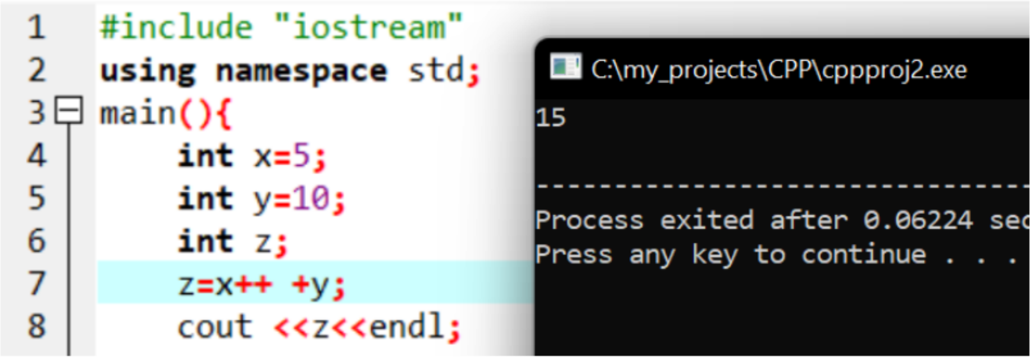
Summary CPP

Sessions No 7(19-10-2022)

- Pre Increment operator - **++x** - here it first increments the value of **x** by 1 and then it will store back in the x variable and then use it.
- Post Increment operator - **x++** - here it will first use this x and after this will increment the value of x by 1 and then store it back to x
- Similar to ++x and x++ we also have - - x and x- - which will decrease the value of x by 1
- Now, when we have multiple operations in an expression, then we need to decide the priority of different operators. And the operator which has more priority will be executed first.

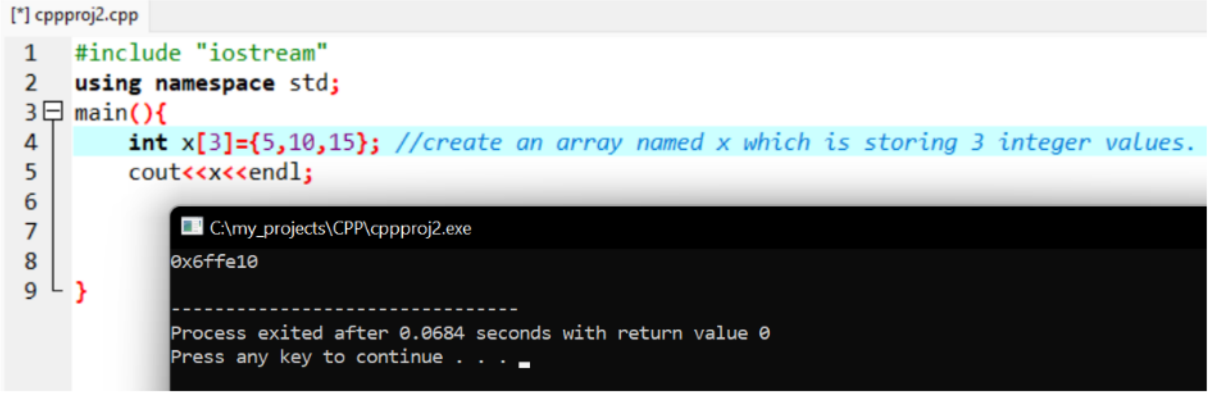


```cpp
#include "iostream"
using namespace std;
main(){
	int x=5;
	int y=10;
	int z;
	z=x++ +y;
	cout <<z<<endl;
```

```
C:\my_projects\CPP\cppproj2.exe
15

--------------------------------
Process exited after 0.06224 sec
Press any key to continue . . .
```

For example here priority of the **Post incitement Operator** is less than "**+**" so here first x+y will execute then "=" will execute(means z=5+10=15) and then in the last **"X++"** will execute and the value of x will increase by 1.

