

Computer Architecture Lab – 147798

Lecturer: Hoàng Văn Hiệp

Mid-term 2023.2 Report

Nguyễn Khánh Nam - 20225749

Bài 22:

(22) Cyclone Word (challenge)

Cyclone words are English words that have a sequence of characters in alphabetical order when following a cyclic pattern.

Example:



Write a function to determine whether a word passed into a function is a cyclone word. You can assume that the word is made of only alphabetic characters, and is separated by whitespace.

```
is_cyclone_phrase("adjourned") # => True
```

```
is_cyclone_phrase("settled") # => False
```

- Cách thực hiện:
 - + Đọc từng character được nhập vào.
 - + Sử dụng 2 con trỏ địa chỉ trỏ vào đầu và cuối chuỗi character được nhập.
 - + Thực hiện dịch chuyển địa chỉ 2 con trỏ đầu - cuối chuỗi character lần lượt so sánh từng character trong chuỗi theo hình xoắn ốc từ ngoài vào chữ cái ở giữa chuỗi cùng chiều kim đồng hồ.
 - + So sánh thứ tự chữ cái tăng dần theo bảng ASCII trả kết quả TRUE, ngược lại thì trả kết quả FALSE.
- Thanh ghi:
 - + \$sp: chứa địa chỉ đỉnh Stack lưu từng character từ word được nhập vào. Giảm 1 byte mỗi lần lưu 1 giá trị word (a-z và A-Z). Chương trình mặc định đọc và chuyển hết về Lower-case để kiểm tra Cyclone Word.
 - + \$s0: chứa địa chỉ character tận cùng bên phải ở lần so sánh 1 và tiếp theo sau lần so sánh thứ LẺ sẽ chứa giá trị địa chỉ con trỏ \$s1 dịch xuống 1 byte.
 - + \$s1: chứa địa chỉ character tận cùng bên trái ở lần so sánh 1 và tiếp theo sau lần so sánh thứ CHẴN sẽ chứa giá trị địa chỉ con trỏ \$s0 dịch lên 1 byte.

+ \$v0: chứa giá trị từng character được nhập vào. Dùng để kiểm tra điều kiện nhập vào.

+ \$t1: đếm số character trong word.

+ \$s2 = \$t1 - 1: đếm số lần so sánh kiểm tra Cyclone Word.

- Chương trình con:

+ check_upper và check_lower: kiểm tra chuỗi character nhập vào nằm trong khoảng (A-Z) và (a-z),

+ check_cyclone_word: chương trình con chính của chương trình. Thay đổi giá trị địa chỉ được trỏ bởi \$s0 và \$s1 và kiểm tra Cyclone Word.

- Source code:

```
.data
    Message_error_input: .asciiz "\nError input\nWord only!"
    Message_false: .asciiz "\nFALSE"
    Message_true: .asciiz "\nTRUE"
    space: .asciiz " "

.text
    addi $s1, $sp, -1
    addi $s0, $sp, -1
    li $t1, 0          #count_characters cnt = 0 $t1
input_loop:           # Each char

    addi $sp, $sp, -1
    li $v0, 12
    syscall
    beq $v0, 0x20, stop_input
    nop
    beq $v0, 0x0a, stop_input
    nop

check_upper:
    blt $v0, 0x41, end_error    #Check Upper A-Z
    nop
    bgt $v0, 0x5A, check_lower
    nop
    addi $v0, $v0, 32           #Chuyển về lower-case
    j skip_check_lower
    nop

check_lower:
    blt $v0, 0x61, end_error    #Check lower a-z
```

```

        nop
        bgt $v0, 0x7A, end_error
        nop

skip_check_lower:
        sb $v0, 0($sp)
        addi $t1, $t1, 1           # cnt++
        addi $s0, $s0, -1         # string[i++]
        j input_loop

stop_input:

        addi $s2, $t1, -1         # $s2 = cnt - 1
                                   # - check for number of comparisons

        addi $s0, $s0, 1          # = last character of the word

check_cyclone_word:
        lbu $t2, 0($s1)           # first half character
        lbu $t3, 0($s0)           # last half character

        slt $t4, $t3, $t2        # compare: $t4 = $t2 <= $t3 ? continue : end
        beq $t4, 1, end_not_cyclone_word
        nop

        addi $s2, $s2, -1         # number of comparisons - 1 after each comparison
        beq $s2, 0, end_check_cyclone_word # number of comparisons = 0 then stop check

        beq $ra, $0, skip        # Check $ra whether returns or not->skip
        nop
        jr $ra

        skip:

# Sau so sánh lần thứ 1, so sánh tiếp những lần tiếp theo

# Sau lần so sánh thứ lẻ,
# địa chỉ s1 đang trỏ ở nửa TRÁI word cần check
# địa chỉ s0 đang trỏ ở nửa PHẢI word cần check

        addi $t5, $s0, 0          # temp = s0(Odd)
        addi $s0, $s1, -1        # s0(Even) = s1(Odd) + 1
        addi $s1, $t5, 0         # s1(Even) = s0(Odd)

# phải giữ nguyên địa chỉ s0 của chữ cái STT nhỏ hơn
# để so sánh lần thứ 2 sau lần so sánh thứ 1
# nên cần jal so sánh luôn lần thứ chẵn (từ lần thứ 2 trở đi)

        jal check_cyclone_word
        nop

```

```

        add $ra, $0, $0                # Reset $ra after each Even - comparison

# Sau lần so sánh thứ CHẴN,
# địa chỉ s1 đang trỏ ở nửa PHẢI word cần check
# địa chỉ s0 đang trỏ ở nửa TRÁI word cần check

        add $t5, $s0, 0                # temp = s0(Even)
        addi $s0, $s1, 1               # s0(Odd) = s1(Even) - 1
        addi $s1, $t5, 0               # s1(Odd) = s0(Even)

        j check_cyclone_word
        nop

end_check_cyclone_word:
        li $v0, 4
        la $a0, Message_true

        syscall
        j end
        nop

end_not_cyclone_word:
        li $v0, 4
        la $a0, Message_false

        syscall
        j end
        nop

end_error:
        li $v0, 4
        la $a0, Message_error_input

        syscall
        j end
        nop

end:

```

- Kết quả:
- + 'adjourned':

Data Segment									
Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)	Value (+14)	Value (+18)	Value (+1c)	
0x7ffffef0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	d \0 \0 \0	u r n e	a d j o	\0 \0 \0 \0	▲
0x7fffff00	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	
0x7fffff20	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	
0x7fffff40	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	
0x7fffff60	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	
0x7fffff80	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	
0x7fffffa0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	
0x7fffffc0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	
0x7fffffe0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	
0x7fffff100	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	▼

current \$sp
☒ Hexadecimal Addresses
☒ Hexadecimal Values
☒ ASCII

Mars MessagesRun I/O

Clear

adjourned
TRUE
-- program is finished running (dropped off bottom) --

+ 'settled':

Data Segment									
Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)	Value (+14)	Value (+18)	Value (+1c)	
0x7ffffef0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	l e d \0	s e t t	\0 \0 \0 \0	▲
0x7fffff00	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	
0x7fffff20	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	
0x7fffff40	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	
0x7fffff60	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	
0x7fffff80	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	
0x7fffffa0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	
0x7fffffc0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	
0x7fffffe0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	
0x7fffff100	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	▼

current \$sp
☒ Hexadecimal Addresses
☒ Hexadecimal Values
☒ ASCII

Mars MessagesRun I/O

Clear

-- program is finished running (dropped off bottom) --

settled
FALSE
-- program is finished running (dropped off bottom) --

Bài 23:

23) Write a program to compute $C(k, n)$, The number of k -combinations from a given set S of n elements.

