Date : 10.02.2025
Lakshana Baskaran
24011103026

LAB - 4
List ADT - Singly Linked List

QUESTION 1:

A. Doubly Linked List: Write a C++ menu-driven program to implement List ADT using a doubly linked list with a tail.
Maintain proper boundary conditions and follow good coding practices. The List ADT has the following operations:

1. Insert Beginning
2. Insert End
3. Insert Position
4. Delete Beginning
5. Delete End
6. Delete Position
7. Search
8. Display
9. Exit

SOURCE CODE:

```
//Implemention of list ADT using doubly linked lists

#include <stdio.h>
#include <iostream>
#include <stdlib.h>
using namespace std;

class node {
    private:
        int data;
        struct node* next;
        struct node* prev;

        struct node* head;
        struct node* tail;

    public:
        void insert_beginning(int);
        void insert_end(int);
        void insert_pos(int, int);

        int delete_beginning();
        int delete_end();
        int delete_pos(int);

        bool search(int);
        void print();
        void print_rev();



    node() {
        head = NULL;
        tail = NULL;
    }

};

//insertion of a of node containing value of x at the beginning of
the DLL
void node::insert_beginning(int x) {
    struct node* newnode = new struct node;
    newnode -> data = x;

    newnode -> next = head;
    newnode -> prev = NULL;

    if (head != NULL) {
        head -> prev = newnode;
    }
    head = newnode;

    if (tail == NULL) {
        tail = newnode;
    }
}

//insertion of a of node containing value of x at the end of the DLL
void node::insert_end(int x) {
    if (head == NULL) {
        insert_beginning(x);
        return;
    }

    struct node* newnode = new struct node;
    newnode -> data = x;

    newnode -> next = NULL;
    newnode -> prev = tail;
    tail -> next = newnode;
    tail = newnode;

    //cout << tail -> data << endl;
}

//insertion of a of node containing value of x at position pos of the
DLL
void node::insert_pos(int x, int pos) {
    if (head == NULL) {
        cout << "List is empty.\n";
        return;
    }

    if (pos == 1) {
        insert_beginning(x);
        return;
    }

    node* temp = head;
    for (int i = 1; i < pos; i++, temp = temp -> next) {
        if (temp -> next == NULL) {
            cout << "position out of range.\n";
            return;
        }
    }

    struct node* newnode = new struct node;
    newnode -> data = x;

    newnode -> prev = temp -> prev;
    newnode -> next = temp;
    temp -> prev -> next = newnode;
    temp -> prev = newnode;
```

```
}

//Deletes the first element of the DLL
int node::delete_beginning() {
    if (head == NULL) {
        cout << "UnderFlowError: List is Empty\n";
        return 0;
    }

    int popped = head -> data;
    if (head -> next == NULL) {
        head = NULL;
        tail = NULL;
        return popped;
    }

    head = head -> next;
    head -> prev = NULL;

    return popped;
}

//Deletes the last element of the DLL
int node::delete_end() {
    if (head == NULL) {
        cout << "UnderFlowError: List is Empty\n";
        return 0;
    }

    int popped = tail -> data;
    if (tail -> prev == NULL) {
        head = NULL;
        tail = NULL;
        return popped;
    }

    tail = tail -> prev;
    tail -> next = NULL;

    return popped;
}

//deletes the element from the specified position from the DLL
```

