Weekly Reading. Ch. 3: 3.5.3; Ch. 5: 5.15.7

Concepts to understand after the readings:

- Basic idea of “axiomatic semantics”
- Review terms: name, address, alias, l-value, r-value, binding, binding time (static vs. dynamic), implicit and explicit type declarations
- Notion of type inference
- Binding lifetime
- Static vs. stack dynamic vs. heap dynamic (explicit and implicit) variables
- Notion of scope (variables) and visibility
- Static vs dynamic scope vs global scope
- Notion of blocks and block-structured languages
- Referencing environment

You should be able to answer the following questions:

- Determine the blocks, block nesting, referencing environments of a MyPL program
- Give an example in MyPL of a function that returns different values depending on whether static or dynamic scoping rules are used. Explain the differences and generally how they work using your example.
- Similar as above but for structured types (i.e., types created using the type reserved word).

Programming Homework: Second Phase of Syntax Analysis.

The two goals of this assignment are to: (a) modify your parser implementation to return an Abstract Syntax Tree (AST) object structure representing the program being analyzed; and (b) to take the AST object structure returned from a parse and navigate it (using the Visitor pattern) to output a “pretty print” version of the corresponding source code. You must use the AST class design provided below (shown graphically and provided via the starter code). When writing your “pretty printer”, you must use the following MyPL code styling rules (also see the test cases provided separately).

1. Indent all statements within a block by two spaces.
2. Each statement should be on a separate line without blank lines before or after the statement. The exceptions to this rule are struct and function declarations.

3. Format variable declarations without explicit types using one space between var and the identifier, and one space before and after the assignment symbol, e.g., var id := expr.

4. Format variable declarations with explicit types the same as for implicit types, but with one space before and after the type name, e.g., var type id := expr.

5. Format variable assignments with one space between set and the identifier, and one space before and after the assignment symbol, e.g., set id := expr.

6. Format type declarations such that the reserved word type and the type name appears on one line, with one space between the two, each variable declaration indented (two spaces from the start of type) and on a separate line (with no blanks between), and end on the next immediate line after the last variable declaration and aligned with type. There should be one blank line before and after the struct declaration. Here is an example:

   type Person
   var age := 0
   var name := ""
   var Person mother := nil
   var Person father := nil
   end

7. Format function declarations such that fun, the return type, the function name, and the parameter list are all on the same line, the body of the function is indented appropriately, and the end is on a separate line, immediately following the last body statement, aligned with the fun reserved word. There should be one space between fun and the type, one space between the type and the function name, and so on. There should be one blank line before and after each function declaration. Here is an example:

   fun int add(int x, int y)
   set sum := x + y
   return sum
   end

8. Format while statements such that while and do occur on the same line, there is one space between the start and end of the Boolean expression, the body of the while loop is appropriately indented, and end is aligned with while and occurs on the line immediately after the last statement of the body.

9. Format for statements such that for and do occur on the same line, each part of the loop specification is separated by a single space, the body of the for loop is appropriately indented, and end is aligned with for and occurs on the line immediately after the last statement of the body.

10. Format if-elif-else statements similar to while statements such that the body of each section is indented, elif and else statements appear on separate lines (with no blank lines before or after), then appears on the same line as its corresponding if or elif, and end statements appear on separate lines with no preceding blank lines.
11. Format simple expressions without any extra spaces. Path expressions should not contain spaces between corresponding dots (e.g., *x.y.z*).

12. Format complex expressions with spaces between their corresponding parts and fully enclose them in parentheses (regardless of whether there were parentheses in the original source code). For example, if the original was written as 3+4+5 the pretty-printed version should be written as *(3 + (4 + 5))*.

13. Boolean expressions should follow the same rules as for complex expressions. For example, `not (x>1) and (y>1)` should print as `not ((x > 1) and (y > 1))`.

14. Format structured type object creation such that there is one space between `new` and the struct type name.

15. Format function calls such that the function name is immediately followed by an opening parenthesis, followed by a comma-separated list of expressions, followed by a closing parenthesis. There should be one space after each comma.

Note that because we are going to be implementing associativity and precedence of operators, some *MyPL* expressions can end up with some strange looking parenthesizations. Thus, associativity and precedence will need to be explicitly stated by a programmer via explicit use of parentheses.

**What to Turn In:** You must hand in the following by the due date for your assignment to be considered complete.

- A cover sheet with your name, the assignment number, and the date filled in
- A hard copy print out of your test files
- A hard copy print out showing your code runs correctly over all tests
- A hard copy print out of your “discussion” write up (see cover sheet)
- All program source code submitted through GitHub (instructions provided separately)