CSCI 2110
Data Structures and Algorithms
TEST NO. 2

TOPICS AND PRACTICE QUESTION BANK

SOLUTIONS

TOPICS

Module 3: Unordered Lists

- Data Structures: The Big Picture
- Abstract Data Type
- Concept of layering in building data structures
- **Unordered Lists:**
    - Definition and examples
    - Implementation choices
    - Linked Lists: Concept and Advantages
    - Generic Node Class
    - Generic Linked List Class: LINKED LIST methods and examples
    - Generic Unordered List Class– Know how the methods work and how to use them
    - Application Examples with Unordered Lists
    - Complexity Analysis of Unordered Lists
    - Smart Search Method

Module 4: Ordered Lists

- Definition
- Examples
- BINARY SEARCH algorithm: Concept and Examples
- Pseudocode for Binary Search
- Complexity of Binary Search of n items – n is a power of 2
- Complexity of Binary Search of n items – n is not a power of 2
- Generic Ordered List Class – Know how the methods work and how to use them
- Application examples with Ordered Lists
- Merging two ordered lists – the two-finger walking algorithm

Types of Questions

1. Multiple Choice Set
2. Short answer concept questions
3. Problem-solution type
4. Coding type

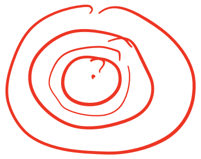
PRACTICE QUESTION BANK FOR TEST NO. 2

Note 1: This is meant to be a question bank covering the types of questions you will get on the test. The actual number of questions that will be designed to fit the test time (1 hour 30 minutes).
Note 2: The question bank contains representative samples of different topics. Please be sure to study all the topics for the test.

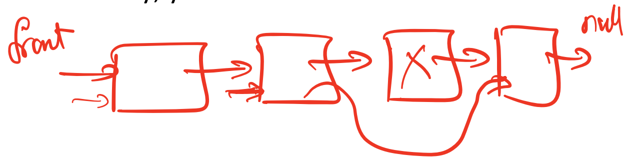
Multiple Choice Set (Choose the most appropriate answer in each of the following questions).

1. 1. We discussed the concept of layering in data structures in the lectures. Which of the following statement(s) is/are true with respect to layering:
   I. Layering means building one data structure from other (smaller) data structures.
   II. Layering means that all tasks of an application are divided into sets; each layer implements one set of tasks.
   III. The outer application layer is able to use the methods in any data structure in any of the inner layers.
   IV. Each layer is allowed to use the methods of only the data structure one layer below it.

Choose one of the following:



   a. I and II
   b. I and III
   c. II and III
   [✓ d. I and IV]
   e. II and IV
2. An abstract data type refers to
   a. the abstract methods in a class file.
   b. the type of data stored in a data structure with implementation details.
   [✓ c. what the data structure contains and what operations can be done without implementation details.]
   d. what the data structure contains and what operations can be done with implementation details.
3. The following exception is generated when you try to reference a node past the last node in a linked list.
   a. LinkedList OutOfBounds exception
   b. Unknown Error exception
   c. IndexOutOfBounds exception
   [✓ d. NullPointer exception]
4. To remove a node that is not the first node of a linked list conveniently, you need to have a reference to
   [✓ a. the node that is before it]
   b. the node that comes after it
   c. the node itself
   d. null



5. The complexity of adding a node to the front of a linked list of n items is:
   a. O(n)
   b. O(n^2)
   [✓ c. O(1)]
   d. O($\log^2 n$)

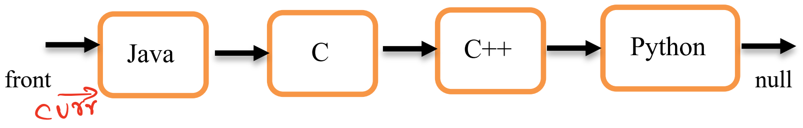
6. The complexity of removing all nodes with a given item in a linked list of n items is:
   [✓ a. O(n)]
   b. O(n^2)
   c. O(1)
   d. O($\log^2 n$)

For questions (7-14), assume that the following linked list has been created using the generic Node class and the generic LinkedList class. front is the reference to the first node and null is the reference from the last node. The linked list contains String data "Java", "C", "C++", and "Python" as shown. (NOTE: front and null are just references, not data).

