Data Structures and Algorithms

Lab Journal - Lab 5

GITHUB LINK:

https://github.com/KingPanda5/DSA-lab-5.git

Name: SYED MURTAZA HAIDER

Enrollment #: 0 1 - 1 3 4 2 3 2 - 1 8 2

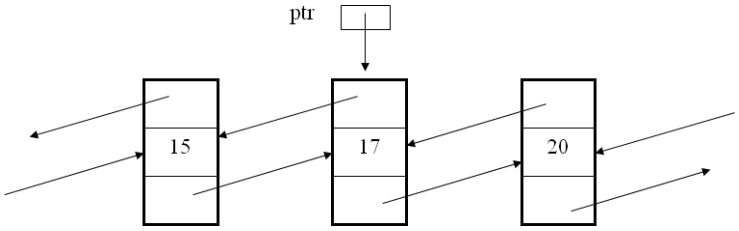
Class/Section: BS(CS) – 3D

Objective

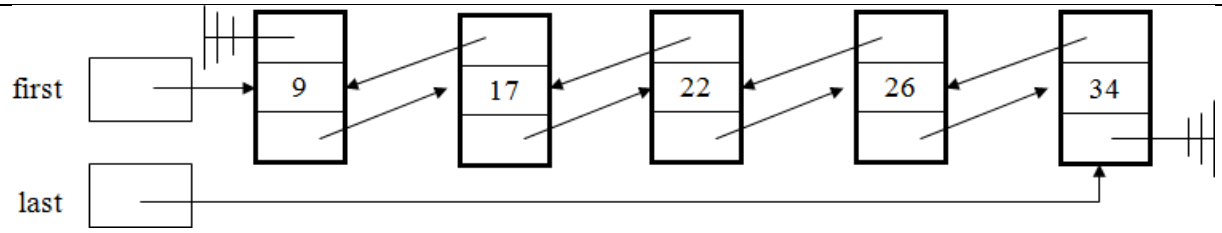
This lab session is aimed at introducing students to doubly linked lists.

Task 1 :

Give answers to the following.

1.	Redraw the following list after the given instructions are executed:
	 
	
	`ptr->next->prev = ptr->prev;`
	`ptr->prev->next = ptr->next;`
	`delete ptr;`

2.	Consider the following doubly linked list :



Write C++ statements to:

```
#include <iostream>
2
3using namespace std;
4
5struct Node {
6  int data;
7  Node *next;
8};
9
10int main() {
11  Node *first, *last, *temp, *n;
12  first = new Node;
13  last = new Node;
14  temp = new Node;
15  n = new Node;
16  first->data = 9;
17  first->next = temp;
18  temp->data = 17;
19  temp->next = n;
20  n->data = 22;
21  n->next = last;
22  last->data = 26;
23  last->next = nullptr;
```

a. Print the value 26 using the pointer ‘last’:

```
cout << "Value of last: " << last->data << endl;
```

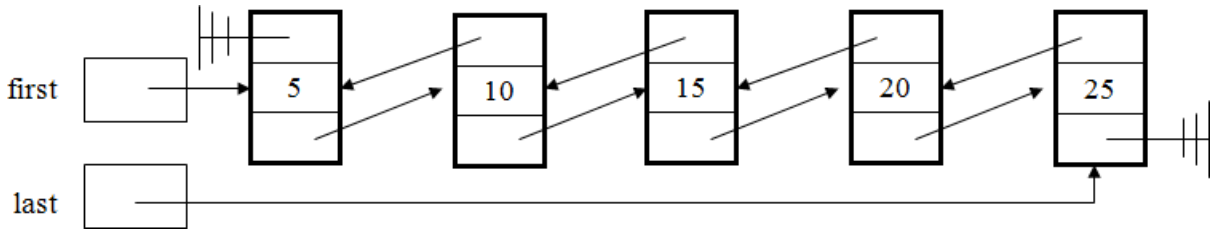
b. Print the value 17 using the pointer ‘first:

```
cout << "Value of first: " << first->next->data << endl;
```

c. Print the address of the node containing value 9 using the pointer 'last':

```
cout << "Address of last: " << last << endl;
```

3. Given the following linked list, state what does each of the following statements refer to?



`first->data;`	5
`last->next;`	NULL
`first->next->prev;`	NULL
`first->next->next->data;`	15
`last->prev->data;`	20

Task 2 :

Implement the following exercises.

Exercise 1

Implement the class Doubly Linked List to create a list of integers. You need to provide the implementation of the member functions as described in the following.

```
class DList
{
private:
      Node *head;
public:
      DList();
      // Checks if the list is empty or not
      bool emptyList();
      // Inserts a new node with value 'newV' after the node
      containing value 'oldV'. If a node with value 'oldV' does
      not exist, does not insert the new node.
      void insert_after(int oldV, int newV);
      // Deletes the node containing the specified value
      void deleteNode(int value);
      // Inserts a new node at the start of the list
      void insert_begin(int value);
      // Inserts a new node at the end of the list
      void insert_end(int value);
      // Displays the values stored in the list starting from head
      void traverse();
```

Cpp file:

```
#include <iostream>

using namespace std;


class Node {

public:

    int data;

    Node* prev;

    Node* next;
```

