# WEEK 10

Derived Class with Resources

# OVERVIEW

- Week 10-1
  - Constructor & Destructors Redux
  - Copy Constructor & Copy Assignment Operator Redux

# CONSTRUCTORS

- A Default No Arg Constructor is provided by the compiler if not explicitly created
- Can have one or many constructors to create objects with
- A derived class' constructor will call its base's default constructor by default
- This can be overridden by explicitly calling a particular base constructor

# DESTRUCTORS

- Default is provided by the compiler if not explicitly created
- Only one destructor that cleans up the object
- A derived class' destructor will call its base destructor automatically

# Copy Constructor

- Default is provided by the compiler if not explicitly created
- The default CC only does shallow copying

# Copy Assignment Operator

- Default is provided by the compiler if not explicitly created
- The default CA only does shallow copying

# DEFAULT CC / CA

Consider the following class:

```
class MyClass
{
int x;
char c;
std::string s;
};
```

What do the default copy constructor and copy assignment operator look like?

# DEFAULT CC

Consider the following class:

```
class MyClass
{
int x;
char c;
std::string s;
};
```

```
MyClass::MyClass(const MyClass& other) : x(other.x), c(other.c), s(other.s)
{}
```

A default copy constructor provided by the compiler ends up looking like this. Notice how the variables are passed

This is known as an initializer list

# DEFAULT CA

Consider the following class:

```
class MyClass
{
int x;
char c;
std::string s;
};
```

```
MyClass& MyClass::operator=( const MyClass&
other ) {
        x = other.x;
        c = other.c;
        s = other.s;
        return *this;
}
```

This is what the default copy assignment provided by the compiler would look like

# CC / CA

- In the case of compiler provided CC and CA, they have no notion of handling dynamic resources (deep copying). For this we would need to deal in a manually defined CC / CA
- How does this notion carry into inheritance / derived classes with resources?

# DERIVED CLASSES WITH RESOURCES

Consider the following:

```
class Compound{
        int weight;
        char* name; // Dynamic
Resources
        ...
}
```

```
class Playdoh : public Compound{
        char* colour; // Dynamic
Resources
        ...
}
```

Are there any considerations for implementing the CC / CA in this context?

# COMPLAY EXAMPLE