You are developing a new App where you can record the movement of a ball in a real-life shuffle board game and then convert it into movements of a blinking ball in a grid on the screen. Unfortunately the trajectory-capturing software records the movement of the real ball into a binary representation and the image-rendering library that you are using was written in JavaScript and expects a string of word commands. An important mobile device in the target market is based on a MIPS microcontroller that uses MIPS assembly, but there is no compiler available for this controller yet. Thus you will have to write MIPS assembly routines to do this conversion. To make the task easier, you have divided the functionality that you need into two separate routines.

Question 4 (20 points): In this question you will write `concatenate`, a subroutine that receives two pointers to null-terminated strings: `$a0` receives the `_to` pointer, which is the address of the string that will be grown by the concatenation; `$a1` receives the `_from` pointer, which is the address of the string that is to be concatenated into the string pointed by `_to`. Figure 1 illustrates the effect of one call to the subroutine concatenate. You can assume that there is enough memory allocated for the `_to` string to enable the concatenation of the `_from` string without overwriting to other data structures in the program. You must follow all the subroutine invocation conventions of MIPS.

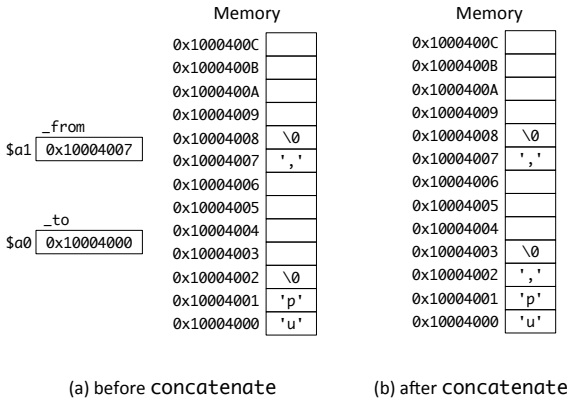

Figure 1: Example of execution of a concatenate function.

Figure 2 shows an implementation of `concatenate`

```
# concatenate
# receives two pointers to null-terminated strings (_to and _from)
# and concatenates the string pointed by the pointer _from
# into the string pointed to by the pointer _to
# example: if _from points to the string "set"
#             and _to points to the string "up"
#          then after concatenate, _to will point to the string "upset"
# arguments:
#   $a0: _to is the address of the string to be grown
#   $a1: _from is the address of the string to be appended to _to
# register usage:
#   $a0: p_to is a pointer to _to array
#   $a1: p_from is a pointer to _from array
concatenate:
    lb      $t0, 0($a0)         # $t0 <- first character of _to
    beq     $t0, $zero, append  # if $t0 == null found the end of _to
nextchar:
    addi    $a0, $a0, 1         # p_to++
    lb      $t0, 0($a0)         # $t0 <- next character in _to
    bne     $t0, $zero, nextchar    # if $t0 != null get next _from character
append:
    # $a0 contains the position where _from should be placed into _to
    lb      $t1, 0($a1)         # $t1 <- next character in _from
    sb      $t1, 0($a0)         # *p_to <- $t1
    addi    $a0, $a0, 1         # p_to++
    addi    $a1, $a1, 1         # p_from++
    bne     $t1, $zero, append  # if $t1 != null append next _from character into _to
    jr      $ra
```

Figure 2: A solution for `concatenate`.