LAB 7 -- add symbol table and type checking

Submit Assignment

Due Apr 17 by 5:30pm **Points** 40 **Submitting** a file upload **File Types** zip and tar

Click **HERE** and **HERE** to get my symbol table code

In this lab you are to add your symbol table code to your semantic action set. The goal is to ensure that used variables have been declared.
Additionally, you will assign new symbols to intermediate values in expressions. This will allow us to allocate memory for the intermediate expressions.
Finally, we will add rudimentary type checking so that we don't mix VOID and INTs.

There are a number of details required in this lab. You will need to maintain how and when to add to the symbol table, when to remove from the symbol table, how to maintain offset values

The rules for insertion is as follows:

a) You need to maintain a static scope counter, called "level" in your grammar (YACC) as a global variable
b) Each time you enter a compound statement, you need to increment level
c) Each time you exit a compound statement you need to decrement the level, whenever you exit a compound statement, you need to remove all the symbols defined at that level and reset your offset by the amount of memory words you removed with the deletion. Here are some sample code that may help

Some of my semantic action for FUNDEC on exit is....

```
offset-=Delete(1);  /* remove all the symbols from what we put in from the function call*/
level=0;  /* reset the level */
offset=goffset;
```

My semantic action for end of compound statement

```
{  $$=ASTCreateNode(BLOCK);
   $$->s1=$2;
```

```
                $$->s2=$3;
                Display();  /* display symbol table as per requirement */
                offset-=Delete(level);  /* decrease the offset count by the size of values allocated at level
*/
                level--;  /* down one level */
               }
```

d) Whenever a variable is declared, it must not have been declared before at that level.
-- if a variable is "new" to this level, then you need to insert the variable into the symbol table, along with the level and its size.
e) each variable insertion also comes with a stored offset. The offset is initially set to 0 and indicates how far into the runtime stack the variable will be found. The offset is incremented by the size of the variable. 1 for scalars, the size of an array for arrays, and 0 for functions.
f) the offset starts at 0 for the start of a function and the function paramaters start taking up offset. When the function is done, we must reset the main offset to a stored global offset (I call mine goffset).

The rules for use of variables are:
a) Whenever a variable is used, it must occur at your level, or any level less, in a decreasing manner.
b) If a variable is not defined, then error.
c) If a variable is either scalar, array or function, then variable name must be used in the correct context. This means that you cannot have a variable name as an INT be used as a function -- this is a form of type checking in that the variable name has to be used in the correct context.

For example

```
void main(void)
begin
  int x;
 x[0] =  1;  -- is illegal since x is not an array
 x= x(10);  -- is illegal (x is not a function)
end
```

To make your symbol table work, you will need to update the actions in AT LEAST the following production rules:
VARDEC
FUNDEC
PARAMS
Wherever ID occurs.

Rules for Type checking:

1) an expression inherits its type from the operations that occurs. You should ***NOT*** allow things like

```
int x;
void y;
x= x + y;
```

2) Each Number is an integer. Each ID has its declared type. Remember that if the ID is a function, then it has a type.

3) Each of the formal parameters of a function should match the length and type of argument list of the function.

4) Types have to agree for assignment statements.

REMINDERS

When you add semantic actions in the middle of a Right Hand side, remember to increment your $$/$1 references so you are still referencing the return values from the subordinate production rules

Symbol Table changes

1) The SymTable structs does not need "label", remove it

The SymTable struct for symbol needs to be a char* not fixed at size 10. Since LEX is doing a "strdup" from the heap, you can change the array in Symtable to a character pointer in your insert, just do an assignment.

You need to add and maintain an element to the structure to identify the type of element (currently INT/VOID)

You need to add an element to the Symbol Table structure to keep track if it is a function

You need to be able to store information to know if it is an array. And if it is an array, the size of the array

For functions, You need to store the number of parameters when declared (or better yet, a pointer to the paramater list), We will add later the types of the parameters

You need to add an element for "level" that the variable is declared. This means that X can be declared in multiple levels.

You will need a Symbol table function to remove variables introduced at level N to be removed when

you move back to level N-1

add to your YACC file an include file for the symbol table BEFORE your "include" of your abstract syntax tree

add an element to your AST node for a pointer to an SymbTab entry.

2) You should consider updating your ASTnode to include a pointer to the Symbol table. This should be set to the symbol table structure that matches the ID you are looking for. You will VERY likely need all the symbol table information. I kept my "name" variable and added a pointer to Symbol table.

Requirements

1) Modify your YACC file to include symbol table.
1a) Symbol table insert should generate a semantic error when the same name is declared at the same level
1b) On completion of a compound statement, print out all variables defined in that context, and then remove them from the symbol table
1c) Whenever an ID occurs, you need to check that the name has been defined. You need to ensure the correct form is used properly (ie as a function, scalar, or array).
1d) You need to implement rudimentary type inheritance. You should not be doing expressions to VOID, for example
1e) Symbol entries need to maintain level. Level =0 for global variables and function names. level=1 for function parameter names and the primary compound statement of the function. level is incremented by one each time a new compound statement is encountered. level is decremented by one each time the compound statement is exited

```
int x;
int z;
int f( int x, int z[])
  begin
       int y;
       begin
         int y;
       end
    end
int t;
```

would result in something like:

```
int x;  -- level 0 offset 0
int z;  -- level 0 offset 1
int f  --- level 0 offset NA   (we will use a register for return values instead of memory)
  ( int x,  level 1, offset 0
    int z[]   level 1,  offset 1
  )
  begin
        int y  -- level 1 offset 2
        begin
             int y -- level 2 offset 3
        end
  end
int t;   level 0 offset 2 -- we are at global variables now
```

2) Submission

a) Your entire project with all supporing files as a ZIP

b) The output of the symbol table for EACH time a compound statement ends AND the symbol table at the end of the parse.

This **INPUT** should generate something close to this **OUTPUT**

This is your **TEST INPUT**, please provide your output in submission

Requirements:

1) All code must be documented , with focus on explaining the actions you add to make symbol table and type checking work

2) Submit your Lex, Yexx, AST, and symbol_table code. You need to also include your AST.h, symbol_table.h

3) Your YACC code should be an enhancement from your prior submission. Major changes (which include changes in the non-terminals and terminals in the YACC code) need to be indicated and APPROVED. I expect your YACC code from the prior lab to be your starting point.