ĐẠI HỌC BÁCH KHOA HÀ NỘI

TRƯỜNG CÔNG NGHỆ THÔNG TIN & TRUYỀN THÔNG

Nhóm 6 - Bài Tập 1, 5

BÁO CÁO CUỐI KỲ

THỰC HÀNH KIẾN TRÚC MÁY TÍNH

Giảng viên hướng dẫn: **Nguyễn Thành Trung**

Sinh viên: **Hoả Đức Việt - 20205046 Việt-Nhật 05 K65**

Mai Hoàng Việt - 20205047 Việt-Nhật 01 K65

MỤC LỤC

Bài Tập 1

I, Đề bài

1. Curiosity Marsbot

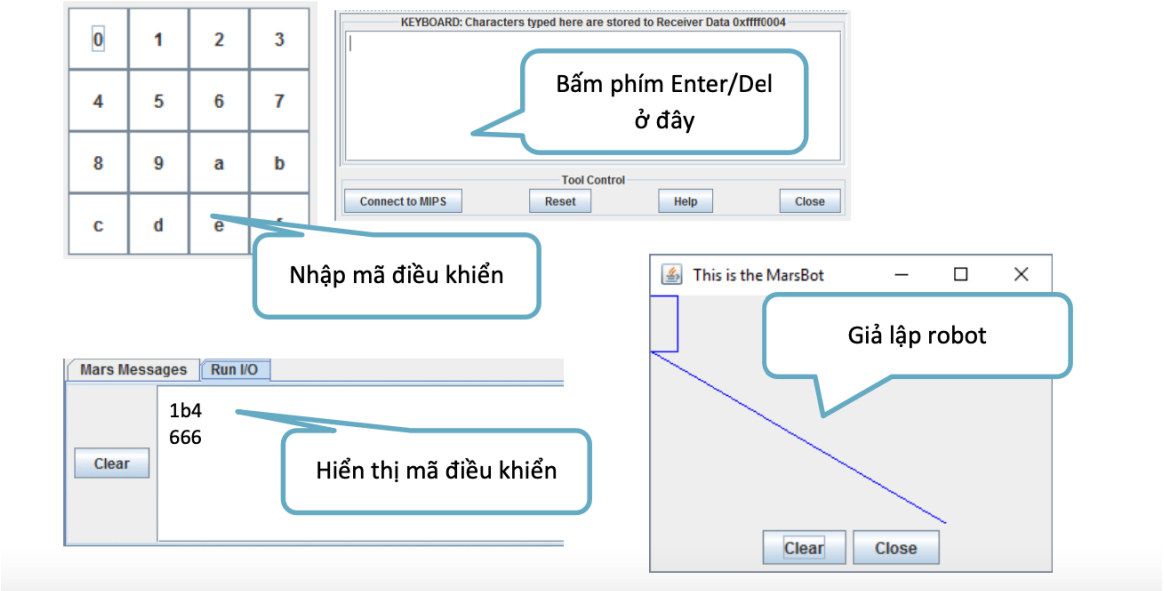
Xe tự hành Curioisity Marsbot chạy trên sao Hỏa, được vận hành từ xa bởi các lập trình viên trên Trái Đất. Bằng cách gửi đi các mã điều khiển từ một bàn phím ma trận, lập trình viên điều khiển quá trình di chuyển của Marbot như sau:

Mã điều khiển	Ý nghĩa
1b4	Marbot bắt đầu chuyển động
c68	Marbot đứng im
444	Rẽ trái 90* so vơi phương chuyển động gần đây và giữ hướng mới
666	Rẽ phải 90* so vơi phương chuyển động gần đây và giữ hướng mới
dad	Bắt đầu để lại vết trên đường
cbc	Chấm dứt để lại vết trên đường
999	Tự động quay trở lại theo lộ trình ngược lại. Không vẽ vết, không nhận mã khác cho tới khi kết thúc lộ trình ngược. Mô tả: Marsbot được lập trình để nhớ lại toàn bộ lịch sử các mã điều khiển và khoảng thời gian giữa các lần đổi mã. Vì vậy, nó có thể đảo ngược lại lộ trình để quay về điểm xuất phát (dù có thể lệch một chút do hàm syscall sleep không thực sự thời gian thực)

Sau khi nhận mã điều khiển, Curioisity Marsbot sẽ không xử lý ngay, mà phải đợi lệnh kích hoạt mã từ bàn phím Keyboard & Display MMIO Simulator. Có 2 lệnh như vậy:

Kích hoạt mã	Ý nghĩa
Phím Enter	Kết thúc nhập mã và yêu cầu Marbot thực thi
Phím Del	Xóa toàn bộ mã điều khiển đang nhập dở dang.

Hãy lập trình để Marsbot có thể hoạt động như đã mô tả.
Đồng thời bổ sung thêm tính năng: mỗi khi gửi một mã điều khiển cho Marsbot, hiển thị mã đó lên màn hình console để người xem có thể giám sát lộ trình của xe.



II, Cách làm

- **B1**: Mỗi khi người dùng nhập 1 ký tự từ **Digital Lab Sim** sẽ tạo interrupt để lưu kí tự được nhập vào bộ nhớ, tạo nên code điều khiển.
- **B2**: Kiểm tra liên tục xem ký tự ***Enter*** có được nhập ở **Keyboard & Display MMIO Simulator** hay không. Khi ký tự Enter được nhập, sẽ kiểm tra code điều khiển có hợp lệ hay không (gồm 3 ký tự), nếu không sẽ thông báo code lỗi và chuyển sang bước số 4.
- **B3:** Lần lượt kiểm tra xem code điều khiển được nhập vào có trùng với các đoạn code điều khiển đã quy định sẵn. Nếu không sẽ thông báo đoạn code bị lỗi. Nếu đúng sẽ thực hiện các thao tác theo chương trình.
- **B4**: In ra console code điều khiển đã nhập và xoá lưu trữ trong bộ nhớ.

III, Phân tích

1. **storePath**
    - Lưu lại thông tin về đường đi của Marsbot vào mảng path
    - Mảng path lưu thông tin về điểm đầu tiên của mỗi cạnh đường đi theo ***structure(x,y,h***) là toạ độ *x*, *y* và hướng đi *h* của cạnh đó.
2. **goBack**
    - Khi muốn quay ngược lại đường đã đi, ta lấy hướng của từng cạnh trên đường đi và đi ngược lại., điều khiển Marsbot đi ngược lại theo lộ trình đã đi về điểm xuất phát.
    - Dữ liệu vào là mảng ***path*** lưu thông tin đường đi, biến ***lengthPath*** lưu kích cỡ của mảng path theo bytes.
    - Mỗi thông tin đường đi trong path là cạnh gồm toạ độ *x*, *y* và hướng đi của cạnh đó. Nên mỗi thông tin chiếm 12 bytes.
3. **go Right và Left**
    - Điều khiển Marsbot di chuyển và quay sang trái (phải) một góc 90* so với hướng di chuyển hiện tại.
4. **GO và STOP**
    - Điều khiển Marsbot chuyển động (**GO**) hay dừng lại (**STOP**) bằng cách load vào địa chỉ **MOVING *(0xFFFF8050)* .**
5. **ROTATE**

- Quay Marsbot theo hướng có giá trị được lưu trong ***nowHeading.***
- load biến now Heading và lưu vào địa chỉ **HEADING (*0xFFFF8010*)** để Marsbot đổi hướng.

