EST 102 CPC

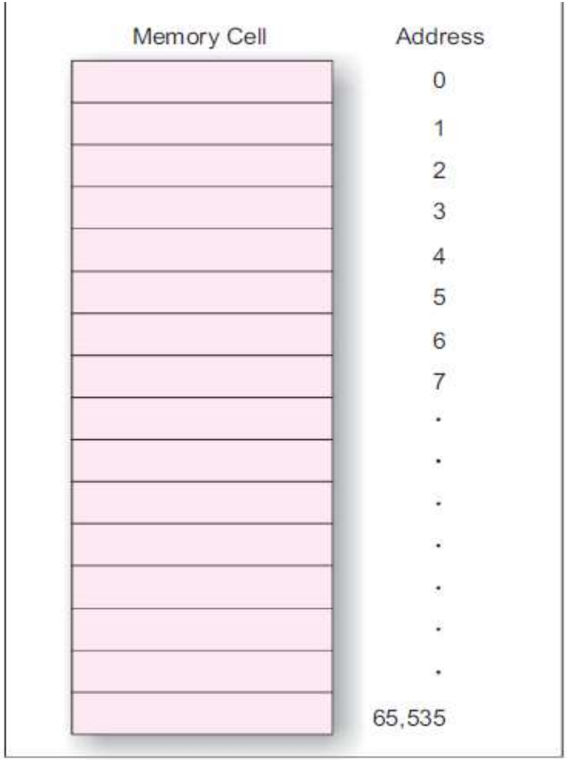
MODULE-5
POINTERS AND FILES

POINTERS:Basics

- A pointer is a **derived data type** in C.
- It is built from one of the fundamental data types available in C.
- Pointers **contain memory addresses are their values**.
- Pointers can be **used to access and manipulate data stored in the memory**.
    - Because, memory addresses are the locations in the computer memory where program instructions and data are stored.
- Pointers are one of the most distinct and exciting features of C language.

Understanding pointers:

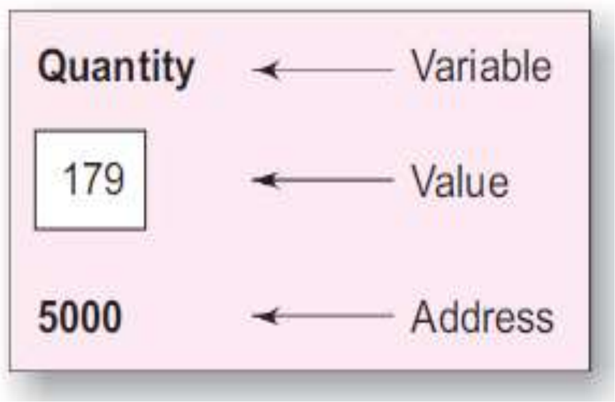
- The computer's memory is a sequential collection of storage cells.
- Associated with each cell, there is an address.
- Typically, addresses are numbered consecutively starting from zero.
- Whenever we declare a variable,the system allocates an appropriate location to hold the value of the variable, somewhere in the memory.
- Since every cell has a unique address number, this location have its own address number.



<u>Eg:</u> Consider the following statement;

int quantity = 179;

- This statement instructs the system to find a location for the integer variable *quantity* and puts the value 179 in that location.
- Assuming that the system has chosen the address location 5000 for *quantity,* we may represent it as;



- During program execution,the system always associates the name *quantity* with address 5000.
- So, to access the value 179, we may have to use either the name *quantity* or the address 5000.

- Memory address, being simply numbers, can be assigned to some variables, that can be stored in memory, like any other variable.
- **Pointer variables**:

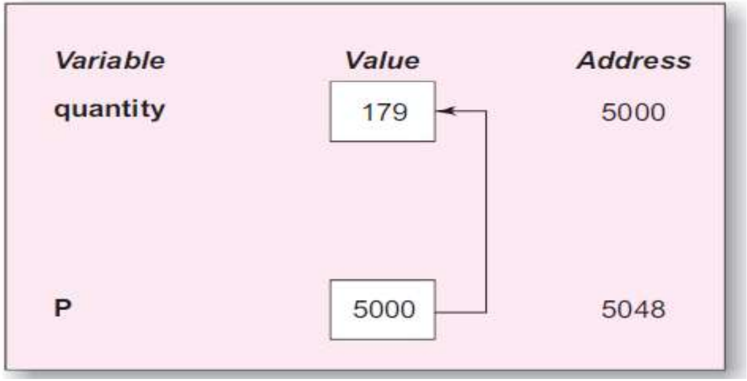
  Variables that hold memory addresses.

