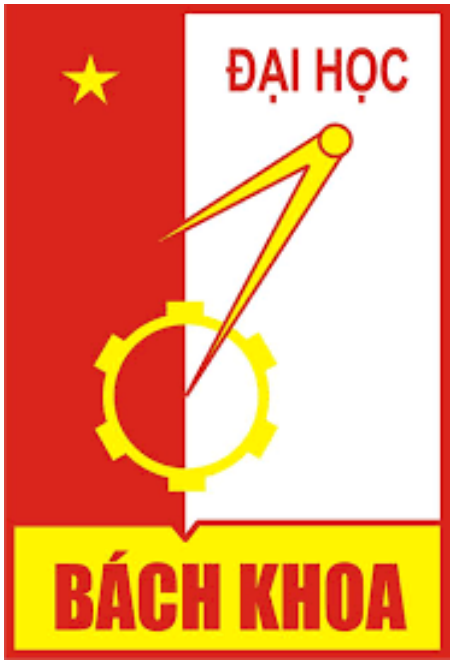
Trường Đại Học Bách Khoa Hà Nội

Viện Công nghệ thông tin và truyền thông

Báo cáo đề tài giữa kỳ KTMT



Giảng viên hướng dẫn: Lê Bá Vui

Sinh viên thực hiện:

Lê Nhật Huy	**20176784**
Đỗ Quang Nam	**20176828**

Hà Nội, tháng 5 năm 2020

1) Project 10 _ Lê Nhật Huy

a) Yêu cầu bài toán:

- Yêu cầu người dùng nhập vào một số nguyên i.
- Tính power(2, i), square(i), hexadecimal(i) và in ra màn hình theo định dạng:

i	power(2,i)	square(i)	Hexadecimal(i)
10	1024	100	0xA
7	128	49	0x7
16	65536	256	0x10

b) Thuật toán sử dụng:

- Viết chương trình nhận một số nguyên từ người dùng và 3 chương trình con lần lượt tính power(2, i), square(i) và hexadecimal(i).

