Chapter I

Basics of C++ Programming

C++ is an object oriented programming (OOP) language that can be extended through the use of classes. A class defines a set of member variables and a set member functions. Member functions define the various operations that can be carried-out on the member variables of the class. In general, member variables can be accessed only through member functions of the same class. An instance of a class is called an object and hence the term “object-oriented programming” or OOP. An instance of a built-in type is called a variable.

Before we delve into OOP programming or class design we will cover the basics of C++ programming. A simple C++ program is shown next:

```
// A simple C++ program
#include <iostream.h>
#include <conio.h>

void main()
  {
  cout << "Hello " << endl;
  cout << "In this book we will learn about OOP" << endl;
  getch();
  }
```

The above program begins with a comment statement preceded by //. In C++ you can include comments by using one of the following forms:

// Comment, proceeds to the end of the line

or

/* multi-line
Comments. */

C++ programs for console type application, i.e. non Windows type application, must include

a function **main()**. Execution of the program always starts from main(). The function main() returns by default an integer value. A return value is a value returned to the program in which execution of the function was initiated, in this case the operating system. You can override the default by preceding main() with the word **void**, i.e, no return value.

The above simple C++ program contains two **#include** instructions. These are directives to the compiler to insert the file named between the brackets < > in the above code. The first include statement includes a file named **iostream.h**. This file defines the input-output streams used in transferring data from or to the program, such as in:

```
cout << "Hello " << endl;
```

cout is defined in iostream.h and the operator "<< " is for inserting the string "Hello" in the output stream or basically on your computer monitor. The **endl** instructs the computer to move to the beginning of a new line after typing Hello. The insertion operator can be cascaded as shown in the above statement.

A C++ statement is always terminated by a semicolon and can extend over several lines.

The library function **getch()** waits for the user to press any key on the keyboard. Since getch() is the last instruction, once a key is pressed the program terminates. getch() is defined in **conio.h**.

Braces { } are used to outline the start and end of a block of statements. In this case the start and end of the program.

1.1 Basics of number Representations.

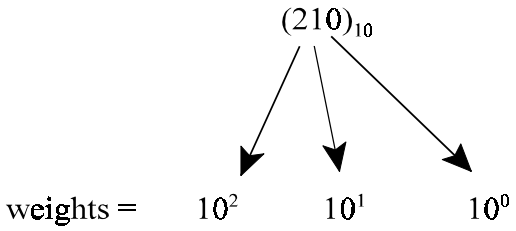
Although you can develop programs without knowing anything about number representations. However, a basic understanding of how numbers are represented will clarify such computer terms as byte, bit, etc.

The number system we are mostly accustomed to is the decimal number system. In this number system only the following digits are allowed:

0 1 2 3 4 5 6 7 8 9

Any number larger than 9 is written using multiple weights of 10. For example, the number 210 is

actually:



or $(210)_{10} = 2\times10^2 + 1\times10^1 + 0\times10^0$.

The weighting system used in the decimal number system extends to other number representations. The weight or base of the decimal system is 10.

Binary number representation has a base of 2 and only 0 and 1 are valid digits. For example the binary number 1101 can be written as:

$$(1101)_2 = 1\times2^3 + 1\times2^2 + 0\times2^1 + 1\times2^0 = 8 + 4 + 0 + 1 = (13)_{10}$$

The digit with the highest weight (the leftmost digit) is called the most significant digit and that with the smallest weight is called the least significant digit.

Octal number representation has a base of 8 and only digits from 0 to 7 are allowed. For example

$$(1127)_8 = 1\times 8^3 + 1\times 8^2 + 2\times 8^1 + 7 \times 8^0 = 512 + 64 + 16 + 7 = (599)_{10}$$

Hexadecimal number representation has a base 16 and only numbers from 0 to 15 are allowed. The following are valid hexadecimal digits:

0 1 2 3 4 5 6 7 8 9 A B C D E F

where A=10, B=11, C=12, D=13, E=14, F=15.

