Programming Technology: Array

Tran Viet Khoa
tvkhoa.husc@gmail.com

Hue University— 01 Feb 2020

Mục lục

1 Giới thiệu

Kiểu dữ liệu Mảng (array) được xem như là một trong những kiểu dữ liệu quan trọng và ứng dụng rất nhiều trong các bài toán lập trình. Nó cho phép lưu trữ được một tập N các đối tượng có cùng kiểu dữ liệu trong bộ nhớ. Để từ đó ta có thể tính toán, xử lý dữ liệu của N đối tượng trên.

Hầu hết các ngôn ngữ lập trình đều có cơ chế cho phép người lập trình khai báo kiểu mảng dựa vào các kiểu dữ liệu cơ bản như số nguyên, số thực, ký tự hoặc kiểu có cấu trúc (struct, pair, tuple).

Kiểu mảng được khai báo ấn định với số phần tử xác định nên dễ dàng xác định được kích thước bộ nhớ và đây cũng chính là nhược điểm của mảng. Để khắc phục nhược điểm này người ta đưa ra cơ chế khai thác bộ nhớ động với biến con trỏ và các kiểu dữ liệu như danh sách liên kết.

Ưu điểm nổi trội của kiểu mảng là tốc độ truy cập các phần tử với độ phức tạp $O(1)$ phù hợp với việc lưu trữ dữ liệu hỗ trợ cho phép tính toán. Thuật toán như quy hoạch động sử dụng mảng làm phần chính trong thể hiện của mình.

Một số cấu trúc dữ liệu nâng cao cũng dùng mảng để biểu diễn như cây phân đoạn (segment tree), dãy tiền tố (suffix array),..Ngoài ra mảng còn dùng lưu trữ dữ liệu cho việc sắp xếp và tìm kiếm dữ liệu.

Bài viết này sẽ đề cập đến các dạng khác nhau của mảng, bao gồm:

- Mảng một chiều, nhiều chiều.
- Mảng động, vector của thư viện STL C++.
- Tìm kiếm và sắp xếp.
- Một số thuật toán trên algorithm.h.

2 Mảng một chiều

2.1 Định nghĩa

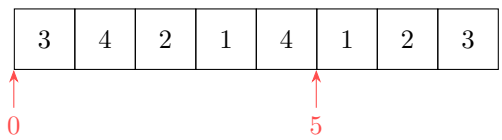
Mảng là kiểu dữ liệu cho phép lưu trữ một tập N phần tử cùng kiểu dữ liệu liên tiếp nhau trong bộ nhớ, do các phần tử có cùng kiểu nên dễ dàng tính được kích thước bộ nhớ cần cung cấp cho mảng. Các phần tử của mảng được tham chiếu (truy cập) bằng chỉ mục, địa chỉ đầu tiên là 0.

Ví dụ: Đoạn mã sau thể hiện trong bộ nhớ như hình vẽ.

index.cpp

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

int main()
{
    int A[] ={3, 4, 2, 1, 4, 1, 2, 3};
    cout <<A[0]<<" "<<A[5]<<endl;
    return 0;
}
```



Hình 1: Mảng và chỉ số

Chương trình trên in ra hai giá trị 3, 1 của mảng tại hai vị trí index là 0 và 5.

2.2 Khai báo và khởi tạo

cú pháp:

```
<TYPE> arrayName[LEN];
```

Trong đó:

- TYPE: Là các kiểu dữ liệu cơ bản, ví dụ như :int, long, double.
- arrayName: Tên biến mảng.
- LEN: Số phần tử tối đa của mảng cần khai báo.

2.2.1 Khai báo tĩnh, toàn cục

Là khai báo ngoài hàm main(), ưu điểm là được khởi tạo giá trị ban đầu. Xét ví dụ sau:

Dec1.cpp

```cpp
#include <iostream>
using namespace std;
const int nmax=100+1;
int dp[nmax];
bool vis[nmax];
struct point { int x,y; };
point ps[nmax];
int main()
{
   // Các phần tử của mảng dp được khởi tạo với giá trị mặc định là 0.
   for (int i=0; i<nmax; i++)
      cout<< dp[i] <<" ";
   cout<<endl;
   // Các phần tử mảng vis có giá trị false.
   for (int i=0; i<nmax; i++)
      cout<< vis[i] <<" ";
   cout<<endl;
   // Các tọa độ điểm ps có giá trị trường x == 0 và y == 0.
   for (int i=0; i<nmax; i++)
      cout<< ps[i].x <<" " <<ps[i].y<<endl;
   return 0;
}
```

Chú ý: Do biết trước số lượng các phần tử nên ta khai báo hằng cho kích thước mảng:

```
const int nmax=100+1;
```

Ở đây có cộng thêm 1 tránh trường hợp truy cập ngoài vùng chỉ mục theo thói quen một số người đánh chỉ mục từ 1.

2.2.2 Khai báo tĩnh, địa phương

Là khai báo trong hàm và không được khởi tạo giá trị đầu. Do vậy, tránh trường hợp rác người ta vừa khai báo vừa gán giá trị.

cú pháp:

```
<TYPE> arrayName[LEN] = {};
```

Xét ví dụ cho hai khai báo của hai mảng sau:

Dec2.cpp

```cpp
#include <iostream>
using namespace std;
const int nmax=100+1;
int main()
{
   int A[nmax] ; // các phần tử mảng A có giá trị rác (garbage)
   for (int i=0; i<nmax; i++)
      cout<< A[i] <<" ";
   cout<<endl;
   int B[nmax] = { } ; // các phần tử mảng B có giá trị là 0.
   for (int i=0; i<nmax; i++)
      cout<< B[i] <<" ";
   cout<<endl;
   return 0;
}
```

Chú ý: Bạn có thể dùng hàm có sẵn của thư viện <algorithm> để điền dữ liệu linh hoạt cho mảng.

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

int main() {
   int A[100];
   fill(A, A + 100, 7); // điền A[i] = 7, với 0 <= i <= 99.
}
```

Phương thức này rất quen thuộc cho những bạn thường hay dùng hàm fillchar() của ngôn ngữ Pascal.

2.3 Sử dụng mảng một chiều

2.3.1 Truy cập

Mảng là cấu trúc tuyến tính, các phần tử đều được đánh chỉ mục cho nên dễ dàng truy cập các phần tử với thời gian $O(1)$.

cú pháp:

```
arrayName[i]; //i là chỉ mục nhận giá trị từ 0 đến n-1
```

Ở đây có hai phần tử cần nắm bắt, một là kích thước tối đa của khai báo mảng (LEN), hai là số phần tử hiện có n của mảng và ràng buộc $n \leq LEN$. Chỉ mục của ta sẽ đọc từ $0..n-1$ hoặc từ $1..n$ theo thói quen sử dụng.

Xét ví dụ: Liệt kê các số chia hết cho 2 trong dãy số $A = \{1, 3, 4, 2, 5, 7, 6, 6, 9\}$.

ListMod2.cpp

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

int main()
{
   int A[]={1, 3, 4, 2, 5, 7, 6, 6, 9} ; // Vừa khai báo vừa gán giá trị.
   int size = sizeof(A)/sizeof(A[0]);
   // size bằng kích thước mảng A chia cho kích thước của phần từ trong mảng.

   for (int i=0; i<size; i++)
      if (A[i] %2 ==0)
         cout<< A[i] <<" ";
   cout<<endl;
   return 0;
}
```

Chú ý: Có thể dễ dàng tham số hóa tên mảng trong một macro để tính kích thước mảng bất kỳ.

```cpp
#define NELEMS(x)  (sizeof(x) / sizeof((x)[0]))
int a[17];
size_t n = NELEMS(a);
```

2.3.2 Nhập, xuất mảng

Để dễ dàng tiếp cận với hai thao tác này, ta thực hành giải bài toán sau:

Bài tập 1: Liệt kê số

Viết chương trình liệt kê các phần tử trong dãy $A = \{a_1, a_2, \ldots, a_n\}$ có giá trị lớn hơn bằng 5 và nhỏ hơn bằng 7.

Input

- Dòng thứ nhất chứa số nguyên dương n là kích thước dãy thỏa $1 \leq n \leq 10^5$.
- Dòng tiếp theo chứa các số nguyên a_i của dãy, các số cách nhau ký tự trắng. Dữ kiện đảm bảo có kết quả.

Output

- In ra số phần từ thỏa điều kiện.

Samples

input	output
7 1 2 3 4 5 6 7	5 6 7

Tham khảo:

Bài tập PT046, http://oj.hueuni.edu.vn/practice/problem/566/details

Lời giải. Bài toán dạng đơn giản với

Dữ Kiện:

- Khai báo mảng, thực ra bài toán này cũng không cần dùng mảng.
- Nhập dữ liệu cho mảng, thể hiện bằng vòng lặp điều khiển chỉ số chạy từ 0 đến $n-1$ và quét dữ liệu bằng cin.
- Duyệt mảng bằng vòng lặp và kiểm tra điều kiện và in nếu thỏa.

Các bạn quan sát mã nguồn thể hiện như sau:

sol.cpp

