

Chapter 8

All Stuff I Didn't Want to Tell You
...Until Now

Problem with the “best way” to do project 2

- Suppose you have this class

```
class Student {  
private:  
    ...//private variables  
    Course* CoursePtr;  
public:  
    //public stuff  
};
```

Problem with the “best way” to do project 2

- What happens if the student object has a dynamic array inside of it and you do something like this:
- `NewStuArrPtr[i] = StuArrPtr[i];`
- Memberwise copy works for most of it, except...
- The dynamic array portion

Shallow Copy

- The pointer gets the contents of the pointer in the other object...
- Can you change what one of the pointer points to without changing what the other pointer points to?
- Worse yet, what if this is inside of grow code?

Grow Code

```
void grow( Student *&sPtr, int& size)
{
    int newsize = size * 2;
    Student * temp = Student [newsize];
    for ( int i=0; i<size; i++ )
    {
        temp[i] = sPtr[i];
    }
    delete [] sPtr;
    sPtr = temp;
    size = newsize;
}
```

Deep Copy

- Solution is to provide what is know as a mechanism for a deep copy
- When you have dynamic data inside of a class, you should always supply three methods
 1. Copy Constructor
 2. Assignment Operator
 3. Destructor

Copy Constructor

- A copy constructor allows you to successfully create an object that is a copy of another
- e.g. `Student NewStudent = OldStudent;`
- This would invoke the copy constructor.
- The copy constructor would take care of creating and copying the course information

Copy Constructor

```
Student::Student(const Student& RHS)
{
    //you perform a memberwise copy
    CoursePtr = new Course[size];
    for ( int i=0; i<Used; i++ )
    {
        CoursePtr[i] = RHS.CoursePtr[i];
    }
}
```

Assignment Operator

- An assignment operator allows you to transfer a copy of an already existing object into an already existing object.
- e.g. StudentA = StudentB;
- This is a simple assignment statement.
- The difference between this and a copy constructor is the missing StudentA already exists

Assignment Operator

```
const Student& Student::operator=(const Student& RHS )
{
    if ( this != &RHS )
    {
        delete [] this.CoursePtr;
        //perform memberwise copy
        CoursePtr = new Course [size];
        for ( int i=0; i<Used; i++ )
        {
            CoursePtr[i] = RHS.CoursePtr[i];
        }
    }
    return *this;
}
```

Destructor

- For classes with dynamic data, a destructor is crucial
- It allows you to reclaim the dynamic memory at the end of the objects lifetime
- It is called automatically when the object expires

Destructor

```
Student::~~Student()  
{  
    delete [] CoursePtr;  
}
```

Stack Class

- How does a stack work?
 - FIFO: First In First Out
- For your next project you will need a stack that can grow and shrink as necessary.
- You will need to pass your stack, by value and hence invoking the copy constructor, into a function that will reverse the stack and print it out.