CSC3002 2023 spring Assignment 4

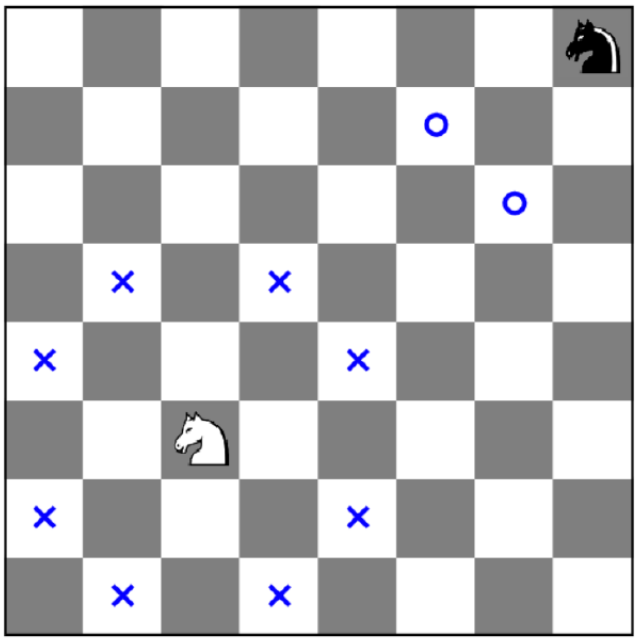
Due 23:59, April 9, 2023

Problem 1 (Exercise 9.6, Points: 25)

Problem Description

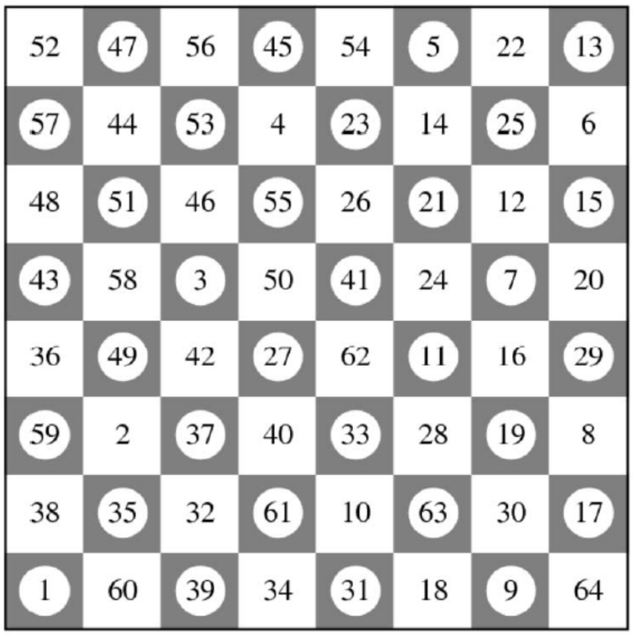
In chess, a knight moves in an L-shaped pattern: two squares in one direction horizontally or vertically, and then one square at right angles to that motion.

For example, the white knight in the upper right side of the following diagram can move to any of the eight squares marked with an $\times$.



The mobility of a knight decreases near the edge of the board, as illustrated by the black knight in the corner, which can reach only the two squares marked with an $\bigcirc$.

It turns out that a knight can visit all 64 squares on a chessboard without ever moving to the same square twice. A path for the knight that moves through all the squares without repeating a square is called a knight's tour. One such tour is shown in the following diagram, in which the numbers in the squares indicate the order in which they were visited:



52	47	56	45	54	5	22	13
57	44	53	4	23	14	25	6
48	51	46	55	26	21	12	15
43	58	3	50	41	24	7	20
36	49	42	27	62	11	16	29
59	2	37	40	33	28	19	8
38	35	32	61	10	63	30	17
1	60	39	34	31	18	9	64

Requirements Write a program that uses backtracking recursion to find a knight's tour. Please fill in the **TODO** part of `findKnightsTour` function in `KnightsTour.cpp`. You can define your own functions in the codes if necessary, but do not modify the provided completed functions.

In & Out Statement

Your program should receives two positive integers that represents the number of rows and columns in this chessboard. These two integers may **NOT** be equal.

- To input a text file:

```
p1.exe < in/p1.txt // for win user
./p1 < in/p1.txt // for Mac user
```

- To input manually:

```
p1.exe // for win user
./p1 // for Mac user
```

If one arbitrary knight's tour is detected, output the tour as the format in the file *out/p1.txt*:

If no knight's tour exists, all elements in the grid should be 0 and a message should be prompt. For instance, when you input `1 3`, the correct output is:

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

About P1

- In all OJ test cases, the number of cells in the chessboard entered will not exceed 75 [1].
- `Grid` class in this code template is different from that in Stanford C++ library. In this problem, `Grid` class is essentially a 2D vector as defined in `p1KnightsTour.h`.

