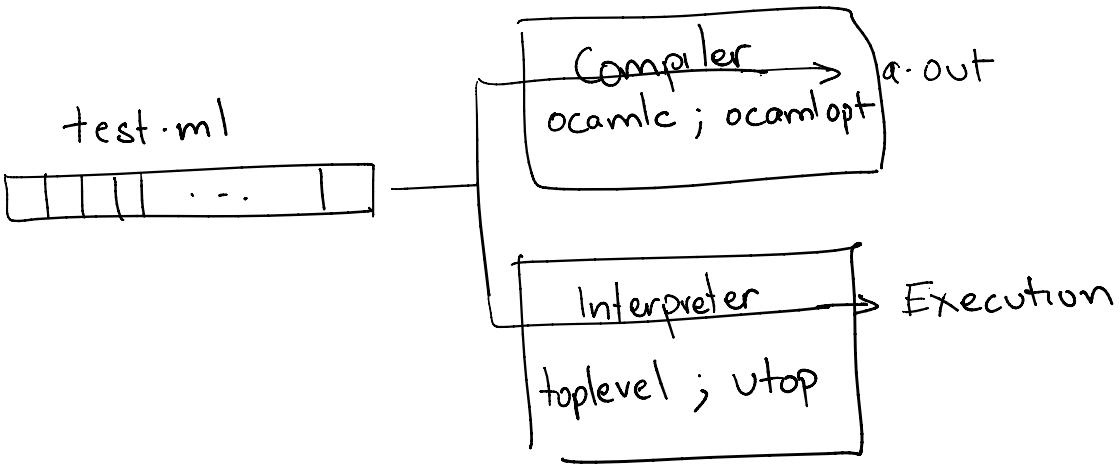
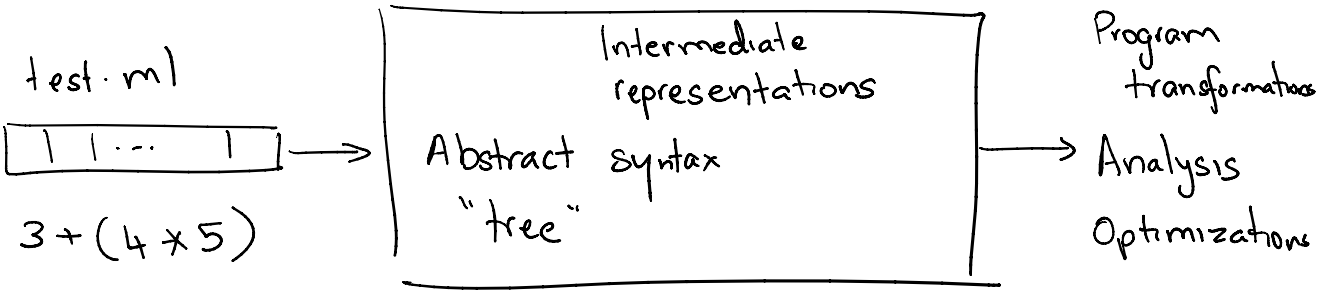
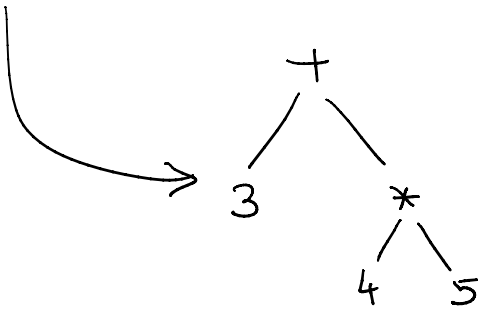
Lecture 12

Thursday, October 1, 2020 3:34 PM







Abstract syntax tree

John Backus

"How do you specify a language?"

- Syntax

"3 < > 4" is badly formed C.

"What is the shape of well-formed code?"

- Semantics

Colourless green ideas sleep furiously.

- Semantics

"What does a program mean?"

Backus-Naur form

Context-free grammars

A Python program means what the CPython interpreter does.

```
void foo() {
    int x = 3;
}
```

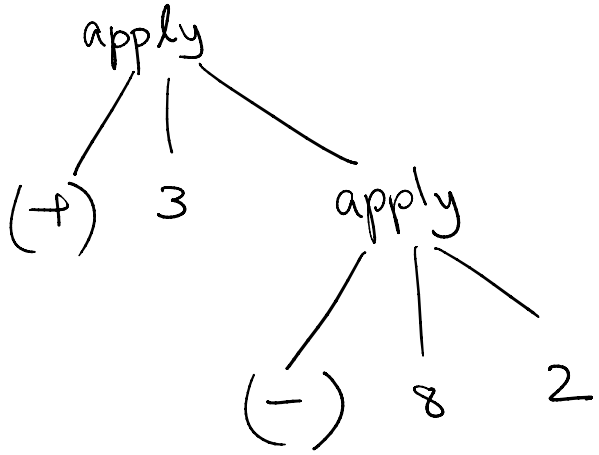
```
void foo() {
    int int = 5;
}
```

```
void >> {
    int = 8
}
```

