μGo: A Simple Go Programming Language

Compiler 2020 Programming Assignment I
Lexical Definition
Due Date: April 23, 2020 at 23:59

Your assignment is to write a scanner for the ***μGo*** language with **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 codes you write are well-structured and able to be revised easily.

1. μGo Language Features

We highlight the features of μGo by comparing it with C language. **It is very important to note that μGo is not Go.**

- μGo is a **static type** and **strong type** language.
- μGo statements do not end with semicolons `;`.
- Conditional expression(s) in `if` and `for` does not enclosed by parentheses.
- μGo does not define `while` in its language.

```
/* Example code. */
var a int32 = 3
var b int32 = 1

if a < 3 {
    b = 2
}

var sum int32 = 0
var i int32
for i = 0; i <= 10; i++ {
    sum += i
}
println(a)   // 3
println(b)   // 1
println(sum) // 55
```

2. Lexical Definitions

Tokens are divided into two classes:

- tokens that will be passed to the parser, and
- 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.

2.1.1 Delimiters

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

Delimiters	Symbols
Parentheses	`(` `)` `{` `}` `[` `]`
Semicolon	`;`
Comma	`,`
Quotation	`"` `"`
Newline	`\n`

2.1.2 Arithmetic, Relational, and Logical Operators

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

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

2.1.3 Keywords

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

The following keywords are reserved words of μC:

Types	keywords
Data type	`int32` `float32` `bool` `string`
Conditional	`if` `else` `for`
Variable declaration	`var`
Build-in functions	`print` `println`

2.1.4 Identifiers

An identifier is a string of letters (`a`~`z`, `A`~`Z`, `_`) and digits (`0`~`9`) and it begins with a letter or underscore. Identifiers are case-sensative; for example, `ident`, `Ident`, and `IDENT` are not the same identifier. Note that keywords are not identifiers.

2.1.5 Integer Literals and Floating-Point Literals

Integer literals: a sequence of one or more digits, such as `1`, `23`, and `666`.

Floating-point literals: numbers that contain floating decimal points, such as `0.2` and `3.141`.

2.1.6 String Literals

A string literal 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 `"`, e.g., `"abc"` and `"Hello world"`.

2.2 Tokens that will be discarded

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

2.2.1 Whitespace

A sequence of blanks (spaces), tabs, and newlines.

2.2.2 Comments

Comments can be added in several ways:

- C-style is texts 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.

2.2.3 Other characters

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

3. What should Your Scanner Do?

3.1 Assignment Requirements

- We have prepared 11 μGo programs, which are used to test the functionalities of your scanner.
- Each test program is 10pt and the total score is 110pt. You will get 110pt if your scanner successfully generates the answers for all eleven programs.
- The output messages generated by your scanner must use the given names of token classes listed below.

Symbol	Token		Symbol	Token		Symbol	Token
`+`	ADD		`&&`	LAND		`print`	PRINT
`-`	SUB		`||`	LOR		`println`	PRINTLN
`*`	MUL		`!`	NOT		`if`	IF
`/`	QUO		`(`	LPAREN		`else`	ELSE
`%`	REM		`)`	RPAREN		`for`	FOR
`++`	INC		`[`	LBRACK		`int32`	INT
`--`	DEC		`]`	RBRACK		`float32`	FLOAT
`>`	GTR		`{`	LBRACE		`string`	STRING
`<`	LSS		`}`	RBRACE		`bool`	BOOL
`>=`	GEQ		`;`	SEMICOLON		`true`	TRUE
`<=`	LEQ		`,`	COMMA		`false`	FALSE
`==`	EQL		`"`	QUOTA		`var`	VAR
`!=`	NEQ		`\n`	NEWLINE			
`=`	ASSIGN						
`+=`	ADD_ASSIGN		Int Number	INT_LIT			
`-=`	SUB_ASSIGN		Float Number	FLOAT_LIT			
`*=`	MUL_ASSIGN		String Literal	STRING_LIT			
`/=`	QUO_ASSIGN		Identifier	IDENT			
`%=`	REM_ASSIGN		Comment	COMMENT			

3.2 Example of Your Scanner Output

The example input code and the corresponding output that we expect your scanner to generate are as follows.

