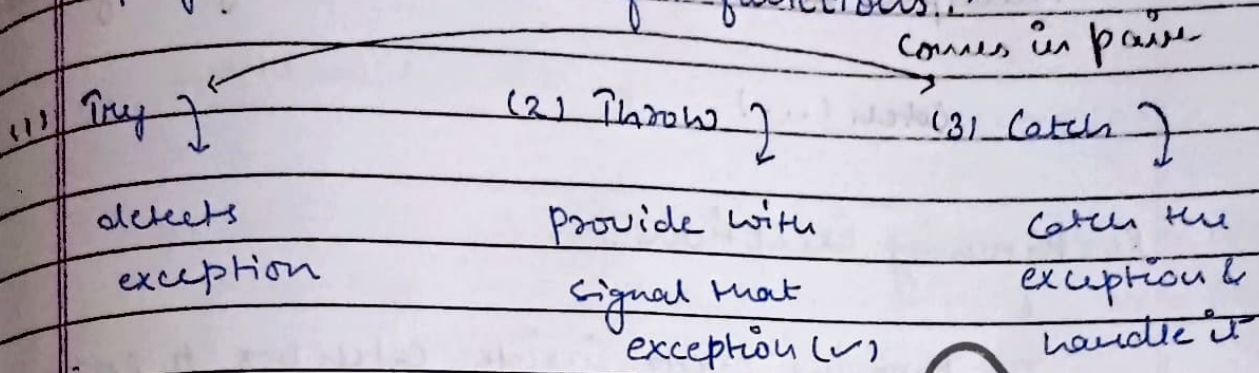


## UNIT-V

### => Exception Handling :-

It allows to manage errors without crashing program. It consists of 3 functions:-



Syntax →

```
try {
    throw exception;
}
catch ( ) {
}
```

eg:-

```
try {
    int age = 21;
    if (age >= 18) {
        cout << "Eligible"; }
```

else {

throw (age); }

can also throw customised error i.e. we can define it  
eg: throw (404);

```
catch (int Num) { (... ) can also be used if try
    cout << "Age NOT eligible"; type not known
    cout << "Age:" << Num;
}
```



Note:- Throw type can be customised

throw MyException("customised error");

- Multiple catch are used for diff types of exceptions.

catch (...) → Handle all exceptions.

Re-throwing Exceptions:-

↳ re-throwing excep. inside catch box to propg. to higher level.

eg:-

```
try {
    try {
        throw ();
    }
    catch ( ) {
        cout << " ";
        throw;
    }
}
catch ( ) {
    cout << " ";
}
```

Standard Exceptions:-

↳ base class

- runtime ⇒ occurs during prog.
- logic error ⇒ due to logical mistake.
- out of range ⇒ out of bound elements
- Invalid argument
- overflow/underflow



Start  $\rightarrow$  try block  $\rightarrow$  excep (X)  $\rightarrow$  print  $\rightarrow$  end  
 |  
 throw excep  $\rightarrow$  catch block  $\rightarrow$  handle error

## 2) Templates :-

### (1) Function

$\downarrow$   
 operates with generic data type.

### (2) Class Template

$\downarrow$   
 allow class to work with any data type.

Syntax  $\rightarrow$

template < typename T >

T add (Ta, Tb) {

return a+b; }

function Name

Parameter

Parameter

template < typename T >

class Box {

public:

T value;

Box(T val): value(val) {}

T getvalue() {

return value; }

};

### Multiple Parameters :-

template < typename T, typename U >

T multiply (Ta, Ub) {

return a\*b;

}

template < typename T >

void \_ ( ) {

cout << " " ; }

Templates can be specialized  
 i.e. print < char > / print < T >

Non - Type Parameter  $\rightarrow$  size, Array size



⇒ Stream class :-

↳ handles I/P, O/P operations

(1) Input

(cin, ifstream)



read data

(2) output

(cout, ofstream)



write data

Common Stream class :-

(1) ifstream (operator >>)



I/P

(2) fstream (operator << >>)



I/P + O/P

(read + write)

operator <

(3) ofstream



O/P

eg:-

O/P { ofstream outfile ("example.txt");  
outfile << " " ;

I/P { ifstream infile ("example.txt");  
Stringline;  
while (getline (infile, line)) {  
cout << " " ; } }

outfile.is\_open()

↳ used to check if file was opened successfully

outfile.close()

↳ closes file after writing

getline

↳ read line from file

infile.close()

↳ closes file after reading



fstream →

```
fstream file("example.txt", ios/out);
if (file.is_open()) {
    string line;
    while (getline(file, line)) {
        cout << line; }
    file << "    ";
    file.close(); }
else {
    cout << "    "; }
return 0; }
```

File opening Mode:-

in → read mode  
 out → write mode  
 app → append mode (data written at end)  
~~ate~~ <sup>open</sup> → move pointer to end  
 binary → open in binary mode (text mode x)

⇒ File Handling:-

- File stream class
- Opening file →

```
ifstream inputfile;
inputfile.open("example.txt");
```

- checking if file open →

```
if (inputfile) {
    cout << "    "; }
```

\* Error Handling

- Binary file Handl.
- File pointer

- Reading file
- Writing file
- Closing file