C++ Exception Handling

Exceptions in C++

- An exception can be any type!
- Exceptions can be programmer defined or exceptions from the C++ standard library.

```
struct Error { } e;
try {
   if ( n < 0 ) throw n;
   else if ( n == 0 ) throw "zero";
   else if ( n == 1 ) throw e;
}
catch (int e1)
    { cout << "integer exception raised" << endl; }
catch (string e2)
    { cout << "string exception " << endl; }
catch (Error e3)
    { cout << "struct Error" << endl; }
```

Standard Exceptions in C++

- C++ defines exception classes in <exception>.
- Hierarchy of classes:
  - exception (top level class)
    - runtime_error
    - logic_error
    - others
- Exceptions can be thrown by C++ language features:

  bad_alloc (thrown by "new")

  bad_cast (thrown by "dynamic_cast")

  bad_exception (generic exception)

Exceptions in C++

Class Hierarchy	include file
exception	<exception>
bad_alloc	<new>
bad_cast	<typeinfo>
bad_exception	<exception>
bad_typeid	<typeinfo>
failure <ios>	
logic_error (has subclasses)	<stdexcept>
runtime_error (has subclasses)	<stdexcept>

- bad_exception is a generic type for unchecked exceptions.

Exception Handler in C++

- Example: catch failure of "new".

```
#include <iostream>
using namespace std;
using std::bad_alloc;
char *makeArray(int nsize) {
   char *p;
   try {
      p = new char[nsize];
   } catch ( bad_alloc e ) {
      cout << "Couldn't allocate array: ";
      cout << e.what( ) << endl;
      p = null;
   }
```

C++ Rethrowing an Exception

In C++ *anything* can be "thrown".

```
try {
     sub(); // sub() can throw exception
} catch ( bad_alloc e ) {
     cerr << "Allocation error " << e.what();
     throw;
}
```

Declaring exceptions

- To declare that your function throws an exception:

```
#include <iostream>
using namespace std;
using std::bad_alloc;
char *makeArray(int nsize) throw(bad_alloc) {
   char *p;
   try {
      p = new char[nsize];
   } catch ( bad_alloc e ) {
      cout << "Couldn't allocate array: ";
      cout << e.what( ) << endl;
      throw; // re-throw bad_alloc exception
   }
```

Declaring no exceptions thrown

- To declare that your function throws no exceptions:

```
#include <iostream>
using namespace std;
using std::bad_alloc;
char *makeArray(int nsize) throw() {
   char *p;
   try {
      p = new char[nsize];
   } catch ( bad_alloc e ) {
      cout << "Couldn't allocate array: ";
      cout << e.what( ) << endl;
      return NULL;
   }
```

Exception Handler in C++

- A function can have multiple "catch" blocks.

```
int main( ) {
   // ... other code goes here ...
   try {
      sub(); /* sub() that throws exceptions */
   } catch ( bad_alloc e ) {
      cerr << "Allocation error " << e.what();
   }
   } catch ( exception e ) {
      cerr << "Exception " << e.what();
   }
   } catch ( ... ) {
      // "..." matches anything:  this catch
      // block catches all other exceptions
      cerr << "Unknown exception " << endl;
   }
```

C++ Default Exception Handler

- If an exception is not caught, C++ provides a default exception handler:
  - If the function didn't use "throw(something)" in its header, then a method named **`terminate()`** is called.
  - If a function declares exceptions in its header, but throws some *other* exception, then the function **`unexpected()`** is called. **`unexpected()`** also calls **`terminate()`**.

C++ Default Exception Handler

- **`unexpected()`** in implemented as a pointer. You can change it to your own exception handler using: **`set_unexpected(`** *your_function* **`)`**
- Similarly, use **`set_terminate()`** to replace **`terminate()`** with some other function.
- Prototypes for set_unexpected() and set_terminate() are defined in the header file <**`exception`**>.

C++ Default Exception Handler

```
#include <exception>
void my_terminator() {
  cerr << "You're terminated!" << endl;
  exit(1);
}
void my_unexpected() {
  cout << "unexpected exception thrown" << endl;
  exit(1);
}
int main() throw() {
  set_unexpected(my_unexpected); // ignore return value
  set_terminate(my_terminator);
  for(int i = 1; i <=3; i++)
  try { f(i); }
  catch(some_exception e) {
     cout << "main: caught " << e.what() << endl;
   throw;
}
```

Rethrow and Exception

Inside a "catch" block, you can throw the same exception by writing "**`throw`**" with no parameter.

```
try {
     fun(x);
}
catch(some_exception e) {
     cout << "main: caught " << e.what() << endl;
    throw;
}
```

Syntax of Try - Catch

If an exception occurs, control branches to the first matching "catch" clause.

```
try {
   statements;
}
catch( ExceptionType1 e1 ) {
   doSomething;
}
catch( ExceptionType2 e2 ) {
   doSomethingElse;
}
```