COP 3331
OBJECT ORIENTED DESIGN
SPRING 2017

WEEK 7: MORE ON OPERATOR OVERLOADING, FRIEND CLASSES, AND BRIEF REVIEW

SCHINNEL SMALL

USF
UNIVERSITY OF SOUTH FLORIDA

MORE ON OPERATOR OVERLOADING

OVERLOADING << AND >>

- In HW 3, you were asked to use functions that overloaded the stream insertion and stream extraction operators.

- As you saw, overloading >> and << required special syntax that was different that the ones used for the assignment, math and increment/ decrement operators

OVERLOADING >> AND <<

- << is actually part of the `ostream` class defined in the C++ runtime library
  - cout is an object that is an instance of ostream
- >> is actually part of the `istream` class defined in the C++ runtime library
  - cin is an object that is an instance of istream
- Referring to these classes require the inclusion of the `iostream` headerfile and the `std` namespace (as of C++ 11)

OVERLOADING >> AND <<

- Recall: "Overloaded stream operators must return reference to `istream`, `ostream` and take `istream`, `ostream` objects as parameters"
  - What does that really mean?

- Let's look at the general syntax for prototypes that overload the << and >> operators:

```
friend ostream& operator<<(ostream&, const className&);
```

```
friend istream& operator>>(istream&, className&);
```

OVERLOADING >> AND <<

```
friend ostream& operator<<(ostream&, const className&);
```

- The ostream reference object is denoted by the syntax `ostream&`
- The `ostream&` parameter will be a reference to the actual `ostream` object on the *left* of the << operator
- The second parameter is a reference to an object of type class, that will appear on the *right* side of the operator

OVERLOADING >> AND <<

- The general syntax for a function that overloads << is:

```
ostream& operator<<(ostream& osObject, const className& cObject)
{
      //local declaration, if any
      //Output the members of cObject.
      //osObject << . . .

      //Return the stream object.
     return osObject;
}
```

- This definition allows us to tell C++ how to handle expressions of the form `ostreamObject << classObject`
  - e.g.: `cout << fraction1`
  - The previous syntax is equivalent to `operator<<(cout, fraction1)`

OVERLOADING >> AND <<

```
friend istream& operator>>(istream&, className&);
```

- Similarly the istream reference object is denoted by the syntax `istream&`
- The `istream&` parameter will be a reference to the actual `istream` object on the *left* of the << operator
- The second parameter is a reference to an object of type class, that will appear on the *right* side of the operator

OVERLOADING >> AND <<

- The general syntax for a function that overloads >> is:

```
istream& operator>>(istream& isObject, className& cObject)
{
        //local declaration, if any
        //Read the data into cObject.
        //isObject >> . . .

        //Return the stream object.
    return isObject;
}
```

- This definition allows us to tell C++ how to handle expressions of the form `istreamObject >> classObject`
  - e.g.: `cin >> fraction1`
  - The previous syntax is equivalent to `operator>>(cin, fraction1)`

OVERLOADING >> AND <<

- The reason that the functions use reference parameters is because it is the most effective way to pass an object

- It also allows us to chain together the operators
  - e.g. `cout << fraction1 << fraction2;`

- The const (in the case of the ostream reference) allows access to the private members of the class without modifying the contents

FRIEND FUNCTIONS AND CLASSES

WHY USE THE WORD FRIEND?

- In most cases, we created functions that were part of our class definition
  - They were either defined inline or declared using their prototype
  - Those functions allows us to access the private member of the class because we can't do so directly

- However, in the case of << and >>, we are using the `ostream` and `istream` classes, which are defined elsewhere

WHY USE THE WORD FRIEND?

- We need the objects of type `ostream` and `istream` to access the members our class, even though we didn't define them

- The `friend` keyword allows a function or class that is not a member of our class to gain access to the private members of the class

- It is essentially an exception to the rule that states that private members be hidden from parts of the program outside the class

WHY USE THE WORD FRIEND?

- A function is declared a friend of the class, by placing the friend keyword in front of a prototype of the function
- The prototype must be included in the specification file, so that the class can keep track of its “friends”
- Once this declaration is made, the function can be defined in the implementation file as normal

WHAT TO EXPECT ON MIDTERM

MIDTERM EXAM INFO

- Date: Friday March 3rd
- Time: 9:00 am*
- Duration: ~75 minutes*
- Location: ENA*

*Time, duration and location may vary for students with prior authorization for accommodations)

MIDTERM EXAM INFO

- Multiple Choice (60%)
  - 15 questions @ 4 points each
  - No room for partial credit ☹
- Free form questions (40%)
  - 2-4 questions
  - Questions may be segmented into parts
  - Partial credit offered

MIDTERM EXAM INFO

- This is a **closed book/note** exam
- The exam will include an honor pledge, which you must sign!
  - **10 point deduction** if not signed
  - Violate pledge -> F
  - Make a scene after caught cheating -> FF, police escort, incident logged on records, etc...

MIDTERM EXAM INFO

- Content based on the following topics:
  - Basic C++ syntax (including those of the C++ 11/14 standard)
  - Functions
  - Arrays and Vectors
  - Pointers
  - Structs and Classes
  - Basic operator overloading concepts (you won't be required to write code for overloading a function)

- (See slides for week 2 through 5)

MIDTERM EXAM INFO

- There will be a full topic list on canvas containing the specific topics that are mentioned in the midterm
  - Keep an eye on that list as I complete the exam
- The multiple choice questions may take the form of conceptual questions
  - Know definitions of terms, “walk through” C++ syntax
- The free form questions require you to write C++ code
  - You will not be expected to write full C++ programs!

(BRIEF) REVIEW

REVIEW

- Recognize the variation of C++ syntax allowed for initialization of a variable

- Recall: int a = 0; is the same as int a{0};
  - int a = {0}; also allowed
  - See week 2, slide 9 for other variants

- Think of (and test out) all of the different ways you can declare and initialize a variable

REVIEW

- In functions, we discussed several terms such as: header, prototype, body, call
  - Make sure you can distinguish between the terms
- We discussed global, local, and static local variables
  - Make sure you know how they work
- We also talked about default parameters
  - Make sure you know the effect of default parameters on function calls

REVIEW

- We recalled the syntax for arrays and discussed the vector syntax

- Make sure you know how to:
  - declare and initialize vectors
  - Use the push_back and pop_back member functions
  - use a range based for loop

REVIEW

- In pointers, we discussed the relationship between arrays and pointers
  - Make sure you can distinguish between pointer notation and array notation
- We also talked about constant pointers, pointers to constants and constant pointers to constants
- We talked about smart pointers
  - Know the syntax for a smart pointer

REVIEW

- We talked about structs and how they can be used with arrays, pointers and other structs

- Make sure you can:
  - Define a nested struct
  - Access members of nested structs

- Make sure you recognize the notation for a struct which has a pointer member (or a struct pointer)

REVIEW

- For classes we discussed the use of constructors, destructors, accessors and mutators
  - Make sure you can distinguish between the terms

- Make sure you can define:
  - classes (with include guards)
  - functions outside of the class definition
    - Will require the scope resolution operator

SUCCESSFUL STUDENTS:

- Practice concepts to help them learn it
  - Better to write an actual C++ statements to learn about syntax than just reading about it on the slide
- Actually consult the text
  - Good for examples, or for covering things not mentioned on slides
- Review past assignments