DATE : / /

PAGE NO. :

UNIT - V

=> **Exception Handling :-**

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

Comes in pair

(1) Try → detects exception

(2) Throw → Provide with signal that exception (v)

(3) Catch → Catch the exception & handle it

Syntax →

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

eg:-

```
try {
  int age = 21;
  if (age >= 18) {
       cout << " Eligible"; }
  else {
     throw (age); }
  }
Catch (int Num) {
  Cout << " ~~Acc~~ Not eligible";
  Cout << " Age :" << Num;
}
```

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

int Num → (...) can also be used if try type not known

DATE: / /
PAGE NO.:

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

DATE : / /

PAGE NO. :

$$\begin{tikzcd}
\text{Start} \arrow[r] & \text{try block} \arrow[r] \arrow[d, dash] & \text{excep (X)} \arrow[r] & \text{Print} \arrow[r] \arrow[d, dash] & \text{end} \\
& \text{throw excep} \arrow[rr] & & \text{catch block} \arrow[r] & \text{Handle error}
\end{tikzcd}$$

⇒ **Templates :-**

(1) Function
↓
operates with generic data type.

(2) class Template
↓
allow class to work with any data type.

eg:- default - parameter

Syntax →

```
template <Typename T>
  T add (T a, T b){
    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 Parameter :-

```
template < Typename T, Typename U >
  T multiply (Ta, Ub){
     return a*b;
  }
```

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

```
Template < Typename T>
void __ ( ) {
cout << " " ; }
```

Non - Type Parameter → size, Arraysize

DATE: / /

PAGE NO.:

⇒ 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 ↓ I/P + O/P (read + write)

(3) ofstream ← operator (<<) ↓ 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

DATE : / /

PAGE NO. :

fstream →

```
fstream file(" example. txt", in|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 →	open & 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 (input-file) {
      cout << "      "; }
```

- Reading file
- writing file
- closing file

*. Error Handling
- Binary file Handl.
- File pointer