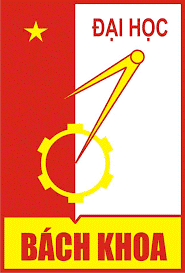
**TRƯỜNG ĐẠI HỌC BÁCH KHOA HÀ NỘI**

**VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG**



Mini-projects

Computer Architecture

**GVHD: Lê Bá Vui**

**Nhóm: 03**

**Thành Viên: Phạm Ngọc Bảo Anh - 20176682**

**Nguyễn Khắc Thắng - 20176869**

**Mã Lớp: 113834**

MỤC LỤC

[I. Project 5: 3](#_Toc40566665)

[1. Phân tích cách thực hiện 3](#_Toc40566666)

[o Phân tích đề bài: 3](#_Toc40566667)

[o Chương trình chính được chia làm 3 phần: 3](#_Toc40566668)

[o Cách chạy chương trình: 3](#_Toc40566669)

[o Cách thực hiện: 3](#_Toc40566670)

[2. Ý nghĩa các thanh ghi 4](#_Toc40566671)

[3. Mã nguồn 4](#_Toc40566672)

[4. Kết quả 7](#_Toc40566673)

[II. Project 7: 8](#_Toc40566674)

[1. Phân tích cách thực hiện 8](#_Toc40566675)

[o Phân tích đề bài 8](#_Toc40566676)

[o Chương trình chính được chia thành các phần như sau: 8](#_Toc40566677)

[o Cách chạy chương trình: 9](#_Toc40566678)

[2. Ý nghĩa của các thanh ghi được sử dụng 9](#_Toc40566679)

[o Trong thủ tục main: 9](#_Toc40566680)

[o Trong thủ tục loop: 9](#_Toc40566681)

[o Trong thủ tục find\_height: 9](#_Toc40566682)

[o Trong thủ tục sort\_height: 9](#_Toc40566683)

[o Trong thủ tục replace: 9](#_Toc40566684)

[o Trong thủ tục output: 10](#_Toc40566685)

[3. Source code 10](#_Toc40566686)

[4. Kết quả: 17](#_Toc40566687)

[a) Sắp xếp mảng đầu vào theo yêu cầu: 17](#_Toc40566688)

[b) Một số ví dụ về xử lý lỗi input: 18](#_Toc40566689)

BÁO CÁO MINI PROJECT MÔN THỰC HÀNH KIẾN TRÚC MÁY TÍNH

## Nhóm 3: Phạm Ngọc Bảo Anh - Nguyễn Khắc Thắng

# **Project 5:**

Write a program to get decimal numbers, display those numbers in binary and hexadecimal.

1. Phân tích cách thực hiện

* Phân tích đề bài:
  + Input là 1 số nguyên hệ thập phân
  + Output là in ra màn hình số hệ nhị phân và hệ thập lục phân
* Chương trình chính được chia làm 3 phần:
  + Chương trình con chuyển đổi hệ thập phân --> hệ nhị phân
  + Chương trình con chuyển đổi hệ thập phân --> hệ thập lục phân
  + Kiểm tra mức độ hợp lệ của dữ liệu nhập từ bàn phím
* Cách chạy chương trình:
  + Nhập 1 số nguyên từ bàn phím, xem kết quả hiển thị ở console
  + Nếu nhập số nguyên là 0 --> Thoát chương trình
  + Nếu nhập 1 chuỗi không hợp lệ --> Nhập lại
* Cách thực hiện:
  + Dựa theo việc khi lưu 1 số nguyên vào thanh ghi thì số nguyên đó được chuyển thành hệ nhị phân (viết gọn lại ở hệ hexa)
  + Convert to binary:
* Trích từng bit trong 32 bit khi lưu 1 số vào thanh ghi Ex: 10 = 0000 0000 0000 0000 0000 0000 0000 1010
* Mã giả:

input = enter from keyboard

count = 32

mask = 1000 0000 0000 0000 0000 0000 0000 0000

while(count != 0)

temp = AND input, mask

if (temp != 0)

temp = 1

print(temp)

srl mask, 1

count--

* + Convert to hexadecimal
* Trích từng 4 bit trong 32 bit khi lưu 1 số. Sau đó chuyển tương ứng theo mã của bảng ASCII Ex: 10 = 0x0000000a
* Mã giả:

input = enter from keyboard

count = 8

mask = 0x0000000f

while(count != 0)

input = rol input, 4 Ex: 0x0000000a -> 0x000000a0

temp = AND input, mask

if (temp <= 9)

temp += 48 (stage number of ASCII table)

else

temp += 55 (stage uppercase alphabet of ASCII table)

storage temp (hex digit) into result

count--

print(result)

