ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN
KHOA CÔNG NGHỆ THÔNG TIN

NHẬP MÔN LẬP TRÌNH

Bài thực hành 06

MẢNG MỘT VÀ HAI CHIỀU

Giảng viên thực hành: Lê Đức Khoan

Thành phố Hồ Chí Minh, 05/2025

Mục lục

1 Khái niệm

Trong phần này, chúng ta sẽ cùng tìm hiểu về mảng một cũng như hai chiều – một trong những cấu trúc dữ liệu cơ bản và thiết yếu trong lập trình. Như đã đề cập ở các phần trước, hầu hết các ngôn ngữ lập trình đều cung cấp sẵn một số kiểu dữ liệu cơ bản để giúp chuyển hóa các bài toán thực tế thành chương trình máy tính. Chẳng hạn, kiểu 'int' dùng để biểu diễn số nguyên, kiểu 'float' dùng cho số thực; hay khái niệm hàm giúp chia nhỏ một bài toán lớn thành các tác vụ nhỏ dễ xử lý hơn.

Tương tự mảng 1 chiều và 2 chiều ra đời để giúp chính ta dễ dàng hiện thực mã nguồn hơn trong nhiều trường hợp. Ví dụ, trong quá trình lập trình, đôi khi chúng ta phải làm việc với nhiều giá trị có cùng kiểu dữ liệu. Thay vì khai báo từng biến riêng lẻ cho mỗi giá trị, ngôn ngữ lập trình cung cấp cho chúng ta một công cụ vô cùng hữu ích - mảng (array).

- Mảng một chiều là một tập hợp các phần tử cùng kiểu dữ liệu, được sắp xếp theo một dãy liên tục và truy cập thông qua chỉ số. Mảng một chiều rất phù hợp cho các bài toán liên quan đến danh sách như: danh sách điểm số, danh sách sản phẩm, danh sách tên người dùng,...

  Ví dụ, để lưu điểm trung bình của 100 học sinh trong một lớp, thay vì khai báo 100 biến số thực, chúng ta chỉ cần một mảng có 100 phần tử. Điều này giúp chương trình ngắn gọn, dễ hiểu và dễ thao tác hơn nhiều.

- Mảng hai chiều mở rộng khái niệm của mảng một chiều, cho phép chúng ta lưu trữ dữ liệu dưới dạng bảng – gồm nhiều dòng và cột. Đây là lựa chọn lý tưởng cho những tình huống dữ liệu mang tính lưới, như ma trận, bảng điểm, bản đồ trò chơi,...

  Chẳng hạn, để lưu điểm từng môn học của 100 học sinh, ta có thể dùng một mảng hai chiều: mỗi dòng đại diện cho một học sinh, mỗi cột tương ứng với một môn học. Như vậy, việc truy xuất và xử lý dữ liệu theo hàng – cột trở nên cực kỳ thuận tiện và trực quan.

2 Các kỹ thuật cơ bản trên mảng một chiều

Trong phần này ta sẽ lần lượt giới thiệu các kĩ thuật cần biết khi sử dụng mảng một chiều trong ngôn ngữ C++.

2.1 Khai báo và khởi gán mảng một chiều

Để khai báo mảng một chiều ta sử dụng cú pháp:

```
<Data type> Name[<size>]
```

Ví dụ ta khai báo một mảng số nguyên có tên là *month* gồm 12 phần tử:

```
int month[12];
```

Với khai báo như trên thì ta có thể hiểu trong RAM có một vùng nhớ liên tục gồm 12 phần tử, mỗi phần tử có kích thước là 4 byte (kích thước tùy ngôn ngữ lập trình quyết định).

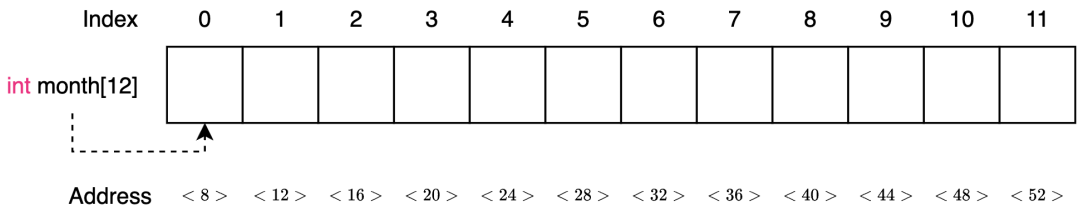

Figure 1: Mảng month được khởi tạo trong RAM

Với câu lệnh khai báo trên thì các giá trị trong mảng chưa được gán giá trị. Ta cần khởi gán giá trị các giá trị cho mảng. Ta có thể gán như sau:

```
int month[12] = {31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31};
```

Như vậy ta vừa định nghĩa giá trị cho mảng month (giá trị này do ta gán, không phải do người dùng nhập thông qua cin). Kết quả ta được như hình 2:

Ngoài ra để khởi tạo toàn bộ mảng với cùng một giá trị ta có thể sử dụng cách như sau. Ở đây ta khởi tạo một mảng check có 12 phần tử và các phần tử có giá trị mặc định là 1.