  Or

  Variable that contains an address, which is a location of another variable in memory.

- Pointer, being a variable, its value is also stored in the memory in another location.

- Suppose we assign the address of *quantity* to a variable *p*, the link between variables *p* and *quantity* can be visualized as below;



Variable	Value	Address
quantity	179	5000
P	5000	5048

- 5048 is the address of *p.*
- Here, the value of variable *p* is the address of the variable *quantity*.
- So, value of the variable *quantity* can be accessed by using the value of *p.* So, we can say that, the variable *p* points to the variable *quantity*.

Accessing the address of a variable:

- **To access the address of a variable in the memory, we can use the operator &.**
- The operator & immediately preceding a variable returns the address of the variable associated with it.
- ‘&’ can be remembered as ‘address of’.
- Eg:

```
p = &quantity;
```

The above statement would assign the address 5000(the location of *quantity*) to the variable *p.*

- The & operator can be used only with a simple variable or an array element.

Advantages of using pointers:

1. Permits references to functions and thereby **facilitate passing of functions as arguments to other functions.**
2. Helps to **save data storage space in memory** if pointer arrays are used to character strings.
3. **Supports dynamic memory management.**
4. **Reduces the length and complexity of programs**.
5. **Increase the execution speed** and thus reduce the program execution time.
6. Efficient in handling arrays and data tables.
7. Can be used to return multiple values from a function via function arguments.

Declaring pointers:

- In C, every variable must be declared for its type.
- Since pointer variables contain addresses that belong to a seperate data type, before using them, we have to declare them.
- The general form of declaring a pointer variable is as follows;

data_type *pt_name;

- This tells the compiler 3 things about the variable *pt_name*.
  - The asterisk(*) tells that the variable *pt_name* is a pointer variable.
  - *pt_name* needs a memory location.
  - *pt_name* points to a variable of type data_type.

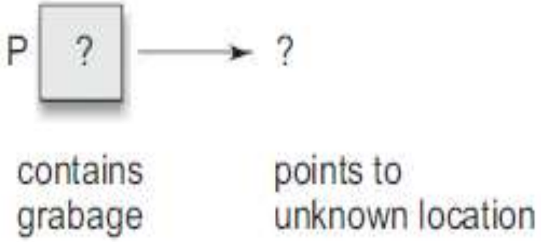
- Eg:1 int *p;

  Declares the variable *p* as a pointer variable that points to an integer data_type.

- Eg:2 float *x;

  Declares *x* as a pointer to a floating-point variable.

- The declarations cause the compiler to allocate memory locations for the pointer variables *p* and *x*.
- Since the memory locations have not been assigned any values, these locations may contain some unknown values in them and therefore they point to unknown locations.



Initialization of pointers:

- **Initialization**: The process of assigning the address of a variable to a pointer variable.
- All uninitialized pointers will have some unknown values that will be interpreted as memory addresses, which may not be valid or they may even point to some values that are wrong.
- Compilers do not detect these errors. So, programs with uninitialized pointers will produce erroneous results.
- Therefore, initializing pointers before using them is very important.
- To initialize a pointer variable which has been declared, we can use the assignment operator.
- Eg:

```
int quantity;

int *p;                    //declaration

p = &quantity;  //initialization
```

We can also combine initialization with declaration as follows;

```
int *p = &quantity;
```

- The pointer variable must always point to the corresponding type of data.

Eg:

```
float a,b;

int x, *p;

p = &a;                /*wrong*/

b = *p;
```

- This will produce erroneous results since we are trying to assign the address of a float variable to an integer pointer.
- Wrong pointer assignments must be avoided.

- It is also possible to combine the declaration of data variable, the declaration of pointer variable and the initialization of the pointer variable in one step.
  - Eg: `int x, *p = &x;`

    This declares *x* as an integer variable and *p* as a pointer variable and then initializes *p* to the address of *x*.

