BÁO CÁO CUỐI KÌ
MÔN THỰC HÀNH KIẾN TRÚC MÁY TÍNH

Họ và tên: Nguyễn Khánh Toàn

MSSV: 20225936

Bài 5:

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

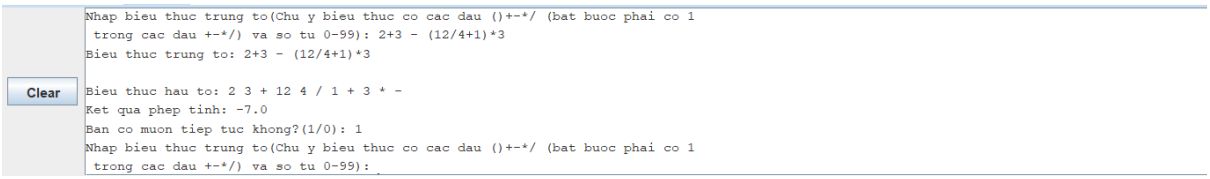
2. In ra biểu thức ở dạng hậu tố, ví dụ: 9 2 + 8 6 * +

3. Tính ra giá trị của biểu thức vừa nhập

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

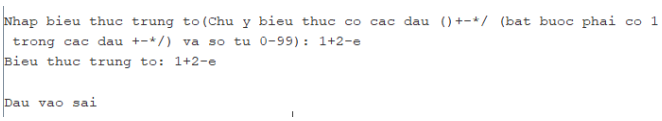
Toán tử bao gồm phép cộng, trừ, nhân, chia lấy thương

Chạy thử chương trình:

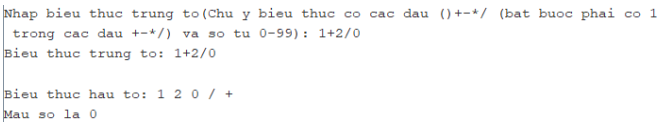


Với các trường hợp đặc biệt:

- Nhập kí tự đặc biệt không phải là chữ số hay toán tử



- Khi gặp một phép chia cho 0 (không thỏa mãn)



- Khi gặp các phép tính không thỏa mãn, không hợp lệ

```
Nhap bieu thuc trung to(Chu y bieu thuc co cac dau ()+-*/ (bat buoc phai co 1
 trong cac dau +-*/) va so tu 0-99): )1-2+3
Bieu thuc trung to: )1-2+3

Dau vao sai
```

```
Nhap bieu thuc trung to(Chu y bieu thuc co cac dau ()+-*/ (bat buoc phai co 1
 trong cac dau +-*/) va so tu 0-99): 1-2+
Bieu thuc trung to: 1-2+

Dau vao sai
```

Giải thích :

- Cách chuyển biểu thức trung tố sang hậu tố
  - Stack (ngăn xếp) dùng để lưu trữ các toán tử
  - Output List (danh sách kết quả) dùng để lưu trữ biểu thức hậu tố
  - Nếu là toán hạng (số hoặc biến): Thêm vào danh sách kết quả.
  - Nếu là dấu ngoặc mở "(": Đẩy vào stack.
  - Nếu là dấu ngoặc đóng ")":
    - Lấy các toán tử từ stack và thêm vào danh sách kết quả cho đến khi gặp dấu ngoặc mở.
    - Bỏ dấu ngoặc mở khỏi stack.
  - Nếu là toán tử:
    - Độ ưu toán tử “*” = “/” > “+” = “-“
    - Trong khi stack không rỗng và toán tử ở đỉnh stack có độ ưu tiên lớn hơn hoặc bằng toán tử hiện tại, lấy toán tử từ stack và thêm vào danh sách kết quả.
    - Đẩy toán tử hiện tại vào stack.
  - Khi đã duyệt hết biểu thức trung tố, lấy tất cả các toán tử còn lại trong stack và thêm vào danh sách kết quả.

- Cách tính toán biểu thức hậu tố:
  - Lấy hai toán hạng từ đỉnh stack (toán hạng 2 được lấy trước, toán hạng 1 lấy sau)
  - Thực hiện phép tính với toán hạng 1 và toán hạng 2
  - Đẩy kết quả phép tính vào stack.
  - Sau khi duyệt hết biểu thức, giá trị trong stack chính là kết quả của biểu thức hậu tố

