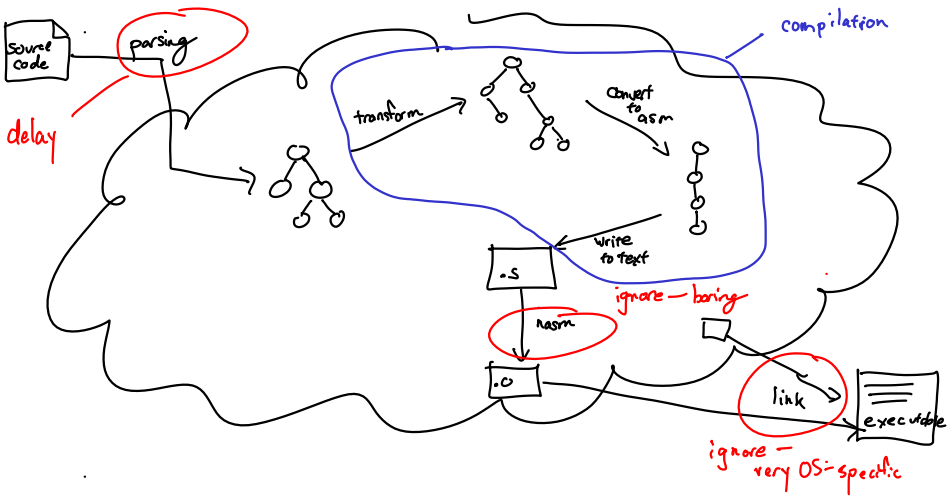

source code
parsing
delay
compilation
transform
convert to asm
write to text
.s
nasm
.o
ignore — boring
link
executable
ignore — very OS-specific

Syntax and Parsing

Auklet Language

<expr> ::= 0 | 1 | -1 | 2 | -2 | ⋯
| after (<expr>)
| before (<expr>)
| <expr> + <expr>
| <expr> - <expr>
| <expr> * <expr>
| (<expr>)

left assoc

high priority

EBNF grammar

3+4 ✓

1 - 2 + after(3) ✓

after(before(4)) ✓

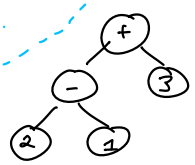
1 - after ✗ syntax error

Concrete syntax

abstract syntax

```
type ast = tree
  | EInt of int
  | EAfter of ast
  | EBefore of ast
  | EPlus of ast * ast
  | EMinus of ast * ast
  | ETimes of ast * ast
```

2 - 1 + 3



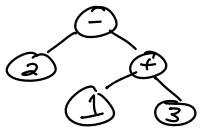
EPlus(EMinus(EInt 2, EInt 1), EInt 3)

Semantics

after(5) ⟹ 6

before(2+3) ⟹ 4

2 - (1+3)



x86 assembly

AT&T

```
mov $4, %eax
```

```
mov eax, 4
```

Intel

```
mov ebx, ecx
```

AMov(AReg(EAX), AConst(4))

```
type register =
    EAX

type arg =
    AReg of register
  | AConst of int

type instruction =
    AMov of arg * arg
    AAdd of arg * arg
```

answer should be in EAX once the instructions are executed

```
let rec compile_expression (e:ast) : instruction list =
  match e with
  | EInt n -> [AMov (AReg EAX, AConst n)]
  | EAfter(e') ->
    let instrs_e' = compile_expression e' in
    instrs_e' @ [AAdd(AReg EAX, AConst 1)]
```

mov eax n

instrs for e'

add eax, 1

Deque

$\alpha \longrightarrow e_1 :: e_2$ (e_2: α list)

α list $\longrightarrow e_1 \ @ \ e_2 \longleftarrow \alpha$ list

```
| EPlus(e1,e2) ->
    let e1 instrs = compile_expression e1 in
    let e2 instrs = compile_expression e2 in
        e1 instrs @ e2 instrs
```