ALGONQUIN COLLEGE

CST8130 – Data Structures

Professor : Dr. Anu Thomas
Email: thomasa@algonquincollege.com
Office: T314

Introduction to Linked Lists

Memory Management Issues of Arrays

Assume that we have a class *DATA that* needs 100 bytes of memory for an object. We are creating n DATA objects.

```
DATA [ ] list = new DATA[n];
for (int i=0; i<n; i++)
list[i] = new DATA ();
```

*Needs **block of memory for array of n references**…and then n blocks of 100 bytes for each DATA object.*

Arrays won’t automatically grow or shrink.

Linked Lists

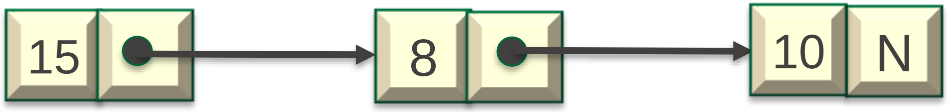
- Help us make use of more modular memory
  - We may not know how many items we need (notice in above example, we still need to know “n”)
- A dynamic data structure grow and shrink at run time
- Collection of data items “linked up in a chain”— insertions and deletions can be made anywhere in a linked list.
- Consists of “nodes” – which are data object reference and a reference to the next item in the list in a block of memory
- Start with a reference to “head” – first item in the list
- Last item in the list “tail” points to NULL

Linked Lists (contd.)

- Typically, a program accesses a linked list via a reference to its first node.
- The program accesses each subsequent node via the link reference stored in the previous node.
- By convention, the link reference in the last node of the list is set to null to indicate “end of list.”
- A linked list is appropriate when the number of data elements to be represented in the data structure is *unpredictable*.
- Linked lists become full only when the system has *insufficient memory* to satisfy dynamic storage allocation requests.

Singly Linked Lists

- Linked list nodes normally are *not stored contiguously* in memory. Rather, they are logically contiguous.



- This diagram presents a singly linked list—each node contains one reference to the next node in the list.
- Often, linked lists are implemented as *doubly linked lists*—each node contains a reference to the next node in the list *and* a reference to the preceding one.

Self-Referential Classes

A self-referential class contains an instance variable that refers to another object of the same class type.

For example, the generic class declaration

```
class Node<T>
{
   private T data;
   private Node<T> nextNode; // reference to next node

   public Node(T data) { /* constructor body */ }
   public void setData(T data) { /* method body */ }
   public T getData() { /* method body */ }
   public void setNext(Node<T> next) { /* method body */ }
   public Node<T> getNext() { /* method body */ }
} // end class Node<T>
```

declares class Node, which has two private instance variables—data (of the generic type T) and Node<T> variable nextNode.

Code….. LLNode class

```
public class LLNode {
        private String data;
        private LLNode next;

        public LLNode() { this.data = null;      this.next = null;  }

        public LLNode (String newData) { this.data = newData);      this.next = null;  }

        public void updateNode (LLNode nextOne) { this.next = nextOne; }

        public String toString () {  return  this.data;   }

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

}
```

Code – List class

```
public class LList {
    private LLNode head;

    public LList() {  head = null; }
    public void addAtHead (String newData) {
        LLNode newNode = new LLNode (newData);
        newNode.updateNode(head);
        head = newNode;
    }

    public void display() {
        LLNode temp = head;
        while (temp != null) {
            System.out.println (temp);
            temp = temp.getNext();
        }
    }
}
```

Code – method main

```
public static void main(String[] args) {
            LList list = new LList();

            list.addAtHead(“Anu");
            list.addAtHead(“Thomas");
            System.out.println("The list is ");
            list.display();
      }
```

```
The list is
Thomas
Anu
```

Delete from Head

What would the code to delete from head looks like???

Delete from Head

In LList class,

```
public LLNode deleteAtHead ( ) {
        LLNode removedOne = head;
        head = head.getNext();
        return removedOne;
}
```

In main method,

```
    LLNode removedOne = list.deleteAtHead();
    System.out.println("After delete, the list is ");
    list.display();
    System.out.println("The one deleted is…" + removedOne);
```

Search and Delete

What would the code to delete a particular String looks like?

Search and Delete

We will do this as Lab 5 !!!

Questions?