LINKED LISTS

Like an array, a linked list is a data abstraction representing a list of items. Unlike an array, the items in the linked list are not necessarily kept in consecutive memory locations.

Each linked list item is referred to as a list node, which contains two sections:

![List Node Diagram]

Application data contains whatever data is needed for the specific problem being solved; the list data contains housekeeping information needed to maintain the list itself. At a minimum the list data indicates the location of the next node in the linked list, which is often referred to as a pointer or a link.

**Example**

Here is a list of students and their heights. The links are drawn as arrows. Tom is the first student in the list, followed by Dick, followed by Harry, who is the last student in the list.
The order of the nodes indicated by the links is called its **logical order** so as to emphasize that the nodes’ physical order is irrelevant. To access the first node in the logical order, we need a special variable that points to it. This is commonly named **first**.

**Example**
Here is a linked list of students, logically ordered alphabetically by name, beginning with Dick, who is pointed to by variable **first**.

Here is a linked list of students in the same physical ordering as that above, but logically ordered by student height. **first** points to Tom, the shortest student. Harry is next and Dick, the tallest student, is last.
An appropriate Java class for building linked list nodes requires fields for the application data and the list data.

---

**Java Linked List Node Class**

```java
public class list node class {
    fields for application data
    public list node class next;
}
```

*list node class* can be whatever class identifier you choose. You’ll use this to declare reference variables that point to list nodes.

The field named *next* is a reference variable containing the address of the next node in the linked list. Its data type must be the same as *list node class*.

---

**Example**

Here is a Java class appropriate for storing the linked list of students and their heights.

```java
public class Student {
    public String name;
    public int feet, inches;
    public Student next;
}
```

You create the nodes of the linked list by building the link objects using the `new` operator, which works the same way as with any object.

---

**Example**

Given the *Student* class above, these Java statements build the two objects shown:

```java
Student tom = new Student();
Student dick = new Student();
```

![Diagram](image-url)
You fill the nodes with application data using the dot (.) notation as usual.

**Example**
These Java statements fill the two *Student* objects with data.

```java
tom.name = "Tom";
tom.feet = 5;
tom.inches = 6;
dick.name = "Dick";
dick.feet = 6;
dick.inches = 1;
```

![Diagram of linked list nodes]

Link the nodes together.

**Example**
This statement makes the node for Tom point to the node for Dick.

```java
tom.next = dick;
```

![Diagram of linked list with Tom pointing to Dick]

Finally define the `first` variable so that it points to the first node in the list. The individual reference variables to the two nodes are no longer needed since all nodes can be accessed starting with the `first` variable and following the links.
Example
This statement makes first point to the node for Tom.

```
Student first = tom;
```

The code for building linked lists can be simplified by implementing clever explicit constructors in the list node class. Such constructors can automatically initialize the instance variables within the node that contain the application data and the link.

Example
This class for the student linked list adds two overloaded constructors. The first (lines 7‒13) initialize the four instance variables of the object to the arguments passed to it. The second constructor (lines 15‒18) calls the first constructor in such a way that the instance variable next is initialized to null.

```
public class Student {
    public String name;
    public int feet, inches;
    public Student next;

    public Student( String n, int f, int i, Student nxt ) {
        name = n;
        feet = f;
        inches = i;
        next = nxt;
    }

    public Student( String n, int f, int i ) {
        this( n, f, i, null );
    }
}
```
Example
Given the Student class above, you can build the linked list containing Tom and Dick using this code:

```java
Student first = new Student( "Tom", 5, 6 );
first.next = new Student( "Dick", 6, 1 );
```

Or, this:

```java
Student dick = new Student( "Dick", 6, 1 );
Student first = new Student( "Tom", 5, 6, dick );
```

To get a good presentation of the linked list in jGRASP’s Viewer, you should add a toString method to your list node class that displays the data that you’re interested in tracking during a debugging session.

