Compiler Construction

Lecture 2: Lexical Analysis I (Introduction)

Thomas Noll

Lehrstuhl für Informatik 2
(Software Modeling and Verification)

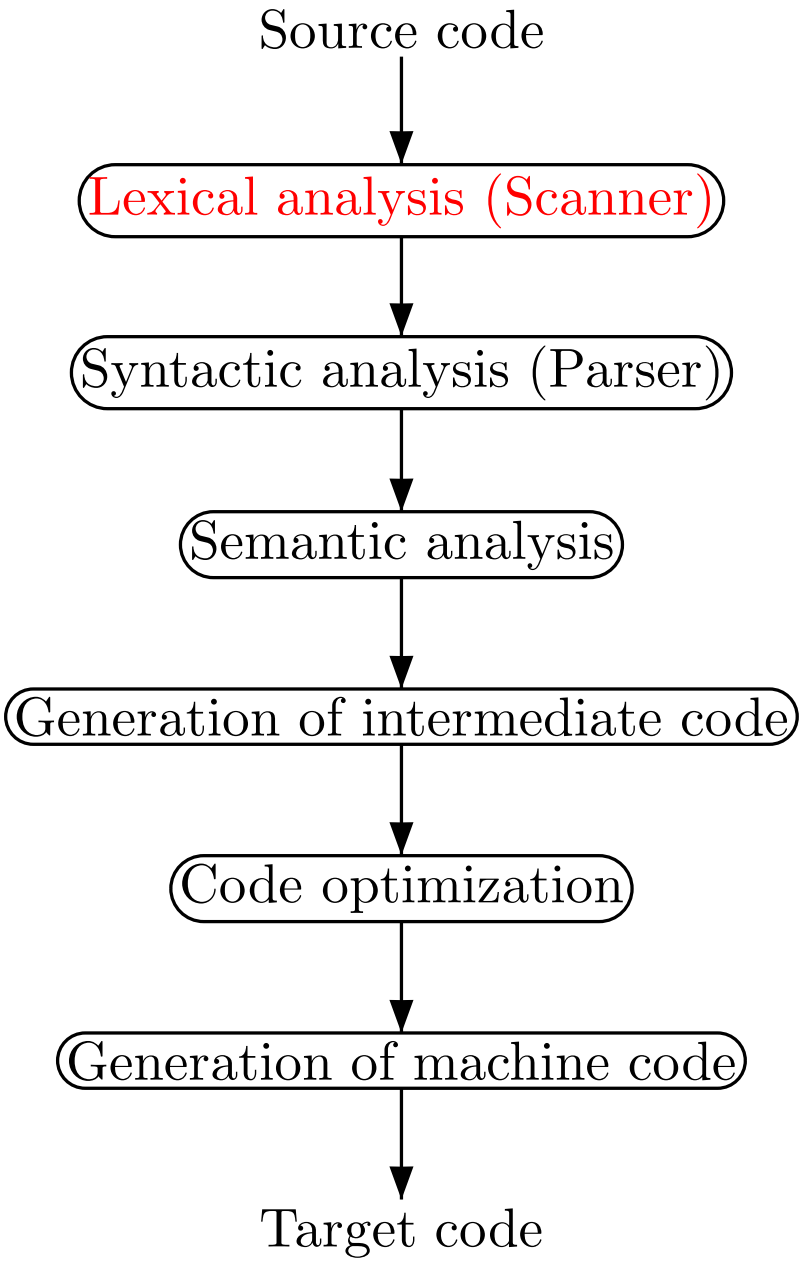
RWTH Aachen University

noll@cs.rwth-aachen.de

http://www-i2.informatik.rwth-aachen.de/i2/cc10/

Winter semester 2010/11

Conceptual Structure of a Compiler



Outline

Lexical Structures

From Merriam-Webster's Online Dictionary

Lexical: of or relating to words or the vocabulary of a language as distinguished from its grammar and construction

Lexical Structures

From Merriam-Webster's Online Dictionary

Lexical: of or relating to words or the vocabulary of a language as distinguished from its grammar and construction

- **Starting point:** source program P as a character sequence
  - Ω (finite) character set (e.g., ASCII, ISO Latin-1, Unicode, ...)
  - $a, b, c, \ldots \in \Omega$ characters (= lexical atoms)
  - $P \in \Omega^*$ source program
    (of course, not every $w \in \Omega^*$ is a valid program)

Lexical Structures

From Merriam-Webster's Online Dictionary

Lexical: of or relating to words or the vocabulary of a language as distinguished from its grammar and construction

