COSC 220 - Computer Science II
Lab 4

Dr. Joe Anderson

Due: 24 September 2019

1 Objectives

1. Practice using C++ classes
2. Develop familiarity with linked lists, and dynamic memory
3. Develop familiarity with advanced class operations: copy constructor, overloaded operators

2 Description

You should read the entire set of instructions before beginning the lab.

You will expand upon the `PayRoll` and `PayRollList` classes you developed in the last lab. You will add the ability for a programmer to perform a "deep copy" of a linked list, via a *copy constructor*. We will add the ability of a programmer to access the list as if it were array: with the `[]` (index) operator, and also query the length of the list. Finally, you will use the new index operator to also allow a user to interact with the `PayRollList` in a more structured way: by specifying a position in the list to remove, assign, or insert.

3 Tasks

1. Design a PayRoll class that has data members for an employees name, hourly pay rate, number of hours worked, and total pay for the week
   (a) **You may modify the PayRoll class you designed last lab!**
   (b) The PayRoll class should implement the following prototype:

```
class PayRoll {
  private:
    string name;
    double rate;
    double hours;

  public:
    PayRoll(); // ctor
    PayRoll(string, double, double); // non-default ctor
    double getTotal();
    double getRate();
    double getHours();
    void setRate(double);
    void setName(string);
```

```
        void setHours(double);
    };
```

2. Implement a linked list whose nodes store PayRoll objects.

   (a) The PayRoll objects should be sorted by pay rate within the list.

   (b) The list should be called "PayRollList" and implement the following class definition:

```
class PayRollList{
  private:
    struct ListNode {
      PayRoll p;
      ListNode* next;
    };

    // points to the first node in the list, or nullptr
    // if the list is currently empty.
    ListNode* head;

  public:
    PayRollList(); // ctor
    ~PayRollList(); // destructor, do this carefully!

    /*
     * copy constructor, takes another list by reference!
     * Should copy each PayRoll into the list being instantiated
     */
    PayRollList(PayRollList&);

    // Returns the number of items in the list
    int length();

    // An overloaded index operator
    // Takes an integer index and (if it exists)
    // returns a *reference to* the PayRoll inside it (why?).
    // To use this, we can say myList[5] to get a reference
    // to the sixth element of the list (if present).
    // You may return a null pointer if the object is not present
    PayRoll* operator[](int);

    // Remove the list item at the specified position
    // (may take advantage of your operator[])
    void remove(int);

    // Copy a PayRoll into a specific position.
    // Be sure to validate the position!
    // (may take advantage of your operator[])
    void assign(int, PayRoll);

    // Inserts the PayRoll into the list so that it is now at the
    // specified position (may take advantage of your operator[]).
    void insert(int, PayRoll);
```

```
            // Include these from last lab
            void insert(string, double, double);
            void insert(PayRoll);
            void printPayChecks();
        };
```

   (c) Note that this class depends on the `PayRoll` class, so will need to `#include` the appropriate header file.

3. Construct your `main` function to test each of the methods above with several hard-coded examples.
   (a) Be sure to test them all with clear output!
4. Use your main program to read a list of employee names, rates, and hours from a file. You may design and use your own convention for the file format, but be sure to include your sample data with your submission.
5. The main function should then print the paychecks for each employee by calling the `printPayChecks` method on the list object.
6. The header files for your classes should use proper include guards.
7. Check your program for memory leaks with `valgrind` and make sure your constructors and destructors appropriately handle your dynamic memory.
8. Include a `README` file to document your program, provide instructions to compile and run it, explain how it works, and provide a reference for the data format of your input file.
9. Your code should be organized between the files: `main.cpp`, `payroll.h`, `payroll.cpp`, `payrolllist.h`, and `payrolllist.cpp`. Also include your `README` and a sample data file.

4 Submission

Upload your project files to the course canvas system in a single `.zip` file.

Turn in (stapled) printouts of your project files, 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.

5 Bonus

(10 pts) Use the guides linked from the course webpage to include a `Makefile` which automatically compiles your code by first compiling into three object files (`main.o`, `payroll.o`, and `payrolllist.o`) and then linking into a single executable.