- **Giải thích code:**

Khai báo các câu prompt, bộ nhớ để chứa các biểu thức trung tố, hậu tố và toán tử

```
.data
        infix: .space 256
        postfix: .space 256
        operator: .space 256
        confirmMsg: .asciiz "Ban co muon tiep tuc khong?(1/0): " # Tin nhắn hỏi người dùng có muốn tiếp tục không
        endMsg: .asciiz "Ket thuc chuong trinh" # Tin nhắn kết thúc chương trình
        errorMsg: .asciiz "Dau vao sai\n" # Tin nhắn lỗi nhập liệu
        startMsg: .asciiz "Nhap bieu thuc trung to(Chu y bieu thuc co cac dau ()+-*/ (bat buoc phai co 1\n trong cac dau +-*/) va so tu 0-99): " #
        prompt_postfix: .asciiz "Bieu thuc hau to: " # Tin nhắn biểu thức postfix
        prompt_result: .asciiz "Ket qua phep tinh: " # Tin nhắn kết quả
        prompt_infix: .asciiz "Bieu thuc trung to: " # Tin nhắn biểu thức infix:
        converter: .word 1 # khai báo biến chuyển đổi có giá trị là 1
        zeroMsg: .asciiz "Mau so la 0\n" # Tin nhắn cảnh báo mẫu số = 0
        temp: .word 1
        stack: .float
```

In cậu lệnh yêu cầu người dùng nhập biểu thức trung tố và in nó ra

```
main:
        li $v0, 4 # in câu mở
        la $a0, startMsg
        syscall

        li $v0, 8
        la $a0, infix
        li $a1, 256
        syscall  #enter infix

        li $v0, 4 # in chuỗi trung tố
        la $a0, prompt_infix
        syscall
        li $v0, 4
        la $a0, infix
        syscall
        li $v0, 11
        li $a0, '\n'
        syscall
```

Định nghĩa sẵn các thanh ghi là giá trị các tham số cần sử dụng trong thuật toán

```
#Khai báo siêu tham số
        li $s6,0                # 0 : none 1 : số 2 :toan tu 3 : ( 4 : )
        li $t9,0                # Đếm số chữ số
        li $t5,-1               # index chuỗi hậu tố
        li $t6,-1               # index của ngăn xếp
        la $t1, infix           # Địa chỉ hiện tại của biểu thức trung tố
        la $t2, postfix         # Địa chỉ hiện tại của biểu thức hậu tố
        la $t3, operator        # Địa chỉ của ngăn xếp lưu toán tử
        li $s6,0                # 0 : none 1 : số 2 :toan tu 3 : ( 4 : )
        addi $t1,$t1,-1         # Vị trí scan đến của biểu thức trung tố
```

ScanInfix thực hiện việc đọc từng kí tự từ chuỗi nhập vào và chuyển nó sang biểu thức hậu tố, đồng thời kiểm tra xem trong chuỗi nhập vào có bị lỗi không để thực hiện thông báo và kết thúc chương trình

```
54  #---------------------------------------------------------------
55  scanInfix:
56          # Kiểm tra tất cả các tùy chọn đầu vào hợp lệ
57          addi $t1,$t1,1                          # Tăng vị trí của biểu thức tru
58          lb $t4, 0($t1)                          # Load ký tự hiện tại trong bi
59          j checkStatus
60          continueScan:
61          j processOp
```

Ở checkStatus trong cụm lệnh có nhãn ScanInfix, ta kiểm tra kí tự mới đọc. Nếu kí tự đó là dấu cách thì đọc tiếp kí tự tiếp theo, nếu đó là dấu xuống dòng chứng tỏ đã đọc hết chuỗi nhảy sang nhãn ‘endReadInfix’. Thanh ghi $t9 có nghĩa là số chữ số gần nhất của tố hiện tại đang duyệt (để tránh trường hợp người dùng nhập quá 2 chữ số)

```
383 checkStatus:
384         beq $t4, ' ', scanInfix         # Nếu là dấu cách, bỏ qua và quét tiếp
385         beq $t4, '\n', endReadInfix                 # Quét kết thúc đầu vào
386         beq $t9,0,readOne                       # Nếu t9 = 0 => k phải là chữ s
387         beq $t9,1,readTwo                       # Nếu t9 = 1 => số có 1 chữ số
388         beq $t9,2,readFail                      # Nếu t9 = 2 => số có 2 chữ số
```

