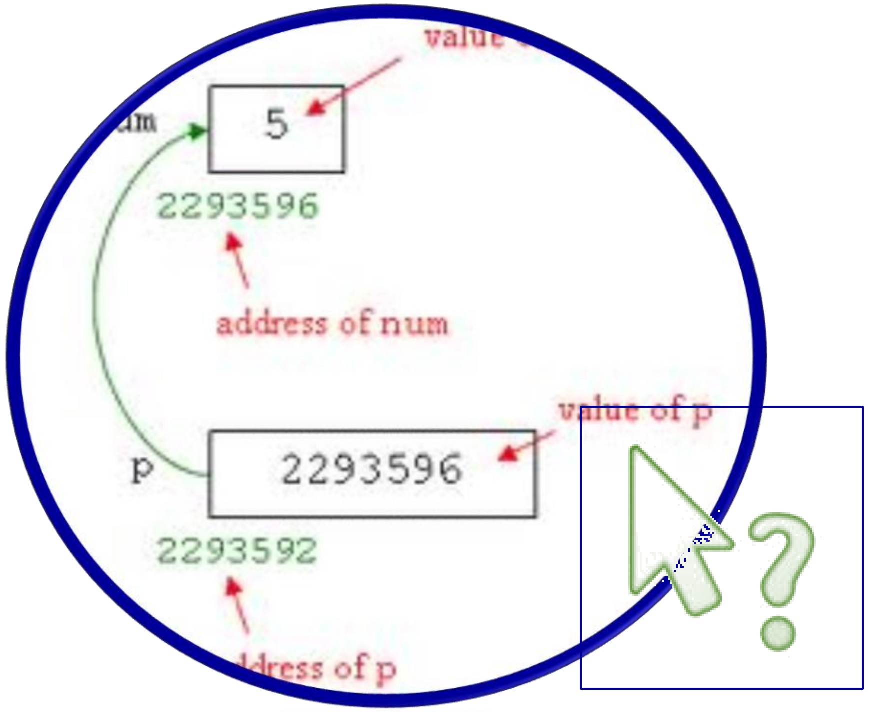


P o i n t e r s L 2 4

Objectives

- To learn and appreciate the following concepts:
  - Pointers – declaration and initialization
  - To access a variable through its pointer

Session outcome

- At the end of session one will be able to:

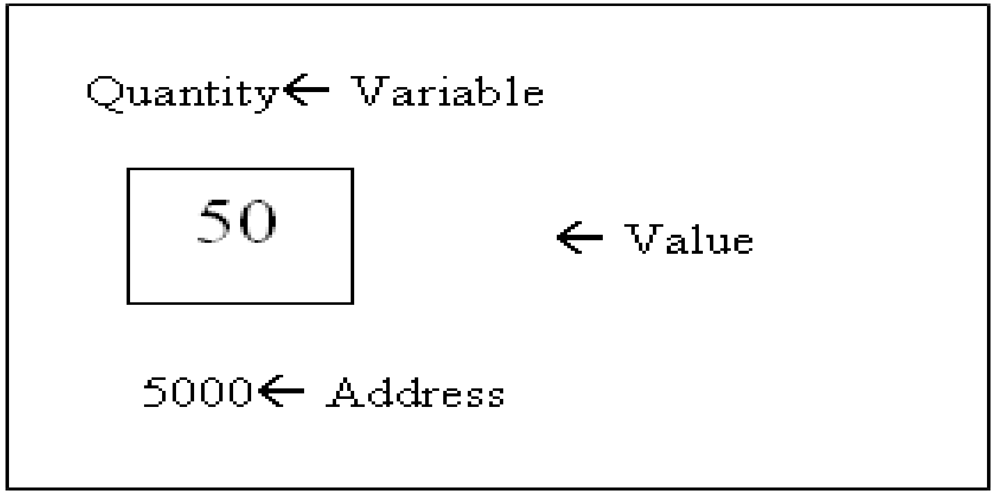
  Understand the overall ideology of pointers

Pointers - Concept

- Consider the following statement

```
int Quantity =50;
```

- Compiler will allocate a memory location for Quantity and places the value in that location. Suppose the address of that location is 5000, then



Pointers - Concept

- During Execution of the program, the system always associates the name **quantity** with the address 5000.

