FUNCTIONS

A named set of statements (subprogram) that may take some values from its caller and returns a value after performing a specific job.

Benefits of using functions:

Provides modularity ,Provides reusability,Makes program modification easier,Makes program smaller and simpler.

Types of Functions in C++

Two types of functions are available in C++.

1. **Library functions:** Provided by C++ compiler in library and can be used by including respective header file.
2. **User defined functions:** Defined by user.

Defining Functions & returning values from functions.

A function is a group of statements that is executed when it is called from some point of the program. The following is its format:

```
<return type> <function name> ( parameter1, parameter2, ...) {
statements }
```

where:

- return type is the data type specifier of the data returned by the function. If function does not returns any value return type should be void.
- name is the identifier by which it will be possible to call the function.
- parameters (as many as needed): Each parameter consists of a data type specifier followed by an identifier, like any regular variable declaration (for example: int x) and which acts within the function as a regular local variable. They allow to pass arguments to the function when it is called. The different parameters are separated by commas.
- statements is the function's body. It is a block of statements surrounded by braces { }.

Example:

```
#include <iostream.h>
int addition (int a, int b)
{
int r;
r=a+b;
return (r);
}
void main ()
{ int z;
z = addition (5,3);
cout << "The result is " << z;
}
```

Output :

```
The result is 8
```

Scope of variables:

The scope of variables declared within a function or any other inner block is only their own function or their own block and cannot be used outside of them.

Therefore, the scope of local variables is limited to the same block level in which they are declared. Nevertheless, we also have the possibility to declare global variables;
These are visible from any point of the code, inside and outside all functions.
In order to declare global variables you simply have to declare the variable outside any function or block; that means, directly in the body of the program.

1. Differentiate between global & local variable with a suitable example.

Ans: **Local variable**: The variables that have a local scope are known as local variables. In other words, variable that can used only in the function or the block where they can declared are known as local variable.
Global Variable: The variable that have a global scope are known as global variables. In other words, variables that can be used in all bocks and function in a program are known as global variables.

Parameters (or Arguments) and their types:

Parameters are the values passed to a function for its working.
There are two types of parameters:
1. **Actual parameters:** Parameters that appears in the calling statement of the function are known as actual parameters.
2. Formal Parameters: Parameters that appears in the function prototype or header statement of the function definition are known as formal parameters.

What is the difference between Actual Parameter and Formal Parameter? Give an example in C++ to illustrate both types of parameters.

Ans. The parameters in the function call statement (or calling function) are called as Actual Parameters.
The parameters in the function definition (or called function) are called as Formal Parameters.

Calling Functions:

There are two ways to call the function by passing the parameters or arguments to a function :
➢ Call by value / Pass by value

➢ Call by reference / Pass by reference

Parameters passed by value:

```
int x=5, y=3, z;
z = addition ( x , y );
```

Parameters passed by reference:

```
#include <iostream>
void duplicate (int& a, int& b, int& c)
{ a*=2;
b*=2;
c*=2;
```

```
}
int main ()
{ int x=1, y=3, z=7;
duplicate (x, y, z);
cout << "x=" << x << ", y=" << y << ", z=" << z;
return 0;
}
```

Output:

```
x=2, y=6, z=14
```

If when declaring the following function: `void duplicate (int& a, int& b, int& c)` we had declared it this way: `void duplicate (int a, int b, int c)` i.e., without the ampersand signs (&), we would have not passed the variables by reference, but a copy of their values instead, and therefore, the output on screen of our program would have been the values of x, y and z without having been modified. For example,

```
// passing parameters by value
#include <iostream>
void duplicate (int a, int b, int c)
{ a*=2;
b*=2;
c*=2;
cout << "a=" << a << ", b=" << b << ", c=" << c;
}
int main ()
{ int x=1, y=3, z=7;
duplicate (x, y, z);
cout << "x=" << x << ", y=" << y << ", z=" << z;
return 0;
}
```

Output :

```
a=2, b=6, c=14
x=1, y=3, z=7
```

Passing by reference is also an effective way to allow a function to return more than one value.

