J0 Programming Language

Reserved Words
CLASS - preceding the class name.
CONST - these are immutable and must be initialized when declared with an integer value.
VAR - these are mutable integer variables.
IF, THEN, ELSE - control structure
READ, WRITE - IO from terminal

All the tokens should be separated by one or more empty space.

Comments are enclosed in "/*" and "*/". They may appear anywhere in a sentence and have no meaning to the translator. Single line comments with //.

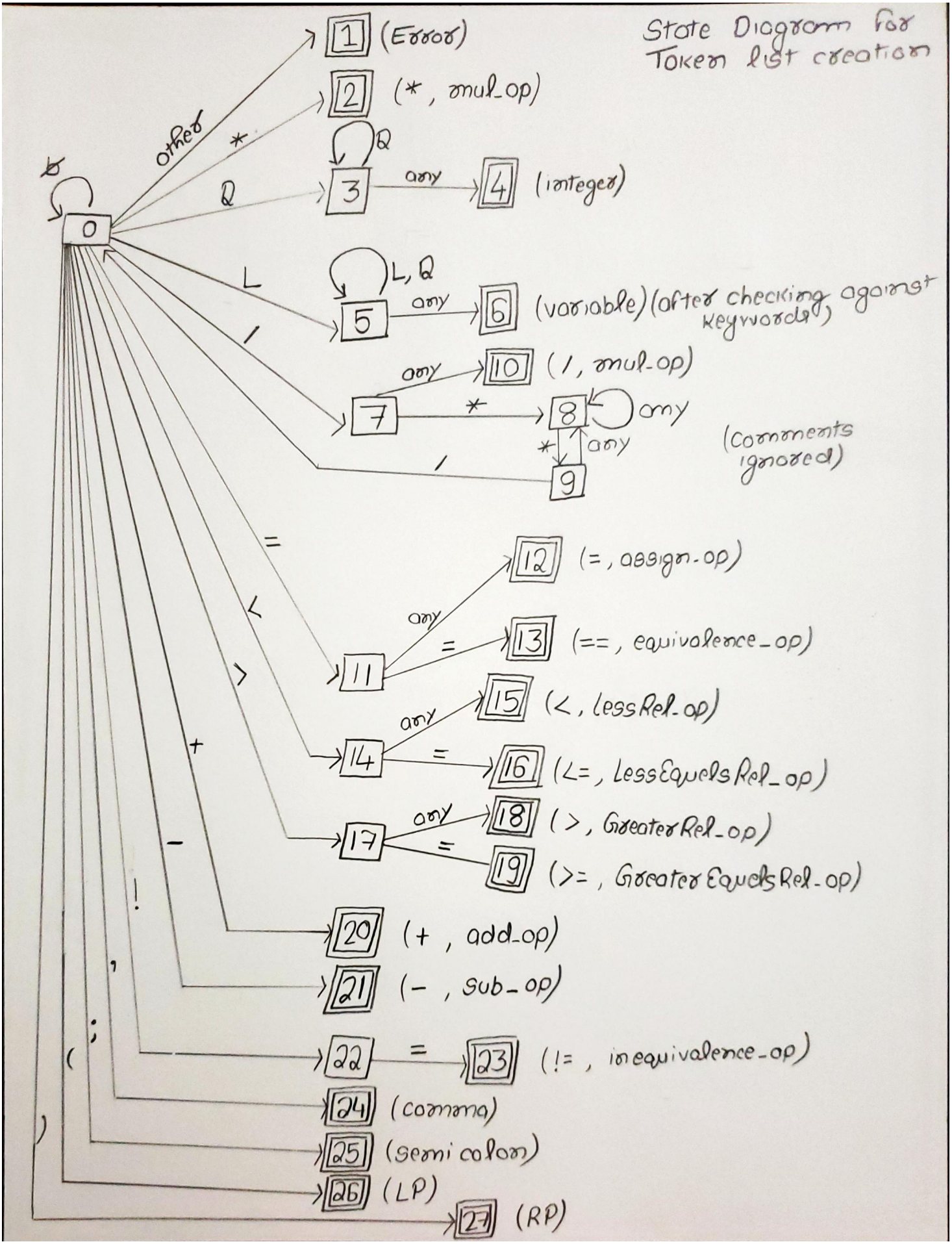
Single Character Delimiters:
= , ; + - * / ( ) < > { }

