Class #1 - 9/26/17

Tuesday, September 26, 2017 6:06 PM

Homework:

1. Get IDE set up
    a. This will likely take some time… so build your "Hello, World!" application ASAP
    b. IDE Options:
        i. Use SU virtual desktop (see Canvas for details…)
        ii. SU UNIX Server
        iii. Eclipse
        iv. Visual Studio (free from Dreamspark)

        * Other options available for MAC --> see Canvas
2. Homework 1 is due on 10/3
    a. A "first draft" of P1
    b. No pre/post conditions for HW1
3. P1 (1st big assignment) is due following Friday on 10/10
4. The Gaddis C++ PowerPoint is a good reference for basic C++ content
5. Recommendation is to start assignment ASAP as it will need to be done in C++
    a. The assumption is most people have a very basic working knowledge of C++
6. Software Documentation
    a. See guidelines on Canvas (https://seattleu.instructure.com/courses/1574151/pages/documentation-hints?module_item_id=16513845)
7. Sign-up for Canvas notifications via e-mail

General Notes:

1. Expectations
    a. Graduate level - learn at an accelerated pace
    b. Turn in your work on time. However, stuff happens so there is a late policy (20% reduction per day).
2. No textbooks are required. Those with red asterisks are recommended.
    a. Scott Meyer book is great reference, but not a beginner book (intermediate level)
3. Professor Oh will be out of town at a conference next week
    a. Her recommendation is to attend the other section this Thursday, 9/28, from 3:45 - 6:25 PM
    b. She will try to record the lecture for those who can't make it
4. Advising period runs 10/23-11/11
    a. Required for international students
    b. Advising will be group sessions due to number of students
    c. She will try to accommodate work schedules due to number of working students
5. Midterm Exam tentatively scheduled for 10/31
    a. She may push back depending on our performance on HW#1
6. Homework assignments
    a. Solutions will not be posted online due to nature of this course --> design driven
    b. Individual designs will vary
7. Canvas --> Modules
    a. Contains important course information. Check it out.
        i. Weekly content & (optional) readings
8. Course Chat
    a. Available on Canvas (similar to Slack)
    b. Professor Oh checks this daily

C++ Intro:

1. Scope resolution operator "::" tells the compiler which namespace the function belongs to
2. "using namespace std"
    a. One option - some very against this
    b. Other option is to use "std::" before "cout" or similar functions
3. Structs vs. classes
    a. Structs - Data
        i. Data is <u>public</u> by default
        ii. Can directly access their members
    b. Classes - Data & Functions
        i. Data is <u>private</u> by default
        ii. Can be made public/private/protected
        iii. Need get/set functions for non-public data
    c. Change "struct" to "class" and add an ";" at the end to convert from struct to class
        i. Made need other modifications and/or new functions to access this data
    d. Both variables and functions can be made public/private in classes
4. C++ allows users to allocate memory on stack or heap
    a. Stack is default
    b. Heap using the "new" keyword
    c. Give users maximum control over memory management (can be dangerous if poorly implemented)
5. Constructors/Destructors
    a. C++ will create a default constructor if you don't define one
6. Getter functions
    a. Use "const" so malicious/dumb user can't modify the private data
    b. e.g. `double getWidth() const { … }`
7. Abstraction - "Information hiding"
    a. What should really be available to the end-user?
    b. Goal: hide the implementation details for many reasons:
        i. Safety - malicious users
        ii. Incompetent users
        iii. Code maintenance/maintainability
            1) Easy to fix implementation details w/o touching higher-level applications
    c. Two roles:
        i. Class designer
        ii. Application programmer or tester
8. This pointer "this->"
    a. We will cover this later
    b. Useful for resolving naming conflicts
9. Class Separation
    a. Move class declaration to .h file (available interface for other users - think API)
        i. Also called the "specification file"
        ii. Avoid functions in header files
    b. Move class implementation to .c file
        i. Also called the "implementation file"
    c. Keep main.cpp file for driver function/app
10. Keep "std" functions like "cout" out of classes as much as possible
    a. Exception: temporary for debugging
11. Passing by value vs. reference
    a. Pass by value: Make a copy and pass the copy. Modifying the copy doesn't modify the original object.
    b. Pass by reference: passing the address of something so the function being called can directly access it

i. Use the "&" operator

ii. e.g. `void print(Rectange r&) { … }`

12. Function prototypes

a. Needed if function is defined after main() so main() can find the function

i. Can be imported via a .h file

b. Function prototype is simply the return type, the name, and the arguments it takes

