

LAB 4

OBJECTIVE: To Demonstrate the Friend Functions in C++.

THEORY:

- Friend functions are functions declared outside a class but have access to the private and protected members of that class.
- They are used to establish a relationship between a class and a function, granting the function the ability to directly manipulate the class's internal data.
- Friend functions are primarily used for tasks that require access to private members but cannot be performed efficiently or logically within the class itself.

Syntax

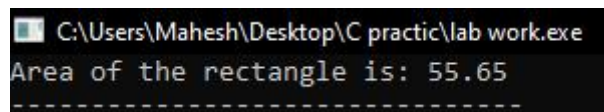
```
class ClassName {  
    friend void friendFunction(ClassName& object); // Declaration  
};  
  
void friendFunction(ClassName& object)  
{    // Access private and protected members of object    }
```

PROGRAMS:

- a. Define a Rectangle class with private data members length and width. Create a friend function getArea that calculates and returns the area of the rectangle.

```
#include<iostream>  
using namespace std;  
  
class Rectangle {  
    double length, width;  
    public:  
        Rectangle(double l,double w): length(l),width(w){}  
        friend double getArea(Rectangle);  
};  
double getArea(Rectangle rect){  
    return rect.length * rect.width ;  
}  
int main(){  
    Rectangle r1(10.5,5.3);  
    double Area = getArea(r1);  
    cout<<"Area of the rectangle is: "<<Area;  
    return 0;  
}
```

Output:



```
C:\Users\Mahesh\Desktop\C practic\lab work.exe  
Area of the rectangle is: 55.65  
-----
```

- b. Define a person class to include a friend function compareAge that compares the ages of two person objects and prints which one has a greater age.

```
#include<iostream>  
using namespace std;  
  
class Person {  
    string name;    int age;  
    public:
```

```

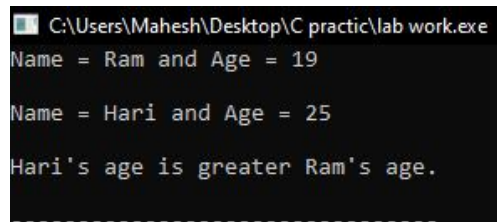
        Person(string n,int a): name(n),age(a){}
        void display(){
            cout<<"Name = "<<name<<" and Age = "<<age<<endl<<endl;
        }
    friend void compareAge(Person, Person);
};

void compareAge(Person p1,Person p2){
    if(p1.age > p2.age)
        cout<<p1.name<<"s age is greater "<<p2.name<<"s age."<<endl;
    else if(p2.age > p1.age)
        cout<<p2.name<<"s age is greater "<<p1.name<<"s age."<<endl;
    else
        cout<<"Both "<<p1.name<<" and "<<p2.name<<" have same age."<<endl;
}

int main(){
    Person p1("Ram",19), p2("Hari",25);
    p1.display();
    p2.display();
    compareAge(p1,p2);
    return 0;
}

```

Output:



```

C:\Users\Mahesh\Desktop\C practic\lab work.exe
Name = Ram and Age = 19
Name = Hari and Age = 25
Hari's age is greater Ram's age.
-----

```

- c. Create classes Circle and Rectangle. Define a friend function compareArea outside both classes that compares the area of a Circle and a Rectangle object and prints the larger one.

```

#include <iostream>
using namespace std;

class Rectangle;
class Circle {
    double radius, area;
public:
    Circle(double r) : radius(r) {
        area = 3.14159 * radius * radius;
    }
    void displayArea() {
        cout << "Area of the Circle is: " << area << endl;
    }
    friend void compareArea(Circle, Rectangle);
};

class Rectangle {
    double length, width, area;
public:
    Rectangle(double l, double w) : length(l), width(w) {
        area = length * width;
    }
    void displayArea() {
        cout << "Area of the Rectangle is: " << area << endl;
    }
    friend void compareArea(Circle, Rectangle);
};

```

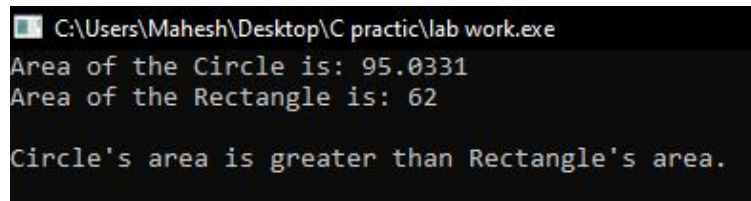
```

void compareArea(Circle c, Rectangle r) {
    cout<<endl;
    if (c.area > r.area) {
        cout << "Circle's area is greater than Rectangle's area." << endl;
    } else if (r.area > c.area) {
        cout << "Rectangle's area is greater than Circle's area." << endl;
    } else {
        cout << "Both have equal area." << endl;
    }
}

int main() {
    Circle circle(5.5);
    Rectangle rectangle(10, 6.2);
    circle.displayArea();
    rectangle.displayArea();
    compareArea(circle, rectangle);
    return 0;
}

```

Output:



```

C:\Users\Mahesh\Desktop\C practic\lab work.exe
Area of the Circle is: 95.0331
Area of the Rectangle is: 62

Circle's area is greater than Rectangle's area.

```

- d. Design a class BankAccount with private data members balance and ownerName. Implement a friend function transferFunds that allows transferring funds between two BankAccount objects.

```

#include<iostream>
using namespace std;

class BankAccount{
    double balance ;   string ownerName;
public:
    BankAccount(double b,string n): balance(b), ownerName(n){}
    void display(){
        cout<<"\nAccount owner Name: "<<ownerName<<endl;
        cout<<"Current Balance: "<<balance<<endl;
    }
    friend void transferFunds(BankAccount&,BankAccount&,double);
};

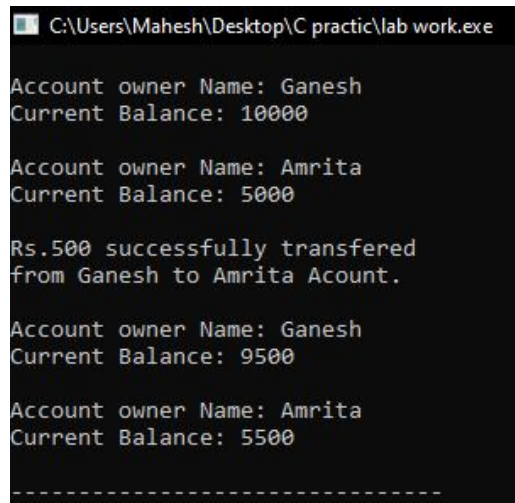
void transferFunds(BankAccount& sender,BankAccount& receiver,double amount){
    if(amount <= sender.balance){
        sender.balance -= amount;
        receiver.balance += amount;
        cout<<endl<<"Rs. "<<amount<<" successfully transfered \nfrom
"<<sender.ownerName<<" to "<<receiver.ownerName<<" Account."<<endl;
    }else
        cout<<"\nUnsufficient balance you want to transfer...!!!"<<endl;
    }

int main(){
    BankAccount p1(10000,"Ganesh"), p2(5000,"Amrita");
    p1.display();
    p2.display();
    transferFunds(p1,p2,500);
    p1.display();
    p2.display();
    return 0;
}

```

}

Output:



```
C:\Users\Mahesh\Desktop\C practic\lab work.exe

Account owner Name: Ganesh
Current Balance: 10000

Account owner Name: Amrita
Current Balance: 5000

Rs.500 successfully transfered
from Ganesh to Amrita Acount.

Account owner Name: Ganesh
Current Balance: 9500

Account owner Name: Amrita
Current Balance: 5500

-----
```

RESULTS AND DISCUSSION:

The friend function can access and manipulate the private and protected members of the class. It allows much flexibility in certain situations where it's necessary to access private data from outside the class, without exposing these members directly to the public interface. After the experiment was successful, the program helps to understand how to use **friend function** in C++ programming language.

CONCLUSION:

This laboratory exercise provided a hands-on experience in C++ programming. Students gained practical knowledge of implementing the basic of friend function in C programming and are now better equipped to undertake more complex programming tasks in the future. By understanding the syntax and use cases of friend functions, we can effectively leverage them in our C++ programs when necessary.

LAB 5

OBJECTIVE:

Passing object as a function argument and returning object from a function in C++.

THEORY:

In C++, you can pass objects as arguments to functions and also return objects from functions. This feature is particularly useful in object-oriented programming as it allows manipulation of class instances directly.

PROGRAM:

- a. WAP to create a rectangle class with data members length, breadth and color. Initialize length and breadth from constructor. Create a none member function paint that takes rectangle object and a color as arguments and returns the colored rectangle.

```
#include <iostream>
using namespace std;

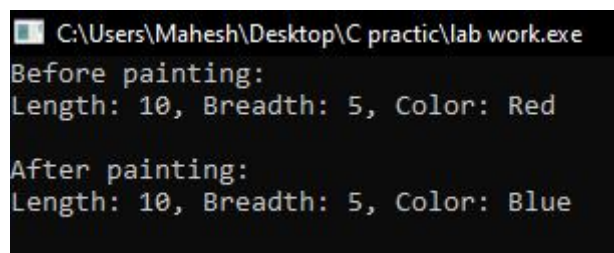
class Rectangle {
    int length; int breadth; string color;
public:
    Rectangle(int l, int b, string c): length(l), breadth(b), color(c) {}

    void display(){
        cout<<"Length: "<<length<<" , Breadth: "<<breadth<<" , Color: "
        "<<color<<endl<<endl;
    }
    void setColor(string c) {
        color = c;
    }
    string getColor() const {
        return color;
    }
};

Rectangle paint(Rectangle rect, string newColor) {
    rect.setColor(newColor);
    return rect;
}

int main() {
    Rectangle rect1(10, 5, "Red");
    cout << "Before painting:" << endl;
    rect1.display();
    Rectangle coloredRect = paint(rect1, "Blue");
    cout << "After painting:" << endl;
    coloredRect.display();
    return 0;
}
```

Output:



```
C:\Users\Mahesh\Desktop\C practic\lab work.exe
Before painting:
Length: 10, Breadth: 5, Color: Red

After painting:
Length: 10, Breadth: 5, Color: Blue
```

- b. Create Car and Driver classes and necessary data members. Write a drive function in car class so that when a driver object is passed as argument, it displays the information about the car condition according to driver skill.

```
#include <iostream>
using namespace std;

class Driver {
private:
    string name;    int skillLevel; // Skill level from 1 to 10
public:
    Driver(string n, int skill) : name(n), skillLevel(skill) {}

    int getSkillLevel() const {
        return skillLevel;
    }
    void display() const {
        cout << "Driver Name: " << name << ", Skill Level: " << skillLevel << endl;
    }
};

class Car {
    string model; int condition; // Condition from 1 to 100
public:
    Car(string m, int c) : model(m), condition(c) {}

    void drive(const Driver &driver) {
        cout << "Car Model: " << model << endl;
        driver.display();

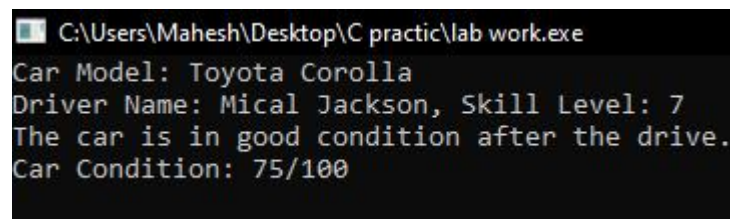
        if (driver.getSkillLevel() >= 8) {
            condition -= 5;
            cout << "The car is in excellent condition after the drive." << endl;
        } else if (driver.getSkillLevel() >= 5) {
            condition -= 15;
            cout << "The car is in good condition after the drive." << endl;
        } else {
            condition -= 30;
            cout << "The car is in poor condition after the drive." << endl;
        }
        cout << "Car Condition: " << condition << "/100" << endl;
    }
};

int main() {
    Driver driver1("Mical Jackson", 7);
    Car car1("Toyota Corolla", 90);

    car1.drive(driver1);

    return 0;
}
```

Output:



```
C:\Users\Mahesh\Desktop\C practic\lab work.exe
Car Model: Toyota Corolla
Driver Name: Mical Jackson, Skill Level: 7
The car is in good condition after the drive.
Car Condition: 75/100
```

- c. Create a ComplexNumber class and necessary members. Also create a member function add, that adds two complex numbers and returns a new complex number as a sum of two.

```
#include <iostream>
using namespace std;

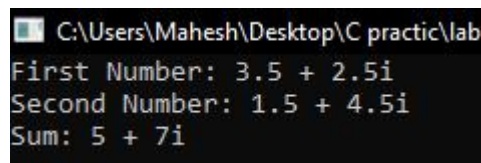
class ComplexNumber {
    double real;
    double imaginary;
public:
    ComplexNumber(double r = 0, double i = 0) : real(r), imaginary(i) {}

    ComplexNumber add(const ComplexNumber& other) const {
        return ComplexNumber(real + other.real, imaginary + other.imaginary);
    }

    void display() const {
        cout << real << " + " << imaginary << "i" << endl;
    }
};

int main() {
    ComplexNumber num1(3.5, 2.5); num1.display();
    ComplexNumber num2(1.5, 4.5); num2.display();
    ComplexNumber sum = num1.add(num2);
    cout << "Sum: ";
    sum.display();
    return 0;
}
```

Output:



```
C:\Users\Mahesh\Desktop\C practic\lab
First Number: 3.5 + 2.5i
Second Number: 1.5 + 4.5i
Sum: 5 + 7i
```

- d. Create a class Point with necessary data members. Write a function that takes two points as arguments and returns the mid point.

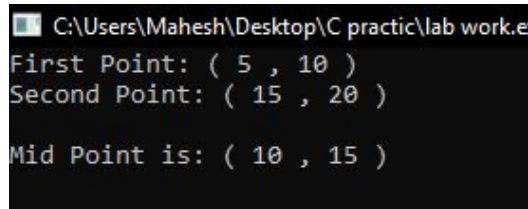
```
#include<iostream>
using namespace std;

class Point{
    double x,y;
public:
    Point(double p1,double p2): x(p1),y(p2){}
    void display(){
        cout<<"( "<<x<<" , "<<y<<" )"<<endl;
    }
    friend Point midPoint(Point,Point);
};

Point midPoint(Point p1,Point p2){
    return Point((p1.x + p2.x)/2,(p1.y + p2.y)/2);
}
```

```
int main(){
    Point p1(5,10), p2(15,20);
    Point mid = midPoint(p1,p2);
    cout<<"First Point: "; p1.display();
    cout<<"Second Point: "; p2.display();
    cout<<"\nMid Point is: "; mid.display();
    return 0;
}
```

Output:



```
C:\Users\Mahesh\Desktop\C practic\lab work.e
First Point: ( 5 , 10 )
Second Point: ( 15 , 20 )
Mid Point is: ( 10 , 15 )
```

CONCLUSION:

The concepts of passing objects as function arguments and returning objects from functions in C++ are fundamental to object-oriented programming. They allow us to create flexible, modular code by enabling functions to interact with and manipulate objects. This program provides a basic implementation of the concept passing object in function as argument and returning the object from function in C++. It serves as a useful example for understanding how to implement classes as blueprint of the program.

LAB 6

OBJECTIVE:

To illustrate the concept of function overloading in C++.

THEORY:

Function overloading in C++ is a feature that allows you to define multiple functions with the same name but different parameters. The compiler determines which function to call based on the number, type, and order of the arguments passed to the function.

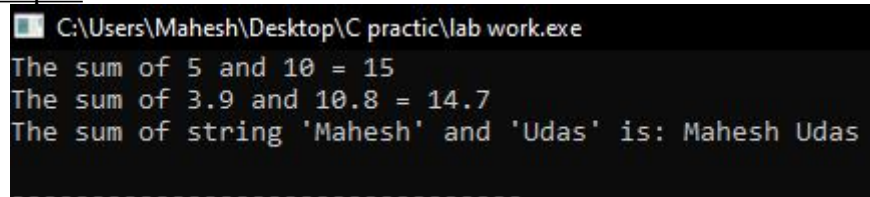
PROGRAMS:

- a. WAP to overload a function sum() that can add two integers, two floats and two strings.

```
#include<iostream>
using namespace std;

int sum(int n1,int n2){
    return n1+n2;
}
float sum(float n1,float n2){
    return n1+n2;
}
string sum(string s1,string s2){
    return s1+" "+s2;
}
int main(){
    cout<<"The sum of 5 and 10 = "<<sum(5,10)<<endl;
    cout<<"The sum of 3.9 and 10.8 = "<<sum(3.9f,10.8f)<<endl;
    cout<<"The sum of string 'Mahesh' and 'Udas' is:
"<<sum("Mahesh","Udas")<<endl;
    return 0;
}
```

Output:



```
C:\Users\Mahesh\Desktop\C practic\lab work.exe
The sum of 5 and 10 = 15
The sum of 3.9 and 10.8 = 14.7
The sum of string 'Mahesh' and 'Udas' is: Mahesh Udas
```

- b. WAP to overload constructor using any class of your choice.

```
#include<iostream>
using namespace std;
class Shape{
    double length, width;
    string shapeName;

public:
    Shape(double l): length(l),width(l){
        shapeName = "Square";
    }
}
```

```

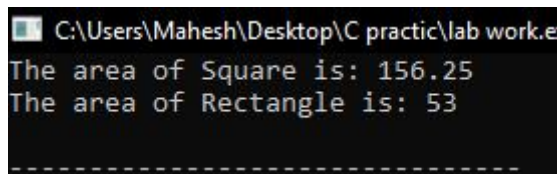
        Shape(double l,double w): length(l),width(w){
            shapeName = "Rectangle";
        }

        void displayArea(){
            cout<<"The area of "<<shapeName<<" is: "<<length*width<<endl;
        }
};

int main(){
    Shape square(12.5), rectangle(5,10.6);
    square.displayArea();
    rectangle.displayArea();
    return 0;
}

```

Output:



```

C:\Users\Mahesh\Desktop\C practic\lab work.e
The area of Square is: 156.25
The area of Rectangle is: 53
-----

```

CONCLUSION :

Function overloading in C++ lets you use the same function name for different tasks by changing the types or number of inputs. It makes your code simpler and easier to understand because you can use one name to do similar but slightly different things

In this lab work, we have understood how to implement function overloading in C++ programming. Students have get well knowledge of concept of constructor in C++ programming language.

LAB 7

OBJECTIVE:

To illustrate the concept of unary and binary operator overloading in C++.

THEORY:

In C++, Operator overloading is a compile-time polymorphism. It is an idea of giving special meaning to an existing operator in C++ without changing its original meaning.

C++ has the ability to provide the operators with a special meaning for a data type, this ability is known as operator overloading. Operator overloading is a compile-time polymorphism. For example, we can overload an operator '+' in a class like String so that we can concatenate two strings by just using +. Other example classes where arithmetic operators may be overloaded are Complex Numbers, Fractional Numbers, Big integers, etc.

PROGRAMS:

- a. WAP to overload the operator ++.

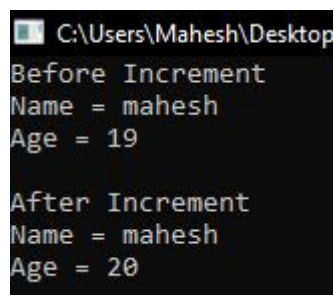
```
#include<iostream>
using namespace std;

class Person{
    int age; string name;
public:
    Person(int a,string n): age(a),name(n){}
    void operator ++(int){ //operator overloaded to increment age by 1.
        age++;
    }
    void display(){
        cout<<"Name = "<<name<<"\nAge = "<<age<<endl;
    }
};

int main(){
    Person p1(19,"mahesh");
    cout<<"Before Increment"<<endl;
    p1.display();
    p1++;
    cout<<"\nAfter Increment"<<endl;
    p1.display();

    return 0;
}
```

Output:



The screenshot shows a terminal window with the following output:

```
C:\Users\Mahesh\Desktop
Before Increment
Name = mahesh
Age = 19

After Increment
Name = mahesh
Age = 20
```

- b. WAP to overload the operator < to compare two person based on their age.

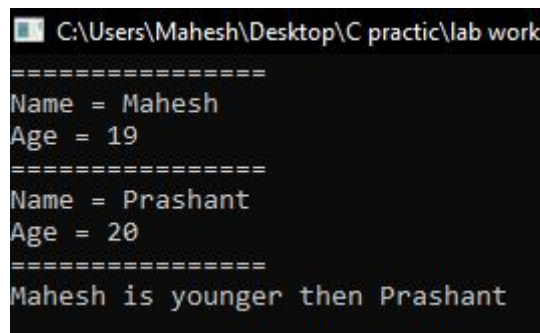
```
#include<iostream>
using namespace std;

class Person{
    int age; string name;
public:
    Person(int a,string n): age(a),name(n){}
    void operator <(Person p){ //operator overloaded to compare age.
        if(age<p.age) cout<<name<<" is younger than
" <<p.name<<endl;
        else cout<<p.name<<" is younger than " <<name<<endl;
    }
    void display(){
        cout<<"Name = " <<name<<"\nAge = " <<age<<endl;
    }
};

int main(){
    Person p1(19,"Mahesh"),p2(20,"Prashant");
    cout<<"===== "<<endl;
    p1.display();
    cout<<"===== "<<endl;
    p2.display();
    cout<<"===== "<<endl;
    p1<p2;

    return 0;
}
```

Output:



```
C:\Users\Mahesh\Desktop\C practic\lab work
=====
Name = Mahesh
Age = 19
=====
Name = Prashant
Age = 20
=====
Mahesh is younger than Prashant
```

CONCLUSION :

Operator overloading in C++ allows us to define how operators like + and - work with custom objects, making your code more intuitive and easier to read. It enables your objects to interact naturally, like built-in types, which improves code readability, reusability, and consistency. This feature is essential for writing clean, object-oriented code that feels familiar and logical to use.

In this lab work, we have understood how to implement operator overloading in C++ programming. Students have got well knowledge of concept of constructor in C++ programming language.