King Fahd University of Petroleum and Minerals

ICS Department

# ICS-202 Data Structures

Assignment 2

First Semester 2021-22

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| Student ID | | | | | | | | | |

| | | | | | | | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Name** | # | # | # | # | # | # | # | # | # | # | # | # | # | # | # | # | # | # | # | # | # | # | # | # | # |

# ICS 202 – Data Structures and Algorithms

**First Semester 2021-22 (20211)**

**SOLUTION to Assignment # 2**

**Due on Saturday, 16 October 2021, 11.59pm**

---

## Submissions Guidelines

- Include a screenshot of a sample run of your programs.
- For each question, submit all java classes as one java file with your name and ID as a comment in the first two lines.
- The java file name must be the same as the main class name, both have the following format (**ID#_Q#**), where ID# will be your ID and Q# will be the question number. For example: **s201706981_Q1, s201706981_Q2.**
- Compress both java files along with the screenshots in one .rar file renamed as your ID.
- The input and the output of the program will be from & to files.
- For the **input file:**
    - The program will take the file name as an argument **(args [0])**.
- For the **output file:**
    - The program will take the output file name as an argument **(args [1]).**
- You are free to explore the input and output methods as you develop the solution to the assignment; but before the submission, you have to prepare all the programs to take the input from a file in **(args [0])** and print the output to the file in **(args [1]).**
- **Attached** to this assignment are sample input and output files for each question. Those files have the same format as the files that your program will be tested through. So, **please adhere to the format in those files, and make sure that your code can read and write the data given in that format.**
- Submit the .rar file through Blackboard.
- Only submissions that follow the above guidelines will be evaluated.

## Question 1 (20 + 20 + 20 = 60 Marks)

**Add the following methods to the class SLL<T>:**

1. **public void insertBefore(T newElem, T existingElem)**
2. **public void insertAfter(T newElem, T existingElem)**
3. **public void deleteBefore(T existingElem)**

Below is shown the way the first one works, and the other two methods will follow the same methodology.

**public void insertBefore(T newElem, T existingElem)**

Inserts an element **newElem** before the element **existingElem**. If no **existingElem** exists, then the method prints `-1` and returns. If more than one instance of **exisingElem** exists, then the methods inserts before the first instance of **existingElem**.
For example, suppose your linked list (of integers) is: [3 5 4 2 9],
Then a call to **insertBefore(new Integer(5), new Integer(9))** would result in the following linked list: [3 5 4 2 5 9]
A call to **insertBefore(new Integer(7), new Integer(5))** would result in [3 7 5 4 2 5 9]
A call to **insertBefore(new Integer(8), new Integer(10))** would result in `-1`

| Sample Input | Sample Output |
|---|---|
| 1 2 3 4 5 | 1 2 2 3 4 5 |
| 5 | 2 1 2 2 3 4 5 |
| ia 2 1 | 2 1 2 3 4 5 |
| ib 2 1 | 1 2 3 4 5 |
| db 3 | -1 |
| db 1 | |
| ia 1 7 | |

As it is shown in the sample input, the first line will have the list of integers that you will work with.
The second line will have an integer N, indicating the number of following lines where each line will have a command as shown.

- **ia 2 1** stands for **insertAfter (2,1)**
- **ib 2 1** stands for **insertBefore (2,1)**
- **db** stands for **deleteBefore (1)**

```
public void insertBefore(T newElem, T existingElem) {
        if(!isInList(existingElem)) { System.out.println("-1"); return;}
        else if (existingElem.equals(head.info)) addToHead(newElem);

        else {
        SLLNode<T> tmp = head;
        while (tmp.next != null) {
                if (tmp.next.info.equals(existingElem)) {
                        SLLNode<T> nextNode = new SLLNode<T>(newElem);
                        nextNode.next = tmp.next;
                        tmp.next = nextNode;
                        break;
                }

                tmp = tmp.next;
        }}}


public void insertAfter(T newElem, T existingElem) {
        if(!isInList(existingElem)) { System.out.println("-1"); return;}

        else {
        SLLNode<T> tmp = head;
        while (tmp!= null) {
                if (tmp.info.equals(existingElem)) {
                        SLLNode<T> nextNode = new SLLNode<T>(newElem);
                        nextNode.next = tmp.next;
                        tmp.next = nextNode;
                        break;
                }

                tmp = tmp.next;
        }}}
```

```
public void deleteBefore(T existingElem) {

        if(!isInList(existingElem)) { System.out.println("-1"); return;}
                else if (existingElem.equals(head.next.info)) deleteFromHead();
                  else if (existingElem.equals(head.info)) System.out.println("nothig before head.");

                        else {
                        SLLNode<T> tmp = head;
                        while (tmp.next.next != null) {
                                if (tmp.next.next.info.equals(existingElem)) {
                                        SLLNode<T> toDelete = new SLLNode<T>();
                                        toDelete = tmp.next;
                                        tmp.next = toDelete.next;



                                        break;
                                }

                                tmp = tmp.next;
                        }}
```

## <u>Question 2 (20 + 20 = 40 Marks)</u>

You are asked to take as input a list of integers that went through several *reverse operations*, and your task is to retrieve the original list, **you may use java.util.Stack and java.util.LinkedList as helpers.**

The operations were as follows:

- Select each subpart of the list that contain even integers only.
  <u>For example</u>, if the list was {5,20,8,7,12,18}, then the selected subparts will be {20,8}, {12,18}.
- Reverse the selected subpart such as {8,20} and {18,12}.

The output should be the original list.

| Sample Input | Sample Output |
|---|---|
| 9<br>2 18 24 3 5 7 9 6 12 | 24 18 2 3 5 7 9 12 6 |

As it is shown in the input sample, the first line will have an integer indicating the number of the elements in the list followed by the second line which has the list of items.

In the sample input, 9 represents the number of items in the list.
As you can see, the subparts that contains only even integers in the given list are {2,18,24} and {6,12}. The corresponding output (which is the original list) is shown in the sample output section.

```
public static void main(String[] args) throws FileNotFoundException {

        LinkedList<Integer> LL = new LinkedList<Integer>();
        Stack<Integer> st = new Stack<Integer>();
        File file = new File(args[0]);
        Scanner sc1 = new Scanner(file);
        int count=sc1.nextInt();
```

```
 for(int i=0;i<count;i++){
    int t=sc1.nextInt();
    LL.add(t);      }
 int tmp1=0,tmp2=0;
 while(tmp1!=LL.size() && tmp2!=LL.size()){
    if(LL.get(tmp2)%2==0)      {
        st.push(LL.get(tmp2));
        ++tmp2;        }
    else {
            while(tmp1<tmp2){
            LL.set(tmp1,st.pop());
            tmp1++;             }
        tmp1++;
        tmp2++;     }       }
 if(!st.empty()) {
   while(tmp1<tmp2){
        LL.set(tmp1,st.pop());
        tmp1++;          }        }

String output="";
for(int i=0;i<LL.size();i++) {
      output+=LL.get(i)+" ";       }
 try {
    File myObj = new File(args[1]);
    if (myObj.createNewFile()) {
        System.out.println("File created: " + myObj.getName());
        FileWriter myWriter = new FileWriter(args[1]);
        myWriter.write(output);

        myWriter.close();
    } else {
        System.out.println("File already exists.");
    }
 } catch (IOException e) {
    System.out.println("An error occurred.");
 }
```