Parsing Files &
Object Oriented Programming

File Input - Using “space” as delimiter

```
ifstream fin;
fin.open("book.txt");
if (!fin.is_open()) {
   return 1;
}
while (!fin.eof()) {
   string tmp_string;
   int tmp_int;
   // read non-blank characters
   fin >> tmp_string >> tmp_int;
   cout << "Text: " << tmp_string << endl;
   cout << “Integer: " << tmp_int << endl;
}
fin.close();
```

File Input Strategies

- What if the input file does not delineate text with spaces?
  - E.g. “student_name,grade,gpa,”
- getline(cin, dest_string)
  - Reads an entire line at once
  - Previously used this when accepting user input from the console
- getline(cin, dest_string, ‘,’)
  - Keeps reading text until reaching the specified character
  - Discards the specified character
  - Can be used to handle an alternate delimiter (e.g. comma)

The Pesky Newline Character

- Most user-readable files use newlines
  - Makes the text much easier to read
- Often used to indicate “new entry”
  - Make sure that your code handles these correctly

Example text file contents:

```
Apples,Oranges,Grapes,Pears,<newline>
Soccer,Baseball,Track,Volleyball<newline>
<blank>
```

The Pesky Newline Character

- std::istream::ignore()
  - Discards one or more characters from the input stream
  - Useful for discarding newline characters
- Common usage:
  - cin.ignore() →Throw away the next character

Example text file contents:

```
Apples,Oranges,Grapes,Pears,<newline>
Soccer,Baseball,Track,Volleyball<newline>
<blank>
```

What about writing to a file?

- You control the delimiters, newlines, etc
- Easier to handle

```
string output_file = "book_stats.txt";
ofstream fout;
fout.open(output_file.c_str(), os::trunc);
if (!fout.is_open()) {
  cout << "Error, unable to open file!" << endl;
  return 1;
}
fout << "Hello world!“ << endl;
fout.close();
```

Classes

- We are now moving into the concept of OOP → Object Oriented Programming
- Classes are very similar to structs
  - Structs are collections of data
  - Classes can have collections of data and perform operations
  - Classes are simply more powerful

Why bother using a class?

- Structs can’t “do” anything
- Classes can have functionality built in
- Example… mystring.length()
  - mystring is a string object
  - mystring has an internal member variable that tracks the length
  - length() is a member function

Basic Example

- Suppose that we create a Point class
  - It contains an X value and a Y value
  - We can create member functions to move the point, display the value, or perform other manipulations
- See code example

Vocab

- Struct: an object without any member functions; collection of data items of diverse types
- Class: an object with both member variables and member functions
- Object: instance of the class
- Member Variable: variable that belongs to a particular struct/class
- Member Function: function that belongs to a particular class
- Encapsulation: the details of implementation of a class are hidden from the programmer who uses the class

Classes v Structs

- Structs
  - No functionality
  - Default public
- Classes
  - Functionality
  - Default private