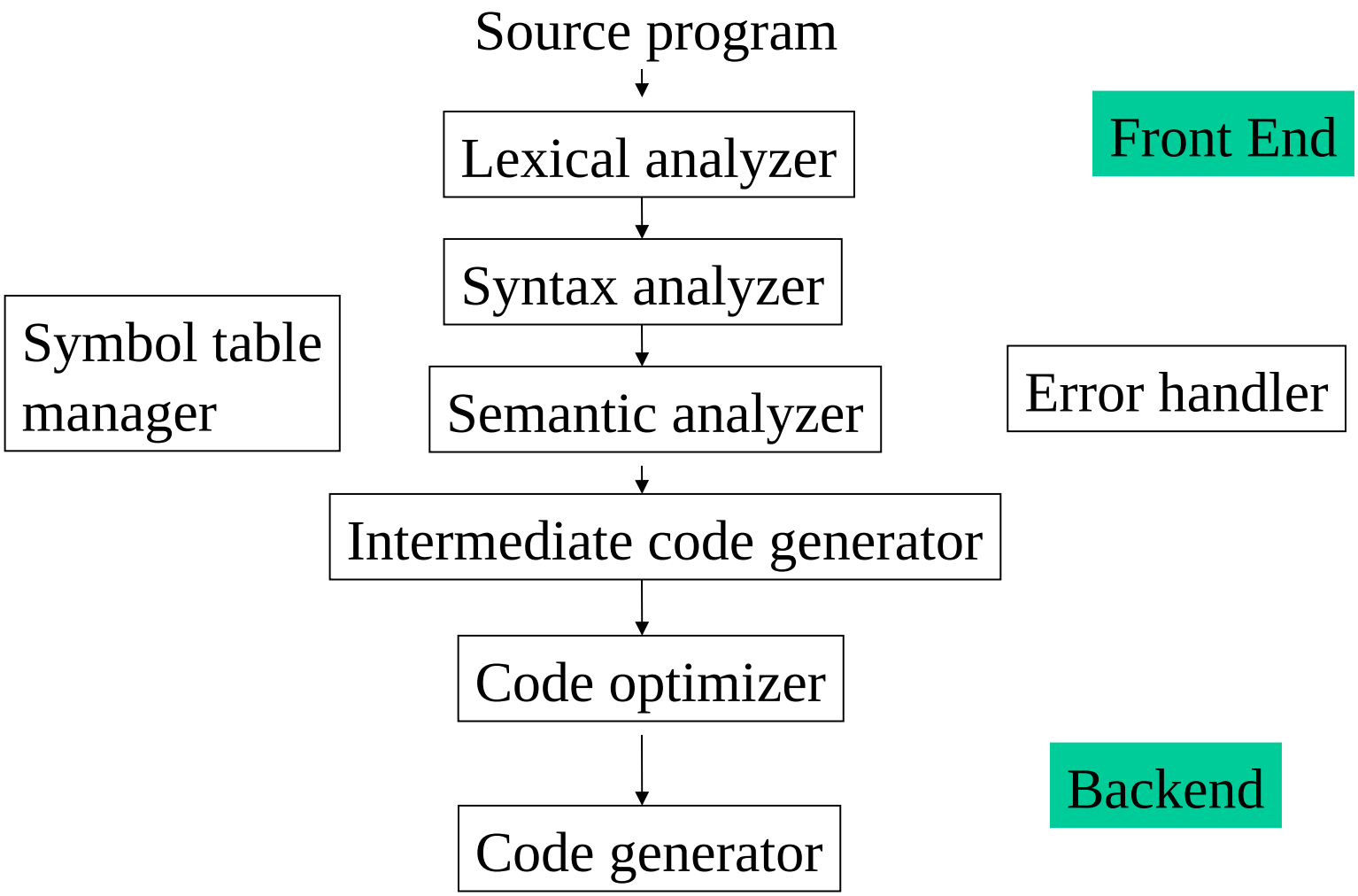
Review: Compiler Phases:



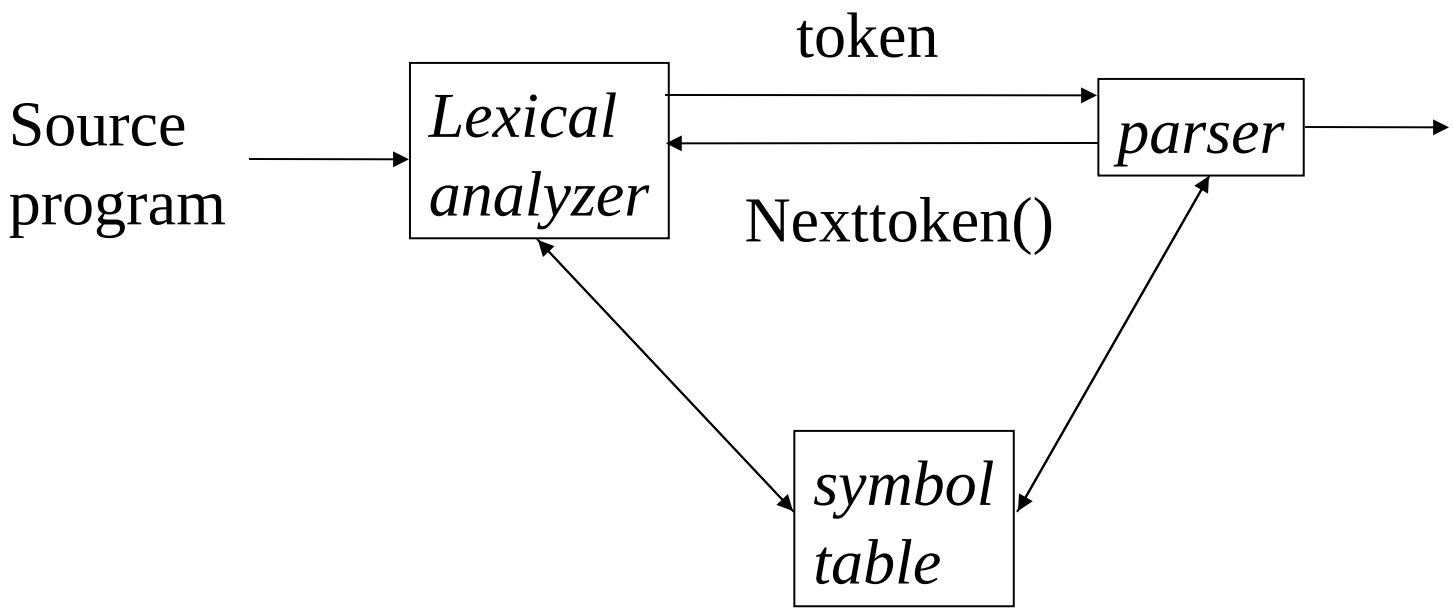
Chapter 3: Lexical Analysis

- Lexical analyzer: reads input characters and produces a sequence of tokens as output (nexttoken()).
  - Trying to understand each element in a program.
  - *Token*: a group of characters having a collective meaning.

    const pi = 3.14159;

    Token 1: (const, -)
    Token 2: (identifier, ‘pi’)
    Token 3: (=, -)
    Token 4: (realnumber, 3.14159)
    Token 5: (;, -)

Interaction of Lexical analyzer with parser



- Some terminology:
  - *Token*: a group of characters having a collective meaning. A *lexeme* is a particular instant of a token.
    - E.g. token: identifier, lexeme: pi, etc.
  - *pattern*: the rule describing how a token can be formed.
    - E.g: identifier: ([a-z]|[A-Z]) ([a-z]|[A-Z]|[0-9])*

- Lexical analyzer does not have to be an individual phase. But having a separate phase simplifies the design and improves the efficiency and portability.

- Two issues in lexical analysis.
  - How to specify tokens (patterns)?
  - How to recognize the tokens giving a token specification (how to implement the nexttoken() routine)?
- How to specify tokens:
  - all the basic elements in a language must be tokens so that they can be recognized.

```
main() {
    int i, j;
    for (I=0; I<50; I++) {
        printf("I = %d", I);
    }
}
```

    - Token types: constant, identifier, reserved word, operator and misc. symbol.
  - Tokens are specified by **regular expressions**.

- Some definitions
  - *alphabet* : a finite set of symbols. E.g. {a, b, c}
  - A *string* over an alphabet is a finite sequence of symbols drawn from that alphabet (sometimes a string is also called a sentence or a word).
  - A *language* is a set of strings over an alphabet.
  - Operation on languages (a set):
    - union of L and M, L U M = {s|s is in L or s is in M}
    - concatenation of L and M

      LM = {st | s is in L and t is in M}
    - Kleene closure of L, L^{i}
    - Positive closure of L, L^{i}
  - Example:
    - L={aa, bb, cc}, M = {abc}

- Formal definition of Regular expression:f
  - Given an alphabet $\sum$,
  - (1) ε is a regular expression that denote { ε }, the set that contains the empty string.
  - (2) For each $a \in \sum$, a is a regular expression denote {a}, the set containing the string a.
  - (3) r and s are regular expressions denoting the language (set) L(r) and L(s). Then
    - (r) | (s) is a regular expression denoting L(r) U L(s)
    - (r) (s) is a regular expression denoting L(r) L (s)
    - (r)* is a regular expression denoting (L (r)) *
  - Regular expression is defined together with the language it denotes.

- Examples:
  - let $\sum = \{a, b\}$

    a | b

    (a | b) (a | b)

    a *

    (a | b)*

    a | a*b

  - We assume that '*' has the highest precedence and is left associative. Concatenation has second highest precedence and is left associative and '|' has the lowest precedence and is left associative
    - (a) | ((b)*(c)) = a | b*c

- Regular definition.
  - gives names to regular expressions to construct more complicate regular expressions.

    d1 -> r1

    d2 ->r2

    …

    dn ->rn
  - example:

    letter -> A | B | C | … | Z | a | b | …. | z

    digit -> 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

    identifier -> letter (letter | digit) *

    - more examples: integer constant, string constants, reserved words, operator, real constant.