Programming Assignment 2

Parser
Due Date: 11:59PM on March 3, 2008

For PA2 you will develop the first phase of a compiler for a Bronco# programming language. This language supports:

- **Nested variable scopes**
  Like ‘C’, you can declare variables within compound blocks.
- **If, while** control structures and recursive functions
- **Type Definitions**
- **Reference Parameters** to functions

To complete this project, your program must be based on the syntax given on Page 2:

1. Your program should have some error recovery features.
2. Error Output should be written to STDOUT.
3. Your program should read an input file from a command line.

**You can choose to use Linux, cygwin (Windows), UNIX or other operating systems where yacc/bison is supported. However, before you submit your assignment, you are required to run your program in SunOS operating system so we can grade your homework easily. You should use an ssh client to connect to csy[01-12].cs.wmich.edu in order to use SunOS system.**

Send your source files to Bilal Abubark at bilal.abubakr@wmich.edu, and cc to zijiang.yang@wmich.edu. with Subject: **Course Number:PA2: Your Name.**
Module

module → unit { unit }                        //{…} represents 0 or many repetitions
unit → declaration ';'                       //global variables
      → function                            //functions
      → type_decl ';'                       //types
      → λ                                    // NULL

Type Declarations

type_decl → type ident '=' type_def
type_def → array ['[ int_const ".." int_const '] of type_id
          → record field_list end
          → enum '{ ident_list '}'
          → pointer of type_id
field_list → field {';' field}
field → type_id ident_list
type_id → ident | base_type                  //’ident’ is a defined type
         → int | float | short        // short takes 2 bytes and others take 4 bytes
base_type → int | float | short

Variable Declarations

declaration_list → declaration { ';' declaration }
declaration → type_id ident_list
ident_list → ident {';' ident}

Function Declarations

function → function return_type ident '(' parameter_list ')' compound_stmt
return_type → type_id | void                  //Function returns nothing
parameter_list → parameter {';' parameter } | λ
parameter → mode type_id ident
mode → ref | λ

Statements

stmt_list → statement { ';' statement }
statement → assign_stmt
           → while_stmt
           → if_stmt
           → function_call                           //Must return VOID
compound_stmt → return_stmt
compound_stmt → print_stmt
compound_stmt → new_stmt
compound_stmt → λ

//NULL statement

assign_stmt → variable "=" expr
while_stmt → while '(' expr ')' statement
if_stmt → if '(' expr ')' statement
if_stmt → if '(' expr ')' statement else statement
function_call → ident '(' expr_list ')
compound_stmt → '{' declaration_list ';' stmt_list '}
compound_stmt → '{' stmt_list '}'
return_stmt → return '(' expr ')' | return
print_stmt → print '(' output_list ')
print_stmt → println '(' output_list ')
print_stmt → println

new_stmt → new variable

Output List

output_list → output_element { ',' output_element }
output_element → expr | string

Expression List

expr_list → expr { ',' expr } | λ

Variables

variable → ident variable_tail
variable_tail → '[' expr ']' variable_tail
variable_tail → '.' ident variable_tail //Ident is defined field
variable_tail → '->' ident variable_tail
variable_tail → λ

Expressions

expr → expr bin_op expr
expr → unary_op expr
expr → '(' expr ')' variable
expr → function_call
expr → int_const
expr → real_const

bin_op → '+' | '-' | '*' | '/' | '%'
→ ‘<’ | ‘>’ | ‘<=’ | ‘>=’ | ‘!=’ | ‘==’
→ ‘&&’ | ‘||’

unary_op → ‘-’ | ‘!’

Standard mathematical rules of precedence and associativity apply (‘%’ has the same associativity as ‘+’). Binary operators are LEFT associative.

Note: All keywords are written in **bold**. All other terminals/tokens are in *italic.*