Aim:

To write a program to create a class `employee` with following specifications :

- `calculate()` : A member function to net salary
- `havedata()` : A member function to accept values
- `displaydata()` : A member function to display all data members

Theory:

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. 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. The public data members are also accessed in the same way given however the private data members are not allowed to be accessed directly by the object. Accessing a data member depends solely on the access control of that data member. This access control is given by Access modifiers in C++. There are three access modifiers : public, private and protected. The member functions defined inside the class are inline functions and declared outside are treated as normal functions.

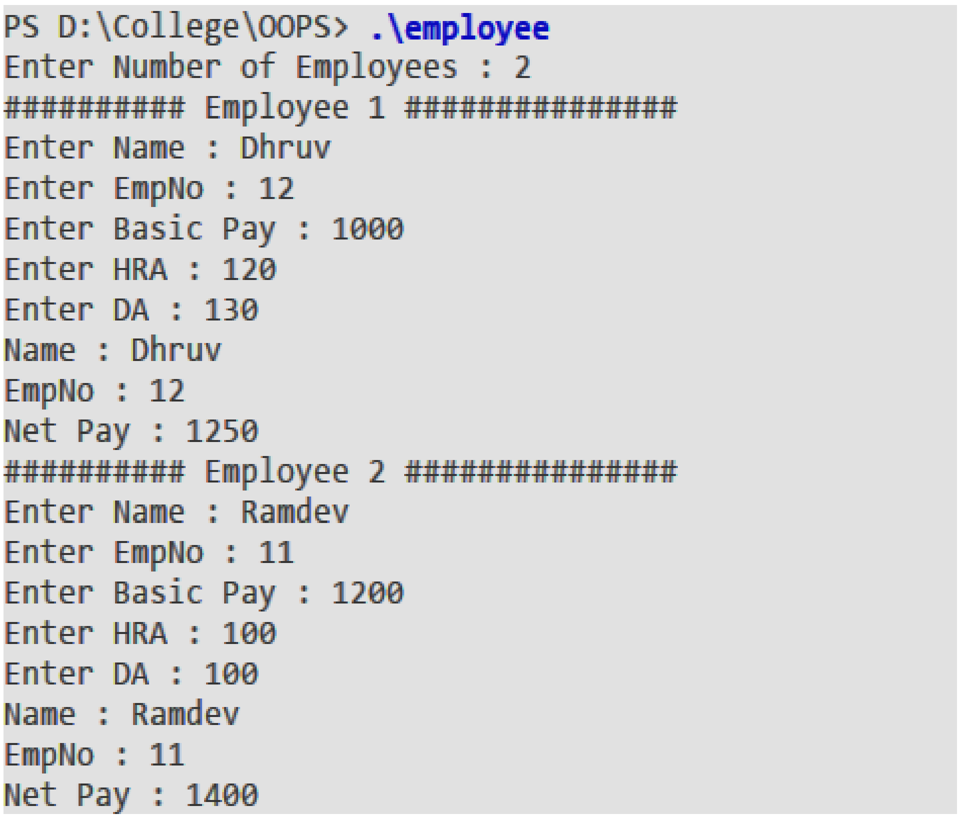
Syntax :

```
// Class Declaration
class name{
    private:
    //data members and member functions
    public:
    //data members and member functions
    protected:
    //data members and member functions
};

// Object Declaration
name object;
```

Code :

```
#include <iostream>
// #include <string>

using namespace::std;

class Employee {
    char name[50] ;
    int empno ;
    float basic ;
    float hra ;
    float da ;
    float netpay ;

    float calculate() {
        return basic + da + hra ;
    }

    public :

    void havedata() {
        cout << "Enter Name : " ;
        cin >> name ;
        cout << "Enter EmpNo : " ;
        cin >> empno ;
        cout << "Enter Basic Pay : " ;
        cin >> basic ;
        cout << "Enter HRA : " ;
        cin >> hra ;
        cout << "Enter DA : " ;
        cin >> da ;
        netpay = calculate() ;
    }

    void displaydata() {
        cout << "Name : "  << name << endl ;
        cout << "EmpNo : "  << empno << endl ;
        cout << "Net Pay : "  << netpay << endl ;
    }
};


int main () {
    int n ;
    cout << "Enter Number of Employees : " ;
    cin >> n ;

    Employee emp[n] ;

    int i = 0 ;
    while(i < n) {
        cout << "##### Employee " << i+1 << " ######" << endl ;
        emp[i].havedata() ;
        emp[i++].displaydata() ;
    }
    return 0 ;
}
```

