Pointers in C++

The pointer in C++ language is a variable, it is also known as locator or indicator that points to an address of a value.

The symbol of an address is represented by a pointer. In addition to creating and modifying dynamic data structures, they allow programs to emulate call-by-reference. One of the principal applications of pointers is iterating through the components of arrays or other data structures. The pointer variable that refers to the same data type as the variable you're dealing with has the address of that variable set to it (such as an int or string).

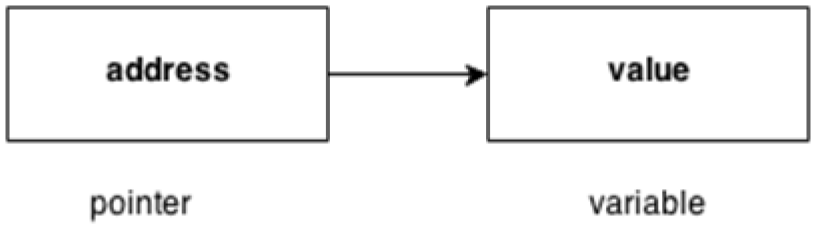
Syntax

1. datatype *var_name;
2. **int** *ptr;   // ptr can point to an address which holds int data

How to use a pointer?

1. Establish a pointer variable.
2. employing the unary operator (&), which yields the address of the variable, to assign a pointer to a variable's address.
3. Using the unary operator (*), which gives the variable's value at the address provided by its argument, one can access the value stored in an address.

Since the data type knows how many bytes the information is held in, we associate it with a reference. The size of the data type to which a pointer points is added when we increment a pointer.



Advantage of pointer

1) Pointer reduces the code and improves the performance, it is used to retrieving strings, trees etc. and used with arrays, structures and functions.

2) We can return multiple values from function using pointer.

3) It makes you able to access any memory location in the computer's memory.

Usage of pointer

There are many usage of pointers in C++ language.

1) Dynamic memory allocation

In c language, we can dynamically allocate memory using malloc() and calloc() functions where pointer is used.

2) Arrays, Functions and Structures

Pointers in c language are widely used in arrays, functions and structures. It reduces the code and improves the performance.

Symbols used in pointer

Symbol	Name	Description
& (ampersand sign)	Address operator	Determine the address of a variable.
* (asterisk sign)	Indirection operator	Access the value of an address.

Declaring a pointer

The pointer in C++ language can be declared using * (asterisk symbol).

```
int *  a; //pointer to int
char *  c; //pointer to char
```

Pointer Example

Let's see the simple example of using pointers printing the address and value.

```
#include <iostream>
using namespace std;
int main()
{
int number=30;
int *  p;
p=&number;//stores the address of number variable
cout<<"Address of number variable is:"<<&number<<endl;
cout<<"Address of p variable is:"<<p<<endl;
cout<<"Value of p variable is:"<<*p<<endl;
```

```
  return 0;
}
```

Output:

```
Address of number variable is:0x7ffccc8724c4
Address of p variable is:0x7ffccc8724c4
Value of p variable is:30
```

Pointer Program to swap 2 numbers without using 3rd variable

```
#include <iostream>
using namespace std;
int main()
{
int a=20,b=10,*p1=&a,*p2=&b;
cout<<"Before swap: *p1="<<*p1<<" *p2="<<*p2<<endl;
*p1=*p1+*p2;
*p2=*p1-*p2;
*p1=*p1-*p2;
cout<<"After swap: *p1="<<*p1<<" *p2="<<*p2<<endl;
  return 0;
}
```

Output:

```
Before swap: *p1=20 *p2=10
After swap: *p1=10 *p2=20
```

What are Pointer and string literals?

String literals are arrays of character sequences with null ends. The elements of a string literal are arrays of type const char (because characters in a string cannot be modified) plus a terminating null-character.

What is a void pointer?

This unique type of pointer, which is available in C++, stands in for the lack of a kind. Pointers that point to a value that has no type are known as void pointers (and thus also an undetermined length and undetermined dereferencing properties). This indicates that void pointers are very flexible because they can point to any data type. This flexibility has benefits. Direct dereference is not possible with these pointers. Before they may be dereferenced, they must be converted into another pointer type that points to a specific data type.

