COMSATS University Islamabad, Vehari Campus

Department of Computer Science

Class: BCS-SP22

Submission Deadline: 9 Oct 2023

Subject: Data Structures and Algorithms-Lab

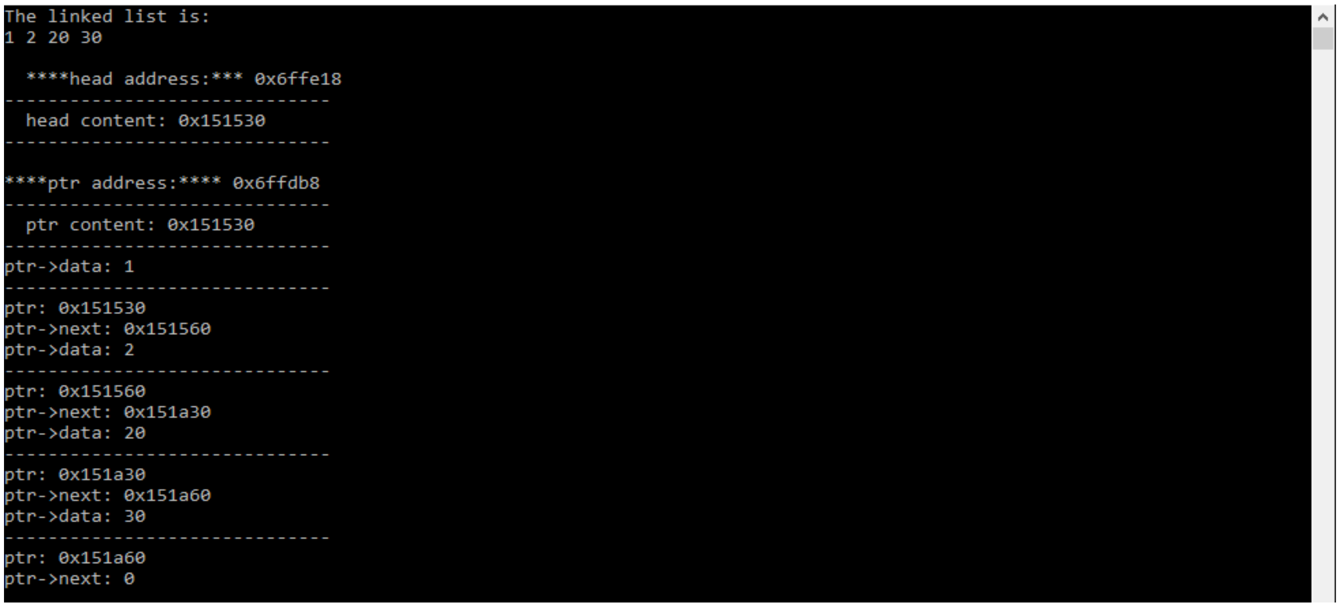
Instructor: Yasmeen Jana

Max Marks: 20

Reg. No: SP22-BCS-005

Activity 1:

Create a function to display linked list output as below:



