CS2310 Computer Programming
2017/18 Semester B
Department of Computer Science
City University of Hong Kong
Assignment TWO
Due Date: 23 March 2018 23:59

All questions should be submitted to PASS.

This assignment consists of 14 pages, including this cover page. It contains 6 questions named as Q1 to Q6.

In Q1-Q6, DO NOT INCLUDE any library other than <iostream>, <iomanip>, and/or <cmath> in your solution. Each solution of Q1-Q6 using any function/facility in other libraries will receive ZERO mark.

Solution Submission Instruction:

- You are required to submit a complete C++ program in one file to PASS.
- You can test the program with VS2017, and submit as many times as you want before the deadline. We will use the latest version of your submission for grading.
- In each question above on PASS, only a small set of test cases are available to you for testing. We will use a more comprehensive set of test cases for grading your solution. Therefore, passing all test casts do NOT mean you will get 100% correct in our grading tests. Try to test your program thoroughly before final submission. Happy coding! ☺

Grading:

Solution correctness (90%) and style (10%).

- **Correctness**: We will use PASS to grade for correctness, basing on how many test cases are correct with your program. If your program cannot be compiled and tested on PASS, you will get **ZERO** mark.
- **Style**: We will check your source code for indentation, variable naming, comments, etc.

Plagiarism Check:

The course will use multiple code plagiarism detection software to check every solution the whole class submits. These software tools include the following two tools. Students are suggested to keep important immediate versions and paper draft of their solutions in case that students are asked to show evidences of self-effort and originality in developing their solutions of each question. All suspicious cases will be forwarded to the CS Departmental Disciplinary Committee for further actions.

- PASS: https://pass3.cs.cityu.edu.hk/index.jsp and
- MOSS: https://theory.stanford.edu/~aiken/moss .

Q1. Write a program which takes as input a sequence of **positive** integers, and outputs the maximum value, minimum value, and the value that occurs the most along with its occurrence. More specifically, the program works as follows:

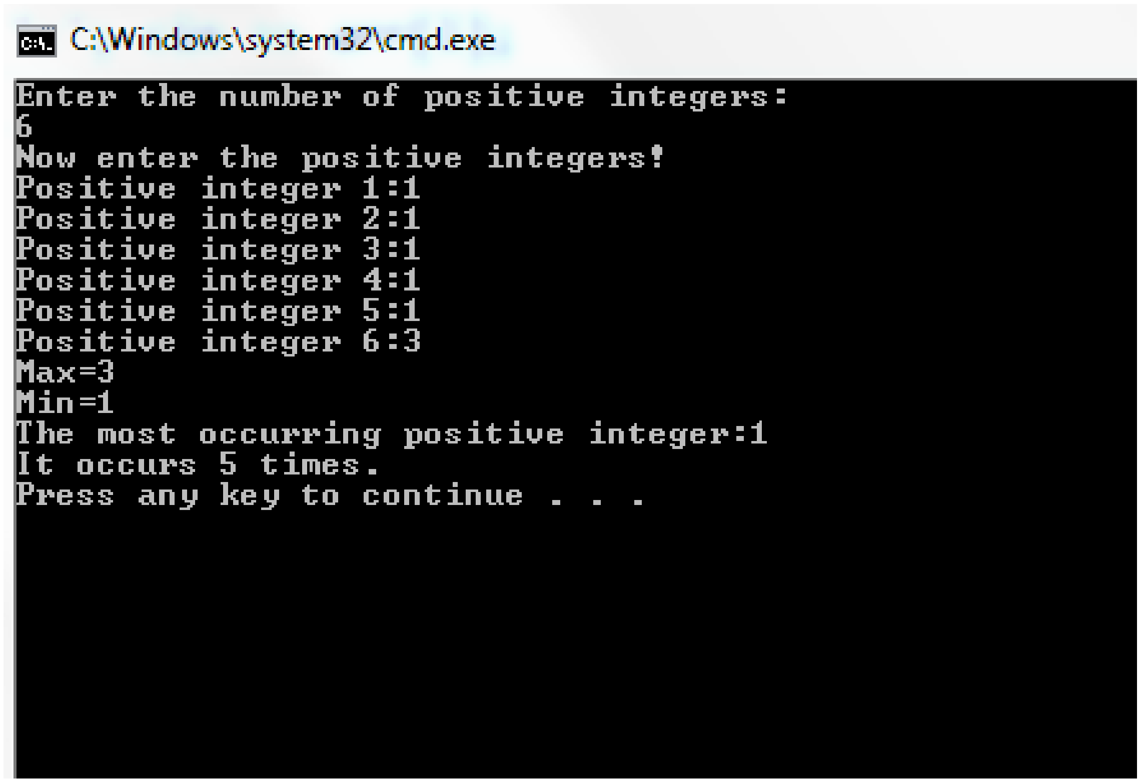
(1) Input the number of positive integers to be received;
(2) Input each positive integer;
(3) Output the maximum value, minimum value, and the value that occurs the most along with its occurrence.

Remarks:

1. You can safely assume the maximum number of input positive integers to be 20 (so in your program you can use an array of size 20 to store the positive integers), and the number of positive integers is > = 4.
2. You can safely assume that the inputs are valid and no validity check is required.
3. If there is no value occurring more than once, your program should print "There is no value occurring more than once." (see Running Sample 2)
4. If there is more than one positive integer value sharing the highest occurrence, your program just needs to print the smallest value. (see Running Sample 3)

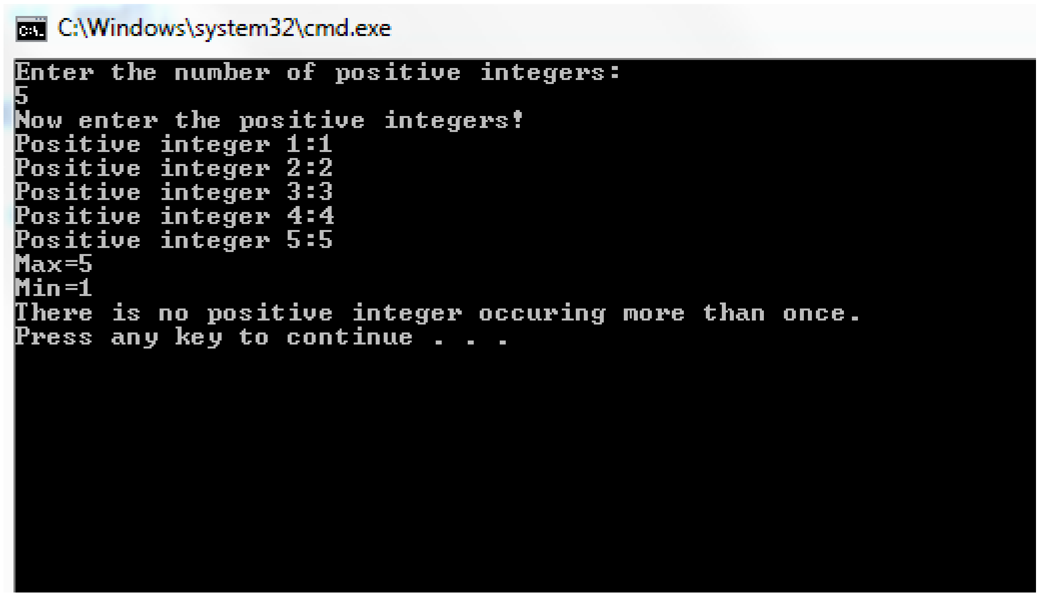
Running samples:

Sample1:



```
C:\Windows\system32\cmd.exe
Enter the number of positive integers:
6
Now enter the positive integers!
Positive integer 1:1
Positive integer 2:1
Positive integer 3:1
Positive integer 4:1
Positive integer 5:1
Positive integer 6:3
Max=3
Min=1
The most occurring positive integer:1
It occurs 5 times.
Press any key to continue . . .
```

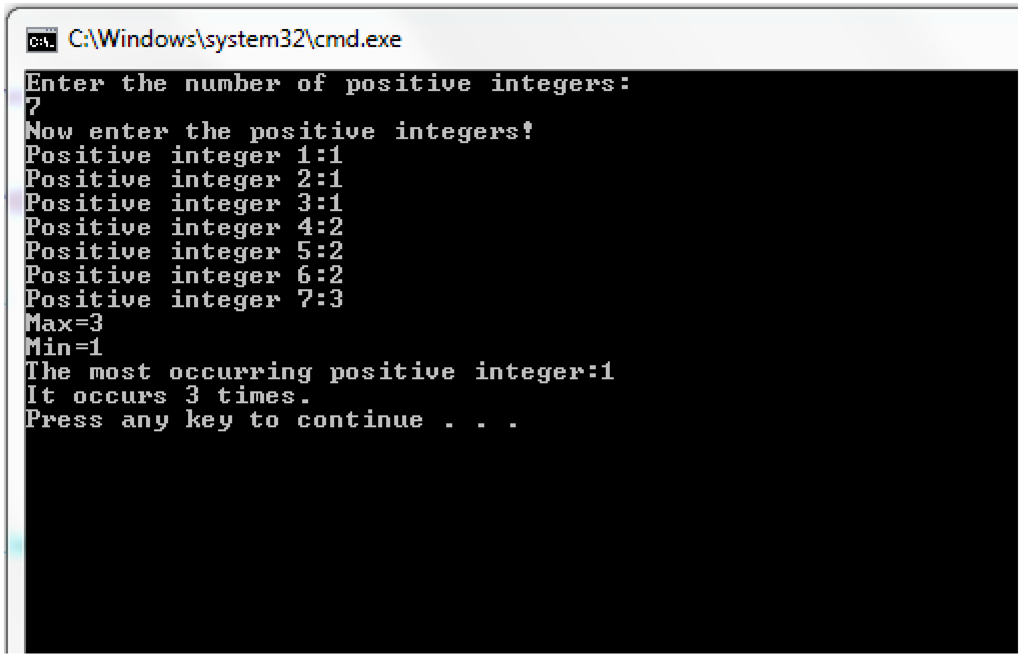
Sample 2:



C:\Windows\system32\cmd.exe

```
Enter the number of positive integers:
5
Now enter the positive integers!
Positive integer 1:1
Positive integer 2:2
Positive integer 3:3
Positive integer 4:4
Positive integer 5:5
Max=5
Min=1
There is no positive integer occuring more than once.
Press any key to continue . . .
```

Sample 3:

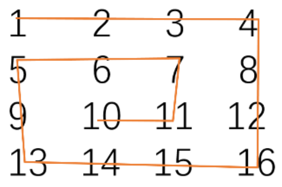


C:\Windows\system32\cmd.exe

```
Enter the number of positive integers:
7
Now enter the positive integers!
Positive integer 1:1
Positive integer 2:1
Positive integer 3:1
Positive integer 4:2
Positive integer 5:2
Positive integer 6:2
Positive integer 7:3
Max=3
Min=1
The most occurring positive integer:1
It occurs 3 times.
Press any key to continue . . .
```

Q2. Write a program which takes as input a square matrix and prints it in spiral form. An example is given below:

Input matrix:



```
1    2    3    4
5    6    7    8
9    10   11   12
13   14   15   16
```

Printed in spiral form:

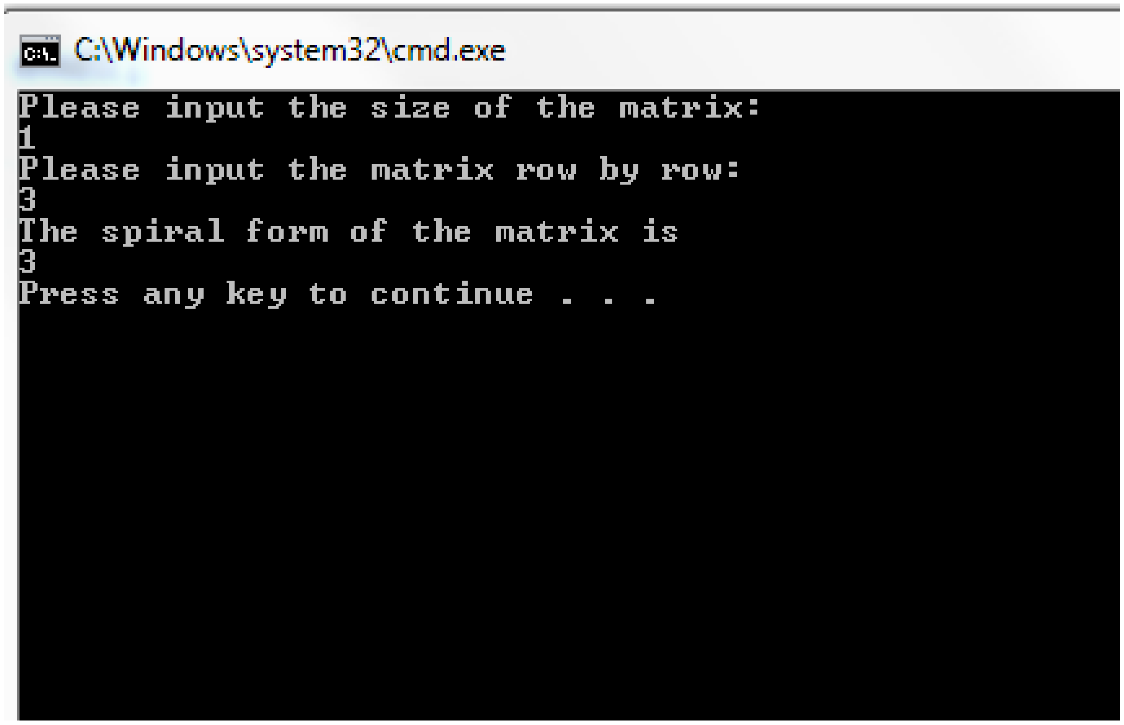
1 2 3 4 8 12 16 15 14 13 9 5 6 7 11 10

Remarks:

1. You can assume that the maximum size of the input matrix is 10 x 10.
2. No validity check is required for the value indicating the size of the matrix.

