Pointers in C Language:-

As you know, 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. Consider the following example, which prints the address of the variables defined –

```c
#include <stdio.h>

int main () {

   int  var1;
   char var2[10];

   printf("Address of var1 variable: %x\n", &var1  );
   printf("Address of var2 variable: %x\n", &var2  );

   return 0;
}
```

Output:-

What are Pointers?

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. The **general form** of a pointer variable **declaration** is −

datatype *var-name;

Here, **datatype** is the pointer's base type; it must be a valid C data type and **var-name** is the name of the pointer variable. The asterisk * used to declare a pointer is the same asterisk used for multiplication. However, in this statement the asterisk is being used to designate a variable as a pointer. Take a look at some of the valid pointer declarations −

```c
int    *ip;    /* pointer to an integer */
double *dp;    /* pointer to a double */
float  *fp;    /* pointer to a float */
char   *ch     /* pointer to a character */
```

The actual data type of the value of all pointers, whether integer, float, character, or otherwise, is the same, a long **hexadecimal number that represents a memory address**.

The only difference between pointers of different data types is the data type of the variable or constant that the pointer points to.

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 and

(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. The following example makes use of these operations −

```
#include <stdio.h> //header file

int main () {

   int  var = 20;   /* actual variable declaration */
   int  *ip;        /* pointer variable declaration */

   ip = &var;  /* store address of var in ip pointer
                                          variable*/

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

   /* address stored in pointer variable */
   printf("Address stored in ip variable: %x\n", ip );

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

   return 0;
}
```

```
Address of var variable: bffd8b3c
Address stored in ip variable: bffd8b3c
Value of *ip variable: 20
```

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. Consider the following program −

```
#include <stdio.h>

int main () {

   int  *ptr = NULL;//ptr - is a readable format for pointer

   printf("The value of ptr is : %x\n", ptr  );

   return 0;
}
```

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

```
The value of ptr is 0
```

In most of the operating systems, programs are not permitted to access memory at address 0 because that memory is **reserved by the operating system**. However, the memory address 0 has special significance; it signals that the pointer is not intended to point to an accessible memory location. But by convention, if a pointer contains the null (zero) value, it is assumed to point to nothing.

To check for a null pointer, you can use an 'if' statement as follows −

```
if(ptr)     /* succeeds if p is not null */
if(!ptr)    /* succeeds if p is null */
```

Pointers in Detail

Pointers have many but easy concepts and they are very important to C programming. The following important pointer concepts should be clear to any C programmer −

Sr.No.	Concept & Description
1	Pointer arithmetic There are four arithmetic operators that can be used in pointers: ++, --, +, -
2	Array of pointers You can define arrays to hold a number of pointers.
3	**Pointer to pointer** C allows you to have pointer on a pointer and so on.
4	Passing pointers to functions in C Passing an argument by reference or by address enable the passed argument to be changed in the calling function by the called function.
5	Return pointer from functions in C

	C allows a function to return a pointer to the local variable, static variable, and dynamically allocated memory as well.

Check the output of program?

```c
#include <stdio.h>

int main()

{

   int num = 10;

   printf("Value of variable num is: %d", num);

   /* To print the address of a variable we use %p
    * format specifier and ampersand (&) sign just
    * before the variable name like &num.
    */
   printf("\nAddress of variable num is: %p", &num);

               //  %p used for printing the value of a pointer in C

   return 0;

}
```

Output:- ????

```c
#include <stdio.h>
int main()
{
   //Variable declaration
   int num = 10;

   //Pointer declaration
   int *p;

   //Assigning address of num to the pointer p
   p = #

   printf("Address of variable num is: %p", p);
   return 0;
}
```

Output:??

C Pointers – Operators that are used with Pointers:-

"Address of"(&) Operator

```
printf("Address of var is: %p", &num);

Output:????
```

"Value at Address"(*) Operator

The * Operator is also known as **Value at address** operator.

How to declare a pointer?

```
int *p1  /*Pointer to an integer variable*/
double *p2  /*Pointer to a variable of data type double*/
char *p3   /*Pointer to a character variable*/
float *p4   /*pointer to a float variable*/
```

By using *** operator** we can access the value of a variable through a pointer.
For example:

```
double a = 10;
double *p;
p = &a;
```

*p would give us the value of the variable a. The following statement would display 10 as output.

```
printf("%d", *p);
```

Similarly if we assign a value to *pointer like this:

```
*p = 200;
```

It would change the value of variable a. The statement above will change the value of **"a"** from 10 to 200.

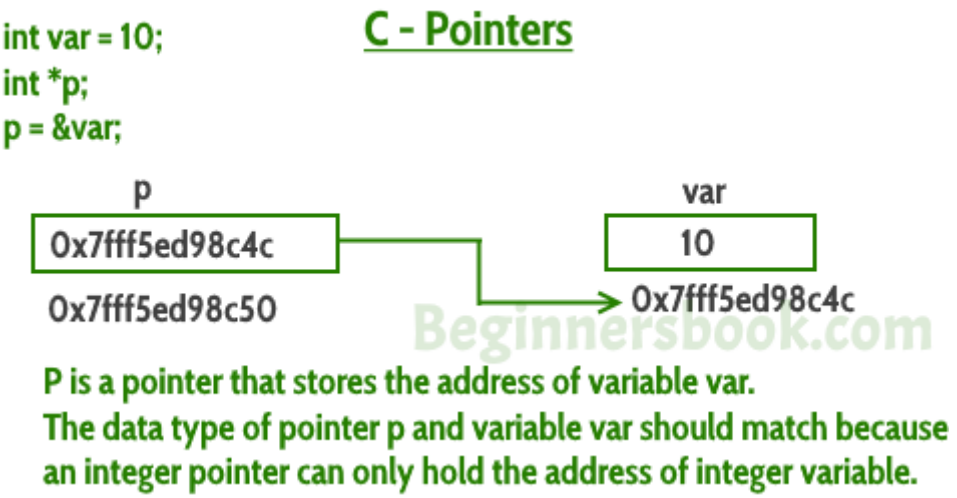
Pointer demonstrating the use of & and *

Sample Code:-

```c
#include <stdio.h>
int main()
{
   /* Pointer of integer type, this can hold the
    * address of a integer type variable.
    */
   int *p;
   int var = 10;

   /* Assigning the address of variable var to the pointer
    * p. The p can hold the address of var because var is
    * an integer type variable.
    */
   p= &var;
   printf("Value of variable var is: %d\n", var);
   printf("\nValue of variable var is: %d\n ", *p);
   printf("\nAddress of variable var is: %p\n ", &var);
   printf("\nAddress of variable var is: %p\n ", p);
   printf("\nAddress of pointer p is: %p\n ", &p);
    *p=250;
    Printf("value of P is %p\n",*p);
   return 0;
}
```

Output:

```
Value of variable var is: 10
Value of variable var is: 10
Address of variable var is: 0x7fff5ed98c4c
Address of variable var is: 0x7fff5ed98c4c
Address of pointer p is: 0x7fff5ed98c50
```



Lets take few more examples to understand it better –
Lets say we have a char variable ch and a pointer ptr that holds the address of ch.

```
char ch='a';
char *ptr;
```

Read the value of ch

```
printf("Value of ch: %c", ch);
or
printf("Value of ch: %c", *ptr);
```

Change the value of ch

```
ch = 'b';
or
*ptr = 'b';
```

The above code would replace the value ‘a’ with ‘b’.