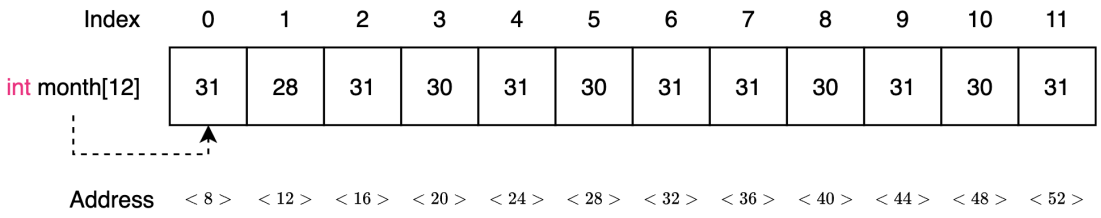

Figure 2: Mảng month được khởi tạo với các giá trị trong RAM

```
1    int check[12] = {1};
```

Ta có kết quả như sau:

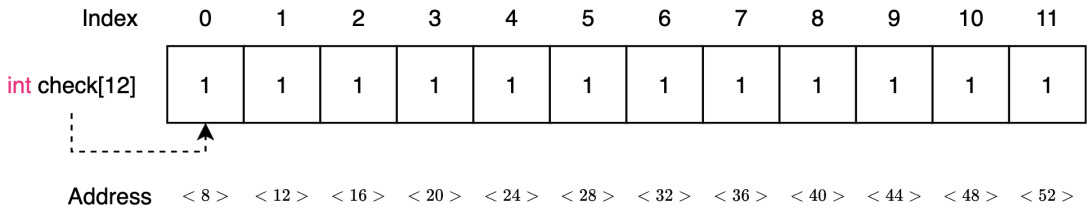

Figure 3: Mảng check được khởi tạo với cùng một giá trị trong RAM

2.2 Xuất nhập mảng một chiều

Tiếp theo ta sẽ học cách nhập xuất mảng một chiều. Kĩ thuật khởi gán giá trị như trên cũng có thể xem là nhập dữ liệu cho mảng một chiều, tuy nhiên đó là ta khởi gán cứng cho mảng. Trong phần này ta sẽ học cách cho người dùng nhập giá trị thông qua bàn phím bằng hàm cin cũng như có thể xuất dữ liệu mảng ra màn hình console. Xét đoạn mã sau minh họa về cách nhập xuất mảng số nguyên từ bàn phím.

```
1    #include <iostream>
2    using namespace std;
3
4    void input_array(int arr[], int n) {
5        cout << "Enter " << n << " integers:" << endl;
6        for (int i = 0; i < n; ++i) {
```

```
            cin >> arr[i];
        }
    }

    void print_array(int arr[], int n) {
        cout << "The array is: ";
        for (int i = 0; i < n; ++i) {
            cout << arr[i] << " ";
        }
        cout << endl;
    }

    int main() {
        int n;
        cout << "Enter number of elements: ";
        cin >> n;

        // Using variable-length array (VLA)
        // Works with some compilers like GCC
        int arr[n];
        input_array(arr, n);
        print_array(arr, n);
        return 0;
    }
```

Đoạn chương trình này minh họa cách **nhập và xuất một mảng số nguyên** trong C++, với cấu trúc rõ ràng thông qua việc tách riêng hai chức năng chính thành các hàm:

- **Hàm input_array**:
  - Nhận vào một mảng số nguyên arr[] và số lượng phần tử n.
  - Yêu cầu người dùng nhập n số nguyên từ bàn phím và lưu vào mảng.
- **Hàm print_array**:
  - Nhận vào mảng arr[] và số lượng phần tử n.

– In ra tất cả các phần tử của mảng theo thứ tự đã nhập.

- **Hàm main**:
  – Bắt đầu bằng việc yêu cầu người dùng nhập vào số lượng phần tử của mảng n.
  – Khai báo một mảng với n phần tử (dạng mảng có độ dài biến đổi - VLA), điều này chỉ tương thích với một số trình biên dịch như GCC.
  – Gọi hàm **input_array** để nhập giá trị cho mảng.
  – Gọi hàm **print_array** để hiển thị nội dung mảng ra màn hình.

Lưu ý:

- Việc truyền mảng vào hàm chỉ cần truyền tên mảng (ví dụ: arr), vì trong C++, tên mảng chính là con trỏ trỏ đến phần tử đầu tiên.
- Biến n được dùng để xác định số lượng phần tử thực tế cần nhập và in ra, tránh truy cập vào vùng nhớ không hợp lệ.

2.3 Một số kỹ thuật cơ bản

2.3.1 Thao tác tìm kiếm

Trong ví dụ này ta sẽ tìm và trả ra vị trí đầu tiên của phần tử được tìm kiếm.

```
1   #include <iostream>
2   using namespace std;
3
4   int linear_search(int arr[], int n, int key) {
5       for (int i = 0; i < n; ++i) {
6           if (arr[i] == key)
7               return i; // return the index where key is found
8       }
9       return -1; // key not found
10  }
11
12  int main() {
13      int n, key;
```

```
14          cout << "Enter number of elements: ";
15          cin >> n;
16          int arr[n];
17          input_array(arr, n);
18
19          cout << "Enter value to search for: ";
20          cin >> key;
21
22          int result = linear_search(arr, n, key);
23
24          if (result != -1) {
25              cout << "Value found at index: " << result << endl;
26          } else {
27              cout << "Value not found in array." << endl;
28          }
29          return 0;
30      }
```

