# Exception Handling with C++

# Fundamentals

- An *exception* is a class
  - Usually derived from one of the system's exception base classes
- If an exceptional or error situation occurs, program *throws* an object of that class
  - Object crawls up the call stack
- A calling program can choose to *catch* exceptions of certain classes
  - Take action based on the exception object

# Class `exception`

- The standard C++ base class for all exceptions
- Provides derived classes with virtual function **`what`**
  - Returns the exception's stored error message

# Example: Handling an Attempt to Divide by Zero

```
// Class DivideByZeroException definition.
#include <stdexcept> // stdexcept header file contains runtime_error
using std::runtime_error; // standard C++ library class runtime_error

// DivideByZeroException objects should be thrown by functions
// upon detecting division-by-zero exceptions
class DivideByZeroException : public runtime_error
{
public:
   // constructor specifies default error message
   DivideByZeroException::DivideByZeroException()
      : runtime_error( "attempted to divide by zero" ) {}
}; // end class DivideByZeroException
```

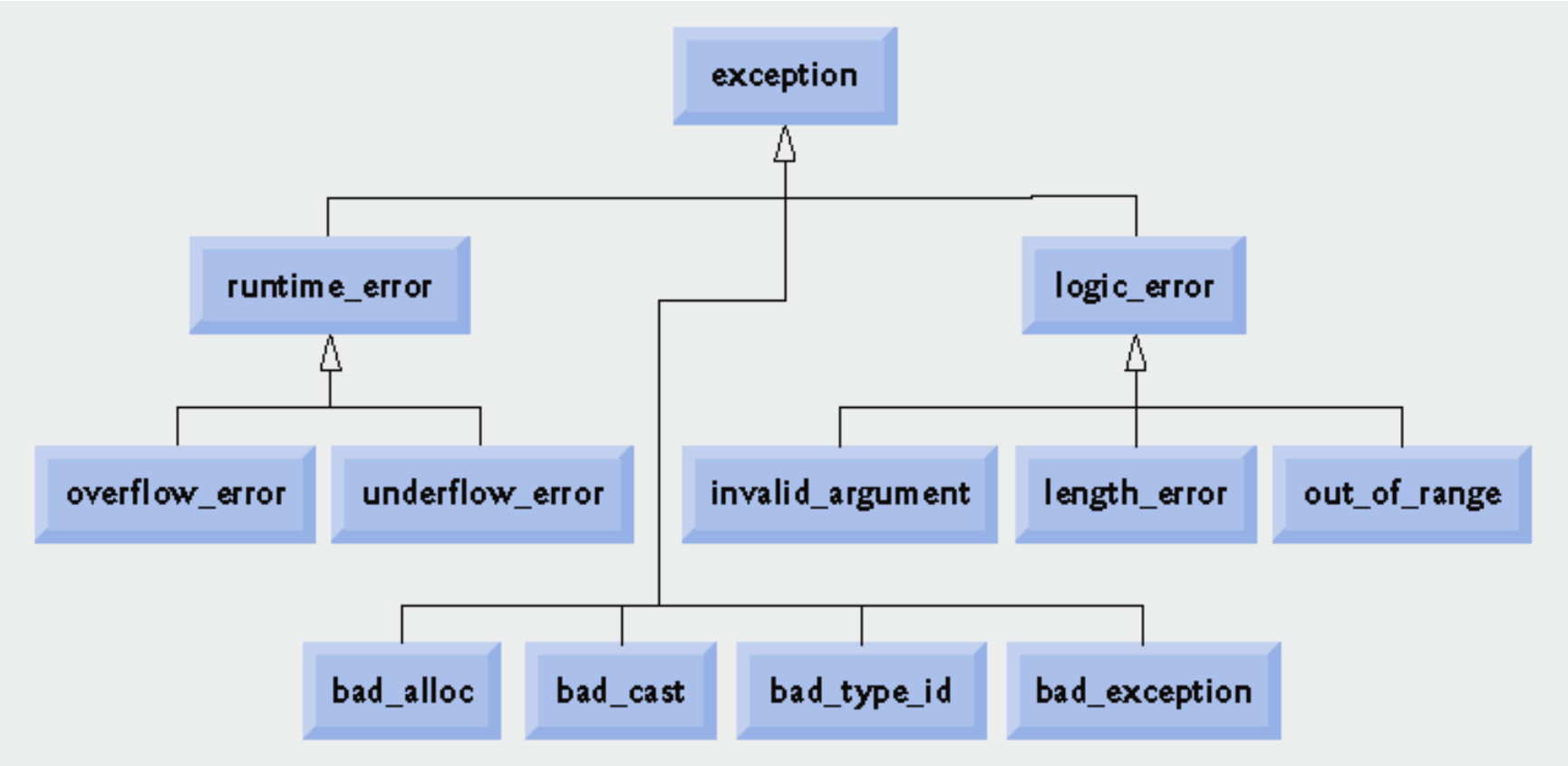
```
// A simple exception-handling example that checks for
// divide-by-zero exceptions.
#include <iostream>
using std::cin;
using std::cout;
using std::endl;

#include "DivideByZeroException.h" // DivideByZeroException class

// perform division and throw DivideByZeroException object if
// divide-by-zero exception occurs
double quotient( int numerator, int denominator )
{
   // throw DivideByZeroException if trying to divide by zero
   if ( denominator == 0 )
      throw DivideByZeroException(); // terminate function

   // return division result
   return static_cast< double >( numerator ) / denominator;
} // end function quotient

int main()
{
   int number1; // user-specified numerator
   int number2; // user-specified denominator
   double result; // result of division

   cout << "Enter two integers (end-of-file to end): ";
```

```

   // enable user to enter two integers to divide
   while ( cin >> number1 >> number2 )
   {
      // try block contains code that might throw exception
      // and code that should not execute if an exception occurs
      try
      {
         result = quotient( number1, number2 );
         cout << "The quotient is: " << result << endl;
      } // end try

      // exception handler handles a divide-by-zero exception
      catch ( DivideByZeroException &divideByZeroException )
      {
         cout << "Exception occurred: "
            << divideByZeroException.what() << endl;
      } // end catch

      cout << "\nEnter two integers (end-of-file to end): ";
   } // end while

   cout << endl;
   return 0; // terminate normally
} // end main
```

