UNIT VI
POINTERS AND IT'S FLEXIBILITY
PART 1

POINTER NOTATION

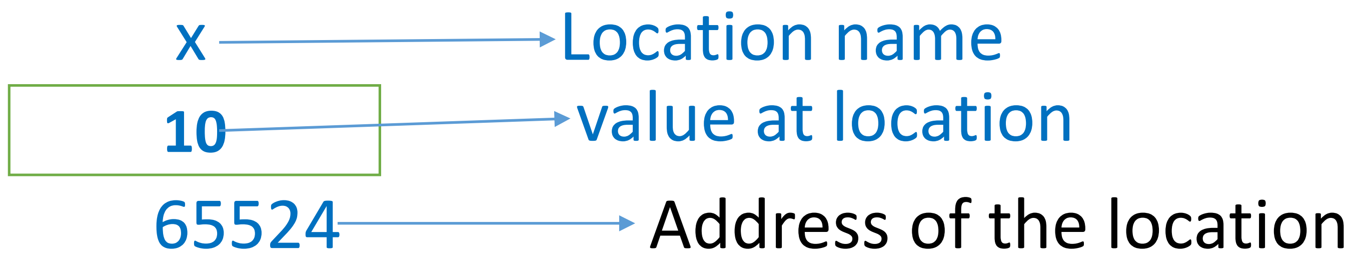
Consider the following declaration:

```
int x=10;
```

This declaration instruct the C compiler to:

1. Reserve a space in memory to hold the integer(**int**) value
2. Associate the **‘x’** with this memory location
3. Store the value **10** at this location

The above declaration can be represented in memory as:



WHAT IS POINTER?

- It is a special data type(feature) that allows us to refer to both its address and value
- It is a variable in which its value is the address of another variable
- There are two types of operators used with pointers namely:
  1. **'*'** taken as '**value at the address**'. Also called as **'indirection operator'/ 'Dereferencing operator'**
  2. **'&'**taken as '**address of**'

General form of Pointer declaration:

data-type *pt_name;

Where,

- 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

For example:

```
        int *p          /*integer pointer*/
        float *x        /*float pointer*/
        char *c         /*character pointer*/
```

Pointer variables capable of holding the addresses

- The type **int** refers to the data type of the variable being pointed to by **p** and not the type of the value of the pointer
- In the above example, the declaration **float *x** doesn't mean that **x** is going to contain a floating-point value. What it means is that, **x** is going to contain the address of a floating-point value
- **int *p** means that it is going to contain the address of an integer value
- **char *c** means that it is going to contain the address of a character value

- Compiler allocates the locations for pointer variable

- Since no value has been assigned, this location may contain some unknown values and therefore they point to unknown locations

- Programmer uses following styles to declare pointer

```
int* p;
int *p;
int * p;
```

INITIALIZATION OF POINTER VARIABLES

- The assignment operator can be used to initialize a pointer variable once it is being declared

For example:

```
int quantity;
int *p;              /*declaration*/
p=&quantity;         /*Initialization*/
```

