CS170#10.1

Exceptions

Vadim Surov

Outline

- Error Handling
- Error Handling Using Exceptions
- `try...catch`
- `throw...catch`

Error Handling

- Some program need to detect and, if possible, recover from error situations, like:
  - `new` not allocating memory
  - out of bounds array subscript
  - division by zero
  - invalid function parameters
  - bad casting
  - construction failure
- These error situations are called **exceptions** in C++
- An exception is something that must be *handled*, or the program must terminate
- Until now, for simplicity, we've ignored the possibility of (many) errors occurring. Exceptions let us handle errors in a more unified and object-oriented way

Error Handling Case 1

- Recall that given a quadratic equation of the form:

$$ax^2 + bx + c = 0$$

- We can solve the equation for its roots with this formula:

$$x_1 = \frac{-b + \sqrt{b^2 - 4ac}}{2a} \qquad x_2 = \frac{-b - \sqrt{b^2 - 4ac}}{2a}$$

- For simplicity, we will only calculate the first root

```cpp
// QRoot.cpp : Defines the entry point for the console application.
//
#include <iostream>
#include <cmath>
using namespace std;

double QRoot(double a, double b, double c)
{
    double determinant = (b * b) - (4 * a * c);
    return (-b + sqrt(determinant)) / (2 * a);
}

void main(void)
{
    // -0.438447
    cout << "QRoot a=1, b=5, c=2: " << QRoot(1, 5, 2) << endl;

    // Error, taking square root of negative number (-1.#IND)
    cout << "QRoot a=1, b=2, c=5: " << QRoot(1, 2, 5) << endl;

    // Error, divide by 0 (-1.#IND)
    cout << "QRoot a=0, b=2, c=5: " << QRoot(0, 5, 2) << endl;
}
```

Error Handling Case 1 (contd)

```
C:\WINDOWS\system32\cmd.exe
QRoot a=1, b=5, c=2: -0.438447
QRoot a=1, b=2, c=5: -1.#IND
QRoot a=0, b=2, c=5: -1.#IND
Press any key to continue . . .
```

- -1.#IND stands for "indefinite“ if a variable hasn't been initialized or “infinity” if caused by a division that produces a result too large to represent
- What are the pros and cons of this approach?

Error Handling Case 2

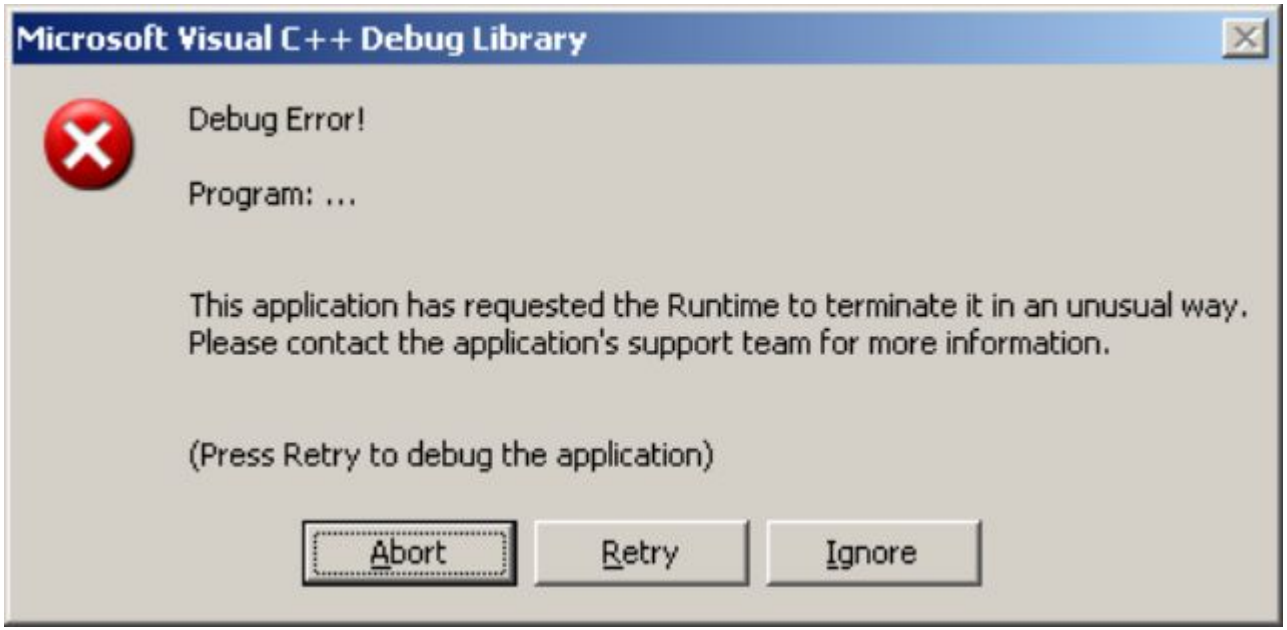
- Let's check the data *before* using it, and if it's bad:
  - Issue an error message
  - Terminate the program
- Otherwise continue as normal

