```
.data
tabToken: .space 240 # Space for 20 entries of 3 words each
inBuf: .space 100 # Input buffer
curToken: .space 9 # Current token string
token_Space: .word 8 # Token space

# State table
tabState:
Q0: .word ACT1
 .word Q1 # T1
 .word Q1 # T2
 .word Q1 # T3
 .word Q1 # T4
 .word Q1 # T5
 .word Q1 # T6
 .word Q11 # T7
Q1: .word ACT2
 .word Q2 # T1
 .word Q5 # T2
 .word Q3 # T3
 .word Q3 # T4
 .word Q4 # T5
 .word Q0 # T6
 .word Q11 # T7
Q2: .word ACT1
 .word Q6 # T1
 .word Q7 # T2
 .word Q7 # T3
 .word Q7 # T4
 .word Q7 # T5
 .word Q7 # T6
 .word Q11 # T7
Q3: .word ACT4
 .word Q0 # T1
 .word Q0 # T2
 .word Q0 # T3
 .word Q0 # T4
 .word Q0 # T5
 .word Q0 # T6
 .word Q11 # T7
Q4: .word ACT4
 .word Q10 # T1
 .word Q10 # T2
 .word Q10 # T3
 .word Q10 # T4
 .word Q10 # T5
 .word Q10 # T6
 .word Q11 # T7
Q5: .word ACT1
 .word Q8 # T1
 .word Q8 # T2
 .word Q9 # T3
 .word Q9 # T4
 .word Q9 # T5
 .word Q9 # T6
 .word Q11 # T7
Q6: .word ACT3
 .word Q2 # T1
 .word Q2 # T2
 .word Q2 # T3
 .word Q2 # T4
 .word Q2 # T5
 .word Q2 # T6
 .word Q11 # T7
Q7: .word ACT4
 .word Q1 # T1
 .word Q1 # T2
 .word Q1 # T3
 .word Q1 # T4
 .word Q1 # T5
 .word Q1 # T6
 .word Q11 # T7
Q8: .word ACT3
 .word Q5 # T1
 .word Q5 # T2
 .word Q5 # T3
 .word Q5 # T4
 .word Q5 # T5
 .word Q5 # T6
 .word Q11 # T7
Q9: .word ACT4
 .word Q1 # T1
 .word Q1 # T2
 .word Q1 # T3
 .word Q1 # T4
 .word Q1 # T5
 .word Q1 # T6
 .word Q11 # T7
Q10: .word RETURN
 .word Q10 # T1
 .word Q10 # T2
 .word Q10 # T3
 .word Q10 # T4
 .word Q10 # T5
 .word Q10 # T6
 .word Q11 # T7
Q11: .word ERROR
 .word Q4 # T1
 .word Q4 # T2
 .word Q4 # T3
 .word Q4 # T4
 .word Q4 # T5
 .word Q4 # T6
 .word Q4 # T7

.text

# Main function
main:
    # Call getLine
    la $s1, Q0 # Initial state = Q0
    li $s0, 1 # T = 1

nextState:
    lw $s2, 0($s1) # Load this state's ACT
    jalr $v1, $s2 # Call ACT, save return addr in $v1
    sll $s0, $s0, 2 # Mult t by 4 for word boundary
    add $s1, $s1, $s0 # Add T to current state index
    sra $s0, $s0, 2 # Divide by 4
    lw $s1, 0($s1) # goto next state
    b nextState


# Function to handle token type 1 (Number)
ACT1:
    # Load character into a register from inBuf
    lb $t0, inBuf($zero)

    # Check if character is a digit
    li $t1, 48 # ASCII code for 0
    blt $t0, $t1, handle_error # If the character is less than 0 not number
    li $t1, 57 # ASCII code for 9
    bgt $t0, $t1, handle_error # If character is greater than 9, not number

    # make current character to curToken string
    sb $t0, curToken($zero)

    # decrease token_Space to indicate that a character was added to current token
    lw $t2, token_Space
    addi $t2, $t2, -1
    sw $t2, token_Space

    #Load token_space value to register
    lw $t3, token_Space

    # Check if token_Space has reached 0 meaning its full
    beq $t3, $zero, token_complete

    # Move to the next character in inBuf
    addi $s6, $s6, 1 # Increment index to inBuf
    j ACT1_continue

ACT1_continue:
    # Continue reading characters from inBuf
    lb $t0, inBuf($s6)

    # Check if the character is a digit
    li $t1, 48 # ASCII code for 0
    blt $t0, $t1, token_complete # If  character < 0, token is complete
    li $t1, 57 # ASCII code for 9
    bgt $t0, $t1, token_complete # If the character > 9, token is complete

    # Append the current character to the curToken string
    sb $t0, curToken($zero)

    # Decrement token_Space to indicate that a character has been added to the current token
    lw $t2, token_Space
    addi $t2, $t2, -1
    sw $t2, token_Space

    #  if token_Space has reached 0 then  token is full
    beq $t2, $zero, token_complete

    # Continue reading characters from inBuf
    addi $s6, $s6, 1 # Increment index to inBuf
    j ACT1_continue

# Function to handle token type 2 (Variable)
ACT2:
    # Load current character into a register from inBuf
    lb $t0, inBuf($zero)

    # Check if the current character is an alphabetic letter (ASCII a to z or A to Z)
    li $t1, 65 # ASCII code for A
