Lab Portion

TA: Patrick S. Li

patrickli.2001@gmail.com

Interests:

- Programming language design
- Type theory
- Compiler implementation
- Stanza programming language. (See www.lbstanza.org)

Lab Portion

GOAL: Develop a high-performance implementation of the Feeny programming language.

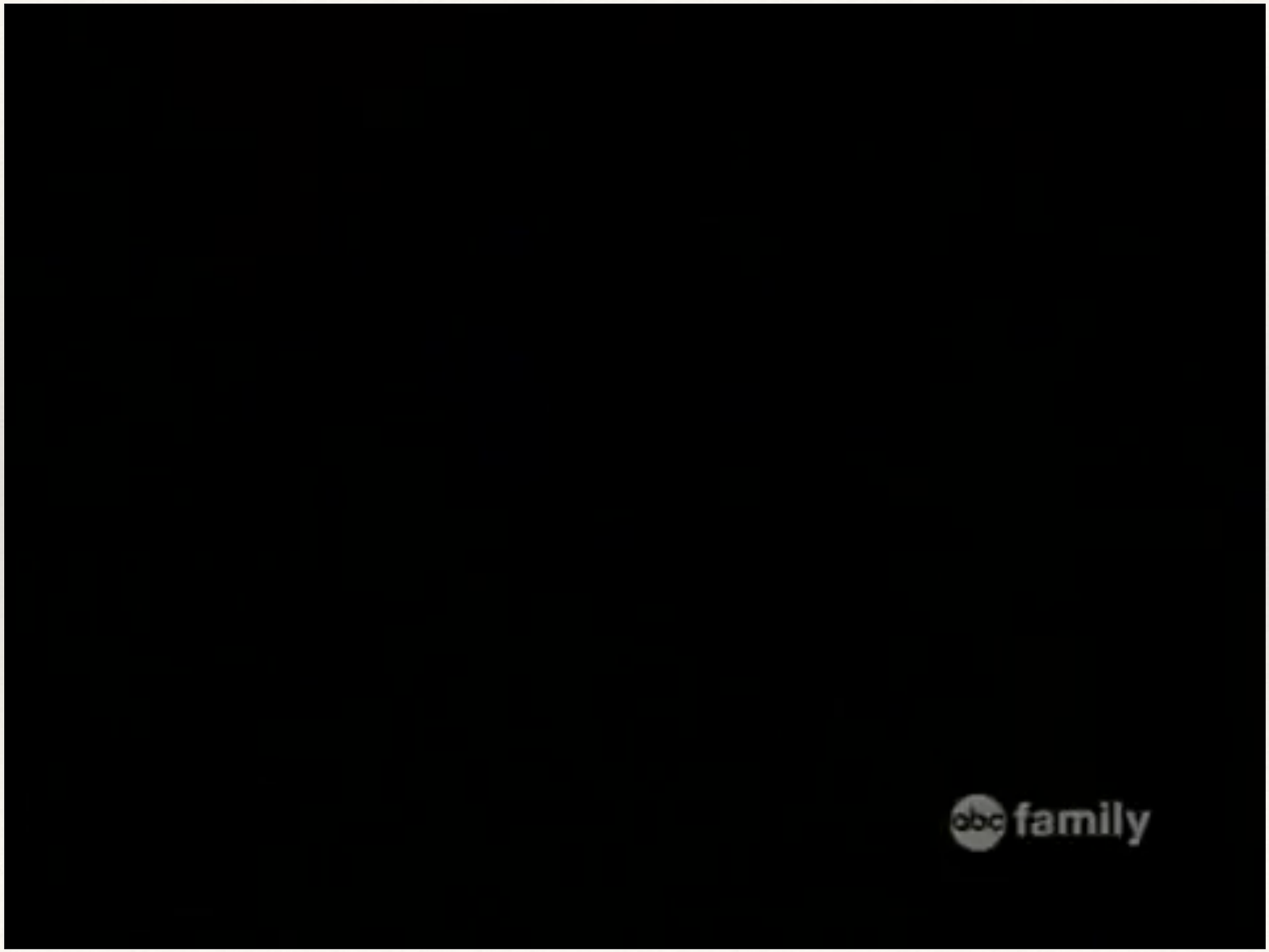
Resources:

- Piazza: Assignments, Harnesses posted here. Questions answered. Etc.
- Weekly Discussion Section