Double Character Delimiters:
== >= <= != /* */

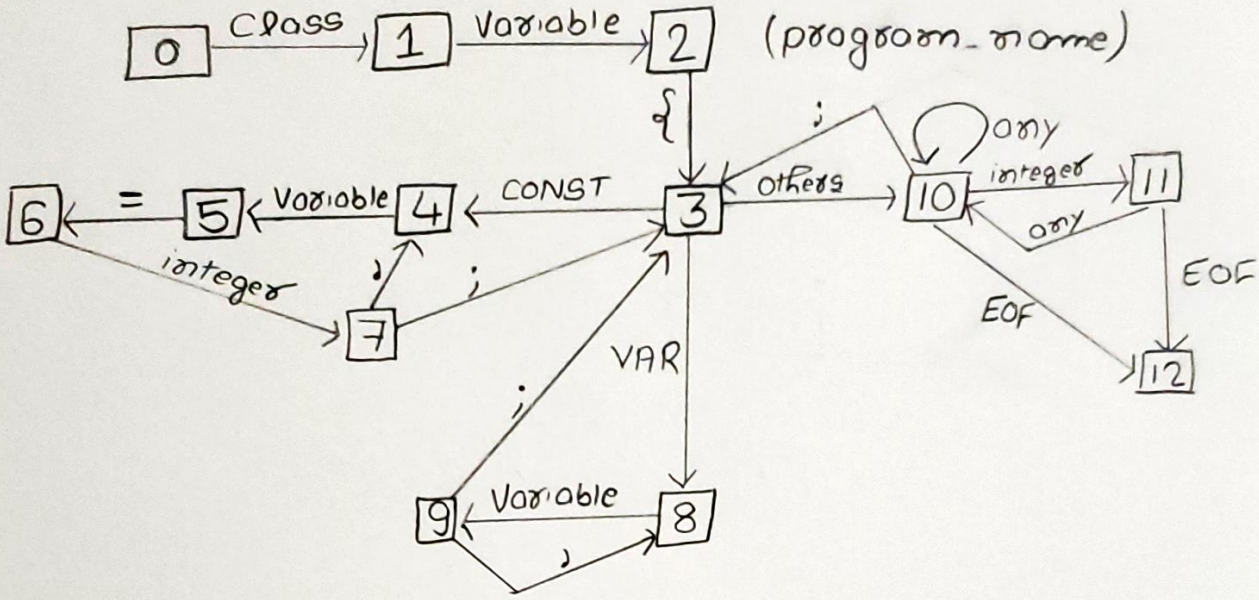
Outline of a JO programs that this compiler will accept.

```
CLASS Pgm {
CONST M = 13 , N = 56 ;
VAR X , Y ;
READ Y ;
X = M * N + 18 - Y ;
WRITE Z ;
}
```

This compiler provides only generic error messages.


State Diagram for Token list creation
0
1 (Error)
2 (*, mul_op)
3
4 (integer)
5
6 (variable) (after checking against keywords)
7
8
9
10 (/, mul_op)
(comments ignored)
11
12 (=, assign_op)
13 (==, equivalence_op)
14
15 (<, lessRel_op)
16 (<=, lessEqualsRel_op)
17
18 (>, GreaterRel_op)
19 (>=, GreaterEqualsRel_op)
20 (+, add_op)
21 (-, sub_op)
22
23 (!=, inequivalence_op)
24 (comma)
25 (semicolon)
26 (LP)
27 (RP)
other
any
Q
L
L, Q
/
*
=
<
>
+
-
!
,
;
(
)

State diagram for making Symbol Table From Token list



Precedence Table used for Push Down Automata of Java 0 parser

	T	=	+	-	(	)	*	/	IF	THEN	==	!=	>	<	>=	<=	{	}	ELSE
T		<							<										
=	>		<	<	<		<	<											>
+	>		>	>	<	>	<	<		>	>	>	>	>	>	>			>
-	>		>	>	<	>	<	<		>	>	>	>	>	>	>			>
(			<	<	<	=	<	<											>
)	>		>	>		>	>	>											>
*	>		>	>	<	>	>	>		>	>	>	>	>	>	>			>
/	>		>	>	<	>	>	>		>	>	>	>	>	>	>			>
IF			<	<	<		<	<		=	<	<	<	<	<	<			=
THEN		<							<								<		=
==			<	<	<		<	<		>									
!=			<	<	<		<	<		>									
>			<	<	<		<	<		>									
<			<	<	<		<	<		>									
>=			<	<	<		<	<		>									
<=			<	<	<		<	<		>									
{																			
}																			
ELSE		<																	

Sample

SomePgm.txt – input source code

```
CLASS Pgm {
VAR area , length , width , height ;
READ lenght ;
READ width ;
READ height ;
area = 2 * ( length * width + width * height + height * length ) ;
WRITE area ;
}
```

lexoutput.txt

Token	Classification
CLASS	$CLASS
Pgm2	variable
{	LB
VAR	$VAR
area	variable
,	comma
length	variable
,	comma
width	variable
,	comma
height	variable
;	semicolon
READ	$READ
lenght	variable
;	semicolon
READ	$READ
width	variable
;	semicolon
READ	$READ
height	variable
;	semicolon
area	variable
=	assign
2	integer
*	mop
(	LP
length	variable
*	mop
width	variable
+	addop
width	variable
*	mop
height	variable
+	addop
height	variable
*	mop
length	variable
(	LP
;	semicolon

WRITE	**$WRITE**
area	**variable**
;	**semicolon**
}	**RB**
EOF	**$EOF**

symboltable.txt

Token	Class	Value	Address	Segment
Pgm2	$ProgramName		0	CS
area	Var	?	0	DS
length	Var	?	2	DS
width	Var	?	4	DS
height	Var	?	6	DS
T1		?	8	DS
T2		?	10	DS
T3		?	12	DS
T4		?	14	DS
T5		?	16	DS
T6		?	18	DS
T7		?	20	DS
T8		?	22	DS
T9		?	24	DS
T10		?	26	DS

quads.txt

*	length	width	T1
*	width	height	T2
+	T1	T2	T3
*	height	length	T4
+	T3	T4	T5
*	2	T5	T6
=	area	T6	!

