Data Structures & Algorithms

Prepared by: Mohamed Ayman

Machine Learning Researcher

spring 2019

sw.eng.MohamedAyman@gmail.com

facebook.com/sw.eng.MohamedAyman

linkedin.com/in/eng-mohamed-ayman

codeforces.com/profile/Mohamed_Ayman

Pointers

References

Agenda

Let's
START UP

Agenda

1- Definition of Pointers

2- Importance of pointers

3- Types of Pointers

4- Incrementing vs. Decrementing Pointers

5- Pointer Comparisons

6- Pointers vs. Arrays

7- Array of Pointers

8- Pointer to a pointer

9- Reference Variables

Definition of Pointers

- A pointer is a variable whose value is the address of another variable.

Like any variable or constant, you must declare a pointer before you can work with it.

- The asterisk you used to declare a pointer is the same asterisk that you use for multiplication. However, in this statement the asterisk is being used to designate a variable as a pointer. Following are the valid pointer declaration:

Definition of Pointers

- There are few important operations, which we will do with the pointers very frequently.

(a) We define a pointer variable.

(b) Assign the address of a variable to a pointer.

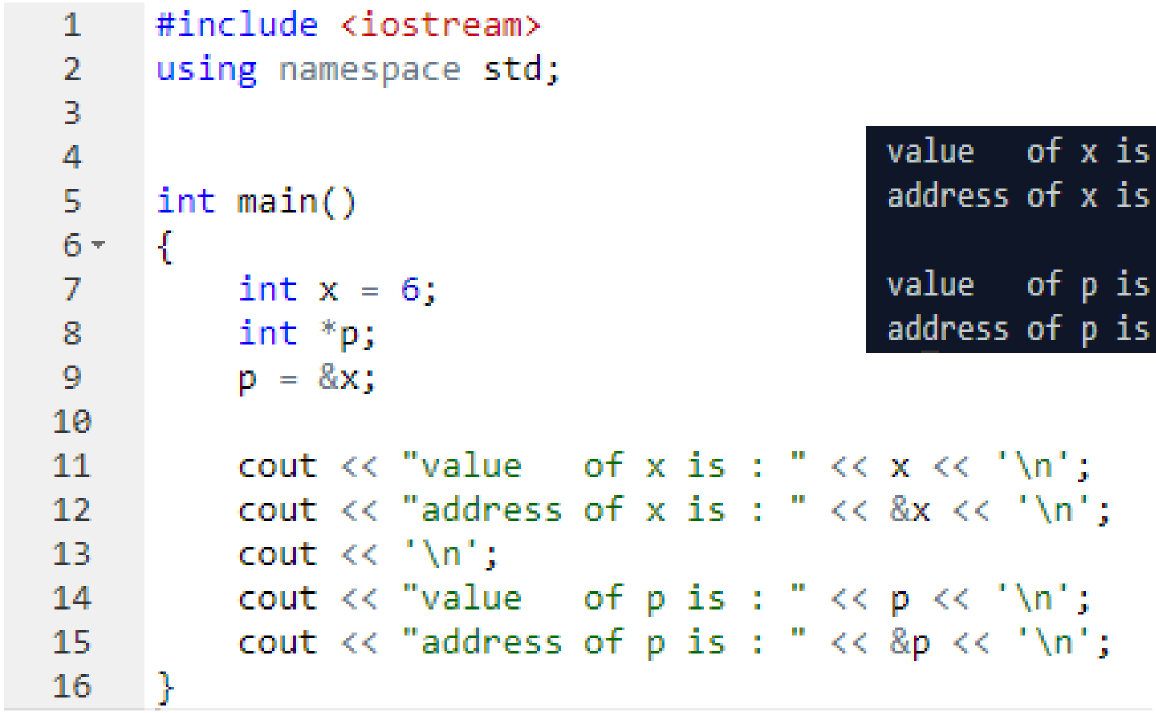
(c) Finally access the value at the address available in the pointer variable.

- This is done by using unary operator * that returns the value of the variable located at the address specified by its operand.

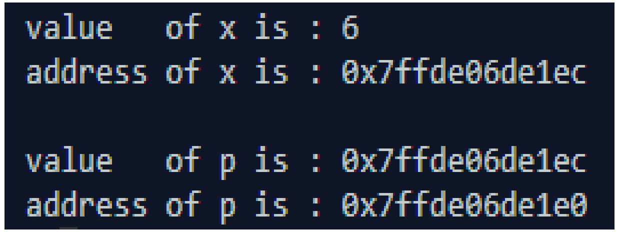
- The asterisk you used to declare a pointer is the same asterisk that you use for multiplication. However, in this statement the asterisk is being used to designate a variable as a pointer. Following are the valid pointer declaration:

Definition of Pointers

Link: repl.it/repls/HighNegativeDesertpupfish



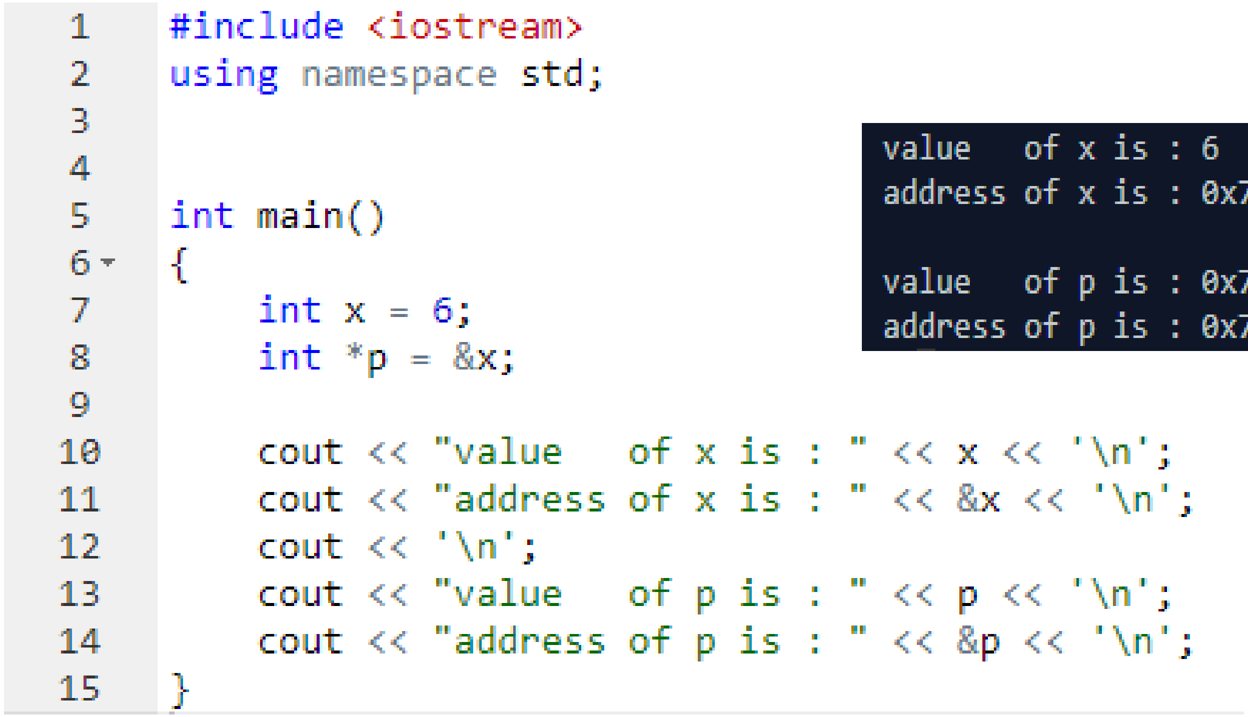
```cpp
#include <iostream>
using namespace std;


int main()
{
    int x = 6;
    int *p;
    p = &x;

    cout << "value   of x is : " << x << '\n';
    cout << "address of x is : " << &x << '\n';
    cout << '\n';
    cout << "value   of p is : " << p << '\n';
    cout << "address of p is : " << &p << '\n';
}
```



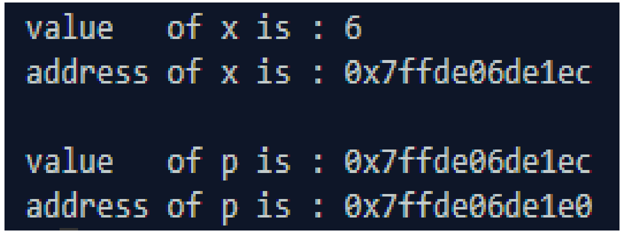
```
value   of x is : 6
address of x is : 0x7ffde06de1ec

value   of p is : 0x7ffde06de1ec
address of p is : 0x7ffde06de1e0
```

