ctions: Functions in programming languages help in organising and managing codes in modular . Following are the main advantages of using functions in a program.

(i) Functions help in breaking down the code into smaller units for easy debugging and managing.

(ii) Functions help in re-using the same code without rewriting it.

A C++ function definition consists of a **function header** and a **function body**. All the parts of a function are as follows:

- **Return Type:** A function may return a value. The **return_type** is the datatype of the value the function returns. Some functions perform the desired operations without returning a value. In such case, the return_type of the function is mentioned by **void**.
- **Function Name:** This is the name of the function. The function name and the parameter list together constitute the function signature.
- **Parameters:** A parameter(s) is/are like a placeholder. When a function is invoked, you pass a value(s) to the parameter(s). The parameter list refers to the type, order, and number of the parameters of a function. Parameters are optional; that is, a function may contain no parameters.
- **Function Body:** The function body contains a collection of statements that define what the function does.

In C++, a few generic functions are already defined in pre-existing header files, which are bundled alongwith the C++ compiler, these functions are known as built-in (predefined) functions.

Header File: `math.h`

Built-in Function	Description	Example	Output

`abs()`	To return absolute value of an integer value	`int N=90,M=-567,L;`
		`L=abs(N);`
		`cout<<L<<endl;`
		`cout<<abs(M)<<endl;`
			90
			567

`fabs()`	To return absolute value of a float value	`float X=-9.45,Y=56.7,Z;`
		`Z=fabs(X);`
		`cout<<Z<<endl;`
		`cout<<fabs(Y)<<endl;`
			9.45
			56.7

`sin()`	To return trigonometric sine of angle X in radians	`const float PIE=3.1416;`
		`float X;`
		`X=sin(PIE/2);`
		`cout<<X<<endl;`
		`cout<<sin(PIE/6)<<endl;`
			1
			0.500001

`cos()`	To return trigonometric cosine of angle X in radians	`const float PIE=3.1416;`
		`float X;`
		`X=cos(PIE/3);`
		`cout<<X<<endl;`
		`cout<<cos(PIE/6)<<endl;`
			0.499998
			0.866025

sqrt()	To return square root of x	float X=25,Y=625,Z;
		Z=sqrt(X);
		cout<<Z<<endl;
		cout<<sqrt(Y)<<endl;
			5
			25

pow()	To return x raised to power y	float X=5,Y=3,Z;
		Z=pow(X,Y);
		cout<<Z<<endl;
		cout<<pow(4,2)<<endl;
			125
			16

log()	To return natural logarithm of x (base e)	float X=5.5,Y=3.3,Z;
		Z=log(X);
		cout<<Z<<endl;
		cout<<log(Y)<<endl;
			1.704748
			1.193922

exp()	To return exponential of x	float X=1.704748,Z; Z=exp(X); cout<<Z<<endl;	5.5

Header File: ctype.h

Built-in Function	Description	Example	Output
toupper()	To return ascii value of uppercase letter for lowercase letter, else ascii value of the character itself.	char P='A',Q='b',R='*',S,T; cout<<toupper(Q)<<":" <<(char)toupper(Q)<<endl; S=toupper(P); T=toupper(R); cout<<S<<":"<<T<<endl;	66:B A:*
tolower()	To return ascii value of lowercase letter for uppercase letter, else ascii value of the character itself.	char P='A',Q='b',R='#',S,T; cout<<tolower(Q)<<":" <<(char)tolower(Q)<<endl; S=tolower(P); T=tolower(R); cout<<S<<":"<<T<<endl;	98:b a:#

isalpha()	To return true (non-zero) value, if the character passed is an alphabet else return false (i.e.0) value.	char P='A',Q='b',R='#';
		if (isalpha(P))
		cout<<P<<" Alphabet\n";
		if (!isalpha(R))
		cout<<R<<" NotAlphabet\n";
		if (isalpha(Q))
		cout<<Q<<" Alphabet\n";
			A Alphabet
			# NotAlphabet
			b Alphabet

isdigit()	To return true (non-zero) value, if the character passed is a digit else return false (i.e.0) value.	char P='8',Q='b';
		if (isdigit(P))
		cout<<P<<" Digit\n";
		if (isdigit(Q))
		cout<<Q<<" Digit\n";
		else
		cout<<Q<<" Not Digit\n";
			8 Digit
			b NotDigit

isalnum()	To return true (non-zero) value, if	char P='8',R='#'; if (isalnum(P))	Alpha/Digit NoAlpha/Digit

	the character passed is a alphabet/digit else return false (i.e.0) value.	cout<<" Alpha/Digit\n";
		if (!isalnum(R))
		cout<<"NoAlpha/Digit\n";
		else
		cout<<"Alpha/Digit\n";	

isupper()	To return true (non-zero) value, if the character passed is an uppercase alphabet else return false (i.e.0) value.	char P='A',R='d';
		if (isupper(P))
		cout<<"UpperCase \n";
		if (!isupper(R))
		cout<<"Not UpperCase\n";
		else
		cout<<"UpperCase\n";
			UpperCase
			Not UpperCase

islower()	To return true (non-zero) value, if the character passed is an lowercase alphabet else return false (i.e.0) value.	char P='b',R='A';
		if (islower(P))
		cout<<"LowerCase \n";
		if (!islower(R))
		cout<<"Not LowerCase\n";
		else
		cout<<"LowerCase\n";
			LowerCase
			Not LowerCase

