#include <bits/stdc++.h>using namespace std;

bool isValidName(const string& name) {

    for (char c : name) {

        if (!isalpha(c) && c != ' ') {

            return false;

        }

    }

    return true;

}

bool isValidNumber(const string& number) {

    for (char c : number) {

        if (!isdigit(c)) {

            return false;

        }

    }return true;

}

bool isValidCGPA(float cgpa) {

    return (cgpa >= 0.0 && cgpa <= 10.0);

}

int main() {

    string name, address, city, phone;

    string prn;

    float cgpa;

    cout << "Enter Name: ";

    getline(cin, name);

    if (!isValidName(name)) {

        cout << "Invalid name!" << endl;

        return 1;

    }

    cout << "Enter Address: ";

    getline(cin, address);

    cout << "Enter City: ";

    getline(cin, city);

    if (!isValidName(city)) {

        cout << "Invalid city!" << endl;

        return 1;

    }

    cout << "Enter Phone: ";

    cin >> phone;

    if (!isValidNumber(phone)) {

        cout << "Invalid phone number!" << endl;

        return 1;

    }

    cout << "Enter PRN: ";

    cin >> prn;

    cout << "Enter CGPA: ";

    cin >> cgpa;

    if (!isValidCGPA(cgpa)) {

        cout << "Invalid CGPA!" << endl;

        return 1;

    }

    cout << "\n--- Person Details ---" << endl;

    cout << "Name: " << name << endl;

    cout << "Address: " << address << endl;

    cout << "City: " << city << endl;

    cout << "Phone: " << phone << endl;

    cout << "\n--- Student Details ---" << endl;

    cout << "PRN: " << prn << endl;

    cout << "CGPA: " << cgpa << endl;

    return 0;

}

#include <bits/stdc++.h> exp2

using namespace std;

int main() {

    string movieName;

    double adultTicketPrice, childTicketPrice;

    int adultTicketsSold, childTicketsSold;

    double charityPercentage;

    cout << "Enter Movie Name: ";

    getline(cin, movieName);

    cout << "Enter Adult Ticket Price: ";

    cin >> adultTicketPrice;

    cout << "Enter Child Ticket Price: ";

    cin >> childTicketPrice;

    cout << "Enter Number of Adult Tickets Sold: ";

    cin >> adultTicketsSold;

    cout << "Enter Number of Child Tickets Sold: ";

    cin >> childTicketsSold;

    cout << "Enter Percentage of Gross Amount to be Donated to Charity: ";

    cin >> charityPercentage;

    // Calculations

    double grossAmount = (adultTicketsSold \* adultTicketPrice) + (childTicketsSold \* childTicketPrice);

    double charityAmount = (charityPercentage / 100) \* grossAmount;

    double netAmount = grossAmount - charityAmount;

    // Displaying results with IO manipulators

    cout << "\n----------------------------------------\n";

    cout << setw(30) << left << "Movie Name:" << movieName << endl;

    cout << setw(30) << left << "Adult Ticket Price:" << fixed << setprecision(2) << adultTicketPrice << endl;

    cout << setw(30) << left << "Child Ticket Price:" << fixed << setprecision(2) << childTicketPrice << endl;

    cout << setw(30) << left << "Adult Tickets Sold:" << adultTicketsSold << endl;

    cout << setw(30) << left << "Child Tickets Sold:" << childTicketsSold << endl;

    cout << setw(30) << left << "Gross Amount:" << "$" << fixed << setprecision(2) << grossAmount << endl;

    cout << setw(30) << left << "Charity Percentage:" << fixed << setprecision(2) << charityPercentage << "%" << endl;

    cout << setw(30) << left << "Charity Amount:" << "$" << fixed << setprecision(2) << charityAmount << endl;

    cout << setw(30) << left << "Net Amount:" << "$" << fixed << setprecision(2) << netAmount << endl;

    return 0;

}

#include <bits/stdc++.h> ##3

using namespace std;

class Date {

private:

    int day, month, year;

    vector<int> daysOfMonth;

public:

    Date() : day(0), month(0), year(0), daysOfMonth({31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31}) {}

    bool setDate(const string& date) {

        char delimiter = (date.find('/') != string::npos) ? '/' : '-';

        stringstream ss(date);

        string temp;

        getline(ss, temp, delimiter);

        day = stoi(temp);

        getline(ss, temp, delimiter);

        month = stoi(temp);

        getline(ss, temp);

        year = stoi(temp);

        return isValidDate();

    }

    string getDate() const {

        stringstream ss;

        ss << (day < 10 ? "0" : "") << day << "/"

           << (month < 10 ? "0" : "") << month << "/"

           << year;

        return ss.str();

    }

    void incrementDate() {

        day++;

        if (day > daysOfMonth[month - 1] + isLeapYearAdjustment()) {

            day = 1;

            month++;

            if (month > 12) {

                month = 1;

                year++;

            }

        }

    }

    void decrementDate() {

        day--;

        if (day == 0) {

            month--;

            if (month == 0) {

                month = 12;

                year--;

            }

            day = daysOfMonth[month - 1] + isLeapYearAdjustment();

        }

    }

    Date dateDifference(const Date& d2) const {

        Date diff;

        diff.day = abs(day - d2.day);

        diff.month = abs(month - d2.month);

        diff.year = abs(year - d2.year);

        return diff;

    }

    int getDay() const { return day; }

    int getMonth() const { return month; }

    int getYear() const { return year; }

private:

    bool isValidDate() const {

        if (month < 1 || month > 12 || day < 1) {

            return false;

        }

        int maxDay = daysOfMonth[month - 1] + isLeapYearAdjustment();

        return day <= maxDay;

    }

    int isLeapYearAdjustment() const {

        return (month == 2 && ((year % 4 == 0 && year % 100 != 0) || (year % 400 == 0))) ? 1 : 0;

    }

};

int main() {

    Date date1, date2;

    string inputDate;

    cout << "Enter the first date (dd/mm/yyyy or dd-mm-yyyy): ";

    cin >> inputDate;

    if (date1.setDate(inputDate)) {

        cout << "Date 1 is valid: " << date1.getDate() << endl;

    } else {

        cout << "Invalid Date 1!" << endl;

        return 1;

    }

    date1.incrementDate();

    cout << "Date after incrementing: " << date1.getDate() << endl;

    date1.decrementDate();

    cout << "Date after decrementing: " << date1.getDate() << endl;

    cout << "Enter the second date (dd/mm/yyyy or dd-mm-yyyy): ";

    cin >> inputDate;

    if (date2.setDate(inputDate)) {

        cout << "Date 2 is valid: " << date2.getDate() << endl;

    } else {

        cout << "Invalid Date 2!" << endl;

        return 1;

    }

    Date diff = date1.dateDifference(date2);

    cout << "Date difference: " << diff.getYear() << " years, "

         << diff.getMonth() << " months, "

         << diff.getDay() << " days." << endl;

    return 0;

}

#include<bits/stdc++.h> ##4

using namespace std;

class EnhancedArray {

private:

    vector<int> arr; // Internal vector to store elements

public:

    EnhancedArray() {}

    void addElement(int value) {

        arr.push\_back(value);

    }

    int getMax() {

        if (arr.empty()) {

            throw runtime\_error("Array is empty. No maximum value.");

        }

        return \*max\_element(arr.begin(), arr.end());

    }

    int getMin() {

        if (arr.empty()) {

            throw runtime\_error("Array is empty. No minimum value.");

        }

        return \*min\_element(arr.begin(), arr.end());

    }

    double getAverage() {

        if (arr.empty()) {

            throw runtime\_error("Array is empty. Cannot calculate average.");

        }

        return static\_cast<double>(accumulate(arr.begin(), arr.end(), 0)) / arr.size();

    }

    void resizeArray(size\_t newSize, int fillValue = 0) {

        arr.resize(newSize, fillValue);

    }

    void displayArray() const {

        if (arr.empty()) {

            cout << "Array is empty." << endl;

            return;

        }

        cout << "Array Elements: ";

        for (int val : arr) {

            cout << val << " ";

