Exceptions

C++ Exceptions

- A C++ exception is an abrupt transfer of control, usually resulting from an error condition.
- When an error condition is encountered, the programmer may choose to *throw* an exception.
- This initiates an *immediate* transfer of control. But to where?
- An assumption is made that if the programmer has chosen to throw an exception, he/she has also provided a place to *catch* the exception.
- Perhaps a simple example would help...

```
enum MathErr { noErr, divByZero, genericOverflow };
float divide(float numerator, float denominator)
{
  if (denominator == 0)
    throw divByZero;
  return numerator/denominator;
}
```

Somebody Catch Me!!!

- An assumption is made that the programmer has set up a place for exceptions to be caught when they occur.
- This is done with a *try block*.
- It looks something like this:

```
int main()
{
  try {
    cout << “3/2 is “ << divide(3,2) << endl;
    cout << “2/0 is “ << divide(2,0) << endl;
  }
  catch(MathErr x) {
    if (x == divByZero)
      cerr << “Divide by zero caught. “ << endl;
    else cerr << “Other error caught. “ << endl;
  }
}
```

Somebody Catch Me!!!

- The `try` statement simply defines a scope inside which any exceptions that occur *might* be caught by catch statements immediately following the `try`.
- The `catch` statement is a little more complicated.
- Its syntax is one of the following:
  - `catch(`*`type variableName`*`) { }`
  - `catch(…) { }`
- The first form is somewhat like a function declaration.
- You specify a variable declaration which will be instantiated by the value thrown *if and only if* that value matches (type wise) the type declared in the `catch` statement.
- Inside the scope of the `catch`, the variable declared in the catch statement is accessible as a local variable.

Somebody Catch Me!!!

- If the value thrown doesn’t match (type wise) the catch statement(s) you supply, the exception is thrown up to the next try block.
- If there are no other try blocks present, the exception is handled by the runtime environment as an *unhandled exception*.
- This usually means a generic error message and/or program termination.
- Now that we’ve spelled it all out, let’s go back to a simple example...

Example #1

A Simple Exception

More About Catching

- For every `try` statement you have, you can have multiple `catch` statements each dealing with a separate type:

```
void executeSomeFunction()
{
  throw 1.4;
}
int main()
{
  try {
    executeSomeFunction();         // Arbitrary function
  }
  catch(int x) {    cerr << “Caught INTEGER: ” << x << endl; }
  catch(string s) { cerr << “Caught STRING: ” << s << endl; }
  catch(…) { cerr << “Generic exception caught” << endl; }
}
```

More About Catching

- When deciding on which `catch()` to pass control to, the compiler does no implicit type conversion to force a match.
- Given the preceding try/catch block, the exception would be caught by the generic block (…)

Example #2

Multiple Catches

More About Throwing

- Recall this example from previous lecture

```
class MyArray {
  MyArray(int s=100);
  int &MyArray::operator[](int index);
private:
  int size;
  int *theData;
};
MyArray::MyArray(int s)
{
  size=s;
  theData=new int[size];
  for (int j=0; j<size; j++) theData[j]=0;
}
```

More About Throwing

- You may also throw user-defined types…
- You can “construct” new instances of classes right in the throw statement by calling a given type’s constructor...

```
class MyIndexError {
  MyIndexError(int i,char *msg) { badIndex=i; theMsg=msg; }
  int getBadIndex() { return badIndex; }
  string getMessage() { return theMsg; }
private:
  int badIndex;
  string theMsg;
};
int &MyArray::operator[](int index)
{
  if ((index < 0) || (index >= size))
    throw MyIndexError(index, “Index out of bounds”);
  return theData[index];
}
```

More About Throwing

- Now, I can set up to catch this exception like this:

```
int main()
{
  MyArray testArray(10);
  try {
    cout << “Element 10 is ” << testArray[10] << endl;
  }
  catch(MyIndexError e)
  {
   cerr << “Error at index ” << e.getBadIndex() << “: ”
         << e.getMessage() << endl;
  }
}
```

- `// This will yield the message:`
- `Error at index 10: Index out of bounds`

Who’s Got It?

- Actually, I could have set up one of four catch statements to catch exceptions of type `MyIndexError`.
- They are:

```
catch(MyIndexError e){}    // Copy of object thrown in e
catch(MyIndexError &e){}   // Reference of object thrown in e
catch(MyIndexError){}      // No access to object thrown
catch(…){}                 // No access to object thrown
```

- If an exception isn’t caught by any of the catch statements in a given try block, the runtime environment would look for any other try blocks further down the stack and try *their* catch statements.
- That would look something like this:

Who’s Got It?

```
void func1()
{
  try {
    func2();
  } catch(ArrayIndexError e) {
    cout << “Array Index Error: ” << e.getMsg() << endl;
  }
}
void func2()
{
  try {
    float x = divide(globalIntArray[15334],globalIntArray[1]);
  } catch(MathErr e) {
    cout << “Math Error encountered: ” << e.getMsg() << endl;
  }
}
```

- If `globalIntArray` is only 50 elements big, what happens?

Example #3

Catching More than You Expect

Even More about Throwing

- Sometimes, when catching an exception, you can only do “so much” to fix the situation.
- Consider a routine to move a robot to a series of positions. When done, you must return the robot to its original position:

```
// Some routine to read an array of Positions from the user
int getPositionSequence(Position *arrayOfPositions)
{ … }

// Call to move robot to a specific position.  If aPos is
// invalid a BadPositionException exception is thrown
void MoveRobot(Position &aPos)
{
   if (badPos(aPos))
     throw BadPositionException(aPos);
   // Continue with move logic…
}
```

Even More about Throwing

- When we execute the code which moves the robot to each successive position, we are prepared to catch a `BadPositionException`.
- When we catch it, we return the robot to its original position.
- But we have no concept of GUI here, how is the user notified?

```
// Move the robot to a succession of positions
void MoveRobot(Position *positions, int numPos)
{
  Position origPos = getCurrentPosition();
  try {
    for (int i=0; i<numPos; i++)
      MoveRobot(positions[i]);
  } catch(BadPositionException e) {
      MoveRobot(origPos);
      throw;   // What does this do?
  }
  MoveRobot(origPos);
}
```

Even More about Throwing

- throw by itself simply re-throws the current exception.
- The assumption is that someone further down the chain is ready to catch it, of course!

```
// Move a robot
void MoveTheRobot()
{
  Position *positionArray;
  int numPositions = getPositionSequence(&positionArray);
  try {
    MoveRobot(positionArray, numPositions);
  } catch(BadPositionException e) {
    cerr << “Error: attempt to move robot to bad ” <<
         << “position ” << endl << “POSITION ” <<
         << e.getBadPosition() << endl;
  }
}
```