Compiler-2018 Project4

1. Changes to my previous parser

- Add some simple checking of semantic performance

2. Ability of parser

The parser will check syntax correctness and semantic correctness.
If an error occurs, it will output error message, otherwise it will output syntax correctness.

Checking semantic correctness

- function
    - At least one function definition in the program
    - First defined function will be treated as the entry point
    - Once a function is declared, it must be defined somewhere after the declaration.
    - A function can be declared or defined only once.
    - A function should be declared or defined before it is invoked.
    - The parameter passing mechanism is call-by-value.
    - Parameters of a function can be either a scalar or array type, while the return value must be a scalar type.
    - Return type matches function declaration/definition.

    - For functions whose return type is not declared as void, its last statement must be a return statement. Early returns are also allowed.
- Variable/Constant Declarations, Initialization
    - Re-assignments to constants are not allowed, and constants cannot be declared in terms of other named constants.
    - In an array declaration, the index must be greater than zero.
    - For variable or array initializations, the type of the left-hand side must be the same as that of the right-hand (after performing type coercion).
    - For array initializations, the number of initializers should be equal to or less than the array size. If there are fewer initializers than elements in the array, the remaining elements are automatically initialized to 0.
- Variable References, Array References and Expressions
    - A variable reference is either an identifier reference or an array reference.
    - The index of array references must be an integer type whose value is equal to or greater than zero. Bound checks are not necessary.
    - Two arrays are considered to be the same type if they have the same number of elements.
    - Array can be used in parameter passing. However, array arithmetic and assignment are not allowed.
    - For arithmetic operators (+, -, *, or /), the operands’

    types must be either integer, float or double, and the operation produces an integer, float or double value. The resulted type is the “most general” type of the two operands.
  - For the modulo operator %, both operands must be integers, and the operation produces an integer value.
  - For logical operators (&&, jj, or !), the operands must be booleans, and the operation produce a Boolean value.
  - For relational operators (<, <=, >=, >, !=, ==), the operands must be integer, float or double types, and the operation produces a Boolean value. For equality comparison operators (== or !=), they must additionally support booleans as operand types.
  - There are no operators that operate on String values, and thus they can only appear in assignment operations, variable declarations, print statements, read statements, function arguments, or return statements.
- Type Coercion
  - An integer (float) type can be implicitly promoted into a float/double (double) type anywhere the latter is expected.
- Compound Statement
  - A compound statement forms an inner scope. Note that declarations inside a compound statement are local to the statements in the block and no longer exist after the block exits.
- Simple Statement

    - Variable references in print and read statements must be scalar type.
    - In assignment statements, the type of the left-hand side must be the same as that of the right-handside (after performing type coercion).
- Conditional and While Statement
    - The conditional expression part of if and while statements must be Boolean types.
- For Statement
    - There are three components of a for loop: initial expression, control expression and increment expression. Variables used in the above three component should be declared before the for statement. The control expression must be of boolean type.
- Jump Statement
    - break and continue can only appear in loop statements.
- function invocation
    - A procedure is a function that has no return value.
    - The types of the actual parameters must be identical to the formal parameters in the function declaration/definition.
    - The number of the actual parameters must be identical to the function declaration/definition.
    - The type of the return value must be identical to the types of the return type in function declaration/definition.
    - One may discard the return value of a function call.

The output symbol table format will be like:

```
printf("==================================================
=========\n");
// Name [29 blanks] Kind [7 blanks] Level [7 blank] Type [
15 blanks] Attribute [15 blanks]
printf("Name Kind Level Type Attribute \n");
printf("--------------------------------------------------
---------\n");
printf{"a function 0(global) int int[2],float\n");
// ....
// Format of Attribute: type,type,type,...
printf("==================================================
=========\n");
```

The error message format will be like:

```
##########Error at Line #N: ERROR MESSAGE.##########
```

If there is no error, output:

```
|-------------------------------------------|
| There is no syntactic and semantic error! |
|-------------------------------------------|
```

3. Platform to run scanner/parser

Use **lex** and **Yacc** to implement scanner/parser

build and execute in Linux/Unix system

Take Ubuntu as example

- Install Flex/Lex and Bison/Yacc

```
% sudo apt-get install Bison flex
```

3. How to run my code?

- To run my scanner/parser, type

```
% make
% ./parser [inputfile]
```

- To delete files except `lex.l` `yacc.y` `Makefile` `SymbolTable.c` `SymbolTable.h`, type

```
% make clean
```