

Department of Computer Science and Engineering
Indian Institute of Technology Kharagpur

Compilers Laboratory: CS39003

3rd Year CSE: 5th Semester

Assignment - 3: Lexer for tinyC

Assign Date: August 12, 2024

Marks: 100

Submit Date: 23:55, August 26, 2024

1 Preamble – tinyC

This assignment follows the lexical specification of C language from the International Standard **ISO/IEC 9899:1999 (E)**. To keep the assignment within our required scope, we have chosen a subset of the specification as given below. We shall refer to this language as **tinyC** and subsequently (in a later assignment) specify its grammar from the Phase Structure Grammar given in the C Standard.

The lexical specification quoted here is written using a precise yet compact notation typically used for writing language specifications. We first outline the notation and then present the Lexical Grammar that we shall work with.

2 Notation

In the syntax notation used here, syntactic categories (non-terminals) are indicated by *italic type*, and literal words and character set members (terminals) by **bold type**. A colon (:) following a non-terminal introduces its definition. Alternative definitions are listed on separate lines, except when prefaced by the words "one of". An optional symbol is indicated by the subscript "opt", so that the following indicates an optional expression enclosed in braces.

$\{ \textit{expression}_{opt} \}$

3 Lexical Grammar of tinyC

1. Lexical Elements

token:

keyword
identifier
constant
string-literal
punctuator

2. Keywords

keyword: one of

auto	enum	restrict	unsigned
break	extern	return	void
case	float	short	volatile
char	for	signed	while
const	goto	sizeof	_Bool
continue	if	static	_Complex
default	inline	struct	_Imaginary
do	int	switch	
double	long	typedef	
else	register	union	

3. Identifiers

identifier:

identifier-nondigit
identifier identifier-nondigit
identifier digit

identifier-nondigit: one of

-	a	b	c	d	e	f	g	h	i	j	k	l	m
	n	o	p	q	r	s	t	u	v	w	x	y	z
	A	B	C	D	E	F	G	H	I	J	K	L	M
	N	O	P	Q	R	S	T	U	V	W	X	Y	Z

digit: one of

0 1 2 3 4 5 6 7 8 9

4. Constants

constant:

integer-constant

floating-constant

character-constant

integer-constant:

integer-constant digit

floating-constant:

fractional-constant *exponent-part*_{opt}

digit-sequence *exponent-part*

fractional-constant:

*digit-sequence*_{opt} . *digit-sequence*

digit-sequence .

exponent-part:

e *sign*_{opt} *digit-sequence*

E *sign*_{opt} *digit-sequence*

sign: one of

+ -

digit-sequence:

digit

digit-sequence digit

' *c-char-sequence* '

c-char-sequence:

c-char

c-char-sequence c-char

c-char:

any member of the source character set except
the single-quote ' , backslash \ , or new-line character

escape-sequence

escape-sequence: one of

\' \\" \? \\
\a \b \f \n \r \t \v

5. String literals

string-literal:

" *s-char-sequence*_{opt} "

s-char-sequence:

s-char

s-char-sequence s-char

s-char:

any member of the source character set except
the double-quote " , backslash \ , or new-line character

escape-sequence

6. Punctuators

punctuator: one of

```
[ ] ( ) { } . ->
++ -- & * + - ~ !
/ % << >> < > <= >= == != ^ | && ||
? : ; ...
= *= /= %= += -= <<= >>= &= ^= |=
, #
```

7. Comments

(a) *Multi-line Comment*

Except within a character constant, a string literal, or a comment, the characters `/*` introduce a comment. The contents of such a comment are examined only to identify multibyte characters and to find the characters `*/` that terminate it. Thus, `/* ... */` comments do not nest.

(b) *Single-line Comment*

Except within a character constant, a string literal, or a comment, the characters `//` introduce a comment that includes all multibyte characters up to, but not including, the next new-line character. The contents of such a comment are examined only to identify multibyte characters and to find the terminating new-line character.

You should maintain a symbol table keeping records of the token names (say, *keyword*, *identifier* etc), and the respective lexemes.

4 The Assignment

1. Write a flex specification for the language of `tinyC` using the above lexical grammar. Name of your file should be `ass3_roll1_roll2.1`. *The `ass3_roll1_roll2.1` should not contain the function `main()`.*
2. Write your `main()` (in a separate file `ass3_roll1_roll2.c`) to test your lexer.
3. Prepare a Makefile to compile the specifications and generate the lexer.
4. Prepare a test input file `ass3_roll1_roll2.test.c` that will test all the lexical rules that you have coded.
5. The execution of your lexical analyzer should print (a) the stream of tokens in the form of $\langle \text{tokenname}, \text{lexeme} \rangle$ (such as $\langle \text{keyword}, \text{int} \rangle$, $\langle \text{identifier}, x \rangle$ etc), and (b) the symbol table.
6. Prepare a tar-archive with the name `ass3_roll1_roll2.tar` containing all the above files and upload to Moodle.

5 Credits

1. Flex Specifications: **60**
2. Main function and Makefile: **20** [15+5]
3. Test file: **20**