# Exception Handling

- Other standard exceptions are defined in the include <stdexcept>.

- Exception types are:



# Extending the Exception class

```
class MyException: public exception
{
        public:
        virtual const char* what() const throw()
        {
                return ”I just don’t like what happened\n";
        }
};
```

# Including Exceptions in Class Specs

```
Stack.h:
#include <stdexcept>
class Stack {
    double pop() throw(logic_error);
    ….
}
```

Can have a comma separated list of exceptions it will throw

Specifies that pop can **only** throw a logic_error exception. Unfortunately, the compiler does NOT check

# Exceptions

- Not specifying a "throw" in a declaration means any kind of exception may be thrown

- Specifying no exception

  ```
  double Stack::pop() throw()
  ```

  means no exceptions will be thrown

- If there is no "catch" AND an exception is thrown, then the program will call **`terminate`**

# Error Note

- The compiler will **<u>not</u>** generate a compilation error if a function contains a **`throw`** expression for an exception not listed in the function's exception specification.

- Error occurs only when that function attempts to **`throw`** that exception at run time.

- To avoid surprises at execution time, carefully check your code to ensure that functions do not **`throw`** exceptions not listed in their exception specifications

# Note

- When an exception is thrown from the constructor for an object that is created in a **`new`** expression, **…**

- **…** the dynamically allocated memory for that object is released.

# Exceptions and Inheritance

- A catch clause may use a exception class to catch "derived exceptions" from that class.
- Example:

```
void foo () throw(DerivedException);

try {
        foo();
} catch (BaseException& x) {
}
```