Parsing

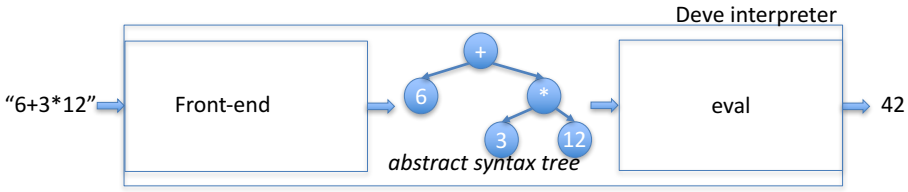
CS 321 Programming Languages

Ozyegin University

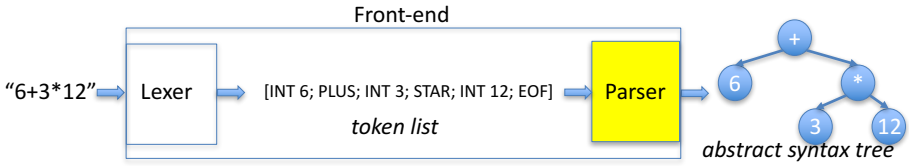
Barış Aktemur

The Big Picture

The Big Picture



The Front-End

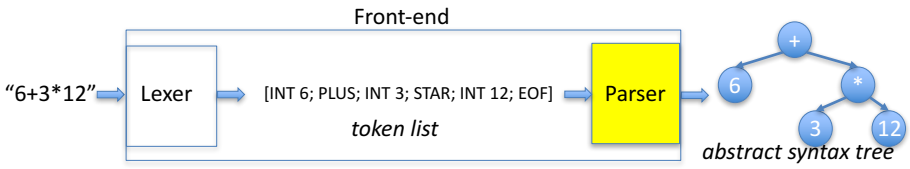


- Parser: Takes the token list, and builds the AST.

Language Grammar

- All languages (natural ones such as Turkish, English, Korean, etc. as well as artificial ones such as Java, OCaml, Python) have a grammar that describe a viable structure for the programs/text written in that language.
- Grammatical rules are about what makes a syntactically correct program.
- A grammatically correct input is not necessarily meaningful (i.e. semantically correct).
  - In terms of grammar, the sentence below is fine. But semantically it has problems
  - "The weather tomorrow was a green bazinga."

The Front-End



- Parser: Takes the token list, and builds the AST.
- Parsing is about checking whether the input is a **syntactically** correct program; it does not perform semantic checks.

Context-Free Grammars

```
main ::= exp EOF
exp  ::= INT
       | NAME
       | exp PLUS exp
       | exp STAR exp
       | LET NAME EQ exp IN exp
       | IF exp THEN exp ELSE exp
```

Note: *This is not runnable code. It is a notation to express grammars. This notation is called the Backus-Naur Form (BNF).*

Context-Free Grammars

```
main ::= exp EOF
exp  ::= INT
       | NAME
       | exp PLUS exp
       | exp STAR exp
       | LET NAME EQ exp IN exp
       | IF exp THEN exp ELSE exp
```

Terminals (tokens)

Note: *This is not runnable code. It is a notation to express grammars. This notation is called the Backus-Naur Form (BNF).*

Context-Free Grammars

```
main ::= exp EOF
exp  ::= INT
       | NAME
       | exp PLUS exp
       | exp STAR exp
       | LET NAME EQ exp IN exp
       | IF exp THEN exp ELSE exp
```

Terminals (tokens)
Non-terminals

Note: *This is not runnable code. It is a notation to express grammars. This notation is called the Backus-Naur Form (BNF).*

Context-Free Grammars

```
A main ::= exp EOF
B exp  ::= INT
C        | NAME
D        | exp PLUS exp
E        | exp STAR exp
F        | LET NAME EQ exp IN exp
G        | IF exp THEN exp ELSE exp
```

Terminals (tokens)
Non-terminals
Productions

Note: *This is not runnable code. It is a notation to express grammars. This notation is called the Backus-Naur Form (BNF).*

