Date:05/07/2024

**Day\_10\_Assignments**

Q.1 File Processing: Design a base class File with a virtual function readData() that has an empty body. Create derived classes like TextFile and ImageFile inheriting from File and overriding readData() with their specific reading procedures. Implement a function that takes a pointer to File as input, attempts to read the data using the readData() function, and handles potential errors based on the actual derived class type (e.g., different file formats).

#include <iostream>

#include <string>

using namespace std;

class File {

public:

virtual bool readData() = 0;

virtual ~File() = default;

};

class TextFile : public File {

public:

bool readData() override {

cout << "Reading data from a text file..." <<endl;

return false;

}

};

class ImageFile : public File {

public:

bool readData() override {

cout << "Reading data from an image file..." <<endl;

return false;

}

};

void processFile(File\* file) {

bool success = file->readData();

if (!success) {

if (dynamic\_cast<TextFile\*>(file)) {

cout << "Failed to read TextFile." <<endl;

} else if (dynamic\_cast<ImageFile\*>(file)) {

cout << "Failed to read ImageFile." <<endl;

} else {

cout << "Failed to read unknown file type." <<endl;

}

} else {

cout << "File read successfully."<<endl;

}

}

int main() {

TextFile textFile;

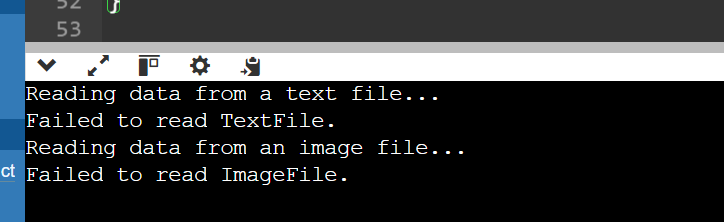
ImageFile imageFile;

processFile(&textFile);

processFile(&imageFile);

return 0;

}



Q.2 Design an abstract factory class hierarchy to create different families of products (e.g., furniture). Use pointers and runtime polymorphism. Define an abstract base class FurnitureFactory with a virtual function createChair(). Create derived classes like ModernFurnitureFactory and ClassicFurnitureFactory that override createChair() to return pointers to concrete chair objects specific to their style. Utilize the factory pattern with runtime polymorphism to allow for flexible furniture creation based on user choice.

#include <iostream>

using namespace std;

class Chair {

public:

virtual void sit() = 0;

virtual ~Chair() {}

};

class FurnitureFactory {

public:

virtual Chair\* createChair() = 0;

virtual ~FurnitureFactory() {}

};

class ModernChair : public Chair {

public:

void sit() override {

cout << "Sitting on a modern chair." <<endl;

}

};

class ClassicChair : public Chair {

public:

void sit() override {

cout << "Sitting on a classic chair." <<endl;

}

};

class ModernFurnitureFactory : public FurnitureFactory {

public:

Chair\* createChair() override {

return new ModernChair();

}

};

class ClassicFurnitureFactory : public FurnitureFactory {

public:

Chair\* createChair() override {

return new ClassicChair();

}

};

void createFurniture(FurnitureFactory\* factory) {

Chair\* chair = factory->createChair();

chair->sit();

delete chair;

}

int main() {

FurnitureFactory\* factory = nullptr;

cout << "Choose furniture style (1: Modern, 2: Classic): ";

int choice;

cin >> choice;

if (choice == 1) {

factory = new ModernFurnitureFactory();

} else if (choice == 2) {

factory = new ClassicFurnitureFactory();

} else {

cout << "Invalid choice." <<endl;

return 1;

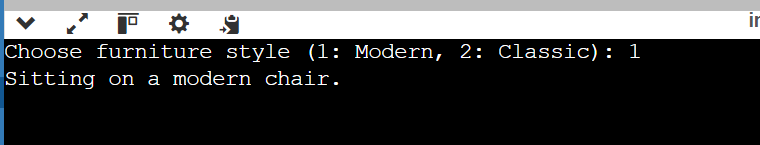
}

createFurniture(factory);

delete factory;

return 0;

}



Q.3 Problem Statement: Flight Booking System (C++ Structures & Functions)

Data Structures: Create a C++ structure named Flight to represent flight information, including:Flight number (string) Departure and arrival airports (strings)Departure and arrival date/time (strings or appropriate data types)Number of available seats (integer)Price per seat (float)Consider creating another structure named Passenger (optional) to store passenger details if needed (name, passport information etc.).Functions:

Develop C++ functions to:Display a list of available flights based on user-specified origin and destination airports (consider searching by date range as well).Book a specific number of seats for a chosen flight (handle cases where insufficient seats are available).Cancel a booking for a specific flight and number of seats (ensure the user cancels the correct booking).Display a list of all booked flights for a specific user (if using Passenger structure).Implement error handling for invalid user input (e.g., trying to book negative seats).Include a function to add new flights to the system (consider adding flights dynamically if needed).