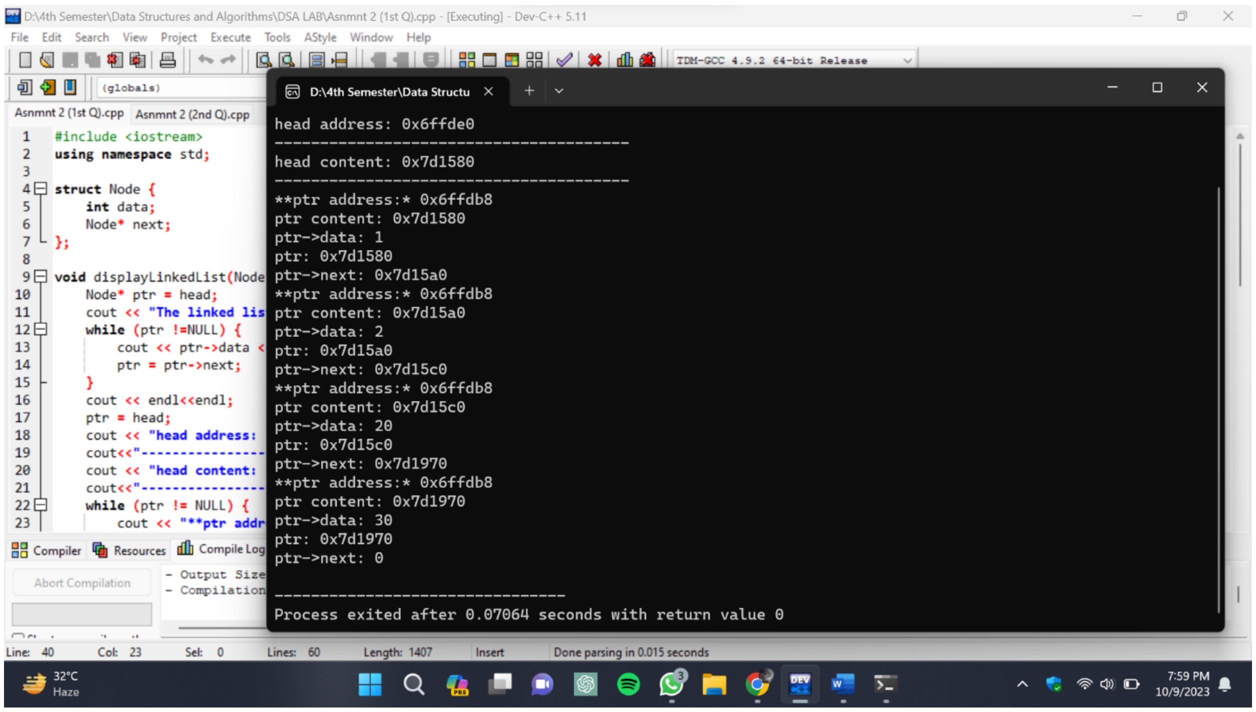
```
The linked list is:
1 2 20 30

  ****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
```

cpp CODE:

```
#include <iostream>

using namespace std;

struct Node {

    int data;

    Node* next;

};

void displayLinkedList(Node* head) {

    Node* ptr = head;

    cout << "The linked list is: "<<endl;

    while (ptr !=NULL) {

        cout << ptr->data << " ";

        ptr = ptr->next;

    }

    cout << endl<<endl;

    ptr = head;

    cout << "head address: " << &head << endl;

    cout<<"---------------------------------------"<<endl;

    cout << "head content: " << head << endl;

    cout<<"---------------------------------------"<<endl;

    while (ptr != NULL) {

        cout << "**ptr address:* " << &ptr << endl;

        cout << "ptr content: " << ptr << endl;

        cout << "ptr->data: " << ptr->data << endl;

        cout << "ptr: " << ptr << endl;

        cout << "ptr->next: " << ptr->next << endl;
        ptr = ptr->next;
    }
}
int main() {
    Node* head = NULL;
    Node* second = NULL;
    Node* third = NULL;
    Node* fourth = NULL;
    head = new Node;
    second = new Node;
    third = new Node;
    fourth = new Node;
    head->data = 1;
    head->next = second;
    second->data = 2;
    second->next = third;
    third->data = 20;
    third->next = fourth;
    fourth->data = 30;
    fourth->next = NULL;
    displayLinkedList(head);
    delete head;
    delete second;
    delete third;
    delete fourth;
```

```
 return 0;

}
```

Output:



```
#include <iostream>
using namespace std;

struct Node {
    int data;
    Node* next;
};

void displayLinkedList(Node
    Node* ptr = head;
    cout << "The linked lis
    while (ptr !=NULL) {
        cout << ptr->data <
        ptr = ptr->next;
    }
    cout << endl<<endl;
    ptr = head;
    cout << "head address:
    cout<<"---------------
    cout << "head content:
    cout<<"---------------
    while (ptr != NULL) {
        cout << "**ptr addr
```

```
head address: 0x6ffde0
---------------------------------------
head content: 0x7d1580
---------------------------------------
**ptr address:* 0x6ffdb8
ptr content: 0x7d1580
ptr->data: 1
ptr: 0x7d1580
ptr->next: 0x7d15a0
**ptr address:* 0x6ffdb8
ptr content: 0x7d15a0
ptr->data: 2
ptr: 0x7d15a0
ptr->next: 0x7d15c0
**ptr address:* 0x6ffdb8
ptr content: 0x7d15c0
ptr->data: 20
ptr: 0x7d15c0
ptr->next: 0x7d1970
**ptr address:* 0x6ffdb8
ptr content: 0x7d1970
ptr->data: 30
ptr: 0x7d1970
ptr->next: 0


--------------------------------
Process exited after 0.07064 seconds with return value 0
```

