**Practical no 1:-**

In second year computer engineering class, group A student‟s play cricket, group B studentsplay badminton and group C students play football. Write a Python program using functions to compute following: -

a) List of students who play both cricket and badminton

b) List of students who play either cricket or badminton but not both

c) Number of students who play neither cricket nor badminton

d) Number of students who play cricket and football but not badminton.

e) (Note- While realizing the group, duplicate entries should be avoided, Do not use SET built- in functions)

def remove\_duplicates(lst):

unique\_lst = []

for student in lst:

if student not in unique\_lst:

unique\_lst.append(student)

return unique\_lst

def both\_cricket\_and\_badminton(groupA, groupB):

# Find intersection of groupA and groupB

result = []

for student in groupA:

if student in groupB and student not in result:

result.append(student)

return result

def cricket\_or\_badminton\_not\_both(groupA, groupB):

cricket\_only = [student for student in groupA if student not in groupB]

badminton\_only = [student for student in groupB if student not in groupA]

result = remove\_duplicates(cricket\_only + badminton\_only)

return result

def neither\_cricket\_nor\_badminton(groupA, groupB, groupC, all\_students):

cricket\_or\_badminton = remove\_duplicates(groupA + groupB)

result = [student for student in all\_students if student not in cricket\_or\_badminton]

return len(result)

def cricket\_and\_football\_not\_badminton(groupA, groupB, groupC):

# Find intersection of groupA (cricket) and groupC (football), but not in groupB (badminton)

result = []

for student in groupA:

if student in groupC and student not in groupB and student not in result:

result.append(student)

return len(result)

# Sample lists of students (can be updated with actual data)

groupA = ["Alice", "Bob", "Charlie", "David", "Eve"]

groupB = ["Bob", "Eve", "Frank", "Grace"]

groupC = ["Charlie", "David", "Eve", "Hannah"]

all\_students = ["Alice", "Bob", "Charlie", "David", "Eve", "Frank", "Grace", "Hannah", "Ivy", "Jack"]

# Removing duplicates from groups to ensure no duplicates within a group

groupA = remove\_duplicates(groupA)

groupB = remove\_duplicates(groupB)

groupC = remove\_duplicates(groupC)

# a) List of students who play both cricket and badminton

print("Students who play both cricket and badminton:", both\_cricket\_and\_badminton(groupA, groupB))

# b) List of students who play either cricket or badminton but not both

print("Students who play either cricket or badminton but not both:", cricket\_or\_badminton\_not\_both(groupA, groupB))

# c) Number of students who play neither cricket nor badminton

print("Number of students who play neither cricket nor badminton:", neither\_cricket\_nor\_badminton(groupA, groupB, groupC, all\_students))

# d) Number of students who play cricket and football but not badminton

print("Number of students who play cricket and football but not badminton:", cricket\_and\_football\_not\_badminton(groupA, groupB, groupC))

**Practical no 2:-**

Write a Python program to store marks scored in subject “Fundamental of Data Structure” byN students in the class. Write functions to compute following: a) The average score of class b) Highest score and lowest score of class c) Count of students who were absent for the test d) Display mark with highest frequency

from collections import Counter

# Function to compute the average score of the class

def average\_score(marks):

return sum(marks) / len(marks)

# Function to compute the highest and lowest score in the class

def highest\_lowest\_score(marks):

highest = max(marks)

lowest = min(marks)

return highest, lowest

# Function to count the number of students who were absent

def count\_absent(marks, total\_students):

# An absent student can be represented by a -1 (or any other placeholder value)

absent\_count = marks.count(-1)

return absent\_count

# Function to display the mark with the highest frequency

def most\_frequent\_marks(marks):

from collections import Counter

mark\_counts = Counter(marks)

most\_frequent = mark\_counts.most\_common(1) # Get the most common mark

return most\_frequent[0][0], most\_frequent[0][1] # Return the mark and its frequency

# Main part of the program

def main():

# Input: Number of students in the class

total\_students = int(input("Enter the number of students: "))

# Input: Marks of students, -1 for absent

marks = []

for i in range(total\_students):

score = int(input(f"Enter marks for student {i+1} (Enter -1 if absent): "))

marks.append(score)

# Calculate average score of the class

avg\_score = average\_score(marks)

print(f"The average score of the class is: {avg\_score:.2f}")

# Calculate highest and lowest score

highest, lowest = highest\_lowest\_score(marks)

print(f"The highest score in the class is: {highest}")

print(f"The lowest score in the class is: {lowest}")

# Count the number of students who were absent

absent\_count = count\_absent(marks, total\_students)

print(f"Number of students who were absent: {absent\_count}")

# Display the mark with highest frequency

frequent\_mark, frequency = most\_frequent\_marks(marks)

print(f"The most frequent mark is {frequent\_mark} with a frequency of {frequency}.")