Definition of Pointers

Link: repl.it/repls/NoxiousInfatuatedStoat



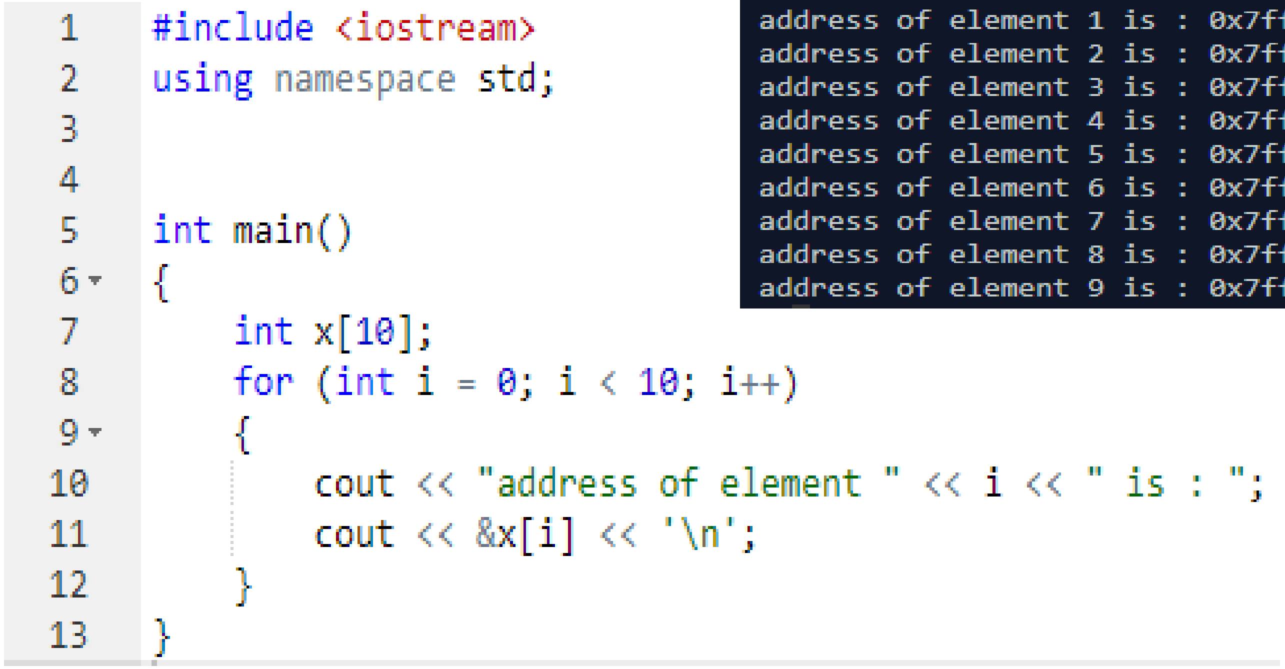
```cpp
#include <iostream>
using namespace std;


int main()
{
    int x = 6;
    int *p = &x;

    cout << "value   of x is : " << x << '\n';
    cout << "address of x is : " << &x << '\n';
    cout << '\n';
    cout << "value   of p is : " << p << '\n';
    cout << "address of p is : " << &p << '\n';
}
```



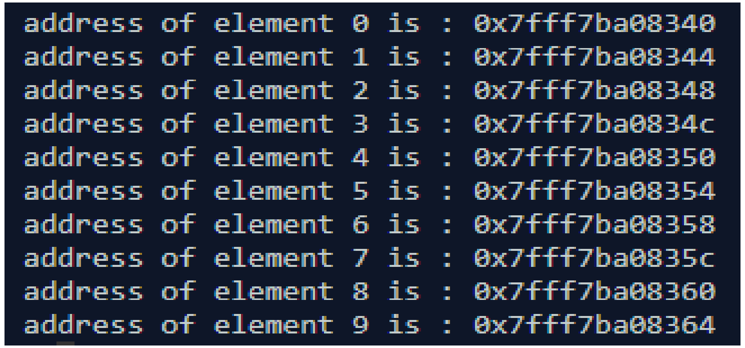
```
value   of x is : 6
address of x is : 0x7ffde06de1ec

value   of p is : 0x7ffde06de1ec
address of p is : 0x7ffde06de1e0
```

Variables address Example

Link: repl.it/repls/MisguidedPapayawhipBull



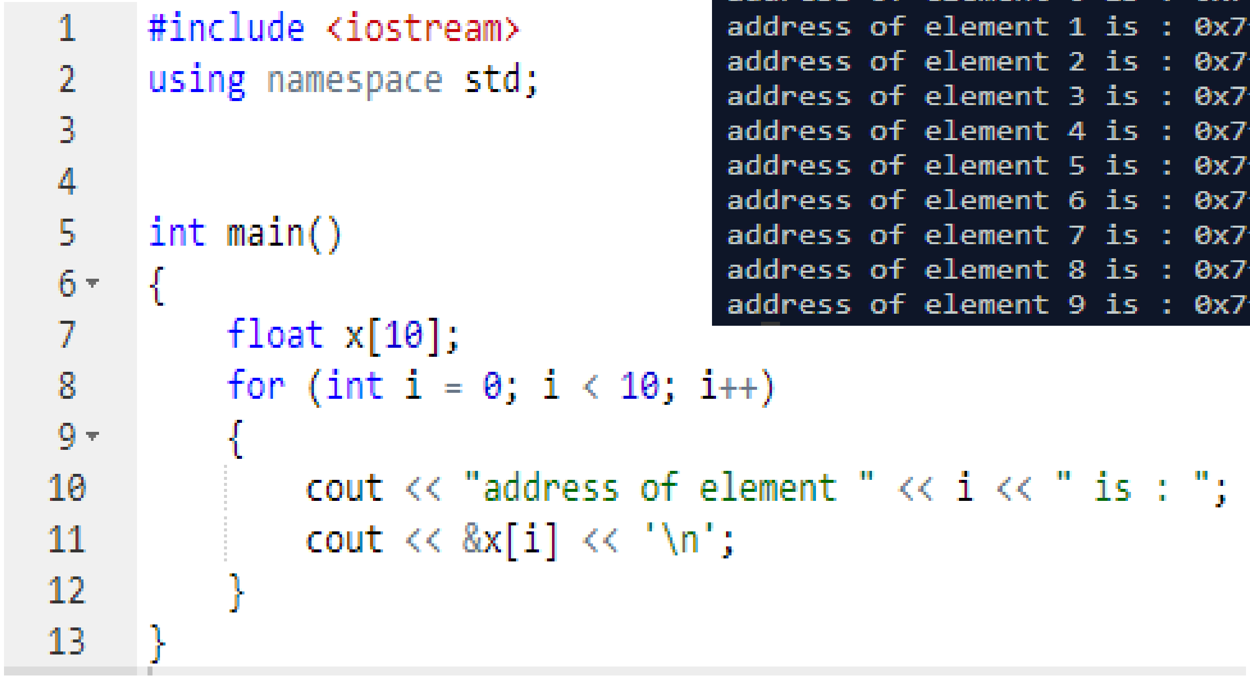
```cpp
#include <iostream>
using namespace std;


int main()
{
    int x[10];
    for (int i = 0; i < 10; i++)
    {
        cout << "address of element " << i << " is : ";
        cout << &x[i] << '\n';
    }
}
```



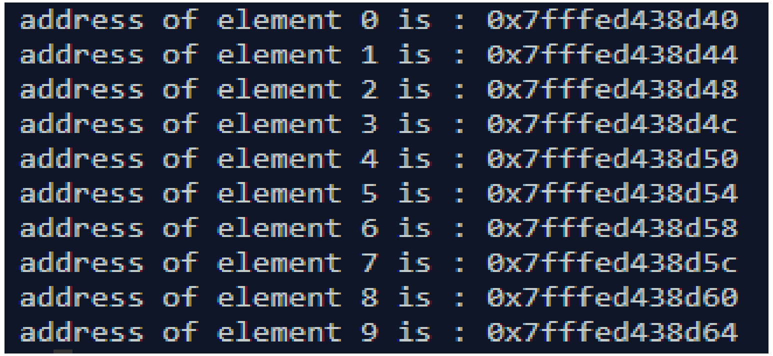
```
address of element 0 is : 0x7fff7ba08340
address of element 1 is : 0x7fff7ba08344
address of element 2 is : 0x7fff7ba08348
address of element 3 is : 0x7fff7ba0834c
address of element 4 is : 0x7fff7ba08350
address of element 5 is : 0x7fff7ba08354
address of element 6 is : 0x7fff7ba08358
address of element 7 is : 0x7fff7ba0835c
address of element 8 is : 0x7fff7ba08360
address of element 9 is : 0x7fff7ba08364
```

Variables address Example

Link: repl.it/repls/DryUtterIndianpalmsquirrel



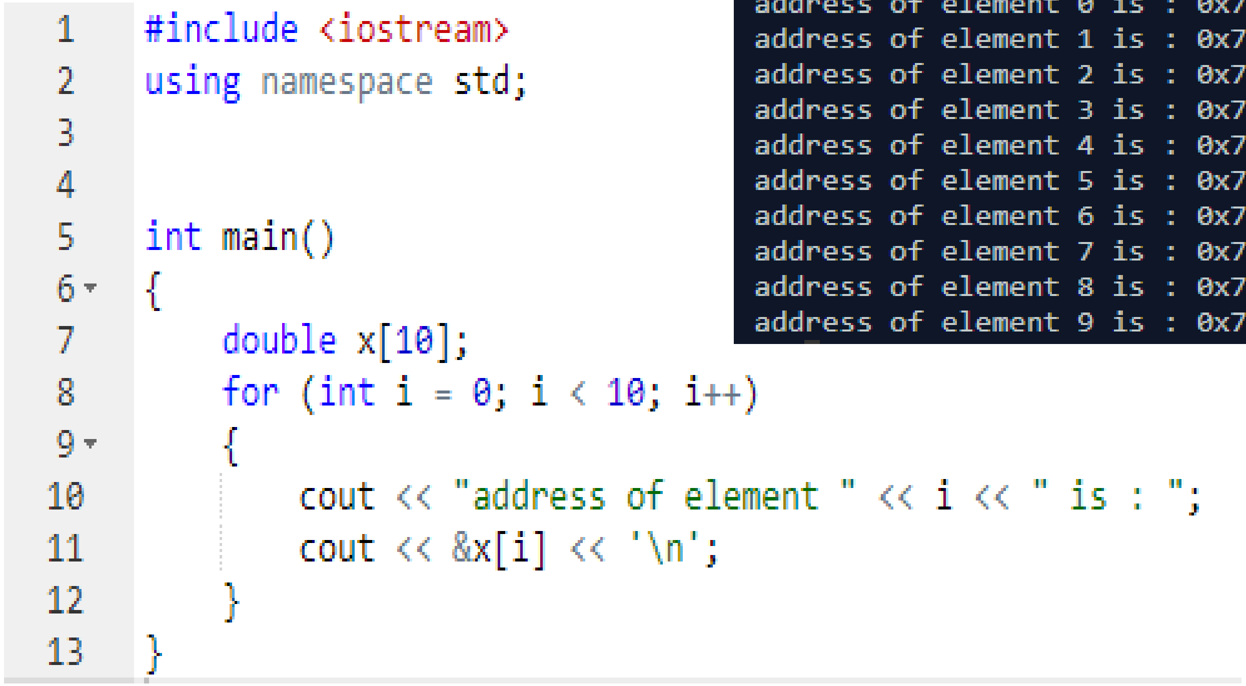
```cpp
#include <iostream>
using namespace std;


int main()
{
    float x[10];
    for (int i = 0; i < 10; i++)
    {
        cout << "address of element " << i << " is : ";
        cout << &x[i] << '\n';
    }
}
```



