CPSC 323 Lexical Analyzer for MiniC

Deadlines

Feb. 22 - Regular expressions for each token and for comments, 30%
Mar. 1 - DFA, 30%
Mar. 8 - Lexical analyzer in programming language of your choice, 40%

MiniC

“C is a general-purpose, procedural computer programming language supporting structured programming, lexical variable scope, and recursion, with a static type system. By design, C provides constructs that map efficiently to typical machine instructions. It has found lasting use in applications previously coded in assembly language. Such applications include operating systems and various application software for computer architectures that range from supercomputers to PLCs and embedded systems.”

Facts about C

- C was invented to write an operating system called UNIX.
- C is a successor of the B language which was introduced in the early 1970s
- Today, C is the most widely used and popular Systems Programming Language.

Sample C and MiniC Program

```
int main() {
/* This is a C program. */
printf("Hello world!\n");
return 0;
}
```

MiniC is a subset of C. The above program is legal in C and MiniC.

Semicolons

In MiniC, all statements are concluded with a semicolon. Each individual statement must end with a semicolon. It indicates the end of one logical entity.

Comments

Comments are ignored by the MiniC lexical analyzer. Comments may exist on a single line starting with two //. Example: // This is a single line comment. Comments may exist on multiple lines between /* and */. Example:
/*

This is
a multiline
comment
*/

Any characters that are allowed in the language are allowed in comments.

Identifiers

A MiniC identifier is a name used to identify a variable, function, or any other user defined item. An identifier must start with a letter A to Z, a to z, or an underscore '_'. The remainder of the identifier name can contain characters a-z, A-Z, _, or 0-9. Identifiers names' cannot be reserved words such as return, int, float, etc.

MiniC does not allow punctuation marks such as @, $, and % within identifiers. C is a case-sensitive programming language. Thus, *Person* and *person* are two different identifiers in C.

Reserved Words

Reserved words are words that cannot be used as identifiers (variables, functions, etc.) because they are reserved by the language. These reserved words must be recognized by the lexical analyzer and correspond to tokens of the same name.

break	else	long	switch
bool	extern	register	typedef
case	float	short	union
char	long	sizeof	typedef
const	register	static	void
continue	for	signed	unsigned
default	if	struct	volatile
do	int	return	while
double	goto	_Packed	

Whitespace in MiniC

Whitespace is the term used in MiniC to describe blanks (0x20), tabs (0x09), newline characters (0x0d, 0x0a) and comments. Whitespace separates one part of a statement from another and

enables the compiler to identify where one element in a statement, such as int, ends and the next element begins. Therefore, in the following statement:

```
int age;
```

There must be at least one whitespace character (usually a space) between int and age for the compiler to be able to distinguish between them. On the other hand, in the following statement:

```
fruit = apples + oranges;
```

No whitespace characters are necessary between fruit and =, or between = and apples.

Line Endings in MiniC

Text files created on Windows machines have different line endings than files created on Linux. Windows uses the carriage return and line feed (“\r\n” or 0x0d and 0x0a) as a line ending, while Linux uses just line feed (“\n”, 0x0a). Assume that all MiniC programs may be written on Windows or Linux.

MiniC Programs

MiniC programs contain a function called ‘main’ that serves as the main entry point to a MiniC program and is called by the operating system upon execution. The main function returns an integer, which indicates if the program terminated properly. A function that has a void return type returns no value. MiniC programs may have additional functions defined.

Tokens in MiniC programs may be surrounded by any number of whitespace characters, including none. Identifiers, Booleans, character literals, string literals, and number literals are delimited by whitespace, semicolon, opening and closing parentheses (opening before another token and closing after another token), operations, comparators, and more.

Tokens in MiniC Programs

The following tokens are in MiniC programs and must be recognized by the lexical analyzer. In addition, the lexical analyzer must recognize each of the reserved words. The lexical analyzer will need to recognize comments and whitespace, so they can be ignored by the lexical analyzer.

Token Name	Description	Symbol
addOp	Addition and subtraction binary operators.	+, -
assignOp	Assignment operators.	=, +=, -=, *=, /=
char	Character literal.	‘X’, ‘o’, ‘\\’, ‘\’’, ‘\”’
comma	Comma.	,
comparator	Comparison operator.	<, >, >=, <=, ==, !=

floatLiteral	Floating point value.	3.1459, 314159E-5
incrOp	Increment or decrement operator.	++, --
identifier	Identifier	main, printf, _x, myFunc
intLiteral	Integer value	0, 212, 215u, 30ul
leftBrace	Left brace.	{
logicalNot	Logical unary operator.	!
leftParen	Left parenthesis.	(
multOp	Multiplication and division operators.	*, /
rightBrace	Right brace.	}
rightParen	Right parenthesis.	)
semicolon	Semicolon.	;
string	String.	“Hello\tWorld\n\n”, “hello, dear”, “hell\”o”

Integer Literals

An integer literal may only appear in decimal form (no octal or hexadecimal). An integer literal can have a suffix that is a combination of U and L, for unsigned and long, respectively. Notice that integer literals in MiniC cannot be signed.

Examples:

```
212         /* Legal */
215u        /* Legal */
215Lu       /* Legal */
0           /* Legal */
-212        /* Illegal, MiniC does not allow signed number literals. */
```

Other legal examples:

```
85          /* Int. */
30u         /* Unsigned int. */
45l         /* Long. */
30ul        /* Unsigned long. */
```

Float Literals

A floating point number has an integer part, a decimal point, a fractional part, and an exponent part. While representing floating point literals, you must include the decimal point, the exponent, or both. When giving the decimal point, you must include the integer part, fractional part, or both. The signed exponent is introduced by e or E.

