NATIONAL UNIVERSITY OF SINGAPORE
SCHOOL OF COMPUTING

Practical Examination 2 (PE2) for Semester 1, AY2014/5
CS1010 Programming Methodology

1 November 2014 Time Allowed: 2 hours

INSTRUCTION TO CANDIDATES

1. You are only allowed to read this cover page and the last page. Do **not** read the questions until you are told to do so.
2. This paper consists of **2** exercises on 8 pages. Each exercise constitutes 50%.
3. This is an open-book exam. You may bring in any printed material, but **not** electronic devices, including but not limited to laptop, thumb-drive, electronic dictionary and calculator. You are to switch off/silence your mobile phone and keep it out of view.
4. You may turn to the last page (page 8) to read some advice.
5. You will be logged into a special Windows account at the beginning of the PE. Do not log off until the end of the PE. Do not use your own NUSNET account.
6. A plab account slip will be issued to you at the beginning of the PE. Bring your matriculation card for identification when you collect it. <u>Please leave your matriculation card on the desk in front of you throughout the PE</u>.
7. You are to write your program in the given **plab account**. The host name is **plab4** (not sunfire!). <u>No activity should be done outside this plab account</u>.
8. You **do not** need to submit your programs to CodeCrunch. We will retrieve your programs and submit them to CodeCrunch after the PE.
9. Skeleton programs and some test data files are already residing in your plab account. Please leave the programs in the home directory, and use the same program names as specified in the paper. Do **not** create subdirectory to put your programs there or we will <u>not</u> be able to find them!
10. **Only your source codes (.c programs)** from your plab account will be collected after the PE. Hence, how you name your executable files is not important.
11. Please read carefully and follow all instructions in the question. If in doubt, please ask. Raise your hand and the invigilator will attend to you.
12. Any form of communication with other students or the use of unauthorised materials is considered cheating and you are liable to disciplinary action.
13. Please save your programs regularly during the PE.
14. When you are told to stop, please do so **<u>immediately</u>**, or you will be penalised.
15. At the end of the PE, please **log out from your plab account**.
16. Please check and take your belongings (especially matriculation card) before you leave.
17. We will make arrangement for you to retrieve your programs after we have finished grading. Grading may take a week or more.

ALL THE BEST!

Exercise 1: Most Frequent Letter *n*-gram [50 marks]

Problem Statement

A **letter *n*-gram** is a sequence of *n* letters. Letter *n*-grams with *n*=1 are called **unigrams** (or we normally just call them letters, *e.g.*, "p", "a" and "c"), while letter *n*-grams with *n*=2 are called **bigrams** (*e.g.,* "ae", "qb" and "ff").

Counting the frequencies of letter *n*-grams is very useful in cryptology. For example, if in a piece of cryptic English text, two letters frequently appear together, it is very likely that they correspond to "th" because "th" is the most frequently encountered bigram in English text.

In this exercise, you are to write a program **ngram.c** to find the most frequent letter unigram or bigram in an English text and report its frequency. The text comprises English words in <u>lower case with no punctuation</u>. All neighbouring words are separated by exactly one space.

For example, if the given text is "a friend in need is a friend indeed", the most frequent letter unigram is "e" which appears 6 times, while the most frequent letter bigram is "nd" which appears 3 times.

Do take note that for two letters to form a bigram, they must appear next to each other without any other letters or spaces in between them. In the previous example, the letters "a" and "f" never appears without any letters or spaces in between, therefore, bigram "af" does not appear in the text and its frequency is 0.

In the case where two unigrams or two bigrams have the same frequency, the one which appears earlier in alphabetical order should be reported.

Your program should read in a string which contains a sequence of English words in lowercase, and an integer, which is either 1 or 2. If the integer is 1, it should find and display the most frequent unigram with the frequency. If the integer is 2, it should find and display the most frequent bigram with the frequency.

You may assume that the input is valid and the maximum length of the string is 100 characters.

Write on the skeleton file **ngram.c** given to you. You must include the following two functions in your program:

- **`int mostFrequentUnigram(char text[], char result[])`**
  which takes in the string in the char array **`text`**, and returns the frequency of the most frequent unigram as an **`int`**. It should also return the unigram through the char array **`result`**.
- **`int mostFrequentBigram(char text[], char result[])`**
  which does the same as the previous function except that it is for bigram.

