1. Write a MIPS assembly language program that reads a single line of keyboard input from the user, capitalizes the first letter of each English word in that line, and prints the result as output. You may assume the user will only enter capital letters (A-Z), lowercase letters (a-z), spaces, and a single linefeed \([LF]\) character at the end (when the user press the enter key to send input to the program). You may assume words are separated by spaces. Think carefully about corner cases.

For example: if the user entered \texttt{I like computer science}\([LF]\), your program should print \texttt{I Like Computer Science}\([LF]\).

To receive full marks, your code must run without errors on the \texttt{noargs} simulator available on the UWaterloo student computer science server. There are no subroutines involved in this question. You do not need to follow the register conventions. Format and comment your code to be readable by the markers.

Submit the file \texttt{a3q1.asm} (10 Marks).

Answer: Here is just one of many possible correct solutions.

```assembly
lis $1
.l.word 0xFFFF000C ; standard output
lis $2
.l.word 0xFFFF0004 ; standard input
addi $3, $0, 1 ; register $3 will indicate if the previous
; character was a space, because if the
; previous character was a space, then we
; will capitalize the next letter.
; start $3 at 1, since we want to
; capitalize the very first letter

read: lw $5, 0($2) ; read a character from standard in
addi $4, $0, 10 ; 10 (base-10) is ACSII for [LF]
beq $5, $4, printlf ; if it's a linefeed, print it and end
addi $4, $0, 32 ; 32 (base-10) is ACSII for space
bne $5, $4, nospace ; if it's not a space, skip to nospace
addi $3, $0, 1 ; otherwise set the indicator that the
; last char was a space
beq $0, $0, print ; and go to printing the char
```
nospace:
    beq $3, $0, print ; if the last character was not a
          ; space, then we can just print this
          ; character
    addi $3, $0, 0 ; otherwise the last character
          ; was a space, so clear the indicator
          ; so we’re not confused about the
          ; next character
    addi $4, $0, 97 ; lowercase letters are ASCII 97-122 (base-10)
    slt $6, $5, $4 ; if the character is less than 97
    bne $6, $0, print ; just print it
    addi $4, $0, 122
    slt $6, $4, $5 ; if the character is greater than 122
    bne $6, $0, print ; also just print it
    addi $5, $5, -32 ; the difference between 'a' and 'A'
          ; in ASCII is -32 (base-10), so if we
          ; subtract 32 from a lowercase letter,
          ; we will get it’s uppercase equivalent
          ; now we continue to print

print:    sw $5, 0($1) ; print the character
          beq $0, $0, read ; and read the next one

printlf:  sw $5, 0($1) ; write out the final linefeed

end:      jr $31 ; end the program
2. Write a MIPS assembly language program that contains a subroutine `squarediff` which calculates the squared difference between two integers passed as arguments.

For example: \( \text{squarediff}(6_{10}, 8_{10}) = (6_{10} - 8_{10})^2 = 4_{10} \)

Call this subroutine on each pair of integers in an input array, and calculate the average of the results.

For example: in the input array \([6_{10}, 8_{10}, 3_{10}, 7_{10}]\) there are two pairs \((6_{10}, 8_{10})\) and \((3_{10}, 7_{10})\). The `squarediff` results on these pairs are \(4_{10}\) and \(16_{10}\), and the average of those squares (and the final result of the computation) is \(10_{10}\).

If the value of the average is a fraction, round down (use the quotient result from division). You can assume the input array will be of even length, and contain at least two values. You can assume, at the beginning of your program, that the memory address of the start of the array is in register $1$, and the number of items in the array is in register $2$ (this is how the array simulator sets up your program). Place the final result of your computation in register $3$. You may assume all input numbers, intermediate computation results, and final results are valid 32-bit two’s complement integers.

To receive full marks, your code must run without errors on the array simulator available on the UWaterloo student computer science server. Your subroutine must follow the register conventions to receive full marks. Format and comment your code to be readable by the markers.

Submit the file `a3q2.asm` (10 Marks).

