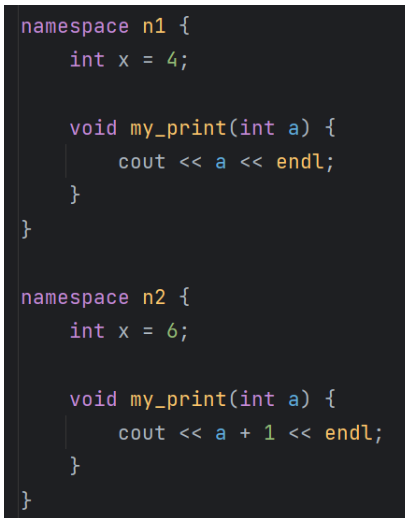
SDL2 Course Day-0

Intro to C++

Namespaces:

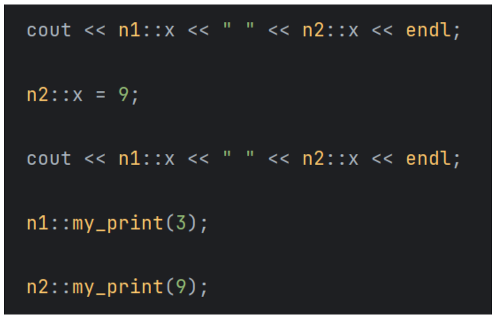
Namespaces = fancy word for block of code

Defined by: namespace <name> { <declarations or definitions> }



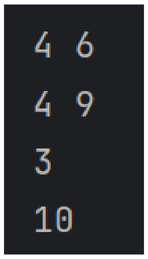
```cpp
namespace n1 {
    int x = 4;

    void my_print(int a) {
        cout << a << endl;
    }
}

namespace n2 {
    int x = 6;

    void my_print(int a) {
        cout << a + 1 << endl;
    }
}
```

Namespaces variables and functions are accessed by “scope resolution operator” ::



```cpp
cout << n1::x << " " << n2::x << endl;

n2::x = 9;

cout << n1::x << " " << n2::x << endl;

n1::my_print(3);

n2::my_print(9);
```

output:

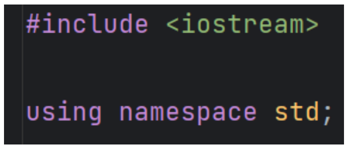


```
4 6
4 9
3
10
```

Each namespace has their own memory and even if there are items which have the same name in two different namespaces, changing one will not change the other

std is a namespace which is defined by GNU (people who made a standard for C/C++).

If you don’t want to use std:: everywhere then type the following statement below the #include files:

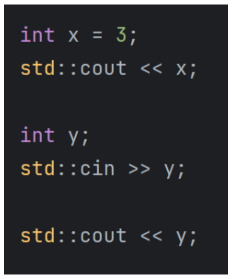


```cpp
#include <iostream>

using namespace std;
```

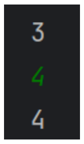
This allows you to use cout and cin directly instead of std::cout and std::cin.

cout and cin: present in <iostream>

Replaces printf and scanf from C, defined by GNU peeps.



```
int x = 3;
std::cout << x;

int y;
std::cin >> y;

std::cout << y;
```

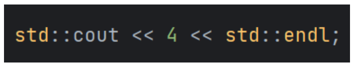
output: 

```
3
4
4
```

cout prints to terminal, cin takes input from terminal (in the above code 3 is printed, then I typed 4 in the terminal and then 4 is printed)

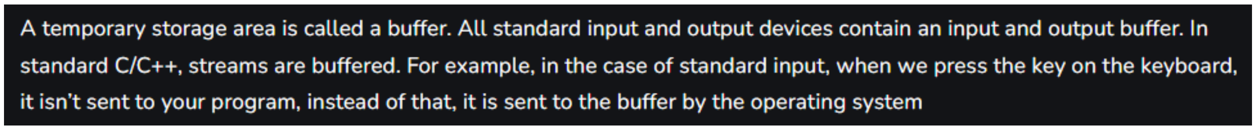
endl:

used to print newline:

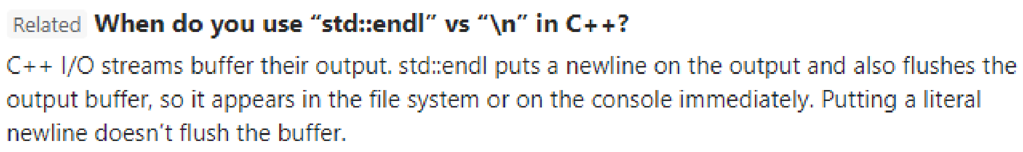


```
std::cout << 4 << std::endl;
```

Why endl?



