NATIONAL UNIVERSITY OF SINGAPORE

SCHOOL OF COMPUTING
PRACTICAL EXAMINATION II FOR
Semester 1 AY2018/2019

CS1010 Programming Methodology

November 2018 Time Allowed 2 Hours 30 Minutes

INSTRUCTIONS TO CANDIDATES

1. This assessment paper contains 5 questions and comprises 8 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 `transpose.c`, `palindrome.c`, `rotate.c`, `marnell.c` and `bracket.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 `transpose.c`, `palindrome.c`, `rotate.c`, `marnell.c` and `bracket.c` directly under your home directory will be graded. Make sure that you write your solution in the correct file.
10. Marking criteria differ by the question. Please see the end of each question for details.

1 Transpose (6 marks)

A matrix can be transposed by flipping it along its diagonal, a row of the matrix becomes a column in the transposed matrix, and vice versa. For instance, the matrix

$$\begin{pmatrix} 0 & 2 \\ 1 & 1 \\ 0 & 9 \end{pmatrix}$$

when transposed, becomes

$$\begin{pmatrix} 0 & 1 & 0 \\ 2 & 1 & 9 \end{pmatrix}$$

Write a program `transpose` that takes an $m \times n$ matrix M_1 and transpose it into an $n \times m$ matrix M_2. Your program must implement the following function to transpose a matrix `m1` into `m2`:

```
void transpose_matrix(double **m1, double **m2, long nrows, long ncols);
```

where the pointer `m1` points to the matrix M_1 and the pointer `m2` points to the matrix M_2. The parameters `nrows` and `ncols` correspond to m and n (the number of rows and number of columns of M_1) respectively.

Your program should reads the following from the standard input:

- the integers m and n ($m \geq 1, n \geq 1$), followed by
- m rows of real numbers, each row containing n real numbers. Each row corresponds to a row of elements of the matrix M_1.

Your program then prints the transposed matrix M_2 to the standard output, each row of the output corresponds to a row of numbers in M_2.

Sample Run

```
ooiwt@pe101:~$ ./cat inputs/transpose.1.in
2 4
-1.0001 -2.0001 -3.0001 -4.0001
-5.0001 -6.0001 -7.0001 -8.0001
ooiwt@pe101:~$ transpose < inputs/transpose.1.in
-1.0001 -5.0001
-2.0001 -6.0001
-3.0001 -7.0001
-4.0001 -8.0001
```

Grading Criteria

- **Documentation (1 mark)** Write the Doxygen documentation block for the function `transpose_matrix` as well as other functions that you write.
- **Efficiency (1 mark)** Your implementation to transpose the matrix must take no more than $O(mn)$ time.
- **Style (1 mark)** Your code should be neat, readable, and easy to understand. Follow the CS1010 guidelines to avoid getting penalized for the style.

- **Correctness (3 marks)** Your code should not only produce the correct output but also uses the programming constructs provided by C appropriately. Further, you need to ensure that you explicitly free any memory that you allocate in your program.

You will only receive the marks for style, efficiency, and documentation if your code is reasonably correct (as judged by the graders).

2 Palindrome (6 marks)

A palindrome is a string that reads the same backward or forward, if we ignore all characters that are not alphabets (including spaces) and treat upper/lower cases as the same. For instance, the string "Was it a car or a cat I saw?", when converted into lower case letters without spaces and symbols, becomes the string "wasitacaroracatisaw", which reads the same backward or forward.

Write a program `palindrome` that reads, from the standard input, a string s containing at least one alphabet and prints, to the standard output, the string `yes` if the input string s is a palindrome, or the string `no` if s is not a palindrome.

Here are some tips that may be helpful:

- To check if a char `c` is a lowercase alphabet, you can check if `'a' <= c <= 'z'`,
- To check if a char `c` is a uppercase alphabet, you can check if `'A' <= c <= 'Z'`,
- To convert a char `c` from uppercase to lowercase, you can use the expression `c - 'A' + 'a'`.

You can also choose to use the functions `islower`, `isupper`, or `tolower`, available from `ctype.h` to do the above if you wish, but it is not necessary.

You may solve this problem either iteratively (with loops) or recursively.

Sample Runs

```
ooiwt@pe101:~$ ./palindrome
Was it a car or a cat I saw?
yes
ooiwt@pe101:~$ ./palindrome
it's a car
no
ooiwt@pe101:~$ ./palindrome
gg
yes
```

Grading Criteria

- **Documentation (0 marks)** You do not have to write the documentation blocks for each function. Note that you still need to comment your code, especially to explain your logic that might not be obvious to the readers.
- **Efficiency (1 mark)** Your implementation must take no more than $O(n)$ time, for input string of length n. A solution that is slower than $O(n)$ will receive 0 efficiency marks.
- **Style (1 mark)** Your code should be neat, readable, and easy to understand. Follow the CS1010 guidelines to avoid getting penalized for the style.
- **Correctness (4 marks)** Your code should not only produce the correct output but also uses the programming constructs provided by C appropriately. Further, you need to ensure that you explicitly free any memory that you allocate in your program.

You will only receive the marks for style and efficiency if your code is reasonably correct (as judged by the graders).

3 Rotate (9 marks)

Given an array, a rotation operation does the following: it moves every element to the right by one slot, except the last element, which is moved to the front and becomes the first (indexed 0) element. For example, the array `1 2 -4 0 8` becomes `8 1 2 -4 0` after one rotation.

Suppose we are given an array of n integers that is originally sorted but has been rotated 0 or more times (but we do not know how many times). Write a program that searches for an element in this array in $O(\log n)$ time.