6. **TRACK và UNTRACK**
    - Điều khiển Marsbot để lại hướng hoặc không
    - Load địa chỉ vào **LEAVETRACK (*0xFFFF 8020*)**: nếu muốn để lại vết thì load 1 và load 0 nếu muốn kết thúc vết.
7. **ErrorMessage**
    - Thông báo lỗi người dùng khi code điều khiển không đúng
8. **EqualString**
    - So sánh các code được nhập từ bàn phím so với code điều khiển được đặt địa chỉ trong ***$s3.***
    - Nếu đúng thì tiếp tục thực hiện chương trình, ngược lại thông báo lỗi và thoát.
9. **remoteControlCode**
    - Xoá lần lượt các ký tự nhập vào tại input Control Code bằng cách gán các giá trị trong xâu bằng **'\0'.**
10. **.kext *0x8000180***
    - Nơi lưu giữ và tổ chức thiết lập địa chỉ của các ký tự trong **Digital Lab Sim.**
    - Dựa vào mã được trả về ghi kí tự tương ứng vào bộ nhớ và các thao tác xử lí interrupt

IV, Mã Nguồn

-Xem ở trong file đã nộp kèm: ***n01_g06_HoaDucViet_MaiHoangViet.asm***

MARSBOT

```
.eqv IN_ADRESS_HEXA_KEYBOARD 0xFFFF0012        #DIGITAL
.eqv OUT_ADRESS_HEXA_KEYBOARD 0xFFFF0014       #RUN I/O
#KEYBOARD
.eqv KEY_CODE  0xFFFF0004        # ASCII code from keyboard, 1 byte =1 if has a new
keycode ?
.eqv KEY_READY 0xFFFF0000        # Auto clear after lw
# MARSBOT
```

```
.eqv HEADING 0xffff8010             # Integer: An angle between 0 and 359
                                    # 0 : North (up)
                                    # 90: East (right)
                                    # 180: South (down)
                                    # 270: West (left)
.eqv MOVING 0xffff8050              # Boolean: whether or not to move
.eqv LEAVETRACK 0xffff8020          # Boolean (0 or non-0):
                                    # whether or not to leave a track
.eqv WHEREX 0xffff8030              # Integer: Current x-location of MarsBot
.eqv WHEREY 0xffff8040              # Integer: Current y-location of MarsBot
# KEY VALUE
        .eqv KEY_0 0x11
        .eqv KEY_1 0x21
        .eqv KEY_2 0x41
        .eqv KEY_3 0x81
        .eqv KEY_4 0x12
        .eqv KEY_5 0x22
        .eqv KEY_6 0x42
        .eqv KEY_7 0x82
        .eqv KEY_8 0x14
        .eqv KEY_9 0x24
        .eqv KEY_a 0x44
        .eqv KEY_b 0x84
        .eqv KEY_c 0x18
        .eqv KEY_d 0x28
        .eqv KEY_e 0x48
        .eqv KEY_f 0x88
.data
#Code control
        MOVE_CODE:     .asciiz   "1b4"
        STOP_CODE:     .asciiz   "c68"
        LEFT_CODE:     .asciiz   "444"
        RIGHT_CODE:    .asciiz   "666"
        TRACK_CODE:    .asciiz   "dad"
        UNTRACK_CODE:  .asciiz   "cbc"
        BACK_CODE:     .asciiz   "999"
        WRONG_CODE:    .asciiz   "Ma dieu khien khong dung!!!"
        inputControlCode: .space 500
        lengthControlCode: .word 0
        nowHeading: .word 0
```

```
        path: .space 600
        lengthPath: .word 12
.text
main:   li $k0, KEY_CODE
        li $k1, KEY_READY
        li $t1, IN_ADRESS_HEXA_KEYBOARD
        li $t3, 0x80                    # bit số 7 có giá trị bằng 1 để bật
        sb $t3, 0($t1)
        addi $t7, $zero, 0
loop:   nop
WaitForKey:
        lw $t5, 0($k1)                  # $t5 là giá trị sẵn sàng
        beq $t5, $zero, WaitForKey
        nop
        beq $t5, $zero, WaitForKey
ReadKey:
        lw $t6, 0($k0)                  # $t6 = [$k0] = KEY_CODE
        beq $t6, 127, continue          # Nếu $t6 = kí tự xoá thì chuyển đến continue
                                        # 127 là kỹ tự xóa trong mã ascii
        bne $t6, '\n' , loop
        nop
        bne $t6, '\n' , loop
CheckControlCode:
        la  $s2, lengthControlCode
        lw $s2, 0($s2)
        bne $s2, 3, ErrorMessage            #Nếu độ dài CODE khác 3 thì thông báo lỗi
        la $s3, MOVE_CODE
        jal EqualString
        beq $t0, 1, go
        la $s3, STOP_CODE
        jal EqualString
        beq $t0, 1, stop
        la $s3, LEFT_CODE
        jal EqualString
        beq $t0, 1, goLeft
        la $s3, RIGHT_CODE
        jal EqualString
        beq $t0, 1, goRight
        la $s3, TRACK_CODE
```

```
        jal EqualString
        beq $t0, 1, track
        la $s3, UNTRACK_CODE
        jal EqualString
        beq $t0, 1, untrack
        la $s3, BACK_CODE
        jal EqualString
        beq $t0, 1, goBack
        beq $t0, 0, ErrorMessage      #Nếu CODE không thuộc dữ kiện đề cho thì báo lỗi
printControlCode:
        li $v0,4
        la $a0, inputControlCode
        syscall
        nop
continue:
        jal removeControlCode
        nop
        j loop
        nop
        j loop
storePath:
        #Sao lưu vào ngăn xếp
        addi $sp, $sp, 4
        sw $t1, 0($sp)
        addi $sp, $sp, 4
        sw $t2, 0($sp)
        addi $sp, $sp, 4
        sw $t3, 0($sp)
        addi $sp, $sp, 4
        sw $t4, 0($sp)
        addi $sp, $sp, 4
        sw $s1, 0($sp)
        addi $sp, $sp, 4
        sw $s2, 0($sp)
        addi $sp, $sp, 4
        sw $s3, 0($sp)
        addi $sp, $sp, 4
        sw $s4, 0($sp)
        # Xử lý dữ liệu
        li $t1, WHEREX
```

```
        lw $s1, 0($t1)
        li $t2, WHEREY
        lw $s2, 0($t2)
        la $t3, lengthPath
        lw $s3, 0($t3)          # $s3 = lengthPath
        la $s4, nowHeading
        lw $s4, 0($s4)          # $s4 = nowHeading
        la $t4, path            # Gán địa chỉ mảng path cho $t4
        add $t4, $t4, $s3       # Chuyển đến địa chỉ chưa lưu giá trị của mảng path
        sw $s1, 0($t4)              # Lưu x
        sw $s2, 4($t4)              # Lưu y
        sw $s4, 8($t4)              # Lưu heading
        addi $s3, $s3, 12       # update lenghtPath: 3(word) x 4(bytes) = 12
        sw $s3, 0($t3)
        #Khôi phục dữ liệu
        lw $s4, 0($sp)
        addi $sp, $sp, -4
        lw $s3, 0($sp)
        addi $sp, $sp, -4
        lw $s2, 0($sp)
        addi $sp, $sp, -4
        lw $s1, 0($sp)
        addi $sp, $sp, -4
        lw $t4, 0($sp)
        addi $sp, $sp, -4
        lw $t3, 0($sp)
        addi $sp, $sp, -4
        lw $t2, 0($sp)
        addi $sp, $sp, -4
        lw $t1, 0($sp)
        jr $ra
        nop
        jr $ra
# QUAY LẠI BAN ĐẦU
# BEGIN GOBACK
goBack:
        #backup
        addi $sp,$sp,4
        sw $s5, 0($sp)
        addi $sp,$sp,4
```

```
        sw $s6, 0($sp)
        addi $sp,$sp,4
        sw $s7, 0($sp)
        addi $sp,$sp,4
        sw $t7, 0($sp)
        addi $sp,$sp,4
        sw $t8, 0($sp)
        addi $sp,$sp,4
        sw $t9, 0($sp)
        jal UNTRACK             # Xoá lựa chọn vết khi quay về,
        nop
        la $s7, path            # $s7 = mảng path
        la $s5, lengthPath
        lw $s5, 0($s5)
        add $s7, $s7, $s5
        # dùng path như 1 ngăn xếp
begin:
        addi $s5, $s5, -12      # Lùi lại 1 structure
        addi $s7, $s7, -12      # Lùi về cạnh cuối cùng
        lw $s6, 8($s7)          # đưa hướng vị trí cuối cùng vào thanh ghi $6
        addi $s6, $s6, 180      # Quay ngược hướng ban đầu
        la $t7, nowHeading
        sw $s6, 0($t7)
        jal ROTATE
loop_goBack:
        lw $t9, 0($s7)          # Toạ độ x xuất phát
        li $t8, WHEREX          # Toạ độ x hiện tại
        lw $t8, 0($t8)
