MAHARISHI UNIVERSITY OF MANAGEMENT

COMPUTER SCIENCE DEPARTMENT

COMP 440 Compilers (DE)

Programming Assignment 1

CP Scanner

Your next project step will be to write a scanner module for the programming language C-Plus (CP). You will use SableCC, a scanner generation tool similar to the Unix tool lex (described in the lecture). SableCC will also be used to generate the parser. Future assignments will involve a CP parser, type checker, and interpreter.

The CP Scanner

The CP scanner will be generated using SableCC. Your main task will be to create the file CP_lexer.scc, the input to SableCC. CP_lexer.scc specifies the regular expression patterns for all the CP tokens.

The first step will be to download the SableCC tool and documentation from the course web site. Read chapters three and four, pp. 21-40, of the SableCC documentation (thesis). Chapter 3 gives an overview of SableCC, and Chapter 4 explains the syntax and use of the scanner generator part of SableCC.

CP Tokens

CP contains the following classes of tokens:

Reserved words:

bool break char const continue class do else false float if int

new null private protected public return static string struct

this true while void

Identifiers: An identifier is a sequence of one or more letters, digits andunderscore (_) characters.

However, identifiers must start with a letter or underscore, and reserved words are excluded. The token for recognizing identifiers must appear after the reserved words (make sure you know why).

Integer Literals: An integer literal is a sequence of one or more digits (0| 1 | ... | 9). However, only the single digit 0 can begin with 0; all other integer literals must start with (1 | 2 | ... | 9).

String Literals. A string literal is any sequence of printable characters,delimited by double quotes. A double quote within the text of a string must be escaped (i.e. \Ã¢â,¬Â) to avoid being misinterpreted as the end of the string. Tabs and newlines etc. within a string are escaped as usual(i.e., \n is a newline, \t is a tab, \r is a carriage return, \b is a backspace, and \f is a form feed). Backslashes within a string must also be escaped (as \\). Strings may not cross line boundaries.

Character Literals: A character literal is any printable character, enclosed within single quotes. A single quote within a character literal must be escaped (i.e. \Ã¢â,¬â„¢) to avoid being misinterpreted as the end of the literal. Tabs and newlines etc. must be escaped (e.g., '\n' is a newline, '\t' is a tab, \r is a carriage return, \b is a backspace, and \f is a form feed). A backslash and double quote must also be escaped (as '\\' and \Ã¢â,¬Â respectively).

Floating Point Literals: Floating point literals contain a decimal point (e.g., 3.14159) with an optional integer exponent part (e.g., 0.314159E01). The character Ã¢â,¬Å“eÃ¢â,¬Â that separates the mantissa and exponent parts may be in upper or lower case. In either notation, the decimal point must be either preceded or succeeded by at least one digit. For a positive exponent an optional Ã¢â,¬Ëœ+Ã¢â,¬â„¢ sign may be present. Thus thefollowing are all valid floating point literals: 1.5, .5, 1. , 1.e05, .5E+1, and 1.77E-03. However, .e05 is not a valid float.

Other Tokens: These are miscellaneous one- or two-character symbols representing operators and delimiters.

( ) [ ] = + - * / % == != && || < > <= >= ! . , { } ; :

Comments and white space, as defined below, are not tokens because they are not returned by the scanner. Nevertheless they must be matched (and skipped) when they are encountered.

A Single Line Comment: As in C++ and Java, this comment begins with a pair of slashes and ends at the end of the current line. Its body can include any character other than an end-of-line.

LineComment = // Not(NewLine) * NewLine

A Multi-Line Comment. This comment begins with the pair

/* and ends with the pair */.

Its body can include any character sequence that does not contain the sequence */ (like C, C++, and Java).

White Space. This separates tokens; otherwise it is ignored.

WhiteSpace = (Blank | Tab | NewLine) + Any character that cannot be scanned as part of a valid token, comment, or white space is illegal and will automatically generate an error message.

Considerations/Requirements

- Because reserved words Ã¢â‚¬Å“look likeÃ¢â‚¬Â identifiers, you must be careful not to mis-scan them as identifiers.

- You should include distinct token definitions for each reserved word before your definition of identifiers.

- You should not assume any limit on the length of identifiers.

- You should not assume any limit on the length of input lines that are scanned.

- Although SableCC's regular expression syntax is designed to be very similar to that of scanner generators like Lex, Flex and JLex, it is not identical. If you are familiar with such scanner generators, read the manual carefully and note the differences. One notable difference is that token names cannot contain uppercase letters! Another difference is that regular expressions defining a token occur between single

  quotes (Ã¢â‚¬Ëœ).

Older versions of java and javac (prior to JDK 1.3) use an environment variable CLASSPATH to define the directories to be searched to find class files stored in libraries; this is not a problem in the latest versions of JDK. If you are using JDK, carefully follow the installation instructions so you donÃ¢â‚¬â„¢t have problems.

To install SableCC, unzip the SableCC tool and follow the installation instructions in the README.html file in the top level directory, sablecc-3.2.

You will have to edit the bin\sablecc.bat file as described in its comment lines (change the path to the sablecc.jar file).

Download the ZIP file lab2.zip from the course web page. After unzipping lab2.zip, you will find a LexerDriver.java file and a skeleton CP_lexer.scc file. LexerDriver.java is a main program for testing your scanner. If SableCC has been properly installed, you can generate a "skeleton scanner" by typing the following at the DOS prompt:

```
sablecc CP_lexer.scc
```

Make sure your CLASSPATH is set up so the compiler can find the lexer and node packages generated by sablecc; then compiling LexerDriver will automatically cause the classes in those packages to be compiled. To test your lexer, type the following at the DOS command prompt:

```
java LexerDriver your_test_file
```

The scanner test program, LexerDriver.java, acts like the test program illustrated below, reading a stream of characters from the command line file and printing out the tokens matched to the standard output, one per line in the following format:

[line, column] TokenClassName: token

For example, if the contents of the command line file is:

```
 class T {
  // hello, this is
  // a test

  const
  Const
  "hello"

  ^

  10;
```

```
The test program should produce:

[1, 1] TKClass: class

[1, 7] TIdentifier: T

[1, 9] TLCurly: {

[2, 1] TSLComment: // hello, this is

[3, 1] TSLComment: // a test

[4, 1] TKconst: const

[5, 4] TIdentifier: Const

[6, 1] TStringLit: "hello"

[7, 1] Unknown token: ^

[8, 1] TIntegerLit: 10

[8, 3] TSemicolon: ;
```

Your program may not exactly follow this format, but it should provide an easy to follow listing of scanned tokens.

Your output will be different depending on the names of the tokens you define; the output will also show white space tokens that are not shown here.

Use the test data provided to test your program.

In the test cases there is only one unknown token, the Ã¢â‚¬Ëœ^Ã¢â‚¬â„¢ character as shown in the above example. All other character sequences in the tests should produce valid tokens.

Make sure you run the tests for the keywords after testing identifiers, i.e., run the rw_test after running the others to make sure reserved words are not scanned as identifiers.

I provide the token count for each set of tests in TokenCounts.txt. However, note that the count can be correct, while the recognized tokens are incorrect.

For example, a reserved word may be incorrectly recognized as an identifier unless the identifier token specification is placed after the reserved words in the Tokens section of the CP_lexer.scc file.

What to hand in

Hand in a ZIP file containing your CP_lexer.scc file and a brief README file giving a status report of

your progress.