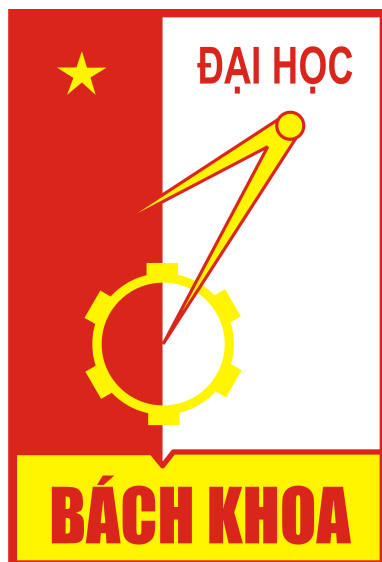


**ĐẠI HỌC BÁCH KHOA HÀ NỘI**  
**TRƯỜNG CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG**



**BÁO CÁO BÀI TẬP LỚN**  
**THỰC HÀNH KIẾN TRÚC MÁY TÍNH**  
**IT3280**

**Học kì 20232 - Năm học: 2023 - 2024**

**Giảng viên hướng dẫn: ThS. Lê Bá Vui**

**Danh sách thành viên: Nguyễn Khoa Ninh - 20226117**  
**Dương Thái Anh - 20226099**

**Mã lớp: 147789**

**Hà Nội, 2024**

# **Báo cáo Thực hành Kiến trúc máy tính Bài 5**

**Họ và tên: Nguyễn Khoa Ninh**

**MSSV: 20226117**

### **\*Phân tích cách thực hiện:**

- Đầu vào: Biểu thức trung tố
- Đầu ra: In ra biểu thức ở dạng hậu tố và tính giá trị biểu thức hậu tố

### **\*Ý tưởng và thuật toán:**

a) Đổi biểu thức hậu tố sang trung tố

Để đổi biểu thức trung tố sang hậu tố, ta sẽ dùng ngăn xếp stack và xâu.

Bước 1: Đưa 1 biểu thức trung tố vào 1 xâu kí tự và đặt tên là infix.

Bước 2: Tạo ra 1 xâu mới để lưu biểu thức hậu tố, đặt tên là postfix.

Bước 3: Sắp xếp lại biểu thức:

- Nếu kí tự là số thì lưu vào postfix.
- Nếu kí tự là toán tử, nếu ngăn xếp trống thì đẩy vào ngăn xếp .
- Nếu kí tự là dấu '(' thì cho vào ngăn xếp.
- Nếu gặp kí tự ')' thì sẽ lấy hết kí tự sau dấu '(' cho vào postfix.
- Nếu toán tử đang xét có bậc cao hơn toán tử ở đỉnh ngăn xếp thì đẩy toán tử đó vào ngăn xếp .
- Nếu toán tử đang xét có bậc bằng toán tử ở đỉnh ngăn xếp thì lấy toán tử đỉnh ngăn xếp ra, xếp vào postfix và đẩy toán tử đang xét vào 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.

Bước 4: Thực hiện bước 3 cho đến khi kết thúc biểu thức và tất cả các toán tử toán hạng được xếp vào postfix, khi đó ta có biểu thức hậu tố.

b) Tính giá trị biểu thức

Bước 1: Quét toàn bộ biểu thức từ trái sang phải.

Bước 2: Tạo 1 ngăn xếp mới.

Bước 3: Nếu phần tử được quét là toán hạng thì đưa vào ngăn xếp.

Bước 4: Nếu phần tử được quét là toán tử thì lấy 2 toán hạng trong ngăn xếp ra, sau đó tính toán giá trị của chúng dựa vào toán tử này, sau đó đẩy lại vào ngăn xếp.

Bước 5: Thực hiện bước 3 và bước 4 cho đến khi kết thúc biểu thức và trong ngăn xếp còn 1 giá trị duy nhất. Đó chính là giá trị của biểu thức

**\*Mã giả**

Khởi tạo bộ nhớ cho:

biểu thức trung tố (infix)

biểu thức hậu tố (postfix)

ngăn xếp toán tử (operator stack)

ngăn xếp chung (general stack)

thông báo kết thúc (end message)

thông báo lỗi (error message)

thông báo bắt đầu (start message)

thông báo hậu tố (postfix prompt)

thông báo kết quả (result prompt)

thông báo trung tố (infix prompt)

Bắt đầu chương trình:

In thông báo bắt đầu và yêu cầu người dùng nhập biểu thức trung tố

Nếu người dùng hủy bỏ (input = -2), thoát chương trình

Nếu người dùng nhấn enter (input = -3), bắt đầu lại lời nhắc

In biểu thức trung tố

Khởi tạo các biến trạng thái:

state = 0 # Biến trạng thái

digit\_count = 0 # Đếm số chữ số

postfix\_offset = -1 # Định vị của hậu tố

operator\_offset = -1 # Định vị của toán tử

infix\_pointer = địa chỉ của infix # Con trỏ của biểu thức trung tố

postfix\_pointer = địa chỉ của postfix # Con trỏ của biểu thức hậu tố

operator\_pointer = địa chỉ của operator # Con trỏ của toán tử

Bắt đầu quét biểu thức trung tố:

while (mỗi ký tự trong biểu thức trung tố):

Tăng con trỏ infix thêm 1

Đọc ký tự hiện tại vào t4

nếu (t4 là khoảng trắng), tiếp tục quét

nếu (t4 là newline), kết thúc quét và chuyển tất cả các toán tử sang hậu tố

nếu (state là 0), xử lý như chữ số đầu tiên

nếu (state là 1), xử lý như chữ số thứ hai

nếu (state là 2), xử lý như chữ số thứ ba

Kiểm tra thứ tự ưu tiên của toán tử:

nếu (t4 là '+', '-'):

nếu (toán tử trên đỉnh ngăn xếp là '+', '-'), chuyển toán tử sang hậu tố

nếu (toán tử trên đỉnh ngăn xếp là '\*', '/', '%'), chuyển toán tử sang hậu tố

Đẩy t4 vào ngăn xếp toán tử

nếu (t4 là '\*', '/', '%'):

nếu (toán tử trên đỉnh ngăn xếp là '\*', '/', '%'), chuyển toán tử sang hậu tố

Đẩy t4 vào ngăn xếp toán tử

nếu (t4 là '('), đẩy vào ngăn xếp toán tử

nếu (t4 là ')'), bật từ ngăn xếp sang hậu tố cho đến khi gặp '('

Cập nhật trạng thái và tiếp tục quét

Kết thúc quét:

Chuyển tất cả các toán tử còn lại trong ngăn xếp sang hậu tố

nếu (ngăn xếp toán tử có dấu ngoặc không khớp), hiển thị thông báo lỗi

Xử lý các chữ số:

nếu (state là 1), lưu chữ số đầu tiên

nếu (state là 2), lưu chữ số thứ hai và kết hợp với chữ số đầu tiên

nếu (state là 3), lưu chữ số thứ ba và kết hợp với các chữ số trước đó

Chuyển chữ số sang hậu tố và đặt lại trạng thái

In biểu thức hậu tố

Thoát chương trình

## Code

.data

infix: .space 256

postfix: .space 256

operator: .space 256

stack: .space 256

endMsg: .ascii "Tiep tục??"

errorMsg: .ascii "Input khong chinh xac"

startMsg: .ascii "Nhap vao bieu thuc trung to\nNote: co chua +- \* / % ()\nso tu 00-99"

prompt\_postfix: .ascii "Bieu thuc hau to: "

prompt\_result: .ascii "Ket qua: "

prompt\_infix: .ascii "Bieu thuc trung to:"

.text

start:

# nhap vao bieu thuc trung to

li \$v0, 54 # Load mã hệ thống gọi để đọc chuỗi vào \$v0

```

la $a0, startMsg # Load địa chỉ của chuỗi startMsg vào $a0
la $a1, infix # Load địa chỉ buffer nơi chuỗi đầu vào sẽ được lưu vào $a1
la $a2, 256 # Load độ dài max của chuỗi đầu vào vào $a2
syscall

```

```

beq $a1,-2,end # if 'cancel' syscall = -2 -> end
beq $a1,-3,start # if 'enter' then start

```

```

# in biểu thức trung to
li $v0, 4
la $a0, prompt_infix
syscall

```

```

li $v0, 4
la $a0, infix #Tải địa chỉ của chuỗi infix vào $a0. Chuỗi infix là biểu thức infix
mà người dùng đã nhập vào trước đó.
syscall

```

```

li $v0, 11 #in 1 ký tự
li $a0, '\n'
syscall

```

```

# khởi tạo các trạng thái
li $s7,0 # biến trạng thái $s7
    # trạng thái "1" khi nhận vào số (0 -> 99)
    # trạng thái "2" khi nhận vào toán tử * / + - %
    # trạng thái "3" khi nhận vào dấu "("
    # trạng thái "4" khi nhận vào dấu ")"
li $t9, 0 # Khởi tạo bộ đếm số chữ số nhận được (để theo dõi số chữ số của
một số nguyên)
li $t5, -1 # Khởi tạo offset cho postfix, ban đầu là -1 (vị trí bắt đầu của mảng
postfix)
li $t6, -1 # Khởi tạo offset cho toán tử, ban đầu là -1 (vị trí bắt đầu của mảng
toán tử)
la $t1, infix # Tải địa chỉ của mảng infix vào thanh ghi $t1
la $t2, postfix # Tải địa chỉ của mảng postfix vào thanh ghi $t2
la $t3, operator # Tải địa chỉ của mảng toán tử vào thanh ghi $t3
addi $t1, $t1, -1 # Giảm địa chỉ infix ban đầu đi 1 để chuẩn bị cho việc duyệt
từng ký tự của chuỗi infix

```

# chuyen sang postfix

scanInfix: # For moi ki tu trong postfix

# kiem tra dau vao

addi \$t1, \$t1, 1 # tang vi tri con tro infix len 1 don vi  $i = i + 1$

lb \$t4, 0(\$t1) # lay gia tri cua con tro infix hien tai, giá trị của biến \$t4 sẽ là giá trị của byte đầu tiên trong vùng nhớ

# mà con trỏ \$t1 đang trỏ tới.

beq \$t4, ' ', scanInfix # neu la space tiep tục scan

beq \$t4, '\n', EOF # Scan ket thuc pop tat ca cac toan tu sang postfix

beq \$t9, 0, digit1 # Neu trang thai la 0 => co 1 chu so

beq \$t9, 1, digit2 # Neu trang thai la 1 => co 2 chu so

beq \$t9, 2, digit3 # neu trang thai la 2 => co 3 chu so

continueScan:

beq \$t4, '+', plusMinus # Kiểm tra nếu \$t4 là '+', chuyển đến nhãn plusMinus

beq \$t4, '-', plusMinus # Kiểm tra nếu \$t4 là '-', chuyển đến nhãn plusMinus

beq \$t4, '\*', multiplyDivideModulo # Kiểm tra nếu \$t4 là '\*', chuyển đến nhãn multiplyDivideModulo

beq \$t4, '/', multiplyDivideModulo # Kiểm tra nếu \$t4 là '/', chuyển đến nhãn multiplyDivideModulo

beq \$t4, '%', multiplyDivideModulo # Kiểm tra nếu \$t4 là '%', chuyển đến nhãn multiplyDivideModulo

beq \$t4, '(', openBracket # Kiểm tra nếu \$t4 là '(', chuyển đến nhãn openBracket

beq \$t4, ')', closeBracket # Kiểm tra nếu \$t4 là ')', chuyển đến nhãn closeBracket

wrongInput: # dau vao loi

li \$v0, 55

la \$a0, errorMsg

li \$a1, 2

syscall

j ask

finishScan:

# in bieu thuc infix

# Print prompt:

li \$v0, 4

la \$a0, prompt\_postfix



```
syscall  
li $t6,-1 # set gia tri infix hien tai la $s6= -1
```

printPostfix:

```
addi $t6, $t6, 1 # Tăng offset của postfix hiện tại  
add $t8, $t2, $t6 # Load địa chỉ của postfix hiện tại  
lbu $t7, ($t8) # Load giá trị của postfix hiện tại  
bgt $t6, $t5, finishPrint # Nếu đã in hết postfix, chuyển sang tính kết quả  
bgt $t7, 99, printOperator # Nếu postfix hiện tại > 99, là một toán tử  
# Neu khong thi la mot toan hang  
li $v0, 1  
add $a0,$t7,$zero  
syscall
```

```
li $v0, 11  
li $a0, ''  
syscall  
j printPostfix # Loop
```

printOperator:

```
li $v0, 11  
addi $t7,$t7,-100 # Decode toan tu  
add $a0,$t7,$zero  
syscall  
li $v0, 11  
li $a0, ''  
syscall  
j printPostfix # Loop
```

finishPrint:

```
li $v0, 11  
li $a0, '\n'  
syscall
```

```
# tính toán kết quả  
li $t9,-4 # set offset của đỉnh stack là -4  
la $t3,stack # Load địa chỉ đỉnh stack  
li $t6,-1 # Đặt offset của Postfix hiện tại là -1
```

CalculatorPost:

```

addi $t6,$t6,1 # tang offset hien tai cua Postfix
add $t8,$t2,$t6 # Load dia chi cua postfix hien tai
lbu $t7,($t8) # Load gia tri cua postfix hien tai
bgt $t6,$t5,printResult # tinh toan ket qua va in ra
bgt $t7,99,calculate # neu gia tri postfix hien tai > 99 --> toan tu --> lay ra 2 toan
hang va tinh toan
# neu khong thi la toan hang
addi $t9,$t9,4 # tang offset dinh stack len
add $t4,$t3,$t9 # tang dia chi cua dinh stack
sw $t7, ($t4) # day so vao stack
j CalculatorPost # Loop

```

calculate:

```

# Pop 1 so
add $t4,$t3,$t9
lw $t0,($t4)

```

```

# pop so tiep theo
addi $t9,$t9,-4
add $t4,$t3,$t9
lw $t1,($t4)

```

```

# Decode toan tu
beq $t7,143,plus
beq $t7,145,minus
beq $t7,142,multiply
beq $t7,147,divide
beq $t7, 137, modulo

```

plus:

```

add $t0,$t0,$t1 # tinh tong gia tri cua 2 con tro dang luu gia tri toan hang
sw $t0,($t4) # luu gia tri cua con tro ra $t4
# li $t0, 0 # Reset t0, t1
# li $t1, 0
j CalculatorPost

```

minus:

```

sub $t0, $t1,$t0
sw $t0,($t4)
# li $t0, 0 # Reset t0, t1

```

```
# li $t1, 0
j CalculatorPost
```

multiply:

```
mul $t0, $t1,$t0
sw $t0,($t4)
# li $t0, 0 # Reset t0, t1
# li $t1, 0
j CalculatorPost
```

divide:

```
div $t1, $t0
mflo $t0
sw $t0,($t4)
# li $t0, 0 # Reset t0, t1
# li $t1, 0
j CalculatorPost
```

modulo:

```
div $t1, $t0
mfhi $t0
sw $t0,($t4)
# li $t0, 0 # Reset t0, t1
# li $t1, 0
j CalculatorPost
```

printResult:

```
li $v0, 4
la $a0, prompt_result
syscall
```

```
li $v0, 1
lw $a0,($t4) # load gia tri cua $t4 ra con tro $t0
```

```
syscall
li $v0, 11
li $a0, '\n'
syscall
```

ask: # tiep tục không??

```

li $v0, 50
la $a0, endMsg
syscall
beq $a0,0,start
beq $a0,2,ask
# End program

```

```

end:
li $v0, 10
syscall

```

# Sub program

EOF:

```

    beq $s7,2,wrongInput      # Nếu trạng thái là 2 (toán tử), báo lỗi input không
    hợp lệ
    beq $s7,3,wrongInput      # Nếu trạng thái là 3 (dấu ngoặc mở), báo lỗi input
    không hợp lệ
    beq $t5,-1,wrongInput      # Nếu postfix không có giá trị (offset là -1), báo lỗi
    input không hợp lệ
    j popAllOperatorInStack    # Nếu tất cả điều kiện đều không đúng, chuyển tất
    cả toán tử trong ngăn xếp sang postfix

```

digit1:                   # Kiểm tra và lưu chữ số đầu tiên.

```

    beq $t4,'0',storeDigit1
    beq $t4,'1',storeDigit1
    beq $t4,'2',storeDigit1
    beq $t4,'3',storeDigit1
    beq $t4,'4',storeDigit1
    beq $t4,'5',storeDigit1
    beq $t4,'6',storeDigit1
    beq $t4,'7',storeDigit1
    beq $t4,'8',storeDigit1
    beq $t4,'9',storeDigit1
    j continueScan

```

digit2:                   #Kiểm tra và lưu chữ số thứ hai hoặc lưu số hiện tại vào  
postfix nếu không phải là chữ số.

```

    beq $t4,'0',storeDigit2
    beq $t4,'1',storeDigit2
    beq $t4,'2',storeDigit2

```

```

beq $t4,'3',storeDigit2
beq $t4,'4',storeDigit2
beq $t4,'5',storeDigit2
beq $t4,'6',storeDigit2
beq $t4,'7',storeDigit2
beq $t4,'8',storeDigit2
beq $t4,'9',storeDigit2
# neu khong nhap vao chu so thu 2
jal numberToPostfix
j continueScan

```

digit3:

# Kiểm tra và báo lỗi nếu gặp chữ số thứ ba hoặc lưu số hiện tại vào postfix nếu không phải là chữ số.

```

beq $t4,'0',wrongInput
beq $t4,'1',wrongInput
beq $t4,'2',wrongInput
beq $t4,'3',wrongInput
beq $t4,'4',wrongInput
beq $t4,'5',wrongInput
beq $t4,'6',wrongInput
beq $t4,'7',wrongInput
beq $t4,'8',wrongInput
beq $t4,'9',wrongInput
# neu khong co chu so thu 3
jal numberToPostfix
j continueScan

```

plusMinus: # Input is + -

```

beq $s7,2,wrongInput # Nhan toan tu sau toan tu hoac "("
beq $s7,3,wrongInput
beq $s7,0,wrongInput # nhan toan tu truoc bat ki so nao
li $s7,2 # Thay doi trang thai dau vao thanh 2

```

continuePlusMinus:

```

beq $t6,-1,inputOperatorToStack # Khong co gi trong stack -> day vao
add $t8,$t6,$t3 # Load dia chi cua toan tu o dinh
lb $t7,($t8) # Load byte gia tri cua toan tu o dinh
beq $t7,(',',inputOperatorToStack # neu dinh la ( --> day vao
beq $t7,'+',equalPrecedence # neu dinh la + - -> day vao

```

```

beq $t7,'-',equalPrecedence
beq $t7,'*',lowerPrecedence # neu dinh la * / % thi lay * / % ra roi day vao
beq $t7,'/',lowerPrecedence
beq $t7,'% ',lowerPrecedence

```

```

multiplyDivideModulo: # dau vao la * / %
    beq $s7,2,wrongInput # Nhan toan tu sau toan tu hoac "("
    beq $s7,3,wrongInput
    beq $s7,0,wrongInput # Nhan toan tu truoc bat ki so nao
    li $s7,2 # Thay doi trang thai dau vao thanh 2
    beq $t6,-1,inputOperatorToStack # Khong co gi trong stack -> day vao
    add $t8,$t6,$t3 # Load dia chi cua toan tu o dinh
    lb $t7,($t8) # Load byte gia tri cua toan tu o dinh
    beq $t7,(',',inputOperatorToStack # neu dinh la ( --> day vao
    beq $t7,+',inputOperatorToStack # neu dinh la + - -> day vao
    beq $t7,'-',inputOperatorToStack
    beq $t7,'*',equalPrecedence # neu dinh la * / % day vao
    beq $t7,'/',equalPrecedence
    beq $t7,'% ',equalPrecedence

```

```

openBracket: # dau vao la (
    beq $s7,1,wrongInput # Nhan "(" sau mot so hoac dau ")"
    beq $s7,4,wrongInput
    li $s7,3 # Thay doi trang thai dau vao thanh 3
    j inputOperatorToStack

```

```

closeBracket: # dau vao la ")"
    beq $s7,2,wrongInput # Nhan ")" sau mot toan tu hoac toan tu
    beq $s7,3,wrongInput
    li $s7,4 # Thay doi trang thai dau vao thanh 4
    add $t8,$t6,$t3 # Load dia chi toan tu dinh
    lb $t7,($t8) # Load gia tri cua toan tu o dinh
    beq $t7,(',',wrongInput # Input bao gom () khong co gi o giua --> error

```

```

continueCloseBracket:
    beq $t6,-1,wrongInput # khong tim duoc dau "(" --> error
    add $t8,$t6,$t3 # Load dia chi cua toan tu o dinh
    lb $t7,($t8) # Load gia tri cua toan tu o dinh
    beq $t7,(',',matchBracket # Tim ngoac phu hop
    jal PopOperatorToPostfix # day toan tu o dinh vao postfix

```

j continueCloseBracket # tiếp tục vòng lặp cho đến khi tìm được ngoặc phù hợp

equalPrecedence: # nhân + - vào đỉnh stack là + - || nhân \* / % vào đỉnh stack là \* / %

jal PopOperatorToPostfix # lấy toán tử đỉnh stack ra Postfix

j inputOperatorToStack # đẩy toán tử mới vào stack

lowerPrecedence: # nhân + - vào đỉnh stack \* / %

jal PopOperatorToPostfix # lấy toán tử đỉnh stack ra và đẩy vào postfix

j continuePlusMinus # tiếp tục vòng lặp

inputOperatorToStack: # đẩy dấu vào cho toán tử

add \$t6,\$t6,1 # tăng offset của toán tử ở đỉnh lên 1

add \$t8,\$t6,\$t3 # load địa chỉ của toán tử ở đỉnh

sb \$t4,(\$t8) # lưu toán tử nhập vào stack

j scanInfix

PopOperatorToPostfix: # lấy toán tử ở đỉnh và lưu vào postfix

addi \$t5,\$t5,1 # tăng offset của toán tử ở đỉnh stack lên 1

add \$t8,\$t5,\$t2 # load địa chỉ của toán tử ở đỉnh stack

addi \$t7,\$t7,100 #  $m_i \frac{1}{2} h_i \frac{1}{2}$  toán tử + 100

sb \$t7,(\$t8) # lưu toán tử vào postfix

addi \$t6,\$t6,-1 # giảm offset của toán tử ở đỉnh stack đi 1

jr \$ra

matchBracket: # xóa cặp dấu ngoặc

addi \$t6,\$t6,-1 # giảm offset của toán tử ở đỉnh stack đi 1

j scanInfix

popAllOperatorInStack: # lấy hết toán tử vào postfix

jal numberToPostfix # Gọi hàm numberToPostfix để xử lý số hiện tại (nếu có)

beq \$t6,-1,finishScan # Nếu stack rỗng ( $t6 == -1$ ), nhảy đến finishScan

add \$t8,\$t6,\$t3 # Tính địa chỉ của toán tử ở đỉnh stack

lb \$t7,(\$t8) # Tải giá trị của toán tử ở đỉnh stack

beq \$t7, '(', wrongInput # Nếu là dấu ngoặc mở '(', báo lỗi

beq \$t7, ')', wrongInput # Nếu là dấu ngoặc đóng ')', báo lỗi

jal PopOperatorToPostfix # Gọi hàm PopOperatorToPostfix để lấy toán tử ra và đưa vào postfix

j popAllOperatorInStack # Lặp lại cho đến khi stack rỗng

storeDigit1:

```
beq $s7,4,wrongInput    # Nếu trạng thái là 4 (sau khi gặp)'), thì báo lỗi
addi $s4,$t4,-48         # Chuyển ký tự số thành số nguyên (ASCII '0' là 48)
add $t9,$zero,1          # Đặt trạng thái đếm thành 1
li $s7,1                 # Đặt trạng thái đầu vào thành 1 (nhập số)
j scanInfix              # Quay lại nhãn scanInfix để tiếp tục quét biểu thức
```

storeDigit2:

```
beq $s7,4,wrongInput    # Nếu trạng thái là 4 (sau khi gặp)'), thì báo lỗi
addi $s5,$t4,-48         # Chuyển ký tự số thứ hai thành số nguyên
mul $s4,$s4,10           # Nhân chữ số đầu tiên với 10
add $s4,$s4,$s5          # Thêm chữ số thứ hai vào để tạo thành số nguyên
add $t9,$zero,2          # Đặt trạng thái đếm thành 2
li $s7,1                 # Đặt trạng thái đầu vào thành 1 (nhập số)
j scanInfix              # Quay lại nhãn scanInfix để tiếp tục quét biểu thức
```

numberToPostfix:

```
beq $t9,0,endnumberToPostfix # Nếu không có số để lưu (t9 == 0), chuyển đến
endnumberToPostfix
addi $t5,$t5,1           # Tăng offset của postfix lên 1
add $t8,$t5,$t2           # Tải địa chỉ của phần tử postfix tiếp theo vào $t8
sb $s4,($t8)             # Lưu số (trong $s4) vào postfix
li $t9,0                 # Đặt trạng thái đếm về 0
```

endnumberToPostfix:

```
jr $ra                   # Trở về địa chỉ trả về (return address)
```

Ví dụ chạy

Biểu thức

```
((20 + 30) * 6 - 98) % 3
= (50 * 6 - 98) % 3
= (300 - 98) % 3
= 202 % 3
= 1
```

Kết quả của phép tính  $((20 + 30) * 6 - 98) \% 3$  là 1.



Reset: reset completed.

Bieu thuc trung to:  $((20 + 30) * 6 - 98) \% 3$

Bieu thuc hau to:  $20 \ 30 + 6 * 98 - 3 \%$

Ket qua: 1

Bieu thuc trung to:  $((20 + 30) * 6 - 98) \% 3$

Bieu thuc hau to:  $20 \ 30 + 6 * 98 - 3 \%$

Ket qua: 1

-- program is finished running --

# Báo cáo Thực hành Kiến trúc máy tính Bài 6

Họ và tên: Dương Thái Anh

MSSV: 20226099

## \*Đề bài (Hàm cấp phát bộ nhớ Malloc()):

Chương trình cho bên dưới là hàm malloc(), kèm theo đó là ví dụ minh họa, được viết bằng hợp ngữ MIPS, để cấp phát bộ nhớ cho một biến con trỏ nào đó. Hãy đọc chương trình và hiểu rõ nguyên tắc cấp phát bộ nhớ động.

Trên cơ sở đó, hãy hoàn thiện chương trình như sau: (Lưu ý, ngoài viết các hàm đó, cần viết thêm một số ví dụ minh họa để thấy việc sử dụng hàm đó như thế nào)

- 1) Việc cấp phát bộ nhớ kiểu word/mảng kiểu word có 1 lỗi, đó là chưa bảo đảm qui tắc địa chỉ của kiểu word phải chia hết cho 4. Hãy khắc phục lỗi này.
- 2) Viết hàm lấy giá trị của biến con trỏ.
- 3) Viết hàm lấy địa chỉ biến con trỏ.
- 4) Viết hàm thực hiện copy 2 con trỏ xâu kí tự.
- 5) Viết hàm giải phóng bộ nhớ đã cấp phát cho các biến con trỏ
- 6) Viết hàm tính toàn bộ lượng bộ nhớ đã cấp phát.
- 7) Hãy viết hàm malloc2 để cấp phát cho mảng 2 chiều kiểu .word với tham số vào gồm:
  - a. Địa chỉ đầu của mảng
  - b. Số dòng
  - c. Số cột
- 8) Tiếp theo câu 7, hãy viết 2 hàm `getArray[i][j]` và `setArray[i][j]` để lấy/thiết lập giá trị cho phần tử ở dòng i cột j của mảng.

## \*Phân tích cách thực hiện:

- Input : menu các chức năng người dùng.
- Output : kết quả theo từng yêu cầu đề bài.

## \*Ý tưởng và thuật toán:

1. Ý tưởng:

- Thực hiện các yêu cầu của đề bài theo dạng menu chức năng.
- Mỗi chức năng tương ứng với 1 option trong menu.
- Nửa đầu code là các hàm menu từ 1 – 11.
- Nửa cuối là các chương trình con xử lý.

2. Thuật toán:

- Khởi tạo vùng cấp phát động trước ở hàm `SysInitMem`.
- Đầu tiên ở hàm `main`, là hàm `print_menu` : in ra message các yêu cầu của đề bài, rồi input dialog để yêu cầu nhập từ 1-11.
- Rồi giá trị vừa nhập được lưu ở thanh ghi `$a0` gán vào thanh ghi `$t0`.
- So sánh `$t0` với 1-11, nếu bằng các giá trị tương ứng thì sẽ hiện ra chức năng tương ứng, còn nếu nhập khác từ 1-11 thì nhảy đến hàm `end` : thoát chương trình.

### \* Option 1: Cho kiểu char.

- Ban đầu, chương trình sẽ yêu cầu người dùng nhập số lượng phần tử trong mảng ký tự và giá trị của từng phần tử.

- Tiếp theo, chương trình sẽ kiểm tra xem giá trị nhập vào có nằm trong khoảng từ 0 đến 1000 không bằng cách gọi hàm Check\_value.
- Sau đó, kích thước của mảng ký tự sẽ được gán là 1 byte và chương trình sẽ tiến hành cấp phát bộ nhớ bằng hàm malloc.
- Trong hàm malloc, chương trình sẽ xác định địa chỉ của ô nhớ trống đầu tiên trong Sys\_TopOfFree.
- Nếu kích thước của mỗi phần tử là 4 byte (kiểu word), chương trình sẽ làm tròn kích thước của mảng cần cấp phát về bội số của 4. Nếu không, chương trình sẽ lưu địa chỉ ô nhớ trống vào biến con trỏ và giá trị trả về của hàm malloc vào thanh ghi \$v0.
- Tiếp theo, chương trình sẽ tính kích thước của mảng cần cấp phát và lưu địa chỉ ô nhớ trống đầu tiên.
- Quay lại option1, chương trình sẽ thông báo cấp phát thành công và in giá trị của ô nhớ cấp phát dưới dạng mã hexa. Cuối cùng, chương trình sẽ quay lại menu chính để tiếp tục.

**\* Option2 : Cho kiểu byte.** - Tương tự như kiểu char, kích thước của nó là 1 byte mỗi 1 phần tử Byte.

**\* Option3 : Cho kiểu word.** - Tương tự như option 1, 2, tuy nhiên ở đây đã thực hiện đúng với yêu cầu sửa lỗi của đề bài là quy tắc địa chỉ của word phải chia hết cho 4 :chương trình sẽ làm tròn kích thước của mảng cần cấp phát về bội số của 4.

**\*Option4:Trả về giá trị các biến con trỏ đó.**

- Đầu tiên, chương trình sẽ in ra thông điệp: "Địa chỉ của biến con trỏ theo Char, Byte, Word, Array".
- Sau đó, chương trình sẽ in ra giá trị của các biến con trỏ. \$a0 sẽ được gán giá trị từ 0 đến 3, tương ứng với Char, Byte, Word, Array. Ví dụ, với kiểu char, các kiểu khác sẽ được xử lý tương tự.
- Tiếp theo, chương trình sẽ điều hướng đến hàm ptr\_value để lấy giá trị của các biến con trỏ. Sau đó, nó sẽ nhảy đến hàm print\_value\_or\_address để in giá trị của các biến con trỏ ra dưới dạng mã hexa, cách nhau bởi dấu "," và kết thúc bởi dấu ".". Cuối cùng, chương trình sẽ quay lại menu chính để tiếp tục.

**\*Option5 : In ra địa chỉ các biến con trỏ.**

- Tương tự như option4 nhảy đến hàm ptr\_value thì option5 ta nhảy đến hàm ptr\_address để lấy địa chỉ các biến con trỏ.
- sll để cho các địa chỉ nằm liên tiếp nhau tương ứng cách nhau 4 bytes.
- Nhảy đến hàm print\_value\_or\_address để in giá trị các biến con trỏ đó ra theo mã hexa cách nhau dấu "," và kết thúc bởi dấu "."
- Rồi quay về main tiếp tục thực hiện menu.

**\* Option6 : Copy 2 con trỏ chuỗi ký tự.**

- Gán địa chỉ CharPtr1 vào thanh ghi \$t0.
- Gán địa chỉ copied\_string vào thanh ghi \$t1.
- Ghi giá trị thanh ghi \$t1 vào CharPtr1 -> CharPtr1 trỏ tới copied\_string.
- CharPtr2 trỏ tới Sys\_TopOfFree
- Tiếp theo, copy\_loop : vòng lặp để copy

- Ta copy từng kí tự một tại \$t1 vào thanh ghi \$t3
- Lưu từng kí tự của \$t3 đó vào ô nhớ tại địa chỉ con trỏ xâu hai ở \$t2
- Tăng biến đếm \$t4 : đếm số lượng kí tự string lên 1
- Chuyển sang kí tự tiếp theo của 2 xâu CharPtr1, CharPtr2.
- Kiểm tra \$t3 string đã nhập đó gặp null thì kết thúc chuỗi – nhảy đến exit\_copy , ko thì tiếp tục lặp để copy từng kí tự .
- exit\_copy:
- Lưu \$t4 : số bytes dùng để lưu string vào \$a0.
- Đồng thời load con trỏ xâu 2 CharPtr.
- Lưu xâu đã copy từ \$a0 vào \$a2 – CharPtr2. - In ra nội dung CharPtr2 trở tới.
- Rồi quay về main , tiếp tục thực hiện menu.

***\*Option7 : Giải phóng bộ nhớ đã cấp phát cho các biến con trỏ.***

- Đầu tiên, nó sẽ gọi hàm freeStorage.
- Trong hàm freeStorage: Chương trình sẽ gán địa chỉ của Sys\_TopOfFree và Sys\_MyFreeSpace vào các thanh ghi \$t9 và \$t8. Sau đó, nó sẽ ghi địa chỉ của Sys\_MyFreeSpace từ thanh ghi \$t8 vào Sys\_TopOfFree. Tiếp theo, chương trình sẽ lấy địa chỉ của biến con trỏ đầu tiên và gán \$t2 = 0. Trong vòng lặp: Chương trình sẽ gán giá trị của từng con trỏ bằng 0. In ra thông điệp cho biết bộ nhớ đã được giải phóng. Cuối cùng, chương trình sẽ quay lại menu chính để tiếp tục.

***\* Option8 : Tính lượng bộ nhớ đã cấp phát.***

- Đầu tiên, in ra message lượng bộ nhớ đã cấp phát.
- Rồi nhảy đến chương trình con storage.
- Load giá trị tại địa chỉ còn trống đầu tiên vào thanh ghi \$t9
- Load Sys\_MyFreeSpace vào thanh \$t8, luôn là thanh ghi ngay sau Sys\_TopOfFree.
- Trừ thanh ghi \$t9 với \$t8 lưu vào \$v0 được lượng bộ nhớ đã cấp phát.
- Gán thanh ghi \$v0 vào \$a0 để in ra .
- Rồi cùng quay về main , tiếp tục thực hiện menu.

***\*Option9: : Cấp phát bộ nhớ cho mảng 2 chiều.***

- Đầu tiên, người dùng sẽ được yêu cầu nhập số hàng và số cột, và chương trình sẽ kiểm tra tính hợp lệ của các giá trị này.
- Số hàng và số cột sẽ được lưu vào các thanh ghi \$a1 và \$a2.
- Sau đó, địa chỉ của con trỏ kiểu Array sẽ được tải và chương trình sẽ nhảy đến hàm Malloc2(). Trong hàm này:
- Ý tưởng là cấp phát bộ nhớ cho một mảng có số hàng \* số cột phần tử, sử dụng lại hàm malloc và sử dụng stack.
- Trước tiên, một phần tử sẽ được thêm vào stack và \$ra sẽ được push vào. Nếu số hàng hoặc số cột vượt quá 1000, chương trình sẽ in ra thông báo lỗi.
- Nếu không, số hàng sẽ được lưu vào row và số cột sẽ được lưu vào phần tử tiếp theo.
- Sau đó, số hàng nhân với số cột sẽ được tính để có số lượng phần tử của mảng Array. \$a2 sẽ được gán bằng 4 để xác định Array là mảng kiểu word (4 byte).
- Chương trình sẽ nhảy đến hàm malloc để cấp phát bộ nhớ cho mỗi phần tử của Array, và kết quả sẽ được lưu vào thanh ghi \$v0.
- Lấy lại giá trị \$ra từ ngăn xếp.

- Lưu \$v0 vào \$t0, in ra cấp phát thành công.
- Cuối cùng, chương trình sẽ quay lại menu chính để tiếp tục.

**\* Option10 : *setArray[i][j]***

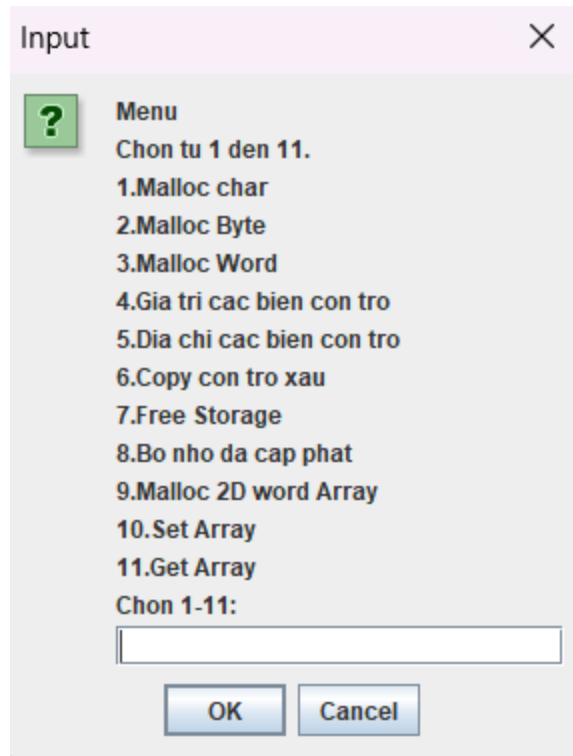
- Đầu tiên, load địa chỉ con trỏ Array lưu vào thanh ghi \$s7.
- Kiểm tra xem nếu = 0 , in ra thông báo con trỏ rỗng.
- Còn không thì, yêu cầu nhập hàng thứ i, cột thứ j.
- Rồi gán lần lượt 2 giá trị đó vào \$s0, \$s1. - Tiếp tục nhập giá trị muốn gán, lưu vào thanh ghi \$a3.
- 2 thanh ghi \$s0, \$s1 chứa i , j lưu vào thanh ghi \$a1, \$a2. Con trỏ Array lưu vào \$a0.
- Nhảy đến chương trình con SetArray. Ở đây, load hàng i \$a1, cột j \$s2 vào \$s0.
- Load số hàng số cột từ \$s0 trong stack vào \$s1, \$s2 . - Kiểm tra \$a1 >= \$s1, \$a2 >= \$s2, in ra thông báo ngoài vùng cho phép.
- Còn không thì nhân với nhau , tức \$a1 \* \$a2 lưu vào \$s0.
- Mỗi phần tử cách nhau 4 bytes.
- $\$s0 = *array + (i*col + j)*4$  là địa chỉ của vị trí muốn gán.
- Store \$a3 ở đây chính là tham số giá trị gán vào .
- Quay về main, tiếp tục thực hiện menu .

**\* Option11 : *getArray[i][j]***

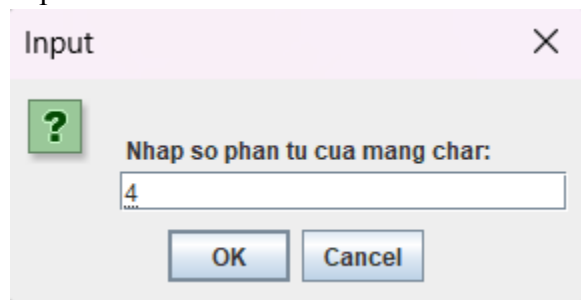
- Đầu tiên, load địa chỉ con trỏ Array lưu vào thanh ghi \$s1.
- Kiểm tra xem nếu = 0 , in ra thông báo con trỏ rỗng.
- Còn không thì, yêu cầu nhập hàng thứ i, cột thứ j, lần lượt lưu vào \$a1, \$a2.
- Gán \$s1 ở trên vào \$a0 chứa địa chỉ con trỏ Array (bắt đầu mảng).
- Nhảy đến chương trình con GetArray để lấy ra giá trị mình đã gán trước đó.
- Load số hàng số cột từ \$s0 trong stack vào \$s1, \$s2 .
- Kiểm tra \$a1 >= \$s1, \$a2 >= \$s2, in ra thông báo ngoài vùng cho phép.
- Còn không thì nhân với nhau , tức \$a1 \* \$a2 lưu vào \$s0.
- Mỗi phần tử cách nhau 4 bytes.
- $\$s0 = *array + (i*col + j)*4$  là địa chỉ của vị trí muốn gán.
- Lưu vào \$v0 rồi quay về option11.
- In ra giá trị mình đã gán trước đó dưới dạng mã hexa.
- Cuối cùng quay về main.

**\*Kết quả thực thi chương trình:**

- Hiện thị menu:



-Option1:



```
-- program is finished running --
```

```
-- program is finished running --
```

```
Malloc success. Dia chi vung nho duoc cap phat : 0x90000004  
Kich thước của vùng nhớ vừa được cấp phát là 0x00000004
```

-Option2:

Input

?

Nhap so phan tu cua mang byte:

12

OK

Cancel

Malloc success. Dia chi vung nho duoc cap phat : 0x90000004  
Kich thuc cua vung nho vua duoc cap phat la 0x0000000c

-Option3:

Input

?

Nhap so phan tu cua mang word:

2

OK

Cancel

Malloc success. Dia chi vung nho duoc cap phat : 0x90000004  
Kich thuc cua vung nho vua duoc cap phat la 0x0000000c  
Malloc success. Dia chi vung nho duoc cap phat : 0x90000010  
Kich thuc cua vung nho vua duoc cap phat la 0x00000008

-Option4:

Malloc success. Dia chi vung nho duoc cap phat : 0x90000004  
Kich thuc cua vung nho vua duoc cap phat la 0x00000004  
Malloc success. Dia chi vung nho duoc cap phat : 0x90000008  
Kich thuc cua vung nho vua duoc cap phat la 0x0000000c  
Malloc success. Dia chi vung nho duoc cap phat : 0x90000014  
Kich thuc cua vung nho vua duoc cap phat la 0x00000008  
Gia tri tai cac bien con tro CharPtr BytePtr WordPtr ArrayPtr la:0x90000004,0x90000008,0x90000014,0x00000000.

-Option5:

Malloc success. Dia chi vung nho duoc cap phat : 0x90000004  
Kich thuc cua vung nho vua duoc cap phat la 0x00000004  
Malloc success. Dia chi vung nho duoc cap phat : 0x90000008  
Kich thuc cua vung nho vua duoc cap phat la 0x0000000c  
Malloc success. Dia chi vung nho duoc cap phat : 0x90000014  
Kich thuc cua vung nho vua duoc cap phat la 0x00000008  
Gia tri tai cac bien con tro CharPtr BytePtr WordPtr ArrayPtr la:0x90000004,0x90000008,0x90000014,0x00000000.  
Dia chi cua cac bien con tro CharPtr BytePtr WordPtr ArrayPtr la:0x10010000,0x10010004,0x10010008,0x1001000c.

-Option6:

## -Option7:

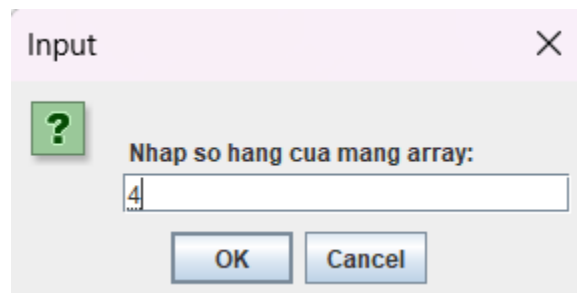
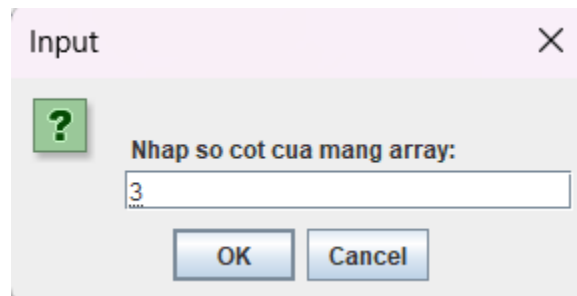
DuongThaiAnh-20226099

```
Malloc success. Dia chi vung nho duoc cap phat : 0x9000001a
Kich thuoc cua vung nho vua duoc cap phat la 0x00000004
Malloc success. Dia chi vung nho duoc cap phat : 0x9000001e
Kich thuoc cua vung nho vua duoc cap phat la 0x0000000c
Malloc success. Dia chi vung nho duoc cap phat : 0x9000002c
Kich thuoc cua vung nho vua duoc cap phat la 0x00000008
Bo nho da cap phat: 48 bytes.
Da giai phong bo nho cap phat!
```

## -Option8:

```
Malloc success. Dia chi vung nho duoc cap phat : 0x9000001a
Kich thuoc cua vung nho vua duoc cap phat la 0x00000004
Malloc success. Dia chi vung nho duoc cap phat : 0x9000001e
Kich thuoc cua vung nho vua duoc cap phat la 0x0000000c
Malloc success. Dia chi vung nho duoc cap phat : 0x9000002c
Kich thuoc cua vung nho vua duoc cap phat la 0x00000008
Bo nho da cap phat: 48 bytes.
```

## -Option9:




```
Malloc success. Dia chi vung nho duoc cap phat : 0x90000004
0x00000003,0x00000004
```

## -Option10 và Option11: +SetArray:




Input ✕

 Nhap i:

Input ✕


 Nhap j:

Input ✕

 Nhap gia tri dau vao:

+GetArray:

Input ✕

 Nhap i:

Input ✕

 Nhap j:

```

Kích thước của vùng nhớ vừa được cấp phát là 0x00000004
Malloc success. Địa chỉ vùng nhớ được cấp phát : 0x90000001e
Kích thước của vùng nhớ vừa được cấp phát là 0x0000000c
Malloc success. Địa chỉ vùng nhớ được cấp phát : 0x90000002c
Kích thước của vùng nhớ vừa được cấp phát là 0x00000008
Bộ nhớ đã cấp phát: 48 bytes.
Đã giải phóng bộ nhớ cấp phát!
Malloc success. Địa chỉ vùng nhớ được cấp phát : 0x900000004
0x00000003, 0x00000004
Giá trị trả về: 0x0000000e

```

---

Giá trị trả về 0x0000000e = 14 => Đúng

## Data Section

.data

CharPtr: .word 0 # Bien con tro, tro toi kieu asciiz

BytePtr: .word 0 # Bien con tro, tro toi kieu Byte

WordPtr: .word 0 # Bien con tro, tro toi mang kieu Word

ArrayPtr: .word 0 # Bien con tro, tro toi mang 2 chieu

CharPtr1: .space 100 #Bien con tro, su dung trong option 6

CharPtr2: .space 100 #Bien con tro, su dung trong option 6

Enter: .asciiz "\n"

row: .word 1

col: .word 1

menu: .asciiz "Menu\nChon tu 1 den 11.\n1.Malloc char\n2.Malloc Byte\n3.Malloc

Word\n4.Gia tri cac bien con tro\n5.Dia chi cac bien con tro\n6.Copy con tro xau\n7.Free

Storage\n8.Bo nho da cap phat\n9.Malloc 2D word Array\n10.Set Array\n11.Get Array\nChon 1-11:"

mal: .asciiz "\nso hang va so cot phai be hon 1000"

char: .asciiz "\n Nhap so phan tu cua mang char:"

word: .asciiz "\n Nhap so phan tu cua mang word:"

byte: .asciiz "\n Nhap so phan tu cua mang byte:"

arr1: .asciiz "\n Nhap so cot cua mang array:"

arr2: .asciiz "\n Nhap so hang cua mang array:"

input\_row: .asciiz "\n Nhap i:"

input\_col: .asciiz "\n Nhap j:"

input\_value: .asciiz "\n Nhap gia tri dau vao:"

output\_value: .asciiz "\n Gia tri tra ve:"

value\_output: .asciiz "\n Gia tri tai cac bien con tro CharPtr BytePtr WordPtr ArrayPtr la:"

address\_output: .asciiz "\n Dia chi cua cac bien con tro CharPtr BytePtr WordPtr ArrayPtr la:"

success: .asciiz "\n Malloc success. Dia chi vùng nhớ được cấp phát : "

notification: .asciiz "\n Kích thước của vùng nhớ vừa được cấp phát là "

bound: .asciiz "\n index out of bound"

null: .asciiz "\nNull Pointer Exception. Chưa khai tạo mảng!!!"

free\_storage\_notification: .asciiz "\nĐã giải phóng bộ nhớ cấp phát! "

storage\_notification: .asciiz "\nBộ nhớ đã cấp phát: "

bytes: .asciiz " bytes."

too\_big: .asciiz "\n Gia tri nhap vao qua lon!!"

too\_small: .asciiz "\n Gia tri nhap vao nho hon hoac bang 0"

copied\_string : .asciiz "DuongThaiAnh-20226099"

```
.kdata
Sys_TopOfFree: .word 1 # Bien chua dia chi dau tien cua vung nho con trong
Sys_MyFreeSpace: # Vung khong gian tu do, dung de cap bo nho cho cac bien con tro
.text
jal SysInitMem #Khoi tao vung nho cap phat dong
```

## Hàm main và các Option 1-11:

```
main:
show_menu:
la $a0,menu #Load dia chi cua menu
jal IntDialog
move $t0, $a0 #Thuc hien chuyen dia chi menu qua $t0, sau do so sanh t0 voi cac gia tri
beq $t0, 1, option1
beq $t0, 2, option2
beq $t0, 3, option3
beq $t0, 4, option4
beq $t0, 5, option5
beq $t0, 6, option6
beq $t0, 7, option7
beq $t0, 8, option8
beq $t0, 9, option9
beq $t0, 10, option10
beq $t0, 11, option11
j end

option1:#Malloc char
la $a0,char #Load dia chi cua char vao $a0
jal IntDialog
jal Check_value #Kiem tra xem gia tri co nam trong khoang xac dinh khong
move $a1,$a0 #Chuyen dia chi cua char vao a1
la $a0,CharPtr #Load dia chi cua char vao a0
li $a2,1 #Cho gia tri tai a2 = 1, tuc la so byte cua bien Char
jal malloc #output $v0 = dia chi bat dau cap phat boi malloc
move $t0,$v0 #Chuyen dia chi tu v0 vao t0
la $a0,success
li $v0,4
syscall # Thuc hien in thong bao malloc success
move $a0,$t0 #Chuyen dia chi tu t0 vao a0
li $v0,34
syscall # mang bat dau tai dia chi : t0
la $a0,notification
li $v0,4
syscall # in thong bao kich thuoc con tro
```

```
move $a0,$t7 #a0 = t7 = kích thước của mảng cấp phát
li $v0,34
syscall
j main
```

```
option2:#Malloc byte
la $a0,byte #Load địa chỉ của byte vào $a0
jal IntDialog
jal Check_value #Kiểm tra xem giá trị có nằm trong khoảng xác định không
move $a1,$a0 #Chuyển địa chỉ của byte vào a1
la $a0,BytePtr #Load địa chỉ của byte vào a0
li $a2,1 #Cho giá trị tại a2 = 1, tức là số byte của biến Byte
jal malloc #output $v0 = địa chỉ bắt đầu cấp phát bởi malloc
move $t0,$v0 #Chuyển địa chỉ từ v0 vào t0
la $a0,success
li $v0,4
syscall
move $a0,$t0 #Chuyển địa chỉ từ t0 vào a0
li $v0,34
syscall
la $a0,notification
li $v0,4
syscall # in thông báo kích thước con trỏ
move $a0,$t7 #a0 = t7 = kích thước của mảng cấp phát
li $v0,34
syscall
j main
```

```
option3:#Malloc word
la $a0,word
jal IntDialog
jal Check_value #Kiểm tra xem giá trị có nằm trong khoảng xác định không
move $a1,$a0 #Chuyển địa chỉ của word vào a1
la $a0,WordPtr #Load địa chỉ của word vào a0
li $a2,4 #Cho giá trị tại a2 = 4, tức là số byte của biến Word
jal malloc #output $v0 = địa chỉ bắt đầu cấp phát bởi malloc
move $t0,$v0 #Chuyển địa chỉ từ v0 vào t0
la $a0,success
li $v0,4
syscall
move $a0,$t0 #Chuyển địa chỉ từ t0 vào a0
li $v0,34
syscall
la $a0,notification
li $v0,4
syscall # in thông báo kích thước con trỏ
```

```

move $a0,$t7 #a0 = t7 = kích thước của mảng cần phát
li $v0,34
syscall
j main

```

```

option4:#In ra giá trị của con trỏ
la $a0,value_output #Chuyển địa chỉ value_output vào a0
li $v0,4
syscall #In ra value_output
li $a0,0 #Trường hợp a0 = 0
jal ptr_value #Thực hiện hàm ptr_value (đòng 385)
jal print_value_or_address #Thực hiện hàm print_value_or_address(đòng 406)
li $a0,1 #Trường hợp a0 = 1
jal ptr_value
jal print_value_or_address
li $a0,2 #Trường hợp a0 = 2
jal ptr_value
jal print_value_or_address
li $a0,3 #Trường hợp a0 = 3
jal ptr_value
jal print_value_or_address
j main

```

```

option5:#In ra địa chỉ của con trỏ
la $a0,address_output
li $v0,4
syscall
li $a0,0 #Trường hợp a0 = 0
jal ptr_address #thực hiện hàm ptr_address(đòng 397)
jal print_value_or_address #Thực hiện hàm print_value_or_address
li $a0,1 #Trường hợp a0 = 1
jal ptr_address
jal print_value_or_address
li $a0,2 #Trường hợp a0 = 2
jal ptr_address
jal print_value_or_address
li $a0,3 #Trường hợp a0 = 3
jal ptr_address
jal print_value_or_address
j main

```

```

# -----
# copy CharPtr1 -> CharPtr2
# print CharPtr2
# a1 = Ptr1
# a2 = Ptr2 -> Sys_TheTopOfFree

```

# -----

option6:#copy string pointer

copy:

```
la $t0,CharPtr1 #Load dia chi CharPtr1 vao t0
la $t1,copied_string #Load dia chi copied_string vao t1
sw $t1,($t0) #Dia chi cua copied_string la gtri cua Ptr1(CharPtr1 -> copied)
la $a2,CharPtr2 #Load dia chi CharPtr2 vao a2
la $a0,Sys_TheTopOfFree #Load dia chi Sys_TheTop vao a0
lw $t5,($a0) #t5 = Gia tri cua Sys_TheTop(Dia chi bat dau cua vung nho tu do)
sw $t5,($a2) #CharPtr2 tro den vung nho tu do(CharPtr2 -> Top)
lw $t2,($a2) #t2 = Dia chi vung nho tu do moi sau khi cap nhat CharPtr2
lw $t4, ($a0) #t4 = Dia chi vung nho tu do truoc khi CharPtr2 duoc cap nhat
```

copy\_loop:

```
lb $t3,($t1) #Load 1byte tu t1 vao t3
sb $t3,($t2) # copy vlueCharPtr1 vao vung bo nho tu do
addi $t4, $t4,1 # tang len de tinh SystopFree moi
addi $t1,$t1,1 # charPtr1[i++]
addi $t2,$t2,1 # charPtr2[i++]
beq $t3,'\0',exit_copy
j copy_loop
```

exit\_copy:

```
sw $t4,($a0) # SystopFree moi
la $a2, CharPtr2 #Load dia chi CharPtr2 vao a2
lw $a0, ($a2) #Load dia chi vung nho tu do
li $v0,4
syscall # in ra noi dung CharPtr2 tro toi
la $a0, Enter
syscall
j main
```

option7:#freeStorage

jal freeStorage #Thuc hien ham freeStorage (dong 319)

la \$a0,free\_storage\_notification

li \$v0,4

syscall #In ra freeStorageNotification

j main

option8:#show storage

la \$a0,storage\_notification

li \$v0,4

syscall

jal storage #Thuc hien ham storage (dong 339)

move \$a0,\$v0 #Chuyen v0 vao a0

```

li $v0,1
syscall #In gia tri bo nho da cap phat
la $a0,bytes
li $v0,4 #In chuoi ki tu bytes ra man hinh
syscall
j main

```

```

option9:#Malloc 2D Array
la $a0,arr1 #Load dia chi arr1 vao $a0
jal IntDialog #doc len in_row
move $s0,$a0 #Load dia chi arr1 vao $s0
la $a0,arr2 #Load dia chi arr2 vao $a0
jal IntDialog #doc len col
move $a1,$s0 # malloc2 2nd param: row
move $a2,$a0 # malloc2 3rd param: col
la $a0,ArrayPtr #Load dia chi con tro ArrayPtr vao $a0
jal Malloc2 # goi malloc2
move $t0,$v0 # luu gia tri tra ve cua Malloc2
la $a0,succcess #In cau lenh succcess
li $v0,4
syscall #In chuoi success
move $a0,$t0
li $v0,34
syscall
li $v0, 4
la $a0, Enter
syscall
move $a0, $s5
li $v0, 34
syscall

```

```

li $v0, 11
li $a0,' '
syscall

```

```

move $a0, $s6
li $v0, 34
syscall
j main

```

```

option10:#setter
la $a0,ArrayPtr
lw $s7,0($a0) # Luu **ArrayPtr vao $s7
beqz $s7,nullptr # if *ArrayPtr==0 error null pointer
la $a0,input_row
jal IntDialog # get row

```

```

move $s0,$a0
la $a0,input_col
jal IntDialog #get col
move $s1,$a0
la $a0,input_value
jal IntDialog #get val
move $a3,$a0 #value
move $a1,$s0 #row
move $a2,$s1 #col
move $a0,$s7 #*ArrayPtr
jal SetArray # SetArray($a0:**ArrayPtr,$a1:hang,$a2:cot,$a3:Gia tri)
j main

```

```

option11:#getter
la $a0,ArrayPtr
lw $s1,0($a0)
beqz $s1,nullptr # if *ArrayPtr==0 error null pointer
la $a0,input_row
jal IntDialog # get row
move $s0,$a0
la $a0,input_col
jal IntDialog #get col
move $a2,$a0 #col
move $a1,$s0 #row
move $a0,$s1 #value
jal GetArray #GetArray(*ArrayPointer,row,col)
move $s0,$v0 # save return value of GetArray
la $a0,output_value
li $v0,4
syscall
move $a0,$s0
li $v0,34
syscall
j main

