Compiler Project 3

Project: C- compiler semantic analyzer — symbol table

Date: November 14, 2018

Platform: FreeBSD / Linux 14.04

How to run:

1. Use make to compile the scanner and the parser

```
waylon@waylon-VirtualBox: /media/sf_ubuntu_share
waylon@waylon-VirtualBox:/media/sf_ubuntu_share$ make
```

2. Execute the parser with testfile(test)

```
waylon@waylon-VirtualBox: /media/sf_ubuntu_share
waylon@waylon-VirtualBox:/media/sf_ubuntu_share$ make
yacc -v -d parser.y
gcc -g -c y.tab.c
gcc -g -o parser symbol_table.o y.tab.o lex.yy.o main.o -lfl
waylon@waylon-VirtualBox:/media/sf_ubuntu_share$ ./parser test
```

3. Get the result

```
waylon@waylon-VirtualBox: /media/sf_ubuntu_share
==========================================================================================
Name                            Kind        Level       Type                Attribute
------------------------------------------------------------------------------------------
a                               variable    0(global)   int
b                               variable    0(global)   int
c                               constant    0(global)   int                 1
d                               constant    0(global)   int                 0
e                               variable    0(global)   float
f                               variable    0(global)   float
h                               constant    0(global)   double              6.62607e-34
pi                              constant    0(global)   float               3.14159
s                               variable    0(global)   string
t                               constant    0(global)   string              this is a const string
flag                            variable    0(global)   bool[1][2][3]
test                            function    0(global)   int                 int[2][2],int
main                            function    0(global)   int
funct                           function    0(global)   void                int[2][2],int[3]
nothing                         function    0(global)   void
==========================================================================================

|-------------------------------|
|  There is no syntactic error!  |
|-------------------------------|
```

4. If the redeclaration of a identifier is detected, the analyzer will print the error message and keep parsing.

```
##########Error at Line #41: a redeclared.##########
##########Error at Line #41: b redeclared.##########
```

5. Pragma is for compiler options. The symbol option enables printing symbol tables. All options are enabled by default.

```
#pragma source on
#pragma token off
#pragma statistic off
#pragma symbol on

int a, b;
int a, b[6] = { 123, 456, 789, 123, 456, 789 };
const int c = true, d = false;
```

Abilities:

With the scanner and the parser from TA, I added the symbol table to the compiler, which specifies the identifiers in different scope levels.

Moreover, the redeclaration detection is also added in this program.

Modifications:

1. Add symbol table source file and header file.

2. Pass literal constant values and id names to yylval in lex.
3. Assign types to declaration-relative tokens and nonterminals in yacc.
4. Push a symbol table when entering a scope and pop it when exiting the scope.
5. Insert entries for variables, constants, parameters, and function declarations/definitions.
6. Lookup entries in the symbol table.