Data Structures and Algorithms Lab(MCSE501P)

Assessment - 3

Name : Mahesh Jagtap
Reg No.24MCS1017

1. Write a program to display all operations of a singly linked list.

a. traverse();
b. insertAtFront();
c. insertAtEnd();
d. insertAtPosition();
e. deleteFirst();
f. deleteEnd();
g. deletePosition();
h. sort();
i. search();

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

void traverse(Node* head) {
    Node* temp = head;
    while (temp) {
        cout << temp->data << " -> ";
        temp = temp->next;
    }
    cout << "NULL" << endl;
}

void insertAtFront(Node*& head, int data) {
    head = new Node{data, head};
    cout<<"data inserted ";
}

void insertAtEnd(Node*& head, int data) {
    Node* newNode = new Node{data, nullptr};
    if (!head) {
        head = newNode;
    } else {
        Node* temp = head;
```

```
        while (temp->next) {
            temp = temp->next;
        }
        temp->next = newNode;
    }
    cout<<"data inserted ";
}

void insertAtPosition(Node*& head, int data, int position) {
    if (position == 0) {
        insertAtFront(head, data);
        return;
    }

    Node* newNode = new Node{data, nullptr};
    Node* temp = head;
    for (int i = 0; temp && i < position - 1; ++i) {
        temp = temp->next;
    }
    if (temp) {
        newNode->next = temp->next;
        temp->next = newNode;
    } else {
        cout << "Position out of range\n";
        delete newNode;
        return;
    }
    cout<<"data inserted ";
}
void deleteFirst(Node*& head) {
    if (head) {
        Node* temp = head;
        head = head->next;
        delete temp;
    } else {
        cout << "List is empty" << endl;
        return;
    }
    cout<<"data deleted ";
}

void deleteEnd(Node*& head) {
    if (!head) {
        cout << "List is empty" << endl;
        return;
    }
    if (!head->next) {
        delete head;
```

```
        head = nullptr;
    } else {
        Node* temp = head;
        while (temp->next->next) {
            temp = temp->next;
        }
        delete temp->next;
        temp->next = nullptr;
    }
    cout<<"node deleted ";
}

void deleteAtPosition(Node*& head, int position) {
    if (position == 0) {
        deleteFirst(head);
        return;
    }

    Node* temp = head;
    for (int i = 0; temp && i < position - 1; ++i) {
        temp = temp->next;
    }
    if (temp && temp->next) {
        Node* toDelete = temp->next;
        temp->next = temp->next->next;
        delete toDelete;
    } else {
        cout << "Position out of range" << endl;
        return;
    }
        cout<<"node deleted ";

}
void sort(Node*& head) {
    if (!head) return;

    bool swapped;
    Node *ptr1;
    Node *lptr = nullptr;

    do {
        swapped = false;
        ptr1 = head;

        while (ptr1->next != lptr) {
            if (ptr1->data > ptr1->next->data) {
                // Swap data
                int temp = ptr1->data;
```

```
            ptr1->data = ptr1->next->data;
            ptr1->next->data = temp;
            swapped = true;
        }
        ptr1 = ptr1->next;
      }
      lptr = ptr1;
   } while (swapped);
    traverse(head);

}

bool search(Node* head, int key) {
   Node* temp = head;
   while (temp) {
      if (temp->data == key) return true;
      temp = temp->next;
   }
   return false;
}

int main() {
   Node* head = nullptr;
   int choice, data, position;

   while (true) {
      cout << "\n1. Insert at Front\n2. Insert at End\n3. Insert at Position\n4. Delete First\n5.
Delete End\n6. Delete at Position\n7. Sort\n8. Search\n9. Traverse\n0. Exit\n";
      cout << "Enter choice: ";
      cin >> choice;

      switch (choice) {
         case 1:
            cout << "Enter data: ";
            cin >> data;
            insertAtFront(head, data);
            break;
         case 2:
            cout << "Enter data: ";
            cin >> data;
            insertAtEnd(head, data);
            break;
         case 3:
            cout << "Enter data and position: ";
            cin >> data >> position;
            insertAtPosition(head, data, position);
            break;
         case 4:
```

```
            deleteFirst(head);
            break;
        case 5:
            deleteEnd(head);
            break;
        case 6:
            cout << "Enter position: ";
            cin >> position;
            deleteAtPosition(head, position);
            break;
        case 7:
            sort(head);
            break;
        case 8:
            cout << "Enter value to search: ";
            cin >> data;
            if (search(head, data)) {
                cout << "Value found in the list\n";
            } else {
                cout << "Value not found in the list\n";
            }
            break;
        case 9:
            cout << "Linked List: ";
            traverse(head);
            break;
        case 0:
            while (head) {
                deleteFirst(head);
            }
            return 0;
        default:
            cout << "Invalid choice" << endl;
    }
  }
}
```

OUTPUT:

```
1. Insert at Front
2. Insert at End
3. Insert at Position
4. Delete First
5. Delete End
6. Delete at Position
7. Sort
8. Search
9. Traverse
0. Exit
Enter choice: 1
Enter data: 24
data inserted
1. Insert at Front
2. Insert at End
3. Insert at Position
4. Delete First
5. Delete End
6. Delete at Position
7. Sort
8. Search
9. Traverse
0. Exit
Enter choice: 2
Enter data: 12
data inserted
1. Insert at Front
2. Insert at End
3. Insert at Position
4. Delete First
5. Delete End
6. Delete at Position
7. Sort
8. Search
9. Traverse
0. Exit
Enter choice: 3
Enter data and position: 62 0
data inserted
```

```
Enter choice: 9
Linked List: 62 -> 24 -> 12 -> NULL

1. Insert at Front
2. Insert at End
3. Insert at Position
4. Delete First
5. Delete End
6. Delete at Position
7. Sort
8. Search
9. Traverse
0. Exit
Enter choice: 8
Enter value to search: 12
Value found in the list

1. Insert at Front
2. Insert at End
3. Insert at Position
4. Delete First
5. Delete End
6. Delete at Position
7. Sort
8. Search
9. Traverse
0. Exit
Enter choice: 7
12 -> 24 -> 62 -> NULL

1. Insert at Front
2. Insert at End
3. Insert at Position
4. Delete First
5. Delete End
6. Delete at Position
7. Sort
8. Search
9. Traverse
0. Exit
Enter choice: 4
data deleted
```

```
Enter choice: 6
Enter position: 1
node deleted
1. Insert at Front
2. Insert at End
3. Insert at Position
4. Delete First
5. Delete End
6. Delete at Position
7. Sort
8. Search
9. Traverse
0. Exit
Enter choice: 4
data deleted
1. Insert at Front
2. Insert at End
3. Insert at Position
4. Delete First
5. Delete End
6. Delete at Position
7. Sort
8. Search
9. Traverse
0. Exit
Enter choice: 9
Linked List: NULL
```

2. Write a program for stack using linked list

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

class Stack {
private:
    Node* top;
public:
    Stack() : top(nullptr) {}

     void push(int value) {
        Node* newNode = new Node();
        newNode->data = value;
        newNode->next = top;
        top = newNode;
        cout << value << " pushed to stack" << endl;
```

```
    }
    void pop() {
        if (isEmpty()) {
            cout << "Stack is empty" << endl;
            return;
        }
        Node* temp = top;
        top = top->next;
        cout << temp->data << " popped from stack" << endl;
        delete temp;
    }

        void peek() {
        if (isEmpty()) {
            cout << "Stack is empty" << endl;
            return;
        }
        cout << "Top element is " << top->data << endl;
    }

    void printStack() {
        if (isEmpty()) {
            cout << "Stack is empty" << endl;
            return;
        }
        Node* current = top;
        cout << "Stack elements are: ";
        while (current != nullptr) {
            cout << current->data << " ";
            current = current->next;
        }
        cout << endl;
    }

    bool isEmpty() {
        return top == nullptr;
    }

     ~Stack() {
        while (!isEmpty()) {
            pop();
        }
    }
};

int main() {
    Stack stack;
    int choice, value;
```

```
    while (true) {
        cout << "\nStack Operations Menu:\n";
        cout << "1. Push\n";
        cout << "2. Pop\n";
        cout << "3. Peek\n";
        cout << "4. Print Stack\n";
        cout << "5. Exit\n";
        cout << "Enter your choice: ";
        cin >> choice;

        switch (choice) {
            case 1:
                cout << "Enter value to push: ";
                cin >> value;
                stack.push(value);
                break;
            case 2:
                stack.pop();
                break;
            case 3:
                stack.peek();
                break;
            case 4:
                stack.printStack();
                break;
            case 5:
                return 0;
            default:
                cout << "Invalid choice. Please enter a number between 1 and 5." << endl;
        }
    }

    return 0;
}
```

