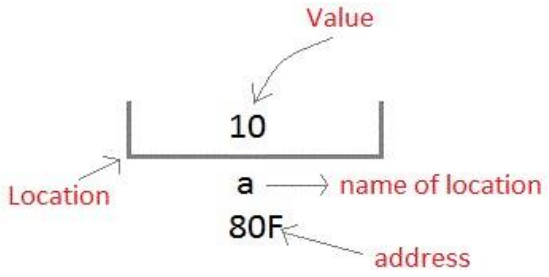
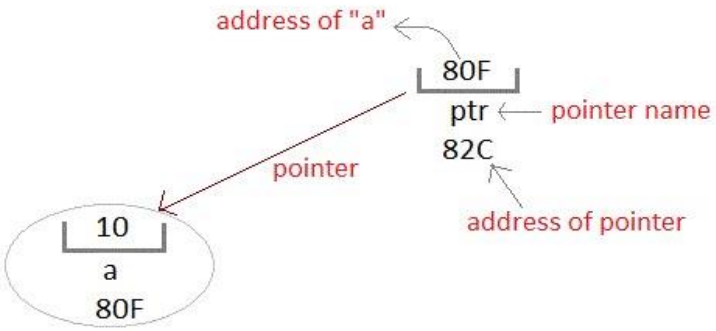
A **pointer** is a variable whose value is the address of another variable, to access or store direct address of the memory location. Like any variable or constant, you must declare a pointer before using it to store any variable address. A pointer can be incremented/decremented, to point to the next/ previous memory location. The purpose of pointer is to save memory space and achieve faster execution time

Whenever a **variable** is declared, system will allocate a location to that variable in the memory, to hold value. This location will have its own address number. Let us assume that system has allocated memory location 80F for a variable **a**. `int a = 10 ;`



We can access the value 10 by either using the variable name **a** or the address 80F. Since the memory addresses are simply numbers they can be assigned to some other variable. The variable that holds memory address are called **pointer variables**. A **pointer** variable is therefore nothing but a variable that contains an address, which is a location of another variable. Value of **pointer variable** will be stored in another memory location.



Declaration of Pc

Syntax:

*datatype *variable_name;*

Example:

int *p

Above statement defines **p** as pointer variable of type **int**.

Similarly,

```
double *p;      /* pointer to a double */
float  *p;      /* pointer to a float */
char   *p;      /* pointer to a character */
```

Reference operator (&) and Dereference operator (*)

- & is called reference operator. It gives you the address of a variable.
- Likewise, there is another operator that gets you the value from the address, it is called a dereference operator (*).

Initialization of Pointer variable

Pointer Initialization is the process of assigning address of a variable to **pointer** variable. Pointer variable contains address of variable of same data type. In C language **address operator** & is used to determine the address of a variable. The & (immediately preceding a variable name) returns the address of the variable associated with it.

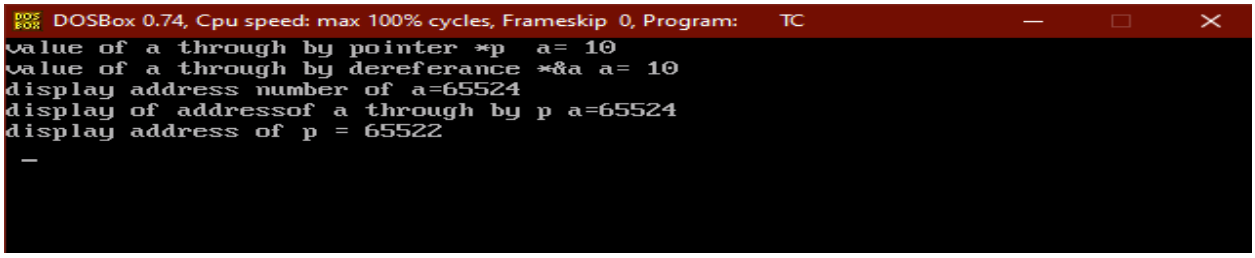
```
int a = 10 ;
int *ptr ;        //pointer variable declaration
ptr = &a ;        //pointer  variable initialization
     or
int *ptr = &a ;   //initialization and declaration together
```

