Chapter 14 - Exception Handling

Spring 2022

Objectives

- In this chapter, you will:
  - Learn what an exception is
  - Learn how to handle exceptions within a program
  - Learn how a `try/catch` block is used to handle exceptions
  - Learn how to throw an exception
  - Become familiar with C++ exception classes and how to use them in a program
  - Learn how to create your own exception classes
  - Discover how to throw and rethrow an exception
  - Explore exception handling techniques
  - Explore stack unwinding

Introduction

- An **exception** is an undesirable event detectable during program execution
- Code to handle exceptions depends on the type of application being developed
  - May or may not want the program to terminate when an exception occurs
- Can add exception-handling code at point where an error can occur

Handling Exceptions Within a Program

- **assert** function:
  - Checks if an expression meets certain condition(s)
  - If conditions are not met, it terminates the program
- Example: division by 0
  - If divisor is zero, **assert** terminates the program with an error message

C++ Mechanisms of Exception Handling

- `try/catch` block: used to handle exceptions
- Exception must be thrown in a `try` block and caught by a `catch` block
- C++ provides support to handle exceptions via a hierarchy of classes

Block (1 of 5)

- Statements that may generate an exception are placed in a `try` block
- The `try` block also contains statements that should not be executed if an exception occurs
- `try` block is followed by one or more `catch` blocks

Block (2 of 5)

- General syntax of the `try/catch` block

```
try {
    // statements
} catch (dataType1 identifier) {
    // exception-handling code
} catch (dataType2 identifier) {
    // exception handling code
} catch (...) {
    // exception handling code
}
```

Block (3 of 5)

- `catch` block:
  - Specifies the type of exception it can catch
  - Contains an exception handler
- If the heading of a `catch` block contains `...` (ellipses) in place of parameters:
  - Block can catch exceptions of all types
- If no exception is thrown in a `try` block:
  - All `catch` blocks are ignored
  - Execution resumes after the last `catch` block

Block (4 of 5)

- If an exception is thrown in a `try` block:
  - Remaining statements (in block) are ignored
- Program searches `catch` blocks in order, looking for an appropriate exception handler
  - If the type of thrown exception matches the parameter type in one of the `catch` blocks:
    - Code of that `catch` block executes
    - Remaining `catch` blocks are ignored

Block (5 of 5)

- A `catch` block can have at most one `catch` block parameter
  - `catch` block parameter becomes a placeholder for the value thrown

Throwing an Exception

- For `try/catch` to work, the exception must be thrown in the `try` block
- General syntax:

```
throw expression;
```

  - where expression is a constant value, variable, or object
- Object being thrown can be a specific object or an anonymous object

- In C++, an exception is a value

Order of catch Blocks

- `catch` block can catch:
  - All exceptions of a specific type
  - All types of exceptions
- A `catch` block with an ellipsis (`...`) catches any type of exception
  - If used, it should be the last `catch` block of that sequence
- Be careful about the order in which you list `catch` blocks

Using C++ Exception Classes (1 of 2)

- C++ provides support to handle exceptions via a hierarchy of classes
- The class `std::exception` is the base class of the exception classes provided by C++
- Function `what()`
  - Contained in class `std::exception`
  - Returns a string containing an appropriate message
- All derived classes of the class `std::exception` override the function `what()` to issue their own error messages

Using C++ Exception Classes (2 of 2)

- Two subclasses of exception (defined in `stdexcept`):
  - `logic_error` includes subclasses:
    - `invalid_argument`: for use when illegal arguments are used in a function call
    - `out_of_range`: string subscript out of range error
    - `length_error`: if a length greater than the maximum allowed for a string object is used
  - `runtime_error` includes subclasses:
    - `overflow_error` and `underflow_error`

Creating Your Own Exception Classes (1 of 2)

- Can create your own exception classes to handle specific exceptions
  - C++ uses the same mechanism to process these exceptions
- `throw` statement: used to throw your own exceptions
- Any class can be an exception class
  - How you use the class makes it an exception class

Creating Your Own Exception Classes (2 of 2)

- Exception class with member variables typically includes:
  - Constructors
  - The function `what()`

Rethrowing and Throwing an Exception (1 of 3)

- When an exception occurs in a `try` block, control immediately passes to one of the `catch` blocks, which either:
  - Handles the exception, or partially processes the exception, then rethrows the same exception
  - Rethrows another exception for the calling environment to handle
- This allows you to provide exception-handling code all in one place

Rethrowing and Throwing an Exception (2 of 3)

- Syntax to rethrow an exception caught by a catch block:
  - If the same exception is to be rethrown:

    ```
    throw;
    ```

  - If a different exception is to be thrown

    ```
    throw expression;
    ```

where **expression** is a constant value, variable, or object

Rethrowing and Throwing an Exception (3 of 3)

- Object being thrown can be:
  - A specific object
  - An anonymous object
- A function specifies the exceptions it throws in its heading using the `throw` clause

  This dynamic exception specification is deprecated as of C++17.
- Example:

```
void foo(int x) throw (int, string, divisionByZero)     {
    ...
    // include the appropriate throw statements
    ...
}
```

Exception-Handling Techniques

- When an exception occurs, the programmer usually has three choices:
  - Terminate the program
  - Include code to recover from the exception
  - Log the error and continue

Terminate the Program

- In some cases, it is best to terminate the program when an exception occurs
- Example: if an input file does not exist when the program executes
  - There is no point in continuing with the program

- Program can output an appropriate error message and terminate

Fix the Error and Continue

- In some cases, you will want to handle the exception and let the program continue
- Example: a user inputs a letter instead of a number
  - The input stream will enter the **fail** state
  - Can include the necessary code to keep prompting the user to input a number until the entry is valid

Log the Error and Continue

- Example: if the program is designed to run a nuclear reactor or continuously monitor a satellite
  - It cannot be terminated if an exception occurs
- When an exception occurs:
  - The program should write the exception into a file and continue to run

Stack Unwinding (1 of 2)

- When an exception is thrown in a function, the function can do the following:
  - Do nothing
  - Partially process the exception and throw the same exception or a new exception
  - Throw a new exception
- In each case, the function-call stack is unwound so that the exception can be caught in the next `try/catch` block

Stack Unwinding (2 of 2)

- When the function call stack is unwound:
  - The function in which the exception was not caught and/or rethrown terminates
  - Memory for its local variables is destroyed
- Stack unwinding continues until:
  - A `try/catch` handles the exception, or
  - The program does not handle the exception
    - The function `std::terminate()` is called to terminate the program

Quick Review

- An exception is an undesirable event detectable during program execution
- **assert** checks whether an expression meets a specified condition; terminates if not met
- `try/catch` block handles exceptions
- Statements that may generate an exception are placed in a `try` block
- A `catch` block specifies type of exception it can catch and contains an exception handler

- If no exceptions are thrown in a `try` block, all `catch` blocks for that `try` block are ignored
- Data type of `catch` block parameter specifies type of exception that `catch` block can catch
- `std::exception` is the base class for exception classes
- `what()` function returns a string containing the exception object thrown by builtin exception classes
- You can create your own exception classes
- A function specifies the exceptions it throws in its heading with the throw clause
- If the program does not handle the exception, then the function `std::terminate()` terminates the program

Questions?