CS2100 - More on C++

Note Title 9/19/2011

Announcements

- HW is due today
- HW2 will be posted today, due in ~1 week

Pointers in a class

Pointers are especially useful in classes.

Often, we don't know all the details of private variables to put in the private declaration.

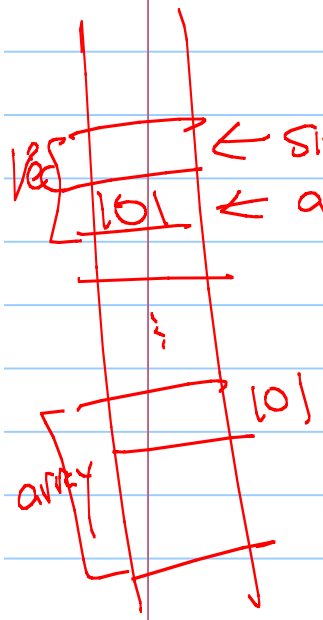
Example: arrays!

What do we need when creating an array?

type & size

↑ pts

```
class MyFloatVec {
  private:
    int size;      // dim of space
    float* a;      // ptr to array holding #s
  public:
    MyFloatVec( int s=10 ) : size(s) {
      a = new float[size];
    }
    :
```



Accessing the array:

With an array, can just pretend the variable isn't a pointer.
(so no * or →)

Ex: override the [] so that x[i] will give back the i^{th} element in the vector:

(x: vector)

```
float operator[](int index) {
    return a[index];
}
```

Function to scale by int (in class):

```
void scale(float value) {
    for (int i=0; i<size; i++)
        a[i] *= value;
}
```

Garbage Collection

In Python, variables that are no longer in use are automatically destroyed.

Pros: easy!, no memory leaks (no junk data reserved)

Cons: speed

C++

In C++, things are sometimes handled for you.

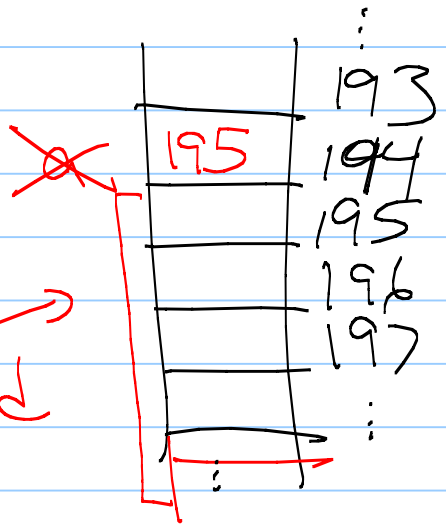
Basically, any standard variable is automatically destroyed at the end of its scope.

This holds for <u>any</u> type of variable!

Problem: Pointers

While the pointer variable is deleted, the spot you created with "new" is not.

```
int main() {
    int * a = new int(5);
    //put stuff in a
    //need:
    delete[] a;
} // a is destroyed
```



Rule: If you have a new, must have a delete!

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:
- private data
- fcn declaration

Point.h

```
#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
```

Compiler should load this class if it hasn't done it before

Cpp files

We then have 2 kinds of cpp files.

- One to declare functions. class
  (might see .tcc) ↳ Point.cpp
- One to test program (& contain the main function).
  ↳ testPoint.cpp

Point.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::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);
}
```

Scope to class

testPoint.cpp

```
#include "Point.h"
#include <iostream>
 - - -
using namespace std;
int main() {
    Point x, y;
    x.setX(6);
    y.setX(x.getX() + 12);
     :
}
```

(include cpp)

Compiling

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

Need to compile in correct order.

So:

```
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, 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 raised in Python?

ValueError ← 6 + "12"

TypeError ← x = 5
x.length()

⋮

C++ Exceptions

The book uses its own error classes.

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

So:

```
#include <stdexcept>
```

↳ more on 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

MyFloatVec : add operator []

Code: at top: #include <stdexcept>
using namespace std;

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

To use:

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

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

↑ prints "Index out of range"

!

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, wanted to values in an array.

Problem: that array was created with a new!

