NATIONAL UNIVERSITY OF SINGAPORE
SCHOOL OF COMPUTING

Practical Examination 1 (PE1) for Semester 2, AY2017/18
CS1010E – Programming Methodology

24 February 2018 Time Allowed: 2 hours

INSTRUCTION TO CANDIDATES

1. You are only allowed to read the **front** and the **back** cover page. Do not read the question paper until you are told to do so.
2. This paper consists of 2 exercises on 6 pages. Each exercise constitutes 50 marks, the total mark is 100.
3. This is an **open-book** exam. You may bring in any printed material, but **not** electronic devices. You are also to switch off/silence your mobile phone and keep it out of view.
4. The entire examination is 2 hours. You are **not allowed** to leave the examination venue early.
5. The computer you have been assigned to is already logged in. Do not log off. **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 must write your programs in the given **plab account**. The host name is **pe10** (**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** are incomplete C programs which you need to complete them for submission. They are already residing in your plab account. Please leave these programs in the home diretory, and use the same program names as given. **Do not** create subdirectory to put your programs there or we will not be able to find them! The program names to be submitted for the two exercises are `numChk.c` and `altSum.c` respectively.
10. **Only your source code (`.c` programs)** from your plab account will be collected after the PE. Hence, how you name your executable files is not important.
11. 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. Don't be shy.
12. Any form of communication with other students, or the use of unauthorized materials is considered cheating and you are liable to disciplinary action.
13. Please save your programs regularly during the PE. Also, **make a copy of the original skeleton programs** so that you won't panic when you accidentally delete the original programs.
14. When you are told to stop, do so **immediately**, or you will be penalized.
15. At the end of the PE, please **log out from your plab account**. The invigilator will then shut down your PC remotely.
16. Please check and keep your belongings (esp. 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.

PLEASE TURN TO THE LAST COVER PAGE
TO READ MORE ABOUT THIS EXAMINATION.

Exercise 1: Number-check Conditions(50 marks)

A positive integer n is said to possess a "maximum distance" max (where max is a non-negative integer) if **no** two digits in n can be of values which are more than max apart. For instance, given the max is 7, the number "229485" possess the "maximum distance 7" property since the difference between any two digits from the number is never greater than 7. On the other hand, "129485" does **not** possess the "maximum distance 7" property because there exists two digits, 1 and 9, the difference of which exceeds 7. Lastly, if the number has only 1 digit, then it always possesses the "maximum distance" max property.

Next, a positive integer n is said to meet a "sum-divide" condition if the sum of the digits in the number is divisible by 3. For instance, "12984" has all its digits sum up to 24, which is divisible by 3, and thus satisfying the "sum-divide" condition. On the other hand, "229486" has all its digits sum up to 31, which is not divisible by 3, and thus does not satisfy the "sum-divide" condition.

Your Task: Complete the given program skeleton named `numChk.c` into a C program that does the following:

> Given that the user inputs a positive integer num, a positive range $range$, and a digit (ranged between 0 and 9) called max, check every integer between num and $(num + range - 1)$ (both ends are included) to determine how many of them **possess only** maximum distance max condition, how many of them **possess only** sum-divide condition, and how many **satisfy both** conditions.

A program skeleton `numChk.c` is provided to you. Beside containing the `main` function (which **you are not allowed to modify**), it also contains the skeletons of the following three programmer-defined functions, which you must make use of (according to their purposes) in your program construction:

1. `int maxDist(int,int)`: This function aims to check the validity of the "maximum distance" property of a number. It returns integer 1 if the number possesses this property, and returns 0 otherwise.
2. `int sumDigits(int)`: This function sums up all the digits in the argument.
3. `void numChk(int,int,int)`: This function performs the main task of checking. It is called by the `main` function, and it must make meaningful call to `maxDist` and `sumDigits` to help complete the checking task. It also prints out the result to the user.

You are required to complete the definitions of these functions when completing your program.

Following is a sample run, with user input shown in **bold**. Here, numbers "89" and "92" satisfies only the "maximum distance 7" condition; "90" satisfies only the "sum-divide"condition; "93" satisfies both requirements; lastly, "91" satisfies none of the two conditions.

```
Enter the starting number: 89
Enter the range: 5
Enter the maximum distance allowed between two digits: 7
Number of numbers only with max distance of 7 is: 2.
Number of numbers only satifying sum-divide requirement is: 1.
Number of numbers meeting both requirements is: 1.
```

Exercise 2: Alternating Sum of an Arithmetic Progression (50 marks)

An *arithmetic progression* (AP) is a number series that can be defined by:

1. The first item in the series is a_1;
2. The difference between an item and its previous in the series is always d; ie., $a_{i+1} - a_i \; = \; d$ for all $i \geq 1$.

Under this definition, the series is of the form:

$$a, \; a+d, \; a+2d, \ldots$$

The n^{th} term of the AP is defined as: $a + (n-1)d$.

Consider an AP with finite number of items, a_1,a_2, ..., a_n, for some positive integer n.

