file points.txt and print the result to the standard output.
*/
#include <iostream>
#include <string>
#include <fstream>
#include <vector>
#include <math.h>
using namespace std;

struct TPoint
{
  int x, y;
};

void getPoints();          //read the points from the
                             file to an array
float go();                //calculate the maximum distance
```

```
 //return the distance between two points
float calcDistance(TPoint, TPoint); //function prototype

vector <TPoint> points;          //global declaration
//-----------------------------------------
int main()
{
  getPoints();
  cout <<"The biggest distance between two points is ";
  cout << go() <<endl;

  system("pause");
  return 0;
}
//-----------------------------------------
void getPoints()
{
  ifstream fin ("points.txt");
  int N;            //number of the points

  fin >> N;
  for (int i=1; i<=N; i++)
  {
    TPoint p;       //a single point
    fin>> p.x>> p.y;
    points.push_back(p);
  }
  fin.close();      //close the file "points.txt"
}
//-----------------------------------------
float go()
{
  float res = calcDistance(points[0], points[1]);

  for(int i=0; i<points.size()-1; i++)
    for(int j=i+1; j<points.size(); j++)
    {
       float tempDistance = calcDistance(points[i], points[j]);
       if (tempDistance > res)
              res = tempDistance;
    }
  return res;
}
//-----------------------------------------
float calcDistance(TPoint p1, TPoint p2)
{
  return sqrt((float)(p1.x-p2.x)*(p1.x-p2.x) +
              (p1.y-p2.y)*(p1.y-p2.y));
}
```



A Vector of Struct

A sample “points.txt” file:

```
5
0 0
1 1
0 -5
3 3
5 5
```

The output:

```
The biggest distance between two points is 11.18
Press any key to continue . . .
```

Exercise: Array as a Struct Member

Modify the program above so that instead of an **array of struct** use a record that keeps all the points in a **member array** like below:

```
struct TPoints
{
     float maxDistance;
     vector <int, int> pointArray;
};
```

Exercise: School Database

Improve the program **“struct of structs”** in this chapter to fulfill all the operations below. Do not use any additional file.

```
1: New Student
2: New Teacher
3: Delete a Record
4: Find
5: Print Students
6: Print Teachers
7: Students by Class
8: Quit

Your choice?
```

SUMMARY

A structure is created using the keyword **struct**.There is a strong relation between structures and classes in C++. We are going to study it in the classes chapter.

A **structure** is a collection of one or more variables grouped together under a single name for convenient handling. The variables in a structure are called **members** and may have any type, including arrays or other structures.

Dot operator ('.') is used to access elements of a structure. Structures can be used as elements of an array and parameters of a function as well as a return value of a function.

REVIEW QUESTIONS

1. Which structure is the best to work with a list of students in a school? The personal information and school information is necessary for each student.
   a) struct
   b) string
   c) struct array
   d) int array

2. Which of the following is not true?
   a) Any struct can be a member of another struct.
   b) Any struct can be a member of itself.
   c) Size of a struct varies depending on its members.
   d) A reference to a struct is allowed.

3. What is the size of the following struct in bytes.

```
struct TCar
{
    char array[10] name;
    int year;
    int price;
    single HP;
    single engineVolume;
};
```

   a) 22
   b) 20
   c) 13
   d) 5

4. What is the output of the following program?

```
#include <iostream>

using namespace std;

struct TTime
{
    int hour, minute, second;
};
float mystery(TTime &);

int main()
{
    TTime time = {23, 15, 20};
    cout << mystery(time)<<endl;
    return 0;
}
float mystery(TTime &t)
{
    return t.hour*60 + t.minute +
(float) t.second/60;
}
```

   a) 1395
   b) 1395.33
   c) 23
   d) 23.63

PROGRAMMING PROBLEMS

1. **(Sum)** Secondary school and high school students usually calculate the sum of the whole numbers without any difficulties. However; rational numbers may become a nightmare. Write a program that calculates the sum of several rational numbers given in fractional form like (a/b). Your program must give the result in the most simplified form.

   a/b + c/d = (a*d + b*c)/(b*d)

   Input

   The first line of input has a single integer N (2<=N<=100) that denotes the number of rational numbers to be added up. Each of the following N lines contains a pair of integers representing a rational number. The first number is the numerator (a) and the second number is the dominator (b).

   Output

   There is a single line with one or two numbers in the output. The first number is the numerator, and the second number is the dominator of the sum. Do not print the dominator when it is equal to 1. In any step of the addition process, the simplified numerator and the dominator don't exceed a 32-bit integer.

   sum.in

   ```
   3
   1 2
   3 4
   5 5
   ```

   sum.out

   ```
   9 4
   ```

2. **(Letter Pairs)** Your are given N letter pairs. All the letters are capital. If any letter belongs to two different pairs, those two pairs are considered related each other and put into the same set. Make a program that finds out all such sets.

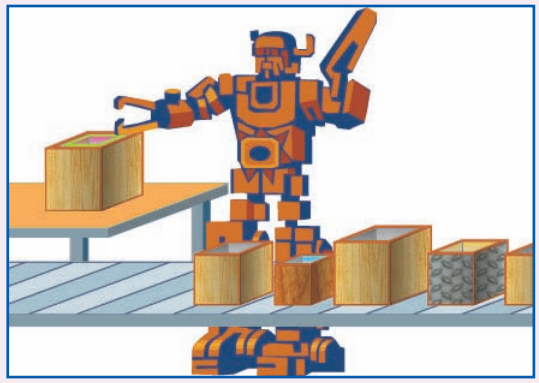
   pairs.in

   ```
   4
   A D
   F K
   A S
   K X
   ```

   pairs.out

   ```
   A S D
   F K X
   ```

3. **(Robot)** Although humans are far superior, robots are better at performing some basic repetitive tasks. In order for a robot to perform a useful function, it must be programmed. You are asked to write a program for a robot which packs the incoming boxes.



Some rectangular open boxes are moving in a straight line on the assembly line. The robot takes the first box and places the following box into the first box, then the third box into the second box, and so on. If any box doesn't fit inside the prior box, the robot sends a signal and another robot takes away the already packed boxes, and the first robot continues the same process with the next box.

The robot always takes the next available box from the assembly line, without skipping any boxes. The robot first checks each incoming box to determine if it will fit inside the open box. To fit a box inside another box, the width, length, and height of the first box must be smaller than the width, length, and height of the second box. The robot can rotate incoming boxes horizontally at 90 degree angles. The sides of the boxes are always parallel to each other.

What is the number of the packets after the robot finishes his task?

Input

The first line of input has a single integer N (2<=N<=1000) that denotes number of the boxes to be packed. Each of the following N lines contains a triple of integers representing the width, length, and height of a box.

Output

The output has a single integer representing the number of the packets after finishing the task.

robot.in