Similarly,

```
int x,*p=&x                /*three in one*/
```

is a valid declaration and initialization

Note: The target variable x needs to be declared first

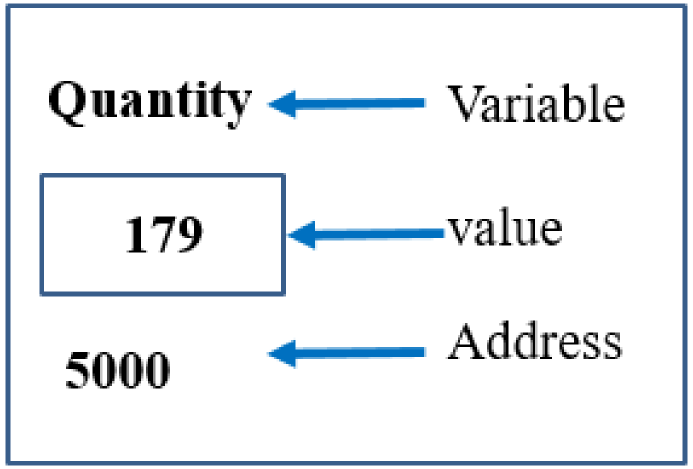
INITIALIZATION OF POINTER VARIABLES

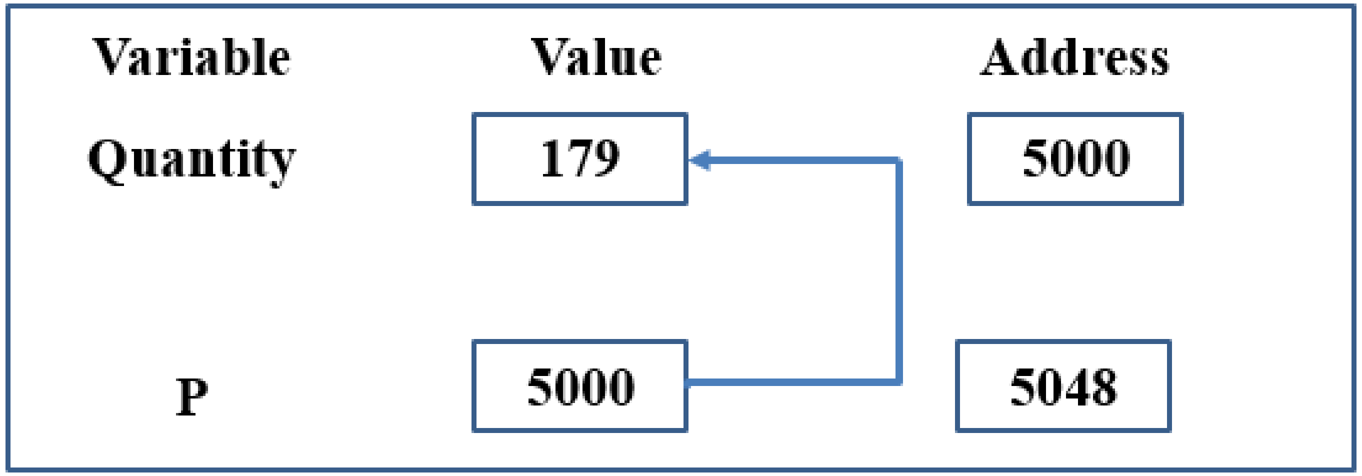
int *p=&x, x; is not a valid declaration and initialization

- We can also define a pointer variable with an initial value of NULL or 0(zero),that is,

```
int *p=NULL;
int *p=0;
```

- Thus, ***POINTER VARIABLE***s are variables that contains the address of another variable in memory
- The pointer's values are also stored in memory in another location





BENEFITS OF POINTER

Some of the benefits:

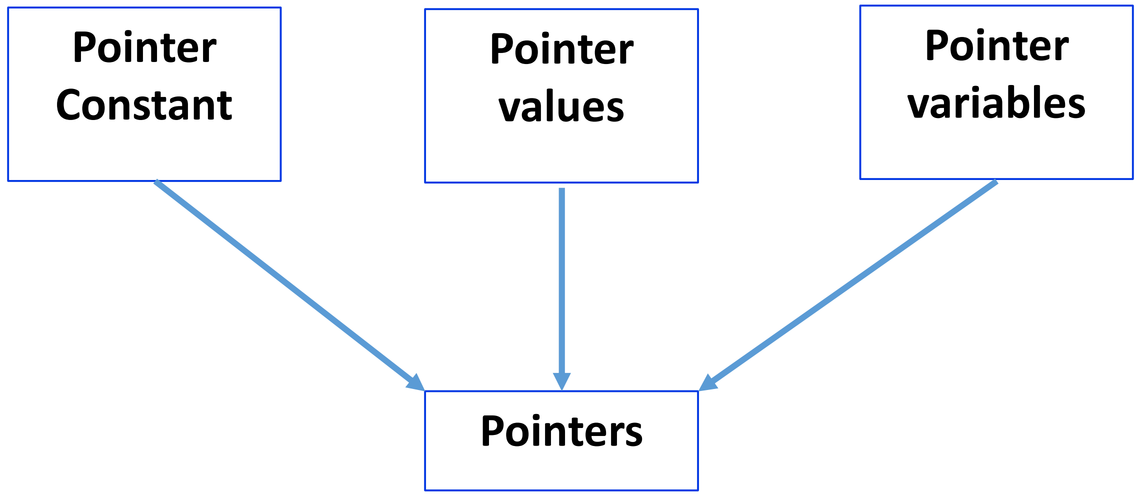
- Pointers are efficient in handling arrays and table data
- Can be used to return multiple values from a function via function argument
- Can permit references to functions
- Pointers reduces length & complexity of programs
- Faster execution time (Direct memory access)
- Pointers provide efficient tool for manipulating dynamic data structures such as structures, linked lists, queues, stacks and trees
- Support dynamic memory management

Accessing the Address

- Memory addresses are system dependent
- ***&*** Operator available in C helps us to determine the address of variable
- Recollect the usage of & operator in *scanf()*
- & operator can be remembered as ‘address of’
- Example *p = &quantity*
- & can be used only with simple variable or array element.
- The following are illegal usage of address operator
  - &123 (pointing at constant)
  - int x[10];
    &x (pointing at array name)
  - &(e+y) (pointing at expression)

