CS180 - Error Handling

Announcements
- Program due tonight
- Office hours 2-3
- Next hw out today or tomorrow

(No check-in for this program.)
- readme, comment & indent
Large Projects

In C++, we often separate a class into multiple files.

- Easier version control.
- Allows division of files.
- Easy reference for later use.
Header files are used to declare the interface of a class or function. Don't actually define or program the code here!

Example: CreditCard.h

private variables

each function is listed
```cpp
#ifndef POINT_H
#define POINT_H
#include <iostream> // need ostream definition for operator<< signature

class Point {
private:
  double _x;
  double _y;
public:
  Point(double initialX=0.0, double initialY=0.0); // in-lined function body
  double getX() const { return _x; } // in-lined function body
  void setX(double val) { _x = val; } // in-lined function body
  double getY() const { return _y; } // in-lined function body
  void setY(double val) { _y = val; } // in-lined function body
  void scale(double factor);
  double distance(Point other) const;
  void normalize();
  Point operator+(Point other) const;
  Point operator*(double factor) const;
  double operator*(Point other) const;
}; // end of Point class

// Free-standing operator definitions, outside the formal Point class definition
Point operator*(double factor, Point p);
std::ostream& operator<<(std::ostream& out, Point p);
#endif
```
Cpp files

We then have 2 kinds of cpp files:

- One to declare functions. (CreditCard.cpp)

- One to test program (contains the main function). TestCreditCard.cpp
```cpp
#include "Point.h"
#include <iostream> // for use of ostream
#include <cmath> // for sqrt definition
using namespace std; // allows us to avoid qualified std:istream syntax

Point::Point(double initialX, double initialY) : x(initialX), y(initialY) {}

void Point::scale(double factor) {
  x *= factor;
  y *= factor;
}

double Point::distance(Point other) const {
  double dx = x - other.x;
  double dy = y - other.y;
  return sqrt(dx * dx + dy * dy); // sqrt imported from cmath library
}

void Point::normalize() {
  double mag = distance(Point()); // measure distance to the origin
  if (mag > 0)
    scale(1/mag);
}

void scale(...) {}
Compiling

Complication: main can’t run without functions or classes!
(include relevant .h file)
Need to compile in correct order.

So:

```
g++ -o TestCreditCard CreditCard.cpp
```

OR

```
g++ CreditCard
```

```
g++ -o TestCreditCard TestCreditCard.cpp
```
Alternative:

Makefiles are used to automate this. I generally provide this.

If you use the names I suggest, you can just type "make" at command prompt.

(I'll post a template of how these work...)
Error Handling

In C++, we do error handling by throwing exceptions.
(These are really just classes themselves.)

What exceptions were there in Python?

Index Out Of Bounds
TypeError
NameError
ValueError

classes
C++ Exceptions

The book uses its own error classes.
(See end of Ch. 2.)

Most of mine will be based on C++'s included exception classes.

So:

```cpp
#include <stdexcept>
```

(check cplusplus.com)
Python:

def sqrt(number):
    if number < 0:
        raise ValueError('number is negative')

C++:

double sqrt(double number) {
    if (number < 0)
        throw domain_error("number is negative");
Example

MyInt Array class needs operator []

Code:

```
int & operator[](int index) {
    if (index >= _size)
        throw out_of_range("Index out of range");
    return _A[index];
}
```
To use:

```
My Int Array myarray;
// code to put data in
```

```
try {
    cout << myarray[73] << endl;
    catch (out_of_range e) {
        cout << e.what() << endl;
    }
    // returns the string of error message
```
Catching exceptions

```cpp
try {
    // any sequence of commands, possibly nested
} catch (domain_error& e) {
    // what should be done in case of this error
} catch (out_of_range& e) {
    // what should be done in case of this error
} catch (exception& e) {
    // catch other types of errors derived from exception class
} catch (...) {
    // catch any other objects that are thrown
}
```

< might just use 1 of these
Other errors

By default, cin doesn't raise errors when something goes wrong. Instead, it sets flags. Use cin.bad(), cin.fail(), etc., to detect these.

Can get a bit long...
Ex (p. 27)

```cpp
number = 0;
while (number < 1 || number > 10) {
    cout << "Enter a number from 1 to 10: ";
    cin >> number;
    if (cin.fail()) {
        cout << "That is not a valid integer." << endl;
        cin.clear(); // clear the failed state
        cin.ignore(std::numeric_limits<int>::max(), '\n'); // remove errant characters from line
    } else if (cin.eof()) {
        cout << "Reached the end of the input stream" << endl;
        cout << "We will choose for you." << endl;
        number = 7;
    } else if (cin.bad()) {
        cout << "The input stream had fatal failure" << endl;
        cout << "We will choose for you." << endl;
        number = 7;
    } else if (number < 1 || number > 10) {
        cout << "Your number must be from 1 to 10" << endl;
    }
}
```
File streams at errors

Similar to cin:

```cpp
void openFileReadRobust(ifstream& source) {
    source.close();  // disregard any previous usage of the stream
    while (!source.is_open()) {
        string filename;
        cout << "What is the filename? ";
        getline(cin, filename);
        source.open(filename.c_str());
        if (!source.is_open())
            cout << "Sorry. Unable to open file " << filename << endl;
    }
}
```