UEE 1303(1067/1069): Object-Oriented Programming

Programming HW #3

Due: 2015/6/14 23:59

1. (20%) **Please use C++ file stream `<fstream>` to implement the file read and write.**

Of the many techniques for compressing the contents of a file, one of the simplest and fastest is known as *run-length encoding*. This technique compress a file by replacing sequences of identical bytes by a pair of bytes: a repetition count followed by a byte to be repeated. For example, suppose that the file to be compressed begins with the following sequence of bytes (shown in hexadecimal)

46 6F 6F 20 62 61 72 21 21 21 20 20 20 20 20

The compressed file contain the following bytes:

01 46 02 6F 01 20 01 62 01 61 01 72 03 21 05 20

Run-length encoding works well if the original file contains many long sequences of identical bytes. In the worst case (a file with no repeated bytes), run-length encoding can actually double the length of the file.

a) Write a program named `compress_file` that uses run-length encoding to compress a file. To run `compress_file`, we'd use a command of the

`compress_file` *original-file*

`compress_file` will write the compressed version of *original-file* to *original-file*`.rle`.

For example, the command

```
compress_file foo.txt
```

will cause `compress_file` to write a compressed version of `foo.txt` to a file named `foo.txt.rle`.

b) Write a program named `uncompress_file` that reverses the compression performed by the `compress_file` program. The `uncompress_file` command will have the form

`uncompress_file` *compressed-file*

compressed-file should have the extension `.rle`. For example, the command

```
uncompress_file foo.txt.rle
```

will cause `uncompress_file` to open the file `foo.txt.rle` and write an uncompressed version of its contents to `foo.txt`. Program `uncompress_file` should display an error message if its command-line

argument doesn't write the `.rle` extension.

2. (30%) Write a program to process "hw3-2.txt" and output to "hw3-2-out.txt".Type the following command to execute the program:

```
> ./hw3-2 hw3-2.txt hw3-2-out.txt
```

The content of the "`hw3-2-out.txt`" file is shown as follows.

```
The number of words read is 78
The longest word has a length of 12
The longest word is elaborations
```

TAs will use different input files to validate your programs.

3. (50%) (Evaluation System for Student) Please develop an evaluation system for students' performance. There are two different ways to evaluate the student's performance: *tests* and *sport*. You need to write a base class `score`, and two derived classes named `test` and `sport` which are inherited from `score`. Moreover, a derived class called `evaluation` is used to conclude the performance of the student and multiply inherited from `test` and `sport`. You may also need a class called `student` which contains three members, `id`, `name`, and `final_score` (`evaluation` class), and a class called `school` to store all results.

   Put the class definition and implementation of `score`, `test` and `sport` classes in **`Score.h`** and **`Score.cpp`**. Put the class definition and implementation of `student` and `school` classes in **`School.h`** and **`School.cpp`**.

   The command-line usage of the evaluation system is

```
>./hw3-3 performace.txt result.txt
```

   In the input file "*performance.txt*", the first line shows the total number of students in the school. Each row indicates a record for one student. As shown in the sample file, the first and second columns denote the student's ID and name, respectively. The later five numbers are the final scores for different subjects and the last five 1/0s indicate win/loss on different sports games. If a student is a winner in one sport game, he/she can obtain extra five points on his/her evaluation report. For example, Tom's average score on five subjects is 70 and he won three games to obtain extra 15 points. Therefore, Tom's final score is 85. The full score is 100. TAs will use different input files to validate your programs.

```
5
991001 Tom 50 60 80 90 70 1 0 0 1 1
991002 Jean 100 90 80 70 60 0 0 0 1 1
991003 Kevin 60 90 100 80 90 0 1 1 0 0
991004 John 100 80 90 70 80 0 0 0 0 1
991005 Marry 50 60 70 90 60 1 1 0 0 0
```

The context in "*result.txt*" is

```
991001 Tom 85
991002 Jean 90
991003 Kevin 94
991004 John 89
991005 Marry 76
Average: 86.8
```

The main function of the program is shown as follows,

```
int main(int argc, char *argv[])
{
    school nctu(argv[1]);
    nctu.report(argv[2]);
    return 0;
}
```