        }

        cout << endl;

    }

    void clearArray() {

        arr.clear();

    }

};

int main() {

    EnhancedArray array;

    int choice, value;

    size\_t newSize;

    while (true) {

        cout << "\n--- Enhanced Array Menu ---" << endl;

        cout << "1. Add Element" << endl;

        cout << "2. Display Array" << endl;

        cout << "3. Get Maximum" << endl;

        cout << "4. Get Minimum" << endl;

        cout << "5. Get Average" << endl;

        cout << "6. Resize Array" << endl;

        cout << "7. Clear Array" << endl;

        cout << "8. Exit" << endl;

        cout << "Enter your choice: ";

        cin >> choice;

        try {

            switch (choice) {

                case 1:

                    cout << "Enter value to add: ";

                    cin >> value;

                    array.addElement(value);

                    break;

                case 2:

                    array.displayArray();

                    break;

                case 3:

                    cout << "Maximum Value: " << array.getMax() << endl;

                    break;

                case 4:

                    cout << "Minimum Value: " << array.getMin() << endl;

                    break;

                case 5:

                    cout << "Average Value: " << array.getAverage() << endl;

                    break;

                case 6:

                    cout << "Enter new size: ";

                    cin >> newSize;

                    cout << "Enter fill value (default is 0): ";

                    cin >> value;

                    array.resizeArray(newSize, value);

                    break;

                case 7:

                    array.clearArray();

                    cout << "Array cleared." << endl;

                    break;

                case 8:

                    cout << "Exiting program." << endl;

                    return 0;

                default:

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

                    break;

            }

        } catch (const exception& e) {

            cout << "Error: " << e.what() << endl;

        }

    }

    return 0;

}

#include <iostream>##5

#include <string>

using namespace std;

int add(int a, int b) {

    return a + b;

}

float add(float a, float b) {

    return a + b;

}

int add(int a, int b, int c) {

    return a + b + c;

}

string add(string a, string b) {

    return a + b;

}

int main() {

    int int1 = 10, int2 = 20, int3 = 30;

    float float1 = 1.5, float2 = 2.5;

    string str1 = "Hello, ", str2 = "World!";

    cout << "Addition of two integers: " << add(int1, int2) << endl;

    cout << "Addition of three integers: " << add(int1, int2, int3) << endl;

    cout << "Addition of two floats: " << add(float1, float2) << endl;

    cout << "Concatenation of two strings: " << add(str1, str2) << endl;

    return 0;

}

#include <iostream>##6

using namespace std;

class Complex {

private:

    float real;

    float imaginary;

public:

    Complex(float r = 0, float i = 0) : real(r), imaginary(i) {}

    Complex operator+(const Complex& other) {

        return Complex(real + other.real, imaginary + other.imaginary);

    }

    Complex operator-(const Complex& other) {

        return Complex(real - other.real, imaginary - other.imaginary);

    }

    void display() const {

        cout << real << " + " << imaginary << "i" << endl;

    }

};

int main() {

    Complex c1(4.5, 7.3), c2(2.5, 3.7);

    Complex sum = c1 + c2;

    Complex difference = c1 - c2;

    cout << "Complex Number 1: ";

    c1.display();

    cout << "Complex Number 2: ";

    c2.display();

    cout << "Sum: ";

    sum.display();

    cout << "Difference: ";

    difference.display();

    return 0;

}

#include <iostream>##7

using namespace std;

class Complex {

private:

    float real;

    float imaginary;

public:

    Complex(float r = 0, float i = 0) : real(r), imaginary(i) {}

    friend istream& operator>>(istream& input, Complex& c) {

        input >> c.real >> c.imaginary;

        return input;

    }

    friend ostream& operator<<(ostream& output, const Complex& c) {

        output << c.real;

        if (c.imaginary >= 0)

            output << " + " << c.imaginary << "i";

        else

            output << " - " << -c.imaginary << "i";

        return output;

    }

};

int main() {

    Complex c1, c2;

    cout << "Enter the first complex number (real and imaginary parts): ";

    cin >> c1;

    cout << "Enter the second complex number (real and imaginary parts): ";

    cin >> c2;

    cout << "\nThe first complex number is: " << c1 << endl;

    cout << "The second complex number is: " << c2 << endl;

    return 0;