13. Header file #include directives to prevent chaos during compilation

a. Potential problem: possible to include a header file multiple times

b. At the top of every header file you create:

i. e.g.

```
#ifndef RECTANGLE_H
#define RECTANGLE_H
```

c. At the bottom:

```
#endif  // RECTANGLE_H
```

d. **""** Use #include "Rectangle.h" for files you generate

i. This searches your local working directory/project for the file

ii. Compiler dependent behavior

e. **<>** Use #include <iostream.h> for system/utility files

14. Constructors

a. Think about realistic default states

i. Zero isn't always the best answer (e.g. a fraction)

b. Think about adding constraints (e.g. does a negative number for width make sense)?

c. A default constructor takes no arguments

d. Can have multiple constructors, some of which take values

e. Can also set default parameters in a constructor that takes arguments

Design Notes - Abstraction, Encapsulation, and Information Hiding:

1. Object state

a. Want to ensure objects are initialized to "good"/"valid" states

i. After constructor is called, all member functions should now be valid to be called

b. Want to ensure member functions can't invalidate the data

2. Coupling

a. How tightly interrelated are two or more objects?

b. Loose coupling is good (minimal dependency)

c. An object should do 1 thing well (e.g. describe and manipulate a rectangle)

i. Doing too many things tightens the coupling

3. Cohesion

a. Singularity of purpose

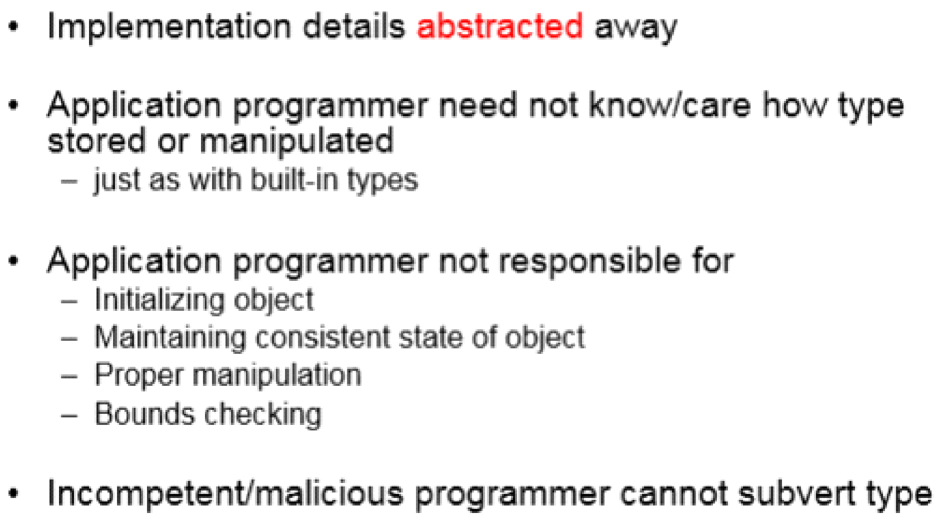
b. Things should fit together; they should belong

c. High cohesion is good

4. Abstract Data Types (ADT)

a. Separating interface from implementation

5. Abstraction



- Implementation details abstracted away
- Application programmer need not know/care how type stored or manipulated
  - just as with built-in types
- Application programmer not responsible for
  - Initializing object
  - Maintaining consistent state of object
  - Proper manipulation
  - Bounds checking
- Incompetent/malicious programmer cannot subvert type

6. Functional decomposition
   a. Avoid massive functions
   b. One function should do 1 thing
   c. If doing multiple things, decompose into separate private helper functions