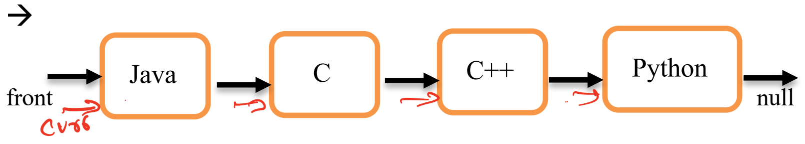
front → Java → C → C++ → Python → null

Study the following codes and select the most appropriate answer.

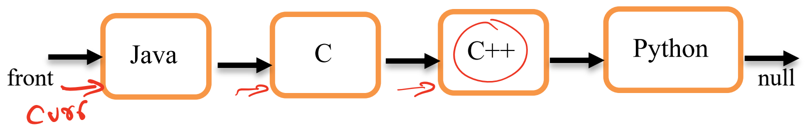
7. What does the following piece of code display?
```
Node<T> curr = front;
while (curr!=null){
    System.out.print(curr.getData() + "→");
    curr = curr.getNext();
}
```
   a. Java → C → C++ → Python → null
   b. front → Java → C → C++ → Python → null
   [✓ c. Java → C → C++ → Python →]
   d. Java → C → C++ →
   e. Null

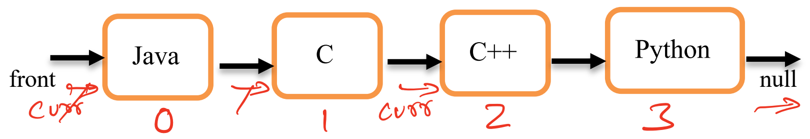


8. What does the following piece of code display?
```
Node<T> curr = front;
while (curr.getNext()!=null){
    System.out.print(curr.getData() + "→");
    curr = curr.getNext();
}
```
   a. Java → C → C++ → Python → null
   b. front → Java → C → C++ → Python → null
   c. Java → C → C++ → Python →
   [✓ d. Java → C → C++ →]
   e. null



9. What does the following piece of code display?
```
Node<T> curr = front;
System.out.println(curr.getNext().getNext().getData());
```
   a. Java
   b. C
   [✓ c. C++]
   d. Python
   e. It will give a NullPointer exception





10. What does the following piece of code display?

```
Node<T> curr = front;
for (int i = 0; i< 2; i++){
    curr = curr.getNext();
}
System.out.println(curr.getData());
```

i = 0
i = 1

a. Java
b. C
[✓ c. C++]
d. Python
e. It will give a NullPointer exception

11. What does the following piece of code display?

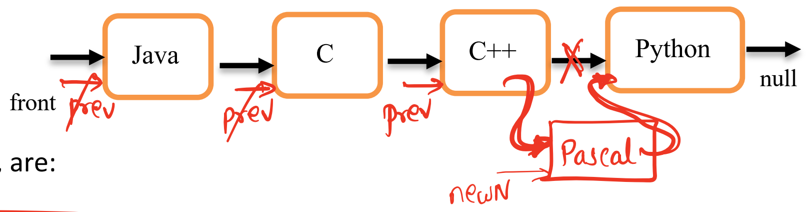
```
Node<T> curr = front;
for (int i = 0; i<=3; i++){
    curr = curr.getNext();
}
System.out.println(curr.getData());
```

a. Java
b. C
c. C++
d. Python
[✓ e. It will give a NullPointer exception]

12. Suppose you want to add a node with data "Pascal" after "C++". That is, the new linked list should be:
front → Java → C → C++ → Pascal → Python → null

Consider the following piece of code to accomplish the above task:

```
Node<T> prev = front;
prev = prev.getNext();
prev = prev.getNext();
//Missing line 1
//Missing line 2
```



The missing lines 1 and 2, respectively, are:

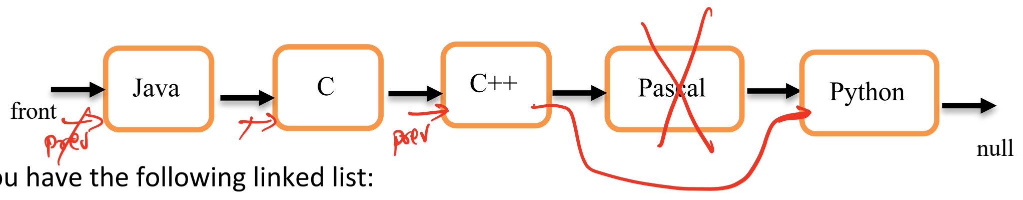
[✓ a. `Node<T> newN = new Node<T> ("Pascal", prev.getNext()); //Missing line 1`
`prev.setNext(newN); //Missing line 2`]

b. `prev.setNext(newN); //Missing line 1`
`Node<T> newN = new Node<T> ("Pascal", prev.getNext()); //Missing line 2`

c. `prev.getNext().setNext(newN); //Missing line 1`
`Node<T> newN = new Node<T> ("Pascal", prev.getNext().getNext());`
`//Missing line 2`

d. `Node<T> newN = new Node<T> ("Pascal", prev.getNext().getNext());`
`//Missing line 1`
`prev.getNext().setNext(newN); //Missing line 2`

e. None of the above



13. Now suppose that you have the following linked list:
front → Java → C → C++ → Pascal → Python → null
You want to delete the node with “Pascal”.
Consider the following piece of code to accomplish the above task:

```
Node<T> prev = front;
prev = prev.getNext();
prev = prev.getNext();
//Missing line
```

Which of the following is the correct code for the missing line?

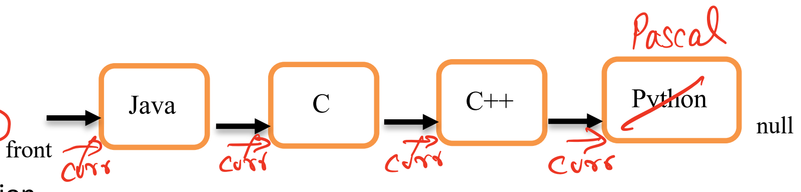
[✓ a. `prev.setNext(prev.getNext().getNext());`]
b. `prev.setNext(prev.getNext());`
c. `prev.setNext(prev.getNext().getNext().getNext();`
d. None of the above

14. Now suppose that you are back to the original linked list:
front → Java → C → C++ → Python → null

What will the following piece of code do?

```
Node<T> curr = front;
curr = curr.getNext().getNext().getNext();
curr.setData("Pascal");
```

a. It will change C++ to Pascal
[✓ b. It will change Python to Pascal]
c. It will change C to Pascal
d. It will give a NullPointer exception



15. The maximum number of searches required by the binary search algorithm to search an ordered list of n items, where n is a power of 2, is
a. n
b. n+1
c. $\log_2 n$
[✓ d. $\log_2 n + 1$]

16. Suppose an ordered list {W, X, Y, Z} is merged with another ordered list {A, B, X, Y} using the 2-finger walking algorithm, what is the resulting ordered list?
a. {W, X, A, B, X, Y, Z}
b. {A, B, W, X, X, Y, Y, Z}
c. {X, Y}
[✓ d. {A, B, W, X, Y, Z}]

{A, B, W, X, Y, Z}

Question 4:

Suppose the generic Node class has been developed as shown below.

```
public class Node<T>{
        private T data;
        private Node<T> next;
        public Node (T data, Node<T> next){
                this.data = data;
                this.next = next;
        }
        public T getData(){
                return data;
        }
        public Node<T> getNext(){
                return this.next;
        }
        public void setData(T data){
                this.data = data;
        }
        public void setNext(Node<T> next){
                this.next = next;
        }

}
```

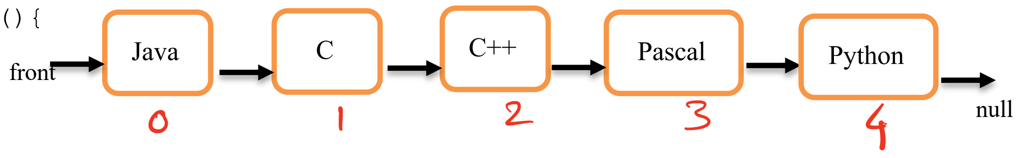
The LinkedList<T> class has the following structure.

```
public class LinkedList<T>
{
        private Node<T> front;
        private int count;

        public LinkedList()
        {
                front = null;
                count=0;
        }
```

Add the following methods to the above LinkedList class:

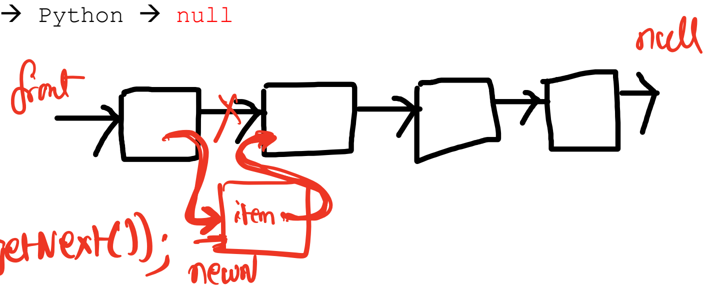
```
//scan the list and display the items in the list
public void enumerate(){
 Node <T> curr = front;
 while (curr != null)
 {
        System.out.print (curr.getData() + "\t");
        curr = curr.getNext();
} }
```

```
//scan the list and display the items with odd indices 1, 3, 5, etc.
//For example, if the list is
// front → Java → C → C++ → Pascal → Python → Cobol → null
// it should display: C  Pascal Cobol
//Note: The list is indexed from 0.

public void displayOdd(){
```

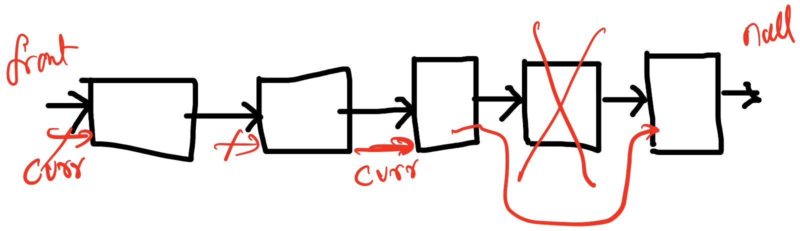


```
Node<T> curr=front;
int i= 0;
while ( curr != null)
{     if (i%2 == 1)
            System.out.print (curr.getData () + "\t");
      curr = curr.getNext();
      i++;
} }
```

```
//Add a given item after the first node. If the list is empty, just return.
//For example, if the list is
// front → Java → C → C++ → Pascal → Python → null
// addAfterFront("Cobol") should change the list to
// front → Java → Cobol → C → C++ Pascal → Python → null

public addAfterFront(T item){
if (front == null) return;
Node<T> newN = new Node<T>(item,
                           front.getNext());
front.setNext(newN);
 count ++;
}
```



```
//Remove the second last item (last but one) in the list. If the list has zero
// nodes or only one node, just return.
//For example, if the list is
// front → Java → C → C++ → Pascal → Python → null
// removeSecondLast() should change the list to
// front → Java → Cobol → C → C++ → Python → null
//As a second example, if the list is
// front → Java → C→ null
// removeSecondLast() should change the list to
// front → C → null

public void removeSecondLast(){
if (front == null ||
      front.getNext() == null)
                         return;
  Node<T>  curr = front;
  while (curr.getNext().getNext().getNext() != null)
                curr = curr.getNext();
  curr.setNext(curr.getNext().getNext());
  count --;
}
                    if (front.getNext().getNext() == null)
                         front = front.getNext();      //2 node case
```



Question 5. Suppose that the Unordered List class has been developed and the methods in the class are as shown below:

<u>**Unordered List methods**</u>

Constructors

`List<T>()`	**Constructs an empty ordered list**

Methods

Name	What it does	Header
`size`	`returns size of the list`	`int size()`
`isEmpty`	`returns true if list is empty`	`boolean isEmpty()`
`clear`	`clears the list`	`void clear()`
`add`	`adds an item to the front of the list`	`void add(T item)`
`first`	`gets the first entry`	`T first ()`
`next`	`gets the next entry`	`T next()`
`enumerate`	`scans the list and prints it`	`void enumerate()`
`contains`	`searches for a given item and returns true or false`	`boolean contains(T item)`
`remove`	`remove an item (first occurrence) from the list.`	`void remove(T item)`
`removeAll`	`remove all occurrences of a specified item from the list.`	`void removeAll(T item)`

*5A. **Using these methods***, implement a method that takes in two unordered lists as arguments to the method and ***creates and returns a third list*** that contains the elements in **list1** and in **list2**.