- **Starting point:** source program P as a character sequence
  - Ω (finite) character set (e.g., ASCII, ISO Latin-1, Unicode, ...)
  - $a, b, c, \ldots \in \Omega$ characters (= lexical atoms)
  - $P \in \Omega^*$ source program
    (of course, not every $w \in \Omega^*$ is a valid program)
- P exhibits lexical structures:
  - natural language for keywords, identifiers, ...
  - mathematical notation for numbers, formulae, ...
    (e.g., $x^2 \rightsquigarrow$ `x**2`)
  - spaces, linebreaks, indentation
  - comments and compiler directives (pragmas)
- Translation of P follows its hierarchical structure (later)

Observations

1. Syntactic atoms (called symbols) are represented as sequences of input characters, called lexemes

First goal of lexical analysis

Decomposition of P into a sequence of lexemes

Observations

1. Syntactic atoms (called symbols) are represented as sequences of input characters, called lexemes

First goal of lexical analysis

Decomposition of P into a sequence of lexemes

2. Differences between similar lexemes are (mostly) irrelevant (e.g., identifiers do not need to be distinguished)
   - lexemes grouped into symbol classes (e.g., identifiers, numbers, ...)
   - symbol classes abstractly represented by tokens
   - symbols identified by additional attributes (e.g., identifier names, numerical values, ...; required for semantic analysis and code generation)

   $\Longrightarrow$ symbol = (token, attribute)

Second goal of lexical analysis

Transformation of a sequence of lexemes into a sequence of symbols

Lexical Analysis

Definition 2.1

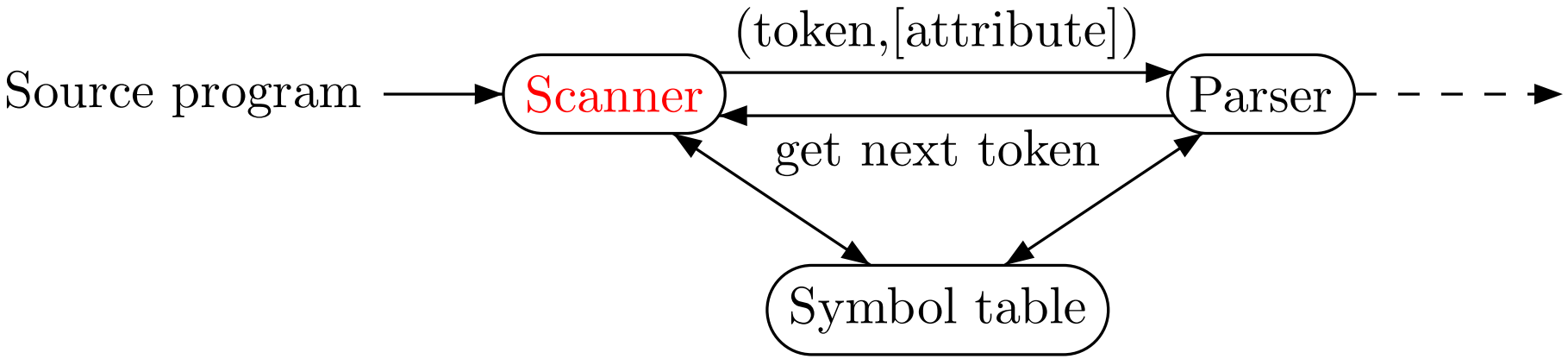
The goal of lexical analysis is to decompose a source program into a sequence of lexemes and their transformation into a sequence of symbols.

Lexical Analysis

Definition 2.1

The goal of lexical analysis is to decompose a source program into a sequence of lexemes and their transformation into a sequence of symbols.

The corresponding program is called a scanner:

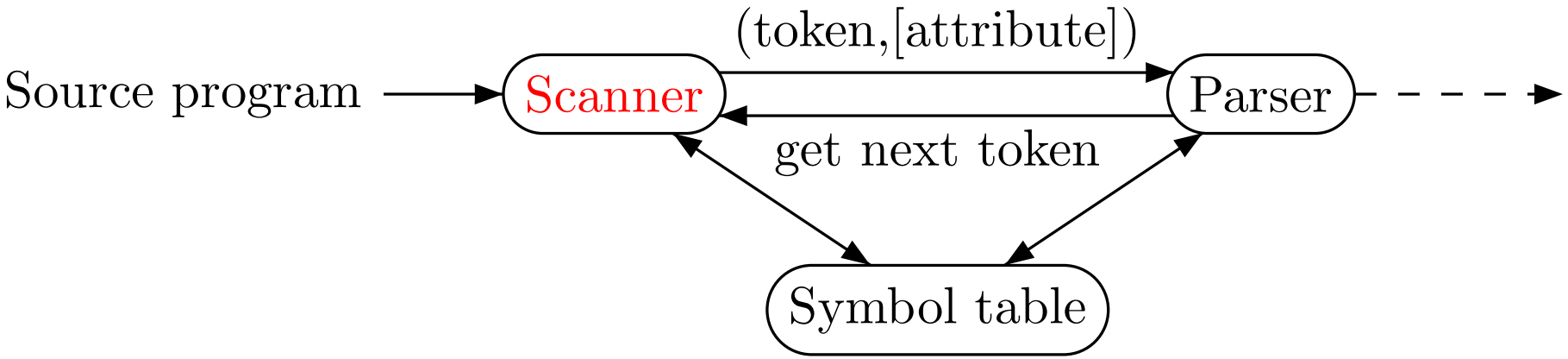


Lexical Analysis

Definition 2.1

The goal of lexical analysis is to decompose a source program into a sequence of lexemes and their transformation into a sequence of symbols.

The corresponding program is called a scanner:



Example:

$$\ldots\,␣\texttt{x1}␣\texttt{:=y2+}␣\texttt{1}␣\texttt{;}␣\ldots$$

$$\Downarrow$$

$$\ldots(\mathsf{id}, p_1)(\mathsf{gets},)(\mathsf{id}, p_2)(\mathsf{plus},)(\mathsf{int}, 1)(\mathsf{sem},)\ldots$$

Important Symbol Classes

Identifiers:

- for naming variables, constants, types, procedures, classes, ...
- usually a sequence of letters and digits (and possibly special symbols), starting with a letter
- keywords usually forbidden; length possibly restricted

Important Symbol Classes

Identifiers:

- for naming variables, constants, types, procedures, classes, ...
- usually a sequence of letters and digits (and possibly special symbols), starting with a letter
- keywords usually forbidden; length possibly restricted

Keywords:

- identifiers with a predefined meaning
- for representing control structures (`while`), operators (`and`), ...

Important Symbol Classes

Identifiers:
- for naming variables, constants, types, procedures, classes, ...
- usually a sequence of letters and digits (and possibly special symbols), starting with a letter
- keywords usually forbidden; length possibly restricted

Keywords:
- identifiers with a predefined meaning
- for representing control structures (`while`), operators (`and`), ...

Numerals: certain sequences of digits, `+`, `-`, `.`, letters (for exponent and hexadecimal representation)

Important Symbol Classes

Identifiers:
- for naming variables, constants, types, procedures, classes, ...
- usually a sequence of letters and digits (and possibly special symbols), starting with a letter
- keywords usually forbidden; length possibly restricted

Keywords:
- identifiers with a predefined meaning
- for representing control structures (`while`), operators (`and`), ...

Numerals: certain sequences of digits, `+`, `-`, `.`, letters (for exponent and hexadecimal representation)

Special symbols:
- one special character, e.g., `+`, `*`, `<`, `(`, `;`, ...
- ... or two or more special characters, e.g., `:=`, `**`, `<=`, ...
- each makes up a symbol class (plus, gets, ...)
- ... or several combined into one class (arithOp)

Important Symbol Classes

Identifiers:
- for naming variables, constants, types, procedures, classes, ...
- usually a sequence of letters and digits (and possibly special symbols), starting with a letter
- keywords usually forbidden; length possibly restricted

Keywords:
- identifiers with a predefined meaning
- for representing control structures (`while`), operators (`and`), ...

Numerals: certain sequences of digits, `+`, `-`, `.`, letters (for exponent and hexadecimal representation)

Special symbols:
- one special character, e.g., `+`, `*`, `<`, `(`, `;`, ...
- ... or two or more special characters, e.g., `:=`, `**`, `<=`, ...
- each makes up a symbol class (plus, gets, ...)
- ... or several combined into one class (arithOp)

White spaces:
- blanks, tabs, linebreaks, ...
- usually for separating symbols (exception: FORTRAN)

Specification and Implementation of Scanners

Representation of symbols: symbol = (token, attribute)

Token: (binary) denotation of symbol class (id, gets, plus, ...)

Attribute: additional information required in later compilation phases

- reference to symbol table,
- value of numeral,
- concrete arithmetic/relational/Boolean operator, ...
- usually unused for singleton symbol classes

Specification and Implementation of Scanners

Representation of symbols: symbol = (token, attribute)

Token: (binary) denotation of symbol class (id, gets, plus, ...)

Attribute: additional information required in later compilation phases

- reference to symbol table,
- value of numeral,
- concrete arithmetic/relational/Boolean operator, ...
- usually unused for singleton symbol classes

Observation: symbol classes are regular sets

$\Longrightarrow$

- specification by regular expressions
- recognition by finite automata
- enables automatic generation of scanners (`[f]lex`)

Outline

Regular Expressions I

Definition 2.2 (Syntax of regular expressions)

Given some alphabet Ω, the set of regular expressions over Ω, RE_Ω, is the least set with

- $\Lambda \in RE_\Omega$,
- $\Omega \subseteq RE_\Omega$, and
- whenever $\alpha, \beta \in RE_\Omega$, also $\alpha \mid \beta, \alpha \cdot \beta, \alpha^* \in RE_\Omega$.