- We may have access to the value 50 by using either the name of the variable **quantity** or the **address 5000.**

- **Since memory** addresses are simply numbers, they can be assigned to some variables which can be stored in memory, like any other variable.

Pointer

A memory location or a variable which stores the address of another variable in memory.

Commonly used in C than in many other languages (such as BASIC, Pascal, and certainly Java, which has no pointers).

Declaring and initializing pointers

Syntax:

data_type * pt_name;

This tells the compiler 3 things about the **pt_name:**

- The **asterisk**(*) tells the variable **pt_name** is a **pointer variable**.
- **pt_name** needs a **memory location**.
- **pt_name** points to a variable of type **data_ type**

Accessing the address of a variable

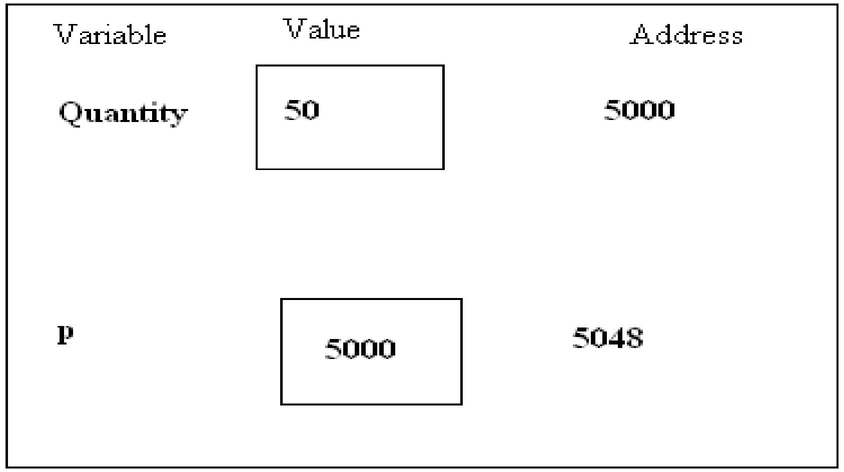
int Quantity=50 ;

- To assign the address 5000 (the location of quantity) to a variable **p**, we can write:

int *p = &Quantity ;

Such variables that hold memory addresses are called

Pointer Variables.



Pointers Concept

The Address-of Operator ‘&’

- To find the address occupied by a variable

Program to illustrate the address of operator

```
#include <stdio.h>
int main() {

int var1 = 11;
   int var2 = 22;
   int var3 = 33;

//print the addresses of these variables
    printf("%x",&var1);
    printf("%x",&var2);
    printf("%x",&var3);

  return 0;
}
```

Output:

```
0x29feec
0x29fee8
0x29fee4
```

Declaring and initializing pointers

Example:

int *p; **//**declares a variable **p** as a **pointer variable** that points to an integer data type.

float *x; **//**declares **x** as a **pointer** to floating point variable.

- Once a pointer variable has been declared, it can be made to point to a variable using an assignment statement :

int quantity = 10;

p = &quantity; // p now contains the address of **quantity**. This is known as **pointer initialization**.

Points to be taken care for pointer usage …

- Before a pointer is initialized, it should not be used.
- We must ensure that the pointer variables always point to the corresponding type of data.
- Assigning an **absolute address** to a pointer variable is prohibited. **i.e p=5000**
- A pointer variable can be initialized in its declaration itself.

Example:

```
int x, *p=&x;   //declares  x as an integer
                            variable and then initializes
                            p to the address of x.
```

Points to be taken care for pointer usage …

The statement

int *p = & x, x;   not valid.

i.e target variable   **‘x’** must be declared first.

