Module 06

Pointers:

1. When variables are declared, memory location is allocated to each variable. The unary operator (&) gives the address of a variable called as the reference operator.
2. If 'var' is a variable in the program then &var will give its address in the memory.
3. Example:

```
#include <stdio.h>
void main(){
    int var = 5;
    printf("Value: %d\n", var);
    printf("Address: %u", &var);
}
```

Output:
Value: 5
Address: 2686778

4. A pointer is a variable which holds the address of another variable. The indirection or dereference operator (*) returns the value stored at particular address of a pointer.
5. Pointer variables are declared with same data type as ordinary variables.
6. Declaration of Pointer: datatype *pointer_variable_name;
   int *p;
7. Example:

```
int v=20,*p;
p=&v;
```

Here, v is a variable of type integer and p is a pointer variable. v stores value and p stores address. Let us assume that address of v is 1000.
Then, v will have value 20 and p will have 1000
And, *p gives the value stored at the address pointed by p i.e. *p=20.

8. Advantages:
   a. It is the only way to express some computations.
   b. It produces compact and efficient code.
   c. It provides a very powerful tool.
9. Disadvantages:
   a. Improper initialization of pointers may lead to disastrous situations.
   b. Using pointers sometimes be confusions.
10. Reference operator (&) and Dereference operator (*):
    a. & is called reference operator. It gives the address of a variable.
    b. Another operator that gets the value from the address is called a dereference operator (*).
11. Example:

```
#include <stdio.h>
void main() {
    int c=22, *pc;
    printf("Address of c:%u\n",&c);
    printf("Value of c:%d\n",c);
    pc=&c;
    printf("Content of pointer pc:%d\n",*pc);
    c=11;
    printf("Content of pointer pc:%d\n",*pc);
    *pc=2;
    printf("Value of c:%d\n",c);
}
```

Output:
Address of c: 2686784
Value of c: 22
Content of pointer pc: 22

Content of pointer pc: 11
Value of c: 2

Passing Arguments to a Function:

There are two ways in which arguments are passed to the function:

A. Call by value:
   1. While Passing Parameters using call by value, xerox copy of original parameter is created and passed to the called function.
   2. Any update made inside called function will not affect the original value of variable in calling function.
   3. That is, changes made to the parameter inside the function have no effect on the argument.
   4. As the scope is limited to only function, the code within a function cannot alter the arguments used to call the function.
   5. Example:

```
#include<stdio.h>
void swap(int, int);
void main()  {
    int a=5,b=7;
    printf("value before call a=%d and b=%d\n",a,b);
    swap(a,b);
    printf("value after call a=%d and b=%d\n",a,b);
}
void swap(int p, int q) {
    int temp;
    temp=p;
    p=q;
    q=temp;
    printf("value after swapping p=%d and q=%d\n",p,q);
}
```

   Output:

```
Value before call a=5 and b=7
Value after swapping p=7 and q=5
Value after a call a=5 and b=7
```

B. Call by reference:
   1. While passing parameter using call by address, the actual address of the variable is passed to the called function.
   2. This copies the address of an argument into the formal parameter.
   3. Any updates made inside the called function will modify the original copy since we are directly modifying the content of the exact memory location.
   4. To pass a value by reference, pointers are passed as arguments to the functions. So we need to declare the function parameters as pointer types.
   5. Example:

```
#include<stdio.h>
void swap(int *, int *);
void main() {
        int a=5, b=7;
        printf("value before call a=%d and b=%d\n",a,b);
        swap(&a,&b);
        printf("value after call a=%d and b=%d\n",a,b);
}
void swap(int *p,int *q) {
        int temp;
        temp=*p;
        *p=*q;
```

```
        *q=temp;
}
```

Output:

Value before call a=5 and b=7

Value after a call a=7 and b=5

Call by Value	Call by Reference
In this case the value of the parameters is passed to the called function.	In this case the address of the parameters is passed to the called function.
In this case the actual parameters are not accessible by the called function.	In this case, the called function can access the actual parameters.
This is implemented by using simple variable names.	This is implemented by the use of pointer variables.
Hence the actual parameters remain unchanged in case of the call by value.	Hence the actual parameters can be altered if required in case of the call by reference method