Push Down Automata Stack

Terminator area
Terminator area =
Terminator area = 2
Terminator area = 2 *
Terminator area = 2 * (
Terminator area = 2 * (length
Terminator area = 2 * (length *
Terminator area = 2 * (length * width
Terminator area = 2 * (T1
Terminator area = 2 * (T1 +
Terminator area = 2 * (T1 + width
Terminator area = 2 * (T1 + width *
Terminator area = 2 * (T1 + width * height
Terminator area = 2 * (T1 + T2
Terminator area = 2 * (T3
Terminator area = 2 * (T3 +
Terminator area = 2 * (T3 + height
Terminator area = 2 * (T3 + height *
Terminator area = 2 * (T3 + height * length
Terminator area = 2 * (T3 + T4
Terminator area = 2 * (T5
Terminator area = 2 * (T5)
Terminator area = 2 * T5
Terminator area = T6
Terminator Terminator
PARSE-COMPLETE

assemblycode.txt

```
sys_exit  equ       1
sys_readequ       3
sys_write         equ       4
stdin             equ       0
stdout            equ       1
stderr            equ       3

section .data

          userMsg           db        'Input integer(less than 32,765): '
          lenUserMsg        equ       $-userMsg
          displayMsg        db        'You entered: '
          lenDisplayMsg     equ       $-displayMsg
          newline           db        0xA
          Ten       DW      10
          Result        db      'Ans = '
          ResultValue       db        'aaaaa'
                            db         0xA
          ResultEnd      equ          $-Result
          num               times 6   db 'ABCDEF'
          numEnd            equ       $-num
                                      ;initialized data from program goes here


section   .bss

          TempChar       RESB    1
          testchar       RESB    1
          ReadInt        RESW    1
          tempint        RESW          1
          negflag        RESB    1
                                      ;uninitialized data from program goes here

          area      RESB      1
          length    RESB      1
          width     RESB      1
          height    RESB      1
          T1        RESB      1
          T2        RESB      1
          T3        RESB      1
          T4        RESB      1
          T5        RESB      1
          T6        RESB      1
          T7        RESB      1
          T8        RESB      1
          T9        RESB      1
          T10       RESB      1

section .text
          global main

main:     nop
Again:  call PrintString
          call GetAnInteger
          mov ax,[ReadInt]
          mov [lenght],ax
```

```
    call PrintString
    call GetAnInteger
    mov ax,[ReadInt]
    mov [width],ax

    call PrintString
    call GetAnInteger
    mov ax,[ReadInt]
    mov [height],ax

    mov ax, [length]
    mul word, [width]
    mov [T1], ax

    mov ax, [width]
    mul word, [height]
    mov [T2], ax

    mov ax, [T1]
    add ax, [T2]
    mov [T3], ax

    mov ax, [height]
    mul word, [length]
    mov [T4], ax

    mov ax, [T3]
    add ax, [T4]
    mov [T5], ax

    mov ax, [2]
    mul word, [T5]
    mov [T6], ax

    mov ax, [T6]
    mov [area], ax

    mov ax, [area]
    call ConvertIntegerToString
    mov eax, 4
    mov ebx, 1
    mov ecx, Result
    mov edx, ResultEnd
    int 80h


fini:
    mov eax,sys_exit
    xor ebx,ebx
    int 80h

PrintString:
    push   ax
    push   dx

    mov eax, 4
    mov ebx, 1
```

```
        mov ecx, userMsg
        mov edx, lenUserMsg
        int         80h
        pop     dx
        pop     ax
        ret

GetAnInteger:
        mov eax,3
        mov ebx,2
        mov ecx, num
        mov edx,6
        int 0x80
        mov edx,eax
        mov eax, 4
        mov ebx, 1
        mov ecx, num
        int 80h

ConvertStringToInteger:
        mov ax,0
        mov [ReadInt],ax
        mov ecx,num
        mov bx,0
        mov bl, byte [ecx]

Next:
        sub bl,'0'
        mov ax,[ReadInt]
        mov dx,10
        mul dx
        add ax,bx
        mov [ReadInt], ax

        mov bx,0
        add ecx,1
        mov bl, byte[ecx]
        cmp bl,0xA
        jne Next
        ret

ConvertIntegerToString:
        mov ebx, ResultValue + 4

ConvertLoop:
        sub dx,dx
        mov cx,10
        div cx
        add dl,'0'
        mov [ebx], dl
        dec ebx
        cmp ebx,ResultValue
        jge ConvertLoop
        ret
```