

Pointers

- To understand pointers, you should first know how data is stored on the computer.
- Each variable you create in your program is assigned a location in the computer's memory. The value the variable stores is actually stored in the location assigned.
- To know where the data is stored, C++ has an **& operator**.
- **The & (reference) operator gives you the address occupied by a variable.**
- **If `var` is a variable then, `&var` gives the address of that variable.**

Example 1: illustrate the Address in C++

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

int main()
{
    int var1 = 3;
    int var2 = 24;
    int var3 = 17;
    cout << &var1 << endl;
    cout << &var2 << endl;
    cout << &var3 << endl;
}
```

Output

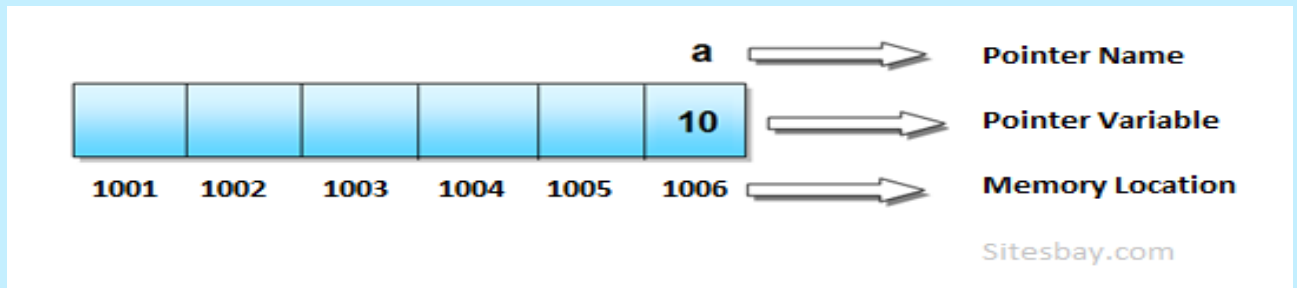
```
0x7fff5fbff8ac
0x7fff5fbff8a8
0x7fff5fbff8a4
```

توضيح

- You may not get the same result on your system.
- The **0x** in the beginning represents the address is in hexadecimal form.
- Notice that first address differs from second by 4-bytes and second address differs from third by 4-bytes.
- This is because the size of integer (variable of type int) is 4 bytes in 64-bit system.

Pointers

- A pointer is the memory address of a variable.
- A **pointer** is a variable that contains the address of a variable.
- Using pointer we can pass argument to the functions. Generally we pass them by value as a copy. So we cannot change them. But if we pass argument using pointer, we can modify them
- Let us imagine that a computer memory is a long array and every array location has a distinct memory location.
- A pointer variable contains a representation of an address of another variable (P is a pointer variable in the following):



Sitesbay.com

Example: (pointer declarations)

`float *p;` //To declare a pointer variable `p` that can "point" to a variable of type `float`

`int *K;` //To declare a pointer variable `K` that can "point" to a variable of type `int`

Example: If a number variable is stored in the memory address `0x123`, and it contains a value `5`.

- The **reference (&)** operator gives the value `0x123`, while the **dereference (*)** operator gives the value `5`.

Pointer Variable Definition

Syntax: *Type *Name;*

Examples:

```
int *P;           // P is variable that can point to an integer var
float *Q;         // Q is a float pointer
char *R;          // R is a char pointer
```

Example:

```
int *AP[5];       /* AP is an array of 5 pointers to ints */
```

Address (&) Operator

- An address used to tell where a variable is stored in memory is a pointer
- Pointer variables must be declared to have a pointer type
- **Reference operator (&)** as discussed above gives the address of a variable.

The Dereferencing Operator

To get the value stored in the memory address, we use the **dereference operator (*)**.

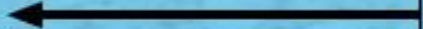
Example: `p1 = &v1;`

- `p1` is now a pointer to `v1`
- `v1` can be called `v1` or "the variable pointed to by `p1`"

Example 2:

```
■ v1 = 0;  
  p1 = &v1;  
  *p1 = 42;  
  cout << v1 << endl;  
  cout << *p1 << endl;
```

v1 and *p1 now refer to the same variable



output:

42

42

Example 3: C++ Pointers

C++ Program to demonstrate the working of pointer.

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

    c = 5;
    cout << "Address of c (&c): " << &c << endl;
    cout << "Value of c (c): " << c << endl << endl;

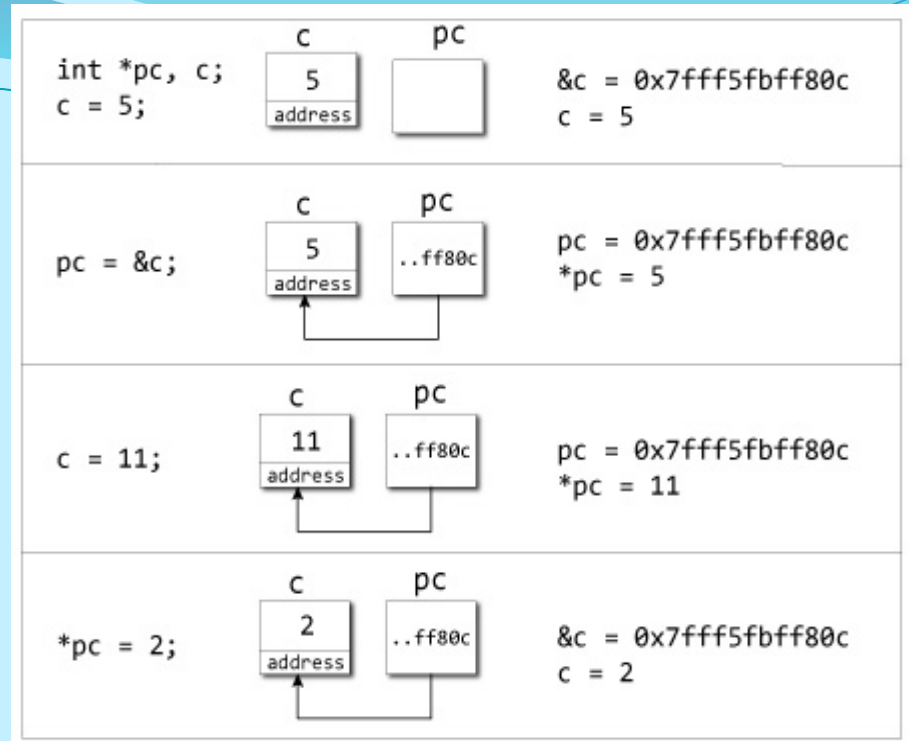
    pc = &c; // Pointer pc holds the memory address of variable c
    cout << "Address that pointer pc holds (pc): " << pc << endl;
    cout << "Content of the address pointer pc holds (*pc): " << *pc << endl;

    c = 11; // The content inside memory address &c is changed from 5 to 11.
    cout << "Address pointer pc holds (pc): " << pc << endl;
    cout << "Content of the address pointer pc holds (*pc): " << *pc << endl;

    *pc = 2;
    cout << "Address of c (&c): " << &c << endl;
    cout << "Value of c (c): " << c << endl << endl;

    return 0;
}
```


The output for Example 3



Address of c (&c): 0x7fff5fbff80c

Value of c (c): 5

Address that pointer pc holds (pc): 0x7fff5fbff80c

Content of the address pointer pc holds (*pc): 5

Address pointer pc holds (pc): 0x7fff5fbff80c

Content of the address pointer pc holds (*pc): 11

Address of c (&c): 0x7fff5fbff80c

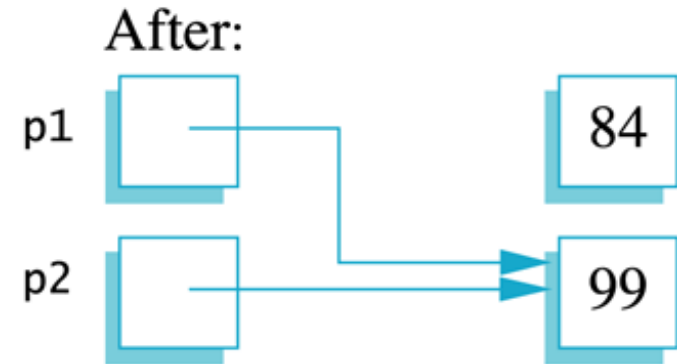
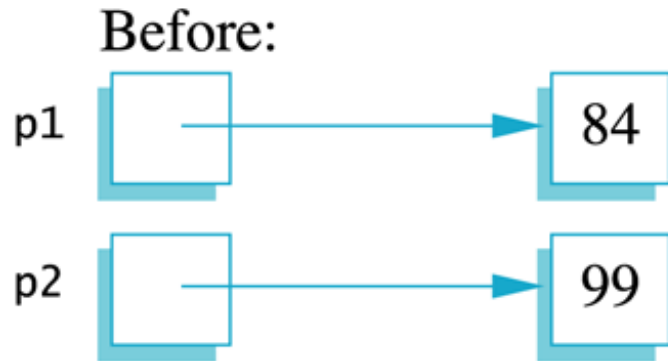
Value of c (c): 2

