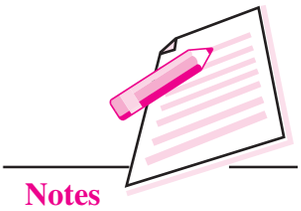


Notes

15

FUNCTIONS

In the previous lesson you have learnt about control statements of C++ programming language. In this lesson you will learn about functions in C++. A function is a group of statements that together perform a task. Every program in C++ has main() function by default. C++ standard library provides lot of built-in functions that you can call or use in your program.

OBJECTIVES

After reading this lesson, you will be able to:

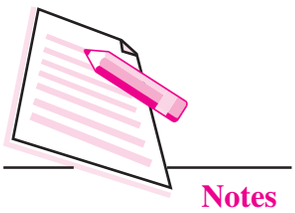
- define function;
- use # include directive;
- list library functions;
- call functions;
- define inline function.

15.1 FUNCTION

Functions are blocks of code that perform a task to accomplish something productive. When you want to group same statements depending upon their actions like finding total, average, etc., you can put them in a function. Whenever you need these set of statements you can call this function in your program. A function may be user defined function or library function. C++ provides a lot of built in functions which you will learn in the next section. To use library functions in a C++ program you have to include header files.

A header file may be included in one of the two ways. # include <iostream.h> or # include "iostream.h". The header file in angle brackets <and> means that file reside in standard include directory. The header file in double quotes means that file resides in current directory. The # include directive instructs the compiler

to read and include another file in the current file. The compiler compiles the entire code.



Notes

15.2 LIBRARY FUNCTIONS

You know that C ++ provides many built in functions that save programming time and effort required.

Mathematical Functions

Mathematical functions like sin ( ), cos ( ), etc., are defined in header file math.h. Some of the important mathematical functions are given in the following table.

Some standard mathematical functions

Function	Meaning
sin(x)	Sin of an angle x (measured in radians)
cos(x)	cosine of an angle x (measured in radians)
tan(x)	Tangent of an angle x (measured in radians)
asin(x)	Sin^{-1} where x (measured in radians)
acos(x)	Cos^{-1} where x (measured in radians)
exp(x)	Exponential function of x (e^x)
log(x)	Logarithm of x
log10(x)	Logarithm of number x to the base 10
sqrt(x)	Square root of x
pow(x,y)	x raised to the power y
abx(x)	Absolute value of integer number x
fabs(x)	absolute value of real number x

Character Functions

All the character functions require ctype.h header file. The following table lists the function.

String	Functions
isalpha(c)	It returns True if c is a letter (A to Z or a to z) otherwise False.
isdigit(c)	It returns True if c is a digit (0 through 9) otherwise False.
isalnum(c)	It returns True if c is a digit from 0 through 9 or an alphabetic character (either uppercase of lowercase) otherwise False.
islower(c)	It returns True if c as a lowercase letter otherwise False.

isupper(c)	It returns True if c is an uppercase letter otherwise False.
toupper(c)	It converts c to uppercase letter.
tolower(c)	It converts c to lowercase letter.

String Functions

The string functions are present in the string.h header file. Some string functions are given below:

strlen(S)	It gives the no. of characters including spaces present in a string S.
strcat(S1, S2)	It concatenates the string S2 onto the end of the string S1. The string S1 must have enough storage location to hold S2.
strcpy(S1, S2)	It copies character string S2 to string S1. The string S1 must have enough storage locations to hold S2.
strcmp((S1, S2) = = 0) strcmp((S1, S2)>0) strcmp((S1, S2)<0)	It compares S1 and S2 and finds out whether S1 equals to S2, S1 greater than S2 or S1 less than S2.

