| | |
|---|---|
| **Status** | Finished |
| **Started** | Friday, 6 December 2024, 12:38 AM |
| **Completed** | Friday, 6 December 2024, 7:17 AM |
| **Duration** | 6 hours 39 mins |
| **Grade** | **10.00** out of 10.00 (**100**%) |

**Question 1**

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<br>78 167 90 119 99 | 0.306 0.655 0.353 0.467 0.388 |

**Answer:**

```cpp
#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 normalize(double *out, int *in, int n)
{
    cin >> n;
    for(int i = 0; i < n; i++)
    {
        cin >> *(in + i);
        *(out + i) = *(in + i) * 1.0 / 255.0;
        //cout << fixed << setprecision(3) << *(out + i) << " ";
    }
}
```

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

Passed all tests! ✓

Correct

Marks for this submission: 10.00/10.00.

Back to Course