UNIT- 6

Files:

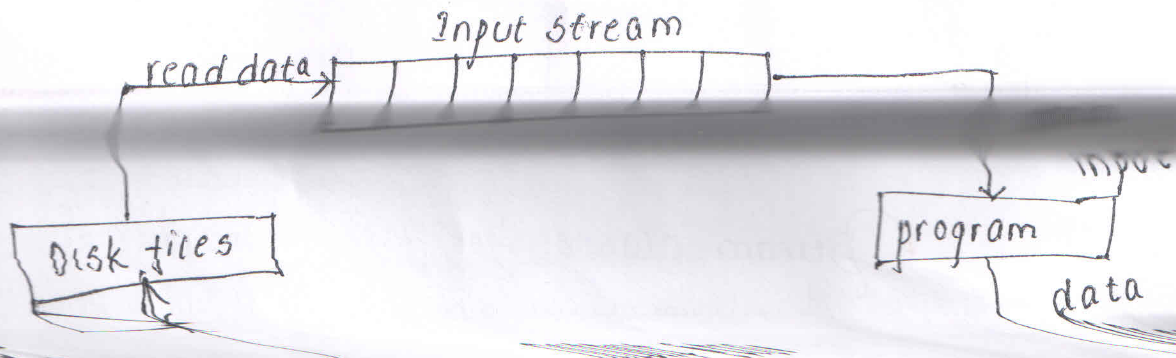
A file is a collection of related data stored in a particular area on the disk

programs can be designed to perform the read and write Operations on these files.

The I/o System of C++ handles file operations. It uses file streams as an interface between the programs and the files.

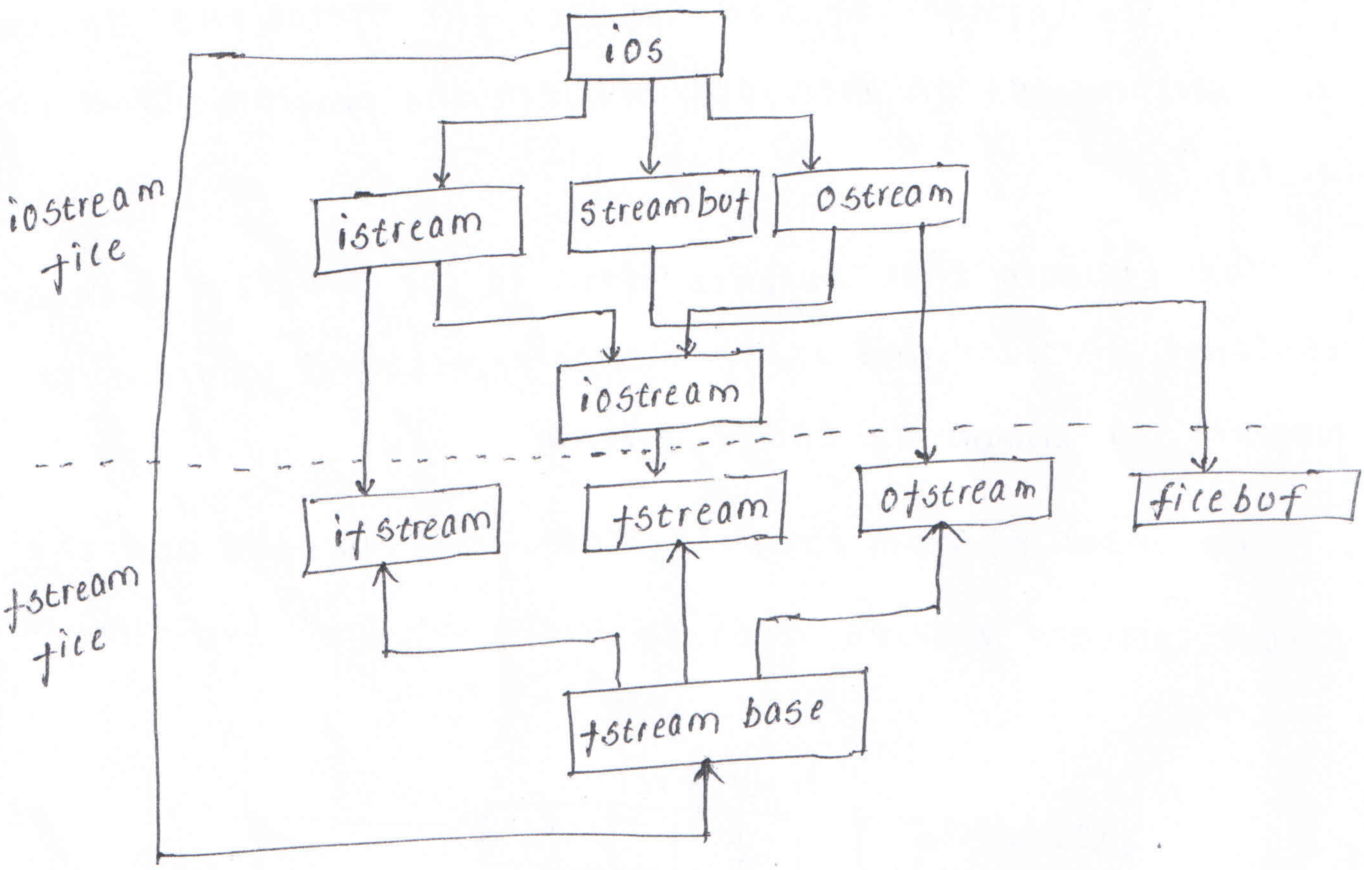
The Stream that supplies data to the program is known as input Stream and the One that receives data from the program is known as output stream.

