MyAnatomy

Pointers

What is Address in C++?

If you have a variable var in your program, &var will give you it's address in the memory.

What is Pointer in C++?

A Pointer in C++ language is a variable that holds a memory address.

This memory address is the address of another variable(mostly) of same data type.

In simple words, if one variable stores the address of second variable then the first variable can be said to point towards the second variable.

Whenever a variable is declared in a program, system allocates a location i.e an address to that variable in the memory, to hold the assigned value.

Benefits of using pointers in C++?

Pointers are more efficient in handling Arrays in C++ and Structures in C++.

Pointers allow references to function and thereby helps in passing of function as arguments to other functions.

Pointers also provide means by which a function can change it's calling arguments.

It reduces length of the program and it's execution time as well.

It allows C++ language to support Dynamic Memory management.

Pointer Operators

There are two pointer operators in C++,

1) & operator (address of operator / reference operator):

The & operator returns the memory address of its operand. In the variable 'a', the memory address of the variable b will get stored.

e.g. a = &b;

2) * operator (indirection operators / dereference operator):

The * operator is the complement of &. This operator returns the value located at the given address. e.g. if 'a' contains the memory address of the variable b, then the following line will store the value of the variable b into c.

e.g. c = *a;

Access memory address of any variable using pointer

```
int var = 7;

cout << "Value of the variable var is: \n" << var;

cout << "Memory address of the variable var is: " <<  &var;
```

Pointer Variables

The variables which are used to hold memory addresses are called Pointer variables.

A pointer variable is therefore nothing but a variable which holds an address of some other variable. And the value of a pointer variable gets stored in another memory location.

Declaration of pointer

Syntax- type *name;

type is the data type of the pointer, and the name is the name of the pointer variable.

And the * operator with the name, informs the compiler that the variable is a pointer variable.

Examples

int* p;
int *p1;
int * p2;

Note:

- The data type of the pointer variable should be the same as of the variable to which the pointer is pointing.
- The declaration int *p doesn't mean that a is going to contain an integer value. It means that a is going to contain the address of a variable of int type.

Assigning value to Pointers

While declaring a pointer variable, if it is not assigned to anything then it contains garbage value. which means it could be pointing anywhere in the memory.

And that can lead to unexpected errors in your program. Hence, it is recommended to assign some value to it.

Example –

```
int *pc, c;

c = 5;

pc = &c;
```

Here, 5 is assigned to the c variable. And, the address of c is assigned to the pc pointer.

Access value & memory address of any variable using pointer

To get the value of the thing pointed by the pointers, we use the * operator.

Example –

```
int *pc, c;

c = 5;

pc = &c;

cout << *pc;          // Output: 5

cout << pc;           // memory address
```

Here, the address of c is assigned to pc pointer. To get the value stored in that address, we used *pc

Changing Value of variables Pointed by Pointers

Example1 –

```
int* pc, c;
c = 5;
pc = &c;
c = 1;
cout << c;
cout << *pc;
```

Example2 –

```
int *pc, c, d;
c = 5;
d = -15;
pc = &c;
cout << *pc;
pc = &d;
cout << *pc;
```

Null pointer in C

While declaring a pointer variable, if it is not assigned to anything then it contains garbage value. And that can lead to unexpected errors in your program. Hence, it is recommended to assign a NULL value to it.

When we assign NULL to a pointer, it means that it does not point to any valid address. NULL denotes the value 'zero'.

A pointer that is assigned a NULL value is called a NULL pointer in C++.

We can give a pointer a null value by assigning it zero to it.

```
int *ptr = 0;
```

We can also use the NULL keyword, which is nothing but a predefined constant for null pointer.

```
int *ptr = NULL;
```

Common mistakes when working with pointers

```
int c, *pc;

pc = c;          // pc is address but c is not so it's an Error

*pc = &c;        // &c is address but *pc is not so it's an Error

pc = &c;         // both &c and pc are addresses. // Not an error

*pc = c;         // both c and *pc are values. // Not an error
```

Working of Pointers

```
#include <iostream>
using namespace std;
int main(void)
{
   int* pc, c;

   c = 22;
   cout << "Address of c: " << &c << endl;
   cout << "Value of c: " << c << endl << endl;

   pc = &c;
   cout << "Address of pointer pc: " << pc << endl;
   cout << "Content of pointer pc: " << *pc << endl << endl;

   c = 11;
   cout << "Address of pointer pc: " << pc << endl;
   cout << "Content of pointer pc: " << *pc << endl << endl;

   *pc = 2;
   cout << "Address of c: " << &c << endl;
   cout << "Value of c: " << c << endl;
}
```

Explanation:

1. int* pc, c; Here, a pointer pc and a normal variable c both of type int are created.
2. c = 22; This assigns 22 to the variable c. That is, 22 is stored in the memory location of variable c.