Dereferencing of Pointer

Once a pointer has been assigned to the address of a variable. To access the value of variable, pointer is dereferenced, using the **indirection operator** *.

```
#include <conio.h>
#include <stdio.h>
void main()
{
    int a,*p;
    a = 10;
    p = &a;

   printf("value of a through by pointer *p  a= %d \n",*p);      //this will print the value of a.
   printf("value of a through by dereferance *&a a= %d \n",*&a);   //this will also print the value of a.
   printf("display address number of a=%u \n",&a);      //this will print the address of a.
   printf("display of addressof a through by p a=%u \n",p);       //this will also print the address of a.
   printf("display address of p= %u \n ",&p);     //this will also print the address of p.
   getche();
}
```



How to use Pointers?

There are a few important operations, which we will do with the help of pointers very frequently.

a) We define a pointer variable

b) assign the address of a variable to a pointer

c) Finally access the value at the address available in the pointer variable. This is done by using unary operator ***** that returns the value of the variable located at the address specified by its operand.

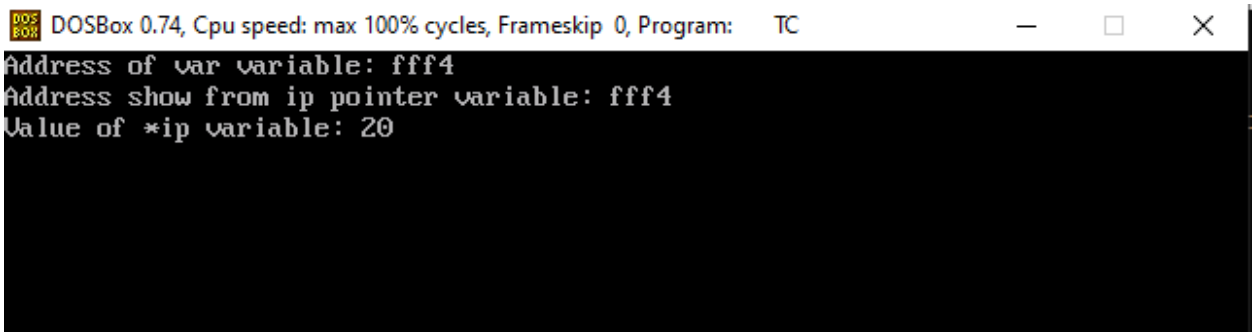
Example C program:

```
#include <stdio.h>
#include <conio.h>
 void main ()
{
    int  var = 20;     // actual variable declaration
    int  *ip;          // pointer variable declaration
   ip = &var;         // store address of var in pointer variable

   printf("Address of var variable: %x\n", &var  );   // address show of  variable var

   printf("Address show from ip pointer variable: %x\n", ip );  // address show using the
                                                                //pointer variable

    printf("Value of *ip variable: %d\n", *ip );    // Access the var value using the pointer variable ip
}
```



NULL Pointers

It is always a good practice to assign a NULL value to a pointer variable in case you do not have an exact address to be assigned. This is done at the time of variable declaration. A pointer that is assigned NULL is called a null pointer.

The NULL pointer is a constant with a value of zero defined in several standard libraries.

Example C program:

```
#include <stdio.h>
 void main( )
{
  int  *ptr = NULL;
  printf("The value of ptr is : %x\n", ptr  );
}
```

When the above code is compiled and executed, it produces the following result −

The value of ptr is 0.

Why we use pointer

1) Pointers save memory space.
2) Execution time with pointers is faster because data are manipulated with the address, that is, direct access to memory location.
3) Memory is accessed efficiently with the pointers and Pointers are used to allocate memory dynamically.
4) Pointers are used with data structures. They are useful for representing two-dimensional and multi-dimensional arrays.
5) Pointers are used for file handling.

Types of pointers

There are eight different types of pointers they are:

1) Null pointer
2) Void pointer
3) Wild pointer
4) Dangling pointer
5) Complex pointer
6) Near pointer
7) Far pointer
8) Huge pointe

Void Pointer

It is a pointer that has no associated data type with it. A void pointer can hold addresses of any type and can be typecast to any type.

It is also called a generic pointer and does not have any standard data type. It is created by using the keyword void.

