C Programming

Trainer : Nisha Dingare

Email : nisha.dingare@sunbeaminfo.com

Pointers – Introduction :

- Every variable is a memory location and every memory location has its address defined which can be accessed using ampersand (&) operator, which denotes an address in memory.
- A pointer is a variable whose value is the address of another variable, i.e., direct address of the memory location.
- Like any variable or constant, you must declare a pointer before using it to store any variable address.
- Internally it is an unsigned integer.
- It is a derived datatype(based on the primitive datatype).

Uses of pointer :

- To return multiple values from functions.
- To pass arguments by reference.
- For accessing array elements.
- Dynamic memory allocation.
- To implement data structures.
- To communicate information about memory.

Pointer syntax :

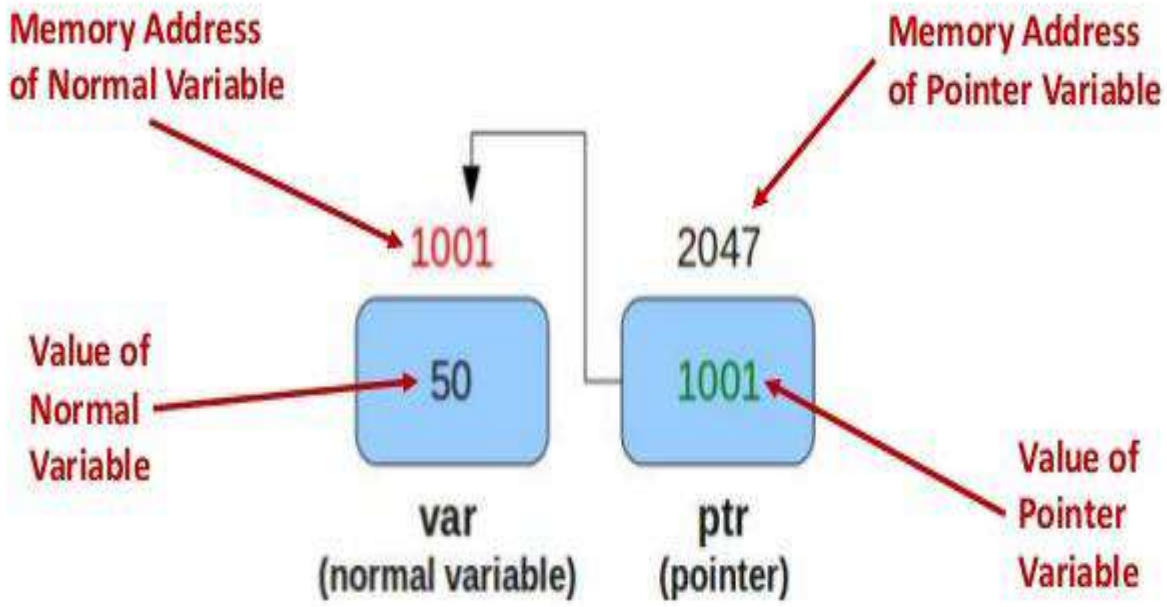
- Declaration :
  - Int *ptr;
- Initialization :
  - Ptr = #
- Dereferencing :
  - Printf(“%d”, *ptr);

- Operator & and *
- & symbol is called as direction / reference / address of operator and is used to get address of any variable.

- * symbol is called as indirection / deference / value at operator and is used to get value at the address stored in a pointer.

Example of pointer :

```
void main()
{
        int var = 50;
        int *ptr;               /*Declaration*/
        ptr = &var;             /*Initialization*/
        printf("%d",*ptr);      /*(50)accessing value*/
        *ptr = 20;              /*setting value*/
        printf("%d",var);       /*(20)*/
}
```



Pointer to functions as arguments :

Call By Value :

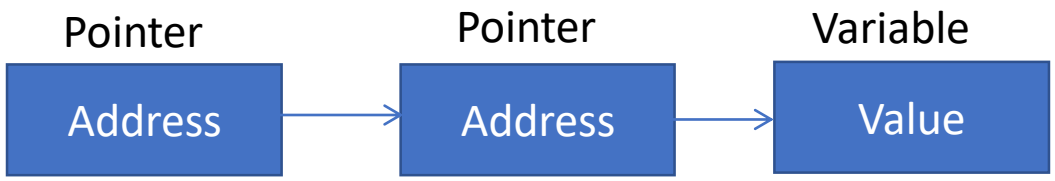
- Formal argument is of same type as of actual argument.
- Actual argument is copied into formal argument.
- Any change in formal argument does not reflect in actual argument.
- Creating copy of argument need more space as well as time (for bigger types).

Call By Address :

- Formal argument is of pointer type (of actual argument type).
- Address of actual argument is collected in formal argument.
- Actual argument can be modified using formal argument.
- To collect address only need pointer. Pointer size is same irrespective of data type.

Pointer to pointer :

- A pointer to a pointer is a form of multiple indirection, or a chain of pointers. Normally, a pointer contains the address of a variable. When we define a pointer to a pointer, the first pointer contains the address of the second pointer, which points to the location that contains the actual value as shown below.



- A pointer to a pointer must be declared by placing an additional asterisk in front of its name. For example, the following declaration declares a pointer to a pointer of type int .

```
int **ptr1;
```

- When a target value is indirectly pointed to by a pointer to a pointer, accessing that value requires that the asterisk operator be applied twice.

Pointer Scale Factor and Pointer arithmetic :

Scale Factor :

- Size of data type of pointer is known as Scale factor.
- Scale factor defines number of bytes to be read/written while dereferencing the pointer.
- Scale factor of different pointers
  - Pointer to primitive types: char*, short*, int*, long*, float*, double*
  - Pointer to pointer: char**, short**, int**, long**, float**, double**, void**
  - Pointer to struct/union.
  - Pointer to enum.

Pointer Arithmetic :

- Scale factor plays significant role in pointer arithmetic.
- n locations ahead from current location
  - ptr+ n = ptr + n * scale factor of ptr
- n locations behind from current location
  - ptr-n = ptr-n * scale factor of ptr
- number of locations in between
  - ptr1 –ptr2 = (ptr1 –ptr2) / scale factor of ptr1

Pointer Arithmetic :

- When pointer is incremented or decremented by 1, it changes by the scale factor.
- When integer ‘n’ is added or subtracted from a pointer, it changes by n * scale factor.
- Multiplication or division of any integer with pointer is not allowed.
- Addition, multiplication and division of two pointers is not allowed.
- Subtraction of two pointers gives number of locations in between. It is useful in arrays.

Pointer and ++

- A pointer in c is an address, which is a numeric value.
- Therefore, you can perform arithmetic operations on a pointer just as you can on a numeric value. There are four arithmetic operators that can be used on pointers: ++, --, +, and -.
- Incrementing a pointer Variable Depends Upon data type of the Pointer variable.
- new value = current address + i * sizeof(data type)

- Three Rules should be used to increment/decrement pointer: •
  - 1. Address+1=Address •
  - 2. Address++ = Address •
  - 3. ++Address = Address

Points to remember :

- Size of the pointer is always same irrespective of its datatype. (as it stores the address).
- When we use sizeof() operator for pointer variable, it gives output as 8 bytes in case of 64 bit compiler and 4bytes in case of 32 bit compiler.

Reason : Address through system is always given in unsigned intformat. Hence size of any pointer will be equals to unsigned intof respective compiler.

- Void Pointer : void * is a pointer which is not associated with any data types. It is a generic pointer.
- void pointer is not aware of scale factor i.e. how many bytes to be dereferred from given base address. Hence if we want to receive value at void pointer it is necessary always typecast prior its to use.

Thank You !