What is a invalid pointer?

A pointer must point to a valid address, not necessarily to useful items (like for arrays). We refer to these as incorrect pointers. Additionally, incorrect pointers are uninitialized pointers.

1. **int** *ptr1;
2. **int** arr[10];
3. **int** *ptr2 = arr+20;

Here, ptr1 is not initialized, making it invalid, and ptr2 is outside of the bounds of arr, making it likewise weak. (Take note that not all build failures are caused by faulty references.)

What is a null pointer?

A null pointer is not merely an incorrect address; it also points nowhere. Here are two ways to mark a pointer as NULL:

1. **int** *ptr1 = 0;
2. **int** *ptr2 = NULL;

What is a pointer to a pointer?

In C++, we have the ability to build a pointer to another pointer, which might then point to data or another pointer. The unary operator (*) is all that is needed in the syntax for declaring the pointer for each level of indirection.

1. **char** a;
2. **char** *b;
3. **char** ** c;
4. a = 'g';
5. b = &a;
6. c = &b;

Here b points to a char that stores 'g', and c points to the pointer b.

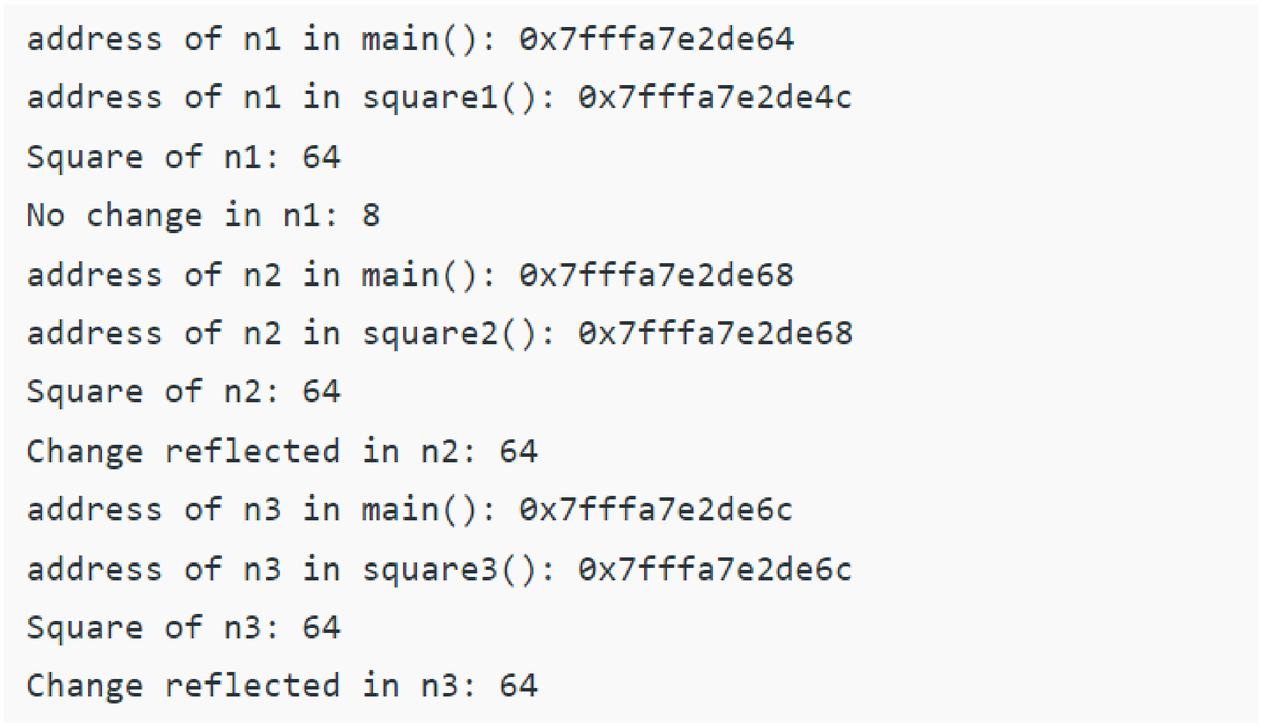
What are references and pointers?