The input Stream extracts data from the file and the Output stream inserts data to file.



Classes for File Stream Operations

The I/O system of C++ contains a set of classes that define the file handling methods. these include ifstream, ofstream and fstream. These classes are derived from fstreambase



Stream classes for file Operations (contained in fstream file)

Opening and Closing a file

If we want to use a disk file, we need to decide the following things about the file and its intended Use :

1. Suitable name for the file
2. Data type and Structure
3. purpose
4. Opening method

A file can be opened in two ways

1. Using the constructor function of the class
2. Using the member function open( ) of the class

* The first method is Useful when we Use only one file in the stream
* The Second method is Used when we want to manage multiple files Using One stream.

Opening Files Using Constructor :

A filename is Used to initialize the file stream object. This involves the following steps

1. Create a file stream object to manage the stream Using the appropriate class
2. Initialize the file object with the desired filename.

The output pointer is used for writing to a given file locatio

Default Actions:

When we open a file in read-only mode, the input pointer is automatically set at the beginning so than we can read the file from the start.

Similarly, when we open a file in write-only mode, the existing contents are deleted and the output pointer is set at the begining. This enables us to write to the file from the start

Functions for Manipulation of File pointers

All the actions on the file pointers take place automatica by default. The file stream classes support the following functions

* seekg() Moves get pointer (input) to a specified location
* seekp() Moves put pointer (output) to a specified locatio
* tellg() Gives the current position of the get pointer
* tellp() Gives the current position of the put pointer

Template:

Templates is one of the features added to C++ recently. It enable us to define generic classes and functions and thus provides support for generic programming

Generic programming is an approach where generic types are used as parameters in algorithms. So that they work for a variety of suitable datatypes & data structures

A template can be used to create a family of classes or functions. For example, a class template for an array class would enable us to create arrays of various data types such as int array and float array. Similarly, we can define a template for a function, say mul(), that would help us create various versions of mul() for multiplying int, float & double type values

A template can be considered as a kind of macro. Since a template is defined with a parameter.

The templates are sometimes called parameterized classes or functions

Class Template :

The general format of a class template is

```
template <class T >
class classname
{ //-----
  // class member specification
  // With anonymous type T
  // Wherever appropriate
  //----
};
```

The class template definition is very similar to an Ordinary class definition except the prefix template <class T> and use of type T.

The prefix tells the compiler that we are going to declare a template and use T as a type name in the declaration. T may be substituted by any data type including the user defined types.

The process of creating a specific class from a class template is called instantiation.