Output :



```
PS D:\College\OOPS> .\employee
Enter Number of Employees : 2
########## Employee 1 ###############
Enter Name : Dhruv
Enter EmpNo : 12
Enter Basic Pay : 1000
Enter HRA : 120
Enter DA : 130
Name : Dhruv
EmpNo : 12
Net Pay : 1250
########## Employee 2 ###############
Enter Name : Ramdev
Enter EmpNo : 11
Enter Basic Pay : 1200
Enter HRA : 100
Enter DA : 100
Name : Ramdev
EmpNo : 11
Net Pay : 1400
```

Discussion :

In the above program we have created a class `Employee` in which number, name and salary are data member and there are two member function for input and output. Now in the `main` program we have created an array of objects which means we have created a number of objects with each object can be represented as an empty which stores the info of each employee and ask user to enter the info of each
employee. Thus, displaying information of each employee as shown in output.

Learning Outcomes :

By studying classes, we learnt the following things: -

- Classes contain, data members and member functions, and the access of these data members and variable depends on the access specifiers .

- Class member functions can be defined inside the class definition or outside the class definition.

- Classes in C++ are similar to structures in C, the only difference being, class defaults to private access control, where as structure defaults to public.
- All the features of OOPS, revolve around classes in C++.
- Objects of class holds separate copies of data members. We can create as many objects of a class as we need.

Aim:

Write a program to add, subtract, multiply and divide two complex number using **Operator Overloading** .

Theory:

Operator overloading is an important concept in C++. It is a type of polymorphism in which an operator is overloaded to give user defined meaning to it. Overloaded operator is used to perform operation on
user-defined data type. For example, '+' operator can be overloaded to perform addition on various data types, like for Integer, String(concatenation) etc.
Almost any operator can be overloaded in C++. However, there are few operator which can not be overloaded. Operator that are not overloaded are follows

- Scope Resolution Operator - `::`
- Member Access or Dot operator - `.`
- Pointer-to-member Operator - `*`
- Ternary or Conditional Operator - `?:`

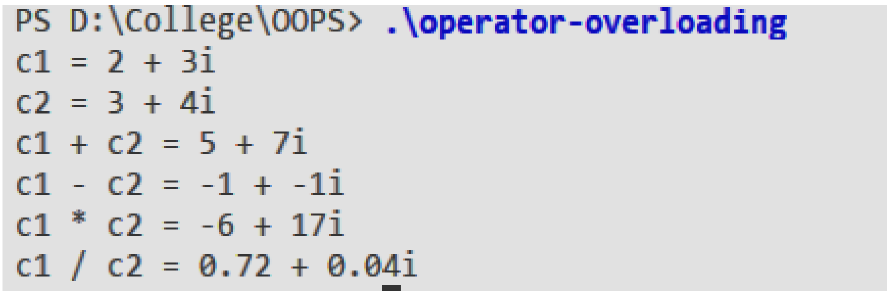
Syntax :

```
data_type classname :: operator symbol (arguments){
    //function body
}
```

Code :

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

class Complex {
    float real ;
    float imag ;

    public :
    Complex () {
        real = 0 ;
        imag = 0 ;
    }
    Complex (float x , float y) {
        real = x ;
        imag = y ;
    }
    void display(){
        cout << real << " + " << imag << "i" << endl ;
    }
    Complex operator + ( Complex x) {
        return Complex( real + x.real , imag + x.imag );
    }
    Complex operator - ( Complex x) {
        return Complex( real - x.real , imag - x.imag );
    }
    Complex operator * ( Complex x) {
        return Complex( real * x.real - imag * x.imag  , real * x.imag + imag * x.real )
    }
    Complex operator / ( Complex x) {
        return Complex( (real * x.real + imag * x.imag) / ( x.real * x.real + x.imag * x
                        (imag * x.real - real * x.imag) / ( x.real * x.real + x.imag * x
    }
} ;

