



# COMSATS University Islamabad, Vehari Campus

Department of Computer Science

**Class: BCS-SP22**

**Submission Deadline: 9 Oct 2023**

**Subject: Data Structures and Algorithms-Lab**

**Instructor: Yasmeen Jana**

**Max Marks: 20**

**Reg. No: SP22-BCS-082**

## Activity 1:

Create a function to display linked list output as below:

```
The linked list is:
1 2 20 30

****head address:*** 0x6ffe18
-----
head content: 0x151530
-----

****ptr address:**** 0x6ffdb8
-----
ptr content: 0x151530
-----
ptr->data: 1
-----
ptr: 0x151530
ptr->next: 0x151560
ptr->data: 2
-----
ptr: 0x151560
ptr->next: 0x151a30
ptr->data: 20
-----
ptr: 0x151a30
ptr->next: 0x151a60
ptr->data: 30
-----
ptr: 0x151a60
ptr->next: 0
```

**Answer:**

```
#include <iostream>
```

```
Struct Node {
```

```
    Int data;
```

```
    Node* next;
```

```
    Node(int value) : data(value), next(NULL) {}
```

```
};
```

```
Void displayLinkedList(Node* head) {
```

```
    If (head == NULL) {
```

```
        Std::cout << "The linked list is empty." << std::endl;
```

```
        Return;
```

```
    }
```

```
    Std::cout << "The linked list is:" << std::endl;
```

```
    Node* ptr = head;
```

```
    While (ptr) {
```

```
        Std::cout << ptr->data << " ";
```

```
        Ptr = ptr->next;
```

```
    }
```

```
    Std::cout << std::endl;
```

```
    Std::cout << "*head address:*" << &head << std::endl;
```

```
    Std::cout << "-----" << std::endl;
```

```
    Ptr = head;
```

```
    While (ptr) {
```

```
        Std::cout << "\nhead content: " << ptr << std::endl;
```

```
        Std::cout << "-----" << std::endl;
```

```

Std::cout << "*ptr address: **" << ptr << std::endl;
Std::cout << "-----" << std::endl;
Std::cout << "ptr content: " << ptr << std::endl;
Std::cout << "-----" << std::endl;
Std::cout << "ptr->data: " << ptr->data << std::endl;
Std::cout << "-----" << std::endl;
Std::cout << "ptr: " << ptr << std::endl;
If (ptr->next) {
    Std::cout << "ptr->next: " << ptr->next << std::endl;
    Std::cout << "ptr->data: " << ptr->next->data << std::endl;
    Std::cout << "-----" << std::endl;
} else {
    Std::cout << "ptr->next: 0

```

### Output:

```

C:\Users\ali\Documents\assign2_prog1.exe
The linked list is:
1 2 20 30
*head address:**0x22fe10
-----
head content: 0x50dd50
*ptr address: **0x50dd50
ptr content: 0x50dd50
-----
ptr->data: 1
ptr: 0x50dd50
ptr->next: 0x50dd70
ptr->data: 2
-----
head content: 0x50dd70
*ptr address: **0x50dd70
ptr content: 0x50dd70
-----
ptr->data: 2
ptr: 0x50dd70
ptr->next: 0x50dd90
ptr->data: 20
-----
head content: 0x50dd90
*ptr address: **0x50dd90
ptr content: 0x50dd90
-----
ptr->data: 20
ptr: 0x50dd90
ptr->next: 0x50df60
ptr->data: 30
-----
head content: 0x50df60
*ptr address: **0x50df60
ptr content: 0x50df60
-----
ptr->data: 30
ptr: 0x50df60
ptr->next: 0

```

## **Activity 2:**

Write a program that will implement single, doubly, and circular linked list operations by showing a menu to the user.

The menu should be:

**Which linked list you want:**

- 1: Single
- 2: Double
- 3: Circular

After the option is chosen by the user:

**Which operation you want to perform:**

- 1: Insertion
- 2: Deletion
- 3: Display
- 4: Reverse
- 4: Seek
- 5: Exit

**Let's suppose, the user has chosen the insertion option then the following menu should be displayed:**

- 1: insertion at beginning
- 2: insertion at end
- 3: insertion at the specific data node

## **Answer:**

```
#include <iostream>
using namespace std;
// Define a Node structure
struct Node {
    int data;
    Node* next;
```

```

    Node* prev; // Used for doubly linked list
};
// Define a LinkedList class
class LinkedList {
private:
    Node* head;
    Node* tail; // Used for doubly linked list
    bool isDoubly;
public:
    LinkedList(bool doubly) {
        head = NULL;
        tail = NULL;
        isDoubly = doubly;
    }
    // Function to insert a node at the beginning of the list
    void insertAtBeginning(int value) {
        Node* newNode = new Node;
        newNode->data = value;
        newNode->next = head;
        if (isDoubly) {
            newNode->prev = NULL;
            if (head != NULL) {
                head->prev = newNode;
            }
        }
        head = newNode;
        if (!isDoubly && tail == NULL) {
            tail = head;
        }
    }
}

```

```

// Function to insert a node at the end of the list
void insertAtEnd(int value) {
    Node* newNode = new Node;
    newNode->data = value;
    newNode->next = NULL;
    if (isDoubly) {
        newNode->prev = tail;
        if (tail != NULL) {
            tail->next = newNode;
        }
        tail = newNode;
    } else {
        if (tail == NULL) {
            head = newNode;
            tail = newNode;
        } else {
            tail->next = newNode;
            tail = newNode;
        }
    }
}

// Function to insert a node after a specific data node
void insertAfter(int target, int value) {
    Node* current = head;
    while (current != NULL) {
        if (current->data == target) {
            Node* newNode = new Node;
            newNode->data = value;
            newNode->next = current->next;
            if (isDoubly) {

