μC: A Simple C Programming Language

Programming Assignment I

Lexical Definition

Due Date: 23:59, April 18, 2019

Your assignment is to write a scanner for the ***μC*** 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 on 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.

The lexical definition below is subject to modification later. You should read it carefully while implementing your assignments, and please make sure that the programs you write are well-structured.

1. 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 (i.e., recognized but not passed to the parser).

1.1 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.

Delimiters	Operators
Parentheses	**( ) { } [ ]**
Semicolon	**;**
Comma	**,**
Quotation	**“ ”**
Newline	**\n**

Arithmetic, Relational, and Logical Operators

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

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

Keywords

The following keywords are reserved words of μC:

int float bool string void print if else for while true false return continue break

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

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 and Floating-Point Constants

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

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

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").

1.2 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, and newlines.

Comments

Comments can be denoted 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.

Other characters

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

2. What should Your Scanner Do?

Assignment Requirements

- Print the recognized token on a separate line and discard whitespace and undefined character sets, **as shown as Section 1.1**. (60pt)
- C and C++ type comment, **as shown as Section 1.2.** (20pt)
- Count code and comment line. (20pt)

Your answer must follow the given in **Token format** below

Token Format

Symbol	Token	Symbol	Token
+	ADD	[	LSB
-	SUB	]	RSB
*	MUL	,	COMMA
/	DIV	;	SEMICOLON
%	MOD	“	QUOTA
++	INC	print	PRINT
--	DEC	if	IF
>	MT	else	ELSE
<	LT	for	FOR
>=	MTE	while	WHILE
<=	LTE	string (keywords)	STRING
==	EQ	int	INT
!=	NE	float	FLOAT

=	**ASGN**	**void**	**VOID**
+=	**ADDASGN**	**bool**	**BOOL**
-=	**SUBASGN**	**true**	**TRUE**
***=**	**MULASGN**	**false**	**FALSE**
/=	**DIVASGN**	**return**	**RET**
%=	**MODASGN**	**continue**	**CONT**
&&	**AND**	**break**	**BREAK**
||	**OR**	**Int Number**	**I_CONST**
!	**NOT**	**Float Number**	**F_CONST**
(	**LB**	**String Constants**	**STR_CONST**
)	**RB**	**Identifier**	**ID**
{	**LCB**	**Comment**	**COMMENT**
}	**RCB**		

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

Example input code and the expected output from your scanner:

Input:

```
1 /*
2  * 2019 Spring Compiler Course Assignment 1
3  */
4
5 int main() {
6     // Declaration
7     int x;
8     int a = 5;
9     string y = "Hello World";
10
11    print(y);
12
13    // if condition
14    if (a > 10) {
15    |   x += a;
16    |   print(x);
17    }
18    return 0;
19 }
```

Output:

```
/*
 * 2019 Spring Compiler Course Assignment 1
 */       C Comment
int       INT
main      ID
(         LB
)         RB
{         LCB
// Declaration    C++ Comment
int       INT
x         ID
;         SEMICOLON
int       INT
a         ID
=         ASGN
5         I_CONST
;         SEMICOLON
string    STRING
y         ID
=         ASGN
"         QUOTA
Hello World       STR_CONST
"         QUOTA
;         SEMICOLON
print     PRINT
(         LB
y         ID
)         RB
;         SEMICOLON
// if condition          C++ Comment
if        IF
(         LB
a         ID
>         MT
10        I_CONST
)         RB
{         LCB
x         ID
+=        ADDASGN
a         ID
;         SEMICOLON
print     PRINT
(         LB
x         ID
)         RB
;         SEMICOLON
}         RCB
return    RET
0         I_CONST
;         SEMICOLON
}         RCB

Parse over, the line number is 19.

comment: 5 lines
```

3. lex Template

You can download this template file from Moodle.

```
%{
#include <stdio.h>
#include <stdlib.h>
#include <ctype.h>
%}

letter [a-zA-Z]
digit [0-9]
id {letter}+({letter}|{digit})*
number                {digit}+

%%

";"                             {printf("%s \t SEM \n",yytext); }
{id}                            {printf("%s \t ID \n",yytext); }
"-"                             {printf("%s \t SUB \n",yytext);}
"+"                             {printf("%s \t ADD \n",yytext);}
[\n]                            {;}

%%

main(int argc,char *argv[]){

        yyin = fopen(argv[1],"r");
        yylex();
}

yywrap(void) {
    return 1;
}
```

4. Environmental Setup

- For Linux
  - Ubuntu 16.04 LTS
  - $ sudo apt-get install flex bison
- For Windows
  - You may like to install VirtualBox to emulate the Linux environment.
  - The installation guide has been upload to Moodle.

Our grading system uses the Ubuntu environment. We will revise your uploaded code to adopt 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: 23:59, April 18, 2019 (THU)
- Compress your files with either the .zip or .rar file. **Other file formats are not acceptable**.
  - Your uploaded assignment must be organized as follow, **otherwise we will ignore it. And you will have to live demo for your homework to get the scores.**

```
Compiler_StudentID_HW1.zip
  ↳ Compiler_StudentID_HW1/
    ↳ Makefile
    ↳ compiler_hw1.l
    ↳ input/
    ↳ output/
```