int main(){

    Complex c1(2.0 , 3.0 ) , c2(3.0 , 4.0) , c3 ;

    cout << "c1 = " ;
    c1.display();
    cout << "c2 = " ;
    c2.display();

    cout << "c1 + c2 = " ;
    c3 = c1 + c2 ;
    c3.display();

    cout << "c1 - c2 = " ;
    c3 = c1 - c2 ;
    c3.display();

    cout << "c1 * c2 = " ;
    c3 = c1 * c2 ;
    c3.display();

    cout << "c1 / c2 = " ;
    c3 = c1 / c2 ;
    c3.display();

    return 0 ;
}
```

Output :



```
PS D:\College\OOPS> .\operator-overloading
c1 = 2 + 3i
c2 = 3 + 4i
c1 + c2 = 5 + 7i
c1 - c2 = -1 + -1i
c1 * c2 = -6 + 17i
c1 / c2 = 0.72 + 0.04i
```

Discussion :

In the above program we created a class complex which is used for carrying operation on complex numbers like addition, subtraction, multiplication and division with the help of operator overloading.
Operator overloading refers to operator polymorphism. For example, + operator can be used to add two number but also can be used to add two strings together. So similar above we overload +, -, *, / with each operator represents its respective operation on complex numbers operators for two complex numbers.

Learning Outcomes :

- We have learned that by using overloading operator our program will be more understandable. However, there are three methods to implement operator overloading that are: -
- Member Function
- Non-Member Function
- Friend Function
- Operator overloading function can be a member function if the Left operand is an Object of that class, but if the Left operand is different, then Operator overloading function must be a non-member function. Operator overloading function can be made friend function if it needs access to the private and protected members of class.
- However, we cannot overload some of the operators that are given below: -
- Scope Resolution Operator - `::`
- Member Access or Dot operator - `.`
- Pointer-to-member Operator - `*`
- Ternary or Conditional Operator - `?:`

Aim:

Write a program to add, subtract, multiply and divide two complex number using **Operator Overloading** .

Theory:

Operator overloading is an important concept in C++. It is a type of polymorphism in which an operator is overloaded to give user defined meaning to it. Overloaded operator is used to perform operation on user-defined data type. For example, ‘+’ operator can be overloaded to perform addition on various data types, like for Integer, String(concatenation) etc. Almost any operator can be overloaded in C++. However, there are few operator which can not be overloaded. Operator that are not overloaded are follows :

- Scope Resolution Operator - `::`
- Member Access or Dot operator - `.`
- Pointer-to-member Operator - `*`
- Ternary or Conditional Operator - `?:`

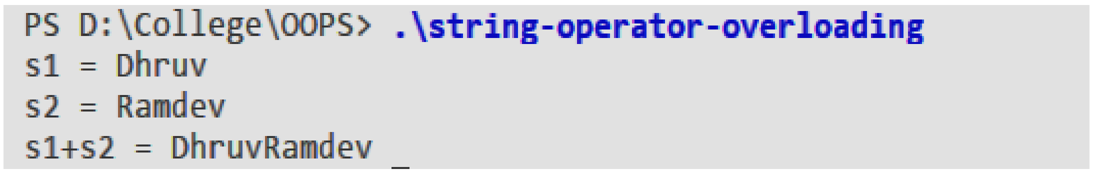
Syntax :

```
data_type classname :: operator symbol (arguments){
    //function body
}
```

Code :

```
#include <iostream>
#include <string.h>

using namespace std;

class String {
    int len ;
    char* str ;

    public :
    String () {
        len = 1 ;
        str = new char ;
        str[0] = '\0' ;

    }
    String (char* s) {
        len = strlen(s) ;
        str = new char[len + 1];
        strcpy( str , s );
    }
    void display(){
        cout << str << endl ;
    }
    String operator + ( String s ) {
        char* t = new char[len + s.len - 1] ;
        strcpy( t , str ) ;
        strcat( t , s.str ) ;
        return String(t) ;
    }

} ;

int main(){

   String s1 = "Dhruv" ;
   String s2 = "Ramdev" ;
   String s3 = s1+s2 ;
   cout << "s1 = " ; s1.display() ;
   cout << "s2 = " ; s2.display() ;
   cout << "s1+s2 = " ; s3.display() ;
    return 0 ;

