µGo: A Simple Go Programming Language

Programming Assignment I

Lexical Definition

Due Date: 23:59, April 26, 2018

Your assignment is to write a scanner for the ***µGo*** language in **lex**. This document gives the lexical definition of the language, while the syntactic definition and code generation will follow in subsequent assignments.

Your programming assignments are based around this division and later assignments will use the parts of the system you have built in the earlier assignments. That is, in the first assignment you will implement the scanner using **lex**, in the second assignment you will implement the syntactic definition in **yacc**, and in the last assignment you will generate assembly code for the Java Virtual Machine by augmenting your **yacc** parser.

This definition is subject to modification as the semester progresses. You should take care in implementation that the programs you write are well-structured and easily changed.

1. *µGo* Language Features

We highlight the features of ***µGo*** by comparing it with C language.

- ***µGo*** statements do not end with semicolons "**;**".
- Conditional expression(s) in ***if*** and ***for*** loops does not enclosed by parentheses.
- ***µGo*** does not define ***while*** in its langauge.

```
/* Example code. */
var a int = 3
var b int

if a < 3 {
        b := 2
}

for i := 0; i < 10; i++ {
        sum += i
}

println(a) // Hello world
```

2. Lexical Definitions

Tokens are divided into two classes:

- Tokens that will be passed to the parser
- Tokens that will be discarded by the scanner (e.g. recognized but not passed to the parser)

2.1 Tokens that will be passed to the parser

The following tokens will be recognized by the scanner and will be eventually passed to the parser:

Delimiters

Each of these delimiters should be passed back to the parser as a token.

parentheses	( ) { }
quotation	" "
newline	\n

Arithmetic, Relational, and Logical Operators

Each of these operators should be passed back to the parser as a token.

Arithmetic	+ - * / % ++ --
Relational	< > <= >= == !=
Assignment	= += -= *= /= %=
Logical	&& || !

Keywords

The following keywords are reversed words of μGo :

var int float32 print println if else for

Identifiers

An identifier is a string of letters and digits beginning with a letter. (Case of letters is relevant, i.e., ident, Ident, and IDENT are not the same identifier. Note that Keywords are not identifiers.)

Integer Constants

A sequence of one or more digits.

String Constants

A string constant is a sequence of zero or more ASCII characters appearing between double-quote (") delimiters. A double-quote appearing with a string must be written after a ". (i.e. "abc" 、"Hello world").

2.2 Tokens that will be discarded

The following tokens will be recognized by the scanner, but should be discarded rather than passing back to the parser.

Whitespace

A sequence of blanks (spaces), tabs.

Comments (Advanced features)

Comments can be denoted in several ways:

- C-*style* is text surrounded by "***/**" and "****/***" delimiters, which may span more than one line;
- C++-*style* comments are a text following a "***//***" delimiter running up to the end of the line.

Whichever comment style is encountered first remains in effect until the appropriate comment close is encountered. For example,

// this is a comment // line */ /* with /* delimiters */ before the end

and

/* this is a comment // line with some and C delimiters */

are both valid comments.

Other characters

The undefined characters or strings should be discarded by your scanner during parsing.

3. Symbol Tables

You must implement symbol tables to store all identifiers. Symbols tables should be designed for efficient insertion and retrieval operations, and hence they are usually organized as hash tables. In order to create and manage the tables, at least the following functions should be provided.

create_symbol(): Creates a symbol table. Declare a data structure and assign the memory space to store variable.

insert_symbol (): Inserts s into a new entry of the symbol table.

lookup_symbol (): Returns index of the entry for string s, or nil if s is not found.

dump_symbol (): Dumps all entries of the symbol table.

Symbol Table example		
Index	**ID**	**Type**
0	Cat	int
1	Dog	float32

4. What Should Your Scanner Do?

Your scanner are expected to offer the **basic features**.

To get **bonus points**, your scanner should be able to provide the **advanced features**.

Basic Features (100pt)

- Print the recognized token on a separate line and discard whitespace and undefined character sets. (50pt)
  (The token could be delimiter, operator, keyword, identifier, or constant.)
- Count the lines of code in the given program. (10pt)
- Implement the symbol table functions.
  - **create_symbol()** (10pt)
  - **insert_symbol()** (10pt)
  - **lookup_symbol()** (10pt)
  - **dump_symbol()** (10pt)

(**create_symbol** and **insert_symbol** functions need to print out message for checking, and at the end of the file scan, your scanner should list all contents in the symbol table.)

The example input code and the corresponding output that we expect your scanner to generate are shown in the next page.

Advanced Features (30pt)

- Discard C and C++ type comment. (10pt)
  (You need to print out the contents of the comments and ignore them during parsing.)
- Count the comment lines. (10pt)

```
/* Declaration */          ← comment line +1
var a int = 3
var b int
println(a) // Hello world  ← comment line +1
```

- Syntax error check. (10pt)
  - Undeclared variables
  - Redefined variables

If you decide to challenge the advanced features in Programming Assignment I, please wirte the **README** to explain what advanced function(s) you implement, and upload the **README** along with your homework files.

Example input code and the expected output from your scanner:

Input:

```
var a int = 3
var b int
println(a)
// Hello world
for (a < b){
  a++
}
```

Output:

```
a         int TYPE VAR
Create a symbol table
Insert a symbol: a
=         Assign
3         Number
b         int TYPE VAR
Insert a symbol: b
println           PRINTLN function
(         LB
a         ID
)         RB
// Hello world   C++ Comment
for       FOR function
(         LB
a         ID
<         Less than
b         ID
)         RB
{         LCB
a         ID
++        Increment
}         RCB

Parse over, the line number is 7.

comment: 1 lines

The symbol table dump:
1         a       int
2         b       int
```

5. Lex Template

You can download this template file on the Moodle.

```
/* Definition section */
%{
        #include <stdio.h>
        #include <stdlib.h>

        /* Symbol table function */
        void create_symbol();
        void insert_symbol();
        int lookup_symbol();
        void dump_symbol();
%}
/* Define regular expression label */
letter [a-zA-Z]
digit [0-9]
id {letter}+({letter}|{digit})*
number {digit}+

/* Rules section */
%%
"+"     { printf("%s \t Add \n", yytext); }
"-"             { printf("%s \t Sub \n", yytext); }
{id}    { printf("%s \t ID \n", yytext); }
\n                      {;}
%%

/*      C Code section */
int yywrap(void)
{
    return 1;
}

void create_symbol() {;}
void insert_symbol() {;}
int lookup_symbol() {;}
void dump_symbol() {;}

int main(int argc,char *argv[])
{
        yyin = fopen(argv[1],"r");
        yylex();
        dump_symbol();
        return 0;
}
```

6. Submission

- Upload your homework to Moodle before the deadline.
- Package your files with either the **.zip** or **.rar** file. Other file ***formats*** are not acceptable.
- The ***organization*** of your homework folder should be as follows.

```
Compiler_StudentID_HW1.zip
└ Compiler_StudentID_HW1/
    └ input/
      Makefile
      compiler_hw1.l
      README ( if required )
```

!!! If you do not comply the above *rules*, it will lead to 10% discount of your score. !!!