You may define additional functions as needed. Check sample runs for input and output format.

Sample Runs

Four sample runs are shown below with user input highlighted in **bold**.

```
Enter text: a friend in need is a friend indeed
Enter option: 1
Most frequent unigram: e
Frequency: 6
```

```
Enter text: a friend in need is a friend indeed
Enter option: 2
Most frequent bigram: nd
Frequency: 3
```

```
Enter text: mississippi is missing
Enter option: 1
Most frequent unigram: i
Frequency: 7
```

```
Enter text: mississippi is missing
Enter option: 2
Most frequent bigram: is
Frequency: 4
```

Exercise 2: 2048 Game [50 marks]

Problem Statement

2048 is a game in which the player moves numbered tiles in four possible directions (i.e., up, down, left and right) on a 4x4 grid. The objective is to merge the tiles to eventually create a tile with the number 2048. Tiles can move as far as possible in the chosen direction until they are stopped by either another tile or the edge of the grid. If a tile is stopped by another tile of the same value, both of them will merge into one tile with the total value of the two combined. Figure 1 shows a 2048 game in progress.

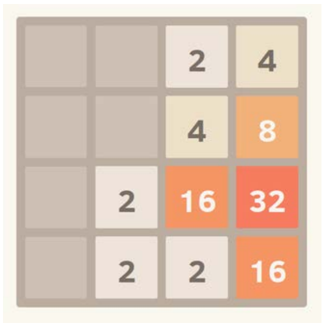

Figure 1. A 2048 game in progress

In this exercise, you are to write a program **2048.c** to simulate a simplified version of the 2048 game. In this simplified version, there are only two possible directions (i.e., up or left) for moving the tiles.

- When the move direction is **up**, the tiles at the higher row (i.e., with lower row index) are moved before the tiles at the lower row (i.e. with higher row index). For tiles in the same row, the tile on the left (i.e., lower column index) is moved before the tile on the right (i.e., higher column index).
- When the move direction is **left**, the tiles at the left column are moved before the tiles at the right column. For tiles in the same column, the tile at the top is moved before the tile at the bottom.

No new tiles are generated after each move.

For example, Figure 2a shows a sample grid with non-zero values denoting the tiles and 0s denoting spaces. If the direction of moving is up, the order of tiles chosen to move up is (0,1), (0,2), (1,0), (1,3) … and so on.

	0	1	2	3
0	0	**2**	**4**	0
1	**2**	0	0	**2**
2	**4**	0	0	**4**
3	0	**2**	**4**	0

Fig.2a. A sample 4x4 grid. Non-zero values denote tiles and 0s denote spaces.

	0	1	2	3
0	**2**	**2**	**4**	**2**
1	0	0	0	0
2	**4**	0	0	**4**
3	0	**2**	**4**	0

Fig.2b. After the tiles at (0,1), (0,2), (1,0) and (1,3) have moved up.

	0	1	2	3
0	**2**	**4**	**8**	**2**
1	**4**	0	0	**4**
2	0	0	0	0
3	0	0	0	0

Fig.2c. After the tiles at (2,0), (2,3), (3,1) and (3,2) have moved up.

Explanation of Figure 2:

> The tiles at (0,1) and (0,2) stay at the same positions since they are already at the edge of the grid, that is, they cannot move up further.
>
> Afterwards, the tiles at (1,0) and (1,3) move up and are stopped at (0,0) and (0,3) by the edge of the grid. *The status of the grid at this point is as shown in Figure 2b.*
>
> Subsequently, the tiles at (2,0) and (2,3) move up and are stopped at (1,0) and (1,3) by the two tiles at (0,0) and (0,3) respectively. Since the values of the tiles are not the same as the ones they are stopped by, no merging occurs.
>
> Lastly, the tiles at (3,1) and (3,2) move up and are stopped by two tiles at (0,1) and (0,2) of the same values respectively. Therefore, merging occurs and results in a tile of value 4 (*i.e.*, 2+2) at (0,1) and another of value 8 (*i.e.*, 4+4) at (0,2). *The final status of the grid is as shown in Figure 2c.*

The simulation starts with a given grid and follows a given sequence of directions. At the end of the simulation, a check will be performed on the grid to decide whether a target number has been formed. For example, in Figure 2c, if the target is 8, then the result of the check is *successful* because a tile of the value 8 can be found at (0,2).

