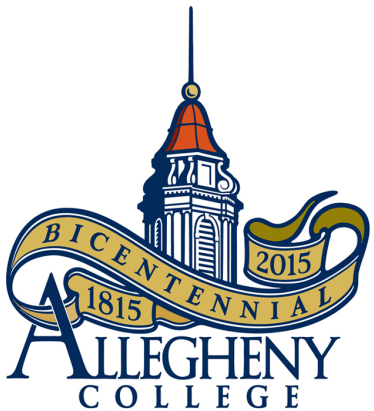
CS101 - Data Abstraction

DS Basics - Module2

Aravind Mohan

Allegheny College

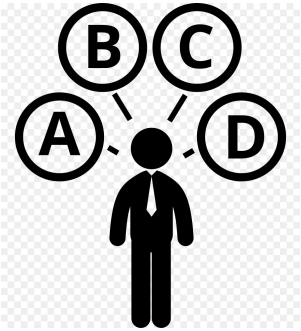
February 25, 2020



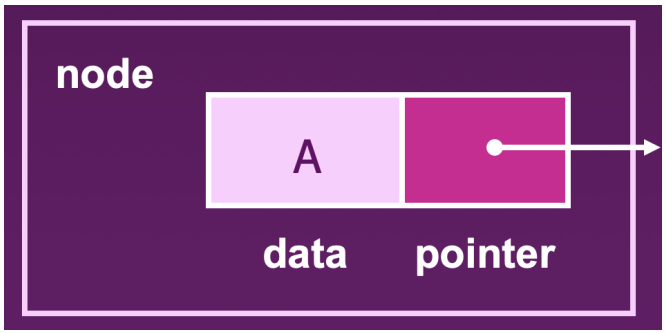
Pros and Cons of Arrays

- **Contiguous** storage of elements in memory.
- **Estimate** of maximum size of list is required.
- **Waste** space.
- **Better** search.
- **Worst** insert and delete.

An alternative Data Structure



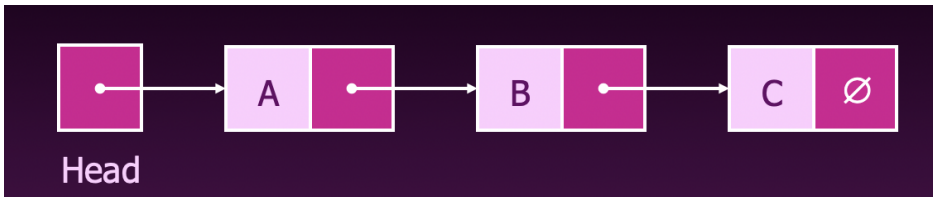
- **Linked List:** A series of connected data items called Nodes.
- A **Node** contains at least a piece of data item (of any type) and a link (pointer) to the next node in the list.



Linked List

Three Properties

- **All nodes** should be linked to each other
- **Head** pointer to the first node
- **Last** node points to null



Pros and Cons of Linked List

- **Not** stored contiguously. Random distribution of data items in memory.
- **Better** space management.
- **Worst** search performance.
- **Better** insert and delete performance.

Core Operations on Linked List

- **IsEmpty** determine whether or not the list is empty
- **InsertNode** inserts a new node at a particular position
- **FindNode** find a node with a given value
- **DeleteNode** delete a node with a given value
- **DisplayList** print all the nodes in the list

Implementation of a Node

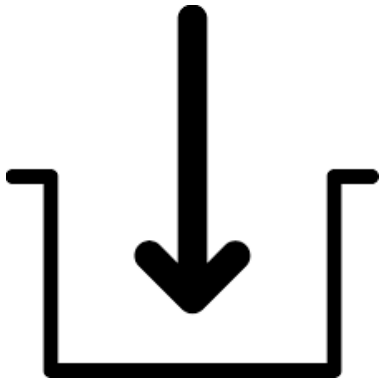
```
public class Node {
  private int data;
  private Node next;
  protected Node(){} // default constructor
  protected Node(int data) {
    this.data = data;
    this.next = null;
  }
  protected Node(int data, Node next) {
    this.data = data;
    this.next = next;
  }
  /* add getters and setters */

}
```