A Comparison:

	Parameters passed by value	Parameters passed by reference
1.	When we call a function by passing parameters by value, the values of actual parameters get copied into the formal parameters but not the variables themselves.	When we call a function by passing parameters by reference, formal parameters create a reference or pointer directly to the actual parameters.
2.	The changes made in the values of formal parameters will not be reflected back to the actual parameters.	The changes made in the values of formal parameters will be reflected back to the actual parameters.

3. Example –

```
// passing parameters by value
#include <iostream.h>
void swap (int a, int b)
{ a=a+b;
b=a-b;
c=a-b;
cout << "Values inside function
\n";
cout<<"a=" << a << ", b=" << b;
}
int main ()
{ int x=1, y=3;
swap (x, y);
cout << "Values inside main \n";
cout << "x=" << x << ", y=" << y;
return 0;
}
Output :
Values inside function
a=3, b=1
Values inside main
x=1, y=3
```

(**Note:** Changes made in formal parameters (a and b) are not reflected back to actual parameters (x and y))

Example -

```
// passing parameters by reference
#include <iostream.h>
void swap (int &a, int &b)
{ a=a+b;
b=a-b;
c=a-b;
cout << "Values inside function
\n";
cout<<"a=" << a << ", b=" << b;
}
int main ()
{ int x=1, y=3;
swap (x, y);
cout << "Values inside main \n";
cout << "x=" << x << ", y=" << y;
return 0;
}
Output :
Values inside function
a=3, b=1
Values inside main
x=3, y=1
```

(**Note:** Changes made in formal parameters (a and b) are reflected back to actual parameters (x and y))

Default values in parameters:

When declaring a function we can specify a default value for each of the last parameters. This value will be used if the corresponding argument is left blank when calling to the function. To do that, we simply have to use the assignment operator and a value for the arguments in the function declaration. If a value for that parameter is not passed when the function is called, the default value is used, but if a value is specified this default value is ignored and the passed value is used instead. Example:

```
// default values in functions
#include <iostream>
using namespace std;
int divide (int a, int b=2)
{ int r;
r=a/b;
return (r);
}
int main ()
{ cout << divide (12);
```

```
cout << endl;
cout << divide (20,4);
return 0;
}
```

1 Differentiate between call by value & call by reference with a suitable example.

Call By Value	Call By Reference
The formal argument holds a copy of the value of the actual argument.	The formal argument is a reference of the actual argument.
The actual argument value cannot be changed by the function.	The actual argument value can be changed by the function.
The actual argument can be a variable, a constant or an expression.	The actual argument must be a variable.

Example:-

Call by value:

```
#include<iostream.h>
void swap(int a,int b)
{
int c;
c = a;
a = b;
b = c;
}
void main( )
{
int x, y ;
cout<<"enter two no";
cin>>a>>b;
cout<<"Before calling swap function value of x = "<<x<<endl;
cout<<"Before calling swap function value of y = "<<y<<endl;
swap(x,y);
cout<<"After calling swap function value of x = "<<x<<endl;
cout<<"After calling swap function value of x = "<<x<<endl;
}
```

Call by Reference:

```
#include<iostream.h>
void swap(int &a,int &b)
{
int c;
c = a;
a = b;
b = c;
}
void main( )
{
int x, y ;
cout<<"enter two no";
cin>>a>>b;
cout<<"Before calling swap function value of x = "<<x<<endl;
cout<<"Before calling swap function value of y = "<<y<<endl;
swap(x,y);
```

cout<<"After calling swap function value of x = "<<x<<endl;
cout<<"After calling swap function value of x = "<<x<<endl; }

1: Find the output of the following program:

```
#include <iostream.h>
void Changethecontent(int Arr[], int Count)
{for (int C = 1;C < Count; C++)
Arr[C-1] += Arr[C];
}
void main()
{int A[] = {3,4,5}, B[]={10,20,30,40}, C[]={900,1200};
Changethecontent(A, 3);
Changethecontent(B, 4);
Changethecontent(C, 2);
for (int L = 0;L < 3;L++) cout<<A[L]<< "#";
cout<<endl;
for (L = 0;L < 4;L++) cout << B[L] << "#";
cout << endl;
for (L = 0;L < 2;L++) cout<<C[L] << "#";
}
```

