EXCEPTION HANDLING IN C++

EXCEPTIONS

- Exceptions are run time anomalies or unusual conditions that a program may encounter during execution.
- Conditions such as
    - Division by zero
    - Access to an array outside of its bounds
    - Running out of memory
    - Running out of disk space
- It was not a part of original C++.
- It is a new feature added to ANSI C++.

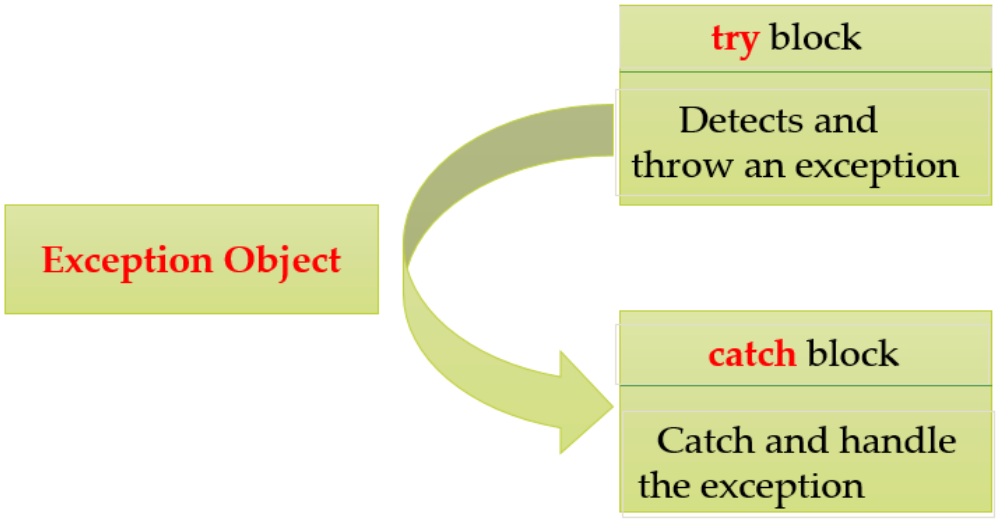
EXCEPTION HANDLING

- Exceptions are of 2 kinds
    - **Synchronous Exception:**
        - Out of rage
        - Over flow
    - **Asynchronous Exception:** Error that are caused by causes beyond the control of the program
        - Keyboard interrupts
- In C++ only synchronous exceptions can be handled.

- Exception handling mechanism
    - Find the problem (Hit the exception).
    - Inform that an error has occurred (Throw the exception).
    - Receive the error information (Catch the exception).
    - Take corrective action (handle the exception).

- Exception handling mechanism

- Find the problem (Hit the exception).
- Inform that an error has occurred (Throw the exception).
- Receive the error information (Catch the exception).
- Take corrective action (handle the exception).



- The keyword **try** is used to preface a block of statements which may generate exceptions.
- When an exception is detected, it is thrown using a **throw** statement in the try block.
- A **catch** block defined by the keyword 'catch' catches the exception and handles it appropriately.
- The catch block that catches an exception must immediately follow the try block that throws the exception.

```
try
{
…                    // Block of statements
                     // which detect and throws an exception
throw exception;
…
}
        catch(type arg)
{
…
```

…

…

}

- Exceptions are objects used to transmit information about a problem.
- If the type of the object thrown matches the arg type in the catch statement, the catch block is executed.
- If they do not match, the program is aborted.

THROWING MECHANISM

- The **throw** statement can have one of the following 3 forms
  - throw(exception)
  - throw exception
  - throw //used to re-throw a exception
- The operand object exception can be of any type, including **constant**.
- It is also possible to throw an object not intended for error handling.
- Throw point can be in a deeply nested scope within a try block or in a deeply nested function call.
- In any case, control is transferred to the catch statement.

CATCHING MECHANISM

- The type indicates the type of exception the catch block handles.
- The parameter arg is an **optional** parameter name.
- The catch statement catches an exception whose type matches with the type of the catch argument.

```
catch(type arg)
{
    …
    …
    …
}
```

- If the parameter in the catch statement is named, then the parameter can be used in the exception handling code.
- If a catch statement does not match the exception it is skipped.
- More than one catch statement can be associated with a try block.

```
try
{
   throw exception;
}
catch(type1 arg)
{
        // catch block 1
}
catch(type2 arg)
{
        // catch block 2
}
…
…
catch(typeN arg)
{
       // catch block N
}
```

- When an exception is thrown, the exception handlers are searched **in order** for a match.
- The first handler that yields a match is executed.
- If several catch statement matches the type of an exception the first handler that matches the exception type is executed.
- Catch all exception

```
catch (…)
{
    // statement for processing all exceptions
}
```

RETHROWING AN EXCEPTION

- A handler may decide to rethrow the exception caught without processing it.
- In such a case we have to invoke **throw** without any arguments as shown below

  throw;

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