EXCEPTIONS

How to deal with errors

Florian Warg, Max Staff

May 18, 2017

EXCEPTION HANDLING

- exception handling is a way of dealing with runtime errors
- if an exception occurs, control is transferred to handlers
- exceptions may be handled gracefully so program execution can continue
- exceptions can occur:
    - when trying to allocate memory
    - when providing an illegal argument to a function
    - when we break math (divide by zero)
    - when we dynamically cast to unrelated types
    - when a variable overflows or underflows
    - …

TRY AND CATCH

- critical parts of program must be surrounded by `try` block
- this is followed by a `catch` block that handles the exception

```
void evil() { throw exception(); }
/* ... */
try {
    evil();
} catch (const exception& e) {
    cerr << e.what() << "\n";
}
```

PRE-DEFINED EXCEPTIONS

- `logic_error (invalid_argument, ...)`
- `runtime_error (overflow_error, ...)`
- `bad_typeid`
- `bad_cast`
- `bad_weak_ptr`
- `bad_function_call`
- `bad_alloc`
- `bad_exception`
- `ios_base::failure`
- ...

http://en.cppreference.com/w/cpp/error/exception

CREATE YOUR OWN EXCEPTIONS

- you can derive from `std::exception`
- your class has to implement the `what()` function

```
class MyException : public std::exception {
    virtual const char* what() const override {
        return ”My exception was thrown!\n”;
    }
};
```

HANDLING DIFFERENT EXCEPTIONS

- you can use several `catch` blocks to handle different exceptions

```
void evil(int x) {
    if (x < 0)
        throw std::exception();
    else
        throw MyException();
}
/* ... */
try { evil(1); }
catch (std::exception e) { /* ... */ }
catch (MyException e) { /* ... */ }
```

THROWING ANYTHING

- you are not limited to exception objects
- you can actually throw **anything**
- (but why would you want to do that)

```
void evil() { throw 42; }
/* ... */
try { evil(); }
catch (int i) {
    cerr << i << '\n';
}
```

CATCHING ANYTHING

- because of polymorphism you can catch all objects of a class hierarchy by reference
- you can also catch anything that is thrown

```
void evil() { throw MyException(); }
/* ... */
try { evil();
} catch (std::exception& e) {
    /* also catches MyException */
} catch (...) {
    /* catches anything */
}
```

EXCEPTION SAFETY GUARANTEES

- there are 4 levels of exception guarantees in C++
- the higher safety guarantees make it easy to recover from exceptions
- levels are in decreasing order (level 1 is the highest safety guarantee)

LEVEL 1: NO-THROW GUARANTEE

- function does not throw exceptions even in exceptional situations
- occuring exceptions are handled internally
- function will success in every situation
- keyword `noexcept` can be used to mark functions

```
int f() noexcept { return 42; }
```

level 2: strong exception safety

- also known as commit or rollback semantics
- function can fail but is guaranteed to have no side effects
- if this function fails, all data will retain their original values

level 3: basic exception safety

- also known as no-leak guarantee
- failed function can have side effects but invariants are preserved and resources are not leaked

level 4: no exception safety

- no guarantees are made ™