Well-formed program

Ill-formed programs

$3 + \longrightarrow\longrightarrow (8 \longrightarrow\longrightarrow -2)$



$* + 3 -$

↑

There is no syntactic structure

$[0-9]^+$ ⇝ Sequence of one or more ASCII digits

[0-9]$^+$ ⇝ Sequence of one or more ASCII digits

Expr ::= IntegerLiteral ⇝ 38 | 42 | 96 | ...

| $Expr_1$ '+' $Expr_2$ ⇝ Two expressions, with a plus sign in between

| $Expr_1$ '−' $Expr_2$ ⇝ Two expressions, with a minus sign in the middle

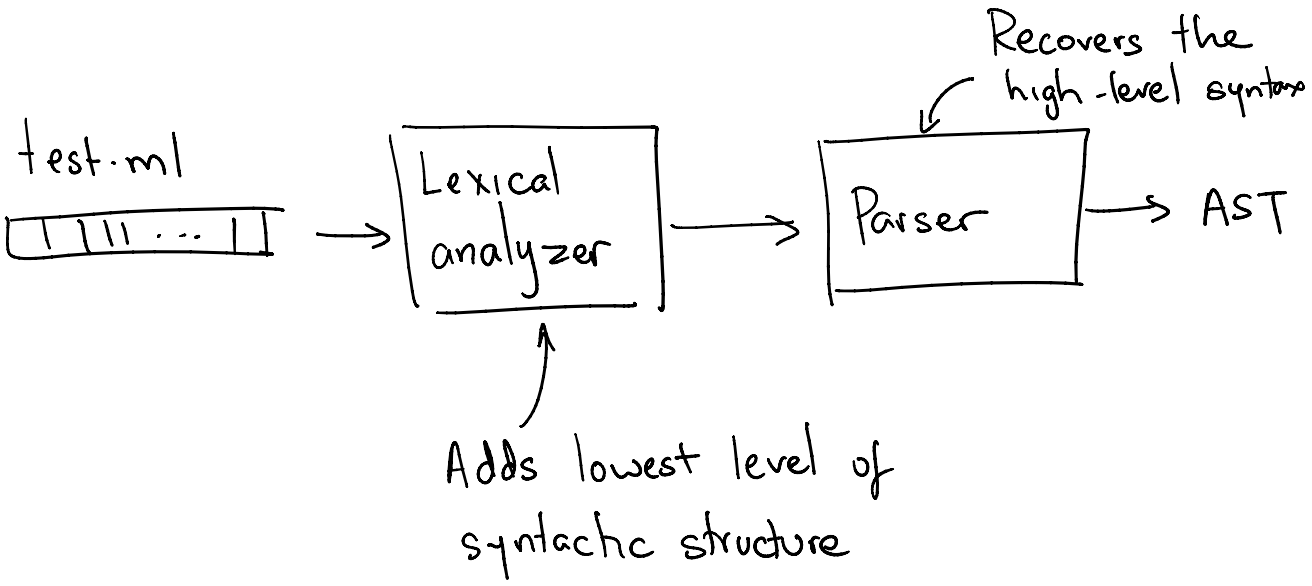
| '(' Expr ')' ⇝ An expression, within parentheses.

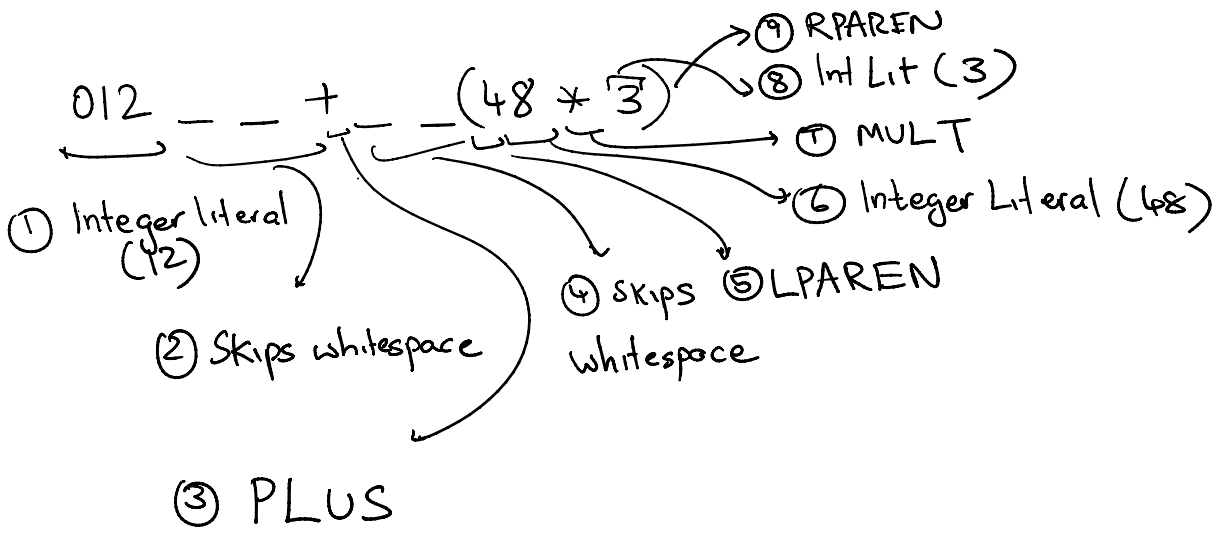
Question: How does one go from sequences of characters to a parse tree?