Context-Free Grammars

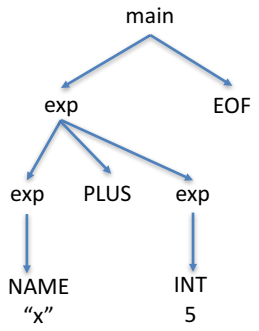
```
A main ::= exp EOF
B exp  ::= INT
C        | NAME
D        | exp PLUS exp
E        | exp STAR exp
F        | LET NAME EQ exp IN exp
G        | IF exp THEN exp ELSE exp
```

Terminals (tokens)
Non-terminals
Productions
Start symbol

Note: *This is not runnable code. It is a notation to express grammars. This notation is called the Backus-Naur Form (BNF).*

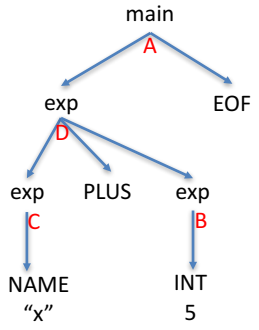
Derivation

- Start with the starting symbol.
- At each step, pick a non-terminal, and expand it by applying one of the productions.
- Stop when no more non-terminals remain (i.e. all we have are terminals).
- The result is a grammatically correct input according to the specified grammar.



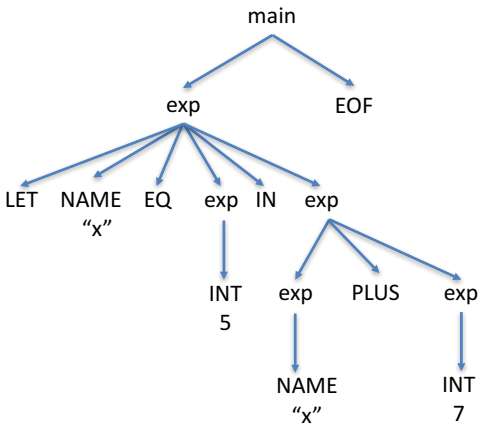

```
A) main ::= exp EOF
B) exp  ::= INT
C)        | NAME
D)        | exp PLUS exp
E)        | exp STAR exp
F)        | LET NAME EQ exp IN exp
G)        | IF exp THEN exp ELSE exp
```

- This is called the "derivation tree" or "concrete syntax tree".
- So, NAME PLUS INT EOF is a valid input in this grammar. E.g. "x + 5"




```
A) main ::= exp EOF
B) exp  ::= INT
C)        | NAME
D)        | exp PLUS exp
E)        | exp STAR exp
F)        | LET NAME EQ exp IN exp
G)        | IF exp THEN exp ELSE exp
```

- This is called the "derivation tree" or "concrete syntax tree".
- So, NAME PLUS INT EOF is a valid input in this grammar. E.g. "x + 5"




```
A) main ::= exp EOF
B) exp  ::= INT
C)        | NAME
D)        | exp PLUS exp
E)        | exp STAR exp
F)        | LET NAME EQ exp IN exp
G)        | IF exp THEN exp ELSE exp
```

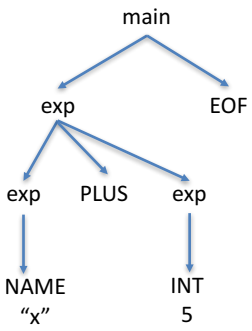
- LET NAME EQ INT IN NAME PLUS INT EOF is a valid input in this grammar.
  - "let x = 5 in x + 7"

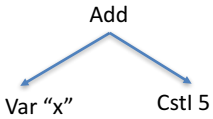
Parsing

- Parsing is the activity of finding whether a given input has a derivation according to the grammar. And, if so, what's that derivation?
- During parsing, we also perform some actions to build an **abstract syntax tree (AST)**. We do not keep the **concrete syntax tree (CST)**, because AST is more useful for evaluation. CST contains lots of unnecessary info that we can throw away.

