Exceptions and assertions in C++

Used to detect and handle unusual or exceptional conditions

Assertion syntax

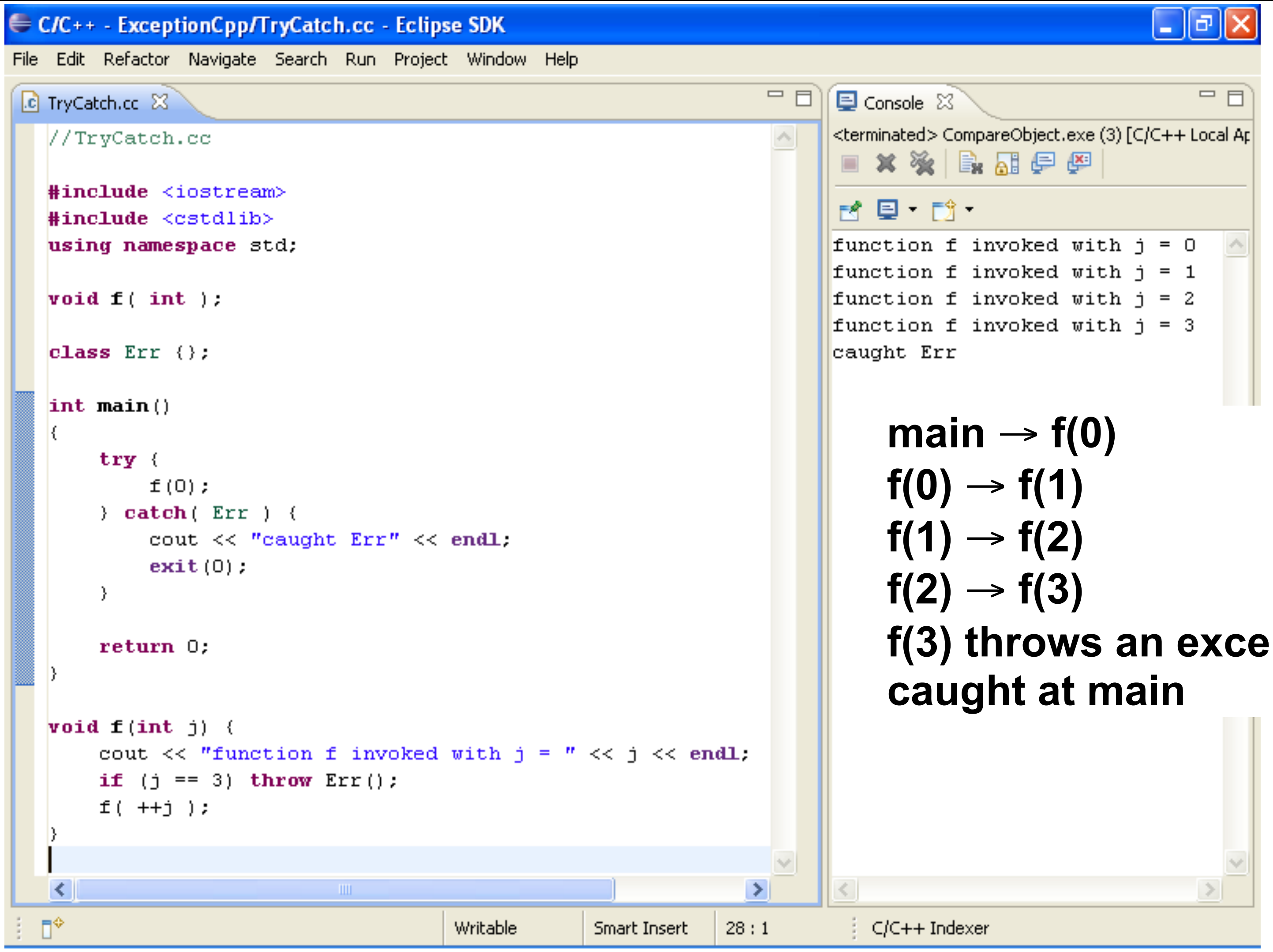
```
#include <assert.h>

int main()
{
  int* array = NULL;
  assert(array != NULL);  // this should fail
  return 0;
}
```

use

`#define NDEBUG`
(or use the flag -DNDEBUG with g++)

to activate assertions. You do not have to physically remove them from your code before shipping it, which can be a good thing.

```
Assertion failed: array != NULL, file test.cpp, line 7

abnormal program termination
```