- Cách thực hiện:
 - + Sử dụng thuật toán đệ quy theo công thức:
$$C(k, n) = C(k - 1, n - 1) + C(k, n - 1)$$
 - + Nhập vào n và k . Sử dụng Stack để lưu giá trị n , k , kết quả và địa chỉ jump return của từng bước đệ quy
 - + Dịch chuyển thanh ghi con trỏ $\$sp$ từng 4 byte để lưu và xuất giá trị n , k , kết quả và địa chỉ return từng bước đệ quy.
 - + Chương trình kiểm tra những trường hợp đặc biệt của tính tổ hợp và đưa ra kết quả ngay lập tức. Nếu không sẽ thực hiện gọi đệ quy để tính toán.
- Thanh ghi:
 - + $\$s0$: chứa n
 - + $\$s1$: chứa k
 - + $\$sp$: chứa địa chỉ đỉnh stack lưu n , k , kết quả $C(k - 1, n - 1)$ và địa chỉ $\$ra$. Được reset lại sau mỗi lần tính được đệ quy $C(k-1, n-1) + C(k,n-1)$.
- Chương trình con:
 - + Base_CKN: dùng để đệ quy chứa trường hợp nền của công thức tính tổ hợp đệ quy trên
$$C(k, n) = \begin{cases} 1 & \text{if } k = 0, k = n \\ n & \text{if } k = 1 \end{cases}$$
 - + findCKN: Thực hiện từng bước đệ quy thay đổi giá trị k và n , gọi lại chương trình con Base_CKN để kiểm tra trường hợp nền. Trả kết quả tính tổ hợp cuối cùng.
- Source code:

```
.data
input_msg1: .asciiz "Input n: "
input_msg2: .asciiz "Input k: "

output_res: .asciiz "C(k,n) = "

error_msg1: .asciiz "wrong input"
```

```

    result: .word 0
.text
#-----INPUT-----
    li $v0, 4
    la $a0, input_msg1
    syscall

    li $v0, 5
    syscall

    addi $s0, $v0, 0 #s0 = n

    li $v0, 4
    la $a0, input_msg2
    syscall

    li $v0, 5
    syscall

    addi $s1, $v0, 0 #s1 = k

#-----CHECKING-----
    blt $s0, $0, error_input1    # n < 0 -> error
    nop
    blt $s1, $0, error_input1    # k < 0 -> error
    nop
    beq $s0, $0, special_case    # n = 0 -> special case
    nop

    bne $s1, 1, skip1            # If k = 1 AND n > 0 -> C(k,n) = n
    nop
    addi $s3, $s0, 0
    j EXIT_result
skip1:

special_case:
    bne $s0, $s1, skip2        # If k = n AND n >= 0 -> C(k,n) = 1
    nop
    addi $s3, $0, 1
    j EXIT_result
skip2:

    bne $s1, 0, skip3          # k = 0 AND n > 0 -> C(k,n) = 1
    nop
    addi $s3, $0, 1
    j EXIT_result
skip3:

    slt $t0, $s0, $s1          #Check n < k -> ERROR
    beq $t0, 1, error_input1

```



```

    nop

#-----Recursive Calculation of C(k,n)-----

#print result
    jal findCKN
    move $s3, $v0
    sw $s3, result

    #display result
    li $v0, 4
    la $a0, output_res
    syscall

    li $v0, 1
    lw $a0, result
    syscall

    #end program
    li $v0, 10
    syscall

    # v0 = result of the function
    #
Base_CKN:
    seq $t0, $s1, $0
    seq $t1, $s0, $s1
    add $t2, $t0, $t1
    blt $t2, 1, continue1    # If (k == 0 || k == n)
    nop
    addi $v0, $0, 1          # return 1
    jr $ra
    continue1:
    bne $s1, 1, continue2    # If (k == 1)
    nop
    addi $v0, $s0, 0          # return n
    jr $ra
    continue2:

    j findCKN                # return C(k - 1, n - 1) + C(k,n-1)

findCKN:
    subu $sp, $sp, 16
    sw $ra 0($sp)

    sw $s1, 4($sp)           #save k
    sw $s0, 8($sp)           #save n
    addi $s1, $s1, -1         # k - 1
    addi $s0, $s0, -1         # n - 1
    jal Base_CKN              # -> C(k - 1, n - 1)

```

```

    lw $s1, 4($sp)      #restore k
    lw $s0, 8($sp)      #restore n
    sw $v0, 12($sp)     #save C(k - 1 , n - 1)

    addi $s0, $s0, -1    # n - 1

    jal Base_CKN         # -> C(k , n - 1)

    lw $t7, 12($sp)     #restore C(k-1,n-1)

    add $v0, $t7, $v0    #return C(k-1,n-1) + C(k,n-1)

    lw $ra, 0($sp)      #restore $ra
    add $sp, $sp, 16     #

    jr $ra

#-----print out-----
error_input1:
    li $v0, 4
    la $a0, error_msg1
    syscall

    j EXIT

EXIT_result:
    li $v0, 4
    la $a0, output_res
    syscall

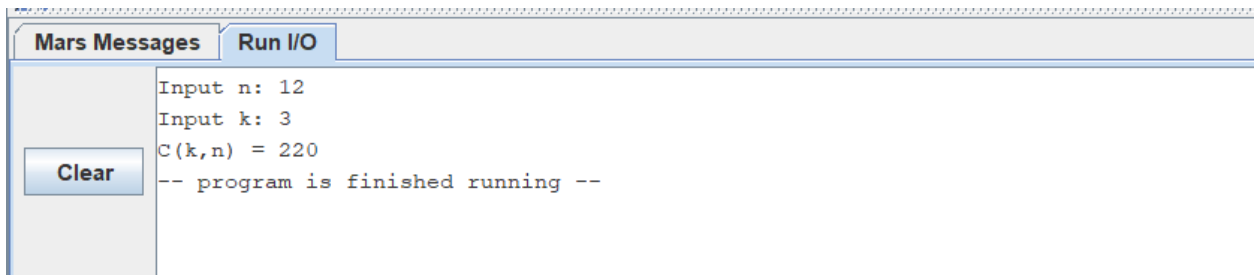
    li $v0, 1
    addi $a0, $s3, 0
    syscall

EXIT:
    li $v0, 10
    syscall

```

- Kết quả:

+ C(12, 3):

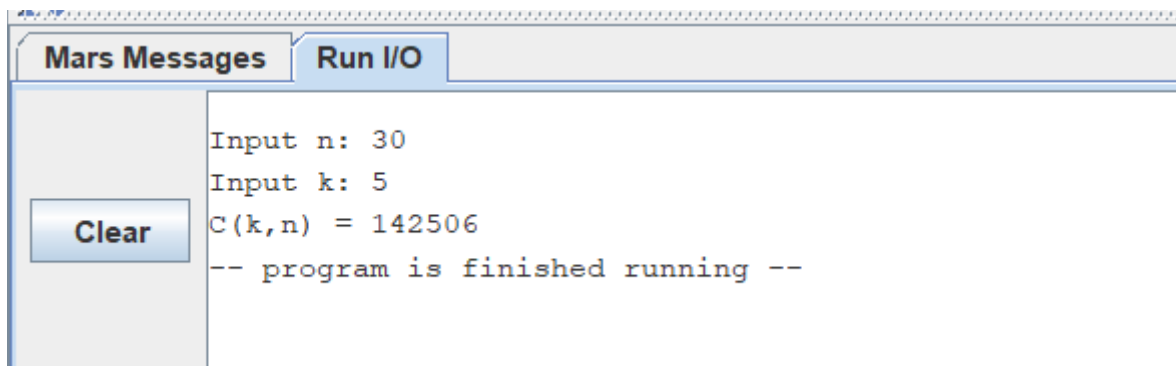


The screenshot shows a window titled "Mars Messages" with a "Run I/O" button. The output text is as follows:

```
Input n: 12
Input k: 3
C(k,n) = 220
-- program is finished running --
```

A "Clear" button is located on the left side of the window.

+ C(30, 5):



The screenshot shows a window titled "Mars Messages" with a "Run I/O" button. The output text is as follows:

```
Input n: 30
Input k: 5
C(k,n) = 142506
-- program is finished running --
```

A "Clear" button is located on the left side of the window.