Week 1 - Tuesday (2019-04-04)

Donald E. Knuth "The Art of Computer Programming"

TEX Was written in Pascal

Literate Programming

- have comments that are written in LaTeX, allowing the code to process the source code as both a piece of software and also as documentation
- main idea is to maintain documentation of source code inside of the source code so you don't have to update two things at once
- example: Java, source code for the Java library is maintained in the documentation in the same file

Quiz Solution:

```
tr -c -s 'A-Za-z' '[ln*]' | sort | uniq -c | sort -rn
```

- tr -c means complement, so translate everything that's not a letter character
- tr -s means squeeze repeated outputs (generates a single newline between each word)
- uniq -c means count the duplicates
- sort -rn means sort by count (reverse and numeric sort)

Main Concepts of the Course

1. Principles and Limitations of programming models
2. Core Programming Paradigms
3. Learn about the various tradoffs for different types of programming languages

Theory	Practice
Language Design	Ocaml
Syntax	Java
Semantics	Prolog
Functions	Scheme
Names	Python
Types	? (Kotlin)
Control	
Objects	
Exceptions	

BASIC Developed on the GE 225 Mainframe (1961)

- 40 micro-seconds to add integers
- 500 micro-seconds to divide integers
- ~40 KB of RAM
- Timeshared with 20 other users

C (C++) Developed on PDR11 (~1975)

- 4 micro-seconds to add
- 16 KB RAM
- 1.2 micro-seconds memory cycle time. (3x faster than adding)
  - This is the reason that C coding revolves around pointer arithmetic.
  - **Note:** that the assumption that following pointers is faster now is wrong

Stable Languages Die, Successfull Languages Evolve!

1. Syntax Evolution: Idea: Can we change the language and add new features without changing the compiler?
   - Example of how to change C language
     ```
     #define ELTS(a) (sizeof(a)/sizeof(a)[0])
     #define CALLMAN(f, args) (f)(ELTS(args), args)
     /* These are now equivalent:
      * Foo(f, args); <=> CALLMAN(f, args);
      */
     #define CALLN(f, ...) CALLMAN(f, ((obj[]) { __VA_ARGS__ }))
     /* This has now changed the language of C through the use of macros,
      * This is called metaprogramming
      */
     ```
   - Steve Bourne created the shell, and modified the C code using this:
     ```
     #define IF if(
     #define THEN ) {
     #define ELSE } else {
     #define FI }
     ```

CS 131 Reading 1 - 4/02/2019

Chapter 1 - Programming Languages

1. There is a very large variety in programming languages
   - Imperative Languages:
     - C for example is an imperative language
       ```
       int fact (int n) {
           int sofar = 1;
           while (n > 0) sofar *= n-- return sofar;
           return sofar;
       }
       ```
     - Imperative languages have two main features: assignment and iteration
   - Functional Languages:
     - Funcional languages are based on two different hallmarks: recursion and single-valued variables
     - Here is an example in ml:
       ```
       fun fact x -
           if x <= a then 1 else x * fact (x ~ 1) ;
       ```
     - Another example in lisp:
       ```
       (defun fact (x)
           (if (<= x 0) 1 (* x (fact (- x 1))))
       ```
   - Logic Programming Languages
     - Perhaps less natural then in functional programming languages, here is fibonnacii with prolog
       ```
       fact(X, 1) :-
           X := 1,
           !.
       fact(X, Fact) :-
           X > 1,
           NewX is X -
           fact(NewX, NF),
           Fact is X * NF.
       ```
     - The first three lines state that the factorial of X is 1 if X = 1
     - The Next five lines state: > "To prove that the factorial of X is Fact, you must do the following things; prove that X is greater than one, prove that NewX is one less than X, prove that the factorial of NewX is MF, and prove that Fact is X times NF."
     - Expressing a program in terms of rulesabout logical inference is the hallmark of logic programming, which is perhaps not the greatest for mathematical expressions
   - Object-Oriented Languages
     - An example is Java, which is object oriented, which means that in addition to being imperative, it also makes it easier to solve programming problems using objects.
       ```
       public class MyInt {
           private int value;
           public MyInt(int value) {
               this.value = value;
           }
           public int getValue() {
               return this.value;
           }
           public MyInt getFact() {
               return new MyInt(fact(this.value));
           }
           private int fact(int n) {
               int sofar = 1;
       ```

```
            while (n > 1) sofar *= n--;
            return sofar;
        }
    }
```

2. Evolution Of Programming Languages
   - Programming Languages have evolved over time, through the use of dialects
     - FORTRAN for example has FORTRAN 77 and FOTRAN 2008 which are different dialects
   - Programming Languages evolve slowly with the help of different industries

Chapter 2 - Defining Program Syntax

Syntax: The syntax of a programming language is the part of the language definition that says how programs look: their form and structure.

