Exception Handling

한양대학교 컴퓨터소프트웨어학부

2017년 2학기

Learning Objectives

- Exception Handling Basics
  - Defining exception classes
  - Multiple throws and catches
  - Exception specifications
- Programming Techniques for Exception Handling
  - When to throw exceptions
  - Exception class hierarchies

Introduction

- Typical approach to development:
  - Write programs assuming things go as planned
  - Get "core" working
  - Then take care of "exceptional" cases
- C++ exception-handling facilities
  - Handle "exceptional" situations
  - Mechanism "signals" unusual happening
  - Another place in code "deals" with exception

Exception Handling

- Exceptions are anomalous or exceptional situations requiring special processing – often changing the normal flow of program execution.[wikipedia]
    - Memory allocation error - out of memory space.
    - Divide by zero.
    - File IO error.
    - …
- Propagating failure through function calls is cumbersome.

Toy Example

- Imagine: people rarely run out of milk:

```
cout << "Enter number of donuts:";
cin >> donuts;
cout << "Enter number of glasses of milk:";
cin >> milk
dpg = donuts/static_cast<double>(milk);
cout        << donuts << "donuts.\n";
            << milk << "glasses of milk.\n";
            << "You have " << dpg
            << "donuts for each glass of milk.\n";
```

- Basic code assumes never run out of milk

Toy Example if-else

- Notice: If no milk→divide by zero error!
- Program should accommodate unlikely situation of running out of milk
  - Can use simple if-else structure:
    ```
    if (milk <= 0)
        cout << "Go buy some milk!\n";
    else
    {…}
    ```
- Notice: no exception-handling here

Exception Handling in C++

```
bool DoSomething(string* error_message) {
  cout << "DoSomething called." << endl;
  // Do something...
  if (something_is_wrong) {
    *error_message = "something is wrong.";
    return false;
  }
  // Do the rest...
  cout << "DoSomething finished." << endl;
  return true;
}
```

```
int main() {
  string error_message;
  if (!DoSomething(&error_message)) {
    cout << "DoSomething failed : '"
         << error_message << "'" << endl;
  }
  cout << "All done." << endl;
  return 0;
}
```

```
Output:
DoSomething called.
DoSomething failed : 'something is wrong.'
All done.
```

Exception Handling in C++

```
bool DoSomething(string* error_message) {
  cout << "DoSomething called." << endl;
  // Do something...
  if (something_is_wrong) {
    *error_message = "something is wrong.";
    return false;
  }
  // Do the rest...
  cout << "DoSomething finished." << endl;
  return true;
}

bool DoSomethingMore(string* error_message) {
  cout << "DoSomethingMore called." << endl;
  if (!DoSomething(error_message)) {
    return false;
  }
  // Do something more...
  if (something_is_wrong) {
    *error_message = "something is wrong.";
    return false;
  }
  // Do the rest...
  cout << "DoSomethingMore finished." << endl;
  return true;
}
```

```
int main() {
  string error_message;
  if (!DoSomethingMore(&error_message)) {
    cout << "DoSomethingMore failed : '"
         << error_message << "'" << endl;
  }
  cout << "All done." << endl;
  return 0;
}
```

```
Output:
DoSomethingMore called.
DoSomething called.
DoSomethingMore failed : 'something is wrong.'
All done.
```

Exception Handling in C++

`try - throw - catch`

- `try`: be prepared to catch certain exceptions specified in the following catch blocks thrown within the block.
- `catch`: catches the exception of the given type, then handles it - either re-throws it or stops propagating it.
- `throw`: invokes (throws) an exception event. It will be caught and handled by the try-catch block.
  (it also is used to specify which exceptions can be thrown in a function.)
- Any object can be thrown as an exception. The thrown object is copied.

