PROGRAMS

DATE:14/03/24

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

#include<iostream>

using namespace std;

int main()

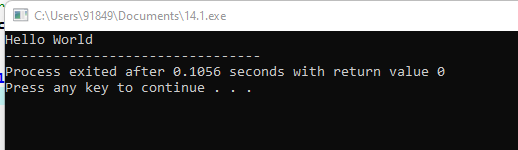
{

cout<<"Hello World";

return 0;

}

OUTPUT:



2. Create a 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>

using namespace std;

int add(int a, int b) {

return a + b;

}

int main() {

int num1, num2, sum;

cout << "Enter two numbers: ";

cin >> num1 >> num2;

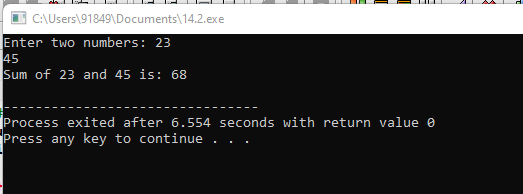
sum = add(num1, num2);

cout << "Sum of " << num1 << " and " << num2 << " is: " << sum << endl;

return 0;

}

OUTPUT:



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

#include<iostream>

using namespace std;

void swap (int &num1, int &num2)

{

int temp;

temp=num1;

num1=num2;

num2=temp;

}

int main()

{

int a,b;

cout<<"enter a and b";

cin>>a>>b;

cout<<"\n Before swapping"<<"\n A = "<<a<<"\n B = "<<b<<endl;

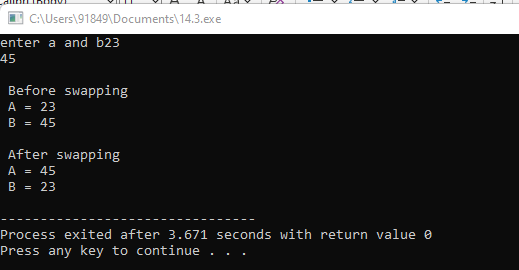
swap(a,b);

cout<<"\n After swapping"<<"\n A = "<<a<<"\n B = "<<b<<endl;

return 0;

}

OUTPUT:



4. Develop a 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>

using namespace std;

void f(int & x)

{

x--;

}

int main()

{

int a ;

cout<<"enter a";

cin>>a;

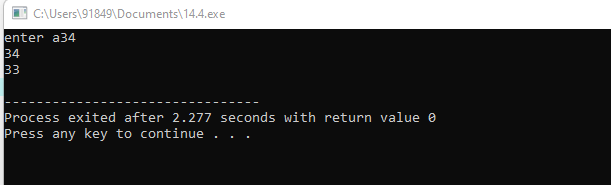
cout << a << endl;

f(a);

cout << a << endl;

}

OUTPUT:



5. . Implement an inline function named **square** that calculates the square of a number. Use this function to square a user-input integer.

#include<iostream>

using namespace std;

class square

{

private:

int n,r;

float n1,r1;

public:

void input();

void calc();

void display();

};

inline void square::input()

{

cout<<"Enter an integer :: ";

cin>>n;

cout<<"\nEnter a float no. :: ";

cin>>n1;

}

inline void square::calc()

{

r=n\*n;

r1=n1\*n1;

}

inline void square::display()

{

cout<<"\nSquare of integer [ "<<n<<" ] = "<<r<<"\n";

cout<<"\nSquare of float [ "<<n1<<" ] = "<<r1<<"\n";

}

int main ()

{

square s;

s.input();

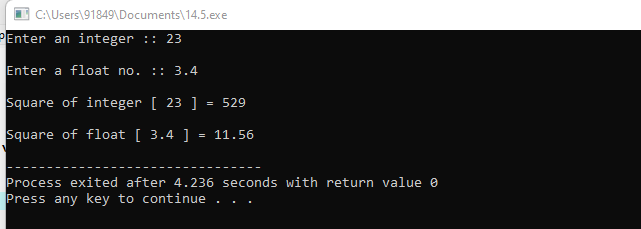
s.calc();

s.display();

return 0;

}

OUTPUT:



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

#include <iostream>

#include <string>

using namespace std;

void printMessage(const string& message = "Hello, World!") {

cout << message << endl;

}

int main() {

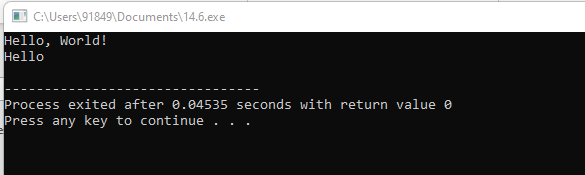
printMessage();

printMessage("Hello");

return 0;

}

OUTPUT:



7. Create a 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>

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);

}

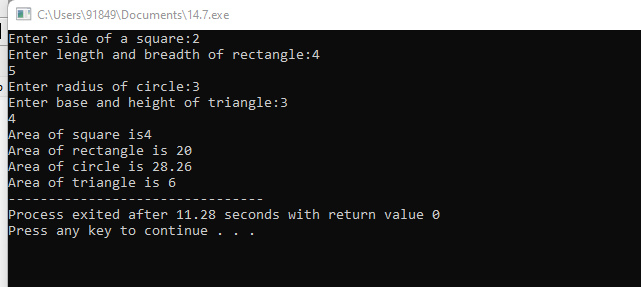
float area(float bs,float ht)

{

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

}

OUTPUT:



8. Define a 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.

PROGRAM:

#include <iostream>

class Rectangle {

private:

double length;

double width;

public:

Rectangle(double len, double wid): length(len), width(wid) {}

double calculateArea() {

return length \* width;

}

double calculatePerimeter() {

return 2 \* (length + width);

}

};

int main() {

double length, width;

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

std::cin >> length;

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

std::cin >> width;

Rectangle rectangle(length, width);

double area = rectangle.calculateArea();

std::cout << "\nArea: " << area << std::endl;

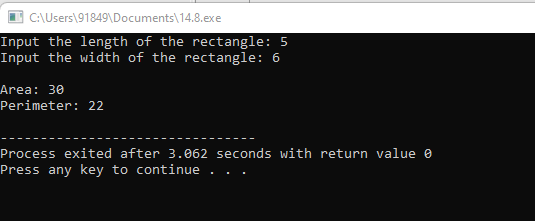
double perimeter = rectangle.calculatePerimeter();

std::cout << "Perimeter: " << perimeter << std::endl;

return 0;

}

OUTPUT:



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.

#include <iostream>

using namespace std;

void printRect(int X[], int Y[], int n)

{

int Xmax = \*max\_element(X, X + n);

int Xmin = \*min\_element(X, X + n);

int Ymax = \*max\_element(Y, Y + n);

int Ymin = \*min\_element(Y, Y + n);

cout << "{" << Xmin << ", " << Ymin << "}" << endl;

cout << "{" << Xmin << ", " << Ymax << "}" << endl;

cout << "{" << Xmax << ", " << Ymax << "}" << endl;

cout << "{" << Xmax << ", " << Ymin << "}" << endl;

}

int main()

{

int X[] = { 4, 3, 6, 1, -1, 12 };

int Y[] = { 4, 1, 10, 3, 7, -1 };

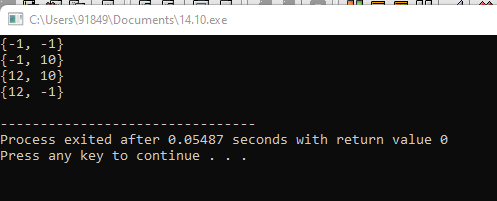
int n = sizeof(X) / sizeof(X[0]);

printRect(X, Y, n);

return 0;

}

OUTPUT:



11. Write a 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>

using namespace std;

float vol(int,int);

float vol(float);

int vol(int);

int main()

{

int r,h,a;

float r1;

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

cin>>r>>h;

cout<<"Enter side of cube:";

cin>>a;

cout<<"Enter radius of sphere: ";

cin>>r1;

cout<<"Volume of cylinder is "<<vol(r,h);

cout<<"\nVolume of cube is "<<vol(a);

cout<<"\nVolume of sphere is "<<vol(r1);

return 0;

}

float vol(int r,int h)

{

return(3.14\*r\*r\*h);

}

float vol(float r1)

{

return((4\*3.14\*r1\*r1\*r1)/3);

}

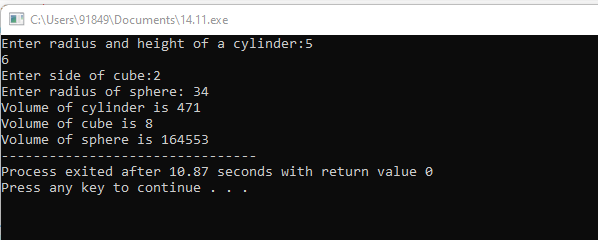
int vol(int a)

{

return(a\*a\*a);

}

OUTPUT:



12. Define a 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>

class Student {

private:

std::string name;

int id;

int grades[3];

public:

Student() : name(""), id(0) {

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

grades[i] = 0;

}

}

void setDetails(const std::string& studentName, int studentId) {

name = studentName;

id = studentId;

}

std::string getName() const {

return name;

}

int getId() const {

return id;

}

void setGrades(int subject1, int subject2, int subject3) {

grades[0] = subject1;

grades[1] = subject2;

grades[2] = subject3;

}

int\* getGrades() {

return grades;

}

};

int main() {

Student;

student.setDetails("John Doe", 123456);

student.setGrades(90, 85, 95);

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

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

int\* grades = student.getGrades();

std::cout << "Grades: ";

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

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

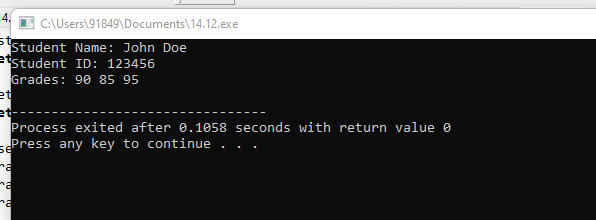
}

std::cout << std::endl;

return 0;

}

OUTPUT:



13. Create a 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;

class Rectangle{

private:

int l, b;

public:

void input(int len, int bre){

l = len;

b = bre;

}

int area(){

return l \* b;

}

};

int main(){

Rectangle r1, r2;

r1.input(10, 9);

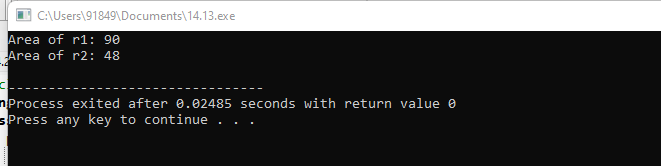
r2.input(8, 6);

cout << "Area of r1: " << r1.area() << endl;

cout << "Area of r2: " << r2.area() << endl;

}

OUTPUT:



14. Define a 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>

using namespace std;

class Employee

{

private:

int empid;

char empname[20];

int bsalary;

int allowance;

public:

Employee()

{

bsalary = 1000;

}

void GetData();

void DispData();

};

void Employee::GetData()

{

cout<<"Enter the employee id: ";

cin>>empid;

cout<<"Enter the employee name: ";

cin>>empname;

cout<<"Enter the employee allowace: ";

cin>>allowance;

}

void Employee::DispData()

{

cout<<endl<<empid<<"\t"<<empname<<"\t"<<bsalary<<"\t"<<allowance;

}

int main()

{

Employee e[3];

cout<<"Enter the employee information:"<<endl;

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

{

e[i].GetData();

}

cout<<endl<<"The employee information is:";

cout<<endl<<"EmpID \t Name \t Bsalary \t Allowance";

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

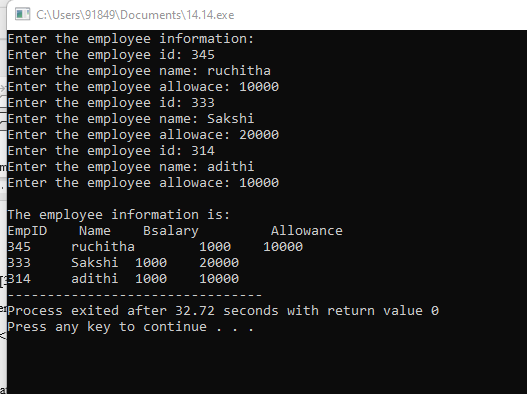
{

e[i].DispData();

}

}

OUTPUT:



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.

#include<bits/stdc++.h>

using namespace std;

class Complex {

public:

int real, imaginary;

Complex(int tempReal = 0, int tempImaginary = 0)

{

real = tempReal;

imaginary = tempImaginary;

}

Complex addComp(Complex C1, Complex C2)

{

Complex temp;

temp.real = C1.real + C2.real;

temp.imaginary = C1.imaginary + C2.imaginary;

return temp;

}

};

int main()

{

Complex C1(3, 2);

cout<<"Complex number 1 : "<< C1.real

<< " + i"<< C1.imaginary<<endl;

Complex C2(9, 5);

cout<<"Complex number 2 : "<< C2.real

<< " + i"<< C2.imaginary<<endl;

Complex C3;

C3 = C3.addComp(C1, C2);

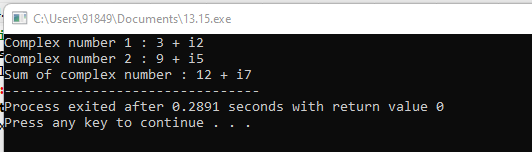
cout<<"Sum of complex number : "

<< C3.real << " + i"

<< C3.imaginary;

}

OUTPUT:



16. Write a 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 <bits/stdc++.h>

using namespace std;

int main()

{

int a = 5, b = 10;

cout << "Before swapping a = " << a << " , b = " << b

<< endl;

swap(a, b);

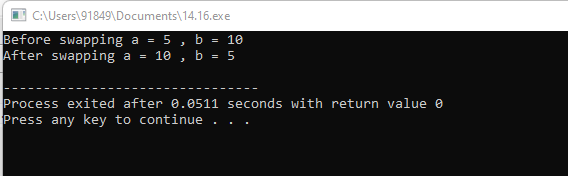
cout << "After swapping a = " << a << " , b = " << b

<< endl;

return 0;

}

OUTPUT:



17. Define a 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>

const double PI = 3.14159;

class Circle {

private:

double radius;

public:

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

double calculateArea() {

return PI \* pow(radius, 2);

}

double calculateCircumference() {

return 2 \* PI \* radius;

}

};

int main() {

double radius;

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

std::cin >> radius;

Circle circle(radius);

double area = circle.calculateArea();

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

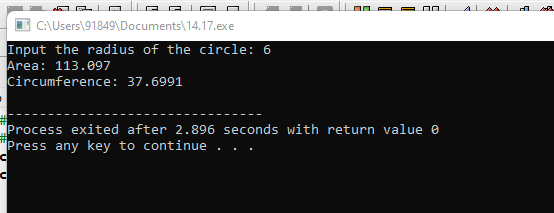
double circumference = circle.calculateCircumference();

std::cout << "Circumference: " << circumference << std::endl;

return 0;

}

OUTPUT:



18. Create a 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>

class Car {

private:

std::string company;

std::string model;

int year;

public:

Car(const std::string & comp, const std::string & mdl, int yr): company(comp), model(mdl), year(yr) {}

std::string getCompany() {

return company;

}

std::string getModel() {

return model;

}

int getYear() {

return year;

}

void setCompany(const std::string & comp) {

company = comp;

}

void setModel(const std::string & mdl) {

model = mdl;

}

void setYear(int yr) {

year = yr;

}

};

int main() {

Car car("AUDI", "A6", 2023);

std::cout << "Company: " << car.getCompany() << std::endl;

std::cout << "Model: " << car.getModel() << std::endl;

std::cout << "Year: " << car.getYear() << std::endl;

car.setCompany("BMW");

car.setModel("M4");

car.setYear(2022);

std::cout << "\nUpdated Company: " << car.getCompany() << std::endl;

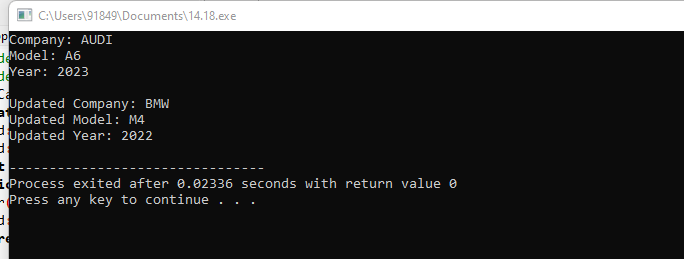
std::cout << "Updated Model: " << car.getModel() << std::endl;

std::cout << "Updated Year: " << car.getYear() << std::endl;

return 0;

}

OUTPUT:



19. Define a 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 <bits/stdc++.h>

using namespace std;

int largest(int arr[], int n)

{

int i;

int max = arr[0];

for (i = 1; i < n; i++)

if (arr[i] > max)

max = arr[i];

return max;

}

int main()

{

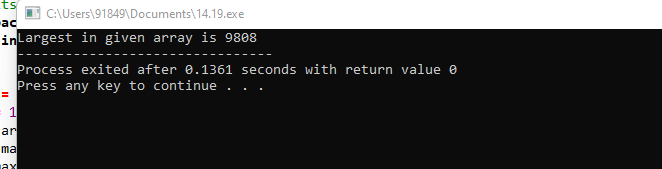
int arr[] = { 10, 324, 45, 90, 9808 };

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

cout << "Largest in given array is " << largest(arr, n);

return 0;

}

OUTPUT:  


20. Write a 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;

int main()

{

int width, lngth, area, peri;

cout << "\n\n Find the Area and Perimeter of a Rectangle :\n";

cout << "-------------------------------------------------\n";

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

cin >> lngth;

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

cin >> width;

area = (lngth \* width);

peri = 2 \* (lngth + width);

cout << " The area of the rectangle is : "<< area << endl;

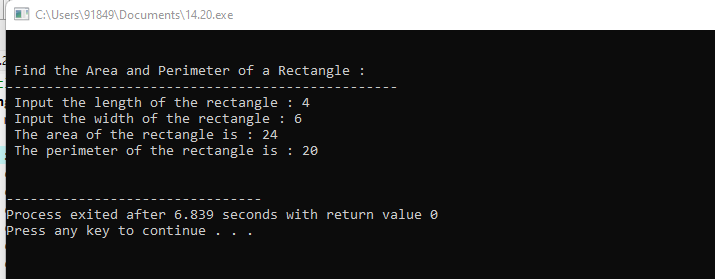
cout << " The perimeter of the rectangle is : "<< peri << endl;

cout << endl;

return 0;

}

OUTPUT:



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);

}

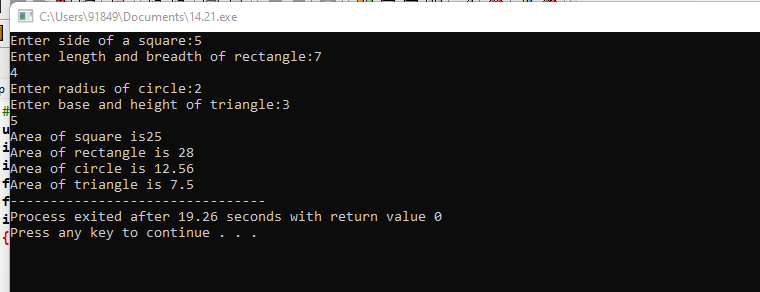
float area(float bs,float ht)

{

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

}

OUTPUT:



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;

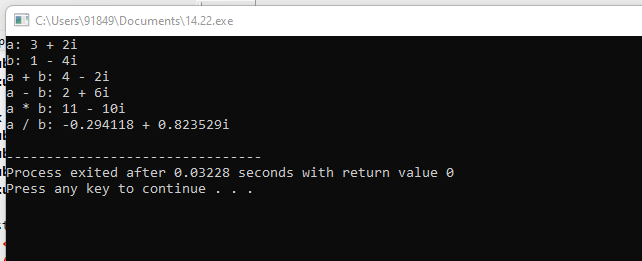
std::cout << "a \* b: " << a \* b << std::endl;

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

return 0;

}

OUTPUT:



23. Create a C++ program that defines a class named **Matrix** to represent a 2D matrix. Implement operator overloading for matrix addition, subtraction, and multiplication. Use dynamic memory allocation for the matrix array within the class.

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();

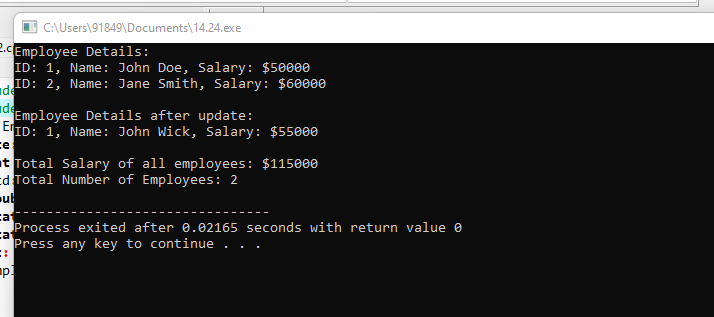
std::cout << "\nTotal Salary of all employees: $" << Employee::getTotalSalary() << std::endl;

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

return 0;

}

OUTPUT:



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;

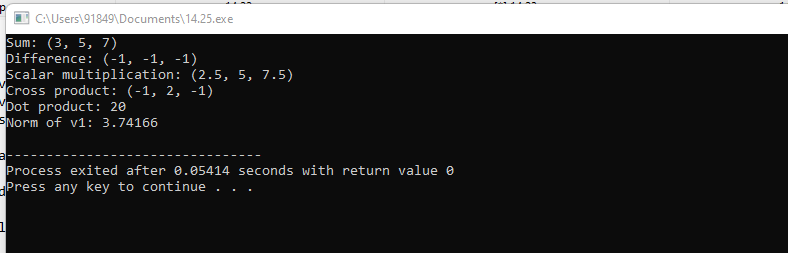
double normV1 = v1.norm();

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

return 0;

}

OUTPUT:



1. 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)

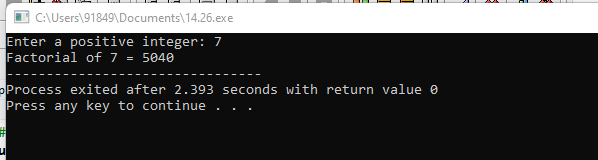
return n \* factorial(n - 1);

else

return 1;

}

OUTPUT:



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);

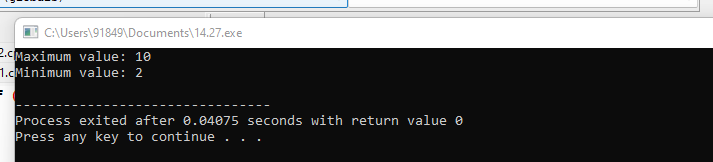
std::cout << "Maximum value: " << result.first << std::endl;

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

return 0;

}

OUTPUT:



28. Create a 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;

double 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(5, 3);

Circle circle(4);

Triangle triangle(3, 4, 5);

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

std::cout << "Area: " << rect.area() << std::endl;

std::cout << "Perimeter: " << rect.perimeter() << std::endl;

std::cout << "\nCircle:" << std::endl;

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

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

std::cout << "\nTriangle:" << std::endl;

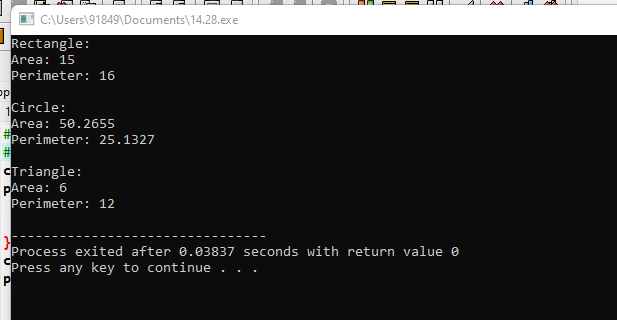
std::cout << "Area: " << triangle.area() << std::endl;

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

return 0;

}

OUTPUT:



29. Write a 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>

template<typename T>

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

int main() {

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

int intSize = sizeof(intArray) / sizeof(intArray[0]);

std::cout << "Average of integers: " << average(intArray, intSize) << std::endl;

double doubleArray[] = {1.5, 2.5, 3.5, 4.5, 5.5};

int doubleSize = sizeof(doubleArray) / sizeof(doubleArray[0]);

std::cout << "Average of doubles: " << average(doubleArray, doubleSize) << std::endl;

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

int floatSize = sizeof(floatArray) / sizeof(floatArray[0]);

std::cout << "Average of floats: " << average(floatArray, floatSize) << std::endl;

return 0;

}

template<typename T>

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

T sum = 0;

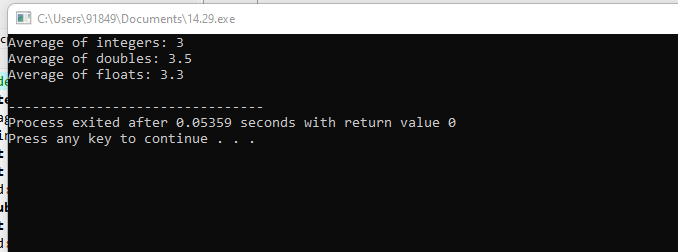
for (int i = 0; i < size; ++i) {

sum += arr[i];

}

return sum / size;

}

OUTPUT:  


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>

using namespace std;

class Swap {

int temp, a, b;

public:

Swap(int a, int b)

{

this->a = a;

this->b = b;

}

friend void swap(Swap&);

};

void swap(Swap& s1)

{

cout << "\nBefore Swapping: " << s1.a << "\n" << s1.b;

s1.temp = s1.a;

s1.a = s1.b;

s1.b = s1.temp;

cout << "\nAfter Swapping: " << s1.a << "\n" << s1.b;

}

int main()

{

Swap s(4, 6);

swap(s);

return 0;

}

OUTPUT:

