# Unit-2

# Experiment: 1

## :

Demonstration of INLINE FUNCTIONS

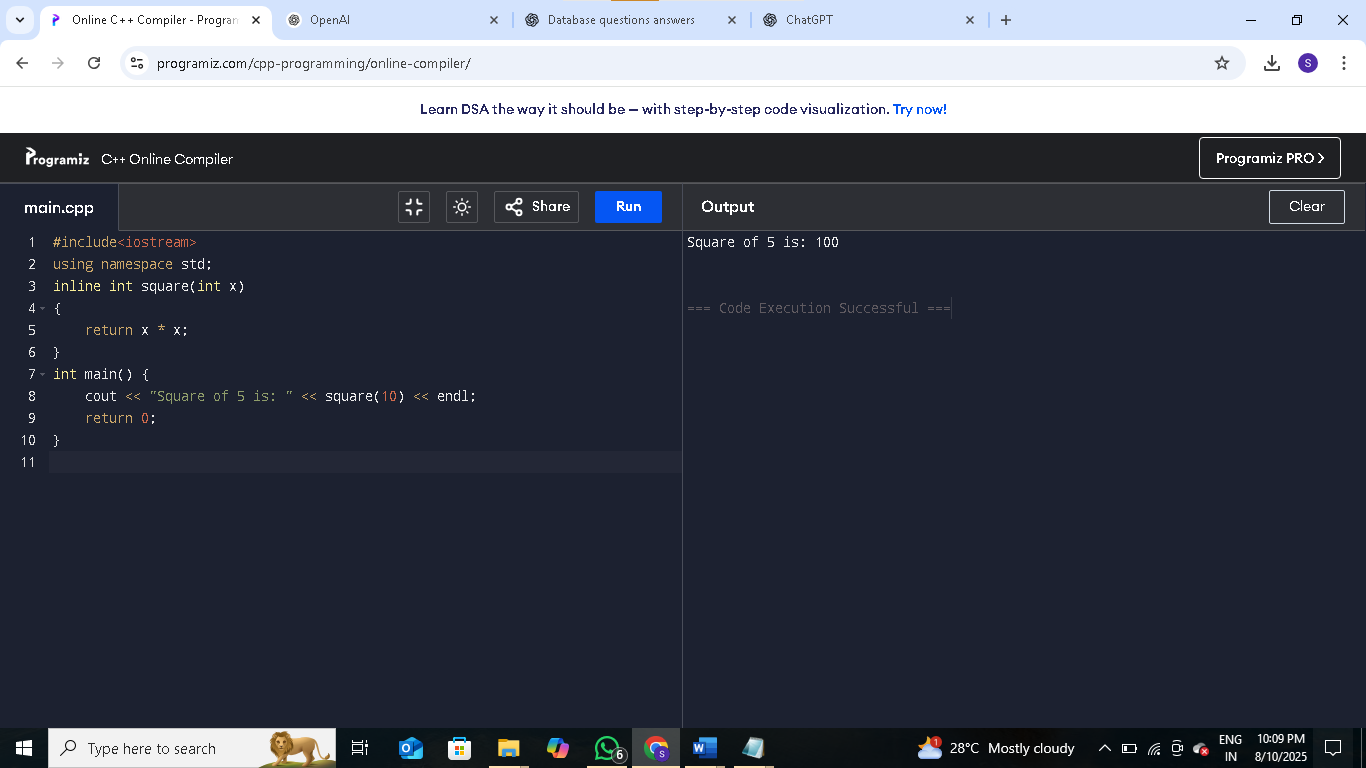
## Description:

This program demonstrates the use of inline functions in C++ where the function body is substituted at compile-time to improve performance.

## Source 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; }

## Sample Input and Output



**Result:**

The program executed successfully and demonstrated the concept as intended.

# Experiment: 2

## Aim:

Adding Two Classes using FRIEND FUNCTIONS

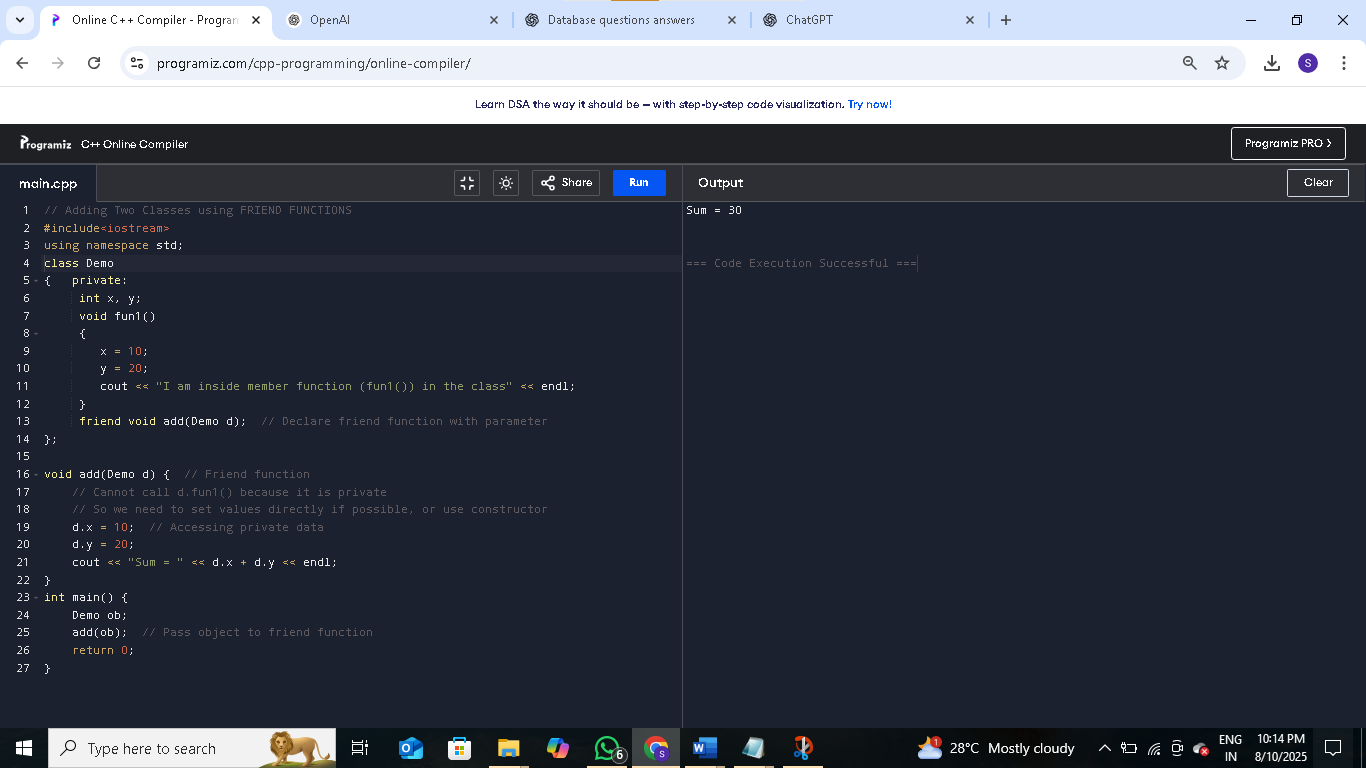
## Description:

This program demonstrates friend functions in C++ which allow non-member functions to access the private data of a class.

## 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;  
}

## Sample Input and Output:



## Result:

The program executed successfully and demonstrated the concept as intended.

# Experiment: 3

## Aim:

Demonstration of OVERLOADING MEMBER FUNCTION

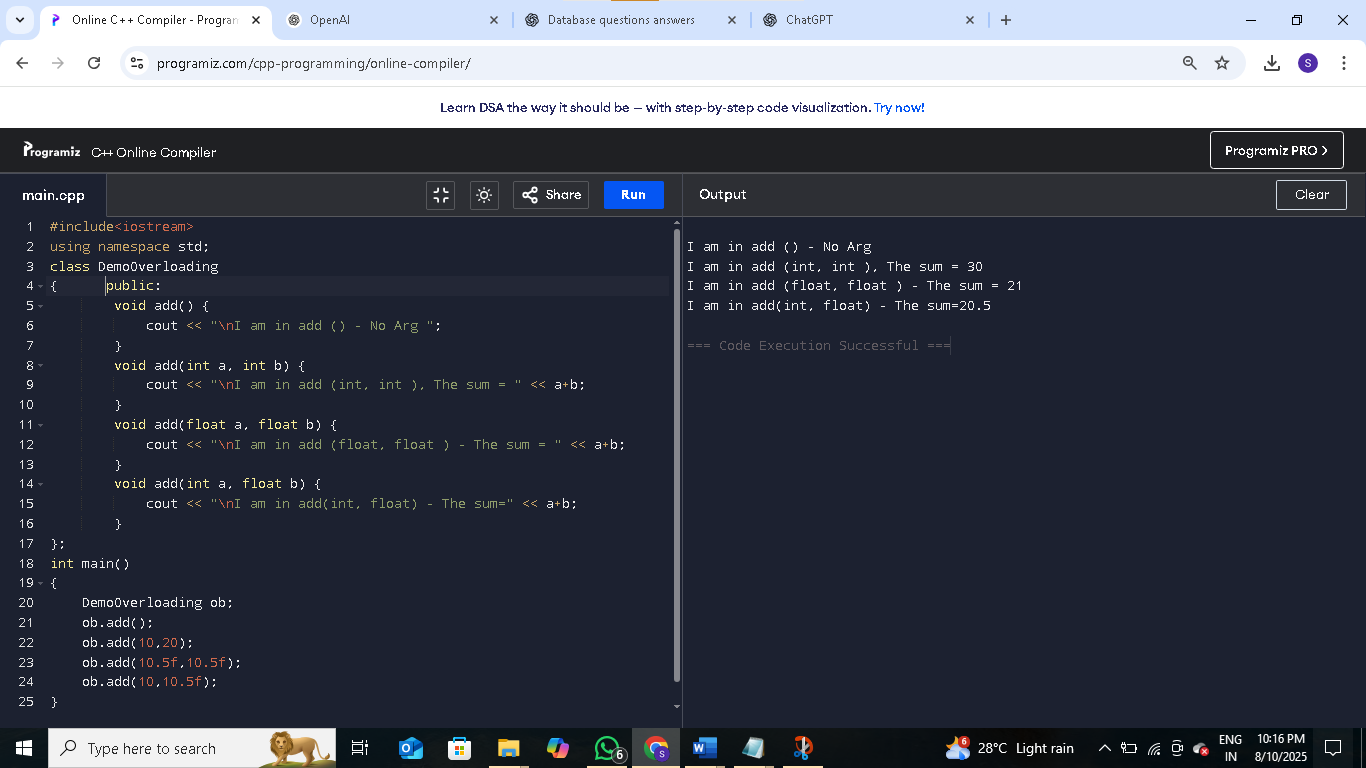
## Description:

This program demonstrates function overloading in C++, where multiple functions share the same name but differ in parameter types or counts.

## 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);  
}

## Sample Input and Output:



## Result:

The program executed successfully and demonstrated the concept as intended.

# Experiment: 4

## Aim:

Demo Constructor and Destructor

## Description:

This program demonstrates how constructors are called when an object is created and destructors when the object goes out of scope.

## 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;  
}

## Sample Input and Output:

A screenshot of a computer

AI-generated content may be incorrect.

## Result:

The program executed successfully and demonstrated the concept as intended.

# Experiment: 5

## Aim:

Constructor Overloading

## Description:

This program demonstrates constructor overloading, where multiple constructors are defined with different parameter lists.

## 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.

## Sample Input and Output:

## 

# Experiment : 6

## Aim:

Demonstration of Copy Constructor

## Description:

This program demonstrates a copy constructor, which creates a new object as a copy of an existing object.

## 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;  
}

## Sample Input and Output:

A computer screen shot of a computer screen

AI-generated content may be incorrect.

## Result:

The program executed successfully and demonstrated the concept as intended.