**FDS**

**Name- Sudhir Singh**

**Department-AI & DS**

**Section-SE A ,**

**Batch – A4**

**Roll No-23152**

**Experiment 7**

**Aim:** To understand and implement singly linked list

**Problem definition:** Department of Computer Engineering has student's club named 'Pinnacle Club'. Students of Second, third and final year of department can be granted membership on request. Similarly, one may cancel the membership of club. First node is reserved for president of club and last node is reserved for secretary of club. Write C++ program to maintain club member‘s information using singly linked list. Store student PRN and Name. Write functions to

**a)** Add and delete the members as well as president or even secretary.

**b)** Compute total number of members of club

**c)** Display members

**d)** Display list in reverse order using recursion

**e)** Two linked lists exist for two divisions. Concatenate two lists.

#include <iostream>

#include <string>

using namespace std;

struct Node {

    string name;

    string prn;

    Node\* next;

    Node(string prn, string name) : prn(prn), name(name), next(nullptr) {}

};

class PinnacleClub {

private:

    Node\* head;

    Node\* tail;

public:

    PinnacleClub() : head(nullptr), tail(nullptr) {}

    void addMember(string prn, string name, bool isPresident = false, bool isSecretary = false) {

        Node\* newNode = new Node(prn, name);

        if (isPresident) {

            newNode->next = head;

            head = newNode;

            if (tail == nullptr) tail = newNode;

        } else if (isSecretary) {

            if (tail != nullptr) tail->next = newNode;

            tail = newNode;

            if (head == nullptr) head = newNode;

        } else {

            if (head == nullptr) {

                head = tail = newNode;

            } else {

                tail->next = newNode;

                tail = newNode;

            }

        }

    }

    void deleteMember(string prn) {

        if (!head) {

            cout << "No members to delete.\n";

            return;

        }

        if (head->prn == prn) {

            Node\* temp = head;

            head = head->next;

            delete temp;

            if (!head) tail = nullptr;

            return;

        }

        Node\* current = head;

        while (current->next && current->next->prn != prn) {

            current = current->next;

        }

        if (current->next) {

            Node\* temp = current->next;

            current->next = temp->next;

            if (temp == tail) tail = current;

            delete temp;

        } else {

            cout << "Member with PRN " << prn << " not found.\n";

        }

    }

    int countMembers() {

        int count = 0;

        Node\* temp = head;

        while (temp) {

            count++;

            temp = temp->next;

        }

        return count;

    }

    void displayMembers() {

        if (!head) {

            cout << "No members in the club.\n";

            return;

        }

        Node\* temp = head;

        cout << "Club Members:\n";

        while (temp) {

            cout << "PRN: " << temp->prn << ", Name: " << temp->name << endl;

            temp = temp->next;

        }

    }

    void displayReverse(Node\* node) {

        if (!node) return;

        displayReverse(node->next);

        cout << "PRN: " << node->prn << ", Name: " << node->name << endl;

    }

    void displayReverse() {

        cout << "Members in Reverse Order:\n";

        displayReverse(head);

    }

    void concatenate(PinnacleClub& other) {

        if (!head) {

            head = other.head;

            tail = other.tail;

        } else if (other.head) {

            tail->next = other.head;

            tail = other.tail;

        }

        other.head = other.tail = nullptr;

    }

    ~PinnacleClub() {

        while (head) {

            Node\* temp = head;

            head = head->next;

            delete temp;

        }

    }

};

int main() {

    PinnacleClub divA, divB;

    divA.addMember("101", "Alice", true);

    divA.addMember("102", "Bob");

    divA.addMember("103", "Charlie", false, true);

    divB.addMember("201", "David", true);

    divB.addMember("202", "Eve");

    divB.addMember("203", "Frank", false, true);

    cout << "Division A:\n";

    divA.displayMembers();

    cout << "\nDivision B:\n";

    divB.displayMembers();

    divA.concatenate(divB);

    cout << "\nAfter Concatenation (Division A + Division B):\n";

    divA.displayMembers();

    cout << "\nDisplaying in Reverse Order:\n";

    divA.displayReverse();

    cout << "\nTotal Members: " << divA.countMembers() << endl;

    divA.deleteMember("102");

