CS 3500 – Programming Languages & Translators
Homework Assignment #2

- This assignment is **due by 8 p.m. on Friday, September 7, 2018**.
- This assignment will be worth **2%** of your course grade.
- You are to work on this assignment **by yourself**.
- You should **take a look at the sample input and output files** posted on the Canvas website **before** you actually submit your assignment for grading. In particular, you should **compare your output with the posted sample output using the *diff* command**, as was recommended in HW #1.

Basic Instructions:

For this assignment you are to use ***bison*** (in conjunction with *flex*) to create a C++ program that will perform **syntax analysis** for the MFPL programming language. If your *flex* file was named **mfpl.l** and your *bison* file was named **mfpl.y**, you should be able to compile and execute them on one of the campus Linux machines (such as rc*nn*ucs213.managed.mst.edu where ***nn*** is **01**-**32**) using the following commands (where *inputFileName* is the name of some input file):

```
flex mfpl.l
bison mfpl.y
g++ mfpl.tab.c -o mfpl_parser
mfpl_parser < inputFileName
```

Your program should process a **single** expression from an input file (although note that that expression could be an expression *list*; see the MFPL grammar that is given later in this document). No attempt should be made to recover from errors; **if your program encounters a syntax error, it should simply output a "syntax error" message which includes the line number** in the input file where the error occurred and terminate. Note that your program should **NOT evaluate** any expressions in the input program as that is not the purpose of lexical analysis or syntax analysis.

MFPL Syntax:

What follows is the context-free grammar for the MFPL programming language for which you are writing the syntax analyzer. To help you distinguish nonterminals from terminals, nonterminal names begin with **N_** and terminal names begin with **T_**.

N_EXPR → N_CONST | T_IDENT |
T_LPAREN N_PARENTHESIZED_EXPR T_RPAREN

N_CONST → T_INTCONST | T_STRCONST | T_T | T_NIL

N_PARENTHESIZED_EXPR → N_ARITHLOGIC_EXPR | N_IF_EXPR |

N_LET_EXPR | N_LAMBDA_EXPR |
N_PRINT_EXPR | N_INPUT_EXPR |
N_EXPR_LIST

N_ARITHLOGIC_EXPR → N_UN_OP N_EXPR | N_BIN_OP N_EXPR N_EXPR

N_IF_EXPR → T_IF N_EXPR N_EXPR N_EXPR

N_LET_EXPR → T_LETSTAR T_LPAREN N_ID_EXPR_LIST T_RPAREN
N_EXPR

N_ID_EXPR_LIST → **ε** | N_ID_EXPR_LIST T_LPAREN T_IDENT N_EXPR
T_RPAREN

N_LAMBDA_EXPR → T_LAMBDA T_LPAREN N_ID_LIST T_RPAREN
N_EXPR

N_ID_LIST → **ε** | N_ID_LIST T_IDENT

N_PRINT_EXPR → T_PRINT N_EXPR

N_INPUT_EXPR → T_INPUT

N_EXPR_LIST → N_EXPR N_EXPR_LIST | N_EXPR

N_BIN_OP → N_ARITH_OP | N_LOG_OP | N_REL_OP

N_ARITH_OP → T_MULT | T_SUB | T_DIV | T_ADD

N_LOG_OP → T_AND | T_OR

N_REL_OP → T_LT | T_GT | T_LE | T_GE | T_EQ | T_NE

N_UN_OP → T_NOT

All other definitions for constructs in this programming language (i.e., tokens and comments) from HW #1 also apply to this assignment.

Sample Input and Output:

You still should output the **token and lexeme information** for every token processed in the input file. In addition, you should output a statement about each **production that is being applied** throughout the parse, and clearly identify when a **syntax error** is encountered and **the line number** on which it occurred.

Given below is some sample input and output; additional sample files are posted on Canvas. Because we are using an automated script (program) for grading, with the exception of whitespace, the output produced by your program **MUST** be **identical** to that of the sample output files! Use **EXACTLY** the same nonterminal names as given in the grammar. However, to shorten things a bit, when you output the **productions** (but **NOT** when you output the TOKEN/LEXEME info from HW #1):

1) drop the **N_** prefix for nonterminal names,

2) use the actual MFPL keywords and symbols rather than their full terminal names (e.g., output (instead of T_LPAREN, * instead of T_MULT, let* instead of T_LETSTAR, etc.) for any token whose lexeme is not unique for its token class,
3) drop the **T_** prefix for the terminals IDENT, INTCONST, STRCONST, T, and NIL,
4) output **epsilon** when **ε** appears on the right hand side of the production.

