NATIONAL UNIVERSITY OF SINGAPORE
SCHOOL OF COMPUTING

Practical Examination 1 (PE1) for Semester 1, AY2015/6
CS1010 Programming Methodology

19 September 2015 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 **8** pages. Each exercise constitutes 50%.
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 8) 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 **plab0** (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 and some test data files are already residing in your plab account. Please leave the 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 or we will **not** be able to find them!
10. **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.
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.
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: Sample Count [50 marks]

Problem Statement

A sample count is performed at the start of the counting process for each electoral division to get an early indication of the possible electoral outcome. This will be released once for each electoral division. Releasing the sample count helps to prevent speculation and misinformation from unofficial sources while the counting is underway, and before the formal election results are announced.

In this exercise, assume two teams A and B are competing in an electoral division with N polling stations, 1 ≤ N ≤ 5. After the end of the polling hours, the election officer will select **100** tickets from **each station**.

Write a program to compute the sample counts for teams A and B. Your program should read in the following inputs (all of type **int**) from the user:

- **numStation:** The total number of polling stations in the electoral division.
- **numVoterDivision:** The total number of voters in the electoral division.

For each polling station, you will read in the following data:

- **numVoteA**: The number of votes for team A.
- **numVoteB:** The number of votes for team B.
- **numVoterStation:** The total number of voters in the polling station.

Your program should compute the **sample count** (of type **float**) for each team weighted by the number of voters in each polling station. To compute the sample count of team A, we have:

$$sampleCountA = \sum_{polling\ station=1}^{numStation} \left(\frac{numVoteA}{100 - numInvalidVote} \times \frac{numVoterStation}{numVoterDivison}\right),$$

where **numInvalidVote** = 100 – numVoteA – numVoteB.

For example, in the first sample run, the numbers of invalid votes in the two stations are 100-80-15 = 5 and 100-65-25 = 10, respectively.

Therefore, the sample count for team A is $\left(\frac{80}{100-5} \times \frac{600}{1000}\right) + \left(\frac{65}{100-10} \times \frac{400}{1000}\right) = 79.42\%$

The sample count for team B can be similarly computed by replacing numVoteA with numVoteB in the equation.

Your program should output the **computed sample counts**. In addition, it should also print a **summary message** based on which team wins and how big is the win margin.

- If there is no difference between the sample counts, output "There is no winner in this election."
- If the difference is less than 5%, say 52% for team A and 48% for team B, output "Team A narrowly wins this election."
- If the difference is between 5% and 30% (both inclusive), say 42.50% for team A and 57.50% for team B, output "Team B wins by a significant margin."

- If the difference is more than 30%, say 79.42% for team A and 20.58% for team B, output "Team A wins by a landslide."

You may assume that the input is valid (*i.e.,* all integers are positive, the total number of voters in the stations is equal to the total number of voters in the division, and the total number of votes for the two teams in each station does not exceed 100).

Write on the skeleton file **election.c** given to you. You need to include one function:

- **`printSummary()`**

  This function prints the summary message based on the sample counts for the two teams. You are to decide the appropriate parameters and return type for this function.

You may define additional functions as needed. However, you are advised to implement the computation of sample counts in the main function instead of in a separate function. Check sample runs for input and output format and read the comments in the skeleton code for additional instructions.

In addition, due to the relatively large number of inputs for this exercise, you are advised to make use of input redirection to test your program with the given input files.

For example, to run your executable code (e.g., a.out) with an input file (e.g., election1.in), in your UNIX command prompt, enter the following command:

```
a.out < election1.in
```

