14.7 — Function try blocks

BY ALEX ON FEBRUARY 6TH, 2017 | LAST MODIFIED BY ALEX ON JANUARY 23RD, 2020

Try and catch blocks work well enough in most cases, but there is one particular case in which they are not sufficient. Consider the following example:

```
class A
{
private:
    int m_x;
public:
    A(int x) : m_x(x)
    {
        if (x <= 0)
            throw 1;
    }
};

class B : public A
{
public:
    B(int x) : A(x)
    {
        // What happens if creation of A fails and we want to handle it here?
    }
};

int main()
{
    try
    {
        B b(0);
    }
    catch (int)
    {
        std::cout << "Oops\n";
    }
}
```

In the above example, derived class B calls base class constructor A, which can throw an exception. Because the creation of object b has been placed inside a try block (in function main()), if A throws an exception, main's try block will catch it. Consequently, this program prints:

Oops

But what if we want to catch the exception inside of B? The call to base constructor A happens via the member initialization list, before the B constructor's body is called. There's no way to wrap a standard try block around it.

In this situation, we have to use a slightly modified try block called a **function try block**.

Function try blocks

Function try blocks are designed to allow you to establish an exception handler around the body of an entire function, rather than around a block of code.

The syntax for function try blocks is a little hard to describe, so we'll show by example:

```
#include <iostream>

class A
{
private:
    int m_x;
public:
    A(int x) : m_x(x)
    {
        if (x <= 0)
            throw 1;
    }
};

class B : public A
{
public:
    B(int x) try : A(x) // note addition of try keyword here
    {
    }
    catch (...) // note this is at same level of indentation as the function itself
    {
                // Exceptions from member initializer list or constructor body are caught her
e

                std::cerr << "Exception caught\n";

                // If an exception isn't explicitly thrown here, the current exception will b
e implicitly rethrown
    }
};

int main()
{
    try
    {
        B b(0);
    }
    catch (int)
    {
        std::cout << "Oops\n";
    }
}
```

When this program is run, it produces the output:

```
Exception caught
Oops
```

Let's examine this program in more detail.

First, note the addition of the "try" keyword before the member initializer list. This indicates that everything after that point (until the end of the function) should be considered inside of the try block.

Second, note that the associated catch block is at the same level of indentation as the entire function. Any exception thrown between the try keyword and the end of the function body will be eligible to be caught here.

Finally, unlike normal catch blocks, which allow you to either resolve an exception, throw a new exception, or rethrow an existing exception, with function-level try blocks, you must throw or rethrow an exception. If you do not

explicitly throw a new exception, or rethrow the current exception (using the throw keyword by itself), the exception will be implicitly rethrown up the stack.

In the program above, because we did not explicitly throw an exception from the function-level catch block, the exception was implicitly rethrown, and was caught by the catch block in main(). This is the reason why the above program prints "Oops"!

Although function level try blocks can be used with non-member functions as well, they typically aren't because there's rarely a case where this would be needed. They are almost exclusively used with constructors!

Function try blocks can catch both base and the current class exceptions

In the above example, if either A or B's constructor throw an exception, it will be caught by the try block around B's constructor.

We can see that in the following example, where we're throwing an exception from class B instead of class A:

```
#include <iostream>

class A
{
private:
    int m_x;
public:
    A(int x) : m_x(x)
    {
    }
};

class B : public A
{
public:
    B(int x) try : A(x) // note addition of try keyword here
    {
        if (x <= 0) // moved this from A to B
            throw 1; // and this too
    }
    catch (...)
    {
                std::cerr << "Exception caught\n";

                // If an exception isn't explicitly thrown here, the current exception will b
e implicitly rethrown
    }
};

int main()
{
    try
    {
        B b(0);
    }
    catch (int)
    {
        std::cout << "Oops\n";
    }
}
```

We get the same output:

```
Exception caught
Oops
```

Don't use function try to clean up resources

When construction of an object fails, the destructor of the class is not called. Consequently, you may be tempted to use a function try block as a way to clean up a class that had partially allocated resources before failing. However, referring to members of the failed object is considered undefined behavior since the object is "dead" before the catch block executes. This means that you can't use function try to clean up after a class. If you want to clean up after a class, follow the standard rules for cleaning up classes that throw exceptions (see the "When constructor fail" subsection of lesson **14.5 -- Exceptions, classes, and inheritance**).

Function try is useful primarily for either logging failures before passing the exception up the stack, or for changing the type of exception thrown.

 **14.8 -- Exception dangers and downsides**

 **Index**

 **14.6 -- Rethrowing exceptions**

C++ TUTORIAL | PRINT THIS POST

15 comments to 14.7 — Function try blocks

Vir1oN
July 19, 2019 at 7:58 am · Reply

Hi

I was playing with your first code example to learn more about what will happen with the derived class (B), when the base class` constructor throws an exception. I was expecting to see the same behavior as described in lesson 14.5 in the section "When constructors fail", but came into the different result.

```
1  #include <iostream>
2
3  class A
4  {
5  private:
6      int m_x;
7  public:
8      A(int x) : m_x(x)
9      {
10         if (x <= 0)
11             throw 1;
12     }
13
14     ~A() { std::cerr << "~A This is never called"; }
15 };
16
17 class Member
```