

**IGDTUW**  
**OBJECT ORIENTED PROGRAMMING**  
**LAB FILE**

**Submitted To:**

Ms. Ankita

**Submitted By:**

Suhani Jain

I.T.-1 (Group-H)

Roll No.-06001032021

## INDEX

S.NO	TOPIC	DATE	SIGNATURE
1.	Prime Number with Flowchart	3-01-2023	
2.	Factorial of a Number with Flowchart	3-01-2023	
3.	Print Hello World	10-01-2023	
4.	Prime Number	10-01-2023	
5.	Factorial of a number	10-01-2023	
6.	Print Fibonacci Series	10-01-2023	
7.	Perform Arithmetic Operations	10-01-2023	
8.	Read Employee Data	17-01-2023	
9.	Check Perfect Number	17-01-2023	
10.	Convert Celsius to Fahrenheit and vice versa	17-01-2023	
11.	Call By value Call By Reference Call By Pointer	17-01-2023	
12.	Calculate area and Perimeter for geometrical figure	17-01-2023	
13.	Check palindrome string	24-01-2023	
14.	Different ways of object creation	24-01-2023	
15.	Inline functions and default arguments	24-01-2023	
16.	Use of friend function of one class, Member function as friend function, Friend function of two class	24-01-2023	
17.	Using member functions	24-01-2023	
18.	Operator Overloading Complex numbers	31-01-2023	
19.	Unary Operators overloading	31-01-2023	
20.	Overload Subscript, Function Call and Assignment Operator	31-01-2023	

21.	Creating Time class using Operator Overloading	31-01-2023	
22.	Vehicle Car Motorcycle class	7-02-2023	
23.	Inheritance	7-02-2023	
24.	Public,private,protected derivation using inheritance	7-02-2023	
25.	B.Tech MCA class	7-02-2023	
26.	Date class using operator overloading	7-02-2023	
27.	Virtual base class	15-02-2023	
28.	Pure virtual function (area eg)	15-02-2023	
29.	Object array of students	15-02-2023	
30.	Overriding(game eg)	15-02-2023	

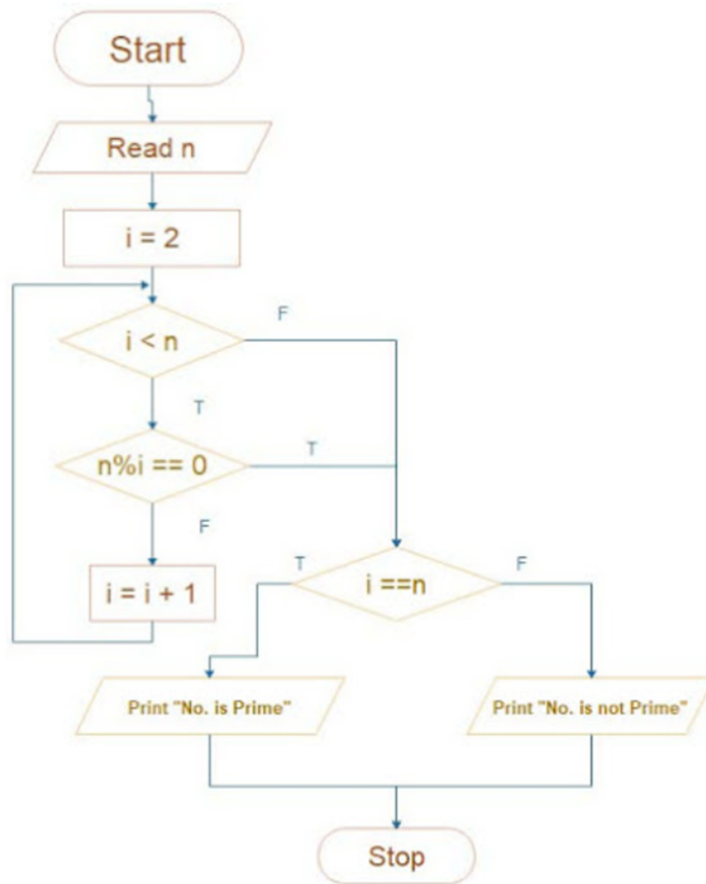
## **LAB EXERCISE-1**

### **1.PRIME NUMBER:**

#### **Algorithm:**

Step 1: Take a number n from user  
 Step 2: Initialize variables as flag=0, i=2 and m=n/2  
 Step 3: Repeat until i<m  
     if(n%i ==0)  
         print "number is not prime"  
         flag=1  
 Step 4: if(flag ==0)  
     Print " number is prime"  
 Step 5: if(n==2)  
     Print "It is a prime number"

#### **Flowchart:**



## 2.FACTORIAL NUMBER:

### Algorithm:

factorial(n)

Step 1: If  $n == 1$  then return 1

Step 2: Else

$f = n * \text{factorial}(n-1)$

Step 3: Return f

Program code

Step 1: Start

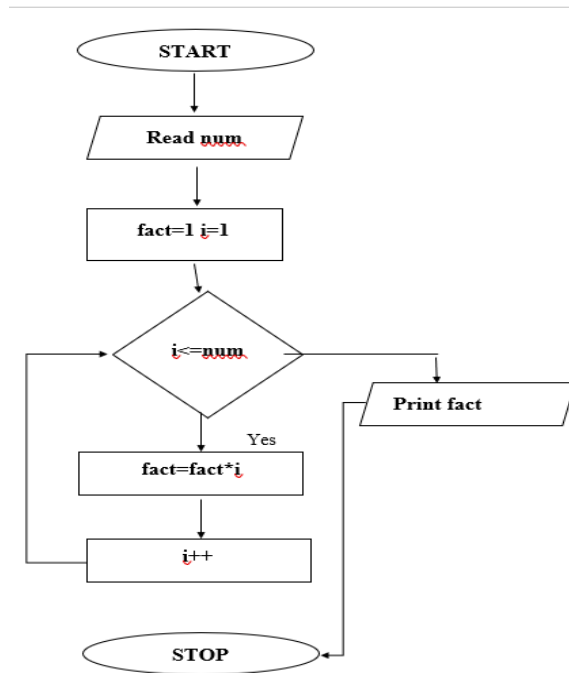
Step 2: Read number n

Step 3: Call factorial(n)

Step 4: Print factorial f

Step 5: Stop

## Flowchart:



## LAB EXERCISE-2

### 1. Write a program to print Hello World in C++

#### Code:

```
#include<iostream>
using namespace std;
int main(){
cout<<"Hello World" <<endl;
return 0;
}
```

#### Output:

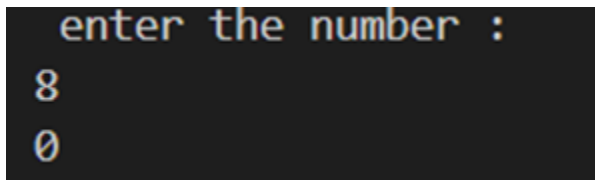
A screenshot of a terminal window with a black background. The text 'Hello World' is displayed in a light blue, monospaced font. Above the text, there are some faint, partially visible characters from the previous line of output.

## 2. Write a program to check whether the number is prime or not

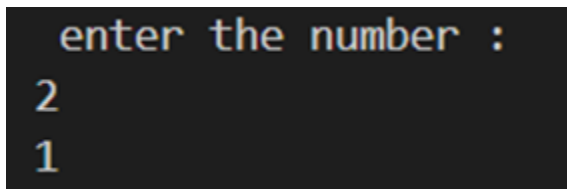
### Code:

```
#include<iostream>
using namespace std;
bool isprime(int n)
{
    if(n==1) return false;
    if(n==2) return true;
    for(int i=2;i<n;i++)
    {
        if(n%i==0) return false;
    }
    return true;
}
int main(){
    int n;
    cout<< " enter the number : " << endl;
    cin>>n;
    bool ans=isprime(n);
    cout<<ans;
    return 0;}
```

### Output:



A screenshot of a terminal window showing the program's output. The prompt "enter the number :" is displayed in a light blue font. The user has entered the number "8", which is shown in a light green font. The program's output, "0", is shown in a light red font, indicating that 8 is not a prime number.



A screenshot of a terminal window showing the program's output. The prompt "enter the number :" is displayed in a light blue font. The user has entered the number "2", which is shown in a light green font. The program's output, "1", is shown in a light red font, indicating that 2 is a prime number.

## 3. Write a program to find the factorial of a given number.

### Code:

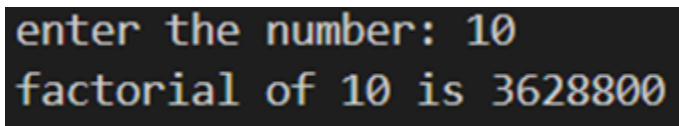
```
#include<iostream>
using namespace std ;
```

```

int main (){
    int n ;
    cout<<"enter the number: " ;
    cin>> n;
    int fact=1;
    if (n==1 || n==0){
        fact=1;
    }
    for(int i=1;i<=n;i++){
        fact=fact*i;
    }
    cout<< "factorial of " << n <<" is " << fact << endl;
    return 0;
}

```

### Output:



```

enter the number: 10
factorial of 10 is 3628800

```

## 4. Write a program to print Fibonacci Series(0,1,1,2,3,5,8....)

### Code:

```

#include<iostream>
using namespace std ;
int fib (int n){
    if(n==0){
        return n;
    }
    if(n==1)
    {
        return n;
    }
    else{
        return fib(n-1)+ fib(n-2);
    }
}
int main(){
    int no; int i=0;
    cout<<"enter the value of n ";

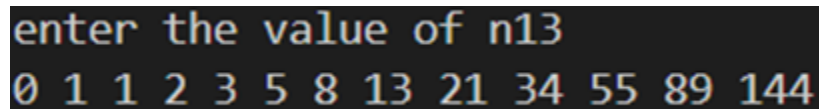
```

```

cin>>no;
while(i< no){
cout<< fib(i)<<" ";
I++;}
return 0;}

```

### Output:



```

enter the value of n13
0 1 1 2 3 5 8 13 21 34 55 89 144

```

### Algorithm:

fib(n)

Step 1: If n==0 then return n

Step 2: If n==1 then return n

Step 3: Else

Return fib(n-1)+fib(n-2)

Program code

Step 1: Start

Step 2: Read number n and initialize a variable i with 0

Step 3: while(i<n) print and call the function fib(i) and increment i by 1

Step 4: Stop

## 5. Write a program to perform arithmetic operations on two numbers.

### Code:

```

#include<iostream>
using namespace std ;
int main (){
    int n ,m;
    cout << "enter the 1 number: ";
    cin>> n;
    cout<<"enter the 2 number: " ;
    cin>>m;
    int sum =n+m;
    int sub=n-m;
    int prod= n*m;
}

```



```
int div=n/m;
cout<<"sum is " << sum << endl;
cout<<"sub is " << sub << endl;
cout<<"product is " << prod << endl;
cout<<"division is " << div << endl;
return 0;
}
```

### Output:

```
enter the 1 number: 8
enter the 2 number: 4
sum is 12
sub is 4
product is 32
division is 2
```

### Algorithm:

- Step 1: Start
- Step 2: Read two numbers n and m from user
- Step 3: Initialize variable sum with n+m and print sum
- Step 4: Initialize variable sub with n-m and print sub
- Step 5: Initialize variable prod with n\*m and print prod
- Step 6: Initialize variable div with n/m and print div

## **LAB EXERCISE-3**

- 1. Write a program to read Employee information ID(int), employee name(string) and employee salary(float).**

### Code:

```
#include<iostream>
#include<string.h>
using namespace std;
int main(){
    int empid;
```

```

    string empname;
    float empsal;
    cout<<"Enter employee ID: ";
    cin>>empid;
    cout<<"Enter employee name: ";
    cin>>empname;
    cout<<"Enter employee salary: ";
    cin>>empsal;
    cout<<endl;
    cout<<"Employee ID: "<<empid<<endl;
    cout<<"Employee name: "<<empname<<endl;
    cout<<"Employee salary: "<<empsal<<endl;
    return 0;
}

```

### Output:

```

Enter employee ID:
123
Enter employee name:
Preeti
Enter employee salary:
1000000
Employee ID: 123
Employee name: Preeti
Employee salary: 1e+06
|

```

## 2. Write a program to show whether a number is perfect number or not .

### Code:

```

#include<iostream>
using namespace std;
int main(){
    int num, div, sum=0;
    cout<<"Enter a number"<<endl;
    cin>>num;
    div=num;
    while(--div){
        if(num%div==0) sum+=div;
    }
}

```

```

if(sum==num)
    cout<<"Perfect Number"<<endl;
else
    cout<<"Not perfect number"<<endl;
return 0;
}

```

### Output:

```

Enter a number
6
Perfect Number

```

### 3. Write a program to convert a) temperature from Celsius to Fahrenheit b) Height from centi-meter to feet and inches and vice versa.

#### a) Code:

```

#include <iostream>
using namespace std;
int tempconvert(int temp,char d)
{
    int converted_temp;
    if(d=='F' || d=='f')
    {
        converted_temp=5*(temp-32)/9;
    }
    else converted_temp=9*temp/5 + 32;
    return converted_temp;
}
int main() {
    char scale; int temp;
    cout<<"Enter the scale for measuring temperature: ";
    cin>>scale;
    cout<<"enter temperature: ";
    cin>>temp;
    cout<<scale;
}

```

```

cout<<"\nthe converted temperature is: ";
temp=tempconvert(temp,scale);
if(scale=='F') scale='C';
else scale='F';
cout<<temp<<" "<<scale;
return 0;}

```

### Output:

```

Enter the scale for measuring temperature: F
enter temperature: 50
F
the converted temperature is: 10 C

```

### b) Code:

```

#include <iostream >
using namespace std;
int main (){
    int n;
    cout<<"enter the length in cm ";
    cin>> n;
    float feet= n/30.48;
    cout<<n<<"cm in feet is " <<feet <<endl;
    float inches =n*0.394;
    cout<<n<<"cm in inches is " <<inches <<endl;
    return 0;
}

```

### Output:

```

/tmp/p3bv1UfpIA.o
enter the length in cm 154
154cm in feet is 5.05249
154cm in inches is 60.676
|

```

**4. Write a program to swap two numbers using call by value, call by reference and pointer.**

## CALL BY VALUE

### Code:

```
#include <iostream >
using namespace std;
void swap(int i,int j){
    int temp;
    temp=i;
    i=j;
    j=temp;
}
int main (){
    int n,m;
    cout<<"enter the first number: ";
    cin>> n;
    cout<<"enter the second number: ";
    cin>>m;
    cout<<"numbers before swapping :" << n<<" " << m<< endl;
    swap(n,m);
    cout<<"numbers after swapping :" << n <<" "<< m<< endl;
    return 0;
}
```

### Output:

```
enter the first number: 10
enter the second number: 20
numbers before swapping :10 20
numbers after swapping :10 20
```

## CALL BY REFERENCE

### Code:

```
#include <iostream>
using namespace std;
void swap(int& x, int& y)
{
    int z = x;
    x = y;
    y = z;
}
int main()
```

```

{
    int a = 45, b = 35;
    cout << "Before Swap\n";
    cout << "a = " << a << " b = " << b << "\n";

    swap(a, b);

    cout << "After Swap with pass by reference\n";
    cout << "a = " << a << " b = " << b << "\n";
    return 0;}

```

### Output:

```

Before Swap
a = 45 b = 35
After Swap with pass by reference
a = 35 b = 45

```

## CALL BY POINTER

### Code:

```

#include <iostream>
using namespace std;

void swap(int *x, int *y)
{
    int z = *x;
    *x = *y;
    *y = z;
}

int main()
{
    int a = 45, b = 35;
    cout << "Before Swap\n";
    cout << "a = " << a << " b = " << b << "\n";

    swap(&a, &b);

    cout << "After Swap with pass by pointer\n";
    cout << "a = " << a << " b = " << b << "\n";
    return 0;}

```

### Output:

```
Before Swap
a = 45 b = 35
After Swap with pass by pointer
a = 35 b = 45
```

## 5. Write a program to calculate area and perimeter of the geometric figures. Use function overloading.

### Code:

```
//Area
#include<iostream>
using namespace std;
void area(int a)
{
    int square=a*a;
    cout<<"The Area Of Square Value is:"<<square<<endl;
}
void area(int l,int b)
{
    int rectangle=l*b;
    cout<<" The Area Of The Rectangle Value is:"<<rectangle<<endl;
}
void area(float r)
{
    float circle=2.34*r*r;
    cout<<"The Area of The Circle Value is:"<<circle<<endl;
}
void area (float le,float br)
{
    float triangle=0.5*le*br;
    cout<<" The Area Of the triangle value is:"<<triangle<<endl;
}
int main()
{
    int a,b,l;
    float r,le,br;
    cout<<"AREA OF SHAPES USING FUNCTION OVERLOADING"<<endl;
    cout<<"Enter value of side for Square:";
```

```

cin>>a;
cout<<endl;
    cout<<"Enter breadth and length for Rectangle:";
cin>>b>>l;
cout<<endl;
cout<<"Enter radius for Circle:";
cin>>r;
cout<<endl;
cout<<"Enter height and base for triangle:";
    cin>>le>>br;
cout<<endl;
cout<<" ____ "<<endl;
area(a);
area(l,b);
area(r);
area(le,br);
cout<<" ____ "<<endl;
return 0;
}

```

### **//Perimeter**

```

#include<iostream>
using namespace std;
void peri(int a)
{
    int square=4*a;
    cout<<"The perimeter Of Square Value is:"<<square<<endl;
}
void peri(int l,int b)
{
    int rectangle=2*(l+b);
    cout<<" The Perimeter Of The Rectangle Value is:"<<rectangle<<endl;
}
void peri(float r)
{
    float circle=2* 3.14*r;
    cout<<"The Circumference of The Circle Value is:"<<circle<<endl;
}
void peri(float s1,float s2, float s3)
{

```



```

float triangle= s1+s2+s3;
cout<<" The Perimeter Of the triangle value is:"<<triangle<<endl;
}
int main()
{
    int a,b,l;
    float r,s1,s2,s3;
    cout<<"PERIMETER OF SHAPES USING FUNCTION OVERLOADING"<<endl;
    cout<<"Enter the value of side for Square:";
    cin>>a;
    cout<<endl;
    cout<<"Enter breadth and length for Rectangle:";
    cin>>b>>l;
    cout<<endl;
    cout<<"Enter radius for Circle:";
    cin>>r;
    cout<<endl;
    cout<<"Enter sides of the triangle:";
    cin>>s1>>s2>>s3;
    cout<<endl;
    cout<<" ____ "<<endl;
    peri(a);
    peri(l,b);
    peri(r);
    peri(s1,s2,s3);
    cout<<" ____ "<<endl;
    return 0;
}

```

**Output:**

```
AREA OF SHAPES USING FUNCTION OVERLOADING
Enter value of side for Square:4
Enter breadth and length for Rectangle:4 3
Enter radius for Circle:2.5
Enter height and base for triangle:3 8
```

```
_____
The Area Of Square Value is:16
The Area Of The Rectangle Value is:12
The Area of The Circle Value is:14.625
The Area Of the triangle value is:12
_____
```

```
PERIMETER OF SHAPES USING FUNCTION OVERLOADING
Enter the value of side for Square: 4
4
Enter breadth and length for Rectangle: 4 3
4 3
Enter radius for Circle:2.5
Enter sides of the triangle:5 6 7
```

```
_____
The perimeter Of Square Value is:16
The Perimeter Of The Rectangle Value is:14
The Circumference of The Circle Value is:11.7
The Perimeter Of the triangle value is:18
_____
```

## **LAB EXERCISE-4**

- 1. Write a program that reverses a given string and checks whether it is palindrome or not. Write the recursive functions for the above program .**

### **Code:**

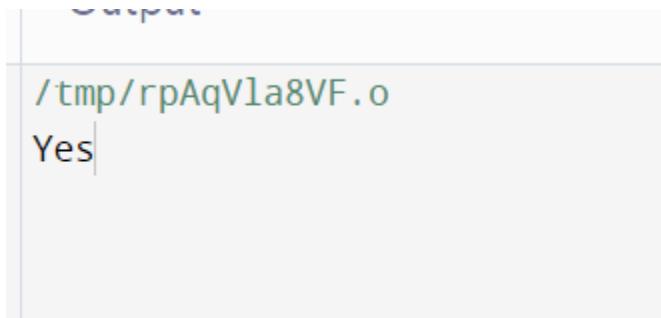
```
#include <bits/stdc++.h>
using namespace std;
bool isPalRec(char str[], int s, int e){
    if (s == e)    return true;
    if (str[s] != str[e])    return false;
    if (s < e + 1)    return isPalRec(str, s + 1, e - 1);
```

```

        return true;
    }
    bool isPalindrome(char str[]){
        int n = strlen(str);
        if (n == 0)    return true;
        return isPalRec(str, 0, n - 1);
    }
    int main(){
        char str[] = "geeg";
        if (isPalindrome(str))    cout << "Yes";
        else    cout << "No";
        return 0;
    }

```

### Output:



## 2. What are the different ways for objects of class to be created? Explain with illustrative examples.

### Code:

#### 1.Through Default constructors

```

#include <bits/stdc++.h>
using namespace std;
class example {
    int x;
public:
    void set(int x){
        this->x = x;
        cout << "The value of x is: "

```

```

        << x << "\n";
    }
};
int main(){
    example obj1;
    obj1.set(12);
    example* obj2 = new example();
    obj2->set(16);
    delete obj2;
    return 0;
}

```

### Output:

```

/tmp/rpAqVla8VF.o
The value of x is: 12
The value of x is: 16

```

## 2.Through Parameterized Constructor

### Code:

```

#include <bits/stdc++.h>
using namespace std;
class example {
    int x;
public:
    example(int y){
        x = y;
        cout << "The value of x is: "
             << x << "\n";
    }
};
int main(){
    example obj1(10);
    example obj2 = example(50);
}

```

```
    example* obj3 = new example(6);
    delete obj3;
    return 0;
}
```

## Output:

```
/tmp/PUWrHR5zV4.o
The value of x is: 10
The value of x is: 50
The value of x is: 6
```

## 3. Copy constructor

### Code:

```
#include <bits/stdc++.h>
using namespace std;
class example {
    int x;
public:
    example(int y){
        x = y;
        cout << "The value of x is: "
              << x << "\n";
    }
    example(example& obj){
        x = obj.x;
        cout << "The value of x in "
              << "copy constructor: "
              << x << "\n";
    };
};

int main(){
    example obj1(4);
    example obj2 = obj1;
    return 0;
}
```

## Output:

```
/tmp/PUWrHR5zV4.o
The value of x is: 4
The value of x in copy constructor: 4
```

## 3. Write a program to illustrate use of inline function and function with default argument.

### Code:

#### Inline Function

##### // Without using class

```
#include <iostream>
using namespace std;
inline int cube(int s) {
    return s*s*s;
}
int main() {
    cout << "The cube of 5 is: " << cube(5) << "\n";
    return 0;
}
```

##### //Using class

```
include <iostream>
using namespace std;
class operation
{
    int a,b,add,sub,mul;
    float div;
public:
    void get();
    void sum();
    void difference();
    void product();
    void division();
};
inline void operation :: get()
{
    cout << "Enter first value:";
```

```

    cin >> a;
    cout << "Enter second value:";
    cin >> b;
}
inline void operation :: sum()
{
    add = a+b;
    cout << "Addition of two numbers: " << a+b << "\n";
}
inline void operation :: difference()
{
    sub = a-b;
    cout << "Difference of two numbers: " << a-b << "\n";
}
inline void operation :: product() {
    mul = a*b;
    cout << "Product of two numbers: " << a*b << "\n";
}
inline void operation ::division()
{
    div=a/b;
    cout<<"Division of two numbers: "<<a/b<<"\n" ;
}
int main()
{
    cout << "Program using inline function\n";
    operation s;
    s.get();
    s.sum();
    s.difference();
    s.product();
    s.division();
    return 0;
}

```

## Output:

**//Without using class**

```

/tmp/PUWrHR5zV4.o
The cube of 5 is: 125

```

**// Using class**

```
Program using inline function
Enter first value:2
Enter second value:5
Addition of two numbers: 7
Difference of two numbers: -3
Product of two numbers: 10
Division of two numbers: 0
|
```

## Function With Default Argument

### Code:

```
#include <iostream>
using namespace std;
int sum(int x, int y, int z = 0, int w = 0){
    return (x + y + z + w);
}
int main(){
    cout << sum(10, 12) << endl;
    cout << sum(10, 15, 26) << endl;
    cout << sum(10, 11, 23, 32) << endl;
    return 0;
}
```

### Output:

```
/tmp/PUWrHR5zV4.o
22
51
76
|
```

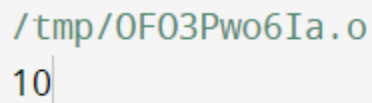
**4. Write a program to illustrate: Use of friend function of one class, Member function as friend function, Friend function of two class .**

### Use of friend function of one class



**Code:**

```
#include <iostream>
using namespace std;
class Student {
    private:
        int rollno;
    public:
        void set_Rollno(int r);
        friend void display(Student s);
};
void Student :: set_Rollno(int r){
    rollno=r;
}
void display(Student s){
    cout<<s.rollno;
}
int main() {
    Student ob1;
    ob1.set_Rollno(10);
    display(ob1);
    return 0;
}
```

**Output:**

```
/tmp/0F03Pwo6Ia.o
10
```

**Member function as friend function****Code:**

```
#include <iostream>
using namespace std;
class truck;
class car{
    private:
        int passenger;
        int speed;
    public:
```

```

        void set(int p, int s);
        int sp_greater(truck t);
};
void car :: set(int p, int s){
    passenger = p;
    speed = s;
}

class truck{
    int weight;
    int speed;
public:
    void set(int w, int s);
    friend int car :: sp_greater(truck t);
};
void truck :: set(int w, int s){
    weight = w;
    speed = s;
}

int car :: sp_greater(truck t){
    return speed - t.speed;
}

int main()  {
    car ob1;
    int s, p;
    cout<<"Enter no of passengers and the speed of the car"<<endl;
    cin>>p>>s;
    ob1.set(p, s);
    truck ob2;
    int s1, w;
    cout<<"Enter weight and the speed of truck"<<endl;
    cin>>w>>s1;
    ob2.set(w, s1);
    cout<<"Speed difference: "<<ob1.sp_greater(ob2);
    return 0;
}

```

**Output:**

```
/tmp/0F03Pwo6Ia.o
Enter no of passengers and the speed of the car
5 120
Enter weight and the speed of truck
90 60
Speed difference: 60|
```

## Friend function of two class

### Code:

```
#include <iostream>
using namespace std;
class truck;
class car{
    private:
        int passenger;
        int speed;
    public:
        void set(int p, int s);
        friend int sp_greater(car c, truck t);
};
void car :: set(int p, int s){
    passenger = p;
    speed = s;
}

class truck{
    int weight;
    int speed;
    public:
        void set(int w, int s);
        friend int sp_greater(car c, truck t);
};
void truck :: set(int w, int s){
    weight = w;
    speed = s;
}

int sp_greater(car c, truck t){
```

```

        return c.speed - t.speed;
    }

int main() {
    car ob1;
    int s, p;
    cout<<"Enter no of passengers and the speed of the car"<<endl;
    cin>>p>>s;
    ob1.set(p, s);
    truck ob2;
    int s1, w;
    cout<<"Enter weight and the speed of truck"<<endl;
    cin>>w>>s1;
    ob2.set(w, s1);
    cout<<"Speed difference: "<<sp_greater(ob1, ob2);
    return 0;
}

```

### Output:

```

/tmp/0F03Pwo6Ia.o
Enter no of passengers and the speed of the car
3 60
Enter weight and the speed of truck
100 40
Speed difference: 20

```

## 5. Design suitable class and write member functions for, Insert, search and modify details of customer.

### Code:

```

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

class customer{
    int n;

```

```

int* id_arr;
string* name;
int last_pos;
public:
customer(int n){
    this->n=n;
    id_arr= new int[n];
    name= new string[n];
    last_pos=0;
}
void insert(pair<int, string> p){
    if(last_pos<n){
        id_arr[last_pos]= p.first;
        name[last_pos]= p.second;
        last_pos++;
    }
}
void search(int id){
    unordered_map <int,string> mp;
    for(int i=0; i<last_pos; i++){
        mp[id_arr[i]]= name[i];
    }
    if(mp[id]!=""){
        string str= mp[id];
        cout<<"The customer exists with name "<<str<< endl;
    }
    else
        cout<<"The customer doesn't exist"<<endl;
}
void modify(int id, string s){
    unordered_map <int,int> mp;
    for(int i=0; i<last_pos; i++){
        mp[id_arr[i]]= i;
    }
    int index= mp[id];
    name[index]= s;
}
void display(){
    for(int i=0; i< last_pos; i++){

```

```

        cout<<"ID is: "<< id_arr[i]<<" ";
        cout<<"Name is: "<< name[i]<<endl;
    }

}

};

int main()
{
    customer obj(5);
    obj.insert({1001, "Ayushi"});
    obj.insert({1002, "Suhani"});
    obj.insert({1003, "Deeksha"});
    obj.insert({1004, "Vani"});
    obj.insert({1005, "Tasbiya"});
    obj.display();
    obj.search(1005);
    obj.search(1009);
    obj.modify(1001,"Pratiksha");
    obj.display();
    return 0;
}

```

## Output:

```

/tmp/uD9UbyHOIU.o
ID is: 1001, Name is: Ayushi
ID is: 1002, Name is: Suhani
ID is: 1003, Name is: Deeksha
ID is: 1004, Name is: Vani
ID is: 1005, Name is: Tasbiya
The customer exists with name Tasbiya
The customer doesn't exist
ID is: 1001, Name is: Pratiksha
ID is: 1002, Name is: Suhani
ID is: 1003, Name is: Deeksha
ID is: 1004, Name is: Vani
ID is: 1005, Name is: Tasbiya

```

## **LAB EXERCISE-5**

**1. Write a program which provides concrete representation for the concept of complex numbers. Using Operator overloading, perform the operation of addition and subtraction using friend and member functions.**

**Using Member function**

**Code:**

```
#include <iostream>
using namespace std;
class ComplexNumbers {
    int real;
    int imaginary;
public:
    ComplexNumbers(int real,int imaginary)
    {
        this->real=real;
        this->imaginary=imaginary;
    }
    void print()
    {
        cout<<real<<" "<<"+"<<" "<<"i"<<imaginary;
    }
    void plus(ComplexNumbers const &c2)
    {
        real=real+c2.real;
        imaginary=imaginary+c2.imaginary;
    }
    void minus(ComplexNumbers const &c2)
    {
        real=real-c2.real;
        imaginary=imaginary-c2.imaginary;
    }
};
int main() {
    int real1, imaginary1, real2, imaginary2;
```

```

cin >> real1 >> imaginary1;
cin >> real2 >> imaginary2;
ComplexNumbers c1(real1, imaginary1);
ComplexNumbers c2(real2, imaginary2);
int choice;
cin >> choice;
if(choice == 1) {
    c1.plus(c2);
    c1.print();
}
else if(choice == 2) {
    c1.minus(c2);
    c1.print();
}
else {
    return 0;
}
}

```

### Output:

```

5 8
4 7
1
9 + i15

```

```

5 8
4 7
2
1 + i1

```

### Using Friend function

#### Code:

```

#include<iostream>
using namespace std;
class complex
{
    float a,b;
public:
    void get()
    {
        cout<<"Enter value for A="<<" ";
        cin>>a;
    }
}

```



```

        cout<<"Enter value for B="<<" ";
        cin>>b;
    }
    void disp()
    {
        cout<<a<<"+"<<"i"<<b<<endl;
    }
    friend complex sum(complex,complex);
    friend complex sub(complex,complex);
};
complex sum(complex c1,complex c2)
{
    complex c3;
    c3.a=c1.a+c2.a;
    c3.b=c1.b+c2.b;
    return c3;
}
complex sub(complex c1,complex c2)
{
    complex c3;
    c3.a=c1.a-c2.a;
    c3.b=c1.b-c2.b;
    return c3;
}
int main()
{
    complex x,y,z,k;
    x.get();
    x.disp();
    y.get();
    y.disp();
    z=sum(x,y);
    z.disp();
    k=sub(x,y);
    k.disp();
    return 0;
}

```

**Output:**

```
Enter value for A= 6
Enter value for B= 8
6+i8
Enter value for A= 9
Enter value for B= 3
9+i3
15+i11
-3+i5
```

**2. Extend the concept of complex number by providing provisions for unary operators + and – which increments and decrements both real and imaginary part of the complex number.**

### Using Friend function

**Code:**

```
#include <iostream>
using namespace std;
class Complex
{
    int a, b;
public:
    Complex()
    {
        a=b= 0;
    }
    Complex(int i, int j)
    {
        a = i;
        b = j;
    }
    void display() {
        cout << a << "+\t" << b << "i" << endl;
    }
    friend Complex operator ++ (Complex & op1);
    friend Complex operator ++ (Complex & op1, int not_used);
    friend Complex operator -- (Complex & op1);
```

```

        friend Complex operator -- (Complex & op1, int not_used);

};
Complex operator ++(Complex & op1)
{
    op1.a ++;
    op1.b ++;
    return op1;
}
Complex operator --(Complex & op1)
{
    op1.a --;
    op1.b --;
    return op1;
}
Complex operator ++ (Complex & op1, int not_used)
{
    Complex temp = op1;
    op1.a ++;
    op1.b ++;
    return temp;
}
Complex operator -- (Complex & op1, int not_used)
{
    Complex temp = op1;
    op1.a --;
    op1.b --;
    return temp;
}
int main()
{
    Complex a (12, 3);
    a.display();
    ++a; // prefix increment
    a.display();
    a++; // postfix increment
    a.display();
    --a;
    a.display();
    a--;
}

```

```

    a.display();
    return 0;
}

```

### Output:

```

12+ 3i
13+ 4i
14+ 5i
13+ 4i
12+ 3i

```

### Using Member function

#### Code:

```

#include<iostream>
using namespace std;
class complex {
    int real, imaginary, c;
public:
    complex() {
    }
    void getvalue() {
        cout << "Enter the real and imaginary part of Number:";
        cin >> real>>imaginary;
    }
    void operator++() {
        real = ++real;
        imaginary = ++imaginary;
    }
    void operator--() {
        real = --real;
        imaginary = --imaginary;
    }
    void operator++(int)
    {
        real=real++;
        imaginary=imaginary++;
    }
    void operator--(int) {

```

```

        real = real--;
        imaginary = imaginary--;
    }
    void display() {
        cout << real << "+\t" << imaginary << "i" << endl;
    }
};

int main() {
    complex obj;
    obj.getvalue();
    ++obj;
    cout << "Post Increment Complex Number\n";
    obj.display();
    obj.getvalue();
    obj++;
    cout << "Pre Increment Complex Number\n";
    obj.display();
    obj.getvalue();
    --obj;
    cout << " Post Decrement Complex Number\n";
    obj.display();
    obj.getvalue();
    obj--;
    cout << " Pre Decrement Complex Number\n";
    obj.display();
    return 0;}

```

## Output:

```

Enter the real and imaginary part of Number:2 3
Post Increment Complex Number
3+ 4i
Enter the real and imaginary part of Number:2 3
Pre Increment Complex Number
2+ 3i
Enter the real and imaginary part of Number:2 3
Post Decrement Complex Number
1+ 2i
Enter the real and imaginary part of Number:2 3
Pre Decrement Complex Number
2+ 3i

```

### 3. Write a program to overload the following operators: Subscript operator, Function Call operator and Assignment Operator.

**Code:**

#### **Subscript operator**

```
#include <iostream>
using namespace std;
int main()
{
    int intArray[1024];
    for (int i = 0, j = 0; i < 1024; i++) {
        intArray[i] = j++;
    }
    cout << intArray[512] << endl;
    cout << 257 [intArray] << endl;
    int* midArray = &intArray[512];
    cout << midArray[-256] << endl;
    cout << intArray[-256] << endl;
    return 0;
}
```

#### **Function Call operator**

```
#include <bits/stdc++.h>
#include <iostream>
using namespace std;
#define N 3
#define M 3
class Matrix {
private:
    int arr[N][M];
public:
    // Overloading of input stream
    friend istream& operator>>(istream&, Matrix&);
    // Overloading of output stream
    friend ostream& operator<<(ostream&, Matrix&);
    int& operator()(int, int);
};

// Function to overload the input
// ">>" operator
```

```

istream& operator>>(istream& cin, Matrix& m)
{
    int x;
    for (int i = 0; i < N; i++) {
        for (int j = 0; j < M; j++) {
            // Overloading operator()
            // to take input
            cin >> m(i, j);
        }
    }
    return cin;
}

ostream& operator<<(ostream& cout, Matrix& m)
{
    for (int i = 0; i < N; i++) {
        for (int j = 0; j < M; j++) {
            cout << m(i, j) << " ";
        }
        cout << endl;
    }
    return cout;
}

int& Matrix::operator()(int i, int j)
{
    return arr[i][j];
}

int main()
{
    Matrix m;
    printf("Input the matrix:\n");
    cin >> m;
    printf("The matrix is:\n");
    cout << m;
    return 0;
}

```

## Assignment Operator

```
#include <iostream>
```

```
using namespace std;
```

```
class Complex {
```

```
private:
```

```
    int real, img; // real, imaginary
```

```
public:
```

```
    // Parameterized Constructor
```

```
    Complex(int r, int i)
```

```
    {
```

```
        real = r;
```

```
        img = i;
```

```
    }
```

```
    // This is automatically called
```

```
    // when '=' operator is
```

```
    // used between C1 and C2.
```

```
    void operator=(const Complex& C)
```

```
    {
```

```
        real = C.real;
```

```
        img = C.img;
```

```
    }
```

```
// function to print
```

```
    void print() { cout << real << "+i" << img << endl; }
```

```
};
```

```
int main()
```

```
{
```

```
    // Assigning by overloading constructor
```

```
    Complex C1(7, 8), C2(9, 10);
```

```
    cout << "BEFORE OVERLOADING ASSIGNMENT OPERATOR" << endl;
```

```
    cout << "C1 complex number: ";
```

```
    C1.print();
```

```
    cout << "C2 complex number: ";
```

```
    C2.print();
```

```
    C1 = C2;
```

```
    cout << "AFTER OVERLOADING ASSIGNMENT OPERATOR" << endl;
```

```
    cout << "C1 complex number: ";
```

```
    C1.print();
```

```
    cout << "C2 complex number: ";
```

```
    C2.print();
```



```
    return 0;  
}
```

**Output:**

### **Subscript operator**

```
512  
257  
256  
-1663203456  
|
```

### **Function Call operator**

```
Input the matrix:  
1 2 3  
4 5 6  
7 8 9  
The matrix is:  
1 2 3  
4 5 6  
7 8 9  
|
```

### **Assignment Operator**

```
BEFORE OVERLOADING ASSIGNMENT OPERATOR  
C1 complex number: 7+i8  
C2 complex number: 9+i10  
AFTER OVERLOADING ASSIGNMENT OPERATOR  
C1 complex number: 9+i10  
C2 complex number: 9+i10  
|
```

**4. Create a Class in C++ which provides representation for the concept of Time(hour, minute, second). Using operator overloading perform the operations of unary increment and decrement of a**

**given Time. Implement operator overloading through friend function and member function.**

### **Using Friend function**

**Code:**

```
#include <iostream>
```

```
class Time {
```

```
    friend Time operator++(Time& t, int);
```

```
    friend Time operator--(Time& t, int);
```

```
public:
```

```
    Time(int h = 0, int m = 0, int s = 0) : hour(h), minute(m), second(s) {}
```

```
    Time operator++() {
```

```
        ++second;
```

```
        if (second >= 60) {
```

```
            second -= 60;
```

```
            ++minute;
```

```
        }
```

```
        if (minute >= 60) {
```

```
            minute -= 60;
```

```
            ++hour;
```

```
        }
```

```
        return *this;
```

```
    }
```

```
    Time operator--() {
```

```
        --second;
```

```
        if (second < 0) {
```

```
            second += 60;
```

```
            --minute;
```

```
        }
```

```
        if (minute < 0) {
```

```
            minute += 60;
```

```
            --hour;
```

```
        }
```

```
        return *this;
```

```
    }
```

```

    void displayTime() const {
        std::cout << hour << ":" << minute << ":" << second << std::endl;
    }

private:
    int hour, minute, second;
};

Time operator++(Time& t, int) {
    Time temp = t;
    ++t;
    return temp;
}

Time operator--(Time& t, int) {
    Time temp = t;
    --t;
    return temp;
}

int main() {
    Time t1(12, 59, 59);
    std::cout << "Original time: ";
    t1.displayTime();
    t1++;
    std::cout << "Incremented time: ";
    t1.displayTime();
    ++t1;
    std::cout << "Incremented time: ";
    t1.displayTime();
    t1--;
    std::cout << "Decrementing time: ";
    t1.displayTime();
    --t1;
    std::cout << "Decrementing time: ";
    t1.displayTime();
    return 0;
}

```

## Output:

```
Original time: 12:59:59
Incremented time: 13:0:0
Incremented time: 13:0:1
Decrement time: 13:0:0
Decrement time: 12:59:59
```

## Using Member function

### Code:

```
#include <iostream>
using namespace std;
class Time {
private:
    int hour;
    int minute;
    int second;
public:
    Time(int h = 0, int m = 0, int s = 0) {
        hour = h;
        minute = m;
        second = s;
    }
    void display() {
        cout << hour << ":" << minute << ":" << second << endl;
    }
    Time operator++() {
        ++second;
        if (second >= 60) {
            second -= 60;
            ++minute;
        }
        if (minute >= 60) {
            minute -= 60;
            ++hour;
        }
        return *this;
    }
    Time operator--() {
        --second;
```

```

        if (second < 0) {
            second += 60;
            --minute;
        }
        if (minute < 0) {
            minute += 60;
            --hour;
        }
        return *this;
    }
};

int main() {
    Time t(10, 30, 45);
    t.display(); // output: 10:30:45
    ++t;
    t.display(); // output: 10:30:46
    --t;
    t.display(); // output: 10:30:45
    --t;
    t.display(); // output: 10:30:44
    ++t; ++t; ++t;
    t.display(); // output: 10:30:47
    return 0;
}

```

### Output:

```

10:30:45
10:30:46
10:30:45
10:30:44
10:30:47

```

## **LAB EXERCISE-6**

**1. Write a C++ program that represents a Vehicle-Car and Vehicle-Motorcycle relationship. Also, add a vehicle and display the vehicles.**

**Code:**

```

#include <iostream>
#include <vector>
using namespace std;
class Vehicle {
protected:
    string brand;
    string model;
    int year;
public:
    Vehicle(string brand, string model, int year) {
        this->brand = brand;
        this->model = model;
        this->year = year;
    }
    void display() {
        cout << "Brand: " << brand << ", Model: " << model << ", Year: " << year << endl;
    }
};

class Car : public Vehicle {
private:
    int num_doors;

public:
    Car(string brand, string model, int year, int num_doors) : Vehicle(brand, model, year) {
        this->num_doors = num_doors;
    }

    void display() {
        cout << "Brand: " << brand << ", Model: " << model << ", Year: " << year << ", Number of
doors: " << num_doors << endl;
    }
};

class Motorcycle : public Vehicle {
private:
    int cc;
public:
    Motorcycle(string brand, string model, int year, int cc) : Vehicle(brand, model, year) {
        this->cc = cc;
    }
}

```

```

void display() {
    cout << "Brand: " << brand << ", Model: " << model << ", Year: " << year << ", CC: " <<
cc << endl;
}
};
int main() {
    vector<Vehicle*> vehicles;
    Vehicle* car = new Car("Honda", "Civic", 2020, 4);
    vehicles.push_back(car);
    Vehicle* motorcycle = new Motorcycle("Harley Davidson", "Softail", 2019, 1600);
    vehicles.push_back(motorcycle);
    for (int i = 0; i < vehicles.size(); i++) {
        vehicles[i]->display();
    }
    return 0;
}

```

### **Output:**

```

Brand: Honda, Model: Civic, Year: 2020
Brand: Harley Davidson, Model: Softail, Year: 2019

```

## **2. Design a single C++ program illustrating the following concept of inheritance: single inheritance, multiple inheritance, multilevel inheritance.**

### **Single inheritance**

#### **Code:**

```

#include <iostream>
using namespace std;
class Reptile {
public:
    string name = "Snake";
};
class Animal: public Reptile {
public:
    string animal = "Monkey";
};
int main(void) {

```

```

    Animal a1;
    cout<<"Reptile Name: "<<a1.name<<endl;
    cout<<"Animal Name: "<<a1.animal<<endl;
    return 0;
}

```

### Output:

```

/tmp/wEMps6kV91.o
Reptile Name: Snake
Animal Name: Monkey

```

## Multiple Inheritance

```

#include <iostream>
using namespace std;
class Base_class {
    public:
    void display() {
        cout << "It is the first function of the Base class " << endl;
    }
};

class Base_class2 {
    public:
    void display2() {
        cout << "It is the second function of the Base class " << endl;
    }
};

class child_class: public Base_class, public Base_class2 {
    public:
    void display3(){
        cout << "It is the function of the derived class " << endl;
    }
};

int main () {
    child_class ch;
    ch.display();
    ch.display2();
    ch.display3();
}

```



```
}
```

```
/tmp/wEMps6kV91.o
```

```
It is the first function of the Base class  
It is the second function of the Base class  
It is the function of the derived class
```

## Multilevel Inheritance

```
#include <bits/stdc++.h>
using namespace std;
class A {
public:
    int a;
    void get_A_data(){
        cout << "Enter value of a: ";
        cin >> a;
    }
};
class B : public A {
public:
    int b;
    void get_B_data(){
        cout << "Enter value of b: ";
        cin >> b;
    }
};
class C : public B {
private:
    int c;

public:
    void get_C_data(){
        cout << "Enter value of c: ";
        cin >> c;
    }
    void sum(){
        int ans = a + b + c;
        cout << "sum: " << ans;
```

```

    }
};
int main(){
    C obj;
    obj.get_A_data();
    obj.get_B_data();
    obj.get_C_data();
    obj.sum();
    return 0;
}

```

```

/tmp/wEMps6kV91.o
Enter value of a: 5
Enter value of b: 4
Enter value of c: 3
sum: 12

```

### 3. Design a single C++ program illustrating the following concept of inheritance: Public derivation, Private derivation and Protected derivation.

#### Code:

```

#include <iostream>
using namespace std;
class Animal {
public:
    void eat() {
        cout << "The animal is eating." << endl;
    }
    void sleep() {
        cout << "The animal is sleeping." << endl;
    }
    void roam() {
        cout << "The animal is roaming." << endl;
    }
};
class Cat : public Animal {
public:

```

```

    void meow() {
        cout << "The cat is meowing." << endl;
    }
};
class Dog : private Animal {
public:
    void bark() {
        cout << "The dog is barking." << endl;
    }
    void doSomething() {
        eat(); // OK: derived class can access public base class members
        sleep(); // OK: derived class can access public base class members
        roam(); // OK: derived class can access public base class members
    }
};
class Horse : protected Animal {
public:
    void neigh() {
        cout << "The horse is neighing." << endl;
    }
    void doSomething() {
        eat(); // OK: derived class can access protected base class members
        sleep(); // OK: derived class can access protected base class members
        roam(); // OK: derived class can access protected base class members
    }
};
int main() {
    Cat cat;
    cat.meow();
    cat.eat(); // OK: public inheritance, derived class can access public base class members
    cat.sleep(); // OK: public inheritance, derived class can access public base class members
    cat.roam(); // OK: public inheritance, derived class can access public base class members
    Dog dog;
    dog.bark();
    //dog.eat(); // Error: private inheritance, derived class cannot access base class members
    //dog.sleep(); // Error: private inheritance, derived class cannot access base class members
    //dog.roam(); // Error: private inheritance, derived class cannot access base class members
    Horse horse;
    horse.neigh();
    //horse.eat(); // Error: protected inheritance, derived class cannot access base class members

```

```
//horse.sleep(); // Error: protected inheritance, derived class cannot access base class members
//horse.roam(); // Error: protected inheritance, derived class cannot access base class members
return 0;
}
```

**4. In C++ Create a class MCA(private: string college\_name ,int marks) and class B Tech(private:string school\_name , int marks), using function compare\_marks( ).Display the marks of both MCA and B Tech and return greatest among them. Note:compare\_mark( ) should not be member function of any class.**

**Code:**

```
#include <iostream>
using namespace std;
class MCA;
class BTech{
    private:
        string school_name;
        int marks;
    public:
        void set(string s, int m);
        friend int compare_mark(MCA mca, BTech btech);
};
void BTech :: set(string s, int m){
    school_name = s;
    marks = m;
}
class MCA{
    string college_name;
    int marks;
    public:
        void set(string c, int m);
        friend int compare_mark(MCA mca, BTech btech);
};
void MCA :: set(string c, int m){
    college_name =c;
    marks = m;
}
int compare_mark(MCA mca, BTech btech){
    if(mca.marks > btech.marks){
        cout<<"MCA has a greater score"<<endl;
```

```

        return mca.marks - btech.marks;
    }
    else if(mca.marks < btech.marks){
        cout<<"Btech has a greater score"<<endl;
        return btech.marks - mca.marks;
    }
    cout<<"Same score in both MCA and Btech"<<endl;
    return 0;
}
int main() {
    BTech ob1;
    string s;
    int btechm, mcam;
    cout<<"Enter school name and the marks of BTech"<<endl;
    cin>>s>>btechm;
    ob1.set(s, btechm);
    MCA ob2;
    string c;
    cout<<"Enter college name and the marks of MCA"<<endl;
    cin>>c>>mcam;
    ob2.set(c, mcam);
    cout<<"Marks difference: "<<compare_mark(ob2, ob1);
    return 0;
}

```

### Output:

```

/tmp/wEMps6kV91.o
Enter school name and the marks of BTech
NEPS    100
Enter college name and the marks of MCA
IGDTUW  90
Marks difference: Btech has a greater score
10|

```

**5. Create a class in C++ which provides representation for the concept of Date(day, month , year). Using operator overloading ,**

**perform the operations of unary increment and decrement of a given Date . Implement operator overloading through member function.**

### **Using Member function**

#### **Code:**

```
#include <iostream>
class Day {
    int day, month, year;
    friend Day operator++(Day& t, int);
    friend Day operator--(Day& t, int);
public:
    Day(int d = 0, int m = 0, int y = 0) : day(d), month(m), year(y) {}
    Day operator++() {
        ++day;
        if ((day > 28 && month==2 && year%4!=0) ||
            (day > 29 && month==2 && year%4==0)) {
            day = 1;
            ++month;
        }
        else if (day>31 && (month<=7 && month%2!=0) || (month>7 && month%2==0)){
            day = 1;
            ++month;
        }
        else if (day>30 && (month<=7 && month%2==0) || (month>7 && month%2!=0)){
            day =1;
            month++;
        }
        if (month>12) {
            month=1;
            ++year;
        }
        return *this;
    }
    Day operator--() {
        --day;
        if (day <1 && month==3 && year%4!=0){
            day=28;
            --month;
        }
    }
}
```

```

    }
    else if(day < 1 && month == 3 && year % 4 == 0) {
        day = 29;
        --month;
    }
    else if(day < 1 && (month <= 7 && month % 2 != 0) || (month > 7 && month % 2 == 0)) {
        day = 31;
        --month;
    }
    else if(day < 1 && (month <= 7 && month % 2 == 0) || (month > 7 && month % 2 != 0)) {
        day = 30;
        month--;
    }
    if (month < 1) {
        month = 12;
        --year;
    }
    return *this;
}

void displayDay() const {
    std::cout << year << ":" << month << ":" << day << std::endl;
}

};

Day operator++(Day& d, int) {
    Day temp = d;
    ++d;
    return temp;
}

Day operator--(Day& d, int) {
    Day temp = d;
    --d;
    return temp;
}

int main() {
    Day d1(1, 1, 2000);
    std::cout << "Original day: ";
    d1.displayDay();
    d1--;
    std::cout << "Decrementing day: ";
    d1.displayDay();
}

```

```

    ++d1;
    std::cout << "Incremented day: ";
    d1.displayDay();

    return 0;
}

```

### Output:

```

/tmp/6FqC2410e1.o
Original day: 2000:1:1
Decrement day: 1999:12:31
Incremented day: 2000:1:1

```

## LAB EXERCISE-7

**1. Write a C++ program that illustrates the concept of virtual base class.**

### Code:

```

#include <iostream>
using namespace std;
class Animal {
public:
    virtual void move() = 0;
};
class Mammal : virtual public Animal {
public:
    void move() {
        cout << "Mammal moves." << endl;
    }
};
class Bird : virtual public Animal {
public:
    void move() {
        cout << "Bird moves." << endl;
    }
};

```



```

class Bat : public Mammal, public Bird {
public:
    void move() {
        cout << "Bat flies." << endl;
    }
};

int main() {
    Animal* animal = new Bat();
    animal->move(); // Output: Bat flies.
    delete animal;
    return 0;
}

```

### Output:

```
Bat flies.
```

**2. Write a C++ program that illustrates the use of pure virtual function. Create an abstract class shape with area() and perimeter() as its member function. These functions are overridden by inherited classes namely rectangle, square, triangle and circle.**

### Code:

```

#include <iostream>
#include <cmath>
using namespace std;
class Shape {
public:
    virtual double area() = 0;
    virtual double perimeter() = 0;
};
class Rectangle : public Shape {
private:
    double width, height;
public:
    Rectangle(double w, double h) : width(w), height(h) {}
    double area() { return width * height; }
    double perimeter() { return 2 * (width + height); }
};

```

```

class Square : public Shape {
private:
    double side;
public:
    Square(double s) : side(s) {}
    double area() { return side * side; }
    double perimeter() { return 4 * side; }
};

class Triangle : public Shape {
private:
    double a, b, c;
public:
    Triangle(double x, double y, double z) : a(x), b(y), c(z) {}
    double area() {
        double s = (a + b + c) / 2;
        return sqrt(s * (s - a) * (s - b) * (s - c));
    }
    double perimeter() { return a + b + c; }
};

class Circle : public Shape {
private:
    double radius;
public:
    Circle(double r) : radius(r) {}
    double area() { return M_PI * radius * radius; }
    double perimeter() { return 2 * M_PI * radius; }
};

int main() {
    Shape *shapes[4];
    shapes[0] = new Rectangle(3, 4);
    shapes[1] = new Square(5);
    shapes[2] = new Triangle(3, 4, 5);
    shapes[3] = new Circle(2);
    for (int i = 0; i < 4; i++) {
        cout << "Shape " << i << ": area = " << shapes[i]->area()
            << ", perimeter = " << shapes[i]->perimeter() << endl;
        delete shapes[i];
    }
    return 0;
}

```

## Output:

```
Shape 0: area = 12, perimeter = 14
Shape 1: area = 25, perimeter = 20
Shape 2: area = 6, perimeter = 12
Shape 3: area = 12.5664, perimeter = 12.5664
```

**3. Write a program in C++ that allows you to create class student (member variable int) and create object array of 10 elements of student class with the following values(3,2,3,5,6,4,5,0,0,0) . The program computes the sum of elements 0 to 6 and stores it at array element 7, computes the average and stores it at array element 8 and identifies smallest value from the array and stores it at 9 th element.**

### Code:

```
#include <iostream>
class Student {
public:
    int marks;
};
int main() {
    // Create an object array of 10 elements of the Student class
    Student students[10] = {3, 2, 3, 5, 6, 4, 5, 0, 0, 0};
    // Compute the sum of elements 0 to 6 and store it at array element 7
    int sum = 0;
    for (int i = 0; i < 6; i++) {
        sum += students[i].marks;
    }
    students[7].marks = sum;
    // Compute the average and store it at array element 8
    float average = (float)sum / 7;
    students[8].marks = (int)average;
    // Identify the smallest value from the array and store it at array element 9
    int smallest = students[0].marks;
    for (int i = 1; i <= 6; i++) {
        if (students[i].marks < smallest) {
            smallest = students[i].marks;
        }
    }
}
```

```

    students[9].marks = smallest;
std::cout << "Object array values:" << std::endl;
    for (int i = 0; i < 10; i++) {
        std::cout << "students[" << i << "].marks = " << students[i].marks << std::endl;
    }
    return 0;}

```

### Output:

```

Object array values:
students[0].marks = 3
students[1].marks = 2
students[2].marks = 3
students[3].marks = 5
students[4].marks = 6
students[5].marks = 4
students[6].marks = 5
students[7].marks = 28
students[8].marks = 4
students[9].marks = 2

```

**4. Create a class Games which is having the function play. Create three more subclasses from Games which is Cricket ,Football, Basketball. Override the play method.**

### Code:

```

#include <iostream>
class Games {
public:
    virtual void play() {
        std::cout << "Playing a game..." << std::endl;
    }
};
class Cricket : public Games {
public:
    void play() override {
        std::cout << "Playing cricket..." << std::endl;
    }
};
class Football : public Games {
public:
    void play() override {

```

```

        std::cout << "Playing football..." << std::endl;
    }
};
class Basketball : public Games {
public:
    void play() override {
        std::cout << "Playing basketball..." << std::endl;
    }
};
int main() {
    Games* game = new Games();
    game->play(); // Output: "Playing a game..."
    Cricket* cricket_game = new Cricket();
    cricket_game->play(); // Output: "Playing cricket..."
    Football* football_game = new Football();
    football_game->play(); // Output: "Playing football..."
    Basketball* basketball_game = new Basketball();
    basketball_game->play(); // Output: "Playing basketball..."
    // Don't forget to delete dynamically allocated objects
    delete game;
    delete cricket_game;
    delete football_game;
    delete basketball_game;
    return 0;}

```

### Output:

```

Playing a game...
Playing cricket...
Playing football...
Playing basketball...

```

## **LAB EXERCISE-8**

**1. Write a program to handle exception different try of exceptions in C++.**

**Code:**

**Output:**

**2. Division by zero is an exception. Write a C++ program to detect such an exception handle it by displaying an appropriate message and exit from the program.**

**Code:**

**Output:**

**3. Write a C++ program to illustrate the concept of re-throwing an exception. Also add throwing specific types of exceptions.**

**Code:**

**Output:**

**4. Write a program to handle an exception thrown by new.**

**Code:**

**Output:**

## **LAB EXERCISE-9**

**1. Explain the role of scope resolution operator.**

**Code:**

**Output:**

**2. Write a program to create a database of student's information system containing the following information: Name, Roll number, Class, Contact address and Telephone number. Construct the database with suitable member functions-add() to add a new student and count() to count total students in the class. Make use of static member function and this pointer in the program.**

**Code:**

**Output:**

**3. Write a program to directly multiply two objects of a class called “Fraction” having numerator and denominator as data members , i.e., if object1=4/15 and object2=7/3 then output should be: 28/45.**

**Code:**

**Output:**

**4. Write a program to describe the behaviour of parameterized constructors during multiple inheritance.**

**Code:**

**Output:**

**5. Write a program to demonstrate who can access private and protected members of a class.**

**Code:**

**Output:**