Q4 Solution

#include <iostream>

using namespace std;

class Furniture {

public:

virtual void display() =0;

};

class Bed : public Furniture {

int length;

int breadth;

public:

Bed(int l, int b)

{length=l;

breadth=b; }

void display() {

cout << "A Bed of length " << length << " and breadth " << breadth << endl;

}

};

class Chair : public Furniture {

int height;

public:

Chair(int h)

{height=h;}

void display() {

cout << "A " << height << " feet high chair" << endl;

}

};

class Inventory {

Furniture\* items[10]; // Assuming a maximum of 10 items

int count;

public:

Inventory()

{count=0;}

void AddBed(int length, int breadth) {

items[count++] = new Bed(length, breadth);

}

void AddChair(int height) {

items[count++] = new Chair(height);

}

void displayFurniture() const {

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

items[i]->display();

}

}

~Inventory() {

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

delete items[i];

}

}

};

int main() {

Inventory inv;

inv.AddBed(10, 5);

inv.AddChair(2);

inv.AddBed(7, 3);

inv.AddChair(4);

inv.displayFurniture();

return 0;

}

Q5 Solution

#include <iostream>

using namespace std;

class Matrix {

int rows, cols;

int\*\* matrix;

public:

// Default constructor

Matrix() {

rows=0;

cols=0;

matrix=nullptr;}

// Parameterized constructor to initialize the matrix with given rows and columns

Matrix(int r, int c)

{

rows=r;

cols=c;

// Dynamically allocate memory for the matrix

matrix = new int\*[rows];

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

matrix[i] = new int[cols];

}

// Input values from the user

cout << "Enter matrix elements:\n";

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

for (int j = 0; j < cols; ++j) {

cin >> matrix[i][j];

}

}

}

// Overloaded assignment operator to assign one matrix to another

Matrix& operator=(const Matrix& other) {

if (this != &other) { // Check for self-assignment

// Deallocate memory for the current matrix

if (matrix != nullptr) {

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

delete[] matrix[i];

}

delete[] matrix;

}

// Copy dimensions

rows = other.rows;

cols = other.cols;

// Dynamically allocate memory for the new matrix

matrix = new int\*[rows];

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

matrix[i] = new int[cols];

// Copy elements from the other matrix

for (int j = 0; j < cols; ++j) {

matrix[i][j] = other.matrix[i][j];

}

}

}

return \*this;

}

// Overloaded addition operator to add two matrices

Matrix operator+(const Matrix& x) const {

// Check if dimensions match

if (rows != x.rows || cols != x.cols) {

cerr << "Error: Matrix dimensions mismatch!\n";

exit(1);

}

// Create a new matrix to store the result

Matrix result(rows, cols);

// Add corresponding elements

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

for (int j = 0; j < cols; ++j) {

result.matrix[i][j] = matrix[i][j] + x.matrix[i][j];

}

}

return result;

}

// Function to print the matrix

void print() const {

cout << "Matrix:\n";

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

for (int j = 0; j < cols; ++j) {

cout << matrix[i][j] << " ";

}

cout << endl;

}

}

// Destructor to deallocate memory

~Matrix() {

// Deallocate memory for the matrix

if (matrix != nullptr) {

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

delete[] matrix[i];

}

delete[] matrix;

}

}

};

int main() {

Matrix M1(2, 3), M2(2, 3), M3;

M3 = M1 + M2;

M3.print();

return 0;

}

Q6 Solution

#include <iostream>

using namespace std;

template <class T, class U>

class Number{

T num;

U num1;

public:

Number(T anum, U anum1)

{

num=anum;

num1=anum1;

}

auto add()

{

return num+num1;

}

auto sub()

{

return num-num1;

}

};

int main() {

Number <int,double>n(3,5.5);

cout<<n.add()<<endl;

cout<<n.sub()<<endl;

return 0;

}