Unit-7

Functions

Functions:

- Functions are the building blocks of a program.
- A function is a self-contained program segment or the block of segments that perform some specific and well defined task.
- A program consists of one or more functions.
- C program must contain a function called main function from where the execution of program begins.
- In a program there may be any number of functions and their definition can appear in any order but they should be independent of one another.
- Each function has a unique name.
- We need to access the function (which is known as calling a function) in order to carry out the intended task.

Advantages of using the function:

- Functions increase code reusability by avoiding rewriting of same code over and over.
- If a program is divided into multiple functions, then each functions can be independently developed. So, program development will be easier.
- Program development will be faster.
- Program debugging will be easier.
- Functions reduces program complexity.
- Length of the program can be reduced by using the functions at appropriate places.
- The recursive call is possible through function.
- Functions increases program readability.

Types of Functions:

There are two types of functions and they are:

1. Library Functions/ Built-in Functions
2. User-Defined Functions

1. Library Functions:

- The Library functions are also know as **built-in functions.**
- The **predefined functions** which are available in the C standard library header files and perform the some specific pre-defined tasks are called library functions.
- We can not change or alter the meaning of library functions.
- We do not need to declare and define the library functions.
- For examples: printf( ), scanf( ), getch( ), strlen( ), strrev( ), pow( ), sqrt( ) etc.

2. User-Defined Functions:

- The functions which are created by the programmer to perform any specific task according to the requirements are called user-defined functions.
- To use user-defined we must declare and define.
- For example: sum( ), find( ), check( ), area( ), factorial( ), area_circle( ) etc.

Differentiate between Library functions and User-Defined functions:

Library Function	User-Defined Function
1. The library function is a predefined function in a header file or preprocessor directive.	1. User-defined functions is not a predefined function, it is created by the programmer according to the need.
2. The programmer can simply use this function by including respective header file.	2. The programmer has to declare, define and use this function by them self.
3. The length of program is short.	3. Usually the length of program is long.
4. Program development time will be faster.	4. Program development time will be usually slower.
5. The program using library functions will be usually simple.	5. The program using user-defined function will be usually complex.
6. This function requires header file to use it.	6. This function requires **function prototype** to use it.
7. Example: printf(), scanf(), getch(), pow(), strlen() etc.	7. Example: sum(), diff(), area(), fact() etc.

Components of User-Defined Functions:

1. Function Prototype (Function Declaration)
2. Function Call
3. Function Definition

1. Function Prototype (Function Declaration):

- Function declaration (also called function prototype) means declaring the properties of a function to the compiler.
- **Syntax:** return_type name_of_function (parameter list);
- **Example**: int sum(int, int);

  float area(int, float);

2. Function Call:

- A function call is a statement that activates the execution of the function.
- When a function call is encountered in a program, the control of the program jumps to the called function and the execution of the function starts.
- **Syntax**: function_name (argument list);
- **Example**: sum(a,b);

  area(r);

3. Function Definition:

- A function definition provides the actual body of the function.
- It contains the statements that will be executed when the function is called.
- Function definition contains function name, list of parameters with type and return type of the function and statements of the function.
- **Syntax**: return_type function_name (parameter list)

```
{
    declarations and statements;
}
```

Return Statement and Void Statement of Function:

- Return statement is used to return a value to the calling function by using **return** keyword.
- **Syntax**: return expression; **Example**: return (a + b);
- When a return statement is executed, it immediately transfers the control back to the calling function or calling program.
- A function definition can contain multiple return statements. However, only one gets executed.
- If a function does not return then we say the function's return type is void. It means this function do not give output to the caller function.

Simple Example:

```
#include<stdio.h>
#include<conio.h>
int sum(int,int); ----------Function Prototype/Function Declaration
int main()
{
    int a=10,b=20,s;
    s=sum(a,b); ----------Function Call(Calling Function)
    printf("Sum=%d",s);
    getch();
}
int sum(int x,int y) ----------Function Definition(Called Function)
{
    int su;
    su=x+y;
    return su;
}
```

Types of User-Defined Function:

Depending on the presence or absence of arguments and whether the value is returned or not, the function can be categorized as:

1. Function with both arguments and return values
2. Function with arguments and but no return values
3. Function without arguments and but with return values
4. Function without any arguments and return values

1. Function with both arguments and return values:

- This type of function has two-way communication. Here, an argument is passed from the calling function to the called function, and there will also be return statement in the called function.