readOne và readTwo để lưu lại số mới được duyệt vàotrong bộ nhớ tạm. Nếu thành ghi $t9 đang có giá trị là 0 có nghĩa chưa đọc được số nào trong biểu thức trung tố đang được lưu, nếu là 1 có nghĩa 1 chữ số đang ở bộ nhớ tạm và đang kiểm tra xem chữ số đó trong số nào không. Nếu $t9 có giá trị 2 và chương trình kiểm tra có thêm 1 chữ số được đọc thì chuyển đến nhãn wrongInput. Các câu lệnh tại nhãn numberToPost có tác dụng đẩy 1 số vào biểu thức hậu tố

```
"
readOne: # nếu quét chữ số đầu tiên
        # Đặt biến để duyệt các ký tự từ '0' đến '9'
        li $s0, 48          # Mã ASCII của '0'
        li $s1, 58          # Mã ASCII của '9' + 1

loopCheckChar1:
        beq $s0, $s1, continueScan  # Nếu $t0 đạt tới '9' + 1 thì kết thúc vòng lặp
        beq $t4, $s0, storeOne # So sánh $t4 với ký tự hiện tại
        addi $s0, $s0, 1       # Tăng ký tự lên 1
        j loopCheckChar1       # Quay lại đầu vòng lặp check
```

```
readTwo: # nếu quét chữ số thứ hai
        # Đặt biến để duyệt các ký tự từ '0' đến '9'
        li $s0, 48            # Mã ASCII của '0'
        li $s1, 58            # Mã ASCII của '9' + 1

loopCheckChar2:
        beq $s0, $s1, out_loop2  # Nếu $t0 đạt tới '9' + 1 thì kết thúc vòng
        beq $t4, $s0, storeTwo # So sánh $t4 với ký tự hiện tại
        addi $s0, $s0, 1       # Tăng ký tự lên 1
        j loopCheckChar2       # Quay lại đầu vòng lặp check
        # Nếu không nhận được chữ số thứ hai
        out_loop2: jal numberToPost
        j continueScan
"
#-----------------------------------------------------
readFail: # nếu quét chữ số thứ ba
        # Đặt biến để duyệt các ký tự từ '0' đến '9'
        li $s0, 48            # Mã ASCII của '0'
        li $s1, 58            # Mã ASCII của '9' + 1

loopCheckChar3:
        beq $s0, $s1, out_loop3  # Nếu $t0 đạt tới '9' + 1 thì kết thúc vòng
        beq $t4, $s0, wrongInput # So sánh $t4 với ký tự hiện tại
        addi $s0, $s0, 1       # Tăng ký tự lên 1
        j loopCheckChar3       # Quay lại đầu vòng lặp check
        # Nếu không nhận được chữ số thứ ba
        out_loop3: jal numberToPost
        j continueScan
```

Tiếp đó chương trình sẽ duyệt xem nếu không phải chữ số thì kí tự có nằm trong các kí tự được quy định không để xử lý theo thuật toán. Các hàm xử lí toán tử sẽ bao gồm việc kiểm tra xem biểu thức nhập vào có hợp lí để tính toán không

```
processOp:
        beq $t4, '+', plusMinus        # Nếu là dấu cộng => xử lý
        beq $t4, '-', plusMinus        # Nếu là dấu trừ => xử lý
        beq $t4, '*', multiplyDivide   # Nếu là dấu nhân => xử lý
        beq $t4, '/', multiplyDivide   # Nếu là dấu chia => xử lý
        beq $t4, '(', openBracket      # Nếu là dấu ngoặc mở => xử lý
        beq $t4, ')', closeBracket     # Nếu là dấu ngoặc đóng => xử lý
#
```

Nếu kiểm tra hết kí tự không thuộc bất kì kí tự nào được quy định sẵn thì chương trình báo lỗi

```
..
wrongInput: # Hiển thị thông báo khi phát hiện trường hợp đầu vào không hợp lệ
        li $v0, 4
        la $a0, errorMsg
        syscall
        j confirm
```

Sau khi in xong chương trình in ra chuỗi hậu tố đã đọc

```
finishScan: # In biểu thức hậu tố
        li $v0, 4
        la $a0, prompt_postfix
        syscall
        li $t6,-1 # Đặt giá trị hiện tại của vị trí biểu thức hậu tố là -1
#-------------------------------------------------------------
#Quy trình printPost
#@brief: Lặp và in từng ký tự một cho đến khi kết thúc biểu thức hậu tố
#@param[in] $t6: Địa chỉ của biểu thức hậu tố
#-------------------------------------------------------------
        printPost:
                addi $t6,$t6,1          # Tăng giá trị hiện tại của vị trí biểu thức hậu tố
                add $t8,$t2,$t6         # Tải địa chỉ của biểu thức hậu tố hiện tại
                lbu $t7,($t8)           # Tải giá trị của biểu thức hậu tố hiện tại
                bgt $t6,$t5,finishPrint# In toàn bộ biểu thức hậu tố --> tính toán
                bgt $t7,99,printOp      # Nếu biểu thức hậu tố hiện tại > 99 --> một toán tử. Nếu không thì biểu thức hậu tố hiện tại là một
                                        #tử lớn hơn 100
                li $v0, 1
                add $a0,$t7,$zero
                syscall
                li $v0, 11
                li $a0, ' '
                syscall
                j printPost             # Lặp lại
                printOp:
                li $v0, 11
                addi $t7,$t7,-100       # Giải mã toán tử
                add $a0,$t7,$zero
                syscall
                li $v0, 11  # in dấu cách
                li $a0, ' '
                syscall
                j printPost             # Lặp lại
        finishPrint:
```