Toy Example with Exception Handling: Display 18.2 Same Thing Using Exception Handling

```
try
{
    cout << "Enter number of donuts:\n";
    cin >> donuts;
    cout << "Enter number of glasses of milk:\n";
    cin >> milk;

    if (milk <= 0)
            throw donuts;

    dpg = donuts/static_cast<double>(milk);
    cout << donuts << " donuts.\n"
         << milk << " glasses of milk.\n"
         << "You have " << dpg
         << " donuts for each glass of milk.\n";
}
catch(int e)
{
    cout << e << " donuts, and No Milk!\n"
         << "Go buy some milk.\n";
}
```

Toy Example Discussion

- Code between keywords *try* and *catch*
  - Same code from ordinary version, except if statement simpler:
    ```
    if (milk <= 0)
        throw donuts;
    ```
  - Much cleaner code
  - If "no milk" → do something exceptional
- The "something exceptional" is provided after keyword *catch*

Toy Example try-catch

- Try block
  - Handles "normal" situation
- Catch block
  - Handles "exceptional" situations
- Provides separation of normal from exceptional
  - Not big deal for this simple example, but important concept

try block

- Basic method of exception-handling is try-throw-catch
- Try block:

```
try
{
    Some_Code;
}
```

  - Contains code for basic algorithm when all goes smoothly

throw

- Inside try-block, when something unusual happens:

```
try
{
    Code_To_Try
    if (exceptional_happened)
        throw donuts;
    More_Code
}
```

  - Keyword *throw* followed by exception type
  - Called "throwing an exception"

catch-block

- When something thrown → goes somewhere
  - In C++, flow of control goes from try-block to catch-block
    - try-block is "exited" and control passes to catch-block
  - Executing catch block called "catching the exception"
- Exceptions must be "handled" in some catch block

catch-block More

- Recall:

```
catch(int e)
{
      cout << e << " donuts, and no milk!\n";
                 << " Go buy some milk.\n";
}
```

- Looks like function definition with int parameter!
  - Not a function, but works similarly
  - Throw like "function call"

catch-block Parameter

- Recall: catch(int e)
- "e" called catch-block parameter
  - Each catch block can have at most ONE catch-block parameter
- Does two things:
  1. type name specifies what kind of thrown value the catch-block can catch
  2. Provides name for thrown value caught; can "do things" with value

Exception Handling in C++

```
void ThrowsException() {
  throw string("Exception!");
}

void DoSomething() {
  cout << "DoSomething called." << endl;
  // Do something...
  if (something_is_wrong) ThrowsException();
  cout << "DoSomething finished." << endl;
}
```

```
int main() {
  try {
    DoSomething();
  } catch (string s) {
    cout << "Caught an exception '"
         << s << "'" << endl;
  }
  cout << "All done." << endl;
  return 0;
}
```

```
Output:
DoSomething called.
Caught an exception 'Exception!'
All done.
```

Exception Handling in C++

- Exceptions can be propagated through several levels of function calls if there is no try-catch block for the exception type.

```
void ThrowsException() {
  throw string("Exception!");
}

void DoSomething() {
  cout << "DoSomething called." << endl;
  // Do something...
  if (something_is_wrong) ThrowsException();
  cout << "DoSomething finished." << endl;
}

void DoSomethingMore() {
  cout << "DoSomethingMore called." << endl;
  DoSomething();
  // Do something more...
  if (something_is_wrong) {
    throw string("error.");
  }
  cout << "DoSomethingMore finished." << endl;
}
```

```
int main() {
  try {
    DoSomethingMore();
  } catch (string s) {
    cout << "Caught an exception '"
         << s << "'" << endl;
  }
  cout << "All done." << endl;
  return 0;
}
```

```
Output:
DoSomethingMore called.
DoSomething called.
Caught an exception 'Exception!'
All done.
```

Defining Exception Classes

- throw statement can throw value of any type
- Exception class
  - Contains objects with information to be thrown
  - Can have different types identifying each possible exceptional situation
  - Still just a class
    - An "exception class" due to how it’s used

Exception Class for Toy Example

- Consider:

```
class NoMilk
{
public:
    NoMilk() { }
    NoMilk(int howMany) : count(howMany) { }
    int getcount() const { return count; }
private:
    int count;
};
```

- throw NoMilk(donuts);
  - Invokes constructor of NoMilk class

Multiple Throws and Catches

- try-block typically throws any number of exception values, of differing types
- Of course only one exception thrown
  - Since throw statement ends try-block
- But different types can be thrown
  - Each catch block only catches "one type"
  - Typical to place many catch-blocks after each try-block
    - To catch "all-possible" exceptions to be thrown

Catching

- Order of catch blocks important
- Catch-blocks tried "in order" after try-block
  - First match handles it!
- Consider:
  catch (...) { }
  - Called "catch-all", "default" exception handler
  - Catches any exception
  - Ensure catch-all placed AFTER more specific exceptions!
    - Or others will never be caught!

