    - Developers can focus on the main logic without getting distracted by error-related clutter.
4. Accurate Error Reporting:
    - Exceptions provide meaningful error messages, making it easier to identify the cause of failures.
    - When an exception occurs, you can include relevant information (such as stack traces) to pinpoint the issue.
5. Facilitating Debugging and Troubleshooting:
    - Exception stack traces help you trace the sequence of method calls that led to the error.
    - Debugging becomes more efficient because you can quickly locate the problematic code.
6. Improved Security:
    - Proper exception handling prevents security vulnerabilities caused by unexpected behavior.
    - By handling exceptions gracefully, you reduce the risk of exposing sensitive information or allowing unauthorized access.
7. Better User Experience:
    - When exceptions are handled well, users experience fewer crashes or abrupt program terminations.
    - A smooth user experience contributes to overall satisfaction with your software.
8. Enabling Error Recovery Mechanisms:
    - Exceptions allow you to recover from errors gracefully.
    - You can catch exceptions, log relevant information, and take corrective actions without disrupting the program flow.

1.3 Exception Handling Mechanism

Exception handling in C++ provides a structured way to handle runtime errors or exceptional conditions that may occur during program execution, allowing the program to recover without crashing.
It involves three key components: try, throw, and catch.

- **try Block:** The try block encloses the code that might throw an exception. If an exception occurs within the try block, it is transferred to the appropriate catch block.
- **throw Statement:** The throw statement is used to explicitly throw an exception. It can be followed by an expression, which is typically an object of an exception class, to provide information about the error.
- **catch Block:** The catch block catches and handles exceptions thrown by the associated try block. It specifies the type of exception it can handle and provides code to deal with the exception.

A. Throwing Mechanism:

- The throwing mechanism is used to signal that an exceptional condition has occurred during program execution. It involves the throw statement.

- **throw Statement:** The throw statement is followed by an expression, typically an object of an exception class. This expression is the exception that is being thrown.
- Throwing Syntax:

```
throw ExceptionType(arguments);
```

- Example:

```
throw runtime_error("Division by zero");
```

B. Catching Mechanism:

- The catching mechanism is used to handle exceptions thrown by the try block. It involves catch blocks.
- **catch Block:** A catch block is associated with a try block and specifies the type of exception it can handle. When an exception of that type is thrown, the corresponding catch block is executed.
- Example:

```
catch (const runtime_error& e) {
    cout << "Error: " << e.what() << endl;
}
```

- try-catch Syntax:

```
try {
    // Code that may throw an exception
} catch (ExceptionType1 e1) {
    // Code to handle ExceptionType1
} catch (ExceptionType2 e2) {
    // Code to handle ExceptionType2
} catch (...) {
    // Catch-all block to handle any other exceptions
}
```

C. Rethrowing an Exception:

- Rethrowing an exception allows an exception caught in one catch block to be thrown again to be handled by another catch block.
- **throw Statement (inside catch block):** In a catch block, we can use the throw statement without any argument to rethrow the caught exception.
- Rethrow Syntax:

```
catch (ExceptionType e) {
    // Code to handle the exception
    throw; // Rethrow the exception
}
```

- Example:

```
catch (const runtime_error& e) {
    cout << "Caught error: " << e.what() << endl;
    throw; // Rethrow the exception
}
```