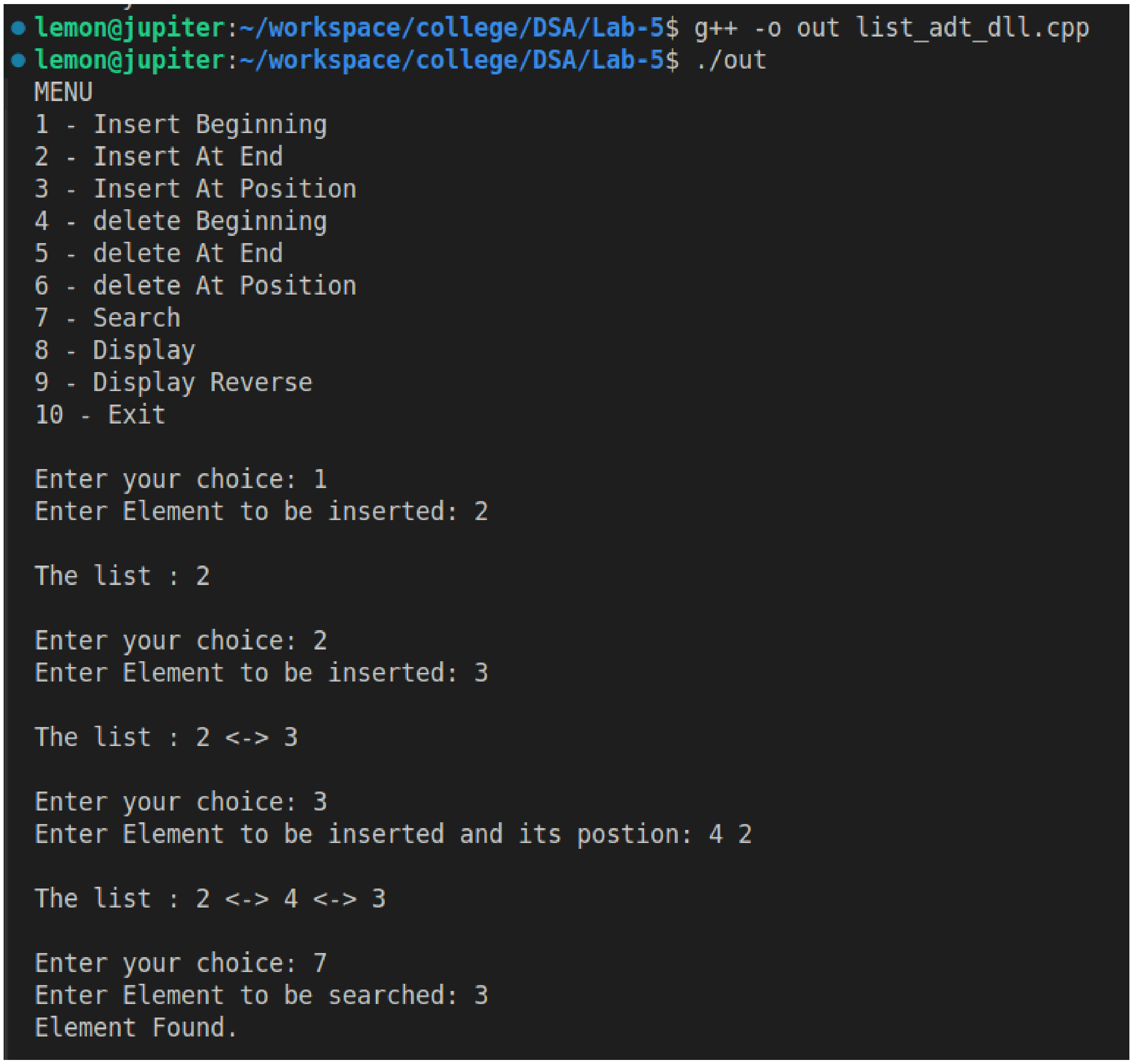
Date : 10.02.2025
Lakshana Baskaran
24011103026

```
int node::delete_pos(int pos) {
    if (head == NULL) {
        cout << "UnderFlowError: List is Empty\n";
        return 0;
    }

    if (pos == 1) {
        delete_beginning();
        return 0;
    }

    node* temp = head;
    for (int i = 1; i < pos; i++, temp = temp -> next) {
        if (temp -> next == NULL) {
            cout << "position out of range.\n";
            return 0;
        }
    }

    if (temp -> next == NULL) {
        delete_end();
        return 0;
    }

    temp -> prev -> next = temp -> next;
    temp -> next -> prev = temp -> prev;

    return 0;
}

//checks if element x is present in the DLL or not
bool node::search(int x) {
    node* temp = head;
    if (head == NULL) {
        return false;
    }

    for (; temp -> next != NULL; temp = temp -> next) {
        if (temp -> data == x) {
            return true;
        }
    }
    if (temp -> data == x) {
        return true;
    }

    return false;

}

//displaying the reverse of the doubly linked list
void node::print_rev() {
    node* temp = tail;
    if (head == NULL) {
        cout << "List is Empty.\n";
        return;
    }

    //cout << "NULL <- ";
    for (; temp -> prev != NULL; temp = temp -> prev) {
        cout << temp -> data << " <-> ";
    }
    cout << temp -> data << "\n";
}

//displaying the doubly linked list
void node::print() {
    node* temp = head;
    if (head == NULL) {
        cout << "List is Empty.\n";
        return;
    }
    //cout << "NULL <- ";
    for (; temp -> next != NULL; temp = temp -> next) {
        cout << temp -> data << " <-> ";
    }
    cout << temp -> data << "\n";
}

int main() {
    node L;
    int x, pos, choice;
    printf("MENU\n1 - Insert Beginning\n2 - Insert At End\n3 - Insert 
At Position\n");
    printf("4 - delete Beginning\n5 - delete At End\n6 - delete At 
Position\n");
```

```
    printf("7 - Search\n8 - Display\n9 - Display Reverse\n10 -
Exit\n");

    while (true) {
        printf("\nEnter your choice: ");
        scanf("%d", &choice);
        switch (choice) {
            case 1:
                printf("Enter Element to be inserted: ");
                scanf("%d", &x);
                L.insert_beginning(x);
                break;

            case 2:
                printf("Enter Element to be inserted: ");
                scanf("%d", &x);
                L.insert_end(x);
                break;

            case 3:
                printf("Enter Element to be inserted and its postion:
");

                scanf("%d", &x);
                scanf("%d", &pos);
                L.insert_pos(x, pos);
                break;

            case 4:
                printf("%d\n", L.delete_beginning());
                break;

            case 5:
                printf("%d\n", L.delete_end());
                break;

            case 6:
                printf("Enter the postion for deletion: ");
                scanf("%d", &pos);
                printf("%d \n", L.delete_pos(pos));
                break;

            case 7:
                printf("Enter Element to be searched: ");
                scanf("%d", &x);

                if (L.search(x) == 1) {
                    printf("Element Found.\n");
                }
                else {
                    printf("Element Not Found.\n");
                }
                break;

            case 8:
                cout << "DISPLAYING LIST: ";
                L.print();
                break;

            case 9:
                cout << "REVERSE OF LIST: ";
                L.print_rev();
                break;

            case 10:
                printf("Exiting...\n");
                return 0;
                break;

            default:
                //printf("\nInvalid choice. Enter again.\n");
                break;
        }
        printf("\nThe list : ");
        L.print();
    }

    return 0;
}
```

