Methodic and Practical Foundations of Computer Science 1

14-Data_Structure

Sayed Ahmad Sahim

Kandahar University
Computer Science Faculty

algprog17@gmail.com

December 10, 2017

Objectives

1. Data Structure and Algorithms
2. Overview on Algorithms
3. Data Structure vs. Algorithm
4. Data Structure Solving Problems
5. Some Initial Definitions
6. The Advantages of lists
7. Linked List
8. What is link?
9. Linked List
10. Singly Linked List
11. Linked List vs. Array
12. List Information
13. Singly Linked List
14. Common Operations on Lists
15. addFirst() Implementation
16. adding to a specified position in the list
17. getElementAT()
18. deleteFirst()

Data Structure and Algorithms

- A data structure is an arrangement of data in a computers memory (or sometimes on a disk).
- Data structures include linked lists, stacks, binary trees, and hash tables.
- Algorithms manipulate the data in these structures in various ways, such as inserting a new data item, searching for a particular item, or sorting the items. Genrally data structure is the Arrangement of data in a computers memory.

Overview on Algorithms

- An algorithm can be thought of as the detailed instructions for carrying out some operation For most data structures, you must know how to do the following:
    1. Insert a new data item.
    2. Search for a specified item.
    3. Delete a specified item.

Data Structure vs. Algorithm

- Data structure is concerned with holding data in memory efficiently.
- Algorithms tell you how to store, retrieve, search or alter data in a data structure.

Data Structure Solving Problems

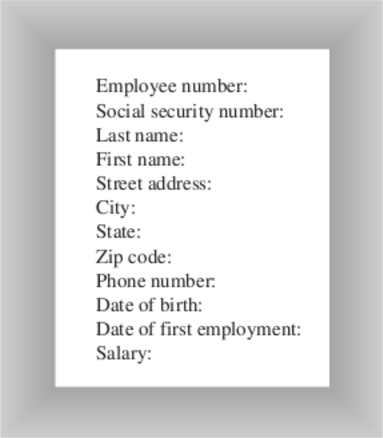
- What sort of problem can you solve with using a Data Structure:
    - Real-world Data Storage: real-world data describes physical entities external to the computer.
    - Some examples are a personnel record that describes an actual human being, an inventory record that describes an existing car part or grocery item, and a financial transaction record that speak about bills.

Some Initial Definitions

1. **Datafile**: refers to a collection of similar data items. As an example, if you create an address book the collection of cards you have created constitutes a datafile.
2. **Record**: A record includes all the information about some entity, in a situation in which there are many such entities. A record might correspond to a person in a **personnel file**, **a recipe in a cookbook file**.
3. **Field**: A record is usually divided into several fields. A field holds a particular kind of data like a persons name.

Record Example

- A Record with multiple Fields



Employee number:
Social security number:
Last name:
First name:
Street address:
City:
State:
Zip code:
Phone number:
Date of birth:
Date of first employment:
Salary:

The Advantages of lists

- Lists can dynamically grow and shrink
- Lists are rather easy to implement
- Adding/removing elements to/from the beginning/end of a list can be fast

Linked List

1. Second most commonly used storage structure after arrays
2. Suitable to use in many general-purpose databases.
3. Size of a list can be increased during execution
4. To delete a list item does not need much efforts

What is link?

- Each data item is fixed in a link.
- A link is an object of a class usually called Link.
- Each link object contains a reference called next, and one or more data items
- The last list element is always connected to null.

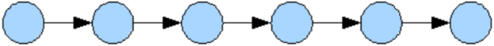
Linked List

There are two types of linked list:

1. Singly linked list
2. Double linked list

Singly Linked List

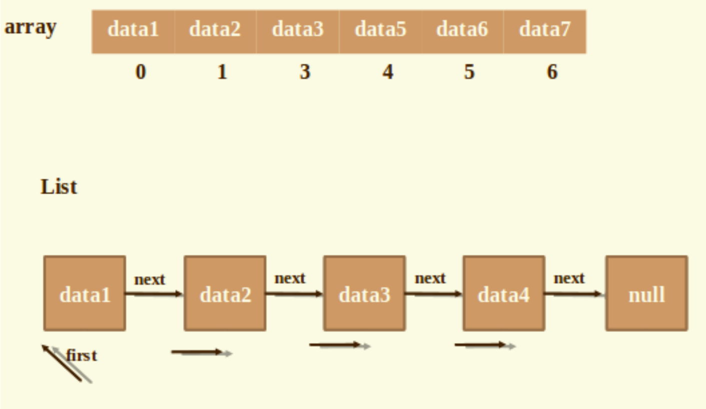
- Singly linked lists are a **linear data structure**.
- Each element has at most one successor, and the last element has no successor.



