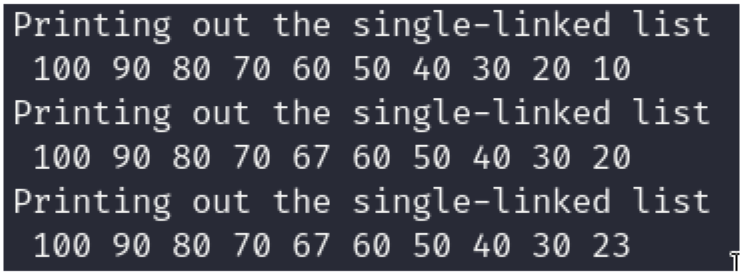
Assignment 1
Due date Sep. 15 by 11.59 Pm
For each day late submission 2 points penalty is incurred.
Copy your program and the out put under word ANSWER in this document and post it is the folder: assignment 1 on the blackboard.

1.Write a program that maintains the top ten scores for a game application, implementing the add and remove methods using a singly linked list instead of an array

Output

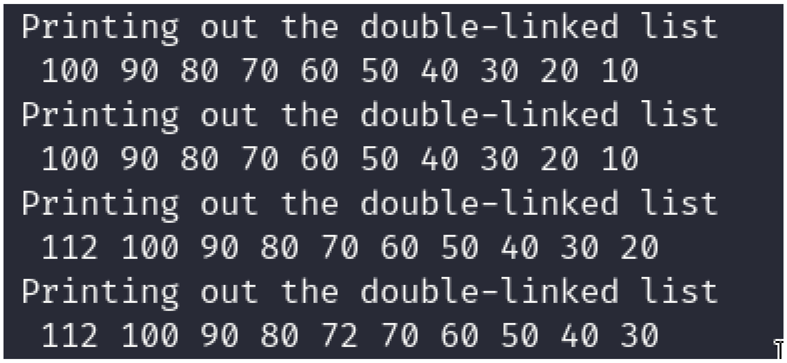


```
Printing out the single-linked list
 100 90 80 70 60 50 40 30 20 10
Printing out the single-linked list
 100 90 80 70 67 60 50 40 30 20
Printing out the single-linked list
 100 90 80 70 67 60 50 40 30 23
```

```
/**
 * The single-linked version of the linked list for assignment #1
 *
 * @author Kyle Guarco
 * @version sll-1
 */
public class SingleLinkedList<T extends Comparable<T>>
{
    public static void main(String[] args)
    {
        SingleLinkedList<Integer> sll = new SingleLinkedList<>();

           for (int i = 100; i > 0; i -= 10)
                sll.add(i);

        sll.print();
        sll.add(67);
        sll.print();
        sll.add(23);
        sll.print();
    }

    private Node<T> first;
    private Node<T> head;
    private int size;

    public void add(T data)
    {
           Node<T> newnode = new Node<>(data);

        if (first == null)
        {
            first = newnode;
            head = first;
            size = 1;
            return;
        }

        boolean checkedFirst = false;
        if (first.data.compareTo(newnode.data) < 0)
        {
            newnode.next = first;
            first = newnode;
            checkedFirst = true;
        }

        if (!checkedFirst)
        {
            Node<T> noderef = first;

            while (noderef.next != null)
            {
                if (noderef.next != null &&
newnode.data.compareTo(noderef.next.data) > 0)
                    break;

                noderef = noderef.next;
            }

            newnode.next = noderef.next;
            noderef.next = newnode;

            while (noderef.next.next != null)
                noderef = noderef.next;
            head = noderef;
        }

        size++;
        if (size > 10)
            remove(head);
    }

    public boolean remove(Node<T> noderef)
    {
        if (noderef == null)
            return false;

        Node<T> nextref = noderef.next;

        // Set the node to the one after it, if it exists.
        if (nextref != null)
            noderef.next = nextref.next;

        // Set the head to the new "front" of the list,
        // using the current position
        // to traverse the rest of the list.
        while (noderef.next != null)
            noderef = noderef.next;

        head = noderef;

        size--;
        return true;
    }

    public void print()
    {
        Node<T> noderef = first;

        System.out.println("Printing out the single-linked list");
        while (noderef != null)
        {
            System.out.print(" " + noderef.data);
            noderef = noderef.next;
        }
        System.out.println();
    }

    static class Node<T>
    {
        public Node<T> next;
        public T data;

        public Node(T data)
        {
            this.data = data;
        }
    }
}
```

2. Perform the previous project, but use a doubly linked list. Moreover, your implementation of remove(i) should make the fewest number of pointer hops to get to the game entry at index i.

Output



```
Printing out the double-linked list
 100 90 80 70 60 50 40 30 20 10
Printing out the double-linked list
 100 90 80 70 60 50 40 30 20 10
Printing out the double-linked list
 112 100 90 80 70 60 50 40 30 20
Printing out the double-linked list
 112 100 90 80 72 70 60 50 40 30
```

```java
/**
 * The double-linked version of the linked list for assignment #1
 *
 * @author Kyle Guarco
 * @version dll-1
 */
public class DoubleLinkedList<T extends Comparable<T>>
{
    public static void main(String[] args)
    {
        DoubleLinkedList<Integer> dll = new DoubleLinkedList<>();

        for (int i = 100; i > 0; i -= 10)
            dll.add(i);

        dll.print();
        dll.add(5);
        dll.print();
        dll.add(112);
        dll.print();
        dll.add(72);
        dll.print();
    }

    private Node<T> first;
    private Node<T> tail;
    private int size;
    public void add(T data)
    {
        Node<T> newnode = new Node<T>(data);

        if (first == null)
        {
            first = newnode;
            tail = first;
            size = 1;
            return;
        }

        boolean checkedFirst = false;
        if (newnode.data.compareTo(first.data) > 0)
        {
            newnode.next = first;
            first = newnode;
            checkedFirst = true;
        }

        if (!checkedFirst)
        {
            Node<T> noderef = tail;

            while (noderef.previous != null)
            {
                if (newnode.data.compareTo(noderef.data) < 0)
                    break;

                noderef = noderef.previous;
            }

            newnode.previous = noderef;
            newnode.next = noderef.next;
            noderef.next = newnode;

            while (noderef.next != null)
                noderef = noderef.next;

            tail = noderef;
        }

        size++;
        if (size > 10)
            remove(tail);
    }

    /**
     * @param noderef
     * @return Was the specified node removed from the list?
     */
    public boolean remove(Node<T> noderef)
    {
        if (noderef == null)
            return false;

        Node<T> nextref = noderef.next;
        Node<T> prevref = noderef.previous;

        if (nextref != null)
            nextref.previous = prevref;
        if (prevref != null)
            prevref.next = nextref;

        // Since noderef hasn't been garbage-collected yet, check and
        // see if either end of the list has been tampered with.
        if (noderef.data == first.data)
            first = nextref;
        else if (noderef.data == tail.data)
            tail = prevref;

        size--;
        return true;
    }

    public void print()
    {
        Node<T> noderef = first;

        System.out.println("Printing out the double-linked list");
        while (noderef != null)
        {
            System.out.print(" " + noderef.data);
            noderef = noderef.next;
        }
        System.out.println();
    }

    static class Node<T>
    {
        public Node<T> previous, next;
        public T data;

        public Node(T data)
        {
            this.data = data;
        }
    }
}
```