An example of a Hexadecimal number is

$$(12EF)_{16} = 1 \times 16^3 + 2 \times 16^2 + 14 \times 16^1 + 15 \times 16^0 = (4847)_{10}$$

In C++ any number that is preceded with a zero is interpreted as an octal number and any number that is preceded with 0x is interpreted as a hexadecimal number. The following program illustrates number interpretation. The program will print out the numbers in decimal representation. If you wish the numbers to be printed as octal and hexadecimal representations, respectively, then you will need

to precede the octal number with **oct** and hexadecimal with **hex** as follows:

cout << oct << 01127; and

cout << hex << 0x12EF;

You can use: cout << dec << 01127; to print an octal number in decimal representation.

```
#include <iostream.h>
#include <conio.h>
main()
 {
 cout << 01127 << endl;  // Octal number equal to 599 decimal
 cout << 0x12EF << endl; // Hexadecimal number equal to 4847 decimal
 getch();
 }
```

Internally all numbers are stored in the computer as binary numbers, e.g. 01101110. Each digit of a binary number is called a bit. An 8 bit number is called a byte (Figure 1.)

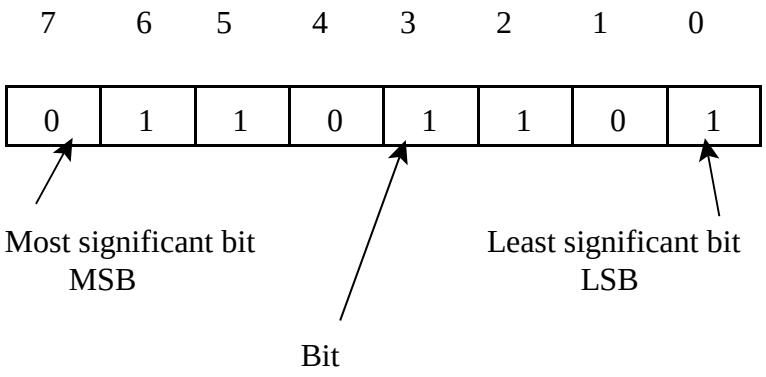


Figure 1. A one byte = 8 bits

1.2 Built-in variable types in C++

Variables in C++ can be of type integer (declared as **int**) or real (declared as **float**) or character (declared as **char**) or wide character (declared as **wchar_t**). All variables to be used in a program must be declared prior to usage. Thus:

```
int i, j;        // i and j are variables of type integer
float a, k;    // a and k are variables of type float
char ch, c;    // ch and c are variables of type character
```

Variables can be declared anywhere in the program prior to usage, but usually are declared at the beginning of the program. C++ is case sensitive, that is, variable I is a different variable than i, and variable MAX is different than Max or mAx. Variable names can be up to 32 characters and can start with any character from the alphabet or the underscore character '_'. Variable names can include numbers, such as _Max1 or var2.

int can be of type **short**, **long**, **unsigned**, or **long unsigned**. Thus integer variables can be declared as follows:

```
short int i, j, k;       // i, j and k are short integers
short i, j;          // same as short integer
```

In 32 bit C++, **short** is 2 bytes, **int** and **long** are 4 bytes and **unsigned** and **long unsigned** are 4 bytes. The following table summarizes the different types:

Table I

Type	size in bytes	Range
int	4 bytes	
short int or short	2 bytes	-32,768 to 32,767
int	4 bytes	-2,147,483,648 to 2,147,483,647
long int or long	4 bytes	-2,147,483,648 to 2,147,483,647
unsigned	4 bytes	0 to 4,294,967,293
long unsigned	4 bytes	0 to 4,294,967,293
float	4 bytes	
float	4 bytes (allows 7 digit precision)	3.4×10^{-38} to 3.4×10^{38}
double	8 bytes (allows 15 digit precision)	1.7×10^{-308} to 1.7×10^{308}
long double	10 bytes (allows 18 digit precision)	3.4×10^{-4932} to 1.1×10^{4932}

char	1 byte	
char	1 byte	-128 to 127
unsigned char	1 byte	0 to 255
signed char (same as char)	1 byte	-128 to 127
wchar_t	2 bytes	

