1. Write a c++ program to overload the ++ operator to increment a variable

Code:

#include <iostream>

class Counter {

private:

int count;

public:

Counter() : count(0) {}

// Overloading the prefix increment operator (++count)

Counter& operator++() {

++count;

return \*this;

}

// Overloading the postfix increment operator (count++)

Counter operator++(int) {

Counter temp = \*this;

++(\*this);

return temp;

}

void display() {

std::cout << "Count: " << count << std::endl;

}

};

int main() {

Counter c;

c.display(); // Output: Count: 0

++c;

c.display(); // Output: Count: 1

c++;

c.display(); // Output: Count: 2

return 0;

}

Output:

Count: 0

Count: 1

Count: 2

1. Write a c++ program to overload the + operator to add two variables

CODE:

#include <iostream>

class MyNumber {

private:

int value;

public:

MyNumber(int val) : value(val) {}

// Overloading the + operator to add two MyNumber objects

MyNumber operator+(const MyNumber& other) {

return MyNumber(value + other.value);

}

// Getter method to retrieve the value

int getValue() const {

return value;

}

};

int main() {

MyNumber num1(5);

MyNumber num2(10);

MyNumber sum = num1 + num2;

std::cout << "Sum: " << sum.getValue() << std::endl; // Output: Sum: 15

return 0;

}

Output:

Sum: 15

3.     Write a c++ program to overload the << operator to print contents of a user defined class

Code:

#include <iostream>

class MyClass {

private:

int data;

public:

MyClass(int d) : data(d) {}

// Overloading the << operator to print the contents of MyClass

friend std::ostream& operator<<(std::ostream& os, const MyClass& obj) {

os << "Data: " << obj.data;

return os;

}

};

int main() {

MyClass obj(42);

// Using the overloaded << operator to print the contents of obj

std::cout << obj << std::endl; // Output: Data: 42

return 0;

}

Output:

Data: 42

4.Write a c++ program to overload the == operator to compare two objects of a user defined class

Code:

#include <iostream>

class MyClass {

private:

int data;

public:

MyClass(int d) : data(d) {}

// Overloading the == operator to compare two MyClass objects

bool operator==(const MyClass& other) const {

return this->data == other.data;

}

};

int main() {

MyClass obj1(5);

MyClass obj2(10);

MyClass obj3(5);

// Comparing obj1 with obj2

if (obj1 == obj2) {

std::cout << "obj1 is equal to obj2" << std::endl;

} else {

std::cout << "obj1 is not equal to obj2" << std::endl;

}

// Comparing obj1 with obj3

if (obj1 == obj3) {

std::cout << "obj1 is equal to obj3" << std::endl;

} else {

std::cout << "obj1 is not equal to obj3" << std::endl;

}

return 0;

}

Output:

obj1 is not equal to obj2

obj1 is equal to obj3

5.Write a c++ program to overload the \* operator to multiply two matrices

Code:

#include <iostream>

#include <vector>

using namespace std;

class Matrix {

private:

vector<vector<int>> mat;

int rows;

int cols;

public:

Matrix(int r, int c) : rows(r), cols(c) {

mat.resize(rows, vector<int>(cols, 0));

}

// Overloading the \* operator for matrix multiplication

Matrix operator\*(const Matrix& other) {

if (cols != other.rows) {

cout << "Matrices cannot be multiplied due to incompatible dimensions." << endl;

return Matrix(0, 0);

}

Matrix result(rows, other.cols);

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

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

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

result.mat[i][j] += mat[i][k] \* other.mat[k][j];

}

}

}

return result;

}

// Function to set values in the matrix

void setValue(int i, int j, int value) {

mat[i][j] = value;

}

// Function to display the matrix

void display() {

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

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

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

}

cout << endl;

}

cout << endl;

}

};

int main() {

Matrix mat1(2, 3);

mat1.setValue(0, 0, 1);

mat1.setValue(0, 1, 2);

mat1.setValue(0, 2, 3);

mat1.setValue(1, 0, 4);

mat1.setValue(1, 1, 5);

mat1.setValue(1, 2, 6);

Matrix mat2(3, 2);

mat2.setValue(0, 0, 7);

mat2.setValue(0, 1, 8);

mat2.setValue(1, 0, 9);

mat2.setValue(1, 1, 10);

mat2.setValue(2, 0, 11);

mat2.setValue(2, 1, 12);

cout << "Matrix 1:" << endl;

mat1.display();

cout << "Matrix 2:" << endl;

mat2.display();

Matrix result = mat1 \* mat2;

cout << "Resultant Matrix:" << endl;

result.display();

return 0;

