Name: Azaan Aziz RegNo: L1F22BSCS0972

Compiler Construction Project Documentation

Project Description:

Neo++ is a simplified beginner-friendly designed for educational purpose. It focuses on clean syntax, readability and the use of minimum Keywords. The theme of Neo++ is clear and Simple every construct every construct is easy to under-Stand. It Support programming Structures like Variables, conditions, loops and functions

Regex Table:

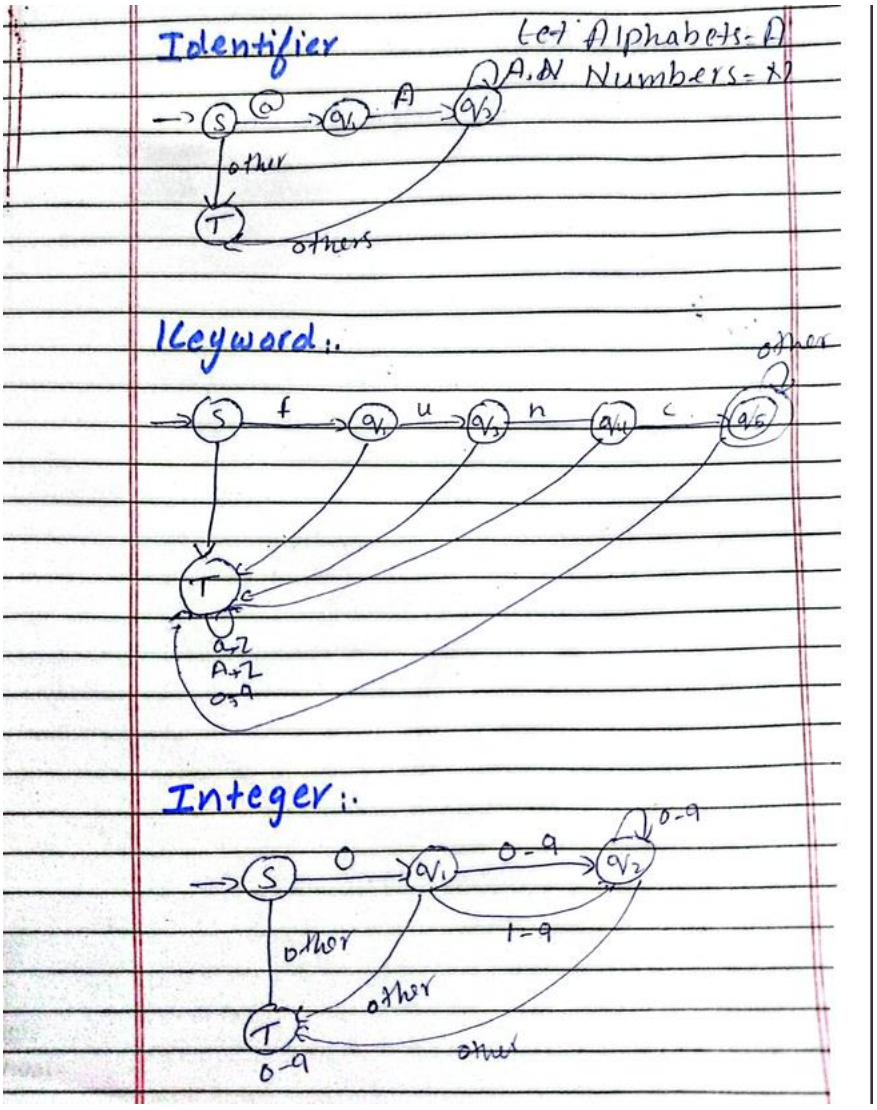
Token Type	Example	Regular expression(Regex)	Explanation
Identifier	@count, @name1, @total_sum	`@[_a-zA-Z][_a-zA-Z0-9]*`	Starts with @ followed by letters, digits, or underscores
Integer Literal	10, 256, 0	`[0-9]+`	One or more digits
Float Literal	12.34, 0.56, 99.0	`[0-9]+\.[0-9]+`	Digits followed by a dot and more digits
Character Literal	|A|, |n|, |9|	`|.|`	Start with | and end with |
String Literal	`Hello`, `Neo++`	`[^]*` `	Start with ` and ends with `
Keywords	Ifx ((n :: 5))	"func"|"num"|"decil"|"txt"|"booll"|"charac"|"arr"|"struc"	Conditional loop ,function , return statement is used
Data Types	Decil num	num, decil, txt, booll, charac, arr, struc	It defines datatype for variables
Boolean Literals	Booll flag::yes;	`` `yes | no ``	Boolean literals represent two possible logical states: `yes` → true, `no` → false
Operators	Num :: 5	`::"|"++"|"--"|""|"~"|"=\?"|"&&&"|"!"`	Used to perform operation like: Addition++, Subtraction--,