The following program uses the built-in function **sizeof** to determine the number of bytes in each of the variable types:

```
#include <iostream.h>
#include <conio.h>
void main()
        {
  cout << "size of int in bytes = " << sizeof(int) << endl;
  cout << "size of long in bytes = " << sizeof(long) << endl;
  cout << "size of short in bytes = " << sizeof(short) << endl;
  cout << "size of float in bytes = " << sizeof(float) << endl;
  cout << "size of double in bytes = " << sizeof(double) << endl;
  cout << "size of long double in bytes = " << sizeof(long double) << endl;
  cout << "size of char in bytes = " << sizeof(char) << endl;
  getch();
  }
```

Note that wide char (wchar_t) may not exist on all C++ compilers.

The following is a simple program that reads in a number using the input stream **cin** and prints-out its square-root using the math library function **sqrt**(double x). The function sqrt( ) takes as argument a value of type double, and returns a value of type double.

```
#include <iostream.h>
#include <conio.h>
#include <math.h>

void main()
  {
  float a;
  cout << "Enter a number --> ";
  cin >> a;
  cout << "The square root of " << a << " = " << sqrt(a);
  getch();
  }
```

C++ has a math library containing the standard math functions. These are listed in Table II.

Table II

Math function	library function	Argument(s) type	Return type
$\cos^{-1}x$	acos(x) acosl(x)	double long double	double (radians) long double (radians)
$\sin^{-1}x$	asin(x) asinl(x)	double long double	double (radians) long double (radians)
$\tan^{-1}x$ $\tan^{-1}(x/y)$	atan(x) atanl(x) atan2(x,y) atan2l(x,y)	double long double double long double	double (radians) long double (radians) double (radians) long double (radians)
round to the next higher integer	ceil(x) ceill	double long double	double (not integer) long double

cos(x)	cos(x) cosl(x)	double (radians) long double (radians)	double long double
hyperbolic cosine	cosh(x) coshl(x)	double long double	double long double
e^x	exp(x) expl(x)	double long double	double long double
absolute value	fabs(x) fabsl(x)	double long double	double long double
truncate to nearest lower integer	floor(x) floorl(x)	double long double	double long double
$\log_e(x)$	log(x) logl(x)	double long double	double long double
$\log_{10}(x)$	log10(x) log10l(x)	double long double	double long double
x^y	pow(x,y) powl(x,y)	double long double	double long double
sin(x)	sin(x) sinl(x)	double long double	double long double
hyperbolic sin	sinh(x) sinhl(x)	double long double	double long double
$\sqrt{x}$	sqrt(x) sqrtl(x)	double long double	double long double

1.2.1 The const qualifier

A constant value like π can be declared as a constant to prevent an accidental modification of its value. The following code demonstrates the use of **const**. The program changes an angle specified in degrees to radians.

```
// Degrees to radians
#include <iostream.h>
#include <conio.h>

const double pi=3.141592654;

void main( )
{
double degrees;
cout << "Enter angle in degrees ";
cin >> degrees;
double radians = degrees * pi/double(180);
cout << degrees << " degrees" << " = " << radians
    << " radians" << endl;
getch();
}
```

Note that the variable "*radians*" was declared and assigned a value at the same time in the statement:

double radians = degrees * pi/double(180);

1.2.2 Arithmetic operators

These include addition (+), subtraction (-), multiplication (*), division (/) and modulus (%) operators. Modulus is the remainder of the division of two integer numbers, for example:
9%8 = 1 and 7%11 = 7, etc.

1.2.3 Arithmetic assignment operators

In C++ the following expressions shown on the left-hand side are equivalent to the expressions shown on the right hand side:

a = a + 2; ↔ a += 2;

a = a * 2; ↔ a *= 2;

a = a / 2; ↔ a /= 2;

a = a % 2; ↔ a %= 2;

1.2.4 Increment operator

The following expressions are equivalent:

a = a + 1; ↔ a ++; or ++a;

where the variable "*a*" is an integer (short, long, unsigned or just int). If the ++ precedes the variable then we call this a prefix increment operator and if it follows the variable we call that postfix increment operator. If the variable is incremented by itself in a single statement i.e., a++; or ++a, then there is no difference between postfix and prefix, it basically means add one to the variable. However, if used in a mathematical expression the difference is as follows:

Postfix increment (i++) means utilize the variable in the expression before incrementing it, and use the incremented value in the following statements.

