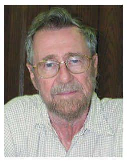
Exceptions

CSCI 334

Stephen Freund

Fortran Control Structure

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



Block Structured Programming

```
if (x < 0) {
  x = -x;
  while (y > 0) {
   y = b - a;
   x = x / z;
    if (x > 10) {
      x = x / 2;
    } else {
      x = x * 2;
    }
  }
}
```

COM Example

```
/* Call A Remote Method on a COM object.  Returns 0 on
  success, 1 on failure. */
int callRemoteMethod() {

  IDispatch pIDispatch;
  CLSIDFromProgID(...);

  CoInitialize(...);

  CoCreateInstance(pIDispatch,...);
  punk->QueryInterface(...);

  pIDispatch->GetIDsOfNames(...);

  pIDispatch->Invoke(...);

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

```
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;
  }
}
```

Stack Example

```
type Stack = int list;

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

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

Stack Example

```
type Stack = int list;

exception DivideByZero;
exception BadOp;

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

Stack Example

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

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

Stack Example (2)

```
type Stack = int list;

exception EvalError of string;

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

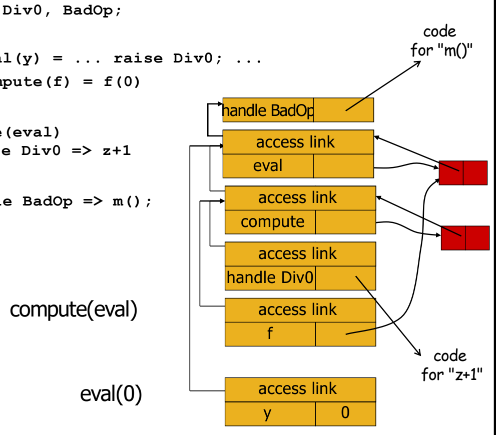
Stack Example (2)

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

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

```
exception Div0, BadOp;
(let
   fun eval(y) = ... raise Div0; ...
   fun compute(f) = f(0)
 in
   compute(eval)
     handle Div0 => z+1

end) handle BadOp => m();
```



Checked Exceptions in Java

```
class IOException extends Exception { ... }

class FileReader {
    public FileReader(String name) ... {
          if (error happens) throw new IOException();
    }
  }

try {
    FileReader in = new FileReader("input.txt");
    ..
  } catch (IOException e) {
    ...
  } catch (NetworkFailureException e) {
    ...
  }
```

Checked Exceptions in Java

```
class IOException extends Exception { ... }

class FileReader {
    public FileReader(String name) throws IOException {
          if (error happens) throw new IOException();
    }
  }

try {
    FileReader in = new FileReader("input.txt");
    ..
  } catch (IOException e) {
    ...
  } catch (NetworkFailureException e) {
    ...
  }
```

Declared Exceptions

```
public FileReader(String name) throws IOException {
  ... if (error happens) throw IOException(); ...
}

void m0() {  new FileReader(...); } ← BAD

void m1() {
  try {
    ... new FileReader(...);
  } catch (IOException e) { ...} ← function handles exn.
}

void m2() throws IOException { ← caller must handle exn.
  ... new FileReader(...);
}

void m3() { m2(); }     ← BAD
```

Resource Management (Memory)

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

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

Resource Management (Other...)

- Files, sockets, DB connections
- Limited number available
  - acquire resource when object created
  - must be released when done with object

```
class File {
  private OSFileHandle osHandle;

  void close() { release(osHandle); }
}
```

Resource Management (Other...)

- Files, sockets, DB connections
- Limited number available
  - acquire resource when object created
  - must be released when done with object

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

Resource Management (Other...)

- Files, sockets, DB connections
- Limited number available
  - acquire resource when object created
  - must be released when done with object

```
File f =
   new File(...);

try {
  ...
  f.read();
} catch (FileException 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 File {
  private OSFileHandle osHandle;

  void close() { release(osHandle); }

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

- Problems?

Try-Finally Blocks

```
File f = new File(...);

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

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

Try-With-Resources

Implements:

```
interface AutoCloseable {
  void close();
}
```

```
try (
  FileOutputStream out =
     new FileOutputStream("out");
  FileInputStream  in1 =
     new FileInputStream("in1");
) {
  // Do something with those 2 streams
} catch (Exception e) {
  // as usual
}
```

Python "with"

- Same idiom
  - open resource, use it, then close it automatically

```
with open("welcome.txt") as file:
  data = file.read()
# do something with data
```

"defer" in Swift...

```
let fileName = "file.txt"
if let file = FileHandle(fileName) {
  defer { file.closeFile() }
  ...
  let data = fetchData()
  file.write(data)
}
```

and Go...

```
file, error := os.Open("/etc/passwd")
if err == nil {
  defer file.Close()
  ...
```