Exception Handling

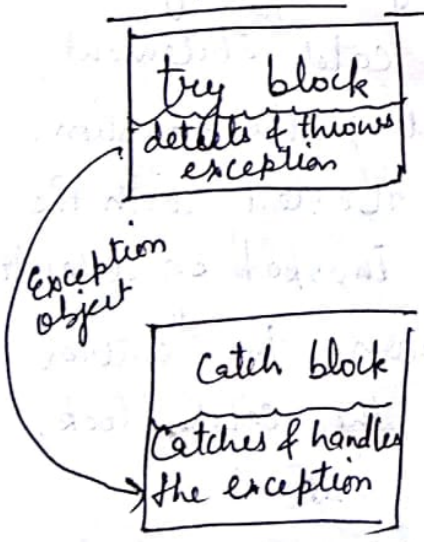
RS-46

Exceptions :- Exceptions are run-time anomalies or unusual conditions that a program may encounter while executing. for e.g. division by zero, access to an array outside of its bounds, running out of memory or disk space.

Exceptions

- Synchronous
  - eg. out-of-range index, overflow
  - ~~-~~
  - ask handled by exception handling mechanism
- Asynchronous
  - eg. keyboard interrupts
  - errors that are caused by events beyond the control of program.
  - not handled by exception handling mechanism.

* Exception Handling Mechanism:-




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

C++ exception handling mechanism consists of [illegible] block shown in fig i.e. try block & catch block.

try block :- The keyword try is used to preface a block of statements which may generate exceptions. This block is called try block. When an exception is detected it is thrown by using a throw statement.

Catch block :- A catch block defined by the keyword catch catches the exception thrown by the 'throw' statement in the 'try' block, and handles it appropriately. The catch block that catches an exception must immediately follow the 'try' block that throws the exception.

- When 'try block' throws an exception, the program control leaves the try block and enters the 'catch' statement of the catch block. If the type of object thrown matches the arg type in the catch statement, then 'catch block' is executed for handling the exception. If they donot 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. That is, catch block is skipped.

ogram :- Try block throwing an Exception.

```
# include <iostream.h>
using namespace std;
void main()
{ int a,b;
cout << "Enter a, b";
cin >> a >> b;
int x= a-b;
try
{ if (x != 0)
  {cout << "Result of a/x=" << a/x ;
  }
  else
  {
    throw (x);
  }
}
Catch (int i)
{
  cout << "Exception caught: Divide by zero";
}
cout << "end";
}
```

o/p.

```
Enter a, b
20 15
Result of a/x = 4
End
```

o/p.

```
Enter a, b
10 10
Exception caught:
Divide by zero
End
```

Invoking function that generates exception :-

```
return type function (arg list)
{
  ...
  throw(object); //throws exception here
  ...
}
...
try
{
  ...
  ... Invoke function here
}
catch (type arg)
{
  ...
  ... Handles exception here
}
...
```

→ function that causes exception

→ throw point

The point at which throw is executed. Once an exception is thrown to catch block control can't return to the throw point

The try block is immediately followed by the catch block, irrespective of the location of the throw point.

eg

```
#include <iostream.h>
using namespace std;
void divide (int x, int y, int z)
{ cout << " Inside function ";
  if ((x-y) != 0)
  { int r = z / (x-y);
    cout << " Result = " << r;
  }
  else
  { throw (x-y); // throw Point
  }
}
int main()
{ try
  { cout << " Inside try ";
    divide (10,20,30);
    divide (10,10,20)
  }
  catch (int i)
  {
  cout << "Caught exception";
  }
  return 0;
}
```

o/p.

```
Inside try
Inside function
Result = -3
Inside function
caught exception
```

Throwing Mechanisms—

1. throw (exception);
2. throw exception;
3. throw;
   ↓
   used for rethrowing an exception.

the operand exception may be of any type including constants. It is also possible to throw objects not intended for error handling.

* Catching Mechanism :-

A catch block looks like a function definition.

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

type → indicates the type of exception that the block handles.

arg → is an optional parameter name.

* Multiple Catch Statements :-

It is possible that a program has more than one condition to throw an exception. In such cases, we can associate more than one catch statement with a try.

```
try
{
  // try block
}
Catch (type1 arg)
  {
    // Catch block 1
  }
Catch (type2 arg)
  {
  }
=
Catch (typeN arg) { ... }
```

When an exception is thrown, the exception handlers are searched in order for an appropriate match. The first handler that matches, is executed & after that control goes to the first statement after last catch block for that try. When no match is found, the program is terminated.

