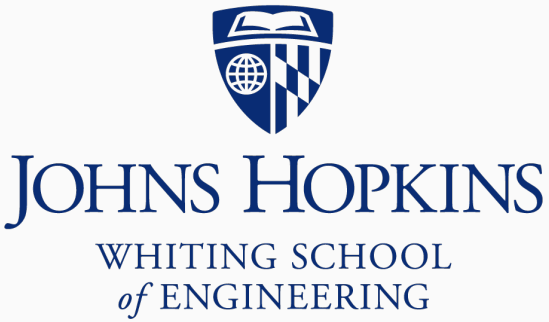
C++ I/O and namespaces

Ben Langmead

ben.langmead@gmail.com

www.langmead-lab.org



Source markdown available at github.com/BenLangmead/c-cpp-notes

C++ I/O and namespaces

We saw this example:

```
#include <iostream>
#include <cassert>  // dropped .h, added c at beginning

using std::cout;
using std::endl;

int main(int argc, char *argv[]) {
    assert(argc > 1); // our old friend assert
    cout << "Hello " << argv[1] << "!" << endl;
    return 0;
}
```

C++: I/O

```
$ g++ -c hello_world_2.cpp -std=c++11 -pedantic -Wall -Wextra
$ g++ -o hello_world_2 hello_world_2.o
$ ./hello_world_2 Everyone
Hello Everyone!
```

Note: argc and argv work just like in C

C++: I/O

`iostream` is the main C++ library for input and output

```
#include <iostream>
```

C++: namespaces

```
using std::cout;
using std::endl;
```

C++ has *namespaces*.

- In C, when two things have the same name, we get errors (from compiler or linker) and confusing situations ("shadowing")
- In C++, items with same name can safely be placed in distinct "namespaces", similar to Java packages / Python modules

C++: namespaces

Most C++ functionality lives in namespace called `std`

If we didn't include:

```
using std::cout;
using std::endl;
```

at the top, then we would have to write the fully qualified name each time:

```
std::cout << "Hello world" << std::endl;
```

C++: namespaces

Do not use `using` in a header file

Doing so affects all the source files that include that header, even indirectly, which can lead to confusing name conflicts

- *Only* use `using` in source `.cpp` files
- This will be enforced in homework grading

Use fully qualified names (e.g. `std::endl`) in headers

C++: namespaces

Do not use `using namespace <id>`

This is bad practice as it brings *all* the names in `namespace <id>` into the current namespace

You will probably see C++ code with `using namespace std`; don't imitate this. You will lose points if you do this on homeworks.

C++: I/O

Text input & output in C++ are simpler than in C, thanks to C++'s stream operators and libraries

(We will probably not cover binary I/O in C++)

C++: I/O

```
cout << "Hello world!" << endl;
```

`cout` is our old friend, the standard output stream

- Like `stdout` in C

`endl` is the newline character

- C++ has `'\n'` too, but `endl` is usually preferred

`<<` is the *insert operator*

- Also known as *insertion operator* or *output operator*

C++: I/O

Insert operator joins all the items to write in a “chain”

Leftmost item in chain is the stream being written to

```
cout << "We have " << inventory << " " << item << "s left,"
     << " costing $" << price << " per unit" << endl;
```

cout is the stream, everything else is the string to write to it

C++: I/O

```cpp
#include <iostream>
using std::cout;
using std::endl;

int main() {
    int inventory = 44;
    double price = 70.07;
    const char *item = "chainsaw";
    bool on_sale = true;

    cout << "We have " << inventory << " " << item << "s left,"
         << " costing $" << price << " per unit."
         << " On sale = " << on_sale << '.' << endl;
    return 0;
}
```

```
$ g++ -c cpp_io_1.cpp -std=c++11 -pedantic -Wall -Wextra
$ g++ -o cpp_io_1 cpp_io_1.o
$ ./cpp_io_1
We have 44 chainsaws left, costing $70.07 per unit. On sale = 1.
```

C++: I/O

No format specifiers (%d, %s etc)

Instead, items to be printed are arranged in printing order; easier to read and understand

```
int inventory = 44;
double price = 70.07;
const char *item = "chainsaw";
bool on_sale = true; // *** note the "bool" type

cout << "We have " << inventory << " " << item << "s left,"
     << " costing $" << price << " per unit."
     << " On sale = " << on_sale << '.' << endl;
```

Note that bool is printed like an integer

C++: I/O

An example of C++ I/O but also an example of *operator overloading*

<< usually does bitwise left-shift; but if operand on the left is a C++ stream (cout), << is the insert operator

```
cout << "Hello world!" << endl;
```

More on this later

C++: I/O

How much of the C library can we use in C++? Most of it.

```
#include <cstdio>

int main() {
    int inventory = 44;
    double price = 70.07;
    const char *item = "chainsaw";
    int on_sale = 1; // *** note the "bool" type

    printf("We have %d %ss left, costing $%f per unit. On sale = %d.\n",
           inventory, item, price, on_sale);
    return 0;
}
```

```
$ g++ -c cpp_io_2.cpp -std=c++11 -pedantic -Wall -Wextra
$ g++ -o cpp_io_2 cpp_io_2.o
$ ./cpp_io_2
We have 44 chainsaws left, costing $70.070000 per unit. On sale = 1.
```

C++: I/O

```cpp
#include <iostream>
#include <string>  // new header -- not used in C

using std::cout;
using std::cin;
using std::endl;
using std::string;

int main() {
    cout << "Please enter your first name: ";
    string name;
    cin >> name;  // read user input into string object
    cout << "Hello, " << name << "!" << endl;
    return 0;
}
```

C++: I/O

```
$ g++ -c cpp_io_3.cpp -std=c++11 -pedantic -Wall -Wextra
$ g++ -o cpp_io_3 cpp_io_3.o
$ echo Ed | ./cpp_io_3
Please enter your first name: Hello, Ed!
```

C++: I/O

```
cin >> name;
```

Reads one whitespace-delimited token from standard input and places the result in string name

>> is the *extraction operator*

- Also called *input operator*

C++: I/O

```cpp
#include <iostream>
#include <string>

using std::cout;
using std::cin;
using std::endl;
using std::string;

int main() {
    string word, smallest;
    while(cin >> word) {
        if(smallest.empty() || word < smallest) {
            smallest = word;
        }
    }
    cout << smallest << endl;
    return 0;
}
```

C++: I/O

```
$ g++ -c smallest_word.cpp -std=c++11 -pedantic -Wall -Wextra
$ g++ -o smallest_word smallest_word.o
$ echo "the quick brown fox" | ./smallest_word
brown
```

C++: I/O

```
while(cin >> word) {
    // ...
}
```

`cin >> word` evaluates to true if the input stream is still in a "good state" (no error, no EOF) after reading the word

C++: I/O

```cpp
#include <iostream>
#include <cctype>

using std::cout;
using std::cin;
using std::endl;

int main() {
    char ch;
    while(cin.get(ch)) { // read single character
        ch = toupper(ch);
        cout << ch; // print single character
    }
    cout << endl;
    return 0;
}
```

C++: I/O

```
$ g++ -c uppercase_cpp.cpp -std=c++11 -pedantic -Wall -Wextra
$ g++ -o uppercase_cpp uppercase_cpp.o
$ echo "The Quick Brown Fox" | ./uppercase_cpp
THE QUICK BROWN FOX
```