#include <iostream>

#include <vector>

#include <string>

using namespace std;

struct Flight {

string flightNumber;

string departureAirport;

string arrivalAirport;

string departureDateTime;

string arrivalDateTime;

int availableSeats;

float pricePerSeat;

};

struct Passenger {

string name;

string passportInfo;

};

void displayAvailableFlights(const vector<Flight>& flights, const string& origin, const string& destination, const string& startDate, const string& endDate) {

cout << "Available flights from " << origin << " to " << destination << " between " << startDate << " and " << endDate << ":" << endl;

for (const Flight& flight : flights) {

if (flight.departureAirport == origin &&

flight.arrivalAirport == destination &&

flight.departureDateTime >= startDate &&

flight.departureDateTime <= endDate) {

cout << "Flight Number: " << flight.flightNumber << endl;

cout << "Departure Airport: " << flight.departureAirport << endl;

cout << "Arrival Airport: " << flight.arrivalAirport << endl;

cout << "Departure Date/Time: " << flight.departureDateTime << endl;

cout << "Arrival Date/Time: " << flight.arrivalDateTime << endl;

cout << "Available Seats: " << flight.availableSeats << endl;

cout << "Price per Seat: " << flight.pricePerSeat << endl;

cout << "---------------------" << endl;

}

}

}

bool bookSeats(vector<Flight>& flights, const string& flightNumber, int numSeatsToBook) {

for (Flight& flight : flights) {

if (flight.flightNumber == flightNumber) {

if (flight.availableSeats >= numSeatsToBook) {

flight.availableSeats -= numSeatsToBook;

cout << "Booking successful!" << endl;

return true;

} else {

cout << "Not enough seats available for booking." << endl;

return false;

}

}

}

cout << "Flight not found." << endl;

return false;

}

bool cancelBooking(vector<Flight>& flights, const string& flightNumber, int numSeatsToCancel) {

for (Flight& flight : flights) {

if (flight.flightNumber == flightNumber) {

flight.availableSeats += numSeatsToCancel;

cout << "Booking cancellation successful!" << endl;

return true;

}

}

cout << "Flight not found." << endl;

return false;

}

void displayBookedFlightsForUser(const vector<Flight>& flights, const string& userName) {

cout << "Booked flights for user " << userName << ":" << endl;

for (const Flight& flight : flights) {

cout << "Flight Number: " << flight.flightNumber << endl;

cout << "Departure Airport: " << flight.departureAirport << endl;

cout << "Arrival Airport: " << flight.arrivalAirport << endl;

cout << "Departure Date/Time: " << flight.departureDateTime << endl;

cout << "Arrival Date/Time: " << flight.arrivalDateTime << endl;

cout << "Available Seats: " << flight.availableSeats << endl;

cout << "Price per Seat: $" << flight.pricePerSeat << endl;

cout << "---------------------" << endl;

}

}

// Function to add new flight to the system

void addNewFlight(vector<Flight>& flights, const Flight& newFlight) {

flights.push\_back(newFlight);

cout << "New flight added successfully!" << endl;

}

int main() {

// Sample data initialization

vector<Flight> flights = {

{"ABC123", "jkj", "LAX", "2024-07-06", "2024-07-06", 100, 350.0},

{"DEF456", "LiX", "JlK", "2024-07-07", "2024-07-07", 150, 400.0},

{"GHI789", "SuO", "Oyt", "2024-07-08", "2024-07-08", 120, 300.0}

};

// Example usage of functions

displayAvailableFlights(flights, "abc", "xyz", "2024-07-05", "2024-07-10");

bookSeats(flights, "ABC123", 2);

cancelBooking(flights, "ABC123", 1);

displayBookedFlightsForUser(flights, "Dhanashri");

