Introduction to Compiler Design by Prof. Yi-Ping You

Project Assignment 3

Symbol Table Construction

Due Date: 23:59, December 5, 2018

Your assignment is to construct symbol tables in the parser. In the previous project, you have done syntactic definition and created a parser using **yacc**. In this assignment you will construct symbol tables for performing some simple checking of semantic correctness in the next project.

1 Assignment

You need to write your symbol table, which should be able to perform the following tasks:

- Push a symbol table when entering a scope and pop it when exiting the scope.
- Insert entries for variables, constants, procedure, and function declarations/definitions.
- Lookup entries in the symbol table.

You then must extend your LALR(1) grammar using **yacc**.

1.1 What to Submit

You should submit the following items:

- revised version of your lex scanner
- revised version of your yacc parser
- report: a file describing what changes you have to make to your scanner/parser since the previous version you turned in, the abilities of your parser, the platform to run your parser and how to run your parser.
- a Makefile in which the name of the output executable file must be named 'parser' (**Please make sure that it works well. TAs will build your parser with this makefile. No further grading will be made if the** *make* **process fails or the executable** *parser* **is not found.**)

1.2 Pragma Directives

In the first assignment, we have defined:

`#pragma source`	`#pragma source on` turns on source program listing, and `#pragma source off` turns it off.
`#pragma token`	`#pragma token on` turns on token (which will be returned to the parser) listing, and `#pragma source off` turns it off.
`#pragma statistic`	`#pragma statistic on` turns on identifier frequencies listing on, and `#pragma statistic off` turns it off.

One more option will be added:

`#pragma symbol` `#pragma symbol on` dumps the contents of the symbol table associated with a block when exiting from that block, and `#pragma symbol off` turns it off.

The format of symbol table information is defined in Section 3.1.

Note that the default of this four #pragma directives are off in this assignment.

1.3 Implementation Notes

You should maintain a symbol table per scope in your parser. Each entry of a symbol table is an identifier associated with its attributes, such as its name, name type (function name, parameter name, variable name, or constant name), scope (local or global), type (variable or constant's data type, function's parameter types or function's return type), constant's value, etc. In effect, the role of a symbol table is to pass information from declarations/definitions to uses. A semantic action "puts" information about identifier x into the symbol table when the declaration of x is analyzed. Subsequently, a semantic action associated with a production such as $factor \rightarrow \mathbf{id}$ "gets" information about the identifier from the symbol table.

2 Semantic Definitions

2.1 Scope Rules

- Scope rules are similar to C.
- Names must be unique within a given scope. Within the inner scope, the identifier designates the entity declared in the inner scope; the entity declared in the outer scope is hidden within the inner scope.
- A compound statement forms an inner scope. Note that declarations inside a compound statement are local to the statements in the block and no longer exist after the block exits.

2.2 Identifier

Only the first 32 characters are significant. That is, the additional part of an identifier will be discarded by the parser.

3 What Should Your Parser Do?

Your parser should dump the symbol tables when `#pragma symbol on` is set.

3.1 More About Symbol Tables

Each entry of a symbol table consists of the name, kind, scope level, type, value, and additional attributes of a symbol. Symbols are placed in the symbol table in order of their appearance in the input file. Precise definitions of each symbol table entry are as follows.

Name	The name of the symbol. Each symbol have the length between 1 to 32.
Kind	The name type of the symbol. There are four kinds of symbols: function, parameter, variable, and constant.
Level	The scope level of the symbol. 0 represents global scope, local scope starts from 1, 2, 3, ...
Type	The type of the symbol. Each symbol is of types int, float, double, bool, string, or the signature of an array. (Note that this field can be used for the return type of a function)
Attribute	Other attributes of the symbol, such as the value of a constant, list of the types of the formal parameters of a function, etc.

For example, given the input:

```
 1: #pragma symbol on
 2: bool func(int a, float b);
 3: const int d = 1;
 4: void main(){
 5:     int x;
 6:     int y[10];
 7:
 8:     if(y[0] == 0){
 9:         float x;      //outer 'x' has been hidden in this scope
10:         double z;
11:         z = 0.1;
12:     }
13: }
14:
15: bool func(int a, float b){
16:     string c = "hello world";
17:     return (b <= 1.0);
18: }
```

After parsing the compound statement in function `main` (at line 12), you should output the symbol table with the following format.

```
Name    Kind         Level      Type         Attribute
------------------------------------------------------------
x       variable     2(local)   float
z       variable     2(local)   double
```

After parsing the definition of function `main` (at line 13), you should output the symbol table with the following format.

```
Name    Kind         Level      Type         Attribute
------------------------------------------------------------
x       variable     1(local)   int
y       variable     1(local)   int[10]
```

After parsing the definition of function `func`(at line 18), you should output the symbol table with the following format.

```
Name    Kind         Level      Type         Attribute
------------------------------------------------------------
```

```
a       parameter    1(local)   int
b       parameter    1(local)   float
c       variable     1(local)   string
```

After parsing the end of the program (at line 18), you should output the symbol table with the following format.

```
Name    Kind         Level      Type         Attribute
-------------------------------------------------------
func    function     0(global)  bool         int,float
d       constant     0(global)  int          1
main    function     0(global)  void
```

Here is the template for outputing symbol table.

```
printf("=============================================================================================\n");
 // Name [29 blanks] Kind [7 blanks] Level [7 blank] Type [15 blanks] Attribute [15 blanks]
printf("Name                              Kind       Level       Type               Attribute   \n");
printf("---------------------------------------------------------------------------------------------\n");
printf{"a                                 function   0(global)   int                int[2],float\n");
     // ....
     // Format of Attribute: type,type,type,...
printf("=============================================================================================\n");
```

3.2 Error Detection

The parser should have all the abilities required in the previous project assignment and also the semantic checking ability in this project assignment (i.e., names must be unique within a given scope). Once your parser encounters a semantic error, it should output an error message containing the line number of the error and an **ERROR MESSAGE** describing the error. The **ERROR MESSAGE** in this case will be `SOME_NAME redeclared`. The whole line must be in the format of:

```
##########Error at Line #N: ERROR MESSAGE.##########
```

Notice that semantic errors should **not** cause the parser to stop its execution. You should let the parser keep working on finding as many semantic errors as possible.

4 How to Submit the Assignment?

Create a directory, named "YourID" and store all files of the assignment under the directory. Zip the directory as a single archive, name the archive as "YourID.zip". Upload the zipped file to the **e-Campus (E3)** system.

Note that the penalty for late homework is **15% per day** (weekends count as 1 day). Late homework will not be accepted after sample codes have been posted. In addition, homework assignments must be individual work. If I detect what I consider to be intentional plagiarism in any assignment, the assignment will receive **zero credit**.