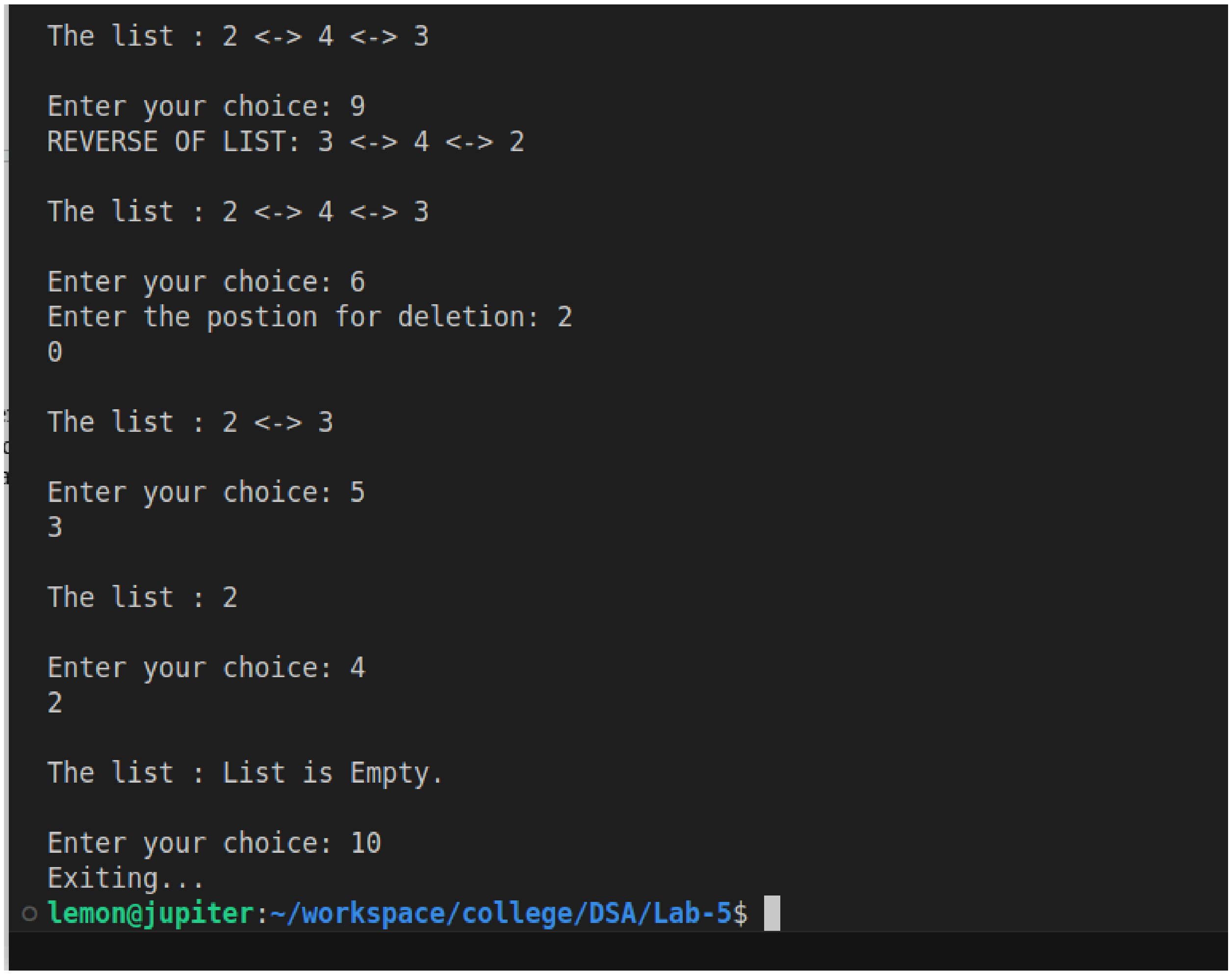
Date : 10.02.2025
Lakshana Baskaran
24011103026