A temporary storage area is called a buffer. All standard input and output devices contain an input and output buffer. In standard C/C++, streams are buffered. For example, in the case of standard input, when we press the key on the keyboard, it isn't sent to your program, instead of that, it is sent to the buffer by the operating system

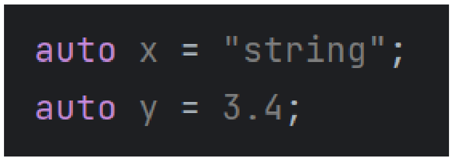


Related **When do you use "std::endl" vs "\n" in C++?**

C++ I/O streams buffer their output. std::endl puts a newline on the output and also flushes the output buffer, so it appears in the file system or on the console immediately. Putting a literal newline doesn't flush the buffer.

Auto:

The auto keyword is used when you don't want to spend time figuring out wtf the data type of the variable is. Can be used only when the variable is declared **and** initialized.

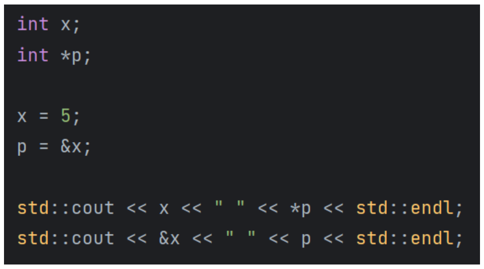


```
auto x = "string";
auto y = 3.4;
```

Note that auto is not a datatype, It just smartly assigns the datatype according to what value is on the right side.

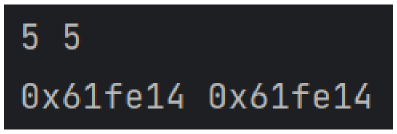
Pointer basics:

& gives address of a variable, using * on a pointer “**dereferences**” it and gives the underlying value.

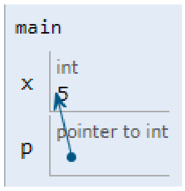


```
int x;
int *p;

x = 5;
p = &x;

std::cout << x << " " << *p << std::endl;
std::cout << &x << " " << p << std::endl;
```

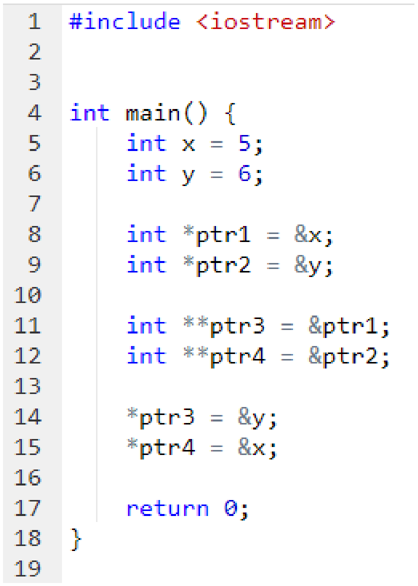
output:



```
5 5
0x61fe14 0x61fe14
```

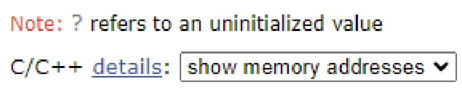


Head over to https://pythontutor.com/cpp.html#mode=edit and type the following code there:



```
#include <iostream>


int main() {
    int x = 5;
    int y = 6;

    int *ptr1 = &x;
    int *ptr2 = &y;

    int **ptr3 = &ptr1;
    int **ptr4 = &ptr2;

    *ptr3 = &y;
    *ptr4 = &x;

    return 0;
}

```

Click on visualize execution to see a step by step execution and change the box to show memory addresses:

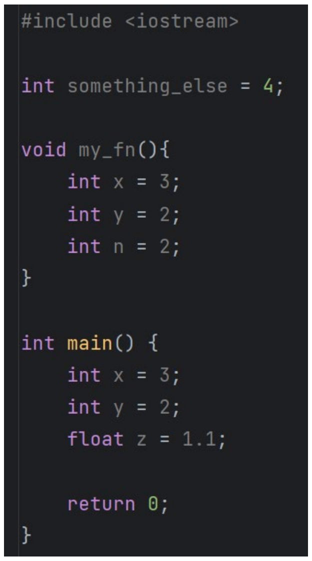


Note: ? refers to an uninitialized value

C/C++ details: show memory addresses

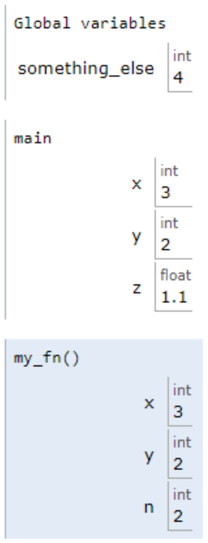
Test out all you want related to pointers on this site, very very very nice tool.

