**Unit-1**

1. **Find the roots of a quadratic equation**

**Aim:** To write a C++ program to Find the roots of a quadratic equation

**Program:**

#include <iostream>

#include <cmath>

using namespace std;

int main()

{

double a, b, c, d, root1, root2, realPart, imagPart;

cout << "Enter a, b, c values: ";

cin >> a >> b >> c;

d = b \* b - 4 \* a \* c;

if (d == 0)

{

root1 = root2 = -b / (2 \* a);

cout << "Roots are real and equal: " << root1 << " " << root2 << endl;

}

else if (d > 0)

{

root1 = (-b + sqrt(d)) / (2 \* a);

root2 = (-b - sqrt(d)) / (2 \* a);

cout << "Roots are real and distinct: " << root1 << " " << root2 << endl;

}

else

{

realPart = -b / (2 \* a);

imagPart = sqrt(-d) / (2 \* a);

cout << "Roots are imaginary (complex): " << endl;

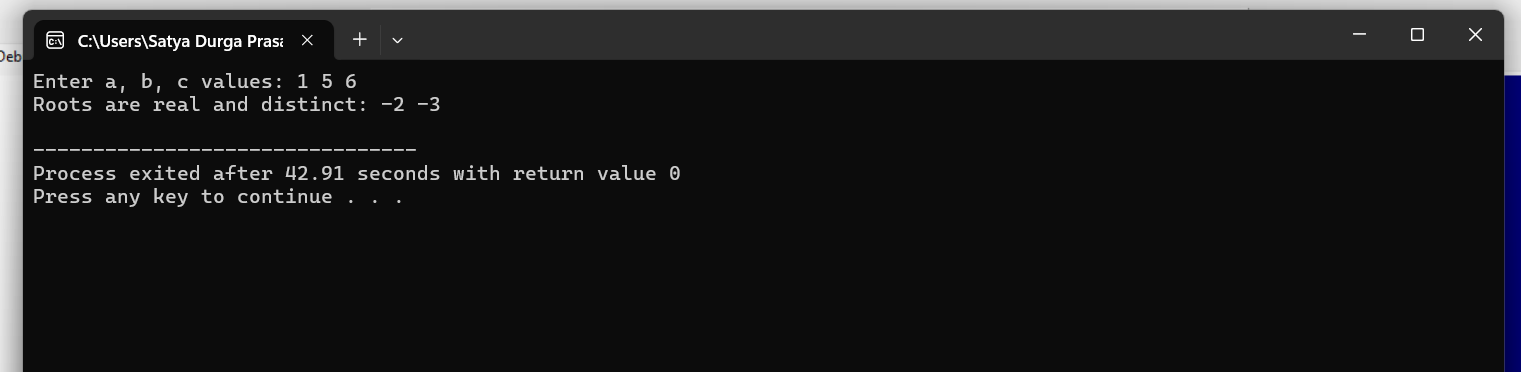
cout << "Root 1 = " << realPart << " + " << imagPart << "i" << endl;

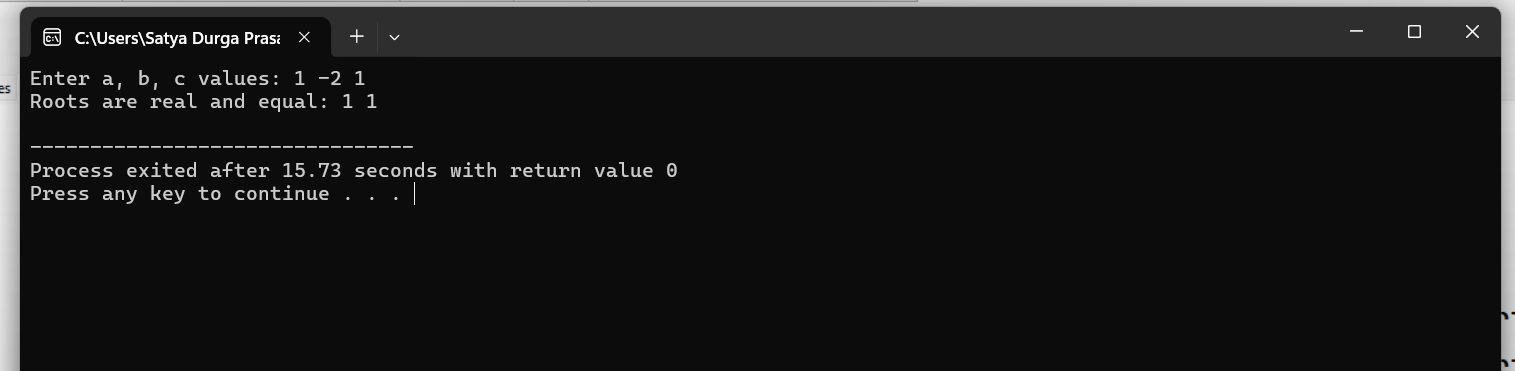
cout << "Root 2 = " << realPart << " - " << imagPart << "i" << endl;

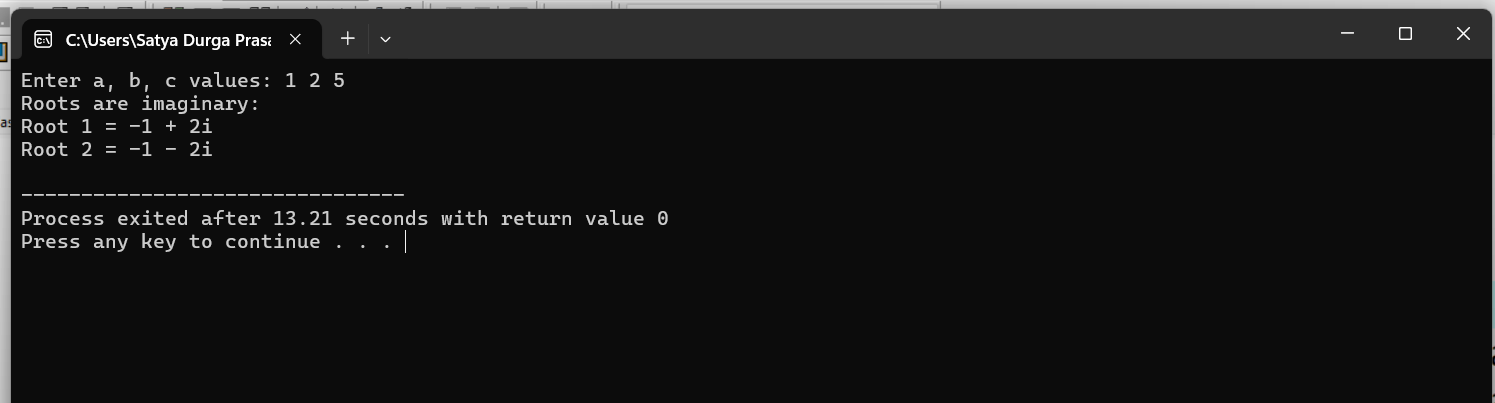
}

return 0;

}

**Output: **

****

****

1. **Find factorial of a given number using recursion**

**Aim:** To write a C++ program to find the factorial of a given number using recursion.

**Program:**

#include <iostream>

using namespace std;

int factorial(int n)

{

if (n == 0 || n == 1)

return 1;

else

return n \* factorial(n - 1);

}

int main()

{

int num;

cout << "Enter a number: ";

cin >> num;

if (num < 0)

{

cout << "Factorial is not defined for negative numbers." << endl;

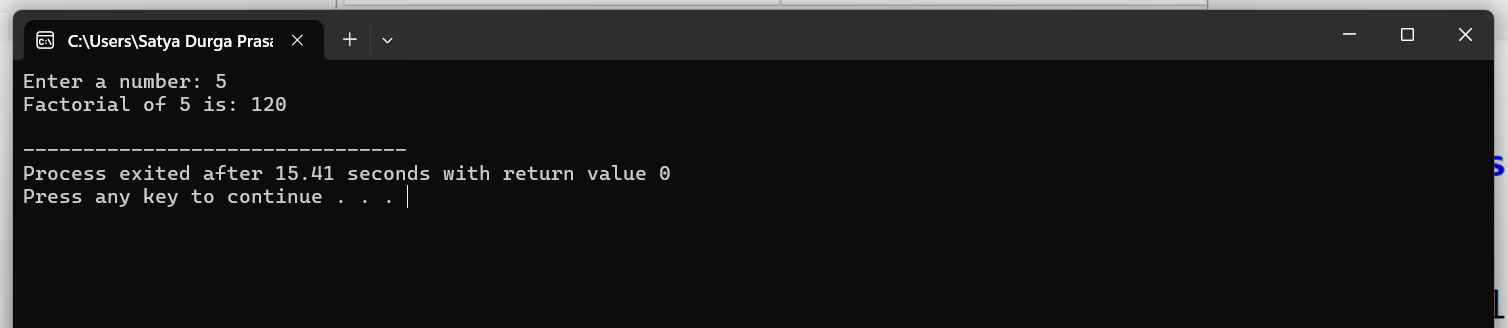
} else {

cout << "Factorial of " << num << " is: " << factorial(num) << endl;

}

return 0;

}

**Output:**

1. **Implement scope resolution and namespaces.**

**Aim:** To demonstrate the use of scope resolution operator and namespaces in C++.

**Program:**

#include <iostream>

using namespace std;

int num = 100;

namespace MyNamespace

{

int num = 200;

void display()

{

cout << "Inside namespace, num = " << num << endl;

}

}

int main()

{

int num = 300;

cout << "Local num = " << num << endl;

cout << "Global num = " << ::num << endl;

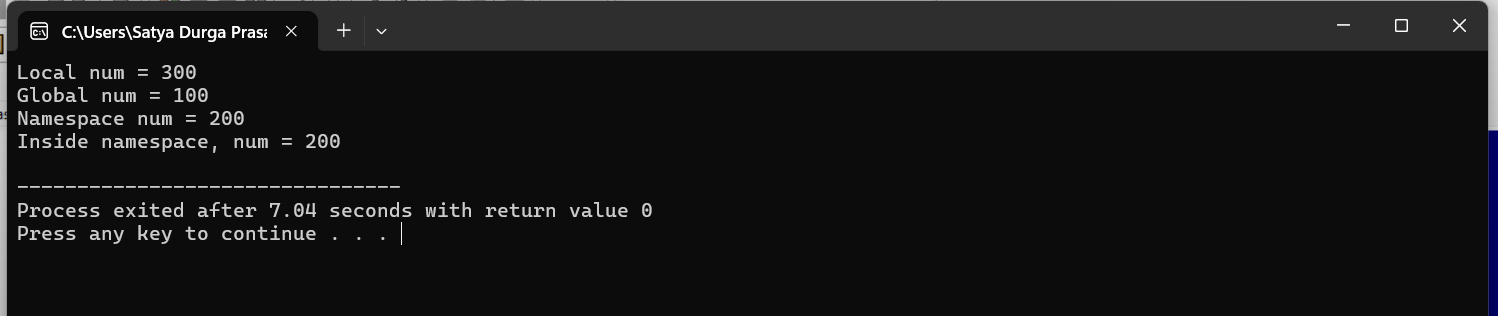
cout << "Namespace num = " << MyNamespace::num << endl;

MyNamespace::display();

return 0;

}

**Output:**



1. **Illustrate the use of default arguments and access specifiers**

**Aim:** To illustrate the use of default arguments in functions and access specifiers (public, private, protected) in a C++ class.

**Program:**

#include <iostream>

using namespace std;

class Example

{

private:

int privateValue;

protected:

int protectedValue;

public:

Example()

{

privateValue = 10;

protectedValue = 20;

}

void display(int x = 5, int y = 10)

{

cout << "x = " << x << ", y = " << y << endl;

cout << "private Value: " << privateValue << endl;

cout << "Protected Value: " << protectedValue << endl;

}

};

int main()

{

Example obj;

cout << "Calling display without arguments:" << endl;

obj.display();

cout << "Calling display with one argument:" << endl;

obj.display(15);

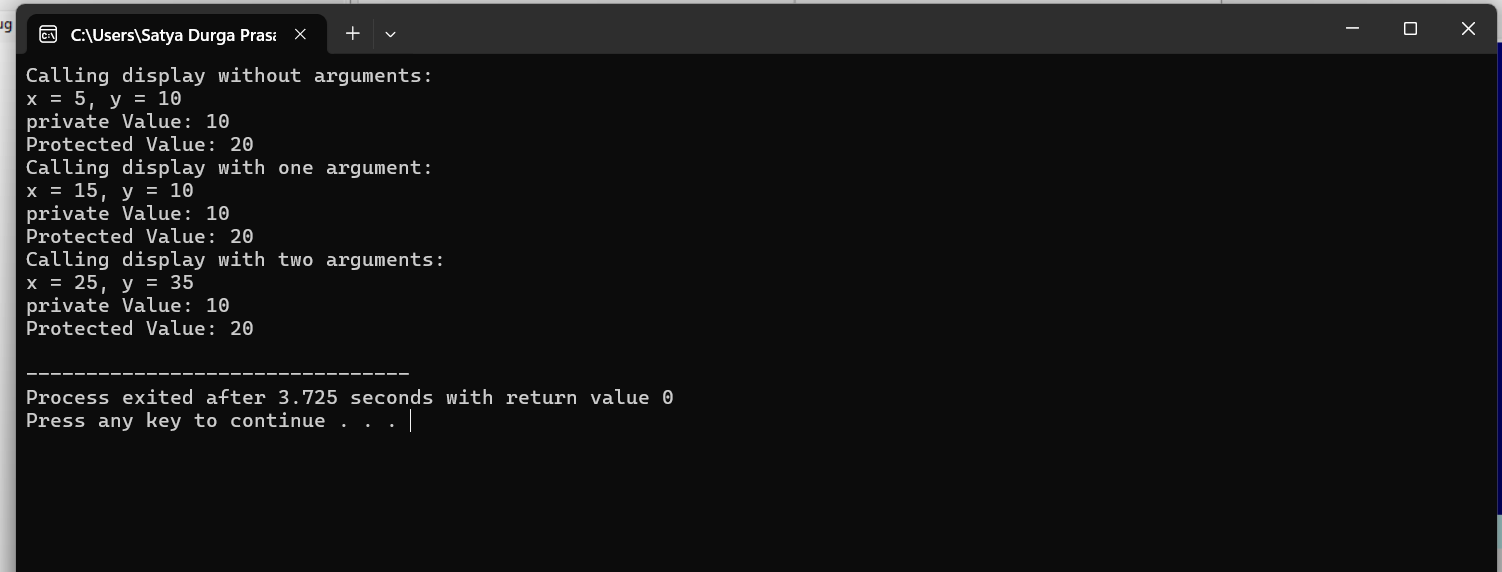
cout << "Calling display with two arguments:" << endl;

obj.display(25, 35);

return 0;

}

**Output:**



**Unit-2**

1. **Program to illustrate inline functions and function overloading.**

**Aim:** To demonstrate the use of inline functions and function overloading in C++

**Program:**

#include <iostream>

using namespace std;

inline int square(int x) {

return x \* x;

}

int add(int a, int b) {

return a + b;

}

float add(float a, float b) {

return a + b;

}

int main() {

int num = 5;

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

int a = 10, b = 20;

float x = 3.5, y = 2.5;

cout << "Addition of integers: " << add(a, b) << endl;

cout << "Addition of floats: " << add(x, y) << endl;

return 0;

}

**Output:** 

1. **Program to illustrate friend function**

**Aim:** To demonstrate the use of a friend function in C++ that can access private members of a class.

**Program:**

#include <iostream>

using namespace std;

class Box

{

private:

int length;

public:

Box(int l)

{

Length=l;

}

friend void showLength(Box b);

};

void showLength(Box b)

{

cout << "Length of the box is: " << b.length << endl;

}

int main() {

Box b1(25);

showLength(b1);

return 0;

}

**Output:**



**3.) Program to illustrate the use of Constructors and Destructors.**

**Aim:** To demonstrate the use of **constructors** and **destructors** in C++ for initializing and cleaning up objects.

**Program:**

#include <iostream>

using namespace std;

class Demo

{

public:

// Constructor

Demo()

{

cout << "Constructor called: Object created." << endl;

}

// Destructor

~Demo()

{

cout << "Destructor called: Object destroyed." << endl;

}

};

int main()

{

cout << "Inside main function." << endl;

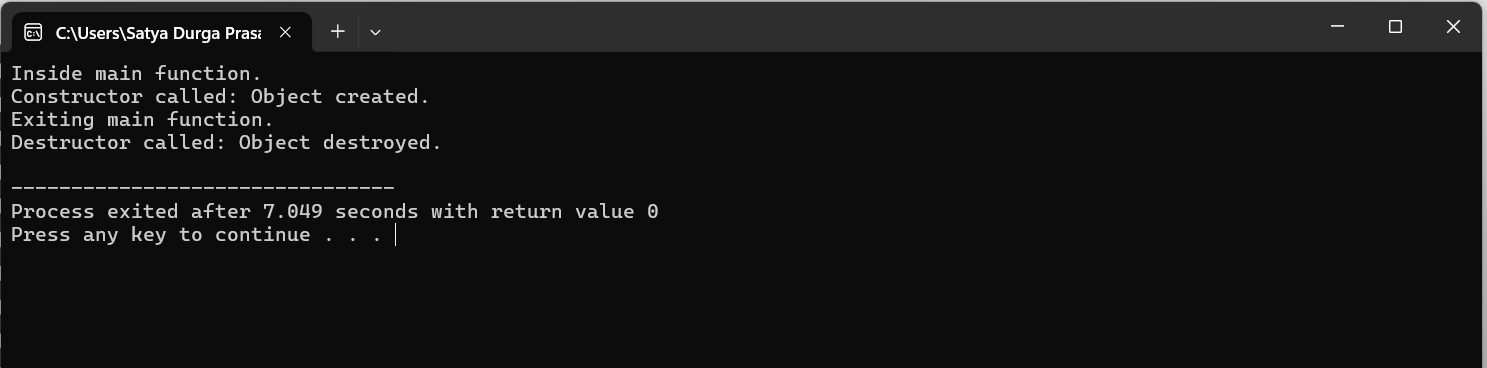
Demo obj;

cout << "Exiting main function." << endl;

return 0;

}

**Output:**



1. **Program illustrating Constructor overloading**.

**Aim:** To demonstrate constructor overloading in C++ by creating multiple constructors with different parameter lists to initialize objects in various ways.

**Program:**

#include <iostream>

using namespace std;

class Rectangle

{

private:

int length, breadth;

public:

Rectangle()

{

length = 0;

breadth = 0;

}

Rectangle(int l)

{

length = l;

breadth = l;

}

Rectangle(int l, int b)

{

length = l;

breadth = b;

}

void displayArea()

{

cout << "Area: " << length \* breadth << endl;

}

};

int main()

{

Rectangle r1;

Rectangle r2(5);

Rectangle r3(4, 6);

cout << "Rectangle r1: ";

r1.displayArea();

cout << "Rectangle r2: ";

r2.displayArea();

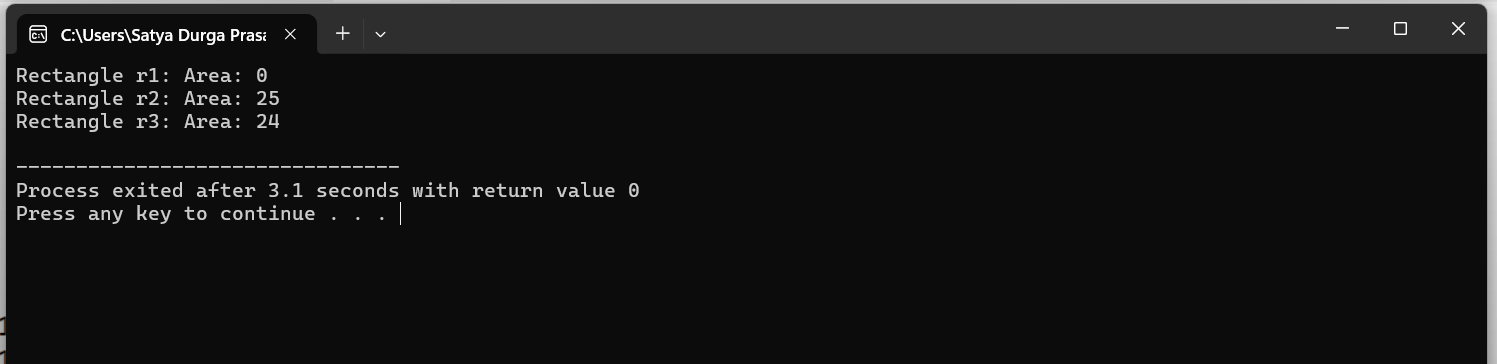
cout << "Rectangle r3: ";

r3.displayArea();

return 0;

}

**Output:**

****

1. **Program illustrating Copy Constructor**.

**Aim:** To demonstrate the use of a **copy constructor** in C++ that creates a new object by copying data from an existing object.

**Program:**

#include <iostream>

using namespace std;

class Student

{

private:

string name;

int age;

public:

Student(string n, int a)

{

name = n;

age = a;

}

Student(const Student &s)

{

name = s.name;

age = s.age;

}

void display()

{

cout << "Name: " << name << ", Age: " << age << endl;

}

};

int main()

{

Student s1("Pinisetty", 20);

Student s2 = s1;

cout << "Original Student: ";

s1.display();

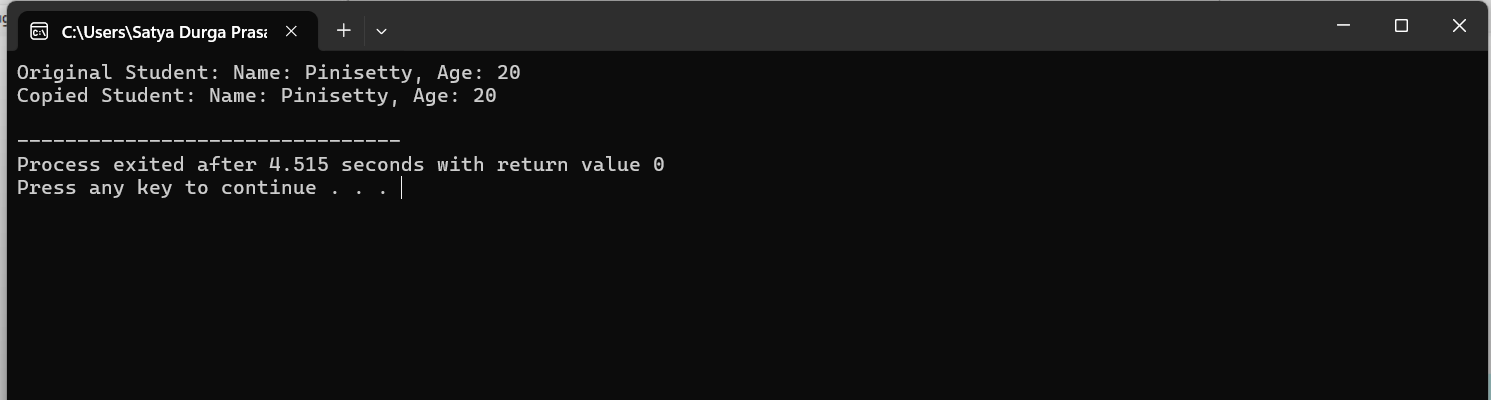
cout << "Copied Student: ";

s2.display();

return 0;

}

**Output:**

****