Can you guess the output of following C program?

```
#include <stdio.h>
int main()
{
    int var =10;
    int *p;
    p= &var;
    printf ( "Address of var is: %p", &var);
    printf ( "\nAddress of var is: %p", p);

    printf ( "\nValue of var is: %d", var);
    printf ( "\nValue of var is: %d", *p);
    printf ( "\nValue of var is: %d", *( &var));
    /* Note I have used %p for p's value as it represents an address*/
    printf( "\nValue of pointer p is: %p", p);
    printf ( "\nAddress of pointer p is: %p", &p);

    return 0;
}
```

Output:

***Other** Topics on Pointers*

1) **Pointer to Pointer** – A pointer can point to another pointer (which means it can store the address of another pointer), such pointers are known as double pointer OR pointer to pointer. Ex: int **x1;

2) **Passing pointers to function** – Pointers can also be passed as an argument to a function, using this feature a function can be called by reference as well as an array can be passed to a function while calling.

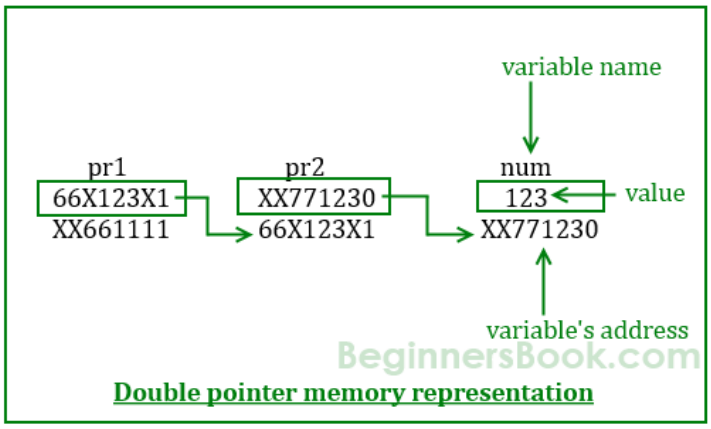
3) **Function pointers** – A function pointer is just like another pointer, it is used for storing the address of a function. Function pointer can also be used for calling a function in C program.

C – Pointer to Pointer (Double Pointer) with example

We already know that a pointer holds the address of another variable of same type. When a pointer holds the address of another pointer then such type of pointer is known as **pointer-to-pointer** or **double pointer**.

How to declare a Pointer to Pointer (Double Pointer) in C?

Let's understand the concept of double pointers with the help of a diagram:



Double pointer memory representation

As per the above diagram, **pr2** is a normal pointer that holds the address of an integer variable **num.** There is another pointer **pr1** in the diagram that holds the address of another pointer **pr2,** the pointer **pr1** here is a **pointer-to-pointer**

Values from above diagram:

```
Variable num has address: XX771230
Address of Pointer pr1 is: XX661111
Address of Pointer pr2 is: 66X123X1
```

Example of double Pointer

Lets write a C program based on the diagram that we have seen above.

```c
#include <stdio.h>
int main()
{
      int num=123;
      //A normal pointer pr2
      int *pr2;
      //This pointer pr2 is a double pointer
      int **pr1;
      /* Assigning the address of variable num to the
       * pointer pr2
       */
      pr2 = &num;
      /* Assigning the address of pointer pr2 to the
       * pointer-to-pointer pr1
       */
      pr1 = &pr2;
      /* Possible ways to find value of variable num*/
      printf("\n Value of num is: %d", num);
      printf("\n Value of num using pr2 is: %d", *pr2);
      printf("\n Value of num using pr1 is: %d", **pr1);

      /*Possible ways to find address of num*/
      printf("\n Address of num is: %p", &num);
      printf("\n Address of num using pr2 is: %p", pr2);
      printf("\n Address of num using pr1 is: %p", *pr1);

      /*Find value of pointer*/
      printf("\n Value of Pointer pr2 is: %p", pr2);
      printf("\n Value of Pointer pr2 using pr1 is: %p", *pr1);

      /*Ways to find address of pointer*/
      printf("\n Address of Pointer pr2 is:%p",&pr2);
      printf("\n Address of Pointer pr2 using pr1 is:%p",pr1);

      /*Double pointer value and address*/
      printf("\n Value of Pointer pr1 is:%p",pr1);
      printf("\n Address of Pointer pr1 is:%p",&pr1);

      return 0;
}
```

