# CS 3500 – Programming Languages & Translators
# Homework Assignment #2

- This assignment is **due by 11:59 p.m. on Wednesday, February 13, 2019**
- This assignment will be worth **3%** 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 Mini-R programming language. If your *flex* file was named **minir.l** and your *bison* file was named **minir.y**, you should be able to compile and execute them on one of the campus Linux machines using the following commands (where *inputFileName* is the name of some input file):

```
flex minir.l
bison minir.y
g++ minir.tab.c -o minir_parser
minir_parser < inputFileName
```

Your program should process expressions from an input file until it processes the expression ***quit()*** or a syntax error is encountered[1]. 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.

## Syntax for the Mini-R Programming Language

What follows is the context-free grammar for the Mini-R programming language for which you are writing the syntax analyzer. To help you distinguish nonterminals from terminals, we've labelled nonterminal names with **N_** and terminal names with **T_** in the grammar. However, to conserve on space, some short terminals like (e.g., '(' and ',') have been written using their symbols rather than their full token names (e.g., T_LPAREN and T_COMMA); in these cases, those symbols have been **boldfaced**.

[1] Note that you can make a C/C++ program terminate (from anywhere in the program) by calling *exit(1)*. See http://www.cplusplus.com/reference/cstdlib/exit/

N_EXPR → N_IF_EXPR | N_WHILE_EXPR | N_FOR_EXPR |
N_COMPOUND_EXPR | N_ARITHLOGIC_EXPR |
N_ASSIGNMENT_EXPR | N_OUTPUT_EXPR | N_INPUT_EXPR |
N_LIST_EXPR | N_FUNCTION_DEF | N_FUNCTION_CALL |
N_QUIT_EXPR

N_CONST → T_INTCONST | T_STRCONST | T_FLOATCONST | T_TRUE | T_FALSE

N_COMPOUND_EXPR → **{** N_EXPR N_EXPR_LIST **}**
N_EXPR_LIST → **;** N_EXPR N_EXPR_LIST | ε

N_IF_EXPR → T_IF **(** N_EXPR **)** N_EXPR | T_IF **(** N_EXPR **)** N_EXPR T_ELSE N_EXPR

N_WHILE_EXPR → T_WHILE **(** N_EXPR **)** N_LOOP_EXPR
N_FOR_EXPR → T_FOR **(** T_IDENT T_IN N_EXPR **)** N_LOOP_EXPR
N_LOOP_EXPR → N_EXPR | N_BREAK_EXPR | N_NEXT_EXPR
N_BREAK_EXPR → T_BREAK
N_NEXT_EXPR → T_NEXT

N_LIST_EXPR → T_LIST **(** N_CONST_LIST **)**
N_CONST_LIST → N_CONST **,** N_CONST_LIST | N_CONST

N_ASSIGNMENT_EXPR → T_IDENT N_INDEX **=** N_EXPR
N_INDEX → **[ [** N_EXPR **] ]** | ε

N_QUIT_EXPR → T_QUIT**( )**

N_OUTPUT_EXPR → T_PRINT **(** N_EXPR **)** | T_CAT **(** N_EXPR **)**

N_INPUT_EXPR → T_READ **(** N_VAR **)**

N_FUNCTION_DEF → T_FUNCTION **(** N_PARAM_LIST **)** N_COMPOUND_EXPR
N_PARAM_LIST → N_PARAMS | N_NO_PARAMS
N_NO_PARAMS → ε
N_PARAMS → T_IDENT | T_IDENT **,** N_PARAMS

N_FUNCTION_CALL → T_IDENT **(** N_ARG_LIST **)**

N_ARG_LIST → N_ARGS | N_NO_ARGS

N_NO_ARGS → ε

N_ARGS → N_EXPR | N_EXPR **,** N_ARGS

N_ARITHLOGIC_EXPR → N_SIMPLE_ARITHLOGIC |
N_SIMPLE_ARITHLOGIC N_REL_OP N_SIMPLE_ARITHLOGIC

N_SIMPLE_ARITHLOGIC → N_TERM N_ADD_OP_LIST

N_ADD_OP_LIST → N_ADD_OP N_TERM N_ADD_OP_LIST | ε

N_TERM → N_FACTOR N_MULT_OP_LIST

N_MULT_OP_LIST → N_MULT_OP N_FACTOR N_MULT_OP_LIST | ε

N_FACTOR → N_VAR | N_CONST | ( N_EXPR ) | T_NOT N_FACTOR

N_ADD_OP → T_ADD | T_SUB | T_OR

N_MULT_OP → T_MULT | T_DIV | T_AND | T_MOD | T_POW

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

N_VAR → N_ENTIRE_VAR | N_SINGLE_ELEMENT

N_SINGLE_ELEMENT → T_IDENT **[ [** N_EXPR **] ]**

N_ENTIRE_VAR → T_IDENT

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 for 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 and terminal names as given in the sample output.

