Table of Contents

- First we should know what is Lexical Analyzer?
- Lexical analyzers are divided into a cascade of two processes:
  - 1. Scanning
  - 2. Lexical Analysis
- What is Token ?
  - Lexem Token example
- What is Pattern ?
- What is Lexeme ?

Input Buffering

- Three general approaches for the implementation of a lexical analyzer
- What is Buffer Pairs ?
- What is Sentinels ?
- Disadvantages
- Advantages

First we should know what is Lexical Analyzer?

The main task of the lexical analyzer is to read the input characters of the source program group them into lexemes and produce as output a sequence of tokens for each lexeme in the source program.

When the lexical analyzer discovers a lexeme constituting an identifier, it needs to enter that lexeme into the symbol table.

The lexical analyzer not only identifies the lexemes but also pre-processes the source text like removing comments, white spaces, etc.

Lexical analyzers are divided into a cascade of two processes:

1. Scanning

It consists of simple processes that do not require the tokenization of the input such as deletion of comments, compaction of consecutive white space characters into one.

2. Lexical Analysis

This is the more complex portion where the scanner produces sequence of tokens as output.

What is Token ?

A Token is pair consisting of a token name and an optional attribute value.

The token name is an abstract symbol representing the kind of lexical unit.

Lexem Token example

Lexeme	Token
=	EQUAL_OP
*	MULT_OP
,	COMMA
(	LEFT_PAREN

What is Pattern ?

A pattern is a description of the form that the lexemes of a token may take. In case of a keyword as a token the pattern is just a sequence of characters that form the keyword.

What is Lexeme ?

A Lexeme is a sequence of characters in the source program that matches the pattern for a token and is identified by the lexical analyzer as an instance of that token.

INPUT BUFFERING

To ensure that a right lexeme is found, one or more characters have to be looked up beyond the next lexeme.

Hence a two-buffer scheme is introduced to handle large lookaheads safely.

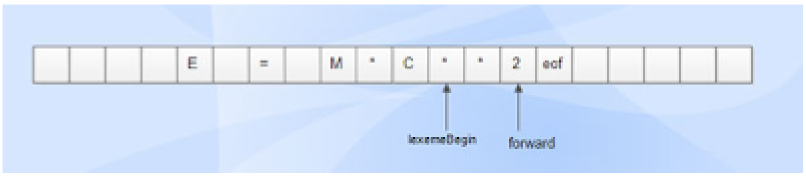
Techniques for speeding up the process of lexical analyzer such as the use of sentinels to mark the buffer end have been adopted.

Three general approaches for the implementation of a lexical analyzer

1. By using a lexical-analyzer generator: In this, the generator provides routines for reading and buffering the input.
2. By writing the lexical analyzer in a conventional systems-programming language, using I/O facilities of that language to read the input.
3. By writing the lexical analyzer in assembly language and explicitly managing the reading of input.

What is Buffer Pairs ?

A specialized buffering techniques0 used to reduce the amount of overhead, which is required to process an input character in moving characters.



Buffer pairs

- Consists of two buffers, each consists of N-character size which are reloaded alternatively.
- Two pointers lexemeBegin and forward are maintained.
- Lexeme Begin points to the beginning of the current lexeme which is yet to be found.
- Forward scans ahead until a match for a pattern is found.
- Once a lexeme is found, lexeme begin is set to the character immediately after the lexeme which is just found and forward is set to the character at its right end.
- Current lexeme is the set of characters between two pointers.

What is Sentinels ?

Sentinels is used to make a check, each time when the forward pointer is moved, a check is done to ensure that one half of the buffer has not moved off. If it is done, then the other half must be reloaded.

Therefore the ends of the buffer halves require two tests for each advance of the forward pointer. Test 1: For end of buffer. Test 2: To determine what character is read. The usage of sentinel reduces the two tests to one by extending each buffer half to hold a sentinel character at the end.

The sentinel is a special character that cannot be part of the source program. *(*eof character is used as sentinel).

Disadvantages

- This scheme works well most of the time, but the amount of lookahead is limited.
- This limited lookahead may make it impossible to recognize tokens in situations where the distance that the forward pointer must travel is more than the length of the buffer.

Advantages

- Most of the time, It performs only one test to see whether forward pointer points to an eof.
- Only when it reaches the end of the buffer half or eof, it performs more tests.
- Since N input characters are encountered between eofs, the average number of tests per input character is very close to 1.