Đoạn mã trên là một chương trình C++ thực hiện thuật toán **tìm kiếm tuyến tính** trong mảng một chiều. Chương trình bắt đầu bằng việc yêu cầu người dùng nhập vào số lượng phần tử của mảng và các giá trị tương ứng. Sau đó, người dùng nhập một giá trị cần tìm (gọi là key). Hàm **linear_search** sẽ duyệt tuần tự từng phần tử trong mảng để so sánh với giá trị cần tìm. Nếu tìm thấy, hàm sẽ trả về chỉ số (index) của phần tử đầu tiên khớp với giá trị đó. Ngược lại, nếu không tìm thấy, hàm sẽ trả về -1. Dựa vào kết quả trả về, chương trình sẽ in ra thông báo cho biết giá trị cần tìm nằm ở vị trí nào trong mảng, hoặc thông báo rằng giá trị không tồn tại.

2.3.2 Thao tác sắp xếp

Một kĩ thuật khác cũng khá phổ biến trong mảng một chiều đó là thao tác sắp xếp. Giả sử đoạn mã sau sẽ sắp xếp tăng mảng số nguyên một chiều.

```
1   #include <iostream>
2   using namespace std;
3
```

```
4   void sort_array(int arr[], int n) {
5       for (int i = 0; i < n - 1; ++i) {
6           int min_index = i;
7           for (int j = i + 1; j < n; ++j) {
8               if (arr[j] < arr[min_index])
9                   min_index = j;
10          }
11          int temp = arr[i];
12          arr[i] = arr[min_index];
13          arr[min_index] = temp;
14      }
15  }
16
17  int main() {
18      int n;
19      cout << "Enter number of elements: ";
20      cin >> n;
21
22      int arr[n];
23      input_array(arr, n);
24      sort_array(arr, n);
25      cout << "Sorted array: ";
26      print_array(arr, n);
27
28      return 0;
29  }
```

Đoạn mã trên thực hiện thuật toán **sắp xếp chọn** để sắp xếp mảng số nguyên theo thứ tự tăng dần. Chương trình bao gồm hai phần chính:

1. **Hàm sort_array:** Hàm này sắp xếp mảng sử dụng thuật toán sắp xếp chọn. Nó duyệt qua từng phần tử của mảng và tìm phần tử nhỏ nhất trong phần mảng chưa được sắp xếp, sau đó hoán đổi phần tử nhỏ nhất này với phần tử ở vị trí hiện tại. Quá trình này được lặp lại cho đến khi mảng được sắp xếp hoàn chỉnh.

2. **Hàm main:** Hàm này thực hiện việc nhập dữ liệu và xuất kết quả. Nó yêu cầu người dùng nhập số lượng phần tử của mảng và các phần tử của mảng, sau đó gọi hàm **sort_array** để sắp xếp mảng và gọi hàm **print_array** để in ra mảng đã sắp xếp.

Thuật toán sắp xếp chọn hoạt động theo các bước sau:

1. Bắt đầu từ phần tử đầu tiên, thuật toán tìm phần tử nhỏ nhất trong phần mảng chưa được sắp xếp.
2. Hoán đổi phần tử nhỏ nhất với phần tử ở vị trí đầu tiên của phần mảng chưa được sắp xếp.
3. Tiếp tục lặp lại quá trình này cho đến khi mảng được sắp xếp hoàn chỉnh.

Đây là một thuật toán sắp xếp đơn giản với độ phức tạp thời gian là $O(n^2)$, trong đó n là số lượng phần tử của mảng. Mặc dù thuật toán này dễ hiểu và cài đặt, nhưng không phải là thuật toán hiệu quả khi làm việc với các mảng lớn.

2.4 Một số kỹ thuật nâng cao

2.4.1 Thêm một phần tử vào một vị trí

```cpp
#include <iostream>
using namespace std;

bool insert_element(int arr[], int& n,
                    int max_size, int element, int position) {
    if (position < 0 || position > n || n >= max_size) {
        return false;
    }
    for (int i = n; i > position; --i) {
        arr[i] = arr[i - 1];
    }

    arr[position] = element;
    ++n;
```

```
15          return true;
16      }
17
18      int main() {
19          int max_size = 10;
20          int arr[max_size];
21          int n = 5;
22          arr[0] = 1;
23          arr[1] = 2;
24          arr[2] = 3;
25          arr[3] = 4;
26          arr[4] = 5;
27          cout << "Initial array: ";
28          print_array(arr, n);
29
30          int element = 99;
31          int position = 2;
32
33          if (insert_element(arr, n, max_size, element, position)) {
34              cout << "Array after element added: ";
35              print_array(arr, n);
36          } else {
37              cout << "Insert fail!" << endl;
38          }
39          return 0;
40      }
```

Đoạn mã C++ bên trên định nghĩa một hàm có tên **insert_element** dùng để chèn một phần tử vào một mảng số nguyên một chiều tại vị trí bất kỳ. Hàm trả về giá trị kiểu `bool`, biểu thị việc chèn thành công (`true`) hoặc thất bại (`false`).

