CAM2003C - Data Structures and Algorithms with C and C++

Lab Exercise: Singly Linked Lists & Doubly Linked List in C and C++

By the end of this lab, students will be able to implement all fundamental operations on Singly and Doubly Linked Lists in C/C++.

Singly Linked List

Perform the followings

Q1. Write the code snippet to define a Node structure for a Singly Linked List in C and C++.

C

```
struct Node {
   // your code here
};
```

C++

```
class Node{
   // your code here
};
```

Q2. Write function prototypes for the following operations:

- Create a new node
- Insert at beginning
- Insert at end
- Insert after a specific position
- Delete from beginning
- Delete from end
- Delete a specific node
- Search an element
- Display the list

Q3. Read the following Algorithms for SLL Operations and implement them in C and C++

Algorithm 1: Create Node

Algorithm CreateNode(value)

1. Allocate memory for newNode
2. Set newNode→data ← value
3. Set newNode→next ← NULL
4. Return newNode

Algorithm 2: Insert at Beginning

Algorithm InsertAtBeginning(head, value)
1. Create newNode using CreateNode(value)
2. Set newNode→next ← head
3. Set head ← newNode
4. Return head

Algorithm 3: Insert at End

Algorithm InsertAtEnd(head, value)
1. Create newNode using CreateNode(value)
2. If head is NULL:
   a. Return newNode
3. Set temp ← head
4. While temp→next ≠ NULL:
   a. temp ← temp→next
5. temp→next ← newNode
6. Return head

Algorithm 4: Insert at Position (0-based index)

Algorithm InsertAtPosition(head, value, pos)
1. If pos = 0:
   a. Return InsertAtBeginning(head, value)
2. Create newNode using CreateNode(value)
3. Set temp ← head
4. Loop (i = 0 to pos-2):
   a. If temp = NULL: return head (Invalid position)
   b. temp ← temp→next
5. newNode→next ← temp→next
6. temp→next ← newNode
7. Return head

Algorithm 5: Delete from Beginning

Algorithm DeleteFromBeginning(head)
1. If head = NULL: return NULL
2. Set temp ← head
3. Set head ← head→next
4. Free temp
5. Return head

Algorithm 6: Delete from End

```
Algorithm DeleteFromEnd(head)
1. If head = NULL: return NULL
2. If head→next = NULL:
   a. Free head
   b. Return NULL
3. Set temp ← head
4. While temp→next→next ≠ NULL:
   a. temp ← temp→next
5. Free temp→next
6. temp→next ← NULL
7. Return head
```

Algorithm 7: Search Element

```
Algorithm Search(head, key)
1. Set temp ← head
2. While temp ≠ NULL:
   a. If temp→data = key: return true
   b. temp ← temp→next
3. Return false
```

Algorithm 8: Display List

```
Algorithm Display(head)
1. Set temp ← head
2. While temp ≠ NULL:
   a. Print temp→data
   b. temp ← temp→next
```

Q3. Write the code snippet for each operation one by one using the algorithm provided above. Use the structure:

```
struct Node* insertAtBeginning(struct Node* head, int value) {
   // code here
}
```

Repeat this for each operation.

Q4. Write a main() function that:

- Declares and initializes the linked list
- Displays a menu with choices for each operation
- Takes user input for choice
- Calls the appropriate function accordingly

Sample menu format:

```
1. Insert at Beginning
2. Insert at End
```

3. Insert at Position
4. Delete from Beginning
5. Delete from End
6. Delete a Node
7. Search an Element
8. Display List
9. Exit

Q5. Combine all code snippets into a single working file for Singly Linked List.
Test each operation thoroughly with sample input/output.

Q6. Create a program for the implementation of a Singly Linked List with all operations in C++.

Doubly Linked List

Q7. Write the code snippet to define a Node structure for a Doubly Linked List in C/C++.