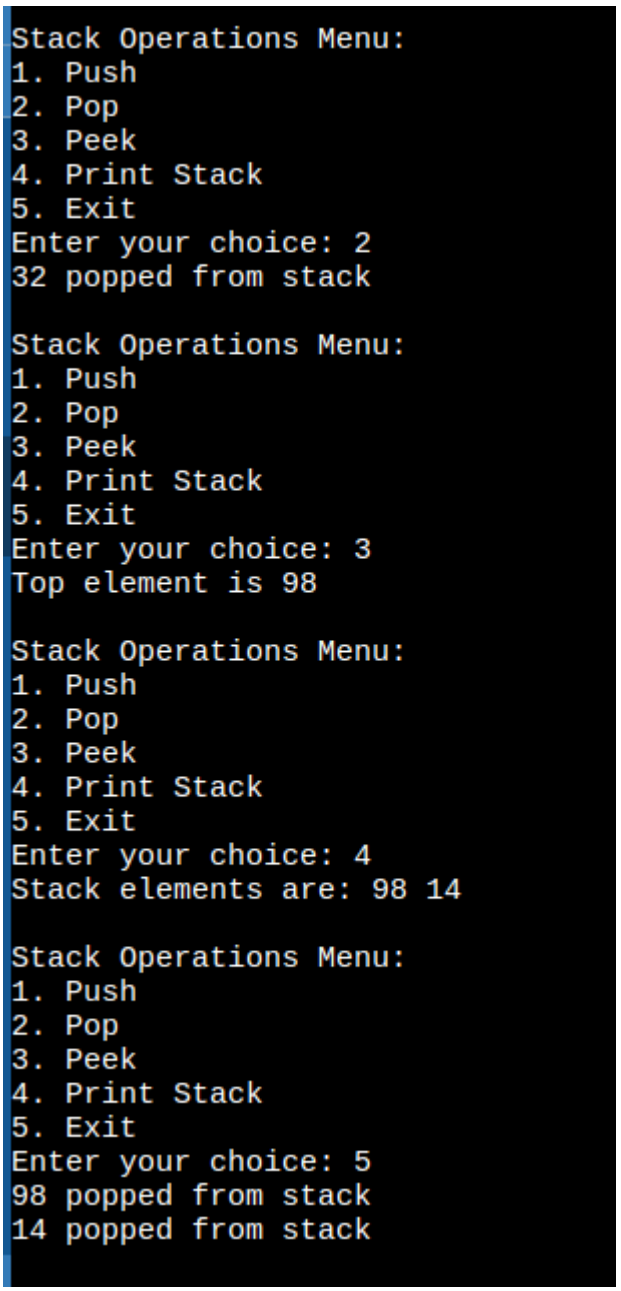
OUTPUT:

```
Stack Operations Menu:
1. Push
2. Pop
3. Peek
4. Print Stack
5. Exit
Enter your choice: 1
Enter value to push: 14
14 pushed to stack

Stack Operations Menu:
1. Push
2. Pop
3. Peek
4. Print Stack
5. Exit
Enter your choice: 1
Enter value to push: 98
98 pushed to stack

Stack Operations Menu:
1. Push
2. Pop
3. Peek
4. Print Stack
5. Exit
Enter your choice: 1
Enter value to push: 32
32 pushed to stack

Stack Operations Menu:
1. Push
2. Pop
3. Peek
4. Print Stack
5. Exit
Enter your choice: 4
Stack elements are: 32 98 14
```