```
#include <iostream>
using namespace std;
const int nmax=100001;
int a[nmax];
int main()
{
   int n;
   cin >> n ;
   for (int i = 0 ; i < n ; i++)
      cin >> a[i] ;
   for (int i = 0 ; i < n ; i++ )
      if (a[i] >= 5 && a[i] <= 7 )
         cout << a[i] << " " ;
   return 0;
}
```

Nhận xét:

- Dữ liệu bài toán phụ thuộc giá trị n với ràng buộc cho trước, tuy nhiên đoạn chương trình luôn dùng một mảng với cố định số phần tử cho trước.
- Vấn đề gì sẽ xảy ra nếu số phần tử của bài toán lớn hơn, ví dụ $1 \leq n \leq 10^6 \ldots 10^8$.
- Tương tự nếu dữ liệu không là số nguyên kiểu int mà có thể là số nguyên long long hoặc số thực double.
- Mảng động giải quyết được một số vấn đề vừa nêu.

2.4 Mảng động

Là cách khai thác vùng nhớ dynamic (heap) của bộ nhớ khác với vùng nhớ static hạn chế ở trên. Để sử dụng được vùng nhớ này C, C++ cung cấp cơ chế rất đặc biệt đó là biến con trỏ, bài viết chi tiết các bạn có thể tham khảo [1].

Một mảng động là một mảng có kích thước có thể thay đổi trong quá trình thực thi chương trình. Mảng động phổ biến nhất trong C ++ là cấu trúc vectơ, có thể được sử dụng gần giống như một mảng thông thường. Đoạn mã sau đây tạo ra một vectơ trống và thêm ba phần tử vào nó:

[1] http://oj.hueuni.edu.vn/blog/article/ham-va-con-tro-ham

```
vector<int> v; // Khai báo vector v, kích thước không định trước.
v.push_back(3);// Bổ sung 3 vào cuối vector với độ phức tạp O(1), v={3}.
v.push_back(2);// Tương tự v={3,2}.
v.push_back(5);// Tương tự v={3,2, 5}.
```

Cách truy cập các phần tử hoàn toàn giống như mảng. Do vậy, bạn chỉ cần thay đổi khai báo thôi phần còn lại hoàn toàn tương tự. Đoạn mã sau khai báo và in các phần tử của vecto.

```
// Gán vector với kích thước 10 phần tử, 0 là giá trị khởi tạo.
vector<int> v(10); //vector<int> v(10, 5); tương tự nhưng giá trị khởi tạo là 5.
for (int i = 0; i < v.size(); i++) {
   cout << v[i] << "\n";
}
//v.size() là hàm tính số phần tử hiện hành của vector v.
```

Xét ví dụ cho bài tập sau:

Bài tập 2: Đếm số lớn nhất

Viết chương trình tìm phần tử có giá trị lớn nhất trong dãy $A = \{a_1, a_2, \ldots, a_n\}$. Sau đó đếm xem có bao nhiêu phần tử bằng giá trị với phần tử vừa tìm thấy.

Input

- Dòng thứ nhất chứa số nguyên dương n là kích thước dãy thỏa $1 \leq n \leq 10^5$.
- n dòng tiếp theo chứa các số a_i của mảng thỏa $0 \leq a_i \leq 1000$.

Output

- In ra phần tử lớn nhất và số đếm cần tìm, hai số cách nhau ký tự trắng.

Samples

input	output
7 34 5 2 5 34 2 32	34 2

Tham khảo:

Bài tập OLP047, http://oj.hueuni.edu.vn/practice/problem/562/details

Lời giải.

Dữ Kiện về bài toán:

- Khai báo mảng động bằng vecto, thực ra khai báo mảng tĩnh cũng được đối với bài toán này.
- Nhập dữ liệu cho mảng, thể hiện bằng vòng lặp điều khiển chỉ số chạy từ 0 đến $n-1$ và quét dữ liệu bằng cin.
- Duyệt mảng bằng vòng lặp tìm giá trị max.
- Duyệt mảng lần nữa và đếm số phần tử bằng giá trị max.

Các bạn quan sát mã nguồn thể hiện như sau:

sol.cpp

