**B.Sc. SEM-VI/Electronics/Paper-I/Unit-III**

**Pointers**

**Introduction:-**A pointer is a special variable that stores/points the address of another variable. The contents of the C pointer always be a whole number i.e. address. Two pointers can be subtracted, but pointer addition, multiplication, divisions are not allowed. The size of any pointer is 2 byte (for 16 bit compiler).

Let’s try to understand the concept.

Address 1001 2047

50

1001

Value

Variable Var \*ptr

(Normal variable) (Pointer variable)

As shown in the above diagram:

1) A normal variable ‘var’ has a memory address of 1001 and holds a value 50

2) A pointer variable ‘ptr’ has its own address 2047 but stores 1001 which is

the address of the variable ‘var’.

**Advantages of pointer in C:-**

1. Pointers provide direct access to memory.
2. Pointers provide a way to return more than one value to the functions.
3. Pointers reduce the storage space and complexity of the program.
4. Pointers reduce the execution time of the program.
5. Pointers provide an alternate way to access array elements.
6. Addresses of objects can be extracted using pointers
7. Pointers helps us to built complex data structures like linked list, stack, queues, trees, graphs etc.

**Drawbacks of pointer in C:-**

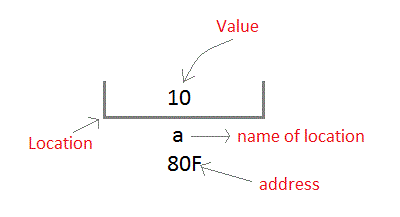
1. Pointers are slower than normal variables
2. If pointers are updated with incorrect values, it might lead to memory corruption.
3. Uninitialized pointers might cause segmentation fault.
4. Pointers bugs are difficult to debug. Its programmer’s responsibility to use pointers effectively and correctly.

**Concept of Pointers:-**

Whenever a **variable** is declared in a program, system allocates a location i.e an address to that variable in the memory, to hold the assigned value. This location has its own address number, which we just saw above.

Let us assume that system has allocated memory location 80F for a variable a.

int a = 10;



We can access the value 10 either by using the variable name a or by using its address 80F.

The question is how we can access a variable using it's address? Since the memory addresses are also just numbers, they can also be assigned to some other variable. The variables which are used to hold memory addresses are called **Pointer variables**.

A **pointer** variable is therefore nothing but a variable which holds an address of some other variable. And the value of a **pointer variable** gets stored in another memory location.

**Address operator:-** The actual location of a variable in the memory location can be determine with the help of the operator &(ampersand) available in C. We have already use this address operator in the scanf function. The operator & immediately preceding a variable return the address of the variable associated with it. For example

p= &quantity;

Will assign the address of the location of quantity (e.g. if address of quantity is 5000) then after executing the above statement, p= 5000

The address operator (&) can be used only with a simple variable or an array element. It cannot used with constant, array names and expressions. For example following are illegal use of address operator:

1. &165 (address of constant)

2. &x (address of array name)

3. &(x+y) (address of expression)

**Declaring and initializing pointers**

**Declaration of C Pointer variable:-**

General syntax for declaration of pointer variable is

Data type \*pointer name;

Where pointer name point to a variable of data type

and Star \* indicates that the variable pointer name is a pointer variable.

Note that the data type of pointer must be same as the data type of the variable to which the pointer variable is pointing.

For example,

int \*p; // declares p as a pointer variable to an integer data type//

float \*p; // declares p as a pointer variable to a float data type//

char \*p; // declares p as a pointer variable to a char data type//

**Initialization of C Pointer variable:-**

Pointer initialization is the process of assigning address of a variable to a pointer variable using an assignment statement such as

p = &quantity;

This means that, p now contains the address of quantity. This is known as pointer initialization. Before a pointer is initialized, it should not be used.

Pointer variable can only contain address of a variable of the same data type.

For example,

main( )

{

int a = 10;

int \*ptr; // pointer declaration //

ptr = &a; // pointer initialization//

}

Note:- A pointer variable can be initialized in its declaration itself. For example

int x, \*p = &x

It declares that x as an integer variable and p as a pointer variable and then initializes p to the address of x. and also note that the target variable x is declared first. e.g. The following statement

int \*p = &x, x;

is not valid.

**Indirection operator (\*)** :- In C, we can access the value of the variable stored at the particular address of memory location through its pointer. This is done by using another unary operator \*(asterisk), usually known as indirection operator. For example consider the following statements:

int quantity, \*p, n; /declares quantity and n as int variable and p as a pointer variable/

quantity = 179; / assigning the value 179 to quantity/

p = &quantity; / initializing of pointer variable with the address of quantity/

n = \*p; / accessing the value of quantity through its pointer/

The fourth line contains the indirection operator \* before a pointer variable which returns the value of the variable quantity to n variable. Thus the value of n will be 179.

Problem: Write a program to illustrate the use of indirection operator

/\* Accessing variable using pointer \*/

main( )

{

int x,y;

int \*ptr;

x = 10;

ptr = &x;

y = \*ptr;

printf(“Value of x is %d\n”,x);

printf(“%d is stored at addr %u\n”, x,&x);

printf(“%d is stored at addr %u\n”, \*&x,&x);

printf(“%d is stored at addr %u\n”, \*ptr,ptr);

printf(“%d is stored at addr %u\n”, y,&\*ptr);

printf(“%d is stored at addr %u\n”, ptr,&ptr);

printf(“%d is stored at addr %u\n”, y,&y);

\*p = 25;

printf(“Now x = %d\n”,x);

}

Output

Value of x is 10

10 is stored at addr 4104

10 is stored at addr 4104

10 is stored at addr 4104

10 is stored at addr 4104

4140 is stored at addr 4106

10 is stored at addr 4108

Now x = 25