OUTPUT:



```
● lemon@jupiter:~/workspace/college/DSA/Lab-5$ g++ -o out list_adt_dll.cpp
● lemon@jupiter:~/workspace/college/DSA/Lab-5$ ./out
  MENU
  1 - Insert Beginning
  2 - Insert At End
  3 - Insert At Position
  4 - delete Beginning
  5 - delete At End
  6 - delete At Position
  7 - Search
  8 - Display
  9 - Display Reverse
  10 - Exit

  Enter your choice: 1
  Enter Element to be inserted: 2

  The list : 2

  Enter your choice: 2
  Enter Element to be inserted: 3

  The list : 2 <-> 3

  Enter your choice: 3
  Enter Element to be inserted and its postion: 4 2

  The list : 2 <-> 4 <-> 3

  Enter your choice: 7
  Enter Element to be searched: 3
  Element Found.
```

Date : 10.02.2025
Lakshana Baskaran
24011103026



```
The list : 2 <-> 4 <-> 3

Enter your choice: 9
REVERSE OF LIST: 3 <-> 4 <-> 2

The list : 2 <-> 4 <-> 3

Enter your choice: 6
Enter the postion for deletion: 2
0

The list : 2 <-> 3

Enter your choice: 5
3

The list : 2

Enter your choice: 4
2

The list : List is Empty.

Enter your choice: 10
Exiting...
lemon@jupiter:~/workspace/college/DSA/Lab-5$
```

Date : 10.02.2025
Lakshana Baskaran
24011103026

QUESTION 2:

B. Circular Linked List: Write a C++ menu-driven program to implement List ADT using a circular linked list. Maintain proper boundary conditions and follow good coding practices. The List ADT has the following operations:

1. Insert Beginning
2. Insert End
3. Insert Position
4. Delete Beginning
5. Delete End
6. Delete Position
7. Search
8. Display
9. Exit

SOURCE CODE:

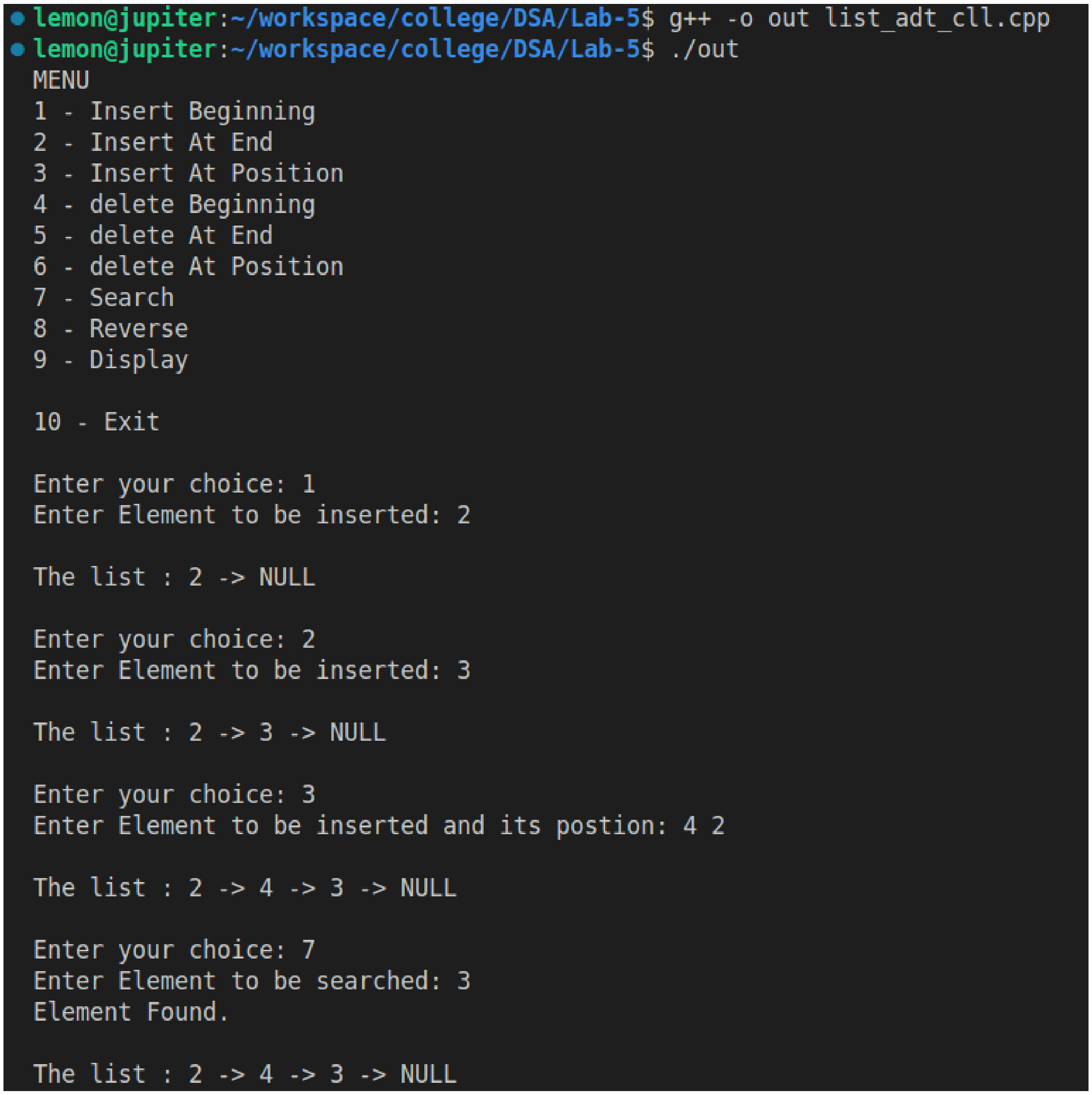
```
//Singly Linked List using list ADT
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>
using namespace std;

//class definition for CLL
class node {
    private:
        int data;
        struct node *next;

    public:
        void insert_beginning(int);
        void insert_end(int);
        void insert_position(int, int);

        int delete_beginning();
        int delete_end();
        int delete_position(int);

        bool search(int);

        void reverse_display(struct node*);
        node* reverse(node*);
        int len();
        void print();

} *head = NULL;

//method to insert element x at the start of the CLL
void node::insert_beginning(int x) {
    struct node* newnode = (struct node*) malloc (sizeof(struct
node));
    newnode -> data = x;
    newnode -> next = head;
    head = newnode;
}

//method to insert element x at the end of the CLL
void node::insert_end(int x) {
    struct node* newnode = (struct node*) malloc (sizeof(struct
node));
    newnode -> data = x;
    newnode -> next = NULL;

    if (head == NULL) {
        head = newnode;
        return;
    }

    struct node* temp = head;
    for (; temp -> next != NULL; temp = temp -> next) {
    }
    temp -> next = newnode;

}

//method to insert element x at the specified position in the CLL
void node::insert_position(int x, int pos) {
    struct node *newnode = (struct node*) malloc (sizeof(struct
node));

    if (pos > len() || pos < 1) {
        printf("Invalid Postion.\n");
        return;
    }
    if (pos == 1) {
        insert_beginning(x);
        return;
    }

    newnode -> data = x;
    struct node* temp = head;
    for (int i = 1; (temp -> next != NULL) && i < pos-1; i++) {
        temp = temp -> next;
    }
    newnode -> next = temp -> next;
    temp -> next = newnode;
}

//method to delete the first element of the CLL
int node::delete_beginning() {
    if (head == NULL) {
        printf("UnderFlow Error: List is Empty.\n");
        return 0;
    }

    int elem;
    elem = head -> data;
    head = head -> next;

    return elem;
}

//method to delete last element of the CLL
int node::delete_end() {
    if (head == NULL) {
        printf("UnderFlow Error: List is Empty.\n");
        return 0;
    }

    int elem;
    if (head -> next == NULL) {
        elem = head -> data;
        head = NULL;
        return elem;
    }
```

