**NATIONAL UNIVERSITY OF SINGAPORE**
**SCHOOL OF COMPUTING**

Practical Examination 1 (PE1) for Semester 2, AY2015/6
# CS1010 Programming Methodology

17 February 2016 Time Allowed: 2 hours

## INSTRUCTION TO CANDIDATES

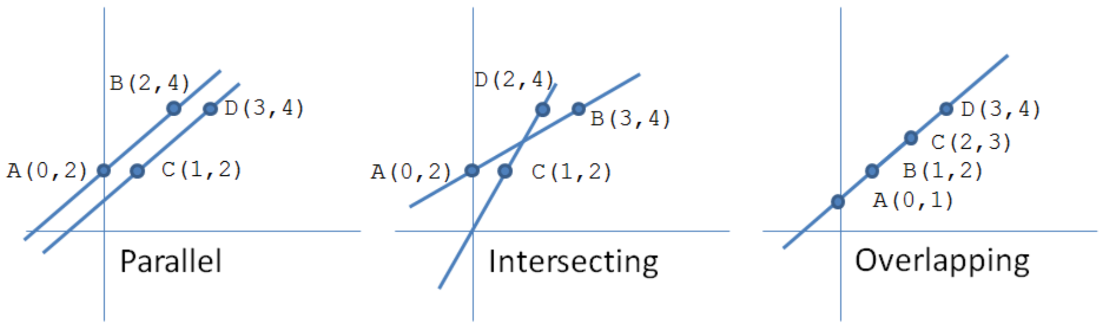
1. You are only allowed to read this cover page and the last page. Do **not** read the questions until you are told to do so.
2. This paper consists of **2** exercises on **6** pages. Each exercise constitutes 50%.
3. This is an open-book exam. You may bring in any printed material, but **not** electronic 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 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 an 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: Lines [50 marks]

### Problem Statement

In 2D geometry, given 4 distinct points A, B, C and D on a plane, the two lines, AB and CD, can be parallel (no intersection), intersecting (exactly one intersection) or overlapping (infinite intersections).



One way to determine which case it is (without calculating the number of intersections) is to make use of the **slopes** of AB, CD and AC.

The slope ***k*** of a line can be calculated using the following formula:

$$k = \frac{y_1 - y_2}{x_1 - x_2}$$

where ($x_1$, $y_1$) and ($x_2$, $y_2$) are the coordinates of two distinct points on the line.

Based on the slopes of the lines, AB and CD are:

- **Parallel** if the slopes of AB and CD are the same but different from that of AC.
- **Intersecting** if the slopes of AB and CD are different.
- **Overlapping** if the slopes of AB, CD and AC are the same.

Write a program to 1) read in the coordinates of the 4 points (all of **int** type), 2) determine whether the two lines AB and CD are parallel, intersecting or overlapping, and 3) print a message accordingly.

Write on the skeleton file **line.c** given to you. You need to include three functions:

- **`determineType()`**: This function takes in the coordinates of 4 points and determines whether the two lines are parallel, intersecting or overlapping.
- **`computeK()`**: This function takes in the coordinates of two distinct points on a line and computes the slope of the line.
- **`printMessage()`**: This function prints a message based on whether the two lines are parallel, intersecting or overlapping.

You are to decide the appropriate parameters and return types for these functions. You may define additional functions as needed.

You need to define constants for the three different types (e.g., 1 for parallel, 2 for intersecting, and 3 for overlapping) and use them in the program.

You may assume that the inputs are valid (i.e., all the coordinates are integers).

You do **not** need to handle the division by zero problem in the computation of slope (i.e., you may assume that $x_1$ and $x_2$ are always different).

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., line1.in), in your UNIX command prompt, enter the following command:

```
a.out < line1.in
```

## Sample Runs

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

```
Enter the coordinates of A: 0 2
Enter the coordinates of B: 2 4
Enter the coordinates of C: 1 2
Enter the coordinates of D: 3 4
The two lines are parallel.
```

```
Enter the coordinates of A: 0 2
Enter the coordinates of B: 3 4
Enter the coordinates of C: 1 2
Enter the coordinates of D: 2 4
The two lines are intersecting.
```

```
Enter the coordinates of A: 0 1
Enter the coordinates of B: 1 2
Enter the coordinates of C: 2 3
Enter the coordinates of D: 3 4
The two lines are overlapping.
```

