CSCE 120/121
Introduction to Program Design & Concepts

A First Look at Exceptions

Dr. Tim McGuire

Grateful acknowledgment to Dr. Philip Ritchey and Dr. Michael Moore for some of the material on which these slides are based.

Communicating Error Information: Use return value of function

- Return an invalid value to indicate an error, e.g. if a function returns an area, return negative numbers to indicate there was an error in the function.
- Not all functions have an invalid value to use
- What if the function uses other functions that can have errors?
  - So each function must know errors of each function it calls and be able to pass that information in a return value. Maybe not one but many values.
  - Messy!

Error returned as a value

```cpp
#include <iostream>
#include <cmath>
using namespace std;

int main() {
  double a = 0;

  //division by zero
  cout << "a (zero):          " << a << endl;
  a = 7.0/0;
  cout << "a (div by zero):  " << a << endl;

  //negative square root
  a = sqrt(-5.0);
  cout << "a (neg sqr root): " << a << endl;
}
```

Output:

```
a (zero):          0
a (div by zero):   inf
a (neg sqr root):  -nan
```

floatingPointExceptions.cpp

Use return value of function: Problems

- Not all functions have an invalid value to use
- What if the function uses other functions that can have errors?
  - So each function must know errors of each function it calls and be able to pass that information in a return value. Maybe not one but many values.
  - Messy!

Communicating Error Information: Use separate channel

- Bypass return value of function
- Functions don't have to be omniscient!

- Exceptions!

Exceptions

- Indicate that something unexpected has occurred or been detected
- Allow program to deal with the problem in a controlled manner
- Can be as simple or complex as program design requires

Exceptions - Terminology

- Exception: object or value that signals an error
- Throw an exception: send a signal that an error has occurred
- Catch/Handle an exception: process the exception; interpret the signal

Exceptions – Key Words

- `throw` – followed by an argument, is used to throw an exception
- `try` – followed by a block `{ }`, is used to invoke code that throws an exception
- `catch` – followed by a block `{ }`, is used to detect and process exceptions thrown in preceding `try` block. Takes a parameter that matches the type thrown.

Exceptions – Flow of Control

1) A function that throws an exception is called from within a try block
2) If the function throws an exception, the function terminates and the try block is immediately exited. A catch block to process the exception is searched for in the source code immediately following the try block.
3) If a catch block is found that matches the exception thrown, it is executed. If no catch block that matches the exception is found, the program terminates.

Exceptions – Example (1)

```
// function that throws an exception
int totalDays(int days, int weeks)
{
    if ((days < 0) || (days > 7))
      throw string("invalid number of days“);
// the argument to throw is the character string
   else
      return (7 * weeks + days);
}
```

Exceptions – Example (2)

```
try // block that calls function
{
    totDays = totalDays(days, weeks);
    cout << "Total days: " << days;
 }
 catch (string msg) // interpret exception
 {
    cout << "Error: " << msg;
 }
```

Exceptions – What Happens

1) `try` block is entered. `totalDays` function is called
2) If 1st parameter is between 0 and 7, total number of days is returned and `catch` block is skipped over (no exception thrown)
3) If exception is thrown, function and `try` block are exited, `catch` blocks are scanned for 1st one that matches the data type of the thrown exception. `catch` block executes

See also ExceptionCaught.cpp

Exceptions and Objects

- An <u>exception class</u> can be defined in a class and thrown as an exception by a member function
- An exception class may have:
  - no members: used only to signal an error
  - members: pass error data to `catch` block
- A class can have more than one exception class

What Happens After `catch` Block?

- Once an exception is thrown, the program cannot return to throw point. The function executing `throw` terminates (does not return), other calling functions in `try` block terminate, resulting in unwinding the stack

Recall

- When an unexpected condition happens, you "**throw**" an exception
- The **try block** is the part of your code where an exception might occur
- You "**catch**" exceptions and deal with them in an exception handler

Handling runtime errors

```
cout << “Enter a month (1-12): ”;
int month;
cin >> month;
if (month < 1 || month > 12) {
    cout << “Invalid month.” << endl;
}
```

Handling runtime errors

```
cout << "Enter a month (1-12): ";
int month;
cin >> month;
if (month < 1 || month > 12) {
    throw 1;
}
```

Throwing an int!

Notes on Exceptions

- We can throw any type of variable as an exception, so:
  - We may want to include more detailed information about an error
  - We can create/use special variable types just for exceptions (Objects & Classes)
- zyBook uses the runtime_error.
  - Good for you to use for most of the class.

When an exception is thrown

1. Function stops executing immediately
2. Scan down looking for a catch
3. Either
    1. Catch block found
        1. If type matches thrown object
            1. Execute catch block
            2. Resume execution after catch block(s) (not back to where throw happened)
        2. Else continue scan (note there might be another catch immediately)
    2. End of function reached

When an exception is thrown

2. End of function reached
    1. Remove function from call stack
        - Variables go out of scope
            - Includes calling destructors (for example, file streams will close their files)
        - If the function removed from stack is main, the program terminates
    2. Throw exception to calling function
        - Return to function where call occurred
        - Scan down looking for a catch (i.e. go back to main step 2)

Exceptions propagate down call stack until caught or program terminates.

Separation of Concerns

- Recall that a function should do one thing well.

- What to do when a function gets bad data?
  - Temptation: Do cout/cin to get a replacement value.
    - Not all programs are interactive.
    - What if the data is from a sensor?
    - What if the interactive program is using another language?

  - Action: Throw exception
    - Function identifies error and throws an exception.
    - Elsewhere in the program the exception is caught and handled.

Avoid mixing I/O with another action within a function.

Exceptions

Stops execution immediately when throw occurs.

```
try {
        // ...
        if (problem)
                throw runtime_error("problem");
        // ...      does not get executed
}
catch ( runtime_error &e ) {
        // handle exception
}
```

Throwing/Catching in same function

```
void myfunction () {
        try {
                if (problem)
                        throw runtime_error("problem");
        }
        catch(runtime_error &e) {

        }
}
```

Generally NOT done.

Handle in else part of if statement.

Throwing in a function

```
void myfunction () {
        // ...
        if (problem)
                throw runtime_error("problem");
        // ...
}
```

Function only has to identify issue, not resolve it.

Catching in calling function

```
void myfunction () {/* throws exception */ }

int main () {
        try {
                myfunction();
        }
        catch (runtime_error &e) { // handle error
        }
}
```

Deal with problem where it can be addressed.

Propagation of exception down call stack

```
void myfunction () {/* throws exception */ }

void second_function() {
        myfunction()
}

int main () {
        try {
                second_function();
        }
        catch (runtime_error &e) { // handle error
        }
}
```

You should catch it somewhere.

If it propagates past main, program will crash!