1. Call-By-Value
2. Call-By-Reference with a Pointer Argument
3. Call-By-Reference with a Reference Argument

Example

```
#include
using namespace std;
// Pass-by-Value
int square1(int n)
{cout << "address of n1 in square1(): " << &n << "\n";
n *= n;
return n;
}
// Pass-by-Reference with Pointer Arguments
void square2(int* n)
{
cout << "address of n2 in square2(): " << n << "\n";
*n *= *n;
}
// Pass-by-Reference with Reference Arguments
void square3(int& n)
{

cout << "address of n3 in square3(): " << &n << "\n";
n *= n;
}
void example()
{
    // Call-by-Value
    int n1 = 8;
    cout << "address of n1 in main(): " << &n1 << "\n";
    cout << "Square of n1: " << square1(n1) << "\n";
    cout << "No change in n1: " << n1 << "\n";

    // Call-by-Reference with Pointer Arguments
    int n2 = 8;
    cout << "address of n2 in main(): " << &n2 << "\n";
    square2(&n2);
    cout << "Square of n2: " << n2 << "\n";
    cout << "Change reflected in n2: " << n2 << "\n";

    // Call-by-Reference with Reference Arguments
    int n3 = 8;
    cout << "address of n3 in main(): " << &n3 << "\n";
    square3(n3);
    cout << "Square of n3: " << n3 << "\n";
    cout << "Change reflected in n3: " << n3 << "\n";
}
// Driver program
int main() { example(); }
```

Output



```
address of n1 in main(): 0x7fffa7e2de64
address of n1 in square1(): 0x7fffa7e2de4c
Square of n1: 64
No change in n1: 8
address of n2 in main(): 0x7fffa7e2de68
address of n2 in square2(): 0x7fffa7e2de68
Square of n2: 64
Change reflected in n2: 64
address of n3 in main(): 0x7fffa7e2de6c
address of n3 in square3(): 0x7fffa7e2de6c
Square of n3: 64
Change reflected in n3: 64
```

POINTER TO OBJECTS:

A Variable That Holds an Address value is called a Pointer variable or simply pointer. We already discussed about pointer that's point to Simple data types likes int, char, float etc. So similar to these type of data type, Objects can also have an address, so there is also a pointer that can point to the address of an Object, This Pointer is Known as **This Pointer.**

C++ DECLARATION AND USES OF OBJECT POINTERS:

Just like Other Pointers, the object pointers are declared by placing in front of a object pointer's name. The Syntax is:

class_name * Object_pointer_name;

In above Syntax, class_Name is the name of an already defined class and object_pointer_name is the pointer to an object of this class type. For example: To declared an Pointer ptr an an object pointer of class Date, We shall write:

Date * ptr;

In Above syntax, **Date** is already defined class. When accessing members of a class using an object pointer, the Error Operator -> is used instead of Dot Operator. Let's take an simple example, This Example illustrates how to access an object given a pointer to it. The following Example illustrates the use of object pointer.

```
#include<iostream>

using namespace std;


class Date

{

   private:

      short int dd, mm, yy;


   public:

      Date() //constrctor:

         {

            dd = mm = yy = 0;

         }


      void getdata(int i, int j, int k)

         {

            dd = i;

            mm = j;

            yy = k;

         }



      void prndata(void)

         {

            cout<<"\nData is "<<dd<<"/"<<mm<<"/"<<yy<<"\n";

         }
};

main()
{
   Date D1; //simple object having type Data:
   Date *dptr; //Pointer Object having type Date:

   cout<<"Initializing data members using the object, with values 19, 10, 2016"<<endl;
   D1.getdata(19,10,2016);

   cout<<"Printing members using the object ";
   D1.prndata();

   dptr = &D1;
   cout<<"Printing members using the object pointer ";
   dptr->prndata();

   cout<<"\nInitializing data members using the object pointer, with values 20, 10,
2016"<<endl;
   dptr->getdata(20, 10, 2016);
   cout<<"printing members using the object ";
   D1.prndata();

   cout<<"Printing members using the object pointer ";
   dptr->prndata();

   return 0;
}
```

The above Example is self-explanatory. To Access public members using an object the Dot (**.**) Operator is used, and to access public members using an Object pointer, the Arrow (**-**

>)operator is used.