Console I/O functions

The following are the list of functions

(i) getchar( )

(ii) putchar( )

(iii) gets( )

(iv) puts( )

The header file for above functions is stdio.h. The first two functions deal with single character and last two functions deal with string (group of characters).

getchar( ) function

The getchar( ) function returns a single character from a standard input device (keyboard). It takes no parameter and the returned value is the input character. The general form of the getchar( ) function is:

A = getchar( );

The variable A is of the type character. It inputs a character from the keyboard and assigns it to variable A.

putchar( ) function

The putchar( ) function takes one argument, which is the character to be sent to output device. It also returns this character as a result. The general form of the putchar( ) function is :

putchar(ch);

where ch is a variable of type character.

Example 1

```
# include <iostream.h >

# include <stdio.h>

void main ( )

{

        char ch;

        ch = getchar ( ) ;

        putchar (ch);

}
```

The output of the above program is

The above program takes a character from the keyboard and prints it on the screen.

gets( ) function

The gets( ) function gets a string terminated by a newline character from the standard input stream stdin. The gets ( ) replaces the newline by a null character (\0). It also allows input string to contain white space characters (spaces, tabs).

This statement gets a string and stores it in the variable A.

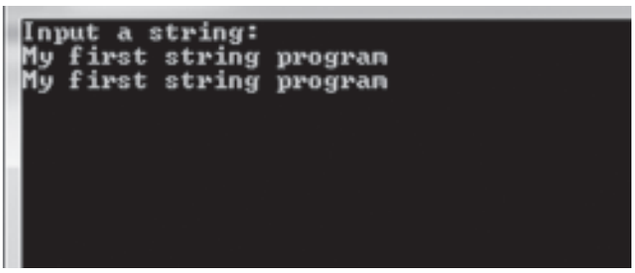
Example:

gets (A) ;

Example 2

```
# include < stdio.h >
# include < iostream.h >
void main ( )
{
        char A [ 100 ];
        cout << "Input a string: \n";
        gets (A);
        puts (A);
}
```

The output of the above program is



Some More Functions

The getch( ) and getche( ) functions

The general form of the getch( ) and getche( ) is

ch = getche( ) ;

ch1 = getch( ) ;

ch and ch1 are the variables of type character. They take no argument and require the conio.h header file. On execution, the cursor blinks, the user must type a character. The value of the character returned from getche ( ) is assigned to ch. The getche ( ) function echoes the character to the screen. That's why there's an e in getche. Another function, getch ( ), is similar to getche ( ) but does not echo the character to the screen.

Example 3

To count the number of characters and words in a sentence entered by the user.

```
# include < iostream.h >
# include < conio.h >
void main ( )
{
        char ch;
        int chcnt = 0, wdcnt = 1;
        while ( (ch = getche ( ) ) ! = '\r')
        {
                if (ch = = ' ')
                        wdcnt + + ;
                else
                        chcnt + + ;
        }
cout < < "No. of characters" < < chcnt << "\n"
cout < < "No. of words" < < wdcnt;
}
```

The above program counts the no. of characters and no. of words in a sentence terminated by return key. The '\r' reads the return key character.

INTEXT QUESTIONS 15.1

1. Name the header file that is required for the following :

   (i) abs ( )

   (ii) strcmp ( )

   (iii) clrscr ( )

   (iv) tolower ( )

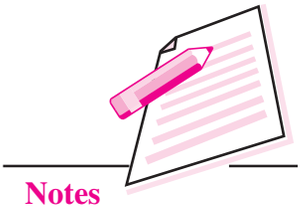
2. Fill in the blanks :

   a) All header files have an extension .....

   b) Character functions are available in ..... header file.

   c) The function toupper (C) converts C to .....

   d) In strcpy (S1, S2) the string..... copies onto the string.....



Notes

e) The function which counts the length of the string is .....

f) The sin ( ) function is available in ..... header file.

3. State whether the following statements are true or false:

a) The isdigit ( ) function checks whether the entered character is digit or not.

b) All string functions are available in stdio.h.

c) The iostream.h contains all input/output functions.

d) The tolower (C) converts C to lowercase.

e) strcat (S1, S2) splits two strings into three strings.

f) pow (x, y) claculates y to the power x.

g) cos (x) returns the cosine of angle (x).

h) tolower ( ) function is available in ctype.h.

i) The log (x) is available in string.h.

