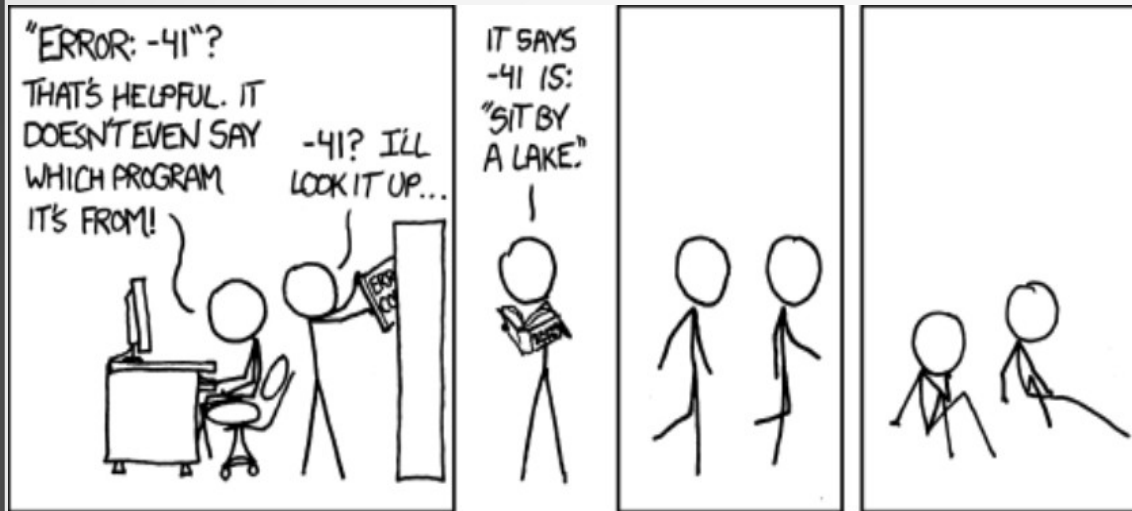


C++ Exception Handling (Part II)



<http://xkcd.com/1024/>



Krishna Kumar

Exceptions: They are not errors!

“some part of the system couldn’t do what it was asked to do”

- Catching Exception:

- ```
void f() {
 try {
 throw E {};
 }
 catch(H) {
 // when do we get here?
 }
}
```

# Catching Multiple Exceptions

```
try {
 // Code to Try
}

catch(arg_1) {
 // One Exception
}

catch(arg_2) {
 // Another Exception
}
```

# Object lifetimes

// Example code

```
{
 Parrot& perch = Parrot();
}
```

- When does an object's lifetime begin?
  - When its constructor completes successfully and returns normally. That is, when control reaches the end of the constructor body or completes an earlier *return* statement
- When does an object's lifetime end?
  - When its destructor begins. That is, when control reaches the beginning of the destructor body

# Constructor exception

```
{ Parrot& perch = Parrot(); }
```

- The state of the object before its lifetime begins is exactly the same as after its lifetime ends: There is no object.
- What does emitting an exception from a constructor mean?
  - It means that construction has failed, the object never existed, its lifetime never began. Indeed, the only way to report the failure of construction—namely, the inability to correctly build a functioning object of the given type—is to throw an exception. Incidentally, this is why a destructor will never be called if the constructor didn't succeed—there's nothing to destroy

# What exactly happens when a constructor emits an exception?

```
class C : private A
{
 B b_;
};
```

In the **C** constructor, how can you catch an exception thrown from the constructor of a base subobject (such as **A** ) or a member object (such as **b\_** )?

# Function try blocks

```
C::C()
```

```
try
```

```
 : A (/*...*/), b_(/*...*/) {}
```

```
catch(...)
```

```
{
```

```
// We get here if either A::A() or B::B() throws.
```

```
}
```

- If A::A() succeeds and then B::B() throws, the language guarantees that A::~~A() will be called to destroy the already-created A base subobject before control reaches this catch block
- **Is this needed at all? In C++, if construction of any base or member subobject fails, the whole object's construction must fail.**

# Morals about function try blocks

- **Moral #1:** Constructor function try block handlers are only good for translating an exception thrown from a base or member subobject constructor. They are not useful for any other purpose.
- **Moral #2:** Destructor function try blocks have little or no practical use. Destructor should NEVER emit an exception.
- **Moral #3:** All other function try blocks have no practical use.
- **Moral #4:** Always perform unmanaged resource acquisition in the constructor body, never in initializer lists, i.e., Resource Allocation Is Initialisation (RAII).



# Exception-safe function calls

// Example 1(a)

```
f(expr1, expr2);
```

// Example 1(b)

```
f(g(expr1), h(expr2));
```

- What is the order of evaluation of the functions f, g, and h and the expressions expr1 and expr2? Assume that expr1 and expr2 do not contain more function calls.