Write a MIPS program that repeatedly reads one line at a time from console. For each line that you read, print two lines on the console: echo print of the input line and types of individual characters on the line as defined by:

<table>
<thead>
<tr>
<th>Characters</th>
<th>Type</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 1 .. 9</td>
<td>1</td>
<td>Digits</td>
</tr>
<tr>
<td>A B .. Z a b .. z</td>
<td>2</td>
<td>Letters</td>
</tr>
<tr>
<td>* + - /</td>
<td>3</td>
<td>Operators</td>
</tr>
<tr>
<td>. ( ) , :</td>
<td>4</td>
<td>Delimiters</td>
</tr>
<tr>
<td>b</td>
<td>5</td>
<td>blank</td>
</tr>
<tr>
<td>#</td>
<td>6</td>
<td>End of the Line</td>
</tr>
</tbody>
</table>

For example, when a line of the input below

\[\text{THISLOOP: LWU R2, 3 \#}\]

the output from the program will be like

\[\text{THISLOOP: LWU R2, 3 \#} \\
\text{222222224 222 2141 6}\]

The blank type of 5 can be left out.
And this process repeats itself by inputting another line, etc.

**Approaches:**

1. Reading input lines

Each input line can be read by calling a procedure, called getline, by allocating the maximum of 80 character space and calling a syscall for reading a string. Make sure that your main program calls getline repeatedly until a special symbol ("\#") is entered to terminate the loop. A procedure call to getline is made by the following MIPS statement:

\[\text{jal getline}\]

An example of getline procedure to read an input string:

\[
\begin{align*}
\text{getline:} \\
\text{la} & \quad \$a0, \text{st\_prompt} \quad \text{# Prompt to enter a new line} \\
\text{li} & \quad \$v0, 4 \\
\text{syscall} \\
\text{la} & \quad \$a0, \text{inBuf} \quad \text{# read a new line} \\
\text{li} & \quad \$a1, 80 \\
\text{li} & \quad \$v0, 8 \\
\text{syscall} \\
\text{jr} & \quad \$ra
\end{align*}
\]
2. Linear search for a character

A simple approach to finding the type of a character is to arrange all characters and their types into an array. This makes a change to the character set simple. Use the following table and write a linear search program to perform the search for an input character from the input string. Be careful that a character from the input string is stored in a byte whereas the search table below is organized in units of words (4-byte each). The linear search part in your program HAS TO BE coded as a function. When you compare a letter from the input string to characters in charTab, make sure that you use the ‘lb’ instruction to move only a byte of the input string to one of the registers.

```
.data
charTab:  .word ' ', 5
          .word '#', 6
          .word '(', 4
          .word ')', 4
          .word '*', 3
          .word '+', 3
          .word ',', 4
          .word '-', 3
          .word '.', 4
          .word '/', 3
          .word '0', 1
          .word '1', 1
          .word '2', 1
          .word '3', 1
          .word '4', 1
          .word '5', 1
          .word '6', 1
          .word '7', 1
          .word '8', 1
          .word '9', 1
          .word ':', 4
          .word 'A', 2
          .word 'B', 2
          .word 'C', 2
          .word 'D', 2
          .word 'E', 2
          .word 'F', 2
          .word 'G', 2
          .word 'H', 2
```
What to hand in:

Submit source code.

`submit cs203 s1.prog2 your_directory`

NOTE: Comments in an assembly program are very important. Especially, it is difficult to follow which registers contain what values. Too many comments clutter the program and make it harder to read. Be judicious about comments so that one can follow the flow of the program you’re your comments.