5

* program to illustrate the use of a Vector class template for performing the scalar product of int type vectors as well as float type vectors

Example of class template

```
#include <iostream>
#include <conio.h>
const size = 3 ;
template <class T>
class Vector
{
  T* v;              // type T vector
public :
  Vector()
  {
    v = new T[size];
    for(int i=0 ; i< size; i++)
    v[i] = 0 ;
  }
  vector(T* a )
  {
    for (int i=0 ; i<size; i++ )
    v[i] = a[i] ;
  }
```

```
T operator* (vector &y)
{
 T sum = 0;
 for (int i=0; i<size ; i++)
 sum += this→v[i] * y.v[i];
 return sum ;
 }
};

int main()
{
   int x[3] = {1,2,3};
   int y[3] = {4,5,6};
   vector <int> v1;
   vector <int> v2 ;
   v1 = x;
   v2 = y;
   int R = v1 * v2 ;
   cout << "R = " << R << "\n";
   return 0;
}
```

class Templates with Multiple parameters

```
template <class T1, class T2, ....>
class classname
{
    .....
    .....            ( Body of the class )
    .....

};
```

Two Generic Data Types in a class Definition

```
#include <iostream>
Using namespace std;
template <class T1, class T2 >
class Test
{
    T1 a;
    T2 b;
  public :
   Test (T1 x, T2 y)
    {  a=x;
       b=y;
    }
   void show()
  {
```

```
int main ()
{
   Test <float, int> test1 (1.23, 123);
   Test <int, char> test2 (100, 'W');
   test1. show();
   test2. show();
   return 0;
};
```

Function Templates

The general format of a function template is

```
template <class T>
returntype functionname (arguments of type T)
{
   // -----
```

The following example declares a swap() function template that will swap two values of a given type of data

```
template <class T>
void swap(T &x, T &y)
{
 T temp = x;
   x = y;
   y = temp;
}
```

program shows how a template function is defined and implemented

```
#include <iostream>
#include <conio.h>
template <class T>
void swap(T &x, T &y)
{ T temp = x;
    x = y;
    y = temp;
}
void fun(int m, int n, float a, float b)
{
 cout << "m and n before swap: " << m << " " << n << "\n"
```

```
int main()
  display(....);
  display(12.34,1234);
  return 0;
}
```

Overloading Of Template functions :

A template function may be overloaded either by template functions or ordinary functions. In such cases, the overloading resolution is accomplished as follows

1) call an ordinary function that has an exact match
2) call a template function that could be created with an exact match
3) Try normal overloading resolution to ordinary functions

Function with two Generic types

```
#include <iostream>
#include <string>
using namespace std;
template<class T1, class T2>

...

void display(int x) //overloads the generic display
{
cout << "Explicit display: " << x << "\n";
}
}
int main()
{
    display(100);
    display(12.34);
    display('C');
    return 0;
}
```

