CS 335 Semester 2023–2024-II: Assignment 1

14th January 2024

Due Your assignment is due by Jan 28 2024 11:59 PM IST.

General Policies

- You should do this assignment ALONE.
- Do not plagiarize or turn in solutions from other sources. You will be PENALIZED if caught.

Submission

- Your solution SHOULD use Flex. We recommend using `flex` and not `flex++`. You can include C++ STL code in the flex specification, and use `g++` to compile the generated C file.

  Do NOT use alternate software like RE/flex, JFlex, and PLY for this assignment.
- Submission will be through Canvas.
- Upload a zip file named "⟨`roll`⟩`-assign1.zip`". The zipped file should contain a directory ⟨`roll`⟩`-assign1`, two sub-directories `problem1` and `problem2`, and the following files:
  (i) The Flex specification files,
  (ii) A `PDF` file that INCLUDES compilation and execution instructions. You may optionally include other relevant details (e.g., corner cases).
- We encourage you to use the LaTeX typesetting system for generating the PDF file.
- You will get up to TWO LATE days to submit your assignment, with a 25% penalty for each day.

Evaluation

- Write your code such that the EXACT output format (if any) is respected.
- We will evaluate your implementations on a Unix-like system, for example, a recent Debian-based distribution. Check for compilation and linking errors.
- We will evaluate the implementations with our OWN inputs and test cases (where applicable), so remember to test thoroughly.
- We WILL deduct marks if you disregard the listed rules.

Problem 1 [40 marks]

The following is the lexical specification for a new language “Kanpur”. The file extension for Kanpur programs is `.knp`. Write a lexical analyzer (i.e., scanner) for Kanpur programs using Flex. Name the flex specification `prob1.l`. Illegal characters and ill-formed strings (i.e., quotes are not paired) should be reported along with the line numbers in the source code. The exact error message depends upon you.

Keywords `ARRAY BEGIN BOOLEAN COMMENT CONTINUE DO DOUBLE ELSE END FALSE FOR IF INTEGER LABEL LIST LONG OWN PROCEDURE STEP SWITCH THEN TRUE UNTIL VALUE WHILE`

Operators AND OR LEQ LT GEQ GT NOT EQL NEQ := + − * / % ^ | & `«` `»` `<=` `>=`

A single <, >, !, and = character is an invalid operator. Note that « and » consist of two characters and are not a Unicode symbol.

Identifier A valid identifier begins with a letter and is followed by any number of occurrences of digits and letters.

Strings A string literal is a collection of zero or more ASCII characters encapsulated either in single quotes or double quotes. A string literal with double quotes cannot have single quotes included in the literal, and vice versa. Strings can be multiline.

`‘‘valid` is an ill-formed string (). An ill-formed string will gobble up the rest of the input file.

Delimiters ; : , ' " [ ] { } ( )

The character ' is a single quote and " is a double quote.

Numeric literals Numeric literals can be decimal (base 10) or hexadecimal (base 16), and are whole numbers. Leading 0s are not allowed. For example, 0 is valid but numbers of the form 0001 and 00 are invalid. Hex literals start with 0x or 0X followed by one or more hex digits (both upper and lower cases are allowed for `A-F`). As before, 0x0 is valid, but 0x00 and 0X0001 are invalid.

A floating point number is formed by one or more digits before the decimal point followed by at least one and up to six digits after the decimal. For example, floating-point numbers 0.00 and 0.500 are valid, but .01, 1., and 123.4567890 are ill-formed. Floating point numbers are only in base 10.

Scientific notations are not allowed.

Comments Comments are any text between { and }. The first occurrence of the terminating character } ends the comment; comments can be multiline but cannot be nested.

White Space White space is defined as the ASCII space character, horizontal tab character, form feed character, and line terminator characters.

Blanks, tabs, ends of lines, and comments are considered to be delimiters. The Lexical Analyzer consumes these but does not return anything.

Other rules

- Keywords and operators are case-insensitive (i.e., `UNTIL` and `until` have the same meaning). Identifiers are case-sensitive.
- All the numbers are unsigned (i.e., whole numbers).

