Chapter 9: Exception Handling

Department of Computer Science and Engineering
Kathmandu University

Instructor: Rajani Chulyadyo, PhD

Contents

Introduction

- An **exception** is an indication of a problem that occurs during a program's execution.
  - Examples: Diving a number by zero, not being able to open a file, array subscripts out of range etc.
- **Exception handling** enables you to create applications that can resolve (or handle) exceptions. It might include
  - Allowing a program to continue executing as if no problem had been encountered, or
  - Notifying the user of the problem before terminating in a controlled manner.

Why Do We Need Exceptions?

Basic exception handling

Exceptions in C++ are implemented using three keywords that work in conjunction with each other: **`throw`**, **`try`**, and **`catch`**.

General structure of a program with try-catch block:

```
try {
  // Statements that we want to monitor for errors (that throw an exception)
}
catch (type1 arg) {
  // Process the exception
}
catch (type2 arg) {
  // Process the exception
}
```

Basic exception handling

Throwing exceptions

A `throw` statement is used to signal that an exception has occurred (to *raise an exception*).

A throw statement consists of the throw keyword, followed by a value of any data type you wish to use to signal that an error has occurred. Typically, this value will be an *error code*, a description of the problem, or a *custom exception class*.

Basic exception handling

Looking for exceptions

The `try` block acts as an observer, looking for any exceptions that are thrown by any of the statements within the try block.

Note that the try block doesn't define HOW we're going to handle the exception. It merely tells the program, "Hey, if any of the statements inside this try block throws an exception, grab it!".

Basic exception handling

Handling exceptions

A try block is followed by a `catch` block, where the actual handling of exceptions is done. It handles exceptions for a single data type.

`Try` blocks and `catch` blocks work together -- a `try` block detects any exceptions that are thrown by statements within the `try` block, and routes them to a `catch` block with a matching type for handling. A `try` block must have at least one `catch` block immediately following it, but may have multiple `catch` blocks listed in sequence.

Once an exception has been caught by the `try` block and routed to a `catch` block for handling, the exception is considered handled, and execution will resume as normal after the `catch` block.

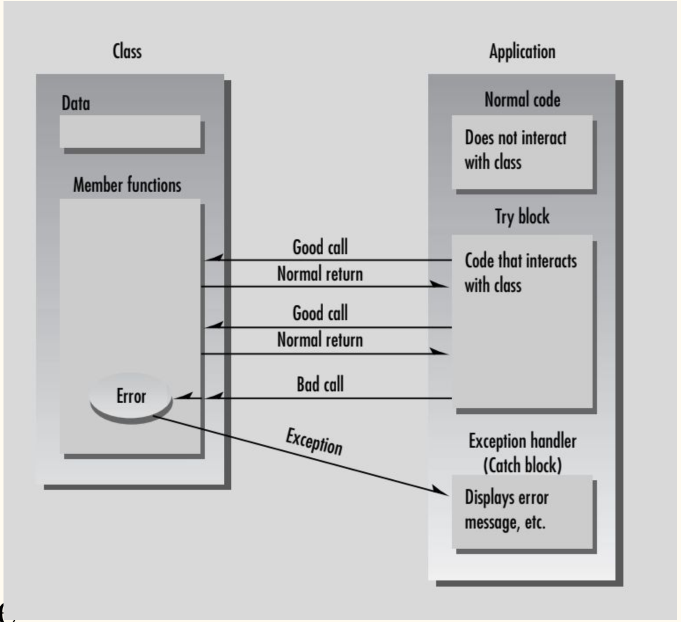
Basic exception handling

Suppose an application creates and interacts with objects of a certain class.

Any code in the application that uses objects of the class is enclosed in a **try** block.

If an exception occurs while calling a member function of the class, the member function informs the application that an error has occurred (i.e., *it* **throws** *an exception*).

The application contains a separate section of code (called *exception handler* or **catch** *block*) to handle the error. It *catches* the exceptions thrown by the member function.



Basic exception handling: Example 1

```
#include <iostream>

int factorial(int n)
{
  if (n < 1 ) {
    throw "n must be greater than 0";
  }
  int result = 1;
  for (int i = 1; i <= n; i++)
  {
    result *= i;
  }

  return result;
}
```

```
int main()
{
  int n;
  std::cout << "n = ? ";
  std::cin >> n;

  try
  {
    int f = factorial(n);
    std::cout << n << "! = " << f << "\n";
  }
  catch (const char* msg)
  {
    std::cerr << "Error: " << msg << "\n";
  }
}
```

Basic exception handling: Example 2

```
//int sumOfNaturalNumbers(int n) throw(int) // Before C++17
int sumOfNaturalNumbers(int n) noexcept(false) // From C++17
{
  if (n < 1) {
    throw 505;
  }

  int result = 0;
  for (int i = 1; i <= n; i++) {
    result += i;
  }

  return result;
}
```

Basic exception handling: Example 2

```
int main()
{
  int n;
  std::cout << "n = ? ";
  std::cin >> n;

  try {
    int f = factorial(n);
    std::cout << n << "! = " << f << "\n";

    int s = sumOfNaturalNumbers(n);
    std::cout << "Sum of natural numbers upto "  << n << " = " << s << "\n";
  }
  catch (const char* msg){
    std::cerr << "Error: " << msg << "\n";
  }
  catch (const int &errorcode) {
    std::cerr << "Error " << errorcode << " occurred!" << std::endl;
  }
}
```

Basic exception handling: Example 3

```cpp
// Example: Deriving your own class from std::exception
#include <iostream>

class MyException : public std::exception {
private:
  std::string m_error;
public:
  MyException(std::string error) : m_error{error} { }

  // return the std::string as a const C-style string
  // const char* what() const { return m_error.c_str(); } // pre-C++11 version
  const char *what() const noexcept // C++11 version
  {
    return m_error.c_str();
  }
};
```

Basic exception handling: Example 3

```
int main() {
  try {
    int i;
    std::cout << "Enter a number: ";
    std::cin >> i;

    if (i > 0) {
      throw MyException("Error message goes here!");
    }
    else {
      throw std::runtime_error("Bad things happened");
    }
  }
  catch (const MyException &exception) {
    std::cerr << "MyException: " << exception.what() << '\n';
  }
  catch (const std::exception &exception) {
    std::cerr << "Standard exception: " << exception.what() << '\n';
  }
return 0;
}
```

Re-throwing an exception

- A handler may decide not to process an exception caught by it. In such cases we can re-throw the exception.
- The most likely reason for doing so is to allow multiple handler access to the exception, e.g. perhaps one exception handler manages one aspect of an exception and a second handler copes with another exception.
- An exception can only be re-thrown from within a catch block (or from any function call from within that block). When we re-throw an exception, it will not be re-caught by the same catch block (statement). It will propagate to an outer (next) catch statement (block).

Re-throwing an exception: An example

```cpp
#include <iostream>
#include <exception>

void foo() noexcept(false) {
  throw 100;
}

void bar() noexcept(false) {
  try {
    foo();
  }
  catch(int i) {
    throw;
  }
}
```

```cpp
int main() {
  try {
    bar();
  }
  catch (int i) {
    std::cout << "Caught " << i << "\n";
  }
}
```

References

1. https://www.learncpp.com/cpp-tutorial/the-need-for-exceptions/
2. https://www.learncpp.com/cpp-tutorial/basic-exception-handling/
1. Lafore, R. Object Oriented Programming in C++. Sams Publishing.