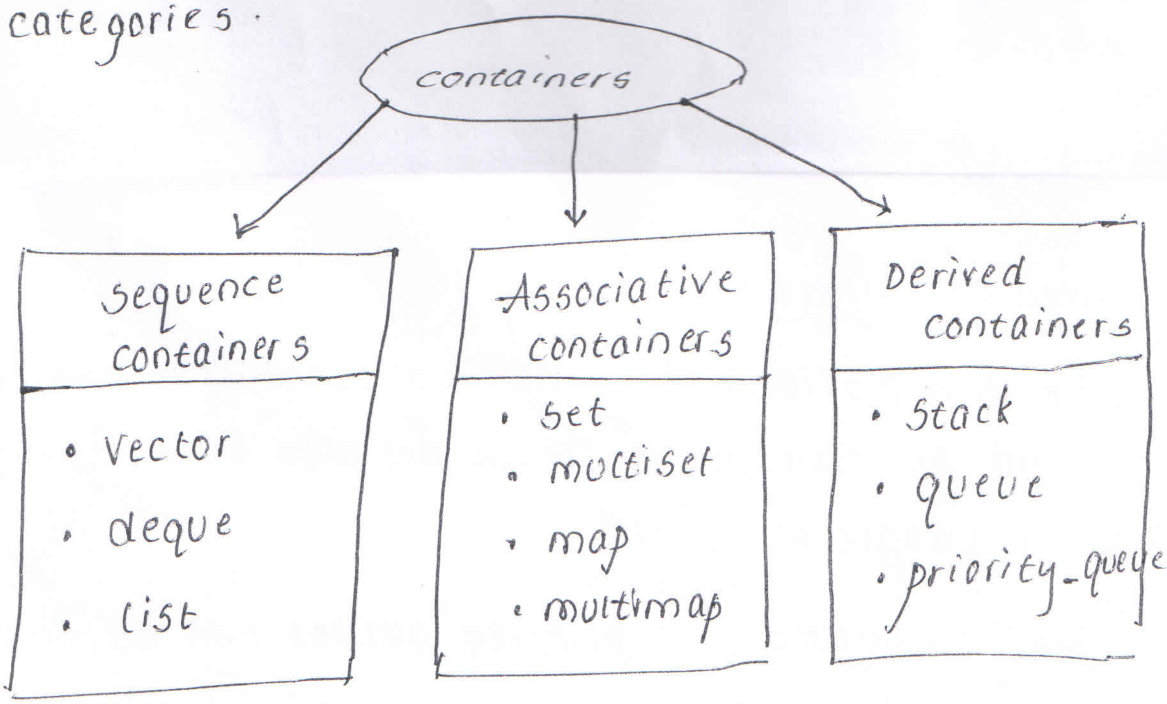
and call the one that matches.

An error is generated if no match is found

Member function Templates

The member functions of the template classes

three categories.



Standard Template Library (STL):

The collection of the generic classes and functions is called STL

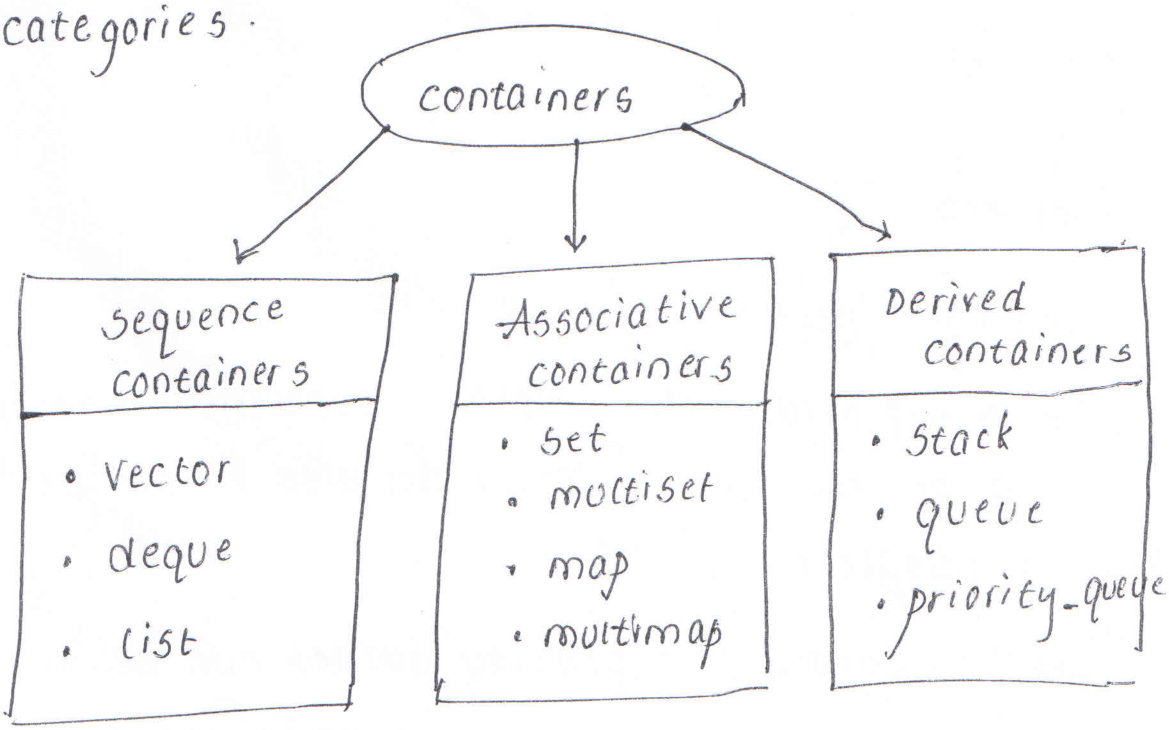
components of STL:

The STL contains several components. But there are three key components they are

- containers
- algorithms and
- iterators

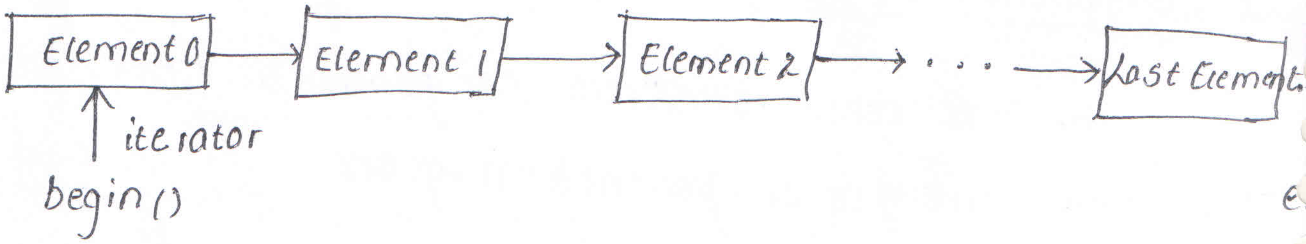
Containers:

containers are objects that hold data (of same type) The STL defines ten containers which are grouped in to three categories.



Sequence containers :

Sequence containers store elements in a linear sequen[ce] like a line. Each element is related to other elements by its position along the line



Associative Containers :

Associative containers are designed to support direct access to elements using keys. They are not sequenti[al]. There are four types of associative containers.

* set
* multiset
* map
* multimap.

support iterators and therefore we can not use them for data manipulation.

Algorithm:

An algorithm is a procedure that is used to process the data contained in the containers.

STL algorithms reinforce the philosophy of reusability. By using these algorithms, programmers can save a lot of time and effort.

Iterators:

Iterators behave like pointers and are used to access container elements. They are often used to traverse from one element to another.

→ there are five types of iterators

Iterator	Direction of movement	I/O capability
Input	forward only	Read Only
Output	forward only	Write Only
forward	forward only	Read/Write
Bidirectional	forward & Backward	Read/Write
Random	Forward & Backward	Read/Write

⇒ It is very rare that a program works correctly first time. ①

⇒ It might have bugs

⇒ The two most common types of bugs are logic errors and syntactic errors.

⇒ The logic errors occur due to poor understandin of the problem.

⇒ The syntatic errors arise due to poor understar -ing of the language itself.

⇒ we often ~~come~~ come across some peculiar probl -ms other than logic or syntax errors.

⇒ They are known as 'exceptions'. Exceptions are ru time anomalies or unusual conditions that a ~~prog~~ Program may encounter while executing.

⇒ Anomalies might include conditions such as divisic by zero, access to an array outside of its bour or running out of memory or disk space.

⇒ ANSI C++ provides built-in language refers to detec and handle exceptions.

⇒ BASICS OF EXCEPTION HANDLING:-

⇒ Exceptions are of two kinds, namely synchronous exceptions & asynchronous exceptions.

⇒ Errors such as "out-of range index" and "over-flo belong to the synchronous type exceptions.

control of the program are called asynchronous exceptions.

⇒ The exception handling suggests a seperate error handling code that performs the following tasks

1. Find the problem (Hit the exception)
2. Inform that an error has occured (Throw the excep
3. Receive the error information (catch the exception)
4. Take corrective actions (Handle the exception.