Pointer Assignment

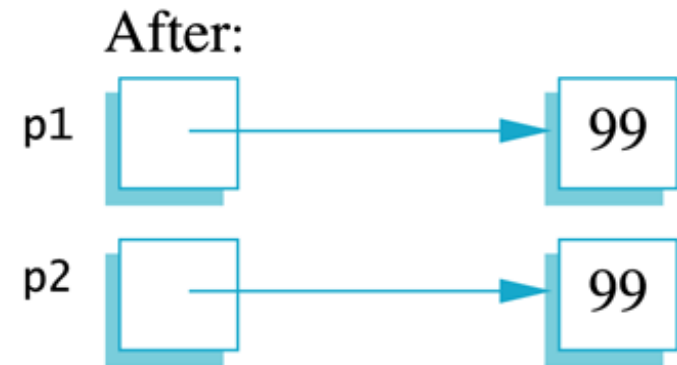
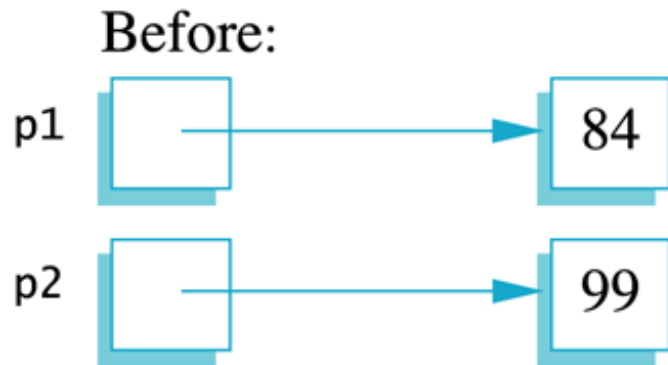
- The assignment operator = is used to assign the value of one pointer to another
 - Example: If p1 still points to v1 (previous slide) then
 $p2 = p1;$
 - causes *p2, *p1, and v1 all to name the same variable
- Some care is required making assignments to pointer variables
 - $p1 = p2;$ // changes the location that p1 "points" to
 - $*p1 = *p2;$ // changes the value at the location that p1 "points" to

