CSCI 1320 - Computer Science I: Engineering Applications

Instructor: Maciej Zagrodzki

Fall 2018

Announcements

- Lab 11 Due Friday, Dec. 14, by 6pm
- Final Project: interview grading this week
- Final Exam:
  - Dec 17th, 7:30 **PM**, 90 minutes, in Muenzinger E050 (here)
  - Multiple choice
  - 1 page of notes allowed (front and back)

Day's Objectives

- Structs vs Classes
- Recap: the constructor function
- Function overloading
- Overloading the constructor
- The scope resolution operator

Structs vs Classes

Both are used to group members associated with a single data entity.

- Both get defined globally (typically) .
- When we declare an object of struct or class, it is called an instance.
- By default, struct members are public and class members are private.
- No private members in structs.
- No member functions in structs.
- Above rules are a convention. Actually, structs and classes can do all the same things in C++.

The Constructor

Constructor: a special kind of member function that is called every time a new object is created.

- The constructor allocates the space for all the member variables.
- Can be user defined to take in arguments and initialize variables.
- Gets the same name as the class.
- No return type. (essentially a void function, but not specified as such)

The Constructor

The constructor can be defined in different ways, depending on what functionality is desired.

- Initialize the member variables to some default values.

Constructor Example

```
1  class Time12
2  {
3  private:
4      int hour;
5      int minute;
6      string mer;
7  public:
8      Time12( ){
9          hour = 12;
10         minute = 59;
11         mer = "AM";
12     }
13 };
```

- Any instance of this class will be initialized to 12:59.

- Member data can still be modified via mutator functions (not shown).

The Constructor

Alternatively, we can define the constructor to have input arguments which will initialize the member variables.

- Now declaration *requires* to be called with inputs.

Constructor Example

```
1  class Time12
2  {
3  private:
4      int hour;
5      int minute;
6      string mer;
7  public:
8      Time12( int h, int m, string me ){
9          hour = h;
10         minute = m;
11         mer = me;
12     }
13 };
14 int main()
15 {
16     Time12 t(5,20,"PM");
17     return 0;
18 }
```

Note: cannot declare w/o arguments now:

```
Time12 t;
```

User Defined Class: Example

Update the Time Class from previous example to add a constructor.

- Take in arguments and assign them to proper member vars.
- Object instantiation should look like this:

```
1    Time12 t(5,20,"PM");
```

Go to timeEg.cpp

Function overloading

C++ allows us to design functions to have multiple definitions.

- We define a function with the same name, but different arguments.
- Can have different number of arguments or different argument types.
- When the function is called, the program will use the definition that matches the arguments.
- Function overloading is often used within classes, especially for constructor definitions.
- Let's start with an example of a standard function:

Overloaded function

```
1  int foo( int x )
2  {
3     return 2*x;
4  }
5
6  int foo( int x, int y)
7  {
8     return x*y;
9  }
```

Go to overloadSB.cpp

User Defined Class: Example

Update the Time Class from previous example:

- Overload the constructor to have 2 definitions:

1. If declared with no arguments, assign some default values.
2. If declared with arguments, take the arguments and assign them to proper member vars.

Go to timeEg.cpp

Member functions: declarations + definitions

So far we have been defining member functions within the class definition.

- Can become cumbersome when class grows to have long function definitions.
- Just like with regular functions, we can place our declarations and definitions separately.
- The declaration goes inside of function definition, the definition goes after main.
- Use the scope resolution operator. ::

Scope resolution operator ::

```
1  //declaration inside of class def
2  void printDate();
3  .
4  .
5  .
6  //definition goes after main
7  void Time12::printDate()
8  {
9      cout << hour << ":" << minute << mer << endl;
10 }
```

Go to timeEgB.cpp