NATIONAL UNIVERSITY OF SINGAPORE

SCHOOL OF COMPUTING
PRACTICAL EXAMINATION II FOR
Semester 1 AY2020/2021

CS1010 Programming Methodology

November 2020 Time Allowed 2 Hours 30 Minutes

INSTRUCTIONS TO CANDIDATES

1. This assessment paper contains 5 questions and comprises 12 printed pages, including this page.
2. The total marks for this assessment is 45. Answer **ALL** questions.
3. This is an **OPEN BOOK** assessment.
4. You can assume that all the given inputs are valid.
5. You can assume that the types `long` and `double` suffice for storing integer values and real values respectively, for the purpose of this examination.
6. Login to the special account given to you. You should see the following in your home directory:
   - The skeleton code `tictactoe.c`, `sun.c`, `replace.c`, `soil.c` and `substring.c`
   - A file named `Makefile` to automate compilation and testing
   - A file named `test.sh` to invoke the program with its test cases
   - Two directories, `inputs` and `outputs`, within which you can find some sample inputs and outputs
7. You can run the command `make` to automatically compile and (if compiled successfully) run the tests.
8. Solve the given programming tasks by editing the given skeleton code. You can leave the files in your home directory and log off after the examination is over. There is no need to submit your code.
9. Only the code written in `tictactoe.c`, `sun.c`, `replace.c`, `soil.c` and `substring.c` directly under your home directory will be graded. Make sure that you write your solution in the correct file.
10. Marking criteria are (i) correctness and (ii) style. One mark is allocated for style for each question. The rest is allocated for correctness. For instance, a 4-mark question has 1 mark allocated for style and 3 marks allocated for correctness.
11. Note that correctness is defined in the broad sense of using the various programming constructs in C (type, function, variable, loops, conditionals, arithmetic expressions, logical expressions) properly – *not just producing the correct output*.

12. You should write code that is clean, neat, and readable to get the mark allocated to style.

Note: The original Practical Exam II is conducted as an online exam. The questions have been reformatted to fit into paper format. The questions are set by Dr. Daren Ler and Ooi Wei Tsang.

1 Tic Tac Toe (6 marks)

The game of Tic-Tac-Toe is typically played on a game board that consists of a 3-by-3 grid. Two players called X and O, take turns to mark each grid cell with X and O respectively. Player X wins if she put X in either the same column, the same row, or diagonally on the game board. Similarly, Player O wins if she put O in either the same column, the same row, or diagonally on the game board.

Write a program `tictactoe` that reads in a Tic-Tac-Toe board from the standard input. The board is represented by three lines of text, each with three characters consisting of either `X`, `O` or `?`. `?` indicates that the cell is empty and has not been marked by either player yet. You can assume that the board represents a valid Tic-Tac-Toe game.

Your program must determine if the game has ended and there is already the winner, given the game board. If there is a winner, print out the winner (X or O). Otherwise, print out the string "no winner". You can assume that there is at most one winner.

Your program must contain a boolean function called `has_won` that takes in the game board and a player, and returns whether the given player has won.

Marking Scheme

Style	1 marks
Correctness	3 marks
Efficiency	0 marks
Documentation	2 marks

Efficiency: No marks is allocated for efficiency. Your code is expected to run in $O(1)$ time. We may deduct marks if your implementation performs obvious redundant or duplicated work.

Documentation: You should document every function in this question according to the CS1010 documentation requirement.

Sample Runs

```
ooiwt@pe101:~/$ ./tictactoe
XO?
X?O
X??
X
ooiwt@pe101:~/$ ./tictactoe
OX?
XO?
X?O
O
ooiwt@pe101:~/$ ./tictactoe
OXO
XX?
O?O
no winner
```

2 Sun (6 marks)

A prime number is a positive integer larger than 1 that is divisible only by 1 and itself.

A triangular number is a number of the form $\frac{x(x+1)}{2}$, where x is a positive integer. The first few triangular numbers are 1, 3, 6, 10, 15. There are $O(\sqrt{n})$ triangular numbers smaller than n.

Wei-Zhi Sun, a mathematician, conjectured in 2008 that, any positive integer, except 216, can be expressed as $p + t$, where p is either 0 or a prime number, and t is either 0 or a triangular number.

Write a program `sun` that, reads in a positive integer n, and print all possible pairs of p and t, in increasing order of p, such that $p + t = n$. If no such pairs can be found, print nothing.

Marking Scheme

Style	1 marks
Correctness	2 marks
Efficiency	3 marks
Documentation	0 marks

Efficiency: Your code should run in $O(n)$ time. In addition, we will deduct marks if your code has obvious redundant or duplicated work. Your code must be correct or almost correct to receive the efficiency marks.

Documentation: You do not have to write Doxygen documentation for each function. You are still encouraged to comment on your code to help the grader understand your intention.

Sample Runs

```
ooiwt@pe101:~/$ ./sun
10
0 10
7 3
ooiwt@pe101:~/$ ./sun
18
3 15
17 1
ooiwt@pe101:~/$ ./sun
216
```

3 Replace (11 marks)

Given a string, we wish to repeatedly perform the search and replace operation on the string. In each search-and-replace operation, we wish to replace all occurrences of a target substring with another replacement string.

For instance, given the string Tengineering, if we replace the target Tin with the replacement Ter, we will end up with the string Tengereererg. Next, if we replace Tere with TO, we get TengOOrg.

The replacement happens by scanning the input string from left to right. Each occurrence is replaced in sequence. For instance, if we replace the occurrences Taa in the input Taaaa with Tb, the output is Tbb.

You can assume that the target string to replace is either longer or equal to the length of the replacement.

