From project description: The symbol table (ST) is a data structure with an entry for each identifier in the language. Associated with each identifier you will store all relevant information (identifier attributes). For example, for:

-- simple variable, store type
-- array, store type, the number of dimensions and range of each dimension
-- record (struct), store each field (member)
-- function, store return type as well as a parameter list
-- etc.

As the parser recognizes proper groups of tokens, you have it perform actions, essentially displaying the proper code in the target language. Those actions generate the proper code at the appropriate time. In this phase you will implement some of the symbol table and some code generation.

Design your ST using object-oriented design (encapsulation, inheritance, polymorphism). Minimally, your interface will include the functions: lookup or retrieve, insert, functions that deal with scope (scopeEntry, scopeExit), and printST. You may have others, but if they are utility functions, not interface, they would be private in a good design.

Put the actions (code for allocating memory, constructing objects, pushing identifiers on to a stack in a list, function calls to lookup, insert, etc.) directly in the grammar (yacc/bison) when you encounter an identifier (yident) or in the appropriate place in the rule.

Remove some of your couts of identifiers and add couts so that Pascal is translated to C++ code. Do this for all constants and for primitive variables such as integer (int), real (float or double), char (char), and Boolean (bool). You do not need to incorporate error checking yet (although you may if you’d like).

The printST

- If you build one large data structure (lists of lists) containing all identifiers, then call printST at the end of your program. If you lose information when you exit scope, then call printST when you leave a scope.

- In the printST function, display the name of the function and a separator line at the entry of each scope. For example, for proc1, display the name and a line made up of 75 dashes:

```
ENTER proc1
-------------------------------------------------------------------------------------------------
```

When you exit scope, printST should display the name and a line made up of 75 equals:

```
EXIT proc1
-------------------------------------------------------------------------------------------------
```

Deliverables

- Files to compile and run your code to make (using your makefile) executable a.out.

- You will write a readme.txt file that briefly explains your design including how you store information. Which technique do you use to handle scope? Include anything else you need to tell me. Also briefly describe who did what on the implementation.