1. Ý nghĩa các thanh ghi

* $t0: Lưu giá trị số nguyên nhập từ bàn phím
* $t3: Giá trị của biến đếm
* $t1: Giá trị trích xuất 1 bit (4 bit) sau mỗi lần chạy vòng lặp
* $t2 trong print\_binary: mặt nạ dùng trong phép AND để trích từng bit
* $t2 trong print\_hexa: Lưu trữ địa chỉ của kết quả (result2)
* $s6: Lưu trạng thái của data input

1. Mã nguồn

.data

ask\_str: .asciiz "Enter a number PLEASE: "

message\_error1: .asciiz "Error Input Type! Please enter integer input!"

message\_error2: .asciiz "Error Cancel! Please enter integer input!"

message\_error3: .asciiz "Error no data input! Please enter integer input!"

result\_str1: .asciiz "\nBinary equivalent: "

result\_str2: .asciiz "\nHexadecimal equivalent: "

result2: .space 8

.text

main:

# ask and store the number

li $v0, 51 # InputDialogInt

la $a0, ask\_str

syscall

add $s6, $zero, $a1 # s6 = a1 = status value of InputDialogInt

# if s6 == 0: OK

bnez $s6, input\_error # if status != 0 jump input\_error

add $t0, $a0, $zero # $t0 = input number

beqz $t0, end\_main # if input = 0 then exit program

jal print\_bin

jal print\_hexa

li $v0, 11 # New Line

li $a0, 10

syscall

j main

end\_main:

li $v0, 10 # exit

syscall

# =============== Check input data ================ #

input\_error:

li $v0, 55

li $a1, 2 # warning message

beq $s6, -1, error1

beq $s6, -2, error2

beq $s6, -3, error3

error1:

la $a0, message\_error1

syscall

j end\_error

error2:

la $a0, message\_error2

syscall

j end\_error

error3:

la $a0, message\_error3

syscall

j end\_error

end\_error:

j main

# ================== Pseudo Code ================= #

# # Convert decimal number to binary number

# input = enter from keyboard

# count = 32

# mask = 1000 0000 0000 0000 0000 0000 0000 0000

# while(count != 0)

# temp = AND input, mask

# if (temp != 0)

# temp = 1

# print(temp)

# srl mask, 1

# count--

# =============================================== #

print\_bin:

li $v0, 4

la $a0, result\_str1 # call result\_str1

syscall

add $t1, $zero, $zero # $t1 = 0

addi $t2, $zero, 1 # load 1 as a mask

sll $t2, $t2, 31 # move the mask to appropriate position

addi $t3, $zero, 32 # loop counter

loop1:

and $t1, $t0, $t2 # and the input with the mask

beq $t1, $zero, print1 # Branch to print if $t1 = 0

addi $t1, $zero, 1 # else $t1 != 0 -> assign $t1 = 1

print1:

li $v0, 1

move $a0, $t1

syscall # print result

srl $t2, $t2, 1

addi $t3, $t3, -1

bne $t3, $zero, loop1

jr $ra # return to main

# ================== Pseudo Code ================= #

# # Convert decimal number to hexadecimal number

# input = enter from keyboard

# count = 8

# mask = 0x0000000f

# while(count != 0)

# input = rol input, 4 Ex: 0x0000000a -> 0x000000a0

# temp = AND input, mask

# if (temp <= 9)

# temp += 48 (stage number of ASCII table)

# else

# temp += 55 (stage uppercase alphabet of ASCII table)

# storage temp (hex digit) into result

# count--

# print(result)

# =============================================== #

print\_hexa:

la $a0, result\_str2 # call result\_str

li $v0, 4

syscall

li $t3, 8 # counter

la $t2, result2 # where answer will be stored

loop2:

beqz $t3, exit # branch to exit if counter is equal to zero

rol $t0, $t0, 4 # rotate 4 bits to the left

and $t1, $t0, 0xf # mask with 1111

ble $t1, 9, sum # if less than or equal to nine, branch to sum

addi $t1, $t1, 55 # if greater than nine, add 55

j end

sum:

addi $t1, $t1, 48 # add 48 to result -> if $t2 = 1 then $t2 assign "1" in ASCII

end:

sb $t1, 0($t2) # store hex digit into result

addi $t2, $t2, 1 # increment address counter

addi $t3, $t3, -1 # decrement loop counter

j loop2

exit:

la $a0, result2

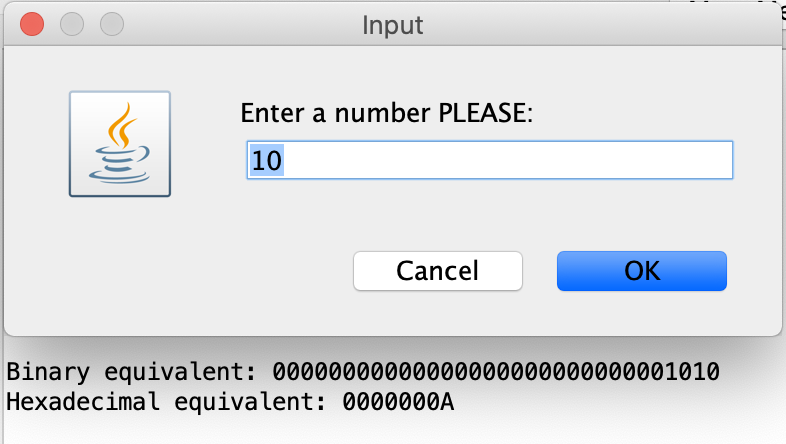
li $v0, 4

syscall

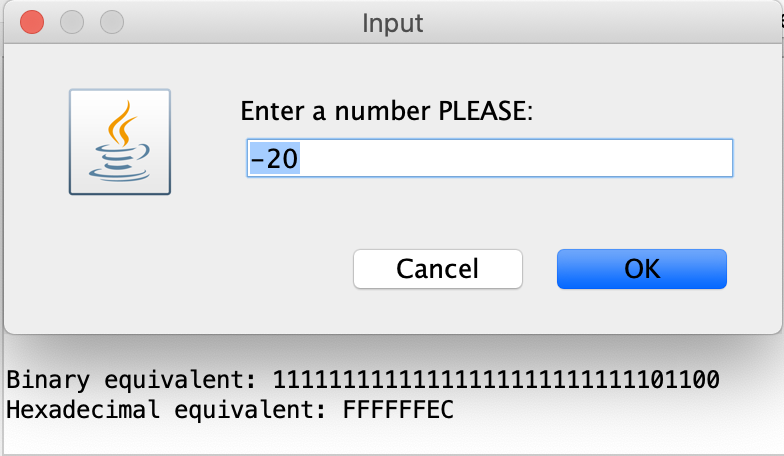
j $ra

1. Kết quả

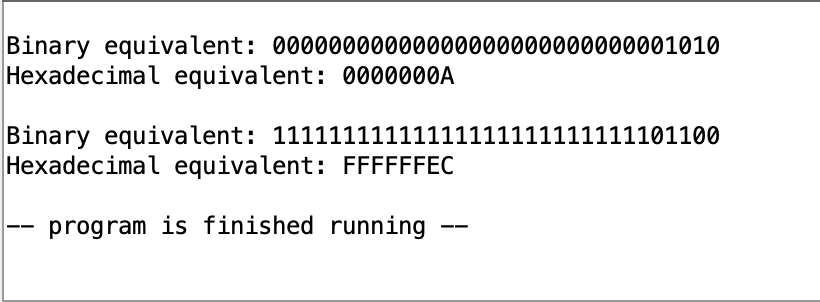
* Input = 10:



* Input = -20:



* Input = 0 🡪 Kết thúc chương trình



# **Project 7:**

Some people are standing in a row in a park. There are trees between them which cannot be moved. Your task is to rearrange the people by their heights in a non-descending order without moving the trees.Example: For a = [-1, 150, 190, 170, -1, -1, 160, 180], the output should be sortByHeight(a) = [-1, 150, 160, 170, -1, -1, 180, 190].

