Part I: Practice and Theory

The following problems are for practise only and will **not be collected**.

Review problems: R9.1-R9.17. **Practice Problems**: P9.2 , P9.7, P9.13, 9.14, 9.16.

Part II: Programming. The following problems will be **collected** and two of them graded. Each graded problem will be worth 50 points. Read instructions carefully!

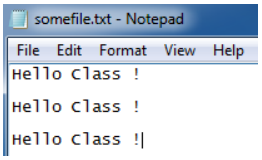
(1) **Problem P9.1**

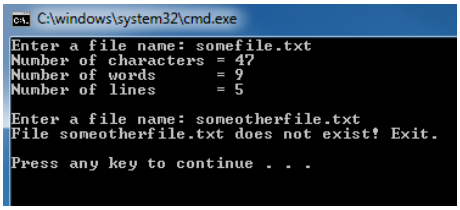
- Submit the solution as `hmw_3_1.cpp`.
- Empty lines are 'lines'. See the sample output below.
- To count the number of characters use the `tellg()` that returns the position number of `get` pointer. This can be done as follows:

```
fs.seekg(0, ios::end); long char_count = fs.tellg();
```

Note: Special charecters, such as the newline character, occupy different amount of bytes on different OS. Thus, Mac users may get different results from the one presented in the samples. Do not worry about it.

- Text between two spaces is a 'word'. To count the words, turn a string into `stringstream` and use the operator >> to count words. An empty word is not a word!
- Sample input-output:





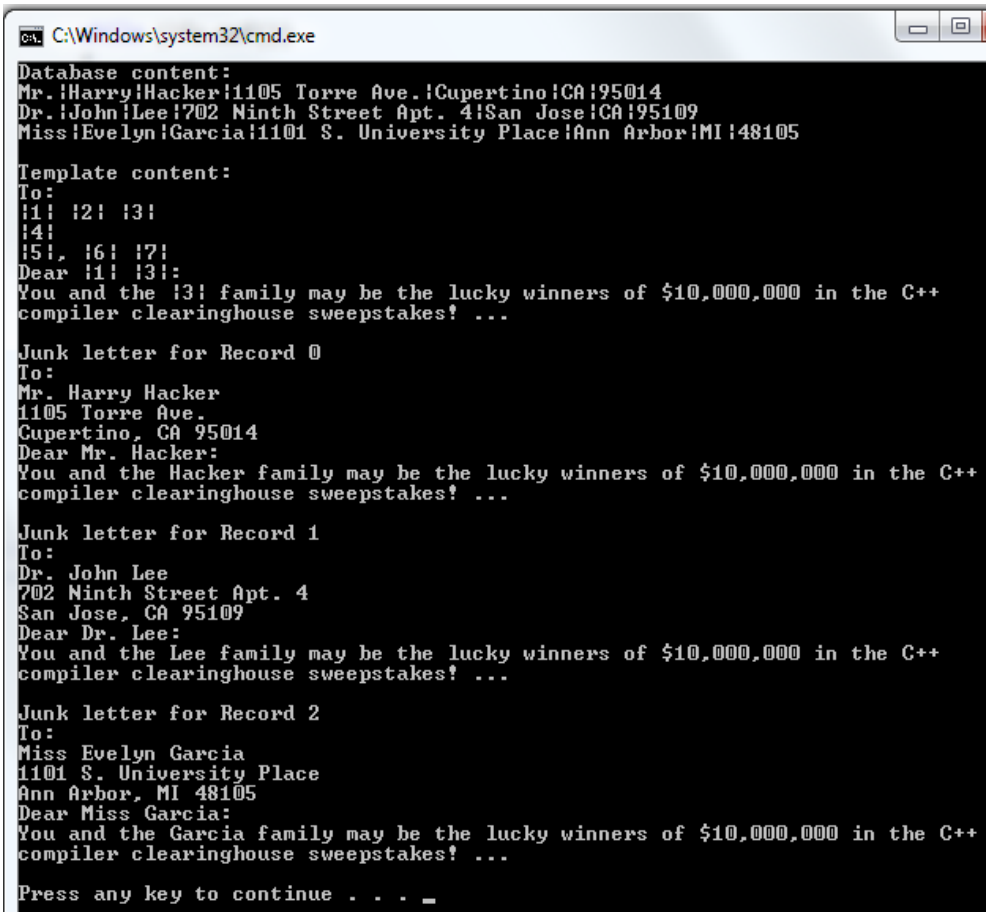
(2) • **Problem P9.6**

- Submit the solution as `hmw_3_2.cpp`.
- Assume that the database file is called `database.txt` and the template file is called `template.txt`. For simplicity, assume that both files exist.
- If `database.txt` is empty and contains no records, then display the message `"The database database.txt is empty. Exit!"` and exit the program.
- If the database is not empty, display the content of `database.txt`, then display the content of `template.txt`. Then create a junk letter for each record and display it on the screen.

• A useful hint. To parse the data fields for each record in the database, read the record (line) into a string. Then create a `stringstream`, say `ss`, associated with the record. Then, in a loop, use `getline(ss, field_val,'|')` to read off the value of each field from the stream to the `field_val`. Here `'|'` indicates a condition when to stop instead of default newline character. This approach makes parsing easy.

• Another useful hint: To replace $|1|, |2|, \ldots, |7|$ in the template file, read the template text line by line. Then replace in each line the above characters using `string::find()` to locate the position of $|1|, |2|, \ldots, |7|$ and `string::replace()` to replace the content.

•Sample input-output



(3) **Problem P9.19**

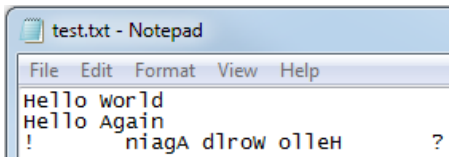
• Submit the solution as `hmw_3_3.cpp`.

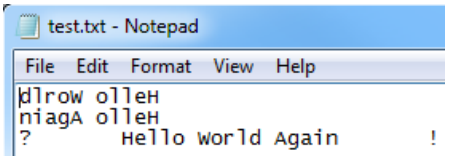
• Write a program that asks the user for the name of the file whose lines are reversed. Then display the content of the file. Then reverse each line in the file (in place by overwriting the

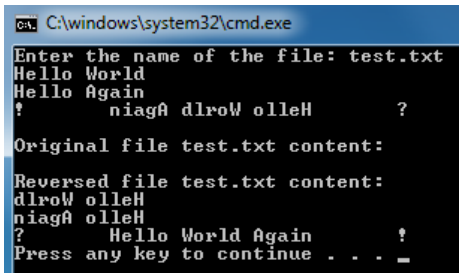
old content). Then display the new content of the reversed file. If the file does not exist, then display the error message.

- Note, when reaching the end-of-file, the stream will not write into file. Thus, to re-write the line that was read last, which sets EOF to `true`, you need to clear the flag by calling fstream::clear().

- Sample input-output:

Before 

After 

Output 

If the file does not exist output: