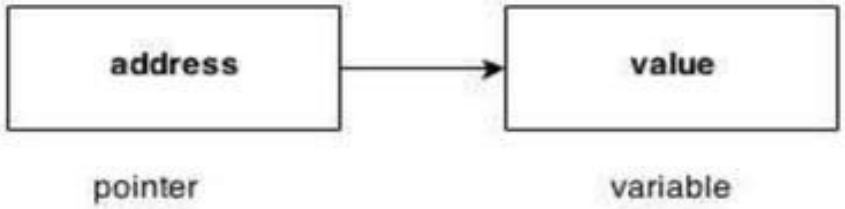
Pointers in C

Definition

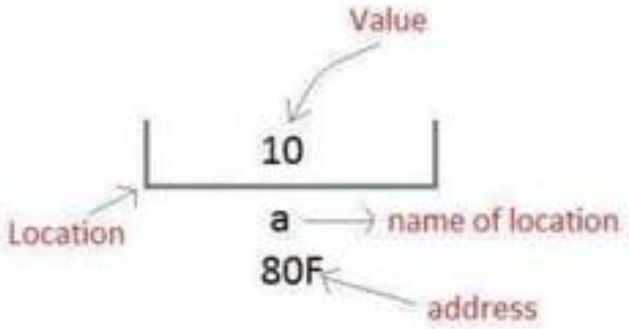
- A Pointer is a variable that holds address of another variable of same data type.
- also known as locator or indicator that points to an address of a value.



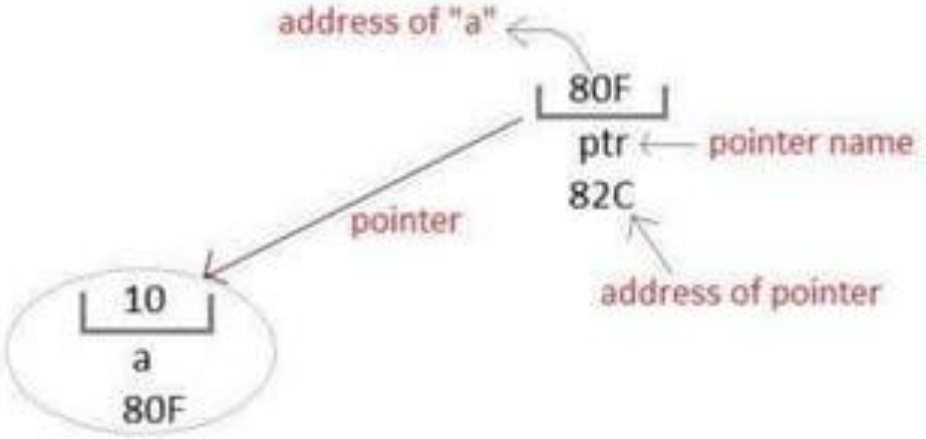
Concept of pointer

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

```
int a = 10 ;
```



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 ir



Declaring a pointer variable

- General syntax of pointer declaration is

```
data-type *pointer_name;
```

Initialization of Pointer variable

- 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 declaration
ptr = &a ;          //pointer initialization
or,
int *ptr = &a ;     //initialization and declaration together
```

Pointer variable always points to same type of data

```
float a;
int *ptr;
ptr = &a;     //ERROR, type mismatch
```

Note: If you do not have an exact address to be assigned to a pointer variable while declaration, It is recommended to assign a NULL value to the pointer variable. A pointer which is assigned a NULL value is called a null pointer.

Dereferencing of Pointer

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

```
int a,*p;
a = 10;
p = &a;

printf("%d",*p);    //this will print the value of a.

printf("%d",*&a);  //this will also print the value of a.

printf("%u",&a);  //this will print the address of a.

printf("%u",p);  //this will also print the address of a.

printf("%u",&p);  //this will print the address of p.
```

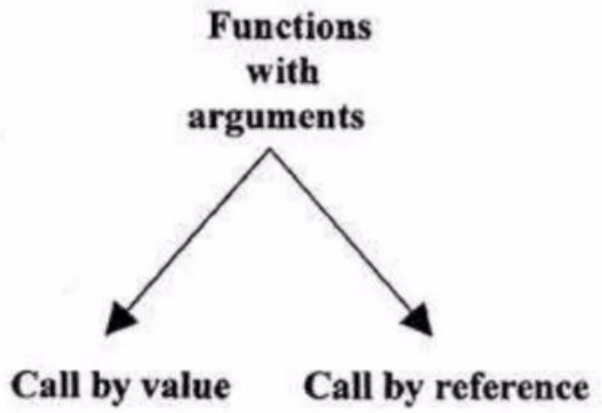
Write a program to declare an integer variable and a pointer to it.
Display the variable's value using both the variable and the pointer.

Write a program to input two integers and find their sum using pointers in a function

Swap Two Numbers Using Pointers Write a function void swap(int *a, int *b) to swap two integers using pointers. Display values before and after swapping.

Find Maximum of Two Numbers Using Pointers Write a program that takes two numbers and uses pointers to find and display the larger number.

THERE ARE TWO WAYS IN WHICH WE CAN PASS ARGUMENTS TO THE FUNCTION



Pass by Value

In pass by value, the function receives a copy of the variable's value.

•Any changes made to the parameter inside the function do not affect the original variable in the caller.

•Two separate copies exist in memory: The original variable in the caller and the function's local copy

```c
#include <stdio.h>

// Function using pass by value
void updateValue(int y)
{

    // y is a local copy of x; changes here do not affect x in main
    y = y + 10;
}

int main()
{
    int x = 50;

    printf("Before function call: %d\n", x);

    // Passes a copy of original
    updateValue(x);

    // original remains unchanged
    printf("After function call: %d\n", x);

    return 0;
}
```

Output

```
Before function call: 50
After function call: 50
```

In pass by reference, the function receives the address of the variable instead of a copy.

•The function can directly modify the original variable in the caller.

•Only one memory location exists for the variable, so changes inside the function affect the original variable.

•In C, this is done using pointers, as C does not have pass by reference like C++. Instead, you can pass the address of a variable to a function using pointers.

```c
#include <stdio.h>

// Function using pass by reference (via pointer)
void updateValue(int *y)
{

    // Modifies the original variable using its address
    *y = *y + 10;
}

int main()
{
    int x = 50;

    printf("Before function call: %d\n", x);

    // Passes the address of x
    updateValue(&x);

    // Original variable is changed
    printf("After function call: %d\n", x);

    return 0;
}
```

Output

```
Before function call: 50
After function call: 60
```

Actual and Formal arguments

- Arguments passed to the function during function call are known as actual arguments

- The arguments we use during a function definition are known as formal arguments.

CALL BY VALUE

- Value of actual arguments passed to the formal arguments.

- Any change made in the formal arguments does not effect the actual arguments.

- When function is called it does not affect the actual contents of the actual arguments.

Call by Reference

Definition: The memory address (reference) of the actual argument is passed to the function's formal parameter (a pointer).

Mechanism: The function can directly access and modify the original argument through its address.

Analogy: Giving someone the original document – any changes they make are permanent.