Uses of the Assignment Operator

`p1 = p2;`

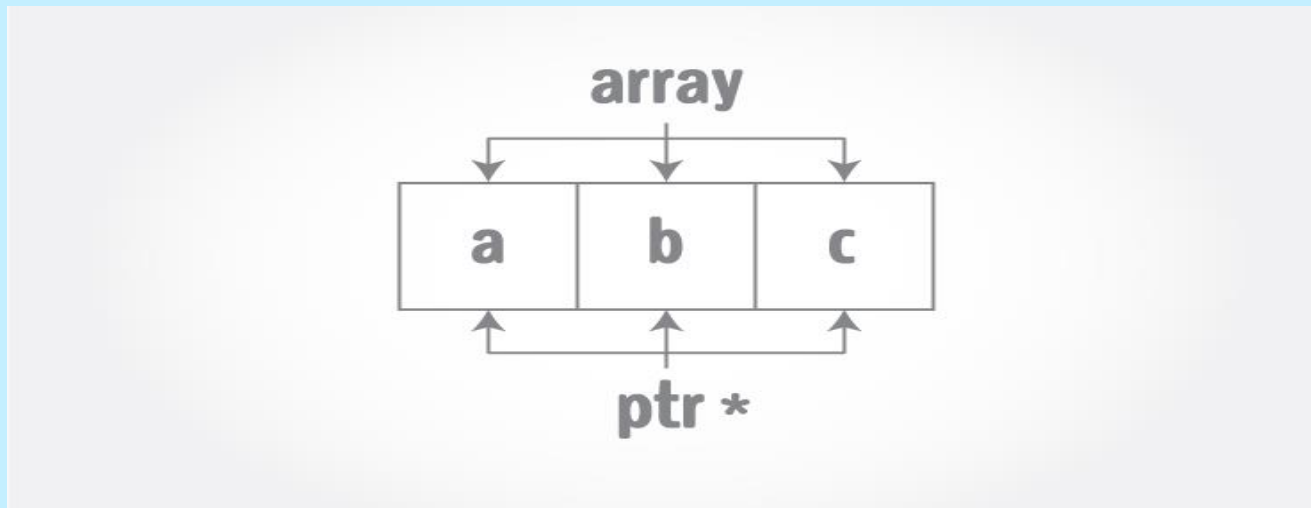


`*p1 = *p2;`



C++ Pointers and Arrays

- In this article, you'll learn about the relation between arrays and pointers, and use them efficiently in your program.



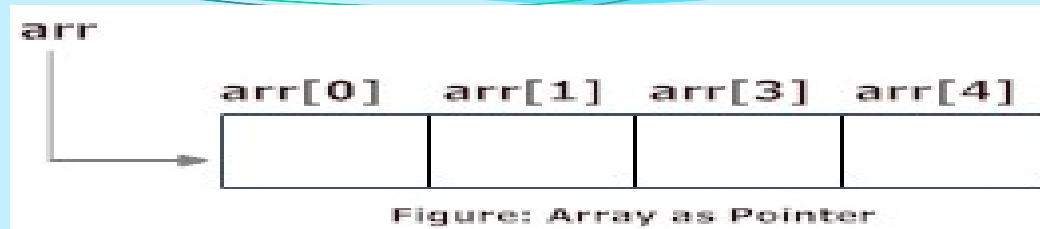
- Pointers are the variables that hold address. Not only can pointers store address of a single variable, it can also store address of cells of an array.

For example:

```
int *ptr;
```

```
int a[5];
```

```
Ptr = &a[2]; // &a[2] is the address of third element of a[5].
```



- Suppose, pointer needs to point to the fourth element of an array, that is, hold address of fourth array element in above case.
- Since **ptr** points to the third element in the above example, **ptr + 1** will point to the fourth element.
- You may think, **ptr + 1** gives you the address of next byte to the **ptr**. But it's not correct.
- This is because pointer **ptr** is a pointer to an **int** and **size of int is fixed for a operating system (size of int is 4 byte of 64-bit operating system)**. Hence, the address between **ptr** and **ptr + 1** differs by 4 bytes.
- If pointer **ptr** was pointer to **char** then, **the address between ptr and ptr + 1 would have differed by 1 byte since size of a character is 1 byte.**

Example 4: C++ Pointers and Arrays

C++ Program to display address of elements of an array using both array and pointers

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

int main()
{
    float arr[5];
    float *ptr;

    cout << "Displaying address using arrays: " << endl;
    for (int i = 0; i < 5; ++i)
    {
        cout << "&arr[" << i << "] = " << &arr[i] << endl;
    }

    // ptr = &arr[0]
    ptr = arr;

    cout << "\nDisplaying address using pointers: " << endl;
    for (int i = 0; i < 5; ++i)
    {
        cout << "ptr + " << i << " = " << ptr + i << endl;
    }

    return 0;
}
```

Output of Example 4:

Displaying address using arrays:

&arr[0] = 0x7fff5bfff880

&arr[1] = 0x7fff5bfff884

&arr[2] = 0x7fff5bfff888

&arr[3] = 0x7fff5bfff88c

&arr[4] = 0x7fff5bfff890

Displaying address using pointers:

ptr + 0 = 0x7fff5bfff880

ptr + 1 = 0x7fff5bfff884

ptr + 2 = 0x7fff5bfff888

ptr + 3 = 0x7fff5bfff88c

ptr + 4 = 0x7fff5bfff890

توضيح

In the above program, a different pointer **ptr** is used for displaying the address of array elements **arr**.

But, array elements can be accessed using pointer notation by using same array name **arr**.