# Run the main function

if \_\_name\_\_ == "\_\_main\_\_":

main()

OUTPUT:-

Enter the number of students: 5

Enter marks for student 1 (or 'A' for absent):

Invalid input. Please enter an integer score or 'A' for absent.

Enter marks for student 1 (or 'A' for absent): 87

Enter marks for student 2 (or 'A' for absent): 89

Enter marks for student 3 (or 'A' for absent): 90

Enter marks for student 4 (or 'A' for absent): 98

Enter marks for student 5 (or 'A' for absent): 99

Average score of the class: 92.60

Highest score: 99

Lowest score: 87

Count of absent students: 0

Mark with highest frequency: 87 (occurrences: 1)

=== Code Execution Successful ===

**Practical no 3:-**

3 Write a Python program for department library which has N books, write functions for following:

a) Delete the duplicate entries

b) Display books in ascending order based on cost of books

c) Count number of books with cost more than 500.

d)Copy books in a new list which has cost less than 500

# Function to delete duplicate entries of books

def delete\_duplicates(books):

# We assume books are represented as tuples of (title, cost)

unique\_books = []

seen\_titles = set()

for book in books:

title = book[0]

if title not in seen\_titles:

unique\_books.append(book)

seen\_titles.add(title)

return unique\_books

# Function to display books in ascending order based on their cost

def display\_books\_by\_cost(books):

sorted\_books = sorted(books, key=lambda book: book[1]) # Sort by cost (index 1)

return sorted\_books

def count\_books\_above\_500(books):

count = sum(1 for book in books if book[1] > 500)

return count

# Function to copy books with cost less than 500 into a new list

def books\_less\_than\_500(books):

less\_than\_500\_books = [book for book in books if book[1] < 500]

return less\_than\_500\_books

# Main part of the program

def main():

# Input: List of books as tuples (title, cost)

n = int(input("Enter the number of books: "))

books = []

for i in range(n):

title = input(f"Enter title of book {i+1}: ")

cost = float(input(f"Enter cost of book {i+1}: "))

books.append((title, cost))

print("\nBooks after removing duplicates:")

books\_no\_duplicates = delete\_duplicates(books)

for book in books\_no\_duplicates:

print(f"{book[0]} - {book[1]}")

print("\nBooks sorted by cost in ascending order:")

sorted\_books = display\_books\_by\_cost(books\_no\_duplicates)

for book in sorted\_books:

print(f"{book[0]} - {book[1]}")

print("\nCount of books with cost greater than 500:", count\_books\_above\_500(books\_no\_duplicates))

print("\nBooks with cost less than 500:")

books\_under\_500 = books\_less\_than\_500(books\_no\_duplicates)

for book in books\_under\_500:

print(f"{book[0]} - {book[1]}")

# Run the main function

if \_\_name\_\_ == "\_\_main\_\_":

main()

OUTPUT:-

Original library:

Title: Database Management, Cost: 400

Title: Python Programming, Cost: 450

Title: Python Programming, Cost: 450

Title: Algorithms, Cost: 550

Title: Data Structures, Cost: 600

After deleting duplicates:

Title: Database Management, Cost: 400

Title: Python Programming, Cost: 450

Title: Algorithms, Cost: 550

Title: Data Structures, Cost: 600

Number of books with cost more than 500: 2

Books with cost less than 500:

Title: Python Programming, Cost: 450

Title: Database Management, Cost: 400

=== Code Execution Successful ===

**Practical no 4:-** Write a Python program to store first year percentage of students in array. Write function for sorting array of floating point numbers in ascending order using a) Selection Sort b) Bubble sort and display top five scores.

