

Exercise walk-through

11.4

Lexical analysis

University Computer Science

Lexical analysis: longest match wins

- 1 **Line** `count = a1 + b2;` lexes to IDENT ASSIGN IDENT PLUS IDENT SEMI at columns 1 7 9 12 14 16.
- 2 **Match** `ifx` is one IDENT - greedy longest match beats the keyword prefix; `if` alone is a keyword.
- 3 **Policy** `12abc` is NUM then IDENT: some languages allow it, ours reports it - positions carried for every diagnostic.

Token pipeline

characters in, tokens out — longest match wins

```
count = a1 + b2;
```

1 4 7 10 13 16



DFA per token class (regex-compiled)



policy notes: ifx is one IDENT (longest match beats the keyword prefix); whitespace skipped; positions kept

error policy is part of the specification: 12abc is NUM then IDENT — our lexer reports it

Characters in, **tokens** out - longest match with priority.

Exercises 2.1-2.3

2.1 Why must the lexer keep positions?

[3]

2.2 What wins between `if` and `ident` on the input `ifx`?

[3]

2.3 What is discarded on the way through?

[3]

Exercises 2.1-2.3

2.1 Why must the lexer keep positions? [3]

2.1

- Every later diagnostic (parse and semantic errors) cites them ✓.

2.2 What wins between `if` and `ident` on the input `ifx`? [3]

2.2

- `ident` – the longest lexeme wins; `ifx` is one IDENT ✓.

2.3 What is discarded on the way through? [3]

2.3

- Whitespace and comments ✓.

Exercise 3.1

3.1 Lex `total = ifx + 42;` under the course specification (keyword `if`; IDENT letters/digits; NUM digits; operators `=`, `+`; punctuation `;`).

List the token stream with type, lexeme and column for each. Justify the `ifx` decision by longest-match-plus-priority. State the NUM policy for a following letter (`42abc`) under our spec. State which single token would change if keywords were removed from the spec.

[8]

Exercise 3.1

3.1 Lex `total = ifx + 42;` under the course specification (keyword `if`; IDENT letters/digits; NUM digits; operators `=`, `+`; punctuation `;`).

List the token stream with type, lexeme and column for each. Justify the `ifx` decision by longest-match-plus-priority. State the NUM policy for a following letter (`42abc`) under our spec. State which single token would change if keywords were removed from the spec. [8]

3.1

- IDENT(`total`)@1, ASSIGN(`=`)@7, IDENT(`ifx`)@9, PLUS(`+`)@13, NUM(`42`)@15, SEMI ✓.
- `ifx`: the ident regex matches 3 characters, `if` matches 2 – longest wins; priority would matter only on exact ties ✓.
- Under our spec `42abc` is NUM(`42`) then IDENT(`abc`) – reported as a lexical error (adjacent value tokens with no separator) ✓.
- Without keywords, nothing changes on THIS line – `ifx` stays IDENT; `if` alone would change from KW to IDENT ✓.

Exercise 3.2

3.2 Two error policies for `12abc`: report as an error, or emit `NUM(12)` then `IDENT(abc)`.

State what each policy makes legal downstream. Argue which fits a language where identifiers cannot start with a digit. State the general principle the choice illustrates about lexer specification ✓.

[7]

Exercise 3.2

3.2 Two error policies for `12abc`: report as an error, or emit `NUM(12)` then `IDENT(abc)`.

State what each policy makes legal downstream. Argue which fits a language where identifiers cannot start with a digit. State the general principle the choice illustrates about lexer specification ✓. [7]

3.2

- Report-as-error: the lexer refuses and no parser state is built ✓.
- Emit-both: `NUM` then `IDENT` reaches the parser, which will choke on two adjacent value tokens ✓.
- Identifiers cannot start with a digit here, so `12abc` cannot be one token of either class – the error policy matches the language ✓.
- Principle: the error policy is part of the lexer's specification – chosen, not discovered ✓.