Supplementary Documentation for Phase2: Parsing

Q. What is the output of "test_parser"?
The output of the parser should be an AST in textual or graphical form.

Q. How to handle ambiguous boundaries between parser phase and type checking phase?

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. And save each symbol with its proper symbol type into the proper symbol table (local symbol table, global symbol table).

3. Identifier declaration (variables, function names, procedure names) matching. 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 // last variable 'a' definition is parsing error.

What the type checking has to do: Semantic checks

1. module name checking **(you can do this also in the parser phase)**
: module declaration name and end name have to be same.
For example) module fibonacci;
… statements ...
end fibonacci2. // a type error.

2. variable "value range" checking **(don't do this in the parser phase)**
: it will be discussed in later lectures

3. type matching: "return/assign" type matching **(don't do this in the parser phase)**
: it will be discussed in later lectures

Q. What is the parameter types?

There is no any type definition in formalParam.
In SnuPL/0, all parameters saved as 'integer' type. (for simplified grammar)

Q. How do we handle unary '+' operation ahead of boolean variable?

In the skeleton code, there are some unary operations such as "-" (negation) and "!" (not). But, there is no unary '+' operation.
In the SnuPL/0 syntax, the grammar (simpleexpr = ["+" | "-"] term { termOp factor }) allows expressions

like “A := +1;” or “A := +true”. At that time, the '+' becomes unary operation.
For the unary '+' ahead of an integer or an integer variable, we can just consume (ignore) the '+' operation (e.g., A = +B;). But, in the type system, the unary '+' operation ahead of boolean variable (“A := + true”) is an error. So, we can not consume or ignore the '+' operation because the next type checking system have to be able to check the type error.
Therefore, we need one additional unary operation ('+') so that the next type checker can check its type matching. You need to fix some codes (including ir.cpp/h) to support unary '+' behavior.

(Our previous reference codes just consumed the unary '+' operation. And we realized the codes contain this bug thanks to a student from our class. We are fixing our reference codes as well.)

Q. What is the procedure type?

Each function has its return type. Otherwise procedure does not have any return type. You can just say every procedure return type is NULL using provided TypeManger (type.cpp/h)

Q. What are Input() and Output()?

In the SnuPL/0 example (fibonacci example), you can find that the code has Input() and Output() function without any subroutine declaration.

SnuPL/0 provides input/output through two implicitly defined functions/procedures:
- function Input(): integer

reads and returns an integer value from stdin
- procedure Output(x);

prints integer value ‘x’ to stdout

The compiler must assume that these two functions are defined. An implementation will be provided at later phase. It also should be linked to the compiled code at later phases.

Actually, what we have to do is very simple. We just need to add two (Input() and Output()) functions into the global symbol table. And it is implemented in the InitSymbolTable() function in the parser.cpp.
Therefore, what you have to do is execute InitSymbolTable() function when you make the global symbol table. Even if you didn't consider this, we won't give no penalty for this because we didn't tell you about this before.

Do you have more questions or difficulties?
All questions are always welcome.
And please use eTL forum efficiently.