

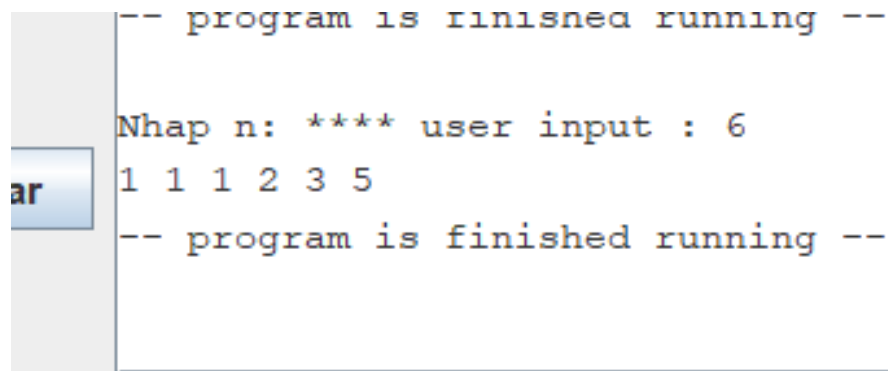
BÁO CÁO THỰC HÀNH KIỂM TRA GIỮA KÌ

KIẾN TRÚC MÁY TÍNH

Họ và tên: Trần Đức Mạnh-MSSV:20225739

Phần A:Nhập số nguyên dương N từ bàn phím, in ra dãy số fibonacci bé hơn N.

Kết quả:



```
-- program is finished running --  
  
Nhap n: **** user input : 6  
1 1 1 2 3 5  
  
-- program is finished running --
```

#code:

.data

prompt: .ascii "Nhap n: "

number: .ascii " "

.text

.globl main

main:

li \$v0, 4 # syscall để in chuỗi

la \$a0, prompt # địa chỉ chuỗi prompt

syscall

li \$v0, 5 # syscall để đọc số nguyên

syscall

move \$s0, \$v0 # lưu giá trị n vào \$s0

li \$t0, 0 # i = 0

print_loop:

move \$a0, \$t0

jal fibonacci # gọi hàm fibonacci(i)

move \$t1, \$v0 # lưu kết quả fibonacci vào \$t1

blt \$t1, \$s0, print_fib # nếu fibonacci(i) < n, in ra

j end_loop

print_fib:

li \$v0, 1 # syscall để in số nguyên

move \$a0, \$t1 # chuyển kết quả fibonacci sang \$a0

syscall

li \$v0, 4 # syscall để in khoảng trắng

la \$a0, number

syscall

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

j print_loop

end_loop:

li \$v0, 10 # syscall để kết thúc chương trình

syscall

Hàm fibonacci

fibonacci:

li \$v0, 0 # f0 = 0

li \$t1, 1 # f1 = 1

li \$t2, 1 # fn = 1

move \$t0, \$a0 # n = \$a0

bltz \$t0, fib_end # nếu n < 0, trả về -1

beqz \$t0, fib_end # nếu n = 0, trả về 0

li \$v0, 1 # Đặt kết quả mặc định là 1 cho n = 1

blez \$t0, fib_end # nếu n <= 1, trả về kết quả

```
li $t3, 2      # i = 2
```

fib_loop:

```
bge $t3, $t0, fib_end # nếu i >= n, kết thúc
```

```
move $t4, $t1    # f0 = f1
```

```
move $t1, $t2    # f1 = fn
```

```
add $t2, $t4, $t1 # fn = f0 + f1
```

```
addi $t3, $t3, 1 # i++
```

```
j fib_loop
```

fib_end:

```
move $v0, $t2    # Trả về kết quả fn
```

```
jr $ra           # Trở về hàm gọi
```

Giải thích:

1. Đoạn khai báo:

.data

prompt: .asciiz "Nhap n: "

number: .asciiz " "

Đoạn trên khai báo dữ liệu được lưu trữ trong bộ nhớ. prompt là một chuỗi thông báo yêu cầu người dùng nhập số n. number là một chuỗi chứa một khoảng trắng, dùng để in ra giữa các số Fibonacci.