C/C++ - ExceptionCpp/TryCatch.cc - Eclipse SDK
File Edit Refactor Navigate Search Run Project Window Help
TryCatch.cc
Console
<terminated> CompareObject.exe (3) [C/C++ Local Ap
Writable
Smart Insert
28 : 1
C/C++ Indexer

```
//TryCatch.cc

#include <iostream>
#include <cstdlib>
using namespace std;

void f( int );

class Err {};

int main()
{
    try {
        f(0);
    } catch( Err ) {
        cout << "caught Err" << endl;
        exit(0);
    }

    return 0;
}

void f(int j) {
    cout << "function f invoked with j = " << j << endl;
    if (j == 3) throw Err();
    f( ++j );
}
```

```
function f invoked with j = 0
function f invoked with j = 1
function f invoked with j = 2
function f invoked with j = 3
caught Err
```

main → f(0)
f(0) → f(1)
f(1) → f(2)
f(2) → f(3)
f(3) throws an exception
caught at main

```
#include <iostream>
using namespace std;

void f( ) {
   throw 29
}
void g(int j) {
   cout << "j = " << j <<endl;
   if (j = 3) {
      throw 17;
   }
   g(++j);
}

int main( ) {
   try {
      f( );
   } catch (int i) {
      cout << "caught it " << i << endl;
   }
   try {
      g(0);
   } catch (int i) {
      cout << "caught it " << i << endl;
   }
   return 0;
}
```

```
caught it 29
j = 0
j = 1
j = 2
j = 3
caught it 17
```

C++, unlike Java, allows exceptions with primitive types!

```
#include <iostream>
using namespace std;

void f( ) {
  throw 29
}
void g(int j) {
  cout << “j = “ << j <<endl;
  if (j = 3) {
    throw 17;
  }
  g(++j);
}

int main( ) {
  try {
    f( );
  } catch (int i) {
    cout << “caught it “ << i << endl;
  }
  try {
    g(0);
  } catch (int i) {
    cout << “caught it “ << i << endl;
  }
  return 0;
}
```

```
caught it 29
j = 0
j = 1
j = 2
j = 3
caught it 17
```

Exceptions with Objects and Declarations

Define classes for two kinds of exceptions

```
//ExceptionUsage4.cc
#include <iostream>
#include <string>
using namespace std;
class MyException {
   string me_message;
public:
   MyException(string msg) : me_message(msg); { }
   void print( ) {
      cout << me_message << endl;
   }
};
```

The class *MyException* creates a class of exceptions with this name

```
Class Err {
   int e_value;
public:
   Err(int i) : e_value(i) { }
   void print( )
      cout << e_value << endl;
   }
};
```

The class *Err* creates another class of exceptions of type *Err*

```
void f(int j) throw(MyException, Err) {
   if (j==1) {
      throw MyException("hello");
   }
   if (j == 2) {
      throw Err(65);
   }
}
```

Using the exceptions

```
//ExceptionUsage4.cc
#include <iostream>
#include <string>
using namespace std;
class MyException {
   string me_message;
public:
   MyException(string msg) {me_message(msg);}
   void print( ) {
      cout << me_message << endl;
   }
};
```

Code that actually throws the exceptions.

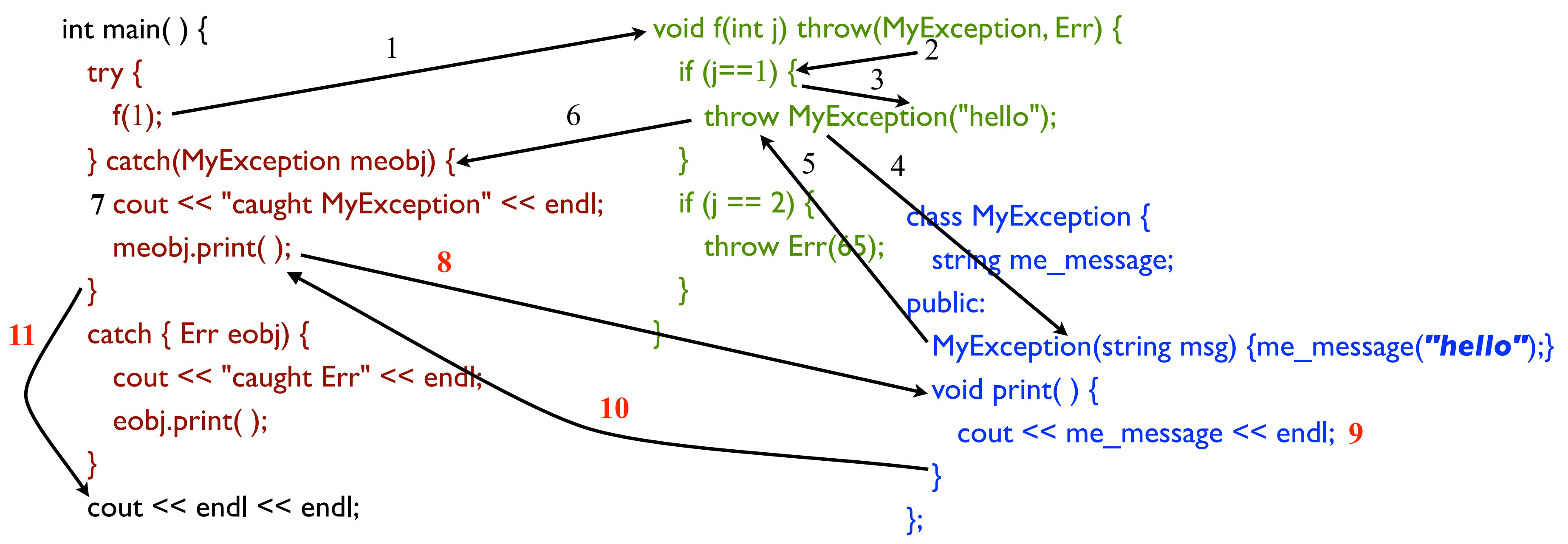
```
Class Err {
   int e_value;
public:
   Err(int i) : e_value(i) { }
   void print( )
      cout << e_value << endl;
   }
};

void f(int j) throw(MyException, Err) {
   if (j==1) {
      throw MyException("hello");
   }
   if (j == 2) {
      throw Err(65);
   }
}
```