Chi tiết hoạt động của hàm:

- **Tham số đầu vào**:

- `arr[]`: mảng số nguyên cần chèn phần tử.
- `n`: số lượng phần tử hiện tại trong mảng.
- `max_size`: kích thước tối đa của mảng.
- `element`: giá trị cần chèn.
- `position`: vị trí cần chèn phần tử (bắt đầu từ 0).

- **Kiểm tra hợp lệ**:
  - Nếu vị trí chèn không hợp lệ hoặc mảng đã đầy, hàm trả về `false`.
- **Thực hiện chèn phần tử**:
  - Dịch các phần tử từ vị trí chèn về cuối mảng sang phải một bước.
  - Gán phần tử mới vào vị trí cần chèn.
  - Tăng số lượng phần tử (`n`) lên 1.
  - Trả về `true` nếu chèn thành công.

Trong hàm main: Một mảng có sẵn 5 phần tử được khởi tạo. Sau đó, phần tử `99` được chèn vào vị trí thứ 2. Nếu chèn thành công, mảng được in ra để hiển thị kết quả.

2.4.2 Xoá phần tử ở một vị trí

```cpp
#include <iostream>
using namespace std;

bool delete_with_order(int arr[], int& n, int pos) {
    if (pos < 0 || pos >= n) return false;
    for (int i = pos; i < n - 1; ++i) {
        arr[i] = arr[i + 1];
    }
    --n;
    return true;
}

```

```cpp
bool delete_without_order(int arr[], int& n, int pos) {
    if (pos < 0 || pos >= n) return false;
    arr[pos] = arr[n - 1];
    --n;
    return true;
}

int main() {
    int arr1[10] = {10, 20, 30, 40, 50};
    int arr2[10] = {10, 20, 30, 40, 50};
    int n1 = 5, n2 = 5;

    delete_with_order(arr1, n1, 2); // Delete second element (30)
    cout << "Remove with order: ";
    print_array(arr1, n1);

    delete_without_order(arr2, n2, 2); // Delete second element (30)
    cout << "Remove without order: ";
    print_array(arr2, n2);

    return 0;
}
```

Trong bài toán xử lý mảng một chiều, một thao tác cơ bản là xoá phần tử tại vị trí bất kỳ. Có hai cách tiếp cận phổ biến:

1. **Xoá có đảm bảo thứ tự**: Sau khi xoá phần tử, tất cả các phần tử phía sau sẽ được dịch trái để giữ nguyên trật tự ban đầu.
2. **Xoá không đảm bảo thứ tự**: Phần tử cần xoá sẽ được thay bằng phần tử cuối cùng của mảng, không cần dịch phần tử, do đó thuật toán nhanh hơn nhưng làm thay đổi trật tự.

Chi tiết hàm thực hiện

- Cả hai hàm đều có kiểu trả về `bool`, trả về `true` nếu xoá thành công, hoặc `false` nếu vị trí xoá không hợp lệ.

- Tham số `n` được truyền tham chiếu để cập nhật lại kích thước của mảng sau khi xoá.
- Trong trường hợp **đảm bảo thứ tự**, sử dụng vòng lặp để dịch các phần tử từ phải sang trái.
- Trong trường hợp **không đảm bảo thứ tự**, chỉ cần gán phần tử cuối vào vị trí cần xoá và giảm kích thước mảng.

3 Các kỹ thuật cơ bản trên mảng hai chiều

3.1 Khai báo và khởi gán mảng hai chiều

Dưới đây là một số cách để khởi tạo mảng hai chiều:

1. **Khai báo mảng hai chiều (không khởi tạo giá trị)**: Khởi tạo mảng hai chiều gồm 3 hàng và 4 cột.

```
        int a[3][4];
```

2. **Khai báo mảng hai chiều và khởi tạo toàn bộ giá trị**: Khởi tạo mảng hai chiều có 2 hàng và 3 cột.

```
        int a[2][3] = {
            {1, 2, 3},
            {4, 5, 6}
        };
```

3. **Khai báo mảng hai chiều và khởi tạo một phần (phần còn lại bằng 0)**

```
        int a[2][3] = {
            {1},
            {2}
        }; // -> a = {{1, 0, 0}, {2, 0, 0}}
```

4. **Khai báo mảng hai chiều và khởi tạo toàn bộ bằng 0**

```
        int a[3][3] = {}; // All elements are 0
```

3.2 Xuất nhập mảng hai chiều

```cpp
#include <iostream>
using namespace std;
#define ROW 100
#define COL 100

void input_array(int rows, int cols, int arr[][COL]) {
    for (int i = 0; i < rows; ++i)
        for (int j = 0; j < cols; ++j) {
            cout << "arr[" << i << "][" << j << "] = ";
            cin >> arr[i][j];
        }
}

void print_array(int rows, int cols, int arr[][COL]) {
    for (int i = 0; i < rows; ++i) {
        for (int j = 0; j < cols; ++j)
            cout << arr[i][j] << "\t";
        cout << endl;
    }
}

int main() {
    int rows, cols;
    cout << "Number of rows: ";
    cin >> rows;
    cout << "Number of columns: ";
    cin >> cols;

    int arr[ROW][COL];

    input_array(rows, cols, arr);
    print_array(rows, cols, arr);
```

```
33
34          return 0;
35      }
```

Đoạn chương trình C++ này thực hiện việc **nhập và xuất một mảng hai chiều** kích thước tối đa 100×100. Cấu trúc chương trình được tổ chức thành các hàm riêng biệt để dễ quản lý và tái sử dụng mã nguồn.

