CSE 4713/6713 – Programming Languages
Assignment 1

In this course, we will design and implement an interpreter for the BullyC language, an extended subset of the C language. It is a subset in that we will not support all constructs in C. It is extended in that we will add elements not found in C.

Our first task is to write a lexical analyzer for BullyC. The job of a lexical analyzer is to return the *lexemes* (i.e., fundamental syntactical elements) in the input program to a parser for further analysis.

Following is a list of lexemes in BullyC:

Keywords	**Token Identifier Value**	**Token Constant**
if	1001	TOK_IF
else	1002	TOK_ELSE
for	1003	TOK_FOR
while	1004	TOK_WHILE
print	1005	TOK_PRINT
return	1006	TOK_RETURN
continue	1007	TOK_CONTINUE
break	1008	TOK_BREAK
debug	1009	TOK_DEBUG
read	1010	TOK_READ
Datatype Specifiers	**Token Identifier Value**	**Token Constant**
int	1100	TOK_INT
float	1101	TOK_FLOAT
string	1102	TOL_STRING
Punctuation	**Token Identifier Value**	**Token Constant**
;	2000	TOK_SEMICOLON
(	2001	TOK_OPENPAREN
)	2002	TOK_CLOSEPAREN
[	2003	TOK_OPENBRACKET
]	2004	TOK_CLOSEBRACKET
{	2005	TOK_OPENBRACE
}	2006	TOK_CLOSEBRACE
,	2007	TOK_COMMA
Operators	**Token Identifier Value**	**Token Constant**
+	3000	TOK_PLUS
-	3001	TOK_MINUS
*	3002	TOK_MULTIPLY
/	3003	TOK_DIVIDE
:=	3004	TOK_ASSIGN
==	3005	TOK_EQUALTO
<	3006	TOK_LESSTHAN
>	3007	TOK_GREATERTHAN
<>	3008	TOK_NOTEQUALTO
&&	3009	TOK_AND

||	3010	TOK_OR
~	3011	TOK_NOT
length	3012	TOK_LENGTH
Useful Abstractions	**Token Identifier Value**	**Token Constant**
identifier	4000	TOK_IDENTIFIER
integer literal	4001	TOK_INTLIT
floating-point literal	4002	TOK_FLOATLIT
string	4003	TOK_STRINGLIT
End of file	5000	TOK_EOF
Unknown lexeme	6000	TOK_UNKNOWN

An identifier is defines as follows: <letter> { <letter> | <digit> | _ } where letter is any upper- or lower-case letter in the English alphabet. Digit is any numeral 0..9. and _ is the underscore character. Therefore, this_is_an identifier is a valid identifier whereas _this_is_not and 1more_bad_example are not a valid identifiers. Identifiers may not be keywords. However, case is significant and keywords are always composed of lowercase characters. Note: an identifier ends when a character not legal for the identifier is encountered.

An integer literal consists of a sequence of digits without a decimal point.

A floating-point literal is a sequence of digits containing an embedded decimal point or ending with a decimal point.

A string literal is a sequence of characters ending within double quotation marks.

Whitespace characters (*i.e.*, space, tab, new-line) act as lexeme terminators. Whitespace should be ignored by your lexical analyzer, except for separating lexemes.

Ambiguity is resolved in favor of longer lexemes. Therefor the word iffiness in the input results in an identifier token and not a keyword token (for if) followed by an identifier token.

The interface for your lexical analyzer is as follows:

Global variables:

1. Input stream yyin
2. Output stream yyout
3. Integer yyleng containing the length of the identified lexeme.
4. character array yytext containing the identified lexeme

Function:

1. yylex; no parameters, returns an integer token identifier value of the identified lexeme.

Organize you program into two separate source files lexer.cpp and driver.cpp. Lexer.cpp should contain your lexical analyzer code in function yylex and manages variables yyleg and yytext. Driver.cpp declares and initializes the global variables, opens the input/output streams and initializes the stream variables. The driver then repeatedly calls yylex() until yylex returns TOK_EOF.

For output, print all the lexemes in an input files on a separate line along with its token identifier.