- Input:

```
var a int32 = 3
var b int32
println(a) /* print a */
b += 10 // Hello world
for a < b {
    a++
}
```

- Output:

```
var          VAR
a            IDENT
int32        INT
```

```
=            ASSIGN
3            INT_LIT
             NEWLINE
var          VAR
b            IDENT
int32        INT
             NEWLINE
println      PRINTLN
(            LPAREN
a            IDENT
)            RPAREN
/* print a */            C Comment
             NEWLINE
b            IDENT
+=           ADD_ASSIGN
10           INT_LIT
// Hello world   C++ Comment
             NEWLINE
for          FOR
a            IDENT
<            LSS
b            IDENT
{            LBRACE
             NEWLINE
a            IDENT
++           INC
             NEWLINE
}            RBRACE

Finish scanning,
total line: 7
comment line: 2
```

4. Environmental Setup

- For Linux
  - Ubuntu 18.04 LTS
  - Install dependencies: `$ sudo apt install flex bison`
- For Windows
  - You may like to install VirtualBox to emulate the Linux environment.

Our grading system uses the **Ubuntu** environment. We will revise your uploaded code to adapt to our environment. In order to facilitate the automated code revision process, we need your help to arrange your code in the following format as specified in 5. Submission.

5. Submission

Upload your homework to Moodle before the deadline.

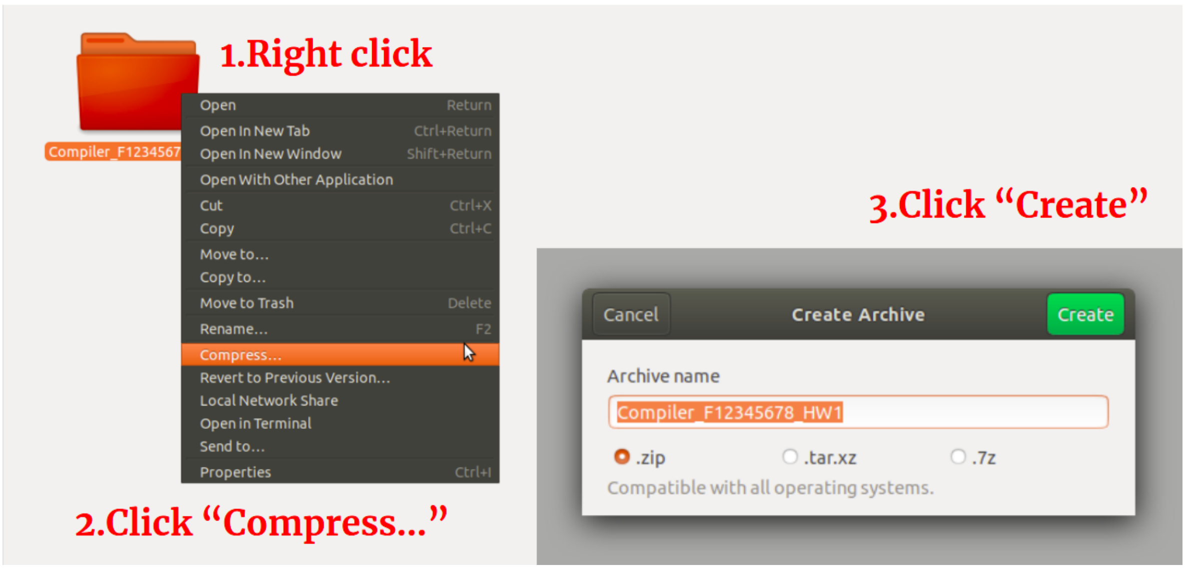
Deadline: April 23, 2020 at 23:59

- Compress your files with `zip` or `rar`. **Other file formats are not acceptable.**
- Your uploaded assignment must be organized illustrated below. Otherwise, our auto-grading tool will ignore it.

- If our tool fails to recognize your homework files because you do not comply the rules, you will have to live demo your homework with our TAs (after you realize your graded score is unacceptable low) and there will be a 10% discount of your score for this assignment.
- As for the filename of your compressed file, you should replace `StudentID` with your student id number. That is, your compressed file should have the name like: `Compiler_F12345678_HW1.zip`.

```
Compiler_StudentID_HW1.zip/
└── Compiler_StudentID_HW1/
    ├── compiler_hw1.l
    └── Makefile
```

Three steps to create zip file for the submission.



Archive name: Compiler_F12345678_HW1

[✓ .zip] .tar.xz .7z

Compatible with all operating systems.

6. References

- Source file src/go/token/token.go: https://golang.org/src/go/token/token.go