```cpp
#include<bits/stdc++.h>
using namespace std;

vector<int> a(n)={};

int main()
{
   //đồng bộ hóa với stdio giải quyết thắt nút cổ chai, tăng tốc quét.
   ios_base::sync_with_stdio(0);
   cin.tie(0);cout.tie(0);
   //nhập kích thước vector.
   cin>>n;
   //nhập các phần tử của vector.
   for (int i = 0; i < n; i++)
      cin >> a[i];
   //tìm max.
   int max=a[0];
   for (int i=1; i<n ; i++)
      if(a[i]>max) max=a[i];
   //Đếm số phần tử bằng giá trị max.
   for (int i=0; i<n ; i++)
      if(a[i]==max) dem++;
   //In kết quả.
   cout << max << " " << dem;

   return 0;
}
```

Nhận xét:

- Lớp chứa vector là một cấu trúc lưu trữ, lớp chứa (container) của C++ STL do vậy nó hoạt động ổn định.
- Là một lớp chứa nên nó có nhiều phương thức hỗ trợ người dùng rất tốt. Bạn hãy tự tìm hiểu thêm các phương thức của nó. https://en.cppreference.com/w/cpp/container/vector
- Đi kèm với vector là thư viện <algorthm> với rất nhiều hàm hỗ trợ vector. bạn tìm hiểu ở https://en.cppreference.com/w/cpp/algorithm
- Trên phương diện lưu trữ bạn nên thao tác tương tự như mảng, vốn quen thuộc về mặt ngữ pháp và cách dùng.

Việc nghiên cứu và vận dụng linh hoạt vector dành cho các bạn yêu thích lập trình, cấu trúc này sẽ còn được dùng một cách tương tự ở nhiều ngôn ngữ lập trình khác và còn được dùng cho triển khai các ứng dụng lớn sau này.

Ví dụ: Tính tổng các phần tử trong một vector.

```cpp
//...
vector< int > v;
//...
int sum = accumulate(all(v), 0); //Tính tổng các phần tử trong vector v.
```

Ví dụ: Tính tích vô hướng của hai vector.

```
//...
vector< int > v1;
vector< int > v2;
//Input data v1={10, 9, 8}. v2={1, 2, 3}.
for(int i = 0; i < 3; i++) {
   v1.push_back(10-i);
   v2.push_back(i+1);
}
// r = v1[0]* v2[0] + v1[1]*v2[1]+v1[2]*v2[2] = 10 * 1 + 9 *2 + 8 *3 =52.
int r = inner_product(all(v1), v2.begin(), 0);
```

Và còn vô số các hàm khác. Mời các bạn tìm hiểu.

3 Mảng nhiều chiều

Trong các ứng dụng toán, cấu trúc ma trận thường xuyên gặp ở nhiều trong thực tế, có thể liệt kê gồm:

- Biểu diễn đồ thị trong lý thuyết đồ thị.
- Ma trận ngẫu nhiên được sử dụng để tìm xích Markov với những trạng thái hữu hạn.
- Tìm nghiệm của các hệ phương trình tuyến tính.
- Và vô số ứng dụng khác.

Tương tự mảng một chiều, mảng hai chiều cũng được tiếp cận theo hai cách, một là khai báo tĩnh, hai là khai báo động. Ưu nhược điểm cũng tương tự như đã phân tích trên.

3.1 Khai báo tĩnh

cú pháp:

```
<TYPE> array2DName[MAXCOL][MAXROW];
```

Trong đó:

- TYPE: Là các kiểu dữ liệu cơ bản, ví dụ như :int, long, double.
- array2DName: Tên biến mảng hai chiều.
- MAXCOL: Số phần tử tối đa tính theo dòng của mảng hai chiều cần khai báo, người ta còn gọi là chỉ số thứ nhất (1st subscript).
- MAXROW: Số phần tử tối đa tính theo cột của mảng hai chiều cần khai báo, người ta còn gọi là chỉ số thứ hai (2nd subscript).

Mảng hai chiều cũng là cấu trúc tuyến tính, các phần tử đều được đánh chỉ mục cho nên dễ dàng truy cập các phần tử với thời gian $O(1)$.

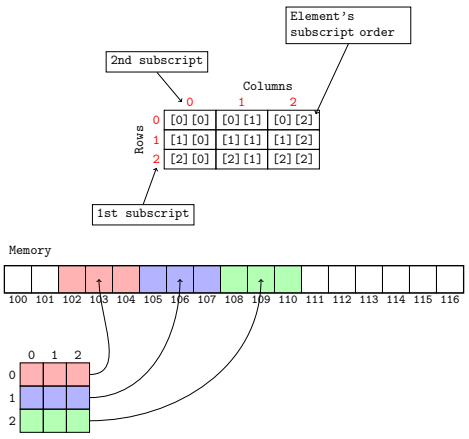
cú pháp:

```
array2DName[1st][2nd];
//1st là chỉ mục nhận giá trị từ 0 đến n-1 tính theo dòng.
//2nd là chỉ mục nhận giá trị từ 0 đến m-1 tính theo cột
//Với mảng hai chiều có cấu trúc n x m phần tử.
```

Xét ví dụ cho khai báo sau:

```
char A[3][3];
```

Hình ảnh sau cho ta thấy các chỉ số, thuộc tính và cấu trúc bộ nhớ của mảng hai chiều vừa khai báo.



Hình 2: Mảng hai chiều và chỉ số

Bài tập 3: Biến đổi ma trận

Cho ma trận $A_{m\times n}$ gồm các số 0 và 1. Hãy biến đổi ma trận trên theo quy tắc nếu có phần tử $a_{i,j}$ nào bằng 1 thì tất cả các phần tử trên cùng dòng và cột của nó đều bằng 1.

Ví dụ: $\begin{bmatrix}0 & 0 & 0\\ 0 & 0 & 1\end{bmatrix}$ sẽ biến đổi thành: $\begin{bmatrix}0 & 0 & 1\\ 1 & 1 & 1\end{bmatrix}$

Hãy viết chương trình thực hiện phép biến đổi trên.

Input

- Dòng đầu tiên chứa chứa số hai số nguyên m, n là kích thước của mảng thỏa $1 \le m, n \le 100$.
- m dòng tiếp theo mỗi dòng có n phần tử dòng của ma trận, các phần tử cách nhau ký tự trắng.

Output

- In ra ma trận sau khi biến đổi.

Samples

input

```
3 4
1 0 0 1
0 0 1 0
0 0 0 0
```

output

```
1 1 1 1
1 1 1 1
1 0 1 1
```

Tham khảo:

Bài tập PT064, http://oj.hueuni.edu.vn/practice/problem/336/details

Lời giải.

Dữ Kiện về bài toán:

- Khai báo mảng hai chiều tĩnh kích thước $n \times m$.
- Nhập dữ liệu cho mảng và biến đổi.

Các bạn quan sát mã nguồn thể hiện như sau:

sol.cpp

```cpp
#include <bits/stdc++.h>
using namespace std;
int m,n,a[101][101];

bool r[101],c[101];

int main()
{
   //Nhập kích thước mảng.
   cin>>n>>m;
   //Nhập mảng và biến đổi dòng, cột nếu a[i][j]=1 theo yêu cầu.
   for(int i = 0;i < n;i++)
      for(int j = 0;j < m;j++){
         cin>>a[i][j];
         if(a[i][j] == 1)
            r[i] = 1,c[j] = 1;
      }
   //In mảng.
   for(int i = 0;i < n;i++){
      for(int j = 0;j < m;j++)
         if(r[i] == 1 || c[j] == 1)
            cout<<1<<" ";
         else cout<<0<<" ";
      cout<<endl;
   }
   return 0;
}
```

Nhận xét:

- Khai báo đơn giản tương tự mảng 1 chiều và khai báo toàn cục.
- Bạn thử chuyển khai báo tĩnh và địa phương xem nếu chưa khởi tạo thì ảnh hưởng gì đến bài toán.
- Kích thước dòng và cột của ma trận sẽ hạn chế vì số phần tử của ma trận = số dòng x số cột.

Bài tập 4: Biến đổi công thức

Về nguyên tắc mảng hai chiều cũng lưu trữ liên tiếp nhau trong bộ nhớ. Như vậy thay vì khai báo mảng hai chiều với cấu trúc hàng cột thì nó có thể khai báo như mảng một chiều. Bài toán đặt ra cho bạn là tìm công thức truy cập phần tử A[i][j] bằng truy cập trên mảng một chiều?

Hãy lập trình nhập vào một mảng hai chiều, sau đó viết hàm tính tổng các phần tử của mảng hai chiều trên với tham số truyền vào là mảng một chiều.

Gợi ý: Hàm sau gán giá trị cho mảng một chiều với hai tham số là hàng và cột tương ứng với mảng hai chiều.

