# CS 246 Fall 2013 - Tutorial 3

October 1, 2013

## 1 Topics

- Strings
- Filestreams
- Constants and Pointers

## 2 Strings

- Accessed through

  ```
  #include <string>
  ```

- Encapsulates a C-string (e.g. a char *)
- Has length, insert, delete, search methods
- Can be accessed like a C-String

```
1 string str = "Gotta go fast";
2 cout << str[5] << endl;
3 // Safer to use the .at() method, which has a range check
4 cout << str.at(5) << endl;
```

## 3 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`)

### 3.1 `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

### 3.2 `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

### 3.3 Basic Example

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

int main(){
  ifstream ifs("infile");
  ofstream ofs("outfile");
  string s;
  ifs >> s;
  if (! ifs.fail())
    for(int i=s.size()-1; i >= 0; --i) ofs << s.at(i);
  ofs << endl;
}
```

**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**.

## 4 Fill in the Blanks - `numberFilter.cpp`

**Problem:** Read the first n lines of a provided filename and write each number in that file that is divisible by a given number to a provided output file.
The program has the following input:

```
> ./numberFilter
n
inFile
10
outFile
```

**Note:** Multiple numbers can occur on a single line. Assume all numbers are integers. Assume there will not be more than 100 numbers. Assume there will not be more than 10 lines. No error checking of input is required.
We can break this problem into 5 subproblems:

**Step 1.** Write the mainline program. Get the 4 arguments from standard in, open the filestreams, call appropriate functions.

**Step 2.** Read the first n lines of the provided input file.

**Step 3.** Read numbers from each the lines read and update the count of numbers seen.

**Step 4.** Filter the numbers by divisibility by the provided divisor and update the number of divisible integers.

**Step 5.** Output the divisible numbers to the appropriate file.

**Step 1** has been done for you.

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

// Step 2.
void readLines(const int numLines, ifstream &file, string lines[]) {
  // Assume provided number of lines exist in the inFile
  for(int i=0; i< numLines; ++i ) {
    getline(file,lines[i]);
  }
}

// Step 3.
void getNumbers( int *numNumbers, const int numLines, string lines[], int numbers[]) {
  for(int i=0; i<numLines; ++i) {
    istringstream ss(lines[i]);
    for(;;) {
      ss >> numbers[*numNumbers];
      if(!ss) break;
      (*numNumbers)++;
    }
  }
}

// Step 4.
void filterNumbers(const int numNumbers, const int numbers[], const int divisor,
                   int filteredNumbers[], int* numFilteredNumbers) {
  for(int i=0; i<numNumbers; ++i) {
    // Why check if divisor is 0 first?
    if(divisor!=0 && numbers[i]%divisor == 0) {
      filteredNumbers[*numFilteredNumbers] = numbers[i];
      (*numFilteredNumbers)++;
    }
  }
}

// Step 5.
void outputFilteredNumbers( const int numFilteredNumbers, ofstream &outfile, const int filteredNumbers[]) {
  for(int i=0; i<numFilteredNumbers; ++i) {
    outfile << filteredNumbers[i] << endl;
  }
}

const int maxLines = 10;
const int maxNums = 100;
int main() {
  int numLines;
  string fileName;
  int divisor;
  string outfileName;

  cin >> numLines;
  cin >> fileName;
  cin >> divisor;
  cin >> outfileName;

  string lines[maxLines];
  int numbers[maxNums];
```

```
  int numNumbers = 0;
  int filteredNumbers[100];
  int numFilteredNumbers = 0;

  // Open Filestreams
  ifstream file(fileName.c_str()); //assume file exists
  ofstream outfile(outfileName.c_str()); //assume file exists

  // Call procedures
  readLines( numLines, file, lines);
  getNumbers(&numNumbers, numLines, lines, numbers);
  filterNumbers(numNumbers, numbers, divisor, filteredNumbers, &numFilteredNumbers);
  outputFilteredNumbers(numFilteredNumbers, outfile, filteredNumbers);
}
```

Notice that we have used pointers as parameters to procedures when we want changes to a variable in a procedure to be reflected in the main function. However, make note that arrays are treated as pointers (read: arrays are never copied by value). However, using pointers to pass back changes appears to be tedious and it would be nice if we had a better mechanism. In addtion, notice that constants are used when we do not want a particular value to be changed. Constant parameters are used to indicate that values should not be changed.

## 4.1 Testing

What would be some good tests to include in a test suite for this problem?

**Answers:**

- An empty file
- Amount of numbers is 100
- Number of lines is 10
- Amount of filtered numbers is 0
- Amount of filtered numbers is 100
- Divisor is 0
- Different amounts of whitespace
- Empty line in infile