DigiPen INSTITUTE OF TECHNOLOGY

CS170#10.2

Unhandled And Unexpected Exceptions

Vadim Surov

Outline

- Exception Specification
- Re-throwing Exceptions
- Unhandled Exceptions
- Unexpected Exceptions

Exception Explicit Specification

- **Deprecated in C++11 !** (because not enforced and performance drain)

- You can "tag" a function with an exception specification

```
double QRoot(double a, double b, double c)
    throw(const char *, double)
{
}
```

Makes it very clear to the user what to expect

- It isn't foolproof: depends on compiler and options you **can still throw other exceptions** (called unexpected)

Exception Explicit Specification (contd)

- To indicate that a function **doesn't throw** any exceptions, use empty parentheses ():

```
int SomeFun(int a, int b, int c) throw() {
}
```

- To indicate that a function **could throw** an exception without type specification use ...:

```
void MyFunc() throw(...) { }
```

What Does The Following Output?

```
double QRoot(   double a,
                double b,
                double c)
    throw(const char *, double)
{
  double determinant =
            (b*b)-(4*a*c);

  if (determinant < 0)
    throw(determinant);
  else if (a == 0)
    throw("Division by 0.");

  return (-b+sqrt(determinant))/
          (2 * a);
}
```

```
void main(void) {
  try {
    cout << "QRoot a=3, b=2, c=1: "
         << QRoot(3, 2, 1) << endl;
  } catch (const char *message) {
    cout << message << endl;
  } catch (double value) {
    cout << value << endl;
  }
  try {
    cout << "QRoot a=0, b=2, c=1: "
       << QRoot(0, 2, 1) << endl;
  } catch (const char *message) {
    cout << message << endl;
  } catch (double value) {
    cout << value << endl;
  }
}
```

What about this?

```
double QRoot(   double a,
                double b,
                double c)
    throw(const char *, double)
{
  double determinant =
            (b*b)-(4*a*c);

  if (determinant < 0)
    throw(determinant);
  else if (a == 0)
    throw("Division by 0.");

  return (-b+sqrt(determinant))/
          (2 * a);
}
```

```
void main(void)
{
  try {
    cout << "QRoot a=3, b=2, c=1: "
     << QRoot(3, 2, 1) << endl;
    cout << "QRoot a=0, b=2, c=1: "
     << QRoot(0, 2, 1) << endl;
  }
  catch (const char *message) {
    cout << message << endl;
  }
  catch (double value) {
    cout << value << endl;
  }
}
```

Exception Specification In C++ 11

- Again: Exception specifications are a C++ language feature that is **deprecated** in C++11

- Use `noexcept(<bool-const-expression>)` to specifies whether a function might throw exceptions.
  - `noexcept` means `noexcept(true)`

Re-throwing Exceptions

- You can propagate exceptions from where they are (first) caught
- Just use the keyword `throw` by itself to re-throw the same (current) exception. See next slide

```
void f1(void)
{
    try {
        QRoot(0, 5, 3);  // division by 0
    }
    catch (const char *s) {
        cout << "Caught error in f1: " << s << endl;
        throw;
    }
}

void main(void)
{
    // protect code
    try {
        f1();
    }
    catch (const char *message) {  // catch a char pointer exception
        cout << message << endl;
    }
    catch (double value) {   // catch an integer exception
        cout << value << endl;
    }
}
```

```
C:\WINDOWS\system32\cmd.exe
Caught error in f1: Division by 0.
Division by 0.
Press any key to continue . . .
```

- You can `catch` one type of exception and `throw` another:

```
void f1(void)
{
    try {
        QRoot(0, 5, 3);  // division by 0
    }
    catch (const char *s) {
        throw("Error! Please call 1-800-DIV-ZERO for help");
    }
}

void main(void)
{
    // protect code
    try {
        f1();
    }
    catch (const char *message) {  // catch a char pointer exception
        cout << message << endl;
    }
}
```

```
C:\WINDOWS\system32\cmd.exe
Error! Please call 1-800-DIV-ZERO for help
Press any key to continue . . .
```

Unhandled Exceptions

- If a matching catch handler cannot be found for the current exception, the predefined `terminate()` run-time function is called
- The default action of `terminate()` is to call `abort()`
- If you want `terminate()` to call some other function in your program before exiting the application, call the `set_terminate()` with the name of the function to be called as its single argument
- You can call `set_terminate()` at any point in your program
- The `terminate()` always calls the last function given as an argument to `set_terminate()`

Unhandled Exceptions (contd)

- The following example code throws a `char *` exception, but does not contain a handler designated to catch exceptions of type `char *`

- The call to `set_terminate()` instructs `terminate()` to call `term_func()`:

```cpp
#include <eh.h>        // For function prototypes
#include <iostream>
#include <process.h>

void term_func()
{
    //...
    std::cout << "term_func was called by terminate." << std::endl;
    exit( -1 );
 }
int main()
{
    try
    {
        // ...
        set_terminate( term_func );
        // ...
        throw "Out of memory!"; // No catch handler for this exception
    }
    catch( int )
    {
        std::cout << "Integer exception raised." << std::endl;
    }
    return 0;
}
```

```
C:\WINDOWS\system32\cmd.exe
term_func was called by terminate.
Press any key to continue . . .
```

Unexpected Exceptions

- The C++ Standard requires that `unexpected()` is called when a function throws an exception that is not on its throw list
- unexpected calls `terminate()` by default
- You can change this default behavior by writing a custom termination function and calling `set_unexpected()` with the name of your function as its argument
- **The C++ 11 will not call** `unexpected()`, Compiler will assume that the function does not throw

The following example calls `unexpected()` directly, which then calls the `unfunction()`, which then calls the `terminate()`

Global Scope)

```cpp
// exception_set_unexpected.cpp
// compile with: /c /EHsc
#include<exception>
#include<iostream>

using namespace std;

void unfunction( )
{
   cout << "I'll be back." << endl;
   terminate( );
}

int main( )
{
   unexpected_handler oldHand = set_unexpected( unfunction );
   unexpected( );
}
```