c) Mã nguồn:

```
# Mid Term Project 10

.data

      message:    .asciiz     "Enter an integer"

      table:      .asciiz     "i    Power(2, i) Square(i)
      Hexadecimal(i)\n"

      space1:     .asciiz     "     "

      space2:     .asciiz     "           "

      newLine:    .asciiz     "\n"

      hex:        .space      10


.text

      li          $v0, 4                # print table header

      la          $a0, table

      syscall

start:

      li          $v0, 51

      la          $a0, message

      syscall
```

```
	beq	$a1, -1, start	# $a1 status value (if input is
				# invalid =>Re-input)
	beq	$a1, -3, start	# $a1 status value (if input is
				# invalid => Re-input)
	beq	$a1, -2, exit	# exit if user click cancel
	blt	$a0, -31, start	# check if i < -31
	bgt	$a0, 30, start	# check if i > 30
	add	$s0, $zero, $a0	# $s0 = i

	jal	power		# call function power(2, i)
	add	$s1, $zero, $v0

	jal	square		# call function square(i)
	add	$s2, $zero, $v0

	jal	hexadecimal	# call function hexadecimal(i)

printResult:
	li	$v0, 1		# print i
	add	$a0, $zero, $s0
	syscall
	li	$v0, 4
	la	$a0, space1
	syscall
checkPowerResult:
	bge	$s0, $zero, printInt
printFloat:
	li	$v0, 2		# print float power(2, i)
	syscall
	li	$v0, 4
	la	$a0, space2
	syscall
	j	printSquare
```

```
printInt:
      li          $v0, 1            # print integer power(2, i)
      add         $a0, $zero, $s1
      syscall
      li          $v0, 4
      la          $a0, space2
      syscall

printSquare:
      li          $v0, 1            # print square(i)
      add         $a0, $zero, $s2
      syscall
      li          $v0, 4
      la          $a0, space2
      syscall

printHex:
      li          $v0, 4            # print hexadecimal(i)
      la          $a0, hex
      syscall
      li          $v0, 4
      la          $a0, newLine
      syscall
      jal         clearHex

      j           start

exit:
      li          $v0, 10
      syscall
```

```
#-----------------------------------------------------------------
# @function power
# @param[in]      $s0         User entered integer i
# @return         $v0         i >= 0 power(2, i)
# @return         $f12        i < 0 power(2, i)
#-----------------------------------------------------------------
power:
      li          $v0, 1
      li          $a0, 0                  # $a0 = 0
      add         $a1, $zero, $s0         # $a1 = $s0
      bge         $s0, $zero, powerLoop   # $s0 >= 0
      sub         $a1, $zero, $s0         # $a1 = -$s0

powerLoop:
      beq         $a0, $a1, powerDone     # $a0 == i ? done : loop
      sll         $v0, $v0, 1             # $v0 = $v0 * 2
      addi        $a0, $a0, 1             # $a0 += 1
      j           powerLoop

powerDone:
      bge         $s0, $zero, done        # $s0 >= 0
      li          $t0, 1
      mtc1        $t0, $f0                # $f0 = 1.0
      cvt.s.w     $f0, $f0

      mtc1        $v0, $f2                # $f2 = (float)$v0
      cvt.s.w     $f2, $f2
      div.s $f12, $f0, $f2                # $f12 = $f0/$f2
done:
      jr          $ra
```

```
#-----------------------------------------------------------------
# @function square
# @param[in]      $s0         User entered integer i
# @return         $v0         square(i)
#-----------------------------------------------------------------
square:
      mul         $v0, $s0, $s0           # $v0 = i * i
      jr          $ra
#-----------------------------------------------------------------
# @function hexadecimal
# @param[in]      $s0         User entered integer i
# @return         none
#-----------------------------------------------------------------
hexadecimal:
      la          $a0, hex                # load hex to $a0
      add         $a1, $zero, $s0         # $a1 = i
      li          $t1, 48                 # add 0x to hex string
      sb          $t1, 0($a0)
      addi        $a0, $a0, 1
      li          $t1, 120
      sb          $t1, 0($a0)
      addi        $a0, $a0, 1
      beqz        $s0, hexZero            # $s0 = 0 => hex = "0x0"
      li          $t0, 8                  # counter loop through 32 bits
      li          $t2, 0                  # flag $t2 ignore 0
hexLoop:
      beqz        $t0, hexDone            # counter == 0 => done
      andi        $t1, $a1, 0xf0000000    # get most left 4 bits
      srl         $t1, $t1, 28            # move 4 bits to most right
      beq         $t1, $t2, continue      # $t1 == $t2 (= 0) => ignore 0
      ble         $t1, 9, less            # $t1 <= 9 => ASCII Code
      addi        $t1, $t1, 55            # [A-F]
```

```
      j           writeHex
less:
      addi        $t1, $t1, 48            # [1-9]
writeHex:
      addi        $t2, $t2, -1            # remove flag ignore 0
      sb          $t1, 0($a0)             # write ASCII Code to hex string
      addi        $a0, $a0, 1
continue:
      sll         $a1, $a1, 4             # shift left to get next 4 bits
      addi        $t0, $t0, -1            # counter -= 1
      j           hexLoop
hexZero:
      li          $t1, 48                 # add 0x to hex string
      sb          $t1, 0($a0)
hexDone:
      jr          $ra
#-----------------------------------------------------------------
# @function clearHex
# @param[in]            none
# @return               none
#-----------------------------------------------------------------
clearHex:
      la          $a0, hex                # load hex to $a0
      li          $a1, 0
clearLoop:
      beq         $a1, 10, doneClear
      sb          $zero, 0($a0)
      addi        $a0, $a0, 1
      addi        $a1, $a1, 1
      j           clearLoop
doneClear:
      jr          $ra
```

d) Giải thích:

i. Hàm main:

- Các thanh ghi sử dụng:

Thanh ghi	Mục đích
$s0	Lưu giá trị i người dùng nhập vào
$s1	Lưu giá trị power(2, i) nếu i >= 0
$f12	Lưu giá trị power(2, i) nếu i < 0
$s2	Lưu giá trị square(i)
$a1	Lưu trạng thái giá trị nhập của người dùng

- Giải thích:
  - Nhận số nguyên người dùng nhập vào và kiểm tra:
    - Nếu hợp lệ => lưu giá trị i vào thanh ghi $s0.
    - Nếu không hợp lệ (không phải số nguyên, số quá lớn, số âm) => yêu cầu người dùng nhập lại.
  - Gọi các chương trình con và lưu kết quả trả về vào các thanh ghi $s1, $s2, $f12 và biến hex.
  - In kết quả ra màn hình
  - Thoát chương trình

ii. Hàm power:

- Các thanh ghi sử dụng:

Thanh ghi	Mục đích
$a0	Lưu giá trị counter
$a1	Lưu giá trị |i|
$v0	Lưu giá trị power(2, i) nếu i >= 0
$f12	Lưu giá trị power(2, i) nếu i < 0

- Giải thích:
  - Khởi tạo $v0 = 1, $a0 = 0, $a1 = |i|.
  - Thực hiện i vòng lặp, mỗi vòng lặp:
    - $v0 = $v0 * 2
    - $a0 += 1
  - Khi $a0 == i, kết thúc vòng lặp:
    - Nếu i >= 0, trả về giá trị $v0 = square(2, i).
    - Nếu i < 0, trả về giá trị $f12 = 1/$v0.

iii. Hàm square:

- Các thanh ghi sử dụng:

Thanh ghi	Mục đích
$v0	Lưu giá trị square(i)
$s0	Lưu giá trị i

- Giải thích:
  - Thực hiện phép nhân $v0 = $s0 * $s0
  - Trả về giá trị $v0 = square(i)

iv. Hàm hexadecimal:

- Các thanh ghi sử dụng:

Thanh ghi	Mục đích
$a0	Lưu địa chỉ biến hex
$a1	Lưu giá trị của i
$t0	Lưu giá trị counter
$t1	Thanh ghi tạm, lưu giá trị của từng byte
$t2	Cờ để xác định có in số 0 hay không

- Giải thích:
  - Biến hex kiểu .space dùng để lưu xâu biểu diễn số hexa chuyển đổi từ i.
  - Nạp địa chỉ của biến hex vào $a0.
  - Ghi xâu "0x" vào biến hex. Nếu i = 0, trả về giá trị hex: "0x0"
  - Khởi tạo biến counter $t0 = 8, biến $t2 = 0 làm cờ:
    - Khi $t2 = 0 => không ghi "0" vào xâu.
    - Khi $t2 = 1 => ghi "0" vào xâu.
  - Thực hiện vòng lặp:
    - Lấy 4 bits ngoài cùng trái của i bằng phép AND lưu vào thanh ghi $t1.
    - Kiểm tra giá trị của $t1:
      - Nếu $t1 <= 9 => $t1 += 48 để ra được mã ASCII của [0-9].
      - Nếu $t1 > 9 => $t1 += 55 để ra được mã ASCII của [A-F].
    - Ghi kí tự vừa lấy được vào xâu hex.

- Dịch trái thanh ghi $a1 4 bits để lấy 4 bits trái ngoài cùng tiếp theo của i.

- Vòng lặp kết thúc khi lặp đủ 8 lần.
- Biến hex đã chứa xâu biểu diễn số hexa chuyển đổi từ i.

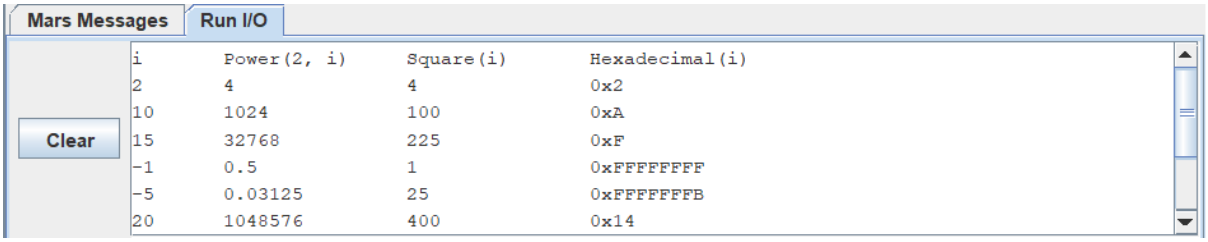
v. Hàm clearHex:

- Các thanh ghi sử dụng:

Thanh ghi	Mục đích
$a0	Lưu địa chỉ biến hex
$a1	Lưu giá trị của counter

- Giải thích:
  - Lặp qua từng ký tự của xâu hex để tạo ra xâu rỗng.

e) Kết quả:



f) Nhận xét:

- Chương trình chỉ biểu diện được các số trong khoảng 2^31 -1, nếu người dùng nhập số i > 30 hoặc i < -31 thì hàm power(2, i) sẽ bị tràn số.
- Đã khắc phục bằng cách giới hạn lại khoảng số người dùng có thể nhập (-31 <= i <= 30).

2) Project 8 _ Đỗ Quang Nam

a) Yêu cầu bài toán:

- Nhập vào số lượng sinh viên trong một class
- Nhập vào tên và điểm của từng sinh viên
- Sắp xếp sinh viên theo điểm và in ra (sắp xếp giảm dần)

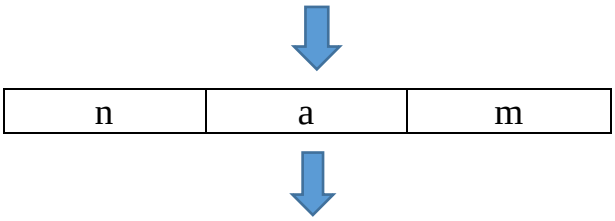
---

b) Thuật toán sử dụng:

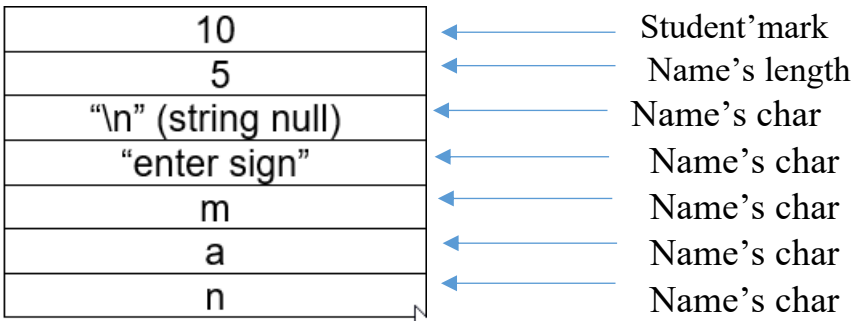
Chương trình được chia ra làm 3 phần chính:

- Đọc số lượng sinh viên
- Đọc điểm và tên sinh viên:
  Input: Name: namdo – Mark: 10

Step 1: Split name to chars



Step 2: Push name's char, name's length and mark to stack



Giải thích: Tên sinh viên sẽ được chia thành nhiều kí tự char và push lần lượt vào stack. Tiếp sau đó là lưu lại độ dài của tên (phục vụ cho việc pop các kí tự ra) và cuối cùng là điểm của sinh viên. Vì vậy thông tin của một sinh viên sẽ được lưu trong stack như hình trên.

- In thông tin sinh viên giảm dần theo điểm:

- Tìm ra sinh viên có điểm số cao nhất trong stack và lưu vào biến "max"
- Sử dụng biến "maxSoFar" để lưu giá trị max trước đó để giá trị của "max" qua mỗi lần duyệt sẽ giảm dần.
- Duyệt lại stack tìm những sinh viên có điểm bằng "max" và in ra màn hình.

→ Tiếp tục quá trình trên cho đến khi tất cả sinh viên được in ra.

