Que. What do you mean by pointers in c programming?How pointer variables are initialised?

Also write a note on pointer arithimetic with proper example in c language.

Ans. A pointer is **a variable that stores the memory address of another variable as its value**.

Initialising a pointer.

* Simply declearing a pointer value is not enough.
* It is important to initialise pointer before use.
* One way to initialise a pointer is to assign address of some value.

data\_type \*pointer;

Example:

int x=5;

int \*ptr;

ptr=&x;

or

int x=5, ptr= &x;

pointer arithmetic with proper example:

pdf 1.0 has ans.

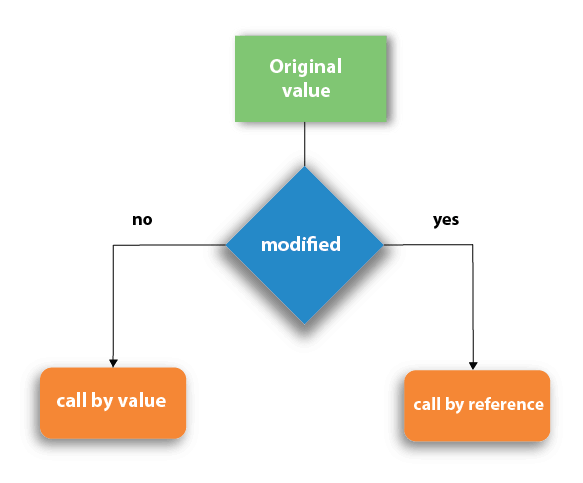
(Must watch)

Que. What do you mean by call by value and call by reference in c programming?

# Ans. Call by value and Call by reference in C

There are two methods to pass the data into the function in C language, i.e., *call by*

*value* and *call by referenc*e



## **Call by value in C**

* In call by value method, the value of the actual parameters is copied into the formal parameters. In other words, we can say that the value of the variable is used in the function call in the call by value method.
* In call by value method, we can not modify the value of the actual parameter by the formal parameter.
* In call by value, different memory is allocated for actual and formal parameters since the value of the actual parameter is copied into the formal parameter.
* The actual parameter is the argument which is used in the function call whereas formal parameter is the argument which is used in the function definition.

Let's try to understand the concept of call by value in c language by the example given below:

1. #include<stdio.h>
2. **void** change(**int** num) {
3. printf("Before adding value inside function num=%d \n",num);
4. num=num+100;
5. printf("After adding value inside function num=%d \n", num);
6. }
7. **int** main() {
8. **int** x=100;
9. printf("Before function call x=%d \n", x);
10. change(x);//passing value in function
11. printf("After function call x=%d \n", x);
12. **return** 0;
13. }

#### **Output**

Before function call x=100

Before adding value inside function num=100

After adding value inside function num=200

After function call x=100

#### **Call by Value Example: Swapping the values of the two variables**

1. #include <stdio.h>
2. **void** swap(**int** , **int**); //prototype of the function
3. **int** main()
4. {
5. **int** a = 10;
6. **int** b = 20;
7. printf("Before swapping the values in main a = %d, b = %d\n",a,b); // printing the value of a and b in main
8. swap(a,b);
9. printf("After swapping values in main a = %d, b = %d\n",a,b); // The value of actual parameters do not change by changing the formal parameters in call by value, a = 10, b = 20
10. }
11. **void** swap (**int** a, **int** b)
12. {
13. **int** temp;
14. temp = a;
15. a=b;
16. b=temp;
17. printf("After swapping values in function a = %d, b = %d\n",a,b); // Formal parameters, a = 20, b = 10
18. }

#### **Output**

Before swapping the values in main a = 10, b = 20

After swapping values in function a = 20, b = 10

After swapping values in main a = 10, b = 20

## **Call by reference in C**

* In call by reference, the address of the variable is passed into the function call as the actual parameter.
* The value of the actual parameters can be modified by changing the formal parameters since the address of the actual parameters is passed.
* In call by reference, the memory allocation is similar for both formal parameters and actual parameters. All the operations in the function are performed on the value stored at the address of the actual parameters, and the modified value gets stored at the same address.

1. #include<stdio.h>
2. **void** change(**int** \*num) {
3. printf("Before adding value inside function num=%d \n",\*num);
4. (\*num) += 100;
5. printf("After adding value inside function num=%d \n", \*num);
6. }
7. **int** main() {
8. **int** x=100;
9. printf("Before function call x=%d \n", x);
10. change(&x);//passing reference in function
11. printf("After function call x=%d \n", x);
12. **return** 0;
13. }

#### **Output**

Before function call x=100

Before adding value inside function num=100

After adding value inside function num=200

After function call x=200

#### **Call by reference Example: Swapping the values of the two variables**

1. #include <stdio.h>
2. **void** swap(**int** \*, **int** \*); //prototype of the function
3. **int** main()
4. {
5. **int** a = 10;
6. **int** b = 20;
7. printf("Before swapping the values in main a = %d, b = %d\n",a,b); // printing the value of a and b in main
8. swap(&a,&b);
9. printf("After swapping values in main a = %d, b = %d\n",a,b); // The values of actual parameters do change in call by reference, a = 10, b = 20
10. }
11. **void** swap (**int** \*a, **int** \*b)
12. {
13. **int** temp;
14. temp = \*a;
15. \*a=\*b;
16. \*b=temp;
17. printf("After swapping values in function a = %d, b = %d\n",\*a,\*b); // Formal parameters, a = 20, b = 10
18. }

#### **Output**

Before swapping the values in main a = 10, b = 20

After swapping values in function a = 20, b = 10

After swapping values in main a = 20, b = 10

## **Difference between call by value and call by reference in c**

|  |  |  |
| --- | --- | --- |
| **No.** | **Call by value** | **Call by reference** |
| 1 | A copy of the value is passed into the function | An address of value is passed into the function |
| 2 | Changes made inside the function is limited to the function only. The values of the actual parameters do not change by changing the formal parameters. | Changes made inside the function validate outside of the function also. The values of the actual parameters do change by changing the formal parameters. |
| 3 | Actual and formal arguments are created at the different memory location | Actual and formal arguments are created at the same memory location |

Que. Discuss various classification of digital computer.

Ans.

Que. Discuss the various classification of computer system.

Ans.

Que. What is the meaning of memory allocation in c programming?

Ans. Memory allotted at compile time is known as STATIC MEMORY allocation.

* Memory allotted is fixed and can not be increased or decreased during run time.

Exp:

#include<stdio.h>

int main()

{

int a[20]= {1,2,3,4,5};

}

It is an a example of static memory allocation.

Problem faced in static memory allocation:

* If you are allocating memory for an array during compile time then we have to

fixe the size of memory at time of decleration. So user can not increse or decrese the size of memory allotted at run time.

* If the value stored by the user in the array is less than the size of specified, then there will be a wastage of memory.
* If the value stored by the user in the array is more than the size of specified, then program may crash or misbehave.

Dynamic memory allocation.

The process of allocating memory at the time of execution. Then it is known as Dynamic memory allocation.

Dynamic memory allocation is of 4 types.

* malloc()
* realloc()
* calloc()
* free()

Note:-

POINTERS PLAYS AN IMPORTANT ROLE IN DYNAMIC MEMORY ALLOCATION . ALLOCATED MEMORY CAN ONLY BE ASSESSED THORUGH POINTERS.