Prefix increment (++i) means increment the variable and use it in the expression and the following expressions.

The following program illustrates this:

```
#include <iostream.h>
#include <conio.h>

void main()
{
int k=2, j=5; // declare and initialize the variables

int i = k * (j++);  // postfix increment
 cout << " result of 2*(5++)" << endl;
 cout << "i = " << i << endl;
 cout << "j = " << j << endl;

 j=5;
 i = k * (++j);   // prefix increment

 cout << "result of 2*(++5)" << endl;
```

```
cout << "i = " << i << endl;
cout << "j = " << j << endl;
getch();
}
```

The print-out of the above program is:

```
result of 2*(5++)
i = 10
j = 6
result of 2*(++5)
i = 12
j = 6
```

Note that you are not allowed to increment a constant such as 5 by typing 5++ or ++5. You can, however, assign 5 to an integer variable and then increment the variable as shown in the above program.

The expression i=k*(j++) multiplies k by j, assigns the result to i and then increments j. The expression i=k*(++j) increments j, multiples k by j and then assigns the result to i.

1.3 Formatted Stream Output

The output can be printed using cout and the insertion operator <<. This form, however, does not provide the programmer with much control over the appearance of the printed result. To format outputs you can use the member functions of the object cout, width() and precision() as follows:

```
#include <iostream.h>
#include <conio.h>

void main()
 {
 int i = 34,j = 65;
 float a=5.6789;
```

```
cout.width(7);
cout << i; cout.width(7);
cout << j; cout.width(10);
cout.precision(3);
cout << a << endl;
getch();
}
```

The output:

```
34    65    5.68
```

The first number is printed right justified in a field width of 7 digits. Similar arrangement for the second number. Third number has a field width of 10. The floating point variable is allowed 3 digits as defined by cout.precision(3).

1.4 Decision Making

1.4.1 if statement

An **if** statement is used for conditional execution of a statement or a block of statements. In constructing an if statement a relational operator is sometimes needed, such as greater than (>) or less than (<) etc. These operators are defined in the following table:

Operator	Meaning
>	greater than
<	less than
= =	equal
! =	not equal
> =	greater than or equal
< =	less than or equal

The following program illustrates usage:

```
#include <iostream.h>
#include <conio.h>
#include <math.h>

void main()
{
double x;
cout << "Enter a number:  ";
cin >> x;
if( x>0)    // x greater than 0
  cout << "square root of " << x
       << " = " << sqrt(x);
getch();
}
```

1.4.2 if-else

The general structure of an if-else appears as follows:

if(*condition*) {
..... }
else{
.....}

The above structure means that if the *condition* is satisfied, i.e. **true**, then the first block statements between the brackets { } are executed, else if the *condition* is **false** the block of statements after the else are executed.

1.4.3 Multiple if-else

One can cascade if-else structures as follows:

```
#include <iostream.h>
#include <stdlib.h>
#include <conio.h>
```

```
void main()
{
float x,y,z;
char ch;
cout << "Enter two numbers: ";
cin >> x >> y;
cout << "Enter operator to be carried \
out on x and y (+, /, -, *) ";
cin >> ch;
if(ch=='+') z=x+y;
else if (ch=='-') z=x-y;
else if (ch=='/') z=x/y;
else if (ch=='*') z=x*y;
else
  {
  cout << "unrecognized operator" << endl;
  getch();
  exit(1);   //defined in stdlib.h
  }
cout << x << ch << y << " = " << z;
getch();
}
```

Note that in C++ single characters are enclosed between single quotes and a string of characters are enclosed between double quotes. Thus the single character + is written as '+' whereas the string "unrecognized operator" is between double quotes. Note also that the statement:

```
cout << "Enter operator to be carried \
out on x and y (+, /, -, *) ";
```

is the same as:

```
cout << "Enter operator to be carried out on x and y (+, /, -, *) ";
```

The backslash "\" allows you to write a string over two lines. You can also rewrite this statement using two separate cout statements as follows:

```
cout << "Enter operator to be carried  ";
cout << "out on x and y (+, /, -, *) ";
```

The exit function used in the above program terminates the program if encountered and is defined in stdlib.h.