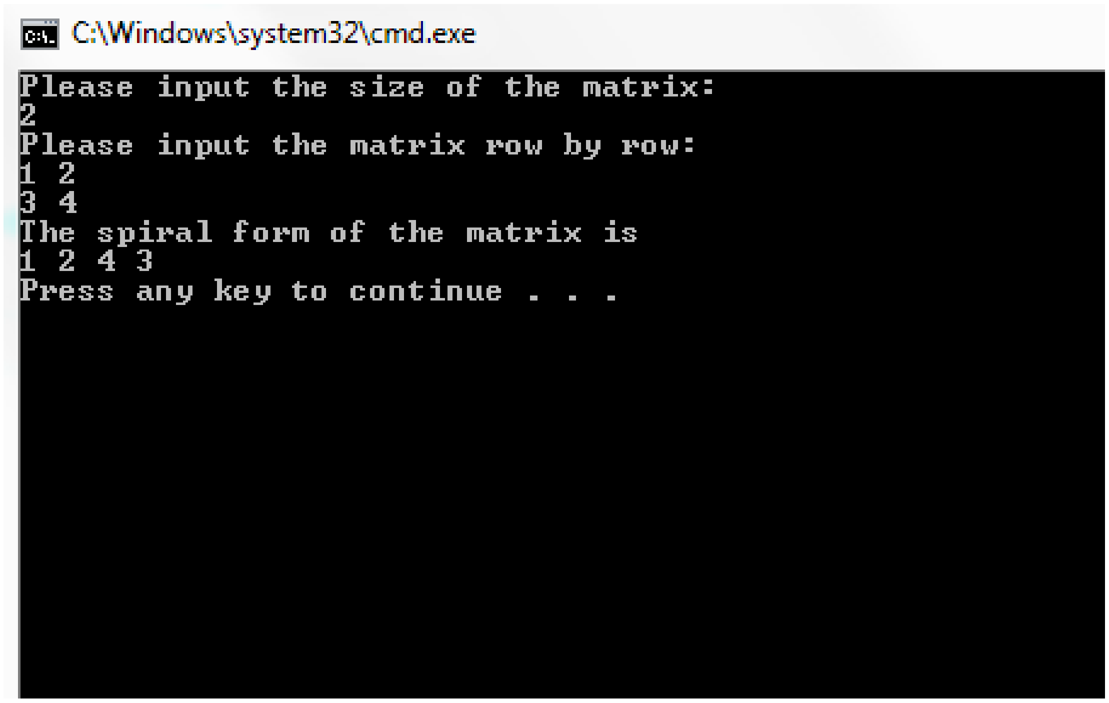
Running samples:

Sample 1:



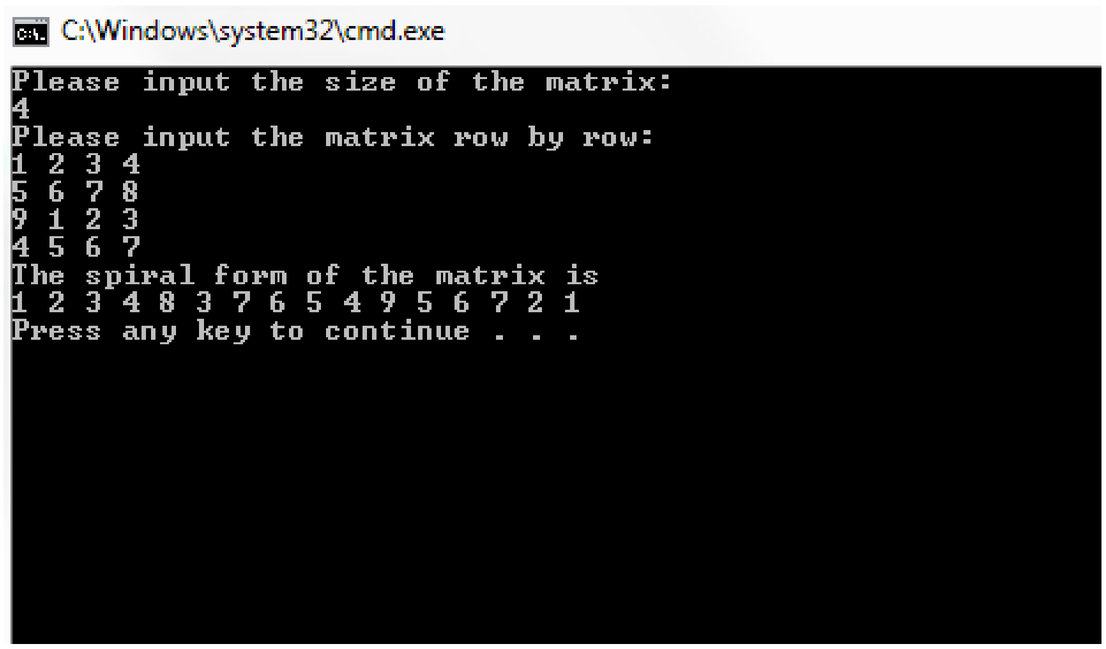
```
Please input the size of the matrix:
1
Please input the matrix row by row:
3
The spiral form of the matrix is
3
Press any key to continue . . .
```

Sample 2:



```
C:\Windows\system32\cmd.exe
Please input the size of the matrix:
2
Please input the matrix row by row:
1 2
3 4
The spiral form of the matrix is
1 2 4 3
Press any key to continue . . .
```

Sample 3:



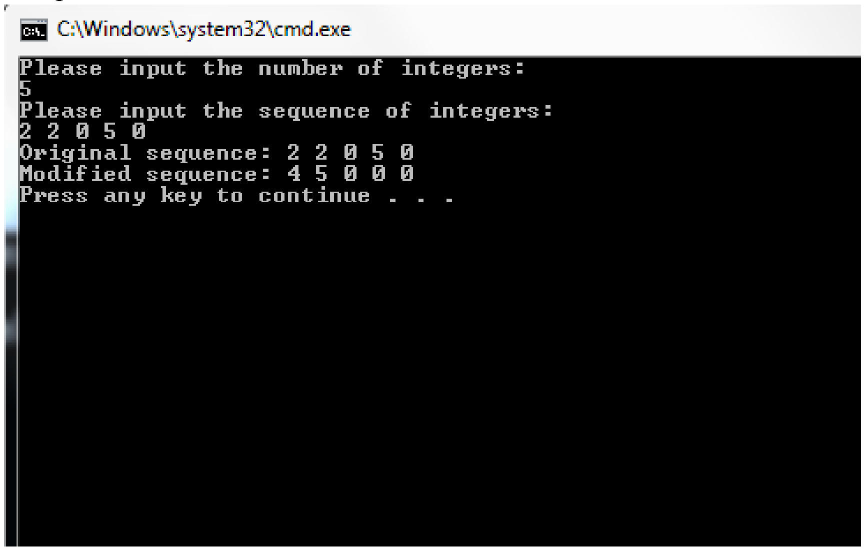
```
C:\Windows\system32\cmd.exe
Please input the size of the matrix:
4
Please input the matrix row by row:
1 2 3 4
5 6 7 8
9 1 2 3
4 5 6 7
The spiral form of the matrix is
1 2 3 4 8 3 7 6 5 4 9 5 6 7 2 1
Press any key to continue . . .
```

Q3. Write a program which receives as input a sequence of integers which will at least contain one Zero. Assume Zero is the invalid number and all other numbers are valid. The program should be able to convert the sequence following these rules:
(1) If next **valid** number is same as current number, double the current number and replace the next valid number with Zero. Note that each checking is done over an updated sequence of numbers from the previous checking and (possible) number doubling;
(2) After all the modification has been done, rearrange the updated sequence such that all 0's are shifted to the end.

For example, given the sequence of numbers {2, 2, 0, 5, 0}, the final converted sequence of numbers is {4, 5, 0, 0, 0}. The conversion procedure is as follows:

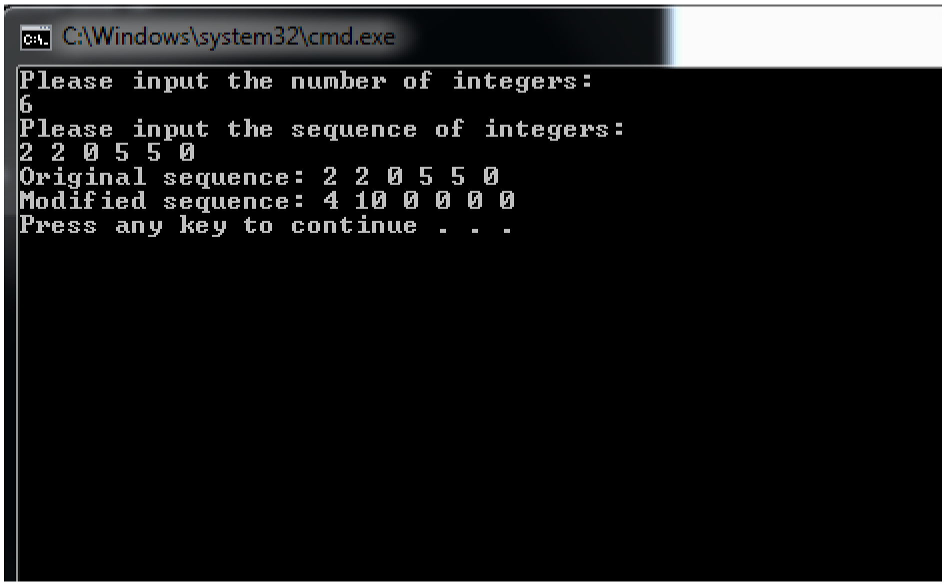
(1) Starting from the leftmost '2', as the next number is valid and is the same as '2', so the current '2' is doubled and becomes '4'. Meanwhile, the next valid number '2' is set to '0'. After this checking, the updated sequence of numbers is {4, 0, 0, 5, 0}.
(2) Such kind of checking is continued until the end of the sequence. As there are no more pairs of neighboring identical valid numbers, the updated sequence of numbers will be {4, 0, 0, 5, 0} when the checking procedure is finished.
(3) Now the program should push all '0s' to the end, so the final converted sequence is {4, 5, 0, 0, 0}.
Remarks:
1. You can assume the maximum number of input integers to be 20 (so in your program you can use an array of size 20 to store the input integers) and that the number of input integers is >=4.
2. No validity check is required for the inputs.

Running samples:

Sample 1:


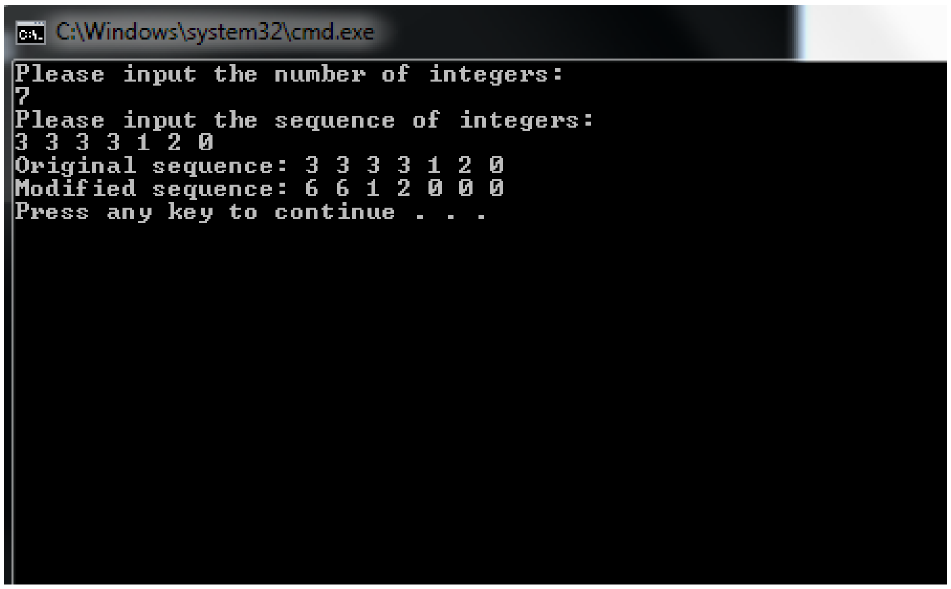
```
Please input the number of integers:
5
Please input the sequence of integers:
2 2 0 5 0
Original sequence: 2 2 0 5 0
Modified sequence: 4 5 0 0 0
Press any key to continue . . .
```

Sample 2:



```
Please input the number of integers:
6
Please input the sequence of integers:
2 2 0 5 5 0
Original sequence: 2 2 0 5 5 0
Modified sequence: 4 10 0 0 0 0
Press any key to continue . . .
```

Sample 3:



```
Please input the number of integers:
7
Please input the sequence of integers:
3 3 3 3 1 2 0
Original sequence: 3 3 3 3 1 2 0
Modified sequence: 6 6 1 2 0 0 0
Press any key to continue . . .
```