Thực hiện tính toán biểu thức hậu tố và in ra giá trị kết quả. Thực hiện phép toán với các lệnh thực hiện các phép toán trên các số thực dấu chấm động theo chuẩn IEEE 754

```
#-------------------------------------------------------------
calPost:
       addi $t6,$t6,1          # Tăng giá trị hiện tại của vị trí biểu thức hậu tố
       add $t8,$t2,$t6         # Tải địa chỉ của biểu thức hậu tố hiện tại
       lbu $t7,($t8)           # Tải giá trị của biểu thức hậu tố hiện tại
       bgt $t6,$t5,printResult# Tính toán cho toàn bộ biểu thức hậu tố --> in kết quả
       bgt $t7,99,calculate    # Nếu biểu thức hậu tố hiện tại > 99 --> một toán tử --> đẩy ra 2 số để tính toán
       # Nếu không thì biểu thức hậu tố hiện tại là một số
       addi $t9,$t9,4          # Tăng giá trị hiện tại của đỉnh ngăn xếp
       add $t4,$t3,$t9         # Tải địa chỉ của đỉnh ngăn xếp hiện tại
       sw $t7,temp
       l.s $f10,temp   # Tải số vào coprocl để chuyển đổi thành số thực
       div.s $f10,$f10,$f0
       s.s $f10, 0($t4)              # Đẩy số vào ngăn xếp
       sub.s $f10,$f10,$f10    # Đặt lại f10 = 0
       # đảm bảo rằng các thanh ghi không chứa giá trị cũ khi thực hiện các phép toán tiếp theo
       j calPost               # Lặp lại
       #-------------------------------------------------------------
       #procedure calculate
       #@brief: Tính toán khi phát hiện + - * /
       #@param[in] $t4: Địa chỉ hiện tại của đỉnh ngăn xếp
       #@param[in] $t7: Giá trị của biểu thức hậu tố hiện tại
       #@param[out] $f1: Kết quả
       #-------------------------------------------------------------
```

```
calculate:
        # Lấy 1 số ra khỏi ngăn xếp
        add $t4,$t3,$t9
        l.s $f3,($t4) #Lấy một số từ đỉnh ngăn xếp vào thanh ghi float $f3
        # Lấy số tiếp theo ra khỏi ngăn xếp
        addi $t9,$t9,-4
        add $t4,$t3,$t9
        l.s $f2,($t4) #Lấy một số từ đỉnh ngăn xếp vào thanh ghi float $f2
        # Giải mã toán tử
        beq $t7,143,plus
        beq $t7,145,minus
        beq $t7,142,multiply
        beq $t7,147,divide
```

SOURCE CODE:

.data

infix: .space 256

postfix: .space 256

operator: .space 256

confirmMsg: .asciiz "Ban co muon tiep tuc khong?(1/0): " # Tin nhắn hỏi người dùng có muốn tiếp tục không

endMsg: .asciiz "\nKet thuc chuong trinh" # Tin nhắn kết thúc chương trình

errorMsg: .asciiz "Dau vao sai\n" # Tin nhắn lỗi nhập liệu

startMsg: .asciiz "Nhap bieu thuc trung to(Chu y bieu thuc co cac dau ()+-*/ (bat buoc phai co 1\n trong cac dau +-*/) va so tu 0-99): " # Tin nhắn bắt đầu chương trình

prompt_postfix: .asciiz "Bieu thuc hau to: " # Tin nhắn biểu thức postfix

prompt_result: .asciiz "Ket qua phep tinh: " # Tin nhắn kết quả

prompt_infix: .asciiz "Bieu thuc trung to: " # Tin nhắn biểu thức infix:

converter: .word 1 # khai báo biến chuyển đổi có giá trị là 1

zeroMsg: .asciiz "Mau so la 0\n" # Tin nhắn cảnh báo mẫu số = 0

temp: .word 1

stack: .float

.text

main:

li $v0, 4 # in câu mở

```
        la $a0, startMsg
        syscall

        li $v0, 8
        la $a0, infix
        li $a1, 256
        syscall  #enter infix

        li $v0, 4 # in chuỗi trung tố
        la $a0, prompt_infix
        syscall
        li $v0, 4
        la $a0, infix
        syscall
        li $v0, 11
        li $a0, '\n'
        syscall
#-----------------------------------------------------------------
#Khai báo siêu tham số
        li $s6,0        # 0 : none 1 : số 2 :toan tu 3 : ( 4 : )
        li $t9,0        # Đếm số chữ số
        li $t5,-1       # index chuỗi hậu tố
        li $t6,-1       # index của ngăn xếp
        la $t1, infix           # Địa chỉ hiện tại của biểu thức trung tố
        la $t2, postfix  # Địa chỉ hiện tại của biểu thức hậu tố
        la $t3, operator  # Địa chỉ của ngăn xếp lưu toán tử
        li $s6,0        # 0 : none 1 : số 2 :toan tu 3 : ( 4 : )
        addi $t1,$t1,-1         # Vị trí scan đến của biểu thức trung tố

# ================== Chuyển thành biểu thức hậu tố  ==================
#-------------------------------------------------------------
#Quá trình scanInfix
```

```
#@brief: Lặp qua từng ký tự trong biểu thức trung tố và kiểm tra xem nó là toán tử hay toán hạng
#@param[in] $t1: Vị trí của biểu thức trung tố
#-------------------------------------------------------------
scanInfix:
    # Kiểm tra tất cả các tùy chọn đầu vào hợp lệ
    addi $t1,$t1,1                  # Tăng vị trí của biểu thức trung tố
    lb $t4, 0($t1)                  # Load ký tự hiện tại trong biểu thức trung tố
    j checkStatus
    continueScan:
    j processOp

#Sau khi check không có kĩ tự nào thuộc các kí tự trên thì kết thúc chương trình
finishScan: # In biểu thức hậu tố
    li $v0, 4
    la $a0, prompt_postfix
    syscall
    li $t6,-1 # Đặt giá trị hiện tại của vị trí biểu thức hậu tố là -1
#-------------------------------------------------------------
#Quy trình printPost
#@brief: Lặp và in từng ký tự một cho đến khi kết thúc biểu thức hậu tố
#@param[in] $t6: Địa chỉ của biểu thức hậu tố
#-------------------------------------------------------------
    printPost:
        addi $t6,$t6,1          # Tăng giá trị hiện tại của vị trí biểu thức hậu tố
        add $t8,$t2,$t6         # Tải địa chỉ của biểu thức hậu tố hiện tại
        lbu $t7,($t8)           # Tải giá trị của biểu thức hậu tố hiện tại
        bgt $t6,$t5,finishPrint # In toàn bộ biểu thức hậu tố --> tính toán
        bgt $t7,99,printOp      # Nếu biểu thức hậu tố hiện tại > 99 --> một toán tử. Nếu không thì biểu thức hậu tố hiện tại là một số (do đã có phép chuẩn hóa toán
                                #tử lớn hơn 100
        li $v0, 1
        add $a0,$t7,$zero
```

```
                syscall
                li $v0, 11
                li $a0, ' '
                syscall
                j printPost             # Lặp lại
                printOp:
                li $v0, 11
                addi $t7,$t7,-100       # Giải mã toán tử
                add $a0,$t7,$zero
                syscall
                li $v0, 11  # in dấu cách
                li $a0, ' '
                syscall
                j printPost             # Lặp lại
        finishPrint:
                li $v0, 11
                li $a0, '\n'
                syscall  # Xuống dòng


# ================== Tính toán biểu thức hậu tố ==================
        li $t9,-4       # Đặt giá trị hiện tại của vị trí đỉnh ngăn xếp là -4
        la $t3,stack            # Tải địa chỉ của ngăn xếp dùng để tính toán
        li $t6,-1       # Đặt giá trị hiện tại của vị trí biểu thức hậu tố là -1
        l.s $f0,converter# Tải giá trị của bộ chuyển đổi

#-------------------------------------------------------------
#procedure calPost
#@brief: Tính toán biểu thức hậu tố
#@param[in] $t6: Địa chỉ của biểu thức hậu tố
#-------------------------------------------------------------
calPost:
```

```
        addi $t6,$t6,1              # Tăng giá trị hiện tại của vị trí biểu thức hậu tố
        add $t8,$t2,$t6             # Tải địa chỉ của biểu thức hậu tố hiện tại
        lbu $t7,($t8)               # Tải giá trị của biểu thức hậu tố hiện tại
        bgt $t6,$t5,printResult     # Tính toán cho toàn bộ biểu thức hậu tố --> in kết quả
        bgt $t7,99,calculate        # Nếu biểu thức hậu tố hiện tại > 99 --> một toán tử --> đẩy ra 2 số để tính toán
        # Nếu không thì biểu thức hậu tố hiện tại là một số
        addi $t9,$t9,4              # Tăng giá trị hiện tại của đỉnh ngăn xếp
