

## Exceptions

CSCI 334  
Stephen Freund

## Fortran Control Structure

```
10 IF (X .GT. 0.0) GO TO 20
11 X = -X
   IF (X .LT. 0.0) GO TO 50
20 IF (X*Y .LT. 0.0) GO TO 30
   X = X-Y-Y
30 X = X+Y
   ...
50 CONTINUE
   X = A
   Y = B-A
   GO TO 11
   ...
```



## COM Example

```
/* Call A Remote Method on a COM object. Returns 0 on
success, 1 on failure. */
int callRemoteMethod() {
    int hResult;
    IDispatch pIDispatch;
    hResult=CLSIDFromProgID(...);
    if (FAILED(hResult)) return 1;
    hResult=CoInitialize(...);
    if (FAILED(hResult)) return 1;
    hResult=CoCreateInstance(pIDispatch,...);
    hResult=punk->QueryInterface(...);
    if (FAILED(hResult)) { pIDispatch->Release(); return 1; }
    hResult = pIDispatch->GetIDsOfNames(...);
    if (FAILED(hResult)) { pIDispatch->Release(); return 1; }
    hResult = pIDispatch->Invoke(...);
    if (FAILED(hResult)) { pIDispatch->Release(); return 1; }
    return 0;
}
```

## COM Example

```
/* Call A Remote Method on a COM object. Returns 0 on
success, 1 on failure. */
int callRemoteMethod() {
    int hResult;
    IDispatch pIDispatch;
    hResult=CLSIDFromProgID(...);
    if (FAILED(hResult)) return 1;
    hResult=CoInitialize(...);
    if (FAILED(hResult)) return 1;
    hResult=CoCreateInstance(pIDispatch,...);
    hResult=punk->QueryInterface(...);
    if (FAILED(hResult)) { pIDispatch->Release(); return 1; }
    hResult = pIDispatch->GetIDsOfNames(...);
    if (FAILED(hResult)) { pIDispatch->Release(); return 1; }
    hResult = pIDispatch->Invoke(...);
    if (FAILED(hResult)) { pIDispatch->Release(); return 1; }
    return 0;
}
```

## COM Example

```
/* Call A Remote Method on a COM object. Returns 0 on
success, 1 on failure. */
int callRemoteMethod() {
    int hResult;
    IDispatch pIDispatch;
    hResult=CLSIDFromProgID(...);
    if (FAILED(hResult)) return 1;
    hResult=CoInitialize(...);
    if (FAILED(hResult)) return 1;
    hResult=CoCreateInstance(pIDispatch,...);
    if (FAILED(hResult)) return 1;
    hResult=punk->QueryInterface(...);
    if (FAILED(hResult)) { pIDispatch->Release(); return 1; }
    hResult = pIDispatch->GetIDsOfNames(...);
    if (FAILED(hResult)) { pIDispatch->Release(); return 1; }
    hResult = pIDispatch->Invoke(...);
    if (FAILED(hResult)) { pIDispatch->Release(); return 1; }
    return 0;
}
```

## Exceptions Preview

```
void callRemoteMethod() {
    IDispatch pIDispatch;
    try {
        CLSIDFromProgID(...);
        CoInitialize(...);
        CoCreateInstance(pIDispatch);
        punk->QueryInterface(...);
        pIDispatch->GetIDsOfNames(...);
        pIDispatch->Invoke(...);
    } catch (Exception e) {
        if (pIDispatch != null) pIDispatch->Release();
        throw e;
    }
}
```

```
void CoInitialize() {
    ...
    if (bad)
        throw new Exception();
    ...
}
```

### Stack Example

```

type Stack = int list;

fun evalX (nil,a::st) = a
| evalX (PUSH(n)::rest,st) = evalX(rest, n::st)
| evalX (ADD::rest, a::b::st) = evalX(rest, (b+a)::st)
| evalX (MULT::rest, a::b::st) = evalX(rest, (b*a)::st)
| evalX (DIV::rest, a::b::st) =
  evalX(rest, (b div a)::st)
| evalX (SUB::rest, a::b::st) = evalX(rest, (b-a)::st)
| evalX (SWAP::rest, a::b::st) = evalX(rest, b::a::st)
| evalX (_,_) = 0
;

fun eval(instrs, stack) =
  print Int.toString(evalX(instrs, stack));

