Lab 3 – Classes, Multi-File programs, and Operator Overload

Nyla Spencer, Ember Roberts, Maggie Lyon

a. A description of the objectives/concepts explored in this assignment including why you think they are important to this course and a career in CS and/or Engineering.

The objectives/concepts explored in this assignment include classes, muti-file programs and operator overloading. These concepts are building blocks in engineering and computer science. Classes allow for the creation of user-defined data types, data encapsulation, and the ability to model real-world scenarios. Multi-file programs are an important concept because understanding how to organize code in multiple files creates a collaborative environment and is a step to further development of code. Finally, operator overloading is useful in making custom data types cooperate within structures.

In task 1, we created a file called Measure.h in which we declared the members of the Measure class, including the private variables; littles, lots, and heaps, as well as public accessors and mutators for these variables. This modularity is an important part of class programming, as keeping data private allows for security and prevents unintentional access, while public accessors/mutators allow this data to be modified in a controlled manner. Also in this file are our class constructors. A constructor is a special member function automatically called when the class is created to initialize the members. We declared 3 constructors that each take a different number of parameters, including the default constructor (no parameters), a constructor that only takes the number of littles, and a constructor that takes values for all 3 private member variables.

In task two, a corresponding .cpp file was created. In this file, the member functions and constructors are defined. No changes were made to the .h file. Having the declaration and the definition of hte functions in two different files allows us to keep our definitions private, as the compiler will create an object file of the .cpp file that can be passed along to other programmers so they cannot access our source code. In task three, the class was extended with operator overloading. Operator overloading is a concept that allows us to redefine the behavior of an operator, so it can be customized to the programmer's needs. In this case, it allowed us to perform operations on the measurement with constraints to ensure the number of littles stays below 7, and lots below 23.

Finally, in task four, we tested the class. Prompts were added for user interaction, result display, and a potential loop to continue if the user wishes. The following screenshot shows some of these operations being performed using the test file.

```
3 - Classes, Multi-File programs and Operator Overloading/"test
(base) maggielyon@Maggies-MacBook-Air Data Structures % cd "/Users/maggielyon/Desktop/Spring 2023/Da
tructures/Lab 3 - Classes, Multi-File programs and Operator Overloading/" && g++ test.cpp -o test &&
sers/maggielyon/Desktop/Spring 2023/Data Structures/Lab 3 - Classes, Multi-File programs and Operatc
erloading/"test
Enter values for obj1 (littles lots heaps): 2 4 6
Enter the number for the operation to perform (1. +, 2. -, 3.  /, 4.  *, 5.  ==): 4
Enter values for obj2 (littles lots heaps): 4 2 2
Result: Littles: 1, Lots: 9, Heaps: 12
Do you wish to continue? (y/n)y
Enter values for obj1 (littles lots heaps): 5 10 2
Enter the number for the operation to perform (1. +, 2. -, 3.  /, 4.  *, 5.  ==): 1
Enter values for obj2 (littles lots heaps): 8 16 1
Result: Littles: 6, Lots: 4, Heaps: 4
Do you wish to continue? (y/n)y
Enter values for obj1 (littles lots heaps): 6 20 40
Enter the number for the operation to perform (1. +, 2. -, 3.  /, 4.  *, 5.  ==): 2
Enter values for obj2 (littles lots heaps): 3 10 20
Result: Littles: 3, Lots: 10, Heaps: 20
Do you wish to continue? (y/n)y
Enter values for obj1 (littles lots heaps): 5 10 15
Enter the number for the operation to perform (1. +, 2. -, 3.  /, 4.  *, 5.  ==): 3
Enter values for obj2 (littles lots heaps): 5 5 5
Result: Littles: 1, Lots: 2, Heaps: 3
Do you wish to continue? (y/n)y
Enter values for obj1 (littles lots heaps): 2 2 2
Enter the number for the operation to perform (1. +, 2. -, 3.  /, 4.  *, 5.  ==): 5
Enter values for obj2 (littles lots heaps): 2 2 2
Equal: True
Do you wish to continue? (y/n)
```

Contributions:

Nyla – lab report, methods research

Ember – Programming for overloaded constructors

Maggie – Programming for .h and test file

https://github.com/margaretlyon21/Lab3/tree/main