

BỘ GIÁO DỤC VÀ ĐÀO TẠO
ĐẠI HỌC BÁCH KHOA HÀ NỘI
Trường Công nghệ thông tin và truyền thông
----- o0o -----



BÁO CÁO FINAL – PROJECT

MÔN: THỰC HÀNH KIẾN TRÚC MÁY TÍNH

Mã học phần: IT3280

Mã lớp: 130939

Sinh viên thực hiện:

Lê Thị Hà

MSSV: 20194544

Phạm Ánh Dương

MSSV: 20194532

Giảng viên hướng dẫn: ThS. Lê Bá Vui

MỤC LỤC

1. Tổng quan	3
1.1 Đề tài được phân công	3
1.1.1 CNC Marsbot	3
1.1.2 Hàm cấp phát bộ nhớ malloc()	3
1.2 Công cụ sử dụng	4
2. Project 4 - Postscript CNC Marsbot	4
2.1 Phân tích cách làm và thuật toán	4
2.2 Mã nguồn	4
2.3 Kết quả chạy mô phỏng	9
- Postscript1: Chữ ‘DCE’	9
- Postscript2: Chữ ‘KTMT’	10
- Postscript3: Chữ ‘LE HA’	10
3. Project 6 - Hàm cấp phát bộ nhớ malloc()	10
3.1 Phân tích cách làm	10
3.2 Thuật toán	10
3.3 Mã nguồn	14
3.4 Kết quả chạy mô phỏng	31
3.4.1 Chức năng 1	31
3.4.2 Chức năng 2	33
3.4.3 Chức năng 3	33
3.4.4 Chức năng 4	33
3.4.5 Chức năng 5	33
3.4.6 Chức năng 6	33
3.4.7 Chức năng 7	33
3.4.8 Chức năng 8	33
3.4.8 Chức năng 9	34

1. Tổng quan

1.1 Đề tài được phân công

1.1.1 CNC Marsbot

Máy gia công cơ khí chính xác CNC Marsbot được dùng để cắt tấm kim loại theo các đường nét được qui định trước. CNC Marsbot có một lưỡi cắt dịch chuyển trên tấm kim loại, với giả định rằng:

- Nếu lưỡi cắt dịch chuyển nhưng không cắt tấm kim loại, tức là Marsbot di chuyển nhưng không để lại vết (Track)
- Nếu lưỡi cắt dịch chuyển và cắt tấm kim loại, tức là Marsbot di chuyển và có để lại vết.

Để điều khiển Marsbot cắt đúng như hình dạng mong muốn, người ta nạp vào Marsbot một mảng cấu trúc gồm 3 phần tử:

- <Góc chuyển động>, <Cắt/Không cắt>, <Thời gian>
- Trong đó <Góc chuyển động> là góc của hàm HEADING của Marsbot
- <Cắt/Không cắt> thiết lập lưu vết/không lưu vết
- <Thời gian> là thời gian duy trì quá trình vận hành hiện tại

Hãy lập trình để CNC Marsbot có thể:

- Thực hiện cắt kim loại như đã mô tả
- Nội dung postscript được lưu trữ cố định bên trong mã nguồn
- Mã nguồn chứa 3 postscript và người dùng sử dụng 3 phím 0, 4, 8 trên bàn phím Key Matrix để chọn postscript nào sẽ được gia công.
- Một postscript chứa chữ DCE cần gia công. Hai script còn lại sinh viên tự đề xuất (tối thiểu 10 đường cắt)

1.1.2 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.

1.2 Công cụ sử dụng

- Chúng em sử dụng công cụ Mars4_5 để thực hiện project

2. Project 4 - Postscript CNC Marsbot

2.1 Phân tích cách làm và thuật toán

- Đọc từ bàn phím Key Matrix để chọn postscript nào sẽ được gia công với hàm polling, nếu người dùng chọn phím 0 chương trình sẽ thực hiện postscript1, người dùng chọn phím 4 chương trình sẽ thực hiện postscript2, người dùng chọn phím 8 chương trình sẽ thực hiện postscript3, nếu chọn phím khác chương trình sẽ không thực hiện.
- Sau khi đã đọc được yêu cầu chạy postscript cụ thể sẽ chạy hàm `READ_PSCRIPT` để đọc nội dung của postscript đó
- Trong hàm `READ_PSCRIPT` có các hàm `READ_ROTATE` để đọc giá trị rotate (quay), `READ_TIME` để đọc giá trị thời gian Marsbot chạy, `READ_TRACK` để đọc giá trị track (giá trị 1 - Marsbot ghi) hoặc untrack (giá trị 0 - Marsbot không ghi)
- Sau khi đã đọc được giá trị thì sẽ gọi các hàm để Marsbot thực hiện

2.2 Mã nguồn

```
# Mars bot

.eqv HEADING 0xffff8010

.eqv MOVING 0xffff8050

.eqv LEAVETRACK 0xffff8020

.eqv WHEREX 0xffff8030

.eqv WHEREY 0xffff8040

# Key matrix
```

```

.eqv OUT_ADRESS_HEXА_KEYBOARD 0xFFFF0014

.eqv IN_ADRESS_HEXА_KEYBOARD 0xFFFF0012

.data

# postscript-DCE => numpad 0

# (rotate,time,0=untrack | 1=track;)

pscript1: .asciiz
"90,2000,0;180,3000,0;180,5790,1;80,500,1;70,500,1;60,500,1;50,500,1;40,500,1;30,500,1;20,500,1;1
0,500,1;0,500,1;350,500,1;340,500,1;330,500,1;320,500,1;310,500,1;300,500,1;290,500,1;280,490,1;9
0,7000,0;270,500,1;260,500,1;250,500,1;240,500,1;230,500,1;220,500,1;210,500,1;200,500,1;190,500,
1;180,500,1;170,500,1;160,500,1;150,500,1;140,500,1;130,500,1;120,500,1;110,500,1;100,500,1;90,50
0,1;90,4000,0;270,2000,1;0,5800,1;90,2000,1;180,2900,0;270,2000,1;90,3000,0;"

# postscript-DAT => numpad 4

pscript2: .asciiz
"90,2000,0;180,3000,0;180,5790,1;0,2695,0;45,3500,1;225,3500,0;135,4000,1;90,3000,0;0,5790,1;270
,2000,1;90,4000,1;90,2000,0;180,5790,1;0,5790,0;135,3000,1;45,3000,1;180,5790,1;90,4000,0;0,5790,
1;270,2000,1;90,4000,1;"

# postscript-DUY => numpad 8

pscript3: .asciiz
"90,4000,0;180,3000,0;180,5790,1;90,2600,1;90,4000,0;270,2000,1;0,5800,1;90,2000,1;180,2900,0;27
0,2000,1;90,3000,0;90,8000,0;90,4000,1;0,2900,0;180,5800,1;270,4000,0;0,5800,1;90,8000,0;195,595
0,1;90,2900,0;345,5950,1;180,3150,0;270,700,0;90,1400,1;90,600,0;"

.text

# <--xu ly tren keymatrix-->

    li $t3, IN_ADRESS_HEXА_KEYBOARD

    li $t4, OUT_ADRESS_HEXА_KEYBOARD

polling:

    li $t5, 0x01          # row-1 of key matrix

    sb $t5, 0($t3)

    lb $a0, 0($t4)

    bne $a0, 0x11, NOT_NUMPAD_0

    la $a1, pscript1

```

```

j START
NOT_NUMPAD_0:
li $t5, 0x02          # row-2 of key matrix
sb $t5, 0($t3)
lb $a0, 0($t4)
bne $a0, 0x12, NOT_NUMPAD_4
la $a1, pscript2
j START
NOT_NUMPAD_4:
li $t5, 0x04          # row-3 of key matrix
sb $t5, 0($t3)
lb $a0, 0($t4)
bne $a0, 0x14, COME_BACK
la $a1, pscript3
j START
COME_BACK: j polling      # khi cac so 0,4,8 khong duoc chon -> quay lai doc tiep
# <!--end-->

# <--xu li mars bot -->
START:
jal GO
READ_PSCRIPT:
addi $t0, $zero, 0      # luu gia tri rotate
addi $t1, $zero, 0      # luu gia tri time
READ_ROTATE:
add $t7, $a1, $t6        # dich ky tu #t7 lưu địa chỉ CỦA A1
lb $t5, 0($t7)           # doc cac ki tu cua pscript
beq $t5, 0, END          # ket thuc pscript

```

```

beq $t5, 44, READ_TIME      # gap ki tu ','
mul $t0, $t0, 10

addi $t5, $t5, -48          # So 0 co thu tu 48 trong bang ascii.
add $t0, $t0, $t5           # cong cac chu so lai voi nhau.
addi $t6, $t6, 1           # tang ky tu can dich chuyen len 1
j READ_ROTATE               # quay lai doc tiep den khi gap dau ','

READ_TIME:                  # doc thoi gian chuyen dong.
add $a0, $t0, $zero
jal ROTATE
addi $t6, $t6, 1
add $t7, $a1, $t6          # ($a1 luu dia chi cua pscript)
lb $t5, 0($t7)
beq $t5, 44, READ_TRACK
mul $t1, $t1, 10
addi $t5, $t5, -48
add $t1, $t1, $t5
j READ_TIME                # quay lai doc tiep den khi gap dau ','

READ_TRACK:
addi $v0,$zero,32          # Keep mars bot running by sleeping with time=$t1
add $a0, $zero, $t1 #T1 LÀ TIME
addi $t6, $t6, 1
add $t7, $a1, $t6
lb $t5, 0($t7)
addi $t5, $t5, -48
beq $t5, $zero, CHECK_UNTRACK # 1=track | 0=untrack
jal UNTRACK

```

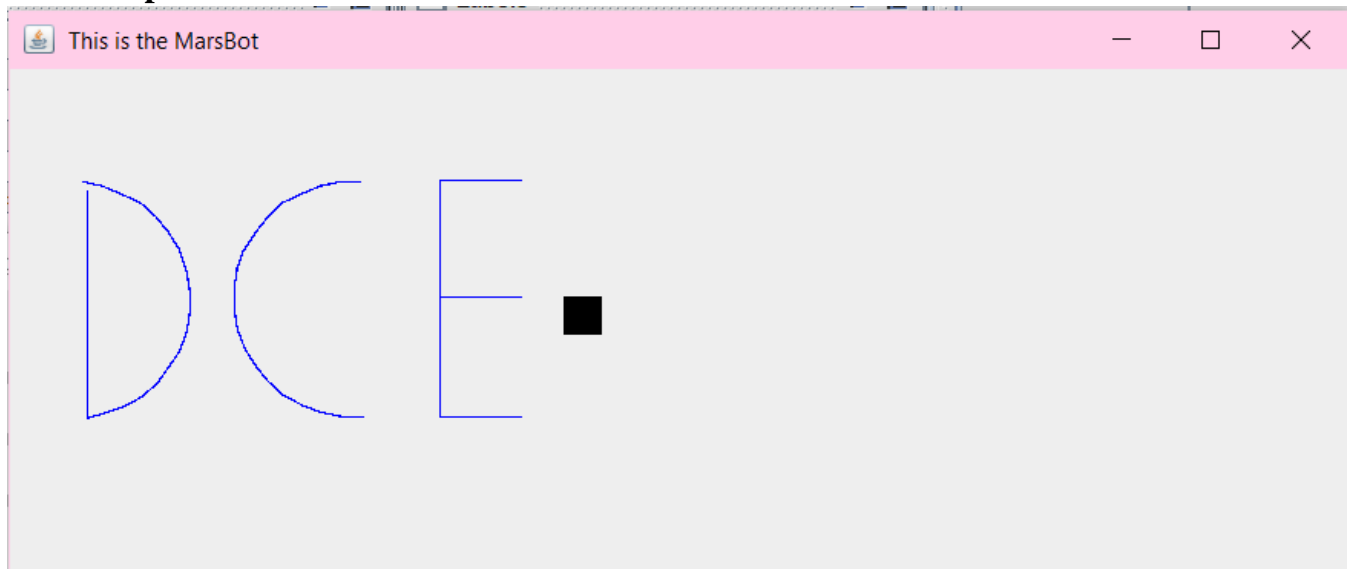
```
jal TRACK
j INCREMENT
CHECK_UNTRACK:
jal UNTRACK
INCREMENT:
syscall
addi $t6, $t6, 2      # bo qua dau ';'
j READ_PSCRIPT
GO:
li $at, MOVING
addi $k0, $zero, 1
sb $k0, 0($at)
jr $ra
STOP:
li $at, MOVING
sb $zero, 0($at)
jr $ra
TRACK:
li $at, LEAVETRACK
addi $k0, $zero, 1
sb $k0, 0($at)
jr $ra
UNTRACK:
li $at, LEAVETRACK
sb $zero, 0($at)
jr $ra
ROTATE:
li $at, HEADING
```



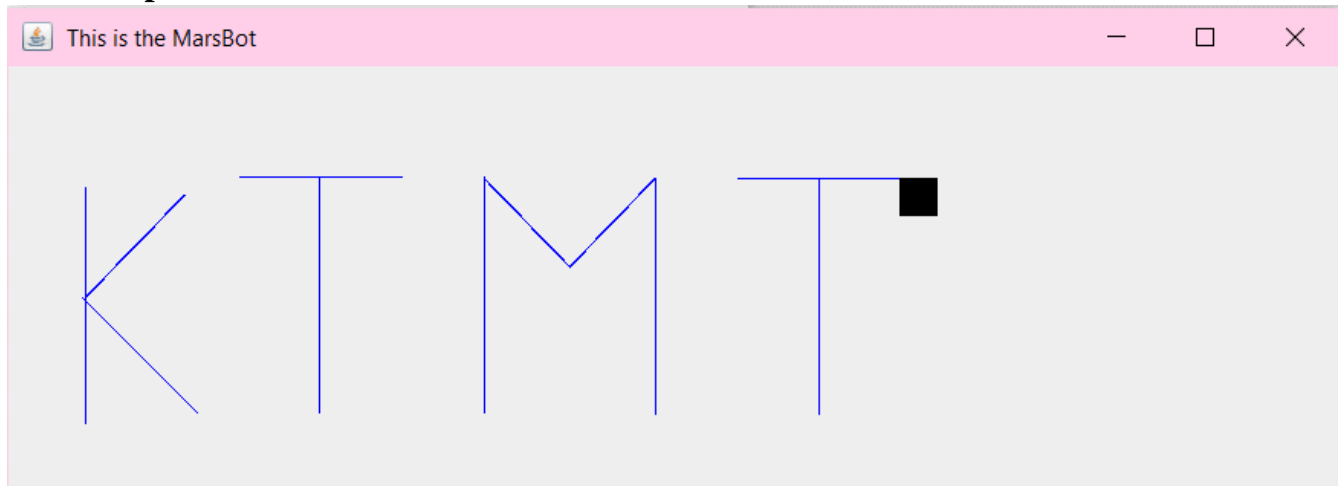
```
sw $a0, 0($at)
jr $ra
END:
jal STOP
li $v0, 10
syscall
j polling
# <!--end-->
```

2.3 Kết quả chạy mô phỏng

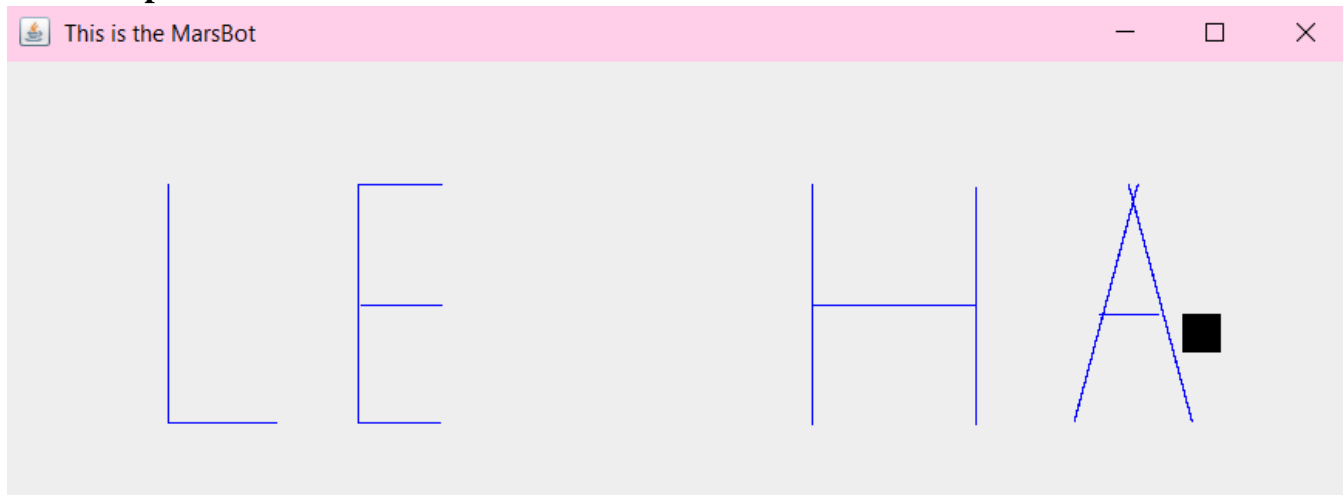
- Postscript1: Chữ ‘DCE’



- Postscript2: Chữ ‘KTMT’



- Postscript3: Chữ ‘LE HA’



3. Project 6 - Hàm cấp phát bộ nhớ malloc()

3.1 Phân tích cách làm

Phân tích đề bài:

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

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.

3.2 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.**

- Message nhập số phần tử mảng char, rồi input dialog để nhập số phần tử, sau đó nhảy đến Check_value để kiểm tra xem có thuộc từ 0-1000 không?
- Sau đó gán cho kích thước của char = 1 byte vào thanh ghi \$a2 = 1.
- Tiếp đó là nhảy đến hàm malloc để cấp phát.
- Đến hàm malloc, 2 dòng đầu là để lấy địa chỉ đầu tiên còn trống trong Sys_TheTopOfFree.
- Tiếp đó, kiểm tra xem \$a2 là thanh ghi chứa kích thước 1 phần tử tính theo byte, nếu = 4 (là kiểu word), thì thực hiện 2 dòng tiếp theo (tăng thêm 3 bit rồi andi để giảm về giá trị gần nhất mà chia hết cho 4).
- Ngược lại thanh ghi \$a2 != 4, nhảy đến skip. Ta sẽ lưu địa chỉ \$t8 đó vào biến con trỏ \$a0, đồng thời là kết quả trả về của hàm malloc lưu vào thanh \$v0.
- Sau đó lấy \$a1 nhân với \$a2 tương ứng với số phần tử cần cấp phát nhân với kích thước 1 phần tử -> ra được kích thước của mảng cần cấp phát.
- Rồi lại tính địa chỉ đầu tiên còn trống, lưu địa chỉ đó vào biến Sys_TheTopOfFree.
- Quay về option1, gán \$t0 = \$v0, thông báo cấp phát thành công, rồi in giá trị từ \$t0 đó thành mã hexa dùng code system là \$v0 = 34.
- Cuối cùng là quay về main để tiếp tục menu.

*** 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.

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

- 3 dòng đầu thì in ra message: địa chỉ của biến con trỏ lần lượt theo Char, Byte, Word, Array.
- 4 block này in ra giá trị các biến con trỏ. Trong đó sẽ gán \$a0 = 0, 1, 2, 3 tương ứng với Char, Byte, Word, Array.
- Ví dụ với kiểu char, các kiểu khác tương tự.
- Nhảy đến hàm value để lấy giá trị các biến con trỏ.
- Nhảy đến hàm print_value_or_address để in giá trị các biến con trỏ đó ra theo mã hexa .cách nhau dấu ; .
- Rồi quay về main tiếp tục thực hiện menu.

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

- Tương tự như option4 nhảy đến hàm value thì option5 ta nhảy đến hàm 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 để in giá trị các biến con trỏ đó ra theo mã hexa cách nhau dấu ;
- Rồi quay về main tiếp tục thực hiện menu.

*** Option6 : Copy 2 con trỏ xâu kí tự.**

- Gắn địa chỉ CharPtr1 vào thanh ghi \$t0.
- Gắn địa chỉ example vào thanh ghi \$t1.
- Ghi giá trị thanh ghi \$t1 vào CharPtr1 -> CharPtr1 trỏ tới example.
- 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 trỏ tới.
- Rồi quay về main , tiếp tục thực hiện menu.

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

- Gọi hàm freeStorage.
- freeStorage:
- Gắn địa chỉ Sys_TopOfFree và Sys_MyFreeSpace lần lượt vào thanh ghi \$t9, \$t8.
- Ghi địa chỉ Sys_MyFreeSpace từ thanh ghi \$t8 vào Sys_TopOfFree.
- Lấy địa chỉ biến con trỏ đầu tiên.
- Gán \$t2 = 0.
- Loop:
- Gán giá trị từng con trỏ bằng 0.
- In ra đã giải phóng bộ nhớ.
- Rồi cùng quay về main , tiếp tục thực hiện menu.

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

- 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.**

- Nhập số hàng (có kiểm tra).
- Nhập số cột (có kiểm tra).
- Lưu giá trị số hàng , số cột vừa nhập lần lượt vào thanh ghi \$a1, \$a2.

- Load địa chỉ con trỏ kiểu Array .
- Nhảy đến hàm Malloc2().
- Ý tưởng cho hàm này là cấp phát bộ nhớ 1 mảng có số hàng * số cột phần tử , sử dụng lại hàm malloc và sử dụng stack.
- Đầu tiên là thêm 1 phần tử vào stack, push \$ra vào, kiểm tra số cột và số hàng có > 1000 thì in ra thông báo lỗi vượt quá bộ nhớ cho phép.
- Còn không thì load row, lưu số hàng đã nhập vào row.
- Lưu số cột đã nhập vào phần tử tiếp theo.
- Nhân \$a1 * \$a2 vào \$a1 được số về phần tử của Array.
- Gán \$a2 = 4 để biết Array là mảng kiểu word (4 bytes).
- Nhảy đến hàm malloc để cấp phát cho mỗi phần tử Array được thanh ghi \$v0 là kết quả trả về hàm.
- Trở về option8, lưu \$v0 vào thanh \$s0, in ra cấp phát thành công.
- In ra giá trị đã cấp phát thành mã hexa.
- Cuối cùng quay về main tiếp tục thực hiện.

*** Option10 : getArray[i][j]**

- 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 dòng 332.
- 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 là tham số giá trị gán vào .
- Quay về main, tiếp tục thực hiện menu .

*** Option11 : lấy ra giá trị đã gán cho array [i][j]**

- 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 ở dòng 352 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.

3.3 Mã nguồn

.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: .word 0

CharPtr2: .word 0

Enter: .asciiz "\n"

row: .word 1

col: .word 1 # tranh nhap nhang

menu: .asciiz "1.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 cot phai nho 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:"

in_row: .asciiz "\n Nhap i:"

in_col: .asciiz "\n Nhap j:"

in_value: .asciiz "\n Nhap gia tri de gan:"

out_value: .asciiz "\n Gia tri tra ve:"

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

```

addr_annoucer: .asciiz "\n Dia chi cua cac bien con tro CharPtr BytePtr WordPtr ArrayPtr la:"
success: .asciiz "\n Malloc success. Dia chi vung nho duoc cap phat : "
annoucer: .asciiz "\n Kich thuoc cua vung nho vua duoc cap phat la "
bound: .asciiz "\n index out of bound"
null: .asciiz "\nNull Pointer Exception. Chua khoi tao mang!!!"
free_storage_annoucer: .asciiz "\nDa giai phong bo nho cap phat! "
storage_annoucer: .asciiz "\nBo nho da cap phat: "
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"
example: .asciiz "example string"
print_matrix: .asciiz "\nSo hang va cot cua mang 2 chieu la:"

```

```
.kdata
```

```

Sys_TheTopOfFree: .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
```

```
main:
```

```
show_menu:
```

```

la $a0,menu
jal IntDialog
move $t0, $a0
beq $t0, 1, op1
beq $t0, 2, op2
beq $t0, 3, op3
beq $t0, 4, op4
beq $t0, 5, op5
beq $t0, 6, op6

```

```

beq $t0, 7, op7
beq $t0, 8, op8
beq $t0, 9, op9
beq $t0, 10, op10
beq $t0, 11, op11
j end

```

op1:#Malloc char

```

la $a0,char
jal IntDialog
jal Check_value
move $a1,$a0
la $a0,CharPtr
li $a2,1
jal malloc          # $v0 = địa chỉ bắt đầu cấp phát bởi malloc
move $t0,$v0
la $a0,success
li $v0,4
syscall            # in thông báo malloc success
move $a0,$t0
li $v0,34
syscall            # mang bắt đầu tại địa chỉ : $t0
la $a0,annoucer
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
op2:#Malloc byte
    la $a0,byte
    jal IntDialog
    jal Check_value
    move $a1,$a0
    la $a0,BytePtr
    li $a2,1
    jal malloc
    move $t0,$v0
    la $a0,success
    li $v0,4
    syscall
    move $a0,$t0
    li $v0,34
    syscall
    la $a0,annoucer
    li $v0,4
    syscall
    move $a0,$t7
    li $v0,34
    syscall
j main
op3:#Malloc word
    la $a0,word
    jal IntDialog
    jal Check_value
    move $a1,$a0
```

```
la $a0,WordPtr
```

```
li $a2,4
```

```
jal malloc
```

```
move $t0,$v0
```

```
la $a0,success
```

```
li $v0,4
```

```
syscall
```

```
move $a0,$t0
```

```
li $v0,34
```

```
syscall
```

```
la $a0,annoucer
```

```
li $v0,4
```

```
syscall
```

```
move $a0,$t7
```

```
li $v0,34
```

```
syscall
```

```
j main
```

op4:#Print value of pointer

```
la $a0,value_annoucer
```

```
li $v0,4
```

```
syscall
```

```
li $a0,0
```

```
jal value
```

```
jal print_value_or_address
```

```
li $a0,1
```

```
jal value
```

```
jal print_value_or_address
```

```
li $a0,2
```

```
    jal value
    jal print_value_or_address
    li $a0,3
    jal value
    jal print_value_or_address
    j main
op5:#Print address of pointer
    la $a0,addr_annoucer
    li $v0,4
    syscall
    li $a0,0
    jal address
    jal print_value_or_address
    li $a0,1
    jal address
    jal print_value_or_address
    li $a0,2
    jal address
    jal print_value_or_address
    li $a0,3
    jal address
    jal print_value_or_address
    j main

# -----
# copy CharPtr1 -> CharPtr2
# print CharPtr2
# a1 = Ptr1
```

```
# a2 = Ptr2 -> Sys_TopOfFree
# -----

op6:#copy string pointer
copy:
    la $t0,CharPtr1
    la $t1, example
    sw $t1,($t0)          #CharPtr1 -> example
    la $a2,CharPtr2
    la $a0,Sys_TopOfFree
    lw $t5,($a0)
    sw $t5,($a2)          #CharPtr2 -> Top
    lw $t2,($a2)
    lw $t4, ($a0)
copy_loop:
    lb $t3,($t1)
    sb $t3,($t2)          # copy vlueCharPtr1 vao vung bo nho tu do
    addi $t4, $t4,1        # tang len de tinh SystopFree
    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
    lw $a0, ($a2)
    li $v0,4
    syscall               # in ra noi dung CharPtr2 tro toi
```

```
    la $a0, Enter
    syscall
    j main
op7:#freeStorage
    jal freeStorage
    la $a0,free_storage_annoucer
    li $v0,4
    syscall
    j main
op8:#show storage
    la $a0,storage_annoucer
    li $v0,4
    syscall
    jal storage
    move $a0,$v0
    li $v0,1
    syscall
    la $a0,bytes
    li $v0,4
    syscall
    j main
op9:#Malloc 2D Array
    la $a0,arr1
    jal IntDialog      #read in_row
    move $s0,$a0
    la $a0,arr2
    jal IntDialog      # read col
    move $a1,$s0        # malloc2 2nd param: row
```

```
move $a2,$a0      # malloc2 3rd param: col
la $a0,ArrayPtr
jal Malloc2        # call malloc2
move $t0,$v0       # save return value of malloc
la $a0,success
li $v0,4
syscall
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
op10:#setter
la $a0,ArrayPtr
lw $s7,0($a0) # Luu **ArrayPtr vao $s7
```

```

beqz $s7,nullptr # if *ArrayPtr==0 error null pointer
la $a0,in_row
jal IntDialog # get row
move $s0,$a0
la $a0,in_col
jal IntDialog #get col
move $s1,$a0
la $a0,in_value
jal IntDialog #get val
move $a3,$a0
move $a1,$s0
move $a2,$s1
move $a0,$s7
jal SetArray # SetArray($a0:**ArrayPtr,$a1:hang,$a2:cot,$a3:Gia tri)
j main

```

op11:#getter

```

la $a0,ArrayPtr
lw $s1,0($a0)
beqz $s1,nullptr # if *ArrayPtr==0 error null pointer
la $a0,in_row
jal IntDialog # get row
move $s0,$a0
la $a0,in_col
jal IntDialog #get col
move $a2,$a0
move $a1,$s0
move $a0,$s1
jal GetArray #GetArray(*ArrayPointer,row,col)

```

```

move $s0,$v0 # save return value of GetArray
la $a0,out_value
li $v0,4
syscall
move $a0,$s0
li $v0,34
syscall
j main

```

#-----

#Giai phong bo nho cap phat cho cac bien con tro

freeStorage:

```

la $t9,Sys_TheTopOfFree
la $t8,Sys_MyFreeSpace
sw $t8, 0($t9)
la $t0,CharPtr      # lay dia chi cua bien con tro dau tien
mul $t2,$t2,0

```

loop:

```

sll $t1, $t2, 2
addi $t2, $t2, 1
addu $t0, $t0, $t1    # lay dia chi tai *CharPtr + 4*$a0
sw $t3, 0($t0)        # lay gia tri cua *---Ptr
beq $t2,4,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_TheTopOfFree
    lw $t9, 0($t9)
    la $t8, Sys_MyFreeSpace
    sub $v0, $t9, $t8
    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 IntDialog

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

#-----

Ham khoi tao cho viec cap phat dong

@param: none

@detail Danh dau vi tri bat dau cua vung nho co the cap phat duoc

#-----

SysInitMem:

la \$t9, Sys_TheTopOfFree # 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

#-----

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 Địa chỉ vùng nhớ được cấp phát
```

```
#-----
```

```
malloc:
```

```
    la $t9, Sys_TopOfFree    # lưu địa chỉ còn trong vào trong $t9
    lw $t8, 0($t9)           # Lấy địa chỉ đầu tiên còn trong trong $t9 (lấy địa chỉ của địa chỉ (C: **))
    bne $a2, 4, skip
    addi $t8, $t8, 3          # trường hợp số không chia hết cho 4
    andi $t8, $t8, 0xfffffc   # lấy địa chỉ chia hết cho 4 về nhất có thể
```

```
skip:
```

```
    sw $t8, 0($a0)           # Cat địa chỉ do vào biến con trỏ
    addi $v0, $t8, 0          # Đồng thời là kết quả trả về của hàm
    mul $t7, $a1, $a2         # Tính kích thước của mảng cần cấp phát
    add $t8, $t8, $t7         # Tính địa chỉ đầu tiên còn trong
    sw $t8, 0($t9)           # Lưu trở lại địa chỉ đầu tiên do vào biến Sys_TopOfFree
    jr $ra
```

```
#-----
```

```
# Hàm cấp phát bộ nhớ động cho mảng 2 chiều
```

```
# @param: [in/out] $a0 Chứa địa chỉ của biến con trỏ cần cấp phát
```

```
# Khi hàm kết thúc, địa chỉ vùng nhớ được cấp phát sẽ lưu trữ vào biến con trỏ
```

```
# @param: $a1 Số hàng
```

```
# @param: $a2 số cột
```

```
# @return: $v0 Địa chỉ vùng nhớ được cấp phát
```

```
#-----
```

```
Malloc2:
```

```
    addi $sp, $sp, -4        # thêm 1 ngăn trong vào stack
    sw $ra, 4($sp)           # push $ra
    bgt $a1, 1000, mal_err   # kiểm tra lỗi số lượng
```

```

    bgt $a2,1000,mal_err      # cua hang (cot)
    la $s0,row                # luu so hang va so cot row[0]= row, row[1]=col
    sw $a1,0($s0)
    sw $a2,4($s0)
    move $s5, $a1             #hang
    move $s6, $a2             #cot
    mul $a1,$a1,$a2
    li $a2,4
    jal malloc
    lw $ra, 4($sp)
    addi $sp,$sp,4
    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

```

```
#-----
```

```
# 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

```

```
#-----
```

```
# ham lay gia tri bien con tro
```

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

```
# @return: $v0 gia tri bien con tro
```

#-----

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 *---Ptr
jr $ra
```

#-----

ham lay dia chi bien con tro

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

@return: \$v0 dia chi bien con tro

#-----

address:

```
la $t0,CharPtr
sll $t1, $a0, 2
addu $v0, $t0, $t1
jr $ra
```

#-----

print value or address of pointer

#-----

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

#-----
# errors
#-----

mal_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

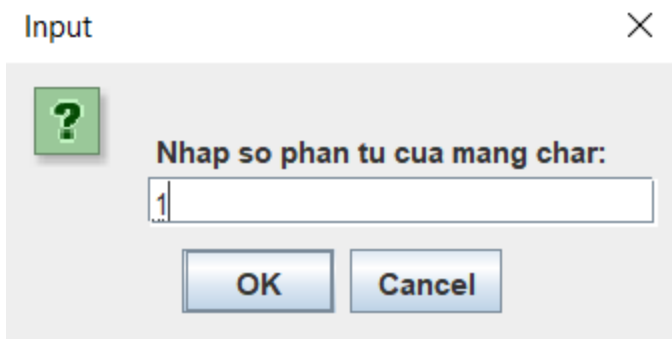
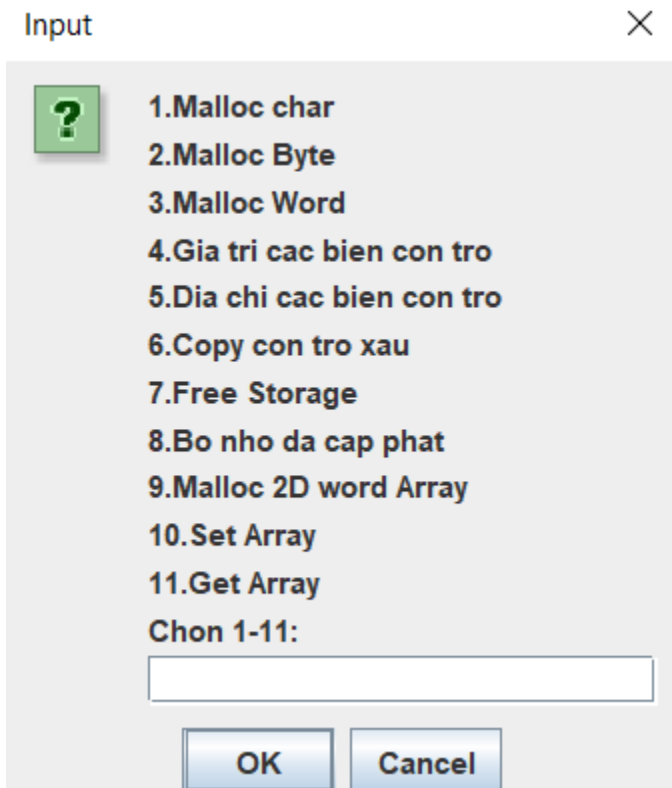
error:        # in ra thông báo lỗi
        li $v0,4
        syscall

end:         # kết thúc chương trình
        li $v0,10
        syscall
```

3.4 Kết quả chạy mô phỏng

3.4.1 Chức năng 1

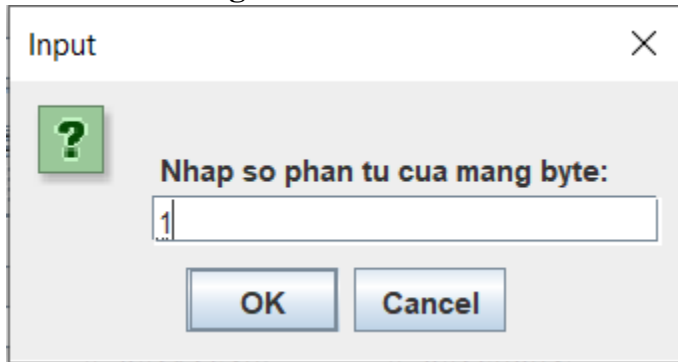
- Input



- **Kết quả:**

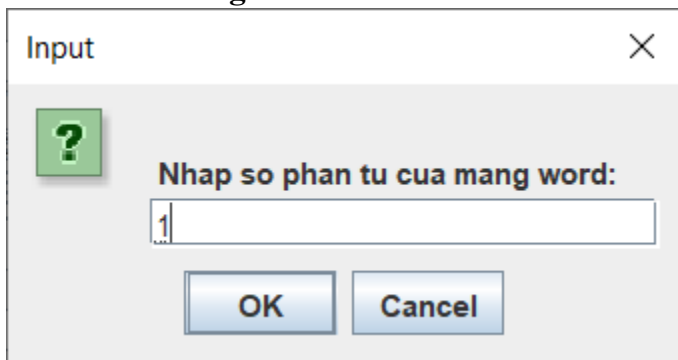
```
Malloc success. Dia chi vung nho duoc cap phat : 0x90000004  
Kich thước của vùng nhớ vừa được cấp phát là 0x00000001
```


3.4.2 Chức năng 2



Malloc success. Địa chỉ vùng nhớ được cấp phát : 0x90000005
Kích thước của vùng nhớ vừa được cấp phát là 0x00000001

3.4.3 Chức năng 3



Malloc success. Địa chỉ vùng nhớ được cấp phát : 0x90000008
Kích thước của vùng nhớ vừa được cấp phát là 0x00000004

3.4.4 Chức năng 4

Giá trị tại các biến con trỏ CharPtr BytePtr WordPtr ArrayPtr là: 0x90000004, 0x90000005, 0x90000008, 0x00000000.

3.4.5 Chức năng 5

Địa chỉ của các biến con trỏ CharPtr BytePtr WordPtr ArrayPtr là: 0x10010000, 0x10010004, 0x10010008, 0x1001000c.

3.4.6 Chức năng 6

example string

3.4.7 Chức năng 7

Đã giải phóng bộ nhớ cấp phát!

3.4.8 Chức năng 8

Bộ nhớ đã cấp phát: 0 bytes.

3.4.8 Chức năng 9

Input

×

?

Nhap so cot cua mang array:

3

OK

Cancel

Input

×

?

Nhap so hang cua mang array:

3

OK

Cancel

Malloc success. Dia chi vung nho duoc cap phat : 0x900000040x00000003,0x00000003