Q. What is the output of “test_parser”?
A. The output of the parser must be an AST in textual or graphical form as well as the symbol table.

Q. What to do, and what not to do in the parsing phase?
A.
What the parser has to do: syntactical checks
1. Build and construct nested symbol tables. Each module has a global symbol table and each subroutine (procedure and function) has its local symbol table. The parser has to save each symbol with its proper symbol type into the correct symbol table between local symbol table and global symbol table.

2. Matching identifier declaration (variables, function names, procedure names). In other words, we can not use any variable which is not declared in the local scope or global scope. We also can not re-define any variables in the same scope.
For example)

```
var a, b, c, a // the definition of the last variable 'a' is a parsing error.
```

3. The module name checking can be done by the parser or the type system, i.e., module declaration name and end name have to be same, but make sure your parser to check this error in the parsing phase as the reference parser checks this error.
For example)

```
module fibonacci;
... statements ...
end fibonacci2. // this is an error.
```

Don't do semantic checks in your parser.
The parser does not perform type (semantic) checking. The examples of the type checking are explained in the below. The details of the type checking will be discussed in later lectures. You may lose your points for your parser performing the followings.

– checking the types of the operands in expressions
– checking the types of the LHS and RHS in assignments
– the return type matching
– checking the types of precedure/function arguments (including an array type)
: also do not implement type conversion between an array type and the pointer of the array type for a function call.
– checking the number of precedure/function arguments
– catching invalid constants (value range checking)

Q. What is the procedure's return type?
A. Functions have their own return type. Procedures do have any return type. Your parser can consider every procedure's return type is NULL. You may use the provided TypeManger (type.cpp/h)

Q. How to deal with the pre-defined open arrays and I/O functions?
A. In the SnuPL/1 programs (refer to the project overview), we have pre-defined functions, 1. DIM/DOFS are used to deal with open arrays, and 2. I/O functions (ReedInt, WriteInt, WriteChar, WriteStr, WriteLn).

The parser assumes that these functions are defined. The implementation and the linking of the functions will be provided in later phases. Therefore, your parser needs to add the functions into the global symbol table. This should be implemented in the InitSymbolTable() function in parser.cpp. In addition, your parser needs to call this function somewhere in the code (maybe in module() in parser.cpp). Actually this is pretty simple. You just need to add the symbols of the functions into the global symbol table.

Q. How to deal with unary '+' operation ahead of boolean variable?
A. The The SnuPL/1 syntax (simpleexpr = [“+” | “-”] term { termOp factor }) allows expressions such as “A := +1;” or “A := +true”. Thus the '+' can become a unary operation (see ir.cpp/h). For '+' ahead of a boolean variable, the type system must catch a type error (not in parsing phase). Thus, do not ignore the unary operation ('+') in this phase so that the type check can check the type matching.

Last modification: 26th September 2017