CS131 Week 1

Week 1 Lecture 1

==========

Introduction and Some Programming Language History

D.E. Knuth : The Art of Computer Programming

- There's an art to computer science

Knuth invented **TEX**

- **batch word processor** that is cleaner layout than MS Word
- written in Pascal

Now Knuth has 3 files

- taocp.tex
- tex.pas
- tex.tex

PROBLEM:

Updates to source code in TEX source code tex.pas has to be updated in his documentation book tex.tex

2 steps

How to eliminate this problem?

- SOLUTION: he makes a layout algo where TEX book and source code are in one file. Changes in source code highlight places in documentation where you have to modify descriptions

Knuth invented an idea called **Literate Programming**

CACM: M.D. McIlroy: Bell Labs guy

- problem: take ASCII text file and output
  - one word per line
  - each preceded by # of occurrences
  - sorted by popularity
- Knuth solved it using invented data structure **hash trie**, which was used in Literate Programming solution in Pascal
- McIlroy responded by saying easier to do using **Unix pipes**
  - tr -cs A-Za-z '[\n]' | sort | uniq -c | sort -nr

TAKEAWAY: some programming languages are better suited for certain problems

Which runs faster?

- Knuth's program has efficiency advantage (designed his own low level hash trie)

- McIlroy's program is quick to implement, doesn't really need documentation, solving related problem is fairly simple

Sapir-Whorf Hypothesis

1. The language we use, to some extent, determines/influences the way we view the world, and how we think
2. The structural diversity of languages is essentially limitless

Sapir-Whorf is controversial for natural language, but fairly true for Programming Languages

QUESTION: Is the structural diversity of programming languages essentially limitless?

Choosing which Language to use

Minimize cost of a project by selecting "best" language
costs:

- reliability
- training
- maintenance
- program development
- (runtime performance)

  Talked about language design issues at end but continued next lecture, so moved to next lecture notes

--- 1/9/18

Week 1 Lecture 2

==========

Topics Covered

Language design issues:

- Orthogonality
- Efficiency
- Simplicity
- Concurrence
- Safety (compile time vs runtime vs no chating?)
- Abstraction

Language Design Issues

1. Orthogonality:

- has functions returning types of values
  - choose type T
  - define function F that returns T
  - t f(int x)
- Orthogonality good: don't want exceptions
  - i.e. in C, can't return arrays because designers wanted return values to fit in register 0 %rax

2. Efficiency (runtime):
   - time (realtime vs. CPU time)
   - memory (RAM vs cache)
   - network (latency vs throughput)
   - power (energy I/O)
3. Simplicity:
   1. Limit exceptions (errors/failures/unusual)
      1. e.g. this function has arguments after header, other function has arguments before header = exception
      2.
4. Concurrency
   1. Achieve level of parallelism for better performance
5. Mutability
   1. Programs can be changed easily
   2. Language evolves
      1. e.g. C: ~1975 was very slow, now it's much faster/mutated to keep up with faster hardware
   3. Example of language that was successful but failed due to non-mutation
      1. Simula 67: first OOP language, failed only because creators thought it was perfect, didn't add updates and Java beat it out
   4. **Preprocessing** using **macros** is a way to mutate a language (without the language standard updating)
      1. e.g. calling a function and passing it arguments in Emacs
      2. #define CALLN(...)
      3. now people can use CALLN() function instead of using Obj args[7]; args[0] = a; args[6] = g; foo(7, args);

Syntax - rules + parser
- form independent of meaning
- e.g. "Colorless green ideas sleep furiously" -Noam Chomsky
- "Time flies" (abiguous: time is adj or noun? flies is verb or noun?)
- "Ireland has leprechauns galore" (why does adj come after noun here, but not elsewhere?)

Syntax issues:
- Simplicity (parser easy to write, easy to understand)

- Syntax is what people already know (**inertia**)
- Readability
- Writeability (e.g. APL [A Programming Language])
- Redundancy (*increases reliability* of code [if there's typos, redundancy catches])
- Ambiguity
  - a - b - c
  - a + b + c (is ambiguous because though math says + is associative, in C you can get overflow adding 2 numbers before the other, rounding error, etc)

Tradeoffs

Simplicity vs Readability
- postfix is simpler but less readable
- e.g. (a + b)/c easier to read, ab+c/ has less symbols (simpler)

Writeability vs Readability
- Eggert's Life program in APL very small to write, hard to read
- his conclusion: scripting languages trade writeability gain for readability loss

