Computer Science and Engineering
IIIT Kalyani, West Bengal

Compilers Design Laboratory (CS 511)
(Autumn: 2019 - 2020)

3rd Year CSE: 5th Semester

Assignment - 4 *Marks: 10*
Assignment Out: *22nd August, 2019* Report on or before: 5th *September, 2019*

In this assignment you need to augment the language of *assignment-3*, and write its interpreter with the following features.

1. An *identifier (id)*, a single letter of English alphabet `[A-Za-qs-z]`. Note that we have excluded '`r`' as it is a *keyword* to read a data from the `stdin`.
2. Two new *operators*, '`=`' (assignment) and '`,`' (comma).

The new definition of a *fully parenthesized expression* is as follows:

1. Every *non-negative integer* (32-bit) is an expression.
2. An *identifier (id)* is an expression whose value is the value assigned to it.
3. `r` is an expression. Its value is the integer read from the `stdin`.
4. If e is an expression and id is an identifier, then $(id = e)$ is an expression. Its value is the value of expression e. The value of the *identifier (id)* is also same, and can be used afterwards.
5. If e_1 and e_2 are expressions, then so are $(e_1 + e_2)$ and $(e_1 * e_2)$ with their usual meaning. $(e_1,\ e_2)$ also is an expression whose value is the value of e_2. In all three cases *inorder* evaluation is performed.
6. Nothing else is an expression.

You need to enhance your C program (the scanner, parser and the interpreter) to incorporate these features. You have to use a *symbol table* to store the values of different identifiers. In the modified version, the input-output looks as follows:

```
$ a.out
a
Value of 'a' not defined
$ a.out
((a = (2 + r)), (b = (5 * a)))
:7
Value: 45
$ a.out
((b = (5 * a)), (a = (2 + r)))
Value of 'a' not defined
```

You are **not allowed** to use any available software or library for scanner, parser, symbol table or interpreter.

1. In the **scanner** there is a new token corresponding to an *identifier (id)*. The value of the token (`val`) may be the ASCII code of the letter (identifier).

   ```
   #include <stdio.h>
   #define END 256
   #define NUM 257
   #define ID  258

   typedef struct { int tokenClass; int val; } token_t;

   extern token_t token;
   extern void getNextToken(void);
   ```

2. In the symbol table you may use the following structure (you are free to choose some other structure as well).

   ```
   #ifndef SYMTAB_H
   #define SYMTAB_H
   #define SIZE 60

   typedef struct {
           char def; // 1: defined, 0: undefined
           int val;  // value assigned to the identifier
   } symRec;

   extern symRec symTab[SIZE];
   void initSymTab();           // every location is undefined
   void updateSymTab(int index, int val); // updates the indexed loc.
   int  getVal(int index, int *vP);     // returns error (1) or OK (0)
                                // *vP is the value of indexed loc.
   #endif
   ```

   It is an array of structures (`symRec`) of size 60. The 0th entry is for the *id* ‘A’ (65), 25th entry is for ‘Z’ (90), 32nd for ‘a’, and the 57th entry is for ‘z’ etc. The `def` field is zero (0) when the corresponding identifier is undefined. It is one (1) when it is defined. The value of the identifier is stored in the `val` field.

   The function `void initSymTab();` initializes the table by making each entry (identifier) undefined.

   The function `void updateSymTab(int index, int val);` updates the the entry corresponding to the given `index` with the `val`. It updates the value and sets the defined flag.

   The function `int getVal(int index, int *vP);` returns zero (0) if the `index`ed location is not defined. Returns one (1) if it is defined. The value of the identifier is available in `*vP`.

3. You need to modify both the parser and the backend interpreter.

4. The modified `Makefile` looks like the following one,

```
objfiles = main.o parser.o lex.o backend.o symTab.o

a.out: $(objfiles)
cc $(objfiles)

main.o: main.c
cc -c -Wall main.c

parser.o: parser.c
cc -c -Wall parser.c
lex.o: lex.c
cc -Wall -c lex.c

backend.o: backend.c
cc -Wall -c backend.c

symtab.o: symtab.c
cc -Wall -c symTab.c
clean :
   rm a.out $(objfiles)
```

item Prepare a `.tar` file with all the files you have with the following command:

```
$ tar cvf <rollNo>.4.tar lex.c lex.h parser.c parser.h main.c
                         backend.c backend.h symTab.h symTab.c Makefile
```

Send it to us on or before the due date.