2. Hàm Main:

1. In ra thông báo yêu cầu nhập n và đọc giá trị n từ bàn phím.
2. Khởi tạo biến đếm i (được lưu trong \$t0) bằng 0.

3. Vào vòng lặp print_loop để tính và in ra các số Fibonacci nhỏ hơn n.

3. Vòng Lặp In Số Fibonacci:

Vòng lặp print_loop gọi hàm fibonacci với tham số là giá trị hiện tại của i để tính số Fibonacci thứ i.

Nếu số Fibonacci nhỏ hơn n, số đó được in ra, sau đó tăng i và tiếp tục vòng lặp.

Khi số Fibonacci không còn nhỏ hơn n, vòng lặp kết thúc.

4. Hàm Fibonacci:

Hàm fibonacci tính số Fibonacci thứ n (với n là tham số đầu vào).

Sử dụng biến \$t0 để lưu giá trị n, \$t1 và \$t2 để lưu giá trị của hai số Fibonacci gần nhất, và \$t3 là biến đếm trong vòng lặp.

Nếu n nhỏ hơn 0, hàm trả về -1 (tuy nhiên, phần này không được sử dụng trong logic hiện tại của chương trình).

Đối với n bằng 0 hoặc 1, hàm trả về n.

Đối với n lớn hơn 1, hàm sử dụng một vòng lặp để tính giá trị Fibonacci thứ n thông qua công thức cổ điển: $F(n) = F(n-1) + F(n-2)$.

Kết quả được trả về qua \$v0.

5. Kết Thúc Chương Trình:

Sau khi in ra tất cả các số Fibonacci nhỏ hơn n, chương trình kết thúc bằng cách gọi syscall với mã 10 để thoát.

Phần B: . Nhập mảng số nguyên từ bàn phím. In ra tổng các phần tử lẻ và tổng các phần tử âm trong mảng.

#code:

.data

```
arraySpace: .space 400 # Dành chỗ cho 100 số nguyên
promptSize: .ascii "Nhap kích thước mảng: "
promptElement: .ascii "\nNhap phần tử: "
sumOddMsg: .ascii "\nTổng các phần tử lẻ là: "
sumNegativeMsg: .ascii "\nTổng các phần tử âm là: "

.text
.globl main
```

main:

```
# Nhập kích thước mảng
```

```
li $v0, 4
```

```
la $a0, promptSize
```

```
syscall
```

```
li $v0, 5 # Đọc số nguyên
```

```
syscall
```

```
move $t0, $v0 # Lưu kích thước mảng vào $t0
```

```
# Kiểm tra kích thước mảng
```

```
blez $t0, exit # Nếu kích thước <= 0 thì thoát
```

```
# Nhập các phần tử mảng
```

li \$t1, 0 # Index i = 0

li \$t2, 0 # Tổng các phần tử lẻ

li \$t3, 0 # Tổng các phần tử âm

la \$t4, arraySpace # Địa chỉ bắt đầu của mảng

input_loop:

In thông điệp nhập phần tử

li \$v0, 4

la \$a0, promptElement

syscall

Đọc phần tử mảng

li \$v0, 5

syscall

sw \$v0, 0(\$t4) # Lưu phần tử vào mảng

Kiểm tra và cập nhật tổng lẻ

andi \$t5, \$v0, 1 # \$t5 = \$v0 % 2

bnez \$t5, update_odd_sum

Kiểm tra và cập nhật tổng âm

bltz \$v0, update_negative_sum

```
j update_index
```

```
update_odd_sum:
```

```
add $t2, $t2, $v0 # Cập nhật tổng lẻ
```

```
update_negative_sum:
```

```
bltz $v0, add_negative_sum # Chỉ cập nhật tổng âm nếu phần tử < 0
```

```
j update_index
```

```
add_negative_sum:
```

```
add $t3, $t3, $v0 # Cập nhật tổng âm
```

```
update_index:
```

```
addi $t4, $t4, 4 # Cập nhật địa chỉ mảng cho phần tử tiếp theo
```

```
addi $t1, $t1, 1 # Tăng index
```

```
blt $t1, $t0, input_loop # Nếu i < kích thước mảng, tiếp tục vòng lặp
```

```
# In tổng các phần tử lẻ
```

```
li $v0, 4
```

```
la $a0, sumOddMsg
```

```
syscall
```

```
li $v0, 1
```



```
move $a0, $t2
```

```
syscall
```

```
# In tổng các phần tử âm
```

```
li $v0, 4
```

```
la $a0, sumNegativeMsg
```

```
syscall
```

```
li $v0, 1
```

```
move $a0, $t3
```

```
syscall
```

exit:

```
li $v0, 10 # Thoát chương trình
```

```
syscall
```

Kết quả:

Mảng nhập vào: 1 -2 -5 4 7

```
Nhap pha tu: **** user input : 7

Tong cac phan tu le la:  3
Tong cac phan tu am la: -7
-- program is finished running --
```

Giải thích:

Khởi tạo và nhập liệu

Khởi tạo dữ liệu: Sử dụng .data để khai báo vùng nhớ cho mảng (arraySpace), các thông điệp (promptSize, promptElement, sumOddMsg, sumNegativeMsg).

Nhập kích thước mảng: Hiển thị thông điệp yêu cầu nhập kích thước mảng (promptSize), sau đó đọc giá trị nhập vào và lưu vào \$t0.

Kiểm tra và nhập mảng

Kiểm tra kích thước mảng: Nếu kích thước mảng nhập vào (\$t0) nhỏ hơn hoặc bằng 0, chương trình sẽ kết thúc.

Nhập các phần tử mảng: Sử dụng một vòng lặp (input_loop) để nhập từng phần tử của mảng. Mỗi phần tử được lưu vào vùng nhớ arraySpace tại địa chỉ được chỉ định bởi \$t4.

Tính toán tổng

Tính tổng các phần tử lẻ và âm: Trong quá trình nhập mỗi phần tử, chương trình kiểm tra xem phần tử đó có phải là số lẻ (sử dụng andi \$t5, \$v0, 1) hoặc số âm (sử dụng bltz \$v0). Nếu đúng, chương trình sẽ cập nhật tổng tương ứng (\$t2 cho tổng số lẻ, \$t3 cho tổng số âm).

In kết quả

In tổng các phần tử lẻ: Sử dụng syscall để in thông điệp sumOddMsg và sau đó in giá trị tổng các số lẻ (\$t2).

In tổng các phần tử âm: Tương tự như trên, in thông điệp sumNegativeMsg và giá trị tổng các số âm (\$t3).

Kết thúc chương trình

Thoát chương trình: Sử dụng syscall với \$v0 được đặt là 10 để kết thúc chương trình.

Phần C: Nhập vào chuỗi ký tự. Đếm số nguyên âm trong câu. (Các nguyên âm là a, i, u, e, o)

#code

.data

inputString: .space 100 # Dành chỗ cho chuỗi ký tự, giả sử tối đa 100 ký tự

prompt: .ascii "Nhập vào: "

countMsg: .ascii "\nSố nguyên âm trong chuỗi là: "

.text

.globl main

main:

Nhập chuỗi ký tự

li \$v0, 4

la \$a0, prompt

syscall

li \$v0, 8 # Đọc chuỗi ký tự

la \$a0, inputString

li \$a1, 100 # Độ dài tối đa của chuỗi

syscall

Đếm số nguyên âm

li \$t0, 0 # Số nguyên âm

li \$t1, 0 # Index i = 0

lowercase_loop:

lb \$t2, inputString(\$t1) # Đọc ký tự tại vị trí i

Chuyển ký tự về chữ thường nếu là chữ hoa

li \$t3, 65 # 'A' - ASCII

li \$t4, 90 # 'Z' - ASCII

blt \$t2, \$t3, not_uppercase

bgt \$t2, \$t4, not_uppercase

addi \$t2, \$t2, 32 # Chuyển chữ hoa về chữ thường

not_uppercase:

sb \$t2, inputString(\$t1) # Lưu ký tự vào chuỗi

addi \$t1, \$t1, 1 # Tăng index i lên 1

bnez \$t2, lowercase_loop # Nếu chưa kết thúc chuỗi, tiếp tục vòng lặp

Đếm số nguyên âm

li \$t1, 0 # Reset index i = 0

count_vowels_loop:

lb \$t2, inputString(\$t1) # Đọc ký tự tại vị trí i

Kiểm tra ký tự có phải nguyên âm không

li \$t3, 0 # Flag để kiểm tra nguyên âm

li \$t4, 97 # 'a' - ASCII

li \$t5, 111 # 'o' - ASCII

blt \$t2, \$t4, not_vowel

bgt \$t2, \$t5, not_vowel

li \$t3, 1 # Đánh dấu là nguyên âm

beq \$t2, 101, is_vowel_e # Kiểm tra nguyên âm 'e'

beq \$t2, 105, is_vowel_i # Kiểm tra nguyên âm 'i'

beq \$t2, 97, is_vowel_a # Kiểm tra nguyên âm 'a'

beq \$t2, 111, is_vowel_o # Kiểm tra nguyên âm 'o'

beq \$t2, 117, is_vowel_u # Kiểm tra nguyên âm 'u'

not_vowel:

addi \$t1, \$t1, 1 # Tăng index i lên 1

bnez \$t2, count_vowels_loop # Nếu chưa kết thúc chuỗi, tiếp tục vòng lặp

In số nguyên âm

li \$v0, 4

la \$a0, countMsg

syscall

li \$v0, 1

move \$a0, \$t0

syscall

Kết thúc chương trình

li \$v0, 10

syscall

is_vowel_a:

is_vowel_e:

is_vowel_i:

is_vowel_o:

is_vowel_u:

addi \$t0, \$t0, 1 # Tăng số nguyên âm lên 1

j not_vowel

Kết quả:

Với xâu nhập vào là TRANMANH

Kết quả thu được là có 2 nguyên âm

```
program is finished running

Nhap xau: **** user input : TRANMANH

So nguyen am trong xau la: 2

-- program is finished running --
```

Giải thích:

1. Khai báo phần dữ liệu:

inputString: .space 100: Khai báo một vùng nhớ có thể chứa tối đa 100 ký tự để lưu xâu ký tự.

prompt: .asciiz "Nhap xau: ": Chuỗi thông điệp hiển thị để yêu cầu người dùng nhập xâu ký tự.

countMsg: .asciiz "\nSo nguyen am trong xau la: ": Chuỗi thông điệp để in ra số nguyên âm trong xâu.

2. Hàm main:

Hiển thị thông điệp "Nhap xau: " và cho phép người dùng nhập xâu ký tự.

Đọc xâu ký tự từ người dùng và lưu vào vùng nhớ inputString.

Khởi tạo biến \$t0 để đếm số nguyên âm và biến \$t1 để đánh dấu vị trí trong xâu.

3. Vòng lặp chuyển chữ hoa về chữ thường:

Duyệt qua từng ký tự trong xâu và chuyển chữ hoa về chữ thường nếu có.

Lưu ký tự đã chuyển vào xâu.

4. Vòng lặp đếm số nguyên âm:

Duyệt qua từng ký tự trong xâu đã chuyển sang chữ thường và kiểm tra xem ký tự đó có phải nguyên âm không.

Nếu là nguyên âm ('a', 'e', 'i', 'o', 'u'), tăng biến đếm lên 1.

5. Xử lý từng nguyên âm:

Nếu ký tự là 'a', 'e', 'i', 'o', hoặc 'u', tăng biến đếm số nguyên âm lên 1.

6. In số nguyên âm và kết thúc chương trình:

In ra số nguyên âm trong xâu.

Kết thúc chương trình.