⇒ exception handling Mechanism:-

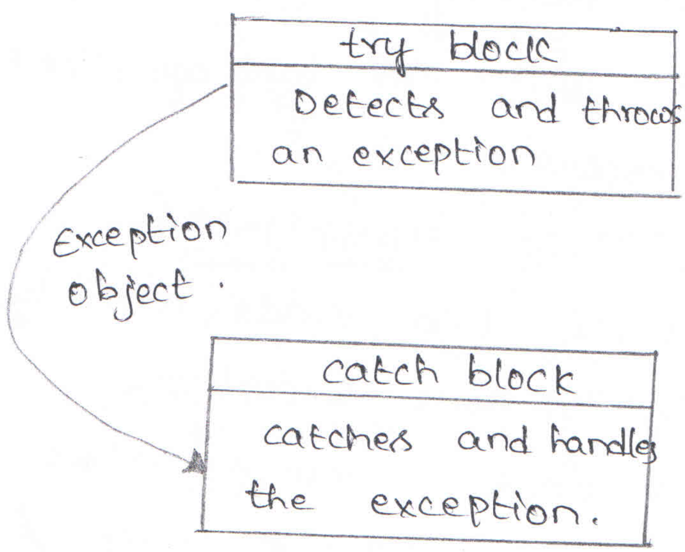
⇒ C++ exception handling Mechanism is basically bui upon three keywords, namely try, throw, and catch

⇒ The keyword try is used to perface a block of Statements (surrounded by braces) which ma generate exceptions.

⇒ This block of statements is known as try bloc

⇒ when an exception is detected, it is thrown usi a throw statement in the try block.

⇒ A catch block defined by the keyword catch catc the exception 'thrown' by the throw statement.



⇒ The general form of these two blocks are as follows:

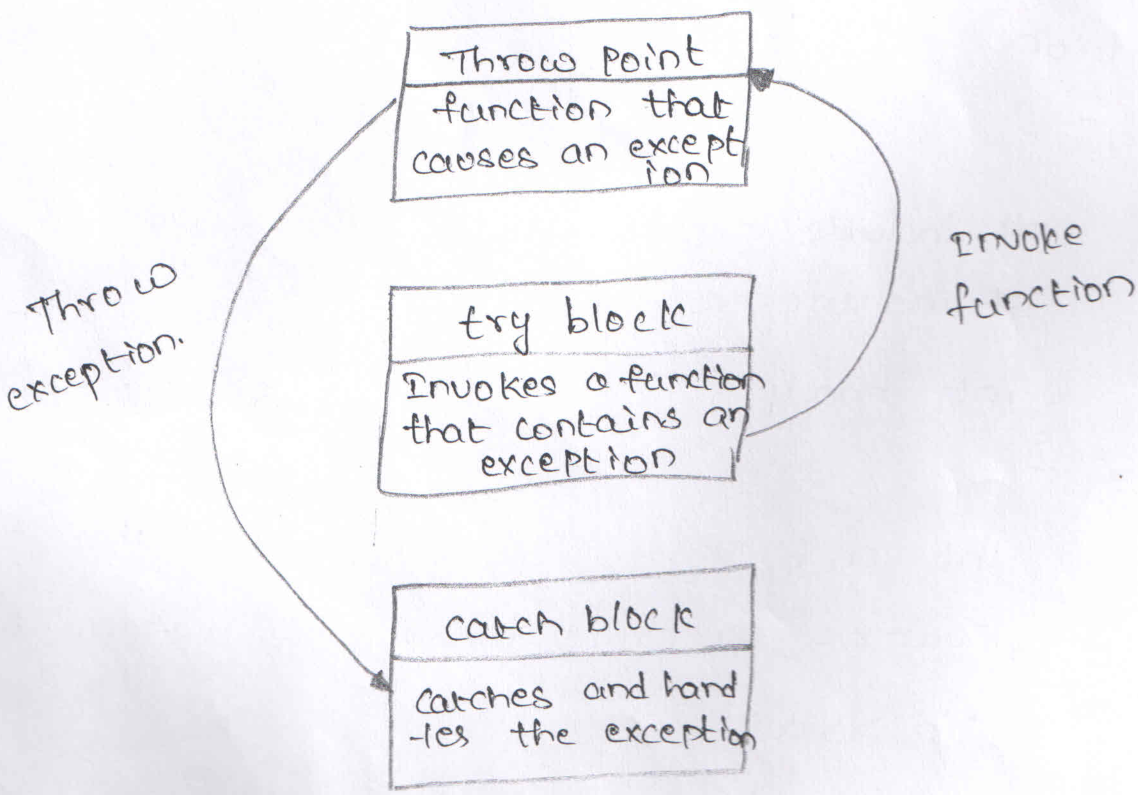
```
- - - - -
 try
{
 throw exception;   // Block of statements which
 - - - - -          // detects and throws an
 - - -                 exception.
}
catch (type arg)        // catches exception.
{
   - - - -
   - - -              // Block of statements that
   - - -
   - - -              // handles the exception.
}
- - -
- - -
```