			Multiply**, Division~ Compare=?, AND operator &&& OR operator |
Punctuation	Decil n::0.5;;	`":"|";;"|"."|"(("|"))"|"{{"|"}}"`	Used for : Colon : Endline;; Separate parameter: **.**
Single-line Comment	//coment	`\/\/[^\n]*`	Starts with // and continues until newline
Multi-line Comment	/\comment/\	`"/\\"([^\\]|\\n)*"/\\"`	For multiline It start/\ and end with /\
Whitespace	Spaces,tabs	`[ \t\r]+`	Ignored in tokenization
Newline	Line break	\n	Used to increment line counter
Unknown/Invalid Token	`@#, $%, ?`	.	Matches any single unrecognized character

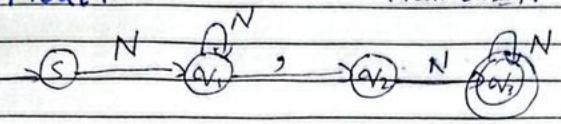
Keywords Explanation:

Keyword	Category	Purpose / Explanation
func	Function Declaration	Used to define a new function or procedure, similar to `void` or `def` in other languages.
num	Data Type	Represents an integer data type (whole numbers). Equivalent to `int`.
decil	Data Type	Represents a floating-point number (decimal type). Equivalent to `float` or `double`.
txt	Data Type	Used for string-type variables to store sequences of characters.
booll	Data Type	Boolean data type, storing logical values `yes` or `no` (true/false).
charac	Data Type	Represents a single character literal. Equivalent to `char`.
arr	Data Structure Keyword	Declares an array type to store multiple elements of the same type.
struc	Data Structure Keyword	Defines a structure or record, grouping multiple variables under one name.
ifx	Control Structure	Conditional statement, similar to `if` in C/C++, used to perform decisions.
elx	Control Structure	Paired with `ifx`, similar to `else`, for alternate execution paths.
wloop	Iteration Control	Represents a `while` loop — executes repeatedly while a condition is true.
dwloop	Iteration Control	Represents a `do-while` loop — executes at least once, then checks the condition.
floop	Iteration Control	Represents a `for` loop — used for counting or fixed-number iterations.
ret	Function Control	Used to return a value from a function, like `return` in C/C++.
show	I/O Operation	Used for displaying output to the user — similar to `print` or `cout`.
take	I/O Operation	Used for taking input from the user — similar to `scanf` or `input`.
giveback	Function Control	Optional return keyword alternative — conceptually used for returning final output from a program.

Transition Diagrams:

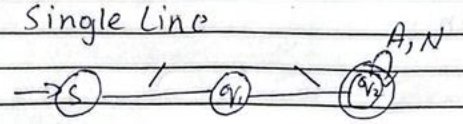


Float:. Number N



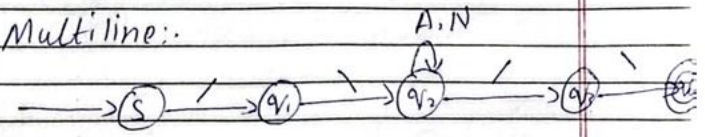
Comments:.

Single Line

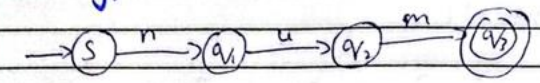


Alphabets - A
Numbers = N

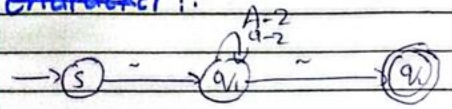
Multiline:.