Activity 2:

Write a program that will implement single, doubly, and circular linked link 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

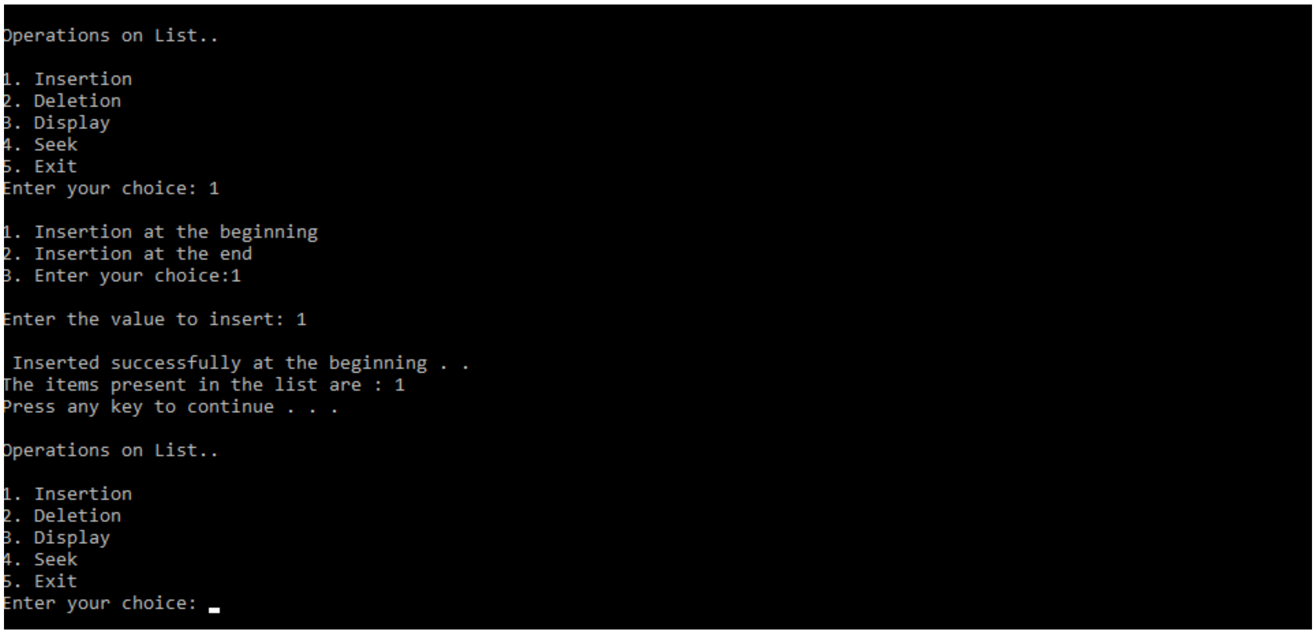
Let's suppose, the user has chosen the insertion option then the following menu should be displayed:

1: insertion at beginning

2: insertion at end

3: insertion at the specific data node

A sample output screenshot is below:



```
Operations on List..

1. Insertion
2. Deletion
3. Display
4. Seek
5. Exit
Enter your choice: 1

1. Insertion at the beginning
2. Insertion at the end
3. Enter your choice:1

Enter the value to insert: 1

 Inserted successfully at the beginning . .
The items present in the list are : 1
Press any key to continue . . .

Operations on List..

1. Insertion
2. Deletion
3. Display
4. Seek
5. Exit
Enter your choice: _
```

