Status	Finished
Started	Friday, 6 December 2024, 7:17 AM
Completed	Tuesday, 10 December 2024, 1:49 PM
Duration	4 days 6 hours
Marks	120.00/120.00
Grade	**10.00** out of 10.00 (**100**%)

Question 1

Correct

Mark 10.00 out of 10.00

Con trỏ - Pointers

Con trỏ (Pointers), tham chiếu (References) và cấp phát bộ nhớ động (Dynamic Memory Allocation) là những tính năng cực kì mạnh mẽ trong ngôn ngữ lập trình C/C++. Những tính năng này cho chép các lập trình viên quản lý và tối ưu tài nguyên của máy tính một cách hiểu quả nhất. Tuy nhiên, đây cũng là nội dung khá phức tạp và gây khó khăn cho người mới bắt đầu.

Con trỏ (Pointers) cho phép chúng ta truy cập và kiểm soát nội dung của các vùng nhớ trong bộ nhớ máy. Nếu được sử dụng đúng cách, con trỏ có thể giúp tăng hiệu năng của chương trình rất nhiều. Tuy nhiên, sử dụng con trỏ không hợp lý có thể dẫn tới rất nhiều vấn đề khác nhau, từ việc làm cho mã nguồn (code) trở nên khó đọc và khó bảo trì cho đến gây ra các lỗi nguy hiểm như thất thoát bộ nhớ (memory leaks) hay tràn bộ nhớ đệm (buffer overflow). Các hacker có thể dựa vào những lỗi này để gây hại cho chương trình của bạn. Nhiều ngôn ngữ lập trình gần đây (như Java hay C#) đã loại bỏ các cú pháp sử dụng con trỏ để tránh những rắc rối mà nó có thể gây ra.

1. Biến con trỏ

Trong máy tính của chúng ta, mỗi ô nhớ (computer memory location) có một địa chỉ (address) và dữ liệu (content) tại ô nhớ đó. Địa chỉ (address) của ô nhớ là một số thường được biểu diễn ở hệ 16 (hexa), ví dụ `0x7ffc7570aa94`. Giá trị của ô nhớ thường được lưu ở dạng nhị phân (binary), người lập trình có thể dịch nó thành các kiểu dữ liệu khác nhau như số nguyên `int`, số thực `double`, kí tự `char` hay xâu `string`.

Tuy nhiên, thông thường các lập trình viên thường gặp khó khăn trong việc sử dụng con trỏ một cách trực tiếp, vì thế các ngôn ngữ lập trình đã đưa ra khai niệm biến (variable). Thông qua tên của biến, lập trình viên có thể truy cập đến dữ liệu của vùng nhớ có địa chỉ mà hệ điều hành đã cấp phát cho biến đó. Cùng với tên biến, lập trình viên cũng phải xác định kiểu của biến (ví dụ: `int`,`double`,`char`) để giúp chúng ta có thể dễ dàng dịch nội dung của biến từ dạng nhị phân (kiểu dữ liệu dùng để lưu trong bộ nhớ máy tính) sang dạng dữ liệu phù hợp mà con người có thể đọc và hiểu được.

Mỗi ô nhớ trong bộ nhớ máy tính thường có kích cỡ 8 bit (1 byte). Để lưu một biến kiểu nguyên `int` 4-bytes trong bộ nhớ, chúng ta cần sử dụng 4 ô nhớ.

Hình vẽ dưới đây biểu diễn mối quan hệ giữa địa chỉ (address) và nội dung (content) của bộ nhớ máy tính (computer memory) với tên (name), kiểu (type) và giá trị (value) của biến được sử dụng khi lập trình.

1.1 BIẾN CON TRỎ (POINTER VARIABLE)

Một biến con trỏ (pointer), giống như một biến thông thường khác như `int` hay `double`, được dùng để lưu một giá trị nào đó. Nhưng thay vì được dùng để lưu các giá trị số nguyên (`int`), số thực (`double`) ..., thì biến con trỏ lưu địa chỉ (address) của một vùng nhớ trong bộ nhớ máy tính.

1.2 KHAI BÁO BIẾN CON TRỎ

Tương các biến thông thường khác, biến con trỏ cần phải được khai báo trước khi sử dụng. Để khai báo một biến con trỏ, chung ta cần phải thêm đấu `*` vào trước tên biến. Biến con trỏ cũng cần được khai báo kiểu, ví dụ `int`, `double`, `boolean`.

```
// Cách khai báo con trỏ
type *ptr; // Cách 1
// hoặc
type* ptr; // Cách 2
// hoặc
type *ptr; // Cách 3
```

Ví dụ, chúng ta có thể khai báo các biến con trỏ như sau:

```
        int *iPtr; // Khai báo một biến con trỏ kiểu int tên là iPtr trỏ đến một vùng nhớ chứa số nguyên.
double * dPtr; // Khai báo một biến con trỏ kiểu double
```

Chú ý rằng, chúng ta cần phải đặt một dấu * trước **từng** tên biến con trỏ mà chúng ta muốn khai báo. Điều này có nghĩa là dấu * chỉ có tác dụng với tên biến đi theo ngay sau đó. Khi viết theo cú pháp khai báo con trò, dấu * không phải là một phép toán nhân mà chúng ta hay dùng.

```
    int *p1, *p2, i; // p1 và p2 là biến con trỏ, i là một biến thông thường kiểu int.
int* p1, p2, i; // p1 là biến con trỏ, p2 và i là biến thông thường kiểu int.
int * p1, * p2, i; // p1 và p2 là biến con trỏ, i là biến thông thường kiểu int.
```

Quy ước đặt tên biến: Chúng ta nên đặt tên cho biến con trỏ bắt đầu bằng chữ `p` hoặc kết thúc bằng chữ `Ptr` để giúp mã nguồn của chúng ta dễ đọc và bảo trì hơn. Ví dụ: `iPtr`, `numberPtr`, `pNumber`, `pStudent`.

1.3 KHỞI TẠO BIẾN CON TRỎ SỬ DỤNG PHÉP & (ADDRESS-OF-OPERATOR)

Ban đầu, khi khai báo một biến con trỏ, nội dung của biến con trỏ sẽ được gán bằng một địa chỉ ngẫu nhiên nào đó mà chúng ta không quan tâm tới (địa chỉ không hợp lệ). Điều này **cực kì nguy hiểm** vì nó có thể gây hại đến các các chương trình khác đang chạy trong máy tính. Vì vậy, chúng ta cần phải khởi tạo biến con trỏ bằng cách gán nó với một địa chỉ mà chúng ta muốn làm việc với nó. Chúng ta có thể gán con trỏ với địa chỉ của một biến thông thường thông qua phép &.

Phép & (address-of-operator) trả về địa chỉ của biến thông thường. Ví dụ, nếu biến `number` là một biến thông thường kiểu `int`, lệnh `&number` sẽ trả về địa chỉ trong bộ nhớ máy tính mà biến `number` được lưu.

Ví dụ:

```
int number = 88; // Khai báo biến thông thường và gán gía trị
int *pNumber; // Khai bảo biến con trỏ
pNumber = &number; // Khởi tạo biến con trỏ, gán giá trị con trỏ bằng địa chỉ của biến number

int *numberPtr = &number; // Chúng ta cũng có thể khởi tạo con trỏ ngay sau lệnh khai báo
```

Như đã mô tả ở hình trên, biến `number` chứa số nguyên `int` có giá trị bằng 88, và được lưu trong 4 ô nhớ bắt đầu từ ô có địa chỉ `0x22ccec`. Câu lệnh `&number` trả về địa chỉ của biến `number` (là địa chỉ của ô nhớ đầu tiên, `0x22ccec`). Địa chỉ này sau đó được gán vào (khởi tạo) biến `pNumber`.

1.4 PHÉP LẤY GIÁ TRỊ CON TRỎ *

Khi sử dụng dấu * trước một biến con trỏ, chúng ta sẽ lấy được giá trị của ô nhớ mà con trỏ đó chỉ tới. Ví dụ, nếu `pNumber` là một biến con trỏ kiểu `int`, lệnh `*pNumber` sẽ trả về giá trị của ô nhớ mà `pNumber` chỉ tới.

Ví dụ:

```
    int number = 98; // Khai báo biến kiểu int
int* pNumber = &number; // Khai báo biến con trỏ kiểu int và khởi tạo nó bằng địa chỉ của biến number
cout << pNumber << endl; // In ra giá trị của con trỏ pNumber, là một địa chỉ
cout << *pNumber << endl; // In ra giá trị của ô nhớ mà biến pNumber chỉ đến
*pNumber = 99; // Sửa giá trị của vùng nhớ mà biến pNumber chỉ tới.
cout << *pNumber << endl; // In ra gía trị của vùng nhớ vừa bị thay đổi
cout << number << endl; // Giá trị của biến number cũng thay đổi theo.
```

Lưu ý: Phép * có ý nghĩa khác nhau khi sử dụng trong câu lệnh khai báo và trong biểu thức. Khi dấu * được sử dụng trong khai báo biến (ví dụ: `int * pNumber;`), nó chỉ ra rằng biến được khai báo ngay phía sau đấu * là một biến con trỏ. Khi sử dụng trong một biểu thức (ví dụ: `*pNumber = 99; std::cout << *pNumber;`), nó trả về giá trị của vùng nhớ mà con trỏ chỉ tới.

1.5 KIỂU CỦA BIẾN CON TRỎ

Trong quá trình khai báo biến con trỏ, lập trình viên phải xác định kiểu dữ liệu của con trỏ đó. Khi đã khai báo, một con trỏ chỉ có thể chứa địa chỉ của vùng nhớ lưu kiểu dữ liệu đã khai báo cho con trỏ đó và không thể chứa kiểu dữ liệu khác.

Ví dụ **đúng**:

```
    int i = 88;
double d = 55.66;
int * iPtr = &i;    // Con trỏ kiểu int trỏ đến vùng nhớ chứa dữ liệu kiểu int
double * dPtr = &d; // Con trỏ kiểu double trỏ đến vùng nhớ chứa dữ liệu kiểu double
```

Ví dụ **sai**:

```
        int i = 88;
double d = 55.66;
int * iPtr = &d;    // LỖI: Con trỏ kiểu int không được chứa địa chỉ vùng nhớ chứa dữ liệu kiểu double
double * dPtr = &i; // LỖI: Con trỏ kiểu double không được chứa địa chỉ vùng nhớ chứa dữ liệu kiểu int
iPtr = i; // LỖI: Con trỏ chỉ chứa địa chỉ, không chứa giá trị.
```

1.6 LỖI CON TRỎ CHƯA KHỞI TẠO

Đoạn code sau đây có một lỗi nguy hiểm mà các lập trình viên hay mắc phải:

```
    int * iPtr;
*iPtr = 55;
cout << *iPtr << endl;
```

Biến con trỏ `iPtr` đã được khai báo nhưng chưa khởi tạo. Vì vậy, hệ điều hành sẽ gán cho nó ngẫu nhiên một giá trị địa chỉ trỏ đến một vùng nhớ ngẫu nhiên nào đó trong máy tính, vùng nhớ này có thể nằm ngoài quyền kiểm soát của chương trình đang chạy. Khi thực thi lệnh `*iPtr = 55;`, chương trình vô hình dung thay đổi giá trị ở một ô nhớ nào đó một cách không hợp pháp. Điều này có thể gây ra lỗi và làm cho chương trình ngừng hoạt động. Hơn nữa, hầu hết các trình biên dịch (compiler) không thông báo lỗi (error) hay cảnh báo (warning) cho lập trình viên về lỗi này trong quá trình biên dịch, nên chúng ta phải tự kiểm tra lại các lỗi kiểu như trên trước khi biên dịch chương trình.

