BCA SEM 2
Advanced C Programming

- Introduction and need of User Defined Functions
- Components of User Defined Functions
- Categories of User Defined Functions
- Recursion

USER-DEFINED FUNCTION(UDF)

Definition: Function:-

A function is a self-contained block of code that performs a particular task.

NEED FOR USER-DEFINED FUNCTION:

The program may become too large and complex and as a result the task of debugging, testing and maintaining becomes difficult. If a program is divided into functional parts, then each part may be independently coded and later combined into a single unit. These independently coded programs are called subprograms. In C, such subprograms are referred to as 'functions'.

Advantage of User-defined Function:-

1) In facilitates top-down modular programming.
2) The length of a source program can be reduced by using function at appropriate places.
3) It easy to locate and isolate faulty function.
4) A function may be used by many other programs.

Components of Function:-

There are three components that are related to function

1) Function Declaration
2) Function Definition
3) Function Call

Function Declaration or function prototype.	
Syntax	returntype functionname (arguments);
Returntype	It indicates function may return value or not, and if return then specify which type of value to be return.
	It may be int, float, char, void etc. It is optional and the by default return type is int.
functionname	Functionname indicates name of the user define function. It must be valid identifier.
Arguments	The parameter list indicates number of parameters and parameters type which is passed in function call.
Example	(1) void max(void); (2) int min(int,int);

Definition of Function:

Function definition includes the following elements. All the six elements are grouped into two parts namely

Function Header	a) function name b) function type c) list of parameters
Function body	d) local variable declaration e) function statement f) return statement

Syntax	returntype function_name(arguments)
	{
	local variable declaration;
	executable statement1;
	executable statement2;
	……
	return statement;
	}

Returntype	The function type specifies the type of the value like float or double that the function is expected to return to the program calling the function. If the function is not returning anything, then we need to specify the return type **void.**
functionname	The function name is any C valid identifier and therefore must follow the same rules of variable name.
Arguments	The parameter list declares the variables that will be receive the data sent by the calling program. They represent actual input values, they are often **referred to as <u>formal parameters.</u>**
Return	A return statement that returns the value evaluated by the function. Return statement can take one of the following form: return; or return (expression);

Example	returtn type ——— function name
	parameter list
	float mul(float x,float y)
	{
	float ans; ← Local Variable
	ans=x*y; ← Executable Statement
	return ans; ← Return Statement
	}

Function Calls:
A function can be called by simply using the function name followed by a list of actual parameters or arguments, if any enclosed within the parentheses. When function is called at that time control transfer to the function defination.

Syntax	function_name(argument list);

Example	void main()
	{
	
	float y;
	y=**mul(10,5);** ← Function Call
	printf("%f",y);
	}

Categories of Function:-

Users define function categories as below.

1) No passing parameter, No Return value
2) Passing parameter and, No Return value
3) No Passing Parameter, Return value
4) Passing Parameter, Return value

1) No Passing Parameter, No Return Value.

In this category function does not take any value, and also does not return any value. It means there is no data transfer between calling function and the called function, **only a transfer control but not data.**

Syntax	**Function Declaration**
	void functionname( );
	Function does not return any value at that time take **void** as a return type. Function does not take any parameter then just write the **empty parenthesis** after function name.
	Function Definition
	void functionname( )
	{
	//body
	}
	Function does not return any value so, **return statement is not include in function body**.
	Function Call
	functionname( );
	Function calls transfer control to the function body. Function does not take any parameter then just write the **empty parenthesis** after function name.

Example	//add two numbers using UDF.

```
#include <stdio.h>
#include <conio.h>
void main()
{
        void addition();   //function declaration
        clrscr();
        addition();        //function call
        getch();
}


void addition()            //function definition
{
        int x,y,ans;
        printf("\n Enter value of X:");
        scanf("%d",&x);
        printf("\n Enter value of Y:");
        scanf("%d",&y);
        ans=x+y;
        printf("\n %d + %d = %d",x,y,ans);
}

/*
Input

Enter value of X:5

Enter value of Y:5

Output

 5 + 5 = 10
*/
```

2) Passing Parameter, No Return Value.

In this category function take parameter but does not return any value. It means parameter read in called function and passed them in function call to the calling function.