3. pc = &c; This assigns the address of variable c to the pointer pc.

4. c = 11; This assigns 11 to variable c.

5. *pc = 2; This change the value at the memory location pointed by the pointer pc to 2.

Accessing array using Pointers

Pointers are variables that hold addresses of other variables. Not only can a pointer store the address of a single variable, it can also store the address of cells of an array.

Example:

```
int *ptr;
int arr[5];
ptr = &arr[0];        // store the address of the first element of arr in ptr
```

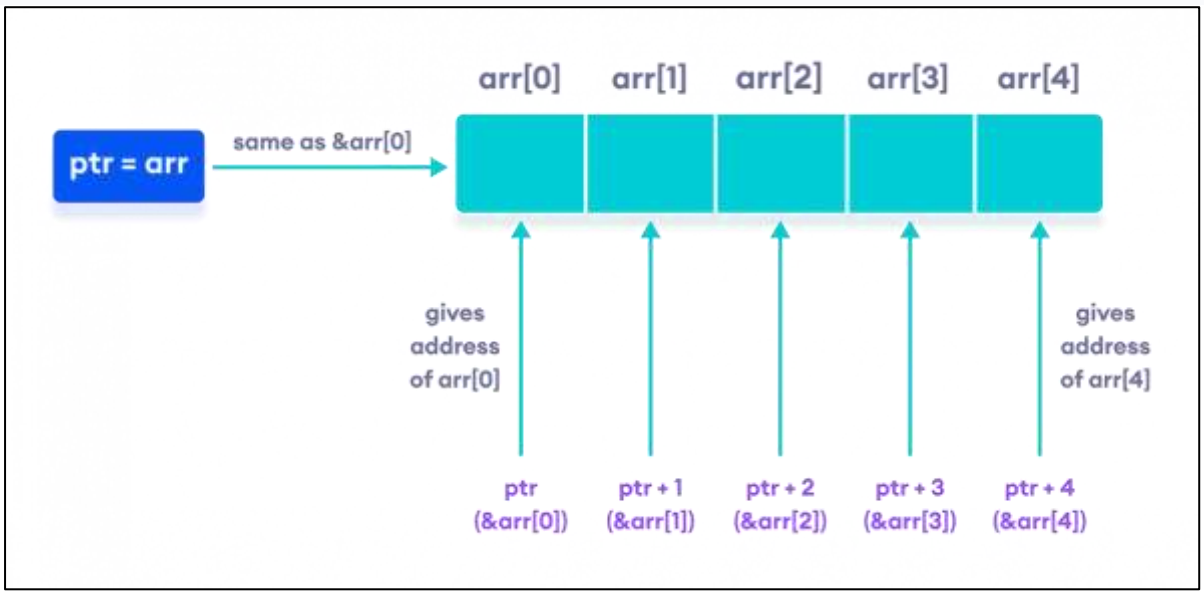
The addresses for the rest of the array elements are given by &arr[1], &arr[2], &arr[3], and &arr[4].

Point to Every Array Elements using single pointer

int *ptr;
int arr[5];
ptr + 1 is equivalent to &arr[1];
ptr + 2 is equivalent to &arr[2];
ptr + 3 is equivalent to &arr[3];
ptr + 4 is equivalent to &arr[4];

We can access the elements using the single pointer.

Example: // use dereference operator
*ptr == arr[0];
*(arr + 1) is equivalent to arr[1];
*(arr + 2) is equivalent to arr[2];
*(arr + 3) is equivalent to arr[3];
*(arr + 4) is equivalent to arr[4];



Program to display address of each element of an array

```
#include <iostream>
using namespace std;
int main(void)
{
   float arr[5];
   float *ptr;
   cout << "Displaying address using arrays: " << endl;
   for (int i = 0; i < 3; ++i)
   {
      cout << "&arr[" << i << "] = " << &arr[i] << endl;
   }
   ptr = arr;      // ptr = &arr[0]

   cout<<"\nDisplaying address using pointer notation: "<< endl;
   for (int i = 0; i < 3; ++i)
   {
      cout << "ptr + " << i << " = "<< ptr + i << endl;
   }
}
```

Program to display element of an array using pointer notation

In this program array name is used as pointer

```
#include <iostream>
using namespace std;
int main(void)
{
   float arr[] = {1.2,3.4,5.6,7.8,9.10};
   cout<<"\nDisplaying Elements using pointer notation: "<< endl;
   for (int i = 0; i < 5; ++i)
   {
      cout << "ptr + " << i << " = "<< *(arr+i) << endl;
   }
}
```

Note: cout >> *(arr + i) is equivalent to cout >> arr[i];