        bne $t8, $t9, loop_goBack
        nop
        bne $t8, $t9, loop_goBack
        lw $t9, 4($s7)
        li $t8, WHEREY          # Toạ độ y xuất phát
        lw $t8, 0($t8)          # Toạ độ y hiện tại
        bne $t8, $t9, loop_goBack
        nop
        bne $t8, $t9, loop_goBack
        beq $s5, 0, finish
        nop
        beq $s5, 0, finish
```

```
        j begin
        nop
        j begin
# END GO BACK
finish:
        jal STOP
        la $t7, nowHeading
        add $s6, $zero, $zero
        sw $s6, 0($t7)                  # update heading = 0
        la $t8, lengthPath
        addi $s5, $zero, 12
        sw $s5, 0($t8)                  # update lengthPath = 12
        #restore
        lw $t9, 0($sp)
        addi $sp,$sp,-4
        lw $t8, 0($sp)
        addi $sp,$sp,-4
        lw $t7, 0($sp)
        addi $sp,$sp,-4
        lw $s7, 0($sp)
        addi $sp,$sp,-4
        lw $s6, 0($sp)
        addi $sp,$sp,-4
        lw $s5, 0($sp)
        addi $sp,$sp,-4
        jal ROTATE
        j printControlCode
go:
        jal GO
        j printControlCode
stop:
        jal STOP
        j printControlCode
track:
        jal TRACK
        j printControlCode
untrack:
        jal UNTRACK
        j printControlCode
```

```
goRight:
        #backup
        addi $sp,$sp,4
        sw $s5, 0($sp)
        addi $sp,$sp,4
        sw $s6, 0($sp)
        beq $t7, 1, next_1
        j goRight_1
goRight_1:
        la $s5, nowHeading          #Gán địa chỉ nowheading vào $t5
        lw $s6, 0($s5)
        addi $s6, $s6, 90
        sw $s6, 0($s5)              #Cập nhật lạ giá trị nowheading
        #restore
        lw $s6, 0($sp)
        addi $sp,$sp,-4
        lw $s5, 0($sp)
        addi $sp,$sp,-4
        jal storePath
        nop
        jal ROTATE
        nop
        j printControlCode
# Xóa và tạo vết mới
next_1:
        jal UNTRACK
        nop
        jal TRACK
        nop
        j goRight_1
goLeft:
        #backup
        addi $sp,$sp,4
        sw $s5, 0($sp)
        addi $sp,$sp,4
        sw $s6, 0($sp)
        beq $t7, 1, next_2
        j goLeft_1
```

```
goLeft_1:
        la $s5, nowHeading              #Gán địa chỉ nowheading vào $t5
        lw $s6, 0($s5)
        addi $s6, $s6, -90
        sw $s6, 0($s5)                  #Cập nhật lạ giá trị nowheading
        #restore
        lw $s6, 0($sp)
        addi $sp,$sp,-4
        lw $s5, 0($sp)
        addi $sp,$sp,-4
        jal storePath
        jal ROTATE
        nop
        j printControlCode
next_2:
        jal UNTRACK
        nop
        jal TRACK
        nop
        j goLeft_1
removeControlCode:
        #backup cac thanh ghi
        addi $sp, $sp, 4
        sw $t1, 0($sp)
        addi $sp, $sp, 4
        sw $s1, 0($sp)
        addi $sp, $sp, 4
        sw $t2, 0($sp)
        addi $sp, $sp, 4
        sw $s2, 0($sp)
        addi $sp, $sp, 4
        sw $t3, 0($sp)
        la $s1, inputControlCode
        la $s2, lengthControlCode
        lw $t3, 0($s2)                  #$t3 = lengthControlCode
        addi $t1, $zero, -1             #$t1 = -1
        addi $t2, $zero, 0              #$t2 = '\0'
        addi $s1, $s1, -1
loop_remove:
        addi $t1, $t1, 1                # i++
```

```
        add $s1, $s1, 1
        sb $t2, 0($s1)                  # Đặt lại giá trị inputControl = 0
        bne $t1, $t3, loop_remove       # $t1 <= 3 thì tiếp tục loop_remove
        nop
        bne $t1, $t3, loop_remove
        add $t3, $zero, $zero
        sw $t3, 0($s2)                          # lengthControlCode = 0
        #restore lai gia tri cac thanh ghi
        lw $t3, 0($sp)
        addi $sp, $sp, -4
        lw $s2, 0($sp)
        addi $sp, $sp, -4
        lw $t2, 0($sp)
        addi $sp, $sp, -4
        lw $s1, 0($sp)
        addi $sp, $sp, -4
        lw $t1, 0($sp)
        addi $sp, $sp, -4
        jr $ra
        nop
        jr $ra
EqualString:
        #backup gia tri
        addi $sp, $sp, 4
        sw $t1, 0($sp)
        addi $sp, $sp, 4
        sw $s1, 0($sp)
        addi $sp, $sp, 4
        sw $t2, 0($sp)
        addi $sp, $sp, 4
        sw $t3, 0($sp)
        add $t0, $zero, $zero       #Khởi tạo giá trị $t0 = 0
        addi $t1, $zero, -1
        la $s1, inputControlCode
loop_equal:
        addi $t1, $t1, 1            #i++
        add $t2, $s1, $t1           #$t2 = inputControl + i
        lb $t2, 0($t2)              #$t2 = inputControl[i]
        add $t3, $s3, $t1
        lb $t3, 0($t3)
```

```
        bne $t2, $t3, not_equal
        bne $t1, 2, loop_equal          # $t1 <= 2 thì tiếp tục loop_equal
        nop
        bne $t1, 2, loop_equal
equal:
        #restore
        lw $t3, 0($sp)
        addi $sp, $sp, -4
        lw $t2, 0($sp)
        addi $sp, $sp, -4
        lw $s1, 0($sp)
        addi $sp, $sp, -4
        lw $t1, 0($sp)
        addi $sp, $sp, -4
        add $t0, $zero, 1           #Nếu String đúng thì update $t0 = 1
        jr $ra
        nop
        jr $ra
not_equal:
        #restore
        lw $t3, 0($sp)
        addi $sp, $sp, -4
        lw $t2, 0($sp)
        addi $sp, $sp, -4
        lw $s1, 0($sp)
        addi $sp, $sp, -4
        lw $t1, 0($sp)
        addi $sp, $sp, -4
        add $t0, $zero, $zero       #Nếu String sai thì update $t0 = 0
        jr $ra
        nop
        jr $ra
ErrorMessage:
        li $v0, 4
        la $a0, inputControlCode        #Hiển thị lại dòng COĐE bị lỗi
        syscall
        nop
        li $v0, 55                  #Hiển thị cảnh báo CODE lỗi
        la $a0, WRONG_CODE
```

```
        syscall
        nop
        nop
        j continue
        nop
        j continue
GO:
        #backup
        addi $sp,$sp,4
        sw $at,0($sp)
        addi $sp,$sp,4
        sw $k0,0($sp)
        li $at, MOVING
        addi $k0, $zero, 1
        sb $k0, 0($at)
        #restore
        lw $k0, 0($sp)
        addi $sp,$sp,-4
        lw $at, 0($sp)
        addi $sp,$sp,-4
        jr $ra
        nop
        jr $ra
STOP:
        #backup
        addi $sp,$sp,4
        sw $at,0($sp)
        li $at, MOVING
        sb $zero, 0($at)
        #restore
        lw $at, 0($sp)
        addi $sp,$sp,-4
        jr $ra
        nop
        jr $ra
TRACK:
        #backup
        addi $sp,$sp,4
        sw $at,0($sp)
        addi $sp,$sp,4
```

```
        sw $k0,0($sp)

        li $at, LEAVETRACK
        addi $k0, $zero, 1
        sb $k0, 0($at)
        addi $t7, $zero, 1
        #restore
        lw $k0, 0($sp)
        addi $sp,$sp,-4
        lw $at, 0($sp)
        addi $sp,$sp,-4
        jr $ra
        nop
        jr $ra
UNTRACK:
        #backup
        addi $sp,$sp,4
        sw $at,0($sp)
        li $at, LEAVETRACK
        sb $zero, 0($at)
        addi $t7, $zero, 0
        #restore
        lw $at, 0($sp)
        addi $sp,$sp,-4
        jr $ra
        nop
        jr $ra
ROTATE:
        #backup
        addi $sp,$sp,4
        sw $t1,0($sp)
        addi $sp,$sp,4
        sw $t2,0($sp)
        addi $sp,$sp,4
        sw $t3,0($sp)
        li $t1, HEADING
        la $t2, nowHeading
        lw $t3, 0($t2)
        sw $t3, 0($t1)
        #restore
```