It is possible that arguments of several catch statement match the type of an exception. In such cases, the first handler that matches the exception type is executed.

eg

```
#include <iostream.h>
using namespace std;
void test (int x)
{
  try
  {
    if (x==1) throw x ;    // int
    else
      if (x==0) throw 'x' ;   // char
    else
      if (x== -1) throw 1.0  // float
      cout << "End of try-block";
  }
catch (char c)
  {
    cout << "Caught character";
  }
catch (int m)
  {
    cout << " Caught integer";
  }
catch (float f)
  {
    cout << " Caught float";
  }
cout << "End of try-catch system";
}
```

```
int main()
{
  cout << "Testing";
  cout << "x==1";
  test(1);
  cout << "x==0";
  test(0);
  cout << "x==-1";
  test(-1);
  cout << "x==2";
  test(2);
  return 0;
}
```

o/p

```
Testing
x==1
Caught integer
End of try-catch system

x==0
Caught character
End of try-catch system

x==-1
Caught float
End of try-catch system

x==2
End of try-block
End of try-catch system
```

→ **Catch All Exceptions :-**

In some situations, we may not be able to anticipate all possible types of exceptions and therefore may not be able to design independent catch handlers to catch them. In such cases, we can force a catch statement to catch all exceptions instead of certain type alone.

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

→ All throws can be caught by catch all statement

→ It should always be placed last in the list of handlers. Placing it before other catch blocks would prevent those blocks from catching exceptions.

Eg

```
#include <iostream.h>
void test (int x)
{
  Try
  {
   if (x==0) throw x;      // int
   if (x==-1) throw 'x' ;  // char
   if (x==1) throw 1.0 ;   // float
  }
  catch (...)
  {
    cout << "Caught Exception";
  }
}
int main()
{
  cout << "Testing";
  test (-1);
  test (0);
  test (1);
  return 0;
}
```

o/p

```
Testing
Caught Exception
Caught Exception
Caught Exception
```

* Rethrowing an Exception :-

A handler may decide to rethrow the exception caught without processing it. In such situations, we invoke throw without any arguments.

```
throw ;
```

This causes the current exception to be thrown to next enclosing try/catch sequence and is caught by catch statement listed after the enclosing try block.

```
#include <iostream.h>
void divide ( double x, double y)
{
  Cout << " Inside function";
  try
  {
    if (y==0.0)
     throw y;      // double thrown
    else
     cout << "Division = " << x/y;
  }
  Catch (double)
  {
   Cout << " Caught double inside function";
   throw ;          // Rethrowing double
  }
  Cout << " End of function;
}
int main()
{
 Cout << " Inside main";
 try
  {
    divide (10.5, 2.0);
    divide (20.0, 0.0);
  }
  Catch (double)
  {
   cout << " Caught double inside main";
  }
  cout << " End of main";
 return 0;
```

output

```
Division = 5.25
End of function
Inside function
Caught double inside function
Caught double inside main
End of main
```

* Specifying exceptions :- It is possible to restrict a function to throw only certain type of exceptions. This is done by adding a throw-list clause to the function definition. i.e.

```
type function-name(arg-list) throw (type-list)
{
  // function body
}
```

type-list → specifies the type of exceptions that may be thrown.

Throwing any other type of exception will cause abnormal program termination. If we wish to prevent a function from throwing any exception, we may do so by making type-list empty.
i.e. we must use `throw();` in the function definition line.

E.g.

```
#include <iostream>
void test(int x) throw (int, double)
{
  if (x==0) throw 'x';      // char → not allowed.
  else
    if (x==1) throw x;      // int → allowed
  else
    if (x==-1) throw 1.0;   // double → allowed
  cout << "End of function block";
}
int main()
{
  try
  {
    cout << "Testing";
    cout << "x==0";
    test(0);
```

```
cout << "x == 1";
test (1);
cout << "x == -1";
test (-1);
cout << "x == 2";
test (2);
}
catch (char c)
{
cout << "Caught character";
}
catch (int m)
{
cout << "Caught int";
}
catch (double d)
{
cout << "Caught double";
}
cout << "End of try-catch system";
return 0;
}
```

o/p.

```
Testing
x == 0
Caught character
End of try-catch system
```

Termination of program

! wrong throw 'char' was not allowed.