Question 5 (30 points): In this question you will write `convertCommandsToString`, a subroutine that receives two parameters:

`$a0:` address of a word that contains the binary representation for commands

`$a1:` address of `stringCommandBuffer`, a buffer of characters where the string commands should be written

There are 16 binary commands in the binary representation of the commands. Each individual command is represented in two consecutive bits. The first command is formed by the two most-significant bits (bits 31-30) and the last command is formed by the two least-significant bits (bits 1-0). This is the correspondence between binary commands and string commands:

Binary	String
00	left
01	right
10	up
11	down

The subroutine `convertCommandsToString` must create a string representation of the commands that starts with the character '<', is followed by the string representation of the commands corresponding to the binary commands, separated by comma, and ends with the character '>'. For instance, if the binary representation of the commands is:

```
010011 ... 1000
```

Then `convertCommandsToString` will create the string:

```
<right,left,down, ... ,up,left>
```

Where `...` in the example above represents other commands that appear in the binary representation and in the string representation.

The person that is passing this task to you already started the organization of the data in memory and created the strings that you need in the data segment. She also has written a subroutine that creates a vector of pointers to the strings `left`, `right`, `up`, and `down`. You should expect that the function `createStringPointerVector` will be invoked before `convertCommandsToString` with the following invocation:

```
la      $a0, stringPointerVector
jal     createStringPointerVector
```

Thus you can be sure that at the address of `StringPointerVector` your subroutine will find a vector of pointers to strings as created by `createStringPointerVector`. You should use this vector when implementing `convertCommandsToString`. You should also invoke `concatenate` to append strings to the buffer when implementing `convertCommandsToString` — You can use `concatenate` in this question even if you did not write a correct solution to the previous question. You must follow all the subroutine invocation conventions of MIPS.

The data segment and the code for `createStringPointerVector` are shown in Figure 1.

```
.data
left:           .asciiz  "left"
right:          .asciiz  "right"
up:             .asciiz  "up"
down:           .asciiz  "down"
comma:          .asciiz  ","
leftBracket:    .asciiz  "<"
rightBracket:   .asciiz  ">"

stringPointerVector:
    .word 4
stringCommandBuffer:
    .byte  0       # buffer is initialized with a null character in the first position
    .space 200

.text
# createStringPointerVector
# arguments:
#   $a0: address of the vector of pointers
# expects:
#   allocation of null-terminated strings with labels "left", "right", "up", "down" in data segment
#
createStringPointerVector:
    la      $t0, left
    sw      $t0, 0($a0)    # stringPointerVector[0] <- &left
    la      $t0, right
    sw      $t0, 4($a0)    # stringPointerVector[1] <- &right
    la      $t0, up
    sw      $t0, 8($a0)    # stringPointerVector[2] <- &up
    la      $t0, down
    sw      $t0, 12($a0)   # stringPointerVector[3] <- &down
    jr
```

Figure 1: Data Segment and subroutine `createStringPointerVector`

MIPS code for `convertCommandsToString`