CSE302: Compiler Design

Instructor: Dr. Liang Cheng
Department of Computer Science and Engineering
P.C. Rossin College of Engineering & Applied Science
Lehigh University

January 25, 2007

Outline

- Recap
  - Introduction (Section 2.1)
  - Syntax definition (Section 2.2)
  - Parsing (Section 2.4)
  - Syntax directed translation (Section 2.3)
- A simple syntax-directed translator (Chapter 2)
  - A translator for simple expressions (Section 2.5)
  - Lexical analysis (Section 2.6)
- Summary and homework

Translation Schemes

- We used semantic rules as a translation scheme
- Now we use semantic actions as a translation scheme to get the same translation result

- Syntax-directed definition for a BNF grammar
  - Associate each grammar symbol (terminals and non-terminals) with a set of attribute
    - Type information for type checking/conversion
    - Notation representation for notation translation
  - Attach a semantic rule or add program fragment to each production in a grammar
    - Computing the values of the attributes associated with the symbols in the production

New BNF Productions and Parse Trees Using Semantic Actions

- Actions are added in the productions

$$
\begin{array}{lcll}
expr & \rightarrow & expr_1 + term & \{\text{print}('+')\} \\
expr & \rightarrow & expr_1 - term & \{\text{print}('-')\} \\
expr & \rightarrow & term & \\
term & \rightarrow & 0 & \{\text{print}('0')\} \\
term & \rightarrow & 1 & \{\text{print}('1')\} \\
 & & \cdots & \\
term & \rightarrow & 9 & \{\text{print}('9')\}
\end{array}
$$

- When drawing a parse tree
  - Indicate an action by constructing an extra child for it, connected by a dashed line to the node that corresponds to the head of the production
- Draw a new parse tree for 9-5+2 with the semantic actions

Actions Translating 9-5+2 into 95-2+

- Perform a postorder depth-first traversal of the parse tree

Outline

- Recap
- A simple syntax-directed translator (Chapter 2)
  - Syntax directed translation (Section 2.3)
  - A translator for simple expressions (Section 2.5)
  - Lexical analysis (Section 2.6)
- Summary and homework

What Can Be Done Now?

- Define language syntax using BNF grammar
- Parsing to detect syntax errors
  - Syntax analysis
- Syntax-directed translation
  - How about we integrate them together?

Integrate What We Have Learned

- Design a BNF grammar for a language that could express a one-digit number, additions and/or subtractions of multiple one-digit numbers in an infix form
- Implement a compiler translating the expression in the above-language to a postfix form

Demo

- Figure 2.27

You should now be able to ...

- Define language syntax using BNF grammar
- Parse sentences and detect syntax errors
- Use syntax-directed definition to perform language translation

Outline

- Recap
- A simple syntax-directed translator (Chapter 2)
  - Syntax directed translation (Section 2.3)
  - A translator for simple expressions (Section 2.5)
  - Lexical analysis (Section 2.6)
- Summary and homework

Lexical Analyzer

- Read input characters and group them into tokens
  - A token object carries attribute values
  - A sequence of input characters that comprises a single token is called a lexeme
- Study the lexical analysis by examples
  - Remove white space
  - Handle constants
  - Recognize keywords and identifiers
  - A lexical analyzer (Appendix A)

Remove White Space

- for (; ; *peek*=next input character) {

      if (*peek* is a blank or a tab) do nothing;
      else if (*peek* is a newline) line=line+1;
      else break;

  }

Handle Constants

- Tokens represent constants as <**num**, **num**.*value*>
- **if** (*peek* holds a digit) {

```
value = 0;
do {
    value = value * 10 + integer value of digit peek;
    peek = next input character;
} while (peek holds a digit);
return token <num,value>;
}
```

Recognize Keywords and Identifiers

- Study the case that keywords are reserved
- Solution: using a table to hold character strings
  - Achieve single representation for ids and keywords
  - Differentiate keywords from ids
- For example, seeds a hashtable with keywords

