Abstract Syntax Trees

- Compilação em uma ou múltiplas passadas
  - Como isso afeta o design da linguagem (exemplo: forward-declarations em C)
- Hoje em dia, preferimos separar tarefas
  - Sempre? (Nimrod)
- Árvore de sintaxe abstrata (Abstract syntax tree, AST) é a estrutura de dados central para as fases pós-parsing

Árvores concretas e abstratas

- Árvore concreta: árvore de derivação
  - Exemplo: S → Digs $; Digs → Digs d | d
- Árvore abstrata
  - Abstratamente Digs representa uma lista de d
    - lembrando que a ordem é importante
  - Nenhuma gramática poderia gerar uma árvore de derivação assim

Uma estrutura de dados eficiente para ASTs

- A AST é tipicamente construída bottom-up: geram-se os filhos e estes são adotados por um nó pai
- Listas de irmãos são tipicamente gerados por regras recursivas: deve ser fácil inserir filhos em qualquer ponta da lista
- Alguns nós têm um número fixo de filhos, outros têm números variáveis

Estrutura do livro Crafting a Compiler

- parent
- leftmost sibling
- right sibling
- leftmost child

Estrutura que eu usei:

- head
- tail
- next
- prev

Design da AST:

- A AST deve ser projetada
  - Primeiro cria-se uma gramática não-ambígua para a linguagem
  - Depois projeta-se uma AST para ela, que descarta detalhes da gramática (pontuação, etc)
- Informação suficiente para representar a execução do programa
  - Tipicamente deve ser possível reconstituir a partir dela um programa equivalente
- Ações semânticas são adicionadas ao parser para construir a AST

Linguagem da página 256

Start → Stmt $
Stmt → id assign E
| if (E) Stmt else Stmt fi
| if (E) Stmt fi
| while (E) do Stmt od
| begin Stmts end
Stmts → Stmts ; Stmt
| Stmt
E → E + T
| T
T → id
| num

Que "tipos" de nó devem existir?
assign, if, while, block, plus

Figuras página 260

L-values e R-values
em uma linguagem como C, o que é "x = y"? qual o assembler equivalente?

x = y → (assign (id x) (deref (id y)))
x = &y → (assign (deref (id x)) (id y))
*x = y → (assign (deref (id x)) (deref (id y)))