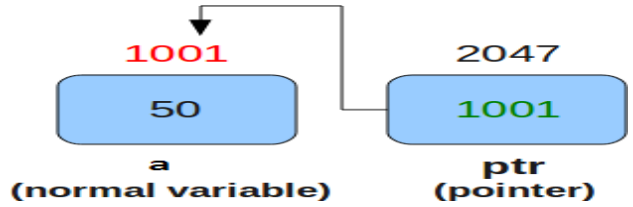
Pointers in C Programming

A pointer is variable which stores address of another variable.

Pointers can only store addresses of other variable.




```
int x=10, y=20;
printf("%u %u", &x, &y);
```

Note :Here **%u** is a format specifier. It stands for unsigned, so it will only display positive values.

You will get output of the above program like below.

605893 605897

&-address of operator.

& is the "address of" operator. It is used to tell the C compiler to refer to the address of variables. Address of any variable can't be negative. This is the reason %u format specifier is used to print the address of variables on the screen.

value at address (*) Operator

This is the second operator used for pointers. It is used to access the value present at some address. And it is used to declare a pointer.

Declaration and initialization of pointers

```
int x=10;
int *ptr;  // Declaration of Pointer variable
ptr=&x; // Storing address of x variable in y pointer variable
```

Example program-1

```
#include<stdio.h>
 void main()
{
int a=6,b=12;
int *x,*y;
x=&a;
y=&b;
printf("%d t %d n",a,b);
printf("%u t %u n",&a,&b);
printf("%u t %u n",x,y);
printf("%d t %d n",*x,*y);
printf("%d t %d",(&a),(&b));
printf("%d t %d",*(&a),*(&b));
}
```

```
6           12
65524       65522
65524       65522
6           12
65524       65522
6           12
```

```
#include <stdio.h>
int 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 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;
}
```

A **pointer** is a variable whose value is the memory address of another variable

syntax

```
type *var-name;
```

Here, **type** is the pointer's base type; it must be a valid C data type and **var-name** is the name of the pointer variable.

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

NULL Pointers

A pointer that is assigned NULL is called a **null** pointer.

The NULL pointer is a constant with a value of zero.

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.

```
#include <stdio.h>
int main ()
{
   int  *ptr = NULL;
   printf("The value of ptr is : %x\n", ptr);
    return 0;
}

output
The value of ptr is 0
```

Incrementing a Pointer(32-bit)

```
#include <stdio.h>
const int MAX = 3;
int main () {
   int  var[] = {10, 100, 200};
   int  i, *ptr;
                        /* let us have array address in pointer */
   ptr = var;
      for ( i = 0; i < MAX; i++) {
      printf("Address of var[%d] = %x\n", i, ptr );
      printf("Value of var[%d] = %d\n", i, *ptr );
                        /* move to the next location */
      ptr++;
   }

   return 0;
}
```

output

```
Address of var[0] = bf882b30
Value of var[0] = 10
Address of var[1] = bf882b34
Value of var[1] = 100
Address of var[2] = bf882b38
Value of var[2] = 200
```

Decrementing a Pointer(32-bit machine)

decreases its value by the number of bytes of its data type.

```
#include <stdio.h>
const int MAX = 3;
int main () {
   int  var[] = {10, 100, 200};
   int  i, *ptr;
                        /* let us have array address in pointer */
   ptr = &var[MAX-1];
    for ( i = MAX; i > 0; i--) {
      printf("Address of var[%d] = %x\n", i-1, ptr );
      printf("Value of var[%d] = %d\n", i-1, *ptr );
                      /* move to the previous location */
      ptr--;
   }

   return 0;
}
```

output

```
Address of var[2] = bfedbcd8
Value of var[2] = 200
Address of var[1] = bfedbcd4
Value of var[1] = 100
Address of var[0] = bfedbcd0
Value of var[0] = 10
```


Program for pointer arithmetic(32-bit machine)


```
#include <stdio.h>
int main()
{
    int m = 5, n = 10, val = 0;
    int *p1;
    int *p2;
    int *p3;

    p1 = &m;    //printing the address of m
    p2 = &n;    //printing the address of n

    printf("p1 = %d\n", p1);
    printf("p2 = %d\n", p2);

    printf(" *p1 = %d\n", *p1);
    printf(" *p2 = %d\n", *p2);


    val = *p1+*p2;
    printf("*p1+*p2 = %d\n", val);//point 1

    p3 = p1-p2;
    printf("p1 - p2 = %d\n", p3); //point 2
```

```
    p1++;
    printf("p1++ = %d\n", p1); //point 3

    p2--;
    printf("p2-- = %d\n", p2); //point 4


    return 0;
}
```

OUTPUT

```
p1 = 2680016
p2 = 2680012

*p1=5;
*p2=10;

*p1+*p2 = 15


p1-p2 = 1
p1++ = 2680020
p2-- = 2680008
```