**Hard programs**

**21.**

#include <iostream>

#include <cmath>

double calculateArea(double length, double width);

double calculatePerimeter(double length, double width);

double calculateArea(double radius);

double calculatePerimeter(double radius);

double calculateArea(double base, float height);

double calculatePerimeter(double side1, double side2, double side3);

int main() {

char choice;

std::cout << "Choose a shape (r for rectangle, c for circle, t for triangle): ";

std::cin >> choice;

switch (choice) {

case 'r': {

double length, width;

std::cout << "Enter the length of the rectangle: ";

std::cin >> length;

std::cout << "Enter the width of the rectangle: ";

std::cin >> width;

std::cout << "Area of the rectangle: " << calculateArea(length, width) << std::endl;

std::cout << "Perimeter of the rectangle: " << calculatePerimeter(length, width) << std::endl;

break;

}

case 'c': {

double radius;

std::cout << "Enter the radius of the circle: ";

std::cin >> radius;

std::cout << "Area of the circle: " << calculateArea(radius) << std::endl;

std::cout << "Circumference of the circle: " << calculatePerimeter(radius) << std::endl;

break;

}

case 't': {

double side1, side2, side3;

std::cout << "Enter the lengths of the three sides of the triangle: ";

std::cin >> side1 >> side2 >> side3;

std::cout << "Area of the triangle: " << calculateArea(side1, side2) << std::endl;

std::cout << "Perimeter of the triangle: " << calculatePerimeter(side1, side2, side3) << std::endl;

break;

}

default:

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

break;

}

return 0;

}

double calculateArea(double length, double width) {

return length \* width;

}

double calculatePerimeter(double length, double width) {

return 2 \* (length + width);

}

double calculateArea(double radius) {

return M\_PI \* radius \* radius;

}

double calculatePerimeter(double radius) {

return 2 \* M\_PI \* radius;

}

double calculateArea(double base, float height) {

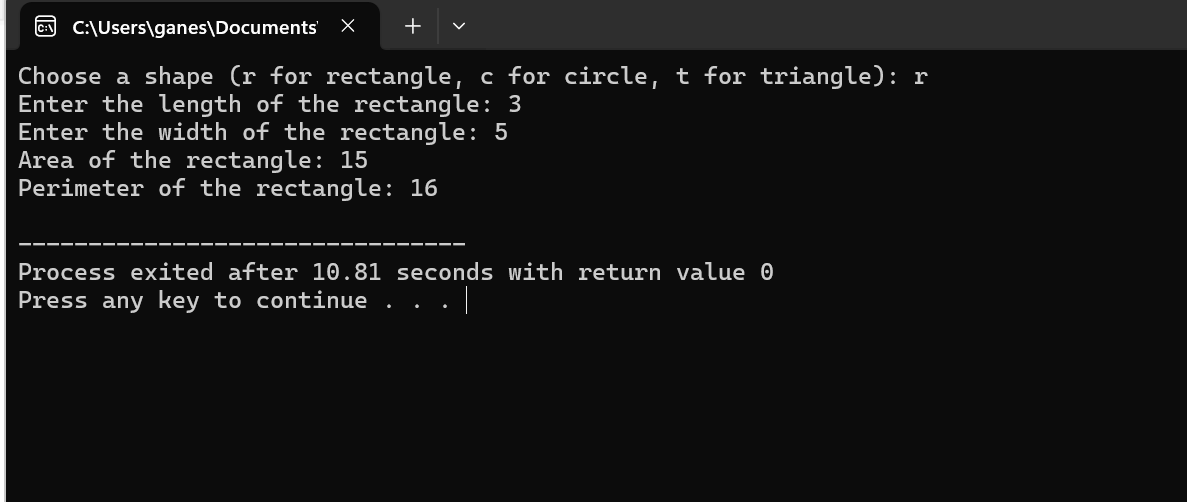
return 0.5 \* base \* height;

}

double calculatePerimeter(double side1, double side2, double side3) {

return side1 + side2 + side3;

}



22.

