CSC3002 2023Fall Assignment 4

Due 23:59, Nov 19, 2023

About this assignment

- You don't need to modify or submit any other files other than `WildcardMatch.cpp`, `buffer.cpp` and `PriorityQueue.cpp`.
- StanfordCPPLib is not needed in this assignment.

Problem 1 (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 `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
```

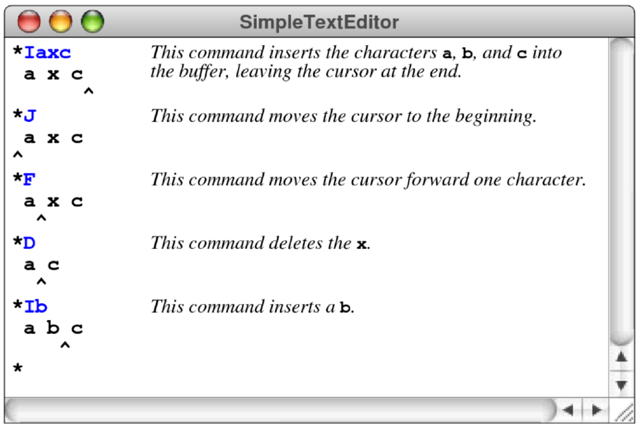
Requirements Please fill in the **TODO** part of the `wildcardMatch` method in the file `WildcardMatch.cpp`. It should be noted that in all OJ test cases, neither the filename nor the patterns will include the space character since it is regarded as a marker of input split.

In & Out Statement The input of this program is a set filename-pattern 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 an input file.

The boolean matching result will be output line-by-line. You may check the file *out/p1.txt* for standard output file.

Problem 2 (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`[1];
- 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 `buffer.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 an input file.

You may check the files *in/p2.txt* and *out/p2.txt* for standard in & out format.

[1]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.

Problem 3 (Exercise 14.8, Points: 25)

Problem Description

In the queue abstraction presented in this chapter, 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 [2] 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.

Requirement Please complete the **TODO** parts in the `enqueue`, `dequeue` and `peek` functions. It should be noted that, in all OJ test cases, neither the values nor the priorities will include the space character since it is regarded as a marker of input split.

In & Out Statement

Your program receives two elements in one line, split by the space: a `string` value to be put into the `PriorityQueue` and the corresponding priority of this element. The priority can be any value that fits the type `double`. You are recommended to test this program using an input file.

Functionalities of your implementation will be tested in the `main()` function, and you may check the files *in/p2.txt* and *out/p2.txt* for standard in & out format.

[2] `Queue` in Stanford C++ library, which is introduced in the lectures.