Exception Handling

임종우 (Jongwoo Lim)

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.

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++

```cpp
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;
}
```

```cpp
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.

Exception Handling in C++

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

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

```cpp
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.
```

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.

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

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

```
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
```