Linked List vs. Array

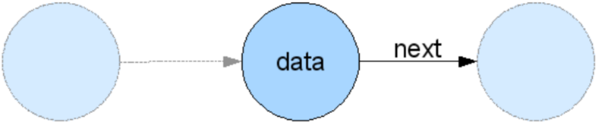
1. Each item of an array occupies a position in the memory. With indexes we can reach the item.
2. Each list element is stored somewhere in the memory and they are connected to each other through a reference.
3. To reach a list element you have to move along the chain of element

Linked List vs. Array



List Information

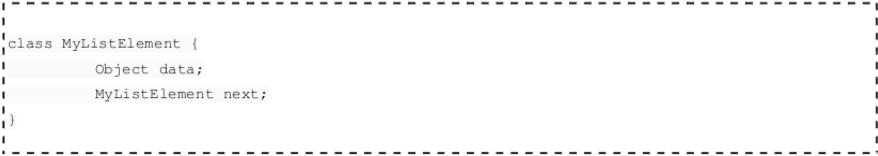
- List elements must store the following information:



- A reference to the data that is represented by the list element
- A reference to the next list element (if it exists)
- Next is a self-refrential variable
- Each Node sotres: element and link to the next node
- It is possible to store additional information in the list elements (such as the predecessor) for convenience.

Singly Linked List

- In Java, each list element is an object



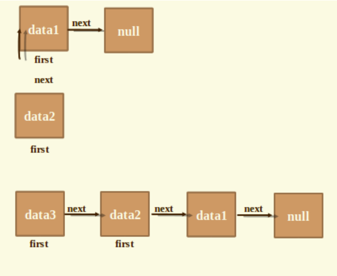
```
class MyListElement {
         Object data;
         MyListElement next;
}
```

- The list elements can then be organized into a list, which contains a reference to the first element and exposes methods to manipulate the list.

Common Operations on Lists

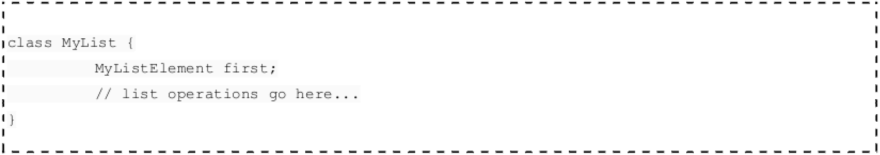
- **addFirst();** to add in the beginning
- **addLast();** to add at the end
- **addAtPosition();** to add in a specified position
- **search();** to search for an element
- **deleteFirst();** to delete the first element
- **deleteLast();** to delete the last element
- **deleteAtPosition();** to delete a specific element
- **printList();** to print list elements

addFirst(data);

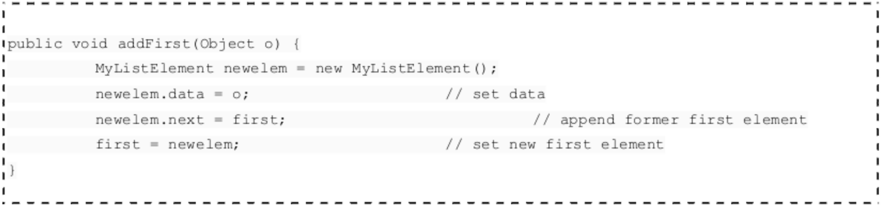


- **addFirst():** Inserts a new element at the beginning

addFirst() Implementation

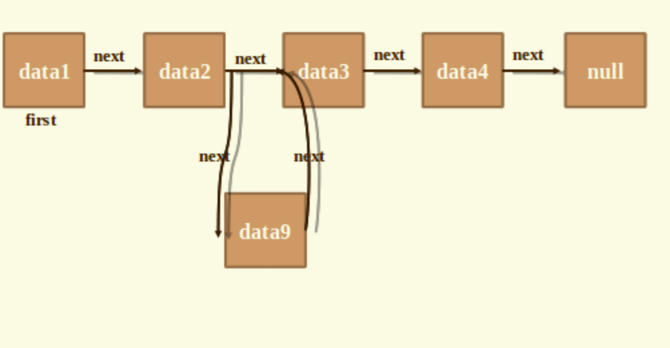


```
class MyList {
        MyListElement first;
        // list operations go here...
}
```