- Xử lí tên sinh viên sau khi lấy từ stack bị đảo kí tự:

- B1: Lấy tất cả kí tự từ stack và lưu trữ lại vào một vùng nhớ (có thể coi như 1 array) (trong chương trình có tên là nameStr[])
- B2: Bắt đầu từ cuối array in từng kí tự của tên ra.
- Minh họa:

Ví dụ tên sinh viên: “namdo” sau khi lấy ra từ stack.

nameStr[]:

o	d	m	a	n

In ngược từ cuối lên ta được kết quả in ra màn hình: “namdo”.

c) Mã nguồn:

```
.data
nameStr: .space 100

inputAmountMes: .asciiz "Input number of students: "
inputNameMes:   .asciiz "Input student's name: "
inputMarkMes:   .asciiz "Input student's mark: "
Message:        .asciiz "Student name:"
errorInputMes:  .asciiz "Mark need to be between 0-10.Please try again!"

#
#     s0 = number of students
#
.text
# **********
# PART 1: READ NUMBER OF STUDENTS
# **********
readAmount:
            li    $v0, 51
            la    $a0, inputAmountMes
            syscall
            la    $s0, 0($a0)             # s0 = number of students

#**********
# PART 2: READ STUDENT’S NAME AND MARK
```

```
#**********
readNameMark:
            li    $s1, 0                  # intialize i = 0 for loop
loop:
            addi  $s1, $s1, 1             # i = i + 1
readName:
            li    $v0, 54
            la    $a0, inputNameMes
            la    $a1, nameStr
            la    $a2, 100
            syscall

            la    $a0, nameStr            # a0 = nameStr[0]
            addi  $t0, $zero, 0           # length = 0 (nameStr[i])
pushNameToStack:
            # read each char of nameStr
            add   $t1, $a0, $t0           # t1 = address(nameStr[0] + i)
            lb    $t2, 0($t1)             # t2 = value(t1) = char

            # push each char to stack
            addi  $sp, $sp, -4
            sw    $t2, 0($sp)
            beq   $t2, $zero, end_of_str  # if 'end of string'
            addi  $t0, $t0, 1             # else length++
            j     pushNameToStack         # and continue to push
end_of_str:
            # After student's name is pushed to stack,
            # push legnth of name to stack
            addi  $sp, $sp, -4
            addi  $t0, $t0, 1
            sw    $t0, 0($sp)
end_of_readName:

readMark:
            li    $v0, 51
            la    $a0, inputMarkMes
            syscall
            blt   $a0, 0, errorInput
            bgt   $a0, 10, errorInput
            j     pushMarkToStack
errorInput:
            li    $v0, 55
            la    $a0, errorInputMes
            syscall
            j     readMark
pushMarkToStack:
            addi  $sp, $sp, -4
            sw    $a0, 0($sp)
end_of_readMark:
```

```
            beq   $s1, $s0, end_of_loop   # if all students is pushed
            j     loop                    # else continue
end_of_loop:
end_of_readNameMark:

# Save current address of stack pointer to loop through stack many times
            add   $fp, $sp, -4

#*********************************************
# PART 3: LOOP THROUGHT STACK TO PRINT STUDENT
#     t1 = maxSoFar
#     t0 = max
#     s3 = number of student have printed
#     t6 = student's mark popped from stack
#     t2 = length of student's name popped from stack
#*********************************************
            li    $t1, 100          # set maxSoFar = 100
            li    $s3, 0            # number of student have printed
loopStack:
            add   $sp, $fp, $zero   # set stack pointer to top of stack

# 3.1: Find max student's mark in stack (max < maxSoFar)
findMax:
            li    $t0, 0            # set max = 0
            li    $t7, 1            # set i for findMax = 1
popMark:
            addi  $sp, $sp, 4
            lw    $t6, 0($sp)                   # get student Mark = t6
            beq   $t6, $t1, popNameLength       # if student's mark >=
maxSoFar -> go through
