

DELHI TECHNOLOGICAL UNIVERSITY

**SHAHBAD DAULATPUR, MAIN BAWANA
ROAD DELHI-110042**



CO-203

Object Oriented Programming Lab File

Submitted by:

Kunal Sinha

Batch -A3(2K17)

Roll No.-2K17/CO164

Submitted to:

Ms. Indu Singh

Assistant Professor

COE Department

.

INDEX

| S.NO. | TOPIC | DATE OF SUBMISSION | TEACHER'S SIGNATURE |
|-------|--|--------------------|---------------------|
| 1. | WAP to implement function overloading | 07.08.18 | |
| 2. | WAP to implement classes and objects | 14.08.18 | |
| 3. | WAP to implement static members | 14.08.18 | |
| 4. | WAP to implement array of objects | 21.08.18 | |
| 5. | WAP to implement inline functions | 21.08.18 | |
| 6. | WAP to find mean of two numbers of different classes using friend function. | 28.08.18 | |
| 7. | WAP to swap two numbers of same class using friend function | 04.09.18 | |
| 8. | WAP to add, subtract, multiply, divide two complex numbers using operator overloading. | 11.09.18 | |
| 9. | WAP to operate strings using operator overloading. | 18.09.18 | |
| 10. | WAP to implement multi-level inheritance | 19.09.18 | |
| 11. | WAP to implement hybrid inheritance | 09.10.18 | |
| 12. | WAP to implement the working of virtual functions. | 16.10.18 | |
| 13. | WAP to implement the concept of template classes. | 18.10.18 | |
| 14. | WAP to implement exception handling: division by zero, array out of bounds error. | 23.10.18 | |
| 15. | WAP to implement file handling: count number of words, sentences and characters. | 25.10.18 | |

PROGRAM 1

➤ AIM:

Write a Program to implement function overloading

➤ Description:

Function overloading is a feature in C++ where two or more functions can have the same name but different parameters. Function overloading can be considered as an example of polymorphism feature in C++.

You can have multiple definitions for the same function name in the same scope. The definition of the function must differ from each other by the types and/or the number of arguments in the argument list. You cannot overload function declarations that differ only by return type.

➤ Algorithm:

1. Start
2. Create a function to perform some task (here a function to calculate area has been made).
3. Overload the function by creating more functions with same name.
4. The function declarations should have parameters of different data types.
5. Create the main function.
6. Call the functions by passing different types of parameters.
7. Stop

➤ Source Code:

```
#include <iostream>
using namespace std;
int area(int l, int b)
{
    return l*b;
}
int area(int s)
{
    return s*s;
}
float area(float r)
{
    return 3.14*r*r;
}
```

```
int main()
{
    int l,b,s;
    float r;
    cout<<"Enter length and breadth"<<endl;
    cin>>l>>b;
    cout<<"Area of rectangle is "<<area(l, b)<<endl;
    cout<<"Enter side of square"<<endl;
    cin>>s;
    cout<<"Area of square is "<<area(s)<<endl;
    cout<<"Enter radius"<<endl;
    cin>>r;
    cout<<"Area of circle is "<<area(r)<<endl;
    return 0;
}
```

➤ Output:

```
C:\Users\home\Desktop\oops\bin\Debug\oops.exe
Enter length and breadth
5
6
Area of rectangle is 30
Enter side of square
10
Area of square is 100
Enter radius
8
Area of circle is 200.96

Process returned 0 (0x0)   execution time : 42.683 s
Press any key to continue.
```

➤ Findings and Learnings

Problem statement was executed.

Functions with same name can be used to perform different tasks differing in their argument list (concept of polymorphism).

Ambiguous parameters should not be used.

The compiler uses some rules to find a match for the function call like integral promotion.

If functions differ in return types then functions are not said to be overloaded.

PROGRAM 2

➤ AIM:

Write a Program to demonstrate the concept of classes and objects.

➤ Description:

The building block of C++ that leads to Object Oriented programming is a Class. It is a user defined data type, which holds its own data members and member functions, which can be accessed and used by creating an instance of that class. A class is like a blueprint for an object.

- A **Class** is a user defined data-type which has data members and member functions.
- Data members are the data variables and member functions are the functions used to manipulate these variables and together these data members and member functions defines the properties and behavior of the objects in a Class.
- An **Object** is an instance of a Class. When a class is defined, no memory is allocated but when it is instantiated (i.e. an object is created) memory is allocated.

➤ Algorithm

1. Start.
2. Declare a class with its private data members and member functions.
3. Define all the member functions i.e. to accept values and to display values.
4. Define the member function which adds two members i.e. the real part and imaginary part of both objects get added separately
5. Create main function.
6. Create two objects and accept the values for them using set ().
7. Compute their sum by calling the add() function in which the two initialized objects are passed as parameters
8. Display the sum using display().
9. Stop.

➤ Source Code:

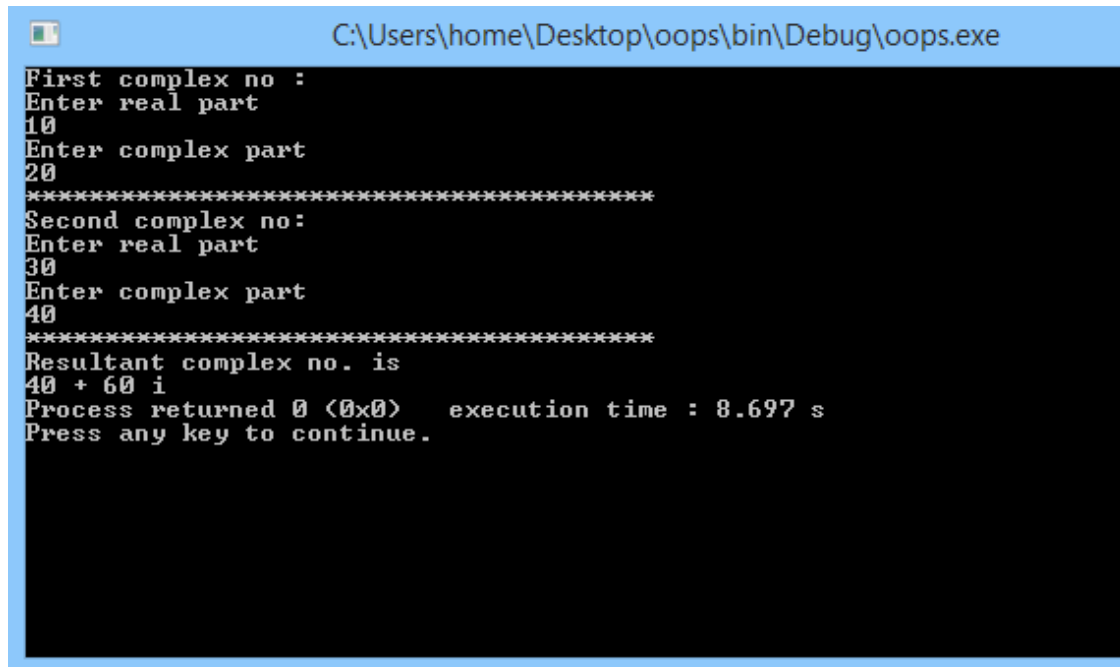
```
#include<iostream>
using namespace std;
```

```

class Complex
{
    int real;                //pvt data memebers
    int imaginary;
public:
    Complex()                //Constructor function
    {
        real = imaginary = 0;
    }
    void set()               // Member function
    {
        cout<<"Enter real part "<<endl;
        int a; cin>>a;
        cout<<"Enter complex part "<<endl;
        int b; cin>>b;
        real = a;
        imaginary = b;
    }
    void display()
    {
        cout<<real<<" + "<<imaginary<<" i";
    }
    void add(Complex& , Complex&);    //Member function prototype
};
void Complex::add (Complex& c1, Complex&c2)
{
    real = c1.real + c2.real;
    imaginary = c1.imaginary + c2.imaginary;
}
int main()
{
    Complex c1, c2,c3;        // object of class created
    cout<<"First complex no : "<<endl;
    c1.set();
    cout<<"*****\n";
    cout<<"Second complex no: "<<endl;
    c2.set();
    c3.add(c1, c2);
    cout<<"*****\n";
    cout<<"Resultant complex no. is \n";
    c3.display();
    return 0;
}

```

➤ Output:



```
C:\Users\home\Desktop\oops\bin\Debug\oops.exe
First complex no :
Enter real part
10
Enter complex part
20
*****
Second complex no:
Enter real part
30
Enter complex part
40
*****
Resultant complex no. is
40 + 60 i
Process returned 0 (0x0)   execution time : 8.697 s
Press any key to continue.
```

➤ Findings and Learning :

Problem statement was executed.

Two numbers (real or complex) can be added using classes and objects. Various other tasks can also be performed using the same concept.

An Object is an instance of the class which can access only the public members and not the private members of the class directly.

So in order to access private members member functions are created in public, as they can access the private members along with public members.

PROGRAM 3

➤ AIM:

Write a Program to implement static member functions.

➤ Description:

- A static member is shared by all objects of the class, all static data is initialized to zero when the first object is created, if no other initialization is present.
- A static member function can only access static data member, other static member functions and any other functions from outside the class.
- By declaring a function member as static, we make it independent of any particular object of the class. A static member function can be called even if no objects of the class exist and the static functions are accessed using only the class name and the scope resolution operator :: .
- We can't put it in the class definition but it can be initialized outside the class as done in the following example by re-declaring the static variable, using the scope resolution operator :: to identify which class it belongs to.

➤ Algorithm:

1. Start.
2. Declare a class with its private data members and member functions.
3. Declare a static member in public.
4. Increment the value of this static variable in constructor.
5. Initialize the value of this static member to 0, outside the class.
6. Create main function.
7. Create two objects of the class.
8. Display the Value of the static member using class name.
9. Stop.

➤ Source Code:

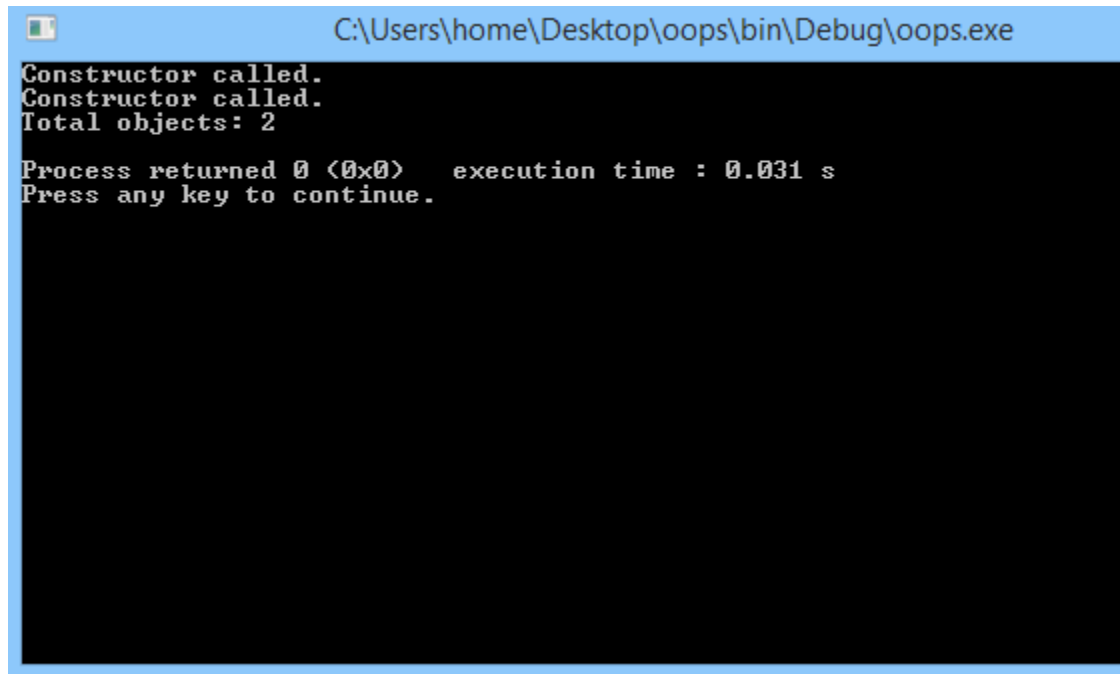
```
#include <iostream>
using namespace std;
class B
{
    int length;
    int breadth;
```

```

    int height;
public:
    static int objectCount;           //static data member
    B(int l = 2, int b = 2, int h = 2) //parameterized constructor
    {
        cout << "Constructor called." << endl;
        length = l;
        breadth = b;
        height = h;
        objectCount++;                //will calculate number of objects
    }
    double Volume()
    {
        return length * breadth * height;
    }
};
// Initialize static member of class Box
int B::objectCount = 0;
int main(void)
{
    B x1(3, 1, 1);
    B x2(8, 6, 2);
    cout << "Total objects: " << B::objectCount << endl;
    return 0;
}

```

➤ Output:



```
C:\Users\home\Desktop\oops\bin\Debug\oops.exe
Constructor called.
Constructor called.
Total objects: 2
Process returned 0 (0x0) execution time : 0.031 s
Press any key to continue.
```

➤ Findings and Learnings

Problem statement was executed.

Value of static variable by default is 0.

The value of static member thus displayed the number of objects,

This is because only one copy of static member is created and shared by all the objects, thus with creation of each object the constructor was called and eventually value of static variable incremented.

It can be used in programs where local variables whose value don't change are required to be created.

PROGRAM 4

➤ AIM:

WAP to create an array of objects by using class to input and display information of employees

➤ Description:

An array of objects, all of whose elements are of the same class, can be declared just as an array of any built-in type. Each element of the array is an object of that class. Being able to declare arrays of objects in this way underscores the fact that a class is similar to a type.

An important aspect of declaring arrays of objects in this way is that all of the objects in the array must be constructed in the same way. It is not possible with this declaration to give each different object in the array a different set of constructor values.

➤ Algorithm:

1. Start.
2. Declare a class with its private data members and member functions.
3. Define all the member functions i.e. to accept values and to display values.
4. Create main function.
5. Create an object of array using **classname object[size of array];**
6. Execute a for loop from 0 to size-1 and accept values using setData() for each index of object.
7. Execute a for loop from 0 to size-1 and Display details using displayData() for each index of object.
8. Stop.

➤ Source Code:

```
#include<iostream>
using namespace std;
class Employee
{
    int age;
    char name[50];
    int emp_id;
    public:
```

```

    void setData();
    void displayData();
};
void Employee::setData()
{
    cout<<"Enter name: ";
    cin>>name;
    cout<<"Enter age: ";
    cin>>age;
    cout<<"Enter employee id: ";
    cin>>emp_id;
}
void Employee::displayData()
{
    cout<<"Name is "<<name<<endl;
    cout<<"Age is "<<age<<endl;
    cout<<"Employee Id is "<<emp_id<<endl;
    cout<<endl;
}

int main()
{
    Employee e1[3];          //Array of objects
    for(int i = 0;i < 3; i++)
    {
        cout<<"Enter info for employee "<<i + 1<<endl;
        e1[i].setData();
        cout<<endl;
    }
    cout<<"\n*****\n";
    for(int i = 0;i < 3; i++)
    {
        cout<<"Info for employee "<<i + 1<<endl;
        e1[i].displayData();
    }
    return 0;
}

```

➤ Output:

➤ Findings and Learnings

Problem statement was executed.

No value has been accepted directly as object cannot access data members directly.

At each index the values of different data members get stored together.

Usually pointers are used to compute operations in places where arrays are used.

This concept can be used in various aspects like storing student , employee, customer database.

PROGRAM 5

➤ AIM:

WAP to perform different functions like addition ,subtraction, multiplication and division by using inline functions

➤ Description:

The CPU then executes the function code, stores the function return value in a predefined memory location/register and returns control to the calling function. This can become overhead if the execution time of function is less than the switching time from the caller function to called function (callee)

C++ provides an inline functions to reduce the function call overhead. Inline function is a function that is expanded in line when it is called. When the inline function is called whole code of the inline function gets inserted or substituted at the point of inline function call. This substitution is performed by the C++ compiler at compile time. Inline function may increase efficiency if it is small.

➤ Algorithm:

1. Create inline function (add) of return type float , accepting two parameter of type float.
2. Add function should return sum of parameters.
3. Create inline function (subtract) of return type float , accepting two parameter of type float.
4. Subtract function should return difference of parameters.
5. Create inline function (multiply) of return type float , accepting two parameter of type float.
6. Multiply function should return product of parameters.
7. Create inline function (divide) of return type float , accepting two parameter of type float.
8. Divide function should return division of parameters.
9. Create main function.
10. Take input of the two numbers from user.
11. Ask for choice of user about the operation to be performed
12. Call the function to perform operation using a switch statement.

13. Display the result
14. Ask user if wants to perform operations again.
15. If true repeat step 10 to step 14.
16. If false exit program.

➤ **Source Code:**

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

inline float add(float a, float b)
{
    return a+b;
}
inline float subtract(float a, float b)
{
    if(a>b)
        return a-b;
    else
        return b-a;
}
inline float multi(float a, float b)
{
    return a*b;
}
inline float divide(float a, float b)
{
    if(b!=0)
        return a/b;
    else
        cout<<"\nInvalid statement\n";
}

int main()
{
    float y,z;
    int x;
    char g;
    do
    {
        cout<<"Enter two number on which operation is to performed\n";
        cin>>y>>z;
```



```
cout<<"Select the operation:\n 1) Addition\n 2)Subtraction\n 3)Multiplication\n 4)Division\n";
cout<<"\nEnter the operation you want to per form: ";
cin>>x;
    switch(x)
    {
        case 1: cout<<"Result of operation "<<add(y,z);
                break;
        case 2: cout<<"Result of operation "<<subrtract(y,z);
                break;
        case 3: cout<<"Result of operation "<<multi(y,z);
                break;
        case 4: cout<<"Result of operation "<<divide(y,z);
                break;
        default: cout<<"Wrong choice";
    }
cout<<"\n Want to perform other operation(Y/N)";
cin>>g;
}while(g=='Y'||g=='y');
return 0;
}
```

➤ Output:

```
arithmetic
Enter two number on which operation is to performed
24 56
Select the operation:
1) Addition
2)Subtraction
3)Multiplication
4)Division

Enter the operation you want to per form: 1
Result of operation 80
Want to perform other operation(T/F)t
Enter two number on which operation is to performed
78 90
Select the operation:
1) Addition
2)Subtraction
3)Multiplication
4)Division

Enter the operation you want to per form: 3
Result of operation 7020
Want to perform other operation(T/F)_.

```

➤ Findings and Learnings

Problem statement was executed.

Functions defined using keywords are inline functions.

When using inline function, function call for function is replaced by function definition which makes program execute faster but at the same time take more memory.

Even if keyword inline is used with a function, but it function definition is big then compiler will ignore the keyword inline.

PROGRAM 6

➤ AIM:

Write a Program to find mean of two numbers belonging to different classes using friend function.

➤ Description:

Friend Class A friend class can access private and protected members of other class in which it is declared as friend. It is sometimes useful to allow a particular class to access private members of other class.

Friend Function A function which is not the member function of the class but still can access the member of the class. Like friend class, a friend function can be given special grant to access private and protected members.

A friend function can be:

- a) A method of another class
- b) A global function

➤ Algorithm:

1. Start
2. Declare the prototype of the friend class.
3. Create the class in which friend function is to be used.
4. Use the keyword friend before the friend class.
5. Define the mean function using suitable parameters.
6. Create the main function.
7. Input the values using getData() function.
8. Call the mean function and return the output.
9. Stop.

➤ Source Code:

```
#include<iostream>
using namespace std;
class B;                      //forward declaration
class A
{
```

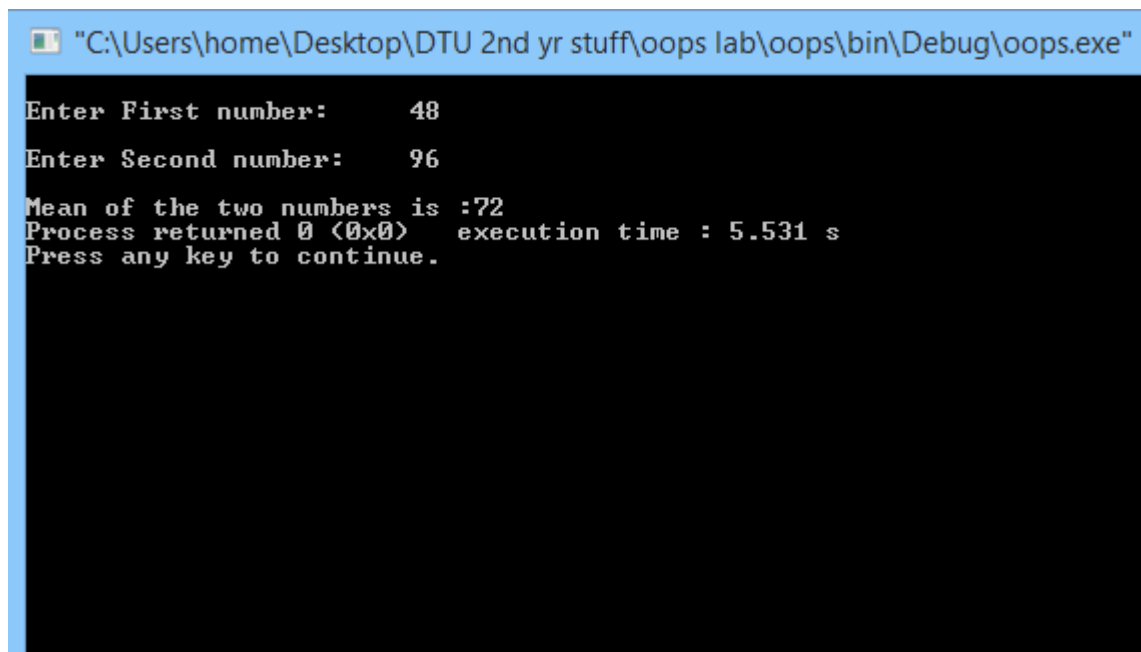
```

    int x;
    friend int mean(A&, B&);
public:
    void getData()
    {
        cout<<"\nEnter First number: \t";
        cin>>x;
    }
};
class B
{
    int y;
    friend int mean(A& , B&);
public:
    void getData()
    {
        cout<<"\nEnter Second number: \t";
        cin>>y;
    }
};
int mean(A& a, B& b)
{
    return (a.x + b.y)/2;
}

int main ()
{
    A a;
    B b;
    a.getData();
    b.getData();
    cout<<"\nMean of the two numbers is : "<<mean(a, b);
    return 0;
}

```

➤ Output:



```
"C:\Users\home\Desktop\DTU 2nd yr stuff\oops lab\oops\bin\Debug\oops.exe"
Enter First number: 48
Enter Second number: 96
Mean of the two numbers is :72
Process returned 0 (0x0) execution time : 5.531 s
Press any key to continue.
```

➤ Discussion

This program has two classes A and B and a friend function mean. The classes have an integer data member. There are member functions which accept the input from the user.

The main function call the getData() functions to accept the integer values for the member of both the classes ,then it calls the mean functions passing the objects as parameter. The mean function being a friend function has the access to the private members and so it extracts the integer values for both the classes and then computes the mean by adding and dividing by 2 and then displays the result using member function.

➤ Finding and Learning

Problem statement was executed.

Forward Declaration was used as class B was used in class A without being defined.

Friends should be used only for limited purpose.

Friendship is not mutual. If a class A is friend of B, then B doesn't become friend of A automatically.

PROGRAM 7

➤ AIM:

Write a Program to exchange two integer members of a class, use friend function, pointer to data member and function.

➤ Description:

Friend Function A function which is not the member function of the class but still can access the member of the class.

Just like pointers to normal variables and functions, we can have pointers to class member functions and member variables. We can use pointer to point to class's data members (Member variables).

Syntax for Declaration :

Datatype class_name:: *pointer_name;

Syntax for Assignment:

Pointer_name=&class_name::datamember_name;

Pointers can be used to point to class's Member functions.

Syntax for member functions:

return_type(class_name::*ptr_name)(argument_type)=&class_name::function_name;

➤ Algorithm

1. Start
2. Create the class in which friend function is to be used.
3. Use the keyword friend before the friend class.
4. Define the swap function and pass the arguments suitably.
5. Use pointer to member method and pointer to object method suitably to swap the variables.
6. Create the main function.
7. Input the values from user using setData() function.
8. Call the swap function and return the output.
9. Stop

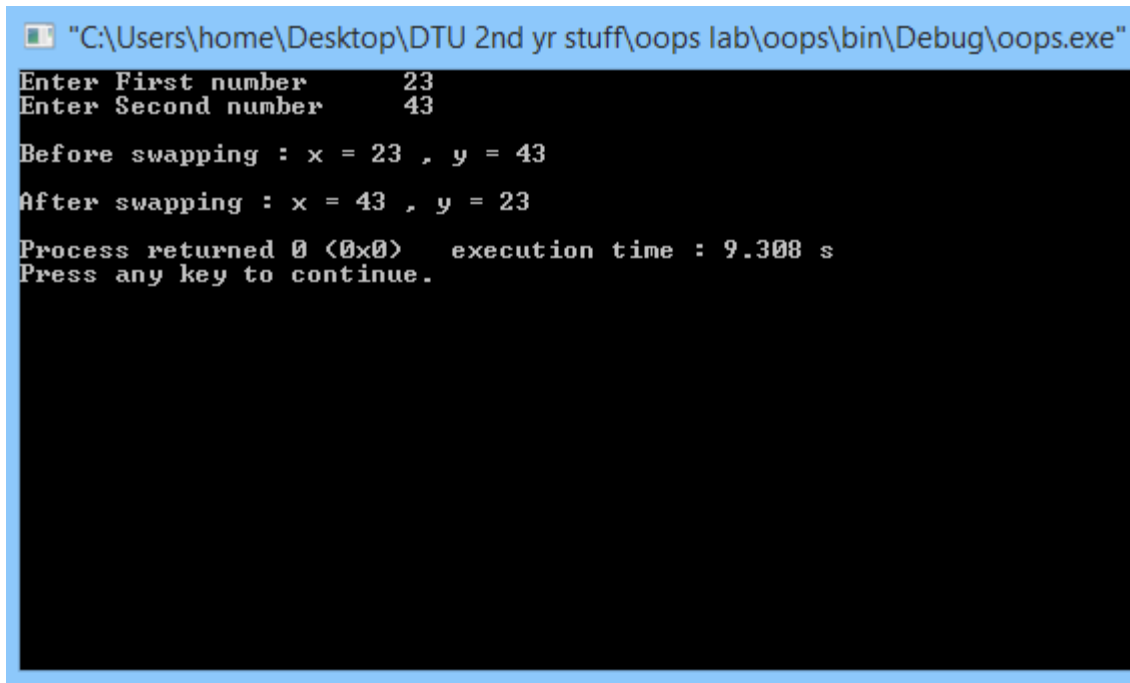
➤ Source Code:

```
#include<iostream>
using namespace std;
class A
{
    int x;
    int y;
public:
    friend void swap(A&);
    void setData(int a, int b)
    {
        x = a; y = b;
    }
};

void swap(A& a)
{
    cout<<"\nBefore swapping : x = "<<a.x<<" , y = "<<a.y<<"\n";
    int A::* px = &A :: x;
    int A::* py = &A :: y;
    A *pm = &a;
    int temp = a.*px;           //object-name .* pointer to member function
    a.*px = pm->*py;           //pointer to object ->* pointer to member function
    pm->*py = temp;
    cout<<"\nAfter swapping : x = "<<a.x<<" , y = "<<a.y<<"\n";
}

int main ()
{
    A a1;
    int a,b;
    cout<<"Enter First number \t";
    cin>>a;
    cout<<"Enter Second number \t";
    cin>>b;
    void (A::*pf)(int , int) = &A :: setData;
    (a1.*pf)(a,b);              //Pointer to member function
    swap(a1);
    return 0;
}
```

➤ Output:



```
"C:\Users\home\Desktop\DTU 2nd yr stuff\oops lab\oops\bin\Debug\oops.exe"
Enter First number      23
Enter Second number     43

Before swapping : x = 23 , y = 43
After swapping : x = 43 , y = 23

Process returned 0 (0x0)   execution time : 9.308 s
Press any key to continue.
```

➤ Discussion :

In this program a class has been created with two integer data members and a friend function swap has been created. The swap function creates two pointers to data members which swap the values of the two data members using reference method i.e using the address of the variable.

The main function calls the get() function which accepts the data from the user . Then it calls the swap function using pointer to member function which performs the swapping of the two data members of the same class and then the display() function displays the output.

➤ Finding and Learning :

Problem statement was executed.

The value and behavior of these pointers can be changed on runtime. It can be pointed to other member function or member variable.

To have pointer to data member and member functions they are declared in public.

Too many functions or external classes are declared as friends of a class with protected or private data, it lessens the value of encapsulation of separate classes in object-oriented programming.

PROGRAM 8

➤ AIM:

Write a Program to add, subtract, multiply and divide two complex numbers using operator overloading.

➤ Description:

Operator overloading is generally defined by a programming language, a programmer, or both. Operator functions are same as normal functions. The only differences are, name of an operator function is always **operator** keyword followed by symbol of operator and operator functions are called when the corresponding operator is used.

Any constructor that can be called with a single argument works as a conversion constructor, means it can also be used for implicit conversion to the class being constructed.

➤ Algorithm:

1. Start
2. Create a class complex number.
3. Add function should return sum of parameters.
4. Create function (subtract) of return type complex No. , accepting two parameter of type complex No.
5. Subtract function should return difference of 2 complex numbers.
6. Create function (multiply) of return type complex No. , accepting two parameter of type complex No.
7. Multiply function should return product of 2 complex numbers.
8. Create function (divide) of return type complex No. , accepting two parameter of type complex No.
9. Divide function should return division of 2 complex numbers.
10. Create main function.
11. Take input of the two complex numbers from user.
12. Call the functions created above using operators.
13. Display the results.
14. Stop.

➤ Source Code:

```
#include <iostream>
using namespace std;
class Complex
{
    float real;
    float imaginary;
public:
    Complex()           //Constructor function
    {
        real = imaginary = 0;
    }
    void set()          // Member function
    {
        cout << "Enter real part: ";
        float a;
        cin >> a;
        cout << "Enter complex part: ";
        float b;
        cin >> b;
        real = a;
        imaginary = b;
    }
    void display()
    {
        cout << real << " + " << imaginary << " i \n";
    }
    Complex operator+(Complex &);
    Complex operator-(Complex &);
    Complex operator*(Complex &);
    Complex operator/(Complex &);
};

Complex Complex::operator+(Complex &c1)
{
    Complex ans;
    ans.real = real + c1.real;
    ans.imaginary = imaginary + c1.imaginary;
    return ans;
}

Complex Complex::operator-(Complex &c1)
{
    Complex ans;
    ans.real = real - c1.real;
    ans.imaginary = imaginary - c1.imaginary;
    return ans;
}

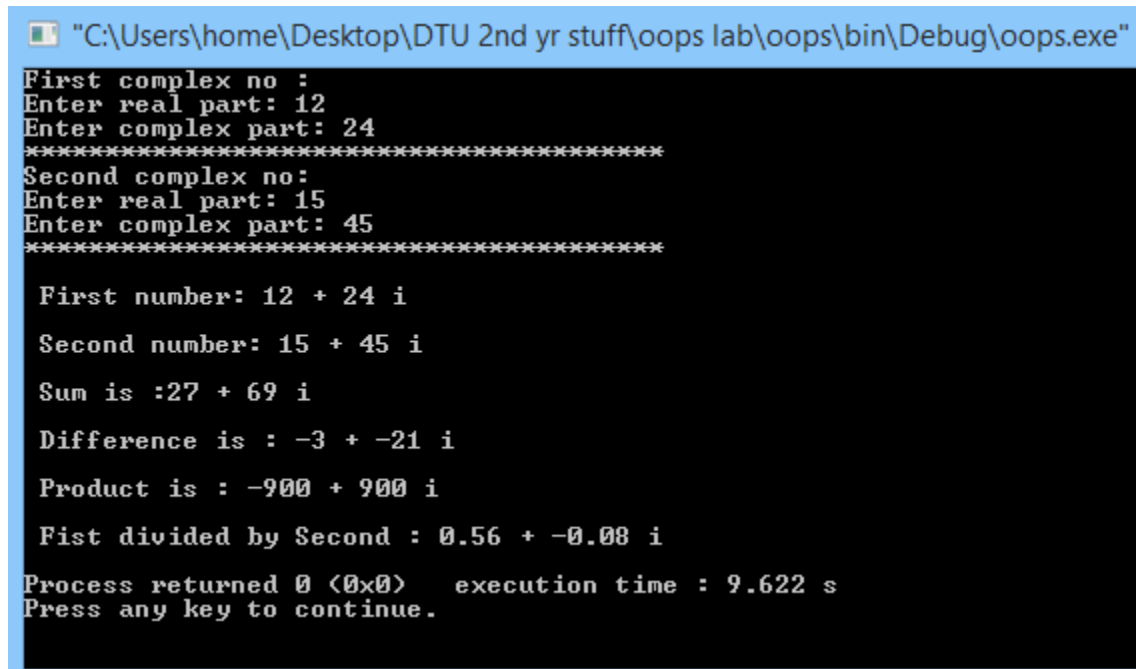
Complex Complex::operator*(Complex &c1)
{
    Complex ans;
    ans.real = real * c1.real - imaginary * c1.imaginary;
```

```

    ans.imaginary = imaginary * c1.real + real * c1.imaginary;
    return ans;
}
Complex Complex::operator/(Complex &c1)
{
    Complex ans;
    float den = c1.real * c1.real + c1.imaginary * c1.imaginary;
    ans.real = (real * c1.real + imaginary * c1.imaginary)/den;
    ans.imaginary = (imaginary * c1.real - real * c1.imaginary)/den;
    return ans;
}
int main()
{
    Complex c1, c2, c3;                                // object of class created
    cout << "First complex no : " << endl;
    c1.set();
    cout << "*****\n";
    cout << "Second complex no: " << endl;
    c2.set();
    cout << "*****\n";
    cout<<"\n First number: ";
    c1.display();
    cout<<"\n Second number: ";
    c2.display();
    c3 = c1 + c2;
    cout << "\n Sum is :";
    c3.display();
    c3 = c1 - c2;
    cout << "\n Difference is : ";
    c3.display();
    c3 = c1 * c2;
    cout << "\n Product is : ";
    c3.display();
    c3 = c1 / c2;
    cout << "\n Fist divided by Second : ";
    c3.display();
    return 0;
}

```

➤ Output:



```
"C:\Users\home\Desktop\DTU 2nd yr stuff\oops lab\oops\bin\Debug\oops.exe"
First complex no :
Enter real part: 12
Enter complex part: 24
*****
Second complex no:
Enter real part: 15
Enter complex part: 45
*****

First number: 12 + 24 i
Second number: 15 + 45 i
Sum is :27 + 69 i
Difference is : -3 + -21 i
Product is : -900 + 900 i
Fist divided by Second : 0.56 + -0.08 i
Process returned 0 (0x0)   execution time : 9.622 s
Press any key to continue.
```

➤ Discussion

In this program a class has been created with two data members which are the real and imaginary parts of a complex number. It also has 4 functions which are overloaded using operator overloading

The main function calls the member functions to accept values and then calls the overloaded functions to add, subtract, multiply and divide the complex numbers respectively. The addition and subtraction are done as usual by adding the real parts together and the imaginary parts together. However in multiplication and division other rules of complex numbers are followed.

➤ Finding and Learning

Problem statement was executed.

Operator keyword is used in the function name followed by operator symbol.

For operator overloading to work, at least one of the operands must be a user defined class object.

Some operators like '.', '::', '?:' and 'sizeof' cannot be overloaded

PROGRAM 9

➤ AIM:

Write a program to perform string operations using operator overloading.

➤ Description:

In programming, **operator overloading**, sometimes termed operator ad hoc polymorphism, is a specific case of polymorphism, where different operators have different implementations depending on their arguments.

In C++, we can make operators to work for user defined classes. 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 Number, Fractional Number, Big Integer, etc

➤ Algorithm:

1. Start
2. Create a function to perform addition task (here a function to add 2 strings has been made).
3. Overload the + & <= operator by creating more functions with same name along with the keyword operator.
4. Create the main function and pass the parameter appropriately.
5. Call the functions using suitable operator.
6. Pass the values "pass by value" method.
7. Display the result.
8. Stop.

➤ Source Code:

```
#include <iostream>
#include <string.h>
using namespace std;
class String
{
    char *s;
    int len;
```

```

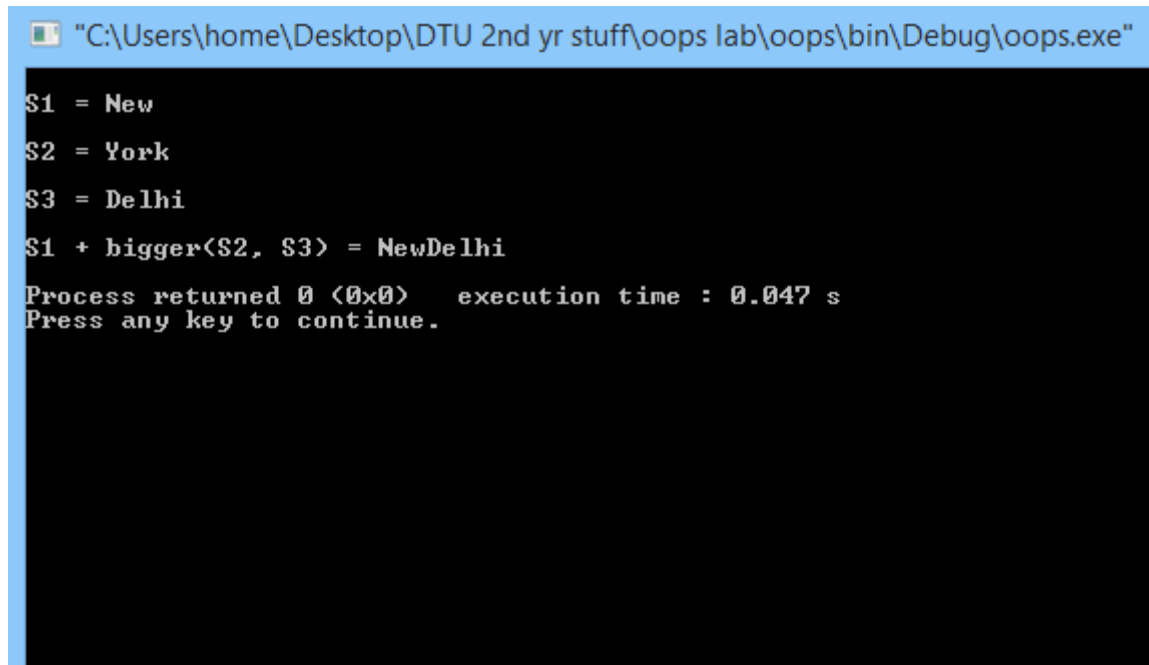
public:
    String()
    {
        //Constructor function
        s = NULL;
        len = 0;
    }
    String(const char *p)
    {
        len = strlen(p);
        s = new char[len + 1];
        strcpy(s, p);
    }
    void display()
    {
        cout << s << "\n";
    }
    friend int operator <= (String &, String&);
    friend String operator+ (String &, String&);

};
String operator+ (String &c1, String &c2)
{
    String ans;
    ans.len = strlen(c1.s) + strlen(c2.s);
    ans.s = new char[ans.len + 1];
    strcpy(ans.s, strcat(c1.s, c2.s));
}
int operator<= (String &c1, String &c2)
{
    int n = strlen(c1.s);
    int m = strlen(c2.s);
    if(n <= m) {
        return 1;
    }
    return 0;
}
int main()
{
    String s1 = "New";
    String s2("York");
    String s3("Delhi");
    cout<<"\nS1 = ";
    s1.display();
    cout<<"\nS2 = ";
    s2.display();
    cout<<"\nS3 = ";
    s3.display();
    String s4 = (s2 <= s3) ? s3 : s2;
    cout<<"\nS1 + bigger(S2, S3) = ";
    (s1 + s4).display();
}

```

```
    return 0;  
}
```

➤ Output:



```
"C:\Users\home\Desktop\DTU 2nd yr stuff\oops lab\oops\bin\Debug\oops.exe"  
  
S1 = New  
S2 = York  
S3 = Delhi  
S1 + bigger(S2, S3) = NewDelhi  
Process returned 0 (0x0)   execution time : 0.047 s  
Press any key to continue.
```

➤ Description

In this program a class is made with a string as a data member and an overloaded function is created.

The main function accepts input strings from the user, then it uses an overloaded function to compare which string is bigger out of string 2 and string 3. Whichever string comes out to be bigger it concatenates it with string 1 using the + operator which is overloaded as a function and then finally displays the output.

➤ Finding and Learning:

Problem statement was executed.

The + operator performs string concatenation whereas the <= performs string comparisons.

Overloaded conversion operators must be a member method. Other operators can either be member method or global method.

Compiler automatically creates a default assignment operator with every class.

PROGRAM 10

➤ AIM:

Write a program to implement multi-level inheritance.

➤ Description:

Inheritance is the process of inheriting properties of objects of one class by objects of another class. The class which inherits the properties of another class is called Derived or Child or Sub class and the class whose properties are inherited is called Base or Parent or Super class.

When a class is derived from a class which is also derived from another class, i.e. a class having more than one parent classes, such inheritance is called **Multilevel Inheritance**. The level of inheritance can be extended to any number of level depending upon the relation. Multilevel inheritance is similar to relation between grandfather, father and child.

➤ Algorithm:

1. Start
2. Make classes student.
3. Store roll number ,name as a data member and a function to accept data.
4. Make classes Test derived publicly from student.
5. Store subject 1 and subject 2 as a data member and some member functions.
6. Make classes Result derived publicly from Test.
7. Store Total as a data member and a member function to calculate the total.
8. Result is inherited from Test which in turn is inherited from Student and hence Multi level Inheritance is achieved.
9. Create the main function.
10. Input the values from user and perform the required task
11. Stop.

➤ Source Code:

```
#include<iostream>
#include <string.h>
using namespace std;
```



```

class Student
{
protected:
    int rn;
    char * name;
public :
    void get(int r, const char * n)
    {
        rn = r;
        name = new char[strlen(n) + 1];
        strcpy(name, n);
    }
    void shown()
    {
        cout<<"\nRoll No. : "<<rn<<"\n";
        cout<<"\nName : "<<name<<"\n";
    }
};

class Test : public Student
{
protected:
    int sub1, sub2;
public:
    void getM(int m1, int m2)
    {
        sub1 = m1;
        sub2 = m2;
    }
    void showM()
    {
        cout<<"\nMarks in subject 1 : "<<sub1<<"\n";
        cout<<"\nMarks in subject 2 : "<<sub2<<"\n";
    }
};

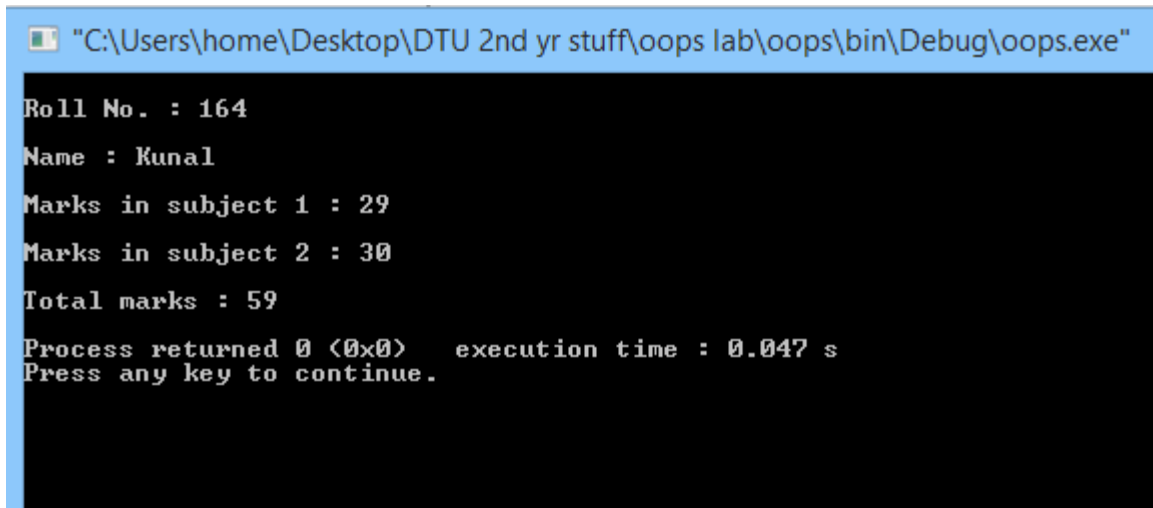
class Result : public Test
{
protected:
int total;
public :
    void display()
    {
        total = sub1 + sub2;
        shown();
        showM();
        cout<<"\nTotal marks : "<<total<<"\n";
    }
};

int main ()
{

```

```
Result r1;  
r1.get(164, "Kunal");  
r1.getM(29, 30);  
r1.display();  
return 0;  
}
```

➤ Output:



```
"C:\Users\home\Desktop\DTU 2nd yr stuff\oops lab\oops\bin\Debug\oops.exe"  
  
Roll No. : 164  
Name : Kunal  
Marks in subject 1 : 29  
Marks in subject 2 : 30  
Total marks : 59  
Process returned 0 (0x0) execution time : 0.047 s  
Press any key to continue.
```

➤ Discussion

In this program 3 classes have been created naming student, test and result with their respective data members. The class result is publicly derived from test which in turn is publicly derived from student. So all the member functions declared in public in class student are accessible in class result. So all the data members are accessible in class result.

The main function creates the object of class result and calls the get function to accept values for all data members of all classes and then calls the total function to add the marks in both subjects and then finally calls the display functions to display the values.

➤ Finding and Learning

Problem statement was executed.

Members declared in private mode cannot be inherited.

If constructors are created then constructor of base class gets called first followed by constructor of derived class.

Since the classes were derived publicly the public member of the base class became public and protected members of the base class became protected in derived class.

PROGRAM 11

➤ AIM:

Write a program to implement Hybrid inheritance.

➤ Description:

Inheritance is the process of inheriting properties of objects of one class by objects of another class. The class which inherits the properties of another class is called Derived or Child or Sub class and the class whose properties are inherited is called Base or Parent or Super class.

The inheritance in which the derivation of a class involves more than one form of any inheritance is called hybrid inheritance. Basically C++ hybrid inheritance is combination of two or more types of inheritance. It can also be called multi path inheritance.

➤ Algorithm:

1. Start
2. Create a class person and input it's details like name and age.
3. Create a class employee which inherits the class person and then declare it's public and private data members such as employeeID and designation of the person.
4. Next we write 2 data member function, get data and put data.
5. Finally we create a class salary and define the increment function according to the factors .
6. Display the new salary and exit the program.

➤ Source Code:

```
#include<iostream>
using namespace std;
typedef enum{Manager, CTO, CEO} Post;
const char *posts[]={ "Manager","CTO","CEO" };
class Person
{
    protected:
        int age;
        char * name;
    public :
        void getR(int r, const char * n)
{
```

```

        age = r;
        name = new char[strlen(n) + 1];
        strcpy(name, n);
    }
    void showR()
    {
        cout<<"Name : "<<name<<"\n";
        cout<<"Age : "<<age<<"\n";
    }
};
class Employee : public Person
{
    protected:
        int empId;
        Post post;
    public:
        void getE(int r, Post p)
        {
            empId = r;
            post = p;
        }
        void showE()
        {
            cout<<"Employee ID : "<<empId<<"\n";
            cout<<"Post : "<<posts[post]<<"\n";
        }
};
class Salary {
    protected: double salary;
    public :
        void showS()
        {

            cout<<fixed<<setprecision(1)<<"Total salary : "<<salary<<" pa \n";
        }

};
class Increment : public Salary, public Employee
{
    protected: double inc;
    public:
        void display()
        {
            switch(post)
            {
                case(0) :
                    inc = 1.0;
                    salary = 1000000;
                    break;
                case(1) :

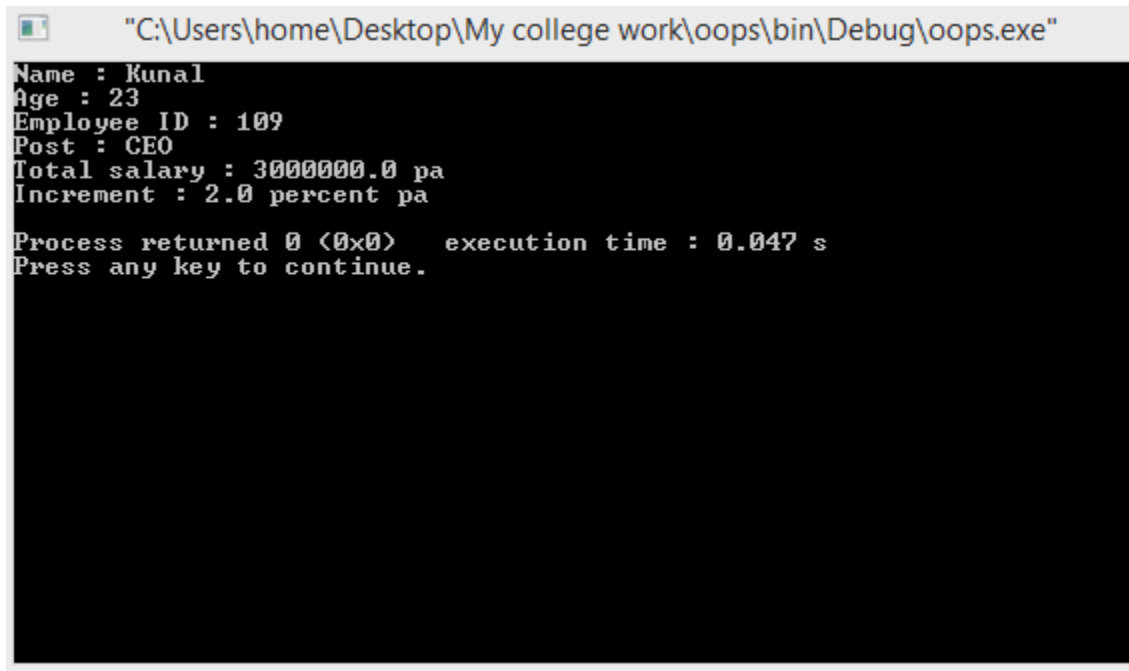
```

```

        inc = 1.5;
        salary = 2000000;
        break;
    case(2) :
        inc = 2.0;
        salary = 3000000;
        break;
    }
    showR();
    showE();
    showS();
    cout<<"Increment : "<<inc<<" percent pa \n";
}
};
int main ()
{
    Increment r1;
    r1.getR(23, "Kunal");
    Post p = CEO;
    r1.getE(109 , p);
    r1.display();
    return 0;
}

```

➤ Output:



```
"C:\Users\home\Desktop\My college work\oops\bin\Debug\oops.exe"
Name : Kunal
Age : 23
Employee ID : 109
Post : CEO
Total salary : 30000000.0 pa
Increment : 2.0 percent pa

Process returned 0 (0x0)   execution time : 0.047 s
Press any key to continue.
```

➤ Discussion

In this program 4 classes have been created naming person, employee, salary and increment with their respective data members. The class employee is derived from person publicly and class increment is derived from employee and salary.

The main function creates the object of class increment, accepts values for all data members of all classes and then calls the display functions to display the values.

➤ Finding and Learning

Problem statement was executed.

Members declared in private mode cannot be inherited.

If constructors are created then constructor of base class gets called first followed by constructor of derived class.

There is a possibility in hybrid classes that a function may be inherited twice by the derived class, such problems are eliminated using virtual functions.

PROGRAM 12

➤ AIM:

Write a program to explain virtual functions by creating a class C-Polygon which has function area() to calculate and return the area of rectangle and triangle respectively

➤ Description:

A virtual function a member function which is declared within base class and is re-defined (Overriden) by derived class. When you refer to a derived class object using a pointer or a reference to the base class, you can call a virtual function for that object and execute the derived class's version of the function.

- Virtual functions ensure that the correct function is called for an object, regardless of the type of reference (or pointer) used for function call.
- They are mainly used to achieve Runtime polymorphism
- Functions are declared with a virtual keyword in base class.
- The resolving of function call is done at Run-time.

➤ Algorithm:

1. Start
2. Create a class polygon and initialize its protected member functions.
3. Create the area function and initialize it with a virtual keyword.
4. Create a class triangle which inherits the class polygon and declare its data member functions.
5. Create a class rectangle which inherits the class polygon and declare its data member functions.
6. Create the main function of the program.
7. End the program

➤ Source Code:

```
#include <iostream>
using namespace std;
class C_Polygon
{
protected:
    double x, y;

public:
    void set(double i, double j)
    {
        x = i;
        y = j;
    }
}
```

```

    }
    virtual void area()
    {
        cout << "Calculate area for diff shapes \n ";
    }
};

class C_triangle : public C_Polygon
{
public:
    void area()
    {
        cout << "Area of triangle with height ";
        cout << x << " and base "<<y<<" is " ;
        cout << x * 0.5 * y << ".\n";
    }
};

class C_rectangle : public C_Polygon
{
public:
    void area()
    {
        cout << "Rectangle with dimensions ";
        cout << x << " and " << y;
        cout << " has an area of ";
        cout << x * y << ".\n";
    }
};

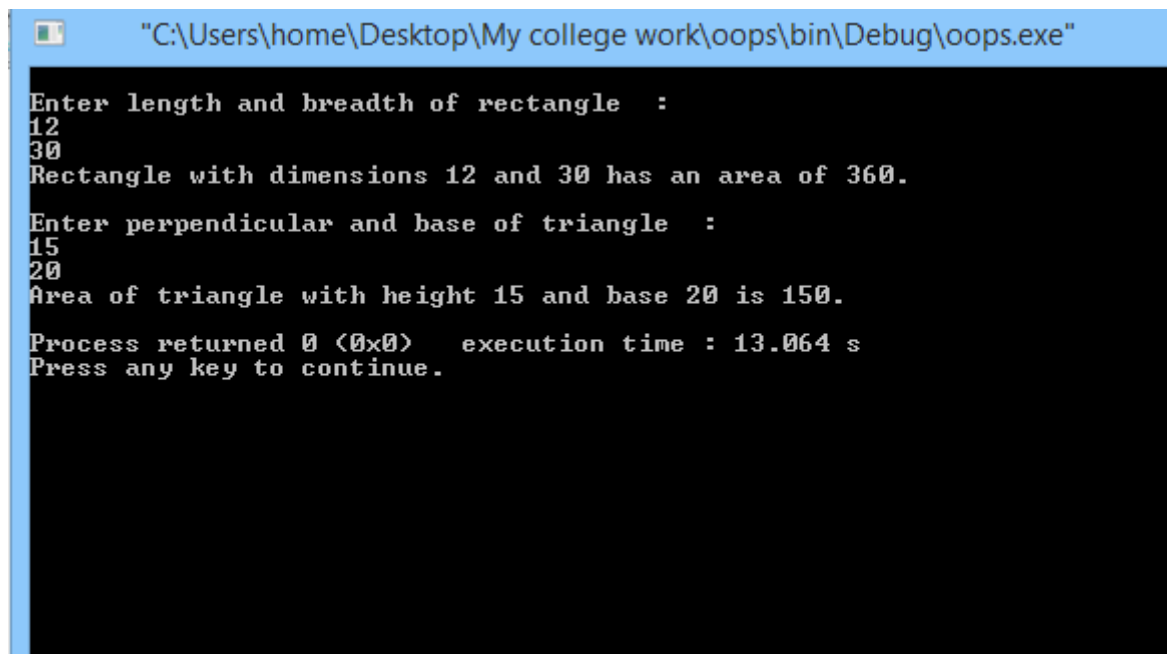
int main ()
{
    C_Polygon *c;
    C_rectangle r;
    C_triangle t;

    c = &r;
    int k,l;
    cout<<"\nEnter length and breadth of rectangle :";
    cin>>k>>l;
    c->set(k, l);
    c->area();

    c = &t;
    cout<<"\nEnter perpendicular and base of triangle :";
    cin>>k>>l;
    c->set(k,l);
    c->area();
}

```


➤ Output:



```
"C:\Users\home\Desktop\My college work\oops\bin\Debug\oops.exe"

Enter length and breadth of rectangle :
12
30
Rectangle with dimensions 12 and 30 has an area of 360.

Enter perpendicular and base of triangle :
15
20
Area of triangle with height 15 and base 20 is 150.

Process returned 0 (0x0) execution time : 13.064 s
Press any key to continue.
```

➤ Discussion

In this program 3 classes have been created naming polygon, rectangle and triangle. The classes triangle and rectangle have their respective data members and member functions and is derived from class polygon. The class polygon has a virtual function area which computes the area of different shapes.

The main function creates the pointer of class and first points to object of rectangle to accept values for all data members of all classes and then calls the area function to compute the area and finally calls the display functions to display the values and does exactly same for triangle.

➤ Finding and Learning

Problem statement was executed.

They Must be declared in public section of class.

Virtual functions should be accessed using pointer or reference of base class type to achieve run time polymorphism.

The prototype of virtual functions should be same in base as well as derived class.

They are always defined in base class and overridden in derived class. It is not mandatory for derived class to override (or re-define the virtual function), in that case base class version of function is used.

PROGRAM 13

➤ AIM:

Write a program to explain class template by creating a template T for a class named pair having 2 data members of type T which are inputted by a constructor and a member function get-max(). Return the greatest of the two numbers using main().

➤ Description:

Templates are powerful features of C++ which allows you to write generic programs. In simple terms, you can create a single function or a class to work with different data types using templates.

Templates are often used in larger codebase for the purpose of code reusability and flexibility of the programs. The concept of templates can be used in two different ways:

- Function Templates
- Class Templates

➤ Algorithm:

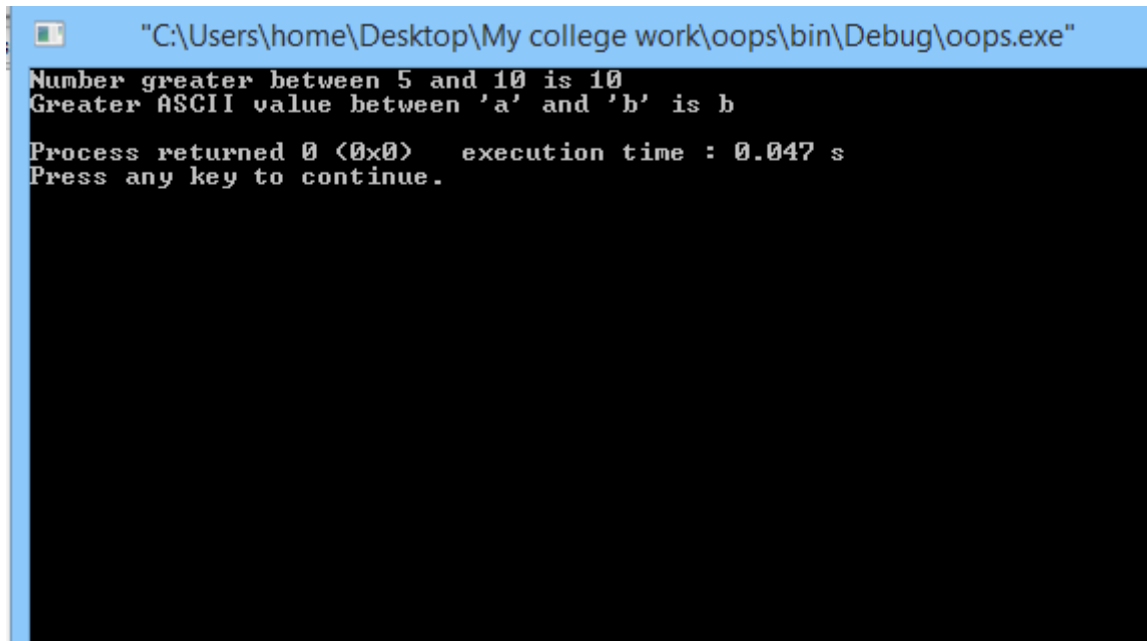
1. Start
2. Create a template class T with 2 data members.
3. In the public section, pair the 2 data members and assign them accordingly.
4. Create a template function which return max value of the 2 parameters passed into it.
5. Use the template function so as to return the max of 2 values of integer type and character type respectively.
6. Create the main function which calls the given functions.
7. End the program .

➤ Source Code:

```
#include <iostream>
using namespace std;
template <class T>
class Pair
{
    T a;
    T b;
public:
    Pair(T x, T y) {
```

```
        a = x;
        b = y;
    }
    T getMax()
{
    return (a > b) ? a : b;
}
};
int main ()
{
    Pair<int> p(5, 10);
    int max = p.getMax();
    cout<<"Number greater between 5 and 10 is "<<p.getMax()<<"\n";
    Pair<char> p1('a' , 'b');
    cout<<"Greater ASCII value between 'a' and 'b' is "<<p1.getMax()<<"\n";
    return 0;
}
```

➤ Output:



```
"C:\Users\home\Desktop\My college work\oops\bin\Debug\oops.exe"  
Number greater between 5 and 10 is 10  
Greater ASCII value between 'a' and 'b' is b  
Process returned 0 (0x0) execution time : 0.047 s  
Press any key to continue.
```

➤ Discussion

In this program a template class has been created with parameters. The parameters have been replaced by int, char, float at runtime. Thus it allowed us to create a generic class which can be used for different data types.

➤ Finding and Learning

Problem statement was executed.

Generic programming is an approach where generic types are used as parameters in algorithms so that they work for a variety of suitable data types and data structures. It is a feature that provides support for generic programming.

It can be used to create a family of classes or functions.

PROGRAM 14

➤ AIM:

Write a program to illustrate division by zero, array index out of bounds exception.

➤ Description:

One of the advantages of C++ over C is Exception Handling. C++ provides following specialized keywords for this purpose.

- *try*: represents a block of code that can throw an exception.
- *catch*: represents a block of code that is executed when a particular exception is thrown.
- *throw*: Used to throw an exception. Also used to list the exceptions that a function throws, but doesn't handle itself.

➤ Algorithm:

1. Start
2. Use the try function to prevent division by zero during runtime.
3. Use the catch block to catch the Null pointer exception if any , in the program.
4. Next , apply the try catch statement for IndexOutOfRangeException for accessing the element of array which is larger than the size of array.
5. Throw a catch statement showing IndexOutOfRangeException in the array.
6. Display the required results.
7. End the program.

➤ Source Code:

```
#include <iostream>
using namespace std;
int main ()
{
    int a,b;
    cout<<"Enter two no's to be divided \n";
    cin>>a>>b;
    try
    {
        if(b == 0)
        {
            throw -1;
        }
        cout<<a/b;
    }
}
```

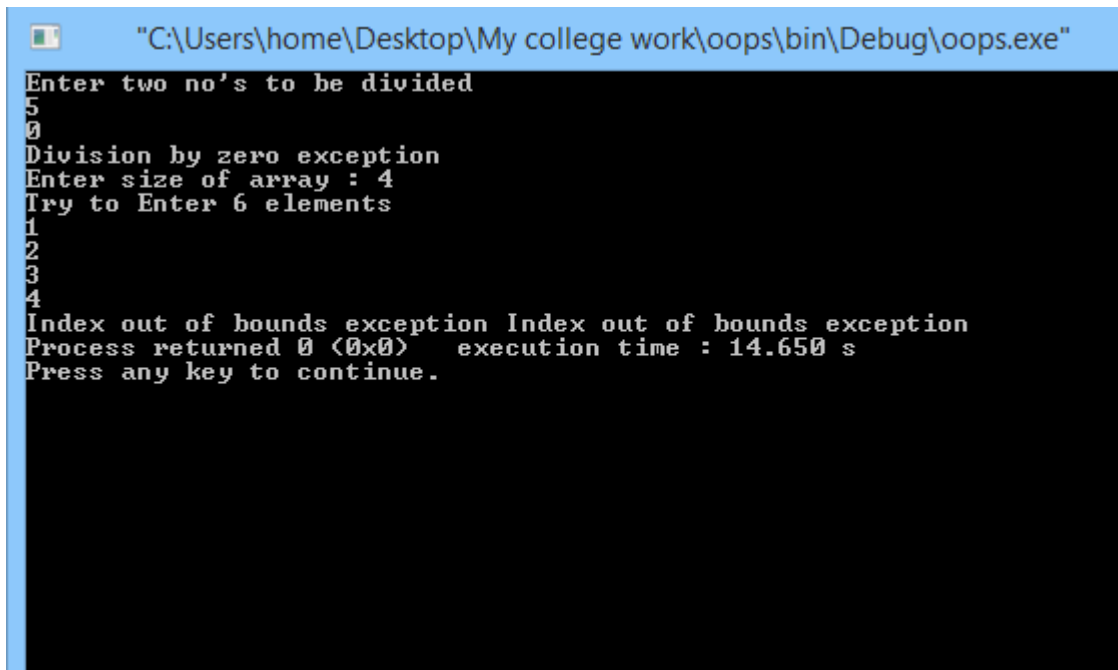
```

catch(int n)
{
    cerr<<"Division by zero exception \n";
}
int n;
cout<<"Enter size of array : ";
cin>>n;
int *arr = new int [n];
cout<<"Try to Enter "<<n + 2<<" elements";
for(int i = 0;i < n + 2; i++) {

    try
    {
        if(i >= n)
        {
            throw -1;
        }
        cin>>arr[i];
    }
    catch(int n)
    {
        cerr<<"Index out of bounds exception ";
    }
}
}

```

➤ Output:



```
"C:\Users\home\Desktop\My college work\oops\bin\Debug\oops.exe"
Enter two no's to be divided
5
0
Division by zero exception
Enter size of array : 4
Try to Enter 6 elements
1
2
3
4
Index out of bounds exception Index out of bounds exception
Process returned 0 (0x0)   execution time : 14.650 s
Press any key to continue.
```

➤ Discussion

In this program exception handling is being tested and implemented. Any number cannot be divide by zero so as soon as zero as encountered as denominator it throws an error .Similarly when we try to enter more elements than the size of array it throws an error and this a very powerful tool of C++.

➤ Finding and Learning

Problem statement was executed.

A function can also re-throw a function using same “throw; “. A function can handle a part and can ask the caller to handle remaining.

When an exception is thrown, all objects created inside the enclosing try block are destructed before the control is transferred to catch block

PROGRAM 15

➤ AIM:

Write a program to implement file handling. Count number of words, characters and sentences.

➤ Description:

In C++, files are mainly dealt by using three classes fstream, ifstream, ofstream available in fstream headerfile.

ofstream: Stream class to write on files

ifstream: Stream class to read from files

fstream: Stream class to both read and write from/to files.

Now the first step to open the particular file for read or write operation. We can open file by

1. passing file name in constructor at the time of object creation
2. using the open method

Files are opened using the command `fin.open('filename.txt');`

➤ Algorithm:

1. Start
2. The program opens a file named abc.txt .
3. It reads it till the end of the file is reached
4. During reading count the number of words encountered, no of characters by the pointer and sentences present in the file after every newline character is encountered .
5. Number of words is counted by number of spaces.
6. Function open() is used to open a file and close() disconnects the file from the stream .
7. Finally after reading the file, we need to show the total number of words ,lines and characters in the document.
8. End the program.

➤ Source Code:

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

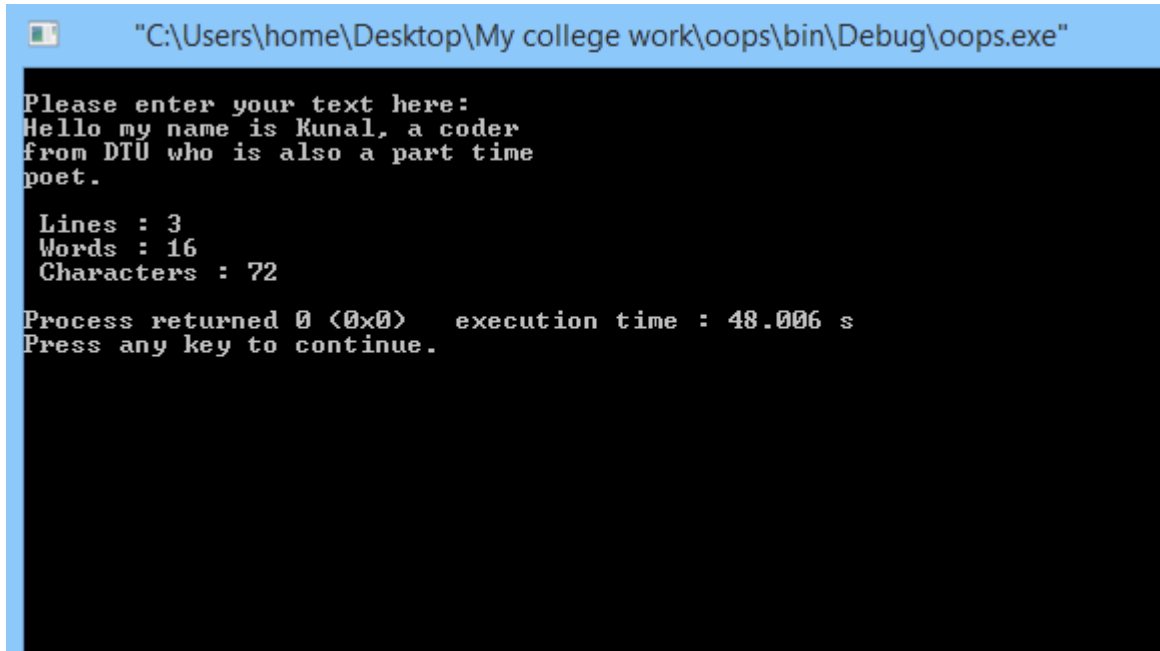


```

int main ()
{
    string result = "";
    string line = "";
    cout<<"\nPlease enter your text here: \n";
    while(getline(cin,line) && !line.empty()) {
        result += line + '\n';
    }
    ifstream infile;
    ofstream outfile;
    outfile.open("abc.txt");
    outfile << result;
    outfile.close();
    char ch;
    int lines = 0;
    int words = 0;
    int chars = 0;
    infile.open("abc.txt");
    while(infile.get(ch)) {
        if(ch == ' '){
            words++;
        } else if(ch == '\n') {
            words++;
            lines++;
        }
        chars++;
    }
    if(words == 0){
        words++;
    }
    else {
        words--;
        cout<<" Lines : "<<lines<<endl;
        cout<<" Words : "<<words<<endl;
        cout<<" Characters : "<<chars<<endl;
    }
    return 0;
}

```

➤ Output:



```
"C:\Users\home\Desktop\My college work\oops\bin\Debug\oops.exe"

Please enter your text here:
Hello my name is Kunal, a coder
from DTU who is also a part time
poet.

Lines : 3
Words : 16
Characters : 72

Process returned 0 (0x0) execution time : 48.006 s
Press any key to continue.
```

➤ Discussion

In this program we input the text from the user in the file and it gets stored in the file. Now we read from the file till end of file is reached. File is read character by character and increments the count of characters. However as ' ' or space is encountered it increments the number of words and as '\n' or enter is pressed it increments the number of lines.

➤ Finding and Learning

Problem statement was executed.

Files are basically used to store data that may vanish if the program is run again.

ofstream and ifstream are objects of class fstream.

File is read in ifstream and is written in ofstream mode.

Text files usually terminate with extension '.txt'.