<u>cpp CODE:</u>

```
#include <iostream>

using namespace std;

struct Node {

    int data;
    Node* next;

    Node* prev;

};


class SingleLinkedList {

private:

    Node* head;


public:

    SingleLinkedList() {

        head = NULL;

    }


    void insertAtBeginning(int value) {

        Node* newNode = new Node{value, head};

        head = newNode;

    }


    void insertAtEnd(int value) {

        Node* newNode = new Node{value, NULL};


        if (!head) {

            head = newNode;

        } else {

            Node* current = head;
            while (current->next) {
                current = current->next;
            }
            current->next = newNode;
        }
    }

    void insertAfterData(int value, int targetValue) {
        Node* newNode = new Node{value, NULL};
        Node* current = head;

        while (current) {
            if (current->data == targetValue) {
                newNode->next = current->next;
                current->next = newNode;
                return;
            }
            current = current->next;
        }

        std::cout << "Target value not found in the list." << std::endl;
    }

    void deleteNode(int value) {
        Node* current = head;
    }

        Node* prev = NULL;

        while (current) {
            if (current->data == value) {
                if (prev) {
                    prev->next = current->next;
                } else {
                    head = current->next;
                }
                delete current;
                return;
            }
            prev = current;
            current = current->next;
        }

        std::cout << "Value not found in the list." << std::endl;
    }

    void display() {
        Node* current = head;
        while (current) {
            std::cout << current->data << " ";
            current = current->next;
        }
        std::cout << std::endl;
    }

    void reverse() {
        Node* prev = NULL;
        Node* current = head;
        Node* nextNode = NULL;

        while (current) {
            nextNode = current->next;
            current->next = prev;
            prev = current;
            current = nextNode;
        }

        head = prev;
    }

    bool seek(int value) {
        Node* current = head;
        while (current) {
            if (current->data == value) {
                return true;
            }
            current = current->next;
        }
        return false;
    }
};

class DoublyLinkedList {
private:
    Node* head;
    Node* tail;

public:
    DoublyLinkedList() {
        head = NULL;
        tail = NULL;
    }

    void insertAtBeginning(int value) {
        Node* newNode = new Node{value, head};
        if (!head) {
            tail = newNode;
        } else {
            head->prev = newNode;
        }
        head = newNode;
    }
```

```
void insertAtEnd(int value) {

    Node* newNode = new Node{value, NULL};


    if (!head) {

        head = newNode;

        tail = newNode;

    } else {

        newNode->prev = tail;

        tail->next = newNode;

        tail = newNode;

    }

}


void insertAfterData(int value, int targetValue) {

    Node* newNode = new Node{value, NULL};

    Node* current = head;


    while (current) {

        if (current->data == targetValue) {

            newNode->next = current->next;

            newNode->prev = current;

            if (current->next) {

                current->next->prev = newNode;

            }
            current->next = newNode;
            return;
        }
        current = current->next;
    }

    std::cout << "Target value not found in the list." << std::endl;
}

void deleteNode(int value) {
    Node* current = head;

    while (current) {
        if (current->data == value) {
            if (current->prev) {
                current->prev->next = current->next;
            } else {
                head = current->next;
            }
            if (current->next) {
                current->next->prev = current->prev;
            } else {
                tail = current->prev;
            }
            delete current;
            return;

        }

        current = current->next;

    }


    std::cout << "Value not found in the list." << std::endl;

}


void display() {

    Node* current = head;

    while (current) {

        std::cout << current->data << " ";

        current = current->next;

    }

    std::cout << std::endl;

}


void reverse() {

    Node* temp = NULL;

    Node* current = head;


    while (current) {

        temp = current->prev;

        current->prev = current->next;

        current->next = temp;
```

```
            current = current->prev;
        }

        if (temp) {
            head = temp->prev;
        }
    }

    bool seek(int value) {
        Node* current = head;
        while (current) {
            if (current->data == value) {
                return true;
            }
            current = current->next;
        }
        return false;
    }
};

class CircularLinkedList {
private:
    Node* head;

public:
```

```
CircularLinkedList() {
    head = NULL;
}

void insertAtBeginning(int value) {