# Selection sort function

def selection\_sort(arr):

for i in range(len(arr)):

min\_index = i

for j in range(i + 1, len(arr)):

if arr[j] < arr[min\_index]:

min\_index = j

arr[i], arr[min\_index] = arr[min\_index], arr[i]

return arr

def bubble\_sort(arr):

n = len(arr)

for i in range(n):

for j in range(0, n - i - 1):

if arr[j] > arr[j + 1]:

arr[j], arr[j + 1] = arr[j + 1], arr[j]

return arr

# Display top 5 scores

def display\_top\_five(arr):

print("Top 5 scores:", arr[-5:][::-1])

# First-year student percentages

percentages = [78.5, 82.3, 67.4, 90.6, 88.2, 76.5, 85.1, 92.3, 70.4, 80.9]

# Sorting and displaying top 5 scores using selection sort

sorted\_selection = selection\_sort(percentages.copy())

display\_top\_five(sorted\_selection)

# Sorting and displaying top 5 scores using bubble sort

sorted\_bubble = bubble\_sort(percentages.copy())

display\_top\_five(sorted\_bubble

5 )**practical no**

Write a Python program to store second year percentage of students in array. Write function for sorting array of floating point numbers in ascending order using : a) Insertion sort b) Shell Sort and display top five scores.

# Insertion Sort function to sort the array in ascending order

def insertion\_sort(arr):

for i in range(1, len(arr)): # Start from the second element

key = arr[i] # The current element to be inserted

j = i - 1 # Position to compare with previous elements

while j >= 0 and arr[j] > key: # Shift elements that are greater than key

arr[j + 1] = arr[j] # Shift element one position to the right

j -= 1

arr[j + 1] = key # Insert the key element at the correct position

return arr

def shell\_sort(arr):

n = len(arr)

gap = n // 2 # Initial gap size

while gap > 0:

for i in range(gap, n): # Start from the element at 'gap' position

temp = arr[i] # Store the current element

j = i

while j >= gap and arr[j - gap] > temp: # Compare with the element at a distance of 'gap'

arr[j] = arr[j - gap] # Shift the element to the right

j -= gap

arr[j] = temp # Place the temp element at its correct position

gap //= 2 # Reduce the gap

return arr

# Function to display the top 5 scores from the sorted array

def display\_top\_five(arr):

print("Top 5 scores:", arr[-5:][::-1]) # Get the last 5 elements and reverse to show in descending order

# Main code

def main():

# List of second year students' percentages

percentages = [68.5, 72.3, 91.4, 85.6, 77.2, 88.9, 65.4, 79.8, 92.1, 84.3]

# Sorting using Insertion Sort

print("Sorting using Insertion Sort:")

sorted\_insertion = insertion\_sort(percentages.copy()) # Make a copy to avoid modifying the original list

display\_top\_five(sorted\_insertion) # Display top 5 scores

# Sorting using Shell Sort

print("\nSorting using Shell Sort:")

sorted\_shell = shell\_sort(percentages.copy()) # Make a copy to avoid modifying the original list

display\_top\_five(sorted\_shell) # Display top 5 scores

# Run the program

if \_\_name\_\_ == "\_\_main\_\_":

main()Original Percentages:

85.50 92.30 78.40 88.10 95.00 76.80 89.20 91.50 80.30 97.50

Sorted Percentages using Insertion Sort:

Top five scores:

91.50

92.30

95.00

97.50

88.10

Sorted Percentages using Shell Sort:

Top five scores:

91.50

92.30

95.00

97.50

88.10

6. practical no

Write a Python program to store first year percentage of students in array. Write function for sorting array of floating point numbers in ascending order using quick sort and display top five scores

# Quick Sort function to sort the array in ascending order

def quick\_sort(arr):

if len(arr) <= 1:

return arr # Base case: a list of length 0 or 1 is already sorted

pivot = arr[len(arr) // 2] # Choose the middle element as the pivot

left = [x for x in arr if x < pivot] # Elements smaller than the pivot

middle = [x for x in arr if x == pivot] # Elements equal to the pivot

right = [x for x in arr if x > pivot] # Elements greater than the pivot

return quick\_sort(left) + middle + quick\_sort(right) # Recursively sort the left and right parts

# Function to display the top 5 scores from the sorted array

def display\_top\_five(arr):

print("Top 5 scores:", arr[-5:][::-1]) # Get the last 5 elements and reverse to show in descending order

# Main code

def main():