Trivial Exception Classes

- Consider:
  ```
  class DivideByZero
  { }
  ```
- No member variables
- No member functions (except default constructor)
- Nothing but it's name, which is enough
  - Might be "nothing to do" with exception value
  - Used simply to "get to" catch block
  - Can omit catch block parameter

Throwing Exception in Function

- Function might throw exception
- Callers might have different "reactions"
  - Some might desire to "end program"
  - Some might continue, or do something else
- Makes sense to "catch" exception in calling function’s try-catch-block
  - Place call inside try-block
  - Handle in catch-block after try-block

Throwing Exception in Function Example

- Consider:

```
try
{
      quotient = safeDivide(num, den);
}
catch (DivideByZero)
{ … }
```

- safeDivide() function throws DividebyZero exception
  - Handled back in caller’s catch-block

Exception Specification

- Functions that don’t catch exceptions
  - Should "warn" users that it could throw
  - But it won’t catch!
- Should list such exceptions:

```
double safeDivide(int top, int bottom)
                        throw (DividebyZero);
```

  - Called "exception specification" or "throw list"
  - Should be in declaration and definition
  - All types listed handled "normally"
  - If no throw list → all types considered there

Throw List

- If exception thrown in function NOT in throw list:
  - No errors (compile or run-time)
  - Function unexpected() automatically called
    - Default behavior is to terminate
    - Can modify behavior
- Same result if no catch-block found

Exception Handling in C++

- Exceptions can be propagated through several levels of function calls if there is no try-catch block for the exception type.

```
void ThrowsException() {
  throw string("Exception!");
}

void DoSomething() {
  cout << "DoSomething called." << endl;
  // Do something...
  if (something_is_wrong) ThrowsException();
  cout << "DoSomething finished." << endl;
}

void DoSomethingMore() {
  cout << "DoSomethingMore called." << endl;
  DoSomething();
  // Do something more...
  if (something_is_wrong) {
    throw string("error.");
  }
  cout << "DoSomethingMore finished." << endl;
}
```

```
int main() {
  try {
    DoSomethingMore();
  } catch (string s) {
    cout << "Caught an exception '"
         << s << "'" << endl;
  }
  cout << "All done." << endl;
  return 0;
}
```

```
Output:
DoSomethingMore called.
DoSomething called.
Caught an exception 'Exception!'
All done.
```

Exception Handling in C++

- Uncaught exceptions cause the program to halt (thus dangerous).

```
void ThrowsException() {
  throw string("Exception!");
}

void CallsOne() {
  ThrowsException();
}

void CallsTwo() {
  try {
    CallsOne();
  } catch (MyException e) {
    cout << "Caught a MyException '"
         << e.msg << "'" << endl;
  }
}
```

```
int main() {
  try {
    CallsTwo();
  } catch (MyException e) {
    cout << "Caught an exception '"
         << e.msg << "'" << endl;
  }
  return 0;
}
```

```
Output (depending on systems):
terminate called throwing an exceptionAbort trap
: 6
```

Exception Handling in C++

- `throw (...)` after a (member) function declaration specifies which exceptions it may generate - but not strictly enforced.

```
void ThrowsException() throw (string) {
  throw string("Exception!");
}

void CallsTwo() throw (string, MyException) {
  ThrowsException();
  throw MyException("test");
}

void CallsOther() throw () {
  // ...
}
```

```
int main() {
  try {
    CallsTwo();
  } catch (MyException e) {
    cout << "Caught an exception '"
         << e.msg << "'" << endl;
  }
  return 0;
}
```

```
Output (depending on systems):
terminate called throwing an exceptionAbort trap
: 6
```