Also, for better readability output **at least one space between each terminal and/or nonterminal** in each grammar production.

Input example with no syntax errors:

```
 ; no syntax errors in this example
(let* ( (x (input))
        (y (* x 100))
        (z 42)
      )
  (if (and (> x z) (not (or (/= x 100) (= y "hello"))))
    t
    nil
  )
)
```

Output for the example with no syntax errors:

```
TOKEN: LPAREN        LEXEME: (
TOKEN: LETSTAR       LEXEME: let*
TOKEN: LPAREN        LEXEME: (
ID_EXPR_LIST -> epsilon
TOKEN: LPAREN        LEXEME: (
TOKEN: IDENT         LEXEME: x
TOKEN: LPAREN        LEXEME: (
TOKEN: INPUT         LEXEME: input
INPUT_EXPR -> input
PARENTHESIZED_EXPR -> INPUT_EXPR
TOKEN: RPAREN        LEXEME: )
EXPR -> ( PARENTHESIZED_EXPR )
TOKEN: RPAREN        LEXEME: )
ID_EXPR_LIST -> ID_EXPR_LIST ( IDENT EXPR )
TOKEN: LPAREN        LEXEME: (
TOKEN: IDENT         LEXEME: y
TOKEN: LPAREN        LEXEME: (
TOKEN: MULT          LEXEME: *
ARITH_OP -> *
BIN_OP -> ARITH_OP
TOKEN: IDENT         LEXEME: x
```

EXPR -> IDENT
TOKEN: INTCONST LEXEME: 100
CONST -> INTCONST
EXPR -> CONST
ARITHLOGIC_EXPR -> BIN_OP EXPR EXPR
PARENTHESIZED_EXPR -> ARITHLOGIC_EXPR
TOKEN: RPAREN LEXEME:)
EXPR -> (PARENTHESIZED_EXPR)
TOKEN: RPAREN LEXEME:)
ID_EXPR_LIST -> ID_EXPR_LIST (IDENT EXPR)
TOKEN: LPAREN LEXEME: (
TOKEN: IDENT LEXEME: z
TOKEN: INTCONST LEXEME: 42
CONST -> INTCONST
EXPR -> CONST
TOKEN: RPAREN LEXEME:)
ID_EXPR_LIST -> ID_EXPR_LIST (IDENT EXPR)
TOKEN: RPAREN LEXEME:)
TOKEN: LPAREN LEXEME: (
TOKEN: IF LEXEME: if
TOKEN: LPAREN LEXEME: (
TOKEN: AND LEXEME: and
LOG_OP -> and
BIN_OP -> LOG_OP
TOKEN: LPAREN LEXEME: (
TOKEN: GT LEXEME: >
REL_OP -> >
BIN_OP -> REL_OP
TOKEN: IDENT LEXEME: x
EXPR -> IDENT
TOKEN: IDENT LEXEME: z
EXPR -> IDENT
ARITHLOGIC_EXPR -> BIN_OP EXPR EXPR
PARENTHESIZED_EXPR -> ARITHLOGIC_EXPR
TOKEN: RPAREN LEXEME:)
EXPR -> (PARENTHESIZED_EXPR)
TOKEN: LPAREN LEXEME: (
TOKEN: NOT LEXEME: not
UN_OP -> not
TOKEN: LPAREN LEXEME: (
TOKEN: OR LEXEME: or
LOG_OP -> or
BIN_OP -> LOG_OP
TOKEN: LPAREN LEXEME: (

