Ben Langmead

ben.langmead@gmail.com

www.langmead-lab.org

Source markdown available at github.com/BenLangmead/c-cpp-notes
std::stringstream and its specialized cousins std::istringstream and std::ostringstream help you get data into and out of strings

First, let's see an example of overloading the extraction (>>) operator
```cpp
std::istream& operator>>(std::istream& is, Complex& c) {
    // Assume format "3.0 + 4.0 i"
    string tmp;
    is >> c.real; // parse real coefficient
    is >> tmp; // skip the +
    is >> c.imaginary; // parse imaginary coefficient
    is >> tmp; // parse the i
    assert(tmp == "i"); // sanity check
    return is;
}
```

Similar to operator<< but with istream instead of ostream and
>> instead of <<

Second argument must be non-const reference so we can modify
stringstream

::stringstream is a stream, like :cout or :cin

Instead of reading or writing to console, it reads and writes to a temporary string (“buffer”) stored inside

The string buffer can be accessed with .str()

```cpp
#include <string>
#include <iostream>
#include <sstream> // for :stringstream

using :cout; using :endl;
using :string; using :stringstream;

int main() {
    stringstream ss; // buffer is empty
    ss << "Hello, world!" << endl; // write message to buffer
    cout << ss.str(); // retrieve buffer with .str()
    return 0;
}
```
$ g++ -c ss1.cpp -std=c++11 -pedantic -Wall -Wextra
$ g++ -o ss1 ss1.o
$ ./ss1
Hello, world!

Why not use std::string and + operator instead?

+ operator overload for std::string only handles std::string or char arguments; often we want other types too

stringstream works with operator<< and operator>>; an operator overload for either will work with stringstream just as well as with cin/cout
#include <string>
#include <iostream>
#include <sstream>

using std::cout; using std::endl;
using std::string; using std::stringstream;

int main() {
    string ss;
    ss << "Hello" << ' ' << 35 << " world"; // mix string, char, int
    string word1, word2;
    int num;
    ss >> word1 >> num >> word2; // read them back out
    cout << word1 "", " " << word2 "!";
    return 0;
}
stringstream

$ g++ -c ss2.cpp -std=c++11 -pedantic -Wall -Wextra
$ g++ -o ss2 ss2.o
$ ./ss2
Hello, world!
# include <string>
#include <iostream>
#include <sstream>
#include <vector>

using std::ostream; using std::istream;
using std::cout; using std::endl;
using std::vector; using std::stringstream;

ostream& operator<<(ostream& os, const vector<int>& vec) {
    for (int i : vec) {
        os << i << ' ';
    }
    return os;
}

istream& operator>>(istream& is, vector<int>& vec) {
    int i;
    while(is >> i) {
        vec.push_back(i);
    }
    return is;
}
```cpp
int main() {
    stringstream ss("1 2 3 4 5");
    vector<int> vec;
    ss >> vec;
    cout << vec << endl;
    return 0;
}
```

```
$ g++ -c ss2.cpp -std=c++11 -pedantic -Wall -Wextra
$ g++ -o ss2 ss2.o
$ ./ss2
1 2 3 4 5
```
Whoa! What was that `for(int i : vec)` business?

```cpp
for(int i : vec) { os << i << ' '; }
```

That’s a “ranged for”, a convenience added in C++11 that we will discuss more later.
**stringstream**

stringstream is an example of *multiple inheritance*

Inherits from:

- istream, which handles input and overloads extraction operator
- ostream, which handles output and overloads insertion operator

If you only need one or the other, you can use istream or ostream
stringstream can be especially useful for testing, especially if you have designed your code to handle input and output as streams.

Example like today’s exercise:

```cpp
// Only using for input
istringstream ss("Eight of Hearts 
  "Ten of Hearts 
  "Jack of Hearts 
  "Nine of Hearts 
  "Queen of Hearts");
assert(straight_flush(ss));
```