## Exercise 2: Skip Counting [50 marks]

### Problem Statement

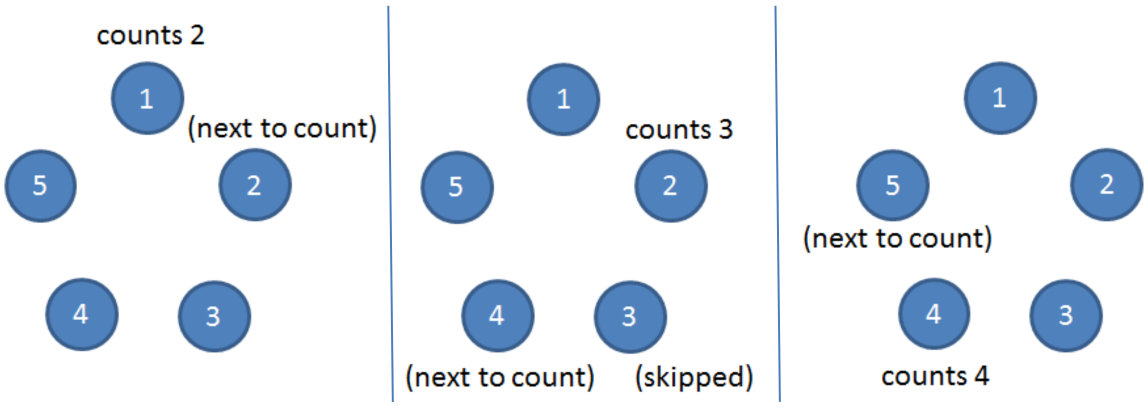
It is the eve of Chinese New Year and the Tan family is having their reunion dinner together as usual. This year, Mr. Tan is planning to give a big red packet to one of his 5 children but he has problem in deciding which child should get it. Therefore, he asks the children to sit around the table and play a game called skip counting.

The rules of the game are as follows:

- The five seats around the table are numbered from 1 to 5 in clockwise direction.
- A starting position, a lower bound and an upper bound are chosen by Mr. Tan.
- The children count the integers from the lower bound to the upper bound (both inclusive) one by one starting from the child at the starting position in clockwise direction. (Since the children are seated around the table, after position 5 comes position 1.)
- If the number being counted contains *n* odd digits, the next *n* positions are skipped. (Do take note that any number, regardless of whether it is odd or even, may contain odd digits. For example, 36 contains 1 odd digit, which is 3, while 35 contains 2 odd digits, which are 3 and 5.
- After the counting is done, the next child to count is the winner of the game.

For example, if the starting position is 1 and the lower bound and upper bound are 2 and 4, respectively. The game proceeds as follows:

- The child at position 1 starts by counting 2. Since 2 does not contain any odd digits, the next child to count is at position 2.
- The child at position 2 counts 3. Since 3 contains *1* odd digit, the next *1* position (i.e., position 3) is skipped and the next child to count is at position 4.
- The child at position 4 counts 4. Since 4 does not contain any odd digit, the next child to count is at position 5.
- At this point, there are no more numbers to count and the child at position 5 gets the red packet.



You are to write a program to play this game and decide which child gets the red packet given a starting position, a lower bound and an upper bound.

Write on the skeleton file **skip.c** given to you. You need to include two functions:

- **`int skipCount(int startPos, int lower, int upper)`**

  This function takes in a starting position `startPos`, a lower bound `lower` and an upper bound `upper`. It returns the ending position (i.e., the next position to count after the skip counting is done).

- **`int countOddDigits(int num)`**

  This function takes in an integer and counts the number of odd digits in it.

You may define additional functions as needed.

You may assume that the inputs are valid: 1) the starting position is an integer between 1 and 5 (both inclusive), 2) the lower bound and the upper bound are both positive integers, and 3) the lower bound is not greater than the upper bound.

## Sample Runs

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

```
Enter starting position: 1
Enter lower bound and upper bound: 2 2
The ending position is 2.
```

```
Enter starting position: 1
Enter lower bound and upper bound: 3 3
The ending position is 3.
```

```
Enter starting position: 5
Enter lower bound and upper bound: 1 2
The ending position is 3.
```

## CS1010 AY2015/6 Semester 2
## 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]**

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 ---**