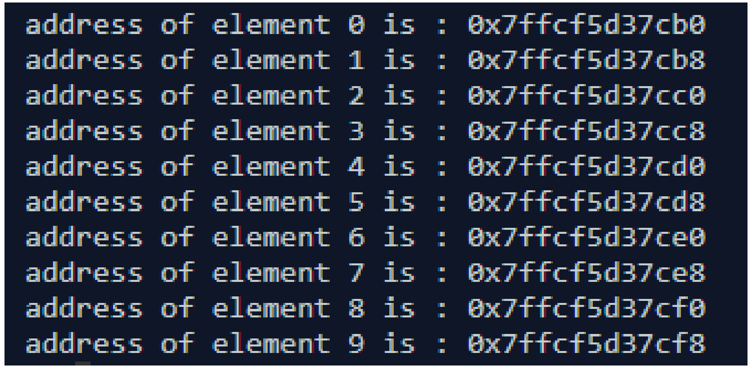
```
address of element 0 is : 0x7fffed438d40
address of element 1 is : 0x7fffed438d44
address of element 2 is : 0x7fffed438d48
address of element 3 is : 0x7fffed438d4c
address of element 4 is : 0x7fffed438d50
address of element 5 is : 0x7fffed438d54
address of element 6 is : 0x7fffed438d58
address of element 7 is : 0x7fffed438d5c
address of element 8 is : 0x7fffed438d60
address of element 9 is : 0x7fffed438d64
```

Variables address Example

Link: repl.it/repls/PalegreenWelcomeRoach



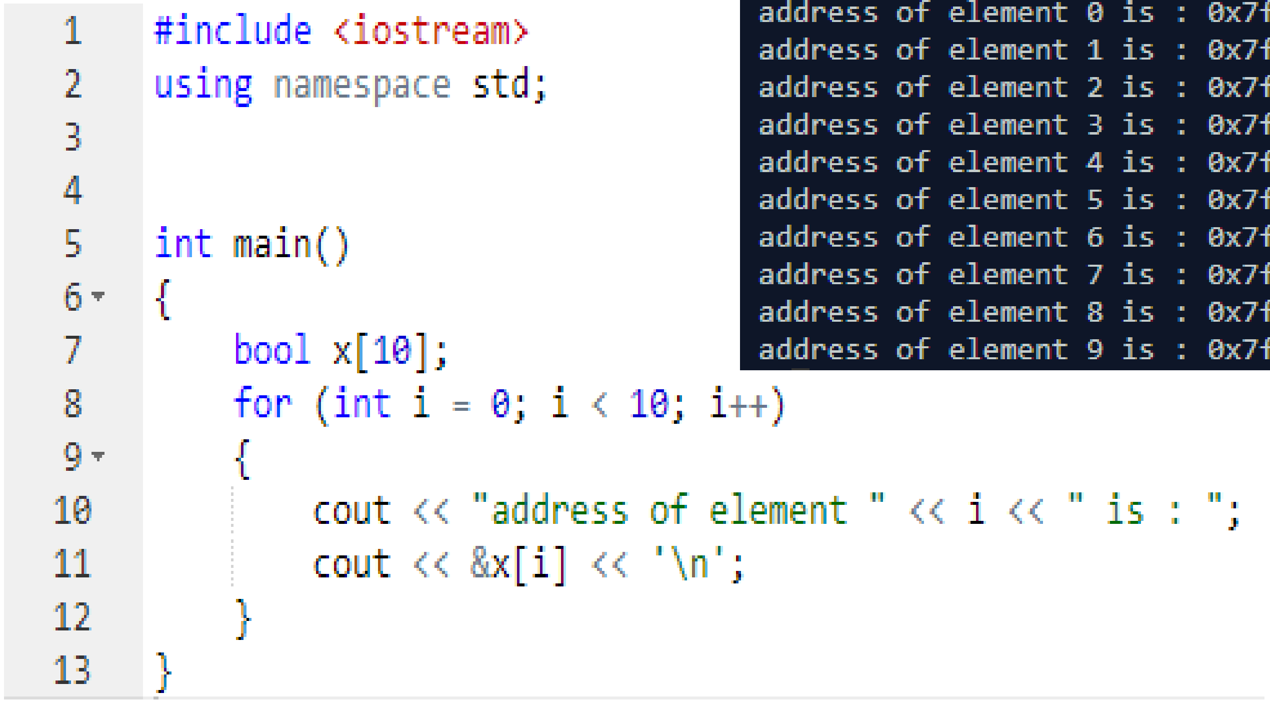
```cpp
#include <iostream>
using namespace std;


int main()
{
    double x[10];
    for (int i = 0; i < 10; i++)
    {
        cout << "address of element " << i << " is : ";
        cout << &x[i] << '\n';
    }
}
```



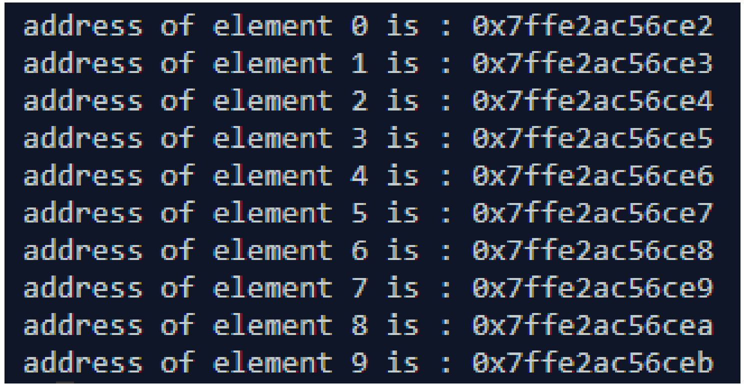
```
address of element 0 is : 0x7ffcf5d37cb0
address of element 1 is : 0x7ffcf5d37cb8
address of element 2 is : 0x7ffcf5d37cc0
address of element 3 is : 0x7ffcf5d37cc8
address of element 4 is : 0x7ffcf5d37cd0
address of element 5 is : 0x7ffcf5d37cd8
address of element 6 is : 0x7ffcf5d37ce0
address of element 7 is : 0x7ffcf5d37ce8
address of element 8 is : 0x7ffcf5d37cf0
address of element 9 is : 0x7ffcf5d37cf8
```

Variables address Example

Link: repl.it/repls/RemorsefulPhonyBudgie



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


int main()
{
    bool x[10];
    for (int i = 0; i < 10; i++)
    {
        cout << "address of element " << i << " is : ";
        cout << &x[i] << '\n';
    }
}
```