```

```

        newNode->prev = current;
        if (current->next != NULL) {
            current->next->prev = newNode;
        }
    }
    current->next = newNode;
    break;
}
current = current->next;
}
}

// Function to delete a node with a specific data value
void deleteNode(int value) {
    Node* current = head;
    Node* prev = NULL;
    while (current != NULL) {
        if (current->data == value) {
            if (prev != NULL) {
                prev->next = current->next;
            } else {
                head = current->next;
            }
            if (isDoubly && current->next != NULL) {
                current->next->prev = current->prev;
            }
            delete current;
            break;
        }
        prev = current;
        current = current->next;
    }
}

```

```
}  
}
```

```
// Function to display the linked list
```

```
void display() {
```

```
    Node* current = head;
```

```
    while (current != NULL) {
```

```
        cout << current->data << " ";
```

```
        current = current->next;
```

```
    }
```

```
    cout << endl;
```

```
}
```

```
// Function to reverse the linked list
```

```
void reverse() {
```

```
    if (isDoubly) {
```

```
        Node* temp = NULL;
```

```
        Node* current = head;
```

```
        while (current != NULL) {
```

```
            temp = current->prev;
```

```
            current->prev = current->next;
```

```
            current->next = temp;
```

```
            current = current->prev;
```

```
        }
```

```
        if (temp != NULL) {
```

```
            head = temp->prev;
```

```
        }
```

```
    } else {
```

```
        Node* prev = NULL;
```

```
        Node* current = head;
```

```
        Node* next = NULL;
```



```

        while (current != NULL) {
            next = current->next;
            current->next = prev;
            prev = current;
            current = next;
        }
        head = prev;
    }
}

// Function to seek a specific data node
bool seek(int value) {
    Node* current = head;
    while (current != NULL) {
        if (current->data == value) {
            return true;
        }
        current = current->next;
    }
    return false;
}

};

// Function to display the linked list after an operation
void displayLinkedList(LinkedList& list) {
    cout << "Linked List: ";
    list.display();
}

int main() {
    int choice;
    bool isDoubly;
    cout << "Which linked list you want:" << endl;

```

```
cout << "1: Single" << endl;
cout << "2: Double" << endl;
cout << "3: Circular" << endl;
cin >> choice;
switch (choice) {
    case 1:
        isDoubly = false;
        break;
    case 2:
        isDoubly = true;
        break;
    case 3:
        // Handle circular linked list
        cout << "Circular linked list is not implemented in this example." << endl;
        return 0;
    default:
        cout << "Invalid choice. Exiting..." << endl;
        return 0;
}
LinkedList list(isDoubly);
while (true) {
    cout << "Which operation you want to perform:" << endl;
    cout << "1: Insertion" << endl;
    cout << "2: Deletion" << endl;
    cout << "3: Display" << endl;
    cout << "4: Reverse" << endl;
    cout << "5: Seek" << endl;
    cout << "6: Exit" << endl;
    cin >> choice;
    switch (choice) {
```

case 1:

```
int insertChoice;  
cout << "1: Insertion at beginning" << endl;  
cout << "2: Insertion at end" << endl;  
cout << "3: Insertion at specific data node" << endl;  
cin >> insertChoice;  
switch (insertChoice) {
```

case 1:

```
int value;  
cout << "Enter value to insert: ";  
cin >> value;  
list.insertAtBeginning(value);  
displayLinkedList(list);  
break;
```

case 2:

```
cout << "Enter value to insert: ";  
cin >> value;  
list.insertAtEnd(value);  
displayLinkedList(list);  
break;
```

case 3:

```
int targetValue;  
cout << "Enter the data value after which to insert: ";  
cin >> targetValue;  
cout << "Enter value to insert: ";  
cin >> value;  
list.insertAfter(targetValue, value);  
displayLinkedList(list);  
break;
```

default:

```

        cout << "Invalid choice." << endl;
    }
    break;
case 2:
    int deleteValue;
    cout << "Enter value to delete: ";
    cin >> deleteValue;
    list.deleteNode(deleteValue);
    displayLinkedList(list);
    break;
case 3:
    displayLinkedList(list);
    break;
case 4:
    list.reverse();
    cout << "Linked List reversed." << endl;
    displayLinkedList(list);
    break;
case 5:
    int seekValue;
    cout << "Enter value to seek: ";
    cin >> seekValue;
    if (list.seek(seekValue)) {
        cout << "Value found in the list." << endl;
    } else {
        cout << "Value not found in the list." << endl;
    }
    break;
case 6:
    cout << "Exiting program..." << endl;

```

```

        return 0;

default:

    cout << "Invalid choice. Please try again." << endl;

    }

}

return 0;

}

```

### Output:

```

C:\Users\ali\Documents\assign2_prog2.exe
Which linked list you want:
1: Single
2: Double
3: Circular
1
Which operation you want to perform:
1: Insertion
2: Deletion
3: Display
4: Reverse
5: Seek
6: Exit
1
1: Insertion at beginning
2: Insertion at end
3: Insertion at specific data node
1
Enter value to insert: 4
Linked List: 4
Which operation you want to perform:
1: Insertion
2: Deletion
3: Display
4: Reverse
5: Seek
6: Exit
1
1: Insertion at beginning
2: Insertion at end
3: Insertion at specific data node
1
Enter value to insert: 6
Linked List: 6 4
Which operation you want to perform:
1: Insertion
2: Deletion
3: Display
4: Reverse
5: Seek
6: Exit

```