COSC 461, COSC 561 **Bowie State Univ.**
Summer-I 2021

Project: Phase II

Undergraduate Requirements: You are to write a parser for the language SIMPL. The LL(1) grammar for SIMPL can be found in the SIMPL documentation. Although tree-building symbols are included, you will have to specify what node types are being built. After finishing the grammar you will need to build the parse table. Then you can begin coding your grammar (I advise you to show me your grammar and parse table before you start coding). You will also need to write a driver program for your parser, which just calls the parser and prints out the resulting syntax tree using the procedure *printTree*.

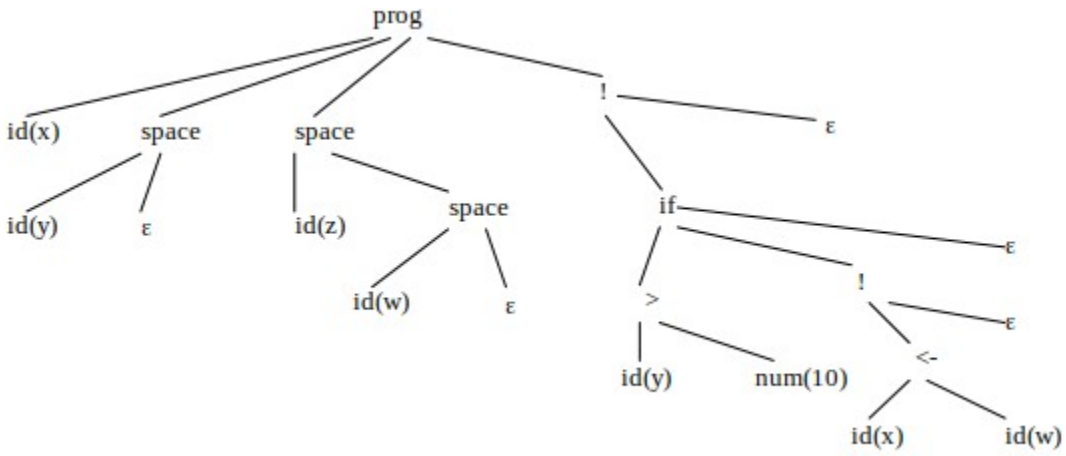
Your parser will use the lexical scanner of Phase I. As with the scanner, your parser should output to standard output, and print any error messages to standard error. Error handling in SIMPL is almost nonexistent. When your compiler encounters a lexical or syntax error it should print an appropriate error message, and abort.

The form of the SIMPL syntax tree is given in the SIMPL documentation. The node listed as type NULL is actually the ␣ node. To summarize the documentation, you will have the following node types:

quaternary: *prog*
ternary: *if*
binary: <-, >, =, *, /, +, -, *space* (a cons operator), ! (a cons operator), *while*
nullary: ␣, *id*, *num*

As an example of a syntax tree, the following SIMPL program produces the following syntax tree:

```
x(y) |z w| if y > 10? x <- w ! : !!
```



Your parser should print the syntax tree using the procedure *printTree*. *PrintTree* can be found in the SIMPL distribution package. In order to use *printTree*, you must use the definition for the *Treenode* class given in the C++ source file. *PrintTree* prints the syntax tree as a sequence of parenthesized expressions. The best way to explain this is by example. The above example syntax tree would be printed as:

```
(prog:1 (x:13) (space:10 (y:13) (epsilon:12)) (space:10 (z:13)
(space:10 (w:13) (epsilon:12))) (!:11 (if:2 (>:4 (y:13) (10:14))
```

```
(!:11   (<-:3   (x:13)   (w:13))   (epsilon:12))   (epsilon:12))
(epsilon:12)))
```

In this expression each node is printed out as a pair of matched parenthesis, enclosing its name, followed by a ":", followed by its type, and then its children.

You will turn in a copy of your grammar, a copy of your parse table, and a listing of your program. You will also demonstrate your program to me. The test files for the demonstration are in the usual directory.

Graduate Requirements: You are to write a parser for the language *notSo*. To do this you must first design a LL(1) grammar for the language, complete with tree building symbols. As for the undergraduate students, you will need to build the parse table, and code the grammar and parser driver, which uses *printTree*.

Your parser will use the lexical scanner of Phase I. The scanner must now be modified so that it does not return the comment token type. Errors should be handled using the scan-set and panic mode method described in class.

The form of the *notSo* syntax tree is given in the *notSo* documentation. In addition to the node types for SIMPL, you will have the folloing types:

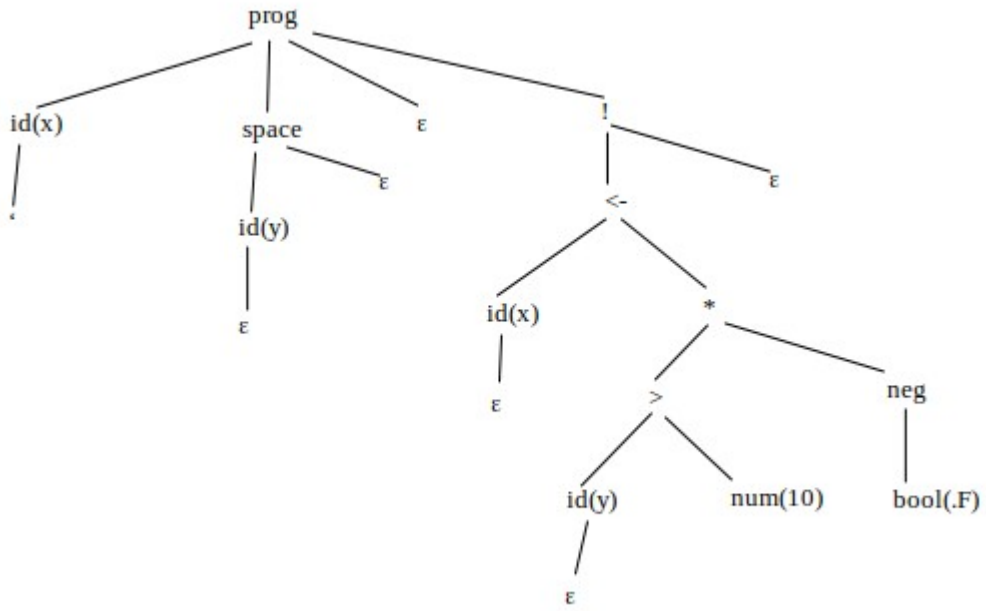
unary: *neg*, *id*

nullary: *bool*, ‘ (a boolean identifier operator)

Notice that the type *id* is now a unary node. As an example, the *notSo* program:

```
x’bool(y) || x <- (y > 10) * -.F!!
```

would form the following syntax tree:



It would be printed by *printTree* as:

```
(prog:24 (x:19 (‘:3)) (space:10 (y:19 (epsilon:16)) (epsilon:16))
(epsilon:16)  (!:18  (<-:4  (x:19  (epsilon:16))  (*:21  (>:23  (y:19
(epsilon:16)) (10:20)) (neg:6 (.F:7)))) (epsilon:16)))
```