**14/03/2024**

**DSA0187:OBJECT ORIENTED PROGRAMMING WITH C++**

N.Ravi Teja Reddy (192210667)

1.Write a simple C++ program with a main function that prints "Hello, World!" to the console

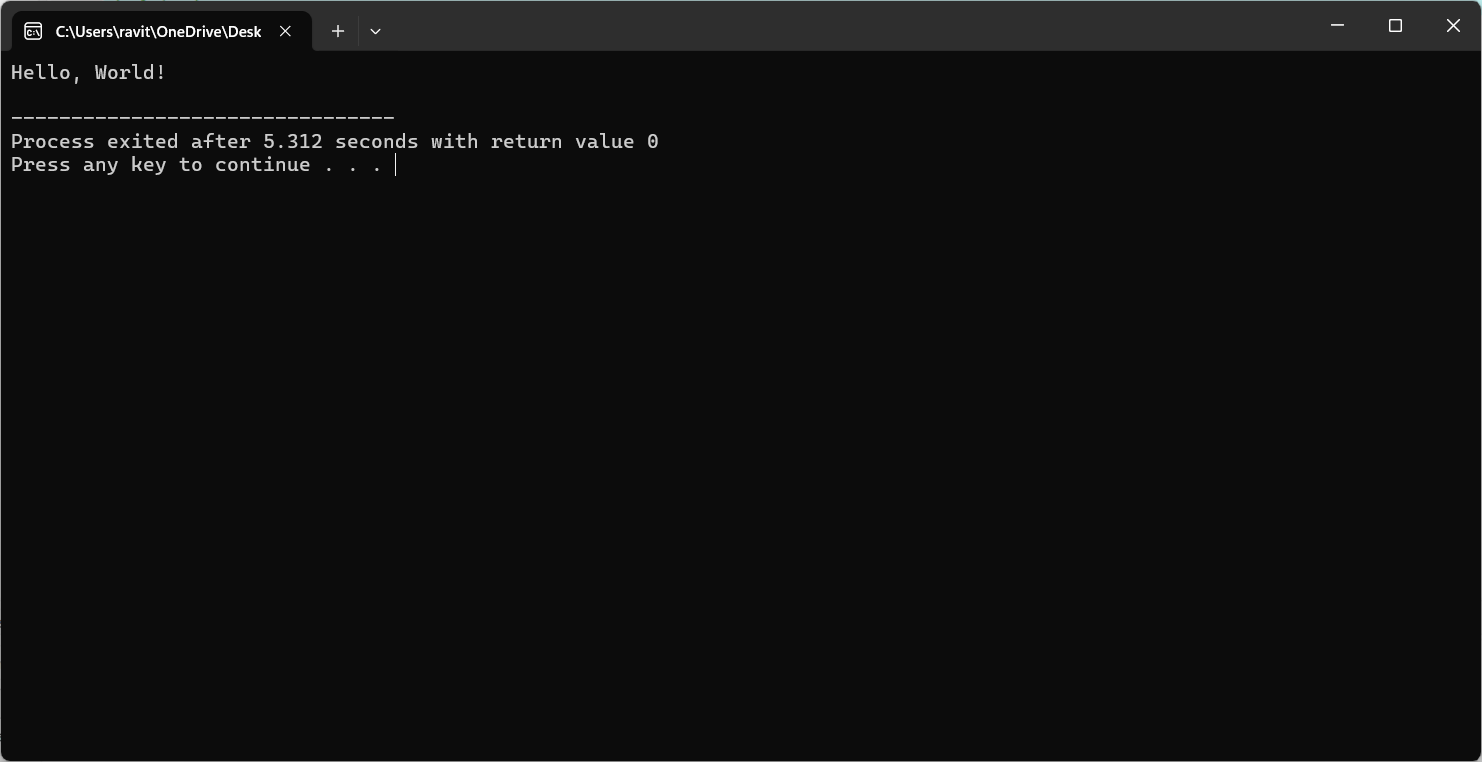
#include <iostream>

int main() {

std::cout << "Hello, World!" << std::endl;

return 0;

}



2. Create a simple C++ program with a function prototype for a function named addNumbers that takes two integers as parameters and returns their sum. Implement the function below the main function and use it to add two numbers.

#include <iostream>

// Function prototype

int addNumbers(int a, int b);

int main() {

int num1 = 5;

int num2 = 10;

// Call the addNumbers function and print the result

int sum = addNumbers(num1, num2);

std::cout << "Sum: " << sum << std::endl;

return 0;

}

// Function definition

int addNumbers(int a, int b) {

return a + b;

}

A screenshot of a computer

Description automatically generated

3. Write a simple C++ program that demonstrates call by reference by swapping the values of two variables.

#include <iostream>

// Function prototype

void swapValues(int &a, int &b);

int main() {

int num1 = 5;

int num2 = 10;

std::cout << "Before swapping: num1 = " << num1 << ", num2 = " << num2 << std::endl;

// Call the swapValues function to swap the values of num1 and num2

swapValues(num1, num2);

std::cout << "After swapping: num1 = " << num1 << ", num2 = " << num2 << std::endl;

return 0;

}

// Function definition

void swapValues(int &a, int &b) {

int temp = a;

a = b;

b = temp;

}

A screenshot of a computer

Description automatically generated

4. Develop a simple C++ program with a function that returns a reference to an integer variable and modifies its value. Use this function to update the value of a variable in the main function.

#include <iostream>

// Function that returns a reference to an integer variable

int& updateValue(int &value) {

// Modify the value

value \*= 2;

// Return a reference to the modified value

return value;

}

int main() {

int num = 5;

std::cout << "Before update: num = " << num << std::endl;

// Call the updateValue function to modify the value of num

updateValue(num);

std::cout << "After update: num = " << num << std::endl;

return 0;

}

A screenshot of a computer

Description automatically generated

5. Implement an inline function named square that calculates the square of a number. Use this function to square a user-input integer. simple c program without comments

#include <iostream>

inline int square(int x) { return x \* x; }

int main() {

int num;

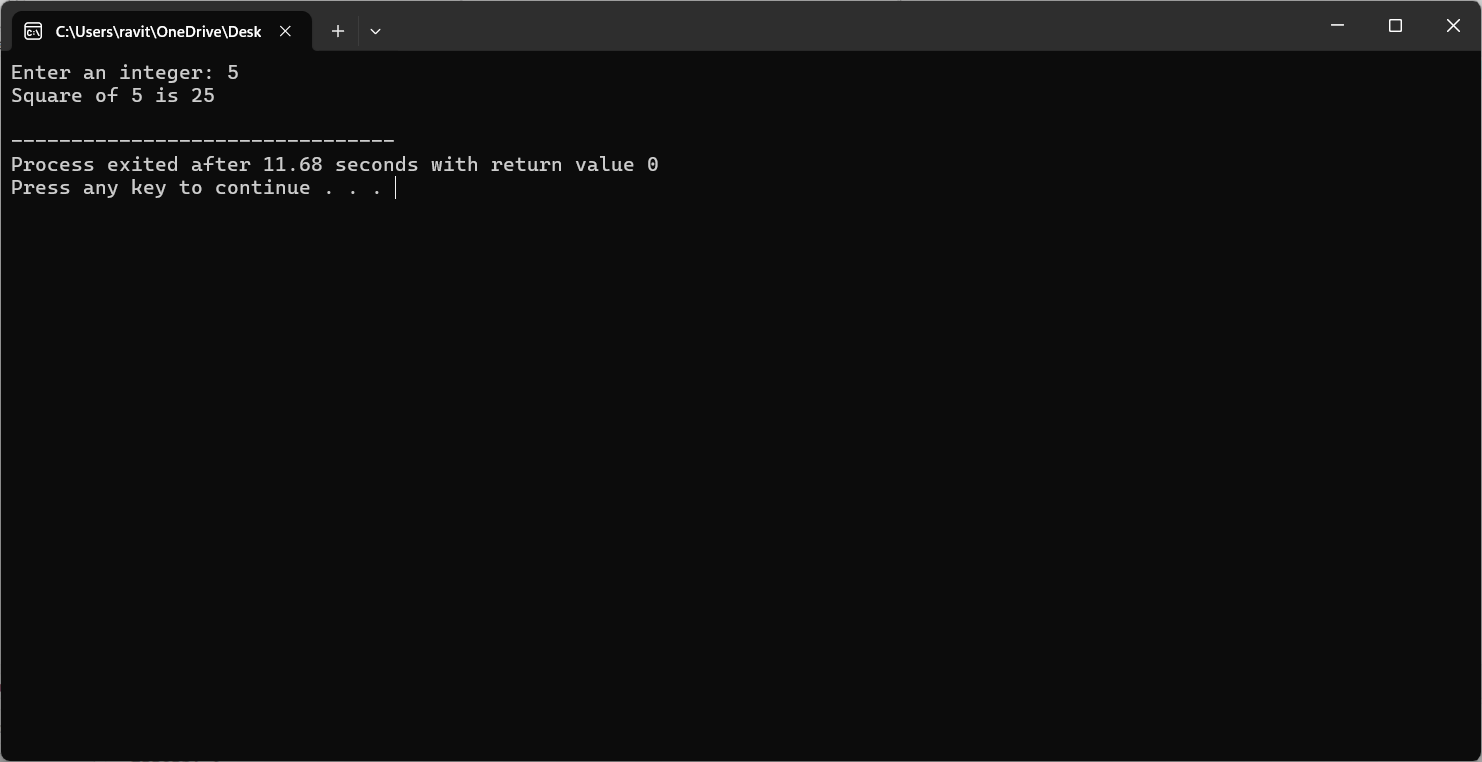
std::cout << "Enter an integer: ";

std::cin >> num;

std::cout << "Square of " << num << " is " << square(num) << std::endl;

return 0;

}



6. Write a simple C++ program that defines a function named printMessage with a default argument "Hello". Call this function without passing any argument

#include <iostream>

void printMessage(std::string message = "Hello") {

std::cout << message << std::endl;

}

int main() {

printMessage(); // Will print "Hello"

return 0;

}

A screenshot of a computer

Description automatically generated

7. Create a simple C++ program with two overloaded functions named area - one to calculate the area of a rectangle (length \* width) and another to calculate the area of a circle (π \* radius^2). Use function overloading to determine which function to call based on the number of arguments.

#include <iostream>

const double PI = 3.14159;

// Function to calculate the area of a rectangle

double area(double length, double width) {

return length \* width;

}

// Function to calculate the area of a circle

double area(double radius) {

return PI \* radius \* radius;

}

int main() {

double length = 5.0;

double width = 3.0;

double radius = 2.0;

// Calculate and print the area of a rectangle

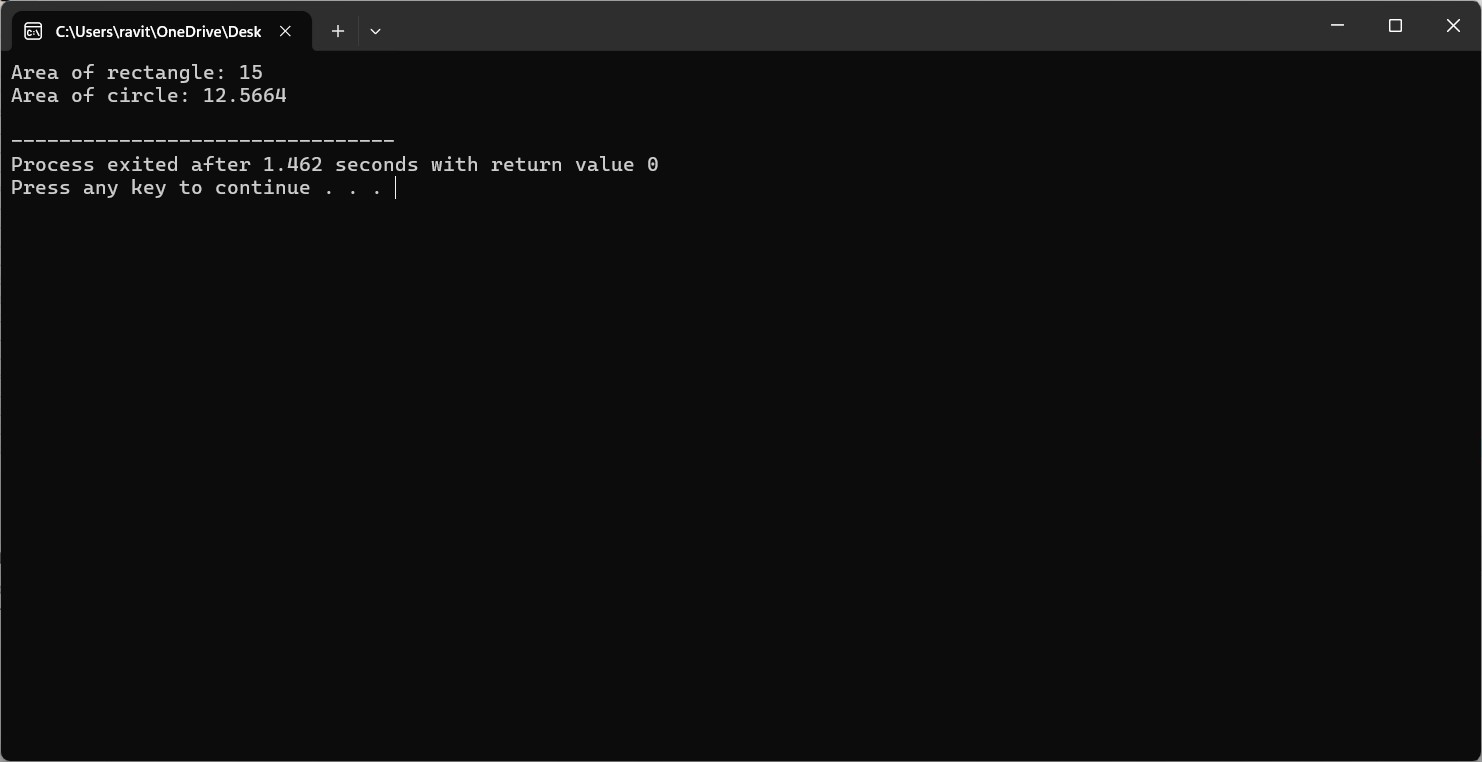
std::cout << "Area of rectangle: " << area(length, width) << std::endl;

// Calculate and print the area of a circle

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

return 0;

}



8. Define a simple C++ class named Rectangle with private member variables length and width. Implement public member functions to set and get the length and width of the rectangle.

#include <iostream>

class Rectangle {

private:

double length;

double width;

public:

void setLength(double len) {

length = len;

}

double getLength() {

return length;

}

void setWidth(double wid) {

width = wid;

}

double getWidth() {

return width;

}

};

int main() {

Rectangle rect;

rect.setLength(5.0);

rect.setWidth(3.0);

std::cout << "Length: " << rect.getLength() << std::endl;

std::cout << "Width: " << rect.getWidth() << std::endl;

return 0;

}

A computer screen with white text

Description automatically generated

9. Extend the Rectangle class to include an array of integers named coordinates to store the (x, y) coordinates of the rectangle. Implement member functions to set and get the coordinates. simple c program without comments

#include <iostream>

class Rectangle {

private:

double length;

double width;

int coordinates[2];

public:

void setLength(double len) { length = len; }

double getLength() { return length; }

void setWidth(double wid) { width = wid; }

double getWidth() { return width; }

void setCoordinates(int x, int y) {

coordinates[0] = x;

coordinates[1] = y;

}

int\* getCoordinates() { return coordinates; }

};

int main() {

Rectangle rect;

rect.setLength(5.0);

rect.setWidth(3.0);

rect.setCoordinates(1, 2);

std::cout << "Length: " << rect.getLength() << std::endl;

std::cout << "Width: " << rect.getWidth() << std::endl;

int\* coords = rect.getCoordinates();

std::cout << "Coordinates: (" << coords[0] << ", " << coords[1] << ")" << std::endl;

return 0;

}

A computer screen with white text

Description automatically generated

10. Add a static member function named countRectangles to the Rectangle class that keeps track of the total number of rectangle objects created. Display the count in the main function. simple c program with comments

#include <iostream>

class Rectangle {

private:

double length;

double width;

int coordinates[2];

static int count; // Static member to keep track of total rectangles

public:

// Constructor to initialize length, width, coordinates

Rectangle(double len, double wid, int x, int y) {

length = len;

width = wid;

coordinates[0] = x;

coordinates[1] = y;

count++; // Increment count when a new object is created

}

// Destructor to decrement count when object is destroyed

~Rectangle() {

count--;

}

void setLength(double len) { length = len; }

double getLength() { return length; }

void setWidth(double wid) { width = wid; }

double getWidth() { return width; }

void setCoordinates(int x, int y) {

coordinates[0] = x;

coordinates[1] = y;

}

int\* getCoordinates() { return coordinates; }

// Static member function to get count of total rectangles

static int getCountRectangles() {

return count;

}

};

int Rectangle::count = 0; // Initialize static member count

int main() {

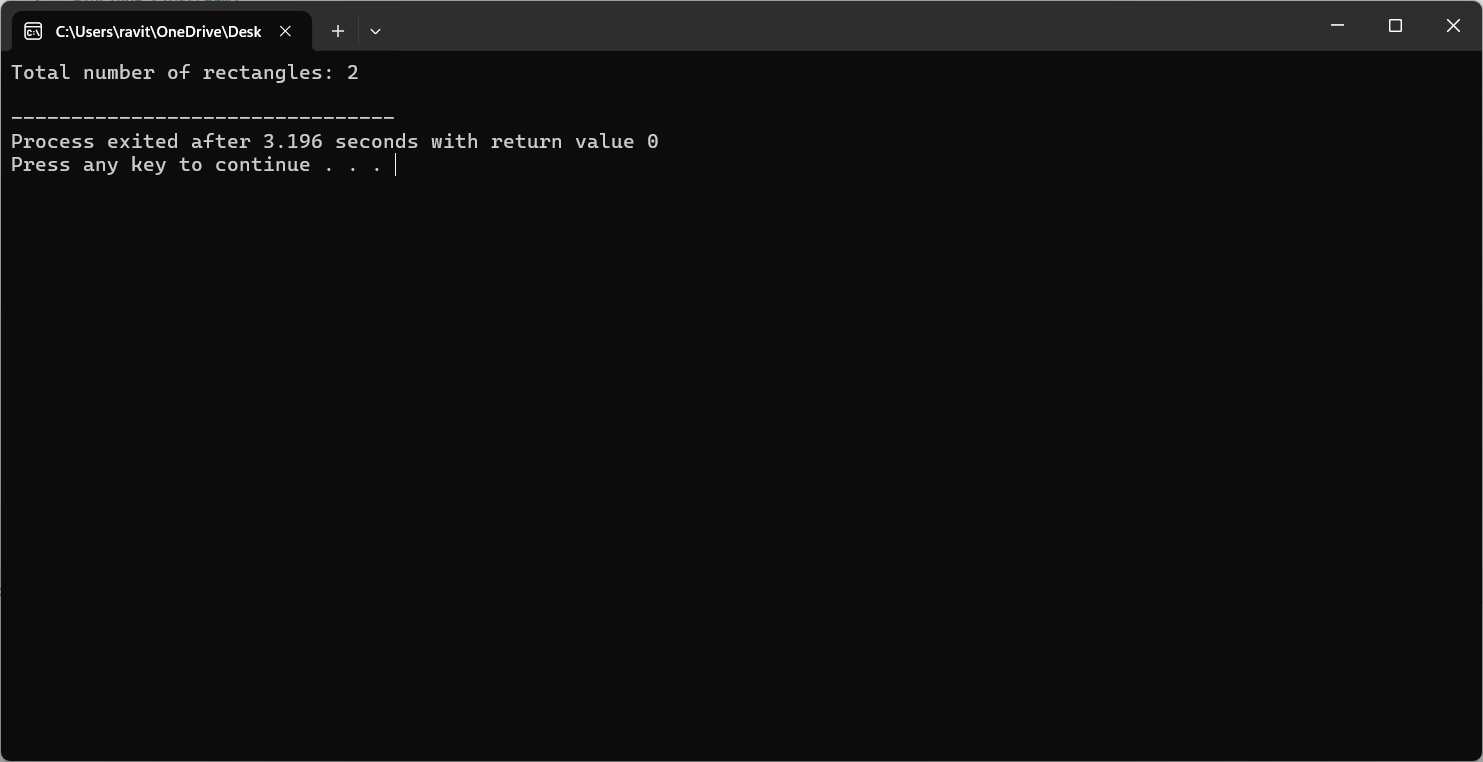
Rectangle rect1(5.0, 3.0, 1, 2);

Rectangle rect2(4.0, 2.0, 3, 4);

std::cout << "Total number of rectangles: " << Rectangle::getCountRectangles() << std::endl;

return 0;

}



11.Write a simple C++ program that calculates the volume of a cube, rectangle, or cylinder based on user choice. Use function prototypes and function overloading to define separate functions for each shape's volume calculation.

#include <iostream>

#include <cmath>

using namespace std;

// Function prototypes

double volume(double side);

double volume(double length, double width, double height);

double volume(double radius, double height);

int main() {

int choice;

cout << "Choose a shape to calculate its volume:" << endl;

cout << "1. Cube" << endl;

cout << "2. Rectangle" << endl;

cout << "3. Cylinder" << endl;

cin >> choice;

switch (choice) {

case 1: {

double side;

cout << "Enter the side length of the cube: ";

cin >> side;

cout << "Volume of the cube: " << volume(side) << endl;

break;

}

case 2: {

double length, width, height;

cout << "Enter the length, width, and height of the rectangle: ";

cin >> length >> width >> height;

cout << "Volume of the rectangle: " << volume(length, width, height) << endl;

break;

}

case 3: {

double radius, height;

cout << "Enter the radius and height of the cylinder: ";

cin >> radius >> height;

cout << "Volume of the cylinder: " << volume(radius, height) << endl;

break;

}

default:

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

}

return 0;

}

// Function to calculate the volume of a cube

double volume(double side) {

return pow(side, 3);

}

// Function to calculate the volume of a rectangle

double volume(double length, double width, double height) {

return length \* width \* height;

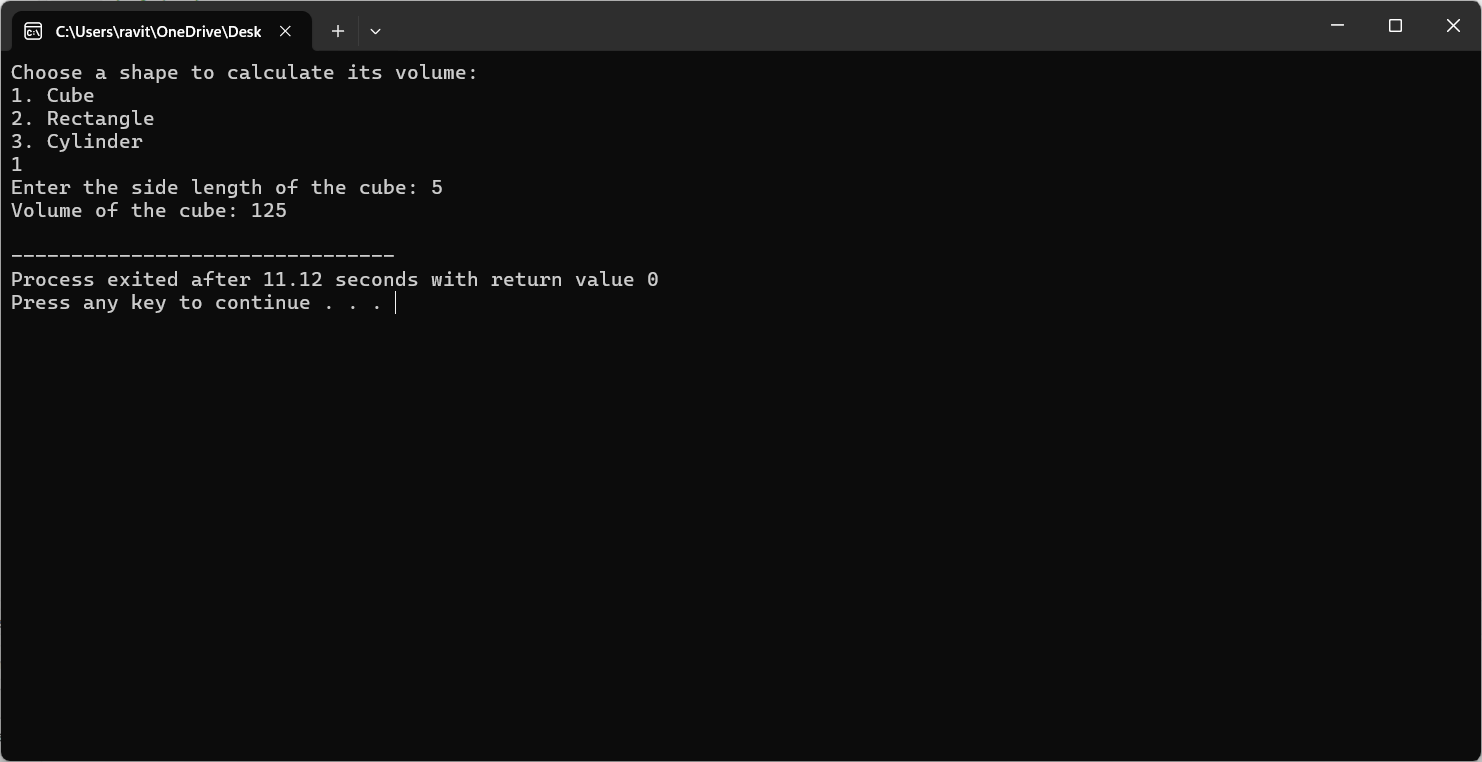
}

// Function to calculate the volume of a cylinder

double volume(double radius, double height) {

return M\_PI \* pow(radius, 2) \* height;

}



12. Define a simple class named Student with private member variables name, id, and an array grades to store the student's grades in three subjects. Implement member functions to set and get the student details and grades.

#include <iostream>

#include <string>

using namespace std;

class Student {

private:

string name;

int id;

int grades[3];

public:

// Constructor

Student(string studentName, int studentId) {

name = studentName;

id = studentId;

for (int i = 0; i < 3; i++) {

grades[i] = 0; // Initialize grades to 0

}

}

// Set student's name

void setName(string studentName) {

name = studentName;

}

// Get student's name

string getName() {

return name;

}

// Set student's ID

void setId(int studentId) {

id = studentId;

}

// Get student's ID

int getId() {

return id;

}

// Set student's grade for a subject (index 0-2)

void setGrade(int subjectIndex, int grade) {

if (subjectIndex >= 0 && subjectIndex < 3) {

grades[subjectIndex] = grade;

} else {

cout << "Invalid subject index." << endl;

}

}

// Get student's grade for a subject (index 0-2)

int getGrade(int subjectIndex) {

if (subjectIndex >= 0 && subjectIndex < 3) {

return grades[subjectIndex];

} else {

cout << "Invalid subject index." << endl;

return -1;

}

}

};

int main() {

Student student("Alice", 12345);

cout << "Student Name: " << student.getName() << endl;

cout << "Student ID: " << student.getId() << endl;

student.setGrade(0, 95); // Set grade for first subject

student.setGrade(1, 85); // Set grade for second subject

student.setGrade(2, 90); // Set grade for third subject

cout << "Grades:" << endl;

cout << "Subject 1: " << student.getGrade(0) << endl;

cout << "Subject 2: " << student.getGrade(1) << endl;

cout << "Subject 3: " << student.getGrade(2) << endl;

return 0;

}

A screenshot of a computer

Description automatically generated

13. Create a simple C++ program that defines an inline function named calculateArea to calculate the area of a rectangle. Provide default arguments for length and width parameters. Use this function to calculate the area of a rectangle with user-input dimensions.

#include <iostream>

using namespace std;

// Inline function to calculate the area of a rectangle

inline double calculateArea(double length = 1.0, double width = 1.0) {

return length \* width;

}

int main() {

double length, width;

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

cin >> length;

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

cin >> width;

double area = calculateArea(length, width);

cout << "Area of the rectangle: " << area << endl;

return 0;

}

A screenshot of a computer

Description automatically generated

14. Define a simple class named Employee with private member variables name and salary. Implement a static member function to calculate the average salary of an array of Employee objects.

#include <iostream>

#include <string>

using namespace std;

class Employee {

private:

string name;

double salary;

public:

Employee(string employeeName, double employeeSalary) {

name = employeeName;

salary = employeeSalary;

}

static double averageSalary(Employee\* employees, int count) {

double totalSalary = 0;

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

totalSalary += employees[i].salary;

}

return totalSalary / count;

}

};

int main() {

const int numEmployees = 3;

Employee employees[numEmployees] = {

Employee("Alice", 50000),

Employee("Bob", 60000),

Employee("Charlie", 70000)

};

double avgSalary = Employee::averageSalary(employees, numEmployees);

cout << "Average salary of employees: " << avgSalary << endl;

return 0;

}

A screenshot of a computer

Description automatically generated

15. Define two classes, Complex and Matrix. Make Matrix a friend of Complex. Implement a function in Matrix class that multiplies a Complex number with a constant and returns the result by reference. simple c program without comments

#include <iostream>

using namespace std;

class Complex;

class Matrix {

public:

Complex multiply(const Complex& c, double constant);

};

class Complex {

private:

double real;

double imag;

public:

Complex(double r = 0, double i = 0) : real(r), imag(i) {}

friend class Matrix;

};

Complex Matrix::multiply(const Complex& c, double constant) {

return Complex(c.real \* constant, c.imag \* constant);

}

int main() {

Complex c(2, 3);

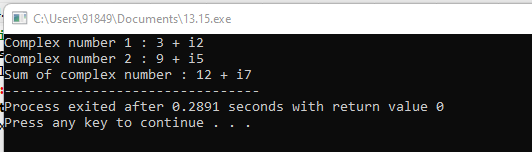
Matrix m;

Complex result = m.multiply(c, 2);

cout << "Result: " << result.real << " + " << result.imag << "i" << endl;

return 0;

}



16. Write a simple C++ program that contains overloaded functions named swap. Define one version that swaps two integers and another version that swaps two arrays of integers. Implement these functions using call by reference

#include <iostream>

using namespace std;

// Function to swap two integers

void swap(int& a, int& b) {

int temp = a;

a = b;

b = temp;

}

// Function to swap two arrays of integers

void swap(int arr1[], int arr2[], int size) {

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

int temp = arr1[i];

arr1[i] = arr2[i];

arr2[i] = temp;

}

}

int main() {

int x = 5, y = 10;

cout << "Before swap: x = " << x << ", y = " << y << endl;

swap(x, y);

cout << "After swap: x = " << x << ", y = " << y << endl;

int arr1[] = {1, 2, 3};

int arr2[] = {4, 5, 6};

int size = sizeof(arr1) / sizeof(arr1[0]);

cout << "Before swap:" << endl;

cout << "arr1: ";

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

cout << arr1[i] << " ";

}

cout << endl;

cout << "arr2: ";

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

cout << arr2[i] << " ";

}

cout << endl;

swap(arr1, arr2, size);

cout << "After swap:" << endl;

cout << "arr1: ";

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

cout << arr1[i] << " ";

}

cout << endl;

cout << "arr2: ";

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

cout << arr2[i] << " ";

}

cout << endl;

return 0;

}

A screenshot of a computer

Description automatically generated

17. Define a simple class named Circle with private member variables radius and area. Implement member functions to set the radius, calculate the area, and display the circle's details

#include <iostream>

#include <cmath>

using namespace std;

class Circle {

private:

double radius;

double area;

public:

void setRadius(double r) {

radius = r;

}

void calculateArea() {

area = M\_PI \* pow(radius, 2);

}

void displayDetails() {

cout << "Radius: " << radius << endl;

cout << "Area: " << area << endl;

}

};

int main() {

Circle circle;

double radius;

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

cin >> radius;

circle.setRadius(radius);

circle.calculateArea();

circle.displayDetails();

return 0;

}

A screenshot of a computer

Description automatically generated

18. Create a simple C++ program that defines a class named Car with private member variables model, year, and price. Implement an array of Car objects and provide default values for year and price.

#include <iostream>

#include <string>

using namespace std;

class Car {

private:

string model;

int year;

double price;

public:

Car(string carModel, int carYear = 2022, double carPrice = 0.0) : model(carModel), year(carYear), price(carPrice) {}

void setModel(string carModel) {

model = carModel;

}

void setYear(int carYear) {

year = carYear;

}

void setPrice(double carPrice) {

price = carPrice;

}

string getModel() const {

return model;

}

int getYear() const {

return year;

}

double getPrice() const {

return price;

}

};

int main() {

const int numCars = 3;

Car cars[numCars] = {

Car("Toyota"),

Car("Honda", 2020),

Car("Ford", 2018, 25000.0)

};

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

cout << "Car " << i + 1 << " Details:" << endl;

cout << "Model: " << cars[i].getModel() << endl;

cout << "Year: " << cars[i].getYear() << endl;

cout << "Price: " << cars[i].getPrice() << endl;

cout << endl;

}

return 0;

}

A screen shot of a computer

Description automatically generated

19. Define a simple C++ function named largestElement that returns a reference to the largest element in an array of integers. Use this function to find the largest element in a user-input array.

#include <iostream>

using namespace std;

int& largestElement(int arr[], int size) {

int maxIndex = 0;

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

if (arr[i] > arr[maxIndex]) {

maxIndex = i;

}

}

return arr[maxIndex];

}

int main() {

int size;

cout << "Enter the size of the array: ";

cin >> size;

int arr[size];

cout << "Enter " << size << " elements: ";

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

cin >> arr[i];

}

int& largest = largestElement(arr, size);

cout << "The largest element in the array is: " << largest << endl;

return 0;

}

A screenshot of a computer

Description automatically generated

20. Write a simple C++ program with function prototypes for calculating the area and perimeter of a rectangle. Implement these functions with default arguments for length and width. Prompt the user to enter the length and width to calculate the area and perimeter

#include <iostream>

using namespace std;

// Function prototype for calculating the area of a rectangle

double calculateArea(double length = 1.0, double width = 1.0);

// Function prototype for calculating the perimeter of a rectangle

double calculatePerimeter(double length = 1.0, double width = 1.0);

int main() {

double length, width;

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

cin >> length;

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

cin >> width;

double area = calculateArea(length, width);

double perimeter = calculatePerimeter(length, width);

cout << "Area of the rectangle: " << area << endl;

cout << "Perimeter of the rectangle: " << perimeter << endl;

return 0;

}

// Function definition for calculating the area of a rectangle

double calculateArea(double length, double width) {

return length \* width;

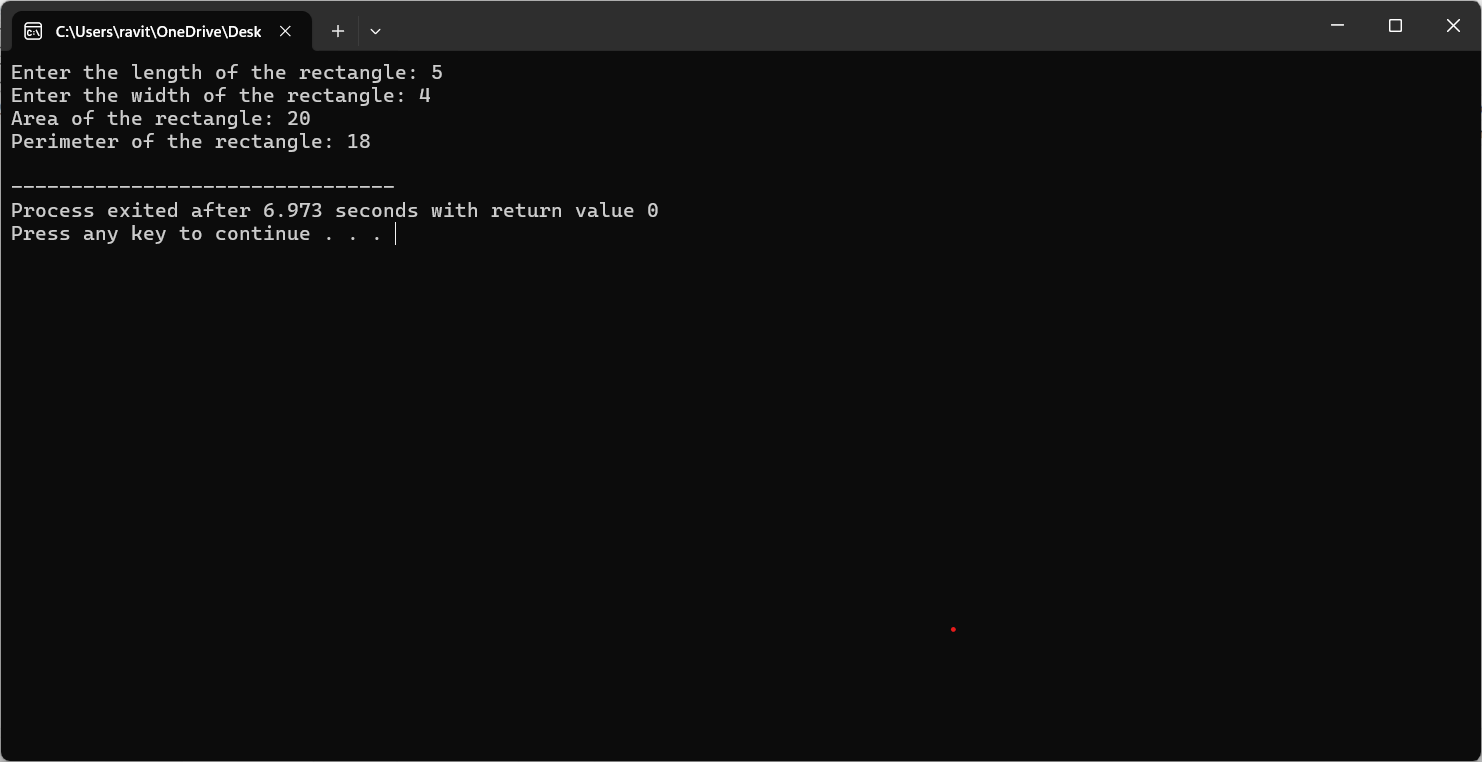
}

// Function definition for calculating the perimeter of a rectangle

double calculatePerimeter(double length, double width) {

return 2 \* (length + width);

}



21. Write a C++ program with function prototypes for calculating the area and perimeter of a rectangle, circle, and triangle. Implement overloaded functions to handle each shape's calculations.

#include<iostream>

using namespace std;

int area(int);

int area(int,int);

float area(float);

float area(float,float);

int main()

{

int s,l,b;

float r,bs,ht;

cout<<"Enter side of a square:";

cin>>s;

cout<<"Enter length and breadth of rectangle:";

cin>>l>>b;

cout<<"Enter radius of circle:";

cin>>r;

cout<<"Enter base and height of triangle:";

cin>>bs>>ht;

cout<<"Area of square is"<<area(s);

cout<<"\nArea of rectangle is "<<area(l,b);

cout<<"\nArea of circle is "<<area(r);

cout<<"\nArea of triangle is "<<area(bs,ht);

}

int area(int s)

{

return(s\*s);

}

int area(int l,int b)

{

return(l\*b);

}

float area(float r)

{

return(3.14\*r\*r);

}

float area(float bs,float ht)

{

return((bs\*ht)/2);

}

A computer screen with white text

Description automatically generated

22. Define a class named **Complex** representing complex numbers with real and imaginary parts. Implement friend functions for addition, subtraction, multiplication, and division of complex numbers.

#include <iostream>

class Complex {

private:

double real;

double imaginary;

public:

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

friend Complex operator+(const Complex& c1, const Complex& c2);

friend Complex operator-(const Complex& c1, const Complex& c2);

friend Complex operator\*(const Complex& c1, const Complex& c2);

friend Complex operator/(const Complex& c1, const Complex& c2);

friend std::ostream& operator<<(std::ostream& os, const Complex& c);

};

Complex operator+(const Complex& c1, const Complex& c2) {

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

}

Complex operator-(const Complex& c1, const Complex& c2) {

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

}

Complex operator\*(const Complex& c1, const Complex& c2) {

double realPart = c1.real \* c2.real - c1.imaginary \* c2.imaginary;

double imaginaryPart = c1.real \* c2.imaginary + c1.imaginary \* c2.real;

return Complex(realPart, imaginaryPart);

}

Complex operator/(const Complex& c1, const Complex& c2) {

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

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

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

return Complex(realPart, imaginaryPart);

}

std::ostream& operator<<(std::ostream& os, const Complex& c) {

os << c.real;

if (c.imaginary >= 0)

os << " + " << c.imaginary << "i";

else

os << " - " << -c.imaginary << "i";

return os;

}

int main() {

Complex a(3, 2);

Complex b(1, -4);

std::cout << "a: " << a << std::endl;

std::cout << "b: " << b << std::endl;

std::cout << "a + b: " << a + b << std::endl;

std::cout << "a - b: " << a - b << std::endl;

std::cout << "a \* b: " << a \* b << std::endl;

std::cout << "a / b: " << a / b << std::endl;

return 0;

}

OUTPUT:

A screenshot of a computer

Description automatically generated

1. 24. Define a class named **Employee** with private member variables **id**, **name**, and **salary**. Implement a static member function to calculate the total salary of all employees. Include advanced member functions for employee details manipulation.

#include <iostream>

#include <vector>

class Employee {

private:

int id;

std::string name;

double salary;

static double totalSalary;

static std::vector<Employee> employees;

public:

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

totalSalary += salary;

employees.push\_back(\*this);

}

static double getTotalSalary() {

return totalSalary;

}

static int getTotalEmployees() {

return employees.size();

}

double getSalary() const {

return salary;

}

static void resetTotalSalary() {

totalSalary = 0.0;

}

void updateDetails(const std::string& empName, double empSalary) {

totalSalary -= salary;

name = empName;

salary = empSalary;

totalSalary += salary;

}

void display() const {

std::cout << "ID: " << id << ", Name: " << name << ", Salary: $" << salary << std::endl;

}

};

double Employee::totalSalary = 0.0;

std::vector<Employee> Employee::employees;

int main() {

Employee emp1(1, "John Doe", 50000.0);

Employee emp2(2, "Jane Smith", 60000.0);

std::cout << "Employee Details:" << std::endl;

emp1.display();

emp2.display();

emp1.updateDetails("John Wick", 55000.0);

std::cout << "\nEmployee Details after update:" << std::endl;

emp1.display();

std::cout << "\nTotal Salary of all employees: $" << Employee::getTotalSalary() << std::endl;

std::cout << "Total Number of Employees: " << Employee::getTotalEmployees() << std::endl;

return 0;

}

OUTPUT:

A screenshot of a computer program

Description automatically generated

25. Develop a C++ program that defines a class named Vector3D representing a 3D vector. Implement operator overloading for addition, subtraction, scalar multiplication, and cross product operations on arrays of Vector3D objects.

#include <iostream>

#include <cmath>

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& vec) const {

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

}

Vector3D operator-(const Vector3D& vec) const {

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

}

Vector3D operator\*(double scalar) const {

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

}

Vector3D operator^(const Vector3D& vec) const {

double \_x = y \* vec.z - z \* vec.y;

double \_y = z \* vec.x - x \* vec.z;

double \_z = x \* vec.y - y \* vec.x;

return Vector3D(\_x, \_y, \_z);

}

double dot(const Vector3D& vec) const {

return x \* vec.x + y \* vec.y + z \* vec.z;

}

double norm() const {

return sqrt(x \* x + y \* y + z \* z);

}

void display() const {

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

}

};

int main() {

Vector3D v1(1.0, 2.0, 3.0);

Vector3D v2(2.0, 3.0, 4.0);

Vector3D sum = v1 + v2;

std::cout << "Sum: ";

sum.display();

std::cout << std::endl;

Vector3D diff = v1 - v2;

std::cout << "Difference: ";

diff.display();

std::cout << std::endl;

Vector3D scaled = v1 \* 2.5;

std::cout << "Scalar multiplication: ";

scaled.display();

std::cout << std::endl;

Vector3D cross = v1 ^ v2;

std::cout << "Cross product: ";

cross.display();

std::cout << std::endl;

double dotProduct = v1.dot(v2);

std::cout << "Dot product: " << dotProduct << std::endl;

double normV1 = v1.norm();

std::cout << "Norm of v1: " << normV1 << std::endl;

return 0;

}

A screenshot of a computer

Description automatically generated

26. Write a C++ program that contains a function to calculate the factorial of a number using call by reference. Use default arguments to provide a starting value for the factorial calculation.

#include<iostream>

using namespace std;

int factorial(int n);

int main() {

int n;

cout << "Enter a positive integer: ";

cin >> n;

cout << "Factorial of " << n << " = " << factorial(n);

return 0;

}

int factorial(int n) {

if(n > 1)

return n \* factorial(n - 1);

else

return 1;

}

OUTPUT:

A screenshot of a computer

Description automatically generated

27. Define a C++ function named **findMaxMin** that finds both the maximum and minimum elements in an array of integers. Return a reference to a **pair** object containing the maximum and minimum values.

#include <utility>

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

if (size == 0) {

static std::pair<int, int> emptyPair(0, 0);

return emptyPair;

}

int maxVal = arr[0];

int minVal = arr[0];

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

if (arr[i] > maxVal) {

maxVal = arr[i];

} else if (arr[i] < minVal) {

minVal = arr[i];

}

}

static std::pair<int, int> result(maxVal, minVal);

return result;

}

#include <iostream>

int main() {

int arr[] = {4, 8, 2, 10, 5};

int size = sizeof(arr) / sizeof(arr[0]);

std::pair<int, int>& result = findMaxMin(arr, size);

std::cout << "Maximum value: " << result.first << std::endl;

std::cout << "Minimum value: " << result.second << std::endl;

return 0;

}

OUTPUT:

A screenshot of a computer

Description automatically generated

28.Create a simple C++ program that defines a base class named Shape with virtual member functions for calculating area and perimeter. Implement derived classes for specific shapes like Rectangle, Circle, and Triangle, overriding the virtual functions accordingly

#include <iostream>

#include <cmath>

class Shape {

public:

virtual double area() const = 0;

virtual double perimeter() const = 0;

};

class Rectangle : public Shape {

private:

double width, height;

public:

Rectangle(double w, double h) : width(w), height(h) {}

double area() const override {

return width \* height;

}

double perimeter() const override {

return 2 \* (width + height);

}

};

class Circle : public Shape {

private:

double radius;

public:

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

double area() const override {

return M\_PI \* radius \* radius;

}

double perimeter() 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 area() const override {

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

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

}

double perimeter() const override {

return side1 + side2 + side3;

}

};

int main() {

Rectangle rect(3, 4);

Circle circle(5);

Triangle triangle(3, 4, 5);

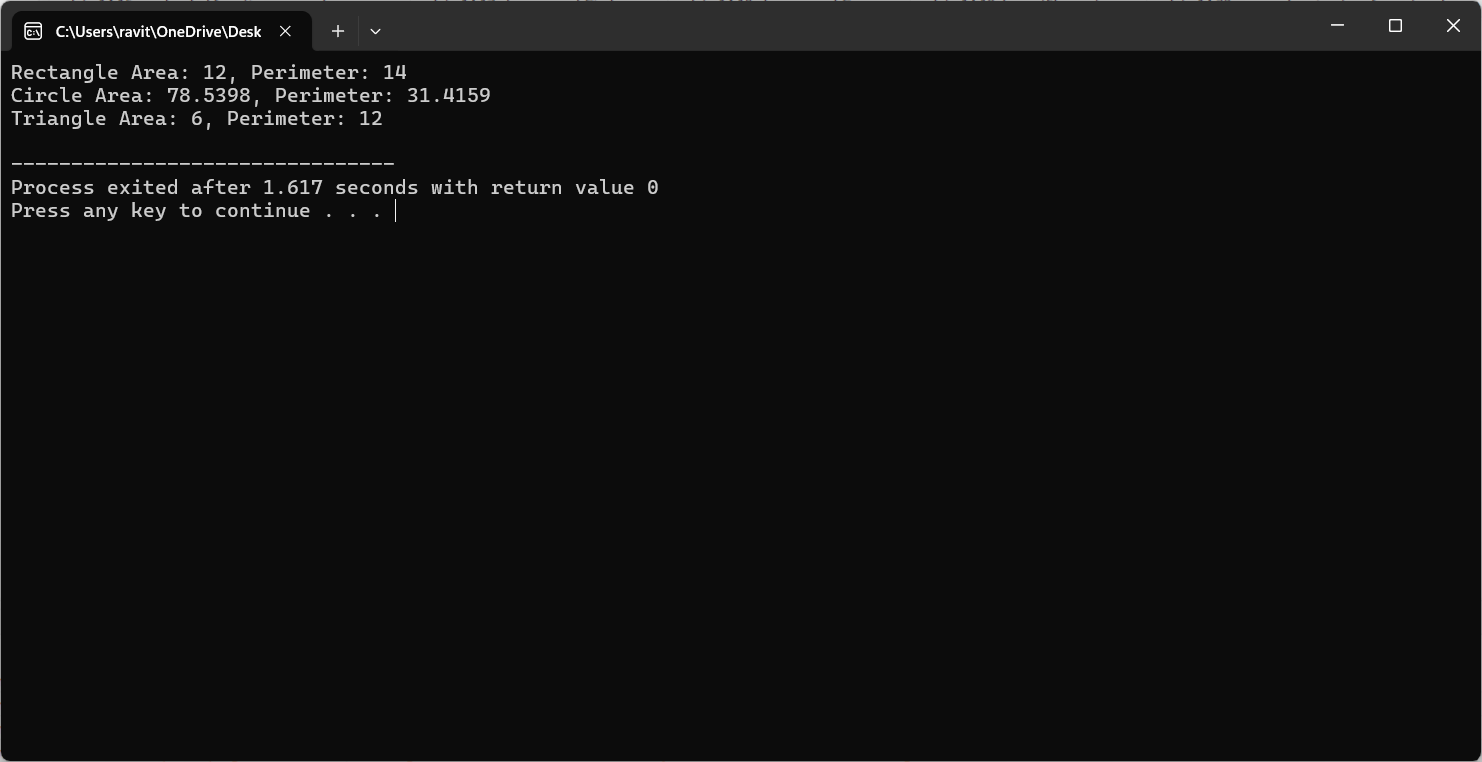
std::cout << "Rectangle Area: " << rect.area() << ", Perimeter: " << rect.perimeter() << std::endl;

std::cout << "Circle Area: " << circle.area() << ", Perimeter: " << circle.perimeter() << std::endl;

std::cout << "Triangle Area: " << triangle.area() << ", Perimeter: " << triangle.perimeter() << std::endl;

return 0;

}



29.Write a simple C++ program that defines a template function named average to calculate the average of elements in an array of any data type. Use function prototypes to declare the template function and test it with different data types

#include <iostream>

// Function prototype for the template function

template <typename T>

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

int main() {

// Test with integers

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

int intAvg = average(intArr, 5);

std::cout << "Average of integers: " << intAvg << std::endl;

// Test with doubles

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

double doubleAvg = average(doubleArr, 5);

std::cout << "Average of doubles: " << doubleAvg << std::endl;

// Test with floats

float floatArr[] = {1.1f, 2.2f, 3.3f, 4.4f, 5.5f};

float floatAvg = average(floatArr, 5);

std::cout << "Average of floats: " << floatAvg << std::endl;

return 0;

}

// Template function definition for calculating average

template <typename T>

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

T sum = 0;

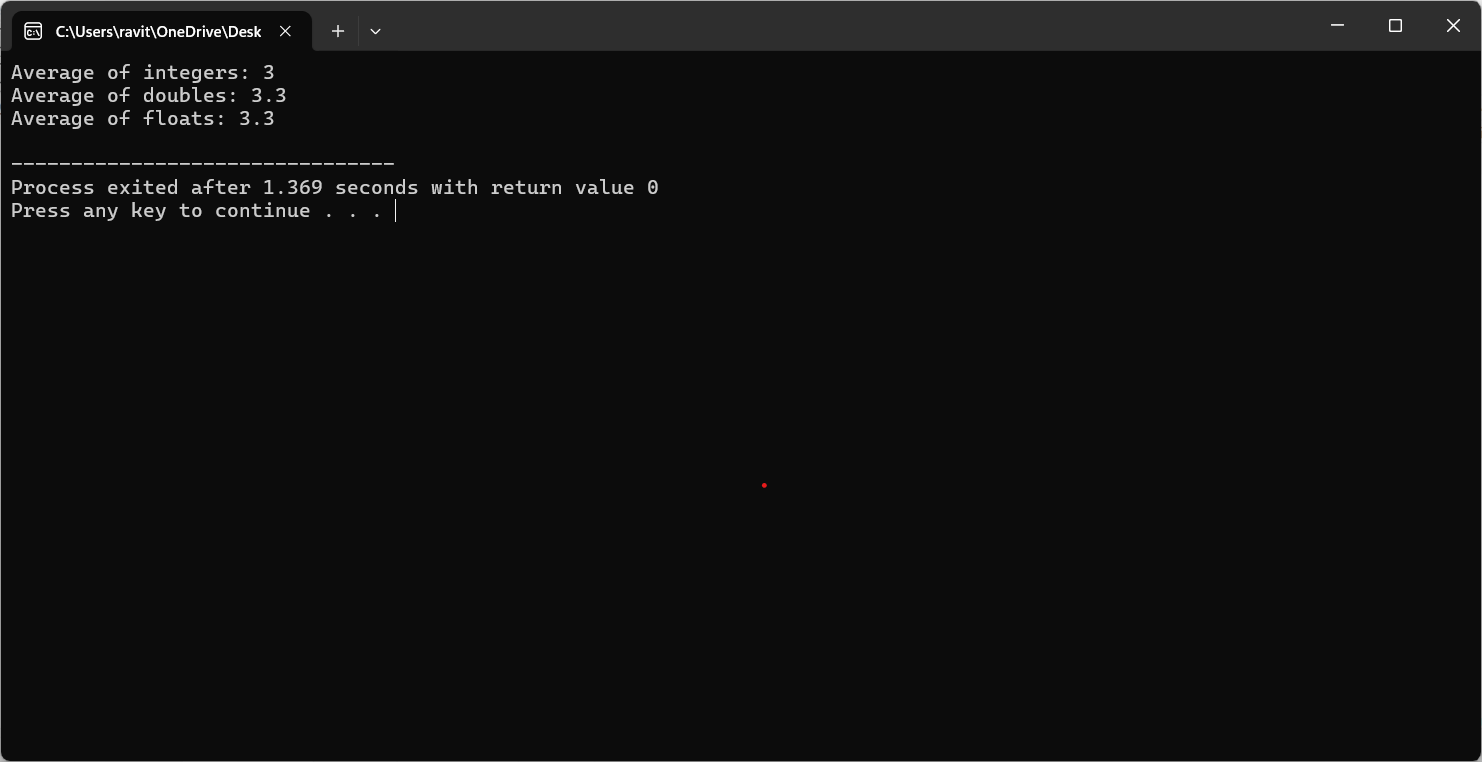
for (int i = 0; i < size; ++i) {

sum += arr[i];

}

return sum / size;

}



30.Define a class named Pair representing a pair of values of any data type. Implement a friend function to swap the values of two Pair objects. Use template specialization to handle swapping for specific data types.

#include <iostream>

#include <utility>

template <typename T1, typename T2>

class Pair {

public:

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

void display() const {

std::cout << "(" << first << ", " << second << ")" << std::endl;

}

template <typename U1, typename U2>

friend void swap(Pair<U1, U2>& pair1, Pair<U1, U2>& pair2);

private:

T1 first;

T2 second;

};

template <typename T1, typename T2>

void swap(Pair<T1, T2>& pair1, Pair<T1, T2>& pair2) {

using std::swap;

swap(pair1.first, pair2.first);

swap(pair1.second, pair2.second);

}

// Template specialization for swapping Pair<int, int>

template <>

void swap(Pair<int, int>& pair1, Pair<int, int>& pair2) {

std::swap(pair1.first, pair2.first);

std::swap(pair1.second, pair2.second);

}

int main() {

Pair<int, double> pair1(1, 2.3);

Pair<int, double> pair2(4, 5.6);

pair1.display();

pair2.display();

swap(pair1, pair2);

pair1.display();

pair2.display();

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

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

intPair1.display();

intPair2.display();

swap(intPair1, intPair2);

intPair1.display();

intPair2.display();

return 0;

}

A screenshot of a computer

Description automatically generated