Topics:

- Pointers and its operators
- Struct
- Pass by value
- Pass by pointers
- Pass by reference
- Pointer arithmetic
- **Exercise:** Swapping elements using pointers
- **Exercise:** Reverse an array

‘Address of’ operator

What if we wanted to **get** the **location** of this variable in the memory?

```
int a = 10;
```

‘Address of’ operator

What if we wanted to **get** the **location** of this variable in the memory?

```
int a = 10;
```

Solution:

We use the ‘**address of**’ operator - **&a**

```
cout<< &a;
```

0x7ffccbbcd804 → Hexadecimal Memory location of the variable **‘a’.**

Pointer Variables

What if we wanted **store** the address of a variable?

```
int a = 10;
```

Solution:

We define a **pointer** variable.

```
int* p = &a;
```

$\Rightarrow$ p is the **pointer variable** to an integer

Pointer Variables

What if we wanted **store** the address of a variable?

```
int a = 10;
```

Solution:

We define a **pointer** variable.

```
int* p = &a;
```

$\longrightarrow$ p is the **pointer variable** to an integer

```
cout<< a ;        Output:
                  10
cout<< p ;        0x7ffccbbcd804
```

Pointer is just a variable that stores a memory address.

Dereferencing a pointer

How to fetch the value from pointer(address)?

cout<< *p << endl ; - 10 - Dereferencing the address

Struct:

If we want to store say multiple fields for a student. We use an user defined type called a ‘struct’.

```
struct student{

        string name;
        string email;
        int birthday;
        string address;

};
```

Pointer to a struct:

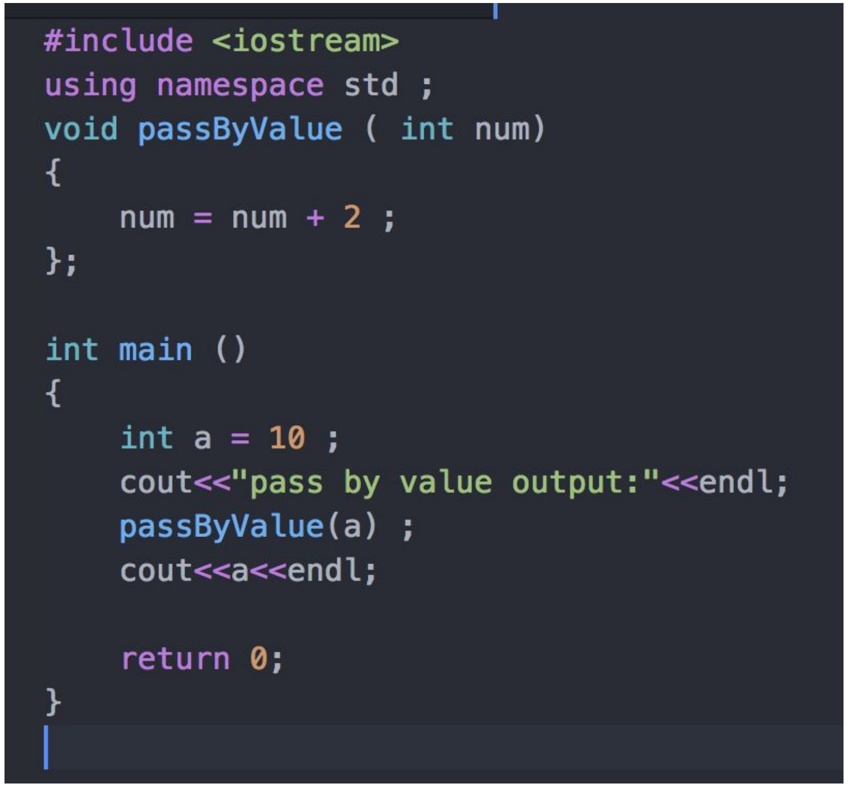
```
student s;
s.name='David';
s.email='David@colorado.edu';

student* ptr;

ptr = &s;
```

<u>Accessing the fields using pointer variable:</u>

```
    ptr−>name
    ptr−>email
```

