**1. Overload the `++` Operator to Increment a Variable**

#include <iostream>

class Number {

int value;

public:

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

// Overload the ++ operator (prefix version)

Number& operator++() {

++value;

return \*this;

}

void display() const {

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

}

};

int main() {

Number num(10);

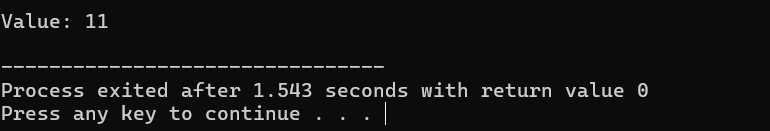
++num;

num.display(); // Output: Value: 11

return 0;

}

**Output**

****

**2. Overload the `+` Operator to Add Two Variables**

#include <iostream>

class Number {

int value;

public:

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

// Overload the + operator

Number operator+(const Number& other) {

return Number(value + other.value);

}

void display() const {

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

}

};

int main() {

Number num1(10), num2(20);

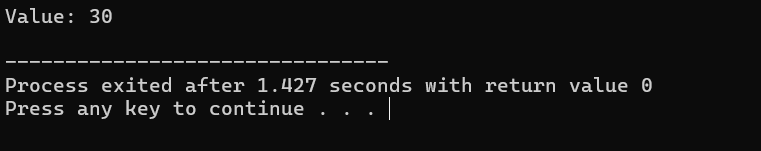
Number sum = num1 + num2;

sum.display(); // Output: Value: 30

return 0;

}

**Output**

****

**3. Overload the `<<` Operator to Print Contents of a User-Defined Class**

#include <iostream>

class Number {

int value;

public:

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

// Overload the << operator

friend std::ostream& operator<<(std::ostream& out, const Number& num) {

out << "Value: " << num.value;

return out;

}

};

int main() {

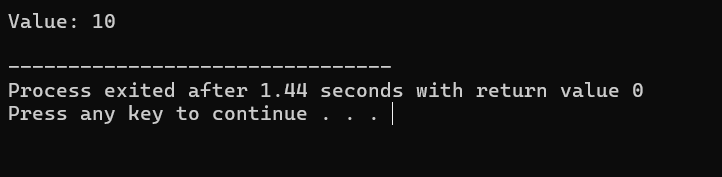
Number num(10);

std::cout << num << std::endl; // Output: Value: 10

return 0;

}

**Output**

****

**4. Overload the `==` Operator to Compare Two Objects of a User-Defined Class**

#include <iostream>

class Number {

int value;

public:

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

// Overload the == operator

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

return value == other.value;

}

};

int main() {

Number num1(10), num2(10);

if (num1 == num2) {

std::cout << "Numbers are equal" << std::endl; // Output: Numbers are equal

} else {

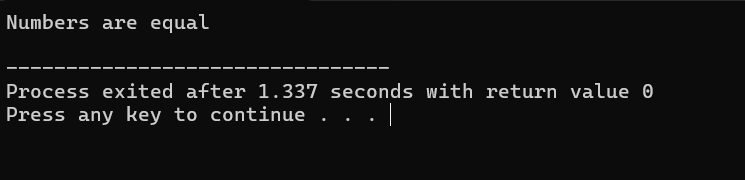
std::cout << "Numbers are not equal" << std::endl;

}

return 0;

}

**Output**

****

**5. Overload the `\*` Operator to Multiply Two Matrices**

#include <iostream>

class Matrix {

int mat[2][2];

public:

Matrix(int a, int b, int c, int d) {

mat[0][0] = a; mat[0][1] = b;

mat[1][0] = c; mat[1][1] = d;

}

// Overload the \* operator

Matrix operator\*(const Matrix& other) {

Matrix result(

mat[0][0] \* other.mat[0][0] + mat[0][1] \* other.mat[1][0],

mat[0][0] \* other.mat[0][1] + mat[0][1] \* other.mat[1][1],

mat[1][0] \* other.mat[0][0] + mat[1][1] \* other.mat[1][0],

mat[1][0] \* other.mat[0][1] + mat[1][1] \* other.mat[1][1]

);

return result;

}

void display() const {

std::cout << mat[0][0] << " " << mat[0][1] << std::endl;

std::cout << mat[1][0] << " " << mat[1][1] << std::endl;

}

};

int main() {

Matrix mat1(1, 2, 3, 4), mat2(2, 0, 1, 2);