- Hằng số `ROW` và `COL` được định nghĩa là 100, giới hạn tối đa cho số hàng và số cột của mảng.
- Mảng hai chiều `arr[ROW][COL]` được khai báo trong hàm `main`.
- Người dùng nhập số hàng và số cột mong muốn từ bàn phím thông qua hai biến `rows` và `cols`.
- Hàm `input_array` thực hiện việc nhập dữ liệu từ bàn phím cho mảng hai chiều.
- Hàm `print_array` xuất toàn bộ nội dung mảng ra màn hình theo dạng bảng.

4 Các điểm lưu ý khi làm việc với mảng

4.1 Chỉ số mảng bắt đầu từ 0

- Trong C++, chỉ số đầu tiên của mảng luôn là 0.
- Nếu chúng ta khai báo int a[10]; thì phần tử đầu là a[0] và phần tử cuối là a[9].
- **Lỗi phổ biến:** Truy cập a[10] sẽ gây lỗi vượt quá giới hạn mảng (**out of bounds**).

4.2 Kích thước của mảng

Kích thước của mảng phải là hằng số hoặc được xác định tại thời điểm biên dịch. Tuy nhiên một số trình biên dịch hỗ trợ **VLA (Variable Length Array)** như **GCC** cho phép kích thước mảng xác định tại thời điểm chạy.

```
1    const int N = 100;
2    int a[N];
3
4    // Variable Length Array (VLA)
5    int n;
6    cin >> n;
7    int a[n];
```

4.3 Truyền mảng vào hàm

- Khi truyền mảng 1 chiều hoặc 2 chiều vào hàm, bạn không cần dùng dấu & vì mảng được truyền theo dạng tham chiếu (reference) mặc định.
- Tuy nhiên, các biến khác (như số phần tử) cần truyền theo tham chiếu nếu muốn thay đổi giá trị từ trong hàm.

```
1    void intput_array(int a[], int& n);
```

4.4 Mảng hai chiều cần xác định số cột khi truyền vào hàm

- Khi truyền mảng 2 chiều vào hàm, chúng ta phải chỉ rõ số cột.

```
1    void input_array(int arr[][100], int rows, int cols);
```

5 Yêu cầu bài nộp

Sinh viên được cung cấp một thư mục mã nguồn **lab_06** chứa thư viện, các định nghĩa hàm. Mã nguồn đã bao gồm đầy đủ các định nghĩa hàm cho tất cả các bài tập trong Lab 06. Sinh viên thực hiện thực các hàm theo prototype đã được định nghĩa ở file **.h** và có thể viết thêm hàm nếu cần thiết (các hàm hỗ trợ chỉ được viết ở file **.cpp** và không được viết thêm hàm vào file **main.cpp**. File **main.cpp** chỉ để nhập xuất dữ liệu và kiểm tra các hàm).

Chú ý: Sinh viên không được tự ý sửa đổi bất kỳ thành phần nào của mã nguồn mẫu. Nếu mã nguồn biên dịch không thành công khi chấm thì sẽ nhận điểm 0 cho bài tập đó.

Khi nộp bài sinh viên **đổi tên thư mục lab_06 thành MSSV và zip toàn bộ mã nguồn** thành file **MSSV.zip** với MSSV là mã số sinh viên được nhà trường cung cấp. Các thành phần ở trong thư mục phải giống như thư mục được cung cấp ban đầu. Không thêm bất kỳ file nay thư mục nào khác.

Tên file zip mẫu:

```
24123456.zip
```

Tất cả các trường hợp làm sai yêu cầu sẽ nhận điểm 0 cho bài thực hành. Vì thế sinh viên cần đọc kỹ và thực hiện đúng yêu cầu.

6 Hướng dẫn chạy file thực hành lab_06

Tổ chức thư mục của lab_06 như sau:

```
lab_06
   |-- 1d_array
   |        |-- array_1d_utils.h
   |        |-- array_1d_utils.cpp
   |
   |-- 2d_array
   |        |-- array_2d_utils.h
   |        |-- array_2d_utils.cpp
   |
   |-- utils
   |     |-- format_utils.h
   |     |-- format_utils.cpp
   |-- main.cpp
   |-- Makefile
```

Vì có nhiều file được include vào trong file main để chạy chương trình do đó để nhanh chóng compile và chạy chương trình. Trong bài lab này chúng ta sẽ tiếp cận với giải pháp dùng Makefile để compile và chạy chương trình. Gõ vào terminal để kiểm tra version của Make:

```
make -v
```

Nếu không xuất hiện version thì máy chưa có Make. Khi đó, các bạn có thể tham khảo hướng dẫn cài đặt Make. Sau khi đã cài xong, các bạn đi đến thư mục đang chưa file **Makefile** và có thể thực hiện các câu lệnh sau:

- **make all**: Compile code ở tất cả các file để tạo ra file thực thi (**.exe**). Đây là default command nên nếu bạn gõ **make** thì nó sẽ tự động hiểu là **make all**.
- **make clean**: Để xoá bỏ file thực thi vừa tạo.
- **make run**: Thực hiện compile code và chạy chương trình.

