CS 469: Special Topics in Computer Science

Assignment 3

Exercise 1: Linked lists

Let consider the following template of linked lists

```
template <typename T>
class LinkedList{
private:
    // Declare a structure for the list
    struct ListNode
    {
        T value;
        struct ListNode *next;
    };
    ListNode *head;        // List head pointer
public:
    LinkedList(void)       // Constructor
        { head = NULL; }

    LinkedList(T);
    ~LinkedList(void); // Destructor
    void appendNode(T);

    T& top();

    T& pop_front();

    bool empty();
    void insertNode(T);
    void deleteNode(T, bool);
    void displayList(void);
    int count(T);
    int length();
```

```
    void sortBySelection();
    void sortByInsertion();

    ListNode *getNode(int);

    T& get(int);

    void clear();

    void reverse();
};
```

Write an algorithm and a C++ program for each of the following methods for this LinkedList class:

1. **LinkedList(T info)**: the constructor to create a linked a list which first node contains the value info and the next pointer is null.
2. **~LinkedList()**: a destructor that remove all elements of the linked list.
3. **void appendNode(T)**: a method to append a node at the end of the linked list.
4. **T& top()**: to get the first element of the linked list.
5. **T& pop_front()**: to remove the first node of the linked list and to return its value.
6. **bool empty()**: to test if the linked list is empty or no.
7. **void insertNode(T info)**: to insert a node in the beginning of the linked list.
8. **void deleteNode(T info, bool removeAll)**: to delete a node from the linked list, if removeAll is false, all occurrence of info will be removed, otherwise, only the first occurrence of info will be removed.
9. **void displayList(void)**: to display all elements of the linked list
10. **int count(T val)**: to count the occurrence of val in the linked list.
11. **int length()**: to count the number of nodes in the linked list.
12. **ListNode *getNode(int i)**: to get the i-th node of the linked list
13. **T& get(int i)**: to get element of the i-th node of the linked list.
14. **void clear()**: to remove all elements of the linked list
15. **void sortBySelection()**: to sort the linked list using the principle of sorting by selection.

16. **void sortByInsertion()**; to sort the linked list using the principle of sorting by insertion.
17. **void reverse()**: to reverse the elements of the linked list. Please, do not use an intermediary linked list to reverse the current linked list.

Hint: some method can reuse other methods, like for example you can reuse the method **clear()** in the destructor, and **void insertNode(T info)** in the constructor **LinkedList(T)**.

Exercise 2: Dynamic arrays

Let consider the following template of dynamic arrays

```
template <typename T>
Class DynamicArray{
        T *array;
        int size; //current size of the array
        int capacity; // the total capacity to be allocated for the array
public:
        DynamicArray(int Capacity); //Constructor
        ~ DynamicArray(); // Desctructor
        void add(T elem);
        void remove(T elem);
};
```

1. Complete the constructor
   public DynamicArray::DynamicArray(int Capacity){

}

2. Complete the desctructor to destroy the array.

   DynamicArray:: ~ DynamicArray(){

   }

3. Complete the function *add* to insert an elem in the dynamic array.

   void DynamicArray::add(T elem){

   }

4. Complete the function remove to search for an element and to delete it from the dynamic array.

   void DynamicArray::remove(T elem){

   } // End of remove

Exercise 3: Stacks

Implement the stacks using the two data structures dynamic array and linked lists. You can reuse the linked list class in the exercise 1 and the template of dynamic arrays in the exercise 2. For that purpose, you have to create a class called *Stack_DynamicArray* for the dynamic arrays based implementation, and another class called *Stack_LinkedList* for the linked lists based implementation.

Exercise 4: Queues

Implement the queues using the three data structures dynamic arrays, circular dynamic arrays and linked lists. You can reuse the linked list class in the exercise 1 and the template of dynamic arrays in the exercise 2. For that purpose, you have to create a class called *Queue_DynamicArray* for the dynamic arrays based implementation, another class called *Queue_Circular_DynamicArray* for the circular dynamic arrays implementation, and another class called *Queue_LinkedList* for the linked lists based implementation.