    blt $t0, $t1, handle_error # If the character is less than A, it's not a variable
    li $t1, 90 # ASCII code for Z
    bgt $t0, $t1, check_lower_case # If the character is greater than 'Z', check if it's lower case
    j variable_complete

check_lower_case:
    # Check if the current character is a lower case alphabetic letter (ASCII a to z)
    li $t1, 97 # ASCII code for a
    blt $t0, $t1, handle_error # If the character is less than a, it's not a variable
    li $t1, 122 # ASCII code for -z
    bgt $t0, $t1, handle_error # If the character is greater than z, it's not a variable

variable_complete:
    #Append the current character to the curToken string
    sb $t0, curToken($zero)

    # Decrement token_Space because a character is added to the current token
    lw $t2, token_Space
    addi $t2, $t2, -1
    sw $t2, token_Space

    # Check if token_Space has reached 0
    beq $t2, $zero, token_complete

    # Move to the next character in inBuf
    addi $s6, $s6, 1 # Increment index to inBuf
    j ACT2_continue

ACT2_continue:
    # Continue reading characters from inBuf
    lb $t0, inBuf($s6)

    # Check if the current character is an alphabetic letter
    li $t1, 65 # ASCII code for A
    blt $t0, $t1, variable_complete # If the character is less than a, token is complete
    li $t1, 90 # ASCII code for Z
    bgt $t0, $t1, check_lower_case_continue # If the character is greater than 'Z', check if it's lower case
    j variable_complete

check_lower_case_continue:
    # Check if the current character is a lower case alphabetic letter (ASCII a to z)
    li $t1, 97 # ASCII code for a
    blt $t0, $t1, handle_error # If the character is less than a, it's not a variable
    li $t1, 122 # ASCII code for z
    bgt $t0, $t1, handle_error # If the character is greater than z

    # Append the current character to the curToken string
    sb $t0, curToken($zero)

    # decrement token_Space  bc a character is added to the current token
    lw $t2, token_Space
    addi $t2, $t2, -1
    sw $t2, token_Space

    # check if token_Space has reached 0 (the token is full)
    beq $t2, $zero, token_complete

    # continue reading from inBuf
    addi $s6, $s6, 1 # Increment index to inBuf
    j ACT2_continue


# function to handle token type 3 (Operator)
ACT3:
    # Load the current character into a register from inBuf
    lb $t0, inBuf($zero)

    # Check if the curent character is a valid operator (*, +, -, /)
    li $t1, 42 # ASCII code for *
    beq $t0, $t1, operator_found
    li $t1, 43 # ASCII code for +
    beq $t0, $t1, operator_found
    li $t1, 45 # ASCII code for -
    beq $t0, $t1, operator_found
    li $t1, 47 # ASCII code for /
    beq $t0, $t1, operator_found
    j handle_error # If the character is not a valid operator, call error
```

```
operator_found:
    # Append the current character to the curToken string
    sb $t0, curToken($zero)

    # Decrement token_Space because a character is added to the current token
    lw $t2, token_Space
    addi $t2, $t2, -1
    sw $t2, token_Space

    # check if token_Space has reached 0, then token is full)
    beq $t2, $zero, token_complete

    # Move to the next character in inBuf
    addi $s6, $s6, 1 # Increment index to inBuf
    j ACT3_continue

ACT3_continue:
    # continue reading characters from inBuf
    lb $t0, inBuf($s6)

    # Check if the current character is a valid operator (*, +, -, /)
    li $t1, 42 # ASCII code for *
    beq $t0, $t1, operator_found
    li $t1, 43 # ASCII code for +
    beq $t0, $t1, operator_found
    li $t1, 45 # ASCI code for -
    beq $t0, $t1, operator_found
    li $t1, 47 # ASCII code for /
    beq $t0, $t1, operator_found

    # If the character is not a valid operator, token is complete
    j token_complete

# Function to handle token type 4 (Delimiter)
ACT4:
    # load the current character into a register from inBuf
    lb $t0, inBuf($zero)

    # Check if the current character is a valid delimiter (. ( ) , : $)
    li $t1, 46 # ASCII code for .
