

## The Task

In this project, you will be writing a program that receives a string of characters via the UART, checks if this string is a palindrome, and then uses a print function to print either “Yes” or “No”. A palindrome is a sequence of characters (typically a word or phrase) that is the same both forwards and backwards. For this project, strings will be terminated using a period (.). You may assume that a string will contain at least one letter in addition to a period (e.g., the input, “b.”, should be considered a palindrome). You will not need to handle empty strings, strings containing only a period, or strings containing characters other than letters, spaces, and periods. Your program should be able to handle multiple strings sent one after another or concatenated together. For example, the string: “abba. data.” should print “Yes” followed by “No” on the next line. Spaces should be ignored when checking for a palindrome and the palindrome should not be case sensitive. For example, “A nut for a jar of Tuna.” would be considered a palindrome.

## Print Function

A template PLP project file is available to download on Canvas. The PLP project includes a second *ASM* file titled, *project3\_lib.asm*. This *ASM* file contains the print function used in this project. PLPTool concatenates all *ASM* files within a PLP project into a single location in memory (unless additional *.org* statements have been added to specify different location for code). No changes to *project3\_lib.asm* should be made.

When called, depending on the value in register *\$a0*, the following string will be displayed on the simulated UART device’s output. If *\$a0* contains a zero then “No” will be displayed and if *\$a0* contains a non-zero value (e.g. one) then “Yes” will be displayed. The print function is called using the following instruction:

```
call project3_print
```

To use the print function, your PLP program needs to initialize the stack pointer (*\$sp*) before performing the function call (or any other operations involving the stack pointer). For this reason, the template project file includes an initialization that sets the stack pointer to 0x10fffffc (the last address of RAM).

### Template Structure

The template project file contains six function stubs that need to be implemented. Five are called from the *main* loop and the sixth is called from “*period\_check*”. The template file contains comments with descriptions of what each function needs to do and how it should be implemented. The table below provides a brief description of the functions.

Function Name	Function Description
poll_UART	Polling loop for UART’s status register, reading new character, and indicating character has been read
period_check	Checks if new character is a period and makes a nested function call to <i>palindrome_check</i> if it is
space_check	Skips saving space characters to array
case_check	Converts new character to uppercase if it is lowercase
array_push	Saves new character to array
palindrome_check	Moves inwards from front and back of array and compares characters at each step to determine if the string is a palindrome. If at any point mirroring characters are not equal, then it should use the print function to print “No”. If the comparison reaches or passes the midpoint then the print function should be used to print “Yes”.

```
.org 0x10000000
```

```
# Initializations
```

```
li $sp, 0x10fffffc    # Starting address of empty stack
li $s0, 0xf0000000    # UART base address
li $s1, 0xf0000004    # UART status address
li $s2, 0xf0000008    # UART receive address
```

```
j main
nop
```

```
array_ptr:            # Label pointing to 100 word array
.space 100
li $a0, array_ptr
move $a1, $a0
```

```
yes:
li $a0, 1
call project3_print    #Print function
j main
nop
```

```
no:
li $a0, 0
call project3_print    #Print function
j main
nop
```

```
#####
# Do not make changes to the main loop
#####
main:
```

```
    jal poll_UART
    nop
    jal period_check
    nop
    jal space_check
    nop
    jal case_check
    nop
    jal array_push
    nop
    j main
    nop
```

```
#####
# Do not make changes to the main loop
#####
```

```
# The "poll_UART" function should poll the status register of the UART.
```

# If the 2<sup>1</sup> bit position (ready bit) is set to 1 then it  
 # should copy the receive buffer's value into \$v0 and send  
 # a clear status command (2<sup>1</sup>) to the command register before  
 # returning (a return statement is already included)

poll\_UART:

```
lw $t0, 0($s3)
li $t1, 0b10
and $t2, $t0, $t1
beq $t2, $zero, poll_UART
nop
li $t6, 0x41 #'A'
li $t7, 0x5A #'Z'
slt $t8, $t7, $v0 # Check for Uppercase
slt $t9, $v0, $t6 # Check for Uppercase
beq $t8, $t9, case_check
lw $v0, 0($s4)
sw $t1, 0($s0)
jr $ra
nop
```

# The "period\_check" function should check if the current character (\$v0)  
 # is a period ("."). If it is a period then the function should go to the  
 # label, "palindrome\_check". If the character is not a period then it  
 # should use the included return.

period\_check:

```
li $t3, 0x2E # Assign period to register t1
beq $v0, $t3, palindrome_check
nop
jr $ra
nop
```

# The "space\_check" function should check if the current character (\$v0)  
 # is a space (" "). If it is then it should jump to "main" so  
 # that it skips saving the space character. If not it should  
 # use the included return.

space\_check:

```
li $t4, 0x20
beq $v0, $t4, poll_UART
jr $ra
nop
```

# The "case\_check" function should check if the current character (\$v0)  
 # is a lowercase character (i.e. greater than the ASCII value of 'Z').  
 # If it is then it should convert the value of \$v0 to the uppercase  
 # equivalent before returning. If it is already uppercase then it  
 # should skip converting and return.

```
case_check:
    addiu $v0, $v0, 32
    j array_push
    jr $ra
    nop
```

# The "array\_push" function should save the current character (\$v0) to the  
# current location of the tail pointer, \$s2. Then it should increment the  
# tail pointer so that it points to the next element of the array. Last  
# it should use the included return statement.

```
array_push:
    sw $v0, 0($s1)
    addiu $s1, $s1, 4
    j poll_UART
    jr $ra
    nop
```

# The "palindrome\_check" subroutine should be jumped to by the period  
# check function if a period is encountered. This subroutine should contain  
# a loop that traverses the array from the front towards the back (using the  
# head pointer, \$s1) and from the back towards the front (using the tail  
# pointer, \$s2). If the string is a palindrome then as the array is traversed  
# the characters pointed to should be equal. If the characters are not equal  
# then the string is not a palindrome and the print function should be used  
# to print "No". If the pointers cross (i.e. the head pointer's address is  
# greater than or equal to the tail pointer's address) and the compared  
# characters are equal then the string is a palindrome and "Yes" should be  
# printed.  
#  
# Remember to restore the head and tail pointers to the first element  
# of the array before the subroutine jumps back to main to begin processing the  
# next string. Also, keep in mind that because the tail pointer is updated at  
# the end of "array\_push" it technically points one element past the last  
# character in the array. You will need to compensate for this by either  
# decrementing the pointer once at the start of the array or using an offset  
# from this pointer's address.

```
palindrome_check:
    lw $t1, 0($s4)
    nop
    addiu $s3, $s3, -4
    nop
    lw $t2, 0($s3)
    nop
    beq $s3, $s4, yes
    nop
```

```
bne $t1, $t2, no  
nop  
addiu $s4, $s4, 4  
nop  
beq $s3, $s4, yes  
nop  
j main  
nop
```