Date : 10.02.2025
Lakshana Baskaran
24011103026

```
    struct node* temp = head;
    for (;temp -> next -> next != NULL; temp = temp -> next) {

    }

    elem = temp -> next -> data;
    temp -> next = NULL;

    return elem;
}

//method to delete the element present in the specified position
int node::delete_position(int pos) {
    if (head == NULL) {
        printf("\nUnderFlow Error: List is Empty.\n");
        return 0;
    }

    if (pos > len() || pos < 0) {
        printf("\nInvalid Position.\n");
        return 0;
    }
    int elem;

    struct node* temp = head;
    if (pos == 1) {
        elem = head -> data;
        head = head -> next;
        return elem;
    }
    for (int i = 1; temp -> next != NULL && i < pos-1; i++) {
        temp = temp -> next;
    }

    elem = temp -> next -> data;
    temp -> next = temp -> next -> next;

    return elem;
}

//method to check if element x is presentin the CLL or not
bool node::search(int x) {
    struct node* temp = head;
    for (int i = 0; temp -> next != NULL; temp = temp -> next){
        if (temp -> data == x) {
            return 1;
        }
        i++;
    }
    if (temp -> data == x) {
        return 1;
    }

    return 0;
}

//method to display the reverse of the CLL
void node::reverse_display(struct node* temp) {
    if (temp -> next == NULL) {
        printf("%d ", temp -> data);
        return;
    }

    reverse_display(temp->next);
    printf("<- %d ", temp->data);
}

//method to reverse the CLL
node* node::reverse(node *head) {
    node *curr = head, *P = NULL, *N;

    while (curr != NULL) {
        N = curr->next;
        curr->next = P;

        P = curr;
        curr = N;
    }

    return P;
}

//method to find the number of elements in the CLL
int node::len() {
    int len = 0;
    struct node* temp = head;
    for (; temp -> next != NULL; temp = temp -> next) {
        len++;
    }

    return len+1;
}

//method to display the singly liked list
void node::print() {
    struct node *temp = head;
    if (temp == NULL) {
        printf("head -> NULL\n");
        return;
    }

    //printf("head -> ");
    for (; temp -> next != NULL; temp = temp -> next) {
        printf("%d -> ", temp -> data);
    }
    printf("%d -> NULL\n", temp -> data);
}


int main() {
    node L;
    int x, pos, choice = 0;
    printf("MENU\n1 - Insert Beginning\n2 - Insert At End\n3 - Insert
At Position\n");
    printf("4 - delete Beginning\n5 - delete At End\n6 - delete At
Position\n");
    printf("7 - Search\n8 - Reverse\n9 - Display\n\n10 - Exit\n");

    while (choice != 10) {
        printf("\nEnter your choice: ");
        scanf("%d", &choice);
        switch (choice) {
            case 1:
                printf("Enter Element to be inserted: ");
                scanf("%d", &x);
                L.insert_beginning(x);
                break;

            case 2:
                printf("Enter Element to be inserted: ");
                scanf("%d", &x);
                L.insert_end(x);
                break;

            case 3:
                printf("Enter Element to be inserted and its postion: 
");
                
                scanf("%d", &x);
                scanf("%d", &pos);
                L.insert_position(x, pos);
                break;

            case 4:
                printf("%d\n", L.delete_beginning());
                break;

            case 5:
                printf("%d\n", L.delete_end());
                break;

            case 6:
                printf("Enter the postion for deletion: ");
                scanf("%d", &pos);
                printf("%d \n", L.delete_position(pos));
                break;

            case 7:
                printf("Enter Element to be searched: ");
                scanf("%d", &x);

                if (L.search(x) == 1) {
                    printf("Element Found.\n");
                }
                else {
                    printf("Element Not Found.\n");
                }
                break;

            case 8:
                printf("Reverse of the list: ");
                L.reverse_display(head);
                printf("\n");
                break;

            case 9:
                head = L.reverse(head);
                break;

            case 10:
                printf("Exiting...\n");
                break;

            default:
                //printf("\nInvalid choice. Enter again.\n");
                break;
        }
        if (choice != 10) {
            if (head == NULL) {
                printf("NULL");
            }
            else {
                printf("\nThe list : ");
                L.print();
            }
        }
    }
}
```