15.3 USER DEFINED C++ FUNCTION

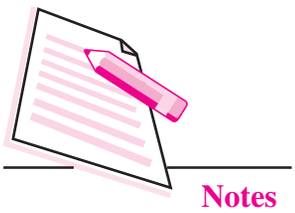
A C++ function is a grouping of program statements in a single unit. Do you know that the main ( ) is an essential function in all C ++ programs? If there is a single function in a program, it should be named main ( ). The main ( ) function is the starting point for the execution of a program. The definition of main ( ) would look like as follows :

```
main ( )
{
// main program statements
}
```

In C ++, the main ( ) returns a value of type int to the operating system. The function that have a return value should use the return statement for termination. The main ( ) function, therefore, is defined as follows :

```
void main ( )
{
        cout<<"Welcome";
}
```

Here, the return type of function is void.



Notes

Consider the program given below :

```
# include < iostream. h >
void main ( )
{
cout < < "Computer Science \n";
}
```

It prints computer science on the screen. We can also write the above program in the following way:

```
# include < iostream.h>
void message ( );
void main ( )
{
        message ( ) ;
}
void message ( )
{
        cout << "Computer Science\n";
}
```

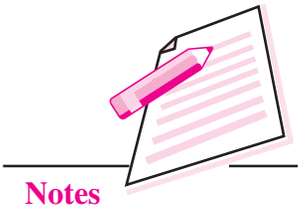
The above program also prints computer science but it uses a function message( ) for printing. In addition to the functions main ( ) and message ( ), the program has an extra line after # include preprocessor directive. The line is void message ( ). This is known as function prototype.

15.3.1 Function Prototype

The function prototype informs the compiler about the functions to be used in a program, the argument they take and the type of value they return. Functions which do not return any value are known as void functions.

15.3.2 Arguments to a function

Sometimes the calling function supplies some values to the called function. These are known as parameters. The variables which supply the values to a calling function are called actual parameters. The variable which receive the value from called statement are termed formal parameters.



Notes

Consider the following example that evaluates the area of a circle.

```
# include < iostream.h >
void area (float);
void main ( )
{
float radius;
cin >> radius;
area (radius ) ;
}
void area (float r)
{
cout << "The area of the circle is " << 3.1416 *r* r << "\n";
}
```

The statement area (radius), that invokes the function, may be written as

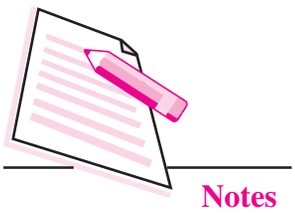
area (6.0);

This will evaluate and display the area of a circle with radius 6.0. Here radius is called actual parameter and r is called formal parameter.

15.3.3 Return Type of a function

In the above program, area of a circle is displayed in the function area ( ) itself. If the area of a circle is required in the calling function main ( ), the function area ( ) may return the area evaluated to function main ( ). This is possible through the return statement. The program may be written as :

```
# include < iostream. h >
float area (float);
void main ( )
{
        float radius, y;
        cin >> radius ;
        y = area (radius) ;
        cout < < "The area of the circle is " < < y;
}
float area (float r)
{
        return ( 3.1416 *r* r) ;
}
```



Notes

In function prototype, void is replaced by **float**, this indicates that value returned by the function area to main function is float. In the calling function variable y is assigned the value returned by function area ( ). Thus, variable y will be assigned the area of a circle.

15.3.4 Global and Local variables

The variable declared in a program may broadly be classified into following two types;

(i) Local variable

(ii) Global variable

Local variable

A variable declared within the body of a function will be evaluated only within the function. The portion of the program in which a variable is retained in memory is known as the scope of the variable. The scope of the local variable is a function where it is defined. A variable may be local to function or compound statement.

Global variable

A variable that is declared outside any function is known as a global variable. The scope of such a variable extends till the end of the program. These variables are available to all functions which follow their declaration. So it should be defined at the beginning, before any function is defined. If in a program, variable a is declared as local as well as global. Then *a* represents the local variable.

