CS 201 Homework 8

Solomon Himelbloom

December 4, 2020

- Repository Link: `https://github.com/techsolomon/cs201`
- Git Commits: `https://github.com/techsolomon/cs201/commits`
- This homework took approximately 6 hours to complete.

1 Design

This ninth homework assignment was designed with refactoring in mind. This made it simple to edit previous programs, add additional features, and fix any bugs from prior versions. I also followed the directions of including the file name in the commit message when code was added or deleted with functions broken down into much smaller parts than before.

The hw8 folder contains the main program (main thermostat), two additional programs (proud polish and least finished), and a practice folder (tf and vacuum cleaner world). Header files were created to allow for linking multiple files and removing repetition for defining each function. For the last time in CS201, each .cpp file starts with the author's name, the computer science class code, the date file was created or last edited, and finally, a short description with an included Makefile.

2 Post Mortem

During this homework assignment, I learned more about classes and the fundamentals of extrapolating information from files. For style and design, I enjoyed using detailed comments and descriptive functions to better organize and reference programs on future assignments.

I have thoroughly valued all of the homework assignments and projects this semester and found that revisiting some of my earlier programs exemplified the progress that I have made while learning C++. Over winter break, I hope to dive deeper into the Standard Library with classes and graphical user interfaces (GUIs) to better prepare myself for CS202 in the Spring 2020 semester.

3 Answers to Questions

1. The main difference between a class and an object is that a class is a group of objects that share common properties whereas an object is an instance of a class with various behaviors and states.

2. We can call a member function of a C++ object using the object of the class with an included dot (.) operator. Requested C++ statement: x.foo();

3. A public member of a C++ class means that it is accessible everywhere, whereas a private member has more restrictions applied. Private class members can only be accessed to functions that are within the designated classes, seldom including references to other functions.

4. Similar to member functions and respective arguments, we can declare a member function using const. In other words, a const member function cannot change the current object and a const pointer does not have the permission for modification of what it is pointing to. This means that we can

ensure that any member functions will not modify the pre-existing object and any attempts to change previously stored data will result in a compile-time error.

5. When we declare a member function const, the "const" keyword goes after the parentheses surrounding the parameter list.

6. Constructors are defined as member functions that are called after a designated object is created. The main purpose of a constructor is to ensure that the object is fully useable within our program.

7. The name given for when a constructor is a member function is called the current object.

8. A default constructor is a constructor with no parameters.

9. One example of overloading as it pertains to constructors is giving two things the same name. You might be asking how this is possible… well, this works as long as each constructor has a unique list of arguments. This should not, however, be confused with namespace pollution.

10. Similar to function overloading, constructor overloading might be useful in situations where you want to use a different number of parameters after initializing an object. Important implications for constructor overloading include that arguments must be passed in for the compiler to know which constructor is needed during the specific call. Another fact is that two global functions can be called, but have the requirement of different numbers and types of parameters.

11. We can define a member outside the class function as long as the function prototype is defined inside the class definition. We use the scope resolution operator (::) between the return type and class name to the left side and the function name and an optional parameter list to the right side of this distinctive operator.

12. Class-wide concerns in a C++ class, as opposed to a particular object, are handled with header and source files. This works since we can move the definition of a member function outside of the traditional class definition.

13. When we declare a data member static within a class, this means that regardless of how many objects might be created in the future, there is only one copy of the static member.

14. Declaring a member function static means that it is no longer dependent on any object within the class. It is worthwhile to mention that after we declare a member function static, the static member function can only access the aforementioned static data member both inside and outside of the class.

15. Two ways that a static member function can be called in C++ include using the class name and scope resolution operator (::) and also through various objects within the class type. The first way is preferred over the second way.

4 Program 1

4.1 Sample Output/Screenshot

Listing 1: Sample Program Output

```
Hello, thermostat.
32 degrees Fahrenheit ( F )

-=-=-=-=-=-

What is the current temperature?

Agent #0
32

-=-=-=-=-=-
```

```
Environment #40
-40

-=-=-=-=-=-

Simulator #100
212
```

4.2 Git Commit Messages

Date	Message
2020-12-06	add: cycle member functions and print actions
2020-12-06	add: constructors and destructors to source files
2020-12-06	add: thermostat.cpp header and source files
2020-12-05	add: Temperature class to thermostat.cpp
2020-12-02	add: hw8 (initial files)

4.3 Source Code

4.4 thermostat.cpp

```cpp
// thermostat.cpp
// Solomon Himelbloom
// 2 December 2020
// Thermostat example for CS 201.

#include <iostream>
#include <stdio.h>
#include <string>
#include <map>
using std::cin;
using std::cout;
using std::endl;
using std::string;
using std::map;

#include "agent.hpp"
#include "environment.hpp"
#include "simulator.hpp"

// A simple divider to signify the end of a printed section.
void printSection() {
    cout << endl;
    cout << "-=-=-=-=-=-" << endl;
    cout << endl;
}