1.4.4 switch statement

A multiple if-else can be replaced in most cases with a switch statement. The structure of a switch statement is as follows:

```
switch(variable)
        {
    case value1:
        statements;
        break;
    case value2:
        statements;
        break;
    .
    .
    .
    default:
        statements;
        }
```

where *variable* can assume *value1* or *value2*, etc. If *variable* is not equal to any of the values specified in any of the cases then the statements proceeding the **default** statement are executed. The **break** statement prevents the execution of other case statements and transfers execution to outside the switch block.

The following example illustrates the usage of the switch statement.

```
#include <iostream.h>
```

```
#include <stdlib.h>
#include <conio.h>

void main()
  {
  float x,y,z;
  char ch;
  cout << "Enter two numbers: ";
  cin >> x >> y;
  cout << "Enter operation (+, -, /, *) -->";
  cin >> ch;
  switch(ch)
    {
    case '+':
        z=x+y;
        break;
    case '-':
        z=x-y;
        break;
    case '/':
        z=x/y;
        break;
    case '*':
        z=x*y;
        break;
    default:
        cout << "No such operation" << endl;
        getch();
        exit(1);
     }
```

```
    cout << x << ch << y << " = " << z;
    getch();
}
```

1.4.5 Logical and bit manipulation operators

The following table describes the various logical and bit manipulation operators.

Operator	Description
&&	**AND** operator.
||	**OR** operator.
|	Bitwise **OR**'ing.
&	Bitwise **AND**'ing.
<<	Shift left.
>>	Shift right.
^	Bitwise **exclusive OR**

The bitwise logical operators work on individual bits of variables types int or char as follows:

Bit value		**Result of**		
E1	**E2**	**E1&E2**	**E1^E2**	**E1|E2**
0	0	0	0	0
0	1	0	1	1
1	0	0	1	1
1	1	1	0	1

An **AND** operator results in 1 only if E1 and E2 are both 1's. An **exclusive OR** results in 1 only if E1 and E2 are different. An **OR** results in 0 only if both E1 and E2 are 0's.

The && and || logical operators work as follows:

A	B	A&&B	A||B
false	false	false	false
false	true	false	true
true	false	false	true
true	true	true	true

In C++ false is equivalent to 0 and true is equivalent to any non-zero value (usually assigned the value 1). Logical AND, &&, results in true only if A and B are true and logical OR, ||, results in false only if A and B are false.

The shift operators can be used to shift the bits stored in an integer or character variable by a specified position. For example the left shift operator in the expression:

```
x=x<<n;   //x shifted left n bits
```

shifts the bits in x to the left by n bits and n 0's are fed from the left side of the variable. If n=1 then Figure 2 explains the operation of the left shift.