Example
This class adds an appropriate toString method to the student list node class (lines 20–24).

```java
1  public class Student
2  {
3      public String name;
4      public int feet, inches;
5      public Student next;
6      
7      public Student( String n, int f, int i, Student nxt )
8      {
9          name = n;
10         feet = f;
11         inches = i;
12         next = nxt;
13      }
14      
15      public Student( String n, int f, int i )
16      {
17          this( n, f, i, null );
18      }
19      
20      public String toString( )
21      {
22          return name + ", "$ + feet + ", "$ + inches + "'";
23      }
24  }
```
Enter the Student class on page 3 into jGRASP, save it to a file, compile it and perform the following experiments using jGRASP’s Interactions pane, Workbench pane and Viewers.

1. In jGRASP’s Interactions pane, enter and execute the statement:

   ```java
   Student tom = new Student();
   ```

   In jGRASP’s Workbench pane, find the entry for tom and click the so that it becomes . What are the values of the object’s instance variables?

2. In jGRASP’s Interactions pane, enter and execute the statement:

   ```java
   Student dick = new Student();
   ```

   In jGRASP’s Workbench pane, find the entry for dick and click the so that it becomes . What are the values of the object’s instance variables?

3. In jGRASP’s Workbench pane, open jGRASP’s Viewer for the two objects by clicking and dragging the blue squares next to the reference variables (e.g. ). If it is not already then set the Viewer to Basic (see the picture to the right).

4. In jGRASP’s Interactions pane, enter and execute these statements:

   ```java
   tom.name = "Tom";
tom.feet = 5;
tom.inches = 6;
dick.name = "Dick";
dick.feet = 6;
dick.inches = 1;
   ```

   What are the values of the instance variables of objects tom and dick?

5. In jGRASP’s Interactions pane, enter and execute the statement:

   ```java
   Student first = tom;
   ```

   In jGRASP’s Workbench pane, open jGRASP’s Viewer for first by clicking and dragging its blue square ( ). If it is not already then set the Viewer to Presentation – Structure Identifier.
6. In jGRASP’s Interactions pane, enter and execute the statement:

   \[ \text{tom.next = dick;} \]

   What is shown by the Viewer for \textit{first}? It should resemble in structure (if not in detail) the linked list pictured at the top of page 5.

---

Enter the \textit{Student} class on page 6 into jGRASP, save it to a file, compile it and perform the following experiments using jGRASP’s Interactions pane, Workbench pane and Viewers.

7. In jGRASP’s Interactions pane, enter and execute this statement:

   \[ \text{Student first = new Student( "Tom", 5, 6 );} \]

   In jGRASP’s Workbench pane, open jGRASP’s Viewer for \textit{first} by clicking and dragging its blue square (first $\rightarrow$). If it is not already then set the Viewer to Presentation – Structure Identifier.

8. In the Viewer for \textit{first}, click the Configure button ( ) and set the Value Expressions text box to \texttt{_node_.toString( )}:

   Value Expressions (separate with #)
   \[ \texttt{_node_.toString( )} \]

9. In the Viewer for \textit{first}, adjust the Scale and Width slider controls to show the entire node contents. The Viewer should appear similar to this:
10. In jGRASP’s Interactions pane, enter and execute this statement:

```java
first.next = new Student( "Dick", 6, 1 );
```

The Viewer for `first` should now show the first two links of this list:

```
first
Tom 5'6"
Dick 6'1"
Harry 5'9"
```

11. In jGRASP’s Interactions pane, enter and execute the statements needed to complete the linked list shown in exercise 10.

Start a new interaction by clicking the End and Clear buttons. Perform the following experiments.

12. In jGRASP’s Interactions pane, enter and execute the statements needed to complete the following linked list:

```
first
Tom 5'6"
Dick 6'1"
Harry 5'9"
```

Open a jGRASP Viewer for `first` and adjust its settings so that the list is displayed like shown in exercise 9.

13. In jGRASP’s Interactions pane, enter and execute the statements needed to transform the linked list into that which follows without destroying any nodes (just reset the first variable and the next links):

```
first
Tom 5'6"
Dick 6'1"
Harry 5'9"
```