#include <bits/stdc++.h>##8

using namespace std;

class Person {

protected:

    string name, address, city, phone;

public:

    Person(string p\_name, string p\_address, string p\_city, string p\_phone) {

        name = p\_name;

        address = p\_address;

        city = p\_city;

        phone = p\_phone;

    }

    void displayPersonDetails() {

        cout << "Name: " << name << "\nAddress: " << address

             << "\nCity: " << city << "\nPhone: " << phone << endl;

    }

};

class Customer : public Person {

private:

    int customerID;

    string customerEmail;

public:

    Customer(string c\_name, string c\_address, string c\_city, string c\_phone, int c\_ID, string c\_email)

        : Person(c\_name, c\_address, c\_city, c\_phone) {

        customerID = c\_ID;

        customerEmail = c\_email;

    }

    void displayCustomerDetails() {

        displayPersonDetails();

        cout << "Customer ID: " << customerID << "\nCustomer Email: " << customerEmail << endl;

    }

};

class Employee : public Person {

private:

    int empID;

    float salary;

public:

    Employee(string e\_name, string e\_address, string e\_city, string e\_phone, int e\_ID, float e\_salary)

        : Person(e\_name, e\_address, e\_city, e\_phone) {

        empID = e\_ID;

        salary = e\_salary;

    }

    void displayEmployeeDetails() {

        displayPersonDetails();

        cout << "Employee ID: " << empID << "\nSalary: " << salary << endl;

    }

};

int main() {

    Person person("John Doe", "123 Elm St", "Metropolis", "555-1234");

    Customer customer("Jane Doe", "456 Oak St", "Gotham", "555-5678", 101, "jane.doe@example.com");

    Employee employee("Jim Beam", "789 Pine St", "Star City", "555-8765", 201, 75000.0);

    cout << "Person Details:" << endl;

    person.displayPersonDetails();

    cout << "\nCustomer Details:" << endl;

    customer.displayCustomerDetails();

    cout << "\nEmployee Details:" << endl;

    employee.displayEmployeeDetails();

    return 0;

}

#include <iostream>##9

using namespace std;

class Stack {

private:

    int \*stack\_array;

    int top;

    int size;

public:

    Stack() {

        size = 10; // Default size

        stack\_array = new int[size];

        top = -1;

    }

    Stack(int stack\_size) {

        size = stack\_size;

        stack\_array = new int[size];

        top = -1;

    }

    ~Stack() {

        delete[] stack\_array;

    }

    bool push(int data) {

        if (isFull()) {

            cout << "Stack Overflow" << endl;

            return false;

        }

        stack\_array[++top] = data;

        return true;

    }

    int pop() {

        if (isEmpty()) {

            cout << "Stack Underflow" << endl;

            return -1;

        }

        return stack\_array[top--];

    }

    bool isEmpty() {

        return (top == -1);

    }

    bool isFull() {

        return (top == size - 1);

    }

};

class CircularQueue {

private:

    int \*queue\_array;

    int front, rear;

    int space\_used;

    int size;

public:

    CircularQueue() {

        size = 10; // Default size

        queue\_array = new int[size];

        front = rear = -1;

        space\_used = 0;

    }

    CircularQueue(int queue\_size) {

        size = queue\_size;

        queue\_array = new int[size];

        front = rear = -1;

        space\_used = 0;

    }

    ~CircularQueue() {

        delete[] queue\_array;

    }

    bool enqueue(int data) {

        if (isFull()) {

            cout << "Queue Overflow" << endl;

            return false;

        }

        if (isEmpty()) {

            front = rear = 0;

        } else {

            rear = (rear + 1) % size;

        }

        queue\_array[rear] = data;

        space\_used++;

        return true;

    }

    int dequeue() {

        if (isEmpty()) {

            cout << "Queue Underflow" << endl;

            return -1;

        }

        int data = queue\_array[front];

        if (front == rear) {

            front = rear = -1;