Q8. Write function prototypes for the following operations:

- Create a new node
- Insert at beginning
- Insert at end
- Insert after a specific position
- Delete from beginning
- Delete from end
- Delete a specific node
- Search an element
- Display forward
- Display backward

Q9. Read the following Algorithms for DLL Operations and implement them in C

Algorithm 1: Create Node

Algorithm CreateDNode(value)
1. Allocate memory for newNode
2. Set newNode→data ← value
3. Set newNode→prev ← NULL
4. Set newNode→next ← NULL
5. Return newNode

Algorithm 2: Insert at Beginning

Algorithm InsertAtBeginning_DLL(head, value)
1. Create newNode using CreateDNode(value)

2. Set newNode→next ← head
3. Set newNode→prev ← NULL
4. If head ≠ NULL:
   a. Set head→prev ← newNode
5. Set head ← newNode
6. Return head

Algorithm 3: Insert at End

Algorithm InsertAtEnd_DLL(head, value)
1. Create newNode using CreateDNode(value)
2. If head = NULL:
   a. Return newNode
3. Set temp ← head
4. While temp→next ≠ NULL:
   a. temp ← temp→next
5. Set temp→next ← newNode
6. Set newNode→prev ← temp
7. Return head

Algorithm 4: Insert at Specific Position (0-based index)

Algorithm InsertAtPosition_DLL(head, value, pos)
1. If pos = 0:
   a. Return InsertAtBeginning_DLL(head, value)
2. Create newNode using CreateDNode(value)
3. Set temp ← head
4. Loop i ← 0 to pos - 2:
   a. If temp = NULL:
      i. Return head (invalid position)
   b. temp ← temp→next
5. Set newNode→next ← temp→next
6. Set newNode→prev ← temp
7. If temp→next ≠ NULL:
   a. temp→next→prev ← newNode
8. Set temp→next ← newNode
9. Return head

Algorithm 5: Delete from Beginning

Algorithm DeleteFromBeginning_DLL(head)
1. If head = NULL:
   a. Return NULL
2. Set temp ← head
3. Set head ← head→next
4. If head ≠ NULL:
   a. head→prev ← NULL
5. Free temp
6. Return head

Algorithm 6: Delete from End

Algorithm DeleteFromEnd_DLL(head)
1. If head = NULL:
   a. Return NULL
2. If head→next = NULL:
   a. Free head
   b. Return NULL
3. Set temp ← head
4. While temp→next ≠ NULL:
   a. temp ← temp→next
5. Set temp→prev→next ← NULL
6. Free temp
7. Return head

Algorithm 7: Delete at Specific Position (0-based index)

Algorithm DeleteAtPosition_DLL(head, pos)
1. If head = NULL:
   a. Return NULL
2. If pos = 0:
   a. Return DeleteFromBeginning_DLL(head)
3. Set temp ← head
4. Loop i ← 0 to pos - 1:
   a. If temp = NULL:
      i. Return head (invalid position)
   b. temp ← temp→next
5. If temp = NULL:
   a. Return head (invalid)
6. Set temp→prev→next ← temp→next
7. If temp→next ≠ NULL:
   a. temp→next→prev ← temp→prev
8. Free temp
9. Return head

Algorithm 8: Search an Element

Algorithm Search_DLL(head, key)
1. Set temp ← head
2. While temp ≠ NULL:
   a. If temp→data = key:
      i. Return true
   b. temp ← temp→next
3. Return false

Algorithm 9: Display Forward

Algorithm DisplayForward_DLL(head)
1. Set temp ← head
2. While temp ≠ NULL:
   a. Print temp→data
   b. temp ← temp→next

Algorithm 10: Display Backward

```
Algorithm DisplayBackward_DLL(head)
1. If head = NULL:
   a. Return
2. Set temp ← head
3. While temp→next ≠ NULL:
   a. temp ← temp→next
4. While temp ≠ NULL:
   a. Print temp→data
   b. temp ← temp→prev
```

Q10. Write the code snippet for each operation using your algorithm logic and the structure:

```
struct DNode* insertAtBeginning(struct DNode* head, int value) {
  // code here
}
```

Q11. Write a main() function for Doubly Linked List similar to Q5 but including:

- Display forward
- Display backward

Sample menu format:

```
1. Insert at Beginning
2. Insert at End
3. Insert at Position
4. Delete from Beginning
5. Delete from End
6. Delete a Node
7. Search an Element
8. Display Forward
9. Display Backward
10. Exit
```

Q12. Combine all code snippets into a single working file for Doubly Linked List and test all operations.

Q13. Create a program for the implementation of a Doubly Linked List with all operations in C++.

Submission Checklist

Before submission, ensure:

- All operations implemented

- Menu-driven program works correctly
- Output screenshots attached for each function
- Proper indentation and comments in code