Example

Following is the C program for the void pointer

```
#include <stdio.h>
void main()
{
  void *p = NULL;     //void pointer
  printf("The size of pointer is:%d\n",sizeof(p));     //size of p depends on compiler
}
```

Output

When the above program is executed, it produces the following result –

```
The size of pointer is:8
```

Wild Pointer:

Pointers that are not initialized are called wild pointers. This pointer may be initialized to a non-NULL garbage value which may not be a valid address.

Example

```
void main()
{
     int *p; // wild pointer
    *p= 10;
}
```

Remember if a pointer p points to any known variable, then it is not a wild pointer. In the below program p is a wild pointer until it points to x.

```
void main()
{
     int = *p; // wild pointer
     int x= 20;
    p= &x // p is not a wild pointer now
}
```

Dangling Pointer:

A pointer that points to a memory location that has been deleted is called a dangling pointer.

Example

```
#include<stdio.h>
#include<stdlib.h>

void main()
{

  int *ptr = (int *)malloc(sizeof(int));
  free(ptr);
  ptr = NULL;
}
```

Near Pointer

Near pointer is used to store 16 bits or 2 bytes addresses means within current segment on a 16 bit machine. The limitation is that we can only access 64kb of data at a time.

Example:

```
#include<stdio.h>
void main()
{
  int a= 300;
  int near* ptr;
  ptr= &a;
  printf("%d",sizeof ptr);
}
```

Far Pointer

A far pointer is typically 32 bits or 4 bytes that can access memory outside current segment. To use this, compiler allocates a segment register to store segment address, then another register to store offset within current segment.

Example:

```
#include<stdio.h>
void main()
{
 int a= 10;
 int far *ptr;
 ptr=&a;
 print("%d", sizeof ptr);
}
```

Huge Pointer

Like far pointer, huge pointer is also typically 32 bit and can access outside segment. In case of far pointers, a segment is fixed. In far pointer, the segment part cannot be modified, but in Huge it can be

Exercise

Theory Questions

1) Define pointer in C/C++;
2) Why we use pointer.
3) What is NULL pointer.
4) Write only list names types of pointers.

Practical Questions

1) Answer the following in one line code.
   a) Declare a pointer to an integer called *address*.
   b) Assign the address of a float variable *balance* to the float pointer *temp.*
   c) Assign the character value 'W' to the variable pointed to by the char pointer *letter.*
   d) Declare a pointer to the text string "Hello" called *message*.
2) What is the output of the following program segment?

```
void main()
{   int  count = 10, *temp, sum = 0;
    temp = &count;
     *temp = 20;
      temp = &sum;
      *temp = count;
  printf("count = %d, *temp = %d, sum = %d\n", count, *temp, sum );
}
```

3) Write a swap( ) function with the following function header:

```
void swap(int *p1, int *p2);
```

Objective MCQ's

1) The far pointer has ____bytes.
   a) 2
   b) 4
   c) 3
   d) 1

2) The near pointer has ____ bits
   a) 64
   b) 32
   c) 16
   d) 8

3) Which is the correct way to define a pointer?
   a) int_ptr x;
   b) int *ptr;
   c) *int ptr;
   d) *ptr

4) Assuming that the address of ***A*** has been assigned to pointer variable ***B***, which of the following expression are correct.
   a) A = &B ;
   b) A = *B ;
   c) B = A ;
   d) B = &A;

5) if we call a function with statement **num*(&a,&b)**, what purpose do ***&x*** and ***&y*** serve?
   a) They are integer values we are passing to the function.
   b) They are the addresses of the function and the calling program.
   c) They are the address of the variables where we want values returned or modified in the calling program.
   d) They are the addresses of library routine need by the function.

6) A pointer that points to a memory location that has been deleted is called a _____
   a) Void pointer
   b) Huge pointer
   c) Near pointer
   d) Dangling pointer.

7) Choose a correct statement about C structure elements.
   a) Structure elements are stored on random free memory locations
   b) structure elements are stored in register memory locations
   c) structure elements are stored in contiguous memory locations
   d) None of the above.

8) Which operator is called reference operator. It gives you the address of a variable.
   a) -
   b) &
   c) .
   d) <-

9) Which operator is called dereference operator. It gives you the value from address of a variable.
   a) -
   b) &
   c) *
   d) <-