    Node* newNode = new Node{value, head};
    if (!head) {
        newNode->next = newNode;
    } else {
        Node* current = head;
        while (current->next != head) {
            current = current->next;
        }
        current->next = newNode;
    }
    head = newNode;
}

void insertAtEnd(int value) {
    Node* newNode = new Node{value, head};
    if (!head) {
        newNode->next = newNode;
        head = newNode;
    } else {
        Node* current = head;
```

```
            while (current->next != head) {

                current = current->next;

            }

            current->next = newNode;

        }

    }


    void insertAfterData(int value, int targetValue) {

        Node* newNode = new Node{value, NULL};

        Node* current = head;


        while (current) {

            if (current->data == targetValue) {

                newNode->next = current->next;

                current->next = newNode;

                return;

            }

            current = current->next;

            if (current == head) {

                std::cout << "Target value not found in the list." << std::endl;

                return;

            }

        }

    }
```

```
void deleteNode(int value) {
    if (!head) {
        std::cout << "Value not found in the list." << std::endl;
        return;
    }

    Node* current = head;
    Node* prev = NULL;

    do {
        if (current->data == value) {
            if (prev) {
                prev->next = current->next;
            } else {
                Node* temp = current;
                while (temp->next != head) {
                    temp = temp->next;
                }
                temp->next = current->next;
                head = current->next;
            }
            delete current;
            return;
        }
        prev = current;
```

```
            current = current->next;
        } while (current != head);

        std::cout << "Value not found in the list." << std::endl;
    }

    void display() {
        if (!head) {
            std::cout << "Circular Linked List is empty." << std::endl;
            return;
        }

        Node* current = head;

        do {
            std::cout << current->data << " ";
            current = current->next;
        } while (current != head);

        std::cout << std::endl;
    }

    void reverse() {
        std::cout << "Reversing a circular linked list is not implemented." <<
std::endl;
    }
    bool seek(int value) {

        Node* current = head;


        if (!current) {

            return false;

        }


        do {

            if (current->data == value) {

                return true;

            }

            current = current->next;

        } while (current != head);


        return false;

    }

};


int main() {

    SingleLinkedList singleLinkedList;

    DoublyLinkedList doubleLinkedList;

    CircularLinkedList circularLinkedList;


    int choice1, choice2, value, targetValue;
```

```
while (true) {

    std::cout << "Which linked list you want:" << std::endl;

    std::cout << "1: Single" << std::endl;

    std::cout << "2: Double" << std::endl;

    std::cout << "3: Circular" << std::endl;

    std::cout << "Enter your choice (1/2/3): ";

    std::cin >> choice1;


    if (choice1 == 1) {

        std::cout << "Which operation you want to perform:" << std::endl;

        std::cout << "1: Insertion" << std::endl;

        std::cout << "2: Deletion" << std::endl;

        std::cout << "3: Display" << std::endl;

        std::cout << "4: Reverse" << std::endl;

        std::cout << "5: Seek" << std::endl;

        std::cout << "6: Exit" << std::endl;

        std::cout << "Enter your choice (1/2/3/4/5/6): ";

        std::cin >> choice2;


        switch (choice2) {

            case 1:

                std::cout << "1: Insertion at beginning" << std::endl;

                std::cout << "2: Insertion at end" << std::endl;

                std::cout << "3: Insertion at specific data node" << std::endl;
```

```
                    std::cout << "Enter your choice (1/2/3): ";
                    std::cin >> choice2;

                    std::cout << "Enter the value to insert: ";
                    std::cin >> value;

                    switch (choice2) {
                        case 1:
                            singleLinkedList.insertAtBeginning(value);
                            break;
                        case 2:
                            singleLinkedList.insertAtEnd(value);
                            break;
                        case 3:
                            std::cout << "Enter the target value after which to insert: 
";
                            std::cin >> targetValue;
                            singleLinkedList.insertAfterData(value, targetValue);
                            break;
                        default:
                            std::cout << "Invalid choice." << std::endl;
                            break;
                    }
                    break;
                case 2:
                    std::cout << "Enter the value to delete: ";
```