```
address of element 0 is : 0x7ffe2ac56ce2
address of element 1 is : 0x7ffe2ac56ce3
address of element 2 is : 0x7ffe2ac56ce4
address of element 3 is : 0x7ffe2ac56ce5
address of element 4 is : 0x7ffe2ac56ce6
address of element 5 is : 0x7ffe2ac56ce7
address of element 6 is : 0x7ffe2ac56ce8
address of element 7 is : 0x7ffe2ac56ce9
address of element 8 is : 0x7ffe2ac56cea
address of element 9 is : 0x7ffe2ac56ceb
```

Agenda

✓ 1- Definition of Pointers

2- Importance of pointers

3- Types of Pointers

4- Incrementing vs. Decrementing Pointers

5- Pointer Comparisons

6- Pointers vs. Arrays

7- Array of Pointers

8- Pointer to a pointer

9- Reference Variables

Importance of pointers

- Some C++ tasks are performed more easily with pointers, and other C++ tasks, such as dynamic memory allocation, cannot be performed without them.

- As you know every variable is a memory location and every memory location has its address defined which can be accessed using ampersand (&) operator which denotes an address in memory.

Agenda

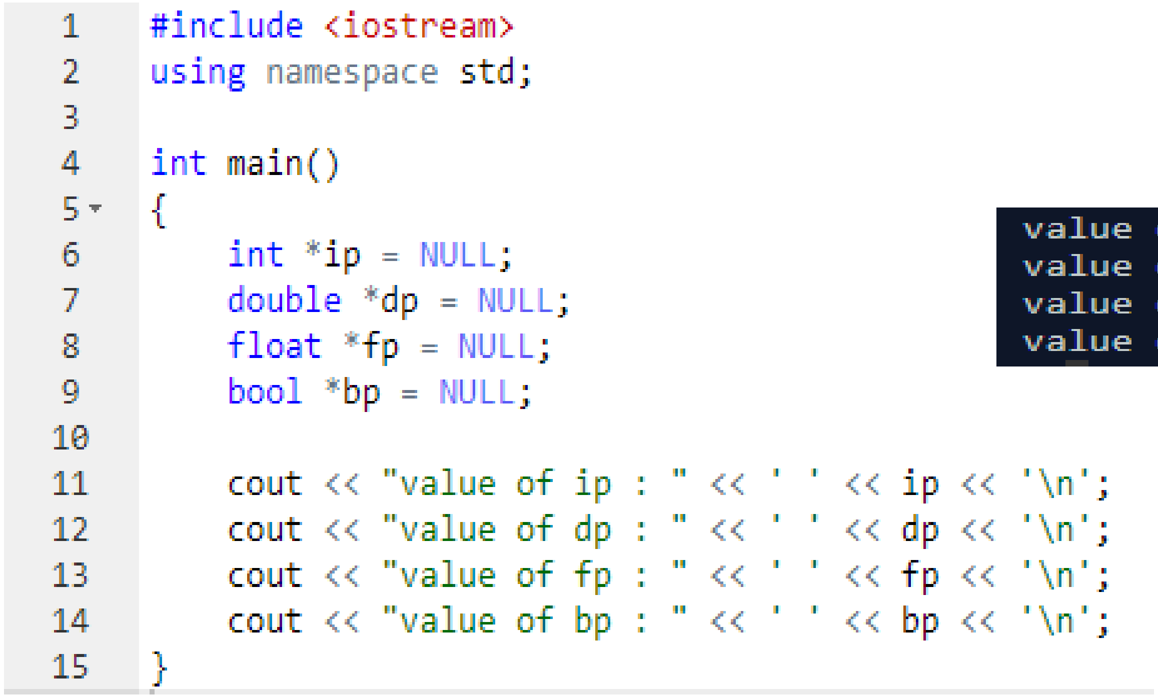
Types of Pointers

- The actual data type of the value of all pointers, whether integer, float, character, or otherwise, is the same, a long hexadecimal number that represents a memory address.

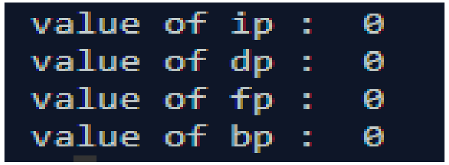
- The only difference between pointers of different data types is the data type of the variable or constant that the pointer points to.

Types of Pointers

Link: repl.it/repls/LavenderNiceCoypu



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

int main()
{
    int *ip = NULL;
    double *dp = NULL;
    float *fp = NULL;
    bool *bp = NULL;

    cout << "value of ip : " << ' ' << ip << '\n';
    cout << "value of dp : " << ' ' << dp << '\n';
    cout << "value of fp : " << ' ' << fp << '\n';
    cout << "value of bp : " << ' ' << bp << '\n';
}
```



```
value of ip :  0
value of dp :  0
value of fp :  0
value of bp :  0
```

Types of Pointers - Null Pointers

- It is always a good practice to assign the pointer NULL to a pointer variable in case you do not have exact address to be assigned.

- This is done at the time of variable declaration. A pointer that is assigned NULL is called a null pointer.

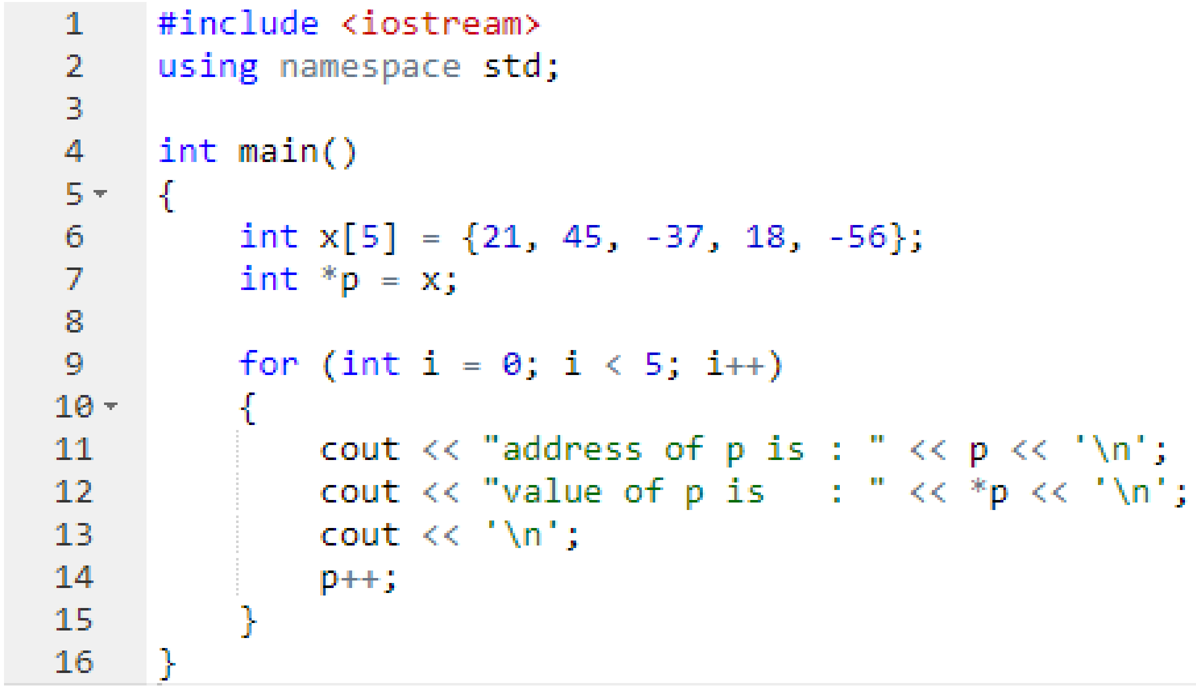
- The NULL pointer is a constant with a value of zero defined in several standard libraries, including ‹iostream›

Agenda

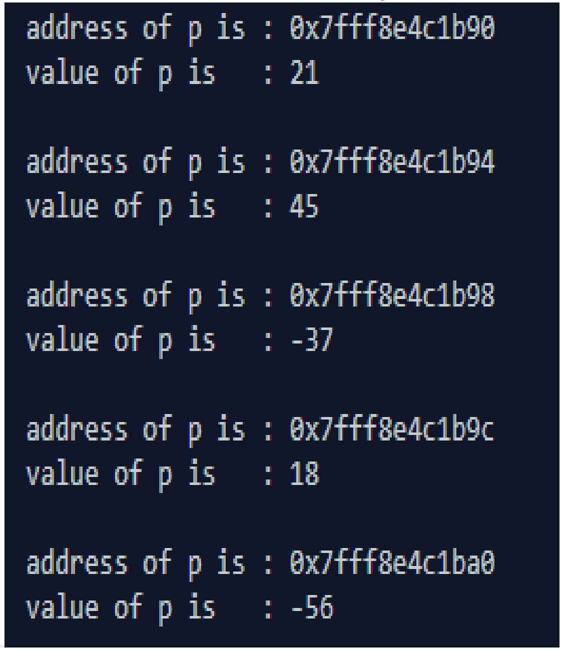
- ✓ 1- Definition of Pointers
- ✓ 2- Importance of pointers
- ✓ 3- Types of Pointers
- 4- Incrementing vs. Decrementing Pointers
- 5- Pointer Comparisons
- 6- Pointers vs. Arrays
- 7- Array of Pointers
- 8- Pointer to a pointer
- 9- Reference Variables

Incrementing vs. Decrementing Pointers

Link: repl.it/repls/ScentedUnlinedUrus



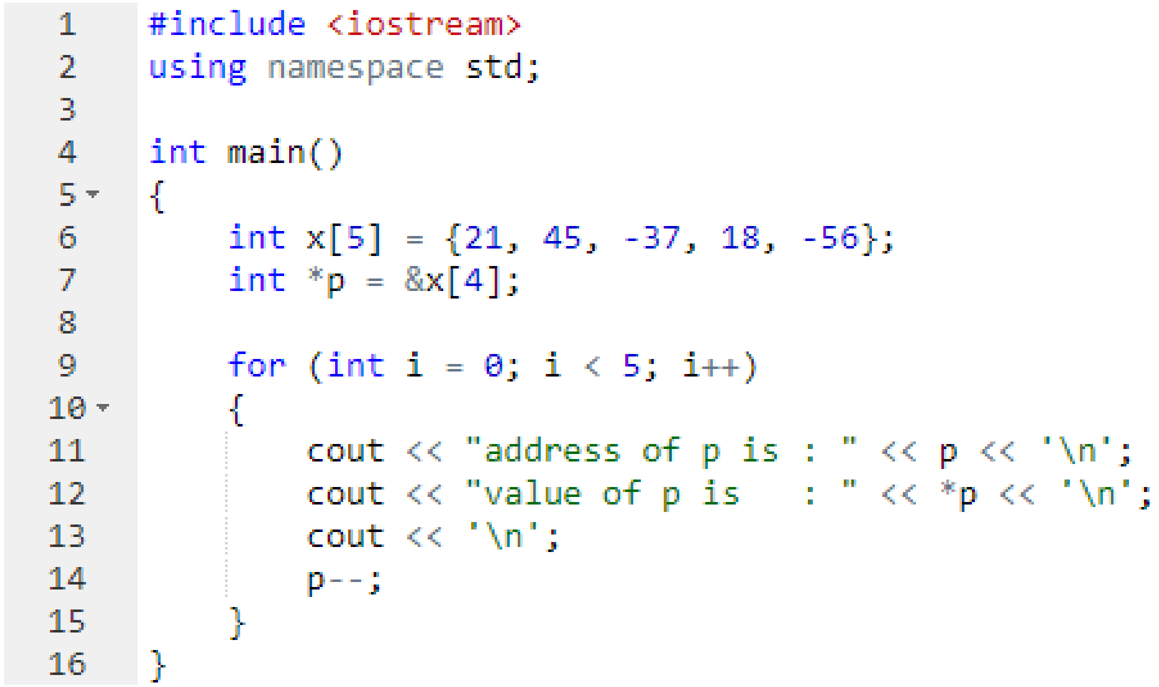
```cpp
#include <iostream>
using namespace std;

int main()
{
    int x[5] = {21, 45, -37, 18, -56};
    int *p = x;

    for (int i = 0; i < 5; i++)
    {
        cout << "address of p is : " << p << '\n';
        cout << "value of p is   : " << *p << '\n';
        cout << '\n';
        p++;
    }
}
```