Syntax

Function Declaration

```
void functionname(parameterlist);
```

Function dose not return any value at that time take **void** as a return type. If Function take any parameter then specify the list of parameter in parenthesis.

Function Definition

```
void functionname(parameterlist)
{
        //body
}
```

Function does not return any value so, **return statement is not include**

	in function body. **Function Call** functionname(parameterlist); Function call transfer control to the function body. Function take parameters so pass parameter in parenthesis after function name.

Example	//add two numbers using UDF.
	#include <stdio.h>
	#include <conio.h>
	void main()
	{
	void addition(int,int); //pass two parameter of int type
	int x,y;
	clrscr();
	printf("\n Enter value of X:");
	scanf("%d",&x);
	printf("\n Enter value of Y:");
	scanf("%d",&y);
	addition(x,y); // x & y is known as actual parameter
	getch();
	}
	void addition(int a,int b) // a & b is known as formal parameter
	{
	int ans;
	ans=a+b;
	printf("\n Addition = %d ",ans);
	}
	/*
	Input
	Enter value of X:5
	Enter value of Y:5
	Output
	Addition = 10
	*/

3) No Passing Parameter, Return Value. In this category function does not take parameter but return value, with the help of return statement we can return a single value.	

Syntax	**Function Declaration**
	return-type functionname( );
	If function return int value then take int as return-type, if return float value then take float as return type and so on. And write empty parenthesis after the function name because no passing parameter.
	Function Definition
	returntype functionname( )
	{

<table>
<tr><td></td><td>
//body
return(exp);
}
Function return value so, return statement is include in function body.
Function Call
varname = functionname( );
Function call transfer control to the function body. The return statement return value which is stored in specified varname.
</td></tr>
<tr><td>Example</td><td>
//add two numbers using UDF.
#include <stdio.h>
#include <conio.h>
void main()
{
int addition(); // function declare with return type int.
int ans;
clrscr();
ans=addition(); // function call and answer is store in ans variable.
printf("\n Addition = %d ",ans);
getch();
}
int addition() // functuion defination
{
int x,y,sum;
printf("\n Enter value of X:");
scanf("%d",&x);
printf("\n Enter value of Y:");
scanf("%d",&y);
sum=x+y;
return(sum); // return value of sum variable to the called function
}
/*
Input
Enter value of X:5
Enter value of Y:5
Output
Addition = 10
*/
</td></tr>
<tr><td colspan="2">
4) Passing Parameter, Return Value.
In this category function take parameter and return value, with the help of return statement we can return a single value.
<table>
<tr><td>Syntax</td><td>Function Declaration
return-type functionname(parameterlist);</td></tr>
</table>
</td></tr>
</table>

If function return int value then take int as return-type, if return float value then take float as return type and so on. And write parameter list in parenthesis after the function name.

Function Definition

```
returntype functionname(parameterlist)
{
        //body
         return(exp);
}
```

Function return value so, **return statement is include in function body**.

Function Call

```
varname = functionname(parameterlist);
```

Function call transfer control to the function body. The return statement return value which is stored in specified varname.

Example

```
//add two numbers using UDF.
#include <stdio.h>
#include <conio.h>
void main()
{
        int addition(int,int);  //retutn type is int and take two int type value
        int ans,x,y;
        clrscr();
        printf("\n Enter value of X:");
        scanf("%d",&x);
        printf("\n Enter value of Y:");
        scanf("%d",&y);
        ans=addition(x,y); //function call and return answer in ans variable.
        printf("\n Addition = %d ",ans);
        getch();
}
int addition(int a,int b)     //function defination.
{
        int sum;
        sum=a+b;
        return(sum);
}
/*    Input
Enter value of X:5

Enter value of Y:5

Output

 Addition = 10
*/
```

Formal Parameter and Actual Parameter:

Actual Parameters are parameters as they appear in function call.
Formal Parameters are parameters as they appear in function declaration.
Example :

```
 Void main()
{
    -----
   ------
   Swap(x,y);     // actual parameters
   getch();
}

Int swap (int a, int b)     // Formal Parameters
{
----
----
----
}
```

Array as an argument to UDF

In C programming, you can pass en entire array to functions. Before we learn that, let's see how you can pass individual elements of an array to functions.

