Creating User Defined Functions:

Defining a function: Defining a function means writing the actual code of the function
The general format of defining a function is:

```
        Return-type Function-name (parameter list)
        {
            Function Body
        }
```

Following are the commonly used terms used in functions:

Function Name:	It is the name of functions chosen by the user
Parameters or Arguments:	These are the values passed to a function
Return value:	It is the value returned by a function
Return type:	This is the data type of the value returned by the function
Function body:	It is the actual statements that defines the function
Function Prototype:	It is the declaration of the function, written at the top of the program.

Example: Program to find largest from two numbers using a function.

```
#include <stdio.h>
int max(int x, int y);          // Function prototype
main()
{
        int a,b,lar;
        printf("\n Enter two numbers ");
        scanf("%d%d",&a,&b);
        lar=max(a,b);           // Calling the function max with actual parameters a,b
        printf("\n Largest is %d",lar);
        }
int max(int x, int y)           // Function header – Return type : int, formal parameters x,y
{
        int big;
        if(x>y)
          big=x;
        else
          big=y;
        return big;             // return value : big
}
```

Function Prototype: Like variables, all functions in a C program must be declared. Declaring a function is called function prototype. The general format of function prototype is:

Return-type Function-name(parameter list);

The function prototype is same as function header, except a semicolon at the last.

Function Calling and parameters: A function can be called by simply using the function name with a list of actual parameters, if any, enclosed in parentheses. When the compiler encounters a function call, the execution control is transferred to the function and statements in that function is executed, and returned back when the ***return*** statement is executed.
Parameters (also called arguments) are the values exchanged between calling function and called function. The parameters used in the function call is termed as *actual parameters* and the parameters used in the function header is termed as *formal parameters*. The actual parameters and formal parameters must match in type, in number and in order.
Depending up on the problems, a function may or may not use parameters and return values. If a function has no arguments, it means that it does not receive any data from calling function. Similarly, if a function has no return value, it means that, that function does not return a value to the calling function.

Call by Value and Call by Reference: A function can be called in two modes – Call by value and call by reference.
Call by Value: In Call by value, the value of actual parameters are copied to the corresponding formal parameters. Any changes made in the formal parameters does not affect the value of actual parameters. It uses value parameters.
Call by reference: In Call by reference, the changes made in formal parameters also automatically changes the value of actual parameters. It uses reference parameters.
Following example shows the difference between Call by value and Call by reference.

Call by Value	**Call by reference**

`#include <stdio.h>`
`void test(int a, int b);`
`main()`
`{   int m=10,n=20;`
`    test(m,n);`
`    printf("Values in main are %d %d",m,n);`
`}`
	`#include <stdio.h>`
	`void test(int *a, int *b);`
	`main()`
	`{   int m=10,n=20;`
	`    test(&m,&n);`
	`    printf("Values in main are %d %d",m,n);`
	`}`

void test(int a, int b)	void test(int *a, int *b)
{	{
printf(“Values in test are %d %d”,a,b);	printf(“Values in test are %d %d”,*a,*b);
a=100;	*a=100;
b=200;	*b=200;
printf(“Values in test are %d %d”,a,b);	printf(“Values in test are %d %d”,*a,*b);
}	}
Output:	Output:
Values in test are 10 20	Values in test are 10 20
Values in test are 100 200	Values in test are 100 200
Values in main are 10 20	Values in main are 100 200

Passing arrays to a function: Like the values of simple variables, it is also possible to pass values (elements) of an array to a function. Following example finds the sum of all elements of an array by passing it to a function – sum.

```
#include <stdio.h>
int sum(int a[], int n);
main()
{
    int num[5]={10,15,20,25,30},s;
    s=sum(num,5);
    printf(“\n Sum of elements is %d”,s);
}
int sum(int a[], int n)
{
  int i,s=0;
  for(i=0;i<n;i++)
    s=s+a[i];
  return(s);
}
```

We know that an array name itself is a pointer which holds the address of first element in that array. When passing an array to a function, we use the name of the array as the actual parameter, ie the address of the first location. So the same memory location is referred in the called function, and any changes made in the array will be reflected in the original array.

Scope of Variables: A variable can be declared inside a function or outside all functions. Scope of a variable means the part of a program in which the variable is available for use. Scope can be *Local* or *Global*.
Local variables are the variables declared within a function. It has Local scope. It means that such variables can be used within that function only. It has no meaning outside that function.

Global variables are the variables declared outside all functions. It has Global scope. Such variables can be used anywhere in the program. Global variables are also known as *external* variables.

Recursion: Recursion is the process of calling a function by itself. A recursive function should have a condition to terminate the execution, otherwise it will execute indefinitely. Recursive functions can be effectively used to solve problems where solution is expressed in terms of successively applying the same solution.
Using recursion in certain programs is simpler than its no-recursive version. A common example is to find factorial of a number.

Example: Find factorial of a number.

Using Recursion	**Without using recursion**
#include <stdio.h>	#include <stdio.h>
int fact(int n);	int fact(int n);
main()	main()
{ int n,f;	{ int n,f;
printf(“\n Enter number “);	printf(“\n Enter number “);
scanf(“%d”,&n);	scanf(“%d”,&n);
f=fact(n);	f=fact(n);
printf(“\n Factorial is %d”,f);	printf(“\n Factorial is %d”,f);
}	}
int fact(int n)	int fact(int n)
{	{
if (n==1)	int f=1,i;
return(1);	for(i=1;i<=n;i++)
else	f=f*i;
return(n*fact(n-1));	return(f);
}	}