→ Program to demonstrate the try Block throwing an exception.

Program:-

```
#include <iostream.h>
#include <conio.h>.
int main ( )
{
  int a, b;
  cout << " enter values of a and b \n";
  cin >> a;
  cin >> b;
  int x = a - b;
 { try
```

```
if (x != 0)
{
cout << "Result (a/x) = " << a/x << "\n";
}
else
{                          // There is an exception
  throw(x);                // Throws int object.
}
}
catch (int i)              // catches the exception
{
cout << " exception caught : x = " << x << "\n";
}
cout << " END";
return 0;
}
```



⇒ THROWING MECHANISM:-

When an exception that is desired to be handled detected, it is thrown using the throw statement in one of the following forms:

```
throw (exception);
throw exception;
throw;              // used for rethrowing an exception
```

⇒ CATCHING MECHANISM:-

⇒ Code for handling exceptions is included in catch blocks.

⇒ A catch block looks like a function definition and is of the form.

```
catch (type arg)
{
    // statements for
    // managing exceptions.
}.
```

⇒ Multiple catch statements:-

⇒ It is possible that a program segment has more than one condition to throw an exception.

⇒ In such cases, we can associate more than one catch statements with a try (much like the conditions in a switch statement) as shown below:

```
try
{
    // try block
}
catch (type1 arg)
```

```
{
    //catch block 1
}
catch (type2 arg)
{
    // catch block 2
}
= = .
= -
catch (type N arg)
{
    // catch block N
}.
```

=> when an exception is thrown, the exception han[dlers] are searched in order for an appropriate match.

=> The first handler that yields a match is exec[uted]. After executing the handler, the control goes to the f[irst] statement after the last catch blocks for that [try].

=> In other words, all other handlers are bypassed. [If] no match is found, the program is terminated.

=> Catch All exceptions:-

=> In some situations, we may not be able to anti-cipate all possible types of exceptions and theref[ore] may not be able to design independent catch han[dlers] to catch them.

=> In such circumstances, we can force a catch st[ate]-ment to catch all exceptions instead of a cert[ain] type alone.

=> This could be achieved by defining the cat[ch] statement using ellipses as follows:

```
catch ( ):
{
  // statements for processing
  // all exceptions
}
```

→ Program to demonstrate the catching all exception

Program:-

```
#include <iostream.h>
#include <conio.h>
void test (int x)
{
  try
  {
    if (x==0) throw x;
    if (x==-1) throw 'x';
    if (x==1) throw 1.0;
  }
  catch (...)
  {
    cout << "caught an exception \n";
  }
}

int main ( )
{
  cout << "Testing Generic catch \n";
  test (-1);
```

⇒ Rethrowing an exception:-

⇒ A handler may decide to rethrow the exception caught without processing it.

⇒ In such situations, we may simply invoke throw without any arguments as shown below:

throw;

⇒ Program to demonstrate the rethrowing an exception

Program:-

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

void divide ( double x, double y)
{
 cout << " Inside function \n";
 try
 {
   if ( y == 0.0 )
      throw y;
   else
   cout << " Division = " << x/y << "\n";
 }
 catch (double).
 {
  cout << " caught double inside function \n
  throw;
 }
 cout << " end of function \n \n";
}
int main ( )
```

```
{
  cout<<" Inside main \n";
  try
  {
   divide (10.5, 2.0);
   divide (20.0, 0.0);
  }
  catch (double)
 {
  cout<<" caught double inside main \n";
 }
 cout <<" end of main \n";
 return 0;
}.
```