Semantics The semantics of a programming language is the part of the language definition that says what programs do: their behavior and meaning.

- Grammar of modern programming languages has evolved to generally contain certain key elements
  - Example is word granularity, i.e. we do not have tokens be characters, we instead seperate different syntactical structures with spaces and typically newlines
  - Some languages also have statement terminators (Ex: C has ';')
  - All languages support comments in the middle of the line
  - Example of how grammars can be formally structured:
    1. <subexp> ::= a | b | c | <subexp> - <subexp>
    2. <subexp> ::= <var> - <subexp> | <var>

       <var> ::= a | b | a
    3. <subexp> ::= <subexp> - <var> | <var>

       <var> ::= a | b | c
- Another way to represent language meta syntax is through syntax diagrams

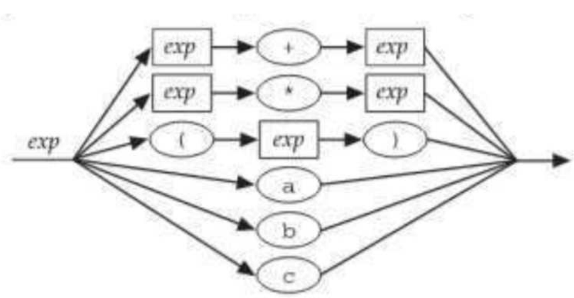

Figure 1: Syntax Diagram with 6 different productions

Chapter 3 - Where Syntax Meets Semantics

1. Operators
   - Almost all modern programming languages also include special syntax for frequently used simple operations like addition, subtraction, multiplication, and division.
   - The inputs to an operator are called its operands

    - Unary Operators take one operand, while binary operators take 2
  - Most modern programming languages use infix operations, so for an arbitrary operator *, operands a and b would be operated like so: a * b
    - Example of a non-infix language is lisp, which has prefix operations
2. Precedence
  - Different Linguages have different ideas about how to ocganize operators into precedence levels.
    - C for example has 15 levels, while Pascal has
  - The lack of proper precedence in a language can lead to very unintuitive responses compared to what humans would expect
  - C with its very complex precedence system tries to mimic human level intuition, but it is still very beneficial to add parentheses
3. Abstract Syntax Trees
  - A grammar for a realistically large grammar will have multiple non-terminal symbols.
  - Language systems usually store an abbreviated version ot a parse tree called the abstract syntax tree or AST
  - Many language systems use an AST as an internal representation of a program,

Chapter 5 - A First Look at ML

1. h

Week 1 - Thursday 2019-04-04

Language Design Choices

1. **Orthogonality**
   - The choice of X does not affect Y
2. **Efficiency**
   - How fast is it?
   - How much RAM does it use?
   - Power/Energy
   - Network throughput/bandwidth/latency
3. **Simplicity**
   - Easy to learn
   - Easy to use with intuitive syntax
   - Good and simple documentation
4. **Convenience**
   - Simple to use
   - Potentially having lots of users is also a convenience
5. **Safety**
   - Static vs Dynamic
     - Static type checking is typically safer to use
   - Type checking
6. **Abstraction**
   - Can your programs scale cleanly?
   - Use of classes is an attempt at abstraction
7. **Exceptions**
   - Proper exception handling
   - Proper indication of errors
8. **Concurrency**
   - Must have proper support for parallel programmng
   - Cannot just be a third party library that adds this functionality
9. **Mutability/Extensibility**
   - Must have support for change in the future
   - Example of C with its macros that allow changes to the language without changing the compiler.

Types of Languages

Imperative	Functional	Logic
C++, C, Java	Lisp, ML, F#	Prolog

1. Imperative Languages are composed of statements
   - Statements are listed in order: `S_1; S_2; S_3;`
2. Functional Languages are composed of functions
   - Functions are linked like:
   $$y = F_1(F_2(x), F_3(x)) \tag{1}$$
   - Functional languages lack I/O
3. Logic Languages are composed of predicates
   - Predicates are linked together through the use of predicates: `(P_1 & P_2) | P_3`

Functional Programming

1. Clarity: use notations from mathematics
2. Parallelizability: performance is good, code structure incentivizes you to write parallel code.

- Function: mapping from a domain to a range
- Functional Form: function where either domain or the range is a function type
  - Example of higher order function in functional form:

$$\int_{1}^{100} f(x)dx \tag{2}$$

  - Example in C++:
    ```
    y = integral(f, 1, 100);
    ```

Ocaml Introduction

1. Static type checking is supported
   - Similar to Java in that all objects have a type
2. Type Inference
   - type checking is done at compile time, types for variables do not have to be declared
3. Automatic Storage Management
   - Garbage Collection is assumed inside of Ocaml system
4. Good Support for Higher-Order Functions
   - Really easy to implement these features (This led to creation of lambda expressions