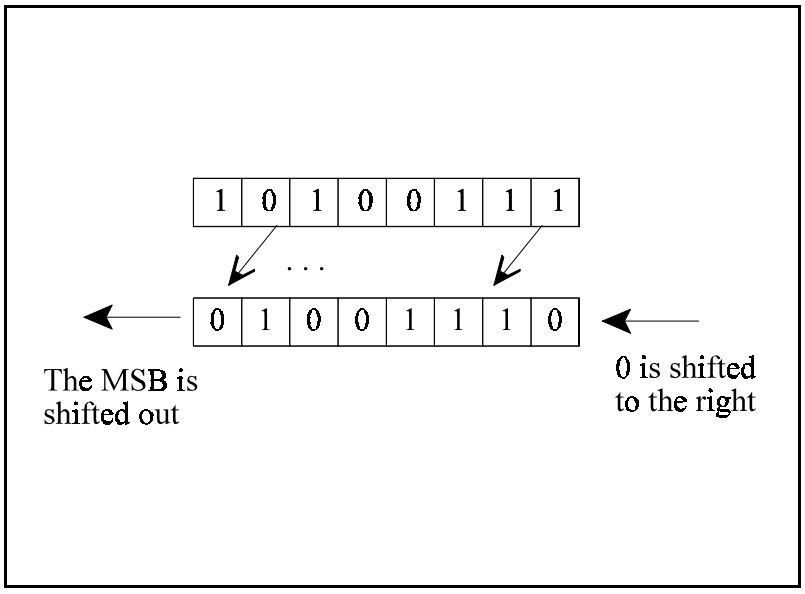


Figure 2 Right shift by one position.

Note that a left shift by n bits is equivalent to multiplying the number by 2^n and a right shift by n bits is equivalent to division by 2^n

1.4.6 The ternary operator ?:

The ternary operator ?: has the syntax:

var1 = (*test*)? *var1*:*var2*;

which means that *var1* = *var2* if *test* = true, otherwise *var* = *var2*.

For example the statement:

y = (x > 0)? x:-x;

assigns to y the absolute value of x.

1.5 Loops

1.5.1 The for loop

The for loop is used for repeating a statement or block of statements several times based on preset conditions. The structure of a for statement is as follows.

for(<*initialization*1>, <initialization2>,..; <*condition for termination*>; <*modification*1>, <*modification2*>, ..);

All statements between < > are considered optional.

Examples of **for** loops:

for(i=0; i<10; i++) { *statements*;}

for (; ;) { *statements*;}

for(i=0, j=5; i<5; i+=2, j++) { *statements*;}

The following program illustrates usage.

```
//Computing factorial
#include <iostream.h>
#include <conio.h>

void main()
  {
 double f=1;
 int n;
```

```
cout << "Enter a number from 1 to 30:  ";
cin >> n;
if(n>0 && n<=30)
 {
 for(int i=1; i<=n; i++)
   f*=double(i);   //i is casted to double

 cout << n << "!=" << f << endl;
 }
else
 cout << "Sorry! number out of range" << endl;

getch();
}
```

The statement

```
f*=double(i);
```

transforms an **int** value into a **double** through casting. Built-in types (int, float, char, etc.) can be casted into one another using the same approach as in the above statement. The above statement can be rewritten in several ways:

```
f*=(double)i;
f=f*double(i);
f=f*(double)i;
```

The **for** loop can also be written in several ways:

```
for(i=1; i<n; )
    f*=double(i++);

int i=1
for( ; i<n; )
   f*=double(i++);
```

Note that variables can be declared anywhere in the program.

1.5.2 The do-while loop

The do-while loop has the following structure

```
do {
   .
   .
   .
   }while(condition);
```

The loop is repeated until *condition* is false.

1.5.3 The while loop

Has the following structure

```
while {
   .
   .
   .
   }
```

1.5.4 The continue and break statement

A **continue** statement forces a jump to the end of a loop if a condition is met. For example:

```
do{
   statement1;
   statement2;
   if(condition1) continue;
   statement3;
   statement4;
    } while(condition2);
```

For the above example *statement3* and *statement4* are skipped if *condition1* is true. The loop, however, continues until *condition2* is false.

A **break** statement, on the other hand, terminates the loop and execution continues with the

first statement after the loop.

1.5.5 Example

Develop a program that sums:

$$sum = 1 + T_1 + T_2 + T_3 + \ldots$$

where $T_1 = x$

$$T_{i+1} = T_i \frac{x}{i+1}$$

and x is any real number.

