450 COMPILERS

COMPUTER SCIENCE

News

- Introducing the B3 JIT Compiler
  - https://webkit.org/blog/5852/introducing-the-b3-jit-compiler/

Administrivia

- Lab 02
  - Due Thursday
- Lab 03
  - Available Now
  - Due in 1 Week,

Review

- Context-Free Grammar (Grammar)
  - Called a Production
  - Four Components
    - 1) Set of tokens, terminals
    - 2) A set of nonterminals
    - 3) A set of Productions
    - 4) Nonterminal Start Symbol

Terminal vs Non-Terminal

For instance, the following represents an integer (which may be signed) expressed in a variant of Backus–Naur form:

```
<digit> ::= '0' | '1' | '2' | '3' | '4' | '5' | '6' | '7' | '8' | '9'
<integer> ::= ['-'] <digit> {<digit>}
```

In this example, the symbols (-,0,1,2,3,4,5,6,7,8,9) are terminal symbols and <digit> and <integer> are nonterminal symbols.

Note: This example supports strings with leading zeroes like "0056" or "0000", as well as negative zero strings like "-0" and "-00000".

Syntax-Directed Translation

- The attributes associated with the constructs needed for a giving programming language
- Syntax-Directed definitions are used for specifying translations for programming language constructs

Review continued

- Syntax-Dedicated Definitions
  - Uses grammar to specify syntactic structure of the input
  - Grammar symbols
    - Set of Attributes
  - Production
    - Set of semantic rules

Syntax-Directed Definitions

- Uses a context-free grammar to specify the syntactic structure of the input
- Each grammar symbol, it associates a set of attributes, and with each production, a set of semantic rules for computing values of the attributes associated with the symbols appearing in that production
- The grammar and the set of semantic rules constitute the syntax directed definitions

Translator for Simple Expressions

- An abstract syntax tree (AST) is a parse tree where the interior nodes are programming constructs rather than nonterminals. The interior nodes in a parse tree are nonterminals.

Symbol Table

- Either the scanner or the parser must create an entry in the symbol table for each identifiers in the program. The scanner creates the entry in the symbol table for the current token if it can. The parser must do so in cases where the scanner is not smart enough to so so - such as those that involve scope. Details of symbol table management in later chapters.

Lexicon Analysis

- What do we want to do? Ex.
  - if(i == j)
  - z = 0;
  - else
  - z = 1;
- The input is just a string of characters:

  \tif(i == j)\n\t\tz = 0;\n\telse\n\t\tz=1;
- Goal: Partition the input string into substrings
  - Where the subrstrings are tokens

What's a Token?

- A syntactic category
  - In English:
    - Noun, verb, adjective, …
  - In Programming Languages:
    - Identifier, Integer, Keyword, Whitespace, ...

Tokens

- Tokens correspond to sets of strings

- Identifier: strings of letters or digits, starting with a letter
- Integer: a non-empty string of digits
- Keyword: “else” or “if” or “switch”
- Whitespace: a non-empty sequence of blanks, newlines, and tabs

What are Tokens for?

- Classify program substrings according to role
- Output of lexical analysis is a stream of tokens…
- ...Which is input to the parser
- Parser relies on token distinctions
  - An identifier is treadted differently than a keyword

Designing a Lexical Analyzer

- Define a finite set of tokens
  - Tokens describe all items of interest
  - Choice of tokens depends on language, design of parser

Example

- Recall
  - \tif(i == j)\n\t\tz = 0;\n\telse\n\t\tz=1;
- Useful tokens for this expression:
  - Integer, Keyword, Identifier, Whitespace,
  - (,),=,;

Designing a Lexical Analyzer: Step 2

- Describe which strings belong to each token
- Recall:
  - Identifiers: strings of letters or digits, starting with a letter
  - Integer: a non-empty string of digits
  - Keyword: “else” or “if” or “switch”
  - Whitespace: a non-empty sequence of blanks, newlines, and tabs

Lexical Analyzer: Implementation

- An implementation must do two things:
  - 1. Recognize substrings corresponding to tokens
  - 2. Return the value or lexeme of the token
    - The lexeme is the substring

Example

- Recall
- \tif(i == j)\n\t\tz = 0;\n\telse\n\t\tz=1;

Lexical Analyzer: Implementation

- The lexer usually discards “uninteresting” tokens that don’t contribute to parsing
- Examples: Whitespace, Comments

True Crimes of Lexical Analysis

- Is it as easy as it sounds?
- Not quite!
- Look at some history …

Lexical Analysis in FORTRAN

- FORTRAN rule: Whitespace is insignificant
- Example: VAR1 is the same as VA R1
- A terrible design

Lexical Analysis in FORTRAN cont.

- Two important points:
  - 1. The goal is to partition the string. This is implemented by reading left-to-right, recognizing one token at a time
  - 2. “Lookahead” may be required to decide where one token ends and the next token begins

Lookahead

- Even our simple example has lookahead issues
  - i vs. if
  - = vs. ==
  - Footnote: FORTAN Whitespace rule motivated by inaccuracy of punch card operators

Lexical Analysis in C++

- Unfortunately, the problems continue today

- C++ template syntax:
  - Foo<Bar>
- C++ stream syntax:
  - cin >> var;
- But there is a conflict with nested templates:
  - Foo<Bar<Bazz>>

Seque

- Regular expressions are simple, almost trivial
  - But they are useful

- Reconsider informal token descriptions

Example: Keyword

- Keyword: “else” or “if” or “switch”

Example: Integers

- Integer: a non-empty strings of digits
  - Digit = ‘0’ + ‘1’ + ‘2’ + …
  - Integer = digit digit*

Example: Identifier

- Letter = ‘A’ + .. + ‘Z’ + ‘a’ + .. ‘z’
- Identifier = letter(letter + digit)*

Example: Whitespace

- Whitespace: a non-empty sequence of blanks, newlines, and tabs
- (‘ ‘ + ‘\n’ + ‘\t’

Example: Phone Numbers

- Consider (650)-732-3232

- Σ = digits U {-,(,)}
- Exchange = digit^3
- Phone = digit^4
- Area = digit^3

- Phone_number = ‘(‘ area ‘)-’ exchange ‘-’ phone

Example: Email Addresses

- Consider anyone@cs.csub.edu
- Σ = letters U {.,@}
- Name = letter+
- Address = name '@' name '.' name '.' name