NATIONAL UNIVERSITY OF SINGAPORE

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

CS1010 Programming Methodology

October 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 30. 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 `vote.c`, `newton.c`, `goldbach.c`, `digits.c` and `square.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 `vote.c`, `newton.c`, `goldbach.c`, `digits.c` and `square.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.

1 Vote (4 marks)

The Planet Earth is having its first democratic election to elect the first President of the World, from two candidates, McNeal and Nixon.

Write a program `vote` that reads, from the standard input, two integers representing the number of votes received by McNeal and Nixon in that order, and print, to the standard output, the percentage of votes received by the two candidates, in the same order.

You can assume that there is at least one vote in the election.

Sample Run 1

```
ooiwt@pe101:~$ ./vote
10 0
100.0000 0.0000
```

Sample Run 2

```
ooiwt@pe101:~$ ./vote
1000000000 3000000000
25.0000 75.0000
```

2 Newton (4 marks)

Newton's method is a numerical method used to determine the root of a mathematical function $f(x)$. In other words, it finds an x such that $f(x)$ is 0. It starts with an initial guess x_0 of a root of $f(x)$, and successively calculates x_i by:

$$x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)},$$

where $f'(x_i)$ is the derivative of $f(x)$. This process continues until $f(x_n)$ is close enough to the root.

In this question, you will write a program that calculates the root of a cubic function $f(x) = ax^3 + bx^2 + cx + d$ using the Newton's method. Recall that, $f'(x) = 3ax^2 + 2bx + c$.

Write a program `newton` that reads, from the standard input, five real values corresponding to a, b, c, d, and x_0, in that order. Print to the standard output the root of the cubic function as obtained by Newton's method starting from the initial guess x_0. For the purpose of this question, we say that x is close enough to a root if $|f(x)| < 0.000000001$.

Example

Suppose we have the input `1 2 0 5 -3`. We wish to find the root to the polynomial $f(x) = x^3 + 2x^2 + 5$ starting with $x_0 = -3$. The derivative $f'(x)$ is $3x^2 + 4x$. Executing the Newton's method, we get the following:

i	$f(x_i)$	$f'(x_i)$	x_{i+1}
0	-4.0000	15.0000	-2.7333
1	-0.4788	11.4800	-2.6916
2	-0.0107	10.9680	-2.6906
3	-0.0000	10.9562	-2.6906

Since $f(x_4)$ is close enough to 0, we output x_4 (`-2.6906`) as the root.

Sample Run 1

```
ooiwt@pe101:~$ ./newton
1 2 0 5 -3
-2.6906
```

Sample Run 2

```
ooiwt@pe101:~$ ./newton
0 1 0 -9 1
3.0000
```

Sample Run 3

```
ooiwt@pe101:~$ ./newton
1.5 2.5 3.5 4.5 11.5
-1.4687
```

3 Goldbach (6 marks)

The Goldbach Conjecture is one of the oldest unsolved problems in mathematics. It states that ”every even number is the sum of two prime numbers”. This conjecture is not yet proven to be true for all even numbers, but have been shown to be true for even numbers up to 400,000,000,000,000.

Write a program, `goldbach`, that, reads, from the standard input, an even number n, and prints, to the standard output, the number of pairs of prime numbers where their sum is n. For example, 10 is the sum of the prime pair (3, 7) and (5, 5). So your program `goldbach` should print 2 when the input is `10`. 100 is the sum of the prime pair (3, 97), (11, 89), (17, 83), (29, 71), (41, 59), and (47, 53) So your program should print 6 when the input is `100`.

Note: A prime number is an integer larger than 1 that can only be divisible by 1 and itself.

Sample Run 1

```
ooiwt@pe101:~$ ./goldbach
2
0
```

Sample Run 2

```
ooiwt@pe101:~$ ./goldbach
10
2
```

Sample Run 3

```
ooiwt@pe101:~$ ./goldbach
100
6
```

4 Digits (8 marks)

An integer consists of multiple consecutive sequences of digits. For instance, the number `800100022`, has five sequences, `8`, `00`, `1`, `000`, `22`. The sequence `000` is the longest.

Write a program `digits` that reads, from the standard input, an integer, and prints to the standard output, the digit that occurs in the longest consecutive sequence. If there are multiple consecutive longest sequences, break ties by printing the digit that is the smallest.

You must not use arrays to solve this problem, or you risk getting 0 marks for this question.

Sample Run 1

```
ooiwt@pe101:~$ ./digits
800100022
0
```

Sample Run 2

```
ooiwt@pe101:~$ ./digits
11112222
1
```

Sample Run 3

```
ooiwt@pe101:~$ ./digits
1234980
0
```

5 Square (8 marks)

Write a program `square` that reads, from the standard input, an integer n ($n < 200$) and prints, to the standard output, a set of squares with width n, $n-4$, $n-8$, .. until we reach either the width of 3, 2, 1, or 0. The smaller square is contained in the larger squares. The squares do not touch each other. A square has exactly one space between itself and the next larger square (if exists), in each direction.

You must not use arrays to solve this problem, or you risk getting 0 marks for this question. The only arrays you are allowed to use are in the form of strings `"#"` and `" "`.

You must draw the squares *recursively*, by defining a recursive function that matches the following declaration:

```
  void draw_square(long row, long width);
```

to draw the `row`-th row of the square with width `width`. As an example of how this function is called, the `main()` function has been given and it looks like:

```
#include "cs1010.h"

int main()
{
  long n = cs1010_read_long();
  for (long i = 0; i < n; i += 1) {
    draw_square(i, n);
    cs1010_println_string("");
  }
}
```

Sample Run 1

```
ooiwt@pe101:~$ ./square
1
#
```

Sample Run 2

```
ooiwt@pe101:~$ ./square
2
##
##
```

Sample Run 3

```
ooiwt@pe101:~$ ./square
3
###
# #
###
```

Sample Run 4

```
ooiwt@pe101:~$ ./square
4
####
#  #
#  #
####
```

Sample Run 5

```
ooiwt@pe101:~$ ./square
5
#####
#   #
# # #
#   #
#####
```

Sample Run 6

```
ooiwt@pe101:~$ ./square
6
######
#    #
# ## #
# ## #
#    #
######
```

Sample Run 7

```
ooiwt@pe101:~$ ./square
7
#######
#     #
# ### #
# # # #
# ### #
#     #
#######
```

Sample Run 8

```
ooiwt@pe101:~$ ./square
10
##########
#        #
# ###### #
# #    # #
# # ## # #
# # ## # #
# #    # #
# ###### #
#        #
##########
```

END OF PAPER

This page is intentionally left blank.