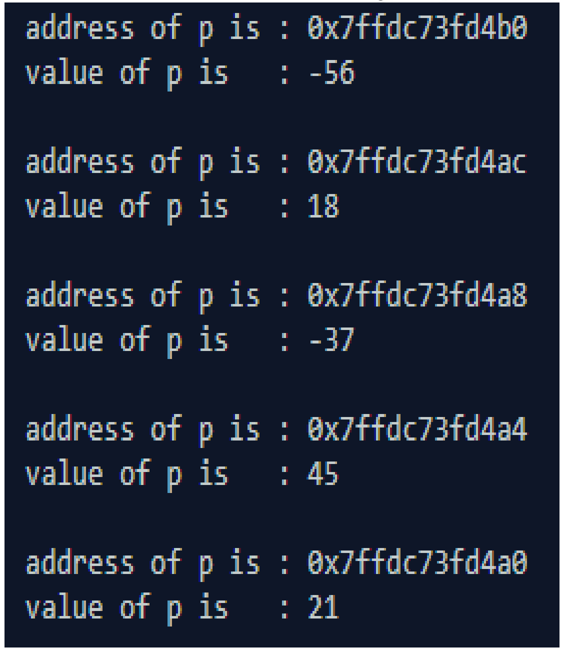
```
address of p is : 0x7fff8e4c1b90
value of p is   : 21

address of p is : 0x7fff8e4c1b94
value of p is   : 45

address of p is : 0x7fff8e4c1b98
value of p is   : -37

address of p is : 0x7fff8e4c1b9c
value of p is   : 18

address of p is : 0x7fff8e4c1ba0
value of p is   : -56
```

Incrementing vs. Decrementing Pointers

Link: repl.it/repls/ForsakenTwinPseudodynerusquadrisectus



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

int main()
{
    int x[5] = {21, 45, -37, 18, -56};
    int *p = &x[4];

    for (int i = 0; i < 5; i++)
    {
        cout << "address of p is : " << p << '\n';
        cout << "value of p is   : " << *p << '\n';
        cout << '\n';
        p--;
    }
}
```



```
address of p is : 0x7ffdc73fd4b0
value of p is   : -56

address of p is : 0x7ffdc73fd4ac
value of p is   : 18

address of p is : 0x7ffdc73fd4a8
value of p is   : -37

address of p is : 0x7ffdc73fd4a4
value of p is   : 45

address of p is : 0x7ffdc73fd4a0
value of p is   : 21
```

Agenda

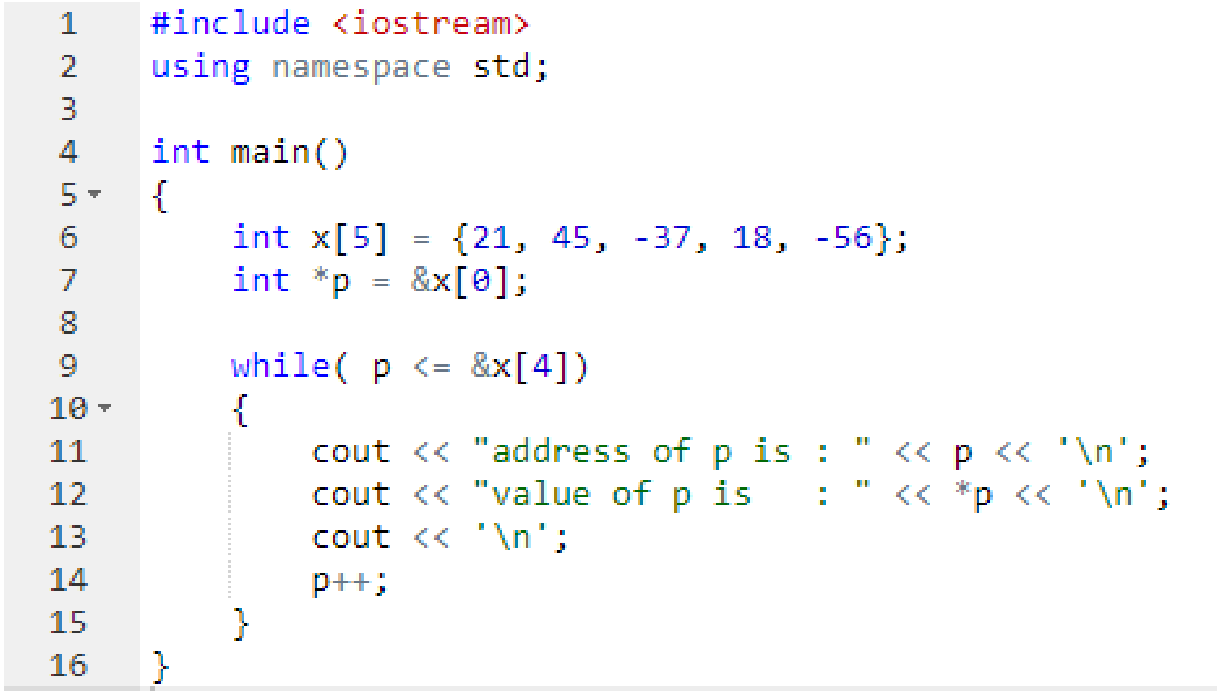
Pointer Comparisons

- Pointers may be compared by using relational operators, such as ==, !=, <, <=, >, >=

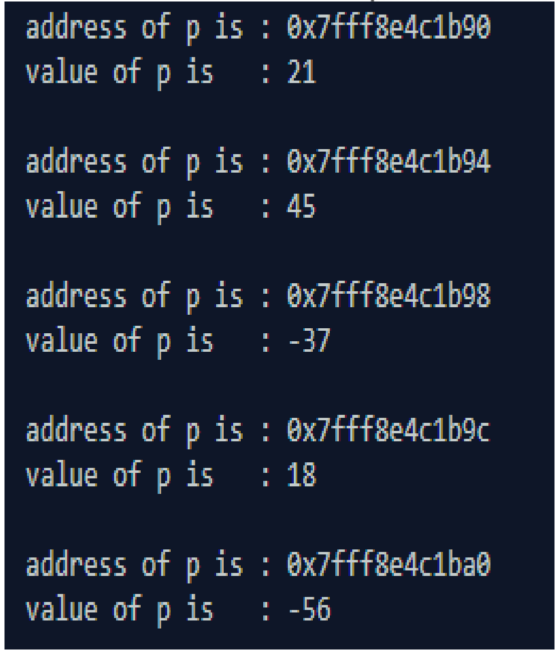
- If p1 and p2 point to variables that are related to each other, such as elements of the same array, then p1 and p2 can be meaningfully compared.

Pointer Comparisons

Link: repl.it/repls/IvoryLightyellowNakedmolerat



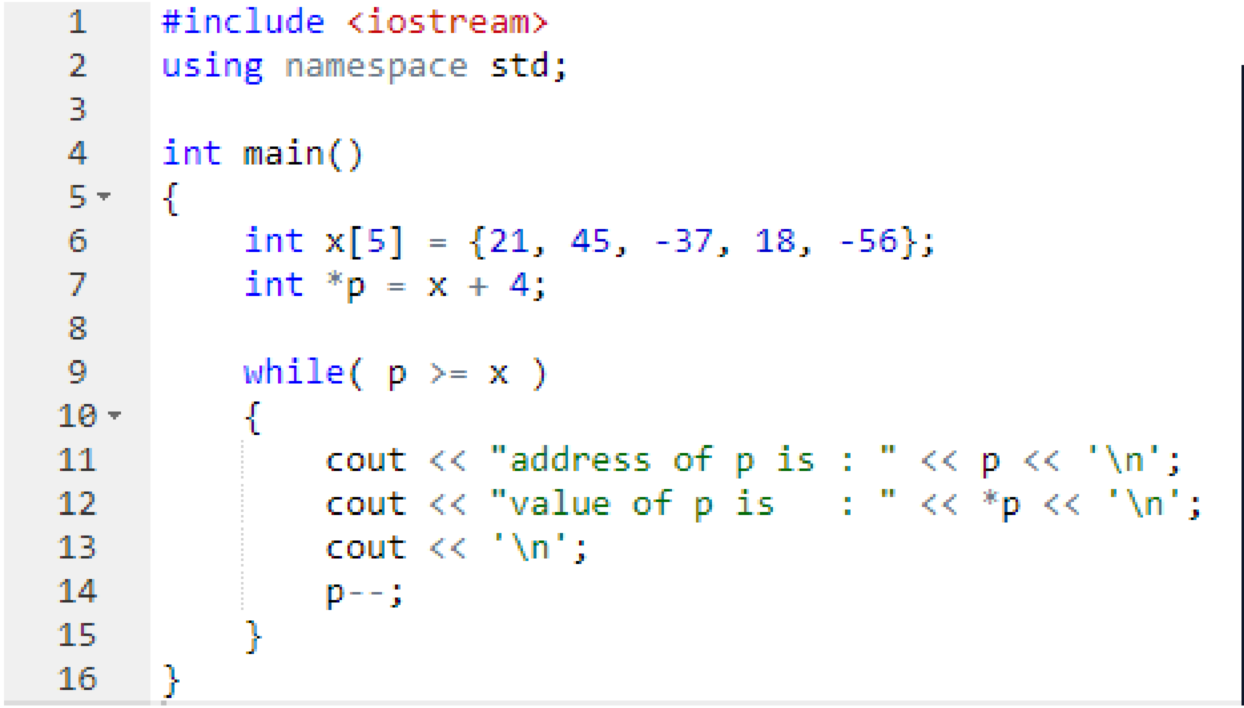
```cpp
#include <iostream>
using namespace std;