        } else {

            front = (front + 1) % size;

        }

        space\_used--;

        return data;

    }

    bool isFull() {

        return (space\_used == size);

    }

    bool isEmpty() {

        return (space\_used == 0);

    }

};

int main() {

    cout << "Stack Operations:" << endl;

    Stack stack(5);

    stack.push(10);

    stack.push(20);

    stack.push(30);

    cout << "Popped element: " << stack.pop() << endl;

    cout << "Popped element: " << stack.pop() << endl;

    cout << "Is stack empty? " << (stack.isEmpty() ? "Yes" : "No") << endl;

    cout << "\nQueue Operations:" << endl;

    CircularQueue queue(5);

    queue.enqueue(1);

    queue.enqueue(2);

    queue.enqueue(3);

    cout << "Dequeued element: " << queue.dequeue() << endl;

    cout << "Dequeued element: " << queue.dequeue() << endl;

    cout << "Is queue empty? " << (queue.isEmpty() ? "Yes" : "No") << endl;

    return 0;

}

#include <iostream>##10

using namespace std;

class Node {

public:

    int data;

    Node \*next;

    Node() : data(0), next(nullptr) {}

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

};

class LinkedList {

private:

    Node \*head;

    Node\* createNode(int data) {

        return new Node(data);

    }

public:

    LinkedList() : head(nullptr) {}

    ~LinkedList() {

        Node \*current = head;

        while (current != nullptr) {

            Node \*nextNode = current->next;

            delete current;

            current = nextNode;

        }

        head = nullptr;

    }

    void attachBegin(int data) {

        Node \*newNode = createNode(data);

        newNode->next = head;

        head = newNode;

    }

    void attachEnd(int data) {

        Node \*newNode = createNode(data);

        if (head == nullptr) {

            head = newNode;

            return;

        }

        Node \*temp = head;

        while (temp->next != nullptr) {

            temp = temp->next;

        }

        temp->next = newNode;

    }

    Node\* detachBegin() {

        if (head == nullptr) {

            cout << "List is empty." << endl;

            return nullptr;

        }

        Node \*temp = head;

        head = head->next;

        return temp;

    }

    Node\* detachEnd() {

        if (head == nullptr) {

            cout << "List is empty." << endl;

            return nullptr;

        }

        if (head->next == nullptr) {

            Node \*temp = head;

            head = nullptr;

            return temp;

        }

        Node \*temp = head;

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

            temp = temp->next;

        }

        Node \*toDelete = temp->next;

        temp->next = nullptr;

        return toDelete;

    }

    void traverse() {

        if (head == nullptr) {

            cout << "List is empty." << endl;

            return;

        }

        Node \*temp = head;

        while (temp != nullptr) {

            cout << temp->data << " -> ";

            temp = temp->next;

        }

        cout << "NULL" << endl;

    }

    Node\* search(int data) {

        Node \*temp = head;

        while (temp != nullptr) {

            if (temp->data == data) {

                return temp;

            }

            temp = temp->next;

        }

        return nullptr;

    }

};

int main() {

    LinkedList list;

    list.attachBegin(10);

    list.attachBegin(20);

    list.attachEnd(30);

    list.attachEnd(40);

    cout << "Linked List: ";

    list.traverse();

    int searchValue = 30;

    Node \*foundNode = list.search(searchValue);

    if (foundNode) {

        cout << "Element " << searchValue << " found in the list." << endl;

    } else {

        cout << "Element " << searchValue << " not found in the list." << endl;

    }

    Node \*deletedNode = list.detachBegin();

    if (deletedNode) {

        cout << "Deleted from beginning: " << deletedNode->data << endl;

        delete deletedNode;

    }

    deletedNode = list.detachEnd();

    if (deletedNode) {

        cout << "Deleted from end: " << deletedNode->data << endl;

        delete deletedNode;

    }

    cout << "Linked List after deletions: ";

    list.traverse();

    return 0;

