Chapter 16:

Exceptions, Templates, and the Standard Template Library (STL)
What do you do when a function receives invalid input?

```cpp
int getValue(int index)
{
    if(index < 0 || index >= size)
    {
        cout << "Invalid index" << endl;
        return -1; // return a dummy value
    }
    else return array[index];
}
```
**ERROR HANDLING**

- But wait....
- Suppose the size of array was 10
- Suppose returned value was used as an index of another array

```java
int result = otherArray[ getValue(12) ];
```

- An error message will be printed (there is no element 12)
- Then, `getValue()` will return a -1.
- Invalid index!

**Conclusion:** naïve error checking is not always enough, and not always safe
**ERROR HANDLING**

- But wait....
- Suppose the size of array was 10
- Suppose returned value was used as an index of another array

```java
int result = otherArray[ getValue(12) ];
```

- An error message will be printed (there is no element 12)
- Then, `getValue()` will return a -1.
- Invalid index!

**Conclusion:** naïve error checking is not always enough, and not always safe
ERROR HANDLING

- What if:
  ```
  double divide(double num, double den)
  {
    if(den == 0) {
      cout << "Error: Division by zero\n";
      return 0; // dummy value
    }
    else return num / den;
  }
  ```

- What if the dummy value is used in further calculations?
- **Once again:** naïve error checking is not always enough, and not always safe
EXCEPTIONS: ELEGANT ERROR HANDLING

- In C++, a **throw statement** is used to signal that an exception or error has occurred.
- “Throw” an exception:

  ```cpp
double divide(double num, double den)
  {
      if(den == 0) {
          throw "Error: Division by zero\n";
      }
      else return num / den;
  }
```

- Instead of returning a meaningless dummy value, we will **terminate the function** immediately and **indicate error**
Exceptions: Elegant Error Handling

What exceptions can you throw?

- `throw -1; // throw a value`
- `throw ENUM_INVALID_INDEX; // throw an enum value`
- `throw "Division by zero"; // throw a char const *`
- `throw dX; // throw a predefined variable`
- `throw MyException("Fatal Error 404"); // Throw an object of class MyException`
A thrown exception can be “caught”:

```cpp
double x = 25, y = 0, result;
try {
    result = divide(x, y); // throws an exception
} 
catch(char const * errMessage) {
    cout << errMessage;
}
```

You can throw more than one exception from the try block, and provide as many catch blocks as necessary
Exceptions - Terminology

- **Exception**: object or value that signals an error

- **Throw an exception**: send a signal that an error has occurred

- **Catch/Handle an exception**: process the exception; interpret the signal

- If the exception was caught, the program carries on executing

- What if you do not catch a thrown exception?
  - **See example code**
**Exceptions – Flow of Control**

1) A function that **throws an exception** is called from within a **try block**.

2) If the function throws an exception, the function **terminates** and the try block is immediately **exited**. A **catch block** to process the exception is searched for in the source code immediately following the try block.

3) If a **catch block** is found that matches the **exception thrown**, it is executed. If no catch block that matches the exception is found, **the program terminates**.
**Exception Not Caught?**

- An exception will not be caught if:
  - it is thrown from outside of a `try{}` block
  - there is no `catch{}` block that matches the data type of the thrown exception
- If an exception is not caught, the program will terminate

```java
double x = 25, y = 0, result;
try
{
    result = divide(x, y); // throws an exception
}
// terminate the program - no “catch”!
```
EXCEPTIONS AND OBJECTS

- **Classes** may require more detailed exceptions than just a single value or an error message.
- An **exception class** can be defined within a class and thrown as an exception by a member function.
- An exception class may have:
  - no members: used only to signal an error
  - members: pass error data to catch block
- A class can have more than one exception class.

```cpp
class Rectangle
{
    public:

    class NegativeWidth{}; // Exception classes
    class NegativeLength{}
    class NegativeHeight{}; // ...followed by the
                               // rest of the class

}; // see example code!
```
WHAT HAPPENS AFTER CATCH BLOCK?

- Once an exception is thrown, the program cannot return to throw point.
- The function executing the throw terminates (does not return)
- Other calling functions in try block terminate
- This is known as unwinding the stack

- If objects were created in the try block and an exception is thrown, they are destroyed. (Destructor is automatically executed)
**Nested Try Blocks**

- **try/catch** blocks can occur within an enclosing **try** block.

- Exceptions caught at an inner level can be passed up to a **catch** block at an outer level:
  ```java
  catch ( int )
  {
      ...
      throw; // pass exception up
  } // to next level
  ```

- Known as “re-throwing exceptions”
double function1(double x) {
    try {
        if(x < 0) throw -1;
        return sqrt(x);
    } catch (int) {
        cout << "negative value";
        throw; // re-throw: let the function who
        // called you handle it
    }
}

double function2(double y) {
    try {
        double x = function1(y); return x;
    } catch(int) { /* handle it!... */ }
}