}

Output:

Matrix 1:

1 2 3

4 5 6

Matrix 2:

7 8

9 10

11 12

Resultant Matrix:

58 64

139 154

6.     Write a c++ program to overload the [] operator to access the elements in an array using index values

Code:

#include <iostream>

using namespace std;

class Array {

private:

int\* arr;

int size;

public:

Array(int s) : size(s) {

arr = new int[size];

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

arr[i] = i; // Initialize array with index values

}

}

// Overloading the [] operator for accessing elements

int& operator[](int index) {

if (index < 0 || index >= size) {

cout << "Index out of bounds!" << endl;

// Returning a reference to a static variable just for demonstration purposes

static int dummy;

return dummy;

}

return arr[index];

}

// Destructor to release dynamically allocated memory

~Array() {

delete[] arr;

}

};

int main() {

const int size = 5;

Array myArray(size);

// Accessing elements using [] operator

cout << "Elements of the array:" << endl;

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

cout << "Element at index " << i << ": " << myArray[i] << endl;

}

// Attempting to access an out-of-bounds index

cout << "Attempting to access an out-of-bounds index:" << endl;

cout << "Element at index " << size << ": " << myArray[size] << endl;

return 0;

}

Output:

Elements of the array:

Element at index 0: 0

Element at index 1: 1

Element at index 2: 2

Element at index 3: 3

Element at index 4: 4

Attempting to access an out-of-bounds index:

Element at index 5: Index out of bounds!

0

7.     Write a c++ program to overload the () to call a function with arguments

Code:

#include <iostream>

using namespace std;

class FunctionCaller {

public:

// Overloading the () operator to call a function with arguments

void operator()(int a, int b) {

cout << "Sum of " << a << " and " << b << " is: " << (a + b) << endl;

}

};

int main() {

FunctionCaller adder;

// Calling the function using the overloaded () operator

adder(5, 7);

return 0;

}

Output:

Sum of 5 and 7 is: 12

8.     Write a c++ program to overload the – operator to subtract two variable

Code:

#include <iostream>

using namespace std;

class Subtract {

private:

int value;

public:

Subtract(int v) : value(v) {}

// Overloading the - operator to subtract two variables

Subtract operator-(const Subtract& other) {

return Subtract(value - other.value);

}

// Function to display the value

void display() {

cout << "Value: " << value << endl;

}

};

int main() {

Subtract var1(10);

Subtract var2(5);

cout << "Before subtraction:" << endl;

cout << "Variable 1:" << endl;

var1.display();

cout << "Variable 2:" << endl;

var2.display();

Subtract result = var1 - var2;

cout << "After subtraction:" << endl;

cout << "Result:" << endl;

result.display();

return 0;

}

Output:

Before subtraction:

Variable 1:

Value: 10

Variable 2:

Value: 5

After subtraction:

Result:

Value: 5

9.     write a c++ program to overload a function to add two integer numbers and two floating point number separately

Code:

#include <iostream>

using namespace std;

// Function to add two integers

int add(int a, int b) {

return a + b;

}

// Function to add two floating-point numbers

float add(float a, float b) {

return a + b;

}

int main() {

int int1 = 5, int2 = 7;

float float1 = 3.5, float2 = 2.5;

// Adding two integers

int sum\_int = add(int1, int2);

cout << "Sum of integers: " << sum\_int << endl;

// Adding two floating-point numbers

float sum\_float = add(float1, float2);

cout << "Sum of floating-point numbers: " << sum\_float << endl;

return 0;

}

Output:

Sum of integers: 12

Sum of floating-point numbers: 6

10. Write a c++ program to overload the += operator to add two objects of a user defined class

Code:

#include <iostream>

using namespace std;

class MyNumber {

private:

int number;

public:

MyNumber(int num) : number(num) {}

// Overloading the += operator to add two MyNumber objects

MyNumber& operator+=(const MyNumber& other) {

number += other.number;

return \*this;

}

// Function to display the value

void display() {

cout << "Number: " << number << endl;

}

};

int main() {

MyNumber num1(5);

MyNumber num2(7);

cout << "Before addition:" << endl;

cout << "Num1:" << endl;

num1.display();

cout << "Num2:" << endl;

num2.display();

num1 += num2; // Using the overloaded += operator to add num2 to num1

cout << "After addition:" << endl;

cout << "Num1:" << endl;

num1.display();

return 0;

}

Output:

Before addition:

Num1:

Number: 5

Num2:

Number: 7

After addition:

Num1:

Number: 12