FUNCTIONS

- Concept of function, using functions, call by value and call by reference mechanism to working with functions-example programs, passing arrays to functions, scope and extent, storage classes, recursion.

Type of argument passing

The different ways of passing parameters to the function are:

- Call by value
- Call by reference

Call by value:

In call by value, the values of actual parameters are copied into formal parameters, that is formal parameters contain only a copy of the actual parameters. So, even if the values of the formal parameters changes in the called function, the values of the actual parameters are not changed.

The concept of call by value can be explained by considering the following program.

Example:

```
#include<stdio.h>
void swap(int,int);
void main()
{
int a,b;
printf("enter values for a and b:");
scanf("%d %d",&a,&b);
printf("the values before swapping are a=%d b=%d \n",a,b);
swap(a,b);
printf("the values after swapping are a=%d b=%d \n",a,b);
getch();
}
```

```
void swap(int c,int d)
{
    int temp;
    temp=c;
    c=d;
    d=temp;
}
```

- Execution starts from function main() and we will read the values for variables a and b, assume we are reading 10 and 20 for a and b respectively.

- We will print the values before swapping it will print 10 and 20.
- The function swap() is called with actual parameters a and b whose values are 10 and 20.
- In the function header of function swap(), the formal parameters c and d receive the values 10 and 20.
- In the function swap(), the values of c and d are exchanged.
- But, the values of actual parameters a and b in function main() have not been exchanged.

Call by Address

In Call by Address, when a function is called, the addresses of actual parameters are sent. In the called function, the formal parameters should be declared as pointers with the same type as the actual parameters. The addresses of actual parameters are copied into formal parameters. Using these addresses the values of the actual parameters can be changed.

The concept of call by reference can be explained by considering the following program.

Example:

```
#include<stdio.h>
void swap(int *a,int *b);
void main()
{
        int a,b;
        printf("enter values for a and b:");
        scanf("%d %d",&a,&b);
        printf("the values before swapping are a=%d b=%d \n",a,b);
        swap(&a,&b);
        printf("the values after swapping are a=%d b=%d \n",a,b);
        getch();
}
void swap(int *c,int *d)
{
         int temp;
         temp=*c;
         *c=*d;
         *d=temp;
}
```

The sequences of operations that are carried out during the execution of the program are:

- Execution starts from function main() and assume that we are reading values 10 and 20 for variables a and b. We will print the values before swapping it will print 10 and 20.

- The function swap() is called by sending the addresses of a and b.
- In the function header of function swap, the formal parameters c and d declared using * operator holds the addresses of a and b.
- In the function swap(), using *c and *d we can access the values of a and b and they are exchanged.

Note:

Pointer: A pointer is a variable that is used to store the address of another variable.

Syntax: datatype *variablename;

Example: int *p;

```
#include<stdio.h>
void main()
{
        int a ,*p;
        p=&a;
}
```

In the above program **p** is a **pointer** variable, which is storing the address of variable **a**.

Differences between Call by Value and Call by reference

Call by Value	Call by Address
✓ When a function is called the values of variables are passed	✓ When a function is called the addresses of variables are passed
✓ The type of formal parameters should be same as type of actual parameters	✓ The type of formal parameters should be same as type of actual parameters, but they have to be declared as pointers.
✓ Formal parameters contains the values of actual parameters	✓ Formal parameters contain the addresses of actual parameters.
✓ Change of actual parameters in the function call will not affect the actual parameters in the calling function.	✓ The actual parameters are changed since the formal parameters indirectly manipulate the actual parameters
✓ Execution is slower since all the values have to be copied into formal parameters.	✓ Execution is faster since only addresses are copied.