Write a program Treplace that performs a sequence of search-and-replace operation on an input string.

The input of the program is as follows:

- The first line contains the string s to be operated on. You can assume that the input string does not contain white spaces.
- The second line contains a positive integer k, which is the number of search-and-replace operations.
- In the next k lines, each line contains two strings, s_1 and s_2, with the length of $s_2 \leq$ length of s_1.

The output of the program is a single line containing the output string.

CONSTRAINT: You are not allowed to call methods from the C string library, except Tstrlen.

Marking Scheme

Style	1 marks
Correctness	5 marks
Memory Management	3 marks
Efficiency	2 marks
Documentation	0 marks

Memory Management: To obtain full marks for memory management, make sure that your code does not have memory leaks, even if memory allocation fails.

Efficiency: To obtain full marks for efficiency, your code only needs to run in $O(nk)$ times for each search-and-replace operation, where n is the length of the input string, and k is the length of the target string. Your code must be correct or almost correct to receive the efficiency marks.

Documentation: You do not have to write Doxygen documentation for each function. You are still encouraged to comment on your code to help the grader understand your intention.

Sample Runs

```
ooiwt@pe101:~/$ ./replace
aaaa
```

```
1
aa b
bb
ooiwt@pe101:~/$ ./replace
engineering
2
in er
ere O
engOOrg
```

4 Soil (11 marks)

In the farm owned by Old McDonald, a plot of land is divided into a grid of m-by-n cells. Old McDonald has hired you to look for a cell with the perfect soil salinity level to plant his crops. Armed with a salinity meter, you planned to check each of the $m \times n$ cells one-by-one, to find the cell with the right salinity level.

Just as you were about to start, Old McDonald pointed to a map of the grid and said, "There is something peculiar about this plot of land. If you walk through the cells row-by-row, from left to right, the soil salinity increases. If you walk column-by-column, from top to bottom, the salinity level increases. Further, no two cells have the same salinity level."

"Perfect," you said, "Now I only need to check $O(m + n)$ of the cells instead of $O(mn)$ cells!"

Write a program `soil` to simulate the process to search for salinity levels in a plot of land, by searching for a given integer in a 2D array of integers. Your program must read the following from the standard input.

The first line contains three positive integers, m, n, and k. m and nn represent the number of rows and columns of the 2D array.

The next m rows in the input contain the values to be stored in the 2D array. Each row has n integers. The following properties are guaranteed in the 2D array:

- In each row, the integers are in strictly increasing order
- In each column, the integers are in strictly increasing order
- The integers are unique in the 2D array.

The next k values represent the queries, i.e., the integers to search for. For each query, your program should look for it in the given 2D array, and either

- print the row and column indices of the 2D array that contains the value, separated by space, on its own line,
- or print the string "not found" if the corresponding value cannot be found.

Example

The following is a valid input 2D array:

```
1 4 8
2 5 9
3 7 10
```

Each row is sorted in increasing order from left to right, and each column is sorted in increasing order from top to bottom.

If the query to search for is 5, the program should print `1 1`. If the query is 6, the program should print `not found`.

Marking Scheme

Style	1 marks
Correctness	3 marks
Memory Management	1 marks
Efficiency	6 marks
Documentation	0 marks

Memory Management: You do not have to handle the case where memory allocation fails, but you should still ensure that your program does not have any memory leaks if it completes successfully.

Efficiency: Your algorithm must take $O(m+n)$ times to search a given query to obtain full marks. An $O(mn)$ solution is trivial and will receive 0 marks for efficiency. Any solution slower than $O(m+n)$ but faster than $O(mn)$ will receive partial marks for efficiency. Your solution must be correct or almost correct to receive the efficiency mark.

Documentation: You do not have to write Doxygen documentation for each function. You are still encouraged to comment on your code to help the grader understand your intention.

Sample Runs

```
ooiwt@pe101:~/$ ./soil
3 3 4
1 4 8
2 5 9
3 7 10
1
5
6
10
0 0
1 1
not found
2 2
ooiwt@pe101:~/$ ./soil
5 5 1
1  2  11 16 26
3  4  12 17 27
5  6  13 18 28
7  8  14 19 29
10 15 20 25 30
9
not found
```

5 Substring (11 marks)

Given a string, print all possible substrings of length k of the string.

The order in which the substrings are printed is defined as follows. Let s be in the input string; x and y be two substrings of s. Suppose that x_i and y_i is the first character that the substrings differ. If x_i appears before y_i in s, then x should be printed before y.

For instance, if $k = 4$ and the input is `singa`, the output should be

```
sing
sina
siga
snga
inga
```

Write a program `substring` that reads in a string s followed by a positive integer k from the standard input. You can assume that s contains only lowercase letters `a` to `z` with no repetition and k is at most the length of s.

The program should print to the standard output, all possible substring of s of length k in the order defined above, one substring per line.

Marking Scheme

Style	1 marks
Correctness	9 marks
Memory Management	1 marks
Efficiency	0 marks
Documentation	0 marks

Memory Management: You do not have to handle the case where memory allocation fails, but you should still ensure that your program does not have any memory leaks if it completes successfully.

Efficiency: There is no requirement for the running time of your algorithm. We may, however, still deduct marks if your code has obvious redundant or duplicated work.

Documentation: You do not have to write Doxygen documentation for each function. You are still encouraged to comment on your code to help the grader understand your intention.

Sample Runs

```
ooiwt@pe101:~/$ ./substring
singa 4
sing
sina
siga
snga
inga
ooiwt@pe101:~/$ ./substring
numbers 7
numbers
ooiwt@pe101:~/$ ./substring
```

```
done 1
d
o
n
e
```

END OF PAPER

This page is intentionally left blank.