Các câu lệnh trong Makefile bao gồm:

```
1  # Compiler and flags
2  CXX = g++
3  CXXFLAGS = -std=c++17 -Wall -g -I utils -I 1d_array -I 2d_array
4
5  # Name of execution file
6  TARGET = main
7
8  # List of source files
9  SRC = main.cpp utils/format_utils.cpp 1d_array/array_1d_utils.cpp\
10                                       2d_array/array_2d_utils.cpp
11
12 # Default command
13 all:
14         $(CXX) $(CXXFLAGS) $(SRC) -o $(TARGET)
15
16 # Clean command
17 clean:
18         rm -f $(TARGET)
19
20 # Compile and run command
21 run: all
22         ./$(TARGET)
23
```

- **Câu lệnh số 2**: Khai báo compiler là **g++**.
- **Câu lệnh số 3**: Khai báo thêm các flags khi compile code. Với các tham số sau **-I** chỉ các thư mục chưa file **.h**. Nếu có thêm nhiều thư viện tự định nghĩa ta sẽ thêm tiếp các cặp **-I library_directory**. Mỗi thư mục sẽ đi với một ký tự **-I**.
- **Câu lệnh số 6**: Khai báo tên file **Thực thi** được tạo ra khi compile mã nguồn.
- **Câu lệnh số 13**: Compile mã nguồn và tạo ra file thực thi. Với câu lệnh tương tự trong

những tuần trước: **g++ main.cpp -o main**. Câu lệnh này sẽ được chạy khi ta gọi: **make all** hoặc **make**.

- **Câu lệnh số 17**: Xoá file thực thi vừa được tạo ra. Lệnh được thực thi khi gọi: **make clean**.
- **Câu lệnh số 21**: Thực hiện compile và chạy file thực thi. Lệnh được thực thi khi gọi: **make run**. Khi gọi lệnh này sẽ thực hiện hai việc liên tục là **make all** và **./main**. Với **make all** đã được trình bày ở trên và **./main** là để chạy file thực thi.

Sau khi đã chạy được toàn bộ chương trình, sinh viên tiến hành làm bài thực hành. Bài thực hành sẽ bao gồm 2 bài lớn tương ứng với 2 thư mục là **1d_array** và **2d_array** theo thứ tự.

7 Bài tập

7.1 Xây dựng thư viện hỗ trợ mảng 1 chiều

Trong bài tập này sinh viên cần xây dựng môt tập tin **.h** để khai báo các chức năng của thư viện hỗ trợ mảng một chiều và một tập tin **.cpp** để định nghĩa các chức năng được khai báo trong file **.h**. Các chức năng được thực thi trong thư viện thời gian bao gồm:

1. Nhập mảng một chiều. Chi tiết tại (7.1.1)
2. Xuất mảng một chiều. Chi tiết tại (7.1.2)
3. Tìm kiếm phần tử. Chi tiết tại (7.1.3)
4. Thêm phần tử vào vị trí. Chi tiết tại (7.1.4)
5. Xoá phần tử. Chi tiết tại (7.1.5)
6. Thực hiện tính toán tổng, tích các phần tử. Chi tiết tại (7.1.6)
7. Tính giá trị trung bình các phần tử. Chi tiết tại (7.1.7)
8. Tính giá trị trung vị của mảng. Chi tiết tại (7.1.8)
9. Tìm giá trị nhỏ, lớn nhất. Chi tiết tại (7.1.9)
10. Sắp xếp các phần tử tăng dần. Chi tiết tại (7.1.10)
11. Sắp xếp các phần tử giảm dần. Chi tiết tại (7.1.11)

Sinh viên được cung cấp hai file:

- **1d_array_utils.h**: để định nghĩa các chức năng của thư viện.
- **1d_array_utils.cpp**: để hiện thực các chức năng. (Sinh viên có thể viết thêm các hàm phụ cần thiết để sử dụng trong các hàm chức năng. Các hàm này được định nghĩa trực tiếp trong file **1d_array_utils.cpp**).

Chi tiết từng chức năng được cung cấp ở phần sau.

7.1.1 Nhập mảng

Viết hàm nhập các phần tử trong mảng vào từ bàn phím.

```
1  // array_1d_utils.h
2
3  void input_array(int arr[], int n);
```

7.1.2 Xuất mảng

Viết hàm in các giá trị trong mảng ra màn hình console cách nhau bằng khoảng trắng.

```
1  // array_1d_utils.h
2
3  void print_array(int arr[], int n);
```

7.1.3 Tìm kiếm phần tử

Viết hàm trả về chỉ số chỉ của vị trí đầu tiên của phần tử được tìm kiếm trong mảng. Nếu không tồn tại trả về -1.

```
1  // array_1d_utils.h
2
3  int search(int arr[], int n, int value);
```

Testcase mẫu:

```
Input: arr = [1 31 20 1 10 20 35], value = 20
Output: 2
```

Giải thích: Chỉ số của vị trí đầu tiên của phần tử 20 trong mảng là 2.

