CS180 - Error Handling

Announcements
- Lab tomorrow
- HW due Friday
- Office hours 9-10am
Large Projects

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

- Easier version control.
- Allows division of files.
- Easy reference for later use.
.h files

Header files are used to declare the interface of a class or function. Don't actually define or program the code here!

Example: `Point.h`

Contains:
- private variables
- function declarations (public ones)
```cpp
#include <iostream>

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);
```

```
We then have 2 kinds of cpp files.

- One to declare functions.
  (Point.cpp)

- One to test program (and contain the main function).
```cpp
#include "Point.h"
#include <iostream> // for use of ostream
#include <cmath> // for sqrt definition
using namespace std; // allows us to avoid qualified std:ostream 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);
}
```
```cpp
int main() {
    Point p1(3, 2);
    Point p2(4, 5);
    cout << p1 + p2 << endl;
}
```
Compiling

Complication: main can't run without functions or classes!

Need to compile in correct order.

So:

```
g++ -o TestPoint Point.cpp  
```

OR

```
g++ Point  
g++ -o TestPoint TestPoint.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?

- Syntax error
- Runtime errors
- Value Error
- Type Error
- NameError
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>
```

[cppplusplus.com](http://cppplusplus.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

My Float Vec * add operator []

Code:

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

```cpp
MyFloatVec v1(3);
```

\[ v1[12] = 82.5 \] — might crash program

```cpp
try {
    cout << v1[5] << endl;
} catch (out_of_range e) {
    cout << e.what() << endl;
} // prints error message
```
Catching exceptions

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
    }

}
Other errors

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

Can get a bit long...

Ex: prompt user for a number between 1-10
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;
    } else if (cin.bad()) {
        cout << "The input stream had fatal failure" << endl;
    } else if (number < 1 || number > 10) {
        cout << "Your number must be from 1 to 10" << endl;
    } else 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;
    } else if (cin.bad()) {
        cout << "The input stream had fatal failure" << endl;
    } else if (number < 1 || number > 10) {
        cout << "Your number must be from 1 to 10" << endl;
    }
}
```
File streams & 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;
    }
}
```
Singly Linked Lists
A collection of nodes that together form a linear ordering.

- Easy to add

Diagram:
- Head
  - Node
  - LAX → STL → MSP
  - Picture
  - Memory
    - Head
      - 263
      - 1025
    - Node 3
      - MSP
      - 825
    - Node 2
      - LAX
      - 263
    - Node 1
      - STL
      - 122
Why this structure?

Note: This is not the same as the list class which we'll write later. (nor is it like Python lists)

This linked structure will show up in a lot of our data structures. Similar to arrays as a building block.

So why?

Certain operations are faster on a linked structure.
Implementation

What is a node and how do we code it?
  - Separate class or struct

Private data?
  - Pointer to the head
  - (may include _size)

Functions?
  - Insert
  - Delete
  - Edit or return date
  - Is empty or size
```cpp
template <typename Object>
class SLinkedList {

private:
    class SNode {
        private:
            Object * elem;
            SNode <Object>* next;
    };

    SNode <Object>* _head;
```
Functions (listed in .h file)

public:

SLinkedList();
SLinkedList(SLinkedList);  // constructor
bool empty() const;
const Object& front() const;
void addFront(const Object& e);
void removeFront();