```

### Stack Example

```

type Stack = int list;

exception DivideByZero;
exception BadOp;

fun evalX (nil,a::st) = a
| evalX (PUSH(n)::rest,st) = evalX (rest, n::st)
| evalX (ADD::rest, a::b::st) = evalX (rest, (b+a)::st)
| evalX (DIV::rest, 0::b::st) = raise DivideByZero
| evalX (DIV::rest, a::b::st) =
  evalX (rest, (b div a)::st)
...
| evalX (_,_) = raise BadOp;

```

### Stack Example

```

fun eval(instrs, stack) =
  (
    print Int.toString(evalX(instrs, stack))
  ) handle BadOp => print "Bad Operation"
    | DivideByZero => print "Div by 0";

- eval([PUSH(3),PUSH(0),DIV],nil);
Div by 0

```

### Stack Example (2)

```

type Stack = int list;

exception EvalError of string;

fun evalX (nil,a::st) = a
| evalX (PUSH(n)::rest,st) = evalX (rest, n::st)
| evalX (ADD::rest, a::b::st) = evalX (rest, (b+a)::st)
| evalX (DIV::rest, 0::b::st) =
  raise EvalError("Div by 0")
| evalX (DIV::rest, a::b::st) =
  evalX (rest, (b div a)::st)
...
| evalX (_,_) =
  raise EvalError("Bad op");

```

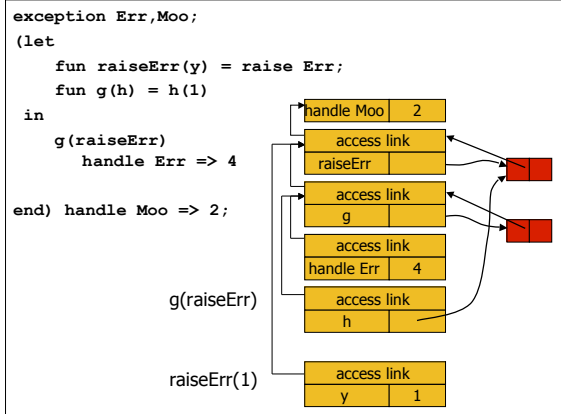
### Stack Example (2)

```

fun eval(instrs, stack) =
  (
    print Int.toString(evalX(instrs,stack))
  ) handle EvalError(msg) => print msg;

- eval([PUSH(3),PUSH(0),DIV], nil);
Div by 0

```



## Resource Management (Memory)

```
try {
    ...
    // creates lots o' memory
    ...
} catch (Exception e) {
    ...
}
```

- GC works well with exceptions to clean up mem.
- C++ or no GC:
  - much harder
  - where do you free mem?

## Resource Management (Other...)

- Files, sockets, DB connections
- Limited number available
  - must be released when done
- Without Exceptions:

```
File f;
...
f = new InputFile("moo.txt");
...
f.read();
...
f.close();
```

## Resource Management (Other...)

- Files, sockets, DB connections
- Limited number available
  - must be released when done

- With Exceptions:

```
File f =
    new InputFile("moo.txt");

try {
    ...
    f.read();
} catch (SocketException e) {
}
}
```

When is f closed?  
• on exit from try-block  
• on exit from handler  
• on exit caused by  
uncaught exception

## Finalizers

- GC calls finalize on objects right before collection:

```
class InputFile {
    ...
    void finalize() {
        this.close();
    }
}
```

- Problems?

## Try-Finally Blocks

```
InputFile f = new InputFile();

try {
    ...
} catch (Exception e) {
    ...
} finally {
    f.close();
}
```

- finally code runs:
  - when control leaves try part (normally or exceptionally)
  - when control leaves exception handler