Exceptions II

Reminder

- Final exam
  - The date for the Final has been decided:
  - Saturday, November 16th
  - 8am – 10am
  - 01-2000

Project

- Clock Problem
  - Given back today
  - Grading
    - Functionality
    - Design
    - Style
- Parking Lot Problem: due Nov 11th

Project Notes

- Change your generic solver?
  - Don't forget to change your Clock and Farmer problem as well.
- Memory Management
  - Using purify
    - Add to Makefile:
      - `CCC = purify CC`
    - Or better yet, create a file `header.mak`
  - `Workshop` also has memory management tools.

New plan

- Today: Exceptions 2
- Monday: Files 1
- Tuesday: Exam / Files 2
- Thursday: Files 3

Enter…the exception

- Exceptions allow a method to tell the caller when an error has occurred
  - Many times it is the calling function that knows what to do when an error occurs.
  - Exceptions allow the caller to respond to the error rather than the method itself.
  - Different callers may wish to respond to particular errors differently.

Throwing exceptions

- In C++, exceptions are thrown by using the `throw` keyword.
  - Unlike Java, there is not a Throwable class.
  - In C++, any item can be thrown
    - Basic datatypes (int, float, etc.)
    - Class objects
    - Pointers to class objects
    - References to class objects

Catching Exceptions

- Like in Java, C++ uses a `try/catch` block for catching exceptions.

```
void f()
{
     try {
        // call to a method that may throw something
    }
    catch (Overflow) {
          // code that handles an overflow error
          ...
    }
    ...
  }
```

Stack unwinding

- If an exception is caught and handled
  - Execution continues from next statement after the try/catch block.

Catching Exceptions

- In C++, there is no `finally` section of the try/catch block.
- In Java, the `finally` code is executed regardless of whether an exception was caught.
  - Allows for cleanup of system resources.

Catching Exceptions

```
void use_file (const char *name)
{
     FILE *f = fopen (…);
     try {
         // some file operation
     }
     catch (…) {
          fclose (f);
          throw;
     }
     fclose(f);
}
```

Stack unwinding

- When an exception is thrown in C++
  - Call stack is searched for first function to catch the data thrown.
    - If none found, program will terminate.
    - If one is found:
      - All local variables from all methods on stack from method that threw the exception to that which caught it, will have it's destructor called.
      - Note that this is not true for objects allocated on the heap.

Stack unwinding

- Advantageous to wrap system resource calls into class objects.
  - The resource can be cleaned up during object destruction.

```
void use_file (const char *name)
 {
       FileObject f (…)
       // do something with f
 }
```

Exceptions and Constructors

- If an exception is thrown during the call to an object's constructor, only the data members that have been completely are destroyed.

Exceptions and Constructors

```
class X {
    Y  yy;
    Z zz;
public:
     X ( char *foo, int bar) :
       yy (foo), zz (bar) {}
     …
}
```

Exceptions and Constructors

- An object is not completely constructed until it's constructor completes.
  - Beware of data members allocated on the heap

Exceptions and Constructors

```
class Y {
private:
    int *p;
    void init();
public:
     Y (int s) { p = new int [s];
  init();}
     ~Y () { delete [] p;}
     …
}
```

Exceptions and Constructors

```
class Y {
private:
    vector<int> p;
    void init();
public:
     Y (int s) { p (s); init();}
}
```

Exceptions and Constructors

- Questions?

Using exceptions

- Important safety tips (from Stroustrup, the inventor of C++)
  - Use exceptions for error handling
  - Throw an exception to indicate failure during construction
  - Use exception specifications (good style…in style guide)
  - Beware of dynamically allocated data members when throwing an exception from a constructor

Using exceptions

- Important safety tips
  - Assume that every exception that can be thrown by a function will be thrown.
  - Don’t assume that exceptions will be derived from the “standard” exception class.
  - Libraries should not terminate a program. Throw an exception instead.
  - Think about error handling early in a design

Exceptions vs. Assertions

- Exceptions
  - Let the caller decide how to handle the error.
- Assertions
  - Aborts the program
  - Debugging tool
    - Should not be included in the release version of software.

Exceptions vs. Assertions

- What about for testing preconditions?

```
// push
//
// Description: adds a new element to "the top of" the
   stack
// Arguments:   the element to be added
// Pre:         stack is not full
// Post:        size has increased by one
// Post:        top is equal to the argument newElement
//
virtual void push( char newElement ) = 0;
```

Exceptions vs. Assertions

- My own humble philosophy
  - Use assertions for errors that are under your control, as a programmer.
  - Use exceptions for error that are under the control of a user of the system or user of your code.

Exceptions

- Questions?

- Do we have time to look at some code?