Answer: Here is just one of many possible correct solutions.

```assembly
add $20, $0, $1 ; move the array address to a saved ; temporary register
add $21, $0, $2 ; move the item count to a saved ; temporary register
addi $16, $0, 0 ; keep a count of how many numbers ; we’ve handled so far
addi $17, $0, 0 ; the results starts as zero
loop: lw $4, 0($20) ; load the first two items of the array ; into the arguments registers for ; the squarediff subroutine
lw $5, 4($20)
add $30, $30, -4
sw $31, 0($30) ; save the return address
jal squarediff ; call squarediff
lw $31, 0($30) ; load the return address
```
addi $30, $30, 4

add $17, $17, $2 ; add the difference to the sum so far
addi $16, $16, 2 ; add 2 to the count of numbers handled
addi $20, $20, 8 ; advance the array memory address
                     ; by two items
bne $16, $21, loop ; loop if we have more numbers left
addi $8, $0, 2 ; divide the length of the array by
div $21, $8 ; 2 to get the number of pairs
mflo $21 ; to divide by for the average
div $17, $21 ; divide by the number of pairs
mflo $3 ; and take the result
jr $31

squarediff:
sub $8, $4, $5 ; compute the difference
mult $8, $8 ; square the difference and put
mflo $2 ; it in the output register
jr $31 ; end the subroutine
3. Write a MIPS assembly language program that contains a recursive subroutine `sumsuffixmax` which calculates the sum of the largest integer in each suffix array of an input array.

For example: the input array [6, 10, 8, 10, 3, 10] has three suffix arrays [6, 10, 8, 10, 3, 10], [8, 10, 3, 10], and [3, 10]. The largest integers in each of these suffixes are 8, 8, and 3. The sum of these largest integers (and the final result of `sumsuffixmax` on this array) is 19.

The `sumsuffixmax` subroutine must take two arguments: the first is the address in memory of the input array, and the second is the length of that array. The `sumsuffixmax` subroutine should call itself recursively on the suffixes of the input array, where each recursive call calculates one maximum. Your program must not contain any code outside of the `sumsuffixmax` subroutine except a short snippet that calls the subroutine with the inputs from the array simulator and then ends the program.

An empty array should produce a result of 0. You can assume, at the beginning of your program, that the memory address of the start of the input array is in register $1, and the number of items in the array is in register $2 (this is how the array simulator sets up your program). Leave the final result of your computation in the register it ends up in when `sumsuffixmax` returns according to the register convention. You may assume all input numbers, intermediate computation results, and final results are valid 32-bit two’s complement integers.

To receive full marks, your code must run without errors on the array simulator available on the UWaterloo student computer science server. Your subroutine must follow the register conventions to receive full marks. Format and comment your code to be readable by the markers.

Submit the file `a3q3.asm` (10 Marks).

Answer: Here is just one of many possible correct solutions.

```assembly
add $4, $0, $1 ; move the array address to an argument
add $5, $0, $2 ; move the item count to an argument
addi $30, $30, -4
sw $31, 0($30) ; save the return address
jal sumsuffixmax ; call sumsuffixmax
lw $31, 0($30) ; load the return address
addi $30, $30, 4
jr $31 ; end the program

sumsuffixmax:
addi $2, $0, 0 ; start with the result as zero```

Page 5 of 7
beq $5, $0, end ; if the length of the array 
    ; is zero, then return the result 
    ; which is still zero (base case)

addi $30, $30, -12
sw $31, 0($30) ; save the return address
sw $4, 4($30) ; save the array start address
sw $5, 8($30) ; save the array length

addi $4, $4, 4 ; add 4 to shorten the input array
    ; to make the next suffix
addi $5, $5, -1 ; and decrement its length

jal sumsuffixmax ; call sumsuffixmax recursively
    ; on the suffix
lw $31, 0($30) ; load the return address
lw $4, 4($30) ; load the array start address
lw $5, 8($30) ; load the array length
addi $30, $30, 12

lw $8, 0($4) ; load the first item from the array
    ; it’s the biggest item so far

loop: addi $4, $4, 4 ; go to the next item in the array
    ; and decrement the number of items left
addi $5, $5, -1
beq $5, $0, sum ; if there are no more items in the
    ; array, then add the max to the sum

lw $9, 0($4) ; load the next item

slt $10, $8, $9 ; check if that item is the new max
beq $10, $0, loop ; if it’s not, loop again
add $8, $0, $9 ; otherwise make it the new max
beq $0, $0, loop ; and continue the loop

sum: add $2, $2, $8 ; add the max to the sum from
jr $31 ; end the subroutine

; the recursive call
; and leave that value as the return value