Input example with no syntax errors:

```
if (boJack)
5
else
print("Mr. Peanutbutter")
```

Output for the example with no syntax errors:

```
TOKEN: IF              LEXEME: if
TOKEN: LPAREN          LEXEME: (
TOKEN: IDENT           LEXEME: boJack
TOKEN: RPAREN          LEXEME: )
ENTIRE_VAR -> IDENT
VAR -> ENTIRE_VAR
FACTOR -> VAR
MULT_OP_LIST -> epsilon
TERM -> FACTOR MULT_OP_LIST
ADD_OP_LIST -> epsilon
SIMPLE_ARITHLOGIC -> TERM ADD_OP_LIST
ARITHLOGIC_EXPR -> SIMPLE_ARITHLOGIC
EXPR -> ARITHLOGIC_EXPR
TOKEN: INTCONST        LEXEME: 5
CONST -> INTCONST
FACTOR -> CONST
TOKEN: ELSE            LEXEME: else
MULT_OP_LIST -> epsilon
TERM -> FACTOR MULT_OP_LIST
ADD_OP_LIST -> epsilon
SIMPLE_ARITHLOGIC -> TERM ADD_OP_LIST
ARITHLOGIC_EXPR -> SIMPLE_ARITHLOGIC
EXPR -> ARITHLOGIC_EXPR
TOKEN: PRINT           LEXEME: print
TOKEN: LPAREN          LEXEME: (
TOKEN: STRCONST        LEXEME: "Mr. Peanutbutter"
CONST -> STRCONST
FACTOR -> CONST
TOKEN: RPAREN          LEXEME: )
MULT_OP_LIST -> epsilon
TERM -> FACTOR MULT_OP_LIST
ADD_OP_LIST -> epsilon
SIMPLE_ARITHLOGIC -> TERM ADD_OP_LIST
ARITHLOGIC_EXPR -> SIMPLE_ARITHLOGIC
EXPR -> ARITHLOGIC_EXPR
OUTPUT_EXPR -> PRINT ( EXPR )
EXPR -> OUTPUT_EXPR
IF_EXPR -> IF ( EXPR ) ELSE EXPR
EXPR -> IF_EXPR
START -> EXPR

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

Input example with a syntax error:

```
while (boJack < 10
  boJack = boJack + 1
```

Output for the example with a syntax error:

```
TOKEN: WHILE            LEXEME: while
TOKEN: LPAREN           LEXEME: (
TOKEN: IDENT            LEXEME: boJack
TOKEN: LT               LEXEME: <
ENTIRE_VAR -> IDENT
VAR -> ENTIRE_VAR
FACTOR -> VAR
MULT_OP_LIST -> epsilon
TERM -> FACTOR MULT_OP_LIST
ADD_OP_LIST -> epsilon
SIMPLE_ARITHLOGIC -> TERM ADD_OP_LIST
REL_OP -> <
TOKEN: INTCONST         LEXEME: 10
CONST -> INTCONST
FACTOR -> CONST
TOKEN: IDENT            LEXEME: boJack
MULT_OP_LIST -> epsilon
TERM -> FACTOR MULT_OP_LIST
ADD_OP_LIST -> epsilon
SIMPLE_ARITHLOGIC -> TERM ADD_OP_LIST
ARITHLOGIC_EXPR -> SIMPLE_ARITHLOGIC REL_OP SIMPLE_ARITHLOGIC
EXPR -> ARITHLOGIC_EXPR
Line 2: syntax error
```

## An Ambiguity in the Grammar

There is one **ambiguity** in the Mini-R grammar, which will result in conflicts in the parse table generated by *bison* unless we do something about it. This problem stems from N_IF_EXPR. Effectively, we want an *"else"* to pair with the nearest "*then*"; our if-expression doesn't use the keyword "*then*", so instead we'll need an *"else"* to pair with the nearest right parenthesis. Using our token names **T_RPAREN** and **T_ELSE**, we can tell *bison* how to correctly deal with this situation by including the following lines in the declaration section of your *bison* specification file, along with your other **%token** lines:

**%nonassoc T_RPAREN**
**%nonassoc T_ELSE**

Use these lines **instead of** **%token T_RPAREN** and **%token T_ELSE**. The **nonassoc** directive says that two occurrences of that token should not be combined. Tokens are given precedence in the order in which they appear in the nonassoc declarations of the

*bison* file, with the token having the lowest precedence listed first. Therefore, you want to be sure you have the **T_RPAREN** defined **before** the **T_ELSE**.

There should be no other conflicts when you run *bison* on your .y file. If conflicts are reported, then you have made a mistake specifying the productions for the grammar (e.g., you typed something wrong, left out a production, etc.). You must fix your bison file so that there are no conflicts!

## 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) |
|---|---|---|---|---|---|
| All productions in the grammar are correctly expressed in the bison file and are output when applied in a derivation | 90 | | | | |
| Error message is output with correct line number when syntax error is detected | 5 | | | | |
| Program terminates when syntax error detected or quit() is processed | 5 | | | | |