Week 8

Diana Gage

www-personal.umich.edu/~drgage

slides by Diana Gage and Leah Bar-On Simmons

Today:

- File I/O
- Classes

File I/O

- Will be key in your next project/some of the final projects!
- Alternative to standard I/O
  - Reading in from keyboard, printing to screen
- File input – reading in from a file
- File output – writing to a file

File I/O

- **#include <fstream>** to have access to these datatypes:
  - Ifstream
  - Ofstream

- With these, you can declare variables so that you can read from/write to files!

<iostream> vs <fstream>

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

int main() {
    int x;


    cin >> x;

}
```

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

int main() {
    int x;
    ifstream input_file;
    input_file.open("filename");
    input_file >> x;
    input_file.close();
}
```

My suggestion: name your ifstream “**fin**”

What about writing to files?

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

int main() {
    int x = 42;
    ofstream output_file;
    output_file.open("filename");
    output_file << x;
    output_file.close();
}
```

My suggestion: name your ofstream “**fout**”

Stream States

- Good

Everything is great!

- Fail

Non-fatal Error - failed to read expected data

Examples: *failed to convert type* or *file does not exist*

- Bad

- EOF

When Reading in From Files:

- **DO NOT** use while (!fin.eof()) {…} to stop reading in → *undefined behavior*, different things on different compilers

- Instead, use these:
  - while (fin >> x) {…}
  - while (!fin.fail()) {…}

- Fail bit will be set to **true** when end of file is reached, and/or when fin fails

Clearing a fail state

- What happens when reading **fails**? → we need to fix it to keep reading!
- cin.clear() or fin.clear()
- Use a **junk** variable to get rid of what caused the fail state!

Remember this:

```
int x = 0;
string junk;
 cin >> x;
while (!cin.fail()) {
   if (cin.fail()) {
        cin.clear();
        cin >> junk;
    }
   cin >> x;
}
```

Goal: read into int variable x.

cin is expecting data of type int, ***but user could enter something else! → cin enters <u>fail state</u>***

To keep reading: clear the fail state, and use junk string variable to store unwanted input

File I/O Exam Practice Questions! ☺

Classes: Review

- A class is a container that can hold **different** types of objects (objects with different data types)

- Another type of container we've learned, where all objects must be the **same data type**: array

Classes: Review

- A class is a container that can hold **different** types of objects (objects with different data types)
- A class is a user-defined data type
  - Just like int and double and string are data types, so is the class you define
- A class is a way to group together related information

Header files vs. source files

- Header files have the .h extension
- Source files have the .cpp extension

- Header files are where class definitions and function declarations go
- Source files are where function implementations go → remember :: (*source resolution operator*)
  - Ex: with Person class, in .cpp file with int getter –

int Person::get_num()

Public vs. Private

- A private member variable or member function means that only members of the class can access it
- Member variables are usually private, and we use setters and getters to access them
- A public member variable or function means that any part of the code can access that function or variable. Setters and getters are always public.

Passing Classes into Functions

- You almost always want to pass classes **by reference** into functions

- Why?
  - *Think of arrays: arrays are **automatically** passed by reference because they are large containers (take up a lot of space in memory)*
  - *Classes are also containers that usually hold a lot of information → **who wants to use pass by value and copy all of that??***

const with functions

- Const after the statement means the member function is not allowed to modify the class's member variables (read-only)
- i.e.
  - string Card::printCard() const{}
- Important to understand which functions should be const, and which shouldn't
- **You cannot call non-const functions from const functions**

const parameters

- ie bool goodCard(const Card& card1,
  const Card& card2){}
- The const means the cards passed in as parameters cannot be modified by the functions

Classes: Practice Question

- Which of the following statements is FALSE?
  a. The entities which follow line 7 cannot be used outside of the class
  b. The private declaration is in effect until it is changed
  c. Only the entity on line 8 is private; line 9 is not private
  d. There is no syntax error on line 4
  e. You can access x and text inside member functions of MyClass

```
1     class MyClass
2     {
3     public:
4         MyClass( int, string );
5         void display();
6
7      private:
8         int x;
9         string text;
10      };
```

Classes: Practice Question

- Which of the following statements is FALSE?
  a. The entities which follow line 7 cannot be used outside of the class
  b. The private declaration is in effect until it is changed
  c. **Only the entity on line 8 is private; line 9 is not private**
  d. There is no syntax error on line 4
  e. You can access x and text inside member functions of MyClass

```
1      class MyClass
2      {
3      public:
4          MyClass( int, string );
5          void display();
6
7       private:
8          int x;
9          string text;
10       };
```

Today: write our own class

- Our own version of Stay in the Blue App ☺
  - Using the charts on: http://www.brad21.org/bac_charts.html

    ***slightly modified….*

- We will have:
  - Header Files
  - .cpp files with implementations
  - Main.cpp to use our classes

- **TOPICS COVERED IN EXERCISE:**
  - Primarily Classes
  - Also: 2D arrays, 1D arrays

In the Header files (.h)

- Some member functions
- Public and private members
- Default constructors
- Non-default constructors
- Getters
- Setters

.cpp files

- Have all the implementations of the declarations in the header files
- Be careful with syntax here!!!

In main.cpp we want to…

- Make 2 instances of the Person class, 3 instances of the Drink class
- Use a default constructor at least once to practice (not in Assignment 4)
- Access elements and change them, using getters and setters

Final Product

- Our final product will be uploaded to discussion resources along with these slides
- Think about ways you could improve our code!

Plan of Attack:

- Implement functions in Drink.cpp first
- Then functions in Person.cpp
  - Some of these require Drink objects, so Drink.cpp needs to be completed!
- Then main!