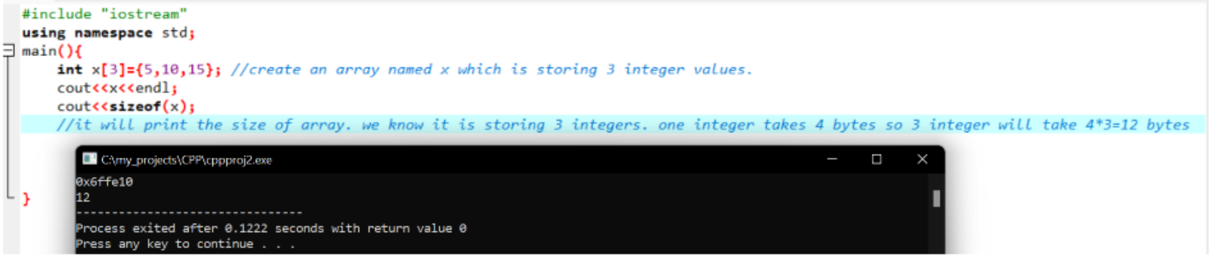
- The **array** is one of the data structures that stores similar data types of values in a continuous manner in RAM. Here in the below example, we have created an array **"x"** which stores three integer values in a continuous manner inside RAM. But when we print the "x" with help of **cout** function, we get some physical address. Hence here our array variable "**x**" is storing the physical address of something, So we can call it a **pointer**. As we know pointer is a special kind of variable that stores the address of other variables. Now here the main point is that "**x**" is storing the address of array.



```
#include "iostream"
using namespace std;
main(){
    int x[3]={5,10,15}; //create an array named x which is storing 3 integer values.
    cout<<x<<endl;



}
```

```
0x6ffe10

--------------------------------
Process exited after 0.0684 seconds with return value 0
Press any key to continue . . .
```

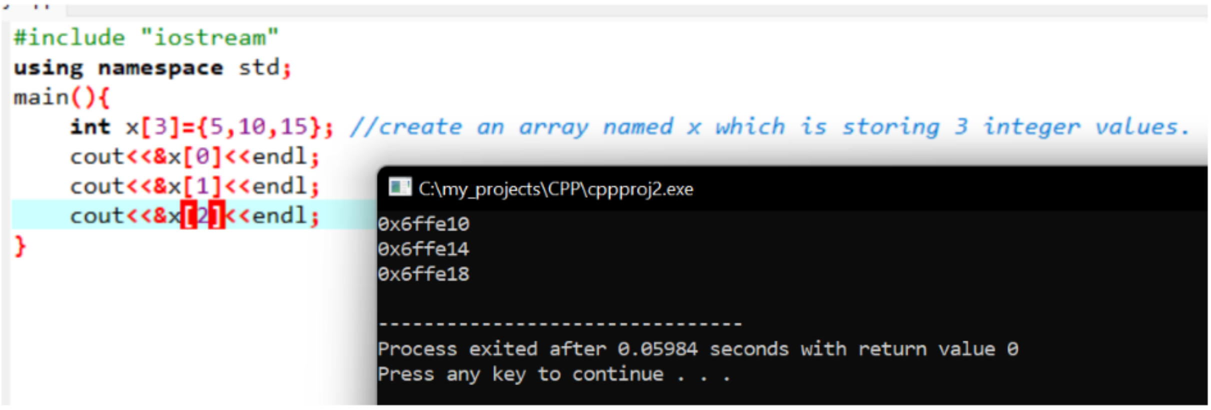
- Check the size of Array -



```
#include "iostream"
using namespace std;
main(){
    int x[3]={5,10,15}; //create an array named x which is storing 3 integer values.
    cout<<x<<endl;
    cout<<sizeof(x);
    //it will print the size of array. we know it is storing 3 integers. one integer takes 4 bytes so 3 integer will take 4*3=12 bytes

}
```

```
0x6ffe10
12
--------------------------------
Process exited after 0.1222 seconds with return value 0
Press any key to continue . . .
```

- To print the first element of array - **cout<<x[0] ;** - *it will print the data stored in the first element of array*
- Now to print the first element we can also print it by **cout<<*x<<endl;** As we know "**x**" is storing just the address of starting(first) element of array, hence to get the data in it, we can write like **"*x"**, It is the same way as we get the value using a pointer.
- To get the address of all the elements of array -



```
#include "iostream"
using namespace std;
main(){
    int x[3]={5,10,15}; //create an array named x which is storing 3 integer values.
    cout<<&x[0]<<endl;
    cout<<&x[1]<<endl;
    cout<<&x[2]<<endl;
}
```

```
0x6ffe10
0x6ffe14
0x6ffe18

--------------------------------
Process exited after 0.05984 seconds with return value 0
Press any key to continue . . .
```

Now we can see here the difference of 4 bytes between all three addresses, it is due to every element we are storing here being of "**int datatype**". And you will also see they are in a continuous manner, which proves Array store data in continuous space in RAM

- Now, As we know "x" here stores the starting address of the first element of the array, so if we try to print x+1, it will not just add "1" to the value of "x", our program understands that x is storing the address, and if we do x+1 it will show us the next element’s starting address and so on. Here in the array, we have stored integers. So on doing x+1 it will automatically go and read the next 4-byte address.