C++ Exception Handling

Introduction to Programming

1/07/2016

Error Handling

- Traditional error handling
  - Terminate the program
    - `exit` and `abort`
  - Return a value representing error
    - string functions and `malloc` functions
  - Return a legal value and leave the program in an illegal state
    - files
  - Call function supplied in case of error

C++ Exception Handling

- C++ error handling is intended to support error handling in programs composed of independently developed components
- The author of a library can detect run-time errors but does not know what to do with them.
- The user of a library know how to cope with errors but cannot detect them
- The fundamental idea of exception is that a function finds a problem it cannot handle `throws` an exception, hoping the caller can handle the problem
- A function that wants to handle that kinds of problem can indicate that it is willing to `catch` that exception
- `Exception` means some part of the system couldn't do what it was asked to do

Exception Class

- An exception is an object of some class representing an exception occurrence.
- Code that detects an errors throws an error object.
- A piece of code expresses desire to handle an exception by a catch clause
- The effect of a throw is to unwind the stack until a suitable catch is found
- Often exceptions fall into families
- Inheritance can be useful to structure exceptions and exception handling
- Example: ext1.cpp

Grouping of Exceptions

- Example

```
class Metherr { };
class Overflow: public Matherr { } ;
class Underflow: public Matherr { } ;
class Zerodivide: public Matherr { } ;
function f()
{
  try { ...
  }
  catch (Overflow) {
    // handle overflow
  }
  catch (Matherr) {
    // handle Matherr except Overflow
  }
}
```

- With this grouping, new exceptions would not necessarily cause program halt

Composite Exceptions

- Exceptions might not always grouped by a tree structure
- Example:

```
class Nwtfilw_err:
    public Network_err, public File_system_err
{
   // ......
};
// ...
try {
  // ...
}
catch (Network_err& e) {
  // catches network part of errors
}
catch (File_system_err& e) {
  // catch file system errors
}
```

Catching Exceptions

- Example

```
void f()
{
  try {
    throw E();
  }
  catch (H) {
  }
}
```

- The handler is invoked if
  - H is the same type as E
  - H is an unambiguous public base of E
  - H and E are pointer types and the above hold for the types they refer to
  - H is a reference and the above hold for the type H refers to
- const can be added to the type used to catch exception
- In principle, an exception is copied when it is thrown

Re-Throw and Practical Exception Handling

- Having caught an exception, it is common for a handler to decide that it can't completely handle the error.
- The error handler perform what it can and then throws the exception again
- A re-throw is indicated by a throw without an operand.
- The exception re-thrown is the original exception caught and not just the part of it.

```
catch (...) { } // catches every exception
```

- Order of handler is significant
- Compiler knows class hierarchy, thus, if a base class is handled first, the its derived errors, listed later, will never get executed
- If an exception is thrown but not caught, the function `std::terminate()` will be called

Exception Specifications

- The following function declaration that this function can only throw two types of faults (or faults derived from these two types)

```
void f(int a) throw (x2, x3);
```

- All other errors are handled by the function (or exited from the function)
- The following declaration can throw any fault

```
void f(int a) ;
```

- And the following function throws no errors

```
void f(int a) throw ();
```

- A virtual function may be overridden only by a function that has an exception-specification as restrictive as its own.

Synchronous Errors

- C++ error handling is designed to handle only synchronous exceptions, such as array range check and I/O errors
- Asynchronous events, such as keyboard interrupts and certain arithmetic errors are not necessarily exceptional and are not handled directly by this mechanism.
- Standard C++ does not have the notion of a thread or a process, and thus exceptions relating to concurrency are not discussed.
- But error handling can be effective in a concurrent program as long as programmer or system enforces basic concurrency rules, such as properly locking a shared data structure when using it.

Summary

- Error handling
- C++ exception handling
- Exception class
- Grouping of exceptions
- Catching exceptions
- Exception specifications
- Synchronous Errors