```
7
7 5 3
4 6 2
5 4 5
4 3 3
2 2 2
1 1 1
1 1 1
```

robot.out

```
3
```

4. **(Library)** Even though digital libraries are taking the role of classical libraries, classical libraries still retain their importance. Because so far, only some percentage of information in libraries have been transferred to a computer-readable form, and many readers (including me) like to touch, smell and flip the pages of books.

Computer software eases the burden of librarians. Such a software may keep titles, categories, authors, price, ISBN, availability in the library, borrowed day, returning day etc, of the books, customers information, lending a book, accepting and returning book, etc. for the librarian.

Make a program for your school library. Some of the operations your program should perform are below:

- New Book
- New Customer
- Remove a Book
- Remove a Customer
- Search for a book
- Report non-returning books
- List by Category
- List by Author
- Borrow a Book
- Book Return

FLOWCHART PROGRAMMING (OPTIONAL)

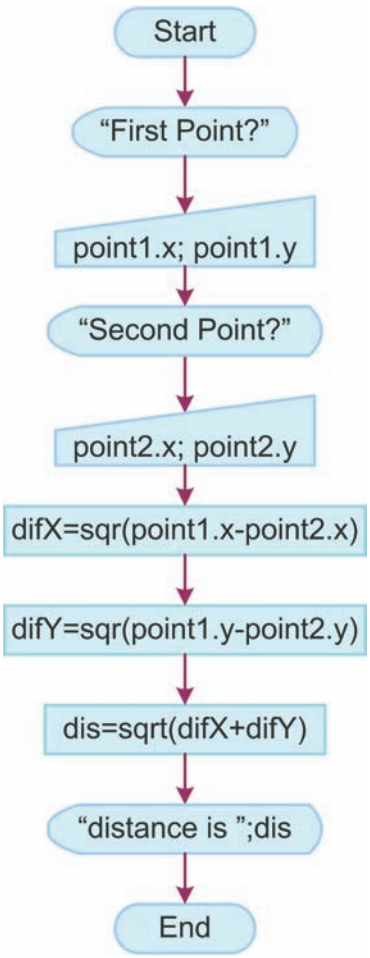
Structures

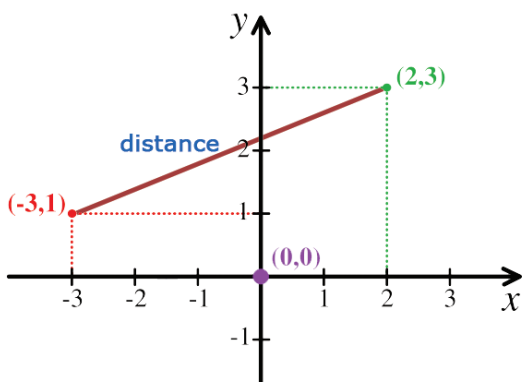
There is no special symbols for structures in flowchart programming. FCPRO manages structures as standard variables. Whenever you use a **dot operator (".")**, FCPRO understands that the identifier on the left of the operator is the name of a structure and the identifier on the right of the operator is a member of the structure.

When you are printing and assigning structures in FCPRO, you don't have to do it member by member with a **dot operator**. FCPRO allows you to print and assign structures directly with their names.

Example: Distance

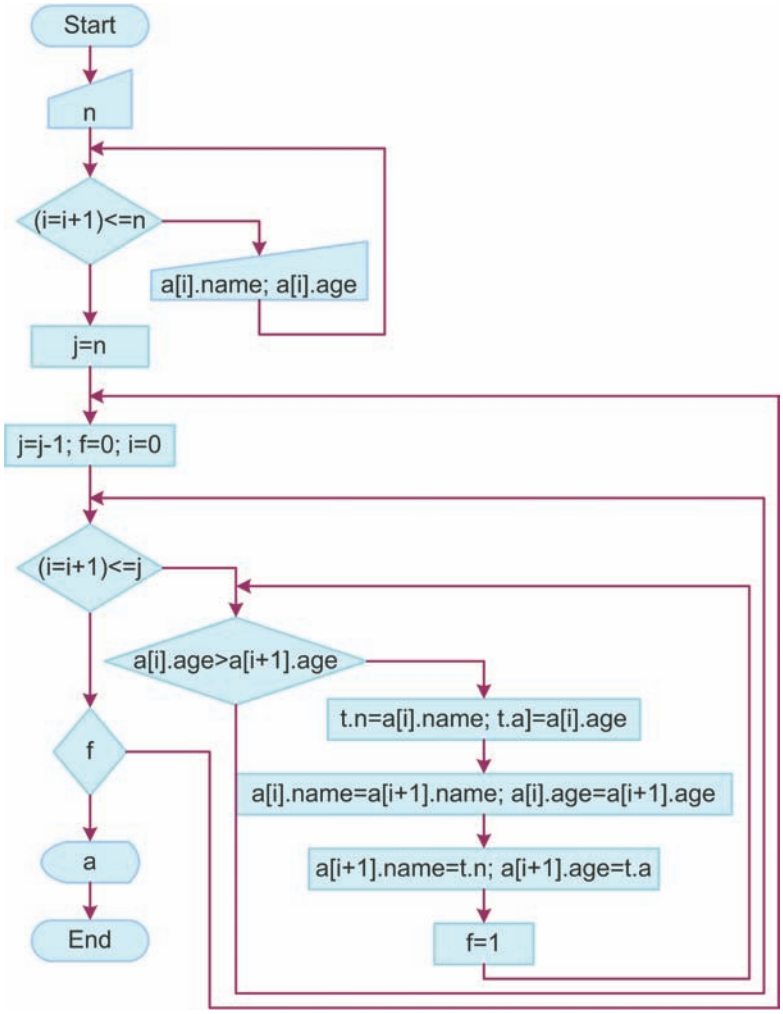
The following flowchart gets (x, y) coordinates of two points on the cartesian plane and then calculates and displays the distance between them.





Example: Names and Ages

The following flowchart reads the names and ages of a list of people and then sorts and prints the list by ages.



Exercise: Birthdays

Make a flowchart that reads the names and birthdays (year/month/day) of a list of people and then prints the names of people from the youngest to the oldest.

Your program should ask the user for the current date.

CHAPTER 7

OBJECT-ORIENTED PROGRAMMING

- Classes and Ojects
- Encapsulation
- Inheritance
- Polymorphism
- Operator Overloading

C++

Introduction

Sequential Programming

Modular Programming

Object-Oriented Programming

Programming has progressed a lot in a very short time. It started with a sequential programming in which the statements were following each other. Sequential programs are OK for basic tasks. However, solving a complex task with a sequential program is a challenging problem. With the increasing complexity of programs, programmers developed the modular programming technique after sequential Programming. A **module** is a sub-program (function in C++) that performs a specific task. Managing larger programs became easier with the help of modular programming. The focus of attention in modular programming is on modules (functions). It became possible for a group of programmers to work on a common project hence the software engineering concept emerged with modular programming.

Programmers have developed a new programming technique called **Object-Oriented Programming (OOP)**. In this technique, computer programs are made up of objects. Programmers manage complexity by managing the individual object instead of the whole program. The focus of attention in OOP is on classes. You should think about how each object will be designed and how these objects interact. Especially visual programs and the Internet programs increased the complexity of programming. OOP is the best programming technique to manage such complex programs.

Understanding Classes and Objects

Sometimes data members of a class ares called **attributes**, and member functions are called **methods** or **behaviours**.

We had variables and functions to manipulate variables in our programs so far. **Object-Oriented Programming (OOP)** encapsulates variables (data) and functions (methods) into packages called classes.

A **class** is the formal definition of an object, and an object is an instance of a class. A class is like a blueprint. Once a class has been defined, objects of that class can be declared from that class. A builder can build a house out of a blueprint. A programmer can instantiate (create) an object out of a class.

There is a strong relation between **structures** and **classes** in C++. Structures and classes both are user-defined data types. Instances of these data types are known as objects, and contain data and methods to manipulate the data. Classes are used commonly in convention when you encapsulate data and methods.

Member Accessibility

Class member access determines if a class member is accessible by other classes. Suppose that "day" is a member of class "date". Class member day can be declared to have one of the following levels of accessibility:public, private, or protected.

- **public:** day can be used anywhere by any class.
- **private:** day can be used only by the members of class date.
- **protected:** day can be used only by the members of class date, and the members of classes derived from class A.

Members of classes declared with the keyword **class** are private by default. Members of **structures** declared with the keyword **struct** are public by default.

Class Definition

Classes are described in a class declaration. A class declaration consists of a class header and body. The class header consists of a **class** keyword and the class name. The class body encapsulates the members of the class, which are the data members and member functions. Usually a class body contains only **prototypes** of member functions. Member functions are implemented after the class definition.

A **function prototype** is a function declaration or definition that includes both the return type of the function and the types of its arguments.

The following declaration defines a new class type **TDate** which has three private class members and six public member functions. The class members are **year**, **month** and **day**. The member functions are **setYear**, **setMonth**, **setDay**, **getYear**, **getMonth**, **getDay**, and **print**. Traditionally the set methods are used to alter the class members and get methods are used to retrieve the values of class members.

//PROG: c7_01class.cpp

```
class TDate
{
private:
  int year, month, day;
public:
  void setYear(int);
  void setMonth(int);
  void setDay(int);
  int getYear();
  int getMonth();
  int getDay();
  void print();
};
void TDate::setYear(int y)
{
  year = y;
}
void TDate::setMonth(int m)
{
  month = m;
}
void TDate::setDay(int d)
{
  day = d;
}
int TDate::getYear()
{
  return year;
}
int TDate::getMonth()
{
  return month;
}
int TDate::getDay()
{
  return day;
}
```

The private class members (year, month, and day) are hiding from other classes. They are accessible only via the public member functions of the class. This is called **data hiding**. Data hiding is a characteristic of OOP.