The integer part, fractional part, and exponential part, if given, must be in decimal without the signed or long indicator. While neither integer literals or float literals can be signed in MiniC, the exponent portion of the exponential numbers may be signed.

Examples:

```
3.14159         /* Legal, decimal point only. */
314159E-5       /* Legal, exponent only. */
0.0             /* Legal. */
0.0e0           /* Legal with decimal point and exponent. */
12.             /* Legal. */
.12             /* Legal. */
.12E-5          /* Legal. */
12.E-5          /* Legal. */
510E            /* Illegal. Incomplete exponent. */
210f            /* Illegal. No decimal or exponent. */
.e55            /* Illegal. Missing integer or fraction. */
```

Character Literals

Character literals are enclosed in single quotes, e.g., ‘x’. These can be a plain character (e.g., ‘x’), or an escape sequence (e.g. ‘\t’).

Escape Sequence Table

Escape Sequence	Meaning
\\	
\’	
\”	
\b	Backspace.
\n	Newline.
\r	Carriage return.
\t	Horizontal tab.

While the whitespace characters space (0x20), horizontal tab (0x09), carriage return (0x0d), and line feed (0x0a) may appear in MiniC source files, they cannot be characters in MiniC and must be represented as a character literal or escape sequence character literal.

String Literals

String literals or constants are enclosed in double quotes “. A string optionally contains characters that are similar to character literals: plain characters, escape sequences, and space (0x20). While the whitespace character space (0x20), horizontal tab (0x09), carriage return (0x0d), and line feed (0x0a) may appear in MiniC source files, only the space (0x20) may appear in strings.

Legal Characters

a-z, A-Z, 0-9, periods, commas, underscore, +, -, *, \, <, >, !, =, /, ‘, “, (,), {, }, ;, space (0x20), horizontal tab (0x09), carriage return (0x0d), and line feed (0x0a).

The plain characters:

a-z, A-Z, 0-9, periods, commas, underscore, +, -, *, \, <, >, !, =, /, ‘, “, (,), {, }, ;

Along with whitespace:

space (0x20), horizontal tab (0x09), carriage return (0x0d), and line feed (0x0a)

are the only characters allowed in MiniC programs. An error should be raised by the lexical analyzer if a character not found in this listing appears in MiniC source code.

Sample MiniC Program with Sample Lexical Analyzer Output

Here is a sample MiniC program:

```
int main(int argc) {
      int x, y, z;
      x = 0;
      y=1;
      z = sum(x, y);
      z += mult(x,y);
      return z;
}

int sum(int a, int b) {
      return a+b;
}

int mult(int a, int b) {
      return a * b;
}
```

Output

Token: int	Lexeme: int
Token: identifier	Lexeme: main
Token: leftParen	Lexeme: (
Token: int	Lexeme: int
Token: identifier	Lexeme: argc
Token: rightParen	Lexeme:)
Token: leftBrace	Lexeme: {
Token: int	Lexeme: int
Token: identifier	Lexeme: x
Token: comma	Lexeme: ,
Token: identifier	Lexeme: y
Token: comma	Lexeme: ,
Token: identifier	Lexeme: z
Token: semicolon	Lexeme: ;
Token: identifier	Lexeme: x
Token: assignOp	Lexeme: =
Token: intLiteral	Lexeme: 0
Token: semicolon	Lexeme: ;
Token: identifier	Lexeme: y
Token: assignOp	Lexeme: =
Token: intLiteral	Lexeme: 1
Token: semicolon	Lexeme: ;

And so on…

Assignment

Create a table driven lexical analyzer for MiniC which uses a scanning table for the identification of tokens and to ignore comments. Once an identifier is found, a reserved word lookup table is used to identify these tokens. To create a table for your lexical analyzer, begin by defining regular expressions for each token. Convert the created regular expressions into an NFA that recognizes all tokens, a DFA, and then the scanning table.

The lexical analyzer interface must meet the following requirements:

Paths to relevant tables: Obtain paths to a scanning, token, and reserved word table. Allows the user to start the program without selecting any files (i.e. using tables at default locations). However, allow the user to specify alternative paths for one or more of the tables. Inform the user if the scanning is attempted, but one or more tables is not present. Provide useful error messages.

Perform analysis on input: Allow the user to choose a test file containing a MiniC program.

Token by token: When a MiniC program is being scanned, allow the user (eventually this will be the parser) to repeatedly request a token, until the MiniC program has been consumed or the user decides to quit scanning. After each request, your program should display the token identified and the lexeme associated with the token, and allow the user to request the next token.

Source code errors: When the analyzer encounters an error in the MiniC program that it is scanning, the analyzer should display an error message that is helpful to the developer of the MiniC program. The error should display what character(s) caused the error and at what location within the file they have occurred. The analyzer should attempt to continue scanning the program once the error has been displayed.

EOF in source file: Recognize the end of the source program and inform the user that the end of the file has been reached.

For full credit:

- Your program must be well commented. Use documentation strings for all public functions and classes. The inputs and outputs of functions must be well described. Comment your intent on confusing code blocks.
- Your program should be well designed.
- Your tables must not be hard coded (use a .csv file for tables).
- Paths to your tables are not hard coded (see Paths to relevant tables above).

You cannot use the tool Lex for this assignment or any other library that performs the task of lexical analysis by default.