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;
    }
```

```
    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.