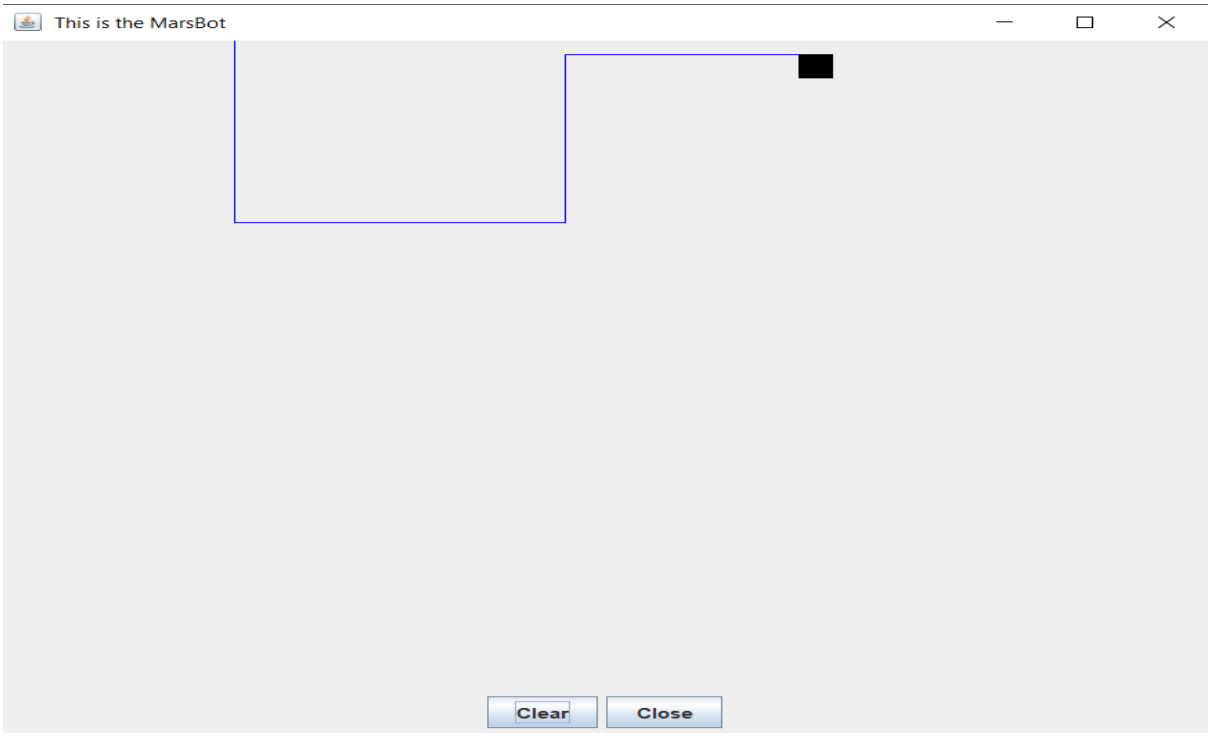
```
        lw $t3, 0($sp)
        addi $sp,$sp,-4
        lw $t2, 0($sp)
        addi $sp,$sp,-4
        lw $t1, 0($sp)
        addi $sp,$sp,-4
        jr $ra
        nop
        jr $ra
#-------------------------------------------------------------------------------------
.ktext 0x80000180
backup:
        addi $sp, $sp, 4
        sw $ra, 0($sp)
        addi $sp, $sp, 4
        sw $at, 0($sp)
        addi $sp, $sp, 4
        sw $a0, 0($sp)
        addi $sp, $sp, 4
        sw $t1, 0($sp)
        addi $sp, $sp, 4
        sw $t2, 0($sp)
        addi $sp, $sp, 4
        sw $t3, 0($sp)
        addi $sp, $sp, 4
        sw $t4, 0($sp)
        addi $sp, $sp, 4
        sw $s0, 0($sp)
        addi $sp, $sp, 4
        sw $s1, 0($sp)
        addi $sp, $sp, 4
        sw $s2, 0($sp)
        addi $sp, $sp, 4
        sw $s3, 0($sp)
get_code:
        li $t1, IN_ADRESS_HEXA_KEYBOARD
        li $t2, OUT_ADRESS_HEXA_KEYBOARD
scan_row1:
        li $t3, 0x81
        sb $t3, 0($t1)
```

```
        lbu $a0, 0($t2)
        bnez $a0, get_code_in_char
scan_row2:
        li $t3, 0x82
        sb $t3, 0($t1)
        lbu $a0, 0($t2)
        bnez $a0, get_code_in_char
scan_row3:
        li $t3, 0x84
        sb $t3, 0($t1)
        lbu $a0, 0($t2)
        bnez $a0, get_code_in_char
scan_row4:
        li $t3, 0x88
        sb $t3, 0($t1)
        lbu $a0, 0($t2)
        bnez $a0, get_code_in_char
get_code_in_char:
        beq $a0, KEY_0, case_0
        beq $a0, KEY_1, case_1
        beq $a0, KEY_2, case_2
        beq $a0, KEY_3, case_3
        beq $a0, KEY_4, case_4
        beq $a0, KEY_5, case_5
        beq $a0, KEY_6, case_6
        beq $a0, KEY_7, case_7
        beq $a0, KEY_8, case_8
        beq $a0, KEY_9, case_9
        beq $a0, KEY_a, case_a
        beq $a0, KEY_b, case_b
        beq $a0, KEY_c, case_c
        beq $a0, KEY_d, case_d
        beq $a0, KEY_e, case_e
        beq $a0, KEY_f, case_f
case_0:         li $s0, '0'
        j store_code
case_1:         li $s0, '1'
        j store_code
case_2:         li $s0, '2'
        j store_code
```