Concrete vs. Abstract Syntax Tree

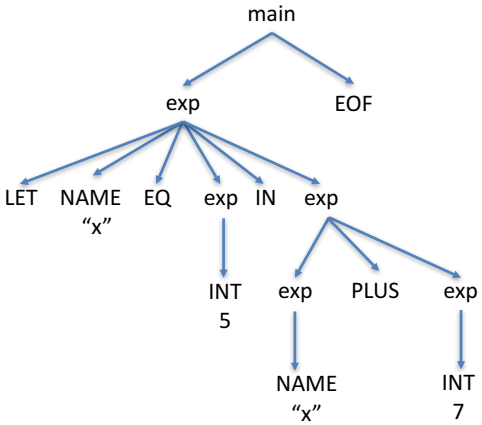
"x + 5"

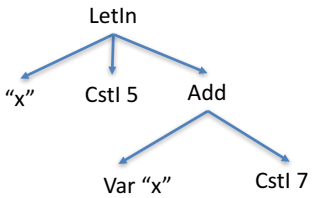




Concrete vs. Abstract Syntax Tree

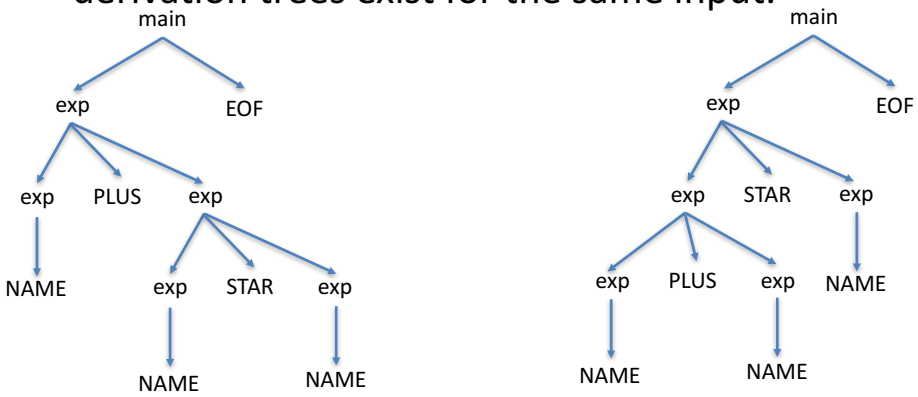
"let x = 5 in x + 7"





Ambiguity

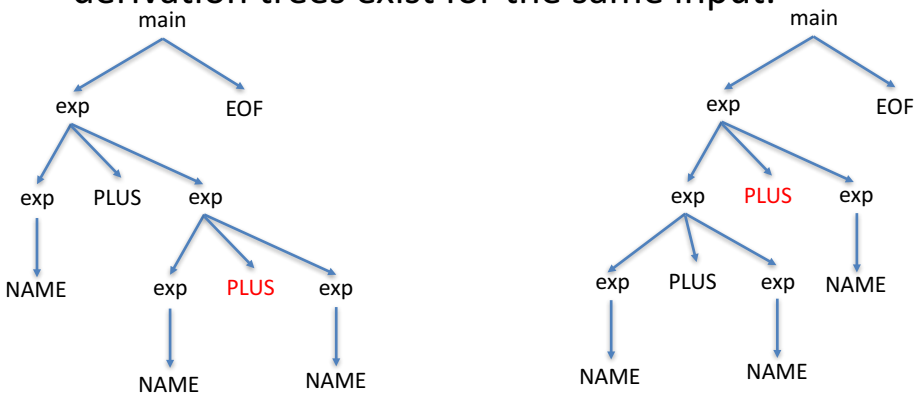
- A grammar is ambiguous if more than one derivation trees exist for the same input.



E.g. "x + y * z"

Ambiguity

- A grammar is ambiguous if more than one derivation trees exist for the same input.



E.g. "x + y + z"

Ambiguity

- Derive two different parse trees for each:
  - "let x = 5 in x + 9"
  - "if c then 42 else 5 + x"
  - "9 – 3 – 2"
- We need **precedence** and **associativity** of the operators defined precisely.