#include <iostream>

class Complex {

private:

double real;

double imag;

public:

Complex(double r = 0.0, double i = 0.0) : real(r), imag(i) {}

friend Complex add(const Complex& c1, const Complex& c2);

friend Complex subtract(const Complex& c1, const Complex& c2);

friend Complex multiply(const Complex& c1, const Complex& c2);

friend Complex divide(const Complex& c1, const Complex& c2);

void display() const {

std::cout << real << " + " << imag << "i";

}

};

Complex add(const Complex& c1, const Complex& c2) {

return Complex(c1.real + c2.real, c1.imag + c2.imag);

}

Complex subtract(const Complex& c1, const Complex& c2) {

return Complex(c1.real - c2.real, c1.imag - c2.imag);

}

Complex multiply(const Complex& c1, const Complex& c2) {

double real = c1.real \* c2.real - c1.imag \* c2.imag;

double imag = c1.real \* c2.imag + c1.imag \* c2.real;

return Complex(real, imag);

}

Complex divide(const Complex& c1, const Complex& c2) {

double denominator = c2.real \* c2.real + c2.imag \* c2.imag;

double real = (c1.real \* c2.real + c1.imag \* c2.imag) / denominator;

double imag = (c1.imag \* c2.real - c1.real \* c2.imag) / denominator;

return Complex(real, imag);

}

int main() {

Complex c1(2.0, 3.0);

Complex c2(1.0, -1.0);

Complex addition = add(c1, c2);

Complex subtraction = subtract(c1, c2);

Complex multiplication = multiply(c1, c2);

Complex division = divide(c1, c2);

std::cout << "Addition: ";

addition.display();

std::cout << std::endl;

std::cout << "Subtraction: ";

subtraction.display();

std::cout << std::endl;

std::cout << "Multiplication: ";

multiplication.display();

std::cout << std::endl;

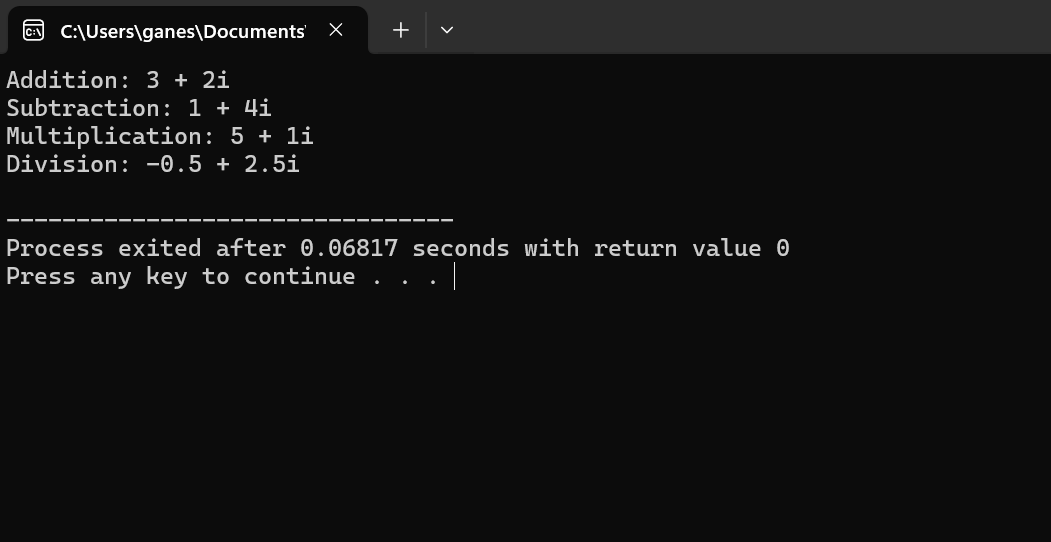
std::cout << "Division: ";

division.display();

std::cout << std::endl;

return 0;

}



23.