Sample Runs

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

```
Enter number of voters in the division: 1000
Enter number of stations: 2
Enter number of voters in station 1: 600
Enter number of votes for Team A: 80
Enter number of votes for Team B: 15
Enter number of voters in station 2: 400
Enter number of votes for Team A: 65
Enter number of votes for Team B: 25
Sample count for Team A = 79.42%
Sample count for Team B = 20.58%
Team A wins by a landslide.
```

```
Enter number of voters in the division: 1000
Enter number of stations: 1
Enter number of voters in station 1: 1000
Enter number of votes for Team A: 50
Enter number of votes for Team B: 50
Sample count for Team A = 50.00%
Sample count for Team B = 50.00%
There is no winner in this election.
```

```
Enter number of voters in the division: 5000
Enter number of stations: 3
Enter number of voters in station 1: 1000
Enter number of votes for Team A: 60
Enter number of votes for Team B: 40
Enter number of voters in station 2: 2000
Enter number of votes for Team A: 40
Enter number of votes for Team B: 60
Enter number of voters in station 3: 2000
Enter number of votes for Team A: 60
Enter number of votes for Team B: 40
Sample count for Team A = 52.00%
Sample count for Team B = 48.00%
Team A narrowly wins this election.
```

```
Enter number of voters in the division: 2000
Enter number of stations: 2
Enter number of voters in station 1: 500
Enter number of votes for Team A: 20
Enter number of votes for Team B: 80
Enter number of voters in station 2: 1500
Enter number of votes for Team A: 50
Enter number of votes for Team B: 50
Sample count for Team A = 42.50%
Sample count for Team B = 57.50%
Team B wins by a significant margin.
```

This page is intentionally left blank.
You can use it for writing your algorithms or for any other purposes.

Exercise 2: Amicable numbers [50 marks]

Problem Statement

In number theory, a pair of **amicable numbers** consists of two **different** numbers such that **the sum of the factors** of each is **equal to the other number**.

For example, 220 and 284 is a pair of amicable numbers:

- The factors of 220 are 1, 2, 4, 5, 10, 11, 20, 22, 44, 55 and 110, of which the sum is 284.
- The factors of 284 are 1, 2, 4, 71 and 142, of which the sum is 220.

*(Note: As shown in the example, we consider only the **proper** factors, which do **not** include the number itself.)*

Given a range [lower, upper], we count a pair of amicable numbers as a *full pair* if both numbers in the pair are within the range. In contrast, we count a pair of amicable numbers as *a half pair* if only one of the numbers in the pair is within the range.

For example, if the range given is [100, 300], 220 and 284 is counted as a full pair because both numbers are within the range. In contrast, if the range given is [250, 300], the same pair is counted as a half pair, since only 284 is within the range.

In this exercise, you are to write a program to compute the number of full pairs and half pairs of amicable numbers in a given range.

For example, given the range [100, 300], your program should output that there is 1 full pair and 0 half pair of amicable numbers. As for the range [250, 300], your program should output that there is 0 full pair and 1 half pair of amicable numbers.

Your program should read in two integers, which represent the lower bound and upper bound of the range (both inclusive), compute the numbers of full pairs and half pairs of amicable numbers in the given range, and print these two numbers in the output messages.

You may assume that the input is valid (*i.e.*, the integers are all positive and the lower bounds are no bigger than the upper bounds).

Write on the skeleton file **amicable.c** given to you. You need to implement the following two functions:

- `void countPairs(int lower, int upper, int results[])`

  This function takes in two integers `lower` and `upper`. It stores the number of full pairs of amicable numbers in the given range `[lower, upper]` in `results[0]`, and the number of half pairs in `results[1]`.

- `int sumFactors(int number)`

  This function takes in one integer `number` and computes the sum of factors for `number`.

You may define additional functions as needed; however, **you are not allowed to change the main function.** Read the comments in the skeleton code for additional instructions.

Sample Runs

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

```
Enter range: 100 300
Number of full pairs: 1
Number of half pairs: 0
```

One pair of amicable numbers is counted: 220 and 284 (full).

```
Enter range: 250 300
Number of full pairs: 0
Number of half pairs: 1
```

One pair of amicable numbers is counted: 220 and 284 (half).

```
Enter range: 1 100
Number of full pairs: 0
Number of half pairs: 0
```

No integer within this range is part of a pair of amicable numbers.

```
Enter range: 200 1200
Number of full pairs: 1
Number of half pairs: 1
```

Two pairs of amicable numbers are counted: 220 and 284 (full), and 1184 and 1210 (half).

CS1010 AY2015/6 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.
- 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 for both exercises is given below.

Rough Marking Scheme For Each Exercise [50 marks]

1. 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?
2. Design: 10 marks
   - Are there correct definition and use of functions?
   - Are function prototypes present?
   - Is the right construct used?
   - Is algorithm not unnecessarily complicated?
3. Correctness: 30 marks
4. 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 ---