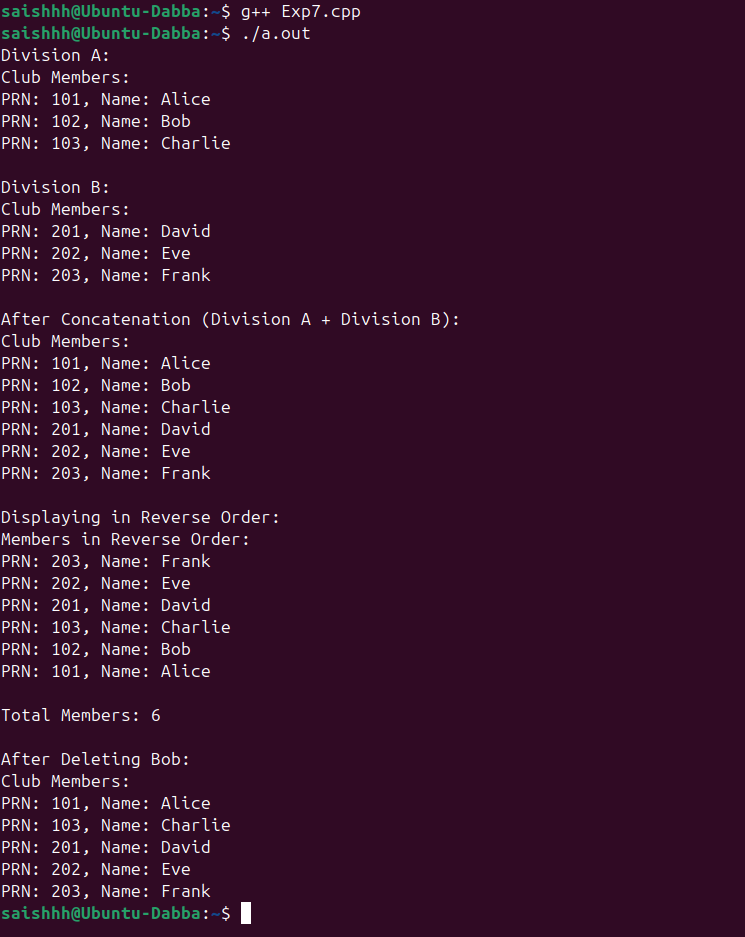
    cout << "\nAfter Deleting Bob:\n";

    divA.displayMembers();

    return 0;

}

**OUTPUT:-**



**FDS**

**Name- Sudhir Singh**

**Department-AI & DS**

**Section-SE A ,**

**Batch – A4**

**Roll No-23152**

**Experiment 8**

**Aim:** To understand and implement set operation using linked list.

**Problem definition:** Second year Computer Engineering class, set A of students like Vanilla Ice-cream and set B of students like butterscotch ice-cream. Write C/C++ program to store two sets using linked list. compute and display

1. Set of students who like either vanilla or butterscotch or both
2. Set of students who like both vanilla and butterscotch
3. Set of students who like only vanilla not butterscotch
4. Set of students who like only butterscotch not vanilla
5. Number of students who like neither vanilla nor butterscotch

#include <iostream>

#include <string>

#include <unordered\_set>

using namespace std;

struct Node {

    string name;

    Node\* next;

    Node(string name) : name(name), next(nullptr) {}

};

class LinkedList {

private:

    Node\* head;

public:

    LinkedList() : head(nullptr) {}

    void addStudent(const string& name) {

        Node\* newNode = new Node(name);

        newNode->next = head;

        head = newNode;

    }

    void display() const {

        Node\* temp = head;

        while (temp) {

            cout << temp->name << " ";

            temp = temp->next;

        }

        cout << endl;

    }

    unordered\_set<string> toSet() const {

        unordered\_set<string> studentSet;

        Node\* temp = head;

        while (temp) {

            studentSet.insert(temp->name);

            temp = temp->next;

        }

        return studentSet;

    }

    ~LinkedList() {

        while (head) {

            Node\* temp = head;

            head = head->next;

            delete temp;

        }

    }

};

void displaySet(const unordered\_set<string>& s) {

    if (s.empty()) {

        cout << "No students found.\n";

        return;

    }

    for (const auto& name : s) {

        cout << name << " ";

    }

    cout << endl;

}

int main() {

    LinkedList vanilla, butterscotch;

    int n, m;

    string name;

    cout << "Enter number of students who like Vanilla: ";

    cin >> n;

    cout << "Enter names of students who like Vanilla:\n";

    for (int i = 0; i < n; ++i) {

        cin >> name;

        vanilla.addStudent(name);

