CSC3002 Assignment 3

Important Notes:

1. The assignment is an individual project, to be finished on one's own effort.
2. **Submission deadline is 23:55pm Sunday, 21 April, 2024 (Week #4). After the deadline, we will reopen OJ for late submission for more 3 days.** Late submission will lead to some penalty – 10% deduction for each day of the delay and no submission would be accepted after 3 days of the due date.

3. Please also submit your final **four programs** and **your one page document**. The four programs should be renamed as "StudentID_A3_p1.cpp", "StudentID_A3_p2.cpp", "StudentID_A3_p3.cpp", and "StudentID_A3_ **p4**.cpp", respectively on the blackboard. For example, a student whose Student ID is "120000001" should submit **four programs** named as "120000001_A3_p1.cpp", "120000001_ A3_p2.cpp", "120000001_A3_p3.cpp", and "120000001_A3_p4.cpp". Furthermore, prepare **one page document to highlight your ideas of your solutions to the given four problems**. The one page document should be named as "StudentID_A3_solutions.pdf".

4. OJ website: http://10.26.200.13/. After opening this website, you must go the contest "**CSC3002 24Spring Assignment 3**". Please use **your student ID** as the user name for registration. We score your assignment only using your student ID. If you are off campus, please use VPN to access the OJ.

5. Each student is only permitted to submit code to OJ **up to 80 times for each problem**. Only the **last submission** will be used in evaluation of assignment marks.

6. Plagiarism is strictly forbidden, regardless of the role in the process. Notably, ten consecutive lines of identical codes are treated as plagiarism. Depending on the seriousness of the plagiarism, 30%-100% of marks will be deducted.

Marking Criterion:

1. The full score of the assignment is 100 marks.
2. **We have 4 problems in this assignment with 5 test cases each. Each test case has 5 marks.**
3. Zero mark is given if: there is no submission; a normal submission fails all test cases.

Running Environment:

1. The submissions will be evaluated in the course's OJ system running **C++17** and Linux platform.
2. All students will have an opportunity to test their programs in the OJ platform prior to the official submission.
3. In the test, each program is required to finish within **5 seconds**, with no more than **64MB memory. This is a strict requirement measured in the server environment!**

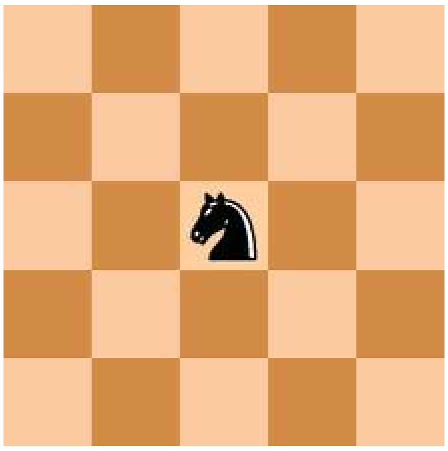
Submission Guidelines:

1. Inconsistency with or violation from the guideline leads to marks deduction.
2. It is the students' responsibility to read this assignment document and submission guidelines carefully and in detail. No argument will be accepted on issues that have been specified clearly in these documents.
3.

Problem 1 Description (The Knight's tour problem):

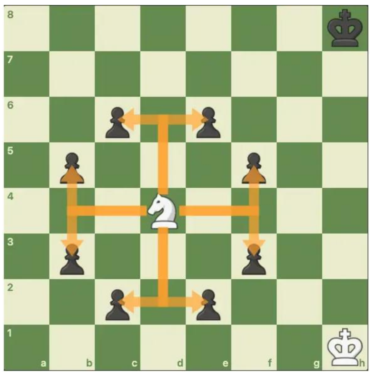
Write a program called **knight.cpp to solve the knight's tour problem.**

The problem requires us to simulate the tour of a knight on a chessboard. Given a 2D-array with dimensions **m by n** (**m rows and n columns**) representing the **chessboard**, and a fixed starting position **(m, 1)** for the **knight**, we need to move the knight around the board in **such a way that every cell is visited exactly once**. The values in the cells of the array should represent the order in which each cell is visited, **starting from 1** at the initial position of the knight.



An animation of an open knight's tour on a 5 × 5 board

A knight in chess **moves in an "L" shape**. In other words, a knight will **take two steps vertically** and then take the third step horizontally or the first two steps horizontally and vertically. **Mathematically, this means that for a move from (r1, c1) to (r2, c2): min(abs(r1 - r2), abs(c1 - c2)) should be 1 and max(abs(r1 - r2), abs(c1 - c2)) should be 2, with (r2, c2) being in the bounds of the board**.



In the diagram above, the knight can take any of the black pawns. The L-shaped arrows show all the possible movements of the knight.

You can read more details about the **knight's tour problem** by

https://en.wikipedia.org/wiki/Knight%27s_tour.

Following is a chessboard with 4 x 5 cells with the knight starting at the **(4,1)**. Numbers in cells indicate the move number of Knight.

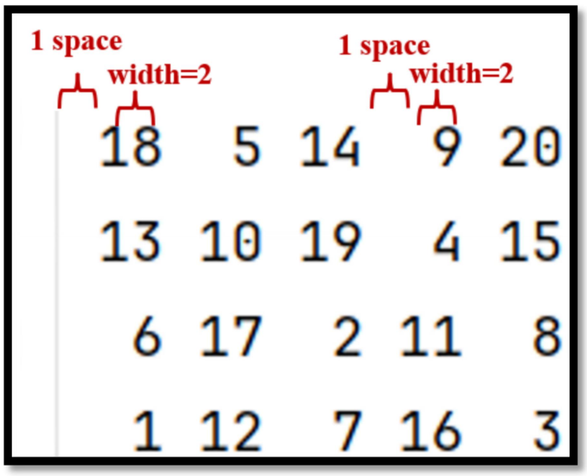
18	5	14	9	20
13	10	19	4	15
6	17	2	11	8
1 (starting point)	12	7	16	3

Input: Given a 2D-array with dimensions **m by n** representing the **chessboard**, and the starting position for the **knight is fixed at (m , 1)**. **1<=m<=9, 1<=n<=9**

```
4 5
```

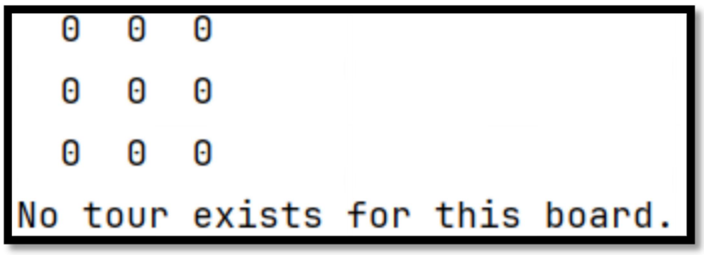
Output:

- If it is possible to visit every cell of the chessboard exactly once with starting position at **(m, 1)**, please output one possible order of each cell in which they are visited.
  Note that, the output format for each cell should begin **with a space**. **Then output the order number with width of 2 and aligned to the right.** You can use the following C++ code line to meet the format requirement.
  cout << " " << setw(2) << board[i][j];



```
 18  5 14  9 20
 13 10 19  4 15
  6 17  2 11  8
  1 12  7 16  3
```

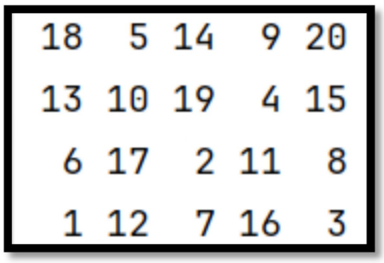
- **It there is no solution for the Knight** to visit every cell exactly once. Please output the chessboard with all 0 following the above format. Then, output the message "No tour exists for this board." with a line.


```
  0  0  0
  0  0  0
  0  0  0
No tour exists for this board.
```

Example 1: the input dimensions of the chessboard, m and n:

```
4 5
```

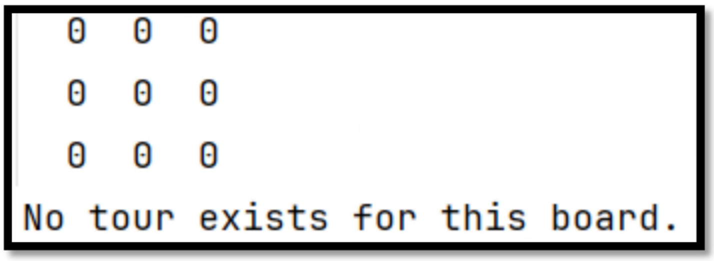
The corresponding output is as follows:


```
 18  5 14  9 20
 13 10 19  4 15
  6 17  2 11  8
  1 12  7 16  3
```

Example 2: the input dimensions of the chessboard, m and n:

```
3 3
```

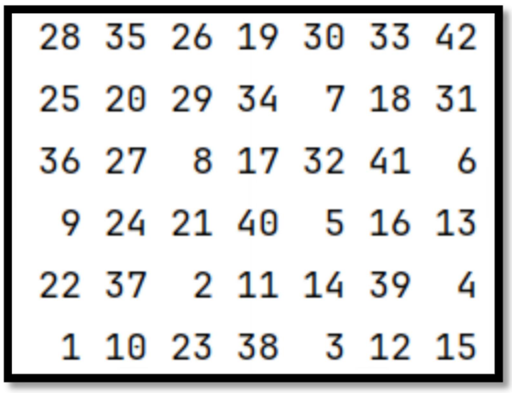
The corresponding output is as follows:


```
  0  0  0
  0  0  0
  0  0  0
No tour exists for this board.
```

Example 3: the input dimensions of the chessboard, m and n:

```
6 7
```

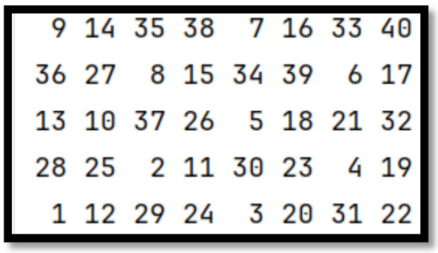
The corresponding output is as follows:


```
 28 35 26 19 30 33 42
 25 20 29 34  7 18 31
 36 27  8 17 32 41  6
  9 24 21 40  5 16 13
 22 37  2 11 14 39  4
  1 10 23 38  3 12 15
```

Example 2: the input dimensions of the chessboard, m and n:

```
5 8
```

The corresponding output is as follows:



```
 9 14 35 38  7 16 33 40
36 27  8 15 34 39  6 17
13 10 37 26  5 18 21 32
28 25  2 11 30 23  4 19
 1 12 29 24  3 20 31 22
```

Problem 2 Description (Program for merge sort)

Write a program called **merge_sort.cpp** that sort a given array from smallest to largest.
Please use the the merge sort algorithm to solve this problem.
We would have **5** test cases for evaluation.

Input: We have two lines as input. The first line is a positive integer, n, specifying the size of the array. **1<=n<=1000**
The second line lists the elements of the input array. It is possible for the input array to contain +/- integers, zeros, repeating integers and so on.

Output: Print the elements of the sorted array from the smallest to the largest.

Example 1: the input is as follows:

```
8
-4 1 2 6 8 11 14 0
```

The corresponding output is as follows:

```
-4 0 1 2 6 8 11 14
```

Example 2: the input is as follows:

```
6
13 19 -1 0 20 6
```

The corresponding output is as follows:

```
-1 0 6 13 19 20
```

Example 3: the input is as follows:

```
40
11 -2 -5 13 -1 5 -2 8 8 -1 10 0 2 15 4 6 -2 -1 10 -4 7 -2 10 0 20 4 -3 -5 2 2 11 -2 4 9
-5 6 17 -5 3 15
```

The corresponding output is as follows:

```
-5 -5 -5 -5 -4 -3 -2 -2 -2 -2 -2 -1 -1 -1 0 0 2 2 2 3 4 4 4 5 6 6 7 8 8 9 10 10 10 11 11
13 15 15 17 20
```

Problem 3 Description

Problem Statement

Design a text editor with a cursor that can do the following:

1. **Add** text to where the cursor is.
2. **Delete** text from where the cursor is (simulating the backspace key).
3. **Move** the cursor either left or right.

When deleting text, only characters to the left of the cursor will be deleted. The cursor will also remain within the actual text and cannot be moved beyond it. More formally, we have that 0 <= cursor.position <= currentText.length always holds.

Method:

Implement the TextEditor class:
TextEditor() Initializes the object with empty text.
string addText(string text) Appends text to where the cursor is. The cursor ends to the right of text.
string deleteText(int k) Deletes k characters to the left of the cursor. Returns the number of characters actually deleted.
string cursorLeft(int k) Moves the cursor to the left k times. Returns the last min(10, len) characters to the left of the cursor, where len is the number of characters to the left of the cursor.
string cursorRight(int k) Moves the cursor to the right k times. Returns the last min(10, len) characters to the left of the cursor, where len is the number of characters to the left of the cursor.

We use **Empty** to represent the empty string.

Testing:

Example 1:
the input is as follows:
We have four operations, addText, deleteText, cursorRight, cursorLeft. Only addText is followed by a word, which means the text we need to add, while other operations are followed by an int number, which indicates the number we need to move or delete. Please follow the operations and simulate the output.

```
addText leetcode deleteText 4 addText practice cursorRight 8 cursorLeft 10 deleteText 10
cursorLeft 2 cursorRight 6
```

The corresponding output is as follows:
The output should be the text after performing each operation.

```
leetcode leet leetpractice leetpractice leetpractice practice practice practice
```

Explanation
textEditor.addText("leetcode"); // The current text is "leetcode|".
textEditor.deleteText(4); //

// The current text is "leet|".
// 4 characters were deleted.
textEditor.addText("practice"); // The current text is "leetpractice|".
textEditor.cursorRight(3); //
// The current text is "leetpractice|".
// The cursor cannot be moved beyond the actual text and thus did not move.
// "etpractice" is the last 10 characters to the left of the cursor.
textEditor.cursorLeft(8); //
// The current text is "leet|practice".
// "leet" is the last min(10, 4) = 4 characters to the left of the cursor.
textEditor.deleteText(10); //
// The current text is "|practice".
// Only 4 characters were deleted.
textEditor.cursorLeft(2); //
// The current text is "|practice".
// The cursor cannot be moved beyond the actual text and thus did not move.
// "" is the last min(10, 0) = 0 characters to the left of the cursor.
textEditor.cursorRight(6); //
// The current text is "practi|ce".
// "practi" is the last min(10, 6) = 6 characters to the left of the cursor.

Example 1:
the input is as follows:

```
cursorLeft 6 cursorRight 8 cursorRight 5 deleteText 4 deleteText 2
```

The corresponding output is as follows:

```
Empty Empty Empty Empty Empty
```

Example 2:
the input is as follows:

```
addText icu deleteText 7 deleteText 4 addText rcoma cursorLeft 4
```

The corresponding output is as follows:

```
icu Empty Empty rcoma rcoma
```

Example 3:
the input is as follows:

```
addText bweyzt deleteText 2 addText fwwxtlomos cursorLeft 3 cursorLeft 4
```

The corresponding output is as follows:

```
bweyzt bwey bweyfwwxtlomos bweyfwwxtlomos bweyfwwxtlomos
```

Problem 4 Description

Problem Statement

In the queue abstraction presented in Lecture Notes #14, new items are always added at the end of the queue and wait their turn in line. For some programming applications, it is useful to extend the simple queue abstraction into a priority queue, in which the order of the items is determined by a numeric priority value. When an item is enqueued in a priority queue, it is inserted in the list ahead of any lower priority items. If two items in a queue have the same priority, they are processed in the standard first-in/first-out order. Using the linked-list implementation of queues as a model, design and implement a class called PriorityQueue, which exports the same methods as the traditional Queue class with the exception of the enqueue method, which now takes an additional argument, as follows: void enqueue(ValueType value, double priority); The parameter value is the same as for the traditional versions of enqueue; the priority argument is a numeric value representing the priority. As in conventional English usage, smaller integers correspond to higher priorities, so that priority 1 comes before priority 2, and so forth.

Method

In this implementation, your program should include functions for enqueue and dequeue to handle queue.

Testing

Input: Input contains two rows. The first row includes the initial queue. The letter "A-E" stands for the value and the number following the letter is the corresponding priority. For simplicity, we use int number "1-9" here. The second row includes the operations we need to perform. The word "dequeue" means finding and removing the element with the highest priority. The word "enqueue" means to insert the item using the rules stated above. There will be one letter "A-E" and one int number following the "enqueue", representing the insert value and corresponding priority.

```
C 2 B 4 C 4 A 5 D 6 C 6 A 7 D 7 E 7 B 9
dequeue enqueue E 1 dequeue dequeue dequeue dequeue
```

Output: Output contains one row, which means the output queue after performing the operations.

```
D C A D E B
```

Example 1: the input is as follows:

```
C 2 B 2 B 3 B 6 A 7 A 9
enqueue A 1 dequeue enqueue D 4 dequeue
```

The corresponding output is as follows:

```
B B D B A A
```

Example 2: the input is as follows:

```
D 2 A 4 E 5 B 9 A 9
```

```
dequeue enqueue D 9 enqueue E 2 dequeue enqueue B 1 enqueue E 7 dequeue
dequeue dequeue enqueue B 1
```

The corresponding output is as follows:

```
B E B A D
```

Example 3: the input is as follows:

```
E 1 D 1 E 1 E 3 A 4 C 8 B 9 A 9 E 9
dequeue enqueue C 1 dequeue enqueue E 3 enqueue A 3 dequeue dequeue
```

The corresponding output is as follows:

```
E E A A C B A E
```