TOKEN: NE LEXEME: /=
REL_OP -> /=
BIN_OP -> REL_OP
TOKEN: IDENT LEXEME: x
EXPR -> IDENT
TOKEN: INTCONST LEXEME: 100
CONST -> INTCONST
EXPR -> CONST
ARITHLOGIC_EXPR -> BIN_OP EXPR EXPR
PARENTHESIZED_EXPR -> ARITHLOGIC_EXPR
TOKEN: RPAREN LEXEME:)
EXPR -> (PARENTHESIZED_EXPR)
TOKEN: LPAREN LEXEME: (
TOKEN: EQ LEXEME: =
REL_OP -> =
BIN_OP -> REL_OP
TOKEN: IDENT LEXEME: y
EXPR -> IDENT
TOKEN: STRCONST LEXEME: "hello"
CONST -> STRCONST
EXPR -> CONST
ARITHLOGIC_EXPR -> BIN_OP EXPR EXPR
PARENTHESIZED_EXPR -> ARITHLOGIC_EXPR
TOKEN: RPAREN LEXEME:)
EXPR -> (PARENTHESIZED_EXPR)
ARITHLOGIC_EXPR -> BIN_OP EXPR EXPR
PARENTHESIZED_EXPR -> ARITHLOGIC_EXPR
TOKEN: RPAREN LEXEME:)
EXPR -> (PARENTHESIZED_EXPR)
ARITHLOGIC_EXPR -> UN_OP EXPR
PARENTHESIZED_EXPR -> ARITHLOGIC_EXPR
TOKEN: RPAREN LEXEME:)
EXPR -> (PARENTHESIZED_EXPR)
ARITHLOGIC_EXPR -> BIN_OP EXPR EXPR
PARENTHESIZED_EXPR -> ARITHLOGIC_EXPR
TOKEN: RPAREN LEXEME:)
EXPR -> (PARENTHESIZED_EXPR)
TOKEN: T LEXEME: t
CONST -> t
EXPR -> CONST
TOKEN: NIL LEXEME: nil
CONST -> nil
EXPR -> CONST
IF_EXPR -> if EXPR EXPR EXPR

```
PARENTHESIZED_EXPR -> IF_EXPR
TOKEN: RPAREN        LEXEME: )
EXPR -> ( PARENTHESIZED_EXPR )
LET_EXPR -> let* ( ID_EXPR_LIST ) EXPR
PARENTHESIZED_EXPR -> LET_EXPR
TOKEN: RPAREN        LEXEME: )
EXPR -> ( PARENTHESIZED_EXPR )
START -> EXPR

---- Completed parsing ----
```

Input example with a syntax error:

```
; there is a syntax error in this example
(let* ( (x (input))
        (y (* x))   ; syntax error in expression on this line
        (z 42)
      )
  (if (and (> x z) (not (or (/= x 100) (= y "hello"))))
     t
     nil
  )
)
```

Output for the example with a syntax error:

```
TOKEN: LPAREN        LEXEME: (
TOKEN: LETSTAR       LEXEME: let*
TOKEN: LPAREN        LEXEME: (
ID_EXPR_LIST -> epsilon
TOKEN: LPAREN        LEXEME: (
TOKEN: IDENT         LEXEME: x
TOKEN: LPAREN        LEXEME: (
TOKEN: INPUT         LEXEME: input
INPUT_EXPR -> input
PARENTHESIZED_EXPR -> INPUT_EXPR
TOKEN: RPAREN        LEXEME: )
EXPR -> ( PARENTHESIZED_EXPR )
TOKEN: RPAREN        LEXEME: )
ID_EXPR_LIST -> ID_EXPR_LIST ( IDENT EXPR )
TOKEN: LPAREN        LEXEME: (
TOKEN: IDENT         LEXEME: y
TOKEN: LPAREN        LEXEME: (
TOKEN: MULT          LEXEME: *
```

ARITH_OP -> *
BIN_OP -> ARITH_OP
TOKEN: IDENT LEXEME: x
EXPR -> IDENT
TOKEN: RPAREN LEXEME:)
Line 3: syntax error

What to Submit for Grading:

You should submit only your *flex* and *bison* files via Canvas, archived as a *zip* file. Name your *flex* and *bison* files using **your last name followed by your first initial** with the correct *.l and .y* file extensions (e.g., Homer Simpson would name his files **simpsonh.l** and **simpsonh.y**). Your zip file should be similarly named (e.g., **simpsonh.zip**). You can submit multiple times before the deadline; only your last submission will be graded.

WARNING: If you fail to follow all of the instructions in this assignment, the automated grading script will reject your submission, in which case it will NOT be graded!!!

The grading rubric is given below so that you can see how many points each part of this assignment is worth. Note that the next assignment builds upon this one, so **it is critical that this assignment works properly in all respects!**

	Points Possible	Mostly or completely incorrect (0% of points possible)	Needs improvement (70% of points possible)	Adequate, but still some deficiencies (80% of points possible)	Mostly or completely correct (100% of points possible)
Flex file correctly processes tokens and is correctly set up for bison file	10				
All productions in the grammar are correctly expressed in the bison file and are output when applied in a derivation	84				
Error message is output with correct line number when syntax error is detected	6				