THIS POINTER:

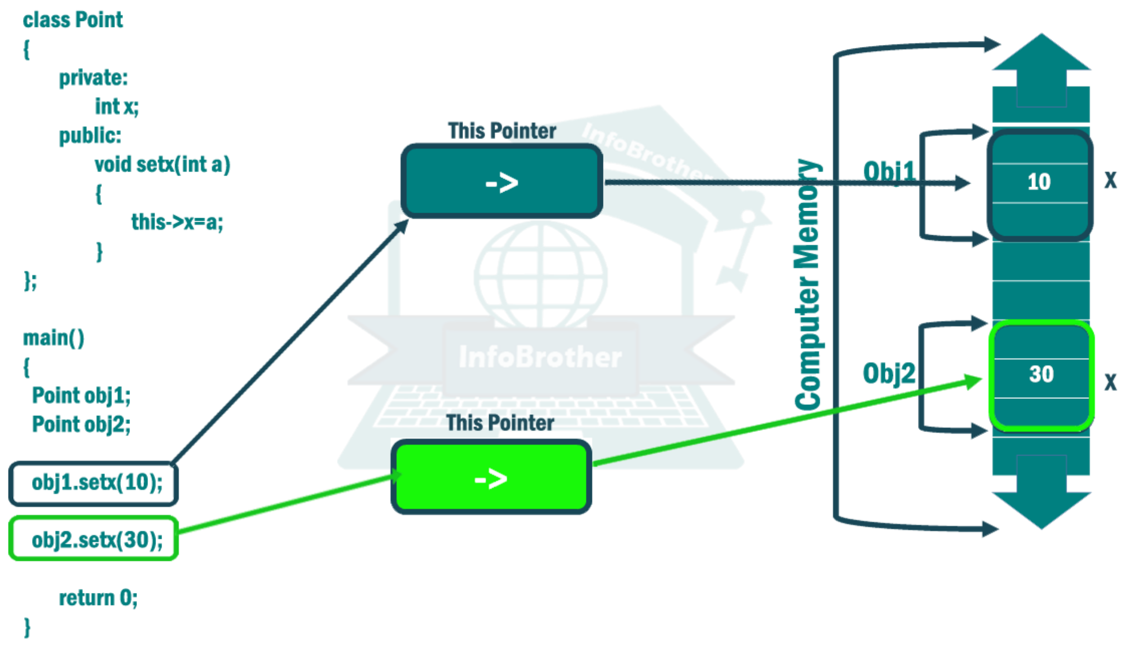
Every Object in C++ has access to its own address through an important pointer called This Pointer. The This Pointer is an implicit parameter to all member functions. Therefore, inside a member function, this may be used to refer to the invoking object. Whenever a member function is called, it is automatically passed an implicit arguments that is This pointer to the invoking object *(i.e. The object on which the function is invoked).*

The This pointer is passed as a hidden argument to all Nonstatic member function calls and is available as a local variable within the body of all Nonstatic functions. This Pointer is a constant pointer that holds the memory address of the current object. This pointer is not available in static member functions as static member functions can be called without any object *(with class name).*

Let's have an simple example of This pointer.

```
#include<iostream>
using namespace std;

class Point
{
  private:
    int x;

  public:
    void setx(int a) //simple function.
    {
      this->x=a;
    }
};
```

```
main()
{
  Point obj1;   //1st Object of Class Point.
  Point obj2;   //2nd Object of Class point.

  obj1.setx(10); //function calling using 1st object.
  obj2.setx(30); //function calling using 2nd object.

  return 0;
}
```



```
class Point
{
    private:
        int x;
    public:
        void setx(int a)
        {
            this->x=a;
        }
};

main()
{
  Point obj1;
  Point obj2;

  obj1.setx(10);

  obj2.setx(30);

    return 0;
}
```