The full set of tokens are KEYWORD, OPERATOR, IDENTIFIER, STRING, DELIMITER, INTEGER, FLOATING_POINT, and HEXADECIMAL. Check the public test cases to learn about the usage. The meaning of the programs is not important for this assignment.

- The output should list and classify all unique lexemes into proper syntactic categories (as enumerated above). You should ignore reporting comments.
- The output should be sorted by `LEXEME` (do not change case while printing). Follow the conventions in the sample output. Not abiding by the output rules will attract a minimum of 20% marks as a penalty.
- Use a single counter for each keyword. If the input has both "unTIL" and "Until", there will be two rows in the output with a count of two.
- Your scanner can terminate after reporting the first error (if any) but only after printing all valid tokens and lexemes before the first error is detected. You may optionally continue beyond the first error.

Example. The output for test case `public1.knp` is as follows.

TOKEN	**COUNT**	**LEXEME**
STRING	1	`"x is greater than y"`
STRING	1	`"y is greater than x"`
DELIMITER	2	`(`
DELIMITER	2	`)`
DELIMITER	1	`,`
INTEGER	1	`10`
INTEGER	1	`20`
OPERATOR	2	`:=`
DELIMITER	5	`;`
KEYWORD	1	`BEGIN`
KEYWORD	1	`ELSE`
KEYWORD	1	`END`
OPERATOR	1	`GT`
KEYWORD	1	`IF`
KEYWORD	1	`INTEGER`
IDENTIFIER	2	`PRINT`
KEYWORD	1	`THEN`
IDENTIFIER	3	`x`
IDENTIFIER	3	`y`

Problem 2 **[60 marks]**

Write a lexer for Fortran 2008. Name the flex specification `prob2.l`. Letters, digits, underscore, and special characters (Table 3.1) make up the Fortran character set. Any illegal character encountered should be reported along with the line number in the source code.

We list the constraints on the specification in the following. Refer to Chapter 3 in the linked PDF for more details.

Keywords `allocatable allocate assign associate asynchronous backspace bind call case class close common contains continue cycle codimension contiguous critical data deallocate deferred dimension do else elsewhere end endfile endif entry equivalence exit external enum enumerator extends forall final flush format function goto generic import if implicit inquire intrinsic include interface intent lock module nopass namelist nullify open only operator optional parameter pause print program pointer private pass protected procedure public read return recursive result rewind rewrite save stop subroutine select sequence submodule target then use value unlock volatile while write`.

Names A valid name begins with a letter and is followed by more letters, digits, or underscore. The maximum length of a name is 63.

Literals Consider only the rules for int-literal-constant, signed-int-literal-constant, real-literal-constant, signed-real-literal-constant, and logical-literal-constant (Section 3.2.3). A string literal is a collection of zero or more characters encapsulated in double quotes. A string literal with double quotes cannot have single quotes included in the literal. Strings can be multiline.

Ignore the optional kind type parameter.

Operator $= + \; - \; * \; / \; < \; > \; \% \; \&$
In addition, consider only the rules defined in intrinsic-operator (Section 3.2.4).

Delimiters ( ) [ ] (/ /)

Comments If the first non-blank character on a line is an "!", the line is a comment (Section 3.3.2.3). The character "!" initiates a comment except where it appears within a string. The comment extends to the end of the line.

Special Characters All other characters that are listed in Table 3.1 but not mentioned above.

Other rules

- Assume that keywords are reserved, and cannot be used as identifiers.
- Extend the operators to include `&&` « » `||`
- Fortran is case-insensitive, i.e., keywords, operators, and names (e.g., variables and function names) written in any case combination are equivalent. Only strings are case-sensitive.
- Ignore labels (Section 3.2.5).
- Ignore nuances based on fixed-format layout.

Your lexer should tokenize input Fortran programs into the given token categories, and print the count of each token along with the category. The full set of tokens are KEYWORD, NAME, INT_LITERAL, REAL_LITERAL, LOGICAL_LITERAL, CHAR_LITERAL, OPERATOR, DELIMITER, and SPECIAL_CHAR.

The output requirements are the same as in Problem 1.