Output:

```
Value of num is: 123
Value of num using pr2 is: 123
Value of num using pr1 is: 123
Address of num is: XX771230
Address of num using pr2 is: XX771230
Address of num using pr1 is: XX771230
```

```
Value of Pointer pr2 is: XX771230
Value of Pointer pr2 using pr1 is: XX771230
Address of Pointer pr2 is: 66X123X1
Address of Pointer pr2 using pr1 is: 66X123X1
Value of Pointer pr1 is:  66X123X1
Address of Pointer pr1 is: XX661111
```

You can also understand the program logic with these simple equations:

```
num == *pr2 == **pr1
&num == pr2 == *pr1
&pr2 == pr1
```

Passing pointer to a function in C with example

You will learn how to pass a pointer to a function as an argument. To understand this concept you must have a basic idea of **Pointers** and **functions in C programming**.

Just like any other argument, pointers can also be passed to a function as an argument. Let's take an example to understand how this is done.

> **In this example, we are passing a pointer to a function. When we pass a pointer as an argument instead of a variable then the address of the variable is passed instead of the value. So any change made by the function using the pointer is permanently made at the address of passed variable. This technique is known as call by reference in C programming.**

```c
#include <stdio.h>
void salaryhike(int  *var, int b)
{
    *var = *var+b;
}
int main()
{
    int salary=0, bonus=0;
    printf("Enter the employee current salary:");
    scanf("%d", &salary);
    printf("Enter bonus:");
    scanf("%d", &bonus);
    salaryhike(&salary, bonus);
    printf("Final salary: %d", salary);
    return 0;
}
```

Output:-

```
Enter the employee current salary:10000
Enter bonus:2000
Final salary: 12000
```

Swapping two numbers using Pointers

This is one of the most popular example that shows how to swap numbers using call by reference.

```c
#include <stdio.h>
void swapnum(int *num1, int *num2)
{
   int tempnum;

   tempnum = *num1;
   *num1 = *num2;
   *num2 = tempnum;
}
int main( )
```

```
{
    int v1 = 11, v2 = 77 ;
    printf("Before swapping:");
    printf("\nValue of v1 is: %d", v1);
    printf("\nValue of v2 is: %d", v2);

    /*calling swap function*/
    swapnum( &v1, &v2 );

    printf("\nAfter swapping:");
    printf("\nValue of v1 is: %d", v1);
    printf("\nValue of v2 is: %d", v2);
}
```

Output:

```
Before swapping:
Value of v1 is: 11
Value of v2 is: 77
After swapping:
Value of v1 is: 77
Value of v2 is: 11
```

C – Function Pointer with examples

In C programming language, we can have a concept of Pointer to a function known as function pointer in C**.**

How to declare a function pointer?

Syntax:-

```
function_return_type(*Pointer_name)(function argument list)
```

For example:

```
double  (*p2f)(double, char)
```

Here double is a return type of function, p2f is name of the function pointer and (double, char) is an argument list of this function. Which means the first argument of this function is of double type and the second argument is char type.

#include<…>

```
int sum (int num1, int num2)
{
```

```
    return num1+num2;
}
int main()
{
    /* The following two lines can also be written in a single
     * statement like this: void (*fun_ptr)(int) = &fun;
     */
    int (*f2p) (int, int);
    f2p = sum;
    //Calling function using function pointer
    int op1 = f2p(10, 13);

    //Calling function in normal way using function name
    int op2 = sum(10, 13);

    printf("Output1: Call using function pointer: %d",op1);
    printf("\nOutput2: Call using function name: %d", op2);

    return 0;
}
```

Output:

```
Output1: Call using function pointer: 23
Output2: Call using function name: 23
```

Some points regarding function pointer:

1. As mentioned in the comments, you can declare a function pointer and assign a function to it in a single statement like this:

```
void (*fun_ptr)(int) = &fun;
```

2. You can even remove the ampersand from this statement because a function name alone represents the function address. This means the above statement can also be written like this:

```
void (*fun_ptr)(int) = fun;
```

Pointer and Array in C programming with example

We will learn how to work with Pointers and arrays in a C program. I recommend you to refer Array and Pointer .