        add $t4,$t3,$t9             # Tải địa chỉ của đỉnh ngăn xếp hiện tại
        sw $t7,temp
        l.s $f10,temp     # Tải số vào coproc1 để chuyển đổi thành số thực
        div.s $f10,$f10,$f0
        s.s $f10, 0($t4)            # Đẩy số vào ngăn xếp
        sub.s $f10,$f10,$f10        # Đặt lại f10 = 0
        # đảm bảo rằng các thanh ghi không chứa giá trị cũ khi thực hiện các phép toán tiếp theo
        j calPost                   # Lặp lại
        #-------------------------------------------------------------
        #procedure calculate
        #@brief: Tính toán khi phát hiện + - * /
        #@param[in] $t4: Địa chỉ hiện tại của đỉnh ngăn xếp
        #@param[in] $t7: Giá trị của biểu thức hậu tố hiện tại
        #@param[out] $f1: Kết quả
        #-------------------------------------------------------------
calculate:
        # Lấy 1 số ra khỏi ngăn xếp
        add $t4,$t3,$t9
        l.s $f3,($t4) #Lấy một số từ đỉnh ngăn xếp vào thanh ghi float $f3
        # Lấy số tiếp theo ra khỏi ngăn xếp
        addi $t9,$t9,-4
        add $t4,$t3,$t9
        l.s $f2,($t4) #Lấy một số từ đỉnh ngăn xếp vào thanh ghi float $f2
        # Giải mã toán tử
```

```
        beq $t7,143,plus
        beq $t7,145,minus
        beq $t7,142,multiply
        beq $t7,147,divide
# Lam đúng các bước cứ duyệt đến dấu trong biểu thức hậu tố ta sẽ lấy 2 số phía trước để thực hiện phép toán
        plus:
                add.s $f1,$f2,$f3
                s.s $f1, 0($t4)
                sub.s $f2,$f2,$f2          # Đặt lại f2 và f3
                sub.s $f3,$f3,$f3
                j calPost
        minus:
                sub.s $f1,$f2,$f3
                s.s $f1, 0($t4)
                sub.s $f2,$f2,$f2          # Đặt lại f2 và f3
                sub.s $f3,$f3,$f3
                j calPost
        multiply:
                mul.s $f1,$f2,$f3
                s.s $f1, 0($t4)
                sub.s $f2,$f2,$f2          # Đặt lại f2 và f3
                sub.s $f3,$f3,$f3
                j calPost
        divide:
                # Kiểm tra nếu $f3 bằng 0
                c.eq.s $f3, $f10        # So sánh $f3 với $f0 (đặt $f0 là 0.0)
                bc1t handle_div_zero    # Nếu $f3 bằng 0, nhảy đến handle_div_zero

                div.s $f1,$f2,$f3
                s.s $f1, 0($t4)
                sub.s $f2,$f2,$f2          # Đặt lại f2 và f3
                sub.s $f3,$f3,$f3
```

```
                j calPost

printResult:
        li $v0, 4
        la $a0, prompt_result
        syscall
        li $v0, 2
        l.s $f12,0($t4)
        syscall
        li $v0, 11
        li $a0, '\n'
        syscall
confirm:                        # Hỏi người dùng có tiếp tục hay không
        li $v0, 4
        la $a0, confirmMsg
        syscall

        li $v0, 5
        syscall

        beq $v0, 1, main
# Kết thúc chương trình
end:
        li $v0, 4
        la $a0, endMsg
        syscall
        li $v0, 10
        syscall

# Sub program
#------------------------------------------------------------
#procedure endReadInfix
```

```
#@brief: Khi hoàn thành việc quét biểu thức infix, kiểm tra xem ngăn xếp có trống hay không
#@param[in]: $s6 Trạng thái quét
#@param[in]: $t5 Trạng thái quét
#-------------------------------------------------------------
endReadInfix:
        beq $s6,2,wrongInput                    # Kết thúc với một toán tử hoặc dấu ngoặc
mở
        beq $s6,3,wrongInput
        beq $t5,-1,wrongInput                   # Không có đầu vào
        j popAll
#-------------------------------------------------------------
# Với mỗi tùy chọn đầu vào hợp lệ
#-------------------------------------------------------------
readOne: # nếu quét chữ số đầu tiên
        # Đặt biến để duyệt các ký tự từ '0' đến '9'
        li $s0, 48          # Mã ASCII của '0'
        li $s1, 58          # Mã ASCII của '9' + 1

loopCheckChar1:
        beq $s0, $s1, continueScan  # Nếu $t0 đạt tới '9' + 1 thì kết thúc vòng lặp
        beq $t4, $s0, storeOne # So sánh $t4 với ký tự hiện tại
        addi $s0, $s0, 1       # Tăng ký tự lên 1
        j loopCheckChar1       # Quay lại đầu vòng lặp check

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

readTwo: # nếu quét chữ số thứ hai
        # Đặt biến để duyệt các ký tự từ '0' đến '9'
        li $s0, 48          # Mã ASCII của '0'
        li $s1, 58          # Mã ASCII của '9' + 1

loopCheckChar2:
        beq $s0, $s1, out_loop2  # Nếu $t0 đạt tới '9' + 1 thì kết thúc vòng lặp