// Example of adding a new flight

Flight newFlight = {"XYZ789", "abc", "xyz", "2024-07-15", "2024-07-16", 100, 500.0};

addNewFlight(flights, newFlight);

return 0;

}

Q.4

#include <iostream>

using namespace std;

int multiply(int a,int b);

int main()

{

cout<<multiply(4,5)<<endl;

cout<<[](int a,int b){ return a\*b; }(4,5)<<endl;

auto f=[](int a,int b){ return a\*b; };

cout<<f(4,5)<<endl;

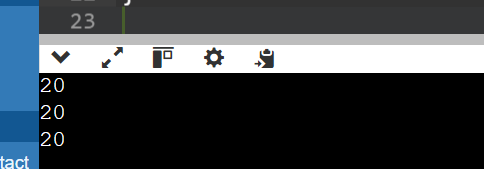
}

int multiply(int a,int b)

{

return a\*b;

}



Q.5

#include <iostream>

using namespace std;

void lamda\_value\_capture() {

int value=1;

auto copy\_value=[value]{

return value;

};

value =100;

auto stored\_value=copy\_value();

cout<<stored\_value<<endl;

}

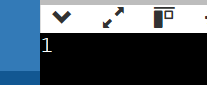
int main()

{

lamda\_value\_capture();

return 0;

}



Q.6

#include <iostream>

using namespace std;

void lamda\_reference\_capture() {

int value=1;

auto copy\_value=[&value]{

return value;

};

value =100;

auto stored\_value=copy\_value();

cout<<"stored value ="<<stored\_value<<endl;

}

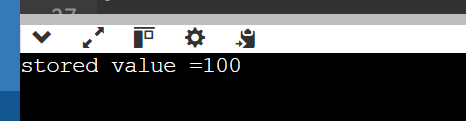
int main()

{

lamda\_reference\_capture();

return 0;

}



Q.7

#include <iostream>

using namespace std;

int main()

{

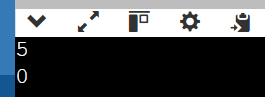
int m=0;

int n=0;

[&,n](int a)mutable{m=++n + a;}(4);

cout<<m<<endl<<n<<endl;

}



Q.8

#include <iostream>

#include<algorithm>

#include<vector>

using namespace std;

void assign(int& v)

{

static int n =1;

v=n++;

}

void print(int v)

{

cout<<v<<" ";

}

int main()

{

vector<int> vec(10);

for\_each(vec.begin(),vec.end(),print);

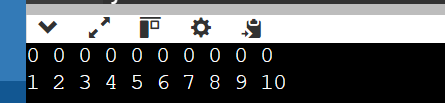
cout<<endl;

for\_each(vec.begin(),vec.end(),assign);

for\_each(vec.begin(),vec.end(),print);

return 0;

}



Q.9 Practice Problem Statement:

Scenario: You're working on a data analysis project where you need to filter a list of integers based on whether they are even or odd. You want to use a lambda expression to achieve this filtering.Task:Define a function named filter\_even\_odds that takes two arguments:const std::vector<int>& numbers: The vector containing the integer values.bool is\_even: A flag indicating whether to filter even (true) or odd (false) numbers.Inside the function, use a lambda expression to iterate through the numbers vector.Within the lambda, check if the current number is even using the modulo operator (%).If the even/odd condition matches the is\_even flag, add the number to a new filtered vector.Return the filtered vector from the filter\_even\_odds function.

#include <iostream>

#include <vector>

#include <algorithm>

using namespace std;

std::vector<int> filter\_even\_odds(const std::vector<int>& numbers, bool is\_even) {

std::vector<int> filtered\_numbers;

//lamda exprssion

auto check\_even\_odd = [is\_even](int num) {

return (is\_even && num % 2 == 0) || (!is\_even && num % 2 != 0);

};

std::copy\_if(numbers.begin(), numbers.end(), std::back\_inserter(filtered\_numbers), check\_even\_odd);

return filtered\_numbers;

}

int main() {

std::vector<int> numbers = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};

std::vector<int> evens = filter\_even\_odds(numbers, true);

cout << "Even numbers: "<<endl;

for (int num : evens) {

cout <<num<< " ";

}

std::vector<int> odds = filter\_even\_odds(numbers, false);

cout <<"\n Odd numbers: "<<endl;

for (int num : odds) {

cout <<num<< " ";

}

return 0;

}

