1 Purpose

- To gain experience using 1-way selection.
- To gain experience using the C++ `while` construct.
- To gain experience implementing a class given a class definition.
- To gain experience using a `Makefile`.

2 One Way Selection

Copy the file `salary.cc` from the `~cs203/Labs/Lab6` to your own `Lab6` directory.

Using the `salary.cc` file, complete problem 7B on page 262 of the Mercer text.

Be sure to hand-check your results so that you know you have done the implementation correctly.

Hand in the program code, and an execution of the program.

3 Repetition

Copy the file `rep.cc` from the `~cs203/Labs/Lab6` to your own `Lab6` directory. The program is supposed to read a sequence of positive values and then display their average.

You are to fill in the main function so that it reads (using the function `getValue()`) a sequence of values and sums them up — the end of the sequence will be marked by the value -1. After all the values have been read, you display the average of the values.

Important: If no values are read (i. e., if the first value read is -1) then a message indicating no data was read should be printed.

Hand in the complete program code, and several executions of the program showing that all the features work.

4 Implementing a Class

Scenario:

> A gate attendant at a concert makes use of a “click-counter” which, when clicked, adds one to the current value of the counter. In this way, when everyone has entered the concert, the number of people admitted can be determined.
>
> The click-counter has two buttons: the counter button, which when pressed increments the counter, and the reset button which sets the counter value to zero. The click-counter also has a window containing a sequence (probably 4) rotating wheels with the digits from 0 through 9. These rotate appropriately to represent the number counted so far. The important thing about the window is that it sets a maximum on the number of things which can be counted — it will automatically roll back to 0 when the maximum number is incremented.

We want to develop a similar kind of counting device which can be used in a program to, for example, count the number of times an object is accessed or the number of times a loop repeats.

Here is a class definition for a software click-counter.

```
#ifndef CLICKER_H
#define CLICKER_H

class Clicker {
public:
    Clicker();
    Clicker(int inMaxCount);
    void click();
    void reset();
    void setMaxCount(int inMaxCount);
    void display() const;
    int getCount() const;
    int getMaxCount() const;
private:
    int mCount;
    int mMaxCount;
};
```

```
#endif
```

Carry out the following activities.

- Copy the files `Clicker.h`, `testClicker.cc` and `Makefile` from `~cs203/Labs/Lab6` to your `Lab6` directory.
- Edit `Clicker.h` and add appropriate pre and postconditions for each method.
- Create a new file `Clicker.cc` and write the implementations for each of the methods (and constructors) of the class definition.

  For the default constructor you can choose a value which will always be used for the maximum (unless it is fixed in some other way).

  For the `display()` method, print the current value of the counter and the maximum count.
- Notice that the **const** after the three methods is a promise that the function doesn't change the state of the object.

- Notice in the file `Clicker.h` that the first two and last lines are compiler directives (since they begin with `#`) and they make sure that if you happen to include `Clicker.h` twice by mistake, the second inclusion will have no effect. Without these lines you would end up with lots of "doubly defined" object and abstraction names.

  The notion `#ifndef CLICKER_H` is read "if `CLICKER_H` is not defined then continue, otherwise skip to the `#endif`. Notice that if `CLICKER_H` is defined, then the lines after the `#ifndef` will *not* be seen by the compiler. The `#define CLICKER_H` line obviously defines the name `CLICKER_H` and then goes on. That will only be done the first time the file is included!

- Examine `testClicker.cc` to see how it works. It has no other purpose than to act as a testing ground for the `Clicker` class.

  Move to your `Lab6` directory. Now, compile your program by typing **make** after the Unix prompt. The command **make** performs the instructions in the `Makefile` you copied. We will learn how to create our own `Makefile` files later.

> After compiling, run the executable to test the class definition. Modify the program to test the following feature of the class. You should show that the clicker object actually rolls back to 0 properly when the maximum value is reached. (How do you suppose you should initialize the maximum count to test this? 10000? 100? Something else?)

Turn in all the code for this part: `Clicker.h`, `Clicker.cc`, and your final version of `testClicker.cc` along with the execution of `testClicker.cc`.

When you are finished, be sure to clean up your directory. Type **make clean** at the Unix prompt. This will remove all of your `.o` files and your `.exe` file. You should clean up your files after each lab.