```

## Các hàm được sử dụng:

```

#-----
#Giai phong bo nho cap phat cho cac bien con tro
freeStorage:
la $t9,Sys_TheTopOfFree #gan dia chi Sys_TheTopOfFree
la $t8,Sys_MyFreeSpace #Gan dia chi Sys_MyFreeSpace

```



```

sw $t8, 0($t9) #Sys_TopOfFree tro den vung nho tu do
la $t0, CharPtr # lay dia chi cua bien con tro dau tien
mul $t2, $t2, 0 #set up t2 = 0
loop:
sll $t1, $t2, 2 #Dich trai 2 bit
addi $t2, $t2, 1 #t2++
addu $t0, $t0, $t1 # lay dia chi tai *CharPtr + 4*$a0
sw $t3, 0($t0) # lay gia tri cua *---Ptr , su dung t3 vi luc nay t3 = 0
beq $t2, 4, exit_free #Neu t2 = 4, nhay den exit_free
j loop
exit_free:
jr $ra
#-----
# Tinh tong luong bo nho da cap phat
# param: none
# return: $v0 - dung luong bo nho da cap phat (byte)
#-----
storage:
la $t9, Sys_TopOfFree #Dia chi cua con tro Sys_TopOfFree
lw $t9, 0($t9) #Dia chi vung nho tu do gan nhat
la $t8, Sys_MyFreeSpace #Dia chi vung nho tu do ban dau
sub $v0, $t9, $t8 #v0 = t8 - t9
jr $ra

```

SysInitMem:

```

la $t9, Sys_TopOfFree # Lay con tro chua dau tien con trong, khoi tao
la $t7, Sys_MyFreeSpace # Lay dia chi dau tien con trong, khoi tao
sw $t7, 0($t9) # Luu lai
jr $ra
#-----
# InputDialogInt
# Lap den khi nao nhap vao thanh cong
#-----
InputDialog:
move $t0, $a0 # luu dia chi cua chuoi ki tu
li $v0, 51
syscall
beq $a1, 0, done # success
beq $a1, -2, end # thoat chuong trinh khi nguoi dung chon "cancel"
move $a0, $t0 # lay lai dia chi cua chuoi ki tu
j InputDialog
done:
jr $ra

#-----
# check: 0<input<1000?

```

#-----

Check\_value:

bge \$a0,1000,over\_value

blez \$a0,negative

jr \$ra

over\_value:

la \$a0,too\_big

j error

negative:

la \$a0,too\_small

j error

#-----

# ptr\_value: ham lay gia tri bien con tro

# param: \$a0 {0:char ; 1:byte ; 2:word ; 3: array}

# return: \$v0 - gia tri cua bien con tro

#-----

ptr\_value:

la \$t0,CharPtr # lay dia chi cua bien con tro dau tien

sll \$t1, \$a0, 2

addu \$t0, \$t0, \$t1 # lay dia chi tai \*CharPtr + 4\*\$a0

lw \$v0, 0(\$t0) # lay gia tri cua bien con tro, luu vao \$v0

jr \$ra

#-----

# ptr\_address: ham lay dia chi bien con tro

# param: \$a0 {0:char ; 1:byte ; 2:word ; 3: array}

# return: \$v0 - dia chi cua bien con tro

#-----

ptr\_address: #Tuong tu nhu ptr\_value, ngoai tru viec luu dia chi cua bien con tro vao v0

la \$t0,CharPtr

sll \$t1, \$a0, 2

addu \$v0, \$t0, \$t1

jr \$ra

#-----

#print\_value\_or\_address: in ra gia tri hoac dia chi cua con tro

#-----

print\_value\_or\_address:

move \$t1,\$a0 # kiem tra lan in cuoi

move \$a0,\$v0

li \$v0,34

syscall

li \$v0,11

beq \$t1,3,end\_print #ngan cach cac output boi cac dau phay, neu \$t1 = 3, tuc gia tri ket thuc,  
nhay den end\_print

```

li $a0, ''
syscall
jr $ra
end_print: li $a0, ''
syscall
jr $ra

```

```

#-----
# Ham cap phat bo nho dong cho cac bien con tro
# param: [in/out] $a0 Chua dia chi cua bien con tro can cap phat
# Khi ham ket thuc, dia chi vung nho duoc cap phat se luu tru vao bien con tro
# param: $a1 - So phan tu can cap phat
# param: $a2 - Kich thuoc 1 phan tu, tinh theo byte
# return: $v0 - Dia chi vung nho duoc cap phat
#-----
malloc:
    la $t9, Sys_TopOfFree # Luu dia chi vao t9
    lw $t8, 0($t9)        # Lay gia tri dau tien cua con tro
    bne $a2, 4, continue  # Neu kich thuoc khong chia het cho 4, bo qua
    addi $t8, $t8, 3       # Do khoi tao cua Sys_Top = 1 nen +3 lam tron thanh gia tri 4 gan
nhat
    andi $t8, $t8, 0xfffffc # Thuc hien phep and tron 3 bit cuoi cung thanh 0, gia tri chia het cho
4 gan nhat
continue:
    sw $t8, 0($a0)        # Luu dia chi vao con tro
    addi $v0, $t8, 0       # Tra ve dia chi cap phat la ket qua cua ham
    mul $t7, $a1, $a2      # Tinh kich thuoc cua mang can cap phat
    add $t8, $t8, $t7      # Tinh dia chi dau tien cua con tro tiep theo
    sw $t8, 0($t9)        # Lu tr? l?i ??a ch? ??u tien vao bi?n Sys_TopOfFree
    jr $ra

```

```

#-----
# Ham cap phat bo nho dong cho mang 2 chieu
# Y tuong: Dua ve cap phat bo nho cho mang 1 chieu voi kich thuoc row * col phan tu, tai su
dung ham malloc
# param: [in/out] $a0 Chua dia chi cua bien con tro can cap phat
# Khi ham ket thuc, dia chi vung nho duoc cap phat se luu tru vao bien con tro
# param: $a1 - So hang
# param: $a2 - so cot
# return: $v0 - Dia chi vung nho duoc cap phat
#-----
Malloc2:

```

```

addi $sp,$sp,-4 # them 1 ngan trong vao stack
sw $ra, 4($sp) # push $ra
bgt $a1,1000,malloc_err # kiem tra loi so luong
bgt $a2,1000,malloc_err # cua hang (cot)
la $s0,row # luu so hang va so cot : row[0]= row, row[1]=col
sw $a1,0($s0) #luu so hang vao row[0]
sw $a2,4($s0) #luu so cot vao row[1]
move $s5, $a1 #hang
move $s6, $a2 #cot
mul $a1,$a1,$a2 #tinh so phan tu va luu vao $a1
li $a2,4 #kich thuoc moi phan tu (word)
jal malloc
lw $ra, 4($sp)
addi $sp,$sp,4
jr $ra

```

```

#-----
# gan gia tri cua trong mang
# param [in] $a0 - Chua dia chi bat dau mang
# param [in] $a1 - hang (i) # @param [in] $a2 cot (j)
# param [in] $a3 - gia tri gan
#-----

```

SetArray:

```

la $s0,row
lw $s1,0($s0)
lw $s2,4($s0)
bge $a1,$s1,bound_err
bge $a2,$s2,bound_err
mul $s0,$s2,$a1
addu $s0,$s0,$a2
sll $s0, $s0, 2
addu $s0,$s0,$a0
sw $a3,0($s0)
jr $ra

```

```

#-----
# lay gia tri cua trong mang
# param [in] $a0 - Chua dia chi bat dau mang
# param [in] $a1 - hang (i)
# param [in] $a2 - cot (j)
# return $v0 - gia tri tai hang a1 cot a2 trong mang
# -----

```

GetArray:

```

la $s0,row # s0 =ptr so ha`ng

```

```
lw $s1,0($s0) #s1 so hang
lw $s2,4($s0) #s2 so cot
bge $a1,$s1,bound_err
bge $a2,$s2,bound_err
mul $s0,$s2,$a1
addu $s0,$s0,$a2 #s0= i*col +j
sll $s0, $s0, 2
addu $s0,$s0,$a0 #s0 = *array + (i*col +j)*4
lw $v0,0($s0)
jr $ra
```

```
#-----
# errors
#-----
malloc_err: # thông báo lỗi số lượng malloc
    la $a0, mal
    j error
bound_err: # thông báo lỗi chỉ số vượt ngoài phạm vi
    la $a0, bound
    j error
nullptr: # thông báo con trỏ rỗng
    la $a0, null
error: # in ra thông báo lỗi
    li $v0,4
    syscall
end: # kết thúc chương trình
    li $v0,10
    syscall
```