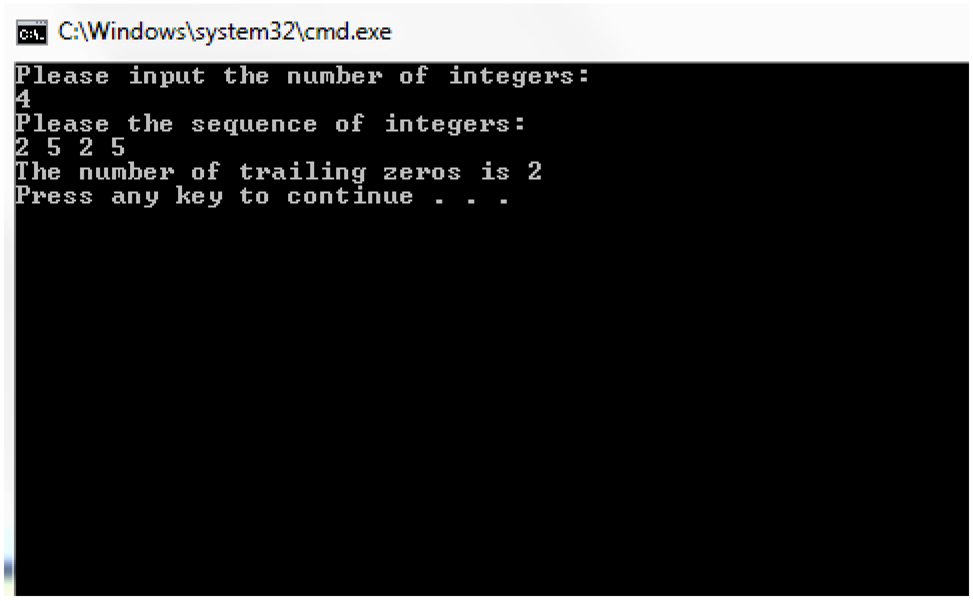
Q4. Write a program which works as follows. First, it receives a sequence of integers, each of which may be **positive, negative, or zero**. Then, the product of all integers is computed. The program should output the number of trailing 0s in the product result. For example, if the input sequence is {2, 3, 5}, the product is 30, so the number of trailing 0s is 1; if the input sequence is {-2, 3, 10, 5}, the product is -300, so the number of trailing 0s is 2. If the input sequence contains 0(s), the program output the number of trailing zeros as 1 (see running sample 3).

Hints: A simple approach to achieving the functionality of this program is to simply multiply all integers first and then count trailing 0s in product. However, this solution may cause integer overflow. So, a better solution is based on the fact that 0s are formed by a combination of 2 and 5. Hence the number of zeros will depend on the number of pairs of 2 and 5 that can be formed. For example, 8 * 3 * 5 * 23 * 17 * 25 * 4 -> $2^3 * 3^1 * 5^1 * 23^1 * 17^1 * 5^2 * 2^2$. In this example there are 5 twos and 3 fives. Hence, we shall be able to form only 3 pairs of (2*5). Hence will be 3 Zeros in the product.

Remarks:
1. You can assume the maximum number of input integers is 20.
2. You can safely assume that the number of input integers is >=4.

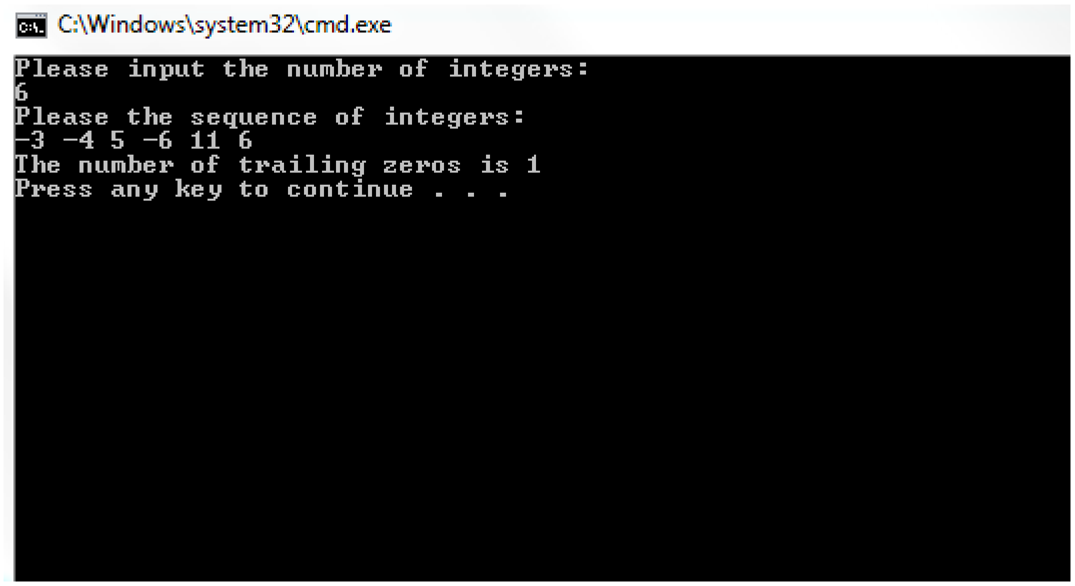
Running samples:

Sample 1:



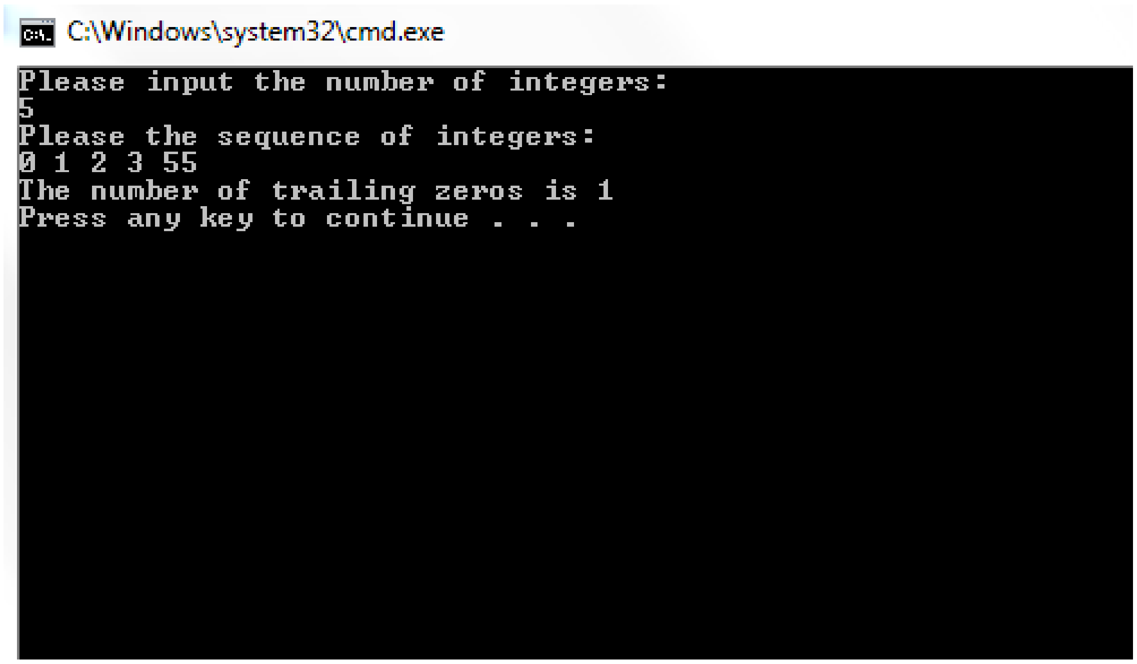
```
C:\Windows\system32\cmd.exe
Please input the number of integers:
4
Please the sequence of integers:
2 5 2 5
The number of trailing zeros is 2
Press any key to continue . . .
```

Sample 2:



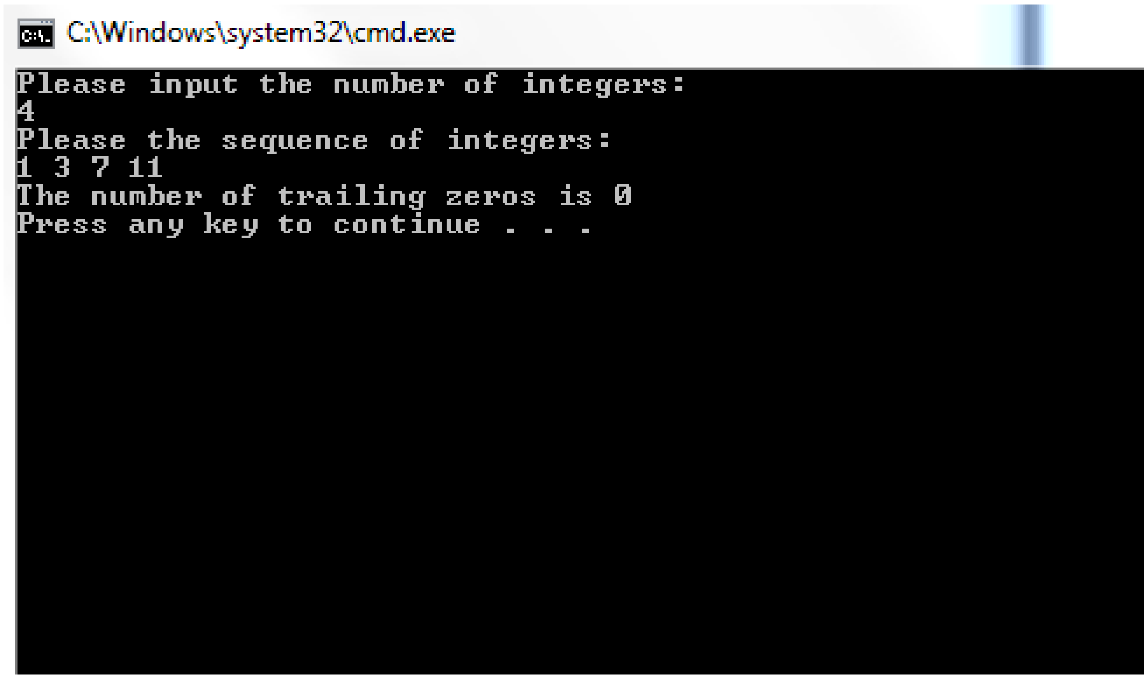
```
Please input the number of integers:
6
Please the sequence of integers:
-3 -4 5 -6 11 6
The number of trailing zeros is 1
Press any key to continue . . .
```

Sample 3:



```
Please input the number of integers:
5
Please the sequence of integers:
0 1 2 3 55
The number of trailing zeros is 1
Press any key to continue . . .
```

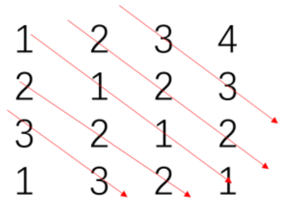
Sample 4:



C:\Windows\system32\cmd.exe

```
Please input the number of integers:
4
Please the sequence of integers:
1 3 7 11
The number of trailing zeros is 0
Press any key to continue . . .
```

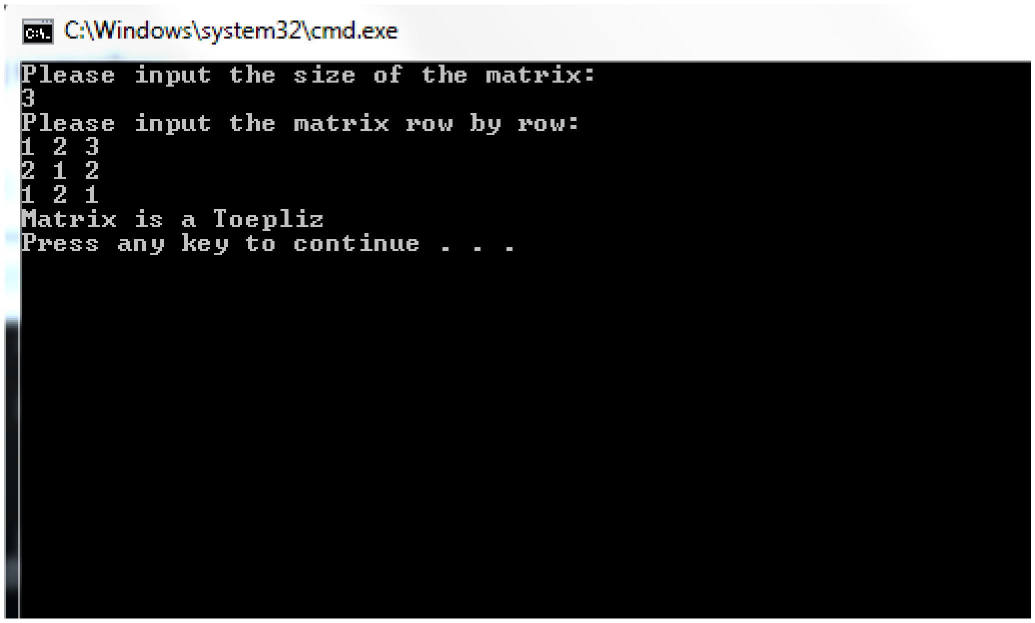
Q5. Given a square matrix, check if it's a Toeplitz matrix or not. A Toeplitz (or diagonal-constant) matrix is a matrix in which **each** descending diagonal from left to right is constant, i.e., all elements on a diagonal are same. An example of Toeplitz matrix is shown below:



Remarks:
1. You can safely assume that the minimum size of the matrix is 3 x 3 and the maximum size of the matrix is 10 x 10.
2. No validity check is required for the inputs.

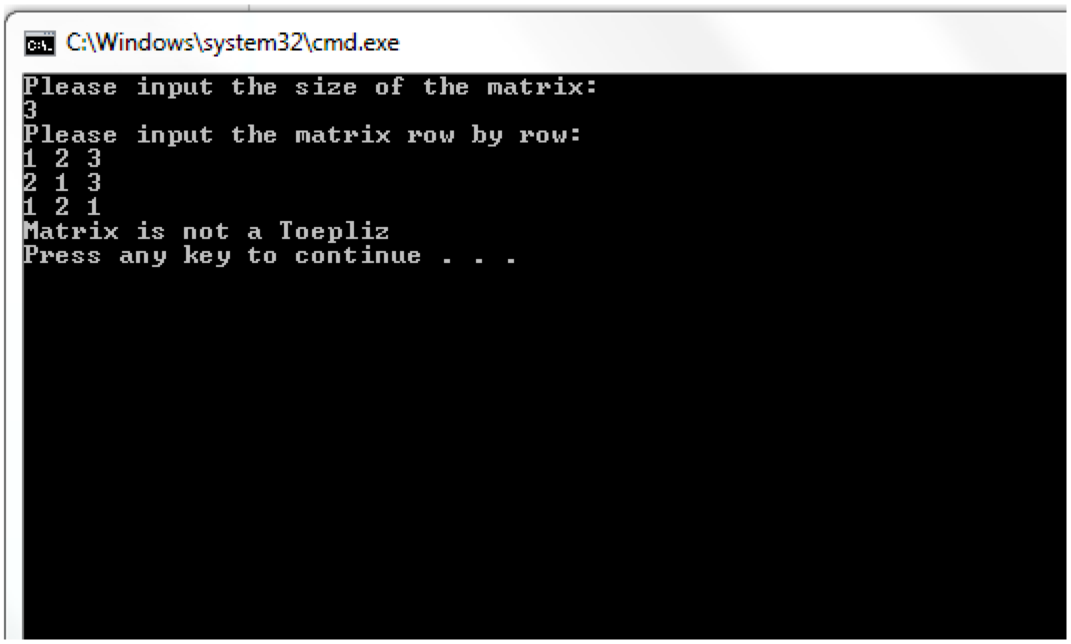
Running samples:

Sample 1:



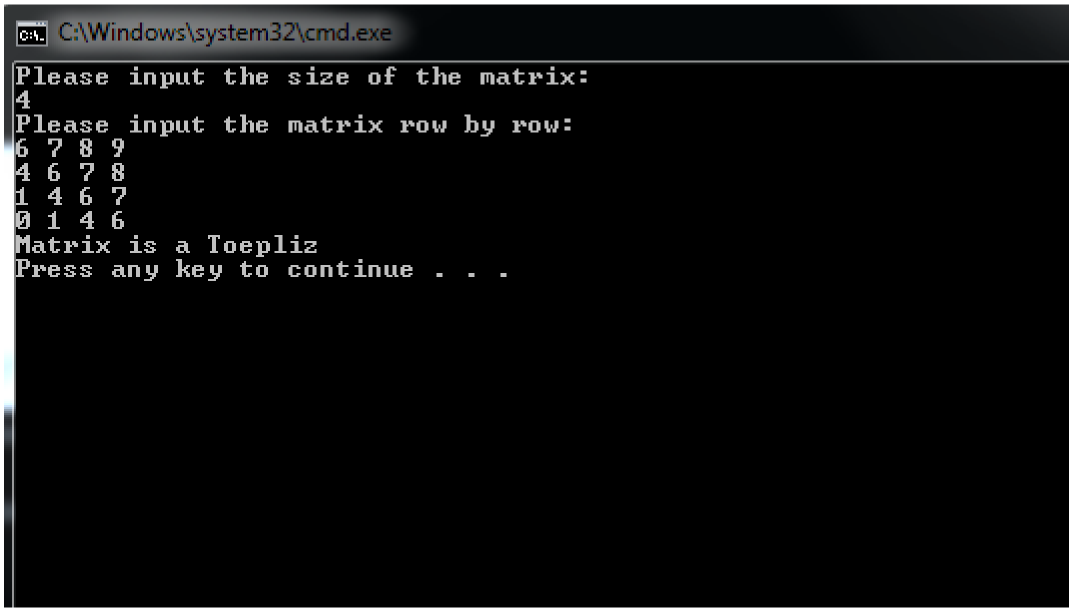
```
Please input the size of the matrix:
3
Please input the matrix row by row:
1 2 3
2 1 2
1 2 1
Matrix is a Toepliz
Press any key to continue . . .
```

Sample 2:



```
C:\Windows\system32\cmd.exe
Please input the size of the matrix:
3
Please input the matrix row by row:
1 2 3
2 1 3
1 2 1
Matrix is not a Toepliz
Press any key to continue . . .
```

Sample 3:



```
C:\Windows\system32\cmd.exe
Please input the size of the matrix:
4
Please input the matrix row by row:
6 7 8 9
4 6 7 8
1 4 6 7
0 1 4 6
Matrix is a Toepliz
Press any key to continue . . .
```

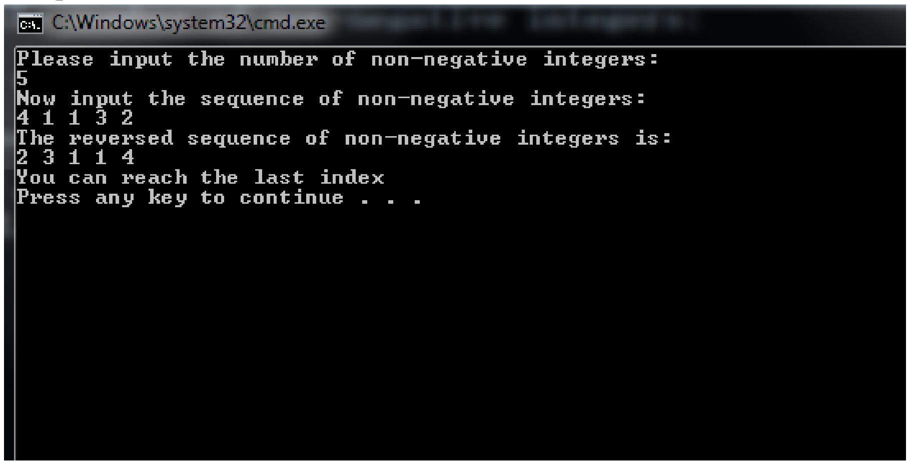
Q6. Write a program which works as follows. Given a sequence of **non-negative** integers, the program should reverse the order of integers. Based on this reversed sequence, the following game is played: Suppose each element in the reversed sequence represents the **maximum** number of steps that can be made forward from that element. You are initially placed on the first index of the sequence. Determine if you can reach the last index. For example, if the reversed sequence is {2, 3, 1, 1, 4}, you can reach the last index; if the reversed sequence is {1, 2, 3, 4}, you can reach the last index; if the reversed sequence is {1, 0, 3, 4}, you cannot reach the last index.

Remarks:

1. You can safely assume that the minimum number of non-negative integers is 4 and the maximum number of non-negative numbers is 10.
2. You can assume the inputs are valid and no validity check is required.

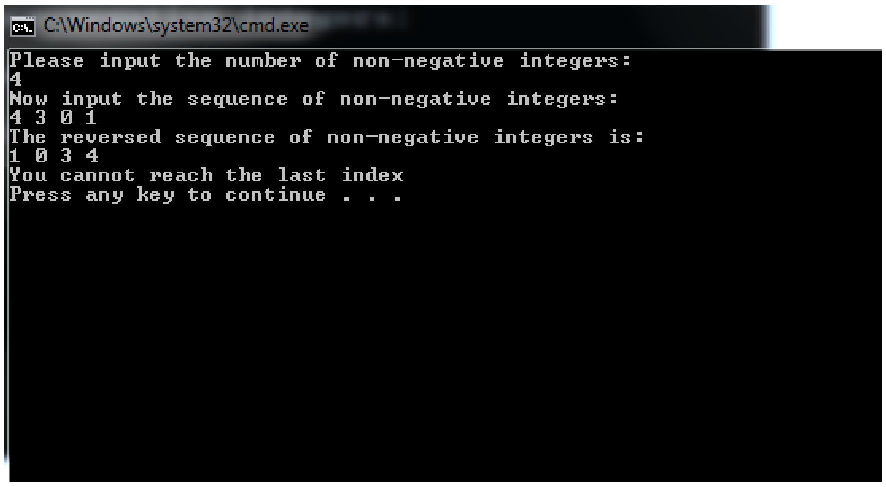
Running samples:

Sample 1:



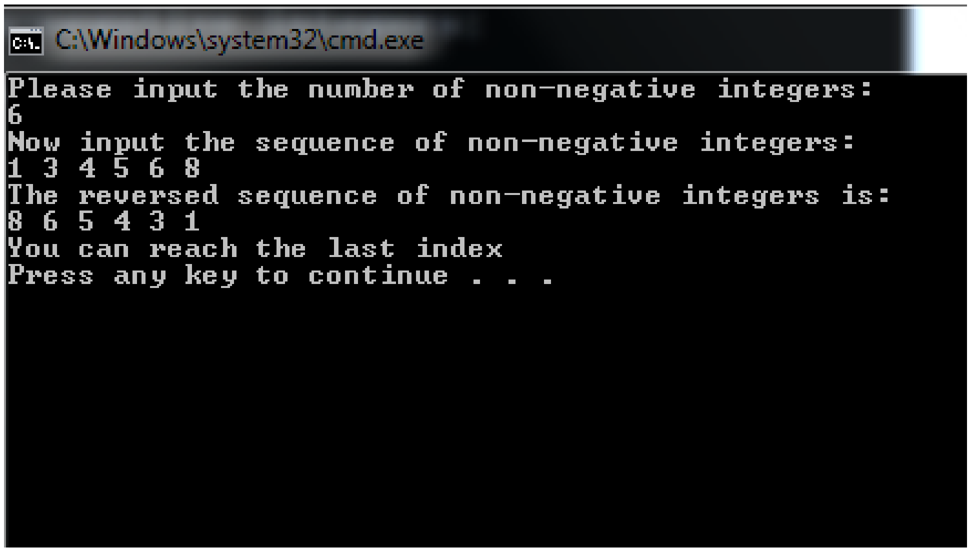
```
C:\Windows\system32\cmd.exe
Please input the number of non-negative integers:
5
Now input the sequence of non-negative integers:
4 1 1 3 2
The reversed sequence of non-negative integers is:
2 3 1 1 4
You can reach the last index
Press any key to continue . . .
```

Sample 2：



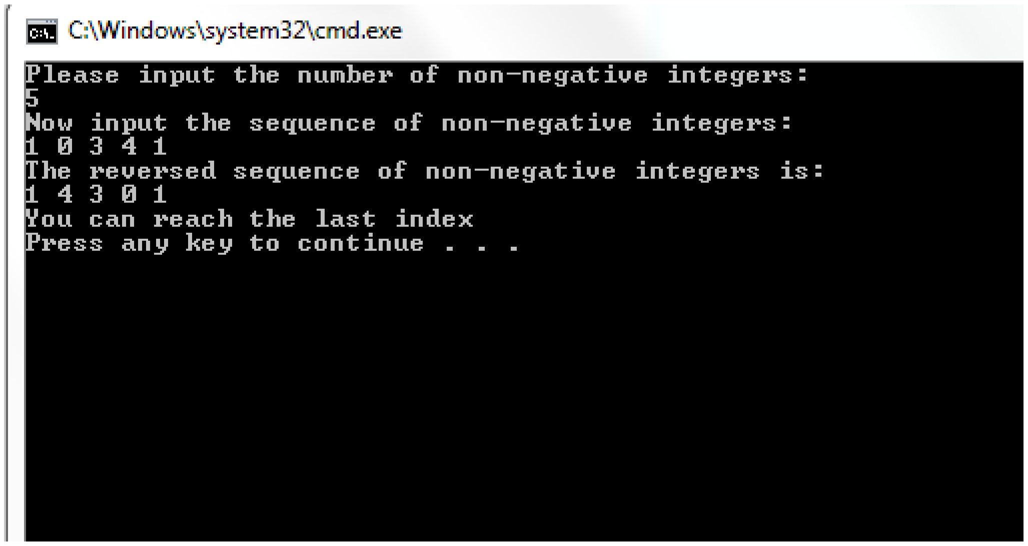
```
C:\Windows\system32\cmd.exe
Please input the number of non-negative integers:
4
Now input the sequence of non-negative integers:
4 3 0 1
The reversed sequence of non-negative integers is:
1 0 3 4
You cannot reach the last index
Press any key to continue . . .
```

Sample 3:



```
Please input the number of non-negative integers:
6
Now input the sequence of non-negative integers:
1 3 4 5 6 8
The reversed sequence of non-negative integers is:
8 6 5 4 3 1
You can reach the last index
Press any key to continue . . .
```

Sample 4:



```
Please input the number of non-negative integers:
5
Now input the sequence of non-negative integers:
1 0 3 4 1
The reversed sequence of non-negative integers is:
1 4 3 0 1
You can reach the last index
Press any key to continue . . .
```