# List of first year students' percentages

percentages = [78.5, 82.3, 67.4, 90.6, 88.2, 76.5, 85.1, 92.3, 70.4, 80.9]

# Sorting using Quick Sort

sorted\_percentages = quick\_sort(percentages) # Sort the array in ascending order

display\_top\_five(sorted\_percentages) # Display the top 5 scores

# Run the program

if \_\_name\_\_ == "\_\_main\_\_":

main()

7. practical no

Department of Computer Engineering has student's club named 'Pinnacle Club'. Students of second, third and 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:

1. Add and delete the members as well as president or even secretary. 2. Compute total number of members of club 3. Display members 4. Two linked lists exist for two divisions. Concatenate two lists

#include <iostream>

#include <string>

using namespace std;

// Define a structure for a single node in the linked list

struct Node {

string prn; // PRN (student ID)

string name; // Name of the student

Node\* next; // Pointer to the next node

// Constructor to initialize node with PRN and Name

Node(string p, string n) {

prn = p;

name = n;

next = nullptr;

}

};

// Class to represent the Pinnacle Club and its members using a singly linked list

class Club {

private:

Node\* head; // Pointer to the first member (head of the list)

public:

// Constructor initializes the head to nullptr (empty list)

Club() {

head = nullptr;

}

// Function to add a new member at the end of the list

void addMember(string prn, string name) {

Node\* newNode = new Node(prn, name); // Create a new node with PRN and Name

if (head == nullptr) {

head = newNode; // If the list is empty, new member becomes the head

} else {

Node\* temp = head;

while (temp->next != nullptr) { // Traverse to the end of the list

temp = temp->next;

}

temp->next = newNode; // Add the new member at the end

}

}

// Function to delete a member by PRN (including president and secretary)

void deleteMember(string prn) {

if (head == nullptr) {

cout << "The club is empty!" << endl;

return;

}

// If the first node contains the PRN (deleting president)

if (head->prn == prn) {

Node\* temp = head;

head = head->next;

delete temp; // Free memory

cout << "Member with PRN " << prn << " has been removed." << endl;

return;

}

// Traverse to find the node with the given PRN

Node\* temp = head;

while (temp->next != nullptr && temp->next->prn != prn) {

temp = temp->next;

}

if (temp->next == nullptr) {

cout << "Member with PRN " << prn << " not found!" << endl;

return;

}

// Delete the node

Node\* deleteNode = temp->next;

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

delete deleteNode; // Free memory

cout << "Member with PRN " << prn << " has been removed." << endl;

}

// Function to compute the total number of members in the club

int getTotalMembers() {

int count = 0;

Node\* temp = head;

while (temp != nullptr) {

count++;

temp = temp->next;

}

return count;

}

// Function to display all the members of the club

void displayMembers() {

if (head == nullptr) {

cout << "The club has no members!" << endl;

return;

}

Node\* temp = head;

cout << "Club Members: " << endl;

while (temp != nullptr) {

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

temp = temp->next;

}

}

// Function to concatenate another club list to this club

void concatenate(Club& otherClub) {

if (head == nullptr) {

head = otherClub.head; // If this club is empty, take the other club's members

otherClub.head = nullptr; // Set the other club's head to nullptr

} else {

Node\* temp = head;

while (temp->next != nullptr) {

temp = temp->next; // Traverse to the end of this club's list

}

temp->next = otherClub.head; // Link the other club's list to the end of this club

otherClub.head = nullptr; // Set the other club's head to nullptr

}

}

};

// Main function to demonstrate the functionality

int main() {

Club divisionA, divisionB;

// Add members to Division A

divisionA.addMember("A101", "Alice");

divisionA.addMember("A102", "Bob");

divisionA.addMember("A103", "Charlie");

// Add members to Division B

divisionB.addMember("B101", "David");

divisionB.addMember("B102", "Eva");

divisionB.addMember("B103", "Frank");

// Display members of Division A

cout << "Division A Members: " << endl;

divisionA.displayMembers();

cout << endl;

// Display members of Division B

cout << "Division B Members: " << endl;

divisionB.displayMembers();

cout << endl;

// Concatenate Division B into Division A

divisionA.concatenate(divisionB);

// Display all members after concatenation

cout << "All Members after concatenation: " << endl;

divisionA.displayMembers();

cout << endl;