[1]This is to get rid of the potential time limit exceed error in OJ.

Problem 2 (Exercise 11.7, Points: 25)

Problem Description

Rewrite the implementation of the merge sort algorithm from Figure 10-3 (shown as follows) so that it sorts an array rather than a vector.

FIGURE 10-3 **Implementation of the merge sort algorithm**

```
/*
 * Function: sort
 * --------------
 * This function sorts the elements of the vector into increasing order
 * using the merge sort algorithm, which consists of the following steps:
 *
 * 1. Divide the vector into two halves.
 * 2. Sort each of these smaller vectors recursively.
 * 3. Merge the two vectors back into the original one.
 */

void sort(Vector<int> & vec) {
   int n = vec.size();
   if (n <= 1) return;
   Vector<int> v1;
   Vector<int> v2;
   for (int i = 0; i < n; i++) {
      if (i < n / 2) {
         v1.add(vec[i]);
      } else {
         v2.add(vec[i]);
      }
   }
   sort(v1);
   sort(v2);
   vec.clear();
   merge(vec, v1, v2);
}

/*
 * Function: merge
 * ---------------
 * This function merges two sorted vectors, v1 and v2, into the vector
 * vec, which should be empty before this operation.  Because the input
 * vectors are sorted, the implementation can always select the first
 * unused element in one of the input vectors to fill the next position.
 */

void merge(Vector<int> & vec, Vector<int> & v1, Vector<int> & v2) {
   int n1 = v1.size();
   int n2 = v2.size();
   int p1 = 0;
   int p2 = 0;
   while (p1 < n1 && p2 < n2) {
      if (v1[p1] < v2[p2]) {
         vec.add(v1[p1++]);
      } else {
         vec.add(v2[p2++]);
      }
   }
   while (p1 < n1) vec.add(v1[p1++]);
   while (p2 < n2) vec.add(v2[p2++]);
}
```

As in the reimplementation follows this basic idea:

1. Copy the array into an array allocated on the stack.

2. Divide the new array into two halves by manipulating the indices.
3. Sort each half of the array recursively
4. Merge the two halves back into the original one.

Requirements Please fill in the **TODO** part of `sort` function in `p2MergeSort.cpp`. You can define your own functions in the codes if necessary, but do not modify the provided completed functions.

In & Out Statement

The input of this program consists of 2 parts: the first line is a positive integer n, which is the size of the array; the second line consists of all the elements in this array. It is possible for the input array to contain +/- integers, zeros, repeating integers and so on.

- To input a text file:

```
p2.exe < in/p2.txt // for win user
./p2 < in/p2.txt // for Mac user
```

- To input manually:

```
p2.exe // for win user
./p2 // for Mac user
```

The sorted array is then output by `printArray` in a certain format. Please check the file *out/p2.txt* for details.

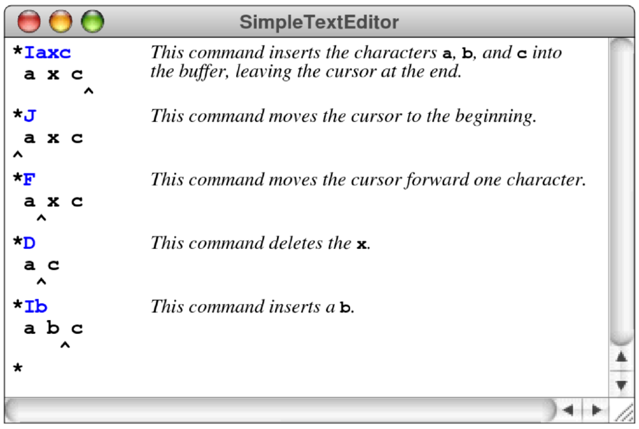
About P2

- In all OJ test cases, the actual size of the input array is identical to the input n.

