COSC 220 - Computer Science II
Lab 9

Dr. Joe Anderson

25 April 2018

1 Objectives

1. Develop familiarity with overloading operators.
2. Practice writing thorough test cases to check complicated code.

2 Tasks

2.1 Pre Lab

1. Before lab, make sure you have a class called `PayRoll` defined in a file called `PayRoll.h`, possibly along with function definitions in `PayRoll.cpp`.
2. Be sure that the `PayRoll` and `PayRollList` classes follow the specifications from Lab 5: the correct copy constructor and assignment operator.

2.2 In Lab

1. Add an overloaded addition operator: `operator+` to the `PayRoll` class to add together *both* the hourly rate and the number of hours worked. This should return a *new* `PayRoll` object, and not modify either of the operands.
2. Add an overloaded comparison operator: `operator<` to the `PayRoll` class to determine whether one payroll has a smaller hourly pay rate than other.
3. Add an overloaded comparison operator: `operator<` to the `PayRollList` class to determine whether one `PayRollList` has a larger *total weekly pay* than the other. This function will need to iterate each item in the list and sum the weekly pay of each employee, then compare the sums for each of the two lists.
4. Add an overloaded stream operator: `operator<<` to the `PayRoll` and `PayRollList` classes.
   (a) These will have prototypes: `std::ostream& operator<<(std::ostream&, const PayRoll&);` and `std::ostream& operator<<(std::ostream&, const PayRollList&);`, respectively.
   (b) The functionality will be relatively straightforward, and can be up to you as to how to format the output.
   (c) If you want to be able to directly access private elements, you may designate these operators as `friend` functions to each class.

5. Finally, overload another operator of your choice *that we have not overloaded yet*! Be sure to document which one you implemented, what it does, and why. Include tests (below) to demonstrate its functionality.

6. **Important: worth many points!** Test your code on various situations, including edge cases (e.g. printing an empty list, self-assignment, etc.). **Thoroughly test and demonstrate correctness of each of your overloaded operators!** You will need to think to yourself: what are all the possible ways that this will be called, and will it always be correct? What needs to be printed/displayed to verify that they function correctly?

7. Include a `Makefile` to compile your code into a binary which runs your test cases.

3 Submission

Be sure your code conforms to the above specifications. Upload your project files to the course canvas system in a single zipped folder: To zip on Linux:

1. To zip a single folder into a single archive called "myZip.zip":

   ```
   zip -r myZip.zip folderName
   ```

2. To zip multiple files into a zip file called "myZip.zip":

   ```
   zip myZip.zip file1.cpp file2.h file3 file4.cpp
   ```

Turn in (stapled) printouts of your source code, properly commented and formatted with your name, course number, and complete description of the code.

Also turn in printouts reflecting several different runs of your program (you can copy/past from the terminal output window). Be sure to test different situations, show how the program handles erroneous input and different edge cases.