1. Phân tích cách thực hiện

* Phân tích đề bài
* Input là 1 mảng số nguyên hệ thập phân.
* Giá trị -1 biểu thị cho tree (cây).
* Output hiển thị ra màn hình mảng input đã được sắp xếp theo thứ tự tăng dần nhưng vẫn giữ nguyên vị tí của các phần tử có giá trị bằng -1 trong mảng input.
* Chương trình chính được chia thành các phần như sau:
* main: người dùng nhập vào số lượng phần tử của mảng input, xử lý các lỗi input.
* loop: người dùng nhập vào các phần tử của mảng input.
* find\_height: lấy các giá trị là chiều cao thật của con người (khác -1) trong mảng input và cho vào mảng height.
* sort\_height: sắp xếp mảng height theo thứ tự không giảm.
* replace: thay thế những giá trị đã được sắp xếp trong mảng height vào mảng input.
* output: in ra mảng đã được sắp xếp ra màn hình console.
* Cách chạy chương trình:
* Nhập số lượng phần tử của mảng input.
* Nhập các phần tử của mảng input.
* Output sẽ hiển thị dãy số đã sắp xếp theo yêu cầu của đề bài tại cửa sổ console.

1. Ý nghĩa của các thanh ghi được sử dụng

* Trong thủ tục main:
* $v0: mode InputDialogInt của syscall
  + - $a0: địa chỉ của message "Enter length of input array", sau khi gọi syscall thì $a0 sẽ lưu giá trị mà người dùng nhập vào
    - $t5: giá trị -1
    - $t6: giá trị -2
    - $t7: giá trị -3
    - $s6: giá trị của status trả về sau khi người dùng nhập từ bàn phím
    - $t8: độ dài của mảng input do người dùng nhập vào (đã kiểm tra hợp lệ)
    - $t0: index của phần tử đầu tiên trong mảng input
    - $t2: địa chỉ của mảng input
* Trong thủ tục loop:
  + $t1: kết quả của phép set less than giữa $t0 và $t8
  + $v0: mode InputDialogInt của syscall
  + $a0: địa chỉ của message "Enter element of input array", sau khi gọi syscall thì $a0 lưu giá trị mà người dùng nhập vào
  + $s6: giá trị của status trả về sau khi người dùng nhập từ bàn phím
  + $t0: index của phần tử kế tiếp trong mảng input
  + $t2: địa chỉ của phần tử kế tiếp trong mảng input
  + $a1: địa chỉ của mảng height
  + $s0: giá trị -1 thể hiện tree
  + $s1: số phần tử của mảng input
  + $s2: số phần tử của mảng height
* Trong thủ tục find\_height:
  + $t0: thể hiện chỉ số i tương ứng với phần tử đang xét của mảng input
  + $t2: thể hiện chỉ số j tương ứng với phần tử đang xét của mảng height
  + $s3: giá trị của phần tử thứ i trong mảng input
  + $t3: địa chỉ của phần tử tiếp theo trong mảng height
  + $t4: kết quả của phép set less than giữa chỉ số i của mảng input và chiều dài mảng input
  + $s2: số lượng phần tử hiện thời của mảng height (tăng dần trong thủ tục find\_height)
* Trong thủ tục sort\_height:
  + $t0: thể hiện chỉ số i tương ứng với phần tử đang xét của mảng input
  + $t1: thể hiện chỉ số j tương ứng với phần tử đang xét của mảng height
  + $t6: kết quả của phép set less than giữa chỉ số i và số lượng phần tử của mảng height
  + $t3: địa chỉ của phần tử tiếp theo trong mảng height
  + $s3: giá trị của phần tử height[j]
  + $s4: giá trị của phần tử height[j+1]
  + $t4: kết quả của phép set less than giữa height[j] và height[j+1]
* Trong thủ tục replace:
  + $t0: thể hiện chỉ số i tương ứng với phần tử đang xét của mảng input
  + $t2: thể hiện chỉ số j tương ứng với phần tử đang xét của mảng height
  + $t1: địa chỉ của phần tử thứ i trong mảng input
  + $s3: giá trị của phần tử thứ i trong mảng input
  + $t3: địa chỉ của phần tử thứ j trong mảng height
  + $s4: dùng để reset giá trị của phần tử thứ j tỏng mảng height về 0
* Trong thủ tục output:
  + $t1: kết quả của phép set less than giữa index i và độ dài mảng input
  + $v0: mode khi gọi syscall
  + $a1: mode message
  + $t2: địa chỉ của phần tử tiếp theo được xét trong mảng input