Matrix result = mat1 \* mat2;

result.display();

// Output:

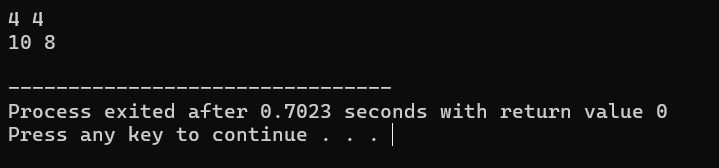
// 4 4

// 10 8

return 0;

}

**Output**

****

**6. Overload the `[]` Operator to Access the Elements in an Array Using Index Values**

#include <iostream>

class Array {

int arr[5];

public:

Array() {

for (int i = 0; i < 5; ++i) arr[i] = i + 1;

}

// Overload the [] operator

int& operator[](int index) {

return arr[index];

}

void display() const {

for (int i = 0; i < 5; ++i)

std::cout << arr[i] << " ";

std::cout << std::endl;

}

};

int main() {

Array array;

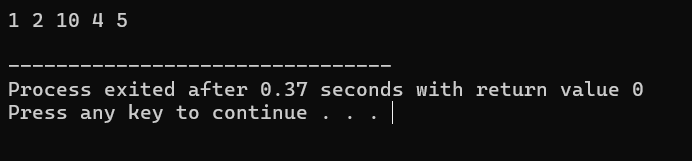
array[2] = 10;

array.display(); // Output: 1 2 10 4 5

return 0;

}

**Output**

****

**7. Overload the `()` Operator to Call a Function with Arguments**

#include <iostream>

class Functor {

public:

// Overload the () operator

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

std::cout << "Sum: " << (a + b) << std::endl;

}

};

int main() {

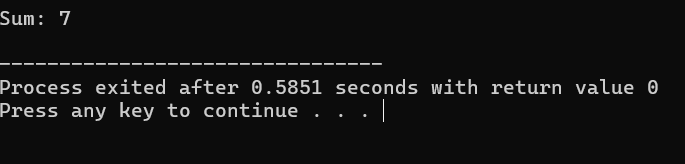
Functor func;

func(3, 4); // Output: Sum: 7

return 0;

}

**Output**

****

**8. Overload the `–` Operator to Subtract Two Variables**

#include <iostream>

class Number {

int value;

public:

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

// Overload the - operator

Number operator-(const Number& other) {

return Number(value - other.value);

}

void display() const {

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

}

};

int main() {

Number num1(20), num2(10);

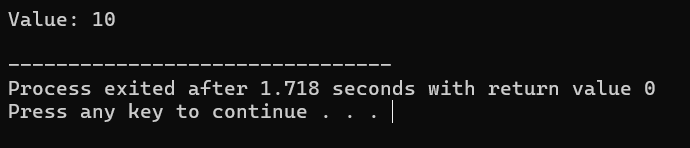
Number diff = num1 - num2;

diff.display(); // Output: Value: 10

return 0;

}

**Output**

****

**9. Overload a Function to Add Two Integer Numbers and Two Floating Point Numbers Separately**

#include <iostream>

class Adder {

public:

// Overload the add function for integers

int add(int a, int b) {

return a + b;

}

// Overload the add function for floats

float add(float a, float b) {

return a + b;

}

};

int main() {

Adder adder;

std::cout << "Sum of integers: " << adder.add(3, 4) << std::endl; // Output: 7

std::cout << "Sum of floats: " << adder.add(3.5f, 4.5f) << std::endl; // Output: 8.0

return 0;

}

**Output**

****

**10. Overload the `+=` Operator to Add Two Objects of a User-Defined Class**

#include <iostream>

class Number {

int value;

public:

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

// Overload the += operator

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

value += other.value;

return \*this;

}

void display() const {

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

}

};

int main() {

Number num1(10), num2(20);

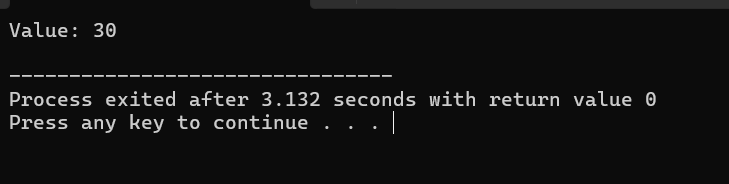
num1 += num2;

num1.display(); // Output: Value: 30

return 0;

}

**Output**

****