CS180 - Classes + Exceptions

Note Title 1/31/2011

Annoncements

- HW2 is posted
- Get a copy of Ch. 2
- Textbook will come out this week
  I'll post details
- HW1 is graded.
  Expect an email w/ your grade in next 1-2 days.
- Lab tomorrow - prelab is due by 10am

A word on cheating

- Do **NOT** look at another student's code
- Do **NOT** look at webpages for solutions!

(I only allow course materials plus cplus plus.com)

More on Classes:

Destructors:

If your class opens files or allocates memory, then can't just use delete.

new

Must create a <u>destructor</u>.

~ClassName() - no arguements, no return type

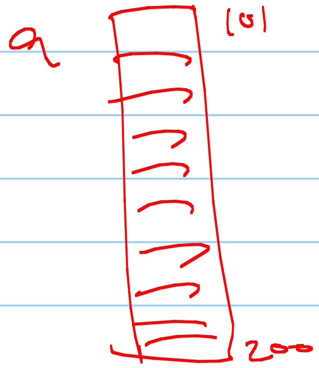
```
~Point() {
    delete commands
}
```

Copy Constructors:

(House keeping functions)

Previously:

```
Point a;
Point b(a);
```



Consider the following:

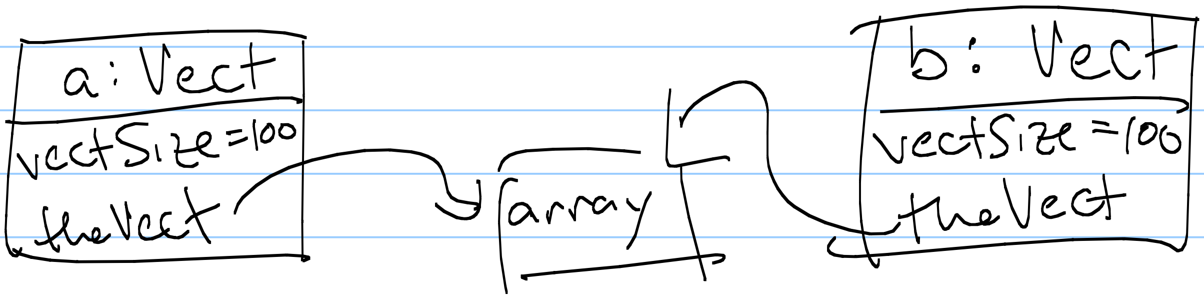
```
Vect a(100);
Vect b(a);
```

What does this do?

(in C++, by default)

Copies each element of a to b:

```
vectSize = a.vectSize;
theVect = a.theVect;
```



shallow copy!

not good - so we'll override this

To fix, write our own copy constructor:

```
  Vect b(a);

    // copy constructor
    Vect (const Vect &a) {
        vectSize = a.vectSize;  // copy size
        theVect = new int[vectSize];  // new Vector
        for (int i = 0; i < vectSize; i++) {
            theVect[i] = a.theVect[i];
        }
    }
```

name of class

Another problem:

```
Vect a(100);
 Vect c;
  c=a;
```

What does this do? shallow copy
by default, copies each parameter:

```
c.vectSize = a.vectSize;
c.theVect = a.theVect;
```

Write $\underbrace{\text{operator =}}_{\text{3rd housekeeping function}}$ to make deep copy of data.

Enum: user defined types

$$\text{enum Color } \{\underset{\substack{\uparrow\\ 0}}{\text{RED}}, \underset{\substack{\uparrow\\ 1}}{\text{BLUE}}, \underset{\substack{\uparrow\\ 2}}{\text{GREEN}}\};$$

Color sky = BLUE;
Color grass = GREEN;

Convention: write in all capital letters
(not enforced by compiler)

Structures: (legacy from C)
useful for holding collections of objects

Ex:

```
enum MealType {NO_PREF, REGULAR, VEG};
                  0        1      2
struct Passenger {
    string name;
    MealType mealPref;
    bool isFreqFlyer;
    String freqFlyerNo;
}
```

Could use a class.
No functions, no public/private —
stripped down class!

Using Structures

Structures can then be used inside the program:

```
Passenger pass = {"John Smith", VEG, true, "1234"};
pass.mealPref = REGULAR;
```

Another example:

```
Passenger *p;                  Can't say p.name = "Erin"
     pointer + new
p = new Passenger;
p->name = "Barbara Wright";
p->mealPref = NO_PREF;
p->isFreqFlyer = false;
p->freqFlyerNo = "NONE";
dereferencing

(*p).mealPref = VEG;
                enum

if (p->mealPref == 1)   // test if meal preference
                        //      is regular
```

Function templates:

```
template <typename T>
T min (T a, T b) {
    if (a<b)
        return a;
    else
        return b;
}
```

tells compiler I don't know what type to use, + needs to work for multiple types

Important: Will work for any class, as long as "<" has been defined!

operator <

Example:

```
int x = 53;
int y(69);

int z = min(x,y);
    ↑ 53

string a = "Hello";
string b = "Goodbye";

cout << min(a,b) << endl;
           ↑ Goodbye (?)
```

Class templates : a vector example

whatever is in < >

```
template <typename Object>
class BasicVector {
private:
        Object* a;          // array of elements
        int capacity;       // length of array a
public:
        BasicVector(int c=10) {     //constructor
                capacity = c;
                a = new Object[capacity]; //allocate storage
        }

        Object& elemAtRank(int r)   //access rth element
                {return a[r];}
        ⋮
```

Back to BasicVector: usage

```
BasicVector <int>    intvec(5);  //vector of 5 ints
BasicVector <string> strvec(10); //vector of 10 strings

intvec.elementAtRank(3) = 8;  //sets 4th element = 8
strvec.elementAtRank(7) = "hello"; //sets 8th elt = "hello"
```

Or even:

```
BasicVector < BasicVector<int> >  myvec(5);
        //vector of 5 BasicVectors of integers

 myvec.elementAtRank(2).elementAtRank(8) = 15;
      //  myvec[2][8] = 15
```

Error Handling

In C++, we do error handling by throwing exceptions.

(These are really just classes themselves.)

What exceptions did we have in Python?

$$\left.\begin{array}{l}\text{Type Error}\\ \text{Name Error}\\ \text{Syntax Error}\\ \text{Add Value Error}\end{array}\right\}\ \text{classes}$$

Exceptions in C++

The book uses its own error classes. (at end of Ch2)

Most of mine are based on C++ default exceptions.

So:

```
#include <stdexcept>
```
← (at top of file)

Example:

In BasicVector, might want to allow you to access the ith element:

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

```
BasicVect <int>   myvec;
 //fill in vector
try {
    cout <<   myvec[73] << endl;
}
```

$\underbrace{\text{catch}}_{\text{next time}}$