// Total number of members in the club

cout << "Total Members in the Club: " << divisionA.getTotalMembers() << endl;

cout << endl;

// Delete a specific member (e.g., president or secretary)

divisionA.deleteMember("A101"); // Deleting Alice (President)

divisionA.deleteMember("B102"); // Deleting Eva

// Display members after deletion

cout << "Members after deletion: " << endl;

divisionA.displayMembers();

cout << endl;

// Display total members after deletion

cout << "Total Members after deletion: " << divisionA.getTotalMembers() << endl;

return 0;

}

8.practical no

Second year Computer Engineering class, set A of students like Vanilla Ice-cream and set B of students like butterscotch ice-cream. Write C++ program to store two sets using linked list. compute and display : Write C/C++ program to store two sets using linked list, compute and display: a. Set of students who like either vanilla or butterscotch or both b. Set of students who like both vanilla and butterscotch c. Number of students who like neither vanilla nor butterscotch

#include <iostream>

#include <string>

#include <unordered\_set>

using namespace std;

// Node structure to represent each student in the linked list

struct Node {

string name; // Name of the student

Node\* next; // Pointer to the next node in the list

// Constructor to initialize the node with a name

Node(string n) {

name = n;

next = nullptr;

}

};

// Class to represent the set of students

class IceCreamSet {

private:

Node\* head; // Head pointer for the linked list

public:

IceCreamSet() {

head = nullptr;

}

void addStudent(string name) {

Node\* newNode = new Node(name); // Create a new node for the student

if (head == nullptr) {

head = newNode; // If list is empty, the new node becomes the head

} else {

Node\* temp = head;

while (temp->next != nullptr) {

temp = temp->next; // Traverse to the last node

}

temp->next = newNode; // Add the new node at the end

}

}

// Function to get all students in the set as an unordered set (for easier set operations)

unordered\_set<string> getStudentsSet() {

unordered\_set<string> studentsSet;

Node\* temp = head;

while (temp != nullptr) {

studentsSet.insert(temp->name); // Add each student's name to the set

temp = temp->next;

}

return studentsSet;

}

// Function to display all the students in the set

void displayStudents() {

Node\* temp = head;

while (temp != nullptr) {

cout << temp->name << endl; // Display student name

temp = temp->next;

}

}

};

// Function to compute the union of two sets (Vanilla or Butterscotch)

IceCreamSet computeUnion(IceCreamSet& setA, IceCreamSet& setB) {

IceCreamSet unionSet;

unordered\_set<string> studentsA = setA.getStudentsSet();

unordered\_set<string> studentsB = setB.getStudentsSet();

// Add all students from both sets (union operation)

for (const string& student : studentsA) {

unionSet.addStudent(student);

}

for (const string& student : studentsB) {

unionSet.addStudent(student);

}

return unionSet;

}

// Function to compute the intersection of two sets (both Vanilla and Butterscotch)

IceCreamSet computeIntersection(IceCreamSet& setA, IceCreamSet& setB) {

IceCreamSet intersectionSet;

unordered\_set<string> studentsA = setA.getStudentsSet();

unordered\_set<string> studentsB = setB.getStudentsSet();

// Add students who are present in both sets (intersection operation)

for (const string& student : studentsA) {

if (studentsB.find(student) != studentsB.end()) {

intersectionSet.addStudent(student);

}

}

return intersectionSet;

}

// Function to compute the complement of the union (students who like neither Vanilla nor Butterscotch)

int computeComplement(IceCreamSet& setA, IceCreamSet& setB, unordered\_set<string>& allStudents) {

IceCreamSet unionSet = computeUnion(setA, setB);

unordered\_set<string> unionStudents = unionSet.getStudentsSet();

int count = 0;

// Count the number of students who are not in the union

for (const string& student : allStudents) {

if (unionStudents.find(student) == unionStudents.end()) {

count++;

}

}

return count;

}

// Main function to demonstrate the functionality

int main() {

// List of all students in the class (for the complement operation)

unordered\_set<string> allStudents = {"Alice", "Bob", "Charlie", "David", "Eva", "Frank", "Grace", "

IceCreamSet vanillaSet;

IceCreamSet butterscotchSet;

// Adding students to the Vanilla Ice-Cream set

vanillaSet.addStudent("Alice");

vanillaSet.addStudent("Bob");

vanillaSet.addStudent("Charlie");