```
void TDate::print()
{
  cout<<day<<"/"<<month<<"/"<<year<<endl;
}
```

Reading and Printing a Class

The following program defines a class TDate and then creates an object of TDate by declaring a variable type of TDate (myDate). It demonstrates how to use public member functions to access the private class members in the main function. Please notice that, we can directly access only public members of the class. For example the statement "`cout<<myDate.day;`" would cause an error in your program.

```
//PROG: c7_02date.cpp
#include <iostream>
using namespace std;

class TDate          //TDate is a class.
{
private:
  int year, month, day;
public:
  void setYear(int);
  void setMonth(int);
  void setDay(int);
  int getYear();
  int getMonth();
  int getDay();
  void print();
};
void TDate::setYear(int y)
{
  year = y;
}
void TDate::setMonth(int m)
{
  month = m;
}
void TDate::setDay(int d)
{
  day = d;
}
int TDate::getYear()
{
  return year;
}
int TDate::getMonth()
{
  return month;
}
int TDate::getDay()
{
```

```
  return day;
}
void TDate::print()
{
  cout<<day<<"/"<<month<<"/"<<year<<endl;
}
int main()
{
  TDate myDate;            //myDate is an object
  myDate.setYear(2007);
  myDate.setMonth(7);
  myDate.setDay(12);
  cout<<"Year is "<<myDate.getYear()<<endl;
  cout<<"Month is "<<myDate.getMonth()<<endl;
  cout<<"Day is "<<myDate.getDay()<<endl;
  myDate.print();
  system("pause");  return 0;
}
```

```
Year is 2007
Month is 7
Day is 12
7/12/2007
Press any key to continue . . .
```

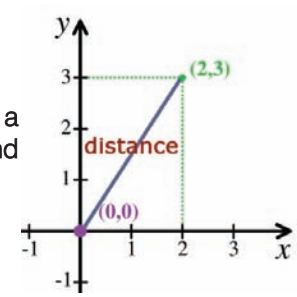
Exercise: Date Format

Modify the print function of the program above so that it prints the date in the format:

June 12, 2007

Example: Distance

Make a program that gets the coordinates (x, y) of a point and prints the distance between the origin and the point.



```
//PROG: c7_03distance.cpp

#include <iostream>
#include <math.h>

using namespace std;

class TPoint
{
private:
  int x, y;
public:
  void setX(int);
  void setY(int);
  double distance();
};
void TPoint::setX(int xx)
{
  x=xx;
}
void TPoint::setY(int yy)
{
  y=yy;
}
double TPoint::distance()
{
  return sqrt((double)x*x + y*y);
}

int main()
{
  TPoint p;
  int x, y;
  cout <<"x and y =?";
  cin >> x >> y;
  p.setX(x);
  p.setY(y);
  cout <<"The distance is "<<p.distance()<<endl;

  system("pause");
  return 0;
}
```

```
x and y =?3 2
The distance is 3.605
Press any key to continue . . .
```

Exercise: The Longest Distance

Make a program to calculate the longest distance between any two points in a set of points. Your program reads x and y coordinates of the points from the file **points.in** and write the result into the file **points.out**.

Use an array of class (**TPoint**) in your program. Modify the **distance** function of **TPoint** to calculate the distance between two points.

The prototype of the function is:

```
double distance(TPoint);
```

The implementation of the function is:

```
Double TPoint::distance(TPoint p2)
{
  double sqrt((p2.x -x)*(p2.x-x) + (p2.y-y)*(p2.y-y));
}
```

The Class Constructor and Initializing Class Members

A **constructor** is a special method on a class that initializes the class members. Constructors have the same name as their class and is executed whenever that object comes into existence. You do not specify a return type for a constructor.

The following program calculates the distance from a point to the origin. The point is been initialized to (3, 4) in the constructor function. The other functions (setX, setY, and distance) have been implemented directly in the body of the class.

```
//PROG: c7_04constructor.cpp
#include <iostream>
#include <math.h>
using namespace std;

class TPoint
{
private:
  int x, y;
public:
  TPoint(){ x=3; y=4;}            //Constructor function
  void setX(int xx){ x=xx;}
  void setY(int yy){ y=yy;}
  double distance()
  {
    return sqrt((double)x*x + y*y);
  }
};
```

Destructors are functions with no arguments (parameters) that are called whenever an object of the class is destroyed. Destructors are declared with the same name as the class except that they are preceded with a "~". All the objects are destroyed when the program is over.

