CS403: Programming Languages

Notes on Lexical Analyisis

Introduction

The first thing one needs to do in implementing a language is to figure what the words are in that language. For example, consider teaching someone to read English. You might start out by explaining the grammar of English and beginning a discourse on nouns and verbs and so on. But there is a more fundamental level that needs explanation first and that is the morphology of the language.
Morphology is the study of shapes and, surprising though it may seem, all languages have shape. This is why it is easy for non-French speakers to recognize spoken and written French (apologies to E. A. Poe and The Murders in the Rue Morgue). English prose is composed of sentences and we know when a sentence ends and another begins when we see some special shapes (punctuation). We also know that a sentence is composed of words and we know that blank spaces separate the words in a sentence. Finally, we know that words can be categorized as to their parts of speech and we learn to recognize certain words as nouns, verbs, adjectives, and other arcane types. All these rules must be understood before one can understand the grammar of a language.

Lexical analysis

Lexical analysis is the process of breaking up a language stream into its words and categorizing those words as to their types. For example, were we to perform lexical analysis on the sentence:

```
The large cow drank the cool water.
```

we might produce the following output:

```
ARTICLE The
ADJECTIVE large
NOUN cow
VERB drank
ARTICLE the
ADJECTIVE cool
NOUN water
PERIOD
```

where the part of speech precedes the actual word. These parts of speech / word pairs are known as lexemes. Formally, a lexical analyzer transforms a stream of characters into a stream of lexemes. The lexeme for a period needs no associated 'word' since there is only one kind of period.
Suppose we wish to implement a simple, whitespace delimited, calculator language. Here are some example sentences in the language:

```
define a;
define zeta;
a = 64 * 2;
zeta = log(a*a,2);
print("the value of zeta is ", zeta);
```

The parts of speech in this language are keywords, variables, punctuation, operators, and numbers (in order of appearance). A scanner is a program that lexically analyzes a character stream and outputs a stream of lexemes. It uses a function named lex, which is responsible for determining the next lexeme in the input. Here is an example scanner:

```
function scanner()
    {
    var lexeme;

    lexeme = lex();
    while (lexeme.type != END)
        {
        lexeme.display();
        newline();
```

```
            lexeme = lex();
            }
        }
```

The *lex* function grabs a character at a time from the input and when enough characters have been read, returns a lexeme. Here is an example *lex* (the implementations of *skipWhiteSpace*, *lexNumber*, *lexVariable*, and *lexString* and the implementations of the *Lexeme* and *Input* classes are left up to the student):

```
    function lex()
        {
        define ch;

        skipWhiteSpace();

        ch = Input.getCharacter();

        if (Input.failed) return new Lexeme(END);

        switch(ch)
            {
            // single character tokens
            case '(':
                return new Lexeme(OPAREN);
            case ')':
                return new Lexeme(CPAREN);
            case ',':
                return new Lexeme(COMMA);
            case '+':
                return new Lexeme(PLUS);
            case '*':
                return new Lexeme(TIMES);
            case '-':
                return new Lexeme(MINUS);
            case '/':
                return new Lexeme(DIVIDES);
            default:
                // multi-character tokens
                // (only numbers, variables/keywords, and strings)
                if (isdigit(ch))
                    {
                    Input.pushback(ch);
                    return lexNumber();
                    }
                else if (isalpha(ch))
                    {
                    Input.pushback(ch);
                    return lexVariable();  //and keywords!
                    }
                else if (ch == '\"')
                    {
                    Input.pushback(ch);
                    return lexString();
                    }
            }

        return new Lexeme(BAD_CHARACTER, ch);
        }
```

The tokens `END`, `PLUS`, `MINUS`, `etc.` are constants that fill out the type component of a lexeme.
Given the input:

```
    define zeta;
    zeta = 5;
    print("zeta is ", zeta);
```

the scanner might print out something like:

```
DEFINE
VARIABLE zeta
SEMICOLON
VARIABLE zeta
ASSIGN
NUMBER 5
SEMICOLON
PRINT
OPAREN
STRING "zeta is "
COMMA
VARIABLE zeta
CPAREN
SEMICOLON
END
```

Given the input:

```
define zeta#
zeta  = 5;
print("zeta is ", zeta);
```

the scanner might generate:

```
DEFINE
VARIABLE zeta
BAD_CHARACTER '#'
VARIABLE zeta
...
```

assuming an octothorpe is not legal for a variable name.
How does the function *lex* decide the type of a particular token? Through its morphology. A variable, for example may have the following shape:

```
VARIABLE: a letter followed by any number of letters or digits
```

An integer has more restrictive shape:

```
INTEGER : a digit followed by any number of digits
```

Often, in a language, a keyword has the same general shape as a variable, yet its specific shape is different. For example, the keyword define has the following exact shape:

```
DEFINE : a 'd' followed by 'e', 'f', 'i', 'n', 'e' in that order
```

In the implementation of lex given above, the function *lexVariable* is charged with looking for the specific shapes of the keywords. If it finds one, it returns a keyword lexeme instead of a variable lexeme.