```

```
        beq $t4, $s0, storeTwo # So sánh $t4 với ký tự hiện tại
        addi $s0, $s0, 1       # Tăng ký tự lên 1
        j loopCheckChar2       # Quay lại đầu vòng lặp check
        # Nếu không nhận được chữ số thứ hai
        out_loop2: jal numberToPost
        j continueScan
#-------------------------------------------------
readFail: # nếu quét chữ số thứ ba
        # Đặt biến để duyệt các ký tự từ '0' đến '9'
        li $s0, 48          # Mã ASCII của '0'
        li $s1, 58          # Mã ASCII của '9' + 1

loopCheckChar3:
        beq $s0, $s1, out_loop3  # Nếu $t0 đạt tới '9' + 1 thì kết thúc vòng lặp
        beq $t4, $s0, wrongInput # So sánh $t4 với ký tự hiện tại
        addi $s0, $s0, 1       # Tăng ký tự lên 1
        j loopCheckChar3       # Quay lại đầu vòng lặp check
        # Nếu không nhận được chữ số thứ ba
        out_loop3: jal numberToPost
        j continueScan
plusMinus:                     # Đầu vào là + -
        beq $s6,2,wrongInput           # Nhận phép toán sau phép toán hoặc dấu ngoặc mở
        beq $s6,3,wrongInput           # Nhận đầu vào ( thì sai
        beq $s6,0,wrongInput           # Nhận phép toán trước bất kỳ số nào
        li $s6,2                   # Thay đổi trạng thái đầu vào thành nhập toán tử
        continuePlusMinus:
        beq $t6,-1,inputToOp           # Không có gì trong ngăn xếp Operator --> đẩy vào
        jal loadTopStack
        beq $t7,'(',inputToOp          # Nếu đỉnh là ( --> đẩy vào
        beq $t7,'+',equalPrecedence    # Nếu đỉnh là + -
        beq $t7,'-',equalPrecedence
        beq $t7,'*',lowerPrecedence    # Nếu đỉnh là * /
```

```
        beq $t7,'/',lowerPrecedence
multiplyDivide:                 # Đầu vào là * /
        beq $s6,2,wrongInput            # Nhận phép toán sau phép toán hoặc dấu ngoặc mở
        beq $s6,3,wrongInput
        beq $s6,0,wrongInput            # Nhận phép toán trước bất kỳ số nào
        li $s6,2                # Thay đổi trạng thái đầu vào thành nhập toán tử
        beq $t6,-1,inputToOp            # Không có gì trong ngăn xếp Operator --> đẩy vào
        jal loadTopStack
        beq $t7,'(',inputToOp           # Nếu đỉnh là ( --> đẩy vào
        beq $t7,'+',inputToOp           # Nếu đỉnh là + - --> đẩy vào
        beq $t7,'-',inputToOp
        beq $t7,'*',equalPrecedence     # Nếu đỉnh là * /
        beq $t7,'/',equalPrecedence
openBracket:                    # Đầu vào là (
        beq $s6,1,wrongInput            # Nhận dấu ngoặc mở sau một số hoặc dấu ngoặc
đóng
        beq $s6,4,wrongInput
        li $s6,3                # Thay đổi trạng thái đầu vào thành 1
        j inputToOp
closeBracket:                   # Đầu vào là )
        beq $s6,2,wrongInput            # Nhận dấu ngoặc đóng sau một phép toán hoặc dấu
ngoặc đóng
        beq $s6,3,wrongInput
        li $s6,4
        add $t8,$t6,$t3                 # Load địa chỉ của phần tử Operator đỉnh
        lb $t7,($t8)                    # Load giá trị byte của phần tử Operator đỉnh
        beq $t7,'(',wrongInput          # Đầu vào chứa () mà không có gì giữa --> lỗi
        continueCloseBracket:
        beq $t6,-1,wrongInput           # Không tìm thấy dấu ngoặc mở --> lỗi
        jal loadTopStack
        beq $t7,'(',matchBracket# Tìm thấy dấu ngoặc mở phù hợp
        jal opToPostfix                 # Lấy phần tử Operator đỉnh và đẩy vào Postfix
```

```
        j continueCloseBracket          # Tiếp tục vòng lặp cho đến khi tìm thấy dấu ngoặc đóng phù hợp hoặc lỗi


# Hàm load dữ liệu đỉnh stack vào thanh ghi $t7
loadTopStack:
        add $t8,$t6,$t3                 # Load địa chỉ của phần tử Operator đỉnh
        lb $t7,($t8)                    # Load giá trị byte của phần tử Operator đỉnh
        jr $ra


# Độ ưu tiên giữa toán tử hiện tại và toán tử đỉnh là như nhau
equalPrecedence:        # Nghĩa là nhận + - và đỉnh là + - || nhận * / và đỉnh là * /
        jal opToPostfix                 # Lấy phần tử Operator đỉnh và đẩy vào Postfix
        j inputToOp                     # Đẩy phần tử Operator mới vào


# Độ ưu tiên toán tử hiện tại bé hơn toán tử đỉnh
lowerPrecedence:        # Nghĩa là nhận + - và đỉnh là * /
        jal opToPostfix                 # Lấy phần tử Operator đỉnh và đẩy vào Postfix
        j continuePlusMinus             # Lặp lại để nhỡ may còn các dấu khác có độ ưu tiên lớn hơn còn trong stack
#-------------------------------------------------------------
#-------------------------------------------------------------