```
class TPoint
{
private:
  int x, y;
public:
  TPoint(){ x=3; y=4;}
  ~TPoint()
  { cout<<"bye!";}
  .
  .
  .
```

```
int main()
{
  TPoint p;
  cout <<"The distance is "<<p.distance()<<endl;
  system("pause");
  return 0;
}
```

```
The distance is 5
Press any key to continue . . .
```

Object-Oriented Techniques

Object-oriented programming is built on three pillars: encapsulation, inheritance, and polymorphism.

Encapsulation involves **hiding data** of a class and allowing access only through a public interface.

a. Encapsulation and Data Hiding

OOP encloses data and the functions manipulate that data all within an object Holding the data and the related method in the same container is called **encapsulation**. Although classes contain both data and functions, not all of the class is accessible to other classes. Programmers let other classes access only the public methods of a class, and keep the rest of the class hidden in a private section. For example, let's think a TV set as an object. You never need to open it to operate it, instead the remote controller or the buttons on TV are enough to operate the TV set.

Overriding is replacing a method in a base class with a specific version in a derived class. That is, redefining a method from a parent class in a child class.

b. Inheritance

New classes can be created from existing classes. This technique provides software re-usability to save time in program development. The new class inherits the data members and member functions of the existing class. The new class is called "**derived class**" and the existing class is called "**base class**". A derived class can add new data members and member functions or its own, or **override** its inherited methods. Thus, a derived class is more specific than its base class.

Don't confuse the concepts of **overloading** and **overriding**.

Overloading deals with multiple methods in the same class with the same name but different parameter lists.

Overriding deals with two methods, one in a parent class and one in a child class, that have the same name and same parameter lists but different operator definition.

c. Polymorphism

There are two main advantages of inheritance: code reuse and polymorphism. C++ and other OOP languages allow objects of different types to respond differently to the same function call. This feature is called polymorphism. Derived classes **override** the methods of base classes in polymorphism. For example, let **TRectangle** and **TTriangle** classes be two different derived classes of the base class **TShape**, and let **calculateArea** be a method of the class **TShape**. The derived classes **TRectangle** and **TTriangle** override the **calclateArea** method to calculate their own areas.

Inheritance

Object-oriented programming allows classes to inherit commonly used class members and member functions from other classes. In the following example **TStudent** class is derived from **TPerson** class. Thus, **TPerson** is the **base class**, and **TStudent** is the **derived class**.

There are three types of inheritance: public, protected, and private. Protected and private inheritance are used rarely and they are beyond the scope of this book.

In the **public inheritance**, **public members** of the base class become public members of the derived class and protected members of the base class become protected members of the derived class. Private members are not inherited. They can be accessed with the help of public methods of the base class.

The syntax for creating a derived class is simple. At the header of your class declaration, add semicolon (**":"**) and **public** keyword, followed by the name of the class to inherit from:

```
  class TStudent:public TPerson  //TPerson is the base class
```

```
//PROG: c7_05inheritance.cpp
#include <iostream>
#include <string>
using namespace std;

class TPerson       //base class
{
protected:
  string nameSurname;
  int age;
  char gender;      //'M' or 'F'
public:
  void setNameSurname(string ns){ nameSurname =ns;}
  void setAge(int a){ age=a;}
  void setGenger(char g){ gender = g;}
  string getNameSurname(){return nameSurname;}
  int getAge(){ return age;}
  char getGender(){ return gender;}
};

class TStudent:public TPerson    //public inheritance
{
private:
  long schoolID;
  int grade;
  string classRoom;
public:
  void setSchoolID(long id){schoolID=id;}
  void setGrade(int g){grade=g;}
  void setClass(string c){classRoom=c;}
  long getSchoolID(){return schoolID;}
```

```
  int getGrade(){return grade;}
  string getClass(){return classRoom;}
};

int main()
{
  TPerson person;
  person.setNameSurname("Alan George is a");
  person.setAge(17);
  person.setGenger('M');
  cout<<person.getNameSurname()<<" ";
  cout<<person.getAge()<<" years old, ";
  if (person.getGender()=='M')
    cout<<"boy."<<endl;
  else cout<<"girl."<<endl<<endl;

  TStudent student;
  student.setNameSurname("Maria Orwell is a");
  student.setAge(16);
  student.setGenger('F');
  student.setSchoolID(2007195);
  student.setGrade(10);
  student.setClass("B");
  cout<<student.getNameSurname()<<" ";
  cout<<student.getAge()<<" years old, ";
  if (student.getGender()=='M')
    cout<<"boy."<<endl;
  else cout<<"girl."<<endl;
  cout<<"ScoolID is "<<student.getSchoolID()<<", ";
  cout<<"grade is "<<student.getGrade()<<", ";
  cout<<"classroom is "<<student.getClass()<<"."<<endl;

  system("pause");
  return 0;
}
```

```
Alan George is a 17 years old, boy.

Maria Orwell is a16 years old, girl.
ScoolID is 2007195, grade is 10, classroom is B.
Press any key to continue . . .
```

Polymorphism

Base classes and **virtual functions** are key concepts in **polymorphism**. A virtual function is a prototype of a function preceding with the keyword **virtual** in the generic base class. The derived classes override those virtual functions so that the same function in different derived classes performs the same task in a specific way.

For example, in the following program, base class **TShape** contains the **getArea** virtual function. Assume that **TRectangle**, **TTriangle** and **TCircle**, are all derived classes from **TShape**. Now the member function **getArea()** takes a different action depending on the shape.

