Homework 5: Class Design

Instructor: Mehmet Emre

CS 32 Spring '22

Due: 4/20 12:30pm
Name & Perm #: Bharat Kathi (5938444)

Homework buddy (leave blank if you worked alone):

Reading: "Class Design", DS 2
In CMPSC 24, you likely already read parts of Chapter 2. I want to invite you to read it again, for two reasons: (1) to review parts you may have skipped or skimmed over the first time (2) to revisit things that you may have read many weeks or months ago, and that may not have fully made sense to you at the time.
I've included some questions here on a few of the important points from Chapter 2, but these are **not** the only things you need from Chapter 2. I don't have enough room on the page to give you homework over everything from Chapter 2 you need to learn. Read few this material a few times between now and the first exam.

1

According to Chapter 2 of DS:

1. (4 pts) In C++ if you write a class, but do not specify any constructors, what happens?

   Answer: The compiler will automatically create a default constructor. This constructor will just call the default contrustors for the member variables of your class.

2. (4 pts) When you implement a so-called "default constructor" in C++, what specific capability does this give the user of your class? (By specific, I mean what capability does the default constructor give that is distinguished from other kinds of constructors?)

   Answer: Declaring a default constructor allows users of your class to declare variables of your class without having to actually provide any arguments for a constructor.

3. (4 pts) Suppose you are writing a class called Student that has two private members, `std::string name` and `int perm`. Write the function prototype (function declaration) **only** for a default constructor for Student. For full credit, include the semicolon that terminates the prototype, but **do not** write the body of the constructor

   Answer: `Student();`

2

Before enrolling in CMPSC 32, you should have learned about stacks in CMPSC 24 (or an equivalent course). We will talk about a few things about an example stack implementation as a C++ `class`, then ask a few questions—these are designed to check your understanding of some important ideas about ADTs and C++ classes from Chapter 2 of DS.

Suppose we implement a class for a stack of integers, called `IntStack`. Our implementation uses a variable `int stack[MAX_SIZE]` and a variable `int size`. The constant `MAX_SIZE` may be declared as `const int MAX_SIZE=100`, for example. I will have methods `push`, `pop`, `isEmpty` and since our implementation has a max size, `isFull`.

If we declare a method with the function prototype `bool isEmpty() const`, I have the option of putting the function body either in a separate IntStack.cpp file, or directly inside the IntStack.h file like this:

```
// If this is in the header file, it should be declared
// ‘inline‘. Otherwise, we get multiple definitions!
bool isEmpty() const { return size == 0; }
```

1. (2 pts) According to C++ recommended practice, in which section of the class should we find the declarations of `int stack[MAX_SIZE];` and `int size;`?

   Circle one: public private

   Answer: private

2. (2 pts) When we put the method body (e.g. `{ return size == 0; }` for `isEmpty` inside the class declaration (i.e. in the `IntStack.h` file) that makes `isEmpty` a special type of member function (three word phrase with initials IMF). What does IMF stand for?

   Answer: IMF stands for Inline Member Function.

3. (3 pts) What is one thing that is different about IMF from the perspective of how they are treated by the compiler?

   Answer: The compiler will recompile the inline member function declaration and place a copy of it in your code. This saves some execution time sicne there is no seperate function call and return, but it also inefficient in space.

4. (4 pts) What is the significance of the keyword `const` that appears after `isEmpty`?

   Answer: This means that the function does not modify any class member variables. This helps protect the function from accidently modifying member variables.

5. (4 pts) Would `const` be appropriate for the `pop` operation?

   Circle one: yes no
   Briefly explain your answer.

   Answer: No, since the function needs to modify the integer stack by removing the top element.

6. (4 pts) Would `const` be appropriate for the `isFull` operation?

   Circle one: yes no
   Briefly explain your answer.

   Answer: Yes, since the function does not need to modify the values of any member variables.

3

DS Chapter 2 refers to something called a macro guard. This is a set of two pre-processor directives that appear at the start of a file, and one that appears at the end of a file. DS's macro guard for a file `throttle.h` uses a symbol `MAIN_SAVITCH_THROTTLE`, but it would be more traditional to use a symbol such as `THROTTLE_H` (the name of the file, with underscores after, and replacing the dot[1]). Another alternative that is common in large code bases is to guard a file `foo/bar/baz.h` with `FOO_BAR_BAZ_H` to encode the full file path (with underscore replacing the slash and the dot).

1. (4 pts) Write a macro guard for a file called `IntStack.h`, using the symbol `INTSTACK_H`. Put an ellipsis `...` in the place between the first two lines and the last line.

   Answer: .

```
#ifndef INTSTACK_H
#define INTSTACK_H
...
#endif
```

[1]We do not put an underscore before because the identifiers beginning with an underscore and a capital letter are reserved for the compiler and the standard library.

2. (5 pts) What is the purpose of a macro guard?

   Answer: Macro gaurds help avoid duplicate class definitions by putting class header definitions inside a compiler directive.