}
```

Output :



```
PS D:\College\OOPS> .\string-operator-overloading
s1 = Dhruv
s2 = Ramdev
s1+s2 = DhruvRamdev _
```

Discussion :

In the above program we have created a class `String` to concatenate two strings by using operator overloading. We have overload `+` operator for adding two strings. First we created two objects of String
class and initialize two strings for two different objects and then we overload + operator or we can say that we invoke the operator function and then we created a string of third type of object which has
size of string one and string two and concatenate in it by using `strcpy()` function.

Learning Outcomes :

- We have learned that by using overloading operator our program will be more understandable. However, there are three methods to implement operator overloading that are: -
  - Member Function
  - Non-Member Function
  - Friend Function

- Operator overloading function can be a member function if the Left operand is an Object of that class, but if the Left operand is different, then Operator overloading function must be a non-member function. Operator overloading function can be made friend function if it needs access to the private and protected members of class.
- However, we cannot overload some of the operators that are given below: -
- Scope Resolution Operator - `::`
- Member Access or Dot operator - `.`
- Pointer-to-member Operator - `*`
- Ternary or Conditional Operator - `?:`

Aim:

To understand the concept of Multilevel Inheritance and implement it with the help of on example.

Theory:

One of the most important concepts in object-oriented programming is that of inheritance. Inheritance allows us to define a class in terms of another class, which makes it easier to create and maintain an application. This also provides an opportunity to reuse the code functionality and fast implementation time. When creating a class, instead of writing completely new data members and member functions, the programmer can designate that the new class should inherit the members of an existing class. This existing class is called the **base class** , and the new class is referred to as the **derived class** . In Multilevel Inheritance, a derived class can also inherited by another class. For instance, if we inherit Base class in Derived1 class and inherit Derived1 class in Derived2 class. So when a derived is inherited by another class then it creates Multiple Levels and this is known as **Multi-level Inheritance** .

Syntax :

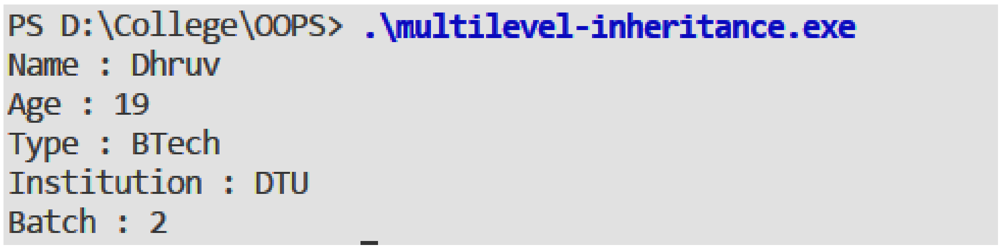
```
class A {
    // Base class
};
class B: public A {
    // B derived from A
};
class C: public B {
   // C derived from B
};
```

Code :

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

class Person {

    protected :
    char name[50] ;
    int age ;

    public :
    Person( char* s , int a ) {
        strcpy( name,s );
        age = a ;
    }

    void display() {
        cout << "Name : " << name << endl ;
        cout << "Age : " << age << endl ;
```

```cpp
    }

};

class Student : public Person  {

    protected :
    char type[10] ;
    char clg_name[50] ;

    public :
    Student( char *s  , int a , char *t , char *cn ) : Person( s , a ) {
        strcpy( type , t );
        strcpy( clg_name , cn );
    }

    void display() {
        Person :: display() ;
        cout << "Type : " << type << endl ;
        cout << "Institution : " << clg_name << endl ;
    }

} ;

class BTech : public Student {
    private :
    int batch ;

    public :
    BTech( char *s  , int a , char *t , char *cn , int b  ) : Student( s, a, t ,cn  ){
        batch = b ;
    }

    void display() {
        Student :: display () ;
        cout << "Batch : " << batch <<endl ;
    }
};

int main(){

    BTech var( "Dhruv" , 19 , "BTech" , "DTU" , 2  );
    var.display();

}
```

Output :



```
PS D:\College\OOPS> .\multilevel-inheritance.exe
Name : Dhruv
Age : 19
Type : BTech
Institution : DTU
Batch : 2
```

Discussion :

The above program illustrates the concept of multilevel inheritance. In this program, a `Person` class is made which has marks as a data member and a constructor to accept data. Class `Person` is inherited publically in derived class `Student` which accepts data and calls the constructor of `Person`. `Student` is inherited by `BTech`. This way a multilevel inheritance is carried out.