}

#include <iostream>##11

using namespace std;

template <typename T>

bool linearSearch(T arr[], int size, T key) {

    for (int i = 0; i < size; i++) {

        if (arr[i] == key) {

            return true; // Key found

        }

    }

    return false; // Key not found

}

int main() {

    int intArr[] = {1, 2, 3, 4, 5};

    int intKey = 3;

    cout << "Searching " << intKey << " in integer array: "

         << (linearSearch(intArr, 5, intKey) ? "Found" : "Not Found") << endl;

    float floatArr[] = {1.1, 2.2, 3.3, 4.4, 5.5};

    float floatKey = 4.4;

    cout << "Searching " << floatKey << " in float array: "

         << (linearSearch(floatArr, 5, floatKey) ? "Found" : "Not Found") << endl;

    char charArr[] = {'a', 'b', 'c', 'd', 'e'};

    char charKey = 'd';

    cout << "Searching " << charKey << " in character array: "

         << (linearSearch(charArr, 5, charKey) ? "Found" : "Not Found") << endl;

    return 0;

}

#include <iostream>##11.1

using namespace std;

// Class Template for Stack

template <typename T>

class Stack {

private:

    T \*stackArray; // Pointer to stack array

    int top;       // Index of tops element

    int capacity;  // Maximum capacity of stack

public:

    Stack(int size) : capacity(size), top(-1) {

        stackArray = new T[size];

    }

    ~Stack() {

        delete[] stackArray;

    }

    void push(T value) {

        if (top == capacity - 1) {

            cout << "Stack Overflow!" << endl;

            return;

        }

        stackArray[++top] = value;

    }

    T pop() {

        if (top == -1) {

            cout << "Stack Underflow!" << endl;

            return T(); // Return default value for the type

        }

        return stackArray[top--];

    }

    void display() {

        if (top == -1) {

            cout << "Stack is empty." << endl;

            return;

        }

        for (int i = 0; i <= top; i++) {

            cout << stackArray[i] << " ";

        }

        cout << endl;

    }

};

int main() {

    Stack<int> intStack(5);

    intStack.push(10);

    intStack.push(20);

    intStack.push(30);

    cout << "Integer Stack: ";

    intStack.display();

    cout << "Popped: " << intStack.pop() << endl;

    intStack.display();

    Stack<float> floatStack(5);

    floatStack.push(1.1);

    floatStack.push(2.2);

    floatStack.push(3.3);

    cout << "Float Stack: ";

    floatStack.display();

    cout << "Popped: " << floatStack.pop() << endl;

    floatStack.display();

    Stack<char> charStack(5);

    charStack.push('A');

    charStack.push('B');

    charStack.push('C');

    cout << "Character Stack: ";

    charStack.display();

    cout << "Popped: " << charStack.pop() << endl;

    charStack.display();

    return 0;

}

#include <iostream>##12

using namespace std;

class Employee {

protected:

    double salary;

public:

    Employee(double sal) : salary(sal) {}

    virtual ~Employee() {}

    virtual void raiseSalary() = 0;

    void showSalary() const {

        cout << "Current Salary: " << salary << endl;

    }

};

class Engineer : public Employee {

public:

    Engineer(double sal) : Employee(sal) {}

    void raiseSalary() override {

        salary += salary \* 0.10; // Raise salary by 10%

    }

};

class TeamLeader : public Employee {

public:

    TeamLeader(double sal) : Employee(sal) {}

    void raiseSalary() override {

        salary += salary \* 0.15;

    }

};

class Manager : public Employee {

public:

    Manager(double sal) : Employee(sal) {}

    void raiseSalary() override {

        salary += salary \* 0.20; // Raise salary by 20%

    }

};

int main() {

    Engineer eng(50000);

    TeamLeader tl(70000);

    Manager mgr(100000);

    Employee\* employees[3];

    employees[0] = &eng;

    employees[1] = &tl;

    employees[2] = &mgr;

    cout << "Initial Salaries:\n";

    for (int i = 0; i < 3; ++i) {

        employees[i]->showSalary();

    }

    cout << "\nRaising Salaries:\n";

    for (int i = 0; i < 3; ++i) {

        employees[i]->raiseSalary();

        employees[i]->showSalary();

    }

    return 0;

}