1. Language
   1. Put these in an API to define syntax and semantics
   2. Syntax
      1. The structure of the expressions, statement. Functions, prog units
      2. Terms:
         1. Sentence
            1. Is a string of characters over some alphabet

Alphabet example {a,b}

* + - 1. Language
         1. Is a set of sentences
      2. Lexeme
         1. The lowest level syntactic unit of a language
         2. Ex:

Int I; //I and sum are identifier token lexemes

Float sum; // float and int are keywords or predefined types token

For(i=0;…) //for is a token

* + - 1. Token
         1. a category of lexemes
    1. Context free Grammar (CFG) (Chomsky Hierarchy)
       1. Designed in mid 1900s by Chomsky (linguist)
       2. Regular expression is the language
          1. Context free grammar sits on top of it

Then context sensitive grammar sits on top of it

Then the turing machine (Recursive enumerable) sits on top of it

* + 1. Backus-Nour Form (BNF)
       1. Used to represent any Context Free Grammar
    2. Grammar Rule
       1. G=(N,T,P,S)
          1. N: set of the non-terminal elements <name>
          2. T: set of the terminal elements {a,b}, {‘=’,’+’,’\*’,’-‘, ‘/’, ‘while’, for’}
          3. P: production rules (syntactic lists)

In out RULE below two rule a<s>b or ab

A lot of times you have to figure everything out from just the production rule

Rule: LHS🡪RHS

LHS will be one non-terminal

RHS will be mixed with terminal and nonterminal

Context sensitive language will have a mix of nonterminal and terminal with at least one terminal

Derivation (Pretty Tedious)

A repeated application of rules starting with the start symbol and ending with a sentence (all terminal symbols)

This derivation lets you check if it is a valid sentence within the language

* + - * 1. S: the start symbol which is non-terminal

Int I,j,k;

A list of value <type> <id\_list>

<id\_list>🡪ID|ID, <id\_list>

EXAMPLE: <s>🡪<var>=<expr>

<expr>🡪<term> + <term>|<term> - <term>

<term>🡪<var>

<var>🡪a|b|c|d

a=b+d

Derivation:

<s>🡪<var>=<expr> //can pick right most or left most

🡪a=<expr>

🡪a=<term>+<term>

🡪a=<var>+<term>

🡪a=b+<term>

🡪a=b+<var>

🡪a=b+d //yes valid

* + 1. Compiler implementation
       1. Parsers
          1. LL or LR
    2. Constructing a parsing tree is much easier see notes sheet
    3. How to construct more complicated languages
  1. Semantics
     1. The meaning
        1. V=expr;
           1. This is an example of assignment expression
  2. RULE: <s>🡪a<s>b|ab

1. Ambiguous Grammar
   1. Two or more parsing trees for the same sentence
      1. Quiz/test questions will pretty much always be ambiguous
2. Syntax analyzer
   1. LL: Top-down Recursive Descent (we use this in this class)
      1. How to write a top-down recursive descent parser
         1. Each non-terminal you write a function
         2. The function body contains the Right hand side of the production rules
         3. <s>🡪<A>c|<B>c
         4. <A>🡪a<A>|a
         5. <B>🡪b<B>|b
         6. Python implementation see script in notes folder
   2. LR: Bottom Up Recursive Descent (Yacc or Bison)