// Adding students to the Butterscotch Ice-Cream set

butterscotchSet.addStudent("Charlie");

butterscotchSet.addStudent("David");

butterscotchSet.addStudent("Eva");

// Display all students who like Vanilla

cout << "Students who like Vanilla Ice-cream:" << endl;

vanillaSet.displayStudents();

cout << endl;

// Display all students who like Butterscotch

cout << "Students who like Butterscotch Ice-cream:" << endl;

butterscotchSet.displayStudents();

cout << endl;

// Compute and display the union (students who like either Vanilla or Butterscotch or both)

IceCreamSet unionSet = computeUnion(vanillaSet, butterscotchSet);

cout << "Students who like either Vanilla or Butterscotch or both (Union):" << endl;

unionSet.displayStudents();

cout << endl;

// Compute and display the intersection (students who like both Vanilla and Butterscotch)

IceCreamSet intersectionSet = computeIntersection(vanillaSet, butterscotchSet);

cout << "Students who like both Vanilla and Butterscotch (Intersection):" << endl;

intersectionSet.displayStudents();

cout << endl;

// Compute and display the complement (students who like neither Vanilla nor Butterscotch)

int studentsNeither = computeComplement(vanillaSet, butterscotchSet, allStudents);

cout << "Number of students who like neither Vanilla nor Butterscotch: " << studentsNeither << endl;

return 0;

}

10. practical no

A palindrome is a string of character that„s the same forward and backward. Typically, punctuation, capitalization, and spaces are ignored. For example, “Poor Dan is in a droop” is a palindrome, as can be seen by examining the characters “poor danisina droop” 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 functions1. To print original string followed by reversed string using stack 2. To check whether given string is palindrome or not

#include <iostream>

#include <stack>

#include <cctype> // For isalpha, isspace, tolower

using namespace std;

// Function to print the original string followed by the reversed string using a stack

void printOriginalAndReversed(const string& input) {

// Print the original string

cout << "Original string: " << input << endl;

// Using stack to reverse the string

stack<char> s;

// Push all characters of the string onto the stack

for (char ch : input) {

s.push(ch);

}

// Pop characters from the stack to get the reversed string

cout << "Reversed string: ";

while (!s.empty()) {

cout << s.top();

s.pop();

}

cout << endl;

}

// Function to check whether a given string is a palindrome or not

bool isPalindrome(const string& input) {

string filteredInput;

// Remove non-alphanumeric characters and convert to lowercase

for (char ch : input) {

if (isalnum(ch)) {

filteredInput.push\_back(tolower(ch));

}

}

// Check if the filtered string is the same forwards and backwards

int left = 0;

int right = filteredInput.length() - 1;

while (left < right) {

if (filteredInput[left] != filteredInput[right]) {

return false; // Not a palindrome

}

left++;

right--;

}

return true; // It is a palindrome

}

int main() {

string input;

// Input string from user

cout << "Enter a string: ";

getline(cin, input); // Using getline to allow spaces in the input

// Print the original and reversed string using stack

printOriginalAndReversed(input);

// Check if the string is a palindrome

if (isPalindrome(input)) {

cout << "The string is a palindrome." << endl;

} else {

cout << "The string is not a palindrome." << endl;

}

return 0

10.

Implement C++ program for expression conversion as infix to postfix and its evaluation using stack based on given conditions: 1. Operands and operator, both must be single character. 2. Input Postfix expression must be in a desired format. 3. Only '+', ' -', '\*' and '/ ' operators are expected

#include <iostream>

#include <stack>

#include <cctype> // For isalnum function to ignore non-alphanumeric characters

using namespace std;

// Function to print the original string and its reversed version using a stack

void printOriginalAndReversed(string str) {

stack<char> s;

// Push all alphanumeric characters (ignoring spaces and punctuation) to the stack

for (char ch : str) {

if (isalnum(ch)) {

s.push(tolower(ch)); // Convert to lowercase for case-insensitivity

}

}

// Print the original string

cout << "Original string: " << str << endl;

// Print the reversed string by popping characters from the stack

cout << "Reversed string: ";

while (!s.empty()) {

cout << s.top();

s.pop();

}

cout << endl;

}

// Function to check whether the string is a palindrome

bool isPalindrome(string str) {

stack<char> s;

