# COMP.2030 HW 5: Scanner

Given an input string, a scanner collects a coherent sequence of characters with the same character types and saves it as a single unit, called a token. The scanner then saves each token into a table with two columns: a token string and token type. To simplify the representation in memory, each token string will contain a maximum of 8 characters. That way, each entry in the token table always takes up exactly 12 bytes (8 bytes for a token and a word for its type).

The numeric encoding of token types is identical to the character types used in HW 3, but does not include the blank character type any longer since the scanner will not save blank tokens.

| | | |
|---|---|---|
| Token type 1 -- Number | : | 0 1 .. 9 |
| Token type 2 -- Variable | : | An alphabetic letter followed by alpha-numeric characters |
| Token type 3 – Operator | : | * + - / |
| Token type 4 – Delimiter | : | . ( ) , : $ |
| Token type 5 -- End of Line: | | # |

For example, given an input string below,

```
Thisloop:   li    $t0,63            #
```

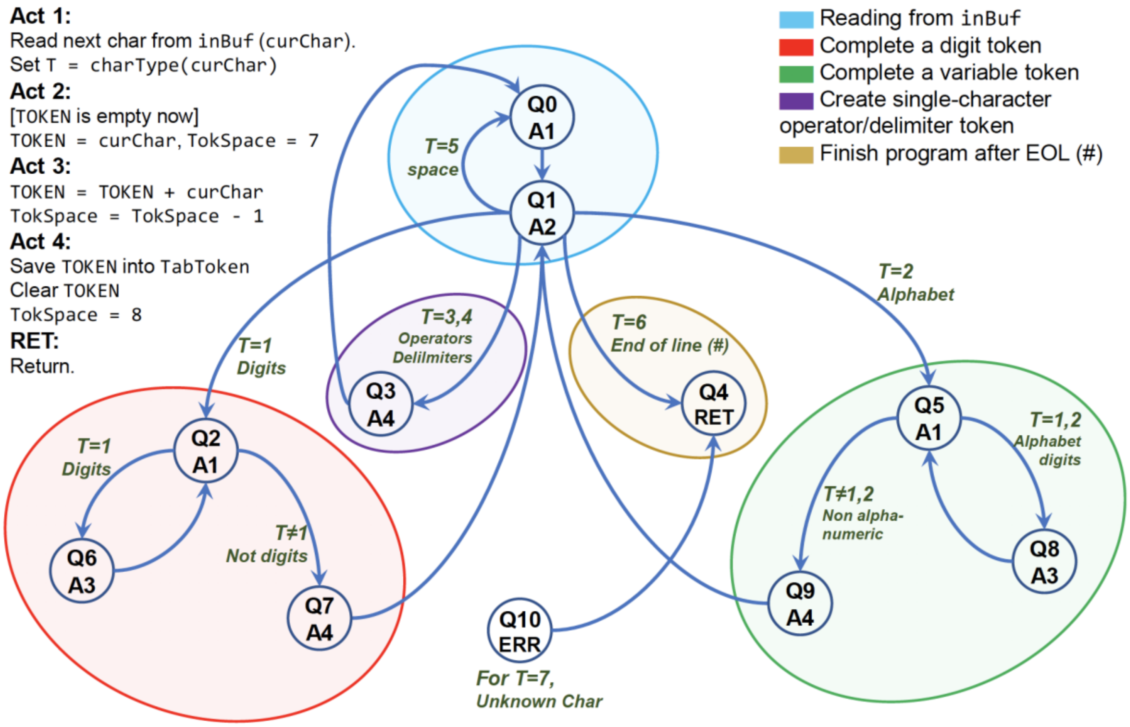
the output of HW4 will be a dump of the token table, tabToken, is as follows.

```
     Token             Token Type
-----------------    ----------------
     Thisloop               2
     :                      4
     li                     2
     $                      4
     t0                     2
     ,                      4
     63                     1
     #                      5
```

In the program, you need to reserve an enough space in memory for the **tabToken** table; 20 entries of 3 words each should be sufficient for a single input string of an MIPS instruction. The same token table will be overwritten each time a new input string is processed.

The behavior of the Scanner can be described (defined) by the state (transition) diagram shown below. To translate this state machine into a program, we can encode the diagram into a state table stored in memory and have Scanner program simulate the finite automata. The Scanner simulation steps through the states in the diagram, beginning in state Q0. At each state, Scanner calls the action function associated with the state and then uses the value of the variable T to look up next state to transition to according to the state transition table **tabState**.

The variable **T** holds the type of the character in inBuf which is being examined. It is identical to the return value of function search in HW. The variable **curToken** holds the token string as it is being assembled. The variable, **tokSpace**, is initialized to 8, and is decremented each time a new character (byte) is appended to the current token in curToken.



**Act 1:**
Read next char from inBuf (curChar).
Set T = charType(curChar)
**Act 2:**
[TOKEN is empty now]
TOKEN = curChar, TokSpace = 7
**Act 3:**
TOKEN = TOKEN + curChar
TokSpace = TokSpace - 1
**Act 4:**
Save TOKEN into TabToken
Clear TOKEN
TokSpace = 8
**RET:**
Return.

🟦 Reading from inBuf
🟥 Complete a digit token
🟩 Complete a variable token
🟪 Create single-character operator/delimiter token
🟨 Finish program after EOL (#)

The state-diagram table for our Scanner Program.

The same information specified through the state transition diagram of a finite state automata can be expressed in a tabular form (the numbers used in state table below is slightly different from the diagram above, but it still accurately describes the same automata). The entire table to be included in the MIPS program is found at the end of this assignment.

| State/T | | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| Q0 | ACT1 | Q1 | Q1 | Q1 | Q1 | Q1 | Q1 | Q11 |
| Q1 | ACT2 | Q2 | Q5 | Q3 | Q3 | Q4 | Q0 | Q11 |
| Q2 | ACT1 | Q6 | Q7 | Q7 | Q7 | Q7 | Q7 | Q11 |
| Q3 | ACT4 | | | | | | | |
| Q4 | ACT4 | Q10 | Q10 | Q10 | Q10 | Q4 | Q10 | Q11 |
| Q5 | ACT1 | | | | | | | |
| Q6 | ACT3 | | | | | | | |
| Q7 | ACT4 | | | | | | | |
| Q8 | ACT3 | | | | | | | |
| Q9 | ACT4 | | | | | | | |
| Q10 | RETURN | Q10 | Q10 | Q10 | Q10 | Q10 | Q10 | Q11 |
| Q11 | ERROR | Q4 | Q4 | Q4 | Q4 | Q4 | Q4 | Q4 |

The algorithm for tracing through the states and a section of the state table are given below. **CUR** holds the current state and **T** has the type of the current character.

Scanner Algorithm

```
1)    Call getLine
2)    CUR = Q0;   T=1
3)    ACT = tabState[CUR][0]
      CALL ACT
4)    CUR = tabState[CUR][T]
5)    GO TO 3
```

**Use `$s0` and `$s1` to hold the value of T and CUR**, respectively. Steps 2 through 4 can be coded in MIPS as follows:

```
            la    $s1, Q0          # Initial state = Q0
            li    $s0, 1           # Initial 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      # Multiply T by 4 for word boundary
            add   $s1, $s1, $s0    # Add T to current state index
            sra   $s0, $s0, 2      # Divide by 4 to restore original T
            lw    $s1, 0($s1)      # Transition to next state
            b     nextState
```

In this assignment, you need to write four short functions (ACT1, ACT2, ACT3 and ACT4) and some bookkeeping functions. You are free to handle ERROR function as you wish.

**Note**: To print the token table, it is easier to copy each entry of `tabToken` to a separate 3-word space and print one row at a time. A function to dump the token table is below, and you are welcome to include it in your program. The `printToken` function assumes that you have `$a3` pointing to the last entry in tabToken in bytes.

```
#######################################################################
#
#  printToken:
#     print Token table header
#     copy each entry of tabToken into prToken and print TOKEN
#
#     in Main(), $a3 has the byte index to last entry in the tabToken
#
########################################################################
```

```
                .data
prToken:        .word  0:3                # space to copy one token at a time
tableHead:      .asciiz "TOKEN    TYPE\n"

                .text
printToken:
                la     $a0, tableHead      # print table heading
                li     $v0, 4
                syscall

                # copy 2-word token from tabToken into prToken
                #  run through prToken, and replace 0 (Null) by ' ' (0x20)
                #  so that printing does not terminate prematurely
                li     $t0, 0
loopTok:        bge    $t0, $a3, donePrTok # if ($t0 <= $a3)

                lw     $t1, tabToken($t0)  #   copy tabTok[] into prTok
                sw     $t1, prToken
                lw     $t1, tabToken+4($t0)
                sw     $t1, prToken+4

                li     $t7, 0x20           # blank in $t7
                li     $t9, -1             # for each char in prTok
loopChar:       addi   $t9, $t9, 1
                bge    $t9, 8, tokType
                lb     $t8, prToken($t9)   #   if char == Null
                bne    $t8, $zero, loopChar
                sb     $t7, prToken($t9)   #       replace it by ' ' (0x20)
                b      loopChar

                # to print type, use four bytes: ' ', char(type), '\n', and Null
                #  in order to print the ASCII type and newline
tokType:
                li     $t6, '\n'           # newline in $t6
                sb     $t7, prToken+8
                #sb    $t7, prToken+9
                lb     $t1, tabToken+8($t0)
                addi   $t1, $t1, 0x30      # ASCII(token type)
                sb     $t1, prToken+9
                sb     $t6, prToken+10     # terminate with '\n'
                sb     $0, prToken+11

                la     $a0, prToken        # print token and its type
                li     $v0, 4
                syscall

                addi   $t0, $t0, 12
                sw     $0, prToken         # clear prToken
                sw     $0, prToken+4
                b      loopTok

donePrTok:
                jr     $ra
```

=========== State table to be copied into a data segment

```
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
```