This exercise takes in an AP, and computes the alternating sum of the series, as follows:

$$\begin{aligned}
\mathit{altSum} &= \sum_{i=1}^{n} (-1)^{i-1} \mathit{altfold}_i \\
\mathit{altfold}_i &= \sum_{j=i}^{n} \mathit{prod}(i,j) \\
\mathit{prod}(i,j) &= \begin{cases} (-1)^{j-i}(a_j \times a_{n-j+i}) & \text{if } j \leq n-j+i \\ 0 & \text{otherwise} \end{cases}
\end{aligned}$$

As an example, consider an AP of 5 items, with $a_1 = 1$ and $d = 1$. The AP is:

$$a_1 = 1; \; a_2 = 2; \; a_3 = 3; \; a_4 = 4; \; a_5 = 5.$$

Then,

$$\begin{aligned}
\mathit{altfold}_2 &= \sum_{j=2}^{5} \mathit{prod}(2,j) \\
&= (a_2 \times a_5) - (a_3 \times a_4) + 0 + 0 \\
&= -2
\end{aligned}$$

Furthermore,

$$\begin{aligned}
\mathit{altSum} &= \sum_{i=1}^{5} (-1)^{i-1} \mathit{altfold}_i \\
&= \mathit{altfold}_1 - \mathit{altfold}_2 + \mathit{altfold}_3 - \mathit{altfold}_4 + \mathit{altfold}_5 \\
&= \;\; ((a_1 \times a_5) - (a_2 \times a_4) + (a_3 \times a_3)) && // \; \mathit{altfold}_1 \\
&\quad - ((a_2 \times a_5) - (a_3 \times a_4)) && // \; \mathit{altfold}_2 \\
&\quad + ((a_3 \times a_5) - (a_4 \times a_4)) && // \; \mathit{altfold}_3 \\
&\quad - (a_4 \times a_5) \; + \; (a_5 \times a_5) && // \; \mathit{altfold}_4 \text{ and } \mathit{altfold}_5 \\
&= 6 - (-2) + (-1) - 20 + 25 = 12.
\end{aligned}$$

Your Task: Complete the given program skeleton named `altSum.c` to form a C program that does the following:

The program will takes in three integers relating to an AP:

1. The value of the first item of the AP;
2. The difference (which can either be positive or negative) between an item and its previous item in the AP; and
3. The number of items (at least 1) in the AP.

It then computes the alternating sum result with regard to this AP.

The program skeleton `altSum.c` is provided in you plab account. Beside having a `main` function (which **you are not allowed to modify**), it also contains the skeletons of the following two programmer-defined functions, which you must make use of (according to their purposes) in constructing your program:

1. `int altSum(int, int, int)`: This is the most important programmer-defined function, which takes in the three values characterizing an AP, and computes the alternating sum. It **must** call the function `altFold` to compute $altFold_i$ for some index i.

2. `int altFold(int, int, int, int)`: This performs the *altFold* operation, which accepts four arguments, the last three characterize the AP, and the first argument is an index (ie., i) pointing to the first item in the AP from which *altFold* begins its computation.

You are required to complete the definitions of these functions when completing your program. Two sample runs are provided, with user input shown in bold.

```
Please enter the value of the first item of the AP: 1
What are the difference between two adjacent items? 1
How many items are there? 5
The final alternating sum of the computation is 12.
```

```
Please enter the value of the first item of the AP: 5
What are the difference between two adjacent items? -5
How many items are there? 3
The final alternating sum of the computation is 0.
```

[This page is intentionally left blank.]

CS1010E AY2017/18 Semester 2
Practical Exam 1 (PE1)

Advice on Writing Your Programs – Please Read!

- You are advised to spend some time thinking over the tasks to design your algorithms, instead of writing the programs right away.
- You are **not** allowed to use arrays, recursion or string functions from `string.h`. If in doubt, please check with the invigilators.
- You may write additional function(s) not mentioned in the question, if you think it necessary. But, you **must** make use of the functions provided to you.
- If you create a function, you must have a corresponding function prototype, and you must put the function definition *after* the `main` function.
- 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. If you are in the 12pm session, you cannot leave the examination venue even if you finish your exercises early.
- The rough marking scheme for both exercises is given below.

Rough Marking Scheme for Each Exercise

1. Style:
   - Top of a program: Are name, matriculation number, plab account-id, tutorial group and description filled accurately?
   - Ensure that there is a good and short description of every function apart from the main function.
   - Proper indentation and naming of variables.
   - Appropriate comments wherever necessary.
2. Design:
   - Correct definition and use of functions.
   - Presence of function prototypes.
   - Is the algorithm simple or unnecessarily complicated?
3. Correctness.
4. Deductions (not restricted to the following):
   - Program cannot be compiled: Severely penalized
   - Compiler issues warning with `-Wall`: Penalized
   - Use of arrays, recursion, built-in string functions from `string.h`: Very severely penalized
   - Use of global variables: Very severely penalized