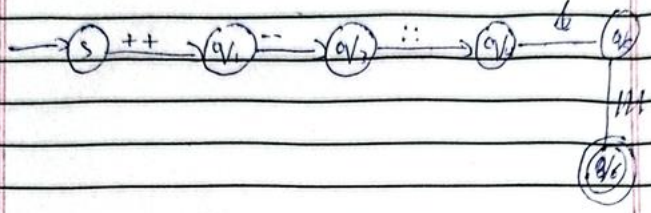
Datatype



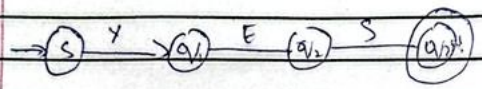
String
~~Character~~ :.



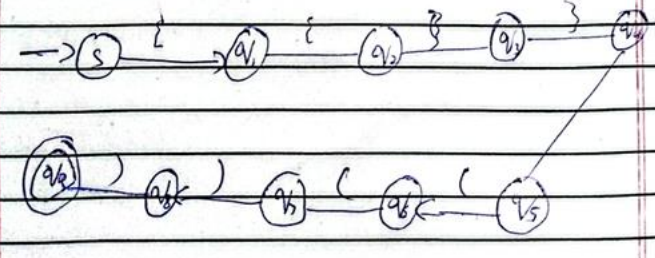
Operators :.



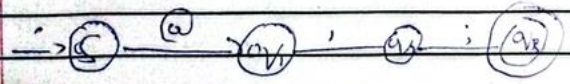
Boolean :.



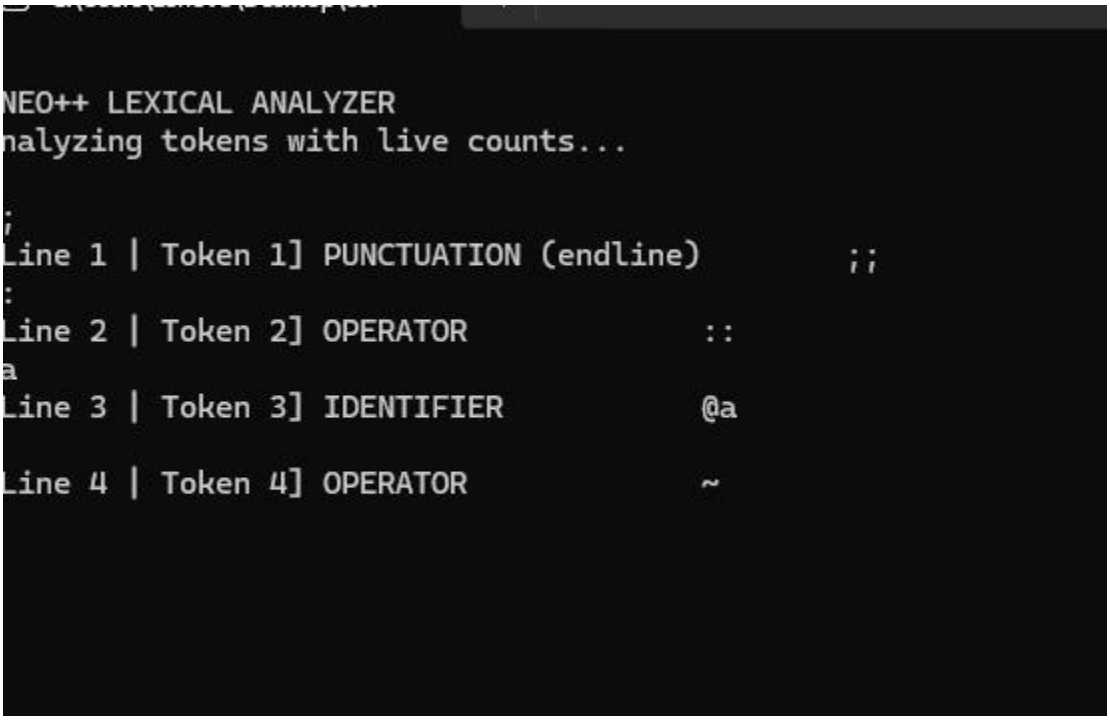
Delimeters :.



Symbols :.



Output:


NEO++ LEXICAL ANALYZER
nalyzing tokens with live counts...
;
Line 1 | Token 1] PUNCTUATION (endline) ;;
:
Line 2 | Token 2] OPERATOR ::
a
Line 3 | Token 3] IDENTIFIER @a
Line 4 | Token 4] OPERATOR ~