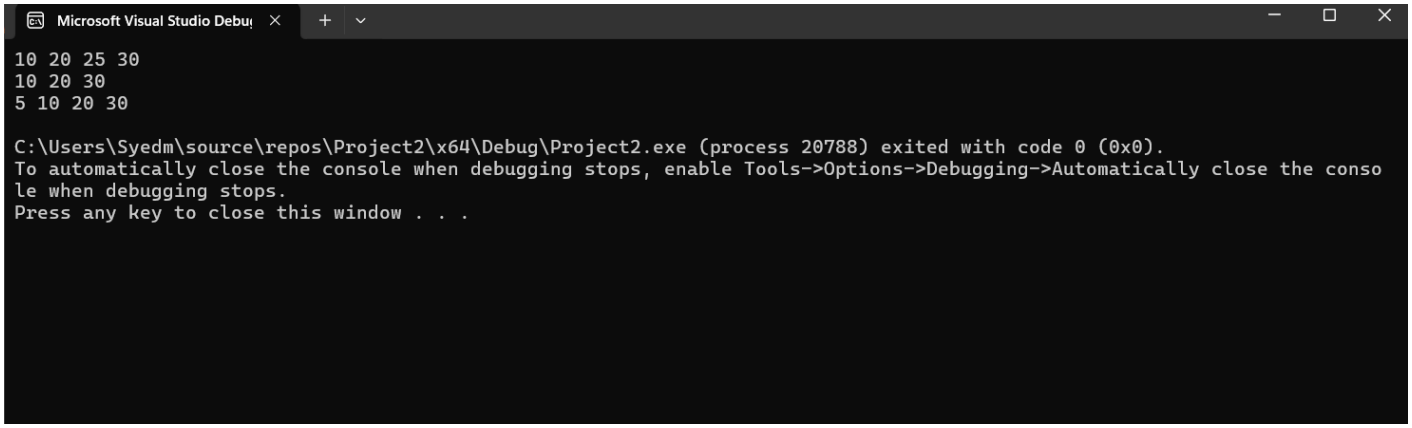
```
    Node(int value) : data(value), prev(nullptr), next(nullptr) {}
};

class DList {
private:
    Node* head;

public:
    DList() : head(nullptr) {}

    // Checks if the list is empty or not
    bool emptyList() {
        return head == nullptr;
    }

    // Inserts a new node at the start of the list
    void insert_begin(int value) {
        Node* newNode = new Node(value);
        if (emptyList()) {
            head = newNode;
        }
        else {
            newNode->next = head;
            head->prev = newNode;
            head = newNode;
        }
    }
```

```
// Inserts a new node at the end of the list
void insert_end(int value) {
    Node* newNode = new Node(value);
    if (emptyList()) {
        head = newNode;
    }
    else {
        Node* temp = head;
        while (temp->next != nullptr) {
            temp = temp->next;
        }
        temp->next = newNode;
        newNode->prev = temp;
    }
}

// Inserts a new node with value 'newV' after the node containing value 'oldV'
void insert_after(int oldV, int newV) {
    Node* temp = head;
    while (temp != nullptr && temp->data != oldV) {
        temp = temp->next;
    }
    if (temp != nullptr) { // oldV found
        Node* newNode = new Node(newV);
        newNode->next = temp->next;
        newNode->prev = temp;
        if (temp->next != nullptr) {
```

```
            temp->next->prev = newNode;
        }
        temp->next = newNode;
    }
}

// Deletes the node containing the specified value
void deleteNode(int value) {
    Node* temp = head;
    while (temp != nullptr && temp->data != value) {
        temp = temp->next;
    }
    if (temp == nullptr) return; // Value not found

    if (temp->prev != nullptr) {
        temp->prev->next = temp->next;
    }
    else {
        head = temp->next;
    }
    if (temp->next != nullptr) {
        temp->next->prev = temp->prev;
    }
    delete temp;
}

// Displays the values stored in the list starting from head
void traverse() {
```

```
        Node* temp = head;
        while (temp != nullptr) {
            cout << temp->data << " ";
            temp = temp->next;
        }
        cout << endl;
    }
};

int main() {
    DList list;
    list.insert_begin(10);
    list.insert_end(20);
    list.insert_end(30);
    list.insert_after(20, 25);
    list.traverse(); // Output: 10 20 25 30

    list.deleteNode(25);
    list.traverse(); // Output: 10 20 30

    list.insert_begin(5);
    list.traverse(); // Output: 5 10 20 30

    return 0;
}
```

OUTPUT:



Exercise 2

Write C++ functions to :

- **Reverse a doubly linked list.**
- **Display the contents of alternate nodes of doubly linked list.**

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

void reverse() {

    Node* temp = nullptr;

    Node* current = head;



    // Swap next and prev for all nodes of the list

    while (current != nullptr) {
```

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

    // Adjust head to be the last element
    if (temp != nullptr) {
        head = temp->prev;
    }
}

// Function to display contents of alternate nodes starting from head
void display_alternate() {
    Node* temp = head;
    bool display = true;
    while (temp != nullptr) {
        if (display) {
            cout << temp->data << " ";
        }
        display = !display;
        temp = temp->next;
    }
```

```
    cout << endl;
}
```

Implement the given exercises and get them checked by your instructor. If you are unable to complete the tasks in the lab session, deposit this journal alongwith your programs (printed) as per the submission date given.

S No.	Exercise	Checked By:

1.	Exercise 1	
2.	Exercise 2	

++++++++++++++++++++++++++