Exceptions

Reminder

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

Announcement

- October 31st is halloween
  - Dress up and win big prizes!
  - 4pm…Refreshments at CS Offices

Project

- Clock Problem
  - Sorry, still not ready…definitely Thursday
- Farmer Problem: due Oct 30th

Project Notes

- Change your generic solver?
  - Don't forget to change your Clock 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
- Thursday: Exceptions 2 / Files1
- Monday: Files 2
- Tuesday: Exam / Files 3

When things go wrong

- When a program comes upon a problem it cannot solve locally it can:
  - Terminate the program
    - E.g. assertion
  - Return an "error" value
  - Return a valid value and leave the program in a "bad" state.
    - E.g. IOStreams
  - Call some "error handling" functions

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.

C++ Exceptions

- The idea behind C++ exceptions is very much like Java exceptions.
- Like all things C++, though, C++ exceptions do have their quirks.

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

Throwing exceptions

```
class Stack
{
public:
    bool isFull();
    void push();
private:
    int size;
    ...
};


void Stack::push()
{
...
if (isFull()) throw size;
}
```

Throwing exceptions

- Like in Java, it is more useful to create a heirarchy of Exception classes.

```
class MathError { };
class Overflow : public MathError {  };
class Underflow : public MathError { };
class DivideByZero : public MathError { };
```

Throwing exceptions

- Exception classes are not special.
  - They can contain methods/data like any other class.

```
class MathError {
    // ...
     virtual void printMessage() const;
   }
```

  - They can also be derived from multiple classes

```
class NetfileError : public NertworErr,
  public FileError { … }
```

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
          ...
    }
    ...
 }
```

Catching Exceptions

- Rules for catching exceptions:

```
try { // something of type E is thrown }
catch (H) { // when is the handler invoked?}
```

- Handler is invoked if
  1. H is the same type as E
  2. E is derived from H
  3. H and E are pointers and 1 & 2 apply to the things they point to.
  4. H is a reference and 1 & 2 hold for the type H refers to.

Catching Exceptions

```
void f()
{
   throw MathError();
}

void g()
{
    try { f (); }
    catch (MathError E) {  E.printMessage(); }
}
```

Copying will occur

Catching Exceptions

```
void f()
{
   throw Overflow();
}

void g()
{
    try { f (); }
    catch (MathError E) {  E.printMessage(); }
}
```

Slicing Will Occur

Catching Exceptions

```
void f()
{
   throw new MathError();
}

void g()
{
    try { f (); }
    catch (MathError *E) {
       E->printMessage();
       delete E;  // to prevent a memory leak
    }
}
```

Catching Exceptions

```
void f()
{
   throw Overflow();
}

void g()
{
    try { f (); }
    catch (MathError &E) {  E.printMessage(); }
}
```

No Slicing Will Occur

Catching Exceptions

- To catch anything, regardless of type, use the … syntax.

```
try {
     // something
 }
 catch (...) {
     // catches anything thrown at you
 }
```

Rethrowing exceptions

- Once caught, an exception can be rethrown by using the throw keyword:

```
try {
     // something
 }
 catch (...) {
     // catches anything thrown at you
     // and throws it back
     throw;
 }
```

Catching exceptions

- Handlers in a try / catch block are tried in the order in which they appear.

```
try {// something}
catch (Overflow)
     { // handle overflow}
catch (MathError)
     { // handle any math error}
catch (...)
     { // handle anything}
```

Catching exceptions

- Erroneous ordering:

```
try {// something}
  catch (...)
       { // handle anything}
  catch (MathError)
       { // it'll never get here}
  catch (Overflow)
       { // or here}
```

Catching exceptions

- Questions

Exception specification

- Like Java, what gets thrown by a method can be declared when defining the function.
- Unlike Java, this declaration is not required.
  - But if there, it is guaranteed to throw only what's specified.

Exception specification

```
void f() throw (Overflow, int)
{
   if ( ) throw Overflow();
   else throw 7;
}
```

Exception specification

- If unspecified (default), the function may throw anything:

```
int f();  // can throw anything
```

- To indicate that a function will never throw an exception

```
int g() throw ();  // throws nothing
```

Exception specification

- In exception hierarchies, derived classes may only restrict what is thrown

```
class B {
    virtual void f();    // can throw anything
    virtual void g() throw (X, Y);
    virtual void h() throw (X);
 }
 class C : public B {
    virtual void f()  throw (X);      // ok
    virtual void g() throw (X);       // ok
    virtual void h() throw (X,Y);     // not okay
 }
```

Exceptions in Java

- When an exception is thrown, the exception gets passed to the calling function.
- This function may:
  - *Catch* the exception, then perform whatever error handling is appropriate or
  - Pass the exception up the call stack to the function that called it.
- If an exception reaches the main method and is not caught and handled, the program will terminate.

Exceptions in C++

- Same is true in C++ except:
  - Pass the exception up the call stack is implicit
  - No need for function to specify this in definition of function.

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

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

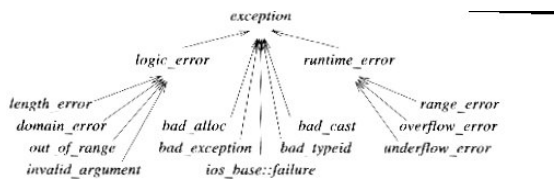
Stack unwinding

```
Stuff *pointer( 0 );
try {
    Stuff direct( "direct" );
    pointer = new Stuff( "dynamic" );
    direct.test();    // int thrown here
    delete pointer;
}
catch( int x )
{
     cout << "Exception #" << x << " caught" << endl;
     delete pointer;
}
```

Standard Exceptions

- There are some standard exceptions
  - `bad_alloc` – thrown by `new`
  - `bad_cast` – thrown by `dynamic_cast`
  - `bad_typeid` – thrown by `typeid`
- STL exceptions
  - `out_of_range`
  - `invalid_argument`
  - `overflow_error`
  - `ios_base::failure`

Standard Exceptions



Standard Exceptions

- There is no guarantee or rule that forces one to derive their exceptions from this hierarchy.

Exceptions

- Questions?

- Next time
  - Using exceptions in practice