Problem 3 (Exercise 13.12, Points: 25)

Problem Description

Implement the `EditorBffer` class using the strategy described in the section entitled "Doubly linked lists" (Page 606). Be sure to test your implementation as thoroughly as you can. In particular, make sure that you can move the cursor in both directions across parts of the buffer where you have recently made insertions and deletions. A sample program testing results are shown as follows (Your program doesn't have to have the same interface).



```
*Iaxc          This command inserts the characters a, b, and c into
 a x c         the buffer, leaving the cursor at the end.
      ^
*J             This command moves the cursor to the beginning.
 a x c
^
*F             This command moves the cursor forward one character.
 a x c
  ^
*D             This command deletes the x.
 a c
  ^
*Ib            This command inserts a b.
 a b c
    ^
*
```

In this implementation, the ends of the linked list are joined to form a ring, with the dummy cell at both the beginning and the end. This representation makes it possible to implement the `moveCursorToEnd` method in constant time, and reduces the number of special cases in the code. The constructor is already given. Methods need to be implemented:

- The destructor that delete all cells;
- Methods to move the cursor, including `moveCursorForward`, `moveCursorBackward`, `moveCursorToStart` and `moveCursorToEnd`[2];
- A `insertCharacter` method that inserts one character into the buffer on the cursor;
- A `deleteCharacter` method that deletes one character after the cursor;
- A `getText` method that returns the content in buffer;
- A `getCursor` method that returns the index of the cursor.

Requirements Please fill in the **TODO** part of the methods in `p3buffer.cpp`. Implement the `EditorBuffer` class using this representation (which is, in fact, the design strategy used in many editors today).

Make sure that your program continues to have the same computational efficiency as the two-stack implementation in the text and that the buffer space expands dynamically as needed.

You can define your own functions in the codes if necessary, but do not modify the provided completed functions.

Hint There are several functions defined in `SimpleTextEditor.cpp`, in which you can check the correspondence between the commands and operations. You don't need to modify any function in this file.

In & Out Statement

The input of this program is a set of commands. You are recommended to test this program using a input file.

- To input a text file:

```
p3.exe < in/p3.txt // for win user
./p3 < in/p3.txt // for Mac user
```

You may check the file *out/p3.txt* for standard output file.

Problem 4 (Exercise 9.11, Points: 25)

Problem Description

Most operating systems and many applications that allow users to work with files support ***wildcard patterns***, in which special characters are used to create filename patterns that can match many different files. The most common special characters used in wildcard matching are `?`, which matches any single character, and `*`, which matches any sequence of characters. Other characters in a filename pattern must match the corresponding character in a filename.

For example, the pattern `*.*` matches any filename that contains a period, such as `EnglishWords.dat` or `HelloWorld.cpp`, but would not match filenames that do not contain a period. Similarly, the pattern `test.?` matches any filename that consists of the name test, a period, and a single character; thus, `test.?` matches

[2]If the cursor is already at the end position, then `moveCursorForward` or `moveCursorToEnd` will not cause any movement to this cursor; the same effect is also available for `moveCursorBackward` or `moveCursorToStart` at the start position.

test.h but not test.cpp. These patterns can be combined in any way you like. For example, the pattern ??* matches any filename containing at least two characters.

Write a function

```
bool wildcardMatch(string filename, string pattern);
```

that takes two strings, representing a filename and a wildcard pattern, and returns true if that filename matches the pattern. Thus,

```
wildcardMatch("US.txt", "*.*")         returns  true
wildcardMatch("test", "*.*")           returns  false
wildcardMatch("test.h", "test.?")      returns  true
wildcardMatch("test.cpp", "test.?")    returns  false
wildcardMatch("x", "??*")              returns  false
wildcardMatch("yy", "??*")             returns  true
wildcardMatch("zzz", "??*")            returns  true
```

Requirements Please fill in the **TODO** part of the `wildcardMatch` method in the file `p4WildcardMatch.cpp`.

In & Out Statement

The input of this program is a set filename-patter pairs, which are split by the space. The string before the space is the filename and the string after the space is the pattern. You are recommended to test this program using a input file.

- To input a text file:

```
p4.exe < in/p4.txt // for win user
./p4 < in/p4.txt // for Mac user
```

The boolean matching result will be output line-by-line. You may check the file *out/p4.txt* for standard output file.

About P4

- In all OJ test cases, neither the filename nor the patterns will include the space character since it is regarded as a marker of input split.

About this assignment

- You don't need to modify or submit any other files other than `p1KnigntsTour.cpp`, `p2MergeSort.cpp`, `p3buffer.cpp` and `p4WildcardMatch.cpp`.
- Submission to OJ platform is required, while submission to Blackboard again is not necessary.
- You can only view your pre-test scores and such scores do NOT equal to your final score. Pre-test cases take up 10-30% cases of the final test (also known as formal test) cases in this assignment. The formal test will be conducted after the assignment. Your final score will ONLY depend on the codes in your latest submission.
- If you are late, 5 points will be deducted from each missing problem immediately after 00:00, and 5 points more every hour afterward until no more points can be deducted.