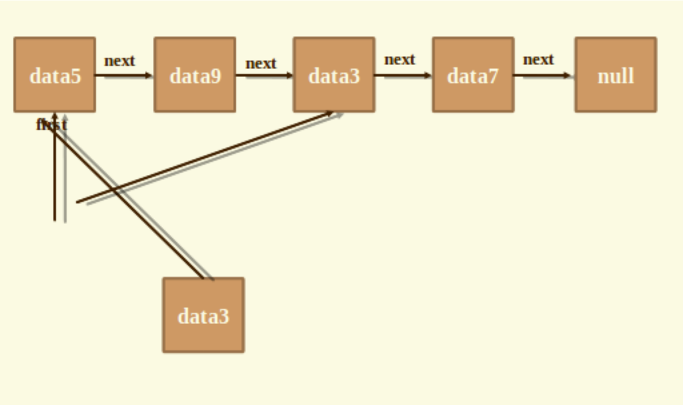
```
public void addFirst(Object o) {
        MyListElement newelem = new MyListElement();
        newelem.data = o;                    // set data
        newelem.next = first;                        // append former first element
        first = newelem;                     // set new first element
}
```

addAtPosition(data,position)

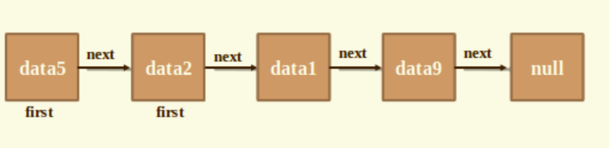


getElementAT()

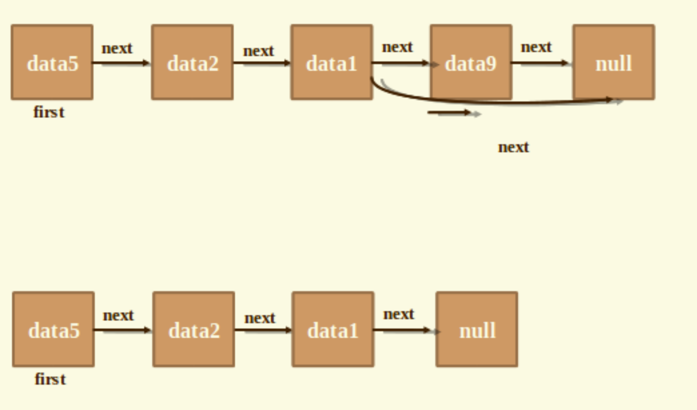
- getElementAT(): Returns the element at the specified position in the list.



deleteFirst()

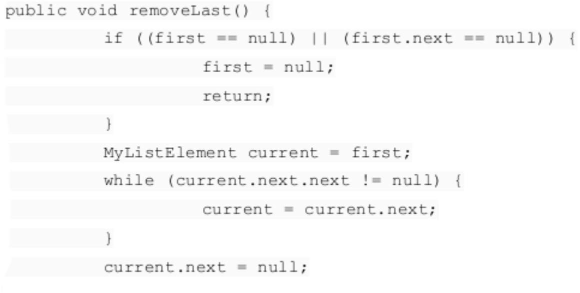


deleteLast();


data5
next
data2
next
data1
next
data9
next
null
first
next
data5
next
data2
next
data1
next
null
first

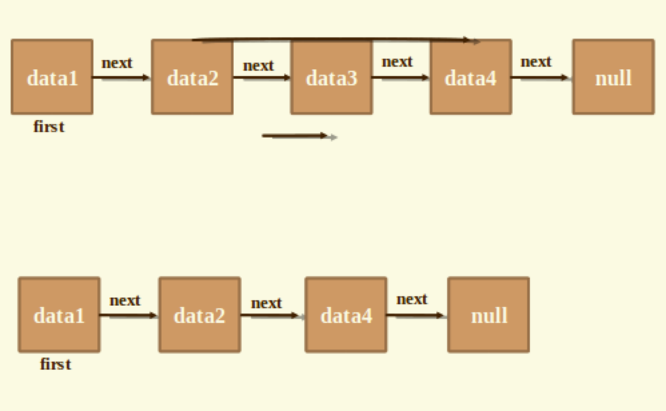
deleteLast();

- Certain operations, such as removing the last element of a list, require going through the entire list. This is inefficient.



```
public void removeLast() {
        if ((first == null) || (first.next == null)) {
                first = null;
                return;
        }
        MyListElement current = first;
        while (current.next.next != null) {
                current = current.next;
        }
        current.next = null;
```

deleteAtPosition();



Question