7.1.4 Thêm phần tử vào tại vị trí

Viết hàm để thêm một giá trị vào một vị trí bất kỳ trong mảng. Trả về True nếu thành công, ngược lại trả về False.

```
// array_1d_utils.h

bool add_at_index(int arr[], int &n, int value, int index);
```

Testcase mẫu:

```
Input: arr = [23 10 30 0 10 0] , value = 5, index = 0
Output: arr = [5 23 10 30 0 10 0]
```

7.1.5 Xoá phần tử tại vị trí

Viết hàm xoá phần tử tại một vị trí bất kỳ và đảm bảo thứ tự xuất hiện của các phần tử. Nếu thành công trả về True, ngược lại trả về False.

```
// array_1d_utils.h

bool remove_at_index(int arr[], int &n, int max_size, int index);
```

Testcase mẫu:

```
Input: arr = [23 10 30 0 10 0] , index = 0
Output: arr = [10 30 0 10 0]
```

7.1.6 Thực hiện phép tính tổng, tích

Viết hàm tính tổng hoặc tích của mảng một chiều. Với biến `op` mang hai giá trị là "+" và "*". Với "+" tương ứng với phép cộng và "*" tương ứng với phép nhân.

```
// array_1d_utils.h

long long compute_array(int arr[], int n, char op = '+');
```

Testcase mẫu:

```
Input: arr = [23 10 30 0 10 0] , op = '+'
Output: 73
```

7.1.7 Thực hiện phép tính giá trị trung bình

Viết hàm trả về giá trị trung bình của các phần tử trong mảng. Làm tròn đến 2 chữ số thập phân sử dụng hàm **round_to_decimal**.

```
1  // array_1d_utils.h
2
3  float compute_mean(int arr[], int n);
```

Testcase mẫu:

```
Input: arr = [23 10 30 0 10 0]
Output: 12.17
```

7.1.8 Thực hiện phép tính giá trị trung vị

Viết hàm trả về giá trị trung vị của các phần tử trong mảng. Làm tròn đến 2 chữ số thập phân sử dụng hàm **round_to_decimal**. Nếu số phần tử lẻ trả về vị trí chính giữa, ngược lại trả về trung bình hai giá trị chính giữa.

```
1  // array_1d_utils.h
2
3  float compute_median(int arr[], int n);
```

Testcase mẫu:

```
Input: arr = [23 10 30 0 10 0]
Output: 10
```

7.1.9 Tìm giá trị nhỏ, lớn nhất

Viết hàm tìm giá trị nhỏ nhất hoặc lớn nhất trong một mảng. Với biến `max` là True nếu lấy giá trị max, ngược lại lấy giá trị min.

```
// array_1d_utils.h

int get_max_min(int arr[], int n, bool max = true);
```

Testcase mẫu:

```
Input: arr = [23 10 30 0 10 0], max = false
Output: 0
```

7.1.10 Sắp xếp tăng dần

Viết hàm sắp xếp các phần tử trong mảng theo thứ tự tăng dần.

```
// array_1d_utils.h

void sort_ascending(int arr[], int n);
```

Testcase mẫu:

```
Input: arr = [23 10 30 0 10 0]
Output: arr = [0 0 10 10 23 30]
```

7.1.11 Sắp xếp giảm dần

Viết hàm sắp xếp các phần tử trong mảng theo thứ tự giảm dần.

```
// array_1d_utils.h

void sort_descending(int arr[], int n);
```

Testcase mẫu:

```
Input: arr = [23 10 30 0 10 0]
Output: arr = [30 23 10 10 0 0]
```

7.2 Xây dựng thư viện hỗ trợ xử lý mảng hai chiều.

Trong bài tập này sinh viên cần xây dựng môt tập tin **.h** để khai báo các chức năng của thư viện hỗ trợ mảng hai chiều và một tập tin **.cpp** để định nghĩa các chức năng được khai báo trong file **.h**. Các chức năng được thực thi trong thư viện thời gian bao gồm:

1. Nhập mảng hai chiều. Chi tiết tại (7.2.1)
2. Xuất mảng hai chiều. Chi tiết tại (7.2.2)
3. Kiểm tra phần tử tồn tại. Chi tiết tại (7.2.3)
4. Thực hiện tính toán tổng, tích các phần tử. Chi tiết tại (7.2.4)
5. Tính giá trị trung bình các phần tử. Chi tiết tại (7.2.5)
6. Tìm giá trị nhỏ, lớn nhất. Chi tiết tại (7.2.6)
7. Sắp xếp các phần tử tăng dần. Chi tiết tại (7.2.7)
8. Sắp xếp các phần tử giảm dần. Chi tiết tại (7.2.8)
9. Hoán vị dòng/cột ma trận vuông. Chi tiết tại (7.2.9)
10. Chuyển vị ma trận vuông. Chi tiết tại (7.2.10)
11. Xoay ma trận vuông. Chi tiết tại (7.2.11)
12. Đảo ma trận vuông. Chi tiết tại (7.2.12)

Sinh viên được cung cấp hai file:

- **2d_array_utils.h**: để định nghĩa các chức năng của thư viện.
- **2d_array_utils.cpp**: để hiện thực các chức năng. (Sinh viên có thể viết thêm các hàm phụ cần thiết để sử dụng trong các hàm chức năng. Các hàm này được định nghĩa trực tiếp trong file **2d_array_utils.cpp**).

Chi tiết từng chức năng được cung cấp ở phần sau.

