CIT237
Chapter 16: Exceptions and Templates (Part 1: Exceptions)

November 12, 2019

Reminders

- Quiz 6 will be held at the start of class on Wednesday, November 20.
- The material covered on Quiz 6 will be:
  - Lectures of October 28 through November 13.
  - Chapters 15, 16 and 17.
- Project 3:
  - We discussed Project 3 in class last week.
  - The due date is December 2.

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 <u>may</u> throw 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 may throw 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.

Example: `try` / `catch` Sequence

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

Example: `throw` an Exception

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

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

Exception Not Caught?

- An exception will not be caught if
  - it is thrown from outside of a `try` block
  - there is no `catch` block that matches the data type of the thrown exception
- If an exception is not caught, the program will terminate

Exceptions and Objects

- An exception class can be defined *within* 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

Exception Class Example (1)

```
class InvalidPayRate {
private:
     double payRateValue;
public:
     InvalidPayRate(double input)
     {
         payRateValue = input;
     }
     double getInvalidPayRate()
     {
         return payRateValue;
     }
};
```

Exception Class Example (2)

```
try
{
        . . .
      testPayRate(payRate);
        . . .
}
catch (InvalidPayRate e)
{
    cout << "Invalid Pay Rate: "
         << e.getInvalidPayRate();
}
```

Exception Class Example (3)

```
bool testPayRate(double rate)
{
   if (rate < 0)
      throw InvalidPayRate(rate);
   return true;
}
```

What Happens After `catch` Block?

- Once an exception is thrown, the program cannot return to the throw point.
- If the **`throw`** is executed inside some function which was called within the **`try`** block, then that function terminates (does not return); other calling functions in the **`try`** block terminate, resulting in unwinding the stack
- If objects were created in the **`try`** block and an exception is thrown, they are destroyed.

Nested `try` Blocks

- `try/catch` blocks can occur within an enclosing `try` block
- Exceptions caught at an inner level can be passed up to a `catch` block at an outer level:

```
catch ( )
{
    ...
    throw;  // pass exception up
}          // to next level
```

Summary

- Exceptions: a (relatively) clean mechanism for handling errors in a program.
  - `throw`
  - `try`
  - `catch`