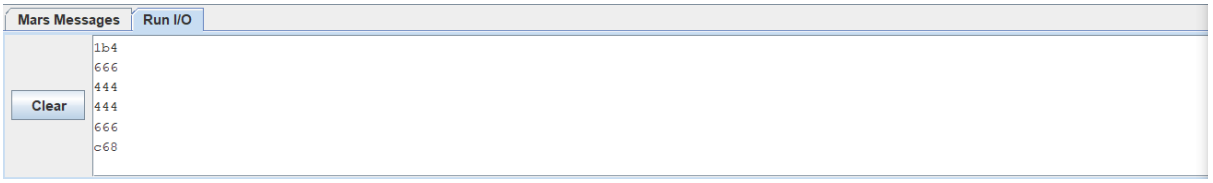
```
case_3:         li $s0, '3'
        j store_code
case_4:         li $s0, '4'
        j store_code
case_5:         li $s0, '5'
        j store_code
case_6:         li $s0, '6'
        j store_code
case_7:         li $s0, '7'
        j store_code
case_8:         li $s0, '8'
        j store_code
case_9:         li $s0, '9'
        j store_code
case_a:li $s0, 'a'
        j store_code
case_b:         li $s0, 'b'
        j store_code
case_c:li $s0, 'c'
        j store_code
case_d:         li $s0, 'd'
        j store_code
case_e:li $s0,  'e'
        j store_code
case_f: li $s0, 'f'
        j store_code
store_code:
        la $s1, inputControlCode
        la $s2, lengthControlCode
        lw $s3, 0($s2)                  # $s3 = lenghtControlCode
        addi $t4, $t4, -1               # $st4 = i
loop_store_code:
        addi $t4, $t4, 1
        bne $t4, $s3, loop_store_code
        add $s1, $s1, $t4               # $s1 = inputControlCode + i
        sb $s0, 0($s1)                  # $s0 = inputControlCode[i]
        addi $s0, $zero, '\n'           # add '\n' để kết thúc chuỗi
        addi $s1, $s1, 1
        sb $s0, 0($s1)
```

```
        addi $s3, $s3, 1
        sw $s3, 0($s2)                  # update lenghtControlCode
next_pc:
        mfc0 $at, $14                   # $at <= Coproc0.$14 = Coproc0.epc
        addi $at, $at, 4                # $at = $at + 4 (next instruction)
        mtc0 $at, $14                   # Coproc0.$14 = Coproc0.epc <= $at
restore:
        lw $s3, 0($sp)
        addi $sp, $sp, -4
        lw $s2, 0($sp)
        addi $sp, $sp, -4
        lw $s1, 0($sp)
        addi $sp, $sp, -4
        lw $s0, 0($sp)
        addi $sp, $sp, -4
        lw $t4, 0($sp)
        addi $sp, $sp, -4
        lw $t3, 0($sp)
        addi $sp, $sp, -4
        lw $t2, 0($sp)
        addi $sp, $sp, -4
        lw $t1, 0($sp)
        addi $sp, $sp, -4
        lw $a0, 0($sp)
        addi $sp, $sp, -4
        lw $at, 0($sp)
        addi $sp, $sp, -4
        lw $s3, 0($sp)
        addi $ra, $sp, -4
return:
        eret
```