int main()
{
    int x[5] = {21, 45, -37, 18, -56};
    int *p = &x[0];

    while( p <= &x[4])
    {
        cout << "address of p is : " << p << '\n';
        cout << "value of p is   : " << *p << '\n';
        cout << '\n';
        p++;
    }
}
```



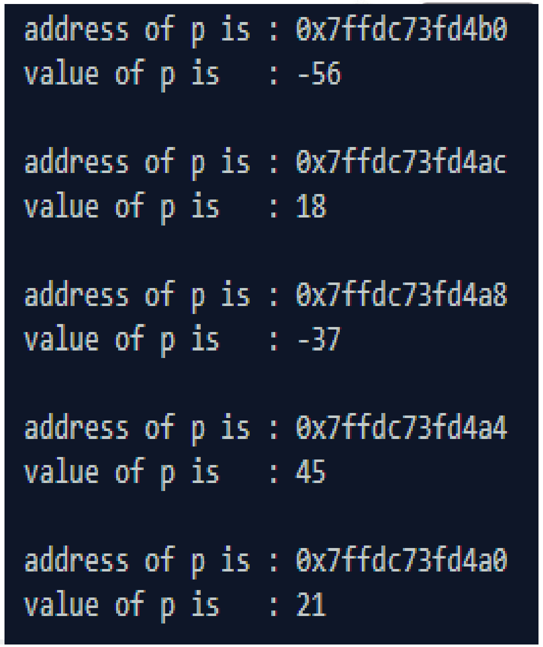
```
address of p is : 0x7fff8e4c1b90
value of p is   : 21

address of p is : 0x7fff8e4c1b94
value of p is   : 45

address of p is : 0x7fff8e4c1b98
value of p is   : -37

address of p is : 0x7fff8e4c1b9c
value of p is   : 18

address of p is : 0x7fff8e4c1ba0
value of p is   : -56
```

Pointer Comparisons

Link: repl.it/repls/GrandLavishMontanoceratops



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

int main()
{
    int x[5] = {21, 45, -37, 18, -56};
    int *p = x + 4;

    while( p >= x )
    {
        cout << "address of p is : " << p << '\n';
        cout << "value of p is   : " << *p << '\n';
        cout << '\n';
        p--;
    }
}
```



```
address of p is : 0x7ffdc73fd4b0
value of p is   : -56

address of p is : 0x7ffdc73fd4ac
value of p is   : 18

address of p is : 0x7ffdc73fd4a8
value of p is   : -37

address of p is : 0x7ffdc73fd4a4
value of p is   : 45

address of p is : 0x7ffdc73fd4a0
value of p is   : 21
```

Agenda

- ✓ 1- Definition of Pointers
- ✓ 2- Importance of pointers
- ✓ 3- Types of Pointers
- ✓ 4- Incrementing vs. Decrementing Pointers
- ✓ 5- Pointer Comparisons
- 6- Pointers vs. Arrays
- 7- Array of Pointers
- 8- Pointer to a pointer
- 9- Reference Variables

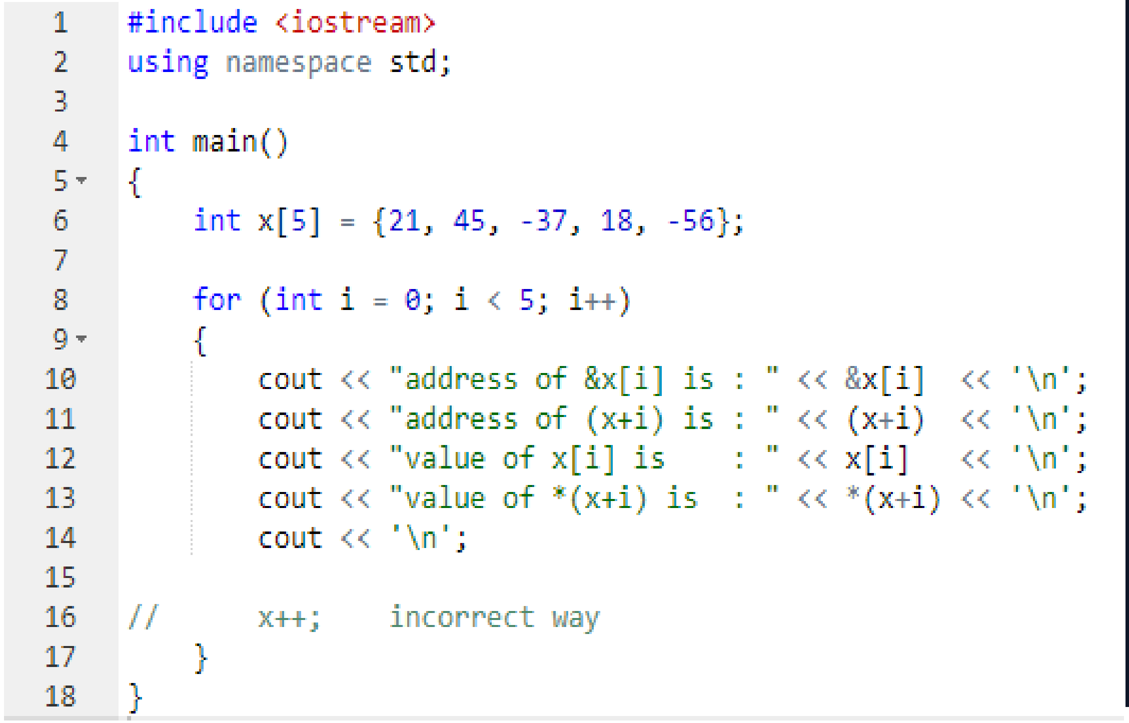
Pointers vs. Arrays

- Pointers and arrays are strongly related. In fact, pointers and arrays are interchangeable in many cases.

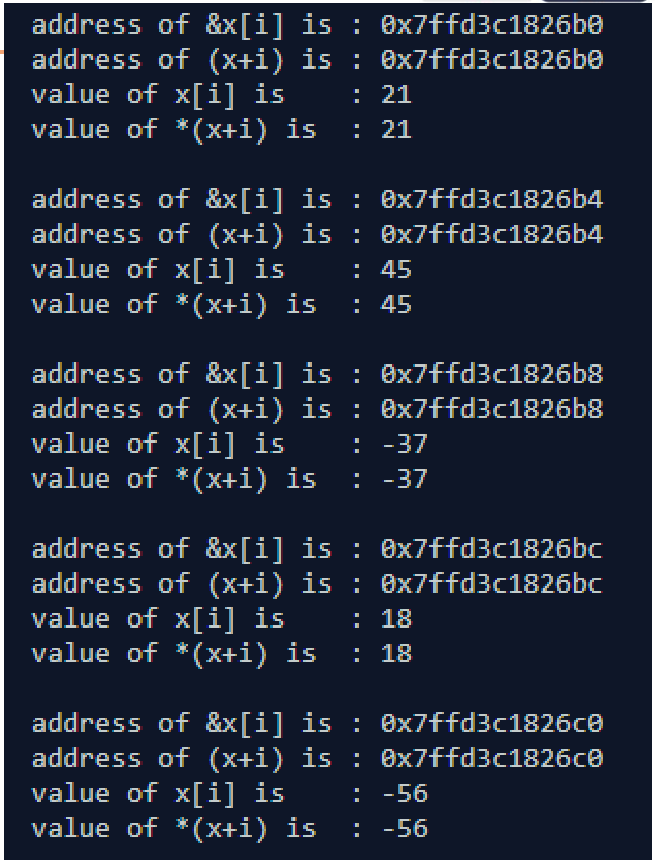
- For example, a pointer that points to the beginning of an array can access that array by using either pointer arithmetic or array-style indexing.

Pointers vs. Arrays

Link: repl.it/repls/UnwieldyResponsibleAntarcticgiantpetrel



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

int main()
{
    int x[5] = {21, 45, -37, 18, -56};

    for (int i = 0; i < 5; i++)
    {
        cout << "address of &x[i] is : " << &x[i]  << '\n';
        cout << "address of (x+i) is : " << (x+i)  << '\n';
        cout << "value of x[i] is    : " << x[i]   << '\n';
        cout << "value of *(x+i) is  : " << *(x+i) << '\n';
        cout << '\n';