Your program, 'rotate', reads, from the standard input, a positive integer n ($n \geq 1$), followed by n distinct integers that correspond to the elements of a sorted but possibly rotated array, followed by an integer q. Your program then prints, to the standard output, either the string 'not found' if q is not part of the array, or the index of q in the array if q is found. The first element in the array is indexed as 0.

Sample Run 1

```
ooiwt@pe101:~$ cat inputs/rotate.1.in
4
1 2 3 4
5
ooiwt@pe101:~$ ./rotate < inputs/rotate.1.in
not found
```

Sample Run 2

```
ooiwt@pe101:~$ cat inputs/rotate.2.in
8
1 2 3 4 5 -3 -2 0
4
ooiwt@pe101:~$ ./rotate < inputs/rotate.2.in
3
```

Grading Criteria

- **Documentation (0 marks)** You do not have to write the documentation blocks for each function. Note that you still need to comment your code, especially to explain your logic that might not be obvious to the readers.
- **Efficiency (5 mark)** Your implementation must take no more than $O(\log n)$ time for an array of size n. A linear solution is trivial and you will not receive any efficiency mark for a $O(n)$ solution.
- **Style (1 mark)** Your code should be neat, readable, and easy to understand. Follow the CS1010 guidelines to avoid getting penalized for the style.
- **Correctness (3 marks)** Your code should not only produce the correct output but also uses the programming constructs provided by C appropriately. Further, you need to ensure that you explicitly free any memory that you allocate in your program.

4 Marnell (12 marks)

In 2004, Geoffrey R. Marnell proposed the following conjecture: "Every number greater than 10 is the sum of a prime number and a semiprime".

A prime number is an integer larger than 1 that can only be divisible by 1 and itself. A semiprime is a number that is a product of two prime numbers (which may or may not be the same prime number). A square of a prime number is, therefore, a semiprime. The first 10 semiprimes are: 4, 6, 9, 10, 14, 15, 21, 22, 25, 26.

Write a program, `marnell`, that, reads, from the standard input, a positive number n, and prints, to the standard output, the number of pairs of prime number and semiprime that sums up to n. For example, 11 is the sum of the pairs:

- 2 and 9: 2 is prime; 9 = 3 $\times$ 3 is a semiprime
- 4 and 7: 7 is prime; 4 = 2 $\times$ 2 is a semiprime
- 5 and 6: 5 is prime; 6 = 2 $\times$ 3 is a semiprime

None of the pairs that sum up to 10 consists of a prime and a semiprime:

- 2 and 8: 2 is prime; 8 is not a semiprime.
- 3 and 7: 7 is prime; 3 is a prime.
- 4 and 6: Both 4 and 6 are semiprime.
- 5 and 5: 5 is a prime.

So your program `marnell` should print `0` when the input is `10`, and `3` when the input is `11`.

Sample Runs

```
ooiwt@pe101:~$ ./marnell
10
0
ooiwt@pe101:~$ ./marnell
11
3
```

Grading Criteria

- **Documentation (0 marks)** You do not have to write the documentation blocks for each function. Note that you still need to comment your code, especially to explain your logic that might not be obvious to the readers.
- **Efficiency (5 mark)** Your implementation must take no more than $O(n\sqrt{n})$ time, for input string of length n. Furthermore, you should avoid redundant work and repetitive work as much as possible.
- **Style (2 marks)** Your code should be neat, readable, and easy to understand. Follow the CS1010 guidelines to avoid getting penalized for the style.
- **Correctness (5 marks)** Your code should not only produce the correct output but also uses the programming constructs provided by C appropriately. Further, you need to ensure that you explicitly free any memory that you allocate in your program.

5 Bracket (12 marks)

Consider a string consisting of only four types of open brackets `(`, `[`, `<` and `{` and the corresponding matching close brackets `)`, `]`, `>` or `}`. We say that such a string is *valid* according to the following rules:

1. An empty string (a string with no character) is always valid;
2. A non-empty string must be either (i) a valid string followed by another valid string, or (ii) starts with an open bracket ends with a matching close bracket, and contain a valid string in between.

Let's look at some examples:

- The string `<>([])` is valid. It consists of two valid strings `<>` and `([])`. `<>` is valid since it starts with an open bracket `<`, ends with a matching close bracket `>`, and contains a valid (empty) string in between. `([])` is also valid since it starts with an open bracket `(`, ends with a matching close bracket `)`, and contains a valid string `[]` in between.
- The string `<(>)` is invalid since we cannot partition it into two valid strings, nor does it contains matching open and close brackets. `<` and `)` do not match.

Write a program `bracket` that reads, from the standard input, a string with at least two characters, consisting of only open and close brackets, and prints, to the standard output, `yes` if the string is valid, `no` if the string is not valid.

NOTE: You must solve this problem using recursion.

Sample Runs

```
ooiwt@pe101:~$ ./bracket
<>([])
yes
ooiwt@pe101:~$ ./bracket
((()))<>[]){}
yes
\begin{Verbatim}
ooiwt@pe101:~$ ./bracket
<(>)
no
```

Grading Criteria

- **Documentation (0 marks)** You do not have to write the documentation blocks for each function. Note that you still need to comment your code, especially to explain your logic that might not be obvious to the readers.
- **Efficiency (5 mark)** Your implementation must take no more than $O(n)$ time, for input string of length n. Furthermore, you should avoid redundant work and repetitive work as much as possible.
- **Style (2 marks)** Your code should be neat, readable, and easy to understand. Follow the CS1010 guidelines to avoid getting penalized for the style.
- **Correctness (5 marks)** Your code should not only produce the correct output but also uses the programming constructs provided by C appropriately. Further, you need to ensure that you explicitly free any memory that you allocate in your program, and you have to solve this problem using recursion.

This page is intentionally left blank.

END OF PAPER