CS 256 – Programming Languages and Translators
Assignment 5

- This assignment is due by 1 p.m. on Friday April 4, 2014
- This assignment will be worth 7% of your grade
- You are to work on this assignment by yourself

Basic Instructions

As before, your program must compile and execute on one of the campus Linux machines (such as rc*nn*xcs213.managed.mst.edu where *nn* is 01-32). If your flex file was named mfpl.l and your bison file was named mfpl.y, we should be able to compile and execute them 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_eval
mfpl_eval inputFileName
```

Note that we are no longer redirecting input using ¡ in the command line. This is necessary in order to be able to get input from the keyboard for the MFPL (input) statement at the time that it actually is evaluated. Consequently, you need to change your main( ) function to process the input file from the command line:

```
int main(int argc, char** argv) {
  if (argc < 2) {
    printf("You must specify a file in the command line!\n");
    exit(1);
  }
  yyin = fopen(argv[1], "r");
  do {
    yyparse();
  } while (!feof(yyin));
  return 0;
}
```

As in HW4, no attempt should be made to recover from non-lexical errors; if your program encounters a syntactical or semantic error, or attempted division by zero (a new error you need to catch!), then it should simply output a meaningful message and terminate execution.

Your program should still output the tokens and lexemes that it encounters in the input file, and still should output an appropriate message whenever it begins or ends a scope, or whenever it makes an entry in the symbol table.

As before, your program should process a single expression from the input file (but remember that a single expression could be an expression list), terminating when it completes processing the expression or encounters an error. After reading in the expression from the input file, your program should evaluate and output the resulting value of the expression; this is known as the read-eval-print process of interpreters. Specifically, the `N_START` rule in your bison file should include the following output statements:

```
printRule("START", "EXPR");
printf("\n---- Completed parsing ----\n\n");
printf("\nValue of the expression is: ");
```

Programming Language

What follows is a brief description about the evaluation rules for the various expressions in MFPL (Mini Functional Programming Language).

N_EXPR $\rightarrow$ N_CONST | T_IDENT | T_LPAREN N_PARENTHESIZED_EXPR T_RPAREN

The resulting value of an `N_EXPR` is the value of the constant if the `N_CONST` rule is applied, or the value of the identifier if the `T_IDENT` rule is applied (i.e., youll have to look up its name in the symbol table to find out its value), or the value of the `N_PARENTHESIZED_EXPR` if that rule is applied.

N_CONST $\rightarrow$ T_INTCONST | T_STRCONST | T_T | T_NIL

The resulting value of an `N_CONST` is the value of an integer constant if the `T_INTCONST` rule is applied, or the value of a string constant if the `T_STRCONST` rule is applied. Be careful what value you choose to internally represent the MFPL constants t and nil; logically, MFPL should treat nil as false and everything else (including 0) as true.

N_PARENTHESIZED_EXPR $\rightarrow$ N_ARITHLOGIC_EXPR | N_IF_EXPR | N_LET_EXPR |
N_LAMBDA_EXPR | N_PRINT_EXPR | N_INPUT_EXPR |
N_EXPR_LIST

The resulting value of an `N_PARENTHESIZED_EXPR` is the value returned by whichever rule is applied. Note that you are no longer required to process `N_LAMBDA_EXPR`.

N_ARITHLOGIC_EXPR $\rightarrow$ N_UN_OP N_EXPR | N_BIN_OP N_EXPR N_EXPR
N_BIN_OP $\rightarrow$ N_ARITH_OP | N_LOG_OP | N_REL_OP
N_ARITH_OP $\rightarrow$ T_MULT | T_SUB | T_DIV | T_ADD
N_LOG_OP $\rightarrow$ T_AND | T_OR
N_REL_OP $\rightarrow$ T_LT | T_GT | T_LE | T_GE | T_EQ | T_NE
N_UN_OP $\rightarrow$ T_NOT

The value of the `N_ARITHLOGIC_EXPR` depends on what the operator is; basically, you have to perform the operation on the values of the `N_EXPR`s and assign the resulting value to `N_ARITHLOGIC_EXPR`. If the operator is division, you must check for attempted division by zero. For the logical operators (and, or, not), treat nil as false and everything else (including 0) as true.

N_IF_EXPR $\rightarrow$ T_IF N_EXPR N_EXPR N_EXPR

If the first `N_EXPR` evaluates to nil, then the resulting value of the `N_IF_EXPR` is the value of the third `N_EXPR`; otherwise, the resulting value of the `N_IF_EXPR`is the value of the second `N_EXPR`. Note that we can now definitively determine the type of the `N_IF_EXPR` (i.e., there no longer should be designation of types `INT_OR_STR`, `INT_OR_STR_OR_BOOL`, etc.); thus you are expected to assign the type of the `N_IF_EXPR` accordingly.

N_LET_EXPR $\rightarrow$ T_LETSTAR T_LPAREN N_ID_EXPR_LIST T_RPAREN N_EXPR
N_ID_EXPR_LIST $\rightarrow$ ϵ | N_ID_EXPR_LIST T_LPAREN T_IDENT N_EXPR T_RPAREN

The value of each `T_IDENT` in `N_ID_EXPR_LIST` will be the value of its `N_EXPR`. This will need to be recorded in the symbol table! Then the resulting value of the `N_LET_EXPR` will be the value of its `N_EXPR`.

$$\begin{aligned} \text{N\_LAMBDA\_EXPR} &\rightarrow \text{T\_LAMBDA T\_LPAREN N\_ID\_LIST T\_RPAREN N\_EXPR} \\ \text{N\_ID\_LIST} &\rightarrow \epsilon \mid \text{N\_ID\_LIST T\_IDENT} \end{aligned}$$

For this assignment, your program will NOT have to handle functions.

$$\text{N\_PRINT\_EXPR} \rightarrow \text{T\_PRINT N\_EXPR}$$

When an `N_PRINT_EXPR` is processed, it should output the value of the `N_EXPR` followed by a newline to standard output. Also note that the resulting value of a `N_PRINT_EXPR` is the value of the `N_EXPR`.

$$\text{N\_INPUT\_EXPR} \rightarrow \text{T\_INPUT}$$

When an `N_INPUT_EXPR` is processed, it should perform a C or C++ getline call into a string (or char array); do not use cin ¿¿ since valid input can contain spaces. If the first character that is input is a digit or a + or a , treat the entire input as an integer; otherwise, treat the input as a string. Note that you should now set the type of an `N_INPUT_EXPR` to either `INT` or `STR` (its no longer `INT_OR_STR`), and dynamically type-check its use in other expressions.

$$\text{N\_EXPR\_LIST} \rightarrow \text{N\_EXPR N\_EXPR\_LIST} \mid \text{N\_EXPR}$$

The resulting value of an `N_EXPR_LIST` is the value of the last `N_EXPR`.

Submission

You will submit this assignment using `cssubmit`. Your ***single*** lexer file ***must*** have a `.l` extension, and your single yacc/bison file must have a `.y` extension. If it does not, you will receive a zero for this assignment. From the directory containing the `.l` file, `.y`, and symbol table files, you will run
`cssubmit 256 a 5`
on the cs213 Linux machines. This will collect your submission and submit it to me. You may submit as many times as you desire; only your last submission will be graded (previous submissions are overwritten). **READ** the output of cssubmit; it may have changed since you last used it.

When grading, I will run the following commands:

```
flex *.l
bison *.y
g++ *.tab.c
```

If you cannot generate an executable by following EXACTLY those steps from the directory you are submitting from, your submission will not receive full credit. If you find yourself having example .l and .y files, or some other file with a `.tab.c` extension, remove them from your submission directory.

Note

Because we are using an automated script to grade your programs, and the output is fairly extensive, with the exception of whitespace and case (upper vs. lower) of the text, your output is expected to EXACTLY MATCH the output specifications given in the homework assignments! It is recommended that you use the diff command (see http://ss64.com/bash/diff.html) to compare your output with the sample output for this assignment.

Extra Credit

You can earn extra credit by making your program evaluate lambda expressions. The amount of extra credit will depend on the extent to which your program addresses this functionality (pun totally intended). A program that can only handle functions, the body of which is an arithmetic expression like `((lambda (x y) (+ x y)) 10 20)`, will not earn as many points as a program that can handle more complex expressions like the following:

```
( (lambda (x y)
     (let* ( (z (* x 5))
               (foo (lambda (bar) (/ bar x)))
           )
        (- (foo z) y)
      )
    )
  10 20
)
```

If you do the extra credit, in addition to submitting your files for grading for assignment 5, you will email the following to Nathan Eloe (nwe5g8@mst.edu)

1. Your flex, bison, and .h files
2. A brief and clear explanation of the extra credit capabilities of your program (what kind of functions can it handle?)
3. Sample input files that clearly and concisely demonstrate the extent of the extra credit capabilities of your program.

The extra credit can be submitted any time before the last day of class this semester. However, even if you do the extra credit, you are still required to submit the regular HW5 by the specified due date.