    }

    cout << "Enter number of students who like Butterscotch: ";

    cin >> m;

    cout << "Enter names of students who like Butterscotch:\n";

    for (int i = 0; i < m; ++i) {

        cin >> name;

        butterscotch.addStudent(name);

    }

    unordered\_set<string> setVanilla = vanilla.toSet();

    unordered\_set<string> setButterscotch = butterscotch.toSet();

    unordered\_set<string> unionSet = setVanilla;

    unionSet.insert(setButterscotch.begin(), setButterscotch.end());

    unordered\_set<string> intersectionSet;

    for (const auto& s : setVanilla) {

        if (setButterscotch.count(s)) {

            intersectionSet.insert(s);

        }

    }

    unordered\_set<string> onlyVanilla;

    for (const auto& s : setVanilla) {

        if (!setButterscotch.count(s)) {

            onlyVanilla.insert(s);

        }

    }

    unordered\_set<string> onlyButterscotch;

    for (const auto& s : setButterscotch) {

        if (!setVanilla.count(s)) {

            onlyButterscotch.insert(s);

        }

    }

    int totalStudents;

    cout << "Enter total number of students in class: ";

    cin >> totalStudents;

    int neitherCount = totalStudents - unionSet.size();

    cout << "\nStudents who like either Vanilla or Butterscotch or both:\n";

    displaySet(unionSet);

    cout << "\nStudents who like both Vanilla and Butterscotch:\n";

    displaySet(intersectionSet);

    cout << "\nStudents who like only Vanilla:\n";

    displaySet(onlyVanilla);

    cout << "\nStudents who like only Butterscotch:\n";

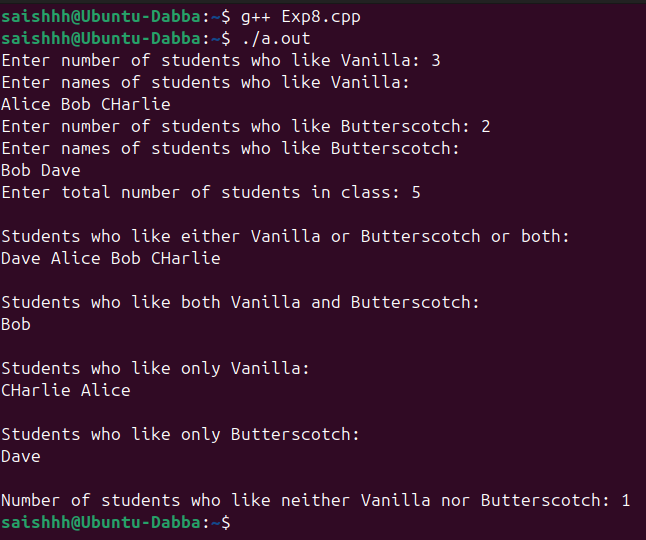
    displaySet(onlyButterscotch);

    cout << "\nNumber of students who like neither Vanilla nor Butterscotch: " << neitherCount << endl;

    return 0;

}

**OUTPUT:-**



**FDS**

**Name- Sudhir Singh**

**Department-AI & DS**

**Section-SE A ,**

**Batch – A4**

**Roll No-23152**

**Experiment 9**

**Aim:** To illustrate the concept of stack and string.

**Problem Statement**: A palindrome is a string of character that‘s the same forward and backward. Typically, punctuation, capitalization, and spaces are ignored. For example, ǁPoor Dan is in a droopǁ is a palindrome, as can be seen by examining the characters ―poor danisina droopǁ and observing that they are the same forward and backward. One way to check for a palindrome is to reverse the characters in the string and then compare with them the original-

in a palindrome, the sequence will be identical. Write C++ program with functions-

1. To check whether given string is palindrome or not that uses a stack to determine whether a string is a palindrome.

2. To remove spaces and punctuation in string, convert all the Characters to lowercase, and then call above Palindrome checking function to check for a palindrome

3. To print string in reverse order using stack

#include <iostream>

#include <stack>

#include <cctype>

#include <string>

using namespace std;

bool isPalindrome(const string& str) {

    stack<char> s;

    for (char ch : str) {

        s.push(ch);