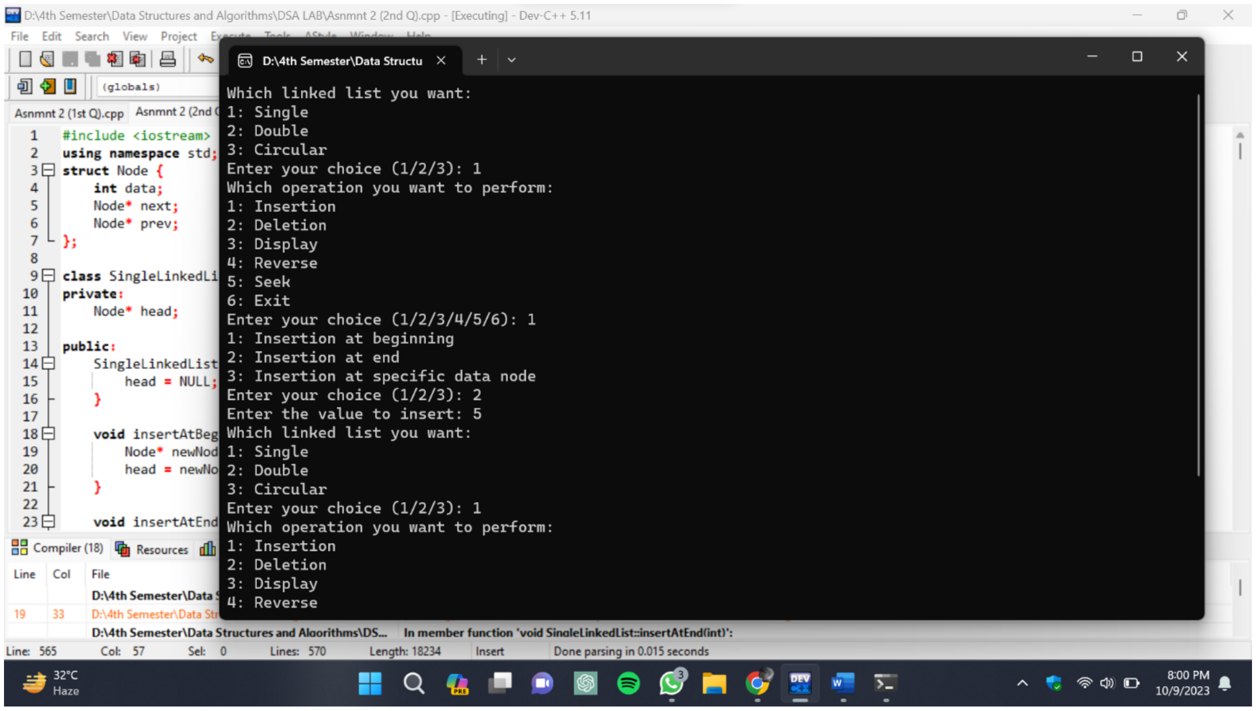
```
    std::cin >> value;

    singleLinkedList.deleteNode(value);

    break;

case 3:

    std::cout << "Single Linked List: ";

    singleLinkedList.display();

    break;

case 4:

    singleLinkedList.reverse();

    std::cout << "Single Linked List reversed." << std::endl;

    break;

case 5:

    std::cout << "Enter the value to seek: ";

    std::cin >> value;

    if (singleLinkedList.seek(value)) {

        std::cout << "Value found in the list." << std::endl;

    } else {

        std::cout << "Value not found in the list." << std::endl;

    }

    break;

case 6:

    exit(0);

default:

    std::cout << "Invalid choice." << std::endl;

    break;
```

```
            }
        } else if (choice1 == 2) {
            std::cout << "Which operation you want to perform:" << std::endl;
            std::cout << "1: Insertion" << std::endl;
            std::cout << "2: Deletion" << std::endl;
            std::cout << "3: Display" << std::endl;
            std::cout << "4: Reverse" << std::endl;
            std::cout << "5: Seek" << std::endl;
            std::cout << "6: Exit" << std::endl;
            std::cout << "Enter your choice (1/2/3/4/5/6): ";
            std::cin >> choice2;

            switch (choice2) {
                case 1:
                    std::cout << "1: Insertion at beginning" << std::endl;
                    std::cout << "2: Insertion at end" << std::endl;
                    std::cout << "3: Insertion at specific data node" << std::endl;
                    std::cout << "Enter your choice (1/2/3): ";
                    std::cin >> choice2;

                    std::cout << "Enter the value to insert: ";
                    std::cin >> value;

                    switch (choice2) {
                        case 1:
                            doubleLinkedList.insertAtBeginning(value);
                            break;
                        case 2:
                            doubleLinkedList.insertAtEnd(value);
                            break;
                        case 3:
                            std::cout << "Enter the target value after which to insert:
";
                            std::cin >> targetValue;
                            doubleLinkedList.insertAfterData(value, targetValue);
                            break;
                        default:
                            std::cout << "Invalid choice." << std::endl;
                            break;
                    }
                    break;
                case 2:
                    std::cout << "Enter the value to delete: ";
                    std::cin >> value;
                    doubleLinkedList.deleteNode(value);
                    break;
                case 3:
                    std::cout << "Doubly Linked List: ";
                    doubleLinkedList.display();
                    break;
                case 4:
```

```
                doubleLinkedList.reverse();

                std::cout << "Doubly Linked List reversed." << std::endl;

                break;

            case 5:

                std::cout << "Enter the value to seek: ";

                std::cin >> value;

                if (doubleLinkedList.seek(value)) {

                    std::cout << "Value found in the list." << std::endl;

                } else {

                    std::cout << "Value not found in the list." << std::endl;

                }

                break;

            case 6:

                exit(0);

            default:

                std::cout << "Invalid choice." << std::endl;

                break;

        }

    } else if (choice1 == 3) {

        std::cout << "Which operation you want to perform:" << std::endl;

        std::cout << "1: Insertion" << std::endl;

        std::cout << "2: Deletion" << std::endl;

        std::cout << "3: Display" << std::endl;

        std::cout << "4: Reverse" << std::endl;

        std::cout << "5: Seek" << std::endl;
```

```
            std::cout << "6: Exit" << std::endl;
            std::cout << "Enter your choice (1/2/3/4/5/6): ";
            std::cin >> choice2;

            switch (choice2) {
                case 1:
                    std::cout << "1: Insertion at beginning" << std::endl;
                    std::cout << "2: Insertion at end" << std::endl;
                    std::cout << "3: Insertion at specific data node" << std::endl;
                    std::cout << "Enter your choice (1/2/3): ";
                    std::cin >> choice2;

                    std::cout << "Enter the value to insert: ";
                    std::cin >> value;

                    switch (choice2) {
                        case 1:
                            circularLinkedList.insertAtBeginning(value);
                            break;
                        case 2:
                            circularLinkedList.insertAtEnd(value);
                            break;
                        case 3:
                            std::cout << "Enter the target value after which to insert:
";
                            std::cin >> targetValue;
                            circularLinkedList.insertAfterData(value, targetValue);

                            break;

                        default:

                            std::cout << "Invalid choice." << std::endl;

                            break;

                    }

                    break;

                case 2:

                    std::cout << "Enter the value to delete: ";

                    std::cin >> value;

                    circularLinkedList.deleteNode(value);

                    break;

                case 3:

                    std::cout << "Circular Linked List: ";

                    circularLinkedList.display();

                    break;

                case 4:

                    std::cout << "Reversing a circular linked list is not implemented."
<< std::endl;

                    break;

                case 5:

                    std::cout << "Enter the value to seek: ";

                    std::cin >> value;

                    if (circularLinkedList.seek(value)) {

                        std::cout << "Value found in the list." << std::endl;

                    } else {
                        std::cout << "Value not found in the list." << std::endl;
                    }
                    break;
                case 6:
                    exit(0);
                default:
                    std::cout << "Invalid choice." << std::endl;
                    break;
            }
        } else {
            std::cout << "Invalid choice." << std::endl;
        }
    }
    return 0;
}
```

