Little Languages

Lecture 35:

Final Thoughts

We will be on the VM today. Go ahead and pull.
cd into lecture 35 / jsx
Then run:

```
$ npm install
```

Final Exam

- Thursday, May 2nd, at 12pm

- Final Exam score can replace a single lower midterm score
  - Will happen automatically if it improves your outcome

- Combines concepts from MT1 and MT2
  - Unlike [Beauty in Squares] – we'll use a modern, real world grammar
  - I'll distribute the grammars Monday but we'll preview them today

Modern ECMAScript

- A case study on the final will revolve around RegExps and JSX in ECMAScript.

- Grammar Notation:
  - https://www.ecma-international.org/ecma-262/8.0/index.html#sec-grammar-notation

- Primary Expressions:
  - https://www.ecma-international.org/ecma-262/8.0/index.html#sec-primary-expression

- RegExps:
  - RegExp Literal: https://www.ecma-international.org/ecma-262/8.0/index.html#sec-literals-regular-expression-literals
  - RegExp Grammar: https://www.ecma-international.org/ecma-262/8.0/index.html#sec-regexp-regular-expression-objects

- Facebook React's JSX Syntax Extension:
  - https://facebook.github.io/jsx/

JavaScript Regular Expressions

- Regular Expression Literal
- Pattern
- Disjunction
- Alternative
- Term
- Assertion (^$)
- Quantifier / Quantifier Prefix

- Play around in your browser with console (Ctrl+Shift+J) or in the VM:

- $ cd 590-material/lecture/35-final-thoughts/regexp
- $ node regexps.js

Babel: A JavaScript Transpiler

- Website: https://babeljs.io/
- Used by many projects to enable modern JavaScript features by translating them into equivalent code expressed in terms of older versions of the language.
- Plugin architecture allows experimental language features to be developed without rewriting a complete JavaScript tokenizer/parser/code generator/etc
- FaceBook's React Project introduced a popular extension to the JavaScript language (JSX) that can be transpiled using babel.

Facebook React JSX

- JSX is an extension to the ECMAScript language
  - Formal grammar linked to in previous slide
- Introduces HTML-like syntax literals to the language
- Transpiles from .jsx files to .js

Demo Directory:

```
$ cd 590-material/lecture/35-final-thoughts/jsx
$ npm install
```

Demo Compiling by reading a file and writing out a file:

```
$ ./node_modules/.bin/babel --out-file example.js example.jsx
```

Demo Compiling with standard input:

```
$ cat example.jsx | ./node_modules/.bin/babel --filename example.jsx
```

Language Concepts

- Regular Expressions
  - Relationship between Regular Expressions and Automata
- Grammars
  - Terminals vs. Non-terminals
  - Derivation of Parse Tree
- Lexemes and Tokenization
- Parsing and Abstract Syntax Tree Representation

Lower Level Language Programming Concepts

- `null`
- Stack values versus heap values
- Lifetime of values
- Addresses, Pointers, and Dereferencing
- Smart pointers in Rust vs. raw pointers in C
  - https://doc.rust-lang.org/book/ch15-00-smart-pointers.html

Programming Language Concepts

- Pattern Matching Statements
  - if-let
  - match
- Algebraic Data Types
  - Rust's Enum
- Operator Overloading

System Operations

- Process Model
  - Process vs. Program
  - Arguments
  - Standard Input/Output/Error
  - Pipes
  - Output Redirection
  - Environment Variables

- Tools of the Trade
  - Shell: Bash
    - $HOME, $PATH, $?
  - Version Control: git
  - Text Searching/Filtering: grep
  - Text Stream Editing: sed
  - Command builder: xargs

The UNIX Philosophy '78 per Doug McIlroy

- **"Make each program do one thing well."**
  - Make each *function* and *class* do one thing well, as well.
- **"Expect the output of every program to become the input to another, as yet unknown, program. Don't clutter output with extraneous information.** [..] Don't insist on interactive input."
- **"Design and build software**, [..], **to be tried early** [..]. **Don't hesitate to throw away the clumsy parts and rebuild them."**
- **Use tools** in preference to unskilled help to lighten a programming task, **even if you have to detour to build the tools and expect to throw some of them out after you've finished using them.**

"Even though the UNIX system introduces a number of innovative programs and techniques, **no single program or idea makes it work well**.

Instead, **what makes it effective is the approach to programming, a philosophy of using the computer**.

Although that philosophy can't be written down in a single sentence, at its heart is the idea that **the power of a system comes more from the relationships among programs than from the programs themselves**.

Many UNIX programs do quite trivial things in isolation, but, combined with other programs, become general and useful tools."

Kernighan and Pike '84

Thank you!