// Push all alphanumeric characters to the stack

for (char ch : str) {

if (isalnum(ch)) {

s.push(tolower(ch)); // Convert to lowercase

}

}

// Check if the string is the same forwards and backwards

int index = 0;

for (char ch : str) {

if (isalnum(ch)) {

if (ch != s.top()) {

return false; // If any character does not match, it's not a palindrome

}

s.pop();

}

}

return true; // If all characters match, it's a palindrome

}

int main() {

string input;

cout << "Enter a string: ";

getline(cin, input); // Read the input string

// Print original and reversed string using stack

printOriginalAndReversed(input);

// Check if the string is a palindrome and display the result

if (isPalindrome(input)) {

cout << "The string is a palindrome!" << endl;

} else {

cout << "The string is not a palindrome!" << endl;

}

return 0;

}

11

#include <iostream>

#include <stack>

#include <cctype> // For isdigit() function

#include <string>

using namespace std;

// Function to return precedence of operators

int precedence(char op) {

if (op == '+' || op == '-') {

return 1; // Lowest precedence

} else if (op == '\*' || op == '/') {

return 2; // Higher precedence

}

return 0;

}

// Function to perform arithmetic operations

int applyOperator(int a, int b, char op) {

switch (op) {

case '+': return a + b;

case '-': return a - b;

case '\*': return a \* b;

case '/': return a / b;

default: return 0;

}

}

// Function to convert infix expression to postfix

string infixToPostfix(string infix) {

stack<char> operators;

string postfix = "";

for (char c : infix) {

if (isalnum(c)) {

postfix += c; // If the character is an operand, add it to the postfix expression

} else if (c == '(') {

operators.push(c); // Push '(' to stack

} else if (c == ')') {

// Pop and append operators until '(' is encountered

while (!operators.empty() && operators.top() != '(') {

postfix += operators.top();

operators.pop();

}

operators.pop(); // Pop '(' from stack

} else if (c == '+' || c == '-' || c == '\*' || c == '/') {

// Pop operators from stack if they have higher or equal precedence

while (!operators.empty() && precedence(operators.top()) >= precedence(c)) {

postfix += operators.top();

operators.pop();

}

operators.push(c); // Push the current operator to stack

}

}

// Pop all remaining operators from the stack

while (!operators.empty()) {

postfix += operators.top();

operators.pop();

}

return postfix;

}

// Function to evaluate a postfix expression

int evaluatePostfix(string postfix) {

stack<int> values;

for (char c : postfix) {

if (isdigit(c)) {

values.push(c - '0'); // Convert char to int and push onto stack

} else if (c == '+' || c == '-' || c == '\*' || c == '/') {

int b = values.top();

values.pop();

int a = values.top();

values.pop();

int result = applyOperator(a, b, c);

values.push(result); // Push result back onto stack

}

}

return values.top(); // The result is the only value left in the stack

}

int main() {

string infix, postfix;

cout << "Enter infix expression: ";

getline(cin, infix);

// Convert infix expression to postfix

postfix = infixToPostfix(infix);

cout << "Postfix expression: " << postfix << endl;

// Evaluate the postfix expression

int result = evaluatePostfix(postfix);

cout << "Evaluated result of postfix expression: " << result << endl;

return 0;

}

12.

#include <iostream>

using namespace std;

class Deque {

private:

int \*arr; // Array to store deque elements

int front, rear, size;

public:

// Constructor to initialize deque

Deque(int n) {

size = n;

arr = new int[size];

front = -1;

rear = -1;

}

// Destructor to clean up dynamically allocated memory

~Deque() {

delete[] arr;

}

// Function to check if deque is full

bool isFull() {

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

}

// Function to check if deque is empty

bool isEmpty() {

return front == -1;

}

// Function to add an element at the front of the deque

void addFront(int value) {

if (isFull()) {

cout << "Deque is full! Cannot add at front." << endl;

return;

}

if (front == -1) { // If the deque is empty

front = rear = 0;

} else if (front == 0) { // Wrap around to the end

front = size - 1;

} else {

front--;

}

arr[front] = value;

cout << "Added " << value << " at the front." << endl;

}

// Function to add an element at the back of the deque

void addBack(int value) {

if (isFull()) {

cout << "Deque is full! Cannot add at back." << endl;

return;

}

if (front == -1) { // If the deque is empty

front = rear = 0;

} else if (rear == size - 1) { // Wrap around to the front

rear = 0;

} else {

rear++;

}

arr[rear] = value;

cout << "Added " << value << " at the back." << endl;

}

// Function to delete an element from the front of the deque

void deleteFront() {

if (isEmpty()) {

cout << "Deque is empty! Cannot delete from front." << endl;

return;

}

cout << "Deleted " << arr[front] << " from the front." << endl;

if (front == rear) { // Only one element left

front = rear = -1;

} else if (front == size - 1) { // Wrap around to the beginning

front = 0;

} else {

front++;

}

}

// Function to delete an element from the back of the deque

void deleteBack() {

if (isEmpty()) {

cout << "Deque is empty! Cannot delete from back." << endl;

return;

}

cout << "Deleted " << arr[rear] << " from the back." << endl;

if (front == rear) { // Only one element left

front = rear = -1;

} else if (rear == 0) { // Wrap around to the end

rear = size - 1;

} else {

rear--;

}

}

// Function to display the elements in the deque

void display() {

if (isEmpty()) {

cout << "Deque is empty!" << endl;

return;

}

cout << "Deque elements: ";

int i = front;

while (i != rear) {

cout << arr[i] << " ";

i = (i + 1) % size; // Wrap around

}

cout << arr[rear] << endl; // Print the last element

}

};

int main() {

Deque dq(5); // Create a deque with capacity of 5 elements

dq.addBack(10);

dq.addBack(20);

dq.addFront(30);

dq.addBack(40);

dq.addFront(50);

dq.display(); // Display elements in the deque

dq.deleteFront(); // Delete from front

dq.deleteBack(); // Delete from back

dq.display(); // Display elements after deletions

return 0;

}

13. #include <iostream>

#include <string>

using namespace std;

class PizzaParlor {

private:

string \*queue; // Array to store orders (strings)

int front, rear, size;

public:

PizzaParlor(int M) {

size = M;

queue = new string[size];

front = rear = -1;

}

~PizzaParlor() {

delete[] queue;

}

// Check if the queue is full

bool isFull() {

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

}

// Check if the queue is empty

bool isEmpty() {

return front == -1;

}

// Add an order to the queue

void placeOrder(string order) {

if (isFull()) {

cout << "Sorry, the pizza parlor is at full capacity. Cannot accept more orders." << endl;

return;

}

if (front == -1) { // If the queue is empty

front = rear = 0;

} else if (rear == size - 1 && front != 0) { // Wrap around if needed

rear = 0;

} else {

rear++;

}

queue[rear] = order;

cout << "Order placed: " << order << endl;

}

void serveOrder() {

if (isEmpty()) {

cout << "No orders to serve." << endl;

return;

}

cout << "Serving order: " << queue[front] << endl;

if (front == rear) { // Only one order in the queue

front = rear = -1;

} else if (front == size - 1) { // Wrap around if needed

front = 0;

} else {

front++;

}

}

void displayOrders() {

if (isEmpty()) {

cout << "No orders in the queue." << endl;

return;

}

cout << "Current orders in the pizza parlor: ";

int i = front;

while (i != rear) {

cout << queue[i] << " ";

i = (i + 1) % size; // Circular increment

}

cout << queue[rear] << endl; // Print the last order

}

};

int main() {

int maxOrders;

cout << "Enter the maximum number of orders the pizza parlor can accept: ";

cin >> maxOrders;

PizzaParlor parlor(maxOrders);

int choice;

string order;

do {

cout << "\nPizza Parlor Menu:" << endl;

cout << "1. Place an Order" << endl;

cout << "2. Serve an Order" << endl;

cout << "3. Display Current Orders" << endl;

cout << "4. Exit" << endl;

cout << "Enter your choice: ";

cin >> choice;

switch (choice) {

case 1:

cout << "Enter the order (Pizza type): ";

cin >> order;

parlor.placeOrder(order); // Place a new order

break;

case 2:

parlor.serveOrder(); // Serve (remove) an order

break;

case 3:

parlor.displayOrders(); // Display all orders in the queue

break;

case 4:

cout << "Exiting the system." << endl;

break;

default:

cout << "Invalid choice. Please try again." << endl;

}

} while (choice != 4);

return 0;

}