Symbol Table Management Procedures

Implement a set of symbol table management routines for a block structured language. Although you may use any data structures that you like, a linked-list of unbalanced binary search trees is suggested. The procedures should include:

- **SYMINIT**, which will initialize the symbol table,
- **SENTBLOCK**, called when entering a new block,
- **SLEVBLOCK**, called when leaving a block,
- **SADDSYM**, to add a symbol to the table, and
- **SLOOKUP**, to look up a symbol.

For code generation, you will need to return the static nesting of the symbol as well as its location (runtime address). Thus, you should include such information either with each symbol or in the heading of the symbol table for each block. In addition, you should try to keep all the necessary information for code generation. For CONST declarations, the type and value are important. For VAR declarations, the type and location are important. For procedures and functions, the number and types of parameters as well as the location are important. Since you don't know everything that will be needed, you cannot select a final data structure yet. If you implement the information as a record that can be modified later you will not go too far wrong.

To test your symbol table routines, integrate them into your parser and, under the compiler directive(s), write out the entire symbol table immediately upon finishing the VAR section of a procedure, function or program, and immediately upon exiting the procedure, function or program. For each symbol table entry, you will print out all the symbol table information pertaining to that entry. Moreover, make sure that I know the lexical scoping, etc. if they are not stored with each symbol table entry. For this assignment set the default for this compiler option to be on.

Any errors that you encounter should be worded with the end user in mind. The test files will be ST*.inp and will be placed on the class website.