Lots of remaining issues?

- Is "for" an identifier or a keyword?
- What about whitespace?
- What about comments?
- What about preprocessor macros?

- What about preprocessor macros?





Set of tokens:

① PLUS ② MINUS ③ MULT

④ LPAREN ⑤ RPAREN

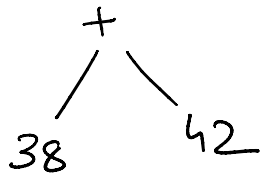
] Simple tokens

⑥ IntegerLiteral(c: int) Token with payload.

"38" ⟼ IntegerLiteral(38)

"38+42" ⟼ (Lexer) [IntegerLiteral(38);
PLUS;
IntLiteral(42)]

↓ Parser

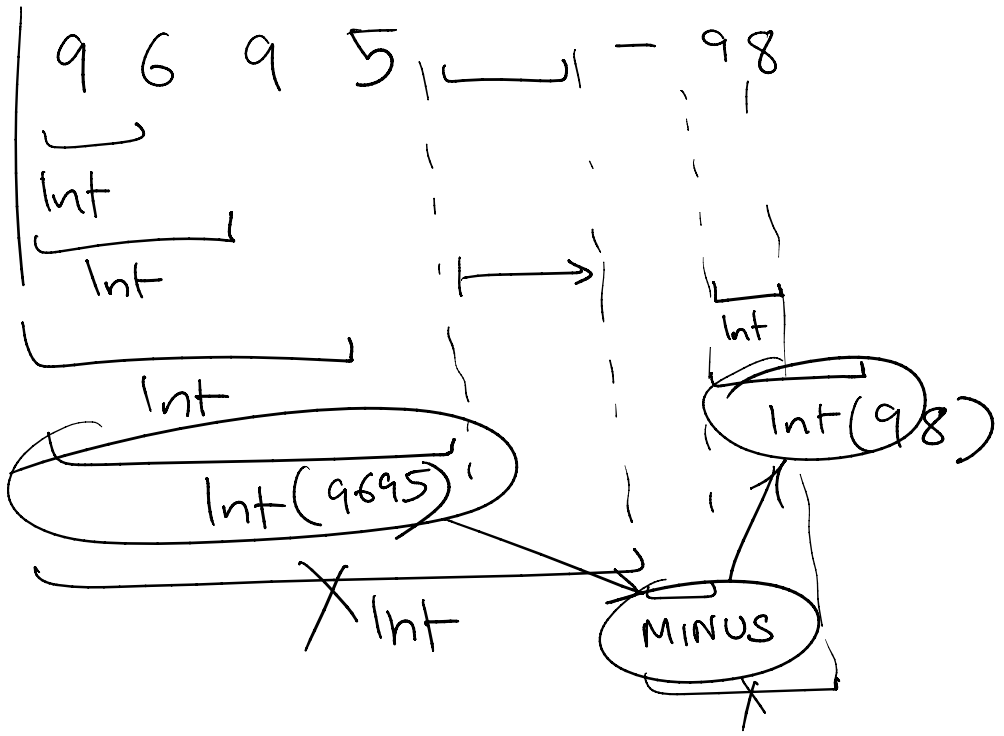


"96 _ 95" ⟶✗ Int(9695)

"9695" →
- [Int(9); Int(6); Int(9); Int(5)]
- [Int(9); Int(695)]
- [Int(96); Int(95)]
- ⋮
- [.]

[Int(9695)]

"Longest match principle".



For practical regex matching,

look for how it is done in Perl.

Simple "academic" regular expressions.

Digit = '0', '1', '2', ..., '9'

= ['0' - '9'] ← A character or a set of characters is a regular expr.

INT = "int"

FLOAT = "float"

A sequence of characters is a regular expr

↓ Regular exprs.

IntLit = DIGIT⊕ | '_' DIGIT⊕

↖ An option between two reg exprs is itself a reg expr.

Ident = Alpha [AlphaNum]⊛

Alpha: 'a' — 'z', 'A' — 'Z'

AlphaNum: 'a' - 'z', 'A' - 'Z', '0' - '9'

A sequence of zero or more occurrences of AlphaNum.

Alpha Num.

If r is a regular expr.

then r^* is also a reg. expr

r^+