Compilers
Homework #7
Due: Friday, February 3, 2023

This assignment the completion of the first part of the your semantic analysis phase of the TOY compiler. By now you should be able to produce a parse tree (actually, we are producing what is typically called an Abstract Syntax Tree since we are not included nodes for every grammar production).

Symbol Table

This portion of the semantic analysis phase should construct a symbol table for each scope (block). Each symbol table should contain

`name`	The name of the symbol table (when the block is the body of a method). Anonymous blocks will not have a name.
`parent`	A reference to the parent symbol table (the symbol table for the immediately enclosing block).
`symbols`	A hash table mapping names to symbol table entries.

There will be three kinds of symbol table entries: variables, parameters and methods. For variables the symbol table entry will contain

`name`	the name of the variable
`type*`	one of `int, char, bool, int[], char[], bool[]`
`size`	for array types, the size of the array

For variables the symbol table entry will contain

`name`	the name of the parameter
`type*`	one of `int, char, bool, int[], char[], bool[]`

For methods the symbol table entry will contain

`name`	the name of the method
`type*`	the result type; one of `void, int, char, bool, int[], char[], bool[]`
`params`	an array of the types of the parameters

Note that the symbol table entries for the parameters themselves will be in the symbol table for the method; the symbol table entry for the method will be in the symbol table for the scope which encloses the method declaration.

*You will undoubtedly want to have a class Type that encapsulates all the information you need about a type:

```
public class Type {
    public static enum Kind { INT, CHAR, BOOL, VOID }
    Kind kind;
}

class ArrayType extends Type {
    int size;
}
```

Traversal

Your semantic analysis phase will perform a traversal of the parse tree and perform various actions as it encounters different kinds of nodes. Some actions will be performed in pre-order (i.e. before traversing the children) and some actions will be performed in post-order (i.e., after traversing the children).

Block* (pre-action):

As you encounter each block, you should create a new symbol table linking it to the symbol table for the enclosing slope. As new declarations (variables, parameters and methods) are encountered during the subsequent traversal of the block, symbol table entries should be created and added to the symbol table for the current block.

*Only for anonymous blocks. The pre-action for the block associated with a method declaration is described below under Method Declarations.

Block (post-action):

As you exit each block, display the symbol table by displaying the name of the table (i.e., the name of the block if it has one) and then all of the symbols that are contained in the symbol table. Suggested format:

```
Scope: print
    Param s: char[]
    Variable i: int
    Variable c: char
    Variable buffer: char[10]
    Method output: int, int, char
```

The parent symbol table will now be restored as the current symbol table.

Variable Declaration (pre-action):

Create a new symbol table entry for this variable in the symbol table for the current scope. If the name is already defined, produce an error message instead. The type information can be inferred from the appropriate children of the variable declaration node.

Parameter Declaration (pre-action):

Create a new symbol table entry for this parameter in the symbol table for the current scope. If the name is already defined, produce an error message instead. The type information can be inferred from the appropriate children of the variable declaration node.

Method Declaration (pre-action):

Create a new symbol table entry for this method in the symbol table for the current scope. If the name is already defined, produce an error message instead. One of the children of this node will contain the list of parameters for this method. Traverse this list of parameters to build the array of parameter types to be included in this symbol table entry.

The next step is to create a new symbol table for the block associated with the method. Its parent symbol table will be the currently active symbol table. You will then need tor traverse the list of parameters once again adding each parameter declaration to this newly created symbol table. Before traversing the block for this method, make this newly created symbol table the current symbol table.

This symbol table will be “popped” off the stack of symbol tables when the post-action for the block is executed.

Identifier (pre-action):

Look up the identifier name in current stack of symbol tables. If a symbol table entry is found, annotate this node in the parse tree with a reference to the symbol table entry found. If no symbol table entry is found, produce an error message indicating that this is an undefined symbol.

Upcoming

The next part of the semantic analysis phase will be annotate the tree with type information and to make sure that usages of names are of the correct kind and type (e.g., the user is not trying to call an integer variable; that the user does not try to store an integer value in a boolean expression, etc).