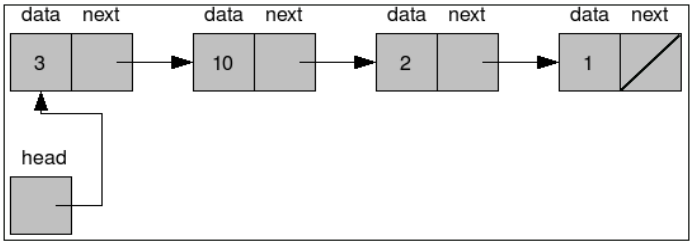
Reference Based Linked Lists

15 Points as lab – See Canvas for due date.

Linked List Starter Kit

A reference-based singly linked list is a basic data structure in computer science. Proficiency with linked lists is part of all CS tracks and represents a substantial portion of the Advancement Programming Exam.



This code listing and exercise gives you practice with designing, coding and testing a reference-based linked list.

Tasks in this exercise:

1. Create an ADT interface
2. Create a Node class (nested, if you like)
3. Create a ListReferenceBased class
4. Create a ListTester class

---

1. Create an ADT interface that specifies method signatures for:

- isEmpty method
- size method
- addNode method that accepts a new item only
- addNode method that accepts a new item and an index
- removeItem method that accepts an index
- removeAll method

---

2. Create a Node class with:

- Instance field (type Object) – item
- Instance field (type Node) – next
- Constructor method that accepts an object only
- Constructor method that accepts an object and a nextNode
- Set and get methods for item
- Set and get methods for next

(Note: If you elect to create your Node class as nested, it will reside within the ListReferenceBased class.)

---

2. Create a ListReferencedBased class that implements:

Your ListInterface interface
All the required methods
A private method – find – that accepts an index and returns a node

---

3. Create a ListTester class to drive your linked list:

```
public class ListTester
{
	public static void main(String[] args)
	{
		ListReferenceBased myList = new ListReferenceBased();

		for (int i = 1; i < 7; i++)
		{
			double temp;
			temp = 1.23456 * i;
			Double d = new Double(temp);
			myList.addNode(d);
		}

		System.out.println(myList);

		myList.removeNode(1);
		System.out.println(myList);

		Double d = new Double(99.0);
		myList.addNode(1, d);
		System.out.println(myList);

		myList.addNode(2, new Double(68));
		System.out.println(myList);
	}
}
```

For lab credit: Submit your code listings on paper, in class. See Canvas for due date.

ListInterface interface:

```
public interface ListInterface
{
	public int getSize();

	public boolean isEmpty();

	public void addNode(Object newItem);

	public void addNode(int index, Object newItem);

	public void removeNode(int index);

	public void removeAll();

	public String toString();

}
```

Node class:

```
public class Node
{
	private Object item;
	private Node next;

	public Node(Object newItem)
	{
		this.item = newItem;
		this.next = null;
	}

	public Node(Object newItem, Node nextNode)
	{
		this(newItem);
		this.next = nextNode;
	}

	public Object getItem()
	{
		return this.item;
	}

	public void setItem(Object newItem)
	{
		this.item = newItem;
	}

	public Node getNext()
	{
		return this.next;
	}

	public void setNext(Node nextNode)
	{
		this.next = nextNode;
	}

	public String toString()
	{
		return this.getItem().toString();
	}
}
```

ListReferenceBased class:

```
public class ListReferenceBased implements ListInterface
{
	private Node head;
	private int numItems;

	public ListReferenceBased()
	{
		head = null;
		numItems = 0;
	}

	@Override
	public boolean isEmpty()
	{
		return numItems == 0;
	}

	@Override
	public int getSize()
	{
		return numItems;
	}

	@Override
	public void addNode(Object newItem)
	{
		Node newNode = new Node(newItem);
		Node curr;

		if(isEmpty())
		{
			this.head = newNode;
		}
		else
		{
			for(curr = head;
			    curr.getNext() != null;
			    curr = curr.getNext());
			curr.setNext(newNode);
		}

		numItems++;
	}

	@Override
	public void addNode(int index, Object newItem)
	{
		Node newNode = new Node(newItem);
		Node prev;

		if(index == 1)
		{
			newNode.setNext(head);
			this.head = newNode;
		}
		else
		{
			prev = find(index - 1);
			newNode.setNext(prev.getNext());
```

```
                prev.setNext(newNode);
            }

            numItems++;
        }

        @Override
        public void removeNode(int index)
        {
            // Special case...
            if (index == 1)
            {
                head = head.getNext();
            }
            else
            {
                Node prev = find(index - 1);
                Node curr = prev.getNext();
                prev.setNext(curr.getNext());
            }
            numItems--;
        }

        @Override
        public void removeAll()
        {
            this.head = null;
            numItems = 0;
        }

        private Node find(int index)
        {
            Node curr = head;

            for (int skip = 1; skip < index; skip++)
            {
                curr = curr.getNext();
            }

            return curr;
        }

        @Override
        public String toString()
        {
            String result = "";

            for(Node curr = this.head; curr != null; curr = curr.getNext())
            {
                result = result + curr.getItem().toString() + "\n";
            }
            return result;
        }
}
```