#include <iostream>

class Matrix {

private:

int rows;

int cols;

int\*\* data;

public:

Matrix(int numRows, int numCols) : rows(numRows), cols(numCols) {

data = new int\*[rows];

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

data[i] = new int[cols];

}

}

// Destructor to deallocate memory

~Matrix() {

// Deallocate memory for the matrix

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

delete[] data[i];

}

delete[] data;

}

// Overloaded operator+ for matrix addition

Matrix operator+(const Matrix& other) const {

Matrix result(rows, cols);

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

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

result.data[i][j] = data[i][j] + other.data[i][j];

}

}

return result;

}

// Overloaded operator- for matrix subtraction

Matrix operator-(const Matrix& other) const {

Matrix result(rows, cols);

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

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

result.data[i][j] = data[i][j] - other.data[i][j];

}

}

return result;

}

Matrix operator\*(const Matrix& other) const {

Matrix result(rows, other.cols);

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

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

result.data[i][j] = 0;

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

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

}

}

}

return result;

}

void setElement(int row, int col, int value) {

if (row >= 0 && row < rows && col >= 0 && col < cols) {

data[row][col] = value;

} else {

std::cerr << "Invalid row or column index." << std::endl;

}

}

void display() const {

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

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

std::cout << data[i][j] << " ";

}

std::cout << std::endl;

}

}

};

int main() {

// Create matrices

Matrix A(2, 2);

A.setElement(0, 0, 1);

A.setElement(0, 1, 2);

A.setElement(1, 0, 3);

A.setElement(1, 1, 4);

Matrix B(2, 2);

B.setElement(0, 0, 5);

B.setElement(0, 1, 6);

B.setElement(1, 0, 7);

B.setElement(1, 1, 8);

Matrix C = A + B;

Matrix D = A - B;

Matrix E = A \* B;

std::cout << "Matrix A:" << std::endl;

A.display();

std::cout << std::endl;

std::cout << "Matrix B:" << std::endl;

B.display();

std::cout << std::endl;

std::cout << "Matrix A + B:" << std::endl;

C.display();

std::cout << std::endl;

std::cout << "Matrix A - B:" << std::endl;

D.display();

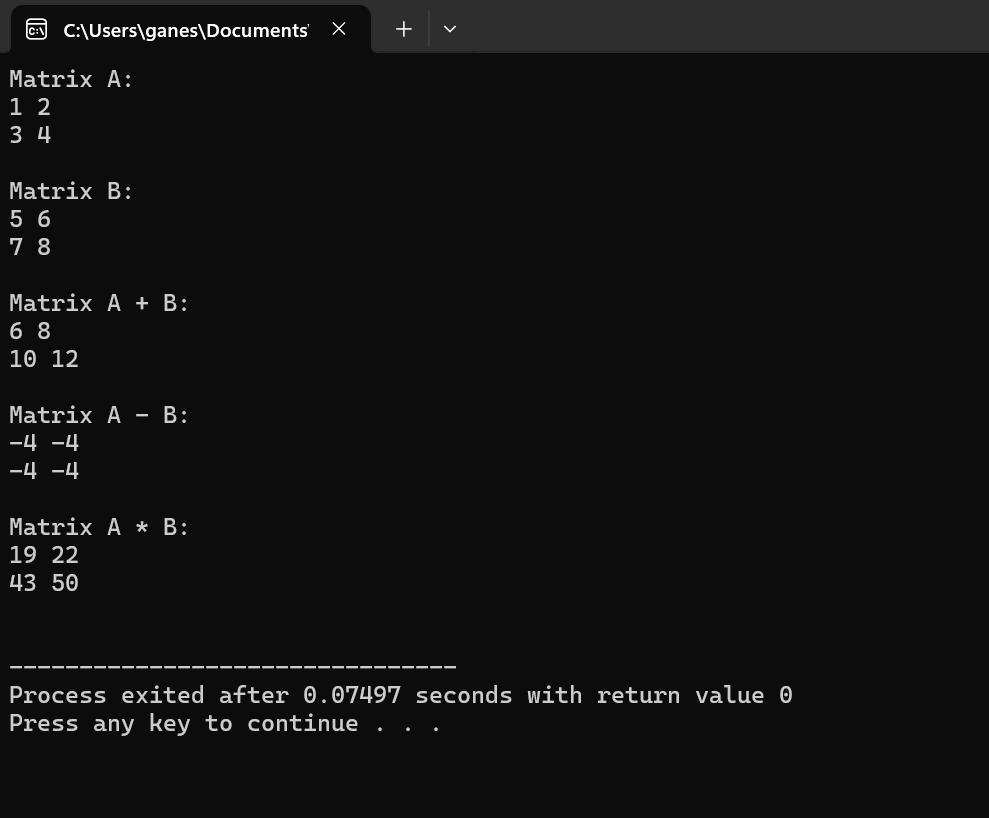
std::cout << std::endl;