```

```
// Prints the current temperature reading (°F)
// given input from setDegrees() and printCurrent().
class Temperature {

public:
    Temperature() : _num(0) {

    }

    Temperature(int num) : _num(num) {

    }

    void setDegrees(int num) {

    }

    void printCurrent() const {
        cout << _num << " " << units() << endl;
    }
private:
    string units() const {
        return "degrees Fahrenheit (°F)";
    }
    int _num;
};

int main() {
    std::cout << "Hello, thermostat." << endl;
    Temperature f(32);
    f.printCurrent();

    printSection();

    // default agent constructor
    Agent a1;
    a1.setID(0);
    cout << "What is the current temperature?" << endl;
    cout << endl;
    cout << a1.toString();
    cout << std::endl;

    // 2nd agent constructor
    Agent a2(32);
    cout << a2.getID();
    cout << std::endl;

    printSection();

    // default environment constructor
    Environment e1;
    e1.setID(40);
    cout << e1.toString();
    cout << std::endl;

    // 2nd environment constructor
    Environment e2(-40);
    cout << e2.getID();
    cout << std::endl;

    printSection();

    // default simulator constructor
    Simulator s1;
    s1.setID(100);
    cout << s1.toString();
```

```
    cout << std::endl;

    // 2nd environment constructor
    Simulator s2(212);
    cout << s2.getID();
    cout << std::endl;

    return 0;
}
```

4.5 agent.cpp

```
// agent.cpp
// Solomon Himelbloom
// 4 December 2020
// Source file agent.cpp

#include <string>
using std::string;
using std::to_string;

#include "agent.hpp"

// constructors
Agent::Agent()
    : _id{0} {

    }

Agent::Agent(int id)
    : _id {id} {

    }

// destructor
Agent::~Agent() {

}

void Agent::setID(int id) {
    _id = id;
}

int Agent::getID() const {
    return _id;
}

string Agent::toString() const {
    return "Agent #" + to_string(_id);
}
```

4.6 agent.hpp

```
// agent.hpp
// Solomon Himelbloom
// 4 December 2020
// Header file agent.hpp

```

```
#ifndef AGENT_HPP
#define AGENT_HPP

#include <string>
using std::string;

class Agent {
public:
    // constructors
    Agent();

    Agent(int id);

    // destructor
    ~Agent();

    void setID(int id);

    int getID() const;

    string toString() const;

    string perceive() const;

    void think();

    string Act() const;

private:
    int _id;
};

#endif // #ifndef AGENT_HPP
```

4.7 environment.cpp

```
// environment.cpp
// Solomon Himelbloom
// 4 December 2020
// Source file environment.cpp

#include <string>
using std::string;
using std::to_string;

#include "environment.hpp"

// constructors
Environment::Environment()
    : _id{0} {

    }

Environment::Environment(int id)
    : _id {id} {

    }

// destructor
Environment::~Environment() {

}

void Environment::setID(int id) {
    _id = id;
```

```
}

int Environment::getID() const {
    return _id;
}

string Environment::toString() const {
    return "Environment #" + to_string(_id);
}

// bool Environment::iteration {
//     int heater();
//     heater = 0;

//     if (heater = 0) {
//         heater += 1;
//         return true;
//     } else {
//         heater -= 1;
//         return false;
//     }
// }
```

4.8 environment.hpp

```
// environment.hpp
// Solomon Himelbloom
// 4 December 2020
// Header file environment.hpp

#ifndef ENVIRONMENT_HPP
#define ENVIRONMENT_HPP

#include <string>
using std::string;

class Environment {
public:
    // constructors
    Environment();

    Environment(int id);

    // destructor
    ~Environment();

    void setID(int id);

    int getID() const;

    string toString() const;

    // Changes the temperature by 1 or -1 depending
    // on whether the header is on or off.
    bool iteration();

private:
    int _id;
};

#endif // #ifndef ENVIRONMENT_HPP
```

4.9 simulator.cpp

```cpp
// simulator.cpp
// Solomon Himelbloom
// 4 December 2020
// Source file simulator.cpp

#include <string>
using std::string;
using std::to_string;

#include "simulator.hpp"

// constructors
Simulator::Simulator()
    : _id{0} {

    }

Simulator::Simulator(int id)
    : _id {id} {

    }

// destructor
Simulator::~Simulator() {

}

void Simulator::setID(int id) {
    _id = id;
}

int Simulator::getID() const {
    return _id;
}

string Simulator::toString() const {
    return "Simulator #" + to_string(_id);
}
```

4.10 simulator.hpp

```cpp
// simulator.hpp
// Solomon Himelbloom
// 4 December 2020
// Header file simulator.hpp

#ifndef SIMULATOR_HPP
#define SIMULATOR_HPP

#include <string>
using std::string;

class Simulator {
public:
    // constructors
    Simulator();

    Simulator(int id);

    // destructor
    ~Simulator();
```

```
21
22      void setID(int id);
23
24      int getID() const;
25
26      string toString() const;
27
28      // Initializes the running sequence.
29      void run();
30
31      void askOwner();
32
33  private:
34      int _id;
35  };
36
37  #endif // #ifndef SIMULATOR_HPP
```