Your program should read in four rows of four integers. The non-zero integers denote tiles while the 0s denote spaces in the grid. It should also read in a sequence of at most 20 characters consisting of 'U' and 'L', which represent moving up and left respectively. Lastly, it should read in an integer and check whether it is formed after moving the tiles in the given directions.

You may assume that the input is valid (*i.e.*, the integers are within the specified range and there are no illegal characters in the directions).

Write on the skeleton file **2048.c** given to you. You must include the following two functions in your program:

- **`void play(int grid[][SIZE], char directions[])`**
  which takes in a **`grid`** and a sequence of **`directions`**, and updates the grid based on the directions.
- **`int exists(int grid[][SIZE], int target)`**
  which takes in a **`grid`** and a number **`target`**, and returns 1 if **`target`** is found in the grid, or returns 0 otherwise.

You may define additional functions as needed. Check sample runs for input and output format.

Sample Runs

Five sample runs are shown below with user input highlighted in **bold**.

```
Enter grid:
0 0 0 0
0 0 0 0
0 0 0 0
1 1 1 1
Enter directions: U
Enter target: 1
1 1 1 1
0 0 0 0
0 0 0 0
0 0 0 0
1 is formed.
```

```
Enter grid:
0 0 0 1
0 0 0 1
0 0 0 1
0 0 0 1
Enter directions: L
Enter target: 2
1 0 0 0
1 0 0 0
1 0 0 0
1 0 0 0
2 is not formed.
```

```
Enter grid:
1 1 1 1
0 0 0 0
0 0 0 0
1 1 1 1
Enter directions: U
Enter target: 4
2 2 2 2
0 0 0 0
0 0 0 0
0 0 0 0
4 is not formed.
```

```
Enter grid:
1 0 0 1
1 0 0 1
1 0 0 1
1 0 0 1
Enter directions: L
Enter target: 2
2 0 0 0
2 0 0 0
2 0 0 0
2 0 0 0
2 is formed.
```

```
Enter grid:
2 0 0 2
0 1 1 0
0 0 0 0
1 1 1 1
Enter directions: UL
Enter target: 4
4 4 0 0
2 0 0 0
0 0 0 0
0 0 0 0
4 is formed.
```

CS1010 AY2014/5 Semester 1
Practical Exam 2 (PE2)

Advice – Please read!

- You are advised to spend the first 10 minutes for each exercise thinking and designing your algorithm, instead of writing the programs right away.
- If you write a function, you must have a function prototype, and you must put the function definition after the main() function.
- You may write additional function(s) not mentioned in the question, if you think it is necessary.
- Any variable you use must be declared in some function. You are **not** allowed to use global variables (variables that are declared outside all the functions).
- You may assume that all inputs are valid, that is, you do not need to perform input validity check.
- Manage your time well! Do not spend excessive time on any exercise.
- Be careful in naming your executable code. Do **not** overwrite your source code with your executable code, especially if you are using the –o option in gcc!
- The rough marking scheme for both exercises is given below.

Rough Marking Scheme For Each Exercise [50 marks]

1. Style: 10 marks
   - Are name, matriculation number, plab-id, DG and description filled at the top of the program?
   - Is there a description written at the top of every function (apart from the main() function)?
   - Are there proper indentation and naming of variables?
   - Are there appropriate comments wherever necessary?
2. Design: 10 marks
   - Are there correct definition and use of functions?
   - Are function prototypes present?
   - Is the right construct used?
   - Is algorithm not unnecessarily complicated?
3. Correctness: 30 marks
4. Deductions (not restricted to the following):
   - Program cannot be compiled: Deduct 10 marks
   - Compiler issues warning with –Wall: Deduct 5 marks.
   - Use of global variables: Deduct 10 marks

--- END OF PAPER ---