CSCI-398 **Homework 2** **List, Queue, Stack**

Download ***hw2.tar.*** To untar, use this command: tar -xvf hw2.tar

Problem 1. Given two single-linked sorted lists, write a method that merges the given lists into a single sorted list: the second list given as a parameter will be empty at the end of merging, and the list, on which we call this method will contain the resulting list sorted in increasing order.

The class list is defined in ***hw2/problem1_List*** and tests to test your program are provided in ***hw2/problem1_List/tests***

To compile, print "make" and press Enter. This will create executable ***run.***

To test use these commands:

./run < tests/t01.in > t01.my

diff t01.my tests/t01.out

Requirements:

1. The running time must be O(n + m), where n is the size of the first list and m is the size of the second list.
2. The space must be O(1). You may use temporary pointers or variables, but do not use any additional data structures such as an array or a vector or an additional list.

Problem 2. Given a double-linked list of integers, and an integer *pivot,* write a recursive member function that partitions the list around the pivot: all integers smaller than pivot will precede the integers that are greater than or equal to the pivot. The public method will take only one parameter, an integer pivot. A recursive private method that is called from this public method can take as many parameters as you need.

You are provided with the class Dlist, that can be used to create an object, a double-linked list, in which each node has a pointer *value* to an integer (data), and two pointers *prev* and *next* to the neighboring nodes. All provided files are inside ***hw2/problem2_Dlist.***

Requirements:

1. Because the recursive functions take space of the running stack, the space may be O(n).
2. The running time must be O(n).
3. Mimic *partition* procedure that is used in *quicksort* algorithm using arrays, i.e. your method should use two pointers that initially point to the head and tail of the list and move toward each other, swapping elements when needed.

Use the Figure below to "mimic" partition of the list.

```
int partition(vector<int> &A, int low, int high){
	int pivot = A[high];
	int i = low, j = high - 1;
	while(i <= j){
		while(i < high && A[i] <= pivot)
			i++;
		while(j >= low && A[j] > pivot)
			j--;
		if(i < j){//swap A[i] and A[j]
			int temp = A[i];
			A[i] = A[j];
			A[j] = temp;
			i++;
			j--;
		}//if
	}//while
	//swap A[i] and pivot
	A[high] = A[i];
	A[i] = pivot;
	return i;
}
```

Problem 3. All provided files for this problem are inside ***hw2/problem3_Stack.***
(A) Given a string over two characters, i.e. the left and right parentheses '(', ')', write a function that returns true if the string consists of the balanced parenthesis. Use stack data structure.
Definition: a string of parentheses is balanced if each left parenthesis matches right parenthesis when reading from left to right.
Example: (())((()())())(((())()) is a sequence of balanced parentheses, but (() is not.

Requirements:

1. Your function does not need to verify that the input string contains only parentheses characters.
2. The running time must be O(n).
3. The space is O(n).
4. Write *main.cpp,* in which you will read an input the string using ***cin***. Then you will call the function that returns Boolean *true* if the input string is a string of balanced parentheses and will return *false,* otherwise.
5. The output of your program will be a single word "True." or "False." followed by ***endl.***

(B) Think how to solve this problem using O(1) space?

Write another function that performs the same task, but does not use stack and runs in O(1) space. You don't have to call this function from main, I will check it by reading your code. You may to test it using the same tests.