UNIT- IV

Ⓐ ①

Exception Handling

* What are Exceptions?

- Exceptions are unusual conditions in a program, that occur at runtime.
- They could be errors that cause the programs to fail.

* Exceptions ~~are~~ of two types:

(i) Synchronous exceptions

- Occur during program execution, due to some fault in input-data, within the program
(Eg). out-of-range index, running out of memory, overflow, underflow, not being able to open a file etc.

(ii) Asynchronous exceptions

- caused by events or faults unrelated (external) to the program and beyond the control of program.
(Eg) keyboard interrupts, hardware malfunctions, disk failure etc.

* The exception handling mechanism in C++ can handle only synchronous exceptions caused within a program.

* The purpose of exception handling mechanism is
- to provide a means to detect and report an "exceptional circumstance" &
- to take appropriate action.

* Also, prevents user from seeing complex technical error msg & display non-technical msgs to user.

* The error handling code in exception handling mechanism performs the following tasks:
(i) Find the problem (Hit the exception)
(ii) Inform that an error has occured (Throw the exception)
(iii) Receive the error information (catch the exception)
(iv) Take corrective actions (Handle the exception)

Exception handling mechanism

* The exception mechanism in C++ uses three keywords :
  - try
  - throw
  - catch

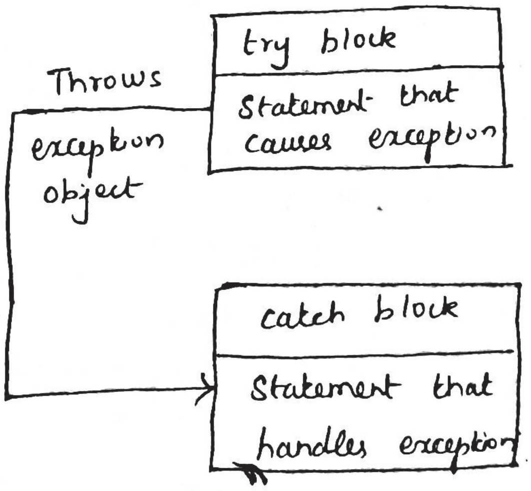
* try :
  - The try block defined by the keyword 'try' encloses the piece of code which may generate exceptions

* throw :
  - 'throw' keyword is used to throw the exception encountered inside try block.
  - After the exception is thrown, the control is transferred to catch block.

* catch :
  - 'catch' keyword defines the catch block that catches the exception thrown by throw stmt from try block
  - The exceptions are then handled inside catch block. Hence the catch statement is called 'Exception handler'

Syntax:

```
try
{
  statements ;
    :
  throw exception;
}
catch (type arg)
{
  statements ;
    :
}
```



try block
Statement that causes exception
Throws
exception object
catch block
Statement that handles exception

Eg) Exception handling for divide by zero.

(i) Single exception handling using single catch block.

```
# include <iostream.h>

void main( )
 {
  int num , den , result ;
  cout << "Enter the numerator:\n";
  cin>> num ;
  cout << "Enter the denominator:\n" ;
  cin>> den ;
  try
  {
   if (den ==0)
     {
      throw den ;
     }
    result = num/den ;
    cout <<" Result : "<< result ;
   }
  catch (int n)
    {
    cout << " Denominator cannot be "<< n;
    }
 }
```

o/p :

```
Enter the numerator:
10
Enter the denominator:
0
Denominator cannot be 0.
```

(ii) Multiple Exceptions handling using Multiple catch block

```
#include <iostream.h>
void main()
 {
 int num, den, result;
  cout << "Enter the Numerator\n";
  cin>> num;
  cout << "Enter the denominator\n";
  cin>>den;
  try
  {
  if (den == 0)
    {
    throw den;   // throws integer parameter.
    }
  else if (den <0)
    {
    throw "Negative denominator not allowed\n"; // throws string
    }                                                as parameter.
    result = num/den;
    cout<<"Result : \n" << result;
   }
  catch (int n)
   {
   cout << "Denominator cannot be "<<n;
   }
   catch (char *msg)
    {
     cout <<msg;
    }
 }
```

* Here the program can throw two exceptions - using integer (or) string parameter.
* Hence two catch blocks are defined - one for integer and other for accepting string parameter.

General purpose catch block for handling all exceptions

- C++ supports a feature to catch all exceptions raised in try block using a single general purpose catch block.
- It is defined by the syntax

```
catch (...)
{
  stmts;
}
```

three dots indicate this catch block is common and can handle all exceptions.

Note: All specialized catch blocks with parameters (if present) should be written before this general purpose catch block

(Eg) (iii)
```
#include <iostream.h>
void main( )
{
int num, den, result;
  :
  :
try
{
if (den==0)
  {
  throw den;
  }
else if (den<0)
  {
   throw "Negative denominator not allowed\n";
  }
  result = num/den;
  cout<< "Result : \n" << result;
}
catch (...)
  {
  cout << "Problem in calculation. Check denominator\n";
  }
}
```

Program to check array index out of bounds

a)

```
#include <iostream.h>
class list
{
 int l[10];
 public :
   int & operator [] (int index)
    {
     if (index > 9)
        throw "Index out of bounds\n";
     return l[index];
    }
};
void main()
{
 list l1;
 try
  {
   l1[2] = 10;
   cout << "Value of l1[2]:" << l1[2] << endl;
   l1[15] = 6;
   cout << "Value of l1[15]:" << l1[15];
  }
  catch (char *msg)
   {
    cout << msg;
   }
 }
```

O/P:

```
Value of l1[2] : 10
Index out of bounds
```

b)

```
#include <iostream.h>

Class list
{
 int l[10];
public :
  class range{}; //abstract clas
  int & operator[] (int index
   {
   if (index > 9)
      throw range(); // throw
                         abstract
   return l[index];        obj.
   }
};
.
.
.
.
.
.

 catch (list :: range)
 {
  cout << "Array index out
             of bounds\n";
  }
}
```