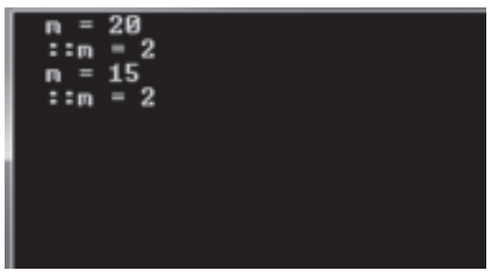
: : *a* represents the global variable. The symbol : : is called scope resolution operator.

Example 4

```
# include < iostream.h >
int m = 2;
void main ( )
{
        int m = 15;
        {
        int m = 10 * : : m;
```

```
        cout < < "m = " < < m < < "\n";
        cout < < ": : m = " < < : : m < < "\n"
        }
        cout < < "m = " < < m < < "\n";
        cout < < ": : m = " < < : :m < < "\n";
}
```

The output of the above program is



Variables and Storage class

The storage class of a variable determines which parts of a program can access it and how long it stays in existence. The storage class can be classified as (i) Automatic (ii) Register (iii) Static (iv) External.

Automatic variable

All variables by default are auto, i.e., the declarations, int a and auto int a are equivalent. Auto variables retain their scope till the end of the function in which they are defined. An automatic variable is not created until the function in which it is defined is called. When the function exits and control is returned to the calling program, the variables are destroyed and their values are lost. The name automatic is used because the variables are automatically created when a function is called and automatically destroyed when it returns.

Example 5

```
# include <iostream.h >
void sum (int);
void main( )
{
        int i;
        for (i = 1; i < =5; i + +)
        sum (i);
}
```

```
void sum (int n)
{
        auto int s = 0;
        s = s + n;
        cout << s << "\n";
}
```

The output of the program is

Register variable

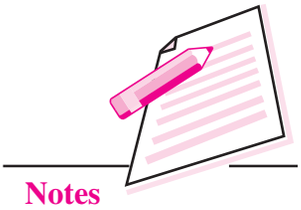
A register declaration is an auto declaration. A register variable has all the characteristics of an auto variable. The difference is that register variable provides fast access as they are stored inside CPU registers rather than in memory. The register and auto can be applied to local variable.

Static variable

The variable may be static automatic variable or static external variable. The later one is meaningful only in multifile programs. A static automatic variable has the visibility of a local variable but the lifetime of an external variable. Thus it is visible only inside the function in which it is defined, but it remains in existence for the life of the program. If in the previous program auto int s = 0 is changed to static int s = 0. The output of the program will be

External variable

A large program may be written by a number of persons in different files. A variable declared global in one file will not be available to a function in another



file. Such a variable, if required by functions in both the files, should be declared global in one file and at the same time declared external in the second file.

Example : extern int a;

where extern is the keyword for external.

15.3.5 Calling of Function

The function can be called using either of the following methods:

(i) call-by-value

(ii) call-by-reference

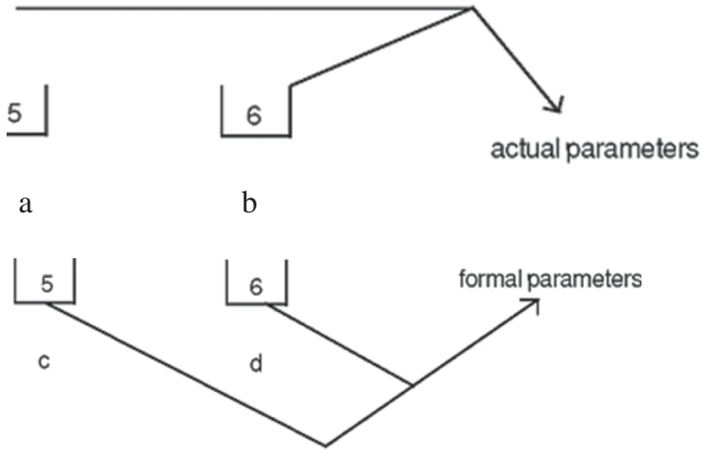
Call-by-value

Consider the following program which will swap the value of two variables using the method call-by-value.

Example 6

```
# include < iostream.h>
void swap (int, int);
void main ( )
{
        int a, b;
        cin >> a >> b;
        swap (a, b);
        cout << a << b << "\n";
}
void swap (int c, int d)
{
        int t;
        t = c;
        c = d;
        d = t;
        cout << c << d << "\n";
}
```

The variables a and b are called actual parameters. When calling the function swap ( ) the value of a is passed onto c and value of b is passed onto d. Here c and d are called formal parameters.



In the function swap ( ), the value are swapped but this change will not be implemented to actual parameters.

The variables which supply the values to a calling function are called actual parameters. The variables which receive the value from called function are called formal parameters.

Call-by-reference

Consider the following program which will swap the value of two variables using call-by-reference method.

Example 7

```
# include < iostream.h >
void swap (int &, int &);
void main ( )
{
        int a , b;
        cin >> a >> b;
        swap (a, b);
        cout << a << b << "\n";
}
```

```
void swap (int &c, int &d)
{
        int t;
        t = c;
        c = d;
        d = t;
        cout << c << d << "\n";
}
```

In this method, when the function is called, the address is transferred from actual to formal parameters. It means variables a and c are sharing the same memory address and variables b and d are sharing the same memory address.

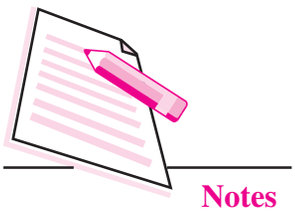
a/c b/d

So, any change in formal parameter will be implemented to actual parameter.

15.4 INLINE FUNCTION

Functions save memory space because all the calls to the function cause the same code to be executed. The function's body need not be duplicated in memory. When the compiler sees a function call, it normally jumps to the function. At the end of the function, it jumps back to the statement following the call.

While the sequence of events may save memory space, it takes some extra time. To save execution time in short functions, inline function is used. Each time there is a function call, the actual code from the function is inserted instead of a jump to the function. The inline function is used only for shorter code.



Notes

Example 8

```
# include < iostream.h >
inline int cube (int r)
{
        return (r*r*r*);
}
void main ( )
{
        int number, y;
        cout << "Enter number";
        cin >> number;
        y = cube (number);
        cout << "The cube of the number is" << y;
}
```

Some important points to be noted:

(i) Function is made inline by putting a word inline in the beginning.

(ii) Inline function should be declared before main ( ) function.

(iii) It does not have function prototype.

(iv) Only shorter code is used in inline function.

(v) If longer code is made inline then compiler ignores the request and it will be executed as normal function.

15.5 FUNCTION WITH DEFAULT ARGUMENTS

C ++ allows to call a function without specifying all its arguments. In such cases, the function assigns a default value to a parameter which does not have a matching argument in the function call. Default value are specified when the function is declared. The compiler knows from the prototype how many arguments a function uses for calling.

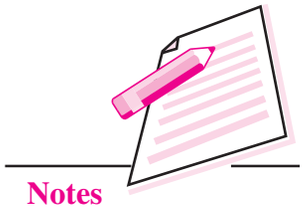
Example 9

```
float result (int marks 1, int marks 2, int marks 3 = 75);
```

A subsequent function call

```
Average = result (60, 70);
```

passes the value 60 to marks 1, 70 to marks 2 and lets the function use default value of 75 for marks 3.



Notes

The call Average = result (60, 70, 80);

passes the value 80 to marks3.

The important point to note is that only the trailing arguments can have default values.

Consider the following program:

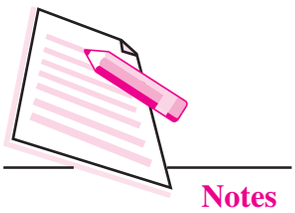
```
# include < iostream.h>
void repchar (char = '=', int = 30);
void main ( )
{
        repchar ( );
        repchar ( '*');
        repchar ('+', 45);
}
void repchar (char ch, int x)
{
        for (int i= 1; i < = x; i + + )
        cout << ch;
        cout << "\n";
}
```

The default argument follows an equal sign, which is placed directly after the type name. In the above program, the function repchar ( ) takes two arguments. It is called three times from main ( ).

- First time it is called with no argument. The function assigns the default value '=' to ch and 30 to x and it print '=' symbol thirty times on the screen.
- Second time it is called with one argument. The function assigns the default value 30 to x and it prints '*' symbol thirty times on the screen.
- Third time it is called with two argument. The actual parameters take priority over the default parameters. It prints '+' symbol forty-five times.

Function Overloading

The same function name can be used to create functions that perform a variety of different tasks. This is known as **function polymorphism** in OOP. The function would perform different operations depending on the argument list in function call. The correct function to be invoked is determined by checking the number and type of arguments. For example, an overloaded function mpy( ) handles different types of data as follows:



Notes


```
/ / Declaration
    void mpy (int, int);              / / prototype 1
    void mpy (int, float);            / / prototype 2
    void mpy (float, int);            / /  prototype 3
    void mpy (float, float);          / / prototype 4
/ / Function calls
    cout < < mpy (3.5, 1.2);          / / uses prototype 4
    cout < < mpy (2, 5);              / / uses prototype 1
    cout < < mpy (5.2, 6);            / / uses prototype 3
    cout < < mpy (7, 6.2);            / / uses prototype 2
```

A function call first matches the prototype having the same number and type of arguments and then calls the appropriate function for execution. A best match must be unique.

The function selection involves the following steps:

(i) The compiler tries to find an exact match in which types of actual arguments are the same and uses that function.

(ii) If an exact match is not found, the compiler uses the integral promotions to the actual arguments, such as

char to int

float to double

to find a match.

(iii) When either of them fails, the compiler tries to use the built in conversions (the implicit conversions) to the actual arguments then uses the function whose match is unique. If the conversion is possible to have multi matches, then the compiler will generate an error message. Suppose, we use the following two functions:-

```
long abc (long n);
double abc (double x);
```

A function call such as

```
abc (10)
```

will cause an error because int argument can be converted to either long or double, thereby creating an ambiguous situation.

(iv) If all the steps fail, then the compiler will try the user defined conversions in combination with integral promotions and built in conversions to find a unique match.

INTEXT QUESTIONS 15.2

1. What will be the output of the following programs ?

(a)
```
# include < iostream.h>
void add (float);
void main ( )
{
    float x = 5.0;
    add (x);
    cout << x ;
}
void add (float x)
{
     x = x + 2;
}
```

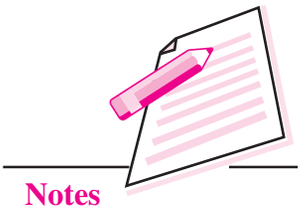
(b)
```
# include < iostream.h >
void add (float);
float x = 5.0;
void main ( )
{
    add (x);
    cout << x ;
}
void add (float x)
{
     x = x + 2;
}
```

(c)
```
# include < iostream.h >
float add (float);
void main ( )
{
    float x = 5.0;
    float y;
```

```
        y = add (x);
        cout << x << " " << y < < "\n"
    }
    float add (float x)
    {
        return (x +2);
    }
```

(d)
```
    # include < iostream.h >
    void P1 (int &, int);
    void main ( )
    {
        int a = 5;
                int b = 6;
                cout << "output of the program \n";
                cout << "values of a and b are " << a << b << "\n";
                P1 (a, b);
                cout << "values of a and b are" << a << b;
        }
    void P1 (int & a, int b)
    {
        a = 7 ;
        b = 10 ;
    }
```

(e)
```
    include < iostream.h >
    int a, b;
    void try2 (int x, int & y);
    void main ( )
    {
        int a = 10 ;
        int b = 7 ;
        try2 (a, b);
        cout << a << b << "\n";
    }
```



Notes

```
void try2 ( int x, int & y)
{
    b = x;
    y = y + b;
    x = y;
    cout << x << y << b << "\n";
}
```

2. Fill in the blanks:

   (a) Any C++ program contains at least ..... function(s).

   (b) After each function has done a job, control returns to .....

   (c) The declaration of a function in the calling program is known as a function .....

   (d) In a program, the function can be called in two ways ..... and .....

   (e) When the return type is not specified, the default return type of function is .....

   (f) To return the control back to the calling function, we use the keyboard .....

   (g) In an ..... function, looping is not allowed.

3. State whether the following statements are true or false:

   (a) There is a limit on the number of functions that might be present in a C++ program.

   (b) A variable which is declared inside the function is called global variable.

   (c) When function is not returning anything, return type is void.

   (d) An inline function is preferred when its size is small.

   (e) In call by reference, any change made in the object inside the function will reflect in the value of the actual object.

   (f) A C ++ program can have more than one function with the same name.

   (g) Inline is a keyword in C++ programming language.

   (h) An inline function must be defined before it is called.

   (i) Any function in C++ can be declared as inline function.

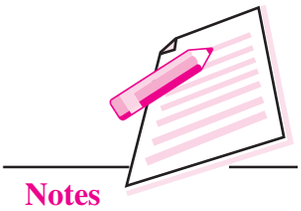
   (j) It is necessary to specify the variable name in the function prototype.

WHAT YOU HAVE LEARNT

- Mathematical functions are defined in math.h file.
- Character functions require ctype.h file.
- String functions are present in the string.h file.
- Functions which do not return any value are known as void functions.
- The variables which supply the values to a calling function are called as actual parameters.
- Variables declared in a program are classified as local variable and global variable.
- Functions can be called using call by value or call by reference methods.
- Inline functions are used in the programs, to save the execution time.

TERMINAL EXERCISE

1. Write a program that will ask the user to enter a single character. Find out whether it is alphabetic, digit or special character.
2. Write a program that will ask the user to enter his/her name. Convert all uppercase letter to lowercase letter and vice-versa.
3. Write a program that will ask the user to enter a character. Check if it is alphabetic or not. If alphabetic, check whether it is in uppercase or lowercase.
4. What is the difference between the following ?
   (i) getchar ( ) & putchar ( )
   (ii) gets ( ) & puts ( )
5. Write a program that will count the number of characters in a sentence entered by user.
6. Write a program that will count the number of words in a sentence entered by the user.
7. Write a program that will count the number of A's, E's, I's, O's and U's in a sentence.
8. Differentiate between the following :
   (i) Actual and Formal parameters
   (ii) Call by value and call by reference
9. What do you mean by inline function ?
10. What are the advantages of function prototype in C++?



11. What is the main advantage of passing the arguments by reference ?
12. When do we need to use default arguments in a function?
13. What do you mean by overloading of a function?
14. Write a program in C++ to call a function which will return the cube of a given number using inline function.
15. Write a program in C++ that prints the largest of two numbers entered from the keyboard. Pass the two numbers to a function as argument, find the largest and return this value to a main function.

ANSWERS TO INTEXT QUESTIONS

15.1

1. (i) math.h (ii) string.h (iii) conio.h (iv) ctype.h
2. (a) h (b) ctype.h (c) uppercase
   (d) S2, S1 (e) strlen ( ) (f) math.h
3. (a) True (b) False (c) True
   (d) True (e) False (f) False
   (g) True (h) True (i) False

15.2

1. (a) 5 (b) 5 (c) 5 7
   (d) output of the program values of a and b are 5 6 values of a and b are 7 6
   (e) 17 17 10 10 17
2. (a) 1
   (b) statement after calling function statement
   (c) prototype
   (d) call by value call by reference
   (e) int
   (f) return
   (g) inline
3. (a) False (b) False (c) True (d) True (e) True
   (f) True (g) True (h) True (i) False (j) False