Scope / Context:



```
#include <iostream>

int something_else = 4;

void my_fn(){
    int x = 3;
    int y = 2;
    int n = 2;
}

int main() {
    int x = 3;
    int y = 2;
    float z = 1.1;

    return 0;
}
```

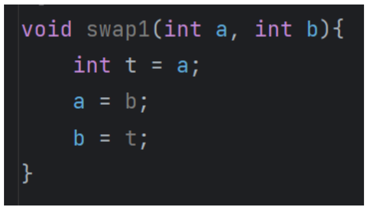
internal view:



By default, all global variables are available in the scope of everything else along with anything else declared above it.

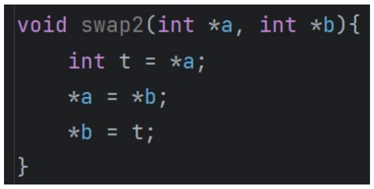
Function parameters:

Pass by value: (creates its own copy of a, b in its scope) (doesn't work, remind yourself of the example of copying someone else's document and making changes in your copy which obviously will not change the original copy)



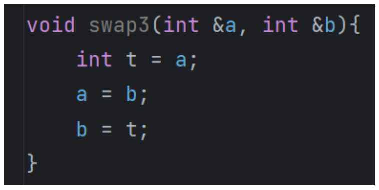
```
void swap1(int a, int b){
    int t = a;
    a = b;
    b = t;
}
```

Pass by address: (creates own copy of pointer but still points to the original thing so hence dereferencing would work)



```
void swap2(int *a, int *b){
    int t = *a;
    *a = *b;
    *b = t;
}
```

Pass by reference: (does not create its own copy, instead refers to the original thing)



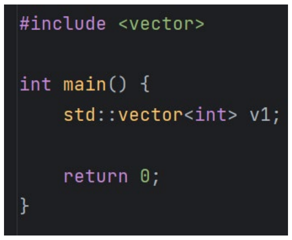
```
void swap3(int &a, int &b){
    int t = a;
    a = b;
    b = t;
}
```

Extra reference if you want https://www.educative.io/answers/pass-by-value-vs-pass-by-reference

Arrays & Vectors: present in <vector>

Refer online for arrays and their limitations.

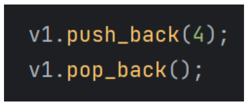
Vectors are not defined by us so please remember that you need to use std:: before the vector or type "using namespace std" at the top of the file.



```
#include <vector>

int main() {
    std::vector<int> v1;

    return 0;
}
```

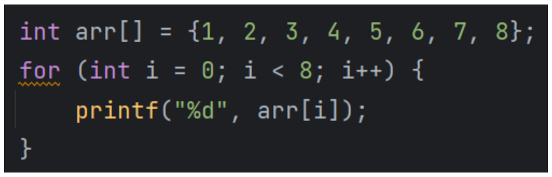
<int> is the datatype of each element here.

push_back(<datatype>) is used to insert at the end of the vector and pop_back() is used to delete from the vector.



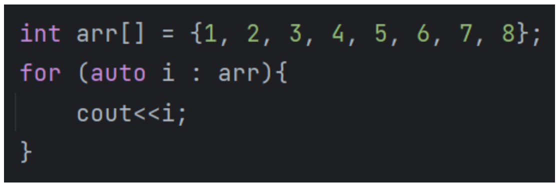
```
v1.push_back(4);
v1.pop_back();
```

To print an array in C normally:



```
int arr[] = {1, 2, 3, 4, 5, 6, 7, 8};
for (int i = 0; i < 8; i++) {
    printf("%d", arr[i]);
}
```

New format of for loop: (range based for loop):



```
int arr[] = {1, 2, 3, 4, 5, 6, 7, 8};
for (auto i : arr){
    cout<<i;
}
```

Refer online for arrays and vectors like geek for geeks or something for more info on these, we'll go through them in class.