Compilation \ PA1 Documentation

***Group members**:*

- *Kalev Alpernas, user: kalevalp, ID: 304385727*
- *Ariel Stolerman, user: arielst1, ID: 039432489*
- *Vadim Stotland, user: stotland, ID: 314282682*

***Code structure**:*

IC.lex is the source for the lexical analyzer which is used to scan the input IC code. The compiler main method runs over the input file, breaks it to tokens using the Lexer class, and handling lexical exceptions using LexerError class.

Package IC:

- *Class Compiler: uses java.io and package Parser (described below). Uses:*
  - *Token: for presenting each token in the input stream.*
  - *Lexer: as the lexical analyzer (generated by IC.lex).*
  - *LexerError: standard error for lexical error exceptions.*
  - *Sym: definition class for the token IDs.*

Package Parser:

- *Token: extends java_cup.runtime.Symbol.*
- *Lexer: the lexical analyzer class generated by JFlex from IC.lex; implements java_cup.runtime.Scanner. uses all other .classes in the package.*
- *LexerError: extends exception.*
- *Sym: independent definitions class.*

IC.lex:

The lex source code from which class Lexer is generated by JFlex. Kept in package Parser but compiled separately.

***Test plan**:*

Main Concerns:

- *Correct recognition of tokens*
  - *Correct tokens*
  - *Correct order*
  - *Comment handling*
- *Error finding and handling*

1. *Token Recognition:*
   1.1. *Simple tests for all token types (including comments)*
   - *keywords (one test for each)*
   - *strings ("quotes") and escape characters in strings: printable ASCII charactes other than " and \, \n, \t, \\, \".*
   - *IDs (ID, CLASS_ID)*
   - *numbers*
   - *comments*

   1.2. *Complex tests including multiple tokens of multiple types, and comments.*

2. *<u>Errors</u>:*
   2.1. *Error finding:*
      - *unrecognized character*
      - *incorrect ASCII characters in strings*
      - *unexpectred EOF (unclosed '/*' comment)*
      - *integer value out of bound (checks that $|n| < 2^{31}$, not that $-2^{31} \leq n \leq 2^{31} - 1$ as specified in the spec)*

   2.2. *Error handling - correct printout for each type of errors*

***Regular expressions for the tokens**:*

Token	**Id**	**Reg. exp**	**Comments**	**Reg. exp**
LP (	1	“(“	Comments “// … “	“//”
RP)	2	“)”	Comments “/* … */”	“/*”
ASSIGN =	3	“=”	**Macros**	
BOOLEAN boolean	4	“boolean"		
BREAK break	5	“break”	DIGIT	[0-9]
CLASS class	6	“class”	LOWER_CASE	[a-z]
CLASS_ID <name>	7	{UPPER_CASE}({ALPHA_NUMERIC})*	UPPER_CASE	[A-Z]
COMMA ,	8	“,”	LETTER	{LOWER_CASE}|{UPPER_CASE}
CONTINUE continue	9	“continue”	ALPHA_NUMERIC	{DIGIT}|{LETTER}|[_]
DIVIDE /	10	“/”	LINE_TERMINATOR	\r|\n|\r\n
DOT .	11	“.”	WHITE_SPACE	{LINE_TERMINATOR}|[\t\f]
EQUAL ==	12	“==”	**States**	
EXTENDS extends	13	“extends”		
ELSE else	14	“else”	YYINITIAL	-
FALSE false	15	“false”	COMMENT1	State for comment of type “//” that ends at newline
GT >	16	“>”		
GTE >=	17	“>=”	COMMENT2	State for comment of type “/* … */” (including newlines in it)
ID <name>	18	{LOWER_CASE}({ALPHA_NUMERIC})*		
IF if	19	“if”		
INT int	20	“int”		
INTEGER <integer>	21	([1-9]({DIGIT})*)|([0]+)		
LAND &&	22	“&&”		
LB[	23	“[“		
LCBR {	24	“{“		
LENGTH length	25	“length”		
NEW new	26	“new”		
LNEG !	27	“!”		
LOR ||	28	“||”		
LT <	29	“<”		
LTE <=	30	“<=”		
MINUS -	31	“-“		
MOD %	32	“%”		
MULTIPLY *	33	“*”		
NEQUAL !=	34	“!=”		
NULL null	35	“null”		
PLUS +	36	“+”		
RB]	37	“]”		
RCBR }	38	“}”		
RETURN return	39	“return”		
SEMI ;	40	“;”		
STATIC static	41	“static”		
STRING string	42	“string”		
QUOTE “<string>”	43	[\"]([!#-\[\]-~]|"\\\\"|"\\\""|"\\t"|"\\n")*[\"]		
THIS this	44	“this”		
TRUE true	45	“true”		
VOID void	46	“void”		
WHILE while	47	“while”		

***Feedback**:*

- *We spent approximately 8-9 hours on all parts of the assignment.*
- *All in all the assignment was in an appropriate level of difficulty. The hardest part was understanding at first how all parts integrate.*