1. Source code

#======================================================================

# Project 7: sortByHeight

# Task: Some people are standing in a row in a park. There are trees between them which cannot be moved.

# Your task is to rearrange the people by their heights in a non- descending order without moving the trees.

# People can be very tall!

#

# Example: For a = [-1, 150, 190, 170, -1, -1, 160, 180]

# the output should be sortByHeight(a) = [-1, 150, 160, 170, -1, # -1, 180, 190].

# =====================================================================

# =====================================================================

# Summary

# @input: A (input array)

# @note: -1 represent trees

# height: store height of person in input array

# @idea: Input array

# 1. @find\_height: find height of person in A array => store result in 'height' array

# 2. @sort\_height: sort the 'height' array in ascending order

# 3. @replace: replace values in A array by a value in 'height' arrays

# =====================================================================

.data

# Input array

message: .asciiz "Enter length of input array: "

message1: .asciiz "Enter element of input array: "

message\_error1: .asciiz "Input type error! Please enter integer input!"

message\_error2: .asciiz "Cancel was choosen! Please enter integer input!"

message\_error3: .asciiz "No data was input! Please enter integer input!"

message\_done: .asciiz "Sorted array: "

blank: .asciiz " "

message\_error: .ascii "Error input type!!! Please enter integer number!"

A: .word 0:100 # input array

height: .word # array to store height of each person

.text

main:

# constructor Array

li $v0, 51 # InputDialogInt

la $a0, message

syscall

addi $t5, $zero, -1 # t5 = -1: input error

addi $t6, $zero, -2 # t6 = -2: cancel error

addi $t7, $zero, -3 # t7 = -3: no data input error

add $s6, $zero, $a1 # s6 = a1 = status value of InputDialogInt

# if s6 == 0: OK

# if s6 == -1: input error -> needhandle

beq $s6, $t5, output\_error1 # input type error

beq $s6, $t6, output\_error2 # cancel error

beq $s6, $t7, output\_error3 # no data input error

add $t8, $a0, $zero # $t8 = length of input array

li $t0, 0 # $t0 = i = 0

la $t2, A # load address of A array

loop:

# Get input array from user

slt $t1, $t0, $t8 # if i < length

beqz $t1, end\_loop

li $v0, 51 # InputDialogInt

la $a0, message1

syscall

add $s6, $zero, $a1 # s6 = a1 = status value of InputDialogInt

# if s6 == 0: OK

# if s6 == -1: input error -> need handle

beq $s6, $t5, error1 # input type error

beq $s6, $t6, error2 # cancel error

beq $s6, $t7, error3 # no data input error

sw $a0, 0($t2) # save value of element in A[i]

addi $t2, $t2, 4

addi $t0, $t0, 1

j loop

end\_loop:

la $a0, A # load address of A array

la $a1, height # load address of height array

addi $s0, $zero, -1 # -1: represent tree in 'A' array

add $s1, $zero, $t8 # n: length of 'A' array

addi $s2, $zero, 0 # m: lenght of 'height' array

j find\_height

after\_find\_height:

j sort\_height

after\_sort:

j replace

after\_replace:

li $t0, 0 # $t0 = i = 0

la $t2, A # load address of A array

li $v0, 4

la $a0, message\_done # print stringn output

syscall

output:

loop\_output:

slt $t1, $t0, $t8 # if i < length

beqz $t1, end\_loop\_output

li $v0, 1 # InputDialogInt

lw $a0, 0($t2) # save value of element in A[i]