    }

    for (char ch : str) {

        if (ch != s.top()) {

            return false;

        }

        s.pop();

    }

    return true;

}

bool checkPalindromeWithPreprocessing(const string& str) {

    string cleanedStr = "";

    for (char ch : str) {

        if (isalnum(ch)) {

            cleanedStr += tolower(ch);

        }

    }

    return isPalindrome(cleanedStr);

}

void printReverseUsingStack(const string& str) {

    stack<char> s;

    for (char ch : str) {

        s.push(ch);

    }

    cout << "Reversed String: ";

    while (!s.empty()) {

        cout << s.top();

        s.pop();

    }

    cout << endl;

}

int main() {

    string input;

    cout << "Enter a string: ";

    getline(cin, input);

    cout << "\nChecking if the string is a palindrome...\n";

    if (checkPalindromeWithPreprocessing(input)) {

        cout << "The string is a palindrome.\n";

    } else {

        cout << "The string is not a palindrome.\n";

    }

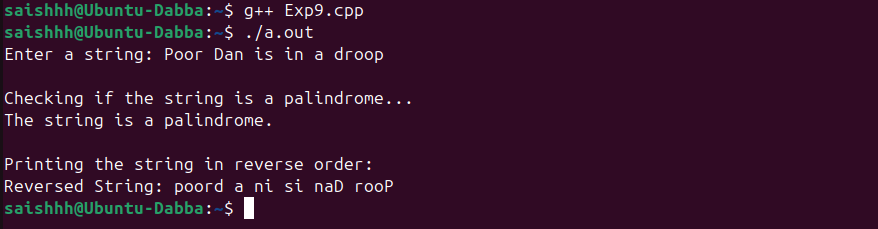
    cout << "\nPrinting the string in reverse order:\n";

    printReverseUsingStack(input);

    return 0;

}

**OUTPUT:-**



**FDS**

**Name- Sudhir Singh**

**Department-AI & DS**

**Section-SE A ,**

**Batch – A4**

**Roll No-23152**

**Experiment 10**

**Aim:** To illustrate the various concept of stack.

**Problem Statement:** In any language program mostly syntax error occurs due to unbalancing delimiter such as (),{},[]. Write C++ program using stack to check whether given expression is well parenthesized or not.

#include <iostream>

#include <stack>

#include <string>

using namespace std;

bool isMatchingPair(char opening, char closing) {

    return (opening == '(' && closing == ')') ||

           (opening == '{' && closing == '}') ||

           (opening == '[' && closing == ']');

}

bool isWellParenthesized(const string& expression) {

    stack<char> s;

    for (char ch : expression) {

        if (ch == '(' || ch == '{' || ch == '[') {

            s.push(ch);

        } else if (ch == ')' || ch == '}' || ch == ']') {

            if (s.empty() || !isMatchingPair(s.top(), ch)) {

                return false;

            }

            s.pop();

        }

    }

    return s.empty();

}

int main() {

    string expression;

    cout << "Enter an expression: ";

    getline(cin, expression);

    if (isWellParenthesized(expression)) {

        cout << "The expression is well parenthesized.\n";

    } else {

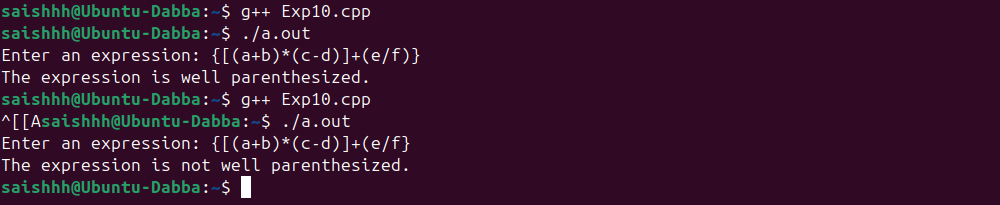
        cout << "The expression is not well parenthesized.\n";

    }

    return 0;

}

**OUTPUT:-**



**FDS**

**Name- Sudhir Singh**

**Department-AI & DS**

**Section-SE A ,**

**Batch – A4**

**Roll No-23152**

**Experiment 11**

**Aim:** To illustrate the concept of queue.

**Problem Statement:** Queues are frequently used in computer programming, and a typical example is the creation of a job queue by an operating system. If the operating system does not use priorities, then the jobs are processed in the order they enter the system. Write C++ program for simulating job queue. Write functions to add job and delete job from queue.