Regular Expressions I

Definition 2.2 (Syntax of regular expressions)

Given some alphabet Ω, the set of regular expressions over Ω, RE_Ω, is the least set with

- $\Lambda \in RE_\Omega$,
- $\Omega \subseteq RE_\Omega$, and
- whenever $\alpha, \beta \in RE_\Omega$, also $\alpha \mid \beta, \alpha \cdot \beta, \alpha^* \in RE_\Omega$.

Remarks:

- abbreviations: $\alpha^+ := \alpha \cdot \alpha^*$, $\varepsilon := \Lambda^*$
- $\alpha \cdot \beta$ often written as $\alpha\beta$
- * binds stronger than $\cdot$, $\cdot$ binds stronger than $\mid$
  (i.e., $a \mid b \cdot c^* := a \mid (b \cdot (c^*))$)

Regular Expressions II

Regular expressions specify regular languages:

Definition 2.3 (Semantics of regular expressions)

The semantics of a regular expression is defined by the mapping

$$[\![.]\!] : RE_\Omega \to 2^{\Omega^*} \text{ where}$$

$$\begin{aligned} [\![\Lambda]\!] &:= \emptyset \\ [\![a]\!] &:= \{a\} \\ [\![\alpha \mid \beta]\!] &:= [\![\alpha]\!] \cup [\![\beta]\!] \\ [\![\alpha \cdot \beta]\!] &:= [\![\alpha]\!] \cdot [\![\beta]\!] \\ [\![\alpha^*]\!] &:= [\![\alpha]\!]^* \end{aligned}$$

Regular Expressions II

Regular expressions specify regular languages:

Definition 2.3 (Semantics of regular expressions)

The semantics of a regular expression is defined by the mapping

$$[\![.]\!] : RE_\Omega \to 2^{\Omega^*} \text{ where}$$

$$\begin{aligned} [\![\Lambda]\!] &:= \emptyset \\ [\![a]\!] &:= \{a\} \\ [\![\alpha \mid \beta]\!] &:= [\![\alpha]\!] \cup [\![\beta]\!] \\ [\![\alpha \cdot \beta]\!] &:= [\![\alpha]\!] \cdot [\![\beta]\!] \\ [\![\alpha^*]\!] &:= [\![\alpha]\!]^* \end{aligned}$$

Remarks: for formal languages $L, M \subseteq \Omega^*$, we have

- $L \cdot M := \{vw \mid v \in L, w \in M\}$
- $L^* := \bigcup_{n=0}^{\infty} L^n$ where $L^0 := \{\varepsilon\}$ and $L^{n+1} := L \cdot L^n$
  $(\implies L^* = \{w_1 w_2 \dots w_n \mid n \in \mathbb{N}, w_i \in L\}$ and $\varepsilon \in L^*)$
- $[\![\varepsilon]\!] = [\![\Lambda^*]\!] = [\![\Lambda]\!]^* = \emptyset^* = \{\varepsilon\}$

Regular Expressions III

Example 2.4

1. A keyword: `begin`

Regular Expressions III

Example 2.4

1. A keyword: `begin`
2. Identifiers: (`a` | ... | `z`)(`a` | ... | `z` | `0` | ... | `9`)*

Regular Expressions III

Example 2.4

1. A keyword: `begin`
2. Identifiers: $(\texttt{a} \mid \ldots \mid \texttt{z})(\texttt{a} \mid \ldots \mid \texttt{z} \mid \texttt{0} \mid \ldots \mid \texttt{9})^*$
3. Integer numbers: $(\texttt{0} \mid \ldots \mid \texttt{9})^+$

Regular Expressions III

Example 2.4

1. A keyword: `begin`
2. Identifiers: $(\texttt{a} \mid \ldots \mid \texttt{z})(\texttt{a} \mid \ldots \mid \texttt{z} \mid \texttt{0} \mid \ldots \mid \texttt{9})^*$
3. Integer numbers: $(\texttt{0} \mid \ldots \mid \texttt{9})^+$
4. Real numbers:
   $((\texttt{0} \mid \ldots \mid \texttt{9})^+ \texttt{.} (\texttt{0} \mid \ldots \mid \texttt{9})^*) \mid ((\texttt{0} \mid \ldots \mid \texttt{9})^* \texttt{.} (\texttt{0} \mid \ldots \mid \texttt{9})^+)$