The driver code

```
int main( ) {
   try {
      f(1);
   } catch(MyException meobj) {
      cout << "caught MyException" << endl;
      meobj.print( );
   }
   catch { Err eobj) {
      cout << "caught Err" << endl;
      eobj.print( );
   }
   cout << endl << endl;
   try {
      f( 2 );
   } catch(MyException meobj) {
      cout << "caught MyException" << endl;
      meobj.print( );
   }
   catch { Err eobj) {
      cout << "caught Err" << endl;
      eobj.print( );
   }
   return 0;
}
```

Two different catch blocks since we are trying two different actions.
Often several things happen in a try block.

Exceptions in action



```
int main( ) {
  try {
    f(1);
  } catch(MyException meobj) {
  7 cout << "caught MyException" << endl;
    meobj.print( );
  }
  catch { Err eobj) {
    cout << "caught Err" << endl;
    eobj.print( );
  }
  cout << endl << endl;
```

```
void f(int j) throw(MyException, Err) {
  if (j==1) {
    throw MyException("hello");
  }
  if (j == 2) {
    throw Err(65);
  }
}
```

```
class MyException {
  string me_message;
public:
  MyException(string msg) {me_message("hello");}
  void print( ) {
    cout << me_message << endl;  9
  }
};
```

7. *caught MyException*
9. *hello*

Exceptions in action two

```
try {
   f( 2 ); 1
} catch(MyException meobj) {
   cout << "caught MyException" << endl;
   meobj.print( );
}
catch { Err eobj) { 8
   cout << "caught Err" << endl; 9
   eobj.print( ); 10
}
return 0;  11
}
```

```
void f(int j) throw(MyException, Err) {2
   if (j==1) { 3
      throw MyException("hello");
   }
   if (j == 2) {  4
  7 throw Err(65);  5
   }
}
```

```
Class Err {
   int e_value;
public:
   Err(int i) : e_value(i) { } 6
11 void print( )
      cout << e_value << endl;
   }
};
```

9. caught Err
10. 65

What about throw(...) clauses in the function signature?

Part of the type when overriding.
Not allowed to be part of the type in a typedef
Does not guarentee that only MyException and Err exceptions are thrown, so not good documentation
  If another exception thrown, an "unexpected( )" exception thrown
  A single global catch for "unexpected"

From http://www.gotw.ca/publications/mill22.htm

While mentioning this material as part of a broader talk at the ACCU conference this past spring, I asked how many of the about 100 people in the room each time had used exception specifications. About half put up their hands. . . . I asked [how many took them out later]; about the same number of hands went up. This is telling.

True, many well-intentioned people wanted exception specifications in the language, and so that's why we have them. This reminds me of a cute poem that I first encountered about 15 years ago as it circulated in midwinter holiday emails. Set to the cadence of "'Twas the Night Before Christmas," these days it's variously titled "'Twas the Night Before Implementation" or "'Twas the Night Before Crisis." It tells of a master programmer who slaves away late at night in the holiday season to meet user deadlines, and performs multiple miracles to pull out a functioning system that perfectly implements the requirements… only to experience a final metaphorical kick in the teeth as the last four lines of the ditty report:

The system was finished, the tests were concluded,
The users' last changes were even included.
And the users exclaimed, with a snarl and a taunt,
"It's just what we asked for, but not what we want!" [4]

The thought resonates as we finish considering our current experience with exception specifications. The feature seemed like a good idea at the time, and it is just what some asked for.

Exceptions in C++ and Java

C++	Java
can throw exceptions of objects or primitive types (such as int)	must be objects of classes derived from Exception
does not have to declare what exceptions may be thrown, but preferred	must declare what exceptions may be thrown
does not have to, but preferred	must be enclosed within a try-catch block, checked by compiler
Can use *} catch (...) {* to catch all exceptions -- however, no object specified, no handle on the exception.	the catch block must identify the object, for example, catch (MyException **meobj**) {
may throw different types of exceptions	same, but the exception thrown must be a class that directly or indirectly extends Exception.
does not have the equivalent	allows finally