Example:

```
int rect_area(int, int);
void main()
{
    int l,b,a;
    printf("Enter l and b:");
    scanf("%d%d",&l,&b);
    a=rect_area(l,b);----------------Calling Function
    printf("The area of rectangle is %d",a);
    getch();
}
int rect_area(int l,int b) ---------------Called Function
{
    int area;
    area=l*b;
    return area;
}
```

2. Function with arguments and but no return values:

- This type of function has one-way communication. Here, an argument is passed from the calling function to the called function, but there is no need of return statement in the called function.

Example:

```
void rect_area(int, int);
void main()
{
    int l,b;
    printf("Enter l and b:");
    scanf("%d%d",&l,&b);
    rect_area(l,b);----------------Calling Function
    getch();
}
void rect_area(int l,int b) ---------------Called Function
{
    int area;
    area=l*b;
    printf("The area of rectangle is %d",area);
}
```

3. Function without arguments and but with return values:

- This type of function also has one-way communication. Here, an argument is not passed from the calling function to the called function, but there is need of return statement in the called function.

Example:

```
int rect_area(void);
void main()
{
    int a;
    a=rect_area( );----------------Calling Function
    printf("The area of rectangle is %d",a);
    getch();
}
int rect_area( ) ---------------Called Function
{
    int area,l,b;
    printf("Enter l and b:");
    scanf("%d%d",&l,&b);
    area=l*b;
    return area;
}
```

4. Function without any arguments and return values:

- If we don't want to return a value we must use the return type void and miss out the return statement. Here, we simply call a function without passing arguments and the called function doesn't have to return any values to the calling function.

Example:

```
void rect_area( );
int main( )
{
    rect_area( );
    getch( );
    return 0;
}
void rect_area( )
{
    int l,b,area;
    printf("Enter l and b:");
    scanf("%d%d",&l,&b);
    area=l*b;
    printf("The area of rectangle is %d",area);
}
```

Recursion Function:

- A recursive function is one which calls itself.
- A function will be recursive if it contains following features:

1. The function should call itself or defined in terms of its previous result.

2. Problem statement must include a stopping condition i.e. we must have an if statement somewhere to force the function to return without the recursive call being executed, otherwise function will never return.

- The concept of using recursive function to repeat the execution of statements many times is known as recursion.
- This process is used for repetitive computations in which each action is stated in term of previous result.

Example:

- C Program to input a number and find its factorial using recursive function.

```
int factorial(int);
void main()
    {
        int n,fact;
        printf("Input a number:");
        scanf("%d",&n);
        fact=factorial(n);
        printf("The factorial of %d is %d",n,fact);
        getch();
    }
int factorial(int n)
    {
        if(n==0 || n==1)
        return 1;
        else
        return (n*factorial(n-1));
    }
```

Example:

- C Program to input a number and calculate the sum of all natural numbers up to that number.

```
int sum(int);
void main()
{
        int n,s;
        printf("Enter a number:");
        scanf("%d",&n);
        s=sum(n);
        printf("Total sum=%d",s);
        getch();
}
int sum(int n)
{
        if(n<=0)
        return 0;
        else
        return (n+sum(n-1));
}
```

Scope of Variable (Local and Global Variable):

- Variable scope refers to the accessibility of a variable in a given program or function. For example, a variable may only be available within a specific function, or it may be available to the entire C program.

Local Variable:

Variables that are declared inside a function or block are called local variables. They can be used only by statements that are inside that function or block of code. Local variables are not known to functions outside their own.

Global Variable:

- Global variables are defined outside a function, usually on top of the program. Global variables hold their values throughout the lifetime of your program and they can be accessed inside any of the functions defined for the program.
- A global variable can be accessed by any function. That is, a global variable is available for use throughout your entire program after its declaration.

Example of Local Variable:

```
int main( )
{
    int a=50,b=60,s;
    s=a+b;
    printf("Sum=%d",s);
}
```

-Here a, b and s are local variables that means these variables are only accessible to main function.

- Scope of this these variable are with in main () function only. Therefor called local variables to main function.

Example of Global variable:

```
int s;
int main()
{
    int a=30,b=90;
    s=a+b;
    printf("Sum=%d",s);
}
```

Here, s is global variable that means we can access from anywhere through out the program.