1.7 CON TRỎ RỖNG - NULL

Chúng ta có thể khởi tạo một biến con trỏ bằng giá trị `0` hoặc `NULL`, khi đó con trỏ sẽ không trỏ đến bất cứ một vùng nhớ nào cả. Truy cập vào con trỏ có giá trị `NULL` sẽ gây ra lỗi `STATUS_ACCESS_VIOLATION`.

```
    int * iPtr = 0; // Khai báo biến con trỏ iPtr và gán nó với giá trị NULL
cout << *iPtr << endl; // LỖI: STATUS_ACCESS_VIOLATION
int *p = NULL; // Cách khác để khai con trỏ Null.
```

Bài tập

Hàm `double* getPointerToPi()` có nhiệm vụ như sau:

- Khai báo một biến kiểu `double` đặt tên là `pi` ở ngoài hàm (biến toàn cục) và gán giá trị cho của số $\pi = 3.14159$ cho biến đó.
- Khai báo một biến con trỏ kiểu `double*` và khởi tạo con trỏ bằng địa chỉ của biến `pi` đã khai báo ở trên.
- Trả về con trỏ đã khai báo.

Hãy viết mã mã C++ để hoàn thành hàm `double* getPointerToPi()` có yêu cầu như trên.

Answer: (penalty regime: 0 %)

```
1 double pi = 3.14159;
2 double* getPointerToPi()
3 {
```

```
    double *ptr = &pi;
    return (ptr);
}

```

Passed all tests! ✓

Correct

Marks for this submission: 10.00/10.00.

Question 2

Correct

Mark 10.00 out of 10.00

[Normalization]

3. Cấp phát bộ nhớ động - (Dynamic Memory Allocation)

3.1 TOÁN TỬ NEW VÀ DELETE

Thay vì khai báo một biến `int` (`int number`) và dùng địa chỉ của biến đó để khởi tạo cho một biến con trỏ khác (`int *pNumber = &number`). Lập trình viên có thể yêu cầu hệ điều hành cấp động một vùng nhớ dụng toán tử `new` và dùng con trỏ để lưu lại địa chỉ của vùng nhớ đó. Trong C++, bất kể khi nào bạn sử dụng toán tử `new` để cấp phát động, bạn cần phải sử dụng toán tử `delete` để giải phòng vùng nhớ đó khi không sử dụng đến nó nữa. Trong một số ngôn ngữ khác, như Java, thì việc dọn rác (gabbage collection) được thực hiện một cách tự động.

Toán tử `new` trả về địa chỉ của vùng nhớ được cấp phát. Toán tử `delete` nhận con trỏ làm đối số và giải phóng vùng nhớ mà con trỏ đó trỏ tới.

Ví dụ:

```
    // Cấp phát tĩnh
int number = 88;
int * p1 = &number;  // Gán địa chỉ của một biến vào con trỏ

// Cấp phát động
int * p2;            // Chưa được khởi tạo, con trỏ được trỏ đến một vùng nhớ ngẫu nhiên trong bộ nhớ
cout << p2 << endl; // In ra địa chỉ trước khi cấp phát bộ nhớ
p2 = new int;       // Cấp phát bộ nhớ động và gán địa chỉ đã cấp phát cho con trỏ

*p2 = 99;
cout << p2 << endl;  // In địa chỉ sau khi đã cấp phát động
cout << *p2 << endl; // In ra giá trị ô nhớ mà con trỏ chỉ tới
delete p2;           // Giải phóng vùng nhớ đã cấp phát động
```

Để khởi tạo vùng nhớ đã được cấp phát, cúng ta có thể sư dụng bộ khởi tạo (initializer) đối với các biến kiểu cơ bản (nguyên thủy) hoặc gọi hàm khởi tạo đối với các đối tượng (object) với từ khóa `new` theo phía trước.

Ví dụ:

```
    // Sử dụng bộ khởi tạo với các biến kiêu nguyên thủy.
int * p1 = new int(88);
double * p2 = new double(1.23);

// Gọi hàm khởi tạo đối với các đối tượng (ví dụ Date, Time)
Date * date1 = new Date(1999, 1, 1);
Time * time1 = new Time(12, 34, 56);
```

3.2 MẢNG ĐỘNG VỚI TOÁN TỬ NEW [] VÀ DELETE []

Thay vì sử dụng mảng có kích thước cố định (mảng tĩnh), chúng ta có thể cấp phát mảng động bằng cách sử dụng toán tử `new[]`. Để giải phóng vùng nhớ mà đã cấp phát cho mảng động, chúng ta sử dụng toán tử `delete []`.

```
/* Cấp phát mảng động */ #include #include using namespace std; int main() { const int SIZE = 5; int * pArray; pArray = new int[SIZE]; // Cấp phát mảng động thông qua toán tử new[] // Gán mỗi phần tử của mảng với một số ngẫu nhiên nằm trong khoảng 1 và 100 for (int i = 0; i < SIZE; ++i) { *(pArray + i) = rand() % 100; } // In ra
```

mảng for (int i = 0; i < SIZE; ++i) { cout << *(pArray + i) << " "; } cout << endl; delete[] pArray; // Giải phóng vùng nhớ thông qua toán tử new[] return 0; }

4. Con trỏ, mảng và hàm

4.1 MẢNG LÀ CON TRỎ

Trong C/C++, tên của mảng là con trỏ, chỉ đến địa chỉ của phần từ đầu tiên của mảng. Ví dụ, biến `numbers` là một mảng kiểu `int`, thì `numbers` đồng thời cũng là một con trỏ kiểu `int`, chỉ đến ô nhớ chứa phần tử `numbers[0]`. Vì vậy, `numbers` là `&numbers[0]`, `*number` là `numbers[0]`, `*(numbers+i)` là `numbers[i]`.

Ví dụ:

```
/* Con trỏ và mảng */
#include
using namespace std;

int main() {
    const int SIZE = 5;
    int numbers[SIZE] = {11, 22, 44, 21, 41};  // Mảng số nguyên int

    // Tên mảng numbers là một con trỏ, trỏ đến phần tử đầu tiền của mảng
    cout << &numbers[0] << endl; // In ra địa chỉ của phần tử đầu tiền của mảng
    cout << numbers << endl;     // Tương tự như trên
    cout << *numbers << endl;          // Giống như numbers[0] (11)
    cout << *(numbers + 1) << endl;    // Giống như numbers[1] (22)
    cout << *(numbers + 4) << endl;    // Giống như numbers[4] (41)
}
```

4.2 CÁC PHÉP TOÁN TRÊN CON TRỎ

Nếu `numbers` là một mảng `int`, nó được xem như là con trỏ chỉ đến phần tử đầu tiên của mảng `numbers[0]`. Khi đó, `numbers+1` trỏ đến phần tử tiếp theo, chứ không phải địa chỉ của ô nhớ tiếp theo trong bộ nhớ. Nhớ rằng, kiểu `int` có kích thước `4` bytes. Vì vậy, `numbers +1` sẽ chỉ đến ô nhớ sau ô nhớ hiện tại mà `numbers` trỏ tới 4 ô.

```
int numbers[] = {11, 22, 33};
int * iPtr = numbers;
cout << iPtr << endl;        // In ra địa chỉ hiện tại (ở dạng hexa): 0x22cd30 (2280752)
cout << iPtr + 1 << endl;    // In ra địa chỉ sau khi cộng 1: 0x22cd34 (2280756 = 2280752 + 4)
cout << *iPtr << endl;       // 11
cout << *(iPtr + 1) << endl; // 22
cout << *iPtr + 1 << endl;   // 12
```

4.3 PHÉP LẤY KÍCH THƯỚC`SIZEOF`

Phép `sizeof(arrayName)` trả về tổng số `bytes` (kích thước) của mảng. Chúng ta có thể tính số phần tử của mảng bằng cách chia số `bytes` của cả mảng cho kích thước của một phần tử trong mảng. Ví dụ:

```
int numbers[100];
cout << sizeof(numbers) << endl;     // Kích cỡ của cả mảng (400)
cout << sizeof(numbers[0]) << endl;  // Kích cỡ của một phần tử trong mảng
cout << "Array size is " << sizeof(numbers) / sizeof(numbers[0]) << endl;  // Số lượng phần tử trong mảng (100)
```

4.4 TRUYỀN MẢNG VÀO HÀM

Khi mảng được truyền vào hàm, trình biên dịch sẽ xem nó như một con trỏ. Khi khai báo đối số cho hàm, chúng ta có thể sử dụng cú pháp mảng `int[]` hoặc cú pháp con trỏ `int *`. Ví dụ, những cách khai báo hàm dưới đây là giống nhau.

```
int max(int numbers[], int size);
int max(int *numbers, int size);
int max(int number[50], int size);
```

Tất cả đối số `numbers` trong các khai báo trên đều được xem như là một biến con trỏ. Hơn nữa, khi truyền vào hàm, thông tin về kích cỡ của mảng sẽ bị mất, vì vậy thông thường chúng ta thường phải truyền thêm một đối số `size` kèm theo tên của mảng để xác định kích cỡ của mảng. Khi truy cập các phần tử của mảng trong hàm, trình biên dịch cũng không kiểm tra liệu chúng ta có truy cập ngoài mảng hay không.Vì thế, chúng ta phải cẩn thận kiểm tra khi sử dụng mảng trong hàm.

Bài tập

Một véc-tơ n chiều: $\vec{x} = (x_1, x_2, \ldots, x_n)$ có thể biểu diễn bằng một mảng gồm n số. Trong nhiều bài toán người ta muốn giá trị các chiều của véc-tơ nằm trong đoạn $[0, 1]$ (hoặc $[-1, 1]$) tức là $x_i \in [0, 1] \forall i$. Một cách để làm việc đó là chia các phần tử của mảng cho số lớn nhất có thể có của các phần tử đó.

Hàm `void normalize(double *out, int *in, int n)` nhận các tham số là:

- Con trỏ trỏ đến mảng đầu vào `in`. Mảng đầu vào chứa các số nguyên trong đoạn $[0, 255]$.
- Con trỏ trỏ đến mảng đầu ra `out`. Mảng đầu ra là mảng chuẩn hóa của mảng đầu vào, chứa các số thực sau khi chia số nguyên tương ứng của mảng đầu vào cho 255.
- Số nguyên `n` là số phần tử của hai mảng.

Nhiệm vụ của hàm `void normalize(double *out, int *in, int n)` là chuẩn hóa các giá trị trong mảng đầu vào `in` về khoảng $[0, 1]$ và lưu vào mảng đầu ra `out`.

Hãy viết mã C++ để hoàn thành hàm `void normalize(double *out, int *in, int n)` thực hiện các yêu cầu trên.