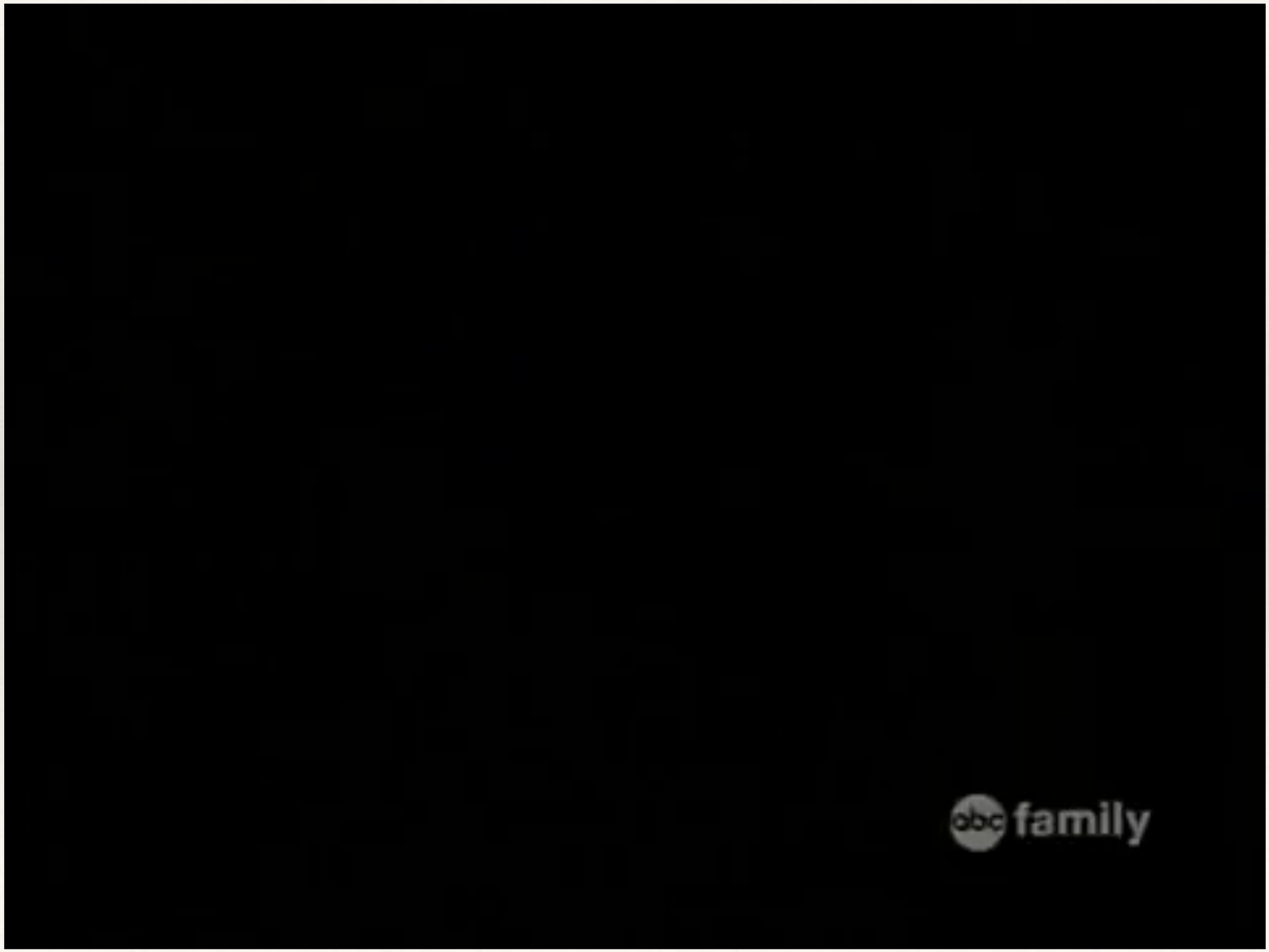
Feeny Programing Language

- Designed to have same features as modern popular scripting languages.
- Easy to Implement. Hard to Optimize.
- Imperative
- Dynamically Typed (ala. Python, Ruby, Lua, Lisp, Smalltalk)
- Prototype Object System (ala. Javascript, Self)
- Garbage Collected
- Restricted in Scope: Objects, Arrays, Integers

How to Pronounce Feeny



How to Pronounce Feeny



Lab Structure

Assignments:

1. Introduction to the Feeny Language
2. Abstract Syntax Tree Interpreter
3. Bytecode Interpreter
4. Bytecode Compiler
5. Garbage Collector
6. Dynamic Compiler
7. Speculation/Cache Optimizations
8. Truffle Framework Implementation

Feeny Example Programs

- Binary Search: Binary search inside sorted array for index of a given number.
- Fibonacci: Compute first 100 fibonacci numbers.
- Complex Numbers: A library for handling complex numbers.
- Inheritance: Demonstrates object inheritance using prototype object system.
- Lists: A library for lists.
- Vectors: A library for automatically growing arrays.
- Sudoku: A program for solving Sudoku puzzles.