For example,
if list1 is {C++, C, C#, COBOL, Python, Ada}and
list2 is {C, Java, Python, Ada, Javascript}
the returned list is {C++, C, C#, COBOL, Python, Ada, Java, Javascript}
Order does not matter. You may also assume that the items are not repeated within each list.

```
public static<T> List<T> union(List<T> list1, List<T> list2)
{
    //continue
    List<T> list3 = new List<T>();
    T item = list1.first();
    while (item != null)
    {   list3.add(item);
        item = list1.next();
    }
    item = list2.first();
    while (item != null)
    {   if (!list3.contains(item))
                list3.add(item);
        item = list2.next();
    }
    return list3;
}
```

5B. ***Using these methods***, implement a method that takes in two unordered lists as arguments to the method and ***creates and returns a third list*** contains the elements in **list2** but not in **list1**.For example,
if list1 is {C++, C, C#, COBOL, Python, Ada}and
list2 is {C, Java, Python, Ada, Javascript}
the returned list is {Java, Javascript}
Order does not matter. You may also assume that the items are not repeated within each list.

```
public static<T> List<T> difference(List<T> list1, List<T> list2)
{
  //continue
     List<T> list3 = new List<T> ();
     T item = list2.first();
     while (item != null)
     {   if (!list1.contains(item))
                list3.add(item);
         item = list2.next();
     }
     return list3;
}
```

Question 6: Using the table, show the step by step process of how the binary search algorithm would work when you **search for the key 105** on the following array of integers. Note: not all the rows in the table may be filled.

	-5	-2	10	12	17	25	35	45	55	63	105
Index	0	1	2	3	4	5	6	7	8	9	10

lo	hi	mid	Key found?	Action
0	10	(0+10)/2 = 5	No	105 > 25 Go right
6	10	(6+10)/2 = 8	No	105 > 55 Go right
9	10	(9+10)/2 = 9	No	105 > 63 Go right
10	10	(10+10)/2 = 10	Yes	

Question 7: Using the table, show the step by step process of how the binary search algorithm would work when you **search for the key "apple"** on the following array of Strings. Note: not all the rows in the table may be filled.

	bonnet	bucket	face	garden	home	insurance	meal	orange	walk	young
Index	0	1	2	3	4	5	6	7	8	9

lo	hi	mid	Key found?	Action
0	9	(0+9)/2 = 4	No	apple < home Go left
0	3	(0+3)/2 = 1	No	apple < bucket Go left
0	0	(0+0)/2 = 0	No	apple < bonnet Go left
0	-1			

Stop. lo > hi
Key not found.

Question 8. Suppose that the Ordered List class has been developed and the methods in the class are as shown below:

```
Ordered List Methods
Constructor
OrderedList()               Constructs an empty unordered list
Methods
int size()                  Returns size of the list
isEmpty()                   Returns true if list is empty
void clear()                Clears the list
T get(int pos)              Gets the item at the specified index
T first ()                  gets the first entry
T next()                    gets the next entry
void enumerate()            scans the list and prints it
int binarySearch(T item)    searches for a given item.
                            returns position (index) if found
                            if not found returns a negative number
void insert(T item)         inserts the item at the correct position
void add(int pos, T item)   adds the given item at the specified index
void remove(T item)         remove the given item
void remove(int pos)        remove the item at the given index
```

8A. ***Using the methods in the class***, write a method that accepts an ordered list of Strings and ***displays the entries with indices 0, 2, 4, etc. in the list.***
For example, if the ordered list is:{A, B, C, D, E}
the method should display A C E
The items are indexed from 0.

Use the method header given below:

```
public static OrderedList<String> evenList(OrderedList<String>
list1)

{
      //continue your code below
    String item = list1.first();
    int i = 0;
    while (item != null)
    {
         if (i % 2 == 0)
              System.out.print(item + "\t");

         i++;
         item = list1.next();
    }

    System.out.println();
}
```

8B. Using the methods in this class, write a method that accepts an ordered list of Strings and a key and returns a new ordered list with all the elements (alphabetically) smaller than the key. For example, if the ordered list is:
{A, C, F, G, Z} and the key is F, return the ordered list {A, C}
Note: The target key may or may not be in the ordered list.

```
public static OrderedList<String> subList(OrderedList<String> list1,
String k1)
{
  //continue your code below
OrderedList<String> result = new OrderedList<String>();
int i = 0;
String check = list1.first();
while (i < list1.size() && check.compareTo(k1) < 0)
{
        result.insert(check);
        check = list1.next();
        i++;
}
  return result;
}
```