**6.** #include <iostream>

using namespace std;

class TreasureMap {

private:

int x;

int y;

public:

TreasureMap(int posX, int posY) : x(posX), y(posY) {}

TreasureMap operator+(const TreasureMap& other) const {

return TreasureMap(this->x + other.x, this->y + other.y);

}

void displayMap() const {

std::cout << "Current Position: (" << x << ", " << y << ")" << std::endl;

}

};

int main() {

int posX, posY;

cout << "Enter current X coordinate: ";

cin >> posX;

cout << "Enter current Y coordinate: ";

cin >> posY;

TreasureMap map1(posX, posY);

cout << "Enter another X coordinate: ";

cin >> posX;

cout << "Enter another Y coordinate: ";

cin >> posY;

TreasureMap map2(posX, posY);

TreasureMap combinedMap = map1 + map2;

cout << "\nFirst Map:" << std::endl;

map1.displayMap();

cout << "\nSecond Map:" << std::endl;

map2.displayMap();

cout << "\nCombined Map:" << std::endl;

combinedMap.displayMap();

return 0;

}

**OUTPUT:**

**Enter current X coordinate: 10**

**Enter current Y coordinate: 20**

**Enter another X coordinate: 11**

**Enter another Y coordinate: 11First Map:**

**Current Position: (10, 20)**

**Second Map:**

**Current Position: (11, 11)**

**Combined Map:**

**Current Position: (21, 31)**

**13.** #include <iostream>

#include <vector>

#include <string>

using namespace std;

// Abstract class Instrument

class Instrument {

public:

virtual void play() const = 0; // Pure virtual function

virtual ~Instrument() {} // Virtual destructor for polymorphic behavior

};

// Derived class Guitar

class Guitar : public Instrument {

public:

void play() const override {

cout << "Playing Guitar: ♫ ♫ ♫" << std::endl;

}

};

// Derived class Piano

class Piano : public Instrument {

public:

void play() const override {

cout << "Playing Piano: ♪ ♪ ♪" << std::endl;

}

};

// Derived class Flute

class Flute : public Instrument {

public:

void play() const override {

cout << "Playing Flute: ˜˜ ˜˜ ˜˜" << std::endl;

}

};

int main() {

vector<Instrument\*> instruments; // Vector to store instrument pointers

char choice;

do {

cout << "Select an instrument to play:" << std::endl;

cout << "1. Guitar\n2. Piano\n3. Flute\nEnter your choice (1-3): ";

int instrumentChoice;

cin >> instrumentChoice;

Instrument\* newInstrument = nullptr;

switch (instrumentChoice) {

case 1:

newInstrument = new Guitar();

break;

case 2:

newInstrument = new Piano();

break;

case 3:

newInstrument = new Flute();

break;

default:

std::cout << "Invalid choice!" << std::endl;

break;

}

if (newInstrument) {

instruments.push\_back(newInstrument);

// Play the selected instrument

newInstrument->play();

cout << "Do you want to play another instrument? (y/n): ";

cin >> choice;

} else {

choice = 'n'; // Set to 'n' to exit loop if choice was invalid

}

} while (choice == 'y' || choice == 'Y');

// Clean up allocated memory for instruments

for (auto& instrument : instruments) {

delete instrument;

}

instruments.clear();

return 0;

}

**OUTPUT = Select an instrument to play:**

**1. Guitar**

**2. Piano**

**3. Flute**

**Enter your choice (1-3): 2**

**Playing Piano: ♪ ♪ ♪**

**Do you want to play another instrument? (y/n):**