CS180 - Other bits of C++

Announcements
- Program 4 due Monday (the 19th), 2011
- Submit lab by Sunday
Last time

- Finished pointers
  - new
  - dereferenced
  - arrays
  - delete
- Destructor

~ClassName() {
  deletes...
}
Copy Constructor

Consider that MyIntArray class.

What if we have 2 MyIntArray arrays, set a = b?

By default, compiler sets each private variable equal to other.

a.size = b.size
a.A = b.A

Shallow copy
Copy constructor

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

MyIntArray(const MyIntArray & other) {
  size = other.size;
  A = new int[-size];
  for (int i = 0; i < size; i++)
    A[i] = other[i];
}

3
Another issue:

MyIntArray c; //
c = a;

What does this do?

In a class, by default, sets each & private variable equal to corresponding variable in a.

(just like copy constructor)

Shallow copy
Solution: rewrite the "=" operation

MyIntArray::operator=(const MyIntArray &other) {
    size = other.size;
    delete A;
    A = new int[-size];
    for (int i = 0; i < -size; i++)
        A[i] = other.A[i];
}

Housekeeping Functions

1. Destructor
2. Copy Constructor
3. Operator =
Enum: user defined types

```cpp
enum Color {RED, BLUE, GREEN};
```

Color sky = BLUE;

Color grass = GREEN;

if (sky == BLUE)
    cout << "It's nice out today!" << endl;
Structs - simple class

useful for simple collections of objects

Ex: enum MealType { NO_PREF, VEG, REGULAR, KOSHER, }

struct Passenger {
    string Name;
    MealType mealPref;
    bool isFrequentFlyer;
    string frequentFlyerNo;
}
Using structs

We can then create instances of a struct in the program:

Passenger pass = new "John Smith", VEG, true, "123-4567890";
Passenger other = new "Jane", REGULAR, false, ""
pass.mealPref = KOSHER;

↑ no private data in a struct
More Complex: use as a pointer

Passenger* p;
p = new Passenger;
p->name = "Barbara Wright";
p->mealPref = REGULAR;
(*p).isFrequentFlyer = false;
(*p).freqFlyerNo = "None";
Templates

If we want a function to work for multiple classes - e.g. `int` and `float` - we can template the variable type.

Ex:

```cpp
template <typename T>
T min(T a, T b) {
    if (a < b)
        return a;
    else
        return b;
}
```
Important: will work for any class with appropriate operators!

Ex: int x = 53;
    int y(96);
    int z = min(x, y);

    for(min just need a class with <= operator.

    string a = "Hello";
    string b = "Goodbye";
    cout << min(a, b) << endl;
Templates in classes

These work in classes, also.

Important in data structures, so our code will make a list of ints or strings or lists.

1st line:

```cpp
template <typename ItemType>
```

```cpp
class List {
    private:
        ItemType* A;
    public:
```
int main() { 

    // Create a list
    List<int> myList; // creates a list of int

    List<string> names;

}
Error Handling