CS 230
Programming Languages

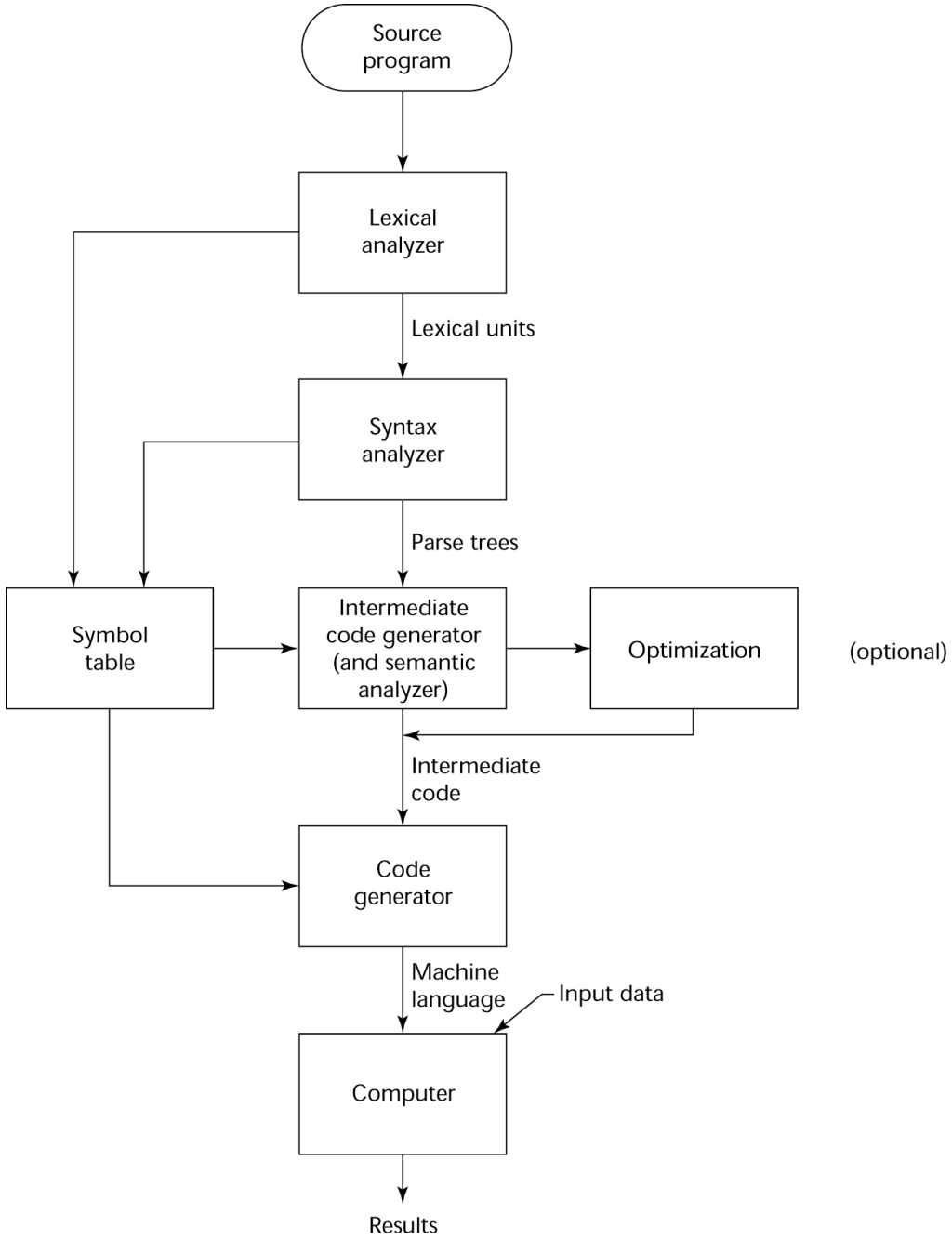
09 / 19 / 2022

Instructor: Michael Eckmann

Today's Topics

- Questions? / Comments?
- Larger grammar example
- Start Parsers (Chapter 4) - Lexical Analyzer

Let's Revisit the Compilation Process



Compilation Process

- Lexical analyzer – groups the code into variable names, reserved words, punctuation, etc.
- Syntax analyzer – constructs parse trees to determine if the lexical units in order are syntactically correct
- Both create items in the **symbol table** to refer to these units
- Intermediate code generator (and semantic analyzer) generates assembly-like code from the symbol table and parse trees.

Chapter 3 so far

- Generator / recognizer
- Is a grammar a generator or recognizer, do you think?
- CFG and BNF are what?
- CFG and BNF are used for what?
- Derivation for a sentence (program)
- Parse tree for a sentence (program)
- What is ambiguity and why is it bad?

Extended BNF

- Meaning of
  - [ ]
  - { }
  - (… | … | … | …)

A more complex grammar

Here's the example sentence we generated in mini-pascal last time:

```
program abc1d ;

    begin
        ad1 = b ;
        if c then
            write ( d )
    end
.
```

A more complex grammar

- Let's in our mind sort of create a parser from this EBNF description and use that to determine if some programs are syntactically correct.

Lexical & Syntax Analysis

- The syntax analysis portion of a language processor nearly always consists of two parts:
  - A low-level part called a **lexical analyzer** (mathematically, a finite automaton – to be defined shortly - based on a regular grammar)
  - A high-level part called a syntax analyzer, or **parser** (mathematically, a push-down automaton based on a context-free grammar, or BNF)
    - The parser can be based directly on the BNF
    - Parsers based on BNF are easy to maintain

Lexical & Syntax Analysis

- Lexical and syntax analysis are separated because of
  - Simplicity - less complex approaches can be used for lexical analysis; separating them simplifies the parser
  - Efficiency - separation allows optimization of the lexical analyzer
  - Portability - parts of the lexical analyzer may not be portable, but the parser always is portable

Lexical Analysis

- Lexical analysis matches patterns and it is a front-end to the parser
- Identifies substrings of the source program that belong together - lexemes
  - Lexemes match a character pattern, which is associated with a token
  - Recall examples of lexemes and tokens

Lexemes and tokens

```
idx = 42 +
count;
```

Lexemes	*Tokens*
idx	identifier
=	equal_sign
42	int_literal
+	plus_op
Count	identifier
;	semicolon

Lexical Analysis

- The lexical analyzer is usually a function that is called by the parser when it needs the next token
- Three approaches to building a lexical analyzer:
  - Write a formal description of the tokens and use a software tool (e.g. lex) that constructs table-driven lexical analyzers given such a description
  - Design a state diagram that describes the tokens and write a program that implements the state diagram
  - Design a state diagram that describes the tokens and hand-construct a table-driven implementation of the state diagram