Exceptions and assertions in C++

Used to detect and handle unusual or exceptional conditions

exceptions.key
Assertion syntax

#include <assert.h>

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

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

abnormal program termination
throw-catch

• When an unexpected situation occurs and the current function cannot handle it, it *throws* an exception.
• The exception is sent to the immediate caller. If it does not *catch* this exception, the exception is thrown to the next caller in the call stack.
• If an exception is not caught anywhere in the call stack, the program terminates (no where else to throw it)
• Should a calling function be aware of exceptions thrown by the called function?
  – In C++, a function may be called even if the caller does not catch the exception that may be thrown by the callee
  – Java requires syntax to make it explicit that the programmer wants the exception re-thrown.
Catching an Exception

- An exception is caught by the closest caller in the stack.
- If both A and B catch the same exception and D throws the exception, B will catch it.
- The handling code can throw the exception again to its call stack; the handling code can also throw a different exception.
- An exception may pass parameters through the call stack.
Exception in C++

```cpp
#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 );
}
```

main → f(0)
f(0) → f(1)
f(1) → f(2)
f(2) → f(3)
f(3) throws an exception caught at main
C++ Exceptions with Primitive Types (e.g. int)
Java does not allow this
```cpp
#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;
}
```
```cpp
#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 = 8
j = 1
j = 2
j = 3
caught it 17
```
Exceptions with Objects and Declarations
//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;
   }
};

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) {
      cout << me_message << endl;
      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;
    }
};

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) {
    cout << me_message << endl;
    if (j==1) {
        throw MyException("hello");
    }
    if (j == 2) {
        throw Err(65);
    }
}
The driver code

```c++
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;
}
```

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

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

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

7. caught MyException
9. hello
Exceptions in action two

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

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

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

9. caught Err
10. 65
## Exceptions in C++ and Java

<table>
<thead>
<tr>
<th>C++</th>
<th>Java</th>
</tr>
</thead>
<tbody>
<tr>
<td>can throw exceptions of objects or primitive types (such as int)</td>
<td>must be objects of classes derived from Exception</td>
</tr>
<tr>
<td>does not have to declare what exceptions may be thrown, but preferred</td>
<td>must declare what exceptions may be thrown</td>
</tr>
<tr>
<td>does not have to, but preferred</td>
<td>must be enclosed within a try-catch block, checked by compiler</td>
</tr>
<tr>
<td>Can use <code>catch (...) {</code> to catch all exceptions -- however, no object specified, no handle on the exception.</td>
<td>the catch block must identify the object, for example, <code>catch (MyException meobj) {</code></td>
</tr>
<tr>
<td>may throw different types of exceptions</td>
<td>same</td>
</tr>
<tr>
<td>does not have the equivalent</td>
<td>allows finally</td>
</tr>
</tbody>
</table>