Accessing variable through a pointer

- A **variable's value** can be accessed by its pointer using **unary operator** *(**asterisk**) known as **indirection operator**.

Consider the following statements:

```
int quantity, *p, n;     // 2 int variables & 1 integer pointer
quantity =50;            // assigns value 50 to quantity
p=&quantity;             // assigns the address of quantity to p
n=*p;                    // contains the indirection operator  *
```

*** Operator - value at address operator**

Example – Accessing variable through a pointer

```
#include <stdio.h>

int main() {
        int var1 = 11;      //two integer variables
  int var2 = 22;
  int *ptr;             //pointer to integer

  ptr = &var1;          //pointer points to var1
  printf("%d",*ptr);    //print contents of pointer (11)

   ptr = &var2;         //pointer points to var2
  printf("%d",*ptr);    //print contents of pointer (22)

   return 0;
}
```

Output :

```
11
22
```

Example - Accessing via pointers.

```
#include <stdio.h>
int main(){

  int var1, var2;          //two integer variables
  int *ptr;          //pointer to integers

   ptr = &var1;            //set pointer to address of var1
  *ptr = 37;               //same as var1=37 ( Dereferencing)
  var2 = *ptr;             //same as var2=var1
  printf("%d", var2);    //verify var2 is 37

   return 0;
}
```

Reference and dereference operators

- **&** is the ‘**reference**’ operator and can be read as "**address of**"
- ***** is the ‘**dereference**’ operator and can be read as “**value at address**” or "**value pointed by**"

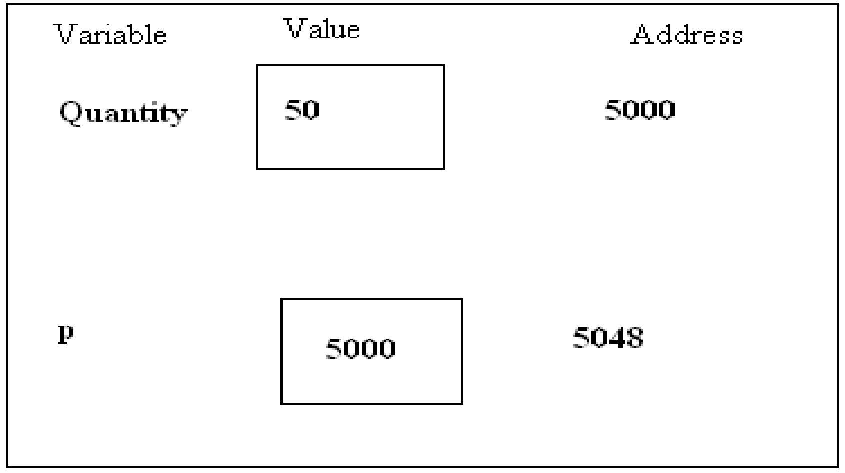
Example- understanding pointers

```
#include <iostream>
using namespace std;
int main() {
int  firstvalue = 5, secondvalue = 15;
 int * p1, * p2;
 p1 = &firstvalue; // p1 = address of firstvalue
 p2 = &secondvalue; // p2 = address of secondvalue
 *p1 = 10;    // value pointed by p1 = 10
 *p2 = *p1;   // value pointed by p2 = value pointed by p1
  p1 = p2;              // p1 = p2 (value of pointer is copied)
 *p1 = 20;    // value pointed by p1 = 20
cout << "firstvalue is " << firstvalue <<endl;
cout << "secondvalue is " << secondvalue;
   return 0;
}
```

Output :

```
firstvalue is  10
secondvalue is  20
```

Summary till now …



```
int Quantity=50;        //defines variable Quantity of type int
int* p;       //defines p as a pointer to int

p = &Quantity;   //assigns address of variable Quantity to  pointer p
```

Now…

```
  *p = 3;     //assigns 3 to Quantity
```

Summary

- Pointer concept
- Reference operator **&**
- Dereference operator *****