Exercise 1

Compilation 0368:3133

Due 30/10/2017

1 Introduction

During the semester we will implement a compiler to an invented object oriented language called RioMare. The entire specification of the RioMare programming language appears inside the relevant folder of the course website. In order to make this document self contained, all the information needed to complete the first exercise is brought here again.

2 Programming Assignment

The first exercise implements a lexical scanner based on the open source tool JFlex. The input for the scanner is a single text file containing a RioMare program, and the output is a (single) text file containing a tokenized representation of the input. The course repository contains a simple skeleton program, and you are encouraged to work your way up from there.

3 Lexical Considerations

Identifiers may contain letters and digits, and must start with a letter.

The keywords in Table 1 can not be used as identifiers.

int	string	void	while
class	nil	new	if
PrintInt	extends	return	end
PrintString			

Table 1: Reserved keywords of RioMare.

White spaces consist of spaces, tabs and newlines characters. They may appear between any tokens. Keywords and identifiers must be separated by a white space, or a token that is neither a keyword nor an identifier.

Comments in RioMare are similar to those used in the C programming language: a comment beginning with the characters // indicates that the remainder of the line is a comment. In addition, a comment can be a sequence of characters that begins with /*, followed by any characters, including newlines, up to the first occurrence of the end sequence */. Unclosed comments are lexical errors.

Integer literals may start with an optional negation sign $-$, followed by a sequence of digits. Non-zero numbers should *not* have leading zeroes. Though integers are stored as 32 bits in memory, they are artificially limited in RioMare to have 16-bits signed values between -2^{15} and $2^{15} - 1$. Integers out of this range are lexical errors.

Precedence	Operator	Description	Associativity
1	:=	assign	right
2	=	equals	left
3	$<, \leq, >, \geq$		left
4	$+, -$		left
5	$*, /$		left
6	[	array indexing	
7	(	function call	
8	`->`	field access	left

Table 2: Binary operators of RioMare along with their associativity and precedence. 1 stands for the lowest precedence, and 9 for the highest.

To create a graph visualization of the AST, please install graphviz and run

```
$ dot -Tjpeg -o ./AST_Graph.jpeg ./AST_Graph.txt
```

from `EX5/LINUX_GCC_MAKE`

4 Input

The input for this exercise is a single text file, the input RioMare program.

5 Output

The output is a single text file that contains a tokenized representation of the input program. Each token should appear in a separate line, together with the line number it appeared on, and the character position inside that line. The list of token names appears in Table 777, and will only be used in this first exercise. Later phases of the compiler will make no use of these token names.

Three types of tokens are associated with corresponding values: integers, identifiers and strings. The printing format for these tokens can be easily deduced from the examples in Table 4.

Token Name	Description		Token Name	Description
LPAREN	(		ASSIGN	:=
RPAREN	)		EQ	=
LBRACK	[		LT	<
RBRACK	]		GT	>
LBRACE	{		CLASS	
RBRACE	}		EXTENDS	
PLUS	+		RETURN	
MINUS	−		WHILE	
TIMES	*		IF	
DIVIDE	/			
COMMA	,		INT(*value*)	*value* is an integer
DOT	.		STRING(*value*)	*value* is a string
SEMICOLON	;		ID(*value*)	*value* is an identifier

Table 3: Token names and printing format for the first exercise. Each line in the output text file should contain a single token. Note that three types of tokens are associated with corresponding values: integers, identifiers and strings. The rest of the tokens encountered should only contain their name, the line number they appeared on, and the character position inside that line.

Printed Lines Examples	Description
INT(74)[3,8]	integer 74 is encountered in line 3, character position 8
STRING("Dan")[2,5]	string "Dan" is encountered in line 2, character position 5
ID(numPts)[1,6]	identifier numPts is encountered in line 1, character position 6

Table 4: Token names and printing format for the first exercise. Each line in the output text file should contain a single token. Note that three types of tokens are associated with corresponding values: integers, identifiers and strings. The rest of the tokens encountered should only contain their name, the line number they appeared on, and the character position inside that line.

6 Submission Guidelines

Open an account on GitHub. Then, visit the academic discount page to enable the free creation of private repositories. One team member should create a new *private* repository called COMPILATION, and then invite other team members and myself as collaborators. My username is OrenGitHub. Please put inside the uppermost folder (COMPILATION) a text file "IDS.txt" with the IDs of all team members (one ID per line). In addition, COMPILATION should contain a sub folder called EX1 where your code will reside. COMPILATION/EX1 should contain a makefile building your source to a runnable jar file called LEXER (note the lack of the .jar suffix). Feel free to use the makefile supplied in the course repository, or write a new one if you want to. Before you submit, make sure that your exercise compiles and runs on the school server: *nova.cs.tau.ac.il*. This is

the formal running environment of the course.

Execution parameters compiler receives 2 input file names:

InputRioMareProgram.txt
OutputStatus.txt