Exception Handling in C++

- Class hierarchy is sometimes useful in defining and catching exceptions - use references.

```cpp
struct MyException : public std::exception {
  int my_counter;
};

struct MySpecializedException
    : public MyException {
  int special_counter;
};
```

```cpp
int main() {
  try {
    // This may throw
    // MySpecializedException.
    CallSpecializedFunction();
    // This may throw MyException.
    CallGeneralFunction();
  } catch (MySpecializedException& e) {
    // ...
  } catch (MyException& e) {
    // ...
  } catch (std::exception& e) {
    // ...
  }
  return 0;
}
```

Exception Handling in C++

```
#include <exception>     // std::exception

class exception {
public:
  exception () noexcept;
  exception (const exception&) noexcept;
  exception& operator= (const exception&) noexcept;
  virtual ~exception();
  virtual const char* what() const noexcept;
}

struct MyException : std::exception {
  string msg;

  MyException(const string& m) : msg(m) {}
};

void DoSomething() {
  cout << "DoSomething called." << endl;
  throw MyException("DoSomething");
}

void DoSomethingElse() {
  cout << "DoSomethingElse called." << endl;
  throw new MyException("DoSomethingElse");
}
```

```
int main() {
  try {
    DoSomething();
  } catch (std::exception e) {
    cout << "Caught an exception" << endl;
  }
  try {
    DoSomethingElse();
  } catch (MyException* e) {
    cout << "Caught a MyException "
         << e->msg << endl;
    delete e;
  }
  return 0;
}
```

```
Output:
DoSomething called.
Caught a MyException DoSomething
DoSomethingElse called.
Caught a MyException DoSomethingElse
```

Throw List Summary

- void someFunction()
  throw(DividebyZero, OtherException);
  //Exception types DividebyZero or OtherException
  //treated normally. All others invoke unexpected()
- void someFunction() throw ();
  //Empty exception list, all exceptions invoke unexpected()
- void someFunction();
  //All exceptions of all types treated normally

Derived Classes

- Remember: derived class objects also objects of base class
- Consider:
  D is derived class of B
- If B is in exception specification →
  - Class D thrown objects will also be treated normally, since it's also object of class B
- Note: does not do automatic type cast:
  - double will not account for throwing an int

unexpected()

- Default action: terminates program
  - No special includes or using directives
- Normally no need to redefine
- But you can:
  - Use set_unexpected
  - Consult compiler manual or advanced text for details

When to Throw Exceptions

- Typical to separate throws and catches
  - In separate functions
- Throwing function:
  - Include throw statements in definition
  - List exceptions in throw list
    - In both declaration and definition
- Catching function:
  - Different function, perhaps even in different file

Preferred throw-catch Triad: throw

-
  ```
  void functionA() throw (MyException)
  {
          …
          throw MyException(arg);
          …
  }
  ```

- Function throws exception as needed

Preferred throw-catch Triad: catch

- Then some other function:

```
void functionB()
{
	…
	try
	{
		…
		functionA();
		…
	}
	catch (MyException e)
	{ // Handle exception
	}
	…
}
```

Uncaught Exceptions

- Should catch every exception thrown
- If not → program terminates
  - terminate() is called
- Recall for functions
  - If exception not in throw list: unexpected() is called
    - It in turn calls terminate()
- So same result

Overuse of Exceptions

- Exceptions alter flow of control
  - Similar to old "goto" construct
  - "Unrestricted" flow of control
- Should be used sparingly
- Good rule:
  - If desire a "throw": consider how to write program without throw
  - If alternative reasonable → do it

Exception Class Hierarchies

- Useful to have; consider:
  DivideByZero class derives from:
      ArithmeticError exception class
  - All catch-blocks for ArithmeticError also catch DivideByZero
  - If ArithmeticError in throw list, then DividebyZero also considered there

Testing Available Memory

- new operator throws bad_alloc exception if insufficient memory:

```
try
{
      NodePtr pointer = new Node;
}
catch (bad_alloc)
{
      cout << "Ran out of memory!";
      // Can do other things here as well…
}
```

- In library <new>, std namespace

Rethrowing an Exception

- Legal to throw exception IN catch-block!
  - Typically only in rare cases
- Throws to catch-block "farther up chain"
- Can re-throw same or new exception
  - rethrow;
    - Throws same exception again
  - throw newExceptionUp;
    - Throws new exception to next catch-block

Q&A