NATIONAL UNIVERSITY OF SINGAPORE
SCHOOL OF COMPUTING

Practical Examination 1 (PE1) for Semester 1, AY2016/17
CS1010 — Programming Methodology

17 September 2016 Time Allowed: 2 hours

INSTRUCTION TO CANDIDATES

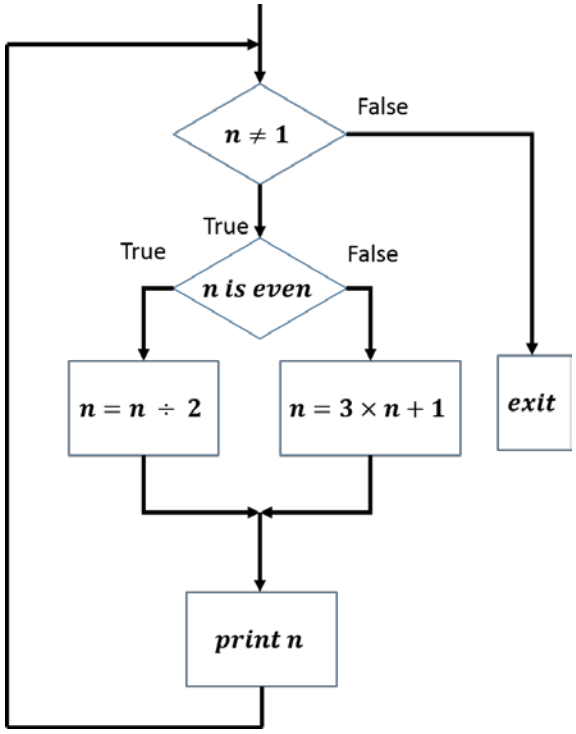
1. You are only allowed to read this cover page and the last page. Do **not** read the questions until you are told to do so.
2. This paper consists of **2** exercises on **6** pages. Each exercise constitutes 50%. The exercises have different levels of difficulty. So, beware of time spent on each exercise.
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 6) 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. Please leave your matriculation card on the desk in front of you throughout the PE.
7. You are to write your program in the given **plab account**. The host name is **plab4** (not sunfire!). No activity should be done outside this plab account.
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 (**collatz.c** and **numerology.c**) are already residing in your plab account. **Write your solutions in these two programs**. Leave these programs in your plab account home directory at the end of the PE. Do **not** change the file names, and do **not** create subdirectory to put your programs there or we will **not** be able to find them!
10. **Only your two source codes (collatz.c** and **numerology.c)** from your plab account will be collected after the PE. Hence, how you name your executable files is not important.
11. Read carefully and follow all instructions in the question. If in doubt, please ask. Raise your hand and the invigilator will attend to you. Don't be shy.
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 **immediately**, 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: Collatz's Problem [50 marks]

Problem Statement

Following is the control flow graph describing the computation of Collatz's sequence.



Given an integer value n, a program with the above control-flow graph prints out the current value of n at every iteration, until control reaches the exit point. Both the division and multiplication operate on integer values.

Write a program **collatz.c** that reads in two positive integers, n and $count$, where n is the input to the Collatz's sequence, and $count$ is the maximum number of iterations (and thus the maximum number of times the value of n is printed) required for execution.

The program should call a function $displayCollatzInternal$ which does the following:

- Compute Collatz's sequence by printing out the values of n at each iteration, for up to $count$ number of iterations.
- Print out the number of times the division statement "$n = n \div 2$" and the multiplication statement "$n = n \times 3 + 1$" have been executed.

You are to decide the appropriate parameter(s), return type and precondition (if any) for this function. You may define additional functions as needed.

You may assume that the inputs are valid (*i.e.*, positive integers).

Sample Runs

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

```
Enter positive numbers for n and count: 8 10

The collatz sequence is: 4 2 1
The division statement has been executed 3 times.
The multiplication statement has been executed 0 times.
```

```
Enter positive numbers for n and count: 28 7

The collatz sequence is: 14 7 22 11 34 17 52
The division statement has been executed 4 times.
The multiplication statement has been executed 3 times.
```

```
Enter positive numbers for n and count: 1 5

The collatz sequence is:
The division statement has been executed 0 times.
The multiplication statement has been executed 0 times.
```

Skeleton Program

A skeleton program **collatz.c** is available in your plab account and is shown below.

```
// CS1010 AY2016/7 Semester 1
// PE1 Ex1: collatz.c
// Name:
// Matriculation number:
// plab account-id:
// Discussion group:
// Description:

#include <stdio.h>

int main(void) {
    // Get user input
    printf("Enter positive numbers for n and count: ");
    printf("\n");

    // display the internal execution of Collatz's sequence

    return 0;
}

/* Sample printf statements for displayCollatzInternal()
    printf("The collatz sequence is: ");
…
*/
```

Exercise 2: Numerology [50 marks]

You are given a task by a numerologist to write a program, **numerology.c**, which gives advice to a client based on a number provided by the client. The number num provided is a non-zero integer and it is the only input to your program.