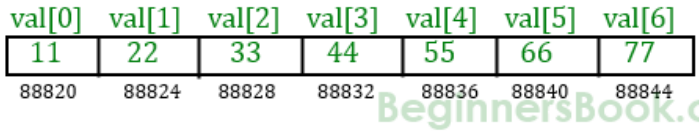
A simple example to print the address of array elements:

```
#include <stdio.h>
int main( )
{
   int val[7] = { 11, 22, 33, 44, 55, 66, 77 } ;
   /* for loop to print value and address of each element of array*/
   for ( int i = 0 ; i < 7 ; i++ )
   {
```

```
      /* The correct way of displaying the address would be using %p format
       * specifier like this:
       * printf("val[%d]: value is %d and address is %p\n", i, val[i], &val[i]);
       * Just to demonstrate that the array elements are stored in contiguous
       * locations, I m displaying the addresses in integer
       */
      printf("val[%d]: value is %d and address is %d\n", i, val[i],
&val[i]);
   }
   return 0;
}
```

Output:

```
val[0]: value is 11 and address is 1423453232
val[1]: value is 22 and address is 1423453236
val[2]: value is 33 and address is 1423453240
val[3]: value is 44 and address is 1423453244
val[4]: value is 55 and address is 1423453248
val[5]: value is 66 and address is 1423453252
val[6]: value is 77 and address is 1423453256
```



All the array elements occupy contigious space in memory. There is a difference of 4 among the addresses of subsequent neighbours, this is because this array is of integer types and an integer holds 4 bytes of memory.

Memory representation of array

In the above example I have used &val[i] to get the address of ith element of the array. We can also use a pointer variable instead of using the ampersand (&) to get the address.

Example – Array and Pointer Example in C

```
#include <stdio.h>
int main( )
{
   /*Pointer variable*/
   int *p;

   /*Array declaration*/
   int val[7] = { 11, 22, 33, 44, 55, 66, 77 } ;

   /* Assigning the address of val[0] the pointer
    * You can also write like this:
    * p = var;
    * because array name represents the address of the first element
    */
   p = &val[0];

   for ( int i = 0 ; i<7 ; i++ )
   {
      printf("val[%d]: value is %d and address is %p\n", i, *p, p);
```

```
      /* Incrementing the pointer so that it points to next element
       * on every increment.
       */
      p++;
   }
   return 0;
}
```

Output:

```
val[0]: value is 11 and address is 0x7fff51472c30
val[1]: value is 22 and address is 0x7fff51472c34
val[2]: value is 33 and address is 0x7fff51472c38
val[3]: value is 44 and address is 0x7fff51472c3c
val[4]: value is 55 and address is 0x7fff51472c40
val[5]: value is 66 and address is 0x7fff51472c44
val[6]: value is 77 and address is 0x7fff51472c48
```

Points to Note:

1) While using pointers with array, the data type of the pointer must match with the data type of the array.
2) You can also use array name to initialize the pointer like this:

```
p = var;
```

Pointer logic

You must have understood the logic in above code so now its time to play with few pointer arithmetic and expressions.

```
if p = &val[0] which means
*p ==val[0]
(p+1) == &val[2]  & *(p+1) == val[2]
(p+2) == &val[3]  & *(p+2) == val[3]
(p+n) == &val[n+1) & *(p+n) == val[n+1]
```

Using this logic we can rewrite our code in a better way like this:

```
#include <stdio.h>
int main( )
{
   int *p;
   int val[7] = { 11, 22, 33, 44, 55, 66, 77 } ;
   p = val;
   for ( int i = 0 ; i<7 ; i++ )
   {
      printf("val[%d]: value is %d and address is %p\n", i,
                                          *(p+i), (p+i));
   }
   return 0;
}
```

We don't need the p++ statement in this program.

