Lecture 23: Classes and header/implementation files

… and in things that look kinda similar to your homework

Announcements and reminders

- HW7 posted, due Wednesday March 13, by 11 pm

Last time on *Intro Computing…*

We saw…

- How to define different types of **member functions**
    - Getters (accessors) and setters (mutators)
- How to access and mutate (get and set) **data members** from inside and outside of the class
    - That dot notation -- when and how to use it!
- What a **constructor** is, and how to use them!
    - Default constructor, vs
    - Parameterized constructor

Header files

Start with our Jedi class all in one function

jedi_allInOne.cpp

header file: `jedi.h`

→ the class definition (stencil)

→ like a function prototype

Header files

Start with our Jedi class all in one function

jedi_allInOne.cpp

header file: **`jedi.h`**

→ the class definition (stencil)

→ like a function prototype

```
#ifndef JEDI_H
#define JEDI_H
#include <string>
using namespace std;

class Jedi {
public:
    Jedi();
    Jedi(string s, int h, int p);
    void rest();
    ... other member functions...
private:
    int health;
    int power;
    string name;
};

#endif
```

Header files

Start with our Jedi class all in one function

jedi_allInOne.cpp

Class implementation file: **`jedi.cpp`**

→ the actual definitions of the class' member functions

Header files

Start with our Jedi class all in one function

jedi_allInOne.cpp

Class implementation file: **`jedi.cpp`**

→ the actual definitions of the class' member functions

```
#include <iostream>
#include <string>
#include "jedi.h"
using namespace std;

Jedi::Jedi() {
    health = 0;
    power = 0;
    name = "Padawan";
}
Jedi::Jedi(string s, int h, int p) {
    health = h;
    power = p;
    name = s;
}
void Jedi::rest() {
    health += 1;
}
... other member functions...
```

Header files

Start with our Jedi class all in one function

jedi_allInOne.cpp

Class implementation file: je

→ the actual definitions of

member functions

```
#include <iostream>
#include <string>
#include "jedi.h"
using namespace std;

Jedi::Jedi() {
    health = 0;
```

Fun fact: *< > versus " "*

<jedi.h> → searches the **include** path for jedi.h

more or less, where the junk native to your C++ would be stored

"jedi.h" → starts searching from the **current** directory

→ "jedi.h" is the one we want!
Because it's in the current directory

```
... other member functions...
```

Header files

Start with our Jedi class all in one function

jedi_allInOne.cpp

Driver file: `jedi_withSepFiles.cpp`

→ tests constructors/member functions

→ checks the data members

→ tests **functions that use our class!**

Header files

Start with our Jedi class all in one function

jedi_allInOne.cpp

Driver file: `jedi_withSepFiles.cpp`

→ tests constructors/member functions

→ checks the data members

→ tests **functions that use our class!**

```
#include <iostream>
#include <string>
#include "jedi.h"
using namespace std;

int main() {

  // test parameterized constructor
  Jedi vader("Darth Vader", 10, 13);
  vader.display();

  // test default constructor
  Jedi luke;
  luke.display();

  // test default constructor
  luke.set_name(“Luke”);
  luke.display();


... other tests of
                  member functions...
```

Arrays of Jedis … or of Jedi? Whatever.

Example: Create an array of everyone's favorite Jedis.

Then, write a function to find the most powerful!

→ ***jediCouncil.cpp***

What just happened?!

We just saw…

- How to define **header** and **implementation** files for our classes
  - ***Modularizes*** our code → easier to modify later! Keeps things tidy

- How to create **arrays** of our class objects

- … and how to use these arrays in **functions**
  - Like… I dunno… maybe an array of ***Book** objects…*?!