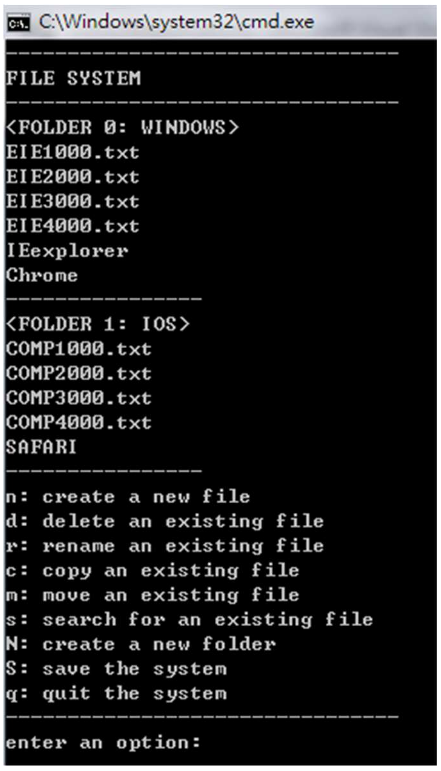
C++ Application Development Assignment – 1

File System

Practical Work

By using Microsoft Visual Studio 2015, develop an object-oriented console application of file system, which can display the names of folders and files, and also perform operations on file names, such as creating a new file name or deleting an existing file name (since this is only a conceptual application, we are merely dealing with file names instead of real files). You are given three text files, which are named “folderName”, “IOS”, and “WINDOWS”. “folderName” stores folder names in this file system; “IOS” stores the names of the files in the folder named IOS, and “WINDOWS” stores the names of the files in the folder named WINDOWS. You can prepare and check the contents of these three files using word processing tools such as Notepad, MS Word or WordPad. The developed file system should be able to read the contents from these three files, and display all the folder names as well as all the file names in each folder. An example is shown below, where “WINDOWS” is the name of the first folder, and “IOS” is the name of the second folder. As shown below, there are 6 file names in the first folder and 5 file names in the second folder. You do not need to follow this style; for example, you can also display the file names in each folder differently, but you must have a way to display all the file names and their corresponding folder names.



C:\Windows\system32\cmd.exe

```
--------------------------------
FILE SYSTEM
--------------------------------
<FOLDER 0: WINDOWS>
EIE1000.txt
EIE2000.txt
EIE3000.txt
EIE4000.txt
IEexplorer
Chrome
----------------
<FOLDER 1: IOS>
COMP1000.txt
COMP2000.txt
COMP3000.txt
COMP4000.txt
SAFARI
----------------
n: create a new file
d: delete an existing file
r: rename an existing file
c: copy an existing file
m: move an existing file
s: search for an existing file
N: create a new folder
S: save the system
q: quit the system
--------------------------------
enter an option:
```

While you are responsible for the final design of the console application of file system, your application is expected to contain a class called `fileSystem`, which stores 1) the number of folders, 2) the names of the folders, 3) the number of files in a folder, and 4) the names of files in a folder. (Hint: you can use multidimensional arrays to store the file names, for example, you can use “`char filename[100][100][100]`” to store the file names in different folders, and `filename[i][j][k]` is the *k*-th character of the *j*-th file’s name in the *i*-th folder. In this way, `filename[i][j]` stores the *j*-th file’s name in the *i*-th folder. Alternatively, you can assume there will be only 2 folders in this application, so you can also use “`char folder1[100][100]`” to store the file names in folder 1 and “`char folder2[100][100]`” to store the file names in folder 2. You can assume that there will be only 2 folders, and in each folder there will be no more than 10 file names, and the length of each file name will be shorter than 50, so that the array size 100 is large enough. Also you can assume that any name contains no space, so that you can simply use `cin` to read `char` strings instead of using `cin.getline()`.) Within the class, apart from the public member functions for displaying the names of files and folders, it should also be able to perform extra tasks on the file names, and you are required to develop public member function(s) to realise one of the following five tasks. Your task should be determined by the remainder obtained from dividing your team number by 5. For example, if your team number is SEVEN, 7%5 = 2, you should do Task 2) of this assignment. Notice that all the tasks below do NOT require you to save the changes into the 3 text files. For simplicity, when doing the tasks below, you can assume that the folder names or folder indexes are always valid, i.e. you don't need to check whether the folder names exist or whether the folder indexes are valid.

