**­­­­Computer Architecture Mini Project Report**

**Full name: Nguyễn Trung Kiên**

**Student ID: 20226110**

**Bài 6:** Given an array of word elements and the number of elements, write a procedure to find the pair of adjacent elements that has the largest product and return that product. Example: For inputArray = [3, 6, -2, -5, 7, 3], the output should be the product of 7 and 3 (21).

1. Source code:

.data

    array: .space 400       # Khai báo mảng có thể chứa tối đa 100 phần tử số nguyên (400 bytes)

    size\_of\_array: .word 0     # Biến lưu kích thước của mảng

    newline: .asciiz "\n"   # Chuỗi dùng để xuống dòng mới

    space: .asciiz " "      # Chuỗi dùng để in dấu cách

    cmd1:  .asciiz "Nhap so luong phan tu n (n<=100) cua day : "  # Chuỗi nhắc nhập số lượng phần tử n

    cmd2:  .asciiz "Nhap so thu "    # Chuỗi nhắc nhập giá trị cho phần tử thứ n

    cmd3:  .asciiz ": "              # Chuỗi kết thúc nhắc nhập, theo sau số thứ tự

    cmd4:  .asciiz "Tich lon nhat cua mot cap phan tu lien ke la: "

# Chuỗi thông báo kết quả

    cmd5:  .asciiz "Hai phan tu lien ke do la: "

# Chuỗi thông báo kết quả

    cmd6:       .asciiz "Loi dau vao: Vuot qua kich thuoc mang"

# Chuỗi thông báo lỗi khi nhập sai

.text

main:

    # Nhập số lượng phần tử của mảng từ bàn phím

    # Yêu cầu người dùng nhập số lượng phần tử n, không quá 30

    li $v0, 4

    la $a0, cmd1

    syscall

    li $v0, 5               # Thiết lập syscall để nhập số nguyên

    syscall                 # Yêu cầu hệ điều hành thực hiện nhập số

    # Kiểm tra nếu n > 100 thì báo lỗi

    bgt $v0, 100, warning

    move $t0, $v0           # Lưu số lượng phần tử vào $t0

    sw $t0, size\_of\_array      # Lưu kích thước mảng vào biến size\_of\_array trong bộ nhớ

    # Nhập các phần tử của mảng từ bàn phím

    la $t1, array           # Lấy địa chỉ bắt đầu của mảng

    li $t2, 0               # Khởi tạo biến đếm lặp

input\_loop:

    bge $t2, $t0, finding # Kiểm tra nếu đã nhập đủ số phần tử thì chuyển đến phần tính toán

    li $v0, 4                    # In ra "Nhap so thu"

    la $a0, cmd2

    syscall

    li $v0, 1                    # In chỉ số phần tử

    addi $a0, $t2, 1

    syscall

    li $v0, 4                    # In ra ": "

    la $a0, cmd3

    syscall

    li $v0, 5               # Thiết lập syscall để nhập số nguyên

    syscall                 # Yêu cầu hệ điều hành thực hiện nhập số

    sw $v0, ($t1)           # Lưu giá trị nhập vào mảng

    addi $t1, $t1, 4        # Di chuyển con trỏ đến phần tử tiếp theo trong mảng

    addi $t2, $t2, 1        # Tăng biến đếm lặp

    j input\_loop            # Quay lại đầu vòng lặp để tiếp tục nhập phần tử tiếp theo

finding:

    lw $t3, size\_of\_array      # Lấy kích thước của mảng

    addi $t4, $t3, -1       # Thiết lập giới hạn duyệt mảng (kích thước mảng - 1 vì duyệt cặp liên kề)

    la $t6, array           # Lấy địa chỉ bắt đầu của mảng

    li $t2, 0               # Khởi tạo lại biến đếm để duyệt mảng

    li $a1, 0               # Khởi tạo giá trị max\_product là 0

finding\_loop:

    bge $t2, $t4, print\_result # Kiểm tra nếu đã duyệt hết mảng thì chuyển đến in kết quả

    lw $t7, ($t6)              # Load giá trị của phần tử hiện tại vào $t7

    lw $t8, 4($t6)             # Load giá trị của phần tử kế tiếp vào $t8

    mul $t9, $t7, $t8          # Tính tích của hai phần tử liên tiếp

    bgt $t9, $a1, update\_max # Nếu tích mới lớn hơn giá trị lớn nhất hiện tại thì cập nhật

    j next\_iteration            # Nếu không, chuyển sang lần lặp tiếp theo

update\_max:

    move $a2, $t7               # Cập nhật phần tử đầu của cặp có tích lớn nhất

    move $a3, $t8               # Cập nhật phần tử thứ hai của cặp có tích lớn nhất

    move $a1, $t9               # Cập nhật giá trị tích lớn nhất

next\_iteration:

    addi $t6, $t6, 4            # Di chuyển con trỏ đến phần tử tiếp theo của mảng

    addi $t2, $t2, 1            # Tăng biến đếm lặp

    j finding\_loop            # Quay lại đầu vòng lặp để tiếp tục tính toán

warning:

    # Xuất thông báo lỗi nếu n > 100

    li $v0, 4

    la $a0, cmd6

    syscall

    j end\_program

print\_result:

    li $v0, 4           #In chuỗi thông báo

    la $a0, cmd4

    syscall

    li $v0, 1                   # Thiết lập syscall để in số nguyên

    move $a0, $a1               # Chuyển giá trị tích lớn nhất vào $a0 để in

    syscall                     # In giá trị tích lớn nhất

    li $v0, 4                   # Thiết lập syscall để in chuỗi

    la $a0, newline             # Chuẩn bị chuỗi xuống dòng để in

    syscall                     # In xuống dòng

    li $v0, 4           #In chuỗi thông báo

    la $a0, cmd5

    syscall

    li $v0, 1                   # Thiết lập syscall để in số nguyên

    move $a0, $a2               # Chuyển phần tử đầu tiên của cặp vào $a0 để in

    syscall                     # In phần tử đầu tiên của cặp

    li $v0, 4                   # Thiết lập syscall để in chuỗi

    la $a0, space             # Chuẩn bị chuỗi xuống dòng để in

    syscall                     # In xuống dòng

    li $v0, 1                   # Thiết lập syscall để in số nguyên

    move $a0, $a3               # Chuyển phần tử thứ hai của cặp vào $a0 để in

    syscall                     # In phần tử thứ hai của cặp

end\_program:

    # Kết thúc chương trình

    li $v0, 10

    syscall

II Giải thích:

1. Tổng quan phương pháp xử lý.

* Chương trình thực hiện một số tác vụ chính:

+ Nhắc người dùng nhập số lượng phần tử và các giá trị của mảng.

+ Duyệt qua mảng để tìm cặp phần tử liền kề có tích lớn nhất.

+ In ra tích lớn nhất tìm được cùng với hai phần tử tạo nên tích đó.

1. Phân tích chi tiết.

* Khởi Tạo và Nhập Dữ Liệu

+ Nhập số lượng phần tử: Chương trình yêu cầu người dùng nhập số lượng phần tử của mảng. Số lượng này được lưu trữ trong thanh ghi $t0 và sau đó được lưu trong bộ nhớ tại vị trí size\_of\_array.

+ Nhập giá trị các phần tử: Với mỗi phần tử, chương trình sử dụng một vòng lặp để nhắc người dùng nhập giá trị. Các giá trị này được lưu trực tiếp vào mảng array trong bộ nhớ.

* Tìm Tích Lớn Nhất

+ Khởi tạo: Các biến cần thiết cho việc tìm kiếm như chỉ số hiện tại và giá trị tích lớn nhất được khởi tạo.

+ Vòng lặp kiểm tra: Chương trình duyệt qua mảng, tính tích của mỗi cặp phần tử liền kề và cập nhật giá trị tích lớn nhất cùng với hai phần tử tạo ra tích đó nếu tích mới lớn hơn giá trị đã lưu.

* Xuất Kết Quả

+ In tích lớn nhất: Chương trình in ra chuỗi thông báo và giá trị tích lớn nhất đã tìm được.

+ In hai phần tử tạo nên tích lớn nhất: Sau khi in giá trị tích, chương trình cũng in ra hai phần tử đã tạo nên tích đó.

Bài 7: Ok tall! Example: For a = [-1, 150, 190, 170, -1, -1, 160, 180], the output should be sortByHeight(a) = [-1, 150, 160, 170, -1, -1, 180, 190].

1. Source code: Ta xây dựng 1 kỹ thuật mới với 2 con trỏ để sắp xếp mảng mà không đổi vị trí các số âm.

.data

array\_1:    .space 120           # Khởi tạo không gian cho 30 số nguyên (120 bytes)

newline:    .asciiz "\n"                     # Chuỗi chứa ký tự xuống dòng

cmd1:       .asciiz "Nhap so n (n <= 30): "  # Chuỗi nhắc nhập số lượng phần tử n

cmd2:       .asciiz "Nhap so thu "           # Chuỗi nhắc nhập giá trị cho phần tử thứ n

cmd3:       .asciiz ": "                     # Chuỗi kết thúc nhắc nhập, theo sau số thứ tự

cmd4:       .asciiz " Loi dau vao: Vuot qua kich thuoc mang # Chuỗi thông báo lỗi khi nhập sai

n:          .word 0                          # Biến lưu trữ số lượng phần tử n

space:      .asciiz " "                      # Chuỗi chứa ký tự khoảng trắng dùng để in giá trị

.text

main:

    # Yêu cầu người dùng nhập số lượng phần tử n, không quá 30

    li $v0, 4

    la $a0, cmd1

    syscall

    # Đọc giá trị n từ bàn phím

    li $v0, 5

    syscall

    # Kiểm tra nếu n > 30 thì báo lỗi

    bgt $v0, 30, warning

    # Lưu giá trị n vừa nhập vào biến n

    sw $v0, n

    # Khởi tạo các biến để nhập các phần tử của mảng

    move $t2, $v0                 # $t2 lưu giá trị n

    addi $t2, $t2, 1              # $t2 = n + 1 để sử dụng trong điều kiện vòng lặp

    li $t1, 1                     # $t1 là biến đếm, bắt đầu từ 1

    li $t0, 0                     # $t0 là chỉ số mảng, bắt đầu từ 0

while\_input:

    # Lặp cho đến khi nhập đủ n phần tử

    beq $t1, $t2, sorting        # Kiểm tra điều kiện thoát khỏi vòng lặp

    li $v0, 4                    # In ra "Nhap so thu"

    la $a0, cmd2

    syscall

    li $v0, 1                    # In chỉ số phần tử

    move $a0, $t1

    syscall

    li $v0, 4                    # In ra ": "

    la $a0, cmd3

    syscall

    li $v0, 5                    # Đọc giá trị nhập vào

    syscall

    sw $v0, array\_1($t0)    # Lưu giá trị từ $s0 vào mảng tại địa chỉ $t0

    addi $t0, $t0, 4           # Di chuyển đến vị trí tiếp theo trong mảng

    addi $t1, $t1, 1             # Tăng chỉ số phần tử

    j while\_input                      # Lặp lại

sorting:

    # Sắp xếp mảng (Thuật toán thao tác với 2 con trỏ (cụ thể trong báo cáo) để sắp xếp số dương và giữ nguyên vị trí số âm)

    lw $t2, n                   # Lấy số phần tử là n

    mul $t2, $t2, 4             # Đổi n thành byte

    li $t1, 0                   # Khởi tạo biến đếm cho vòng lặp ngoài

loop1:

    beq $t1, $t2, print         # Kiểm tra điều kiện thoát vòng lặp ngoài

    add $t3, $t1, 4             # Khởi tạo biến đếm cho vòng lặp trong

loop2:

    beq $t3, $t2, inci          # Kiểm tra điều kiện thoát vòng lặp trong

    lw $s0, array\_1($t3)        # Lấy giá trị phần tử kế tiếp

    lw $s1, array\_1($t1)        # Lấy giá trị phần tử hiện tại

    ble $s0, $s1, checking\_1        # So sánh và nhảy nếu thứ tự đúng

    j skip

checking\_1:

    bgt $s0, 0, checking\_2          # Kiểm tra nếu số dương

    j skip

checking\_2:

    bgt $s1, 0, swaping            # Kiểm tra nếu số dương, nhảy đến đổi chỗ

    j skip

swaping:

    sw $s0, array\_1($t1)        # Đổi chỗ hai phần tử

    sw $s1, array\_1($t3)

skip:

    addi $t3, $t3, 4            # Tăng chỉ số biến đếm trong

    j loop2                     # Tiếp tục vòng lặp trong

inci:

    addi $t1, $t1, 4            # Tăng chỉ số biến đếm ngoài

    j loop1                      # Tiếp tục vòng lặp ngoài

print:

    # In mảng đã sắp xếp

    li $t0, 0

    li $t1, 0

    lw $t2, n

print\_loop:

    beq $t1, $t2, end\_program          # Kiểm tra nếu đã in hết mảng thì thoát

    li $v0, 1

    lw $a0, array\_1($t0)        # Lấy giá trị từ mảng tại vị trí $t0

    syscall                     # In giá trị phần tử

    li $v0, 4

    la $a0, space

    syscall                     # In một khoảng trắng

    addi $t0, $t0, 4            # Chuyển đến phần tử tiếp theo trong mảng

    addi $t1, $t1, 1            # Tăng bộ đếm

    j print\_loop                # Lặp lại cho đến khi in hết mảng

warning:

    # Xuất thông báo lỗi nếu n > 30

    li $v0, 4

    la $a0, cmd4

    syscall

end\_program:

    # Kết thúc chương trình

    li $v0, 10

    syscall

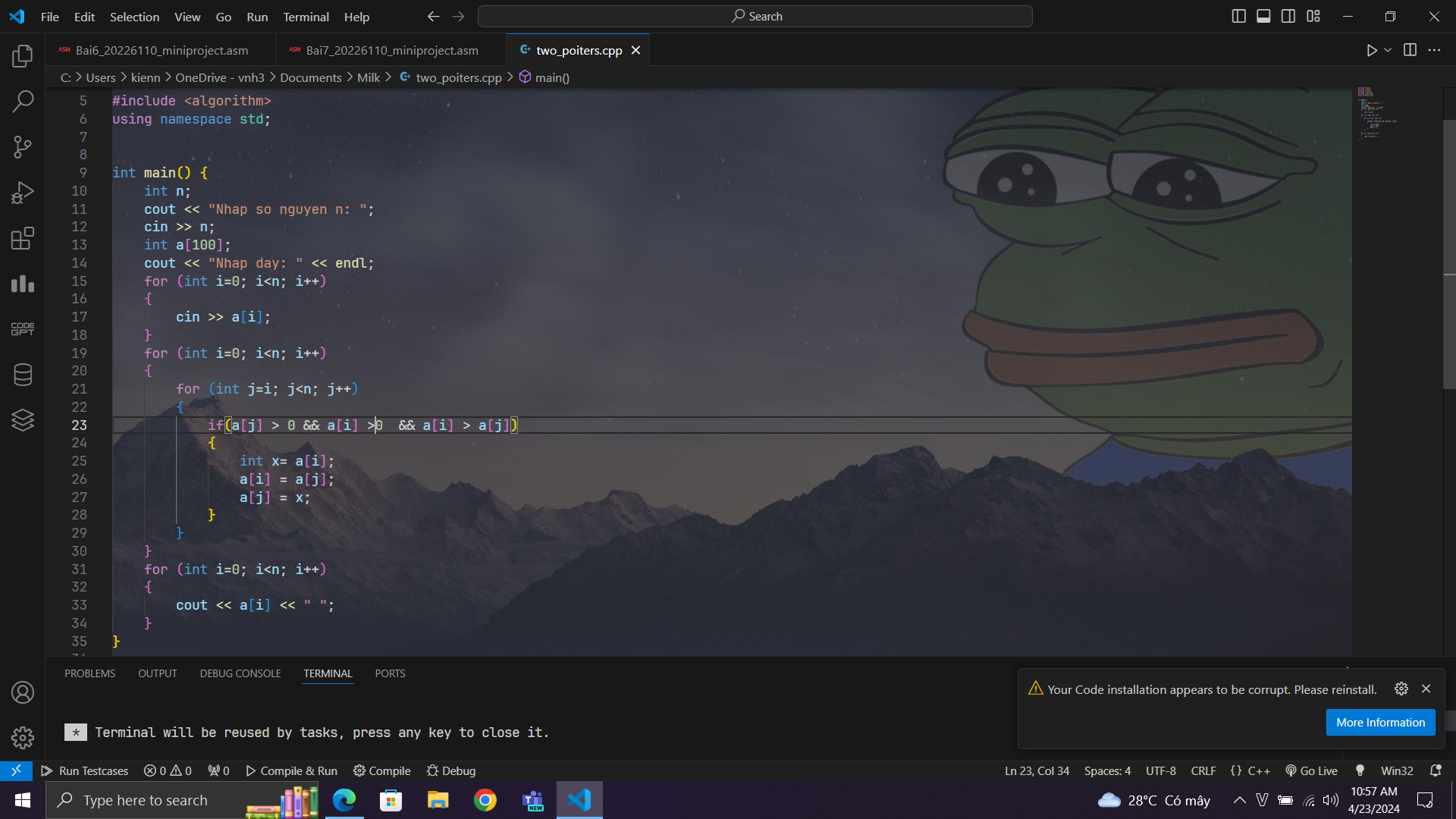
II Giải thích:

1. Tổng quan phương pháp xử lý.

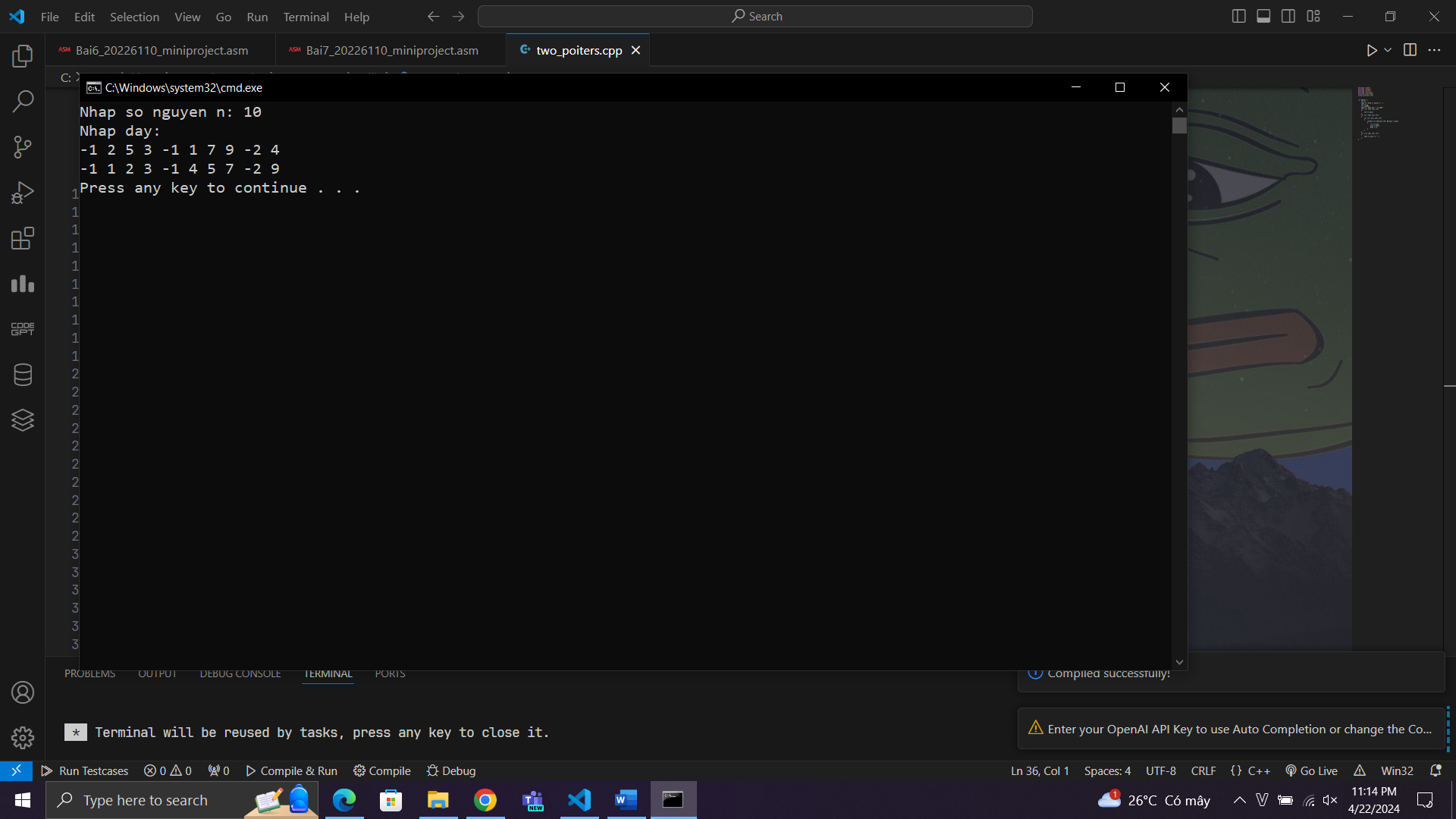
* Ta nhập số phần tử của mảng (tối đa 30) và cho vòng lặp để nhập các giá trị của mảng.
* Sử dụng một thuật toán để sắp xếp lại mảng (bỏ qua những phần tử số nguyên âm, chỉ sắp xếp các số nguyên dương).

1. Thuật toán đề xuất để sắp xếp mảng.

* Bước 1: Đặt con trỏ thứ nhất tại vị trí a[i] (ban đầu i=0). Nếu a[i] < 0 ta nhảy i lên 1.
* Bước 2: Duyệt a[j] (i+1<= j <= n). Nếu a[i] < a[j] ta đổi chỗ 2 giá trị. Như vậy, sau khi duyệt xong 1 vòng của j, số nhỏ nhất được đưa vào vị trí a[i], đồng thời vị trí của các số âm trong mảng vẫn không thay đổi.
* Bước 3: Sau khi duyệt hết 1 vòng j, ta đã có số nhỏ nhất nằm ở vị trí đúng. Tăng index lên 1 và tiếp tục lặp lại vòng lặp.
* Dưới đây là miêu tả code C++ của thuật toán em đề xuất:



* Dưới đây là kết quả thử nghiệm:



1. Phân tích chi tiết.

* Khởi Tạo và Nhập Dữ Liệu

+ Yêu cầu nhập số phần tử: Chương trình bắt đầu bằng cách yêu cầu người dùng nhập số lượng phần tử của mảng, với điều kiện số phần tử không vượt quá 30.

+ Nếu số lượng phần tử nhập vào lớn hơn 30, chương trình sẽ hiển thị thông báo lỗi và kết thúc.

+ Nhập các phần tử của mảng: Người dùng được yêu cầu nhập từng giá trị của mảng. Các giá trị này được lưu vào một mảng được cấp phát bộ nhớ trước.

* Xử Lý Dữ Liệu

+ Kiểm tra điều kiện nhập: Sau khi nhập số phần tử, chương trình kiểm tra xem giá trị đó có vượt quá 30 hay không. Nếu vượt quá, chương trình sẽ chuyển đến nhãn warning để thông báo lỗi và kết thúc chương trình.

+ Sắp xếp mảng: Sử dụng thuật toán đã đề xuất, mảng được sắp xếp theo thứ tự tăng dần với điều kiện chỉ sắp xếp các số dương. Số âm sẽ giữ nguyên vị trí ban đầu trong mảng.

* Xuất Kết Quả

+ In tích lớn nhất: Chương trình in ra chuỗi thông báo và giá trị tích lớn nhất đã tìm được.

+ In mảng đã sắp xếp: Sau khi sắp xếp xong, chương trình sẽ in ra các giá trị của mảng đã được sắp xếp, cách nhau bởi khoảng trắng.