**WEEK 2**

**Experiment 1**

**Title :** Demonstration of INLINE FUNCTIONS

**Description:**

This C++ program demonstrates the use of an inline function to calculate the square of a number.

This makes the program efficient because it avoids the overhead of a regular function call while keeping the code clean and readable.

**Code:**

#include<iostream>  
using namespace std;  
inline int square (int x)   
{  
 return x \* x;  
}  
int main () {  
 cout << "Square of 5 is: " << square(5) << endl;  
 return 0;  
}

**Result:**

The program executed successfully and demonstrated the concept as intended.

**A screenshot of a computer

AI-generated content may be incorrect.**

Experiment No: 2

Title:

Adding Two Classes using FRIEND FUNCTIONS

Description:

This program illustrates the use of a friend function in C++.  
The class Demo has private data members x and y, and a private function fun1().  
The non-member function add() is declared as a friend, so it can access and modify the private members of Demo.  
In main(), add(ob) sets x and y to 10 and 20 and prints their sum, even though these members are private.

Source Code:

#include<iostream>  
using namespace std;  
class Demo   
{  
 private:  
 int x, y;  
 void fun1()   
 {  
 x = 10;  
 y = 20;  
 cout << "I am inside member function (fun1()) in the class" << endl;  
 }  
 friend void add(Demo d);  
};  
void add(Demo d) {  
 d.x = 10;  
 d.y = 20;  
 cout << "Sum = " << d.x + d.y << endl;  
}  
int main() {  
 Demo ob;  
 add(ob);  
 return 0;  
}

Result:

The program executed successfully and demonstrated the concept as intended.

A computer screen shot of a black screen

AI-generated content may be incorrect.

Experiment No: 3

**Title:**

Demonstration of OVERLOADING MEMBER FUNCTION

**Description:**

This program demonstrates function overloading in C++.  
The class DemoOverloading defines four add() methods with different parameter types and counts.  
The compiler chooses the correct version of add() based on the arguments provided in each call.  
Running the program shows different messages and sums depending on which overloaded function is executed.

**Source Code**

#include<iostream>  
using namespace std;  
class DemoOverloading  
{  
 public:  
 void add() {  
 cout << "\nI am in add () - No Arg ";  
 }  
 void add(int a, int b) {  
 cout << "\nI am in add (int, int ), The sum = " << a+b;  
 }  
 void add(float a, float b) {  
 cout << "\nI am in add (float, float ) - The sum = " << a+b;  
 }  
 void add(int a, float b) {  
 cout << "\nI am in add(int, float) - The sum=" << a+b;  
 }  
};  
int main()  
{  
 DemoOverloading ob;  
 ob.add();  
 ob.add(10,20);  
 ob.add(10.5f,10.5f);  
 ob.add(10,10.5f);  
}

Result:

**A screenshot of a computer

AI-generated content may be incorrect.**The program executed successfully and demonstrated the concept as intended.

**Experiment No: 4**

**Title:**

Demo Constructor and Destructor

**Description:**

This program shows how constructors and destructors work in C++.  
When the Demo object is created inside the inner block, the constructor runs and prints a message, and the display() function is called.  
As the block ends, the destructor is automatically invoked to destroy the object before continuing with the rest of main().

**Source Code**

#include <iostream>  
using namespace std;  
class Demo  
{  
public:  
 Demo() {  
 cout << "Constructor called: Object is created." << endl;  
 }  
 ~Demo () {  
 cout << "Destructor called: Object is destroyed." << endl;  
 }  
 void display() {  
 cout << "Inside display function." << endl;  
 }  
};  
int main() {  
 {  
 Demo obj;  
 obj. display();  
 }  
 cout << "Back in main function." << endl;  
 return 0;  
}

**Result:**

The program executed successfully and demonstrated the concept as intended.

**A screenshot of a computer

AI-generated content may be incorrect.**

**Experiment No: 5**

**Title:**

Constructor Overloading

**Description:**

This program shows constructor overloading and the use of a destructor in C++.  
The class Demo has three constructors: a default constructor, one taking an integer, and one taking a character array, each printing a message when called.  
In main(), objects ob, ob1, and ob2 invoke these constructors respectively, displaying different messages.  
When the program ends, the destructor is automatically called for each object, printing its message.

**Source Code**

#include<iostream>  
using namespace std;  
class Demo  
{  
 public:  
 Demo() {  
 cout << "I am in Default Constructor";  
 }  
 Demo(int x) {  
 cout << "\nI am in One arg constructor" << x;  
 }  
 Demo(char str[12]) {  
 cout << "\nI am in one arg string constructor" << str;  
 }  
 ~Demo() {  
 cout << "\nI am in DESTRUCTOR";  
 }  
};  
int main()  
{  
 Demo ob;  
 Demo ob1(10);  
 Demo ob2("CPP Demo Code");  
}

**Result:**

The program executed successfully and demonstrated the concept as intended.

**A screenshot of a computer

AI-generated content may be incorrect.**

**Experiment No: 6**

**Title:**

Demonstration of Copy Constructor

**Description:**

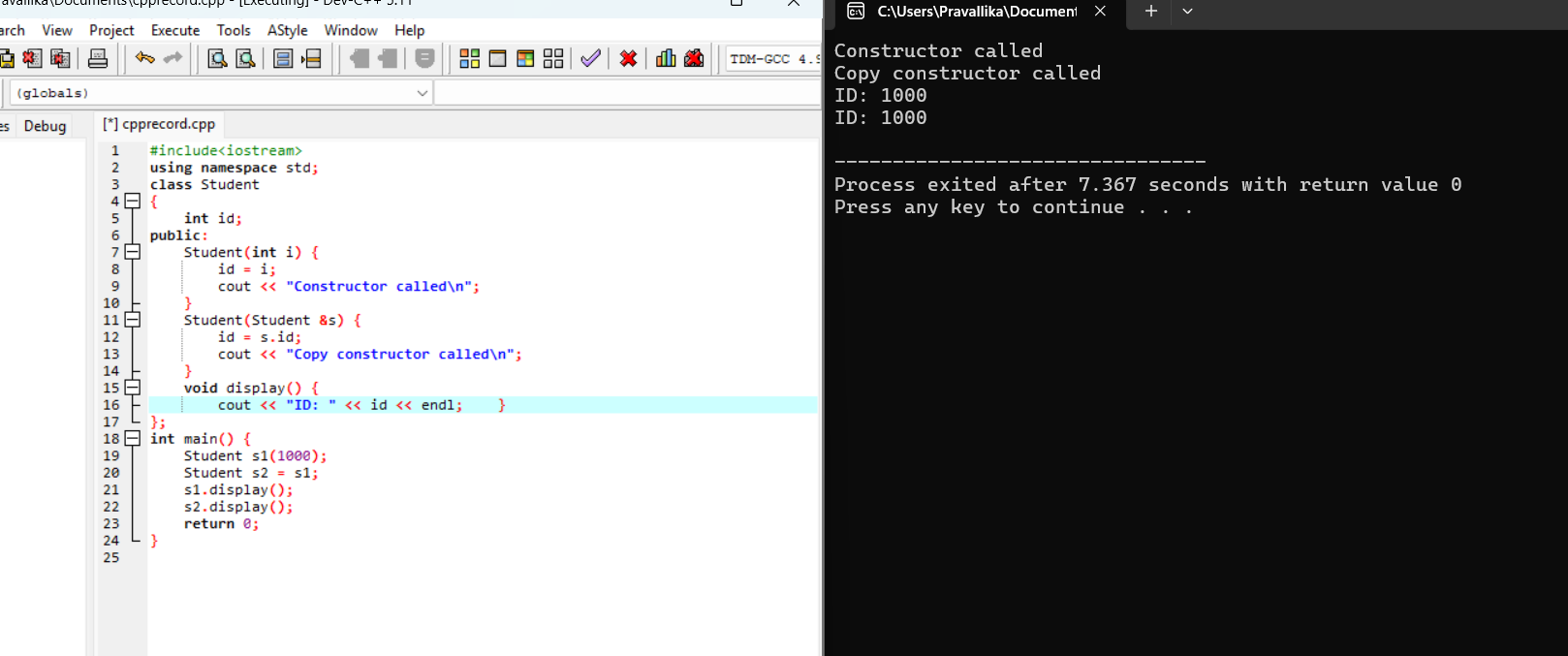
This program demonstrates a copy constructor in C++.  
The Student class has a parameterized constructor to set id and a copy constructor that creates a new object using another Student object’s data.  
In main(), s1 is created with ID 1000, and s2 is initialized as a copy of s1, triggering the copy constructor.  
Both display() calls show the same ID, confirming that the data was successfully copied.

**Source Code:**

#include<iostream>  
using namespace std;  
class Student   
{  
 int id;  
public:  
 Student(int i) {  
 id = i;  
 cout << "Constructor called\n";  
 }  
 Student(Student &s) {  
 id = s.id;  
 cout << "Copy constructor called\n";  
 }  
 void display() {  
 cout << "ID: " << id << endl;  
 }  
};  
int main() {  
 Student s1(1000);  
 Student s2 = s1;  
 s1.display();  
 s2.display();  
 return 0;  
}

**Result:**

The program executed successfully and demonstrated the concept as intended.

****