Implementation of Link class

```
public class Links extends Node {
  private Node head;
  private int size;
  public Links() {
    head = null;
    size = 0;
  } //constructor
  public int findNode(int index) {} //get
  public void insertNode(int index, int data) {
    if(size<1) {
      /* add node to the front */
    } else {
      /* add node to the middle or end */
    }
  } //add
  public void deleteNode(int index) {} //remove
  public void displayList() {} //display
  public int isEmpty() {} // use size
}
```

InsertNode implementation

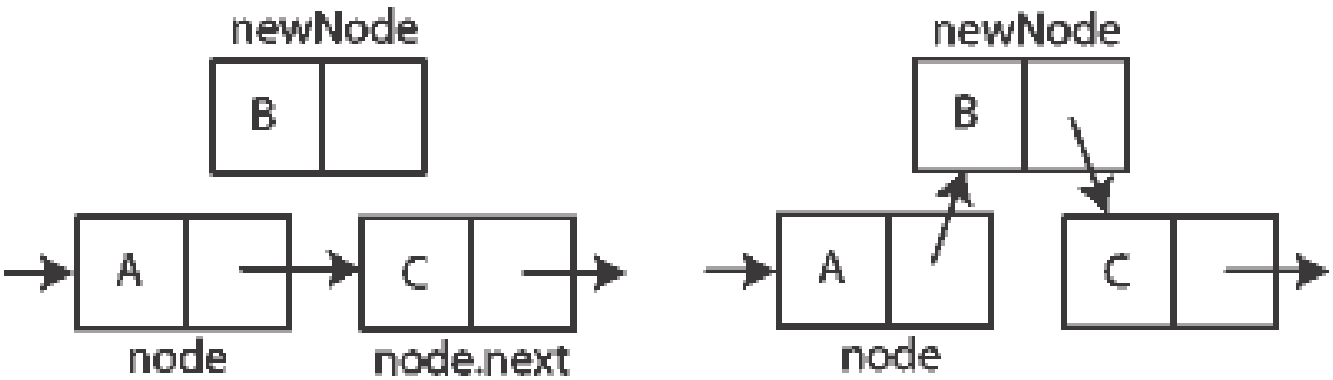


- **Case1** Insert as the first node
- **Case2** Insert in the middle or at the end of the list

InsertNode implementation

How to handle inserts?

- **Locate** the position to insert
- **Allocate** memory for the new node
- **Point** the new node to its successor
- **Point** the new node's predecessor to the new node



FindNode implementation

How to handle search?

- **Search** for a node with a given index in the list
- **Return** the value if such a node exist. Otherwise return not found or -1.

DeleteNode implementation

How to handle remove?

- **Case1** Delete the first node
- **Case2** Delete the middle or at the end of the list

DeleteNode implementation

How to handle remove?

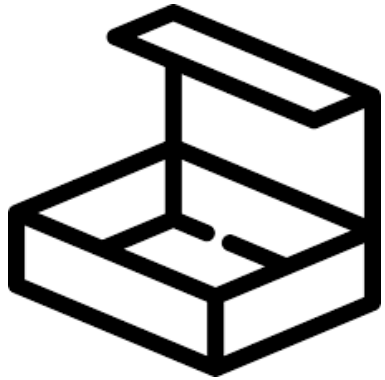
- **Find** the desirable node similar to FindNode
- **Release** the memory occupied by the found node
- **Set** the next link of the predecessor of the found node to the successor of the found node

DisplayNode implementation

How to handle print?

- **Iterate** through the list (based on size) and print the data of all the elements
- **Print** the total number of nodes in the list

IsEmpty implementation



How to handle size?

- **Manage** node inserts by using an entity called Size
- **Size** should be incremented during every node insert in the list
- **Empty** list is determined based on the list size. If the size is 0 then the list is empty. Otherwise list is not empty.

Reading Assignment

GT Chapter 3 - 3.1 and 3.2

More on variations of Linked List.

Questions?

Please ask if there are any Questions!