San José State University
Department of Computer Engineering

CMPE 152
Compiler Design

Instructor: Robert Nicholson

Assignment #2

Assigned:
Due: Next week at start of class
Team assignment, 100 points max

Scanner

Develop a scanner that will recognize the list of keywords shown in the following section. Use the finite state machine / table approach described in class.

Your scanner should be implemented as an object; the nexttoken() function should be a member function; the table should be a private data structure within the object.

Each time the nexttoken () function is called, it should return the next token in the input stream; when no more text remains in the input stream, it should return an EOF indicator. Your scanner should skip space characters (tab, space, and newline).

Write a test program to call the nexttoken (). It should accept input from STDIN, and output to STDOUT, and continue until the scanner function returns an EOF.

The test program should output a text label for each token recognized in the input stream, one token per line.

Tokens to Recognize

Here is the list of tokens to recognize, and the labels to print when each token is recognized.

TOKEN	LABEL
and	AND
array	ARRAY
asm	ASM
begin	BEGIN
break	BREAK
case	CASE
const	CONST
constructor	CONSTRUCTOR
continue	CONTINUE
destructor	DESTRUCTOR
div	DIV
do	DO
downto	DOWNTO
else	ELSE
end	END
FALSE	FALSE
file	FILE
for	FOR
function	FUNCTION
goto	GOTO
if	IF
implementation	IMPLEMENTATION
in	IN
inline	INLINE
interface	INTERFACE
label	LABEL
mod	MOD
nil	NIL
not	NOT
object	OBJECT
of	OF
on	ON
operator	OPERATOR
or	OR

packed	PACKED
procedure	PROCEDURE
program	PROGRAM
record	RECORD
repeat	REPEAT
set	SET
shl	SHL
shr	SHR
string	STRING
then	THEN
to	TO
TRUE	TRUE
type	TYPE
unit	UNIT
until	UNTIL
uses	USES
var	VAR
while	WHILE
with	WITH
xor	XOR
(integer)	INTEGER
(real number)	REAL
(identifier)	INDENTIFIER
+	PLUSOP
-	MINUSOP
*	MULTOP
/	DIVOP
:=	ASSIGN
=	EQUAL
<>	NE
<=	LTEQ
>=	GTEQ
<	LT
>	GT
+=	PLUSEQUAL
-=	MINUSEQUAL
*=	MULTEQUAL
/=	DIVEQUAL
^	CARAT

;	SEMICOLOR
,	COMMA
(	LPAREN
)	RPAREN
[	LBRACKET
]	RBRACKET
{	LBRACE
}	RBRACE
(*	LCOMMENT
*)	RCOMMENT

Your scanner should not be case sensitive. In other words, it should recognize “array”, “ARRAY”, or even “ArRAy.”

Coding trick: Suppose your scanner recognizes word – anything that starts with a letter and contains a sequence of letters and digits. If you find a word, you can then look it up in a table to see if it is one of the special tokens. This will make your state transition table a lot simpler, but it will not perform as well as the purely table-driven scanner, because you’ll need an extra lookup.

The choice is yours, but please document it.

C++ Coding suggestion

Create an enumerated type to represent the token types. Write an output function (or override an output operator) to print the labels.

What to turn in

- A ZIP file containing a folder with the following contents:
- All source files for your program.
- A **makefile**
- A **README.txt** file with any special instructions for building and running your test program.
- A test input file called **test-in.txt**, containing at least 20 and no more than 50 tokens.
- An output file called **test-out.text** produced by your program in response to the test input.
- Your ZIP file should be named: *teamname*-assignment-1.zip

Submit to Canvas: **Assignment #2.**

Only ONE team member should submit the assignment.

Rubric

Your Functional Specification will be graded according to these criteria:

Criteria	**Maximum points**
• Correct makefile and following instructions	**10**
• Correct implementation of table-driven state machine	**20**
• Clean, well-structured code; scanner implemented as an object	**20**
• Correct execution on the test case you provide	**10**
• Correct execution of a test case I provide	**40**
TOTAL	**100**