Recruitment in India 2014 Programming Examination

- Choose examination 1 or 2 :

Write program for examination 1 **OR** 2 attached, in accordance with the following rules:
*Of course, you **CAN** write programs for **BOTH** of two examinations.

- Faster code is better :

Implementation with **faster execution** will lead to higher scores.

- Java or C++ for examination 1:
Java for examination 2:

You **CAN** choose Java **OR** C++ to implement the examination 1.
But you can use **only Java** to implement the examination 2.

(Environment)

-Java

*Compile and execute in Java6 or Java7 environment.
*You **CAN USE** the classes of Java Platform Standard Edition 6/7 API Specification but **CAN NOT USE** the ones from other libraries.

-C++

*Compiler: GCC 4.8.2 or later (The latest version of g++ by apt-get).
*You **CAN** use only C++ Standard Library, but **CAN NOT** use other libraries.

- Use attached code skeleton :

*Do **NOT** change package names written in the attached document.
*Any classes or functions in the skeleton **MUST NOT BE** changed.
(name, variable number, parameters and return value)
*You **CAN** add new classes and/or functions if you need.

- English only :

Use English or derivatives of English for all class names, identifiers, method names, comments, etc.

- Dead line : In 5 days from the presentation at your university.

(Ex. Presentation at 2nd NOV → Deadline : 7th NOV 23:59)

*Show your talent in your source code :

If you pass the examination above, we will interview you including live coding in front of our engineer with your source code for the examination above.

We are looking forward to seeing your creativity and originality in coding!!

<Caution!!>

Do not copy the source code from the other students or from Web sites.
Do not show your source code to the other students or on Web site.
If we find the same source code with the others' , we will not accept it.

See reserve for how to submit your source codes.
Submit your program to the following email address in the indicated format.

Please submit the assignment to the "placement cell".

Attachment: zip format compressed file (Your source codes and directories)
*We will NOT accept *.rar or other compressed format file.

How to upload your source code :

1, Career Page "Apply now" → Choose R&D Engineer's "Show details"→ Click "Apply" → Click "Accept"

2, Input your information according to direction of apply page, then you will find the upload page of the Programming Examination.

*Please be careful to ***choose your university*** correctly.

*When you upload your program, please select which Exam you are going to submit and which language you have used.

(e.g.1) In case of choosing Examination1 in C++

Exam1_Java	×
Exam1_C++	○
Exam2_Java	×

(e.g.2) In case of choosing Examination2 in Java

Exam1_Java	○
Exam1_C++	×
Exam2_Java	○

(e.g.3) In case of writing both examinations, examination1 in C++ examination2 in Java

Exam1_Java	×
Exam1_C++	○
Exam2_Java	○

(e.g.4) In case of writing both examinations, examination1 in Java examination2 in Java

Exam1_Java	○
Exam1_C++	×
Exam2_Java	○

(e.g.5) In case of choosing Examination1 in Java

Exam1_Java	○
Exam1_C++	×
Exam2_Java	×

<Caution!!>

***Submission is allowed only once.**

- Test your program(s) by yourself carefully in various cases before you submit them.

*__**Comment out the code for comments output and debug**__ when you submit source code.

Examination 1

***You can choose Java or C++**

– "Orienteering" –

Abstract

We are planning an orienteering game.
The aim of this game is to arrive at the goal (G) from the start (S) with the shortest distance.
However, the players have to pass all the checkpoints (@) on the map.

An orienteering map is to be given in the following format.

```
########
#@....G#
##.##@##
#..@..S#
#@.....#
########
```

In this problem, an orienteering map is to be given.
Calculate the minimum distance from the start to the goal with passing all the checkpoints.

Specification

* A map consists of 5 characters as following.
  You can assume that the map does not contain any invalid characters and
  the map has exactly one start symbol 'S' and exactly one goal symbol 'G'.

  * 'S' means the orienteering start.
  * 'G' means the orienteering goal.
  * '@' means an orienteering checkpoint.
  * '.' means an opened-block that players can pass.
  * '#' means a closed-block that players cannot pass.

* It is allowed to move only by one step vertically or horizontally (up, down, left, or right) to the next block.
  Other types of movements, such as moving diagonally (left up, right up, left down and right down) and skipping one or more blocks, are NOT permitted.
* You MUST NOT get out of the map.
* Distance is to be defined as the number of movements to the different blocks.
* You CAN pass opened-blocks, checkpoints, the start, and the goal more than once if necessary.
* You can assume that parameters satisfy following conditions.

  * 1 <= width <= 100
  * 1 <= height <= 100
  * The maximum number of checkpoints is 18.

* Return -1 if given arguments do not satisfy specifications, or players cannot arrive at the goal from the start by passing all the checkpoints.

Input

The input is to be given in the following format, from the standard input.

```
W H
Row1
Row2
...
RowH
```

The first row is to describe the width and the height of the orienteering map, sectioned by a space.
From the second row, map information is to be given for exactly the same number of rows as the height.
Each row contains exactly the same number of letters as the width.
See “Specification” for details of the symbols contained in the map.

Output

Output into the standard output, and put a return.