            bgt   $t6, $t1, popNameLength
            blt   $t6, $t0, popNameLength       # if student's mark < current
max -> go through

            add   $t0, $t6, $zero               # else set new max
popNameLength:
            addi  $sp, $sp, 4
            lw    $t2, 0($sp)             # nameLength = t2
popName:
            # loop through stack to get name
            li    $s1, 0                        # intialize i = 0
popLoop:
            addi  $s1, $s1, 1                   # i = i + 1
            addi  $sp, $sp, 4
            bne   $s1, $t2, popLoop       # if i != nameLength
end_of_popName:

            # check condition to continue to loop
            beq   $t7, $s0, end_of_findMax      # if all students were looped
-> exit findMax
```

```
            addi  $t7, $t7, 1             # else i = i + 1
            j     popMark                 # and continue to loop
end_of_findMax:

# Set maxSoFar = max
            add   $t1, $t0, 0

# 3.2: Print student's inf who have current max mark
            add   $sp, $fp, $zero         # point $sp to top of stack to begin
loopForPrint:
            li    $s2, 1                  # set i for loopStack  = 1
popMark2:
            addi  $sp, $sp, 4
            lw    $t6, 0($sp)             # get student Mark = t6

            addi  $sp, $sp, 4
            lw    $t5, 0($sp)             # get student's name length = t5

            beq   $t6, $t0, printName     # if student's mark = current mark
                                            => print student's name
            j     goThrough               # else go through that student
checkCondition:
            beq   $s2, $s0, end_of_loopForPrint # if stack was loop through ->
exit and check to continue to print another student
            addi  $s2, $s2, 1             # else i = i + 1 and
            j     popMark2                # continue to loop through stack
end_of_loopForPrint:

# Check condition to decide if there's need to loop throught stack one more
time
            beq   $s3, $s0, end_of_loopStack    # if all student's name have
printed -> exit program
            j     loopStack               # else loop through stack again
end_of_loopStack:


# @printName function
printName:
            addi  $s3, $s3, 1             # whenever a student have printed,
s3 = s3 + 1 (s3 = number of student have printed)
            li    $s1, 0                  # number of chars have been read

            la    $a0, nameStr            # a0 = address of .space to store
chars popped from stack
            la    $a1, 0($a0)             # a1 = address of current element

# Get all name's chars from stack, and push them to a .space named 'nameStr'
loopForStore:
            # pop char from stack and store in $a1 (a1 = current element
in .space)
```

```
            addi  $sp, $sp, 4
            lw    $t8, 0($sp)
            sb    $t8, 0($a1)
            addi  $a1, $a1, 4             # move to next element in .space
            addi  $s1, $s1, 1                   # i = i + 1 ( s1 = number of
chars have been read)
            bne   $s1, $t5, loopForStore        # if i != nameLength ->
continue loop


# Loop through .space to print student's name
            li    $s1, 0                        # counter of char
printOut:
            addi  $a1, $a1, -4
            lb    $t3, 0($a1)
            li    $v0, 11
            la    $a0, 0($t3)
            syscall
            addi  $s1, $s1, 1
            beq   $s1, $t5, checkCondition
            j     printOut
end_of_printName:

# @goThrough function
goThrough:
            li    $s1, 0
loopThrough:
            addi  $sp, $sp, 4
            addi  $s1, $s1, 1                   # i = i + 1
            bne   $s1, $t5, loopThrough         # if i != nameLength
            j     checkCondition
```