Components of Language
- Tokens (basic building block)
  - smallest piece of language that has meaning

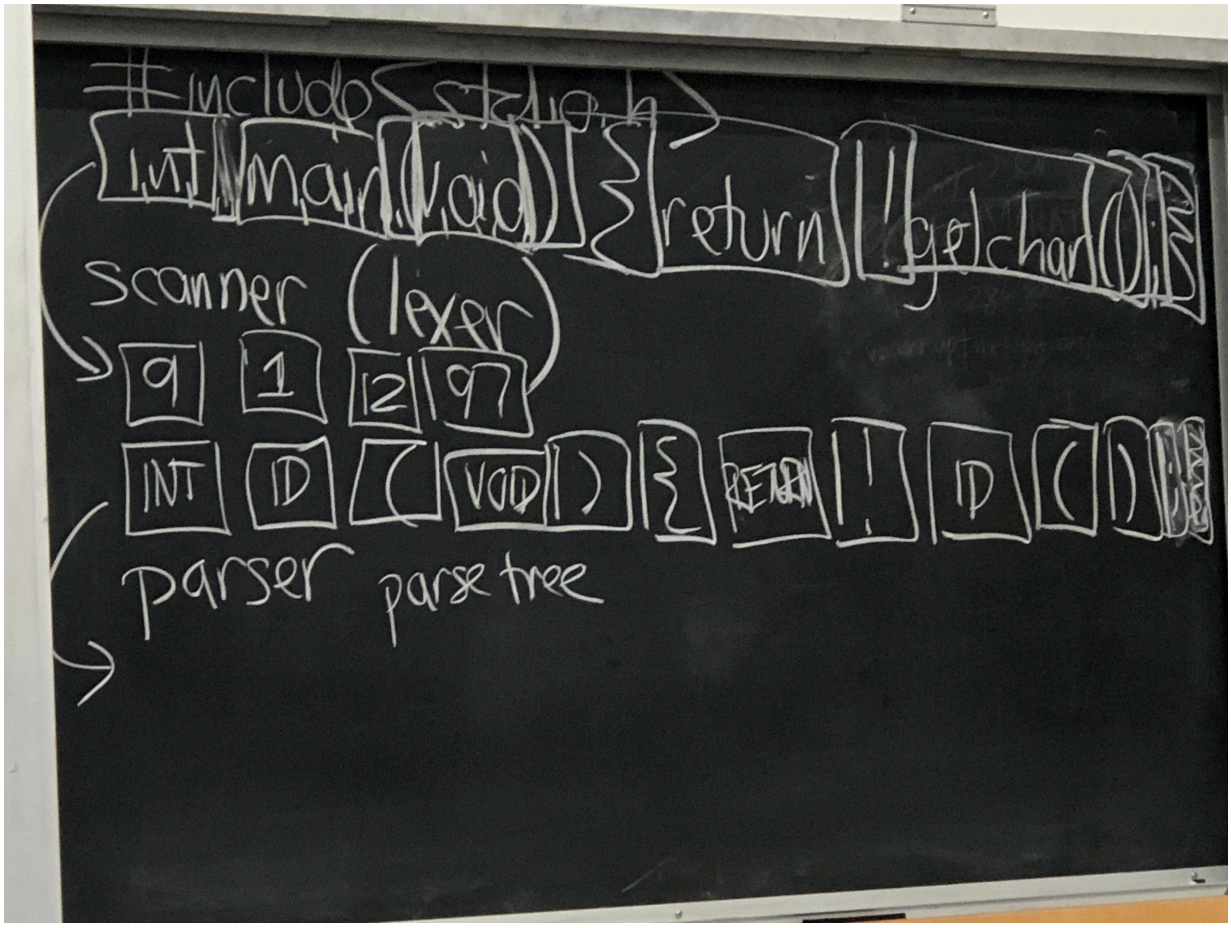

Fig. 1: shows tokens of a language (which scanner will identifier from greedy reading of code / syntax)

- Operators
  - + - * / — etc.
  - tokenization is greedy (left-to-right)
  - operates on character to character basis
  - e.g. *a - - - *b = *a-- -- -*b = ((*a—)—)-(*b))
- Identifiers
  - int **iffy**; (iffy is identifier)
- Keywords
  - reserved words (violate mutability principle)
  - non-reserved keywords (PL/I programming language)

NOTE: Components of language make it difficult to mutate language (the operators, keywords make it difficult to change language)