****DO NOT forget to delete debug outputs before submission** .**

Implementation

<Java>

Write the "jp.co.worksap.global.Orienteering" class, which has executable "main" method.

If other fields, methods, and/or classes are necessary, you can add them to the skeleton, or implement them on other files.

<Skeleton Code of Java>

```
package jp.co.worksap.global;

public class Orienteering {
    public static void main(String[] args) {
        // TODO: Implement your program
    }
}
```

<C++>

Write the "main" function of the "Orienteering" class, included in given skeleton.

If some C++ standard libraries and members are necessary, add them to the skeleton.
You can implement them on other files, but describe them not to cause compile errors with given compile option.

<Skeleton Code of C++>

```
class Orienteering {
public:
    void main();
};

void Orienteering::main() {
  // TODO: Implement this function
}

int main(int argc, char* argv[]) {
  Orienteering o;
  o.main();
  return 0;
}
```

<Compile Options>

The codes you have submitted are to be compiled by the following command.

```
g++ -std=c++11 -O2 -o a.out orienteering.cpp
```

Evaluation Criteria

This problem is to be evaluated by following criteria.

*** Being able to solve the problem correctly.**
*** Being able to get the answer fast.**

Examples

<Notes>

In these examples, the origin of coordinates is left-upper corner, and is described as (0, 0).
The direction of positive X-Axis is to right and positive Y-Axis is to down.

Example1

<Input>

```
5 4
#####
#...#
#S#G#
#####
```

<Output>

```
4
```

<Java Test Example>

```
# current directory has your build.
java -cp . jp.co.worksap.global.Orienteering < example1.txt
```

<C++ Test Example>

```
# current directory has your source code.
g++ -std=c++11 -O2 -o a.out orienteering.cpp
./a.out < example1.txt
```

Result path (x, y): (1, 2) => (1, 1) => (2, 1) => (3, 1) => (3, 2).
The result distance is 4, because the player moves 4 times.

Example2

<Input>

```
5 5
#####
#.@@#
#S###
#..G#
#####
```

<Output>

```
9
```

Result path (x, y): (1, 2) => (1, 1) => (2, 1) => (3, 1) => (2, 1) => (1, 1) => (1, 2) => (1, 3) => (2, 3) => (3, 3).

The result distance is 9, because the player moves 9 times.
Start (1, 2), opened-block (1, 1), and checkpoint (2, 1) appear twice each on the result path, but it is correct because the problem specifications allow to pass the same block for multiple times.

Example3

<Input>

```
5 5
#####
#S..#
##G##
#..@#
#####
```

<Output>

```
6
```

Result path (x, y): (1, 1) => (2, 1) => (2, 2) => (2, 3) => (3, 3) => (2, 3) => (2, 2).

The result distance is 6, because the player moves 6 times.
Opened-block (2, 3) and goal (2, 2) appear twice each on the result path, but it is correct because of the problem specifications.

Examination 2 (Java) : Implement This Skeleton Code

***Java only**

```
package jp.co.worksap.global;

import java.util.NoSuchElementException;

/**
 * The Queue class represents an immutable first-in-first-out (FIFO) queue of objects.
 * @param <E>
 */
public class ImmutableQueue<E> {
	/**
	 * requires default constructor.
	 */
	public ImmutableQueue() {
		// modify this constructor if necessary, but do not remove default constructor
	}

	// add other constructors if necessary

	/**
	 * Returns the queue that adds an item into the tail of this queue without modifying this queue.
	 * <pre>
	 * e.g.
	 * When this queue represents the queue (2, 1, 2, 2, 6) and we enqueue the value 4 into this queue,
	 * this method returns a new queue (2, 1, 2, 2, 6, 4)
	 * and this object still represents the queue (2, 1, 2, 2, 6) .
	 * </pre>
	 * If the element e is null, throws IllegalArgumentException.
	 * @param e
	 * @return
	 * @throws IllegalArgumentException
	 */
	public ImmutableQueue<E> enqueue(E e) {
		return null;
	}

	/**
	 * Returns the queue that removes the object at the head of this queue without modifying this queue.
	 * <pre>
	 * e.g.
	 * When this queue represents the queue (7, 1, 3, 3, 5, 1) ,
	 * this method returns a new queue (1, 3, 3, 5, 1)
	 * and this object still represents the queue (7, 1, 3, 3, 5, 1) .
	 * </pre>
	 * If this queue is empty, throws java.util.NoSuchElementException.
	 * @return
	 * @throws java.util.NoSuchElementException
	 */
	public ImmutableQueue<E> dequeue() {
		return null;
	}

	/**
	 * Looks at the object which is the head of this queue without removing it from the queue.
	 * <pre>
	 * e.g.
	 * When this queue represents the queue (7, 1, 3, 3, 5, 1),
	 * this method returns 7 and this object still represents the queue (7, 1, 3, 3, 5, 1)
	 * </pre>
	 * If the queue is empty, throws java.util.NoSuchElementException.
	 * @return
	 * @throws java.util.NoSuchElementException
	 */
	public E peek() {
		return null;
	}

	/**
	 * Returns the number of objects in this queue.
	 * @return
	 */
	public int size() {
		return -1;
	}
}
```