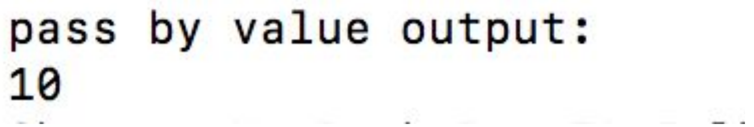
https://www.codepile.net/pile/ZAkvy9wB

Pass By Value



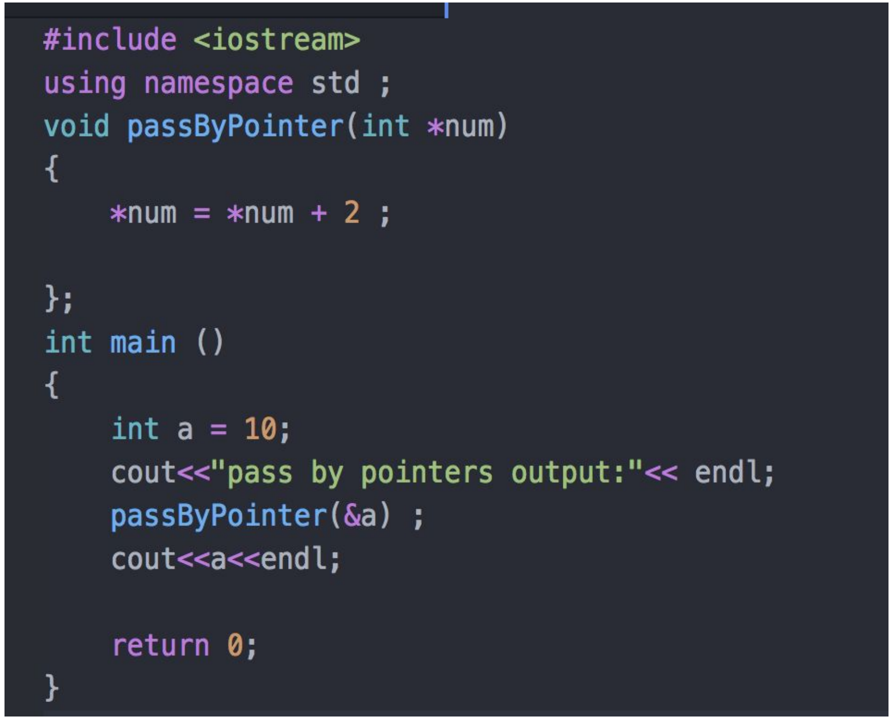
```
#include <iostream>
using namespace std ;
void passByValue ( int num)
{
    num = num + 2 ;
};

int main ()
{
    int a = 10 ;
    cout<<"pass by value output:"<<endl;
    passByValue(a) ;
    cout<<a<<endl;

    return 0;
}
```

Why does it print 10?
The function here creates a local copy and updates it.



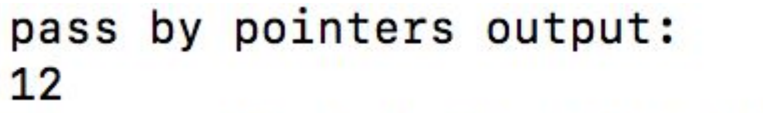
```
pass by value output:
10
```

Pass By Pointers



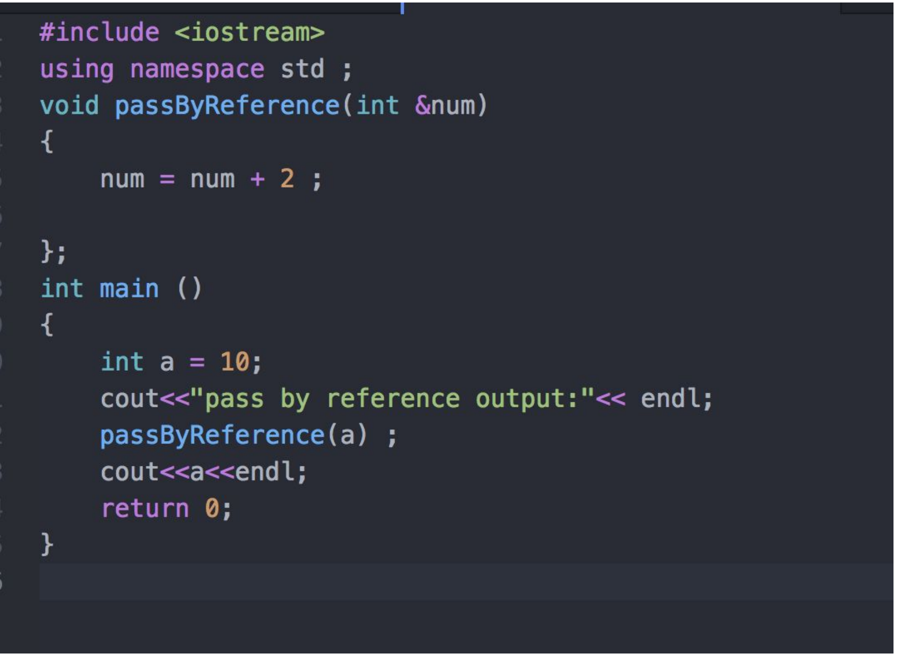
```cpp
#include <iostream>
using namespace std ;
void passByPointer(int *num)
{
    *num = *num + 2 ;

};
int main ()
{
    int a = 10;
    cout<<"pass by pointers output:"<< endl;
    passByPointer(&a) ;
    cout<<a<<endl;

    return 0;
}
```

Why does it print 12?
The function here creates a local copy of the address.
Using the address, it updates the variable at that location.