Exercises in Pointers:

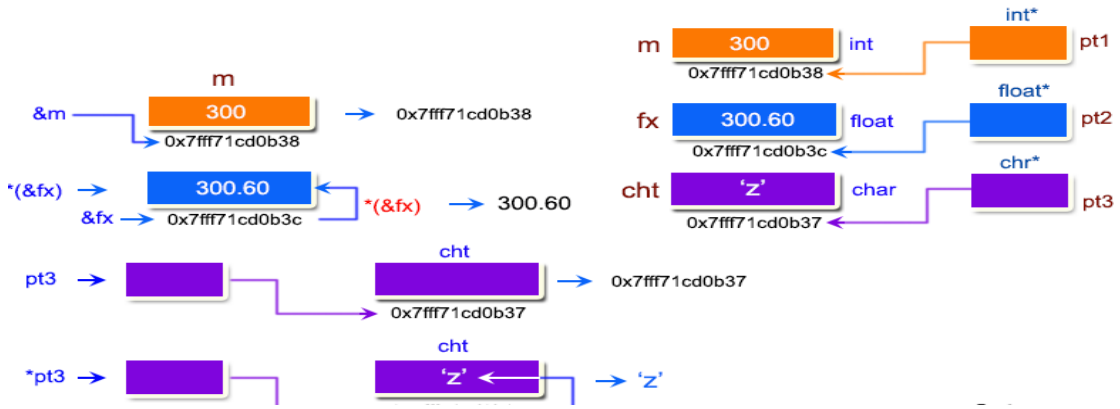
1. Write a program in C to show the basic declaration of pointer.

```c
#include <stdio.h>
void main(void)
{
int m=10,n,o;
int *z=&m ;

     printf("\n\n Pointer : Show the basic declaration of pointer
:\n");
     printf("-----------------------------------------------------
\n");
     printf(" Here is m=10, n and o are two integer variable and *z is
an integer");
     printf("\n\n z stores the address of m  = %p\n",  z); // z is a
pointer so %p would print the address
     printf("\n *z stores the value of m = %i\n",   *z);
     printf("\n &m is the address of m = %p\n",   &m); // &m gives the
address of the integer variable m
                                // so %p is the specifier for that address
     printf("\n &n stores the address of n = %p\n",   &n);
     printf("\n &o  stores the address of o = %p\n",   &o);
     printf("\n &z stores the address of z = %p\n\n", &z); // &z gives
the address, where the pointer z is
                                // stored -> still an address -> %p is the
right
                               // specifier
}
```

Expected Output :

```
 Pointer : Show the basic declaration of pointer :
-----------------------------------------------------
 Here is m=10, n and o are two integer variable and *z is an integer

 z stores the address of m  = 0x7ffd40630d44

 *z stores the value of m = 10

 &m is the address of m = 0x7ffd40630d44

 &n stores the address of n = 0x7ffd40630d48

 &o  stores the address of o = 0x7ffd40630d4c

 &z stores the address of z = 0x7ffd40630d50
```

