NATIONAL UNIVERSITY OF SINGAPORE
SCHOOL OF COMPUTING

Practical Examination 1 (PE1) for AY2017/8 Semester 1
CS1010 Programming Methodology

22 September 2017 Time Allowed: 1 hour 30 minutes

INSTRUCTION TO CANDIDATES

1. Please leave your student card on your desk throughout the PE.
2. You are only allowed to read this cover page and the last page before the start of the PE. Do **not** flip the pages to read the questions inside until you are told to do so.
3. This paper consists of **2** tasks on 6 pages.
4. 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.
5. You may turn to the last page (page 6) to read some advice.
6. 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.
7. A plab account slip will be issued to you at the beginning of the PE.
8. You are to write your program in the given **plab account**. The host name is **pe10** (not sunfire!). No activity should be done outside this plab account.
9. You are not allowed to type your programs in the first ten minutes of the PE.
10. You **do not** need to submit your programs to CodeCrunch. We will retrieve your programs and submit them to CodeCrunch after the PE.
11. Skeleton programs and some test data files are already residing in your plab account. Please leave your 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, and do **not** name your programs differently from what are given, otherwise we will not be able to find them!
12. **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.
13. 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.
14. Any form of communication with other students or the use of unauthorised materials is considered cheating and you are liable to disciplinary action.
15. Please save your programs regularly during the PE.
16. When you are told to stop, please do so **immediately**, or you will be penalised.
17. At the end of the PE, please **log out from your plab account**.
18. Please check and take your belongings (especially your student card) before you leave.
19. 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!

Task 1: 'Beeboo' Value and Code [30 marks]

Task Statement

Given a positive integer (of type **int**), you are to compute its 'beeboo' value and 'beeboo' code.

What is the 'beeboo' value of a number? You take its first digit, let's call it *first*, its last digit, let's call it *last*, and assuming that *digits* is the number of digits in the number, the 'beeboo' value is computed as follows:

$$(first^2 + last^2) \times digits$$

Once you have obtained the 'beeboo' value, you are to determine the 'beeboo' code. The 'beeboo' code is a capital letter obtained by looking at the last digit of the 'beeboo' value, and is generated as follows:

Last digit of 'beeboo' value	0	1	2	3	4	5	6	7	8	9
'Beeboo' code	C	D	E	F	G	H	I	J	K	L

The following table shows a few examples.

Number	12345	9	900	5031	764	113355
'Beeboo' value	130	162	243	104	195	156
'Beeboo' code	C	E	F	G	H	I

A skeleton program **beeboo.c** is given. Do not rename it to something else, or move it into a sub-directory, or we may not be able to find your program.

The input is a positive integer. The output consists of three lines: the first line contains the value entered by user, the second line contains the 'beeboo' value and the third line contains the 'beeboo' code.

You must write a function to compute the 'beeboo' value.

See the sample runs on the next page.

Sample Runs

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

```
Enter a positive integer: 12345
12345
130
C
```

```
Enter a positive integer: 9
9
162
E
```

```
Enter a positive integer: 900
900
243
F
```

Task 2: 'Seesoo' and 'Teetoo' Values [70 marks]

Task Statement

Given a positive integer *n*, which can be a very big number so you are to use the type **long long** (see more information in the box below), find its 'seesoo' value and 'teetoo' value.

The **long long** type allows you to use integers in a bigger range of values. In sunfire, the **long long** type takes 8 bytes, hence it can represent integers in the range from -2^{63} through $2^{63} - 1$, or [-9223372036854775808, +9223372036854775807].

The format specifier for **long long** type in scanf() and printf() is **%lli**

'Seesoo' value

The 'seesoo' value is obtained by summing up all the digits in *n*. If the sum contains more than one digit, you are to sum up all the digits in it again, and repeat this process until eventually the sum contains a single digit, which is the 'seesoo' value of *n*.

For example, if *n* is 123456789012345678, then its 'seesoo' value is 9 (1+2+3+4+5+6+7+8+9+0+1+2+3+4+5+6+7+8 = 81; 8 + 1 = 9). If *n* is 999777555333111, then its 'seesoo' value is 3 (9+9+9+7+7+7+5+5+5+3+3+3+1+1+1 = 75; 7+5 = 12; 1+2 = 3).

You must write a function to compute the 'seesoo' value. Your algorithm should be a general one that works on any arbitrarily large positive integer.

Those who know Number Theory or who have taken CS1231 might be aware that the solution can be obtained by this algorithm:

```
Algorithm seesoo(num) {
    answer = num%9;
    if (answer == 0) return 9;
    else return answer;
}
```

However, for this PE, you are not to use the above algorithm. You are to use repetition statement to solve this task.

'Teetoo' value

The 'teetoo' value of a number *n* is obtained by splitting *n* into groups of width 3 from left to right, with one digit separating two neighbouring groups, and finding the sum of these groups.

For example, if the given positive integer *n* is 123456789012345678, then *n* can be split into groups of width 3 from left to right as follows:

123 4 567 8 901 2 345 6 78

Note that the last group may have fewer than 3 digits.

The sum of these groups is 123 + 567 + 901 + 345 + 78 = 2014. Hence 2014 is its 'teetoo' value.

As another example, if *n* is 10020340567, then *n* can be split as follows:

100 2 034 0 567

The sum of these groups is 100 + 34 + 567 = 701. Hence 701 is its 'teetoo' value.

You must write a function to compute the 'teetoo' value.

A skeleton program **teetoo.c** is given. Do not rename it to something else, or move it into a sub-directory, or we may not be able to find your program.

The input consists of a positive integer of type **long long**. The output consists of three lines: the first line contains the value entered by user, the second line contains the 'seesoo' value and the third line contains the 'teetoo' value.

Sample Runs

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

```
Enter a positive integer: 1234567890123456789
1234567890123456789
9
2014
```

```
Enter a positive integer: 10020340567
10020340567
1
701
```

```
Enter a positive integer: 987
987
6
987
```

CS1010 AY2017/8 Semester 1
Practical Exam 1 (PE1)

Advice – Please read!

- In the first 10 minutes of the PE, you are not allowed to type your programs. Use that time to read the task statements, ask questions and design your algorithms.
- Your program must be compilable! Or you will receive zero mark for correctness.
- You are **not** allowed to use recursion, arrays or string functions from string.h. Heavy penalty will be given (see below) if you use any of them. If in doubt, please ask.
- Any variable used must be declared within some function. You are **not** allowed to use global variables (variables that are declared outside all the functions). Heavy penalty will be given (see below) if you use any global variable.
- If you write a function, you must have a function prototype, and you should put the function definition after the main() function.
- You may write additional function(s) not mentioned in the task statement if you think it is necessary.
- 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 task.
- 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!

Rough Marking Scheme

1. Style: 5 marks
   - Are your name, student 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 variables unnecessarily initialized, or not initialized when they are supposed to be?
   - Are there appropriate comments wherever necessary?
2. Design: 5 marks
   - Is the program modular?
   - Are function prototypes present?
   - Are the functions correctly defined and called?
   - Is function cohesion abided by?
   - Is algorithm not unnecessarily complicated?
3. Correctness: 20 marks for Task 1, 60 marks for Task 2. **Zero mark if cannot be compiled**.
4. Deductions (not restricted to the following):
   - Program cannot be compiled – Zero mark for correctness
   - Compiler issues warning with –Wall: Deduct 5 marks.
   - Use of recursion, array, built-in string functions from string.h: Deduct 10 marks
   - Use of global variables: Deduct 10 marks

--- END OF PAPER ---