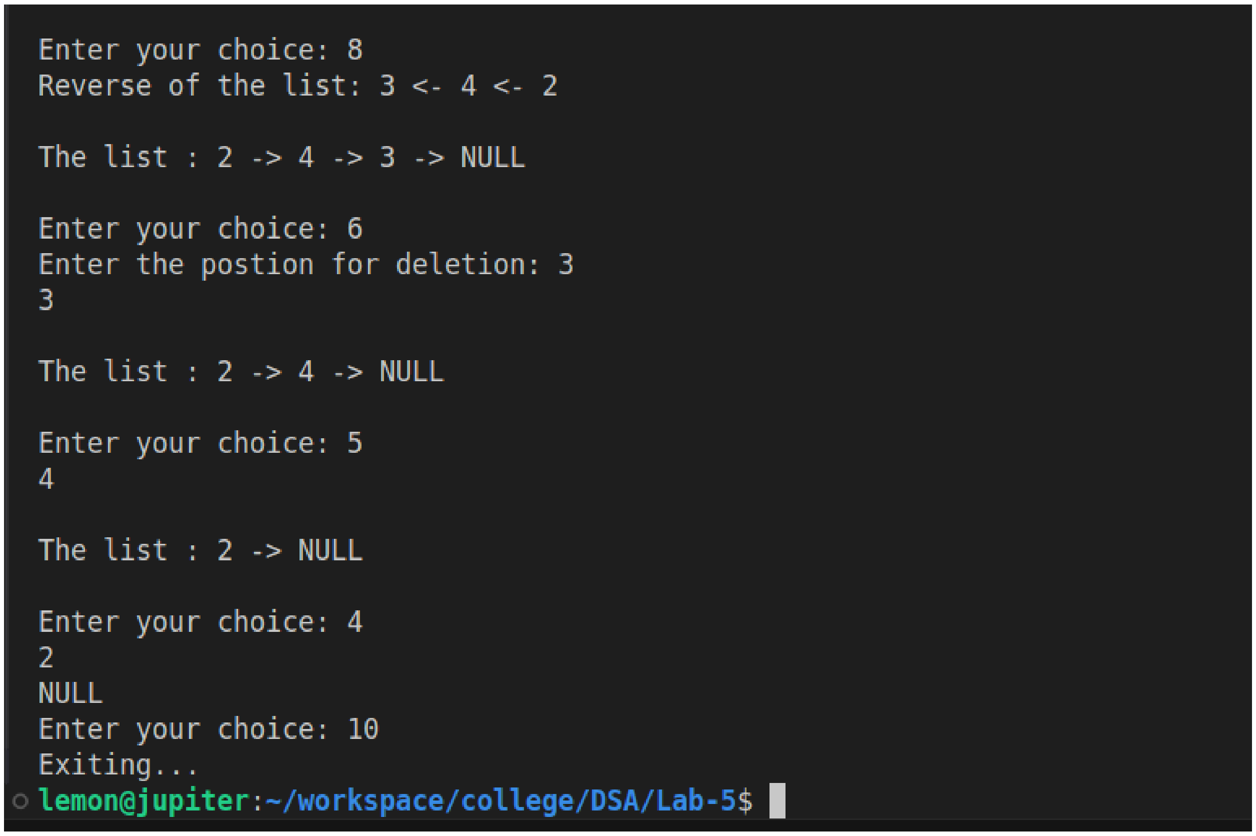
Date : 10.02.2025
Lakshana Baskaran
24011103026