std::cout << "Matrix A \* B:" << std::endl;

E.display();

std::cout << std::endl;

return 0;

}

24.

#include <iostream>

#include <vector>

class Employee {

private:

int id;

std::string name;

double salary;

public:

Employee(int empId, const std::string& empName, double empSalary) : id(empId), name(empName), salary(empSalary) {}

static double calculateTotalSalary(const std::vector<Employee>& employees) {

double totalSalary = 0.0;

for (const auto& emp : employees) {

totalSalary += emp.salary;

}

return totalSalary;

}

void setDetails(int empId, const std::string& empName, double empSalary) {

id = empId;

name = empName;

salary = empSalary;

}

void displayDetails() const {

std::cout << "Employee ID: " << id << std::endl;

std::cout << "Employee Name: " << name << std::endl;

std::cout << "Employee Salary: " << salary << std::endl;

}

};

int main() {

std::vector<Employee> employees;

employees.push\_back(Employee(1, "John Doe", 50000.0));

employees.push\_back(Employee(2, "Alice Smith", 60000.0));

employees.push\_back(Employee(3, "Bob Johnson", 70000.0));

for (const auto& emp : employees) {

emp.displayDetails();

std::cout << std::endl;

}

double totalSalary = Employee::calculateTotalSalary(employees);

std::cout << "Total Salary of all Employees: " << totalSalary << std::endl;

return 0;

}

25.

#include <iostream>

#include <vector>

class Vector3D {

private:

double x, y, z;

public:

Vector3D(double x = 0.0, double y = 0.0, double z = 0.0) : x(x), y(y), z(z) {}

Vector3D operator+(const Vector3D& other) const {

return Vector3D(x + other.x, y + other.y, z + other.z);

}

Vector3D operator-(const Vector3D& other) const {

return Vector3D(x - other.x, y - other.y, z - other.z);

}

Vector3D operator\*(double scalar) const {

return Vector3D(x \* scalar, y \* scalar, z \* scalar);

}

Vector3D crossProduct(const Vector3D& other) const {

return Vector3D(y \* other.z - z \* other.y, z \* other.x - x \* other.z, x \* other.y - y \* other.x);

}

void display() const {

std::cout << "(" << x << ", " << y << ", " << z << ")";

}

};

int main() {

std::vector<Vector3D> vecArray1 = {Vector3D(1.0, 2.0, 3.0), Vector3D(4.0, 5.0, 6.0)};

std::vector<Vector3D> vecArray2 = {Vector3D(7.0, 8.0, 9.0), Vector3D(10.0, 11.0, 12.0)};

std::cout << "Vector Addition:" << std::endl;

for (size\_t i = 0; i < vecArray1.size(); ++i) {

(vecArray1[i] + vecArray2[i]).display();

std::cout << std::endl;

}

std::cout << "Vector Subtraction:" << std::endl;

for (size\_t i = 0; i < vecArray1.size(); ++i) {

(vecArray1[i] - vecArray2[i]).display();

std::cout << std::endl;

}

std::cout << "Scalar Multiplication:" << std::endl;

for (size\_t i = 0; i < vecArray1.size(); ++i) {

(vecArray1[i] \* 2.0).display();

std::cout << std::endl;

}

std::cout << "Cross Product:" << std::endl;

for (size\_t i = 0; i < vecArray1.size(); ++i) {

(vecArray1[i].crossProduct(vecArray2[i])).display();

std::cout << std::endl;

}

return 0;

}

26.

#include <iostream>

void calculateFactorial(int n, unsigned long long &result, int start = 1);

int main() {

int num;

std::cout << "Enter a number to calculate its factorial: ";

std::cin >> num;

unsigned long long factorial;

calculateFactorial(num, factorial);

std::cout << "Factorial of " << num << " is: " << factorial << std::endl;

return 0;

}

void calculateFactorial(int n, unsigned long long &result, int start) {

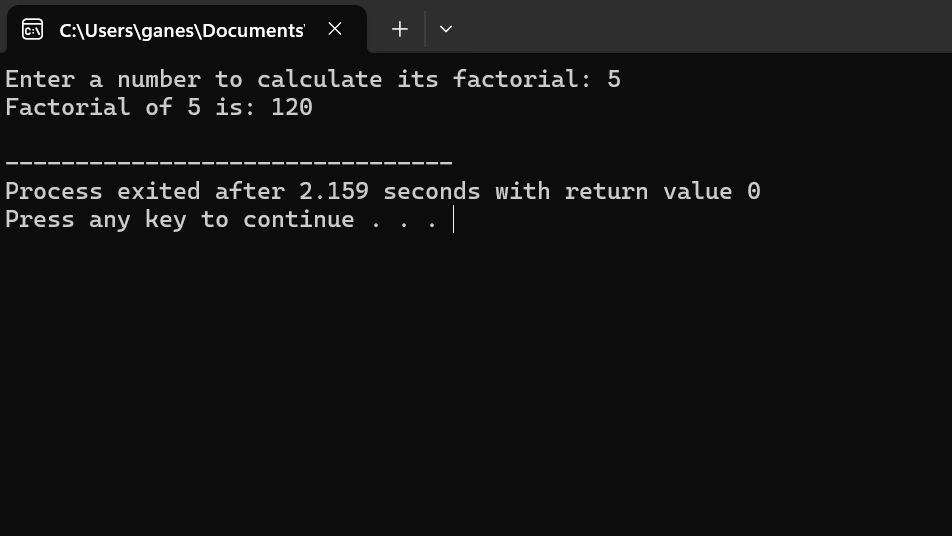
result = 1;

for (int i = start; i <= n; ++i) {

result \*= i;

}

}



27.

#include <iostream>

#include <utility>

std::pair<int&, int&> findMaxMin(int arr[], int size);

int main() {

const int size = 5;

int arr[size];

std::cout << "Enter " << size << " integers:" << std::endl;

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

std::cin >> arr[i];

}

auto& maxMin = findMaxMin(arr, size);

std::cout << "Maximum element: " << maxMin.first << std::endl;

std::cout << "Minimum element: " << maxMin.second << std::endl;

return 0;

}

std::pair<int&, int&> findMaxMin(int arr[], int size) {

int\* maxPtr = &arr[0];

int\* minPtr = &arr[0];

for (int i = 1; i < size; ++i) {

if (arr[i] > \*maxPtr) {

maxPtr = &arr[i];

} else if (arr[i] < \*minPtr) {

minPtr = &arr[i];

}

}

return std::pair<int&, int&>(\*maxPtr, \*minPtr);

}

28.

#include <iostream>

#include <cmath>

class Shape {

public:

virtual double calculateArea() const = 0;

virtual double calculatePerimeter() const = 0;

};

class Rectangle : public Shape {

private:

double length;

double width;

public:

Rectangle(double l, double w) : length(l), width(w) {}

double calculateArea() const override {

return length \* width;

}

double calculatePerimeter() const override {

return 2 \* (length + width);

}

};

class Circle : public Shape {

private:

double radius;

public:

Circle(double r) : radius(r) {}

double calculateArea() const override {

return M\_PI \* radius \* radius;

}

double calculatePerimeter() const override {

return 2 \* M\_PI \* radius;

}

};

class Triangle : public Shape {

private:

double side1, side2, side3;

public:

Triangle(double s1, double s2, double s3) : side1(s1), side2(s2), side3(s3) {}

double calculateArea() const override {

double s = (side1 + side2 + side3) / 2;

return sqrt(s \* (s - side1) \* (s - side2) \* (s - side3));

}

double calculatePerimeter() const override {

return side1 + side2 + side3;

}

};

int main() {

Rectangle rectangle(5.0, 3.0);

Circle circle(4.0);

Triangle triangle(3.0, 4.0, 5.0);

std::cout << "Rectangle:" << std::endl;

std::cout << "Area: " << rectangle.calculateArea() << std::endl;

std::cout << "Perimeter: " << rectangle.calculatePerimeter() << std::endl;

std::cout << std::endl;

std::cout << "Circle:" << std::endl;

std::cout << "Area: " << circle.calculateArea() << std::endl;

std::cout << "Perimeter: " << circle.calculatePerimeter() << std::endl;

std::cout << std::endl;

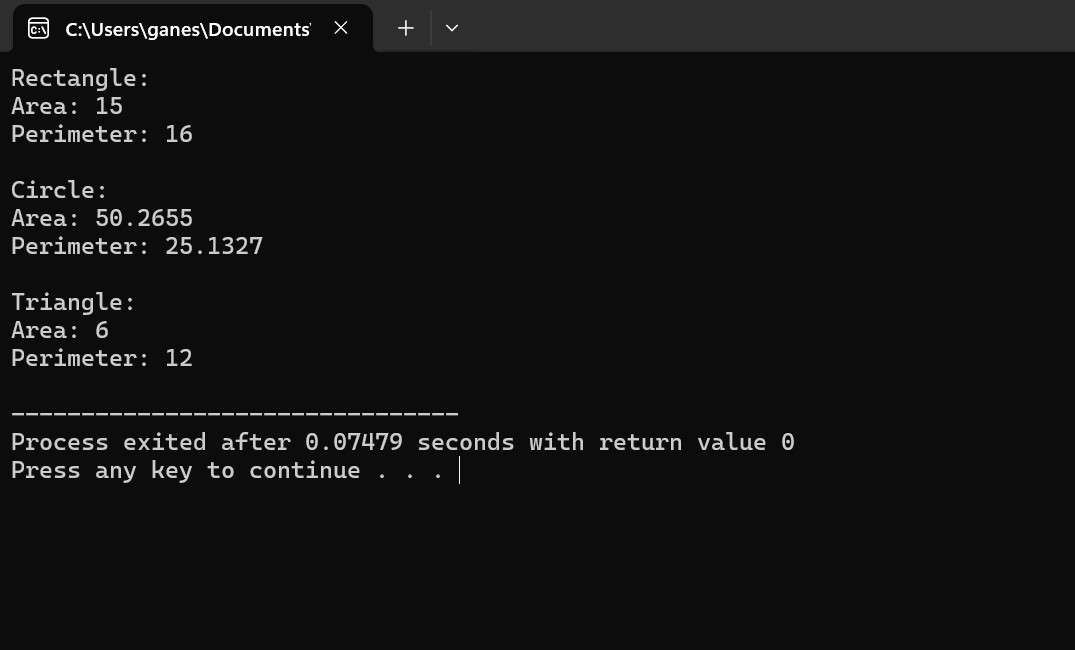
std::cout << "Triangle:" << std::endl;

std::cout << "Area: " << triangle.calculateArea() << std::endl;

std::cout << "Perimeter: " << triangle.calculatePerimeter() << std::endl;

return 0;

}



29.

#include <iostream>

template <typename T>

T average(T arr[], int size);

int main() {

int intArr[] = {1, 2, 3, 4, 5};

int intAvg = average(intArr, 5);

std::cout << "Average of integer array: " << intAvg << std::endl;

double doubleArr[] = {1.1, 2.2, 3.3, 4.4, 5.5};

double doubleAvg = average(doubleArr, 5);

std::cout << "Average of double array: " << doubleAvg << std::endl;

char charArr[] = {'a', 'b', 'c', 'd', 'e'};

char charAvg = average(charArr, 5);

std::cout << "Average of char array: " << charAvg << std::endl;

return 0;

}

template <typename T>

T average(T arr[], int size) {

T sum = 0;

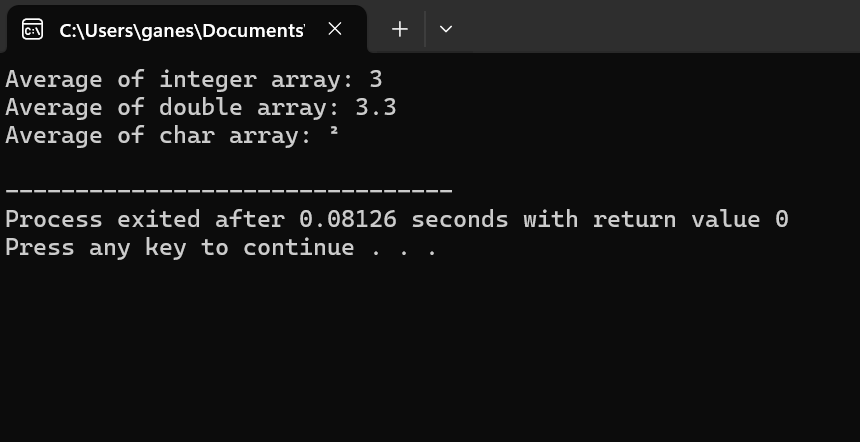
for (int i = 0; i < size; ++i) {

sum += arr[i];

}

return sum / size;

}



30.

#include <iostream>

template<typename T1, typename T2>

class Pair;

template<>

void swapPair(Pair<int, int>& p1, Pair<int, int>& p2) {

std::swap(p1.first, p2.first);

std::swap(p1.second, p2.second);

}

template<>

void swapPair(Pair<double, double>& p1, Pair<double, double>& p2) {

std::swap(p1.first, p2.first);

std::swap(p1.second, p2.second);

}

template<>

void swapPair(Pair<char, char>& p1, Pair<char, char>& p2) {

std::swap(p1.first, p2.first);

std::swap(p1.second, p2.second);

}

template<typename T1, typename T2>

class Pair {

public:

T1 first;

T2 second;

Pair(const T1& f, const T2& s) : first(f), second(s) {}

friend void swapPair<>(Pair& p1, Pair& p2);

};

int main() {

Pair<int, int> intPair1(10, 20);

Pair<int, int> intPair2(30, 40);

std::cout << "Before swapping (intPair1): " << intPair1.first << ", " << intPair1.second << std::endl;

std::cout << "Before swapping (intPair2): " << intPair2.first << ", " << intPair2.second << std::endl;

swapPair(intPair1, intPair2);

std::cout << "After swapping (intPair1): " << intPair1.first << ", " << intPair1.second << std::endl;

std::cout << "After swapping (intPair2): " << intPair2.first << ", " << intPair2.second << std::endl;

Pair<double, double> doublePair1(3.14, 6.28);

Pair<double, double> doublePair2(1.23, 4.56);

std::cout << "Before swapping (doublePair1): " << doublePair1.first << ", " << doublePair1.second << std::endl;

std::cout << "Before swapping (doublePair2): " << doublePair2.first << ", " << doublePair2.second << std::endl;

swapPair(doublePair1, doublePair2);

std::cout << "After swapping (doublePair1): " << doublePair1.first << ", " << doublePair1.second << std::endl;

std::cout << "After swapping (doublePair2): " << doublePair2.first << ", " << doublePair2.second << std::endl;

Pair<char, char> charPair1('a', 'b');

Pair<char, char> charPair2('x', 'y');

std::cout << "Before swapping (charPair1): " << charPair1.first << ", " << charPair1.second << std::endl;

std::cout << "Before swapping (charPair2): " << charPair2.first << ", " << charPair2.second << std::endl;

swapPair(charPair1, charPair2);

std::cout << "After swapping (charPair1): " << charPair1.first << ", " << charPair1.second << std::endl;

std::cout << "After swapping (charPair2): " << charPair2.first << ", " << charPair2.second << std::endl;

return 0;

}