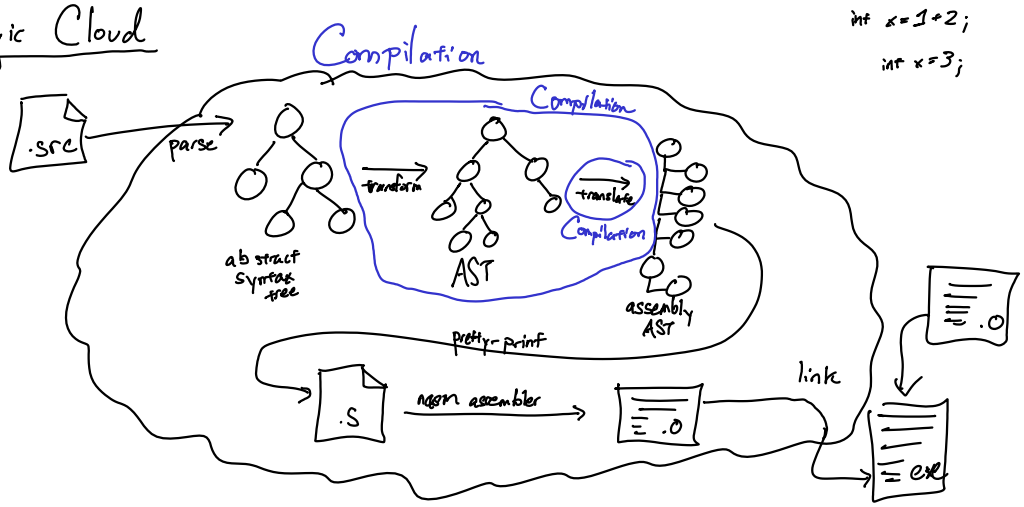
Magic Cloud

```
int x=1+2;
int x=3;
```



Syntax

Grammar (~EBNF) for Aublet

concrete syntax

EBNF Grammar defn

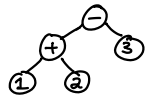
```
<expr> ::= 0 | 1 | -1 | 2 | -2 | ...
         | after(<expr>)
         | before(<expr>)
         | <expr> + <expr>
         | <expr> - <expr>
         | <expr> * <expr>
```

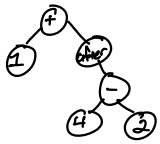
abstract syntax

OCaml source code

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

4 ⟶ (4)

1 + 2 - 3 ⟶  ⟶ EMinus(EPlus(EInt 1, EInt 2), EInt 3)

1 + after(4-2) ⟶ 

7 - after ✗

Semantics

1 + 2 - 3 ⟹ 0

1 + after(4-2) ⟹ 4

A compiler translates (↑ preserve meaning) from one language to another.

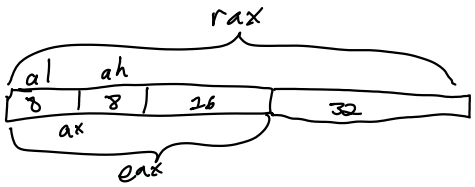
x86 assembly

AT&T

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

Intel

```
mov eax, 4
```



x64 assembly

```
mov rax, 4
```

```
type instr =
  | AsmMov of arg * arg
  | AsmAdd of arg * arg

type arg =
  | ArgReg of reg
  | ArgConst of int

type reg =
  | RAX
```

AsmMov(ArgReg RAX, ArgConst 4)

```
let rec compile (e : expr) : instr list =
```

rec invariant: instructions will leave the answer to the expr in the RAX register

```
  match e with
  | EInt n ->                                   (* mov rax, n *)
      [AsmMov(ArgReg RAX, ArgConst n)]
  | EAfter(e') ->
      compile e' @ [AsmAdd(ArgReg RAX, ArgConst 1)]
```

5

```
mov rax, 5
```

after(5)

```
mov rax, 5
add rax, 1
```