BITS Pilani
Pilani Campus

Lexical Analysis

Dr. Shashank Gupta
Assistant Professor
Department of Computer Science and Information Systems

Today's Agenda

- Challenges in Development of Lexical Analysis.
- Specification and Implementation Issues.

What is Lexical Analysis (LA)?

LA

I/P is any high-level language and O/P is a sequence of tokens.

It generally cleans the code: strips off blanks, tabs, newlines and comments.

Keeps track of the line numbers for associated error messages.

Lexical Analysis

Modelling of LA will be done using Regular Expression.

Implementation of LA will be done using Finite State Machine.

Tokens, Patterns and Lexemes

Token

- A string of characters which logically belong together. *(e.g., Keywords, Number, Identifier, String, etc.)*

Pattern

- The set of strings for which the same token is produced. *L. (L+D)**

Lexeme

- The sequence of characters matched by a pattern to form the corresponding token. *(200.43, temp, while, "This is my lexer")*

Lexical Analysis

Transform numbers to token 'num' and pass the lexeme as its corresponding attribute.

- Integer `43` becomes `<num, 43>`

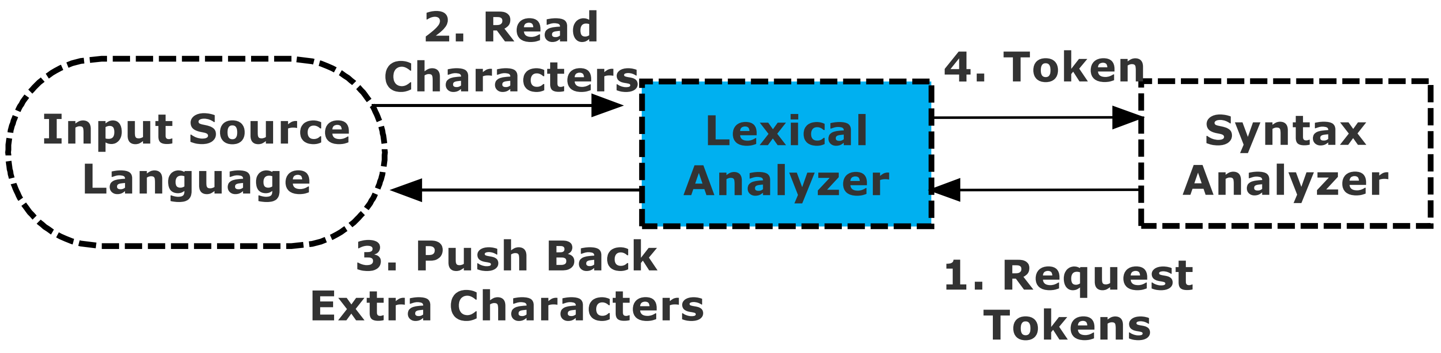
In addition, identify the identifiers and keywords appropriately.

- `temp = else + flag` will be tokenized as follows:
- `id assgnop keyword addop id`

Proper tokenization of keywords and identifiers must be done.

Since, the rules for recognizing the identifiers and keywords are similar.

Interface of Lexical Analysis to other Translation Phases



- Push back operation is necessary due to lookahead for differentiating operators like = and ==.
- Must be implemented through some buffer.

Small Development of a Lexical Analyzer

- Permits numbers and arithmetic operators in an expression.
- Return token and attributes to the syntax analyser.
- A global variable *lexeme* is set to the value of the number.
- Design requires that a finite set of tokens need to be defined and also describe strings belonging to each token.

Small Lexer

```
#include <stdio.h>

int lexeme ;
int lexical_analyzer() {
        int c;          // Character
        while (1) {
        c = getchar ();
        if (c = = ' ' || c = = '\t');  // Discard White Spaces
        else if (isdigit (c) ) {
                        lexeme = c - '0' ;
                         c = getchar ();
                         while (isdigit(c)) {
                                    lexeme = lexeme * 10 + c - '0' ;
                                     c = getchar();}
                      ungetc(c,stdin); // Push back Extra Char
                           return num;}  // Return num Token
        else { lexeme = NULL; return c; } // Return operator
         }
    }
```

Challenges in Lexical Analyzer

Examines lexemes character by character.

Look ahead pointer recognizes

- What kind of token to read?
- When the current token finishes?

First character cannot determine what kind of token we are going to read. (e.g. = or ==)

Lookahead pointer needs to be taken for determining the appropriate token.

Interface of Symbol Table to Lexical Analysis

Injects the information for related translation phases of compiler.

- Lexical analyzer needs the information related to the class of token and the associated lexeme with it.

Operations of the Symbol Table.

- **`Insert(s, t)`**: save lexeme `s` and token t and return pointer.
- **`Lookup(s)`**: return index of entry for lexeme `s` or `'0'` if `s` is not found.