2. Write a program in C to demonstrate the use of &(address of) and *(value at address) operator..




```c
#include <stdio.h>
void main()
{
  int m=300;
  float fx = 300.60;
  char cht = 'z';
  printf("\n\n Pointer : Demonstrate the use of & and * operator :\n");
  printf("-----------------------------------------------------\n");
  int *pt1;
  float *pt2;
  char *pt3;
  pt1= &m;
  pt2=&fx;
  pt3=&cht;
  printf ( " m = %d\n",m);
  printf ( " fx = %f\n",fx);
  printf ( " cht = %c\n",cht);
  printf("\n Using & operator :\n");
  printf("----------------------\n");
  printf ( " address of m = %p\n",&m);
  printf ( " address of fx = %p\n",&fx);
  printf ( " address of cht = %p\n",&cht);
  printf("\n Using & and * operator :\n");
  printf("----------------------------\n");
  printf ( " value at address of m = %d\n",*(&m));
  printf ( " value at address of fx = %f\n",*(&fx));
  printf ( " value at address of cht = %c\n",*(&cht));
  printf("\n Using only pointer variable :\n");
  printf("---------------------------------\n");
  printf ( " address of m = %p\n",pt1);
  printf ( " address of fx = %p\n",pt2);
  printf ( " address of cht = %p\n",pt3);
  printf("\n Using only pointer operator :\n");
  printf("---------------------------------\n");
  printf ( " value at address of m = %d\n",*pt1);
  printf ( " value at address of fx= %f\n",*pt2);
  printf ( " value at address of cht= %c\n\n",*pt3);
}
```

Sample Output:

```
Pointer : Demonstrate the use of & and * operator :
-----------------------------------------------------
 m = 300
 fx = 300.600006
 cht = z

 Using & operator :
-----------------------
 address of m = 0x7fff71cd0b38
 address of fx = 0x7fff71cd0b3c
 address of cht = 0x7fff71cd0b37

 Using & and * operator :
----------------------------
 value at address of m = 300
 value at address of fx = 300.600006
 value at address of cht = z

 Using only pointer variable :
---------------------------------
 address of m = 0x7fff71cd0b38
 address of fx = 0x7fff71cd0b3c
 address of cht = 0x7fff71cd0b37

 Using only pointer operator :
---------------------------------
 value at address of m = 300
 value at address of fx= 300.600006
 value at address of cht= z
```

3. Write a program in C to add two numbers using pointers?

```c
#include <stdio.h>
int main()
{
   int fno, sno, *ptr, *qtr, sum;
   printf("\n\n Pointer : Add two numbers :\n");
   printf("-------------------------------\n");
   printf(" Input the first number : ");
   scanf("%d", &fno);
   printf(" Input the second  number : ");
   scanf("%d", &sno);
   ptr = &fno;
   qtr = &sno;
   sum = *ptr + *qtr;
   printf(" The sum of the entered numbers is : %d\n\n",sum);
   return 0;
}
```

4. Write a program in C to add numbers using call by reference.

```c
#include <stdio.h>
long addTwoNumbers(long *, long *);
int main()
{
   long fno, sno, sum;
```

```
 printf("\n\n Pointer : Add two numbers using call by
 reference:\n");
 printf("-------------------------------------\n");
     printf(" Input the first number : ");
    scanf("%ld", &fno);
    printf(" Input the second  number : ");
    scanf("%ld", &sno);
    sum = addTwoNumbers(&fno, &sno);
    printf(" The sum of %ld and %ld  is %ld\n\n", fno,
                                          sno, sum);
    return 0;
 }
 long addTwoNumbers(long *n1, long *n2)
 {
    long sum;
    sum = *n1 + *n2;
    return sum;
 }
```

5. Write a program in C to find the maximum number between two numbers using a pointer

```
#include <stdio.h>
#include <stdlib.h>
void main()
{
 int firstnum,secondnum,*ptr1=& firstnum,*ptr2=&
secondnum;
   printf("\n\n Pointer:Find the maximum number between
                                  two numbers :\n");
   printf("-------------------------------------\n");
   printf(" Input the first number : ");
   scanf("%d", ptr1);
   printf(" Input the second  number : ");
   scanf("%d", ptr2);
 if(*ptr1>*ptr2)
 {
  printf("\n\n %d is the maximum number.\n\n",*ptr1);
 }
 else
 {
  printf("\n\n %d is the maximum number.\n\n",*ptr2);
 }
}
```