SOURCE CODE:



```
● lemon@jupiter:~/workspace/college/DSA/Lab-5$ g++ -o out list_adt_cll.cpp
● lemon@jupiter:~/workspace/college/DSA/Lab-5$ ./out
  MENU
  1 - Insert Beginning
  2 - Insert At End
  3 - Insert At Position
  4 - delete Beginning
  5 - delete At End
  6 - delete At Position
  7 - Search
  8 - Reverse
  9 - Display

  10 - Exit

  Enter your choice: 1
  Enter Element to be inserted: 2

  The list : 2 -> NULL

  Enter your choice: 2
  Enter Element to be inserted: 3

  The list : 2 -> 3 -> NULL

  Enter your choice: 3
  Enter Element to be inserted and its postion: 4 2

  The list : 2 -> 4 -> 3 -> NULL

  Enter your choice: 7
  Enter Element to be searched: 3
  Element Found.

  The list : 2 -> 4 -> 3 -> NULL
```

Date : 10.02.2025
Lakshana Baskaran
24011103026



```
Enter your choice: 8
Reverse of the list: 3 <- 4 <- 2

The list : 2 -> 4 -> 3 -> NULL

Enter your choice: 6
Enter the postion for deletion: 3
3

The list : 2 -> 4 -> NULL

Enter your choice: 5
4

The list : 2 -> NULL

Enter your choice: 4
2
NULL
Enter your choice: 10
Exiting...
lemon@jupiter:~/workspace/college/DSA/Lab-5$
```

Date : 10.02.2025
Lakshana Baskaran
24011103026

QUESTION 3:

C. Round Robin Scheduling Simulation: An operating system allocates a fixed time slot CPU time for processes using a Round-Robin scheduling algorithm. The fixed time slot will be initialized before the start of the menu-driven program. Implement the Round-Robin scheduling algorithm using the circular linked list. Implement the program with the following operations:

1. Insert Process
2. Execute
3. Exit

The "Insert Process" will get an integer time from the user and add it to the queue.
The "Execute" operation will execute a deletion at the beginning, subtract the fixed time from the process.
If the processing time falls below 0, the process is considered completed.
Otherwise, the remaining time after subtraction is inserted at the end of the circular linked list.

SOURCE CODE:

//".h file":

