Fatima Jinnah Women University

Data Structures & Algorithms Lab

Lab # 9

SUBMITTED TO:

Sir Rehan Ahmed Siddiqui

SUBMITTED BY:

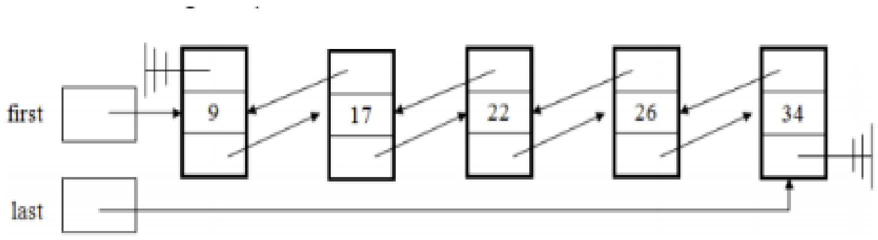
Tanzeela Asghar
2021-BSE-032

SECTION:

BSE III-A

Task 1:

1. **Consider the following doubly linked list**



Write C++ statements to:

a. **Print the value 26 using the pointer 'last':**

cout<<last->prev->data;

b. **Print the value 17 using the pointer 'first:**

cout<<first->next->data;

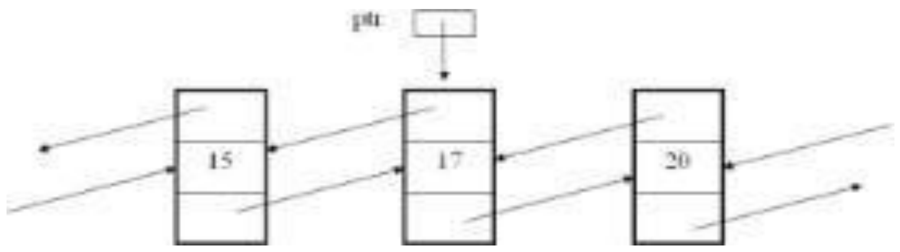
c. Print the address of the node containing value 9 using the pointer 'last':

cout<<last->prev-> prev-> prev-> prev->&;

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

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

3. Redraw the following list after the given instructions are executed:

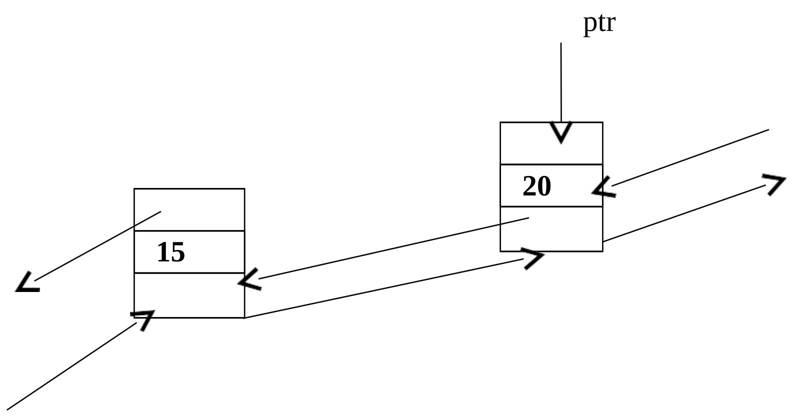


ptr->next->prev = ptr->prev;

ptr->next->prev = ptr->prev;

ptr->prev->next = ptr->next;

delete ptr;

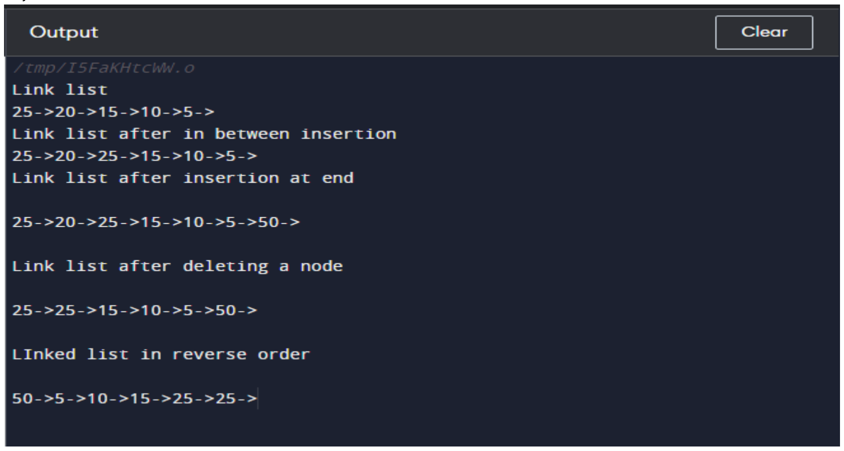


Code Exercise 1:

Implement the class Doubly Linked List, with all provided functions.

```
#include <iostream>
using namespace std;
class DList
{
private:
    struct node
    {
        int data;
        node* next;
        node* prev;
    } *head;
public:
    DList() {
        head = NULL;
    }
    ~DList() {}
    bool emptyList() {
        if (head == NULL)
            return true;
        else
            return false;

    }// Checks if the list is empty or not
    void insertafter(int oldV, int newV) {
        node* ptr;
        ptr = head;
        while (ptr->data != oldV)
        {
            ptr = ptr->next;
        }
        node* p;
        p = new node;
        p->data = newV;
        p->prev = ptr;
        p->next = ptr->next;
        ptr->next = p;
        p->next->prev = p;

    }

    void deleteNode(int value) {
    node* ptr;
        int flag = 0;
        ptr = head;
        if ((ptr->next == NULL) && (ptr->prev == NULL))
        {
            head = NULL;
            delete ptr;
        }
        else if (ptr->data == value)
        {
            head = head->next;
            ptr->next = NULL;
            head->prev = NULL;
            delete ptr;
        }
        else
        {
            while (ptr->next != NULL)
            {
                if (ptr->data == value)
                {
                    ptr->prev->next = ptr->next;
                    ptr->next->prev = ptr->prev;
                    ptr->next = NULL;
                    ptr->prev = NULL;
                    delete ptr;
                    flag++;
                    break;
                }
                ptr = ptr->next;
            }
            if (flag == 0)
            {
                ptr->prev->next = NULL;
                ptr->prev = NULL;
                delete ptr;
            }
        }


    }

    // Deletes the node containing the specified value
    void insert_begin(int value) {
        node*  temp;
        temp = new node;
        temp->data = value;
        temp->next = head;
        temp->prev = NULL;
        if (head != NULL)
            head->prev = temp;
        head = temp;
    }
    // Inserts a new node at the start of the list
    void insert_end(int value) {
        node*  temp;
        temp = new node;
        temp->data = value;
        temp->next = NULL;
        if (head == NULL)
        {
            head = temp;
            temp->prev = NULL;
        }

        else
        {
```

```cpp
            node* p;
            p = head;
            while (p->next != NULL)
            {
                p = p->next;
            }
            p->next = temp;
            temp->prev = p;
        }
    }

    // Inserts a new node at the end of the list
    void traverse() {
        node* p;
        p = head;
        while (p != NULL) {
            cout << p->data << "->";
            p = p->next;
        }
    }
    // Displays the values stored in the list
    void traverse2() {
        node* temp;
        temp = head;
        while (temp->next != NULL)
        {
            temp = temp->next;
        }
        while (temp != NULL)
        {
            cout << temp->data << "->";
            temp = temp->prev;
        }

    }
    // Displays the values stored in the list in reverse order
};
int main()
{
    DList l1;
    cout << "Link list" << endl << endl;
    l1.insert_begin(5);
    l1.insert_begin(10);
    l1.insert_begin(15);
    l1.insert_begin(20);
    l1.insert_begin(25);
    l1.traverse();
    cout << endl << endl;
    cout << "Link list after in between insertion" << endl << endl;
    l1.insertafter(20, 25);
    l1.traverse();
    cout << endl << endl;
    cout << "Link list after insertion at end" << endl << endl;
    l1.insert_end(50);
    l1.traverse();
    cout << endl << endl;
    cout << "Link list after deleting a node" << endl << endl;
    l1.deleteNode(20);
    l1.traverse();
    cout << endl << endl;
    cout << "LInked list in reverse order " << endl << endl;
    l1.traverse2();
    system("pause");
    return 0;
}
```

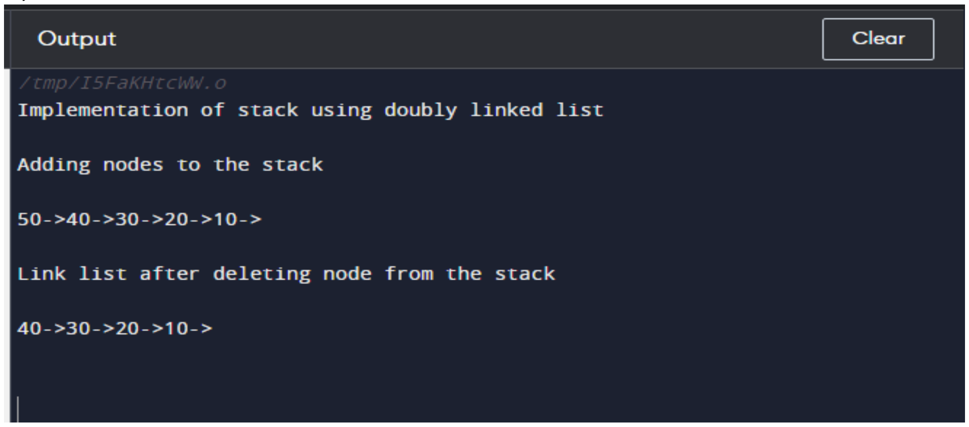


```
/tmp/I5FaKHtcWW.o
Link list
25->20->15->10->5->
Link list after in between insertion
25->20->25->15->10->5->
Link list after insertion at end

25->20->25->15->10->5->50->

Link list after deleting a node

25->25->15->10->5->50->

LInked list in reverse order

50->5->10->15->25->25->
```

<u>Code Exercise 2:</u>

A stack can be implemented using a doubly linked list. The first node can serve as the ‘top’ of Stack and ‘push’ and ‘pop’ operations can be implemented by adding and removing nodes at the head of the linked list. Implement the Stack class using a linked list (Doubly) and provide all the standard member functions.

```
#include <iostream>
using namespace std;
class DList
{
private:
    struct Node {

        struct Node* prev;
        int data;
        struct Node* next;

    }*head;
public:
    DList() {
        head = NULL;
    }
    ~DList()  {}
    bool isempty() {
        if (head == NULL)
            return true;
        else
            return false;
    }

    void push(int data) {

        struct Node* newnode;
        newnode = new Node();

        if (!newnode) {
            cout << "Stack Overflow" << endl;
        }

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

        head = newnode;
    }

    void pop() {

        if (isempty()) {
            cout << "Stack Underflow";
            exit(1);
        }

        Node* tmp;
        tmp = new Node();

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

        delete head;
        head = tmp;

    }

    void display() {

        Node* ptr;
        ptr = head;

        cout << ptr->data << "->";
        while (ptr->prev != NULL) {
            cout << ptr->prev->data << "->";
            ptr = ptr->prev;
        }
        cout << endl;
    }
};

int main ()
{
    DList l1;
    cout << "Implementation of stack using doubly linked list" << endl << endl;
    cout << "Adding nodes to the stack" << endl << endl;
    l1.push(10);
    l1.push(20);
    l1.push(30);
    l1.push(40);
    l1.push(50);
    l1.display();
    cout << endl << endl;
    cout << "Link list after deleting node from the stack" << endl << endl;
    l1.pop();
    l1.display();
    cout << endl << endl;
    system("pause");
    return 0;
}
```



```
/tmp/I5FaKHtcWW.o
Implementation of stack using doubly linked list

Adding nodes to the stack

50->40->30->20->10->

Link list after deleting node from the stack

40->30->20->10->
```