The following program executes the above summation. Note that since we are summing an infinite series we have to set a condition for termination. The condition used in the program is to terminate summation when $|T|$ is less 10^{-7} which means that sum is calculated to an accuracy of up to 7 significant digits.

```
#include <iostream.h>
#include <math.h>
#include <conio.h>

void main()
  {
 float x, sum, T;
 int i=1;

 cout << "Enter a number --> ";
 cin >> x;

 T=float(1);
 sum=0;

 while (fabs(T) > 1.e-7)
```

```
 {
 sum+=T;
 T*=x/float(i);
 i++;
 }
cout << "sum= " << sum << endl;
}
```

1.6 Arrays

1.6.1 Single dimensional arrays

Single dimensional arrays or vectors are declared as follows:

```
float a[5];      //An array of 5 elements
```

The number between the square bracket is the dimension of the array. Arrays can also be declared and initialized as follows:

```
float a[5]={3,6.7,8.9,6, 7.5};
```

which means that vector **a** contains the numbers between the { } brackets. The dimension can be dropped if the array is initialized. The subscript of an array in C++ always starts from 0. Thus for the above array a[0]=3, a[1]=6.7,...,a[4]=7.5.

```
float a[]={3,6.7,8.9,6, 7.5};
```

The following example program determines the maximum, minimum and sum of an array of numbers.

```
#include <iostream.h>
#include <conio.h>

void main()
  {
 int a[5], i;
 cout << "Enter any 5 integer numbers --> ";
 for(i=0;i<5;i++)
    cin >> a[i];
```

```
for(i=0;i<5;i++)
   cout << a[i]<< " ";
cout << endl;
int max, min;
long sum=0;
max=min=a[0];
for(i=0;i<5;i++)
 {
 if(a[i]>max) max=a[i];
 if(a[i]<min) min=a[i];
 sum+=a[i];
 }
cout << "max= " << max << endl;
cout << "min= " << min << endl;
cout << "sum= " << sum << endl;
getch();
}
```

Note that in C++ we can assign several variables to the same constant using a statement as follows:

var1=var2=var3=value;

value is first assigned to var3 and then var3 is first assigned to var2 and so on. The input numbers to the above program can be entered by typing each number followed by a space (except the last number) or by entering each number on a separate line.

The size of an array can be determined during run time by using **sizeof** as follows:

```
int a[]={4,6,7,8,9};
int size=sizeof(a)/sizeof(int);
```

1.6.2 Matrices

A matrix is a two dimensional array of numbers arranged as follows:

$$A = \begin{bmatrix} a_{00} & a_{01} & a_{02} \\ a_{10} & a_{11} & a_{12} \\ a_{20} & a_{21} & a_{22} \end{bmatrix}$$

The above is a 3 rows by 3 columns matrix and the subscripts of each element indicate the row and column positions. Element a_{ij} is located in the $(i+1)^{th}$ row and $(j+1)^{th}$ column. In C++ a matrix is declared as follows:

float a[3][4];

which means that matrix **a** is of type float and can store 3 rows by 4 columns.

The following program allows the user to enter a 3 by 4 matrix and then prints the result in matrix form.

```
#include <iostream.h>
#include <conio.h>

void main()
  {
 float a[3][4];
 int i,j;
 cout << "Enter elements of a 3x4 matrix\
 one row at a time: " << endl;
 for(i=0; i<3; i++)
  for(j=0; j<4; j++)
    cin >> a[i][j];

//printing result
   cout << endl;
for(i=0; i<3; i++)
 {
 for(j=0; j<4; j++)
```

```
    cout << a[i][j] << " ";
  cout << endl;
  }
  getch();
  }
```

1.7 Functions

A function is a block of code that can be called from any location in a program. A function is declared as follows:

return_type function_name(<*arguments*>)

```
{
 code, comments.
}
```

The following program calculates the area of a circle. The code for the area of the circle is included in a function *Acircle*(**float** radius) and the main program calls the function and prints out the result.

```
#include <iostream.h>
#include <math.h>
#include <conio.h>

const float pi=4.0*atan(1);

float Acircle(float radius)
  {
  float Area;
  Area = pi*radius*radius;
  return  Area;
  }

void main()
```

```
{
float r;

do
  {
cout << "Enter radius of circle --> ";
cin >> r;
cout << Acircle(r) << endl;
   }while(r!=0);
 getch();
}
```

The **return** statement returns control and passes back the value of *Area* to the calling program. We will have more to say about functions in later chapters.