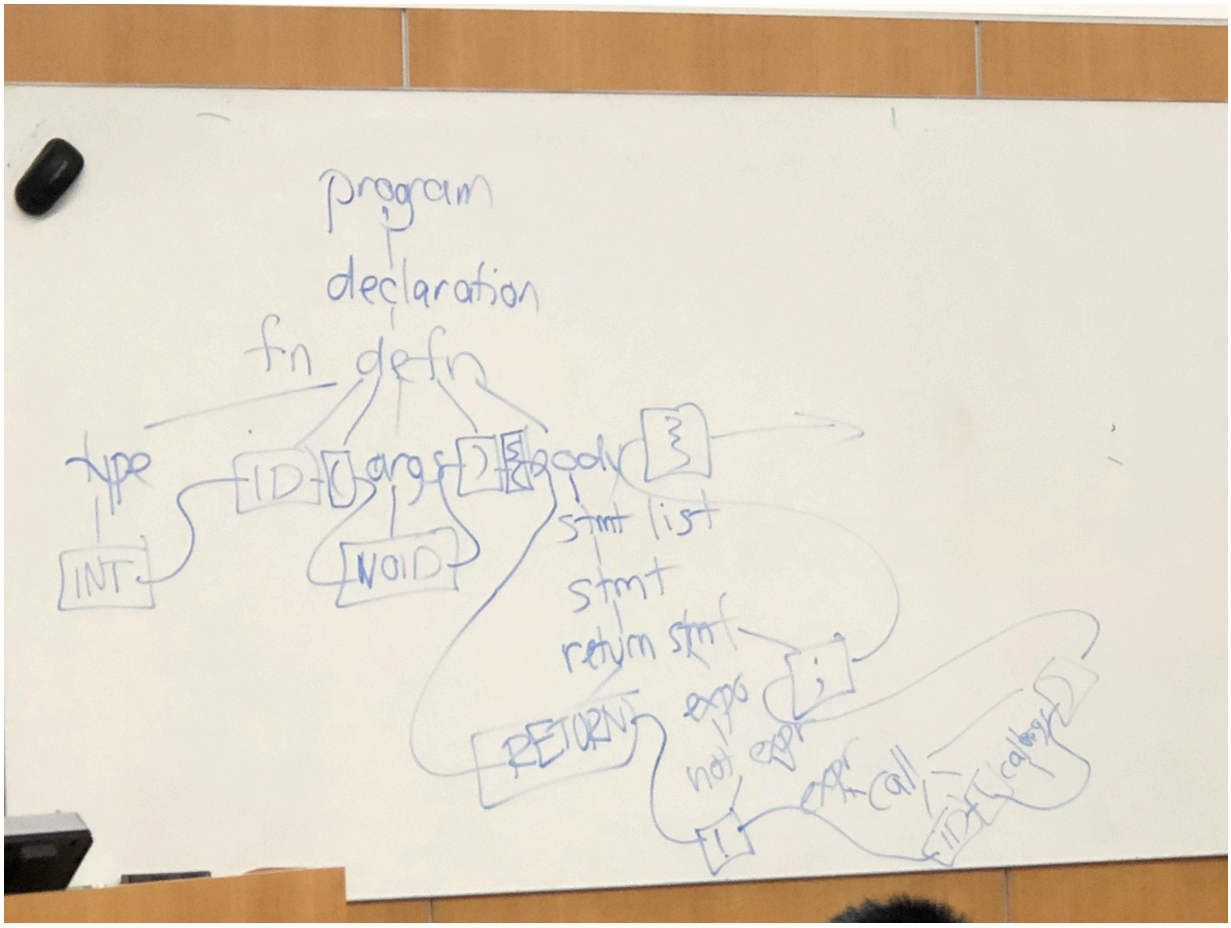

Fig 2: parse tree (parser uses tokens to get meaning of program / semantics)

Context-free grammar

- start symbol
  - finite set of **nonterminal symbols**
  - finite set of **terminal symbols** (tokens)
  - finite set of **production** rules, each looks like
    - nonterminal -> finite sequence of symbols

ex.
recursive grammar
expr -> expr + expr
expr -> expr * expr
expr -> ID
expr -> NUM
expr -> (expr)

context-free grammar
start = expr # start symbol
nt = {expr} # non terminal symbols set

t = {NUM, ID, *, +, (,)} # terminal symbols set
r = { } # ?

Notable Words:

- string = finite sequence of tokens
- sentence = string that belongs to language
- language = set of strings/sentences

—- 1/11/18