To give appropriate advice, we check both the sign (positive or negative) **and** the absolute value of num. From the absolute value of num, we need to derive a digit dgt.

To obtain the digit dgt from the (absolute value) of num, we follow a process called **multiply2Digit** as given below:

- We begin with the given absolute value of num, denoted as n, and multiply all digits in n. This produces a new number n'.
- If n' contains just one digit, then dgt is n'.
- If n' contains more than one digit, then we repeat the **multiply2Digit** process again, but now on n'. We repeat this process until we obtain a number with one digit, and assign that number to dgt.

Shown below are two examples on how the digit dgt is obtained from the number num:

	Example 1	**Example 2**
num	12935	-8281
Absolute value of num	12935	8281
Multiplying the digits	Iteration 1: 1 x 2 x 9 x 3 x 5 = 270 Iteration 2: 2 x 7 x 0 = 0	Iteration 1: 8 x 2 x 8 x 1 = 128 Iteration 2: 1 x 2 x 8 = 16 Iteration 3: 1 x 6 = 6
Digit dgt produced	0	6

Table 1. Examples for the **multiply2Digit** process

The advice given by your program depends on the digit dgt produced from num **and** *the sign of* num (*positive* or *negative*), as follows:

Sign of the input number	**Digit produced**	**Advice to be given**
Positive	0,3,7,9	You should protect life.
	2,5,8	Share your wealth, donate generously.
	1,4,6	Build harmony, bring people together.
Negative	0,3,7,9	Speak honestly.
	2,5,8	Praise others' successes.
	1,4,6	Lend your hand to those who are in need.

Table 2. Advice to be given based on the digit produced

Write on the skeleton file **numerology.c** given to you. You should include the following functions in your program:

- $multiply2Digit$: Compute the digit from the absolute value of the input number, as described in the repetitive process mentioned earlier.
- $giveAdvice$**:** Print out advice given the digit produced and the sign of the input number.

You are to decide the appropriate parameters, return types and preconditions (if any) for these function. You may define additional functions as needed.

You may assume that the input is valid (*i.e.*, a non-zero integer).

Sample Runs

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

```
Enter a non-zero integer: 12935
Advice for you: You should protect life.
```

```
Enter a non-zero integer: -8281
Advice for you: Lend your hand to those who are in need.
```

```
Enter a non-zero integer: 5
Advice for you: Share your wealth, donate generously.
```

Skeleton Program

A skeleton program **numerology.c** is available in your plab account and is shown below.

```
// CS1010 AY2016/7 Semester 1
// PE1 Ex2: numerology.c
// Name:
// Matriculation number:
// plab account-id:
// Discussion group:
// Description:

#include <stdio.h>

int main(void) {
   printf("Enter a non-zero integer: ");

   return 0;
}

/* Sample printf statements for giveAdvice()
   printf("Advice for you: ");
…
*/
```

CS1010 AY2016/7 Semester 1
Practical Exam 1 (PE1)

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.
- You are **not** allowed to use recursion or string functions from string.h. If in doubt, please check with an invigilator.
- If you write a function, you must have a function prototype, and you must put the function definition after the main() function.
- Unless it is specified otherwise, 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 is given below.

Rough Marking Scheme For Each Exercise [50 marks]

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?

Design [10 marks]

- Are there correct definition and use of functions?
- Are function prototypes present?
- Is the right construct used?
- Is the algorithm not unnecessarily complicated?

Correctness [30 marks]

Deductions (not restricted to the following):

- Program cannot be compiled: Deduct 10 marks
- Compiler issues warning with –Wall: Deduct 5 marks.
- Use of recursion, built-in string functions from string.h: Deduct 10 marks
- Use of global variables: Deduct 10 marks

--- END OF PAPER ---