d) Giải thích:

- readAmount:

Thanh ghi	Ý nghĩa
$s0	Lưu số lượng học sinh

- readNameMark:

Thanh ghi	Ý nghĩa
$s1	Biến đếm sử dụng khi lặp
$a0	Địa chỉ nameStr, = nameStr[0]
$t1	= address(nameStr[i])
$t2	= value(nameStr[i]) = char
$t0	= name’s length

Giải thích:

- Thực hiện vòng lặp với biến đếm i ($s1) đọc tên từng sinh viên. Với mỗi tên :
  - Sử dụng "nameStr" để lưu trữ string "name" được nhập.
  - Lưu từng kí tự của string đó vào stack.
  - Sử dụng $t0 lưu độ dài của string.
- Đọc điểm sinh viên và push vào stack
- Khi i = n ($s1 = $s0) → tất cả sinh viên được nhập → kết thúc readNameMark.

- loopStack:

Thanh ghi	Ý nghĩa
$s3	Biến đếm, lưu số lượng sinh viên đã được printed.

Giải thích:

- duyệt stack và làm 2 công việc: tìm giá trị max của điểm trong stack (@findMax) và in ra những sinh viên có điểm = max (@printName). Chi tiết 2 hàm trên xem ở dưới.
- Kết thúc khi $s3 = $s0 || tất cả sinh viên đã được in.

- findMax: tìm số điểm lớn nhất chưa được in ra trong stack.

Thanh ghi	Ý nghĩa
$t0	Lưu giá trị max
$t1	Lưu giá trị maxSoFar
$t6	Điểm được lấy ra từ stack

Giải thích:

- Duyệt qua stack, nếu điểm được lấy ra khỏi stack ($s6) lớn hơn max hiện tại ($t0), gán lại giá trị cho max.
- Max ($t0) cần luôn nhỏ hơn maxSoFar ($t1).

- printName: Lấy tên sinh viên từ stack và lưu tại nameStr[]

Thanh ghi	Ý nghĩa

$s1	Số lượng kí tự char của tên đã lấy từ stack
$a0	=nameStr. Vùng nhớ chứa những kí tự char lấy từ stack
$a1	= nameStr[i]. Địa chỉ của kí tự hiện tại trong nameStr.

Giải thích:

- Thực hiện vòng lặp i ($s1), lấy từng kí tự char của tên ra khỏi stack và lưu vào nameStr[] ($a0).
- Vòng lặp kết thúc khi i = name's length.

- printOut: In tên sinh viên từ nameStr[] ra màn hình

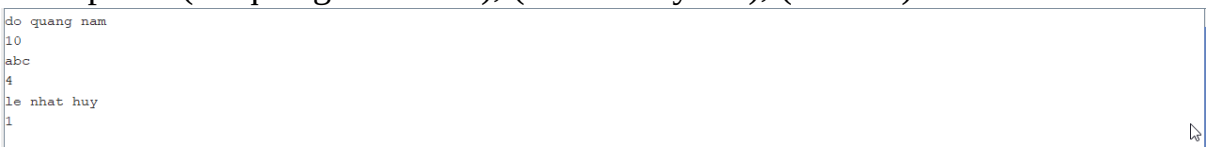
Thanh ghi	Ý nghĩa
$s1	Biến đếm
$a0	= nameStr.
$a1	= nameStr[i]. Địa chỉ của kí tự hiện tại trong nameStr. = nameStr[length - 1] sau khi kết thúc printName.

Giải thích:

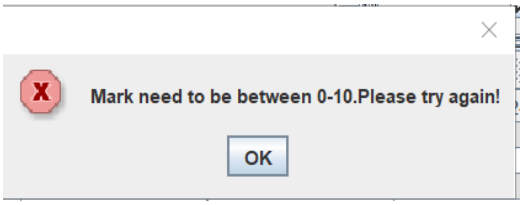
- Thực hiện vòng lặp lấy từng kí tự từ nameStr[] và in ra màn hình. Bắt đầu từ cuối nameStr[] và giảm dần $a1.
- Vòng lặp kết thúc khi i = name's length.

e) Kết quả thực hiện:

Example 1: (do quang nam ~ 10), (le nhat huy ~ 1), (abc ~ 4)



Example 2: (do nam ~ 11) hoặc (do nam ~ -1)



Example 3: (do nam ~ 3), (le huy ~ 3), (duong ~ 10)

```
duong
10
le huy
3
do nam
3
```

f) Nhận xét:

- Thuật toán sử dụng còn phức tạp (VD: Phải tách tên thành các kí tự char push vào stack sau đó lại phải đảo ngược lại kí tự) dẫn đến chương trình trở nên dài, khó maintain và khó đọc.
- Mỗi kí tự char của tên được lưu trong bộ nhớ với 4byte thay vì chỉ 1byte → chiếm dung lượng.