Passing array elements to UDF

Passing array elements to a function is similar to passing variables to a function.

Example:

```
// Program to calculate the sum of array elements by passing to a function

#include <stdio.h>
#include <conio.h>

float calculateSum(float age[]);

void main()
{
   float result, age[] = {23.4, 55, 22.6, 3, 40.5, 18};


   // age array is passed to calculateSum()
   result = calculateSum(age);
   printf("Result = %.2f", result);
   getch();
}
```

```
float calculateSum(float age[]) {

  float sum = 0.0;

  for (int i = 0; i < 6; i++) {
          sum += age[i];
  }

  return sum;
}
```

Output

```
Result = 162.50
```

To pass an entire array to a function, only the name of the array is passed as an argument.

```
result =  calculateSum(age);
```

However, notice the use of [] in the function definition.

```
float calculateSum(float age[])
{
    ... ..
}
```

This informs the compiler that you are passing a one-dimensional array to the function.

Recursion

Recursion is a programming technique that allows the programmer to express operations in terms of themselves.

In C, this takes the form of a function that **calls itself.**

Example.

The following example calculates the factorial of a given number using a recursive function −

```
#include <stdio.h>
long int factorial(int);

void  main()
{
   int i = 5;
   printf("Factorial of %d is %d\n", i, factorial(i));
   getch();
```

```
}

long int factorial(int i)
{

   if(i <= 1)
   {
      return 1;
   }
   return i * factorial(i - 1);
}
```

When the above code is compiled and executed, it produces the following result −

```
Factorial of 5 is 120
```

Advantages of **Recursion**:

1. Reduce unnecessary calling of function.
2. Through **Recursion** one can Solve problems in easy way while its iterative solution is very big and complex.

	Unit-1
	MCQ
1	The parameters used in a function call are called _____. (a) arguments (b) formal **(c) actual** (d) none
2	The variable declared inside a function is called _____. (a) global **(b) local** (c) function (d) none
3	By default _____ is a return type of a C function. (a) void (b) float **(c) int** (d) none
4	In prototype declaration, specifying _____is optional. (a) return type (b) data type (c) semicolon **(d) parameter name**
5	A function which calls itself is known as _____. (a) reverse **(b) recursive** (c) reserve (d) none
6	Structure is a _____data type. (a) built-in **(b) derived** (c) user defined (d) none
7	Function header consists of _____parts. (a) one (b) two **(c) three** (d) none
8	A function definition is also known as _____. **(a) function implementation** (b) function call (c) function type (d) none
9	The parameter is also known as _____ **(a) argument** (b) variable (c) data type (d) array
10	A parameter list in function can be separated by_____ (a) Question marks (?) **(b) Commas (,)** (c) Exclamatory marks (!) (d) none
11	A function can be surrounded by _____. **(a) parentheses** (b) square brackets (c) queerly brackets (d) none
12	The following are wrong declaration in function definition._____ (a) int sum(int a , float b) (b) float sum(int a ,float b) **(c) int sum(int a,b)** (d) float sum(float a, float aa)
13	A _____statement that returns the value evaluated by the function. (a) goto (b) break **(c) return** (d) none
14	When the function is called _____argument is passed. **(a) actual** (b) formal (c) actual & formal (d) none
15	A function declaration is also known as _____ (a) function implementation (b) function call (c) function type **(d) function prototype**.
16	If the functions are declare in the global declaration section the prototype is referred as _____prototyped. **(a) global** (b) local (c) formal (d) none

	Unit-1
	Short Questions
1	Write and explain syntax for function declaration.
2	Write and explain syntax for function definition.
3	Write and explain syntax for function call.
4	List all the categories of user-defined functions.
5	What is actual and formal parameter?
6	Which the advantages are of divide the program into functions?
7	Which are the components of a user-defined function?
8	Explain recursion in brief.
	Unit-1
	Long Questions
1	Explain Function declaration with syntax and example.
2	Explain Function definition with syntax and example.
3	Explain Function call with syntax and example.
4	Explain function with no return type and no parameters.
5	Explain function with no return type and with parameters.
6	Explain function with return type and no parameters.
7	Explain function with return type and with parameters.
8	Explain recursive function with example.
9	Explain passing 1-D array to function with example.