//      x++;    incorrect way
    }
}
```



```
address of &x[i] is : 0x7ffd3c1826b0
address of (x+i) is : 0x7ffd3c1826b0
value of x[i] is    : 21
value of *(x+i) is  : 21

address of &x[i] is : 0x7ffd3c1826b4
address of (x+i) is : 0x7ffd3c1826b4
value of x[i] is    : 45
value of *(x+i) is  : 45

address of &x[i] is : 0x7ffd3c1826b8
address of (x+i) is : 0x7ffd3c1826b8
value of x[i] is    : -37
value of *(x+i) is  : -37

address of &x[i] is : 0x7ffd3c1826bc
address of (x+i) is : 0x7ffd3c1826bc
value of x[i] is    : 18
value of *(x+i) is  : 18

address of &x[i] is : 0x7ffd3c1826c0
address of (x+i) is : 0x7ffd3c1826c0
value of x[i] is    : -56
value of *(x+i) is  : -56
```

Pointers vs. Arrays

- It is perfectly acceptable to apply the pointer operator * to var but it is illegal to modify var value. The reason for this is that var is a constant that points to the beginning of an array and can not be used as l-value.

- Because an array name generates a pointer constant, it can still be used in pointer-style expressions, as long as it is not modified. For example, the following is a valid statement that assigns var [2] the value 500:

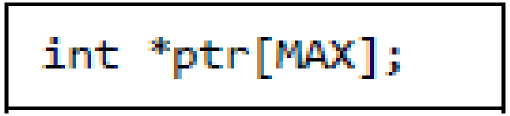
```
*(var + 2) = 500;
```

- Above statement is valid and will compile successfully because var is not changed.

Agenda

Array of Pointers

- There may be a situation, when we want to maintain an array, which can store pointers to an int or char or any other data type available. Following is the declaration of an array of pointers to an integer:

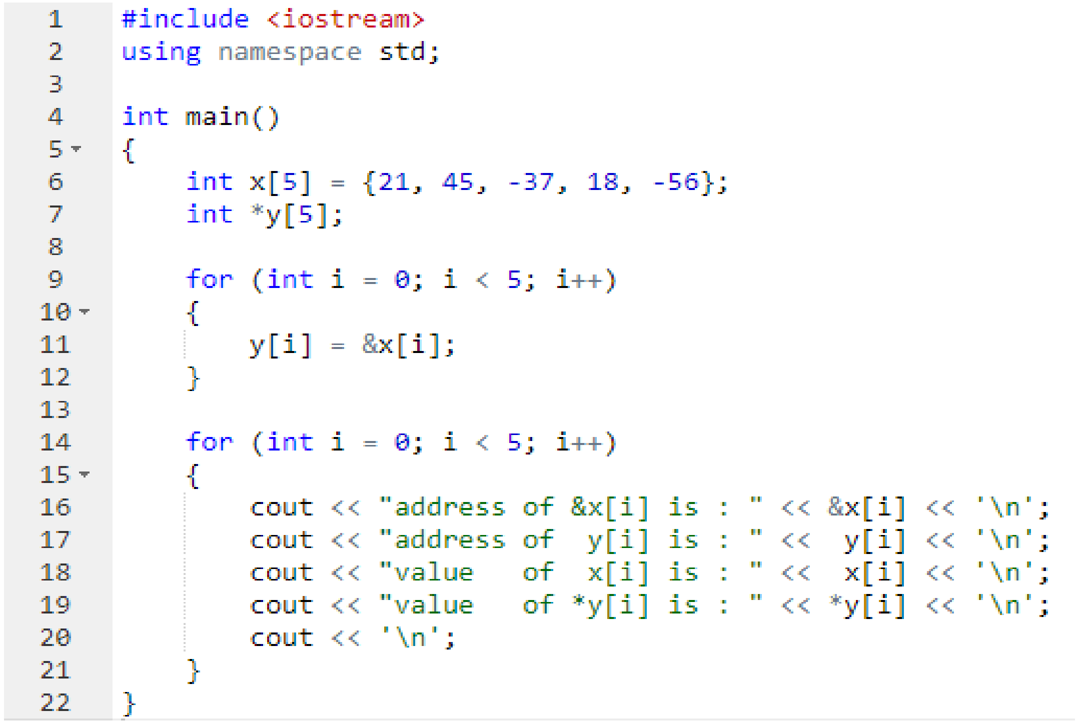


```
int *ptr[MAX];
```

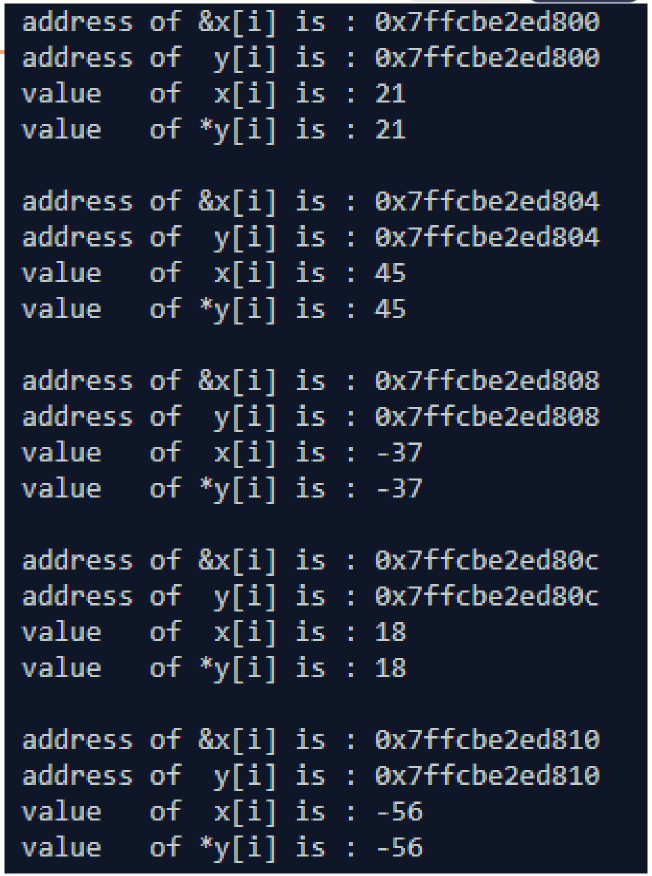
- This declares ptr as an array of 3 integer pointers. Thus, each element in ptr, now holds a pointer to an int value. Following example makes use of three integers which will be stored in an array of pointers as follows:

Array of Pointers

Link: repl.it/repls/LightCaringYorkshireterrier



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

int main()
{
    int x[5] = {21, 45, -37, 18, -56};
    int *y[5];

    for (int i = 0; i < 5; i++)
    {
        y[i] = &x[i];
    }

    for (int i = 0; i < 5; i++)
    {
        cout << "address of &x[i] is : " << &x[i] << '\n';
        cout << "address of  y[i] is : " <<  y[i] << '\n';
        cout << "value   of  x[i] is : " <<  x[i] << '\n';
        cout << "value   of *y[i] is : " << *y[i] << '\n';
        cout << '\n';
    }
}
```

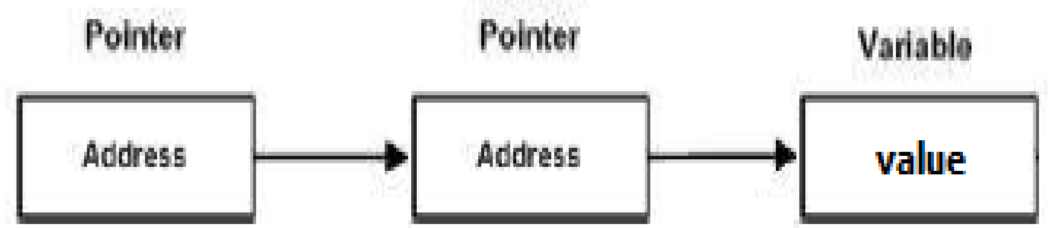


```
address of &x[i] is : 0x7ffcbe2ed800
address of  y[i] is : 0x7ffcbe2ed800
value   of  x[i] is : 21
value   of *y[i] is : 21

address of &x[i] is : 0x7ffcbe2ed804
address of  y[i] is : 0x7ffcbe2ed804
value   of  x[i] is : 45
value   of *y[i] is : 45

address of &x[i] is : 0x7ffcbe2ed808
address of  y[i] is : 0x7ffcbe2ed808
value   of  x[i] is : -37
value   of *y[i] is : -37

address of &x[i] is : 0x7ffcbe2ed80c
address of  y[i] is : 0x7ffcbe2ed80c
value   of  x[i] is : 18
value   of *y[i] is : 18

address of &x[i] is : 0x7ffcbe2ed810
address of  y[i] is : 0x7ffcbe2ed810
value   of  x[i] is : -56
value   of *y[i] is : -56
```

Agenda

Pointer to a pointer

- A pointer to a pointer is a form of multiple indirection or a chain of pointers. Normally, a pointer contains the address of a variable. When we define a pointer to a pointer, the first pointer contains the address of the second pointer, which points to the location that contains the actual value as shown below.



Pointer to a pointer

- A variable that is a pointer to a pointer must be declared as such.

This is done by placing an additional asterisk in front of its name.

For example, following is the declaration to declare a pointer to a pointer of type int:

```
int **var;
```

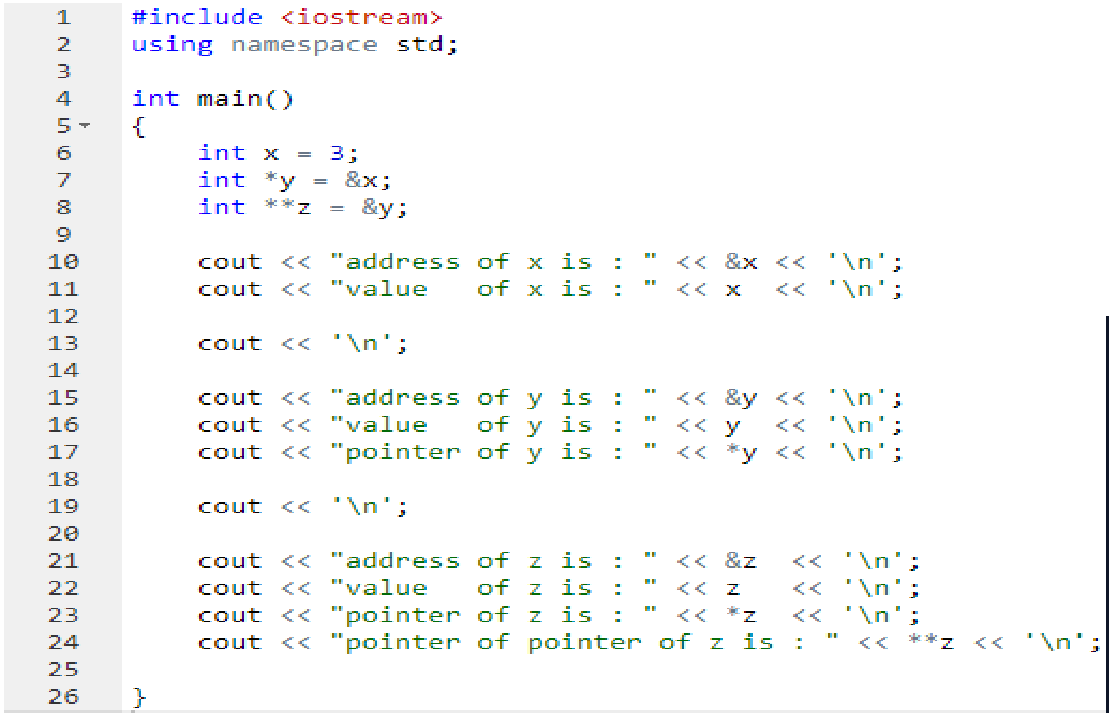
- When a target value is indirectly pointed to by a pointer to a pointer,

