

# Programming Assignment 3

**Due Date: 11:59 PM on Mar 31, 2008**

For PA3 you will further develop the compiler for a Bronco# programming language. Besides the syntactic errors you have found in PA2, your PA3 program is required to find all semantic errors. The semantic errors include:

- No duplicate names are allowed for variables, types, user-defined type names, and function names at the same level. However, a variable declared at the different levels such as different compound blocks can have a duplicate name. Like 'C', you can declare variables within compound blocks.
- Checking for Undefined variables.
- Within a record, enumeration definition, any two fields or two enumeration values should have different names.
- The three types **int**, **float**, and **short** satisfy the following relation:  $\text{short} < \text{int} < \text{float}$ , where  $<$  represents a subtype relation. If  $A < B$ , then A is called subtype and B is called supertype.
- For an assignment, the type of an expression at RHS should be either the same type or subtype of a variable at LHS.
- For all arithmetic binary operations, if the two subexpressions' types are different, the result type of the expression is the supertype of the two subexpressions' type. Otherwise, an error should be reported.
- For a function call, the type rule for an assignment can be applied for the types between a real argument and its corresponding parameter.
- The other requirements imposed on Bronco# should be the same as the requirements for C/C++.
- There is one and only one *main()* method whose return type is *void* in each input file.

To complete this project, your program must find ***all semantic errors*** based on the above requirements:

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](mailto:bilal.abubakr@wmich.edu), and cc to [zjiang.yang@wmich.edu](mailto:zjiang.yang@wmich.edu).

with Subject: *Course Number:PA3: Your Name.*

## INPUT GRAMMAR FOR PA2

### Module

|        |   |   |
|--------|---|---|
| module | → unit { unit }   | //{...} represents 0 or many repetitions                |
| unit   | → declaration ‘;’<br>→ function<br>→ type_decl ‘;’<br>→ λ | //global variables<br>//functions<br>//types<br>// NULL |

### Type Declarations

|            |  |
|------------|--|
| type_decl  | → <b>type</b> <i>ident</i> ‘=’ type_def  |
| type_def   | → <b>array</b> ‘[’ int_const “..” int_const ‘]’ of type_id<br>→ <b>record</b> field_list <b>end</b><br>→ <b>enum</b> ‘{’ ident_list ‘}’<br>→ <b>pointer of</b> type_id |
| field_list | → field { ‘;’ field }  |
| field      | → type_id ident_list   |
| type_id    | → <i>ident</i>   base_type   |
| base_type  | → <b>int</b>   <b>float</b>   <b>short</b>   |

// ‘ident’ is a defined type  
// short takes 2 bytes and others take 4 bytes

### Variable Declarations

|                  |                                     |
|------------------|-------------------------------------|
| declaration_list | → declaration { ‘;’ declaration }   |
| declaration      | → type_id ident_list                |
| ident_list       | → <i>ident</i> { ‘;’ <i>ident</i> } |

### Function Declarations

|                |   |
|----------------|---|
| function       | → <b>function</b> return_type ident ‘(’ parameter_list ‘)’<br>compound_stmt |
| return_type    | → type_id   <b>void</b>   |
| parameter_list | → parameter { ‘;’ parameter }   λ   |
| parameter      | → mode type_id <i>ident</i>   |
| mode           | → <b>ref</b>   λ  |

//Function returns nothing

### Statements

|           |   |
|-----------|---|
| stmt_list | → statement { ‘;’ statement }                                 |
| statement | → assign_stmt<br>→ while_stmt<br>→ if_stmt<br>→ function_call |

//Must return VOID

→ compound\_stmt  
 → return\_stmt  
 → print\_stmt  
 → new\_stmt  
 → λ //NULL statement

assign\_stmt → variable “=” expr  
 while\_stmt → **while** ‘(‘ expr ‘)’ statement  
 if\_stmt → **if** ‘(‘ expr ‘)’ statement  
           → **if** ‘(‘ expr ‘)’ statement **else** statement  
 function\_call → *ident* ‘(‘ expr\_list ‘)’  
 compound\_stmt → ‘{‘ declaration\_list ‘;’ stmt\_list ‘}’  
                   → ‘{‘ stmt\_list ‘}’  
 return\_stmt → **return** ‘(‘ expr ‘)’ | **return**  
 print\_stmt → **print** ‘(‘ output\_list ‘)’  
             → **println** ‘(‘ output\_list ‘)’ | **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  
                   → ‘.’ *ident* variable\_tail //Ident is defined field  
                   → ‘->’ *ident* variable\_tail  
                   → λ

## Expressions

expr → expr bin\_op expr  
       → unary\_op expr  
       → ‘(‘ expr ‘)’  
       → variable  
       → function\_call  
       → *int\_const*  
       → *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*.