Task:

0) The member function adds a new file name to a specific folder. It should ask for the input of a source folder (e.g. the folder index *i* such as 0, 1, 2…, or the folder name; it depends on your design) and the input of the file name. It must check whether the file name exists in the source folder or not, and add the file name only when that file name does not exist. Then when your application displays the file names in that source folder, the newly added file name should also be displayed.
1) The member function deletes an existing file name from a specific folder. It should ask for the input of a source folder (e.g. the folder index *i* such as 0, 1, 2…, or the folder name; it depends on your design) and the input of the file name. It must check whether the file name exists in the source folder or not, and delete the file name only when that file name exists. Then when your application displays the file names in the source folder, the deleted file name should NOT be displayed.
2) The member function renames an existing file name in a specific folder. It should ask for the input of a source folder (e.g. the folder index *i* such as 0, 1, 2…, or the folder name; it depends on your design) and the input of the file name. It must check whether the file name exists in the source folder or not, and rename the file name only when that file name exists. Then when your application displays the file names in the source folder, the modified file name should be displayed instead of the original file name.
3) The member function copies an existing file from a source folder to a destination folder. It should ask for the input of the source folder and the destination folder (e.g. the folder index *i* such as 0, 1, 2…, or the folder name; it depends on your design) and the input of the source file name and the destination file name. It must check whether the source file name exists in the source folder and whether the destination file name exists in the destination folder. It performs the "copy" operation only when the source file name exists in the source folder and the destination file name does NOT exist in the destination folder. Then when your application displays the file names in the source and destination folders, that source file name should still be in the source folder, and the destination file name should be in the destination folder. Notice that for higher flexibility, the source file name can be the same as or different from the destination file name.
4) The member function moves an existing file name from the source folder to the destination folder. It should ask for the input of the source folder and the destination folder (e.g. the folder index *i* such as 0, 1, 2…, or the folder name; it depends on your design) and the input of the source file name. It must check whether the source file name exists in the source folder and the destination folder. It performs the "move" operation only when the source file exists in the source folder but does NOT exist in the destination folder. Then when your application displays the file names, that source file name should NOT be in the source folder, but in the destination folder.

- It is required that the class and the implementation of its member functions should be built as a separate static library and linked into the console application.

- Your console application is expected to provide a text-mode user interface so that users can repeatedly display the folder names, file names, and do the task on file names; until the user chooses to end the application.

- Should you want to get a credit, you should implement a new public member function, which performs file name search. This function should ask for the input of a character string (i.e. a query string), and find out all the file names that match with the query string and their corresponding folder indexes or names. For simplicity, you can assume the query string contains no space. For example, if you have a file name "`abc100`" in some folder, and you input a query string "`abc`", then "`abc100`" should be found out and shown, and its folder index or folder name should also be shown. However, if you input a query string "`abc2`" or "`bc2`" or "`abc1001`", "`abc100`" should NOT be found out, and a message should be shown, indicating that no matched file is found.

- Should you want to get a distinction, you should implement a new public member function to save all your changes made to the file names to the 3 text files of "folderName", "IOS", and "WINDOWS", so that next time when you open the console application again, you have the updated file system.

Report

Your report should include:

Abstract: Summarise the objectives and achievement of your assignment in less than 100 words.

1. **Introduction**: Describe the objectives and requirements of the assignment in detail, and give a brief account of the methodology.

2. **Methodology**: It contains
   - How your team divides the work among the team members (very important, to be used as the basis for assessment)
   - The schedule and stages of developing the project
   - The structure of the developed program, including
     - The specifications of the classes defined, and the public/private member functions/variables inside - explain as far as possible why your group makes such choices of members
     - The flow of program. (It is good to include a flow chart to help illustration.)
   - What problems your group encounters, and how your group solves the problems
   - Testing of your program, which shows
     - How you validate your program, i.e. confirm that the solution is correct.

3. **Results**
   - Include the results of executing your program captured from the screen.

4. **Conclusion and further development**
   - Summarize the experience gained in the assignment
   - Indicate how your program can be extended and improved if more time is allowed.

The report should be in PDF format with your class number, team number, student names, student IDs and task number at the front page. It is NOT required to include the complete source code in the report. Instead, you should zip the folder(s) containing all your project files into a zipped file, which also stores the report. (See the General Description below.)

General Description

1. Each team should comprise TWO students. Students must obtain prior approval from the subject lecturer if they want to form a team with fewer or more team members.

2. Unless you get prior approval from your subject lecturer/tutor, you must observe the following:
   - Do NOT use any technique or C++ constructs not taught in the subject
   - Unless mentioned on this instruction sheet, any library function not mentioned in the subject must NOT be used.

3. Each team should upload the zipped file to the Blackboard (under `Groups`, select `File Exchange` after clicking your team number) on or before **6 Dec. 2017**.

4. It is your duty to assure your submitted application can be built using Visual Studio 2015, and run in `Command Prompt`. Assessment will only be made based on your submitted zipped file. To lower the chance of incomplete submission, for submission, you are advised to zip one whole project folder that contains all folders and files in the project**s** of this assignment.

5. The documentation for your assignment is important. The ability of writing good comments in the program will also be an important factor to the final assessment of your assignment.

6. It is compulsory to use a word processing tool to write your report. The font size must not be bigger than 12 or smaller than 10. Use 1.5 lines spacing on both sides of a page. Including all figures and tables, if any, the length of the report should not be shorter than 7 pages.