Learning Outcomes :

- Multilevel inheritance allows user to inherit classes in a level wise order.
- Reusability - facility to use public methods of base class without rewriting the same.
- Extensibility - extending the base class logic as per business logic of the derived class.
- Data Hiding - base class can decide to keep some data private so that it cannot be altered by the derived class
- Overriding -With inheritance, we will be able to override the methods of the base class so that meaningful implementation of the base class method can be designed in the derived class.

Aim:

To understand the concept of Hybrid Inheritance and implement it with the help of on example.

Theory:

One of the most important concepts in object-oriented programming is that of inheritance. Inheritance allows us to define a class in terms of another class, which makes it easier to create and maintain an application. This also provides an opportunity to reuse the code functionality and fast implementation time. When creating a class, instead of writing completely new data members and member functions, the programmer can designate that the new class should inherit the members of an existing class. This existing class is called the base class, and the new class is referred to as the derived class. Hybrid Inheritance is a type of inheritance. It is implemented by combining more than one type of inheritance. For example: Combining Hierarchical inheritance and Multiple Inheritance.

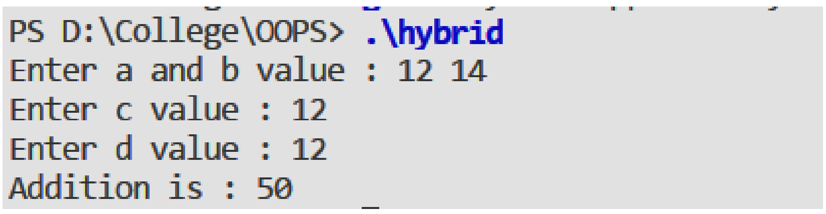
Syntax :

```
class A {

};
class B: public A {

};
class C: {

};
class D: public C public B {

};
```

Code :

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

class A {
    protected:
        int a , b ;
    public:
        void getab() {
            cout<<"Enter a and b value : ";
            cin>>a>>b;
        }
};
class B:public A {
    protected:
        int c ;
    public:
        void getc() {
            cout<<"Enter c value : ";
            cin>>c;
        }
};
class C {
    protected:
        int d ;
    public:
        void getd() {
            cout<<"Enter d value : ";
            cin>>d;
        }
};
class D:public B,public C {
    protected:
        int e ;
    public:
        void result() {
            getab();
            getc();
            getd();
            e=a+b+c+d;
            cout<<"Addition is : "<<e<<endl;
        }
};
int main() {
    D d1;
    d1.result();
    return 0;
}
```

Output :



```
PS D:\College\OOPS> .\hybrid
Enter a and b value : 12 14
Enter c value : 12
Enter d value : 12
Addition is : 50
```

Discussion :

In the above program there are four classes which are showing hybrid inheritance. Classes A and B show multilevel and classes B,C and D show multiple inheritance. So it is a type of hybrid inheritance having
a combination of multilevel and multiple inheritance. However, the invoking method is same as of previous program.

Learning Outcomes :

Inheritance is the major feature of object oriented programming we learn the following terms from inheritance: -

- Reusability : Facility to use public methods of base class without rewriting the same
- Extensibility : Extending the base class logic as per business logic of the derived class

- Data hiding : Base class can decide to keep some data private so that it cannot be altered by the derived class
- Overriding : With inheritance, we will be able to override the methods of the base class so that meaningful implementation of the base class method can be designed in the derived class.

Aim:

To understand the concept of Virtual Function (polymorphism) by creating a base class `c_polygon` which has virtual function area and two derived classes `c_rectangle` and `c_triangle` . Calculate and return area for rectangle and triangle respectively.

Theory:

A virtual function a member function which is declared within base class and is re-defined (Overridden) 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. Some of the rules to define a virtual function are:-

i. They Must be declared in public section of class.
ii. Virtual functions cannot be static and also cannot be a friend function of another class.
iii. Virtual functions should be accessed using pointer or reference of base class type to achieve run time polymorphism.
iv. The prototype of virtual functions should be same in base as well as derived class.
v. 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.
vi. A class may have virtual destructor but it cannot have a virtual constructor.

Syntax :

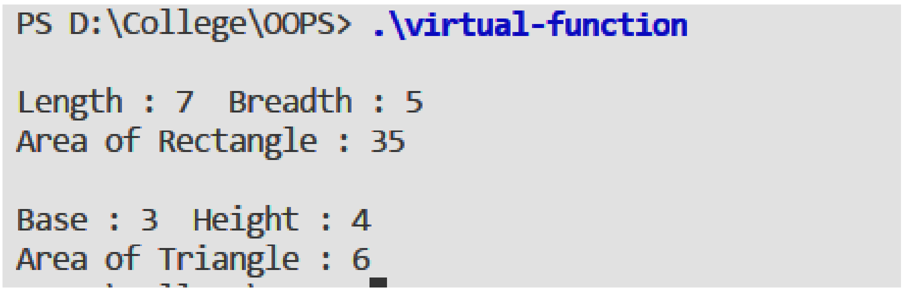
```
class Base {
    public:
        // virtual function definition with keyword virtual
};
class Derived: public A {
    // polymorphism using function overloading
};
```

Code :

```

#include <iostream>
using namespace std;

class c_polygon{
    protected :
    float width, height;
    public :
    c_polygon() { }
    void setParams(float a, float b) {
        width = a;
        height = b;
    }
```

```cpp
    virtual void getArea() {
        cout << "No shape selected" << endl;
    }
};

class c_rectangle : public c_polygon {
    public :
    c_rectangle() { }
    void getArea() {
        float area = width * height;
        cout << "\nLength : " << height << "  Breadth : " << width;
        cout << "\nArea of Rectangle : " << area << endl;
    }
};

class c_triangle : public c_polygon {
    public :
    c_triangle() { }
    void getArea() {
        float area = 0.5 * width * height;
        cout << "\nBase : " << width << "  Height : " << height;
        cout << "\nArea of Triangle : " << area << endl;
    }
};

int main(int argc, const char * argv[]) {
    // insert code here...
    c_polygon *ptr;
    c_rectangle rect;
    c_triangle tr;
    ptr = &rect;
    ptr->setParams(5,7);
    ptr->getArea();
    ptr = &tr;
    ptr->setParams(3,4);
    ptr->getArea();
    return 0;
}
```

Output :



```
PS D:\College\OOPS> .\virtual-function

Length : 7  Breadth : 5
Area of Rectangle : 35

Base : 3  Height : 4
Area of Triangle : 6
```

Discussion :

The above program illustrates the concept of virtual functions. In this program, a polygon class is made which has height and width as data members, a virtual function `getArea()` and function `setParams()` to accept parameters from user. Class `c_polygon` has two derived classes `c_triangle` and `c_rectangle` . Both derived classes have function `getArea()` to calculate area and display. Thus virtual function polymorphism is demonstrated by the three classes.

Learning Outcomes :

- 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.

Aim:

Write a program to explain class template by creating a template for a class named `pair` having two data members of type T which are given by a constructor and a member function get_max() return the greatest of two numbers to main.

Theory:

A template can be used to create a family of classes or functions. A template can be considered as a kind of macro. When an object of a specific type is defined for actual use, the template definition for that class is substituted with the required data type. Since a template is defined as a parameter that would be replaced by a specified data type at the time of actual use of the class or function, the templates are sometimes called parameterized classes or functions.

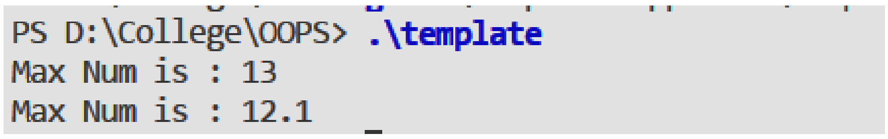
Syntax :

```
template <class T>
class classname{
       //
       // class member specifications
       // with anonymous type T
       // wherever appropriate
       //
}
```

Code :

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

template<typename T>
class Pair{
    T a;
    T b;
public:
    Pair(T x,T y=0){
        a=x;
        b=y;
    }

    T get_max(){
        return a>b?a:b;
    }
};
int main(){

    Pair <int> p1 (10,13);
    cout<< "Max Num is : "<< p1.get_max()<< endl;
    Pair <float> p2 (12.1,11.9);
    cout<< "Max Num is : " <<p2.get_max()<<endl;
    return 0;
}
```

Output :



```
PS D:\College\OOPS> .\template
Max Num is : 13
Max Num is : 12.1
```

Discussion :

The class template definition is very similar to an ordinary class definition except the prefix template `<class T>` and the use of type T. This prefix tells the compiler that we are going to declare a template and use T as a type name in the declaration.

Learning Outcomes :

Templates are a very powerful mechanism which can simplify many things. Its advantages are:

- Reducing the repetition of code (generic containers, algorithms.
- Static polymorphism and other compile time calculations
- Policy based design (flexibility, reusability, easier changes, etc.)
- Templates reduce the effort on coding for different data types to a single set of code.

Aim:

To write a program to demonstrate Exception Handling in C++ by :

- Division by zero.
- Array index out of bounds exception using multiple catch blocks.

Theory:

An exception is a problem that arises during the execution of a program. A C++ exception is a response to an exceptional circumstance that arises while a program is running, such as an attempt to divide by zero.

Exceptions provide a way to transfer control from one part of a program to another. C++ exception handling is built upon three keywords: try, catch, and throw.

- **throw** − A program throws an exception when a problem shows up. This is done using a throw keyword.
- **catch** − A program catches an exception with an exception handler at the place in a program where you want to handle the problem. The catch keyword indicates the catching of an exception.
- **try** − A try block identifies a block of code for which particular exceptions will be activated. It's followed by one or more catch blocks.

Syntax :

```
try {
   // protected code
} catch( ExceptionName e1 ) {
   // catch block
} catch( ExceptionName e2 ) {
   // catch block
} catch( ExceptionName eN ) {
   // catch block
}
```

Code :

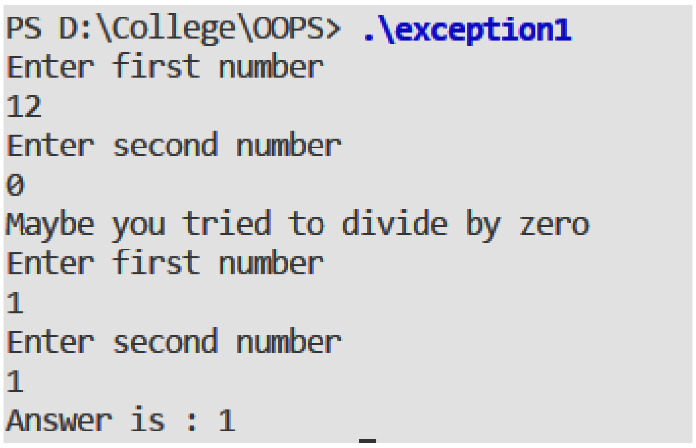
Part 1

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

int main()
{   int a,b,c;
    bool done = false;

    do
    {   cout << "Enter first number" << endl;
        cin >> a;
        cout << "Enter second number" << endl;
        cin >> b;

        try
        {   if (b == 0)
            throw "error";
            c = a/b;
            cout <<"Answer is : "<< c << endl;
            done = true;
        }
        catch(...)
        {   cout << "Maybe you tried to divide by zero"<<endl;
        }
    }
    while (! done);
    return 0;
}
```

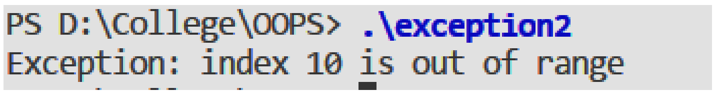
Part 2

```
#include <iostream>
using namespace std;
int main () {
    try
    {
        char * mystring;
        mystring = new char [10];
        if (mystring == NULL) throw "Allocation failure";
        for (int n=0; n<=100; n++)    {
            if (n>9) throw n;
            mystring[n]='z';
        }
    }
    catch (int i)
    {cout << "Exception: ";
        cout << "index " << i << " is out of range" << endl;  }

    catch (char * str)
    { cout << "Exception: " << str << endl;   }
    return 0;
}
```

Output :



```
PS D:\College\OOPS> .\exception1
Enter first number
12
Enter second number
0
Maybe you tried to divide by zero
Enter first number
1
Enter second number
1
Answer is : 1
```



```
PS D:\College\OOPS> .\exception2
Exception: index 10 is out of range
```

Discussion :

When the try block throws an exception, the program control leaves the try block and enters the catch statement of the catch block. Note that exceptions are objects used to transmit information about a problem. If the type of object thrown matches the argument type in the catch statement, then catch block is executed for handling the exception. If they do not match, the program is aborted with the help of `abort()` function which is invoked by default. When no exception is detected and thrown, the control goes to the statement immediately after the catch block.

Learning Outcomes :

- C++ exception handling mechanism is basically build upon three keywords namely. Try, throw, catch.
- The keyword try is used to preface a block of statements which may generate exceptions. This block of statements is known as try block.
- When an exception is detected, it is thrown using a throw statement in the try block.
- A catch block defined by the keyword catch, catches the exception thrown by the throw statement in the try block, it handles is appropriately.

Aim:

Write a program to show file handling and count number of words , characters and sentences.

Theory:

File : The information / data stored under a specific name on a storage device, is called a file.

Stream : It refers to a sequence of bytes.

Text file : It is a file that stores information in ASCII characters. In text files, each line of text is terminated with a special character known as EOL (End of Line) character or delimiter character. When this EOL character is read or written, certain internal translations take place.

Binary file : It is a file that contains information in the same format as it is held in memory. In binary files, no delimiters are used for a line and no translations occur here.

Classes for file stream operation :

- **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.

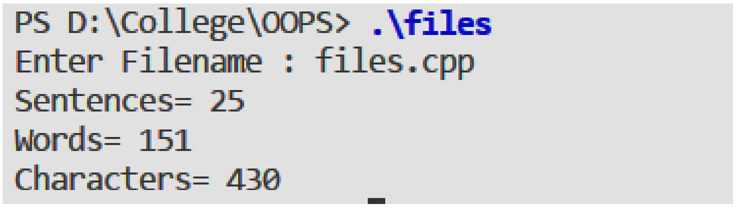
Syntax :

```
// OPENING FILE USING CONSTRUCTOR
ofstream outFile("sample.txt");    //output only
ifstream inFile("sample.txt");  //input only

// OPENING FILE USING open()
Stream-object.open(“filename”, mode)
      ofstream outFile;
      outFile.open("sample.txt");

      ifstream inFile;
      inFile.open(“sample.txt”);

// CLOSING FILE
   outFile.close();
   inFile.close();
```

Code :

```
#include<iostream>
#include<fstream>
using namespace std;
int main(){
    ifstream fin;
    char filename[20];
    cout<< "enter filename: ";
    gets(filename);
    fin.open(filename);
    int line=0,word=0,chars=0;
    char ch;
    fin.seekg(0,ios::end);
    fin.seekg(0,ios::beg);
    while(fin){
        fin.get(ch);
        if(ch!=' ' && ch!='\n') ++chars;
        if(ch==' '|| ch=='\n') ++word;
        if(ch=='\n')  ++line;

    }
    cout<< "Sentences= "<<line<< "\nWords= "<<word<< "\nCharacters= "<<chars<<endl;
    fin.close(); // closing file
    return 0;
}
```

Output :



```
PS D:\College\OOPS> .\files
Enter Filename : files.cpp
Sentences= 25
Words= 151
Characters= 430
```

Discussion :

The program opens a file named files.cpp and reads it till the end of the file is reached and during reading it counts the number of words, characters and sentences present in the file. Function `open()` is used to open a file and `close()` disconnects the file from the stream `object.seekg()` moves the get pointer (input) to a specified location.

Learning Outcomes :

- Here we learned the concept of file handling to read and write data in a file.
- Files are a means to store data in a storage device.
- C++ file handling provides a mechanism to store output of a program in a file and read from a file on the disk.
- A file can be opened in different modes to perform read and write operations.