CS180 - More on C++

Note Title 9/19/2011

Announcements

- There is a grader for this class.
- BBQ today! 4pm, lobby of Ritter
- Lab tomorrow, prelab via email
- Next HW - due Monday.

Large Projects

In C++, we often seperate 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:

- what private variables
- declarations/types for functions

Point.h

if Point hasn't already been loaded

```
#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);
   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
```

private variables

just declarations

Cpp files

We then have 2 kinds of cpp files.

- One to declare functions.

- One to test program (& contain the main function).

Point.~~cpp~~ tcc

```
#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);
}
  ⋮
```

look for my .h file

scope function to Point class

testPoint.cpp

```
#include "Point.h"
#include <iostream>
using namespace std;

int main() {
    Point pt1;
    Point pt2(2.4, 5.1);

    Point pt3 = pt1 + pt2;
    cout << pt3 << endl;
    pt3.normalize();
    cout << pt3 << endl;
}
```

Compiling

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

Need to compile in correct order.

So:

output executable

```
g++ -o TestPoint Point.cpp
                 TestPoint.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, 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 ~~these~~ raised in Python?

ValueError
TypeError
NameError

C++ Exceptions

The book uses its own error classes.
(see cplusplus.com)

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

So:

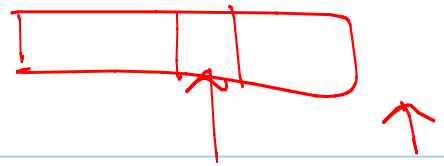
```
#include <stdexcept>
```

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

MyFloatVec : add operator []

Code:

```
float& operator[] (int index) {
    if ((index >= _size) || (index < 0))
        throw out_of_range("Index out of range");
    return _A[index];
}
```

To use:

v1: <0, 0, 0>

```
MyFloatVec v1(3);
// code to put data in

try {
    cout << v1[5] << endl;
}
catch (out_of_range& e) {
    cout << e.what() << endl;
}
```

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(), cin.fail(), etc., to detect these.

Can get a bit long... →

Ex (p.27)

```
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 & errors

Similar to cin:

```
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;
  }
}
```

More on arrays as private variables.

- In MyFloatVec class, want to store values in an array.

<u>Problem</u>:

- When writing the class, don't know how big to make the array!

Should work for any size.

Example Main:

```
int main() {

    MyFloatVec pt1(2);
    MyFloatVec pt2(3);

    pt1[0] = 2;
    pt1[1] = 4.2;

    pt2[0] = pt2[1] = pt2[2] = 0;

} // call to destructor
```

Solution (last time): make appropriate constructor, & array is a pointer:

```
class MyFloatVec {

private:
    int _size;   // size of this array
    float *_A;   // pointer to my array

public:
    MyFloatVec ( int s=10) : _size(s) {
        _A = new float[_size];
    :}
```

① <u>Destructor</u>: for every new, have delete

```
~MyFloatVec() {
    delete[] _A;
}
```

pt1 = pt2;

② Copy Constructor

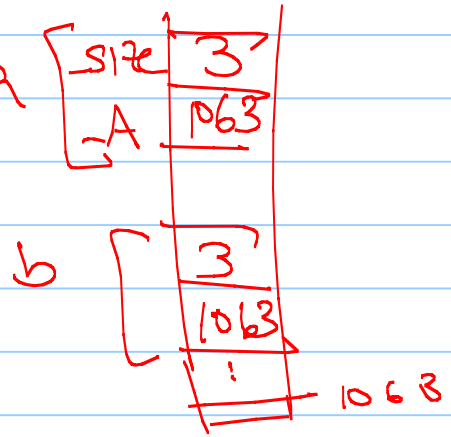
Consider that MyFloatVec class.

What if we have 2 and say a=b, or MyFloatVec b(a)?

By default, goes through private variables & sets them equal.

```
b._size = a._size;
b._A = a._A;
```

Shallow copy.



MyFloatVec b(a);

To avoid shallow copies, we need to make a copy constructor function.

in class!

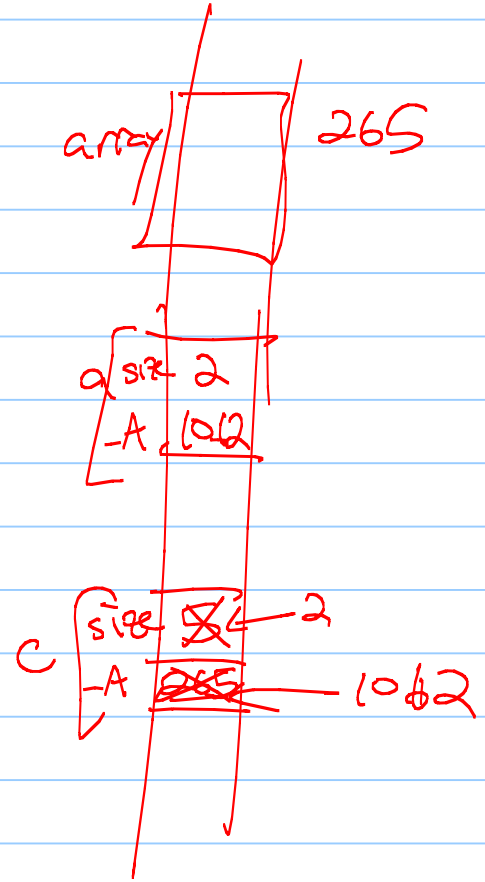
```
MyFloatVec(const MyFloatVec & other) {
    _size = other._size;
    _A = new float[_size];

    for(int i=0; i < _size; i++)
        _A[i] = other._A[i];

    // deep copy of array
}
```

Another issue: operator =

```
MyFloatVec c;
c = a;
```

What does this do?

- shallow copy
- memory leak!



c = a = a; a = b = c;

Solution: rewrite the "=" operation

```
MyFloatVec Operator=(const MyFloatVec& other) {
    if (this != &other) {

    }
    return *this;
}
```

Recap: Housekeeping Functions