```
/*
PROG: c7_06polymorphism
Virtual functions, function Overriding and Polymorphism
*/
#include<iostream>
#include<string>
using namespace std;

class TShape        //base class
{
protected:          //class members are protected
  int width, height;
public:
  virtual double getArea() {return 0;}  //virtual function
  virtual void set(int, int){};         //virtual function

};

class TRectangle:public TShape    //derived class
{
public:
  double getArea()                //polymorphic function
  {
    return width*height;
  }
  void set (int a, int b)         //polymorphic function
  {
    width = a;
    height = b;
  }
};

class TTriangle:public TShape     //derived class
{
public:
  double getArea()                //polymorphic function
  {
    return (double)width*height/2;
  }
  void set (int a, int b)         //polymorphic function
  {
    width = a;
    height = b;
  }
};
```

C++ **virtual function** is a member function of a class, whose functionality can be over-ridden in its derived classes. Virtual functions are usually defined with a minimal functionality.

Pure virtual functions are left without implementation. Any class that has at least one pure virtual function is called an **abstract base class**. You cannot create instances (objects) of abstract base classes.

virtual double getArea() = 0;

```
class TCircle:public TShape
{
private:
  double radius;
public:
  double getArea()         //polymorphic function
  {
    return 3.14*radius*radius;
  }
  void set(double r)       //polymorphic function
  {
    radius = r;
  }
};

int main()
{
  TRectangle rec;   int a, b;
  cout <<"Sides of the rectangle?"<<endl;
  cin >> a>> b;
  rec.set(a, b);
  cout<<"Area of the rectangle is "<<rec.getArea()<<endl;

  TTriangle triangle;
  cout <<"Base and height of the triangle?"<<endl;
  cin >> a >> b;
  triangle.set(a, b);
  cout<<"Area of the triangle is "<<triangle.getArea()<<endl;

  TCircle circle;   double radius;
  cout <<"Radius of the circle:?"<<endl;
  cin >> radius;
  circle.set(radius);
  cout<<"Area of the circle is "<<circle.getArea()<<endl;

  system("Pause");  return 0;
}
```

```
Sides of the rectangle?
3 5
Area of the rectangle is 15
Base and height of the triangle?
7 3
Area of the triangle is 10.5
Radius of the circle:?
1.3
Area of the circle is 5.3066
Press any key to continue . . .
```

Exercise: Shapes

Modify the program above to calculate the perimeters of a rectangle, a triangle and the circumference of a circle in addition of their areas.

Operator Overloading

Operator overloading is a specific case of polymorphism in which some or all of operators like <, +, =, or == have different implementations depending on the types of their arguments. Usually user-defined types require operator overloading. An overloaded operator is called an **operator function**. You declare an operator function with the keyword **operator** preceding the operator.

Many **STL** containers (set, map) and algorithms such as sort, works only after overloading less than ('<') operator with user-defined types.

The **Standard Template Library**, or **STL**, is a C++ library of container classes (vector, list, stack, queue, set, map etc), and algorithms (sort, merge, binary search, swap, min, max, next permutation etc).

Overloading Equal, Assignment, and Smaller Than Operators

The following program demonstrates how to overloaded '==', '=', and '<' operators with in the class **TPoint**. Equal and smaller than operators return a boolean value whereas, an assignment operator returns an object.

```
/*
PROG: c7_07operator.cpp
Operator overloading
*/
#include<iostream>
#include<string>
using namespace std;

class TPoint
{
private:
  int x, y;
public:
  TPoint (){x=0; y=0;}
  void set(int xx, int yy)
  {
    x = xx;  y = yy;
  }
  TPoint& operator = (const TPoint&);
  bool operator == (const TPoint&);
  bool operator < (const TPoint&);
  void print()
  {
    cout <<x<<" "<<y<<endl;
  }
};
```

Many C++ programmers prefer **friend functions** for operator overloading. Friend functions are functions defined outside a class (ie, not member functions), but which the class declares to be friends so that they can access the class's private members.

```
class TPoint
{
friend bool operator<
(const TPoint&, const
TPoint&);

private:
  int x, y;
public:
  .
  .
  .
};
bool  operator<(const
TPoint&   p1,   const
TPoint& p2)
{
  return (p1.x<p2.x ||
  ( p1.x==p2.x &&
    p2.y<p2.y));
}
.
.
.
```

The keyword **this** represents a pointer to the object itself.

```
TPoint& TPoint::operator =(const TPoint& p2)
{
  x = p2.x;
  y = p2.y;
  return *this;
}
bool TPoint::operator ==(const TPoint& p2)
{
  return (x==p2.x && y==p2.y);
}
bool TPoint::operator <(const TPoint& p2)
{
  return (x<p2.x || (x==p2.x && y<p2.y));
}

int main()
{
  TPoint p1, p2;
  p2.print();              //prints "0 0"
  p1.set(1, 2);            //p1 gets (1,2)
  cout<< (p2<p1) <<endl;   //prints "1", that is true
  p2 = p1;                 //p2 gets p1, that is (1, 2)
  p2.print();              //prints "1, 2"
  if (p1==p2)
    cout<<"duplicated points."<<endl;
  cout<< (p1<p2) <<endl;   //prints "0", that is false
  system("pause");  return 0;
}
```

```
0 0
1
1 2
duplicated points.
0
Press any key to continue . . .
```

Example: Sorting the Points

Make a program that randomly generates coordinates (x, y) of some points on the plane and prints the points starting from the most left-button point and ending with the right-up point.

STL offers some common containers and algorithms. Sorting some items is one of the most used algorithm. STL sort algorithm sorts the elements of a vector in a range.

The following program generates some random points, keeps them in a vector, sorts the points in the vector and prints the vector before and after the sorting process.

The sorting algorithm needs to compare the items (points). The '<' operator must be defined for the comparison.