Output:



D:\4th Semester\Data Structures and Algorithms\DSA LAB\Asnmnt 2 (2nd Q).cpp - [Executing] - Dev-C++ 5.11

```
#include <iostream>
using namespace std;
struct Node {
    int data;
    Node* next;
    Node* prev;
};

class SingleLinkedLi
private:
    Node* head;

public:
    SingleLinkedList
        head = NULL;
    }

    void insertAtBeg
        Node* newNod
        head = newNo
    }

    void insertAtEnd
```

D:\4th Semester\Data Structures and Algorithms\DS... In member function 'void SingleLinkedList::insertAtEnd(int)':

```
Which linked list you want:
1: Single
2: Double
3: Circular
Enter your choice (1/2/3): 1
Which operation you want to perform:
1: Insertion
2: Deletion
3: Display
4: Reverse
5: Seek
6: Exit
Enter your choice (1/2/3/4/5/6): 1
1: Insertion at beginning
2: Insertion at end
3: Insertion at specific data node
Enter your choice (1/2/3): 2
Enter the value to insert: 5
Which linked list you want:
1: Single
2: Double
3: Circular
Enter your choice (1/2/3): 1
Which operation you want to perform:
1: Insertion
2: Deletion
3: Display
4: Reverse
```

Line: 565 Col: 57 Sel: 0 Lines: 570 Length: 18234 Insert Done parsing in 0.015 seconds