Underlying Concepts

- Pointers are built on three underlying concepts



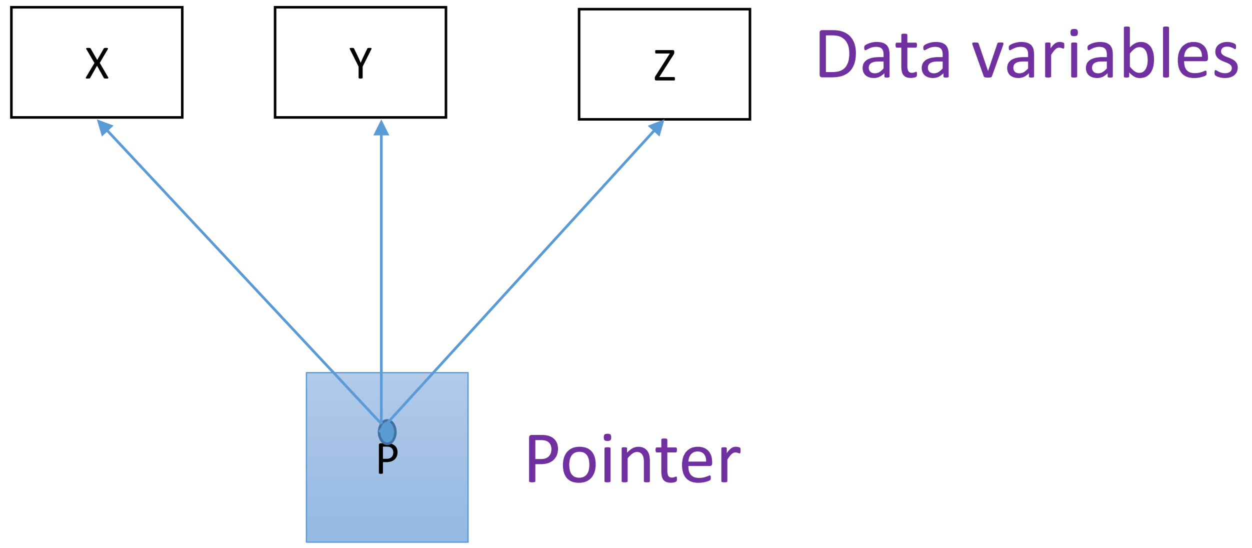
- Memory address within computer (pointer constant)
- Values obtained using address operator (&) (pointer value)
- Variable that store pointer value (pointer variable)

Pointer Flexibility

- Pointers are flexible
- We can make the pointers to point to different data variables in different statements

Example:

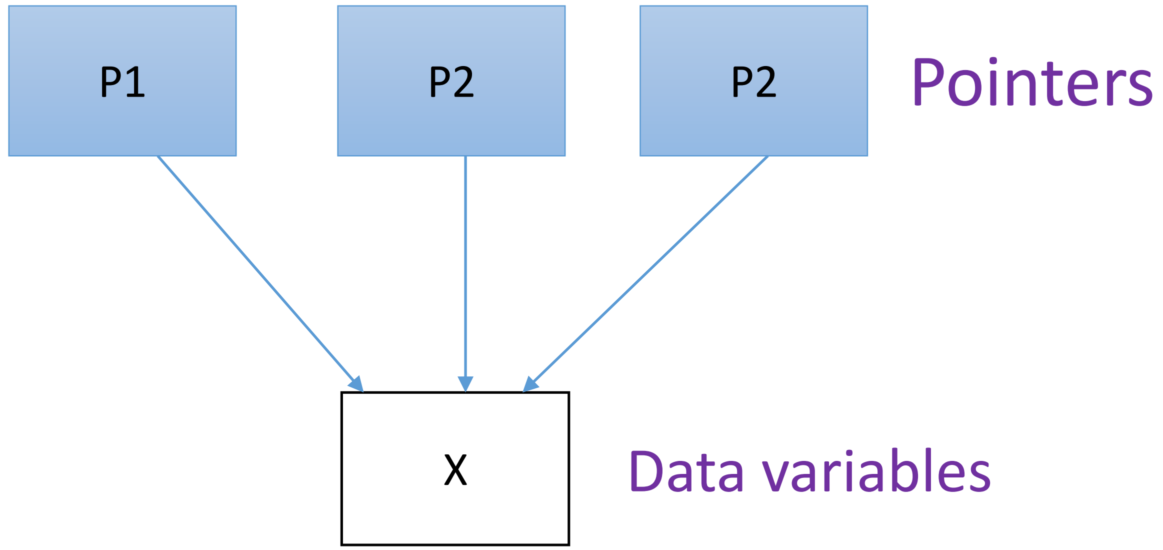
```
int x,y,z,*p;
p=&x;
p=&y;
p=&z;
```



- We can also use different pointers to point to the same data variable

Example:

```
    int x;
    int *p1=&x;
    int *p2=&x;
    int *p3=&x;
```



- No other constant can be assigned to pointer variable other than NULL or 0

Accessing the Variable’s value through Pointer

- The values of the variable can be accessed using the unary * operator usually known as *indirection operator*
- It is also know as *dereferencing operator*
- General form to access the values through pointers

    *variable1 = *pointer_name;*
- When the operator * is placed before a pointer variable in an expression, the pointer returns the value of variable of which pointer value is the address

Accessing the Variable’s value through Pointer

Example

```
    int quantity, *p, n;
    quantity=179;
    p=&quantity;
    n= *p; or n=*&quantity → n=quantity
```

* can be remembered as ‘value at address’