Introduction to Compiler Design by Prof. Yi-Ping You

Project Assignment 3

Symbol Table Construction

Due Date: 23:59, December 5, 2017

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 program declarations.
- 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 it works well. TAs will rebuild 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 Pseudocomments

In the first assignment, we have defined:

`S` `&S+` turns on source program listing, and `&S-` turns it off.

`T` `&T+` turns on token (which will be returned to the parser) listing, and `&T-` turns it off.

One more option will be added:

`D` Dump the contents of the symbol table associated with a block when exiting from that block if D is on (i.e. `&D+`).

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

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 (program name, 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 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 $factor \rightarrow$ **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 is exited.

2.2 For Loop

- A counting iterative control statement has a variable, called the **loop variable**, in which the count value is maintained. The scope of the loop variable is the range of the loop.
- The variable is implicitly declared at the **for** statement and implicitly undeclared after loop termination.
- In a nested loop, each loop must maintain a distinct loop variable. In the same way, there should not be a varible with same name declared in the for statement.
- The loop variable should not apear in the symbol table dump.

2.3 identifier

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

3 What Should Your Parser Do?

Your parser should dump the symbol tables when `pseudocomment D` is set.

Notice that semantic errors should not cause the parser to stop its execution. You should let the parser keep working on finding semantic errors as much as possible. An error should be reported with the following format:

```
char error_msg[] = "symbol 'a' redeclared";
printf("<Error> found in Line %d: %s\n", linenum, error_msg);
```

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 five kinds of symbols: program, 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 integer, real, boolean, 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: test;
 2:
 3: // no global declaration(s)
 4:
 5: func( a:integer ; b:array 1 to 2 of array 2 to 4 of real ): boolean;
 6: begin
 7:         var c: "hello world!";
 8:         begin
 9:                 var d: real;
10:                 return (b[1][4] >= 1.0);
11:         end
12: end
13: end func
14:
15: begin
16:         var a: integer;
17:         begin
18:                 var a: boolean;  // outer 'a' has been hidden in this scope
19:         end
20: end
21: end test
```

After parsing the compound statement in function `func` (at line 11), you should output the symbol table with the following format.

```
Name    Kind         Level      Type         Attribute
------------------------------------------------------------
d       variable    2(local)    real
------------------------------------------------------------
```

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

```
Name    Kind         Level      Type         Attribute
```

```
------------------------------------------------------------
a       parameter   1(local)    integer
b       parameter   1(local)    real [2][3]
c       constant    1(local)    string       "hello world!"
------------------------------------------------------------
```

After parsing the compound statement in the main block (at line 19), you should output the symbol table with the following format.

```
Name    Kind         Level      Type         Attribute
------------------------------------------------------------
a       variable    2(local)    boolean
------------------------------------------------------------
```

After parsing the main block (at line 20), you should output the symbol table with the following format.

```
Name    Kind         Level      Type         Attribute
------------------------------------------------------------
a       variable    1(local)    integer
------------------------------------------------------------
```

After parsing the program's definition (at line 21), you should output the symbol table with the following format.

```
Name    Kind         Level      Type         Attribute
------------------------------------------------------------
test    program     0(global)   void
func    function    0(global)   boolean      integer, real [2][3]
------------------------------------------------------------
```

The output format of symbol table is as follows:

```
void dumpsymbol()
{
    int i;
    for(i=0;i< 110;i++)
        printf("=");
    printf("\n");
    printf("%-33s%-11s%-11s%-17s%-11s\n","Name","Kind","Level","Type","Attribute");
    for(i=0;i< 110;i++)
        printf("-");
    printf("\n");
    {
        printf("%-33s", "func");
        printf("%-11s", "function");
        printf("%d%-10s", 0,"(global)");
        printf("%-17s", "boolean");
        printf("%-11s", "integer, real [2][3]");
        printf("\n");
    }
    for(i=0;i< 110;i++)
```

```
        printf("-");
    printf("\n");

}
```

Sample(s) will be later announced at the course forum.

3.2 Error Detection

The parser should have all the abilities required in the previous project assignment and also the semantic checking ability (i.e., names must be unique within a given scope) in this project assignment. 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`. Notice that semantic errors should **not** cause the parser to stop its execution. You should let the parser keep working on finding semantic errors as much as possible. An error should be reported with the following format:

```
char error_msg[] = "symbol 'a' is redeclared";
printf("<Error> found in Line %d: %s\n", linenum, error_msg);
```

4 How to Submit the Assignment?

You should create a directory, named "YourID" and store all files of the assignment under the directory. Once you are ready to submit your program, zip the directory as a single archive, name the archive as "YourID.zip", and upload it 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 reduced or, usually, **zero credit**.