Feeny Lexical Structure

```
defn bshelper (xs, i, n, v) :
    if n == 1 :
        if xs[i] == v : i
        else : -1
    else :
        var n1 = n / 2
        var a = xs[i + n1 - 1]
        if a < v : bshelper(xs, i + n1, n - n1, v)
        else : bshelper(xs, i, n1, v)
```

Indent

Feeny Lexical Structure

Dynamically Typed

```
defn bshelper (xs, i, n, v) :
    if n == 1 :
        if xs[i] == v : i
        else : -1
    else :
        var n1 = n / 2
        var a = xs[i + n1 - 1]
        if a < v : bshelper(xs, i + n1, n - n1, v)
        else : bshelper(xs, i, n1, v)
```

Feeny Lexical Structure

Dynamically Typed

Indentation Structuring

```
defn bshelper (xs, i, n, v) :
    if n == 1 :
        if xs[i] == v : i
        else : -1
    else :
        var n1 = n / 2
        var a = xs[i + n1 - 1]
        if a < v : bshelper(xs, i + n1, n - n1, v)
        else : bshelper(xs, i, n1, v)
```

Feeny Lexical Structure

Dynamically Typed

```
defn bshelper (xs, i, n, v) :
    if n == 1 :
        if xs[i] == v : i
        else : -1
    else :
        var n1 = n / 2
        var a = xs[i + n1 - 1]
        if a < v : bshelper(xs, i + n1, n - n1, v)
        else : bshelper(xs, i, n1, v)
```

Indentation Structuring

Commas are Whitespace

Feeny Lexical Structure

Dynamically Typed

```
defn bshelper (xs, i, n, v) :
    if n == 1 :
        if xs[i] == v : i
        else : -1
    else :
        var n1 = n / 2
        var a = xs[i + n1 - 1]
        if a < v : bshelper(xs, i + n1, n - n1, v)
        else : bshelper(xs, i, n1, v)
```

Indentation Structuring

Commas are Whitespace

Whitespace Delimited

Null, Integers, and Printing

```
null
42
printf("Hello World\n")
printf("~ + ~ = ~", 2, 3, 5)
```

Arithmetic

```
10 + 23          (10).add(23)
10 - 23          (10).sub(23)
10 * 23          (10).mul(23)
10 / 23          (10).div(23)
10 % 23          (10).mod(23)
```

Comparisons

```
10 < 23                    (10).lt(23)
10 <= 23                   (10).le(23)
10 > 23                    (10).gt(23)
10 >= 23                   (10).ge(23)
10 == 23                   (10).eq(23)
```

Variables

```
var x = 23
printf(“x = ~”, x)
x = 10
printf(“x = ~”, x)
```

If Expressions

all numbers are true, while No object means false.

```
if 2 < 3 :
    printf("Smaller")
else :
    printf("Bigger")
```

If Expressions

```
if 2 < 3 :
   printf(“Smaller”)
```

```
if 2 < 3 :
   printf(“Smaller”)
else :
   null
```

While Expressions

```
var i = 0
while i < 10 :
    printf(“i = ~”, i)
    i = i + 1
```

While Expressions

```
var i = 0
while i < 10 :
    printf("i = ~", i)
    var j = i + 1
    i = j
```

scope.

Arrays

```
var a = array(10, 42)

a[0] = 1

a[1] = 5

printf(“First element: ~”, a[0])

printf(“Second element: ~”, a[1])

printf(“Length: ~”, a.length())
```

```
var a = array(10, 42)

a.set(0, 1)

a.set(1, 5)

printf(“First element: ~”, a.get(0))

printf(“Second element: ~”, a.get(1))

printf(“Length: ~”, a.length())
```

Functions

```
defn double (x) :
   var y = x + x

   printf(“x = ~\n“, x)

   printf(“y = ~\n”, y)

   y


var twenty = double(10)

printf(“twenty = ~”, twenty)
```

Null: The Empty Object

```
var x = null
x.mymethod(10) ;ERROR
x.myslot ;ERROR
```

Objects with Slots

```
var p = object(null) :
    var x = 10
    var y = 20
    printf(“p = (~, ~)”, p.x, p.y)


p.x = 30
p.y = 40
printf(“p = (~, ~)”, p.x, p.y)
```

Objects with Slots

```
var p = object :
    var x = 10
    var y = 20
    printf(“p = (~, ~)”, p.x, p.y)


p.x = 30
p.y = 40
printf(“p = (~, ~)”, p.x, p.y)
```

Objects with Methods

```
var p = object :
    var x = 10
    var y = 20
    method print () :
        printf("(~, ~)", this.x, this.y)
```

receiver object

```
p.print()
```

Objects: Method Scope

```
defn pair (px, py)
  object :
    var x = px
    var y = py
    method print () :
      printf("(~, ~)", this.x, this.y)
```

single expression

```
pair(20,30).print()
```

Objects: Inheritance

```
defn pair2 (px, py) :
    object(pair(px, py)) :
        method print-twice () :
            this.print()
            this.print()

val p = pair2(1, 3)
p.print-twice()
p.print()
```

Objects: Late Binding

```
defn maxer () :
    object :
        method max (x, y) :
            if this.lt(x, y) : y
            else : x
        method lt (x, y) :
            x < y

val m = maxer()
m.max(30, 40)
```

Objects: Late Binding

```
defn pair-maxer () :
    object(maxer()) :
        method lt (a, b) :
            if a.x < b.x :
                a.y < b.y


val m = pair-maxer()
m.max(pair(10,30), pair(2, 5))
```

Assignment 1: Programming in Feeny

Three Exercises

1. Implementation of Towers of Hanoi.
2. Implementation of a simple Stack library.
3. Implementation of a better Towers of Hanoi.