3. Write a program for Queue using linked list

```
#include <iostream>
using namespace std;
struct Node {
    int data;
    Node* next;
};
class Queue {
private:
    Node* front;
    Node* rear;
```

```
public:
    Queue() : front(nullptr), rear(nullptr) {}

    void enqueue(int value) {
        Node* newNode = new Node();
        newNode->data = value;
        newNode->next = nullptr;
        if (rear == nullptr) {
            front = rear = newNode;
        } else {
            rear->next = newNode;
            rear = newNode;
        }
        cout << value << " enqueued to queue" << endl;
    }

      void dequeue() {
        if (isEmpty()) {
            cout << "Queue is empty" << endl;
            return;
        }
        Node* temp = front;
        front = front->next;
        if (front == nullptr) {
            rear = nullptr;
        }
        cout << temp->data << " dequeued from queue" << endl;
        delete temp;
    }


    void frontElement() {
        if (isEmpty()) {
            cout << "Queue is empty" << endl;
            return;
        }
        cout << "Front element is " << front->data << endl;
    }

    void printQueue() {
        if (isEmpty()) {
            cout << "Queue is empty" << endl;
            return;
        }
        Node* current = front;
        cout << "Queue elements are: ";
        while (current != nullptr) {
```

```
            cout << current->data << " ";
            current = current->next;
        }
        cout << endl;
    }

    bool isEmpty() {
        return front == nullptr;
    }

    ~Queue() {
        while (!isEmpty()) {
            dequeue();
        }
    }
};

int main() {
    Queue queue;
    int choice, value;

    while (true) {
        cout << "\nQueue Operations Menu:\n";
        cout << "1. Enqueue\n";
        cout << "2. Dequeue\n";
        cout << "3. Front Element\n";
        cout << "4. Print Queue\n";
        cout << "5. Exit\n";
        cout << "Enter your choice: ";
        cin >> choice;

        switch (choice) {
            case 1:
                cout << "Enter value to enqueue: ";
                cin >> value;
                queue.enqueue(value);
                break;
            case 2:
                queue.dequeue();
                break;
            case 3:
                queue.frontElement();
                break;
            case 4:
                queue.printQueue();
                break;
            case 5:
                return 0;
```

```
            default:
                cout << "Invalid choice. Please enter a number between 1 and 5." << endl;
        }
    }

    return 0;
}
```

OUTPUT:

```
Queue Operations Menu:
1. Enqueue
2. Dequeue
3. Front Element
4. Print Queue
5. Exit
Enter your choice: 1
Enter value to enqueue: 15
15 enqueued to queue

Queue Operations Menu:
1. Enqueue
2. Dequeue
3. Front Element
4. Print Queue
5. Exit
Enter your choice: 1
Enter value to enqueue: 56
56 enqueued to queue

Queue Operations Menu:
1. Enqueue
2. Dequeue
3. Front Element
4. Print Queue
5. Exit
Enter your choice: 1
Enter value to enqueue: 32
32 enqueued to queue

Queue Operations Menu:
1. Enqueue
2. Dequeue
3. Front Element
4. Print Queue
5. Exit
Enter your choice: 4
Queue elements are: 15 56 32
```

```
Queue Operations Menu:
1. Enqueue
2. Dequeue
3. Front Element
4. Print Queue
5. Exit
Enter your choice: 2
15 dequeued from queue

Queue Operations Menu:
1. Enqueue
2. Dequeue
3. Front Element
4. Print Queue
5. Exit
Enter your choice: 3
Front element is 56

Queue Operations Menu:
1. Enqueue
2. Dequeue
3. Front Element
4. Print Queue
5. Exit
Enter your choice: 5
56 dequeued from queue
32 dequeued from queue
```