Question One [50 marks]

File names

- Use cycling.c if you are writing your program in C.
- Use cycling.cpp if you are writing your program in C++.
- Use Cycling.java if you are writing your program in Java.

Note that case matters.

Problem Description

The scenario:

- *N* signs have been set up along a straight road, each displaying a number (which can be positive or negative).
- A cyclist must choose a position on the road at which to start cycling and a position at which to stop.
- The cyclist starts with 0 points. As they cycle from the chosen start position to the end position, they must add the number on any sign that they pass (whether positive or negative) to their score.
- The challenge is to choose a start and end position such that the point score is maximised.

Write a program that, given the numbers from a series of *N* road signs (in the order in which the signs appear), calculates the maximum point score that can be achieved.

Example

Assume a road with *N* = 6 consecutive signs with the following values:

Sign	**S_1**	**S_2**	**S_3**	**S_4**	**S_5**	**S_6**
Value	1	-2	4	-1	5	-3

Choosing to start cycling before the first sign and to stop after the fifth sign, would give a score of 1 + (-2) + 4 + (-1) + 5 = 7 points.

This is not, however, the maximum achievable score. Starting before the third sign and stopping after the fifth would render 4 + (-1) + 5 = 8 points. Thus, the correct answer for this case would be 8.

Note that it is possible to start AND stop before the first sign, (or any other sign), giving a score of 0. While not true for this example, there may be situations in which that is the best choice.

Input and Output

Program input and output will make use of stdio streams (System.in and System.out in Java) i.e., not file I/O.

Input consists of a series of integer values, each on a separate line. The first value is *N*, the number of signs on the road, followed by the point values $P_1, \ldots, P_N$, for those signs. The values of the signs are given in the order they appear on the road.

Output consists of a single integer, *K*, the maximum point score that can be achieved, followed by a line break — in Java, for example, use System.out.println, not System.out.print. The automatic marker expects this precise form.

Sample Input:

```
6
1
-2
4
-1
5
-3
```

Sample output:

```
8
```

Constraints

$1 \leq N \leq 2{,}000$
$-1{,}000{,}000 \leq P_i \leq 1{,}000{,}000$ (for $1 \leq i \leq N$)

The maximum achievable point score will fit within a 32-bit signed integer type.

Scoring

Each test case that is answered correctly will score 10 points.

Question Two [50 marks]

File names

- Use path.c if you are writing your program in C.
- Use path.cpp if you are writing your program in C++.
- Use Path.java if you are writing your program in Java.

Note that case matters.

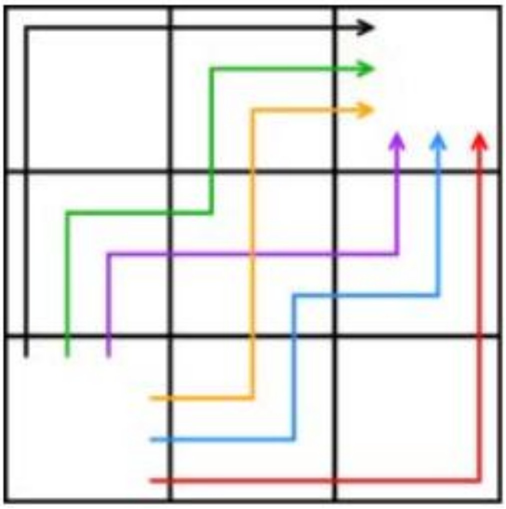
Problem Description

Write a program that computes the number of paths that a robot may take when navigating through a given terrain.

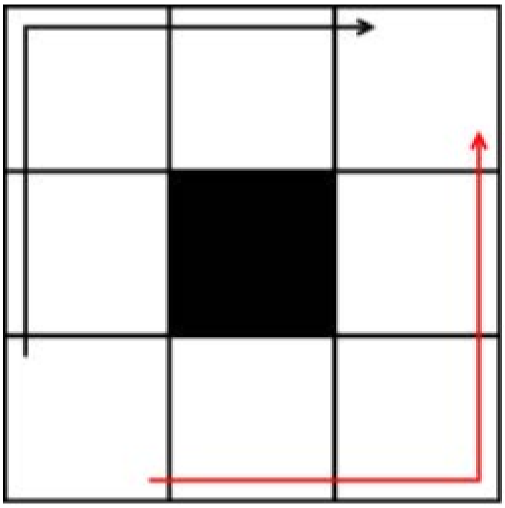
- A terrain is represented by an NxN grid of squares.
- The robot starts in the lower left square of grid, referred to as square (1, 1), and needs to move to the upper right square, referred to as (*N*, *N*).
- A terrain also contains *K* obstacles, each of which blocks a single square on the grid.
- The robot cannot move into a square containing an obstacle, so these reduce the number of possible paths through the grid.
- No two obstacles block the same square, and the starting and ending squares will never contain obstacles.
- The robot can only move a single square up or right with each step.

Example

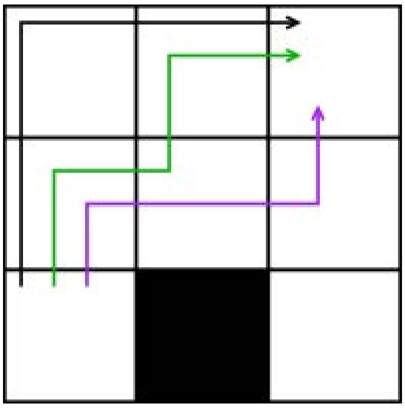
Given a *3×3* grid with no obstacles, there are a total of 6 paths from the bottom left square to the top right, shown below:



Given the same grid with one obstacle on square (2, 2) there are 2 paths:



Given the same grid with one obstacle on square (2, 1) there are 3 paths:



Input and Output

Program input and output will make use of stdio streams (System.in and System.out in Java) i.e., not file I/O.

Input consists of a series of lines, each containing up to two integer values.

- The first line of input contains a single integer, *N*, representing the size of the grid.
- The second line contains a single integer *K*, representing the number of obstacles on the grid.
- The following *K* lines of input each contain two integers separated by a space. Each of these lines represents the coordinates on the grid of one of the obstacles.

Output consists of a single integer *P*, the number of paths from the bottom left grid square to the top right grid square,followed by a line break — in Java, for example, use System.out.println, not System.out.print. The automatic marker expects this precise form.

Sample Input:

```
3
1
2 1
```

Sample output:

```
3
```

Constraints

$1 < N < 20$ $0 < K < N$

All obstacles will have coordinates within the dimensions of the grid.

The answer, P, will be bounded by $0 \le P \le 1{,}000{,}000{,}000$

Scoring

Each test case that is answered correctly will score 5 points.