HIGH LEVEL VIEW OF A COMPILER

- must recognize legal (and illegal) programs.
- must generate correct and efficient code.
- must manage storage of all variables (& code).
- must agree with os & linker on format for object code.

+ big step up from assembly language
use higher level notations.

TRADITIONAL 2-PASS COMPILER

FRONT END $\longrightarrow$ IR $\longrightarrow$ BACKEND
$O(n) / O(n \log n)$ NPC

classic principle from software engineering:
separation of concerns.

- admits multiple frontends ~~and multiple~~.
- admits multiple passes

Structure of a 3 phase compiler:
Optimizer that outputs IR.
(must preserve "meaning" of the code

LEVELS OF IR

- high level IRs

  abstract syntax trees

  ↳ loop transformations
  procedure inlining } useful for

- intermediate level IR

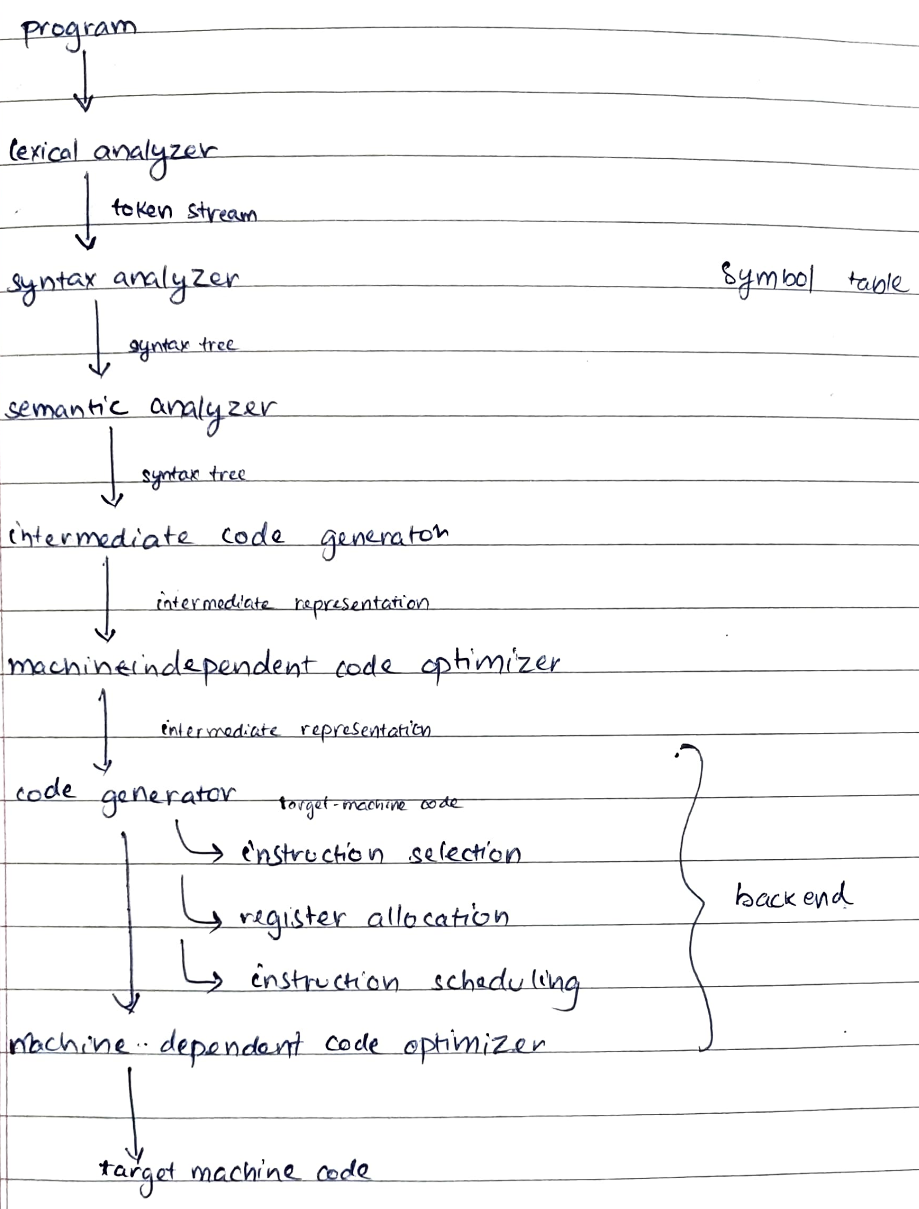
  three address code

  static signgle assignment form

  ↳ constant propagation } useful for.

- low-level IR

  ↳ good for low level machine dependent code.
  optimizations


program
lexical analyzer
token stream
syntax analyzer
symbol table
syntax tree
semantic analyzer
syntax tree
intermediate code generator
intermediate representation
machine-independent code optimizer
intermediate representation
code generator
target-machine code
instruction selection
register allocation
instruction scheduling
backend
machine-dependent code optimizer
target machine code