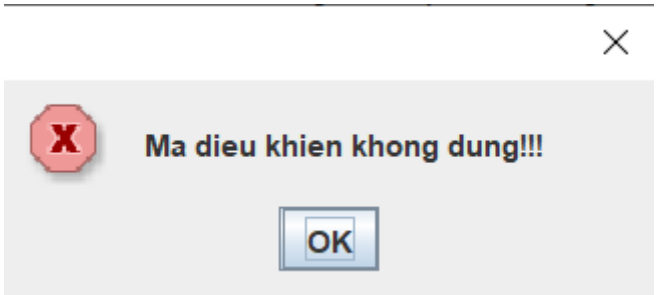
V, Hình Ảnh Mô Phỏng



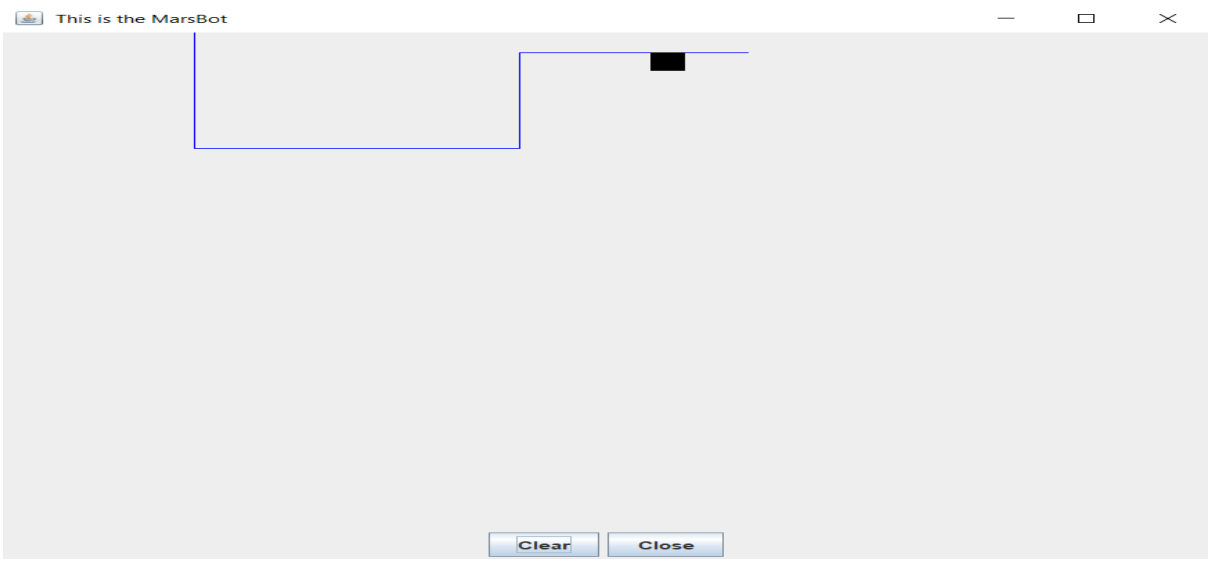
Đường đi của Marsbot

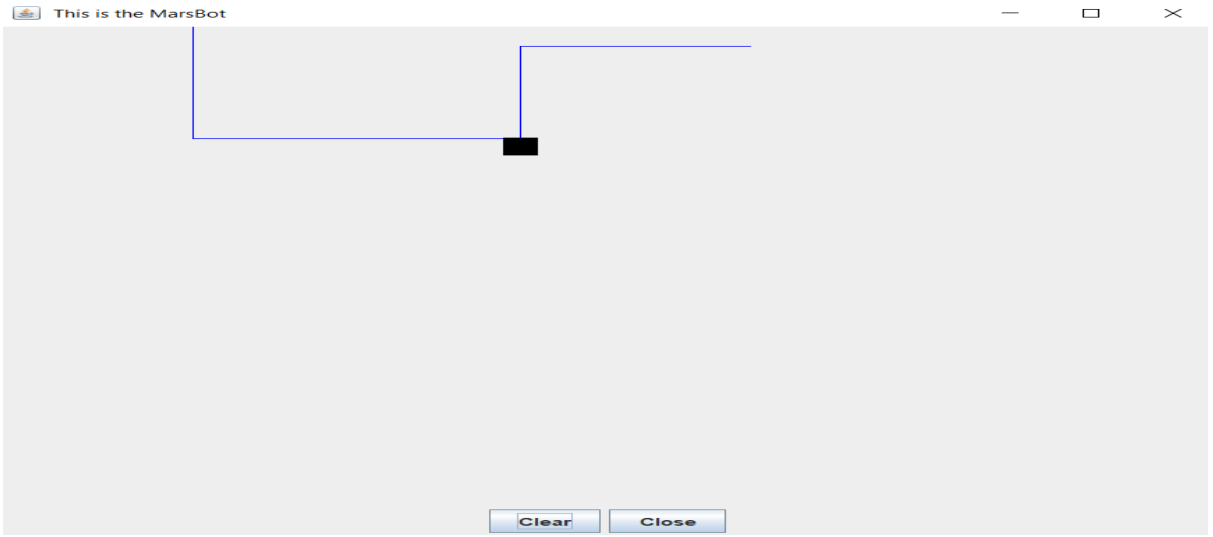


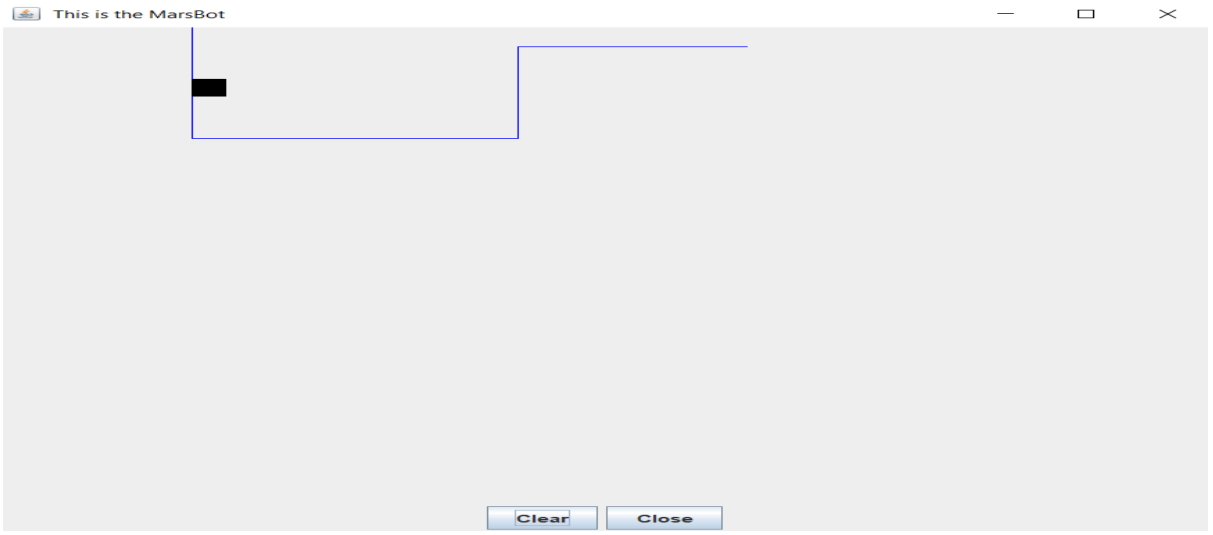
Mã lệnh được nhập



Thông báo khi người dùng nhập sai







Quá trình đi ngược của Marsbot

Bài Tập 5

5. Biểu thức trung tố hậu tố

Viết chương trình tính giá trị biểu thức bất kỳ bằng phương pháp duyệt biểu thức hậu tố.

Các yêu cầu cụ thể:

1. Nhập vào biểu thức trung tố, ví dụ: 9 + 2 + 8 * 6
2. In ra biểu thức ở dạng hậu tố, ví dụ: 9 2 + 8 6 * +
3. Tính ra giá trị của biểu thức vừa nhập

Các hằng số là số nguyên, trong phạm vi từ 0 → 99.

Toán tử bao gồm phép cộng, trừ, nhân, chia lấy thương

I,Cách làm:

1,Nhập vào biểu thức trung tố và in ra ở dạng hậu tố

-> bài này ta sẽ dùng ngăn xếp và xâu để giải quyết bài toán

- **B1**: Đưa biểu thức trung tố vào xâu kí tự ***infix***
- **B2**: Tạo xâu kí tự mới để lưu dạng hậu tố ***postfix***
- **B3:** Sắp xếp:

  Đọc từng kí tự trong xâu ***infix***

  + Nếu kí tự là số thì lưu vào xâu ***postfix***
  + Nếu kí tự là toán tử đang xét có bậc cao hơn toán tử ở đỉnh ngăn xếp thì đẩy toán tử đang xét vào ***stack***.
  + Nếu toán tử đang xét có bậc bằng toán tử ở đỉnh ngăn xếp thì lấy toán tử ở đỉnh ngăn xếp ra khỏi ***stack*** và đẩy vào ***postfix***, sau đó đẩy toán tử đang xét vào đỉnh ngăn xếp.
  + Nếu toán tử đang xét có bậc nhỏ hơn toán tử ở đỉnh ngăn xếp thì lấy toán tử đang xét và xếp vào ***postfix***.

- B4: Thực hiện bước 3 cho đến khi gặp kí tự kết thúc của ***infix***, ta được biểu thức hậu tố ở ***postfix***

2, Tính giá trị của biểu thức vừa nhập bằng biểu thức hậu tố

- **B1:** Tạo ngăn xếp mới
- **B2:** Lấy từng phần tử trong biểu thức hậu tố ***(postfix***) ra
  + Nếu phần tử là toán hạng thì đưa vào ngăn xếp

+ Nếu phần tử là toán tử thì lấy 2 toán hạng ở đỉnh ngăn xếp ra, sau đó tính giá trị của 2 phần tử đó thông qua toán tử đang xét, được kết quả lại đẩy vào đỉnh ngăn xếp.

- **B3:** Thực hiện bước 2 cho đến khi kết thúc biểu thức hậu tố và kết quả là phần tử giá trị duy nhất trong ngăn xếp.