inputToOp:              # Đẩy đầu vào vào ngăn xếp Operator
        add $t6,$t6,1                   # Tăng giá trị top của offset Operator
        add $t8,$t6,$t3                 # Load địa chỉ của phần tử Operator đỉnh ( do đầu vào bị giới hạn 99 nên chỉ cần mỗi ô nhớ stack 1 byte là đủ)
        sb $t4,($t8)                    # Lưu đầu vào vào Operator
        j scanInfix
opToPostfix:            # Lấy phần tử Operator đỉnh và đẩy vào Postfix
        addi $t5,$t5,1                  # Tăng giá trị top của offset Postfix
        add $t8,$t5,$t2                 # Load địa chỉ của phần tử Postfix đỉnh
        addi $t7,$t7,100        # Mã hóa phép toán + 100 ( để tránh việc nhầm với số bình thường )
        sb $t7,($t8)                    # Lưu phép toán vào Postfix
        addi $t6,$t6,-1                 # Giảm giá trị top của offset Operator
```

```
        jr $ra
matchBracket:                   # Bỏ qua một cặp dấu ngoặc phù hợp
        addi $t6,$t6,-1                 # Giảm giá trị top của offset Operator
        j scanInfix
popAll:                         # Lấy tất cả các phần tử Operator và đẩy vào Postfix
        jal numberToPost                #Check đẩy số trước vì có trường hợp kí tự cuối
cùng của chuỗi trung tố là số
        beq $t6,-1,finishScan           # Operator trống --> kết thúc
        jal loadTopStack
        beq $t7,'(',wrongInput          # Dấu ngoặc không phù hợp --> lỗi
        beq $t7,')',wrongInput
        jal opToPostfix
        j popAll                                        # Lặp lại cho đến khi Operator trống
#-------------------------------------------------------------
# Đổi chữ số thành số
#-------------------------------------------------------------
storeOne:
        beq $s6,4,wrongInput            # Nhận chữ số sau dấu ngoặc đóng )
        addi $s4,$t4,-48        # Lưu chữ số đầu tiên vào biến số ( $s4 để tính và lưu số )
        add $t9,$zero,1                 # Thay đổi trạng thái thành 1 chữ số
        li $s6,1
        j scanInfix
storeTwo:
        beq $s6,4,wrongInput            # Nhận chữ số sau dấu ngoặc đóng )
        addi $s5,$t4,-48        # Lưu chữ số thứ hai vào biến số
        mul $s4,$s4,10
        add $s4,$s4,$s5                 # Số được lưu = chữ số đầu * 10 + chữ số thứ hai
        add $t9,$zero,2                 # Thay đổi trạng thái thành 2 chữ số
        li $s6,1
        j scanInfix
#-------------------------------------------------------------
# Thêm số vào chuỗi hậu tố
#-------------------------------------------------------------
```

```
numberToPost:
        beq $t9,0,endnumberToPost # rào để khi popAll không bị đẩy liên tục
        addi $t5,$t5,1
        add $t8,$t5,$t2
        sb $s4,($t8)                    # Lưu số vào Postfix
        add $t9,$zero,$zero             # Thay đổi trạng thái thành 0 chữ số
        endnumberToPost:
        jr $ra
#-------------------------------------------------------------
# Xử lí mẫu số bằng 0
#-------------------------------------------------------------
handle_div_zero:
      li $v0, 4
        la $a0, zeroMsg
        syscall
        j confirm
#-------------------------------------------------------------
#Xử lí đọc đầu vào
#-------------------------------------------------------------


checkStatus:
        beq $t4, ' ', scanInfix         # Nếu là dấu cách, bỏ qua và quét tiếp
        beq $t4, '\n', endReadInfix                     # Quét kết thúc đầu vào --> đẩy tất
cả toán tử vào biểu thức hậu tố
        beq $t9,0,readOne                   # Nếu t9 = 0 => k phải là chữ số
        beq $t9,1,readTwo                   # Nếu t9 = 1 => số có 1 chữ số
        beq $t9,2,readFail                  # Nếu t9 = 2 => số có 2 chữ số


processOp:
        beq $t4, '+', plusMinus         # Nếu là dấu cộng => xử lý
        beq $t4, '-', plusMinus         # Nếu là dấu trừ => xử lý
        beq $t4, '*', multiplyDivide    # Nếu là dấu nhân => xử lý
        beq $t4, '/', multiplyDivide    # Nếu là dấu chia => xử lý
```

```
        beq $t4, '(', openBracket # Nếu là dấu ngoặc mở => xử lý
        beq $t4, ')', closeBracket          # Nếu là dấu ngoặc đóng => xử lý
#-------------------------------------------------------------
# Kết thúc chương trình khi gặp sự cố
#-------------------------------------------------------------
wrongInput: # Hiển thị thông báo khi phát hiện trường hợp đầu vào không hợp lệ
        li $v0, 4
        la $a0, errorMsg
        syscall
        j confirm
```