NULL pointer:

- A NULL pointer is a **pointer which points to nothing.**
- We can define a pointer variable with an initial value of NULL or 0 (zero) as follows;

int *p = NULL;

Or

int *p = 0;

- Some uses of the null pointer are:

a) To initialize a pointer variable when that pointer variable isn’t assigned any valid memory address yet.

b) To pass a null pointer to a function argument when we don’t want to pass any valid memory address.

c) To check for null pointer before accessing any pointer variable so that we can perform error handling in pointer related code.

- With the exception of NULL and 0, no other constant value can be assigned to a pointer variable.
  - Eg: int *p = 5000;   /*Wrong*/

Accessing a variable through its pointer:

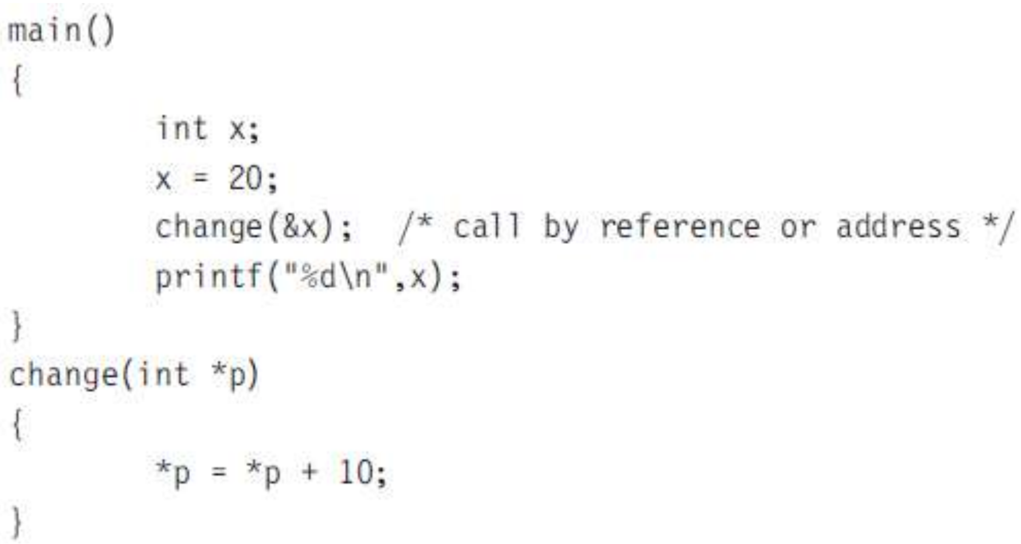
- Once a pointer has been assigned the address of a variable, we must know how **to access the value of the variable using pointer**.
- This is done by using a unary operator ***** (asterisk), usually known as the **indirection operator** or **dereferencing operator**.
- ‘*’ can be remembered as ‘value at address’.
- **Eg:**

```
int quantity, *p, n;  /*Declaring integer variables quantity and n and pointer variable p*/

quantity = 179;              /*Value 179 is assigned to the variable quantity*/

p = &quantity;               /*Address of quantity is assigned to the pointer variable p*/

n = *p;                                        /*Value of the variable quantity is assigned to the integer
variable n. ie,179*/
```

Pass by reference:

- Also known as “**Call by reference**” or “**Call by pointers**”.
- Pass by reference is a **parameter-passing mechanism** in which **the address of variable is passed as an argument to a function.**
- When we pass addresses to a function**, the parameter receiving the addresses should be pointers.**
- This process of **calling a function using pointers** is known as Pass by reference.
- The function which is called by ‘reference’ can change the value of the variable used in the call.

Eg: Consider the code below;



```
main()
{
        int x;
        x = 20;
        change(&x);  /* call by reference or address */
        printf("%d\n",x);
}
change(int *p)
{
        *p = *p + 10;
}
```

- When the function **change( )** is called, the address of the variable **x**, not the value, is passed into the function **change( )**.
- Inside **change( )**, the variable **p** is declared as pointer and therefore **p** is the address of the variable **x**.
- The statement,
  ***p = *p + 10;**
  means ‘add 10 to the value stored at the address p’.
- Since **p** represents the address of **x**, the value of **x** is changed from 20 to 30.
- Therefore the output will be 30.