```
//PROG: c7_08sorting.cpp
#include<iostream>
#include<string>
#include<vector>
#include<algorithm>        //for sorting algorithm
#include<time.h>           //for srand and rand functions
using namespace std;

class TPoint
{
private:
  int x, y;
public:
  TPoint (){x=0; y=0;}
  void set(int xx, int yy)
  {
    x = xx;  y = yy;
  }
  //sort algorithm works only after defining the '<' operator.
  bool operator < (const TPoint&);
  void print()
  {
    cout <<"("<<x<<", "<<y<<") "<<endl;
  }
};
bool TPoint::operator <(const TPoint& p2)
{
  return (x<p2.x || (x==p2.x && y<p2.y));
}

//TPointVector is a user-defined type
typedef vector<TPoint> TPointVector;

//pass a reference to modify the vector
void generatePoints(TPointVector &);
//pass a reference to save space
void printVector(TPointVector &);

int main()
{
  TPointVector points;
  int n;
  cout<<"How many points? ";
  cin >> n;
```

C++ allows the definition of user-defined types based on other existing data types. You can define your own types using the keyword **typedef.**

```
typedef existing_type
new_type_name;
```

Begin returns an iterator (pointer to an item) referring to the first element in the vector container and **end** returns an iterator referring to the past-the-end element in the vector container.

srand() function initializes the random number generator so that **rand()** function generates a different succession of results in the subsequent calls.

```
  points.resize(n); //size of the points has been set to n
  generatePoints(points);
  cout<<"Original points:"<<endl;
  printVector(points);
  sort(points.begin(),points.end()); //sort the vector points
  cout<<"After sorting the points:"<<endl;
  printVector(points);
  system("pause");  return 0;
}

void generatePoints(TPointVector& points)
/*
Generate random points whose coordinates
are between 0 and 99.
*/
{
  srand (time(NULL));      //Initialize random number generator
  for (int i=0; i<points.size(); i++)
    points[i].set(rand()%100, rand()%100);
};

void printVector(TPointVector& points)
//Print all the points in the vector points
{
  for (int i=0; i<points.size(); i++)
    points[i].print();
  cout<<endl;
}
```

```
How many points? 5
Original points:
(68, 63)
(26, 86)
(22, 90)
(14, 27)
(27, 89)

After sorting the points:
(14, 27)
(22, 90)
(26, 86)
(27, 89)
(68, 63)

Press any key to continue . . .
```

SUMMARY

An **object** is a software bundle of related data (variables) and methods (functions). In **Object-Oriented Programming**, software objects are often used to model the real-world objects that you find in everyday life. A **class** is a blueprint or prototype from which objects are created.

Encapsulation, **inheritance** and **polymorphism** are three fundamental principles of OOP (Object Oriented Programming). **Encapsulation** is the grouping of data and the code that manipulates it into a single object. **Inheritance** is a way to form new classes using pre-defined objects or classes where new ones simply take over old ones' implementations and characteristics. It is intended to help the re-use of existing code with little or no modification. **Polymorphism** allows objects to be represented in multiple forms. Even though classes are derived or inherited from the same parent class, each derived class will have its own behavior.

Class members and member functions are not open to other classes. Private, protected and public sections determine the accessibility level of class members and methods by other classes. **Public** section is accessible by all other classes, **protected** section is accessible by derived (child) classes, and **private** section is not accessible by other classes. There is no accessibility restriction for **friend functions** and **friend classes**.

It's can be useful to define a meaning for an existing operator for objects of a new class. This technique is called **operator overloading**. The purpose of operator overloading is to make programs clearer by using conventional meanings for '<', '==', +, etc.

C++'s **Standard Template Library (STL)**, standardized in 1999, solves many standard data structure and algorithm problems.

REVIEW QUESTIONS

1. Which programming technique is the most comprehensive?
   a) sequential
   b) modular
   c) object-oriented
   d) classical

2. Members of struct by default are _____, and member of class by default are _____.
   a) public - private
   b) private - public
   c) public - public
   d) private - private

3. _____ members of a class are accessible by other classes, however _____ members of a class are not accessible by any other classes?
   a) public - private
   b) public - protected
   c) private - protected
   d) private - public

4. What is the output of the following program?

```
#include <iostream>
using namespace std;
class myClass
{
private:
    int a, b;
public:
  myClass(){a=1; b=2;}
```

```
  void set(int aa, int bb)
          { a=aa; b=bb;}
  void print()
    { cout <<a<<" "<<b<<endl;}
};
int main()
{
    myClass x;
    x.print();
    x.set(5,6);
    x.print();

    return 0;
}
```

5. What is wrong in the following code?

```
class myClass
{
private:
          int a, b;
public:
    myClass(){a=1; b=2;}
    .
    .
    .
};

int main()
{
    myClass x = {3, 5};
    .
    .
    .
};
```

6. What is the output of the following program?

```
#include <iostream>
using namespace std;
class myClass
{
private:
         int a, b;
public:
        void set(int x)
        {  a=b=x;}
        void set(int x, int y)
        { a=x; b=y;}
        void print()
        {cout <<a<<" "<<b<<endl;}
};
int main()
{
    myClass x;
    x.set(5);
    x.print();
    x.set(3, 9);
    x.print();
    return 0;
}
```

7. What is the output of the following program?