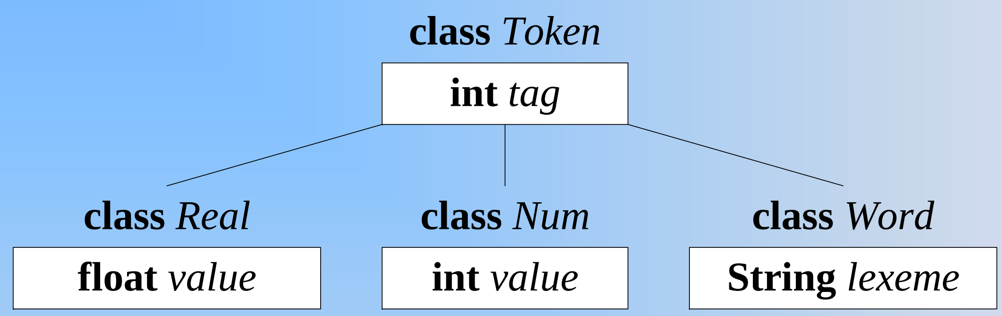
  Hashtable *words* = **new** Hashtable();

  ...

  if (*peek* holds a letter) {
     collect letter and/or digits into a buffer *b*;
     s = string formed from the characters in *b*;
     w = token returned by *words*.*get*(*s*);
     if (*w* != **null**) **return** *w*;
     else {
        enter the key-value pair (*s*, <**id**,*s*>) into *words*;
        return token <**id**,*s*>;
     }
  }

Lexical Analyzer (Appendix A)

- Data structure of tokens



- Tag.java: constants for tokens
- Token.java: tokens' data structure
- Num.java: tokens of integer numbers
- Real.java: tokens of floating-point numbers
- Word.java: tokens of reserved words, ids, and composite tokens like &&, ||, ==, etc.
- Lexer.java: method scan() removes white space and recognizes numbers, ids, and reserved words

Outline

- Recap
- A simple syntax-directed translator (Chapter 2)
  - Syntax directed translation (Section 2.3)
  - A translator for simple expressions (Section 2.5)
  - Lexical analysis (Section 2.6)
- Summary and homework

You should now be able to ...

- Implement a simple language
- Understand lexical analysis implementation

Implement A Simple Language

- Define language syntax using BNF grammar
- Parse sentences and detect syntax errors
- Use syntax-directed definition to perform language translation

You should now be able to ...

- Implement a simple language
- Understand lexical analysis implementation
  - Remove white space
  - Handle constants
  - Recognize keywords and identifiers
  - Understand the lexer package in Appendix A

Homework (Due on 01/29 at 11:55 PM)

- 2.3. (20 points)
  - (a) Define a BNF grammar for a language that could express a one-digit number, additions and/or subtractions of multiple one-digit numbers in a prefix notation (e.g., -xy is the prefix notation for x-y and the prefix notation of an infix notation 4+5-2+6 is +-+4526); (5 pts)
  - (b) Construct a syntax-directed translation scheme that translates the above-defined one-digit arithmetic expressions from prefix notation into infix notation; (5 pts)
  - (c) Implement an executable and correct program to perform the above-mentioned translation. (10 pts)

Homework (Due on 01/29 at 11:55 PM)

- 2.1. (10 points) Rewrite the following BNF to give + precedence over * and force + to be right associative.
  - <assign> → <id> = <expr>
  - <id> → A | B | C
  - <expr> → <expr> + <term> | <term>
  - <term> → <term> * <factor> | <factor>
  - <factor> → (<expr>) | <id>
- 2.2. (10 points) Implement a correct and executable recursive-descent parser based on the pseudo code illustrated in 01/23 lecture:
  - <expr> → <term> <rest>
  - <rest> → + <term> <rest> | - <term> <rest> | ε
  - <term> → 0 | 1 | 2 | ... | 9

Reading Assignment

- Sections 2.3, 2.5 and 2.6
- For next Tuesday class
  - Sections 2.7 and 2.8