    beq $t0, $t1, delimiter_found
    li $t1, 40 # ASCII code for (
    beq $t0, $t1, delimiter_found
    li $t1, 41 # ASCII code for )
    beq $t0, $t1, delimiter_found
    li $t1, 44 # ASCII code for ,
    beq $t0, $t1, delimiter_found
    li $t1, 58 # ASCII code for :
    beq $t0, $t1, delimiter_found
    li $t1, 36 # ASCII code for $
    beq $t0, $t1, delimiter_found
    j handle_error # If the character is not a valid delimiter, call an error

delimiter_found:
    # Append the current character to the curToken string
    sb $t0, curToken($zero)

    # decrement token_Space because a character is added to the current token
    lw $t2, token_Space
    addi $t2, $t2, -1
    sw $t2, token_Space

    #check if token_Space has reached 0
    beq $t2, $zero, token_complete

    #Move to the next character in inBuf
    addi $s6, $s6, 1 # Increment index to inBuf
    j ACT4_continue

ACT4_continue:
    # Continue reading characters from inBuf
    lb $t0, inBuf($s6)

    # Check if the current character is a valid delimiter (. ( ) , : $)
    li $t1, 46 # ASCII code for .
    beq $t0, $t1, delimiter_found
    li $t1, 40 # ASCII code for (
    beq $t0, $t1, delimiter_found
    li $t1, 41 # ASCII code for )
    beq $t0, $t1, delimiter_found
    li $t1, 44 # ASCII code for <
    beq $t0, $t1, delimiter_found
    li $t1, 58 # ASCII code for :
    beq $t0, $t1, delimiter_found
    li $t1, 36 # ASCII code for $
    beq $t0, $t1, delimiter_found
    # if the character is not a valid delimiter, token is complete
    j token_complete


# Function to handle errors
ERROR:
handle_error:
    # print an error message
    li $v0, 4
    la $a0, error_message
    syscall

    # exit program
    li $v0, 10
    syscall

# Function to return from state 10
RETURN:
    jr $ra

# Function to print the token table
printToken:
    # Load the address of the token table header into $a0
    la $a0, tableHead

    # print the token table header
    li $v0, 4
    syscall

    # Initialize index for looping through tabToken
    li $t0, 0

print_loop:
    #calculate the address of the current entry in tabToken
    la $t1, tabToken
    add $t1, $t1, $t0

    # Load the token string and type from tabToken
    lw $t2, 0($t1)    # Load token_string
    lw $t3, 4($t1)    # Load token type

    # Print the token string
    move $a0, $t2     # Load token_string into $a0
    li $v0, 4         # Load print string syscall code into $v0
    syscall           # Print token_string

    # Print the token type
    li $v0, 1         # Load print integer syscall code into $v0
    move $a0, $t3     # Load tokentype into $a0
    syscall           # Print token type

    #Print a newline character
    li $v0, 4         # Load print string syscall code into $v0
    la $a0, newline   # Load newline character address into $a0
    syscall           # Print newline character

    # increment index to point to the next entry in tabToken
    addi $t0, $t0, 8

    # Check if the end of tabToken has been reached
    bge $t0, $a3, print_done

    # Repeat loop
    j print_loop

print_done:
    jr $ra  # Return from printToken
```