5 Program 2

5.1 Sample Output/Screenshot

Listing 2: Sample Program Output

```
What is your desired chess square? 9

SQUARE #1:
current square: 1
previous square total: 0

SQUARE #2:
current square: 2
previous square total: 1

SQUARE #3:
current square: 4
previous square total: 3

SQUARE #4:
current square: 8
previous square total: 7

SQUARE #5:
current square: 16
```

```
previous square total: 15

SQUARE #6:
current square: 32
previous square total: 31

SQUARE #7:
current square: 64
previous square total: 63

SQUARE #8:
current square: 128
previous square total: 127

SQUARE #9:
current square: 256
previous square total: 255

cs201: hw8 (proud-polish) questions...
At least 1,000 (grains of rice): SQUARE #9
At least 1,000,000 (grains of rice): SQUARE #19
At least 1,000,000,000 (grains of rice): SQUARE #29
Largest number (int): SQUARE #31
Largest number (float): SQUARE #1024
```

5.2 Git Commit Messages

Date	Message
2020-12-06	fix: running total error (proud-polish.cpp)
2020-12-05	refactor: current square equation (total rice)
2020-12-05	add: hw2 rice comments to proud-polish.cpp
2020-12-05	add: rice + testing to proud-polish folder
2020-12-02	add: hw8 (initial files)

5.3 Source Code

5.4 proud-polish.cpp

```
// proud-polish.cpp
// Solomon Himelbloom
// 2 December 2020
// Proud polished program example for CS 201.

#include <iostream>
#include <stdio.h>
#include <string>
#include <map>
using std::cin;
using std::cout;
using std::endl;
using std::string;
using std::map;

/*
hw2 (rice.cpp) --> proud-polish revision:

The rice program contains incorrect calculations.

You may notice that in your "previous square total",
the number will always be one less than the next power
of two (based on the mathematical sum of the series 2^i
from i=0 to n being equal to 2^(n+1) - 1).

Thus, we can see that there will actually be
1023 grains of rice on the 9th square, and similarly the
19th and 29th for one million and one billion, respectively.
*/

double currentSquare (int & square) {
    return pow(2, square - 1);
}

double previousSquare (int & square) {
    double total = 0;
    for (int i = 1; i <= square; i++) {
        total = currentSquare(i);
    }

    return total;
}

int main() {
    double i, requested_square = 0;
    double square_number = 0;
    double current_square = 0;
    double previous_square = 0;

    cout << "What is your desired chess square? ";
    cin >> requested_square;
    cout << " " << endl;

    for (i = 0; i < requested_square; i++) {
        previous_square += current_square;
        square_number += 1;
        current_square = pow(2, square_number - 1);

        cout << "SQUARE #" << square_number << ":" << endl;
```

```
60          cout << "current square: " << current_square << endl;
61          cout << "previous square total: " << previous_square << endl;
62          cout << " " << endl;
63      }
64
65      cout << "cs201: hw8 (proud-polish) questions..." << endl;
66      cout << "At least 1,000 (grains of rice): SQUARE #9" << endl;
67      cout << "At least 1,000,000 (grains of rice): SQUARE #19" << endl;
68      cout << "At least 1,000,000,000 (grains of rice): SQUARE #29" << endl;
69      cout << "Largest number (int): SQUARE #31" << endl;
70      cout << "Largest number (float): SQUARE #1024" << endl;
71  }
```

6 Program 3

6.1 Sample Output/Screenshot

Listing 3: Sample Program Output

```
FLTK: cs201 hw8
Truncate
Quit
```

6.2 Git Commit Messages

Date	Message
2020-12-06	add: truncate text and quit window button
2020-12-05	add: fltk template in least-finished.cpp
2020-12-05	add: fltk-trunc to least-finished folder
2020-12-02	add: hw8 (initial files)

6.3 Source Code

6.4 least-finished.cpp

```
1  // least-finished.cpp
2  // Solomon Himelbloom
3  // 2 December 2020
4  // Least finished program example for CS 201.
5
6  #include <FL/Fl.H>
7  #include <FL/Fl_Window.H>
```

```
#include <FL/Fl_Button.H>
#include <FL/Fl_Box.H>
#include <FL/Fl_Pixmap.H>

#include <iostream>
#include <stdio.h>
#include <string>
#include <map>
using std::cin;
using std::cout;
using std::endl;
using std::string;
using std::map;

void TruncateText() {
    cout << "FLTK: Truncate Text" << endl;
    exit(1);
}

void QuitWindow() {
    cout << "FLTK: Quit Window" << endl;
    exit(2);
}

int main(int argc, char **argv) {

    Fl_Window *window = new Fl_Window(340,180, "FLTK: cs201 hw8");

    // Create the FLTK application window.
    Fl_Box *box = new Fl_Box(20,40,300,100);
    box->box(FL_UP_BOX);
    box->labelfont(FL_BOLD+FL_ITALIC);
    box->labelsize(36);
    box->labeltype(FL_SHADOW_LABEL);

    // Create two instances of regular buttons.
    Fl_Button *but1 = new Fl_Button(100,60,140,25,"Truncate");
    Fl_Button *but2 = new Fl_Button(100,90,140,25,"Quit");

    window->end();
    window->show(argc, argv);
    return Fl::run();

}
```