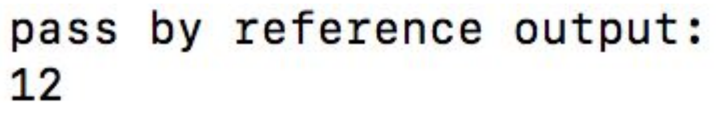
```
pass by pointers output:
12
```

Pass By Reference



```
#include <iostream>
using namespace std ;
void passByReference(int &num)
{
    num = num + 2 ;

};
int main ()
{
    int a = 10;
    cout<<"pass by reference output:"<< endl;
    passByReference(a) ;
    cout<<a<<endl;
    return 0;
}
```

Why does it print 12?
Here, we pass an alias to the variable.
Hence it edits the same thing.



```
pass by reference output:
12
```

Arrays and pointers:

In c++, array name is a ***constant pointer***.
So, you can access array elements by using pointer notation.

For example,

```
arr[3] = {1,2,3};

*(arr) will be 1
*(arr+1) will be 2
*(arr+2) will be 3
```

This is because,
Array is made of ***contiguous memory elements***. (each consecutive element is stored contiguously in memory).

Pointer arithmetic:

A pointer is a numeric value.
So, it can be incremented/decremented.

When incremented, it has the memory address of the next element in the array.

For example,

```
arr[3] = {1,2,3};

int *p = arr;

    Address of arr[0] = Value of p = 0xbfa088b0

p = p +1;

    Address of arr[1] = Value of arr+1= 0xbfa088b4
```

Since it an integer, the next memory location is after 4 bytes. This is the size of an integer.

Exercise:

Print all the addresses of the given elements.

Using pointers, print elements. Use dereferencing here.

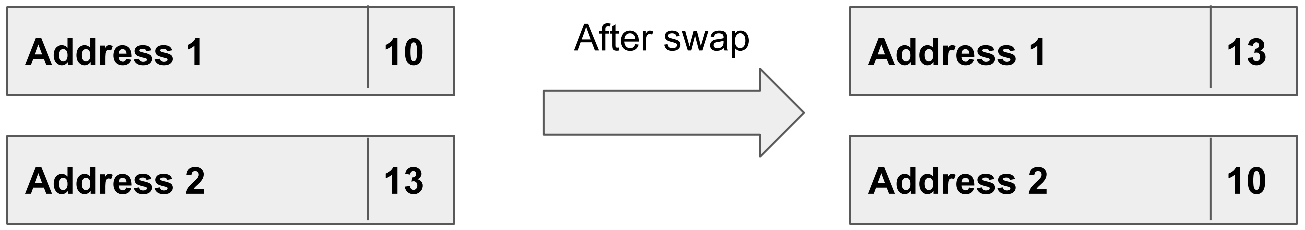
Exercise:

Swap two variables:

```
void swap(int n1, int n2) {
  int temp;
  temp = n1;
  n1 = n2;
  n2 = temp;
}
```

Exercise:

Swap two variables using pointers:



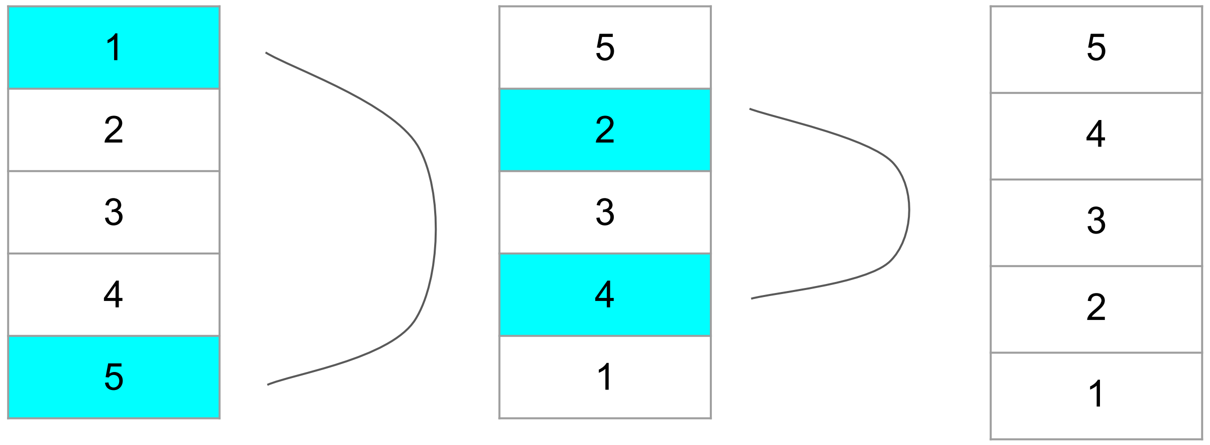
```
void swap(int *n1, int *n2) {

//TODO

}
```

Exercise:

Reverse an array:

Swap first and last elements.

Keep moving inward till you reach the mid-point.


1
2
3
4
5
5
2
3
4
1
5
4
3
2
1