Header File: stdlib.h

Built-in Function	Description	Example	Output

random()	To return a randomly generated value between 0 and N-1. Where N is an integer passed to the function.	int A,B,C;
		A=random(10);//0..9
		B=random(5); //0..4
		cout<<A<<"x"<<B<<"?";
		cin>>C;
		if (C==A*B)
		cout<<"Correct!";
		else
		cout<<"Incorrect!";
			Ist Time Run
			5X4?21
			Incorrect
			
			IInd Time Run
			5X4?20
			Correct

randomize()	randomize() function initializes the random number generator with a random value, so that every time random() function generates different set of random values.	int A,B,C;
		randomize();
		A=random(10);//0..9
		B=random(5); //0..4
		cout<<A<<"x"<<B<<"?";
		cin>>C;
		if (C==A*B)
		cout<<"Correct!";
		else
		cout<<"Incorrect!";
			Ist Time Run
			6X2=12
			Correct
			
			IInd Time Run
			7X4=26
			Incorrect

User Defined Function

User defined function is a function, which is written by the programmer in the program for a particular task.

Function Definition: It is a module of a program, which requires Function Header and Function Body. Function Header contains <Return Type>, <Function Name> and <Formal Parameter List>, whereas Function Body has { } enclosure to hold C++ statements, which are executed, when the function is called.

Syntax for **defining a function** in C++ is as follows:

```
<Return Type> <Function Name>(<Formal Parameter List>)      [Function Header]
{                                                                  +
  <Statement1>;                                             [Function Body]
  <Statement2>;
  <Statement3>;
  :
}
```

Example 1: To display a message on screen	Example 2: To display N integers on screen

`void Display()`
`{`
`  cout<<"Hello!"<<endl;`
`  cout<<"I m user defined funct."<<endl;`
`}`
	`void Count(int N)`
	`{`
	`  for (int I=1;I<=N;I++)`
	`  cout<<I<<endl;`
	`}`

Once the function is defined in the program, it can be called and executed any number of times with different set of parameters (as per number of parameters and their respective data types).

To call a function, we use the following syntax:

```
<Function Name>(<Actual Parameter List>);
```

Example 1 (Contd..)	Example 2 (Contd..)

`void main()`
`{`
`  Display(); //Function Call`
`  cout<<"---------------"<<endl;`
`  Display(); //Function Call`
`  cout<<"###############"<<endl;`
`  Display(); //Function Call`
`}`
`/* Output`
`Hello!`
`I m user defined funct.`
`---------------`
`Hello!`
`I m user defined funct.`
`###############`
`Hello!`
`I m user defined funct.`
 	
`*/`
	`void main()`
	`{`
	`  Count(4); //Function Call`
	`  cout<<"------"<<endl;`
	`  int N;`
	`  cout<<"N:";cin>>N;`
	`  Count(N); //Function Call`
	`}`
	`/* Output`
	`1`
	`2`
	`3`
	`4`
	`N:3`
	`1`
	`2`
	`3`
	`*/`

Note:

(1) It is essential to mention the datatype of each FORMAL Parameter explicitly.

(2) Mention of datatype of Actual Parameter is not required.

Example 3: To display table of M, N times

```
void Table(int M,int N)
{
  for (int I=1;I<=N;I++)
    cout<<M<<"x"<<I<<"="<<M*I<<endl;
}

void main()
{
  Table(2,3);
  int L,Times;
  cout<<"Number:";cin>>L;
  cout<<"Times:";cin>>Times;
  Table(L,Times);
  Table(L+2,Times+2);
}
```

Note: In this example, we have used constant, variable as well as expression as actual Parameters

```
/* --- Sample Output ---
2x1=2
2x2=4
2x3=6
Number:5
Times:4
5x1=5
5x2=10
5x3=15
5x4=20
7x1=7
7x2=14
7x3=21
7x4=28
7x5=35
7x6=42
7x7=49
*/
```

Example 4: To display triangle pattern of N rows

```
void Line(int N)    //Function 1
{
  for (int I=1;I<=N;I++)
    cout<<'*';
  cout<<endl;
}

void Pattern(int N)//Function 2
{
  for (int I=1;I<=N;I++)
  {
    for (int J=1;J<=I;J++)
      cout<<J;
    cout<<endl;
  }
}

void main()
{
  Pattern(3);    //Call for Function 2
  Line(10);      //Call for Function 1
  Pattern(5);    //Call for Function 2
}

/* --- Output ---
1
12
123
**********
1
12
123
1234
12345
*/
```

Function with return type other than void

The function in which the return type is **other than void**, will essentially have one last executable statement as **return <Value>**. This return keyword is responsible for returning a result from the function. In the Even() function of Example 5, you see two return statements, out of which only one will be **executable return** as it is part of if statement. Sum()function in Example 6 has only one return statement, which is responsible for returning value of sum of series.

