Lab Week - 2
Singly Linked List

The purpose of this lab session is to acquire skills in working with singly-linked lists.

Activity Outcomes:

This lab teaches you the following topics:

- Creation of singly linked list
- Insertion in a singly linked list
- Deletion from the singly linked list
- Traversal of all nodes

1) Useful Concepts

A *list* is a finite ordered set of elements of a certain type. The elements of the list are called *cells* or *nodes*. A list can be represented *statically*, using arrays or, more often, *dynamically*, by allocating and releasing memory as needed. In the case of static lists, the ordering is given implicitly by the one-dimension array. In the case of dynamic lists, the order of nodes is set by *pointers*. In this case, the cells are allocated dynamically in the heap of the program. Dynamic lists are typically called *linked lists*, and they can be singly- or doubly-linked.

The structure of a node may be:

```
class Node
  {
   int  data;

  Node next=NULL;   // link   to   next   node, assigned NULL so that should not point garbage
  };
```

A singly-linked list may be depicted as in Figure 1.1.

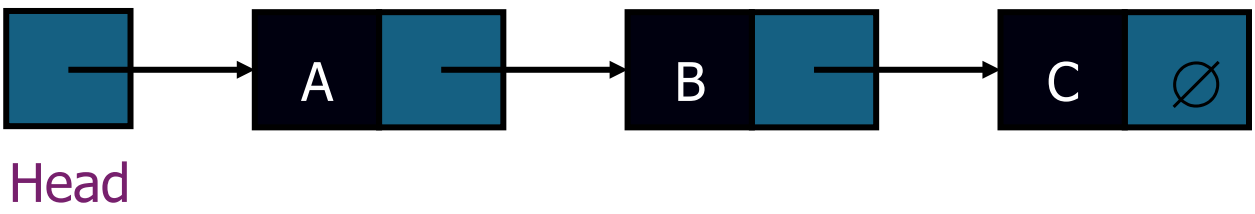

Figure 1.1.: A singly-linked list model.

2) Lab Activities

i. Insert at the end of the list
ii. Insertion at the start of the linked list
iii. Accessing the nodes of a linked list
iv. Insert after specific value
v. Deleting the first node from single linked list
vi. Deleting the last node from single linked list
vii. Deleting a node given by user
viii. Complete deletion of single linked list

NOTES:

When implementing a linked list, different cases need to be handled depending on the operation you are performing. Below are the specific cases for each of the operations you mentioned.

Insert at Start

1. **Empty list**:
    - If the list is empty, both head and tail should point to the new node.
2. **Non-empty list**:
    - The new node should be set as the new head, and its next should point to the current head.

Insert at End

1. **Empty list**:
    - If the list is empty, both head and tail should point to the new node.
2. **Non-empty list**:
    - The new node is added after the current tail, and tail is updated to point to this new node.

Insert After Some Value

1. **Empty list**:
    - If the list is empty, insertion is not possible, as there's no node to insert after.
2. **Value found**:
    - Traverse the list to find the node containing the specific value. Insert the new node after that node by adjusting the next pointers.
3. **Value not found**:
    - If the value is not found, inform the user that the insertion could not be completed.

Delete at Start

1. **Empty list**:
    - If the list is empty, deletion is not possible.
2. **Single-node list**:
    - If there's only one node in the list, both head and tail should be set to null after deleting.
3. **Multiple-node list**:
    - Move the head to point to the next node, effectively removing the first node.

Delete at End

1. **Empty list**:
    - If the list is empty, deletion is not possible.

2. **Single-node list**:
    - If there's only one node, set head and tail to null.
3. **Multiple-node list**:
    - Traverse the list to find the second-to-last node, update its next to null, and update tail to point to this node.

Delete Some Value

1. **Empty list**:
    - If the list is empty, deletion is not possible.
2. **Value is the head node**:
    - If the head contains the value, update the head to the next node. If the head was the only node, set both head and tail to null.
3. **Value found**:
    - Traverse the list to find the node before the one with the specific value, and update its next to bypass the node containing the value.
4. **Value not found**:
    - If the value is not found, inform the user that the value is not in the list.

Delete the Whole List

1. **Empty list**:
    - If the list is already empty, no action is required.
2. **Non-empty list**:
    - Set both head and tail to null, effectively removing all nodes. Optionally, traverse the list and explicitly remove each node to release memory if needed.