Example of Global and Local Variables:

```
#include<stdio.h>
#include<conio.h>
int sum(int , int);
int s=100;--------------Here s is global variable.
int main()
{
      int a=90,b=10;--------Here a and b are local variable to main function only.
      s=sum(a,b);
      printf("%d\n",res);
      getch();
      return 0;
}
int sum(int x,int y)
{
      int res;   -------Here res is local variable to sum function only.
      res=(x+y+s);
      return res;
}
```

Life Time of Variable:

- The lifetime of a variable defines the duration for which the computer allocates memory for it (the duration between allocation and deallocation of memory).
- In C language, a variable can have automatic, static or dynamic lifetime.

Automatic − A variable with automatic lifetime are created. Every time, their declaration is encountered and destroyed. Also, their blocks are exited.

Static − A variable is created when the declaration is executed for the first time. It is destroyed when the execution stops/terminates.

Dynamic − The variables memory is allocated and deallocated through memory management functions.

Nested Function:

- Nested function means calling a function inside the definition of another function.
- When one or more functions are utilized under a particular function, it is known as nesting function in C Programming Language.
- We can not define a function within another function in C language(nested function is not supported by C language). we can only declare a function within another function in C(not define).

Example:

```
#include<stdio.h>
#include<conio.h>
void fun1();
void fun2();
int main()
{
	fun2();
	getch();
}
void fun1()
{
	printf("This is function one.");
}
void fun2()
{
	fun1();
	printf("\nThis is function two.");
}
```

Example:

```
#include<stdio.h>
#include<conio.h>
int fun1();
int fun2(int,int);
int main()
{
	int x;
	x=fun1();
	printf("Total Sum=%d",x);
	getch();
}
int fun1()
{
	fun2(100,50);

}
int fun2(int a,int b)
{
	return (a+b);
}
```

Example:

```
#include<stdio.h>
#include<conio.h>
int square(int);
int cube(int);
void main()
{
	int l,result;
	printf("Enter the value of l:");
	scanf("%d",&l);
	result=square(l);
	printf("Total result=%d",result);
	getch();
}
int square(int a)
{
	int x;
	x=(a*a)+cube(5);
	return x;
}
int cube(int m)
{
	return (m*m*m);
}
```

Passing Array as an Argument to the Function:

- Just like variables, array can also be passed to a function as an argument .
- While passing array from calling function to called function, We actually pass the base address of an array not whole array.
- When passing array as a parameter this does not create new array that means works on the existing array.

Example:

```
#include<stdio.h>
#include<conio.h>
void check(int a[]);
void main()
{
	int a[5]={5,10,15,20,25},i;
	check(a);
	printf("New array elements are:");
	for(i=0;i<5;i++)
	{
		printf("%d\t",a[i]);
	}
	getch();
}
void check(int x[])
{
	int i;
	for(i=0;i<5;i++)
	{
		x[i]=x[i]/5;
	}
}
```

Example:

```
#include<stdio.h>
#include<conio.h>
int check(int a[]);
void main()
{
	int a[5]={5,10,15,20,25},i,sum=0;
	check(a);
	for(i=0;i<5;i++)
	{
		sum=sum+a[i];
	}
	printf("Total sum=%d",sum);
	getch();
}
int check(int x[])
{
	int i;
	for(i=0;i<5;i++)
	{
		return x[i];
	}
}
```

Passing Strings as an Argument to the Function:

- Same like passing primitive data types (int, float, char, void) we can also pass the derived data type(array, string, pointer) as an argument to the function.
- In simple, Passing strings to the function means passing one dimensional character array to the function.

Example:

```
#include<stdio.h>
#include<conio.h>
void string(char* str);
void main()
{
    char str[40]="Kathmandu Nepal";
    string(str);
    getch();
}
void string(char* str)
{
    printf("Inut String is %s",str);
}
```

Example:

```
#include<stdio.h>
#include<conio.h>
int string(char str[]);
void main()
{
	char str[50];
	int l;
	printf("Enter the string: ");
	gets(str);
	l=string(str);
	printf("Length of string=%d",l);
	getch();
}
int string(char str[])
{
	int i,l=0;
	for(i=0;str[i]!='\0';i++)
	{
		l++;
	}
	return l;
}
```