Another thing to note is that while calling such functions, you will require to assign the returned value in a variable (as shown in first call of Sum() function in main() of Example 6) **or** directly use the value in if statement (as in main() function of Example 5) **or** in any part of expression **or** directly display the value in cout(as shown in second call of Sum() function in main() of Example 6).

Example 5: To check if a number is even or not

```
int Even(int N)
{
  if (N%2==0)
    return 1;
  else
    return 0;
}
void main()
{
  int N;
  cout<<"Number to check:";cin>>N;
  if (Even(N))
    cout<<"It is an Even Number";
  else
    cout<<"It is an Odd Number";
}
/* --- Output ---
Number to check:86
It is an Even Number
*/
```

Example 6: To find sum of 2+4+6+.... N times

```
int Sum(int N)
{
  for (int I=1,S=0;I<=N;I++)
    S+=(2*I);
  return S;
}
void main()
{
  int Terms,SS;
  cout<<"Terms:";cin>>Terms;
  SS=Sum(Terms);
  cout<<"Sum Series 1:"<<SS<<endl;
  cout<<"Sum Series 2:"<<Sum(5)<<endl;
}
/* --- Output ---
Terms:10
Sum Series 1:110
Sum Series 2:30
*/
```

Example 7: To calculate Simple Interest

```
float SI(float P,float R,int T)
{
  return (P*R*T)/100;
}

void main()
{
  float Pr,Rt,Simple;int Time;
  cout<<"Principle  :";cin>>Pr;
  cout<<"Rate%      :";cin>>Rt;
  cout<<"Time(Years):";cin>>Time;
  Simple=SI(Pr,Rt,Time);
  cout<<"Simple Interest:"
      <<Simple<<endl;
  cout<<"Amount with Interest:"
      <<Pr+Simple<<endl;
}

/* --- Output ---
Principle  :5000
Rate%      :10
Time(Years):5
Simple Interest:2500
Amount with Interest:7500
*/
```

Example 8: To find Grade for given Marks

```
char GradeMe(float Marks)
{
  if (Marks>=85)
    return 'A';
  else if (Marks>=70)
    return 'B';
  else if (Marks>=55)
    return 'C';
  else
    return 'D';
}
void main()
{
  int M;char Ch;
  do
  {
    cout<<"Enter Marks:";cin>>M;
    cout<<"Your Grade:"<<GradeMe(M)
        <<endl;
    cout<<"More(Y/N)?";cin>>Ch;
  }
  while (Ch!='N');
}
/* --- Output ---
Marks:95
Your Grade:A
More(Y/N)?Y
Marks:56
Your Grade:C
More(Y/N)?N
*/
```

Function Prototype

In the earlier examples, you can notice that we have written function definitions before the function is called. The reason for this is that if we write the definition of function after the function call, C++ compiler will not be able locate the function which will result in syntax error. However, calling a function before it's definition is possible by adding a **Function Prototype** before the function call. Writing Function Prototype will allow us to call Functions from anywhere in the program. Function Prototype should be written in the beginning of the program and has the following characteristics:

1. It is similar to the Function Header, terminated with a semicolon and does not have Function Body attached to it.
2. In Function Prototype, mentioning the name(s) of the parameter(s) is optional, whereas mentioning the data type is a must.

Example 9a: In this program, the functions Display() and Show() can not be called from main()

```
void main()
{
  Display();//Function can not be called
  Show();   //Function can not be called
}
void Display()
{
  cout<<"Function Display"<<endl;
}
void Show()
{
  cout<<"Function Show"<<endl;
  Display();//Function Display can be called
}
```

Note:Here the function Display() can be called from function Show() as it is defined above it but vice versa will not be possible i.e.,Show() can not be called from Display().

Example 9b: In this program, all the functions Display() and Show() can be called from any function of the program.

```
void Display();//Function Prototype
void Show();   //Function Prototype
void main()
{
  Display();//Function can be called
  Show();   //Function can be called
}
void Display()
{
  cout<<"Function Display"<<endl;
}
void Show()
{
  cout<<"Function Show"<<endl;
  Display();
}
```

The following programs programs will run identically, one has parameter names in the prototypes and the other does not.

Example 10a: To display tables separated by line

```
void Table(int M,int N);
void Line(int N);
void main()
{
  Table(5,3);
  Line(5);
  Table(3,4);
}
void Table(int M,int N)
{
  for (int I=1;I<=N;I++)
    cout<<M<<"x"<<I<<"="<<M*I<<endl;
}
void Line(int N)
{
  for (int I=1;I<=N;I++)
    cout<<'*';
  cout<<endl;
}
```

Example 10b: To display tables separated by line

```
void Table(int ,int );//No parameter names
void Line(int );//No parameter name
void main()
{
  Table(5,3);
  Line(5);
  Table(3,4);
}
void Table(int M,int N)
{
  for (int I=1;I<=N;I++)
    cout<<M<<"x"<<I<<"="<<M*I<<endl;
}
void Line(int N)
{
  for (int I=1;I<=N;I++)
    cout<<'*';
  cout<<endl;
}
```