Global Scope) | QRoot(double a, double b, double c)

```
}
*/

double QRoot(double a, double b, double c)
{
  double determinant = (b * b) - (4 * a * c);

  if (determinant < 0) // protected against: sqrt(-x)
  {
    cout << "Can't take square root of a negative number." << endl;
    abort();
  }
  else if (a == 0)      // protected against: x / 0
  {
    cout << "Division by 0." << endl;
    abort();
  }

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

void main(void)
{
    // -0.438447
    cout << "QRoot a=1, b=5, c=2: " << QRoot(1, 5, 2) << endl;
```

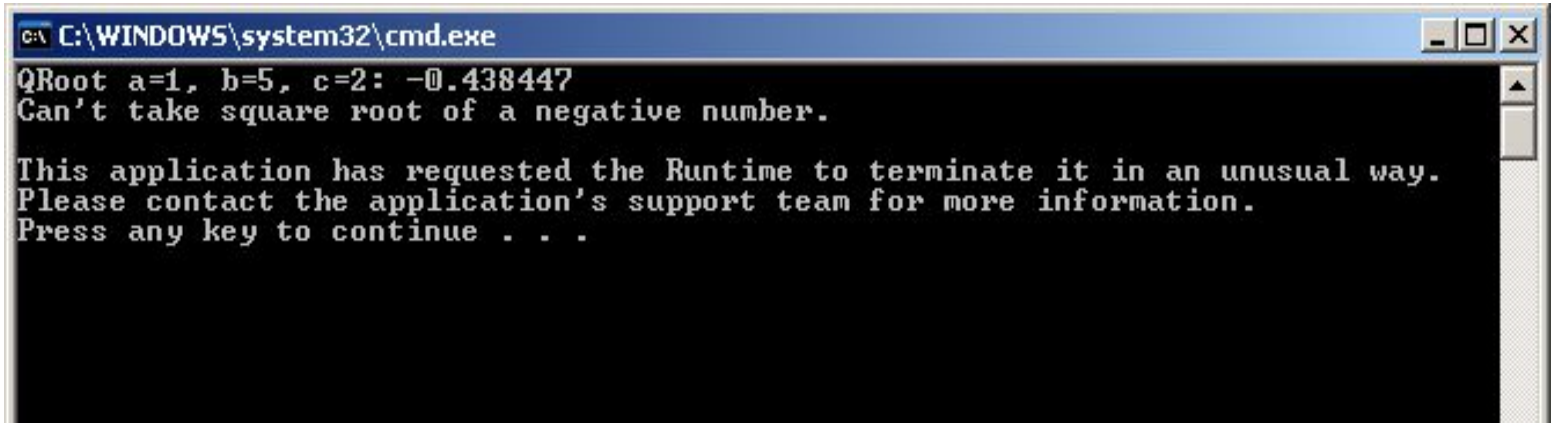
Error Handling Case 2 (contd)

- The result when running in debug mode:



Error Handling Case 2 (contd)

- The result when running in Release mode:



- What are the pros and cons of this approach?