So what do we need (somewhere?)

delete! (somewhere)

Example Main:

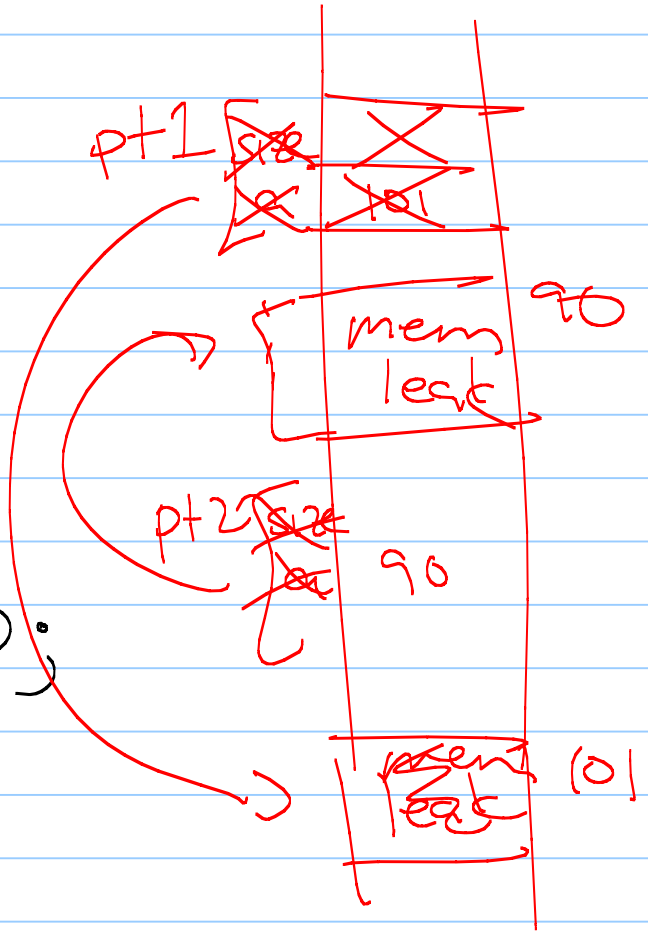
```
int main() {

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

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

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

} // pt1 & pt2 get destroyed
```



Destructor:

in class

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

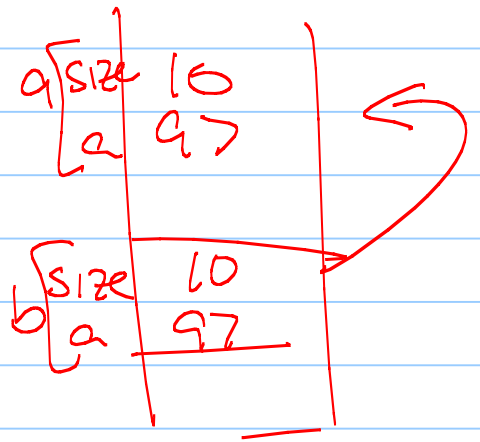
Copy Constructor

Consider that MyFloatVec class.

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

by default, sets private variable =.

Shallow copy (Bad!)



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

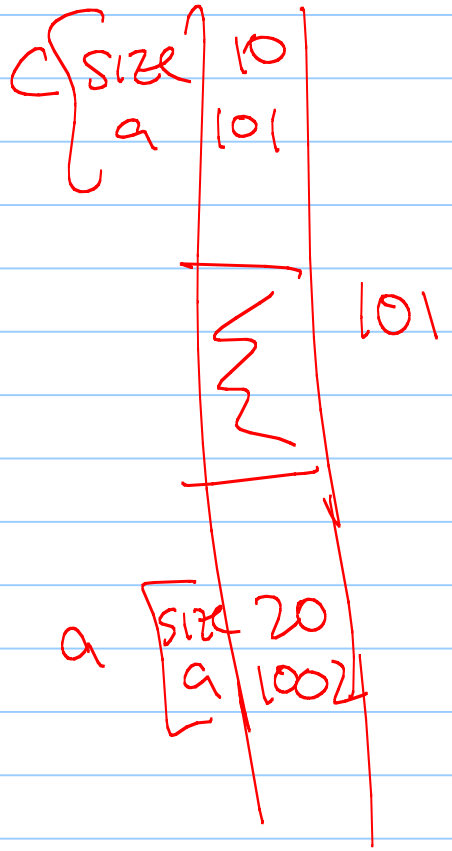
```
MyFloatVec(const MyFloatVec & other) {
    size = other.size;
    a = new float[size];

    for (int i=0; i<size; i++) {
        a[i] = other.a[i];
    }
}
```

Another issue:

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

What does this do?



Solution: rewrite the "=" operation

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

    }
    return *this;
}
```

Recap: Housekeeping Functions