Find the output of the following program:

```
#include<iostream.h>
void Indirect(int Temp=20)
{for (int 1=10; I<=Temp; I+=5)
cout<<I<<", ";
cout<<endl;
}
void Direct (int &Num)
{ Num+=10;
```

}

1 What do you understand by enum or Enumerated data type?

Ans: **Enumerated data type**

The enum specifier defines the set of names which are stored internally as integer constant. The first name was given the integer value 0, the second value 1 and so on. for example: `enum months{jan, feb, mar, apr, may} ;` It has the following features:

- It is user defined.
- It works if you know in advance a finite list of values that a data type can take.
- The list cannot be input by the user or output on the screen.

OR

The enumeration is a set of integer constant, written as identifiers, specifying the possible value that can be assigned to the enumeration variable. These set of possible values are known as enumerators. By default the first enumerator in the enumeration data type is assign the value zero.
Example:
enum color(red,green,blue};
cout<<blue;
Output 2

2 **What do you understand by union?**

Ans: Unions, like a structures, consist of members of the same data type or different data types. The only difference is that all the members of a union share common storage are, but in structure all the member assigned its own unique storage area.

3 **Define the term typedef with suitable example.**
Ans: The typedef keyword allows the programmers to define a new data types based on the existing data type. The typedef define a new name for an existing data type. Example: typedef int integer;

↓↓↓
LIBRARY FUNCTION C++ provides many built in functions that saves the programming time

Mathematical Functions Some of the important mathematical functions in header file **math.h** are

Function	Meaning
sin(x)	Sine 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 (x) where x (measured in radians)
acos(x)	Cos-1 (x) where x (measured in radians)
exp(x)	Exponential function of x (ex)
log(x)	logarithm of x
log 10(x)	Logarithm of number x to the base 10
sqrt(x)	Square root of x
pow(x, y)	x raised to the power y
abs(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.

Function	Meaning
isalpha(c)	It returns True if C is an uppercase letter and False if c is lowercase.
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 or lowercase) otherwise False.
islower(c)	It returns True if C is 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 locations to hold S2.
strcpy(S1, S2)	It copies character string S2 to string S1. The 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 equal to S2, S1 greater than S2 or S1 less than S2.
strcmpi((S1, S2)==0) strcmpi((S1, S2)>0) strcmpi((S1, S2) <0)	It compares S1 and S2 ignoring case and finds out whether S1 equal to S2, S1 greater than S2 or S1 less than S2.
strrev(s)	It converts a string s into its reverse
strupr(s)	It converts a string s into upper case
strlwr(s)	It converts a string s into lower case

Console I/O functions The following are the list of functions are in **stdio.h**

getchar()	It returns a single character from a standard input device (keyboard). It takes no parameter and the returned value is the input character.
putchar()	It takes one argument, which is the character to be sent to output device. It also returns this character as a result.
gets()	It gets a string terminated by a newline character from the standard input stream stdin.
puts()	It takes a string which is to be sent to output device.

General purpose standard library functions

The following are the list of functions are in **stdlib.h** randomize() It initializes / seeds the random number generator with a random number random(n)
It generates a random number between o to n-1
atoi(s) It converts string s into a numerical representation.
itoa(n) It converts a number to a string

Some More Functions

the getch() and getche() functions
the general for 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 and press enter key.
The value of the character returned from getche() is assigned to ch. The getche() fuction echoes the character to the screen.
Another function, getch(), is similar to getche() but does not echo character to the screen.

ASSIGNMENT

Question.

Give the output of the following program segment

```
char *NAME = "CoMPutER";
for (int x = 0: x < strlen(NAME); x++)

if (islower(NAME [x]))

      NAME [x] = toupper(NAME[x]);

else
      if (isupper(NAME[x]))

      if (x%2 ==0)
      NAME[x] = tolower(NAME[x]);
      else
      NAME[x] =NAME[x-1];
      puts(NAME);
```