```
#include <iostream>
using namespace std;
class baseClass
{
protected:
         int a, b;
public:
        void set(int x, int y)
        { a=x; b=y;}
        virtual double process(){};
};

class derivedClass1:public baseClass
{
public:
        double process()
        {  return a+b;}
};

class derivedClass2:public baseClass
{
public:
        double process()
        { return a*b;}
};

int main()
{
   derivedClass1 a;
 derivedClass2 b;
   a.set(3,5);
   b.set(3,5);
   cout<<a.process()<<endl;
```

```
    cout<<b.process()<<endl;
    return 0;
}
```

8. What is the output of the following program?

```
#include <iostream>
using namespace std;
class TPoint
{
private:
    int x,y;
public:
    TPoint& operator +(const TPoint&);
    TPoint& operator -(const TPoint&);
    bool operator <(const TPoint&);
    void set(int xx, int yy)
          { x=xx; y=yy;}
    void print()
          {cout<<x<<" "<<y<<endl;}
};
TPoint& TPoint::operator+
(const TPoint& p2)
{
    x += p2.x;
    y += p2.y;
    return *this;
};
TPoint& TPoint::operator-
(const TPoint& p2)
{
    x -= p2.x;
    y -= p2.y;
    return *this;
};
bool TPoint::operator<(const TPoint& p2)
{
   return (x+y)>(p2.x + p2.y);
};
int main()
{
     TPoint p1, p2;
     p1.set(3,5);
     p2.set(4,2);
     if (p1<p2)
        p1 = p1+p2;
     else
         p1 = p1-p2;
     p1.print();
     p2.print();
     return 0;
}
```

9. What is the output of the following program? ("recFunction" is a recursive function. Recursive functions call themselves.)

```
#include <iostream>
using namespace std;
class TMystery
{
public:
    void recFunction (int);
};
void TMystery::recFunction(int x)
{
    if (x>0)
    {
          cout<<x<<" ";
          recFunction(x-1);
    }
}

int main()
{
   TMystery c;
   c.recFunction(10);
   system("pause");
   return 0;
}
```

PROGRAMMING PROBLEMS

1. **(Chess)** Chess is known as one of the most famous and oldest mental games. It is played between two players. One of the goals of early computer scientists was to create a chess-playing machine, and today's chess is deeply influenced by the abilities of current chess programs. In 1997, a match between Garry Kasparov, then World Champion, and IBM's Deep Blue chess program proved for the first time that computers are able to beat even the strongest human players.

You are going to make a part of a chess program. Suppose that you are in the middle of a chess game and you don't want to make a big mistake not to lose the game. Make a program that gets all the pieces and their positions on a chess-board and tells you all your pieces that are threatened when it is your turn to play.

Input

The first line of the input contains an integer N (2 <= N <= 32) that represents the number of all the pieces on the board. Each of the following N lines contains three letters and an integer: The first letter denotes the color of a piece and the second letter denotes the type of a piece. The third letter and the integer denotes the position of a piece. The color is 'B' (Black) or 'W' (White) and the type is 'S' (Shah or King), W (Wazir or Queen), B (Bishop), K (Knight), R (Rook), and P (Pawn). Your color is white.

Output

The output contains positions and types of your pieces that are threatened by your opponent.

chess.in

```
24
B R A 8      B P G 7
B W A 6      W P G 4
W P A 3      B K G 3
W R A 1      W P F 3
W S E 1      W P E 3
W K G 1      B P B 7
W R H 1      W P B 4
W P H 3      W W C 2
B P H 7      W K D 4
B R H 8      B P C 6
B K G 8      B B D 6
B S E 8      B P F 7
```

chess.out

```
S E 1
P A 3
P B 4
R H 1
```

ANSWER KEY

Chapter 1

1. I am
   learning C++
2. 1 23 5
3. 1 2 3
   5 -1 3
   0 1 2
4. unsigned short, string,char, bool, unsigned short, float, int

Chapter 2

1. if, if/else, switch, ?
2. if (a>b)
   c=a;
   else
   c=b;
3. 21
4. c

Chapter 3

1. b
2. d
3. a
4. c
5. c
6. D C B A
   D C B
   D C
   D
7. 0 2 16

Chapter 4

1. b
2. a
3. c
4. 1 3
   4 8
5. 1 3
   3 2
   1 3
6. 5 11
7. 4 5
8. Reads three integers, and then prints them sorted in increasing order.
9. Reads an integer, and then prints its digits in reverse order, separated with a space.
10. Calculates and displays sum of the prime numbers between 1 and N.
11. Reads two integers (B and P), and then calculates and prints B to the power P.
12. This program prints the Fibonacci numbers smaller than or equal to N, instead of the first N Fibonacci numbers.
13. The go function returns the least significant digit before cutting it form the number. Thus, N remains the same and the while structure becomes an infinite loop.

Chapter 5

1. a
2. c
3. 20
4. 5
5. 4
6. 14
   15
7. Generates two random binary vectors with K elements, and prints out, in how many tries the program generated the vectors.
8. Reads a line of text, and then converts the lower case letters to upper case, and upper case letters to lower case in the text.

Chapter 6

1. c
2. b
3. a
4. b

Chapter 7

1. c
2. a
3. a
4. 1 2
   5 6
5. Objects of myClass had already been initialized in the constructed function. Attempting to initialize the object x in the declaration causes an error.
6. 5 5
   3 9
7. 8
   15
8. 7 7
   4 2
9. 10 9 8 7 6 5 4 3 2 1

INDEX

G

H

I

J

L

M

N

O

P

Q

R

S

T

U

V

W

X

Z