07 Apr 2025 - Compilers -1 - Week 02

syntax analyser = passer

annotated syntax tree = abstracted .. ..

transformations ⇝ semantically equivalent

input = high-level stream

~~unofficially~~ preprocessor

replace comments with blank

- Appendix A, KR
  ↳ 6 classes of tokens

→ Comments do not nest

→ Comments ⟿ greedy algorithm

→ Preprocess

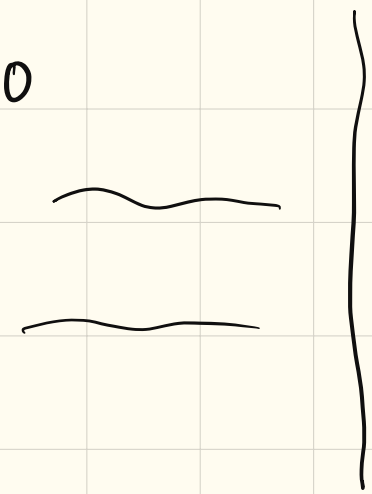
/* ok /* ok */ */

warning

error

if 0

commented out



end if

→ we will not build preproc.

→ Why modular?

↳ good for software design.

→ LA ⟶ not a clear difference

preprocessor

→ preprocessor might do more work sometimes.

LA ⇝ finite automata

→ keywords

→ first step: propose a language.

identifiers $l\,(\,l + d + \_\,)^*$

old C ⇝ 31 char

→ array

new C ⇝ stdstring

Will regex for identifier also match keywords?

Yes

Mini - languages

→ High - performance computing.

→ Token ⟶ given by language designer
→ literal strings " ... "

→ Pattern

→ Lexeme

→ Brackets LISP

( )

[ ]

< > ⟶ templates, (#include < > ⟶ preprocessor)

{ } ⟶ blocks

→ comma , colons , etc.

↓

:

ternary operator

include → keyword technically but eaten up by preproc.

IDE , code - completion

↓ ↳ make the ML model take and o/p tokens

has a small abstract syntax tree to suggest code

09 Apr 2025

→ Missed extra class.

Little Languages

Domain-specific languages.

Regular Expressions

$$r = c^* (a \underset{\text{or}}{+} bc^*)^*$$

L = set of all strings over $\{a, b, c\}$ that do not contain ac.

⇝ o Proof

Integer constants K & R

octal $0\ (0 + 1 + \cdots + 7)^*$

hex $(0x + 0X)(0 + \cdots + 9 + A + \cdots + F + a + \cdots + f)^*$

→ may be suffixed by u or U, L or l;

Is the limitation set by language or the compiler?

→ Code compliance → lU works but l can be confused with 1.
→ LU ✓ lU ~ disallowed / warn by some compilers.

unsigned ⟶ one more bit

unsigned ⟶ type qualifier

unsigned ⟶ keyword

```
unsigned int a = ~~
int b = a;
```

```
64 ⟶ double aa;
32 ⟶ float b1 = aa;
```

4, 8, 16 } Neural net

useless precision { 32, 64, 80, 128

LEX file `.l` ⇝ lexical analyser rules

D [0-9]

L ← letter

H ← hex

E ← exponent

auto = conditional

char or long

↓

longest

auto ~~ = ~~~~

↓

if this is compile type evaluation

→ auto keyword in C

→ auto in C++

"pythonification"

→ gcc and clang source code.

Explore source code if manuals don't work.

→ Backward compatibility is very important

— Intel.

→ auto in function argument → f(auto, ...)

OOPs will

→ Character constants octal \000 → ??

→ Floating Constant