#include <iostream>

#include <queue>

#include <string>

using namespace std;

struct Job {

    int id;

    string description;

    Job(int id, const string& desc) : id(id), description(desc) {}

};

class JobQueue {

private:

    queue<Job> q;

public:

    void addJob(int id, const string& description) {

        Job newJob(id, description);

        q.push(newJob);

        cout << "Job added: [ID: " << id << ", Description: " << description << "]\n";

    }

    void deleteJob() {

        if (q.empty()) {

            cout << "No jobs to delete. The queue is empty.\n";

        } else {

            Job job = q.front();

            cout << "Job deleted: [ID: " << job.id << ", Description: " << job.description << "]\n";

            q.pop();

        }

    }

    void displayJobs() const {

        if (q.empty()) {

            cout << "The queue is empty.\n";

            return;

        }

        queue<Job> temp = q;

        cout << "Jobs in the queue:\n";

        while (!temp.empty()) {

            Job job = temp.front();

            cout << "[ID: " << job.id << ", Description: " << job.description << "]\n";

            temp.pop();

        }

    }

};

int main() {

    JobQueue jobQueue;

    int choice, id;

    string description;

    do {

        cout << "\n1. Add Job\n2. Delete Job\n3. Display Jobs\n4. Exit\n";

        cout << "Enter your choice: ";

        cin >> choice;

        switch (choice) {

            case 1:

                cout << "Enter Job ID: ";

                cin >> id;

                cin.ignore();  // Ignore trailing newline

                cout << "Enter Job Description: ";

                getline(cin, description);

                jobQueue.addJob(id, description);

                break;

            case 2:

                jobQueue.deleteJob();

                break;

            case 3:

                jobQueue.displayJobs();

                break;

            case 4:

                cout << "Exiting...\n";

                break;

            default:

                cout << "Invalid choice. Please try again.\n";

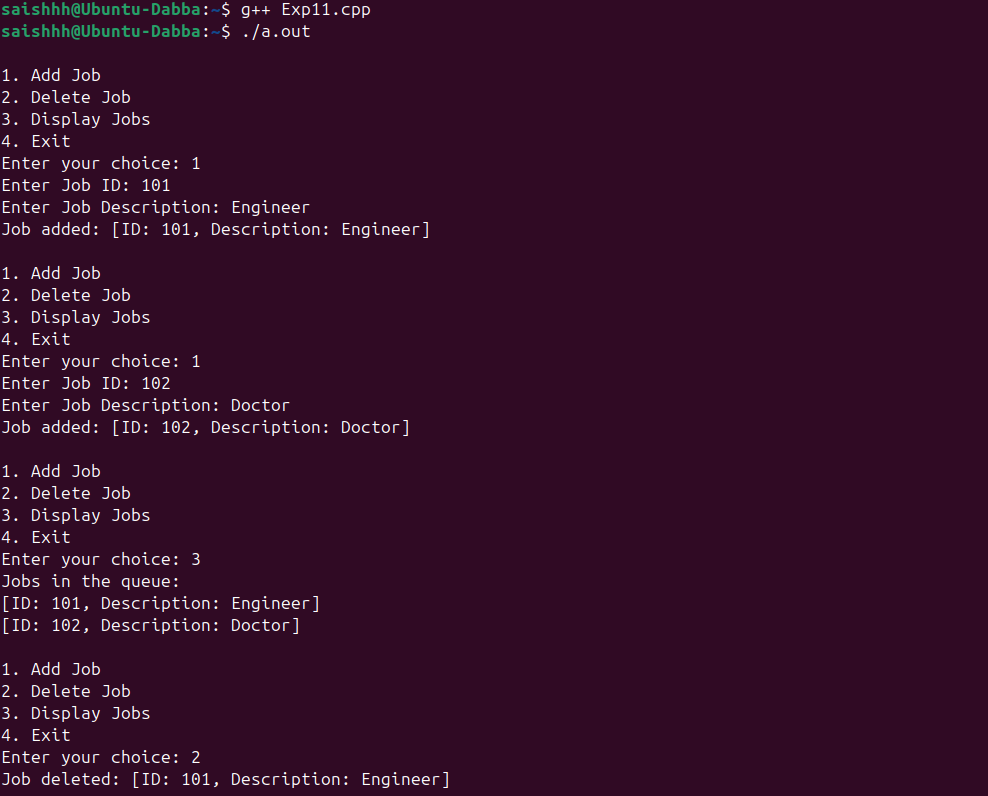
        }

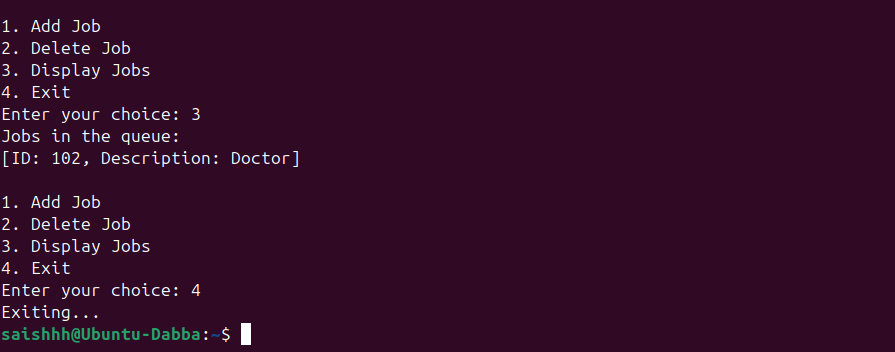
    } while (choice != 4);

    return 0;

}

**OUTPUT:-**





**FDS**

**Name- Sudhir Singh**

**Department-AI & DS**

**Section-SE A ,**

**Batch – A4**

**Roll No-23152**

**Experiment 12**

**Aim:** To illustrate the concept of double-ended queue (deque).

**Problem Statement:** A double-ended queue (deque) is a linear list in which additions and deletions may be made at either end. Obtain a data representation mapping a deque into a one- dimensional array. Write C++ program to simulate deque with functions to add and delete elements from either end of the deque.

#include <iostream>

using namespace std;

class Deque {

private:

    int \*arr;

    int front, rear, size;

public:

    Deque(int n) {

        size = n;

        arr = new int[size];

        front = -1;

        rear = -1;

    }

    ~Deque() {

        delete[] arr;

    }

    bool isFull() {

        return (front == 0 && rear == size - 1) || (rear + 1 == front);

    }

    bool isEmpty() {

        return front == -1;

    }

    void insertFront(int data) {

        if (isFull()) {

            cout << "Deque is full. Cannot insert at front.\n";

            return;

        }

        if (isEmpty()) {

            front = rear = 0;

        } else if (front == 0) {

            front = size - 1;

        } else {

            front--;

        }

        arr[front] = data;

        cout << "Inserted " << data << " at front.\n";

    }

    void insertRear(int data) {

        if (isFull()) {

            cout << "Deque is full. Cannot insert at rear.\n";

            return;

        }

        if (isEmpty()) {

            front = rear = 0;

        } else if (rear == size - 1) {

            rear = 0;

        } else {

            rear++;

        }

        arr[rear] = data;

        cout << "Inserted " << data << " at rear.\n";

    }

    void deleteFront() {

        if (isEmpty()) {

            cout << "Deque is empty. Cannot delete from front.\n";

            return;

        }

        cout << "Deleted " << arr[front] << " from front.\n";

        if (front == rear) {

            front = rear = -1;

        } else if (front == size - 1) {

            front = 0;

        } else {

            front++;

        }

    }

    void deleteRear() {

        if (isEmpty()) {

            cout << "Deque is empty. Cannot delete from rear.\n";

            return;

        }

        cout << "Deleted " << arr[rear] << " from rear.\n";

        if (front == rear) {

            front = rear = -1;

        } else if (rear == 0) {

            rear = size - 1;

        } else {

            rear--;

        }

    }

    void display() {

        if (isEmpty()) {

            cout << "Deque is empty.\n";

            return;

        }

        cout << "Deque elements: ";

        int i = front;

        while (true) {

            cout << arr[i] << " ";

            if (i == rear) break;

            i = (i + 1) % size;

        }

        cout << endl;

    }

};

int main() {

    int size, choice, value;

    cout << "Enter the size of the deque: ";

    cin >> size;

    Deque deque(size);

    do {

        cout << "\n1. Insert at Front\n2. Insert at Rear\n3. Delete from Front\n4. Delete from Rear\n5. Display\n6. Exit\n";

        cout << "Enter your choice: ";

        cin >> choice;

        switch (choice) {

            case 1:

                cout << "Enter value to insert at front: ";

                cin >> value;

                deque.insertFront(value);

                break;

            case 2:

                cout << "Enter value to insert at rear: ";

                cin >> value;

                deque.insertRear(value);

                break;

            case 3:

                deque.deleteFront();

                break;

            case 4:

                deque.deleteRear();

                break;

            case 5:

                deque.display();

                break;

            case 6:

                cout << "Exiting...\n";

                break;

            default:

                cout << "Invalid choice. Please try again.\n";

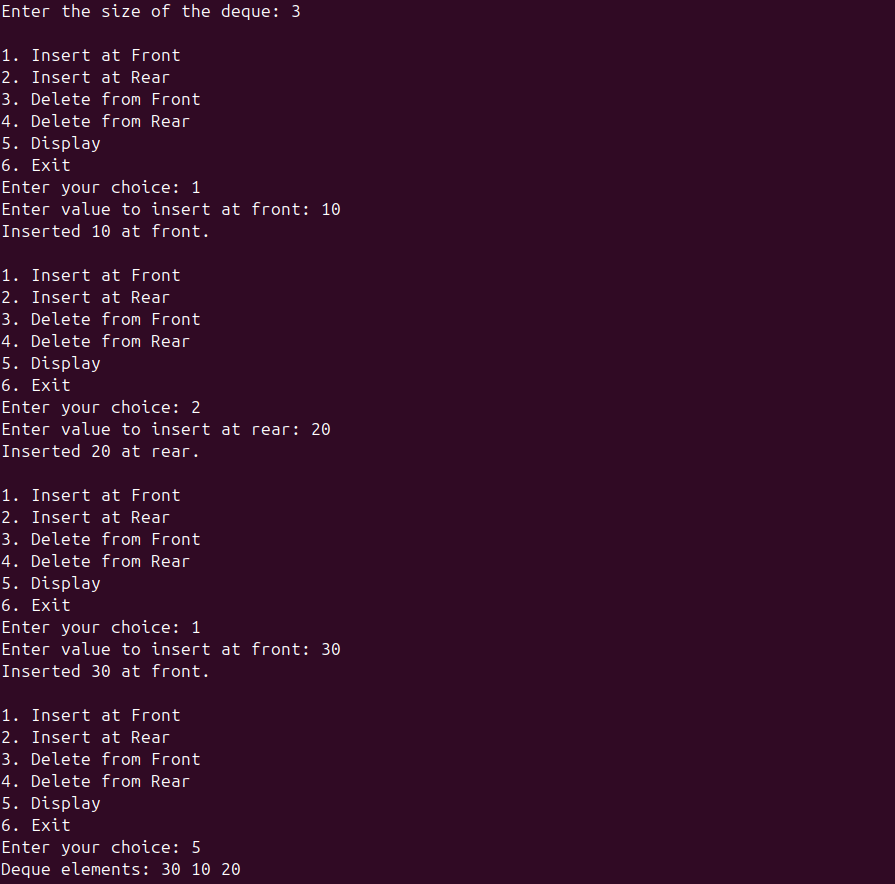
        }

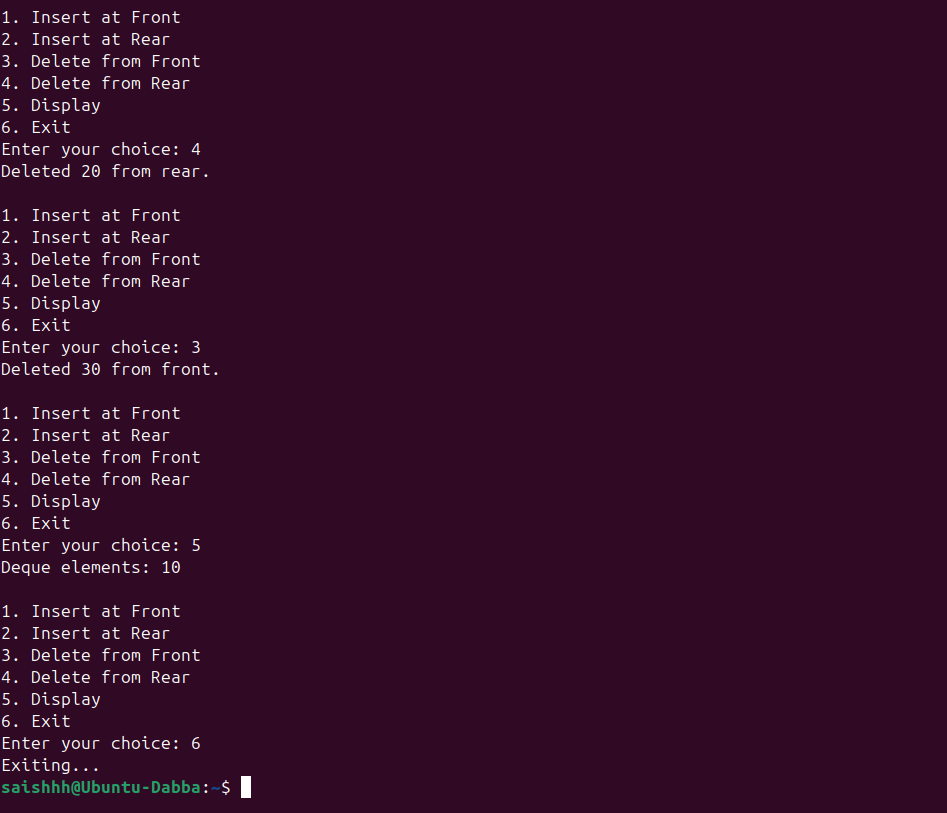
    } while (choice != 6);

    return 0;

}

**OUTPUT:-**





**FDS**

**Name- Sudhir Singh**

**Department-AI & DS**

**Section-SE A ,**

**Batch – A4**

**Roll No-23152**

**Experiment 13**

**Aim:** To illustrate the concept of circular queue.

**Problem Statement:** Pizza parlor accepting maximum M orders. Orders are served in first come first served basis. Order once placed cannot be cancelled. Write C++ program to simulate the system using circular queue using array.

#include <iostream>

using namespace std;

class PizzaOrderQueue {

private:

    int \*orders;

    int front, rear, maxSize, count;

public:

    // Constructor to initialize the queue

    PizzaOrderQueue(int size) {

        maxSize = size;

        orders = new int[maxSize];

        front = -1;

        rear = -1;

        count = 0;

    }

    // Destructor to release allocated memory

    ~PizzaOrderQueue() {

        delete[] orders;

    }

    bool isFull() {

        return count == maxSize;

    }

    bool isEmpty() {