7.2.1 Nhập mảng hai chiều

Viết hàm nhập các phần tử vào mảng hai chiều. Theo thứ tự từ trái sang phải, từ trên xuống dưới.

```
// array_2d_utils.h
const int MAX_ROWS = 100;
const int MAX_COLS = 100;

void input_2d(int arr[][MAX_COLS], int rows, int cols);
```

7.2.2 Xuất mảng hai chiều

Viết hàm in ra các phần tử của mảng hai chiều. Theo thứ tự từ trái sang phải, từ trên xuống dưới cách nhau bằng khoảng trắng.

```
// array_2d_utils.h
const int MAX_ROWS = 100;
const int MAX_COLS = 100;

void print_2d(int arr[][MAX_COLS], int rows, int cols);
```

7.2.3 Tìm kiếm phần tử tồn tại

Viết hàm kiểm tra sự tồn tại của một giá trị trong mảng 2 chiều. Trả về True nếu tồn tại, ngược lại trả về False.

```
// array_2d_utils.h
const int MAX_ROWS = 100;
const int MAX_COLS = 100;

bool check_exist(int arr[][MAX_COLS], int rows, int cols, int value);
```

7.2.4 Thực hiện phép tính tổng, tích

Viết hàm tính tổng hoặc tích của mảng hai chiều. Với biến `op` mang hai giá trị là "+" và "*". Với "+" tương ứng với phép cộng và "*" tương ứng với phép nhân.

```
// array_2d_utils.h
const int MAX_ROWS = 100;
const int MAX_COLS = 100;

long long compute_2d(int arr[][MAX_COLS],
                     int rows,
                     int cols,
                     char op = '+');
```

7.2.5 Thực hiện phép tính giá trị trung bình

Viết hàm trả về giá trị trung bình của các phần tử trong mảng hai chiều. Làm tròn đến 2 chữ số thập phân sử dụng hàm **round_to_decimal**.

```
// array_2d_utils.h
const int MAX_ROWS = 100;
const int MAX_COLS = 100;

float compute_mean_2d(int arr[][MAX_COLS],
                      int rows,
                      int cols);
```

7.2.6 Tìm giá trị nhỏ, lớn nhất

Viết hàm tìm giá trị nhỏ nhất hoặc lớn nhất trong mảng 2 chiều. Với biến `max` là True nếu lấy giá trị max, ngược lại lấy giá trị min.

```
// array_2d_utils.h
const int MAX_ROWS = 100;
```

```
3     const int MAX_COLS = 100;
4
5     int get_max_min_2d(int arr[][MAX_COLS],
6                                 int rows,
7                                 int cols,
8                                 bool max = true);
```

7.2.7 Sắp xếp các phần tử tăng dần

Viết hàm sắp xếp các phần tử trong mảng hai chiều tăng dần. Với biến `row_based` là true, sẽ sắp xếp các phần tử tăng dần theo hàng, ngược lại sẽ theo cột.

```
1     // array_2d_utils.h
2     const int MAX_ROWS = 100;
3     const int MAX_COLS = 100;
4
5     void sort_ascending_2d(int arr[][MAX_COLS],
6                                 int rows,
7                                 int cols,
8                                 bool row_based = true);
```

7.2.8 Sắp xếp các phần tử giảm dần

Viết hàm sắp xếp các phần tử trong mảng hai chiều giảm dần. Với biến `row_based` là true, sẽ sắp xếp các phần tử giảm dần theo hàng, ngược lại sẽ theo cột.

```
1     // array_2d_utils.h
2     const int MAX_ROWS = 100;
3     const int MAX_COLS = 100;
4
5     void sort_descending_2d(int arr[][MAX_COLS],
6                                 int rows,
```

```
                                int cols,
                                bool row_based = true);
```

7.2.9 Hoán vị dòng/cột ma trận vuông

Viết hàm hoán vị hai dòng hoặc hai cột của ma trận vuông. Với biến `row_based` là true, hoán vị hai hàng, ngược lại sẽ hoán vị hai cột.

```
// array_2d_utils.h
const int MAX_ROWS = 100;
const int MAX_COLS = 100;

void swap_2d(int arr[][MAX_COLS],
                                int rows,
                                int cols,
                                int first_index,
                                int second_index,
                                bool row_based = true);
```

7.2.10 Chuyển vị ma trận vuông

Viết hàm chuyển vị ma trận vuông.

```
// array_2d_utils.h
const int MAX_ROWS = 100;
const int MAX_COLS = 100;

void transpose_2d(int arr[][MAX_COLS],
                                int rows,
                                int cols);
```

7.2.11 Xoay ma trận vuông

Viết hàm xoay ma trận vuông. Với biến `n` đại diện cho số lần xoay phải 90 độ. Ví dụ: nếu `n`=1 thì xoay phải 90 độ, `n` = 3 thì xoay phải 270 độ.

```
// array_2d_utils.h
const int MAX_ROWS = 100;
const int MAX_COLS = 100;

void rotate_2d(int arr[][MAX_COLS],
                                int rows,
                                int cols,
                                int n);
```

7.2.12 Đảo ma trận vuông

Viết hàm đảo ma trận vuông. Với biến `row_based` là true, ta đảo ma trận theo hàng, ngược lại đảo ma trận theo cột.

```
// array_2d_utils.h
const int MAX_ROWS = 100;
const int MAX_COLS = 100;

void reverse_2d(int arr[][MAX_COLS],
                                int rows,
                                int cols,
                                bool row_based = true);
```