# CS 246 Fall 2015 - Tutorial 3

October 9, 2015

## 1 Summary

- C++ I/O
- Stringstreams
- Filestreams
- C++ Command Line Arguments

## 2 C++ I/O

- C++ I/O is different than the C I/O you may be used to
- For this course, you should not use C I/O (and other C libraries) unless told otherwise
- Recall, that C++ has three default input and output streams:
  - **cout** - standard output
  - **cerr** - standard error (unbuffered - prints immediately)[1]
  - **cin** - standard input
- Let's see an example that will take in a number and output a phrase. (phrases.cpp)

```
#include <iostream>
#include <string>
using namespace std;

int main() {
  int choice;
  int numChoices = 5;
  string phrases[] = {"More Vespene Gas required.","The sun is shining. But the ice is slippery.",
                      "Gotta go fast!", "Autobots, roll out!", "Do or do not. There is no try."};
  cout << "Please choose a number from 1-5: ";
  while(cin >> choice) { // cin needs to be read at least once before it can hit EOF or fail
    if(choice > numChoices)
      cerr << "Invalid number" << endl;
    else
      cout << phrases[choice-1] << endl;
    cout << "Please choose a number from 1-5: ";
  }
}
```

- This program will end when either EOF is reached or invalid input is given (e.g. non-integer input). Why?
- Accordingly, this program is not very robust. How could we make it more so?

[1] Technically, there are four. The fourth is clog and is basically bufferred cerr

  - Explicitly checking for failure/EOF by using `cin.fail()` and `cin.eof()` and not using the implicit conversion to boolean value
  - If we are in a failure case then we could use `cin.ignore()` to ignore the next character of input and then `cin.clear()` to reset the failure flag.
  - Does the order of `cin.ignore()` and `cin.clear()` matter?
  - What part of stdin does `cin.ignore()` actually ignore? How does it interact with whitespace?
  - Why do we reset the failure flag?

- Recall that cin ignores any and all whitespace (unless you use the I/O manipulator `noskipws` to stop skipping whitespace).
- Suppose we wanted to get an entire line. How could we do this?
  - By using **getline**, e.g. `string s; getline(cin, s)`
  - Thus we take a line from cin and store it in the string s.
  - But how do we process this line now? Using **stringstreams**!
- Sometimes, cin doesn't realize that something is an error until too late.

```
#include <iostream>
using namespace std;

int main() {
  int i;
  //Try to interpret the start of the stream as a number
  if (cin >> i) {
    cout << "Found number " << i << endl;
  //Try to interpret the start of the stream as a string
  } else {
    cin.clear();
    string s;
    cin >> s;
    cout << "Found string " << s << endl;
  }
}
```

- What happens when we run this with "+ 50" or "- 50"? How can we fix this?

# 3 Stringstreams

- Accessed through

  `#include <sstream>`

- Stringstreams are a type of stream that we typically use to interact with string objects
- We can place data in a stringstream exactly how we place data in `cout`
- We can also place data in a stringstream when we create it
- We can remove data from a stringstream exacly how we remove it from `cin`
- A stringstream can be used for input and output at the same time
  - If we want only to retrieve data, we can use an istringstream (works like cin only)
  - If we want only to store data, we can use an ostringstream (works like cout only)
  - Typically we use regular stringstreams but use either can serve as a sanity check
- The easiest way to check if a string is an int/float in C++ is to use a stringstream

```
int num = 0;
string test = ...;  // possibly an int
stringstream ss(test);  // creates a stringstream containing the string test
if ( ss >> num ){
        // if this succeeds, the first thing in ss was an int
} else {
        // the first thing in ss was not an int
}
```

# 4 Filestreams

- Accessed through

  ```
  #include <fstream>
  ```

- read data from a specific file instead of `stdin`
- write data to a specific file instead of `stdout`
- `ifstream` stands for input file stream (like `stdin`)
- `ofstream` stands for output file stream (like `stdout`)

## 4.1 `ifstream`

- Works like `cin`
- Opened via the following:

  ```
  ifstream ifs("infile");
  ```

- Takes input from file rather than cin
- Note that filename must be a C-string and not a C++ string

## 4.2 `ofstream`

- Works like `cout`
- Opened via the following:

  ```
  ofstream ofs("outfile");
  ```

- Writes to a file rather than stdout
- Note that filename must be a C-string and not a C++ string
- If the file doesn't exist, will create it
- If the file exists, will overwrite it

## 4.3 Basic Example

```
#include <fstream>
using namespace std;

int main(){
  ifstream ifs("infile");
  ofstream ofs("outfile");
  string s;
  ifs >> s;
  if (! ifs.fail())
```

```
10      for(int i=s.size()-1; i >= 0; --i) ofs << s.at(i);
11    ofs << endl;
12  }
```

**Note:** If we were to replace every instance of `ifs` with `cin` and `ofs` with `cout`, the program will still work. However, it will read from standard input and write to standard output instead of the respective files. Typically, wherever we can use `cin` or `cout` we can use the equivalent **filestream** or **stringstream**.

# 5 C++ Command Line Arguments

- You may remember that when we were writing bash scripts, we could access the command line arguments using $0, $1, $2, etc
- These arguments were passed to a program without using redirection from stdin (`./myscript one two three`)
- We can also setup a C++ program to accept command line arguments
  - To do this, we give main two arguements, an int, which will hold the number of command line arguments, and an array of char*'s, where each element is one whitespace delimited arguement
  - argv[0] corresponds to $0, argv[1] to $1, etc, so argv[0] is the name of the executable.
- `args.cc` contains the following program. It prints each argument to standard output.

```
#include <iostream>
using namespace std;

int main(int argc, char *argv[]) {
  cout << "Number of arguments: " << argc - 1 << endl;
  cout << "Arguments: " << endl;
  for (int i = 1; i < argc; ++i) {
    string theArg = argv[i];
    cout << " " << i << ": " << theArg << endl;
  }
}
```