II, Mã nguồn

-Xem ở trong file đã nộp kèm: ***n05_g06_HoaDucViet_MaiHoangViet.asm***

Trung Tố - Hậu Tố

```
.data
infix:          .space  256
postfix:        .space  256
stack:          .space  256
notification:   .asciiz "Enter String infix\n(Note) Input expression has number must be integer and positive number beetwen 0 and 99:"
string:  .asciiz "\n"
message_postfix:.asciiz "Postfix is: "
message_result: .asciiz "Result is: "
message_infix:      .asciiz "Infix is: "
message_error1:     .asciiz "Error! \n Bieu thuc khong thoa man, Vui long nhap lai"
message_error2:     .asciiz "So nguyen nam ngoai khoang 0-99.\n Vui long nhap lai"

.text
#Nhập và hiển thị chuỗi trung tố
                li      $v0, 54
                la      $a0, notification
                la      $a1, infix
                la      $a2, 256
                syscall
                la      $a0, message_infix
                li      $v0, 4
                syscall
                la      $a0, infix
                li      $v0, 4
                syscall

# Chuyển trung tố sang hậu tố
```

```
        li      $s6, -1             # counter
        li      $s7, -1             # Scounter
        li      $t7, -1             # Pcounter
loop:
    la      $s1, infix          # $s1 = infix
    la      $t5, postfix        # $t5 = postfix
    la      $t6, stack          # $t6 = stack
        addi    $s6, $s6, 1         # counter ++
        add     $s1, $s1, $s6
        lb      $t1, 0($s1)         # t1 = infix[counter]
        beq     $t1, '+', operator
        nop
        beq     $t1, '-', operator
        nop
        beq     $t1, '*', operator
        nop
        beq     $t1, '/', operator
        nop
        beq     $t1, '\n', n_operator
        nop
        beq     $t1, ' ', n_operator
        nop
        beq     $t1, $zero, end_loop
        nop
        beq     $t1, '=', case_special  #Kiem tra cac ki tu dac biet
        nop
        beq     $t1, '.', case_special
        nop
        beq     $t1, ',', case_special
        nop
        j       next        # Neu khong thuoc truong hop dac biet thi chuyen
xuong next de bo qua bao loi
    case_special:
        j       noti_error
    noti_error:
        li      $v0, 55
        la      $a0, message_error1
        li      $a1, 0
        syscall
    next:
```

```
# push so tu infix
        addi    $t7, $t7, 1
        add     $t5, $t5, $t7
        sb      $t1, 0($t5)
        lb      $a0, 1($s1)
        jal     check_number
        beq     $v0, 1, n_operator
        nop

add_space:
        add     $t1, $zero, ' '
        sb      $t1, 1($t5)
        addi    $t7, $t7, 1
        j       n_operator
        nop
operator:
# add to stack
        beq     $s7, -1, pushToStack
        nop
        add     $t6, $t6, $s7
        lb      $t2, 0($t6)             # t2 = value of stack[counter]
# check t1 precedence
        beq     $t1, '+', equal1
        nop
        beq     $t1, '-', equal1
        nop
        li      $t3, 2
        j check_t2
        nop
equal1:
        li      $t3, 1

# check t2 precedence
check_t2:
        beq     $t2, '+', equal2
        nop
        beq     $t2, '-', equal2
        nop
        li      $t4, 2
        j       compare_precedence
```

```
                nop
        equal2:
                li      $t4, 1
compare_precedence:
                beq     $t3, $t4, equal_precedence
                nop
                slt     $s1, $t3, $t4
                beqz    $s1, t3_large_t4
                nop
# t3 < t4
# pop t2 from stack  and t2 ==> postfix
# get new top stack do again
                sb      $zero, 0($t6)
                addi    $s7, $s7, -1            # scounter ++
                addi    $t6, $t6, -1
                la      $t5, postfix            # $t5 = postfix
                addi    $t7, $t7, 1
                add     $t5, $t5, $t7
                sb      $t2, 0($t5)
                j       operator
                nop
t3_large_t4:
# push t1 to stack
                j       pushToStack
                nop
equal_precedence:
# pop t2  from stack  and t2 ==> postfix
# push to stack                 sb      $zero, 0($t6)
                addi    $s7, $s7, -1            # scounter ++
                addi    $t6, $t6, -1
                la      $t5, postfix            # postfix = $t5
                addi    $t7, $t7, 1             # pcounter ++
                add     $t5, $t5, $t7
                sb      $t2, 0($t5)
                j       pushToStack
                noppushToStack:
                la      $t6, stack              # stack = $t6
                addi    $s7, $s7, 1             # scounter ++
                add     $t6, $t6, $s7
                sb      $t1, 0($t6)
```

```
n_operator:                          # Khi gap 1 so ki tu dac biet se bo qua va sang
ki tu tiep theo
                j       loop
                nop
end_loop:       addi    $s1, $zero, 32
                add     $t7, $t7, 1
                add     $t5, $t5, $t7
                la      $t6, stack
                add     $t6, $t6, $s7
popallstack:
                lb      $t2, 0($t6)          # t2 = value of stack[counter]
                beq     $t2, 0, endPostfix
                sb      $zero, 0($t6)
                addi    $s7, $s7, -2
                add     $t6, $t6, $s7
                sb      $t2, 0($t5)
                add     $t5, $t5, 1
                j popallstack
                nop
endPostfix:
# Hiển thị chuỗi hậu tố
                la      $a0, message_postfix
                li      $v0, 4
                syscall
                la      $a0, postfix
                li      $v0, 4
                syscall
                la      $a0, string
                li      $v0, 4
                syscall
#Calculater
                li      $s3, 0               # counter
                la      $s2, stack           # $s2 = stack
# postfix to stack
                loop_post_to_stack:
                la      $s1, postfix         #$s1 = postfix
                add     $s1, $s1, $s3
                lb      $t1, 0($s1)

                beqz    $t1 end_loop_post_stack
```

```
                nop
                add     $a0, $zero, $t1
                jal     check_number
                nop
                beqz    $v0, is_operator
                nop
                jal     add_number_to_stack
                nop
                j       continue
                nop
is_operator:    jal     pop
                nop
                add     $a1, $zero, $v0
                jal     pop
                nop
                add     $a0, $zero, $v0
                add     $a2, $zero, $t1
                jal caculate
continue:       add     $s3, $s3, 1             # counter++
                j       loop_post_to_stack
                nop
caculate:
                sw      $ra, 0($sp)
                li      $v0, 0
                beq     $t1, '*', case_mul
                nop
                beq     $t1, '/', case_div
                nop
                beq     $t1, '+', case_plus
                nop
                beq     $t1, '-', case_sub
        case_mul:
                mul     $v0, $a0, $a1
                j       cal_push
        case_div:
                div     $a0, $a1
                mflo    $v0
                j       cal_push
        case_plus:
                add     $v0, $a0, $a1
```

```
                j       cal_push
        case_sub:
                sub     $v0, $a0, $a1
                j       cal_push
        cal_push:
                add     $a0, $v0, $zero
                jal     push
                nop
                lw      $ra, 0($sp)
                jr      $ra
                nop
# $s3 : counter for postfix string
# $s1 : postfix string
# $t1 : current value
add_number_to_stack:
                #backup $ra
                sw      $ra, 0($sp)
                li      $v0, 0
        loop_adds:
                beq     $t1, '0', case_0
                nop
                beq     $t1, '1', case_1
                nop
                beq     $t1, '2', case_2
                nop
                beq     $t1, '3', case_3
                nop
                beq     $t1, '4', case_4
                nop
                beq     $t1, '5', case_5
                nop
                beq     $t1, '6', case_6
                nop
                beq     $t1, '7', case_7
                nop
                beq     $t1, '8', case_8
                nop
                beq     $t1, '9', case_9
                nop
        case_0:         j       store_stack
```

```
case_1:         addi    $v0, $v0, 1
        j       store_stack
        nop
case_2:         addi    $v0, $v0, 2
        j       store_stack
        nop
case_3:         addi     $v0, $v0, 3
        j       store_stack
        nop
case_4:         addi    $v0, $v0, 4
        j       store_stack
        nop
case_5:         addi    $v0, $v0, 5
        j       store_stack
        nop
case_6:         addi    $v0, $v0, 6
        j       store_stack
        nop
case_7:         addi    $v0, $v0, 7
        j       store_stack
                nop
case_8:         addi    $v0, $v0, 8
        j       store_stack
        nop
case_9:         addi    $v0, $v0, 9
        j       store_stack
                nop
store_stack:
        add     $s3, $s3, 1                     # counter++
        la      $s1, postfix                    # $s1 = postfix
        add     $s1, $s1, $s3
        lb      $t1, 0($s1)
        beq     $t1, $zero, end_loop_adds
        beq     $t1, ' ', end_loop_adds
        mul      $v0, $v0, 10
        j       loop_adds
end_loop_adds:
        add     $a0, $zero, $v0
        slti    $t0, $v0, 100                   # Neu so nam ngoai 0-99 thi bao loi
        beq     $t0, 1, tieptheo
```

```
            li      $v0, 55
            la      $a0, message_error2
            li      $a1, 0
            syscall
        tieptheo:
            jal     push
            # restore $ra
            lw      $ra, 0($sp)
            jr      $ra
            nop
        check_number:
            li      $t8, '0'
            li      $t9, '9'
            beq     $t8, $a0, check_number_true
            beq     $t9, $a0, check_number_true
            slt     $v0, $t8, $a0
            beqz    $v0, check_number_false

            slt     $v0, $a0, $t9
            beqz    $v0, check_number_false
        check_number_true:
            li      $v0, 1
            jr      $ra
            nop
        check_number_false:
            li      $v0, 0
            jr      $ra
            nop
pop:        lw      $v0, -4($s2)
            sw      $zero, -4($s2)
            add     $s2, $s2, -4
            jr      $ra
            nop
push:       sw      $a0, 0($s2)
            add     $s2, $s2, 4
            jr      $ra
            nop
end_loop_post_stack:                    # print postfix
            la      $a0, message_result
            li      $v0, 4
```

```
syscall
jal     pop
add     $a0, $zero, $v0
li      $v0, 1
syscall
la      $a0, string
li      $v0, 4
syscall
```

III, Phân tích

-Về cơ bản sau khi nhập xong biểu thức trung tố vào xâu infix, thì sẽ được kiểm tra điều kiện xem có thoả mãn hay không.