        return count == 0;

    }

    void placeOrder(int orderID) {

        if (isFull()) {

            cout << "Order queue is full. Cannot place new order.\n";

            return;

        }

        if (isEmpty()) {

            front = rear = 0;

        } else {

            rear = (rear + 1) % maxSize;

        }

        orders[rear] = orderID;

        count++;

        cout << "Order " << orderID << " placed successfully.\n";

    }

    void serveOrder() {

        if (isEmpty()) {

            cout << "No orders to serve. The queue is empty.\n";

            return;

        }

        cout << "Order " << orders[front] << " is served.\n";

        if (front == rear) {

            front = rear = -1;  // Reset the queue if it becomes empty

        } else {

            front = (front + 1) % maxSize;

        }

        count--;

    }

    void displayOrders() {

        if (isEmpty()) {

            cout << "No orders in the queue.\n";

            return;

        }

        cout << "Current Orders: ";

        int i = front;

        do {

            cout << orders[i] << " ";

            i = (i + 1) % maxSize;

        } while (i != (rear + 1) % maxSize);

        cout << endl;

    }

};

int main() {

    int size, choice, orderID;

    cout << "Enter the maximum number of orders: ";

    cin >> size;

    PizzaOrderQueue queue(size);

    do {

        cout << "\n1. Place Order\n2. Serve Order\n3. Display Orders\n4. Exit\n";

        cout << "Enter your choice: ";

        cin >> choice;

        switch (choice) {

            case 1:

                cout << "Enter Order ID: ";

                cin >> orderID;

                queue.placeOrder(orderID);

                break;

            case 2:

                queue.serveOrder();

                break;

            case 3:

                queue.displayOrders();

                break;

            case 4:

                cout << "Exiting...\n";

                break;

            default:

                cout << "Invalid choice. Please try again.\n";

        }

    } while (choice != 4);

    return 0;

}

**OUTPUT:-**