For example:

Input	Result
5 78 167 90 119 99	0.306 0.655 0.353 0.467 0.388

Answer: (penalty regime: 0 %)

```
#include <bits/stdc++.h>
void normalize(double *out, int *in, int n)
{
    for(int i = 0; i < n; i++)
    {
        cin >> *(in + i);
        *(out + i) = *(in + i) * 1.0 / 255.0;
    }

}
```

	Input	Expected	Got	
✓	5 78 167 90 119 99	0.306 0.655 0.353 0.467 0.388	0.306 0.655 0.353 0.467 0.388	✓
✓	20 245 23 104 106 252 139 243 252 136 58 97 196 158 253 53 234 203 95 83 214	0.961 0.090 0.408 0.416 0.988 0.545 0.953 0.988 0.533 0.227 0.380 0.769 0.620 0.992 0.208 0.918 0.796 0.373 0.325 0.839	0.961 0.090 0.408 0.416 0.988 0.545 0.953 0.988 0.533 0.227 0.380 0.769 0.620 0.992 0.208 0.918 0.796 0.373 0.325 0.839	✓
✓	10 208 203 241 194 179 126 162 188 83 201	0.816 0.796 0.945 0.761 0.702 0.494 0.635 0.737 0.325 0.788	0.816 0.796 0.945 0.761 0.702 0.494 0.635 0.737 0.325 0.788	✓
✓	64 241 32 145 162 65 86 100 240 245 94 196 213 228 45 44 200 36 54 82 36 169 54 11 253 38 29 2 130 143 236 140 225 25 148 179 179 49 45 51 181 70 23 167 179 57 173 226 6 253 138 157 240 199 104 5 85 176 0 141 237 45 55 166 29	0.945 0.125 0.569 0.635 0.255 0.337 0.392 0.941 0.961 0.369 0.769 0.835 0.894 0.176 0.173 0.784 0.141 0.212 0.322 0.141 0.663 0.212 0.043 0.992 0.149 0.114 0.008 0.510 0.561 0.925 0.549 0.882 0.098 0.580 0.702 0.702 0.192 0.176 0.200 0.710 0.275 0.090 0.655 0.702 0.224 0.678 0.886 0.024 0.992 0.541 0.616 0.941 0.780 0.408 0.020 0.333 0.690 0.000 0.553 0.929 0.176 0.216 0.651 0.114	0.945 0.125 0.569 0.635 0.255 0.337 0.392 0.941 0.961 0.369 0.769 0.835 0.894 0.176 0.173 0.784 0.141 0.212 0.322 0.141 0.663 0.212 0.043 0.992 0.149 0.114 0.008 0.510 0.561 0.925 0.549 0.882 0.098 0.580 0.702 0.702 0.192 0.176 0.200 0.710 0.275 0.090 0.655 0.702 0.224 0.678 0.886 0.024 0.992 0.541 0.616 0.941 0.780 0.408 0.020 0.333 0.690 0.000 0.553 0.929 0.176 0.216 0.651 0.114	✓
✓	100 63 111 67 123 244 98 148 66 27 115 126 159 90 211 131 90 82 91 185 49 121 248 150 185 211 176 97 221 148 239 148 226 80 57 98 151 252 134 35 243 159 69 205 251 188 74 71 42 68 37 44 204 92 57 177 140 32 48 213 221 1 55 157 101 246 60 97 170 241 24 29 168 228 86 86 6 2 65 75 137 163 182 206 189 113 51 62 113 128 110 26 33 214 72 181 185 246 218 30 164	0.247 0.435 0.263 0.482 0.957 0.384 0.580 0.259 0.106 0.451 0.494 0.624 0.353 0.827 0.514 0.353 0.322 0.357 0.725 0.192 0.475 0.973 0.588 0.725 0.827 0.690 0.380 0.867 0.580 0.937 0.580 0.886 0.314 0.224 0.384 0.592 0.988 0.525 0.137 0.953 0.624 0.271 0.804 0.984 0.737 0.290 0.278 0.165 0.267 0.145 0.173 0.800 0.361 0.224 0.694 0.549 0.125 0.188 0.835 0.867 0.004 0.216 0.616 0.396 0.965 0.235 0.380 0.667 0.945 0.094 0.114 0.659 0.894 0.337 0.337 0.024 0.008 0.255 0.294 0.537 0.639 0.714 0.808 0.741 0.443 0.200 0.243 0.443 0.502 0.431 0.102 0.129 0.839 0.282 0.710 0.725 0.965 0.855 0.118 0.643	0.247 0.435 0.263 0.482 0.957 0.384 0.580 0.259 0.106 0.451 0.494 0.624 0.353 0.827 0.514 0.353 0.322 0.357 0.725 0.192 0.475 0.973 0.588 0.725 0.827 0.690 0.380 0.867 0.580 0.937 0.580 0.886 0.314 0.224 0.384 0.592 0.988 0.525 0.137 0.953 0.624 0.271 0.804 0.984 0.737 0.290 0.278 0.165 0.267 0.145 0.173 0.800 0.361 0.224 0.694 0.549 0.125 0.188 0.835 0.867 0.004 0.216 0.616 0.396 0.965 0.235 0.380 0.667 0.945 0.094 0.114 0.659 0.894 0.337 0.337 0.024 0.008 0.255 0.294 0.537 0.639 0.714 0.808 0.741 0.443 0.200 0.243 0.443 0.502 0.431 0.102 0.129 0.839 0.282 0.710 0.725 0.965 0.855 0.118 0.643	✓

Passed all tests! ✓

Correct

Marks for this submission: 10.00/10.00.

Question 3

Correct

Mark 10.00 out of 10.00

[Cube]

Lũy thừa mũ 3 được sử dụng để tính thể tích của khối lập phương.

Viết hàm `void cube(double *p)` nhận tham số là con trỏ đến một biến thực. Hàm thực hiện phép chỉnh sửa giá trị của biến mà con trỏ trỏ đến thành lập phương giá trị đó.

For example:

Input	Result
124.45319	1927605.24319

Answer: (penalty regime: 0 %)

```
#include<bits/stdc++.h>
void cube(double *p)
{
    //p = new double;
    //cin >> *p;
    *p = pow(*p, 3);
    //cout << fixed << setprecision(5) << *p;
}

```

	Input	Expected	Got	
✓	124.45319	1927605.24319	1927605.24319	✓
✓	60.20472	218218.52843	218218.52843	✓
✓	50.08412	125631.96202	125631.96202	✓
✓	68.60589	322912.01745	322912.01745	✓

	Input	Expected	Got	
✓	267.24079	19085706.49073	19085706.49073	✓

Passed all tests! ✓

Correct

Marks for this submission: 10.00/10.00.

Question 4

Correct

Mark 10.00 out of 10.00

[HigherOrLower]

Cho một dãy gồm \(n\) số nguyên và một ngưỡng nguyên (threshold). Viết hàm kiểm tra xem các số trong dãy cao hơn hay thấp hơn ngưỡng cho trước.

Hàm `bool* isHigher (int* arr, int num, int thres)` nhận đầu vào là mảng \(arr\) có \(n\) số nguyên và một ngưỡng \(thres\).

Hàm kiểm tra và trả về một mảng số kiểu \(bool\) với phần tử thứ \(i\) là \(true\) nếu số nguyên thứ \(i\) trong mảng \(arr\) lớn hơn hoặc bằng ngưỡng \(thres\), và bằng \(false\) trong trường hợp ngược lại.

For example:

Input	Result
6 67 13 99 14 41 20 15	1 0 1 0 1 1

Answer: (penalty regime: 0 %)

```
bool* isHigher (int* arr, int num, int thres)
{
   bool* result = new bool[num];
   for(int i = 0; i < num; i++)
   {
       *(result + i) = (*(arr + i) >= thres);
   }
   return result;
}
```

	Input	Expected	Got	
✓	6 67 13 99 14 41 20 15	1 0 1 0 1 1	1 0 1 0 1 1	✓
✓	15 27 52 81 82 21 31 63 72 9 63 84 73 10 39 12 17	1 1 1 1 1 1 1 1 0 1 1 1 0 1 0	1 1 1 1 1 1 1 1 0 1 1 1 0 1 0	✓
✓	11 99 57 96 13 40 6 98 95 28 53 28 17	1 1 1 0 1 0 1 1 1 1 1	1 1 1 0 1 0 1 1 1 1 1	✓
✓	21 83 67 74 87 39 35 36 56 65 65 40 6 12 3 29 96 15 81 66 38 85 52	1 1 1 1 0 0 0 1 1 1 0 0 0 0 0 1 0 1 1 0 1	1 1 1 1 0 0 0 1 1 1 0 0 0 0 0 1 0 1 1 0 1	✓
✓	57 53 88 76 73 55 62 72 17 20 65 22 88 94 67 81 30 52 41 18 40 41 62 14 95 77 8 79 19 53 72 71 66 28 39 68 85 38 15 63 74 88 89 18 42 50 62 52 69 98 98 60 91 49 36 22 60 65 97	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0	✓

Passed all tests! ✓

Correct

Marks for this submission: 10.00/10.00.

Question 5

Correct

Mark 10.00 out of 10.00

[ImageFilter]

Giả sử bạn được thuê xây dựng một hệ thống xử lý ảnh.

Trong đó có một bước lọc ảnh. Tại đây, bạn phải đọc ảnh vào và kiểm tra giá trị tại từng điểm ảnh (**pixel**) có lớn hơn hoặc bằng một ngưỡng **(threshold)** cố định không.

Nếu có, giá trị tại pixel đó giữ nguyên, nếu không, giá trị tại đó được gán giá trị bằng 0.

Viết hàm `int** getImage (int nRows, int nCols)` đọc vào một ảnh có kích thước $nRows \times nCols$. Hàm trả về một con trỏ trỏ tới con trỏ lưu trữ ảnh có kiểu **int****.

Viết hàm `void fillImage (int** image, int nRows, int nCols, int threshold)` thực hiện việc lọc ảnh với ngưỡng cho trước $threshold$. Hàm lọc và thay đổi giá trị của ảnh lưu bằng con trỏ $image$.

Cuối cùng, viết hàm `void print (int** image, int nRows, int nCols)` để in ra ảnh sau khi đã lọc.

Lưu ý: Các điểm ảnh là số nguyên có giá trị nằm trong khoảng từ 0 đến 255.

For example:

Input	Result
5 2 126 82 26 132 81 153 106 185 207 85 95	126 0 0 132 0 153 106 185 207 0

Answer: (penalty regime: 0 %)

```
#include <bits/stdc++.h>
using namespace std;
#define ll long long
#define ull unsigned long long
#define el "\n"
#define se second
#define fi first
#define en end()
#define be begin()
#define sz size()
#define Faster ios_base::sync_with_stdio(0); cin.tie(0); cout.tie(0);

int** getImage (int nRows, int nCols)
{
    int** image = new int*[nRows];
    for(int i = 0; i < nRows; i++)
    {
        image[i] = new int[nCols];
    }
    for(int i = 0; i < nRows; i++)
    {
        for(int j = 0; j < nCols; j++)
            cin >> image[i][j];
    }
    return image;
}
void fillImage (int** image, int nRows, int nCols, int threshold)
{
    for(int i = 0; i < nRows; i++)
```

```
    {
        for(int j = 0; j < nCols; j++)
        {
            if(image[i][j] < threshold) image[i][j] = 0;
        }
    }
}
void print (int** image, int nRows, int nCols)
{
    for(int i = 0; i < nRows; i++)
    {
        for(int j = 0; j < nCols; j++)
            cout << image[i][j] << " ";
        cout << '\n';
    }
}
```

	Input	Expected	Got	
✓	5 2 126 82 26 132 81 153 106 185 207 85 95	126 0 0 132 0 153 106 185 207 0	126 0 0 132 0 153 106 185 207 0	✓
✓	15 9 94 253 109 105 36 233 95 207 205 186 196 203 29 69 231 27 103 1 252 165 133 87 97 70 49 44 185 191 201 71 64 56 185 173 239 204 41 253 195 216 28 36 94 118 133 144 57 54 235 93 46 215 57 59 189 197 82 212 26 1 43 249 29 52 1 219 73 243 232 122 234 122 114 231 99 177 239 154 4 136 217 211 58 250 27 7 57 0 61 218 45 236 72 20 30 15 145 243 149 94 100 89 86 53 74 157 185 31 93 235 91 217 192 215 6 130 138 26 249 230 23 35 211 146 154 197 104 127 131 169 131 115 102 162 109 29	94 253 109 105 36 233 95 207 205 186 196 203 29 69 231 0 103 0 252 165 133 87 97 70 49 44 185 191 201 71 64 56 185 173 239 204 41 253 195 216 0 36 94 118 133 144 57 54 235 93 46 215 57 59 189 197 82 212 0 0 43 249 29 52 0 219 73 243 232 122 234 122 114 231 99 177 239 154 0 136 217 211 58 250 0 0 57 0 61 218 45 236 72 0 30 0 145 243 149 94 100 89 86 53 74 157 185 31 93 235 91 217 192 215 0 130 138 0 249 230 0 35 211 146 154 197 104 127 131 169 131 115 102 162 109	94 253 109 105 36 233 95 207 205 186 196 203 29 69 231 0 103 0 252 165 133 87 97 70 49 44 185 191 201 71 64 56 185 173 239 204 41 253 195 216 0 36 94 118 133 144 57 54 235 93 46 215 57 59 189 197 82 212 0 0 43 249 29 52 0 219 73 243 232 122 234 122 114 231 99 177 239 154 0 136 217 211 58 250 0 0 57 0 61 218 45 236 72 0 30 0 145 243 149 94 100 89 86 53 74 157 185 31 93 235 91 217 192 215 0 130 138 0 249 230 0 35 211 146 154 197 104 127 131 169 131 115 102 162 109	✓

	Input	Expected	Got	
✓	16 3 217 64 12 91 107 46 70 204 109 128 222 184 194 216 127 112 49 134 214 69 101 166 93 38 240 190 151 152 59 225 72 214 48 35 0 139 31 132 253 137 28 218 195 226 1 93 172 25 17	217 64 0 91 107 46 70 204 109 128 222 184 194 216 127 112 49 134 214 69 101 166 93 38 240 190 151 152 59 225 72 214 48 35 0 139 31 132 253 137 28 218 195 226 0 93 172 25	217 64 0 91 107 46 70 204 109 128 222 184 194 216 127 112 49 134 214 69 101 166 93 38 240 190 151 152 59 225 72 214 48 35 0 139 31 132 253 137 28 218 195 226 0 93 172 25	✓

	Input	Expected	Got	
✓	42 22 13 167 0 87 219 60 71 44 221 84 28 104 47 133 143 213 22 201 20 153 124 66 221 66 153 114 52 105 117 224 67 179 59 228 62 154 176 112 174 48 6 125 128 216 249 124 58 103 151 179 227 203 224 238 5 193 172 90 131 184 208 104 183 156 134 8 229 35 60 241 14 21 176 188 125 211 194 135 20 104 220 90 52 168 130 42 216 93 153 161 76 84 152 247 247 197 241 46 130 178 204 46 8 171 193 143 35 92 227 35 235 62 158 22 72 114 196 222 58 114 245 160 170 194 226 194 70 218 99 149 87 64 227 188 222 230 248 4 174 81 181 5 148 212 34 82 182 169 239 55 221 246 136 124 116 17 24 74 230 85 2 176 225 204 12 129 133 26 13 234 183 185 154 191 46 180 207 224 86 60 14 211 239 198 38 90 234 144 185 82 124 76 196 160 246 163 70 126 98 61 88 216 97 85 111 201 66 42 91 113 160 38 6 66 3 238 135 5 72 103 99 161 234 113 101 232 250 250 88 249 222 220 206 137 165 88 9 212 178 25 34 145 37 238 134 191 26 73 55 100 119 61 232 191 69 58 68 25 88 2 216 209 41 18 122 176 30 79 250 158 14 167 81 86 6 28 189 187 212 226 120 129 201 155 224 200 181 33 209 79 202 180 252 170 71 56 183 162 7 193 143 186 208 186 70 173 171 184 78 192 50 150 71 216 145 33 223 153 39 217 62 106 31 134 60 100 104 15 231 226 150 240 140 244 243 42 33 184 39 40 82 39 174 141 106 85 159 13 170 63 244 42 0 186 225 103 171 237 73 218 194 221 124 131 9 100 83 13 162 16 235 180 148 79 244 4 97 93 158 185 83 178 170 228 28 169 142 54 26 211 71 58 3 50 94 214 50 47 26 233 32 182 249 145 6 146 10 37 73 120 16 243 32 92 57 20 148 109 90 24 42 178 64 184 154 154 217 226 59 13 201 207 254 191 120 152 123 185 34 53 225 73 77 127 65 84 121 98 0 23 56 105 252 237 173 163 106 161 100 121 148 144 254 20 125 40 63 47 167 46 11 223 188 0 180 208 133 154 46 44 36 105 109 222 101 66 237 53 252 29 169 42 70 198 109 191 48 191 172 91 235 93 22 29 60 202 122 171 130 168 131 196 159 144 78 209 248 188 51 115 247 119 171 233 84 67 238 145 16 206 204 68 193 114 212 136 178 155 124 237	0 167 0 0 219 0 0 0 221 0 0 104 0 133 143 213 0 201 0 153 124 0 221 0 153 114 0 105 117 224 0 179 0 228 0 154 176 112 174 0 0 125 128 216 249 124 0 103 151 179 227 203 224 238 0 193 172 0 131 184 208 104 183 156 134 0 229 0 0 241 0 0 176 188 125 211 194 135 0 104 220 0 0 168 130 0 216 93 153 161 0 0 152 247 247 197 241 0 130 178 204 0 0 171 193 143 0 0 227 0 235 0 158 0 0 114 196 222 0 114 245 160 170 194 226 194 0 218 99 149 0 0 227 188 222 230 248 0 174 0 181 0 148 212 0 0 182 169 239 0 221 246 136 124 116 0 0 0 230 0 0 176 225 204 0 129 133 0 0 234 183 185 154 191 0 180 207 224 0 0 0 211 239 198 0 0 234 144 185 0 124 0 196 160 246 163 0 126 98 0 0 216 97 0 111 201 0 0 0 113 160 0 0 0 0 238 135 0 0 103 99 161 234 113 101 232 250 250 0 249 222 220 206 137 165 0 0 212 178 0 0 145 0 238 134 191 0 0 0 100 119 0 232 191 0 0 0 0 0 0 216 209 0 0 122 176 0 0 250 158 0 167 0 0 0 0 189 187 212 226 120 129 201 155 224 200 181 0 209 0 202 180 252 170 0 0 183 162 0 193 143 186 208 186 0 173 171 184 0 192 0 150 0 216 145 0 223 153 0 217 0 106 0 134 0 100 104 0 231 226 150 240 140 244 243 0 0 184 0 0 0 0 174 141 106 0 159 0 170 0 244 0 0 186 225 103 171 237 0 218 194 221 124 131 0 100 0 0 162 0 235 180 148 0 244 0 97 93 158 185 0 178 170 228 0 169 142 0 0 211 0 0 0 0 94	0 167 0 0 219 0 0 0 221 0 0 104 0 133 143 213 0 201 0 153 124 0 221 0 153 114 0 105 117 224 0 179 0 228 0 154 176 112 174 0 0 125 128 216 249 124 0 103 151 179 227 203 224 238 0 193 172 0 131 184 208 104 183 156 134 0 229 0 0 241 0 0 176 188 125 211 194 135 0 104 220 0 0 168 130 0 216 93 153 161 0 0 152 247 247 197 241 0 130 178 204 0 0 171 193 143 0 0 227 0 235 0 158 0 0 114 196 222 0 114 245 160 170 194 226 194 0 218 99 149 0 0 227 188 222 230 248 0 174 0 181 0 148 212 0 0 182 169 239 0 221 246 136 124 116 0 0 0 230 0 0 176 225 204 0 129 133 0 0 234 183 185 154 191 0 180 207 224 0 0 0 211 239 198 0 0 234 144 185 0 124 0 196 160 246 163 0 126 98 0 0 216 97 0 111 201 0 0 0 113 160 0 0 0 0 238 135 0 0 103 99 161 234 113 101 232 250 250 0 249 222 220 206 137 165 0 0 212 178 0 0 145 0 238 134 191 0 0 0 100 119 0 232 191 0 0 0 0 0 0 216 209 0 0 122 176 0 0 250 158 0 167 0 0 0 0 189 187 212 226 120 129 201 155 224 200 181 0 209 0 202 180 252 170 0 0 183 162 0 193 143 186 208 186 0 173 171 184 0 192 0 150 0 216 145 0 223 153 0 217 0 106 0 134 0 100 104 0 231 226 150 240 140 244 243 0 0 184 0 0 0 0 174 141 106 0 159 0 170 0 244 0 0 186 225 103 171 237 0 218 194 221 124 131 0 100 0 0	✓

	Input	Expected	Got	
	102 16 135 121 146 78 2 133 120 62 129 217 194 103 116 2 11 218 156 137 126 163 45 240 94 56 176 232 173 212 252 157 76 209 109 14 103 238 134 86 150 109 205 232 174 83 198 110 42 64 238 231 61 129 202 14 241 198 1 5 152 248 231 167 122 184 75 131 226 187 52 116 234 50 223 203 160 159 121 243 72 74 163 152 189 212 146 238 180 108 64 176 159 68 65 86 201 66 141 118 44 101 240 241 142 163 59 220 71 74 109 196 13 204 119 85 57 200 161 97 136 40 150 35 93 74 204 156 171 107 37 104 180 166 127 95 34 191 7 252 17 31 176 18 246 82 70 166 226 30 32 246 149 216 38 180 183 226 131 162 158 69 108 203 139 32 55 148 50 94 223 161 147 240 162 242 233 196 8 150 58 77 86 0 167 87 114 133 65 23 33 18 30 129 193 89 73 183 10 100 244 67 8 193 82 125 181 110 195 121 202 145 228 196 239 143 100 238 25 220 230 253 82 16 245 2 174 210 126 82 11 250 237 23 194 98 159 221 197 10 234 146 82 246 144 64 215 10 65 217 120 68 86 13 181 201 189 56 208 220 152 166 45 248 218 207 69 242 225 200 200 228 203 207 230 34 211 2 176 74 203 138 24 172 125 22 110 228 190 22 187 20 48 207 199 39 49 163 66 102 26 100 71 48 219 112 196 239 75 138 17 48 17 0 59 195 179 226 158 38 177 228 97 204 53 43 187 113 231 129 96 243 13 157 174 15 129 211 4 198 8 86 190 215 82 148 68 17 38 196 39 237 218 152 102 179 42 57 142 13 97 130 62 248 188 223 146 3 65 22 232 245 250 51 199 81 168 36 139 185 187 196 160 38 93	214 0 0 0 233 0 182 249 145 0 146 0 0 0 120 0 243 0 0 0 0 148 109 0 0 0 178 0 184 154 154 217 226 0 0 201 207 254 191 120 152 123 185 0 0 225 0 0 127 0 0 121 98 0 0 0 105 252 237 173 163 106 161 100 121 148 144 254 0 125 0 0 0 167 0 0 223 188 0 180 208 133 154 0 0 0 105 109 222 101 0 237 0 252 0 169 0 0 198 109 191 0 191 172 0 235 93 0 0 0 202 122 171 130 168 131 196 159 144 0 209 248 188 0 115 247 119 171 233 0 0 238 145 0 206 204 0 193 114 212 136 178 155 124 237 102 0 135 121 146 0 0 133 120 0 129 217 194 103 116 0 0 218 156 137 126 163 0 240 94 0 176 232 173 212 252 157 0 209 109 0 103 238 134 0 150 109 205 232 174 0 198 110 0 0 238 231 0 129 202 0 241 198 0 0 152 248 231 167 122 184 0 131 226 187 0 116 234 0 223 203 160 159 121 243 0 0 163 152 189 212 146 238 180 108 0 176 159 0 0 0 201 0 141 118 0 101 240 241 142 163 0 220 0 0 109 196 0 204 119 0 0 200 161 97 136 0 150 0 93 0 204 156 171 107 0 104 180 166 127 95 0 191 0 252 0 0 176 0 246 0 0 166 226 0 0 246 149 216 0 180 183 226 131 162 158 0 108 203 139 0 0 148 0 94 223 161 147 240 162 242 233 196 0 150 0 0 0 0 167 0 114 133 0 0 0 0 0 129 193 0 0 183 0 100 244 0 0 193 0 125 181 110 195 121 202 145 228 196 239 143 100 238 0 220 230 253 0 0 245 0 174 210 126 0	162 0 235 180 148 0 244 0 97 93 158 185 0 178 170 228 0 169 142 0 0 211 0 0 0 0 94 214 0 0 0 233 0 182 249 145 0 146 0 0 0 120 0 243 0 0 0 0 148 109 0 0 0 178 0 184 154 154 217 226 0 0 201 207 254 191 120 152 123 185 0 0 225 0 0 127 0 0 121 98 0 0 0 105 252 237 173 163 106 161 100 121 148 144 254 0 125 0 0 0 167 0 0 223 188 0 180 208 133 154 0 0 0 105 109 222 101 0 237 0 252 0 169 0 0 198 109 191 0 191 172 0 235 93 0 0 0 202 122 171 130 168 131 196 159 144 0 209 248 188 0 115 247 119 171 233 0 0 238 145 0 206 204 0 193 114 212 136 178 155 124 237 102 0 135 121 146 0 0 133 120 0 129 217 194 103 116 0 0 218 156 137 126 163 0 240 94 0 176 232 173 212 252 157 0 209 109 0 103 238 134 0 150 109 205 232 174 0 198 110 0 0 238 231 0 129 202 0 241 198 0 0 152 248 231 167 122 184 0 131 226 187 0 116 234 0 223 203 160 159 121 243 0 0 163 152 189 212 146 238 180 108 0 176 159 0 0 0 201 0 141 118 0 101 240 241 142 163 0 220 0 0 109 196 0 204 119 0 0 200 161 97 136 0 150 0 93 0 204 156 171 107 0 104 180 166 127 95 0 191 0 252 0 0 176 0 246 0 0 166 226 0 0 246 149 216 0 180 183 226 131 162 158 0 108 203 139 0 0 148 0 94 223 161 147 240 162 242 233 196 0 150 0 0 0 0 167 0 114 133 0 0 0 0 0 129 193 0 0 183 0 100 244 0 0 193 0	

	Input	Expected	Got	
		0 250 237 0 194 98 159 221 197 0 234 146 0 246 144 0 215 0 0 217 120 0 0 0 181 201 189 0 208 220 152 166 0 248 218 207 0 242 225 200 200 228 203 207 230 0 211 0 176 0 203 138 0 172 125 0 110 228 190 0 187 0 0 207 199 0 0 163 0 102 0 100 0 0 219 112 196 239 0 138 0 0 0 0 0 195 179 226 158 0 177 228 97 204 0 0 187 113 231 129 96 243 0 157 174 0 129 211 0 198 0 0 190 215 0 148 0 0 0 196 0 237 218 152 102 179 0 0 142 0 97 130 0 248 188 223 146 0 0 0 232 245 250 0 199 0 168 0 139 185 187 196 160 0	125 181 110 195 121 202 145 228 196 239 143 100 238 0 220 230 253 0 0 245 0 174 210 126 0 0 250 237 0 194 98 159 221 197 0 234 146 0 246 144 0 215 0 0 217 120 0 0 0 181 201 189 0 208 220 152 166 0 248 218 207 0 242 225 200 200 228 203 207 230 0 211 0 176 0 203 138 0 172 125 0 110 228 190 0 187 0 0 207 199 0 0 163 0 102 0 100 0 0 219 112 196 239 0 138 0 0 0 0 0 195 179 226 158 0 177 228 97 204 0 0 187 113 231 129 96 243 0 157 174 0 129 211 0 198 0 0 190 215 0 148 0 0 0 196 0 237 218 152 102 179 0 0 142 0 97 130 0 248 188 223 146 0 0 0 232 245 250 0 199 0 168 0 139 185 187 196 160 0	

	Input	Expected	Got	
✓	19 34 72 113 210 82 103 243 48 222 86 166 2 161 213 246 167 58 219 131 114 64 47 80 135 236 206 242 71 211 106 83 251 200 42 177 231 103 146 238 45 241 202 91 140 204 64 135 194 135 134 250 81 251 226 19 177 18 61 48 10 247 94 10 139 30 13 204 235 144 21 221 249 110 52 122 233 104 63 98 137 91 163 181 167 67 222 156 61 101 251 111 186 113 23 231 114 150 66 30 33 94 143 187 64 226 60 206 196 209 206 243 64 97 0 252 58 60 227 162 30 139 116 165 6 125 205 78 108 138 50 86 116 168 31 214 53 12 168 97 97 137 191 74 7 226 175 14 114 59 125 157 42 89 202 81 6 109 58 253 148 110 196 3 116 164 147 21 221 230 55 58 13 190 97 103 120 226 43 76 106 180 120 226 132 166 49 70 14 198 19 109 104 74 31 25 234 9 131 204 157 74 52 23 177 57 41 168 163 225 217 250 174 217 69 35 50 37 104 195 57 88 220 143 241 185 240 155 47 103 98 185 237 216 247 219 53 208 250 82 157 154 60 180 87 190 76 188 69 121 48 71 48 38 134 67 147 129 114 63 73 113 79 102 163 166 59 228 177 179 212 129 40 97 23 177 161 11 136 135 171 59 75 52 24 8 19 188 106 4 78 144 151 133 51 115 237 173 211 58 195 4 220 122 8 181 56 228 213 16 96 27 132 24 20 191 252 88 106 213 79 204 106 149 194 126 179 252 206 202 86 112 170 183 254 8 8 111 193 0 59 108 145 244 46 243 67 226 214 158 189 10 95 101 231 188 82 66 217 216 238 121 94 74 241 199 138 132 46 160 189 68 81 119 116 2 129 127 111 213 241 43 227 165 186 187 167 38 230 226 14 188 118 243 102 210 184 249 38 140 188 64 48 34 23 94 128 122 138 152 45 94 183 192 1 134 13 63 181 34 199 16 68 169 152 228 46 49 158 101 187 246 225 188 250 75 72 196 61 145 55 97 176 102 221 218 223 254 33 168 163 87 148 41 189 104 45 178 108 103 143 170 162 130 224 39 184 33 162 70 204 242 85 241 224 164 188 220 220 66 17 112 107 38 185 125 128 131 126 192 109 236 49 123 163 87 28 240 250 167 15 30 62 240 166 215 121 1 178 63 112 61 6 73 245 7 220 208 1 177 230 76 114 9 205 247 143 45 71 182 37 73 192 229 16 100 124 94 28 215 145 168 138 243 86 121 55 178 6 21 49 234 183 162 251 118 37 24 119	0 0 0 0 0 243 0 0 0 0 0 0 0 246 0 0 0 0 0 0 0 0 0 0 0 242 0 0 0 0 251 0 0 0 0 0 0 0 0 241 0 0 0 0 0 0 0 0 0 250 0 251 0 0 0 0 0 0 0 247 0 0 0 0 0 0 0 0 0 0 249 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 251 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 243 0 0 0 252 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 253 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 250 0 0 0 0 0 0 0 0 0 0 0 0 241 0 240 0 0 0 0 0 0 0 247 0 0 0 250 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 252 0 0 0 0 0 0 0 0 0 0 252 0 0 0 0 0 0 254 0 0 0 0 0 0 0 0 244 0 243 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 241 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 241 0 0 0 0 0 0 0 0 0 0 0 0 243 0 0 0 249 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 246 0 0 250 0 0 0 0 0 0 0 0 0 0 0 0 254 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 242 0 241 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 240 250 0 0 0 0 240 0 0 0 0 0 0 0 0 0 0 245 0 0 0 0 0 0 0 0 0 247 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 243 0 0 0 0 0 0 0 0 0 0 251 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0 0 0 0 0 243 0 0 0 0 0 0 0 246 0 0 0 0 0 0 0 0 0 0 0 242 0 0 0 0 251 0 0 0 0 0 0 0 0 241 0 0 0 0 0 0 0 0 0 250 0 251 0 0 0 0 0 0 0 247 0 0 0 0 0 0 0 0 0 0 249 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 251 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 243 0 0 0 252 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 253 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 250 0 0 0 0 0 0 0 0 0 0 0 0 241 0 240 0 0 0 0 0 0 0 247 0 0 0 250 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 252 0 0 0 0 0 0 0 0 0 0 0 252 0 0 0 0 0 254 0 0 0 0 0 0 0 0 244 0 243 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 241 0 0 0 0 0 0 0 0 0 0 0 0 0 0 241 0 0 0 0 0 0 0 0 0 0 0 243 0 0 0 249 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 246 0 0 250 0 0 0 0 0 0 0 0 0 0 0 0 254 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 242 0 241 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 240 250 0 0 0 0 240 0 0 0 0 0 0 0 0 0 0 245 0 0 0 0 0 0 0 0 0 247 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 243 0 0 0 0 0 0 0 0 0 251 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	✓

	Input	Expected	Got	
	68 0 113 180 32 171 149 188 228 22 213 52 126 37 227 221 25 69 204 113 7 143 27 195 94 138 43 172 166 4 73 205 236 216 54 70 186 220 86 150 241 44 180 53 122 67 227 49 167 237 254 146 144 28 200 192 168 120 4 183 83 200 74 180 108 79 160 41 250 137 113 116 73 159 187 43 248 76 97 183 45 226 236 40 177 77 79 165 7 240	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 241 0 0 0 0 0 0 0 0 0 254 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 250 0 0 0 0 0 0 0 248 0 0 0 0 0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 241 0 0 0 0 0 0 0 0 0 254 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 250 0 0 0 0 0 0 0 248 0 0 0 0 0 0 0 0 0 0 0 0	

Passed all tests! ✓

Correct

Marks for this submission: 10.00/10.00.

Question 6

Correct

Mark 10.00 out of 10.00

[Multiply]

Khi truyền một con trỏ vào hàm, mọi thay đổi với biến con trỏ sẽ làm thay đổi giá trị của biến tương ứng bên ngoài hàm.

Viết hàm `void multiply (int* n, int k)` thực hiện phép nhân giá trị biến \(n \) lên \(k \) lần.

Biết hàm nhận đối số lần lượt là kiểu con trỏ và tham trị.

For example:

Input	Result
2 3	6

Answer: (penalty regime: 0 %)

```
void multiply(int *n, int k)
{
    *n *= k;
}
```

	Input	Expected	Got	
✓	2 3	6	6	✓
✓	-122 45	-5490	-5490	✓
✓	123 11	1353	1353	✓

Passed all tests! ✓

Correct

Marks for this submission: 10.00/10.00.

Question 7

Correct

Mark 10.00 out of 10.00

[PointerToArray]

Viết hàm `int* getPointerToArray(int n)`. Hàm này khai báo một con trỏ kiểu nguyên, cấp phát bộ nhớ động cho con trỏ đó \(n\) phần tử kiểu nguyên và sau đó gán giá trị cho \(n\) phần tử đó các số được nhập từ bàn phím. Hàm trả về con trỏ được khai báo.

For example:

Input	Result
3 -5021 4497 -9846	-5021 4497 -9846

Answer: (penalty regime: 0 %)

```
#include <bits/stdc++.h>
using namespace std;
#define ll long long
#define ull unsigned long long
#define el "\n"
#define se second
#define fi first
#define en end()
#define be begin()
#define sz size()
#define Faster ios_base::sync_with_stdio(0); cin.tie(0); cout.tie(0);

int* getPointerToArray(int n)
{
    int *ptr = new int[n];
    for(int i = 0; i < n; i++) cin >> ptr[i];
    return ptr;
}
void Print()
{
    int n; cin >> n;
    int *ptr = getPointerToArray(n);
    for(int i = 0; i < n; i++) cout << ptr[i] << " ";
    delete[] ptr;
}

```

	Input	Expected	Got	
✓	3 -5021 4497 -9846	-5021 4497 -9846	-5021 4497 -9846	✓

	Input	Expected	Got	
✓	16 8948 -301 6740 4125 6514 7332 7390 4547 5507 -6894 -9818 -1065 -4962 7746 5690 -5259	8948 -301 6740 4125 6514 7332 7390 4547 5507 -6894 -9818 -1065 -4962 7746 5690 -5259	8948 -301 6740 4125 6514 7332 7390 4547 5507 -6894 -9818 -1065 -4962 7746 5690 -5259	✓
✓	80 -9849 5239 -627 5864 -5636 -578 753 9474 -1362 9909 1007 -1055 1422 1988 8792 2458 -8720 -697 4733 -234 -6538 57 -1116 9797 -4540 -5924 1824 7688 -2977 6631 -9555 -7784 1356 -3256 -2966 6415 4605 5644 -514 2842 3686 2119 -1173 -3325 -9371 5325 -3710 4362 5603 -4205 -4336 344 -2294 -6984 -9429 607 9374 -4523 -311 9268 7430 -2751 8830 -3061 2166 5946 6150 7410 7982 -9216 3384 -2 4582 1105 -2627 -1705 8867 -7383 6971 -500	-9849 5239 -627 5864 -5636 -578 753 9474 -1362 9909 1007 -1055 1422 1988 8792 2458 -8720 -697 4733 -234 -6538 57 -1116 9797 -4540 -5924 1824 7688 -2977 6631 -9555 -7784 1356 -3256 -2966 6415 4605 5644 -514 2842 3686 2119 -1173 -3325 -9371 5325 -3710 4362 5603 -4205 -4336 344 -2294 -6984 -9429 607 9374 -4523 -311 9268 7430 -2751 8830 -3061 2166 5946 6150 7410 7982 -9216 3384 -2 4582 1105 -2627 -1705 8867 -7383 6971 -500	-9849 5239 -627 5864 -5636 -578 753 9474 -1362 9909 1007 -1055 1422 1988 8792 2458 -8720 -697 4733 -234 -6538 57 -1116 9797 -4540 -5924 1824 7688 -2977 6631 -9555 -7784 1356 -3256 -2966 6415 4605 5644 -514 2842 3686 2119 -1173 -3325 -9371 5325 -3710 4362 5603 -4205 -4336 344 -2294 -6984 -9429 607 9374 -4523 -311 9268 7430 -2751 8830 -3061 2166 5946 6150 7410 7982 -9216 3384 -2 4582 1105 -2627 -1705 8867 -7383 6971 -500	✓
✓	90 3172 3094 -5447 9503 849 -6219 -7943 4223 5200 1233 3201 2972 -6153 6332 2566 8931 -9307 -9509 -7624 -6024 -9759 7715 6451 -8462 5536 -1893 -6058 -8139 4732 -3097 -4412 1962 5915 -7670 3659 -6502 6254 7283 5071 -2977 9858 -9310 -9035 8775 1973 -1663 5367 4450 1755 8973 1771 -725 -918 -2010 9319 8946 -239 2201 -7783 -8594 1857 1687 6041 -1702 6823 -2792 -388 -3566 7863 9896 -8312 3893 3540 -5050 -8057 4922 7796 4268 -7522 2875 -7151 3936 863 6541 8539 -6609 7506 3460 7385 446	3172 3094 -5447 9503 849 -6219 -7943 4223 5200 1233 3201 2972 -6153 6332 2566 8931 -9307 -9509 -7624 -6024 -9759 7715 6451 -8462 5536 -1893 -6058 -8139 4732 -3097 -4412 1962 5915 -7670 3659 -6502 6254 7283 5071 -2977 9858 -9310 -9035 8775 1973 -1663 5367 4450 1755 8973 1771 -725 -918 -2010 9319 8946 -239 2201 -7783 -8594 1857 1687 6041 -1702 6823 -2792 -388 -3566 7863 9896 -8312 3893 3540 -5050 -8057 4922 7796 4268 -7522 2875 -7151 3936 863 6541 8539 -6609 7506 3460 7385 446	3172 3094 -5447 9503 849 -6219 -7943 4223 5200 1233 3201 2972 -6153 6332 2566 8931 -9307 -9509 -7624 -6024 -9759 7715 6451 -8462 5536 -1893 -6058 -8139 4732 -3097 -4412 1962 5915 -7670 3659 -6502 6254 7283 5071 -2977 9858 -9310 -9035 8775 1973 -1663 5367 4450 1755 8973 1771 -725 -918 -2010 9319 8946 -239 2201 -7783 -8594 1857 1687 6041 -1702 6823 -2792 -388 -3566 7863 9896 -8312 3893 3540 -5050 -8057 4922 7796 4268 -7522 2875 -7151 3936 863 6541 8539 -6609 7506 3460 7385 446	✓
✓	24 2774 -567 -2701 -9067 9837 418 -7935 4937 2756 -517 3981 5548 -6575 254 2385 -8120 -7676 -149 -2200 -3833 520 4328 1840 -4811	2774 -567 -2701 -9067 9837 418 -7935 4937 2756 -517 3981 5548 -6575 254 2385 -8120 -7676 -149 -2200 -3833 520 4328 1840 -4811	2774 -567 -2701 -9067 9837 418 -7935 4937 2756 -517 3981 5548 -6575 254 2385 -8120 -7676 -149 -2200 -3833 520 4328 1840 -4811	✓

Passed all tests! ✓

Correct

Marks for this submission: 10.00/10.00.

Question 8

Correct

Mark 10.00 out of 10.00

[Reference Variable - Swap]

2. Biến tham chiếu (Reference Variable)

Một biến tham chiếu là một **alias**, hay là một tên khác của một biến đã tồn tại. Khi một biến được khai báo là tham chiếu của một biến khác, chúng ta có thể truy cập một biến sử dụng tên thật hoặc tên tham chiếu của nó.

Tham chiếu chủ yếu được dùng để truyền vào các hàm (pass-by-reference). Khi tham chiếu được truyền vào hàm, hàm sẽ làm việc trực tiếp với biến gốc thay vì tạo ra một bản copy (clone) và làm việc trên bản copy đó. Như vậy, mọi thay đổi trên biến sẽ được lưu lại trên biến gốc sau khi kết thúc hàm.

Biến tham chiếu có nhiều điểm tương tự với biến con trỏ. Trong nhiều trường hợp, biến tham chiếu có thể được sử dụng thay thế cho con trỏ, đặc biệt khi được sử dụng làm đối số của các hàm.

2.1 THAM CHIẾU &

Ở bài học trước, chúng ta sử dụng phép & để lấy địa chỉ của một biến trong các biểu thức. Ở phần này, chúng ta sẽ học thêm một cách sử dụng nữa của phép & để khai báo biến tham chiếu của một biến khác.

Phép & có ý nghĩa khác nhau khi sử dụng trong biểu thức và khi khai báo biến. Khi được sử dụng trong một biểu thức, phép & đóng vai trò là phép lấy địa chỉ (address-of operator), trả về địa chỉ của một biến (Ví dụ: nếu `number` là một biến kiểu `int`, `&number` sẽ trả về địa chỉ của vùng nhớ chứa biến `number`).

Tuy nhiên, khi & được sử dụng trong khai báo biến, thì biến được khai báo ngay sau đó được xem là biến tham chiếu. Biến tham chiếu thường được dùng như tên thay thế của một biến khác đã được khai báo trước đó.

Để khai báo một biến tham chiếu, chúng ta sử dụng cú pháp sau:

```
type &newName = existingName;
// hoặc
type& newName = existingName;
// hoặc
type & newName = existingName;
```

Ở các cách khai báo trên, biến `newName` tham chiếu đến biến `existingName`. Chúng ta có thể truy cập đến biến `existingName` sử dụng chính tên gốc của biến đó hoặc qua tham chiếu `newName`.

Ví dụ:

```
/* Khai báo và khởi tạo tham chiếu */
#include
using namespace std;

int main() {
    int number = 88;          // Khai báo biến number
    int & refNumber = number; // Khai báo biến refNumber tham chiếu đến biến number.

    cout << number << endl;    // In ra giá trị của biến number (88)
    cout << refNumber << endl; // In ra giá trị của tham chiếu (88)

    refNumber = 99;            // Gán giá trị mới cho tham chiếu refNumber
    cout << refNumber << endl;
    cout << number << endl;    // Giá trị của biến number cũng thay đổi theo (99)

    number = 55;               // Gán gía trị mới cho biến number
    cout << number << endl;
    cout << refNumber << endl; // Giá trị của tham chiếu cũng thay đổi (55)
}
```

2.3 THAM CHIẾU HOẠT ĐỘNG NHƯ THẾ NÀO?

Tham chiếu hoạt động như một con trỏ. Một tham chiếu được được sử dụng như tên thứ \(2\) của một biến. Nó chứa địa chỉ của biến, như mô tả dưới đấy:

2.4 Tham chiếu và con trỏ

Con trỏ và tham chiếu có chức năng khá giống nhau trừ một số đặc điểm sau:

- 1. Một tham chiếu (reference) là một biến hằng số lưu địa chỉ. Tham chiếu phải được khởi tạo ngay sau khi khai báo.

  ```
  int & iRef;   // Lỗi: 'iRef' được khai báo là tham chiếu nhưng chưa được khai báo.
  ```

  Khi đã khai báo và khởi tạo, biến tham chiếu không thể thay đổi giá trị.
- 2. Để truy vấn giá trị mà con trỏ chỉ đến, chúng ta sử dụng phép *. Tương tự, để gán giá trị của con trỏ với địa chỉ của một biến bình thường, chúng ta sử dụng phép &. Tuy nhiên, đối với biến tham chiếu, hai quy trình này được thực hiện một cách ngầm định, chúng ta không cần phải sử dụng các phép toán như trong con trỏ.

Ví dụ:

```
    /* Tham chiếu vs. Con trỏ */
#include
using namespace std;

int main() {
    int number1 = 88, number2 = 22;

    // Tạo một con trỏ trỏ đến biến number1
    int * pNumber1 = &number1;  // Khởi tạo con trỏ
    *pNumber1 = 99;             // Thay đổi giá trị mà con trỏ chỉ đến
    cout << *pNumber1 << endl;  // 99
    cout << &number1 << endl;   // 0x22ff18
    cout << pNumber1 << endl;   // 0x22ff18 (Giá trị của <a class="autolink" title="Biến con trỏ"
href="https://dev.uet.vnu.edu.vn/mod/quiz/view.php?id=4833">biến con trỏ</a> pValue)
    cout << &pNumber1 << endl;  // 0x22ff10 (Địa chỉ của <a class="autolink" title="Biến con trỏ"
href="https://dev.uet.vnu.edu.vn/mod/quiz/view.php?id=4833">biến con trỏ</a>)
    pNumber1 = &number2;        // Con trỏ có thể đổi giá trị để trỏ đến một địa chỉ khác

    // Tạo một tham chiếu cho biến number1
    int & refNumber1 = number1;  // Tham chiếu ngầm định ( không phải &number1)
    refNumber1 = 11;             // Thay đổi giá trị mà tham chiếu chỉ đến  - không cần sử dụng phép *
    cout << refNumber1 << endl;  // 11
    cout << &number1 << endl;    // 0x22ff18
    cout << &refNumber1 << endl; // 0x22ff18
    //refNumber1 = &number2;     // Lỗi: Tham chiếu không thể thay đổi giá trị, nó là một hằng số.
    refNumber1 = number2;        // refNumber1 vẫn là tham chiếu của biến number1
                                 // Lệnh trên chỉ copy giá trị của biến number2 vào tham chiếu refNumber1 (hay number1)
    number2++;
    cout << refNumber1 << endl;  // 22
    cout << number1 << endl;     // 22
    cout << number2 << endl;     // 23
}
```

2.4 TRUYỀN THAM CHIẾU VÀO HÀM SỬ DỤNG BIẾN THAM CHIẾU HOẶC BIẾN CON TRỎ

Truyền bằng giá trị

Trong ngôn ngữ lập trình C/C++, các đối số được truyền vào hàm bằng gía trị (ngoại từ mảng `array`, bởi vì mảng thực chất là con trỏ). Có nghĩa là, các bản sao (clone) của các đối số sẽ được tạo ra và truyền vào hàm. Những thay đổi tác động lên bản sao của các đối số trong hàm sẽ không làm thay đổi bản gốc của đối số - được khai báo bên ngoài hàm.

Ví dụ:

```
    /* Truyền bằng giá trị */
#include
using namespace std;

int square(int);

int main() {
    int number = 8;
    cout <<  "In main(): " << &number << endl;  // 0x22ff1c
    cout << number << endl;         // 8
    cout << square(number) << endl; // 64
    cout << number << endl;         // 8 - bản gốc của đối số : gía trị không thay đổi
}

int square(int number) {  // non-const
    cout <<  "In square(): " << &number << endl;  // 0x22ff00
    number *= number;           // Chỉnh sửa trên bản sao
    return number;
}
```

Ở đoạn code trên, biến `number` ở hàm `main()` và biến `number` ở hàm `square()` là hoàn toàn khác nhau. Việc thay đổi biến `number` ở hàm `square()` không làm thay đổi giá trị của biến `number` ở hàm `main()`.

Truyền tham chiếu sử dụng đối số con trỏ.

Trong nhiều trường hợp, chúng ta muốn chỉnh sửa trực tiếp lên bản gốc của một biến chứ không phải bản copy. Để làm được điều này, chúng ta có thể truyền biến con trỏ làm đối số của hàm.

Ví dụ:

```
    /* Truyền tham chiếu sử dụng biến con trỏ */
#include
using namespace std;

void square(int *);

int main() {
    int number = 8;
    cout <<  "In main(): " << &number << endl;  // 0x22ff1c
    cout << number << endl;   // 8
    square(&number);          // Truyền địa chỉ của biến thay vì truyền tên biến
    cout << number << endl;   // 64
}

void square(int * pNumber) {  // Hàm nhận đối số con trỏ
    cout <<  "In square(): " << pNumber << endl;  // 0x22ff1c
    *pNumber *= *pNumber;      // Thay đổi vào ô nhớ mà biến pNumber chỉ đến.
}
```

Trong ví dụ trên, biến `pNumber` ở hàm `square()` và biến `number` ở hàm `main()` cùng trỏ đến một vùng nhớ trên máy tính, nên thay đổi ở hàm `square()` sẽ tác động lên hàm `main()` và được lưu lại sau khi hàm kết thúc.

Truyền tham chiếu với đối số tham chiếu

Thay vì truyền con trỏ vào hàm, biến tham chiếu có thể được sử dụng thay thế để tránh những rắc rối mà con trỏ đem lại.

Ví dụ:

```
/* Truyền tham chiếu sử dụng đối số tham chiếu */
#include
using namespace std;

void square(int &);

int main() {
    int number = 8;
    cout <<  "In main(): " << &number << endl;  // 0x22ff1c
    cout << number << endl;  // 8
    square(number);          // Truyền tham chiếu
    cout << number << endl;  // 64
}

void square(int & rNumber) {  // Hàm nhận đối số là tham chiếu kiểu int
    cout <<  "In square(): " << &rNumber << endl;  // 0x22ff1c
    rNumber *= rNumber;
}
```

Bài tập

Phép hoán đổi giá trị của hai biến được sử dụng trong nhiều bài toán, chẳng hạn như bài toán sắp xếp. Thông thường, người lập trình sẽ xây dựng một hàm riêng thực hiện phép toán này nhằm mục đích tối ưu việc viết code.

Là một lập trình viên, bạn hãy viết hàm `void swap(int& a, int& b)` thực hiện việc hoán đổi giá trị của hai biến , trong đó hàm nhận đối số đầu vào là hai biến tham số `a` và `b`.

For example:

Input	Result
1 2	2 1

Answer: (penalty regime: 0 %)

```
#include <bits/stdc++.h>
using namespace std;
#define ll long long
#define ull unsigned long long
#define el "\n"
#define se second
#define fi first
#define en end()
#define be begin()
#define sz size()
#define Faster ios_base::sync_with_stdio(0); cin.tie(0); cout.tie(0);

void swap(int &a, int &b)
{
    int tmp = a;
    a = b;
    b= tmp;
}
void Print()
{
    //Faster;
    int a,b; cin >> a >> b;
    swap(a,b);
    cout << a << " " << b;
    //return 0;
}
```

```
27
```

	Input	Expected	Got	
✓	1 2	2 1	2 1	✓
✓	3 4	4 3	4 3	✓
✓	123456789 987654321	987654321 123456789	987654321 123456789	✓
✓	100 1000	1000 100	1000 100	✓
✓	444 555	555 444	555 444	✓

Passed all tests! ✓

Correct

Marks for this submission: 10.00/10.00.

Question 9

Correct

Mark 10.00 out of 10.00

[Swap]

Việc hoán đổi giá trị của hai biến được sử dụng trong nhiều bài toán, chẳng hạn như bài toán sắp xếp. Viết hàm `void swap(int* a, int* b)` thực hiện việc hoán đổi giá trị của hai biến \(a\) và \(b\).

For example:

Input	Result
1 2	2 1

Answer: (penalty regime: 0 %)

```
#include <bits/stdc++.h>
using namespace std;
#define ll long long
#define ull unsigned long long
#define el "\n"
#define se second
#define fi first
#define en end()
#define be begin()
#define sz size()
#define Faster ios_base::sync_with_stdio(0); cin.tie(0); cout.tie(0);

void swap(int *a, int *b)
{
    int tmp = *a;
    *a = *b;
    *b= tmp;
}
void Print()
{
    int c, d; cin >> c >> d;
    int *a = &c, *b = &d;
    swap(a,b);
    cout << *a << " " << *b;
}

```

	Input	Expected	Got	
✓	1 2	2 1	2 1	✓
✓	3 4	4 3	4 3	✓
✓	123456789 987654321	987654321 123456789	987654321 123456789	✓
✓	100 1000	1000 100	1000 100	✓
✓	444 555	555 444	555 444	✓

Passed all tests! ✓

Correct

Marks for this submission: 10.00/10.00.

Question 10

Correct

Mark 10.00 out of 10.00

[TheMatrix]

Viết hàm `int** inputMatrix(int nRows, int nCols)` đọc từ bàn phím một ma trận số nguyên có số hàng \(nRows\) và số cột \(nCols\), lưu vào một mảng động hai chiều và trả về con trỏ trỏ đến mảng động này.

Viết hàm `void printMatrix(int** matrix, int nRows, int nCols)` nhận tham số là con trỏ đến mảng động hai chiều \(matrix\), số hàng \(nRows\) và số cột \(nCols\) của ma trận. Hàm này in ma trận đầu vào ra màn hình, các phần tử trên cùng một hàng cách nhau bởi một dấu cách.

For example:

Input	Result
2 3 1 2 3 3 4 5	1 2 3 3 4 5

Answer: (penalty regime: 0 %)

```
#include <bits/stdc++.h>
using namespace std;
#define ll long long
#define ull unsigned long long
#define el "\n"
#define se second
#define fi first
#define en end()
#define be begin()
#define sz size()
#define Faster ios_base::sync_with_stdio(0); cin.tie(0); cout.tie(0);

int** inputMatrix(int nRows, int nCols)
{
    int** matrix = new int*[nRows];
    for(int i = 0; i < nRows; i++)
    {
        matrix[i] = new int[nCols];
    }
    return matrix;
}
void printMatrix(int** matrix, int nRows, int nCols)
{
    for(int i = 0; i < nRows; i++)
    {
        for(int j = 0;  j< nCols; j++)
        {
            cin >> matrix[i][j];
        }
    }
    for(int i = 0; i < nRows; i++)
    {
        for(int j = 0;  j< nCols; j++)
        {
            cout << matrix[i][j] << " ";
        }
        cout << el;
    }
}
void Print()
{
    int nRows, nCols; cin >> nRows >> nCols;
    int ** matrix = inputMatrix(nRows, nCols);
```

```
44      printMatrix(matrix, nRows, nCols);
45  }
46
```

	Input	Expected	Got	
✓	2 3 1 2 3 3 4 5	1 2 3 3 4 5	1 2 3 3 4 5	✓
✓	3 3 1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	✓
✓	3 2 -1 -2 -3 -4 -5 -6	-1 -2 -3 -4 -5 -6	-1 -2 -3 -4 -5 -6	✓
✓	5 5 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25	1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25	1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25	✓
✓	1 1 1	1	1	✓

Passed all tests! ✓

Correct

Marks for this submission: 10.00/10.00.

Question 11

Correct

Mark 10.00 out of 10.00

[SymmetricMatrix]

Trong đại số tuyến tính, một **ma trận đối xứng** là một ma trận vuông, *A*, bằng chính ma trận chuyển vị của nó.

Mỗi phần tử của một ma trận đối xứng thì đối xứng qua đường chéo. Do vậy, nếu các phần tử được viết dưới dạng \(*A=a_{ij}*\), thì \(a_{ij} = a_{ji}\).

Ta có thể biểu diễn một ma trận bằng một mảng hai chiều trong máy tính.

Viết hàm `void inputMatrix(int** matrix, int nRows, int nCols)` nhận tham số là con trỏ đến mảng hai chiều biểu diễn ma trận, số hàng và số cột của ma trận. Hàm này lưu các phần tử của ma trận nhập từ bàn phím vào mảng hai chiều biểu diễn ma trận.

Viết hàm `int isSymmetric(int** matrix, int nRows, int nCols)` nhận tham số là con trỏ đến mảng hai chiều biểu diễn ma trận, số hàng và số cột của ma trận. Hàm trả về \(1\) nếu ma trận đầu vào là ma trận đối xứng, ngược lại trả về \(0\).

For example:

Input	Result
3 3 1 2 3 2 3 4 3 4 5	1

Answer: (penalty regime: 0 %)

```
#include <bits/stdc++.h>
using namespace std;
#define ll long long
#define ull unsigned long long
#define el "\n"
#define se second
#define fi first
#define en end()
#define be begin()
#define sz size()
#define Faster ios_base::sync_with_stdio(0); cin.tie(0); cout.tie(0);

int** createMatrix(int nRows, int nCols)
{
    int** matrix = new int*[nRows];
    for(int i = 0; i < nRows; i++)
    {
        matrix[i] = new int[nCols];
    }
    return matrix;
}
void inputMatrix(int** matrix, int nRows, int nCols)
{
    for(int i = 0; i < nRows; i++)
    {
        for(int j = 0;  j< nCols; j++)
        {
            cin >> matrix[i][j];
        }
    }
}
int isSymmetric(int** matrix, int nRows, int nCols)
{
    if(nRows != nCols) return 0;
    for(int i = 0; i < nRows; i++)
```

```
    {
        for(int j = 0;  j< nCols; j++)
        {
            if(matrix[i][j] != matrix[j][i]) return 0;
        }
    }
    return 1;
}
void Print()
{
    int nRows, nCols; cin >> nRows >> nCols;
    int ** matrix = createMatrix(nRows, nCols);
    inputMatrix(matrix, nRows, nCols);
    cout << isSymmetric(matrix, nRows, nCols);
}

```

	Input	Expected	Got	
✓	3 3 1 2 3 2 3 4 3 4 5	1	1	✓
✓	3 3 1 2 3 2 3 4 3 2 5	0	0	✓
✓	1 1 1	1	1	✓
✓	2 3 1 2 3 4 5 6	0	0	✓
✓	2 3 1 1 1 1 1 1	0	0	✓

Passed all tests! ✓

Correct

Marks for this submission: 10.00/10.00.

Question 12

Correct

Mark 10.00 out of 10.00

[EvenNumberOnly]

Viết hàm `int** keepEven (int** matrix, int nRows, int nCols)` kiểm tra ma trận hai chiều.

Hàm nhận đầu vào là ma trận \(matrix\) có kích thước \(nRows \times nCols\). Hàm trả về một ma trận mới sao cho tất cả các giá trị là số lẻ trong ma trận ban đầu được gán giá trị \(0\) và giữ nguyên các số chẵn.

Answer: (penalty regime: 0 %)

```
#include <bits/stdc++.h>
using namespace std;
#define ll long long
#define ull unsigned long long
#define el "\n"
#define se second
#define fi first
#define en end()
#define be begin()
#define sz size()
#define Faster ios_base::sync_with_stdio(0); cin.tie(0); cout.tie(0);

int** createMatrix(int nRows, int nCols)
{
    int** matrix = new int*[nRows];
    for(int i = 0; i < nRows; i++)
    {
        matrix[i] = new int[nCols];
    }
    return matrix;
}
void inputMatrix(int** matrix, int nRows, int nCols)
{
    for(int i = 0; i < nRows; i++)
    {
        for(int j = 0;  j< nCols; j++)
        {
            cin >> matrix[i][j];
        }
    }
}
int **keepEven(int** matrix, int nRows, int nCols)
{
    for(int i = 0; i < nRows; i++)
    {
        for(int j = 0;  j< nCols; j++)
        {
            if(matrix[i][j] % 2) matrix[i][j] = 0;
        }
    }
    return matrix;
}
void Print()
{
    int nRows, nCols; cin >> nRows >> nCols;
    int ** matrix = createMatrix(nRows, nCols);
    inputMatrix(matrix, nRows, nCols);
    matrix = keepEven(matrix, nRows, nCols);
    for(int i = 0; i < nRows; i++)
    {
        for(int j = 0; j < nCols; j++)
        {
```

	Input	Expected	Got	
✓	6 5 9 11 29 26 21 26 8 1 11 27 5 25 28 11 6 4 3 4 19 3 8 12 26 11 14 12 8 19 12 9	0 0 0 26 0 26 8 0 0 0 0 0 28 0 6 4 0 4 0 0 8 12 26 0 14 12 8 0 12 0	0 0 0 26 0 26 8 0 0 0 0 0 28 0 6 4 0 4 0 0 8 12 26 0 14 12 8 0 12 0	✓
✓	16 3 22 14 19 25 5 46 43 20 21 27 37 42 42 3 15 40 17 17 19 10 6 28 41 1 13 19 46 21 47 45 44 32 11 40 0 18 28 8 41 6 13 22 0 15 37 11 0 19	22 14 0 0 0 46 0 20 0 0 0 42 42 0 0 40 0 0 0 10 6 28 0 0 0 0 46 0 0 0 44 32 0 40 0 18 28 8 0 6 0 22 0 0 0 0 0 0	22 14 0 0 0 46 0 20 0 0 0 42 42 0 0 40 0 0 0 10 6 28 0 0 0 0 46 0 0 0 44 32 0 40 0 18 28 8 0 6 0 22 0 0 0 0 0 0	✓

	Input	Expected	Got	
✓	18 14 244 69 125 141 130 59 110 28 111 236 213 186 66 241 180 199 208 86 28 209 232 107 207 193 33 85 77 176 170 84 213 122 135 36 169 155 216 205 242 191 224 193 131 103 132 102 193 223 32 75 0 59 222 119 103 29 147 92 107 205 45 60 83 89 194 138 39 47 42 142 250 242 185 188 85 129 10 85 71 145 81 146 122 139 196 115 85 239 170 15 15 63 118 91 16 30 236 150 181 200 214 202 6 79 9 123 137 3 173 144 211 31 96 150 13 203 29 192 137 178 110 159 167 32 172 18 39 151 97 31 135 76 80 220 39 111 200 41 224 95 43 195 133 232 225 84 173 151 49 10 143 23 210 91 75 176 242 174 184 227 149 192 147 248 28 212 125 73 71 87 94 168 66 185 145 197 113 214 223 1 232 79 191 205 177 169 203 0 199 130 204 33 137 82 57 133 186 71 10 46 106 122 79 67 96 33 155 40 176 196 243 4 249 170 36 245 120 212 143 24 154 228 21 150 248 76 164 130 69 63 135 106 93 133 148 182 187 40 94 179 155 133 54 123 147 96 251 123 207 29 100 22	244 0 0 0 130 0 110 28 0 236 0 186 66 0 180 0 208 86 28 0 232 0 0 0 0 0 0 176 170 84 0 122 0 36 0 0 216 0 242 0 224 0 0 0 132 102 0 0 32 0 0 0 222 0 0 0 0 92 0 0 0 60 0 0 194 138 0 0 42 142 250 242 0 188 0 0 10 0 0 0 0 146 122 0 196 0 0 0 170 0 0 0 118 0 16 30 236 150 0 200 214 202 6 0 0 0 0 0 0 144 0 0 96 150 0 0 0 192 0 178 110 0 0 32 172 18 0 0 0 0 0 76 80 220 0 0 200 0 224 0 0 0 0 232 0 84 0 0 0 10 0 0 210 0 0 176 242 174 184 0 0 192 0 248 28 212 0 0 0 0 94 168 66 0 0 0 0 214 0 0 232 0 0 0 0 0 0 0 0 130 204 0 0 82 0 0 186 0 10 46 106 122 0 0 96 0 0 40 176 196 0 4 0 170 36 0 120 212 0 24 154 228 0 150 248 76 164 130 0 0 0 106 0 0 148 182 0 40 94 0 0 0 54 0 0 96 0 0 0 0 100 22	244 0 0 0 130 0 110 28 0 236 0 186 66 0 180 0 208 86 28 0 232 0 0 0 0 0 0 176 170 84 0 122 0 36 0 0 216 0 242 0 224 0 0 0 132 102 0 0 32 0 0 0 222 0 0 0 0 92 0 0 0 60 0 0 194 138 0 0 42 142 250 242 0 188 0 0 10 0 0 0 0 146 122 0 196 0 0 0 170 0 0 0 118 0 16 30 236 150 0 200 214 202 6 0 0 0 0 0 0 144 0 0 96 150 0 0 0 192 0 178 110 0 0 32 172 18 0 0 0 0 0 76 80 220 0 0 200 0 224 0 0 0 0 232 0 84 0 0 0 10 0 0 210 0 0 176 242 174 184 0 0 192 0 248 28 212 0 0 0 0 94 168 66 0 0 0 0 214 0 0 232 0 0 0 0 0 0 0 0 130 204 0 0 82 0 0 186 0 10 46 106 122 0 0 96 0 0 40 176 196 0 4 0 170 36 0 120 212 0 24 154 228 0 150 248 76 164 130 0 0 0 106 0 0 148 182 0 40 94 0 0 0 54 0 0 96 0 0 0 0 100 22	✓
✓	7 11 62 39 72 67 16 44 1 50 72 25 63 39 6 63 39 8 68 28 60 54 46 61 7 45 29 35 11 76 42 75 6 53 34 52 30 71 16 57 76 67 36 4 43 17 2 53 50 25 28 63 13 76 4 49 16 15 29 64 74 66 44 52 54 60 28 7 76 16 6 10 9 37 72 20 17 49 13	62 0 72 0 16 44 0 50 72 0 0 0 6 0 0 8 68 28 60 54 46 0 0 0 0 0 0 76 42 0 6 0 34 52 30 0 16 0 76 0 36 4 0 0 2 0 50 0 28 0 0 76 4 0 16 0 0 64 74 66 44 52 54 60 28 0 76 16 6 10 0 0 72 20 0 0 0	62 0 72 0 16 44 0 50 72 0 0 0 6 0 0 8 68 28 60 54 46 0 0 0 0 0 0 76 42 0 6 0 34 52 30 0 16 0 76 0 36 4 0 0 2 0 50 0 28 0 0 76 4 0 16 0 0 64 74 66 44 52 54 60 28 0 76 16 6 10 0 0 72 20 0 0 0	✓

	Input	Expected	Got	
✓	19 9 108 163 69 53 38 64 74 75 136 54 165 94 138 74 40 10 30 79 8 41 21 43 20 169 8 63 83 137 60 66 76 149 68 127 86 50 38 150 34 96 159 158 42 64 121 44 19 111 139 11 57 65 131 107 68 111 9 105 99 77 150 71 99 25 0 86 75 26 82 124 103 89 58 96 99 164 94 37 17 129 27 100 54 43 153 21 112 31 48 159 56 110 80 6 23 170 45 88 10 2 95 134 74 53 104 21 13 162 24 144 37 162 23 72 134 81 76 108 146 111 159 161 118 116 51 163 13 78 154 106 83 146 14 93 55 65 102 17 41 167 98 58 101 158 73 90 146 121 91 78 35 166 67 98 74 168 54 4 96 42 10 98 114 16 14 28 93 67 61 75 99	108 0 0 0 38 64 74 0 136 54 0 94 138 74 40 10 30 0 8 0 0 0 20 0 8 0 0 0 60 66 76 0 68 0 86 50 38 150 34 96 0 158 42 64 0 44 0 0 0 0 0 0 0 0 68 0 0 0 0 0 150 0 0 0 0 86 0 26 82 124 0 0 58 96 0 164 94 0 0 0 0 100 54 0 0 0 112 0 48 0 56 110 80 6 0 170 0 88 10 2 0 134 74 0 104 0 0 162 24 144 0 162 0 72 134 0 76 108 146 0 0 0 118 116 0 0 0 78 154 106 0 146 14 0 0 0 102 0 0 0 98 58 0 158 0 90 146 0 0 78 0 166 0 98 74 168 54 4 96 42 10 98 114 16 14 28 0 0 0 0 0	108 0 0 0 38 64 74 0 136 54 0 94 138 74 40 10 30 0 8 0 0 0 20 0 8 0 0 0 60 66 76 0 68 0 86 50 38 150 34 96 0 158 42 64 0 44 0 0 0 0 0 0 0 0 68 0 0 0 0 0 150 0 0 0 0 86 0 26 82 124 0 0 58 96 0 164 94 0 0 0 0 100 54 0 0 0 112 0 48 0 56 110 80 6 0 170 0 88 10 2 0 134 74 0 104 0 0 162 24 144 0 162 0 72 134 0 76 108 146 0 0 0 118 116 0 0 0 78 154 106 0 146 14 0 0 0 102 0 0 0 98 58 0 158 0 90 146 0 0 78 0 166 0 98 74 168 54 4 96 42 10 98 114 16 14 28 0 0 0 0 0	✓

Passed all tests! ✓

Correct

Marks for this submission: 10.00/10.00.

Back to Course