syscall

li $v0, 4

la $a0, blank # print blank

syscall

addi $t2, $t2, 4 # next element in A

addi $t0, $t0, 1 # i = i + 1

j loop\_output

end\_loop\_output:

li $v0, 10 # terminate

syscall

output\_error1:

li $v0, 55

li $a1, 2 # warning message

la $a0, message\_error1

syscall

j main

output\_error2:

li $v0, 55

li $a1, 2 # warning message

la $a0, message\_error2

syscall

j main

output\_error3:

li $v0, 55

li $a1, 2 # warning message

la $a0, message\_error3

syscall

j main

error1:

li $v0, 55

li $a1, 2 # warning message

la $a0, message\_error1

syscall

j loop

error2:

li $v0, 55

li $a1, 2 # warning message

la $a0, message\_error2

syscall

j loop

error3:

li $v0, 55

li $a1, 2 # warning message

la $a0, message\_error3

syscall

j loop

end\_output:

end\_main:

#-----------------------------------------------------------

# 1. @find\_height: find height of person in A array => store result in 'height' array

# @input: A array (input array)

# @output: 'height' array

#-----------------------------------------------------------

find\_height:

# initialize i, j

addi $t0, $zero, 0 # i = 0

addi $t2, $zero, 0 # j = 0

fh\_loop:

sll $t1, $t0, 2 # $t1 = 4\*i

add $t1, $t1, $a0 # $t1 stores address of A[i]

lw $s3, 0($t1) # load value of A[i]

beq $s3, $s0, fh\_continue # if A[i] == -1 => continue(ignore tree)

sll $t3, $t2, 2 # $t3 = 4\*j

add $t3, $t3, $a1 # $t3 store address of height[j]

sw $s3, 0($t3) # store value of height[j] in $s3

addi $t2, $t2, 1 # j = j + 1

addi $s2, $s2, 1 # m = m + 1 (increase number of elements in 'height' array)

fh\_continue:

addi $t0, $t0, 1 # i = i + 1

slt $t4, $t0, $s1 # if i < n => True: return 1; False: return 0

bne $t4, $zero, fh\_loop

fh\_end\_loop:

j after\_find\_height

#### Sort 'height' array in ascending order using BubbleSort

#-------------------------------------------------------------------------

# 2. @sort\_height: Sort 'height' array in ascending order

# @input: height - random order

# @output: height - sorted in ascending order

#-------------------------------------------------------------------------

sort\_height:

# Initialize index i of loop\_1 to 0

addi $t0, $zero, 0 # i = 0

loop\_1:

# Initialize index j of loop\_2 to 0

addi $t1, $zero, 0 # j = 0

addi $t0, $t0, 1 # i = i + 1

sub $t2, $s2, $t0 # m - i - 1

# If i < m - 1

slt $t6, $t0, $s2

beq $t6, $zero, end\_loop\_1

loop\_2:

# If j < m - i - 1

slt $t5, $t1, $t2 # j < m - i - 1: True return 1; else return 0

beq $t5, $zero, end\_loop\_2

sll $t3, $t1, 2 # $t3 = 4\*j

add $t3, $t3, $a1 # $t3 stores address of height[j]

lw $s3, 0($t3) # load value of height[j] to $s3

lw $s4, 4($t3) # load value of height[j+1] to $s4

if:

slt $t4, $s3, $s4 # if A[j] < A[j+1] => True: return 1

# False: return 0

bne $t4, $zero, end\_if

# If height[j] > height[j + 1] => Swap height[j] and height[j+1]

sw $s4, 0($t3)

sw $s3, 4($t3)

end\_if:

addi $t1, $t1, 1 # j = j + 1

j loop\_2

end\_loop\_2:

j loop\_1

end\_loop\_1:

j after\_sort

#-------------------------------------------------------------------------

# 3. @replace: replace values in A array by a value in 'height' array

# @input: A (input array)

# @output: A (sorted input array in ascending order)

# @note: Reset values of 'height' array = 0

#-------------------------------------------------------------------------

replace:

# Initialize i, j

addi $t0, $zero, 0 # i = 0

addi $t2, $zero, 0 # j = 0

i\_loop:

sll $t1, $t0, 2 # $t1 = 4\*i

add $t1, $t1, $a0 # $t1 stores address of A[i]

lw $s3, 0($t1) # load address of A[i]

beq $s3, $s0, i\_continue # if A[i] == -1 => continue

sll $t3, $t2, 2 # $t3 = 4\*j

add $t3, $t3, $a1 # $t3 stores address of height[j]

lw $s3, 0($t3) # load value of height[j]

sw $s3, 0($t1) # A[i] = height[j]

addi $s4, $zero, 0

sw $s4, 0($t3) # Reset value0 of height[j] = 0

addi $t2, $t2, 1 # j = j + 1

i\_continue:

addi $t0, $t0, 1 # i = i + 1

slt $t4, $t0, $s1 # if i < n: True return: 1; False return: 0

bne $t4, $zero, i\_loop

i\_end\_loop:

j after\_replace

#-------------------------------------------------------------------------

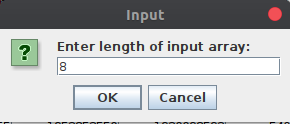
# END

#-------------------------------------------------------------------------

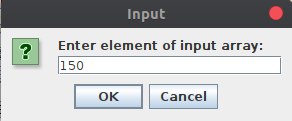
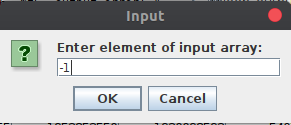
1. Kết quả:
2. Sắp xếp mảng đầu vào theo yêu cầu:

Đầu vào:

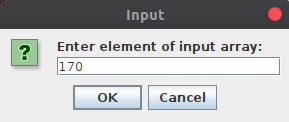
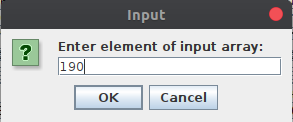
* Đầu tiên ta cần nhập số lượng phần tử của mảng: ở đây ta nhập là 8 (ảnh 1)
* Sau đó ta cần nhập các phần tử của mảng input. (lần lượt các ảnh 2 -> ảnh 9)



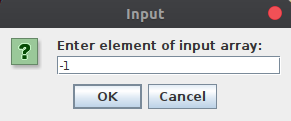
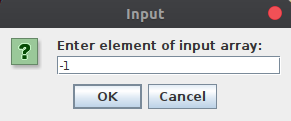
(Ảnh 1)



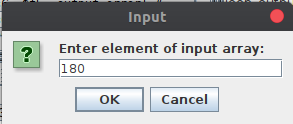
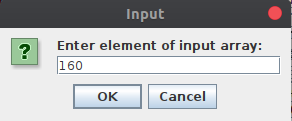
(Ảnh 2) (Ảnh 3)



(Ảnh 4) (Ảnh 5)

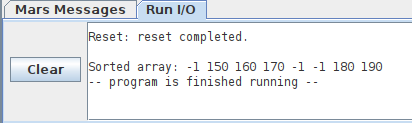


(Ảnh 6) (Ảnh 7)



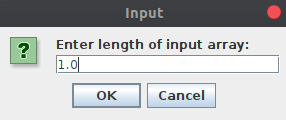
(Ảnh 8) (Ảnh 9)

Kết quả:

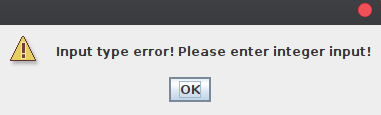


1. Một số ví dụ về xử lý lỗi input:

**Lỗi 1**: Nhập sai kiểu dữ liệu Integer -> Float

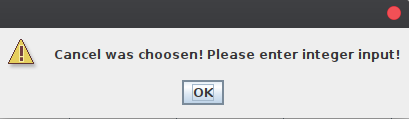


Chương trình báo lỗi và cho người dùng nhập lại:



**Lỗi 2: Người dùng không nhập mà nhấn Cancel:**

Chương trình báo lỗi và cho người dùng nhập lại:



**Lỗi 3: Người dùng không nhập mà nhấn OK:**

Chương trình báo lỗi và cho người dùng nhập lại:

