CS180 - Other bits of C++

Note Title 9/12/2011

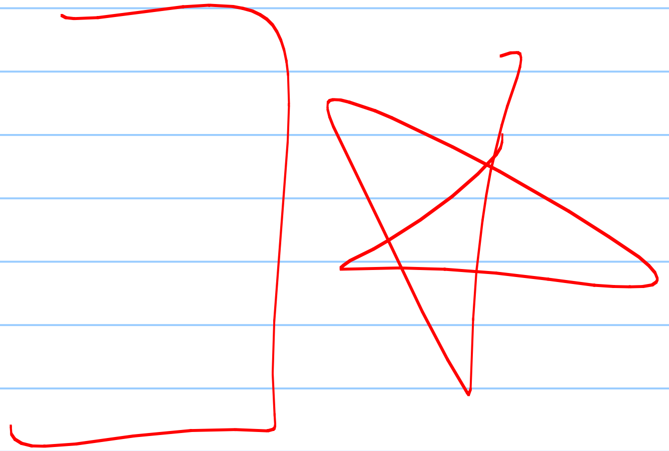
Announcements

- Program 1 due Monday (the 19th), 2011
- Submit lab by Sunday

Last time

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

```
~ClassName() {
    deletes...
}
```



Copy Constructor

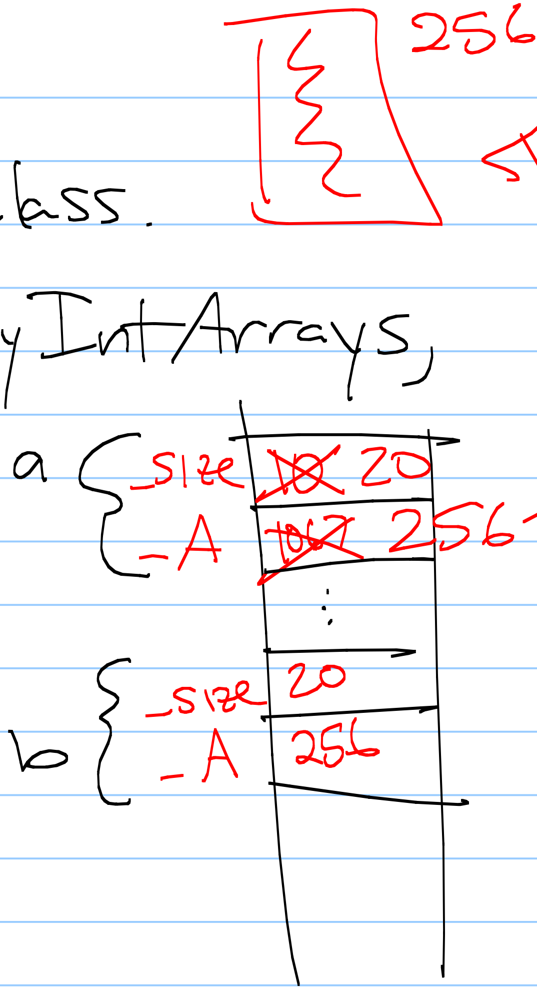
Consider that MyIntArray class.

What if we have 2 MyIntArrays, + 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

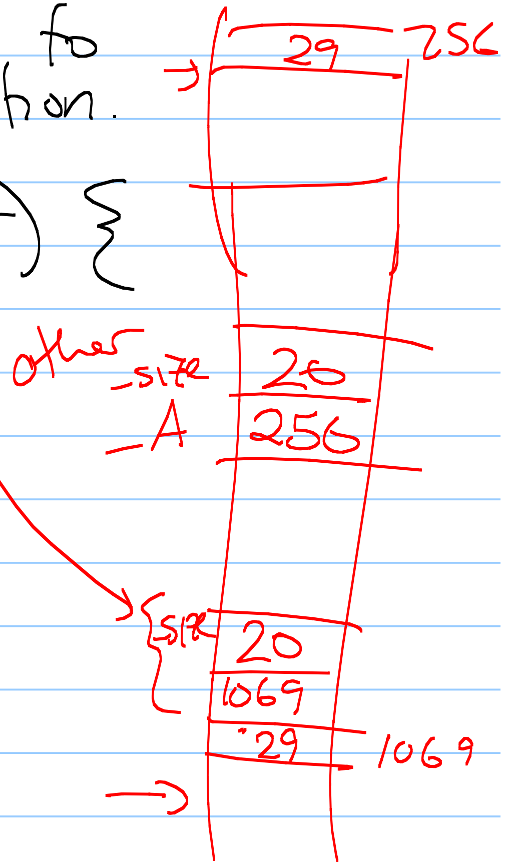
$\frac{i}{0}$

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._A[i];

}
```



Another issue:

```
MyIntArray c; ~~2~~
c = a;
```

What does this do?

In a class, by default, sets each of c's private variable equal to corresponding variable in a.

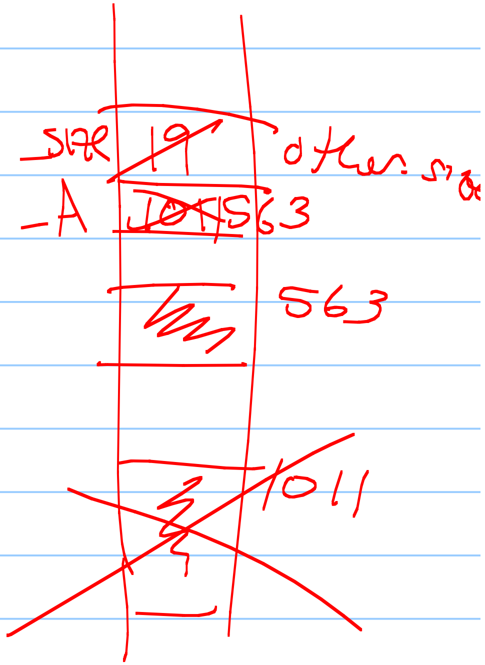
(just like copy constructor)

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];

}
```



House keeping Functions

1. Destructor
2. Copy Constructor
3. Operator =

book generally doesn't do these

Enum: user defined types

enum Color {RED, BLUE, GREEN };

" 0   " 1   " 2

Color sky = BLUE;

Color grass = GREEN;

if (sky == ~~BLUE~~ 1)

cout << "It's nice out today!" << endl;

int i;

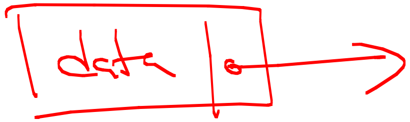
Structs - simple class

useful for simple collections of objects

Ex:

```
enum   MealType   {NO_PREF, VEG,
                   REGULAR, KOSHER};

struct Passenger {
      string    name;
      MealType   mealPref;
      bool      isFreqFlyer;
      string    freqFlyerNo;
}
```



Using structs

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

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

no private data in a struc

More Complex : use ~~as~~ a Pointer

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

Templates

If we want a function to work for multiple classes - eg int and floats - we can template the variable type.

Ex:

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

ItemType
Data
(variable)

type

Important: will work for any class with appropriate operators!

Ex:

```
int x = 53;
int y(96);

int z = min (x,y);


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

cout << min (a,b) << endl;
```

for min, just need a class with < operator.

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:

```
template <typename ItemType>
→ class List {
     private:
         ItemType* _A;
  } public:
```

```
int  main( )  {
   //create a List
   List<int> mylist;        <— creates a list
                                   of ints

   List <string> names;


}
```

Error Handling