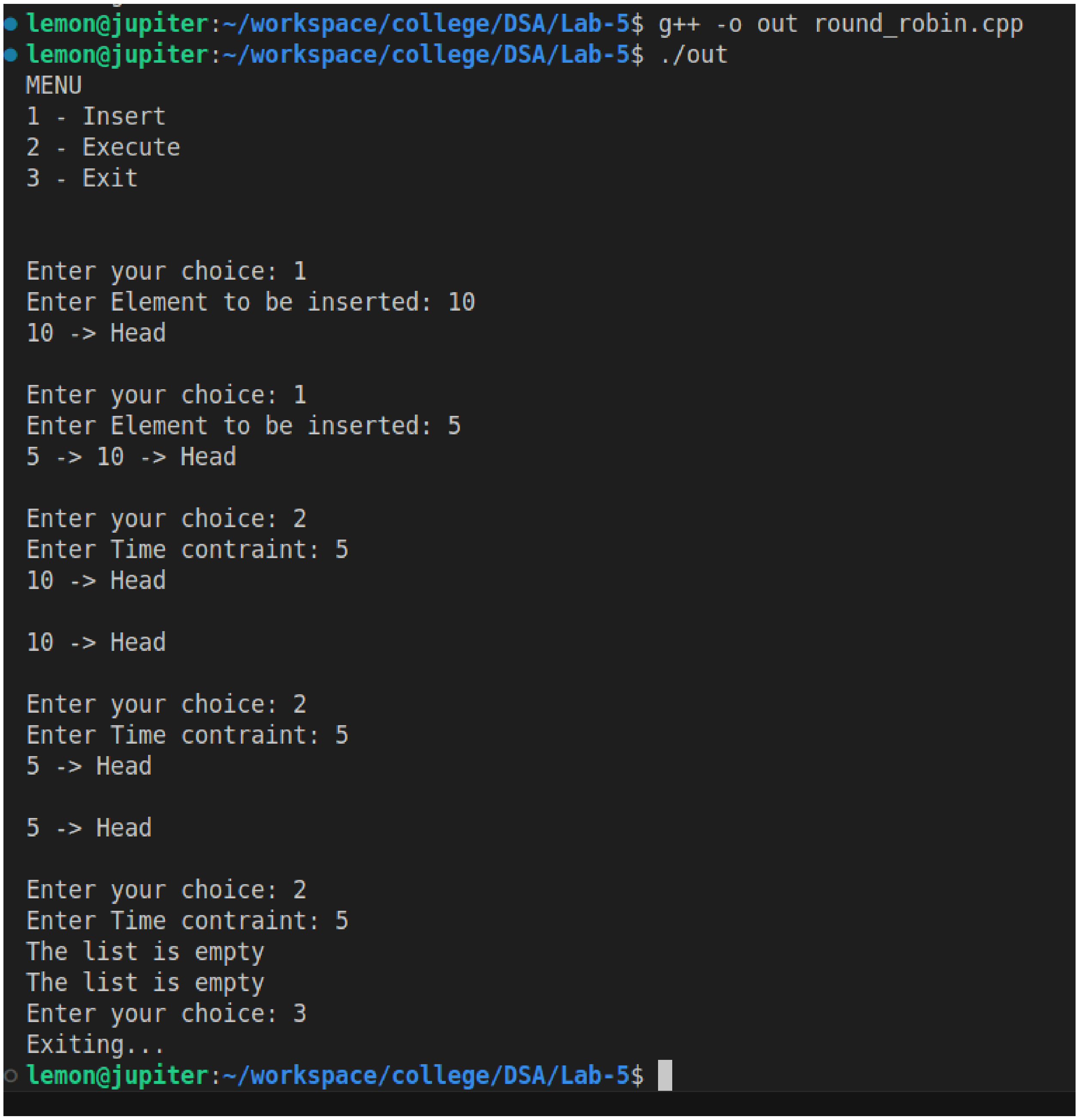
```
//methods that help simulate a round robin scheduling process
#include<iostream>
#include<stdio.h>
using namespace std;


#include<cstdio>
#include<cstdlib>

class List{
    private:
    struct Node{
    int data;
    Node* next;

    };
    struct Node *head;

    public:
        void insert(int);
        void insertatend(int);
        void deleteatbeg();
        void display();
        void execute(int);

    List(){
        head = NULL;
    }

};

//inserts element into the queue
void List::insert(int num){
    struct Node* newNode = (struct Node*)malloc(sizeof(Node));
    newNode->data = num;
    if(head == NULL){
        newNode->next = newNode;
        head = newNode;
    }
    else{
        Node*temp = head;
        while(temp-> next!= head ){
            temp = temp ->next;}

            newNode->next = head;
            temp -> next = newNode;
            head = newNode ;

    }
}


void List::insertatend(int num){
    struct Node* newNode = (struct Node*)malloc(sizeof(Node));
    newNode->data = num;
    if(head == NULL){
        newNode -> next = newNode;
        head = newNode;}
    else{
        Node*temp = head;
        while(temp ->next != head){
            temp = temp->next;
            }

            temp ->next = newNode;
            newNode -> next = head;
    }
}


void List::deleteatbeg(){
    if (head == NULL){
        printf("The list is empty\n");
        return;
    }
    if(head->next == head){
    free(head);
    head = NULL;
    return;}

    Node* temp = head;
    while(temp->next != head){
        temp = temp->next;}

        Node* deletea = head;
        head = head ->next;
        temp ->next = head;
        free(deletea);

}

void List::display(){
    if(head == NULL){
        printf("The list is empty");
    }

    else{
        Node* temp = head;
        while(temp -> next  != head){
            printf("%d -> ", temp->data);
            temp = temp ->next;}

        printf("%d -> Head\n", temp->data);
    }
}

void List::execute(int x) {
    if (head == NULL) {
        return;
    }

    if ((head -> data) - x <= 0) {
        deleteatbeg();
    }

    else {
        int a = (head -> data) - x;
        deleteatbeg();
        insertatend(a);
    }

    display();
    cout << endl;
}
```

//".cpp file":

```
#include "robin.h"

int main() {
    List L;
    int x, pos, choice = 0;
    printf("MENU\n1 - Insert\n2 - Execute\n3 - Exit\n\n");

    while (choice != 3) {
        printf("\nEnter your choice: ");
        scanf("%d", &choice);
        switch (choice) {
            case 1:
                printf("Enter Element to be inserted: ");
                scanf("%d", &x);
                L.insert(x);
                L.display();
                break;

            case 2:
                printf("Enter Time contraint: ");
                scanf("%d", &x);
                L.execute(x);
                L.display();
                break;
            case 3:
                printf("Exiting...\n");
                break;
            default:
                printf("invalid option\n");
                break;
        }
    }
}
```

Date : 10.02.2025
Lakshana Baskaran
24011103026

SOURCE CODE:



```
lemon@jupiter:~/workspace/college/DSA/Lab-5$ g++ -o out round_robin.cpp
lemon@jupiter:~/workspace/college/DSA/Lab-5$ ./out
MENU
1 - Insert
2 - Execute
3 - Exit


Enter your choice: 1
Enter Element to be inserted: 10
10 -> Head

Enter your choice: 1
Enter Element to be inserted: 5
5 -> 10 -> Head

Enter your choice: 2
Enter Time contraint: 5
10 -> Head

10 -> Head

Enter your choice: 2
Enter Time contraint: 5
5 -> Head

5 -> Head

Enter your choice: 2
Enter Time contraint: 5
The list is empty
The list is empty
Enter your choice: 3
Exiting...
lemon@jupiter:~/workspace/college/DSA/Lab-5$
```

–