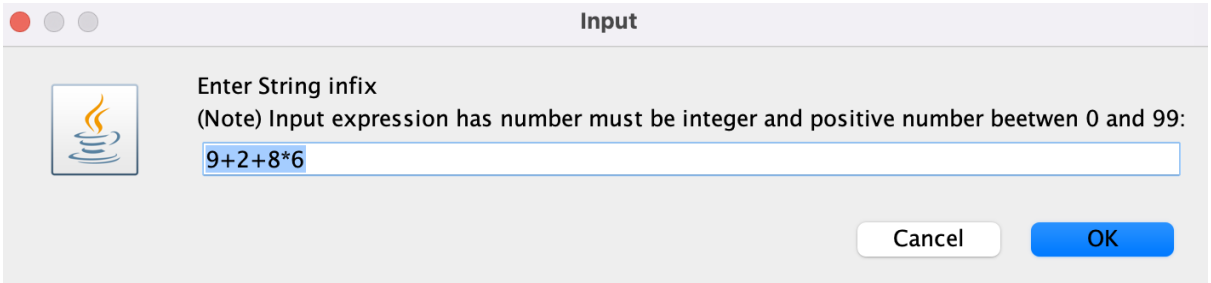
+Nếu trong biểu thức có các kí tự đặc biệt không thỏa mãn đề bài như: **, . =** thì sẽ hiển thị thông báo lỗi và kết thúc chương trình.

+ Nếu trong biểu thức có các số nằm ngoài vùng ***0-99*** như đề bài yêu cầu thì sẽ kết hiển thị thông báo lỗi số hạng và kết thúc chương trình.

+ Nếu trong biểu thức tồn tại các kí tự như khoảng trắng hay ***'\n'*** thì chương trình sẽ bỏ qua và xét tiếp đến ký tự tiếp theo của biểu thức nhập vào.

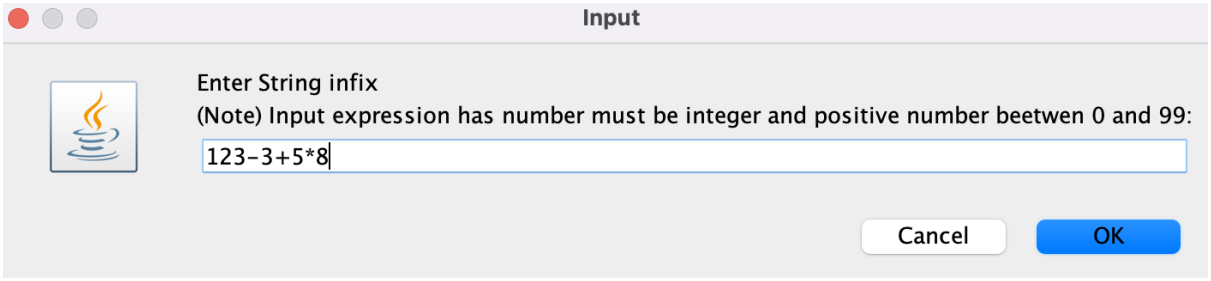
IV, Hình ảnh minh hoạ

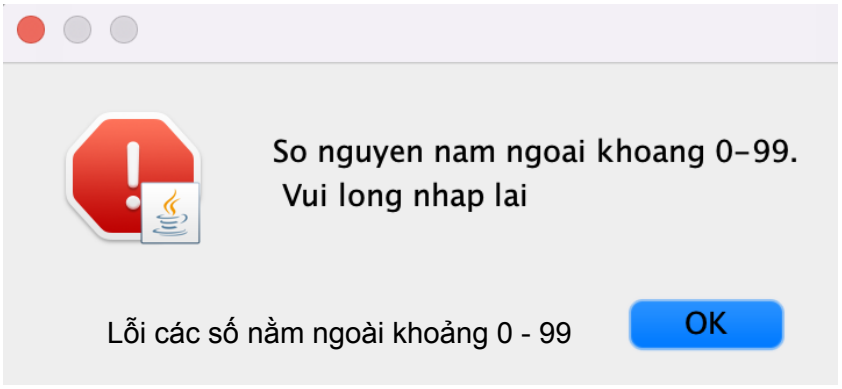
Minh hoạ biểu thức:

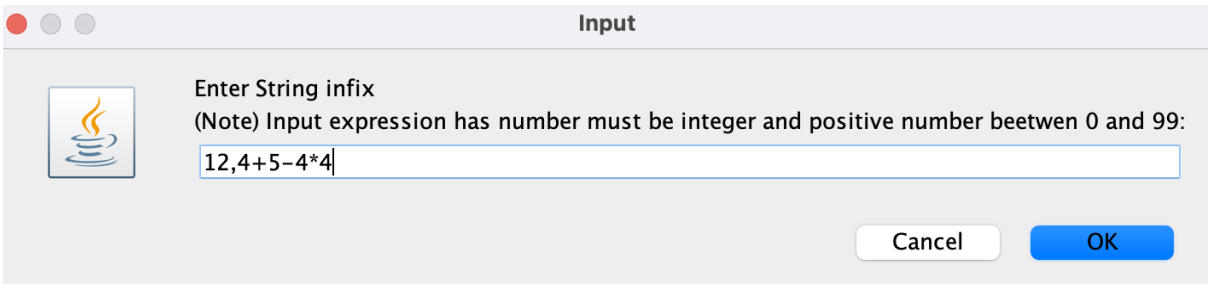


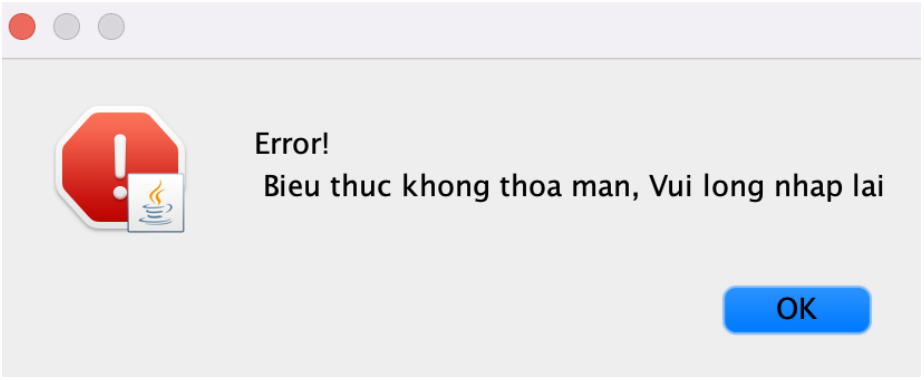

```
Infix is: 9+2+8*6
Postfix is: 9 2 +8 6 *+
Result is: 59

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

Thông báo lỗi:









Lỗi không phải là số nguyên (hay có kí tự đặc biệt ‘ , ‘)