accessing that value requires that the asterisk operator be applied twice,

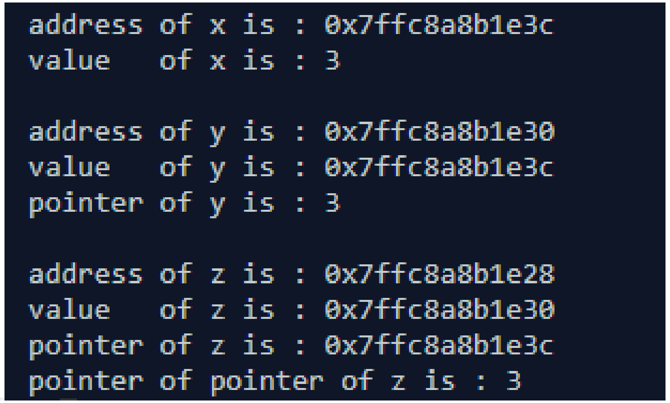
as is shown below in the example:

Pointer to a pointer

Link: repl.it/repls/AnotherScientificOrangutan



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

int main()
{
    int x = 3;
    int *y = &x;
    int **z = &y;

    cout << "address of x is : " << &x << '\n';
    cout << "value   of x is : " << x  << '\n';

    cout << '\n';

    cout << "address of y is : " << &y << '\n';
    cout << "value   of y is : " << y  << '\n';
    cout << "pointer of y is : " << *y << '\n';

    cout << '\n';

    cout << "address of z is : " << &z  << '\n';
    cout << "value   of z is : " << z   << '\n';
    cout << "pointer of z is : " << *z  << '\n';
    cout << "pointer of pointer of z is : " << **z << '\n';

}
```



```
address of x is : 0x7ffc8a8b1e3c
value   of x is : 3

address of y is : 0x7ffc8a8b1e30
value   of y is : 0x7ffc8a8b1e3c
pointer of y is : 3

address of z is : 0x7ffc8a8b1e28
value   of z is : 0x7ffc8a8b1e30
pointer of z is : 0x7ffc8a8b1e3c
pointer of pointer of z is : 3
```

Agenda

- ✓ 1- Definition of Pointers
- ✓ 2- Importance of pointers
- ✓ 3- Types of Pointers
- ✓ 4- Incrementing vs. Decrementing Pointers
- ✓ 5- Pointer Comparisons
- ✓ 6- Pointers vs. Arrays
- ✓ 7- Array of Pointers
- ✓ 8- Pointer to a pointer
- **9- Reference Variables**

Reference Variables

- A reference variable is an alias, that is, another name for an already existing variable.

Once a reference is initialized with a variable, either the variable name or the reference name may be used to refer to the variable.

- References are often confused with pointers but three major differences between references and pointers are:

- You cannot have NULL references. You must always be able to assume that a reference is connected to a legitimate piece of storage.
- Once a reference is initialized to an object, it cannot be changed to refer to another object. Pointers can be pointed to another object at any time.
- A reference must be initialized when it is created. Pointers can be initialized at any time.

Reference Variables

Think of a variable name as a label attached to the variable's location in memory.

You can then think of a reference as a second label attached to that memory location.

```
int    i = 17;
```

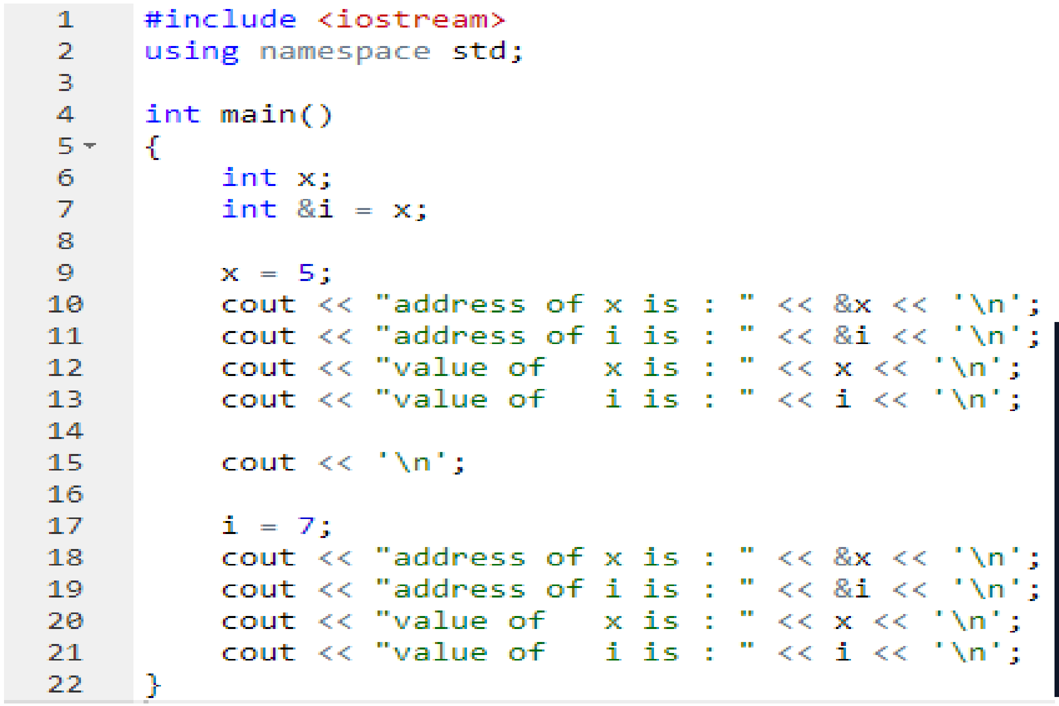
Therefore, you can access the contents of the variable through either the original variable name or the reference. For example, suppose we have the following example:

We can declare reference variables for i as follows.

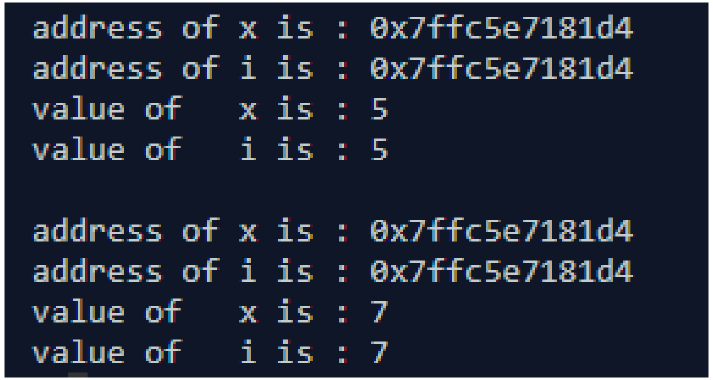
```
int&    r = i;
```

Reference Variables

Link: repl.it/repls/HarshFrostyAfricanbushviper



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

int main()
{
    int x;
    int &i = x;

    x = 5;
    cout << "address of x is : " << &x << '\n';
    cout << "address of i is : " << &i << '\n';
    cout << "value of   x is : " << x << '\n';
    cout << "value of   i is : " << i << '\n';

    cout << '\n';

    i = 7;
    cout << "address of x is : " << &x << '\n';
    cout << "address of i is : " << &i << '\n';
    cout << "value of   x is : " << x << '\n';
    cout << "value of   i is : " << i << '\n';
}
```



```
address of x is : 0x7ffc5e7181d4
address of i is : 0x7ffc5e7181d4
value of   x is : 5
value of   i is : 5

address of x is : 0x7ffc5e7181d4
address of i is : 0x7ffc5e7181d4
value of   x is : 7
value of   i is : 7
```

Agenda

- ✓ 1- Definition of Pointers
- ✓ 2- Importance of pointers
- ✓ 3- Types of Pointers
- ✓ 4- Incrementing vs. Decrementing Pointers
- ✓ 5- Pointer Comparisons
- ✓ 6- Pointers vs. Arrays
- ✓ 7- Array of Pointers
- ✓ 8- Pointer to a pointer
- ✓ 9- Reference Variables

DO
MORE.

Assignment

References

[01]	Online Course YouTube Playlists	https://bit.ly/2Pq88rN
[02]	Introduction to Algorithms Thomas H. Cormen	https://bit.ly/2ONhuSn
[03]	Competitive Programming 3 Steven Halim	https://nus.edu/2z4OvyK
[04]	Fundamental of Algorithmics Gilles Brassard and Paul Bartley	https://bit.ly/2QuvwbM
[05]	Analysis of Algorithms An Active Learning Approach	http://bit.ly/2EgCCYX
[06]	Data Structures and Algorithms Annotated Reference	http://bit.ly/2c37XEv
[07]	Competitive Programmer's Handbook	https://bit.ly/2APAbeG
[08]	GeeksforGeeks	https://geeksforgeeks.org
[09]	Codeforces Online Judge	http://codeforces.com
[10]	HackerEarth Online Judge	https://hackerearth.com
[11]	TopCoder Online Judge	https://topcoder.com

Questions ?