1.8 Scope and Storage Class of variables

In the previous program pi is a global variable accessible to all functions (including main) following its declaration. The scope for such a variable is global. Variables declared within a function or a block of statements (statements contained within the brackets { }) are local variables and are accessible only by the function or within the block in which they are defined.

Four storage classes are defined within C++: **auto**, **extern**, **register** and **static**.

auto is used only with local scope variable declarations. It conveys local (automatic) duration. Local variables are **auto** by default and therefore the usage of this specifier is rare.

extern can be used with functions as well as variables. It indicates external linkage.

register specifier is allowed only for local variables. It is equivalent to auto, but it makes a request to the compiler to allocate the variable to a register if possible. A register is an internal storage built in the microprocessor, and hence accessing a register should be faster than accessing an external memory location.

static specifier can be used with variables and functions to indicate internal linkage. It also indicates that the variable has static duration. Static variables are initialized to 0 or null by the compiler in the

absence of any initialization in the program.

The following program shows usage of some of the above specifiers.

```
#include <iostream.h>
#include <math.h>
#include <conio.h>

const float pi=4.0*atan(1);     //global variable

void main()
  {
 int i=1;   //same as auto int

 for(int j=0;j<10;j++)
  {
  static int r=2;  //local static variable
  cout << i << ". Area of circle of radius " << r
   << " = " << pi*r*r << endl;
  r++;
  i++; // i is accessible within the block
  }
//  cout << r ;   //r is inaccessible
 getch();
 }
```

If the specifier **static** is removed from the declaration of **r** and initialized in the block of statements bracketed after the **for** loop then r will always remain at 2 for all the iterations of the loop. By declaring **r** as static its updated value is retained for the next iteration and so on.

1.8.1 Scope resolution operator (::)

If the same name is used as a global variable and a local variable, then the compiler will use the local value. If, however, you need to refer to the global variable then precede the variable with the scope resolution operator :: as follows:

```
int c=4;
void main( )
{
 int c=5;
 cout << c << endl;      // reference to local variable
 cout << ::c << endl;    // reference to global variable
  }
```

Problems

1. (a) Convert the octal number $(327)_8$ to decimal representation.

   (b) Write down the result of

   ```
   cout << 0112;
   cout << 0x4EF;
   cout << oct << 0112;
   ```

2. Develop a program that prompts the user to enter an angle in degrees and then prints out the sine and cosine of the entered value. Note that the library functions sin(x) and cos(x) expect x to be in radians.

3. Complete the following program:

   ```
   void main()
     {
   float b[] = { -3, 5, 7, -9, 8, -2};
   int size = .....;      //calculate number of elements in b[]
   float sum=0.0;
   ```

```
for( int i = 0; i < size; i++)
  {
  ..................        // sum up values of b
  }
// Calculate average
  .................
// Print result
```

4. Develop a program to test the mathematical ability of the user. The program is to generate any two numbers between 1 and 20 using a random number generator function (check your compiler's library function list - Microsoft Visual C++ has srand() and rand() and Borland C++ or C++Builder has randomize() and random() functions for that purpose. The program is then to prompt the user if he/she wishes to be tested on addition, subtraction, multiplication or division. The program then prompts the user to enter the arithmetic result of the two numbers generated. If the answer is correct the program prints: OK, otherwise the program prints: wrong answer and prints the correct answer. The program then prompts the user if he/she wishes to continue, if the user enters Y or y then the user is tested on a new set of randomly generated numbers otherwise it exists.

5. Develop a program for adding two user supplied matrices. For matrix addition the two matrices should have equal number of columns and rows. Addition of two matrices A and B is carried out as follows:

$$c_{ij} = a_{ij} + b_{ij}$$

6. Develop a program for obtaining the transpose of a user supplied matrix. Transpose of a matrix A is obtained by converting the rows into columns as follows:

$$b_{ji} = a_{ij}$$

7. Write down the output of the following code segment:

```
double x=4.0*atan(1.0);
double y=(x>0.0)?sqrt(x):-999.99;
cout << x << “ ” << y;
```