Error Handling Case 3

- Check the data *before* using it, and if it's bad:
  - Don't use the data
  - Set the output to some "safe" value
  - Return immediately with a status of "FAIL"
- Otherwise continue as normal
- Return a status of "OK“
- What are the pros and cons of this approach?

(Global Scope) | QRoot(double a, double b, double c, double * result)

```
// Error handling 3
bool QRoot(double a, double b, double c, double *result)
{
    double determinant = (b * b) - (4 * a * c);

    // protected against: sqrt(-x), x / 0
    if ( (determinant < 0) || (a == 0) )
    {
    *result = 0.0; // have to return something
    return false;  // indicates there was a problem
    }

    *result = (-b + sqrt(determinant)) / (2 * a);
    return true;  // indicates all is well
}

void main(void)
{
    double answer;
    double success = QRoot(1, 5, 2, &answer);
    if (success)
    cout << "QRoot a=1, b=5, c=2: " << answer << endl;
    else
    cout << "QRoot failed for some reason" << endl;
}
```

C:\WINDOWS\system32\cmd.exe

```
QRoot a=1, b=5, c=2: -0.438447
Press any key to continue . . .
```

Error Handling Using Exceptions

- There's a better way: **exceptions**
- It's built into the C++ language
- More powerful and flexible, but also more complex
- Can start off simple and then grow into complex
- When you detect an exceptional situation, you `throw` an exception
- When you want to handle the exception, you `catch` it
- Program code that wants to `catch` an exception must be placed in a `try` block
- These three facilities (`try`, `throw`, and `catch`) are the foundation of the C++ exception mechanism

Error Handling Using Exceptions (contd)

- In client code:
  - Protect potential "bad" code by placing a `try` block around it
  - Provide a `catch` block that will handle any exceptions thrown in the `try` block.
    - `catch` block can catch exceptions by value, reference, const reference, or pointer
- In the non-client code:
  - Check the data *before* using it
  - If it's bad, `throw` an exception that identifies the problem
  - If it's good, use it as normal

Format Of The `try...catch` Mechanism

```
void main(void)
{

   try
   {
      // code that might cause an exception (throw)
      // and needs to be protected
   }
   catch (...) // which kind(s) of exceptions to catch?
   {
      // code that will handle the exception (catch) from
      // the try block above
   }

}
```

You Can Catch Multiple Exceptions

```
try {

  // code that might cause an exception (throw)
  // and needs to be protected

} catch (const char *p) { // catch a char pointer
    // code that will handle the char pointer exception
    // from the try block above
} catch (int i) { // catch an integer
    // code that will handle the integer exception
} catch (exception e) { // catch an "exception" object
      // code that will handle the "exception" object
} catch (...) { // catch any type (wild card)
      // code that will handle any other exception
}
```

Order Of The Catch Blocks

- The order of the `catch` handles is <u>important</u>:
  - The catch handles are evaluated in the order they appear in the source code
  - If a catch handle is of the correct (matched) type, it will be used to handle the exception
    - The type must be an **exact match**, **or** it must be **derived** from the type that is declared (see classes inheritance)
    - **No implicit conversions** are done (such as from `int` to `double`)

`throw...catch`

- `throw` is sort of like the return; the program "jumps" out immediately
- Where does it "jump" to? One of the `catch` blocks
- Which `catch` block? The one associated with the most recent enclosing `try` block (The exception type must match the type in the `catch` block)
- Exceptions that are not caught end up calling the `abort()` method and terminating the program

(Global Scope)

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

    // protected against sqrt(-x) and division by 0
  if (determinant < 0)
    throw("Can't take square root of a negative number.");
  else if (a == 0)
    throw("Division by 0.");

    // We only reach this point if no exception was thrown
  return (-b + sqrt(determinant)) / (2 * a);
}

void main(void)
{
  try  // protect code
  {
    cout << "QRoot a=0, b=5, c=2: " << QRoot(0, 5, 2) << endl;
  }
  catch (const char *message)  // catch a char pointer exception
  {
    cout << message << endl;
  }
}
```

C:\WINDOWS\system32\cmd.exe

```
Division by 0.
Press any key to continue . . .
```