```cpp
int array[width * height];

int SetElement(int row, int col, int value){
   array[width * row + col] = value;
}
```

3.2 Khai báo động bằng vector

cú pháp:

```
vector< vector<TYPE> > v2D;
// Chú ý: có ký tự trắng sau, trước cặp dấu ngoặc nhọn đầu tiên.
```

Xét ví dụ cho đoạn thuật toán nhân hai ma trận $C_{n\times p} = A_{n\times m} \times B_{m\times p}$ sau:

```
typedef long long int lli;
typedef vector<lli> vi;
typedef vector<vi> vvi;

...
vvi mul(vvi v1, vvi v2, int n, int p, int m) {
   vvi r(n, vi(m, 0));
   for (int i = 0; i < n; i++) {
      for (int j = 0; j < m; j++) {
         for (int k = 0; k < p; k++) {
            r[i][j] = r[i][j] + v1[i][k] * v2[k][j];
         }
      }
   }
   return r;
}
```

4 Tìm kiếm và sắp xếp

Tìm kiếm và sắp xếp là hai giải thuật được sử dụng nhiều nhất trong lập trình, ta có thể tìm hiểu kỷ hơn trong khóa học về "Cấu trúc dữ liệu và giải thuật". Trong nội dung này chỉ đề cập đến các hàm cung cấp sẵn, giúp ta thực hiện các chức năng trên một cách thuận lợi.

C++ cung cấp các hàm sau:

- hàm sort() sắp dữ liệu với độ phức tạp $O(nlogn)$.
- hàm find() tìm kiếm tuần tự với độ phức tạp $O(n)$.
- hàm binary_search() tìm kiếm nhị phân với độ phức tạp $O(logn)$.
- các hàm trên đều thuộc thư viện <algorithm> của C++ STL. Các bạn tham khảo ở https://www.geeksforgeeks.org/c-magicians-stl-algorithms/

Xét ví dụ sau:

Bài tập 5: Liệt kê số

Một bài toán quen thuộc cần lập trình là sắp xếp dữ liệu. Bánh muốn Anh chị lập trình cho bài toán sắp xếp sau: Cho dãy số nguyên $a_1, a_2, \ldots, a_n$. Hãy sắp xếp lại dãy trên theo thứ tự số chẵn sắp trước, số lẻ sắp sau và tăng dần.

Thách thức là viết dưới dạng hàm để có thể truyền tham số như hàm qsort() của thư viện stdlib hay sort() của STL.

Input

- Dòng đầu tiên chứa số nguyên dương n kích thước của dãy thỏa $1 \leq n \leq 10^3$.
- Dòng tiếp theo chứa n số nguyên a_i thỏa $|a_i| \leq 10^3$, mỗi số cách nhau dấu cách.

Output

- In ra dãy sau khi sắp theo yêu cầu.

Samples

input	output
5 2 3 1 4 5	2 4 1 3 5

Tham khảo:

Bài tập PT061 http://oj.hueuni.edu.vn/practice/problem/536/details

Lời giải.

Dữ Kiện:

- Chỉ sắp thứ tự và in ra.
- Tuy nhiên việc sắp hơi phức tạp dựa vào một số điều kiện và hàm sort() có tham số hàm cho việc này.

Các bạn quan sát mã nguồn thể hiện như sau:

sol.cpp

```cpp
#include <bits/stdc++.h>
using namespace std;

//Hàm so sánh, chẵn ưu tiên trước, lẻ sau, và tăng dần.
bool sosanh(int a,int b){
   if((a%2==0 && b%2==0) || (a%2!=0 && b%2!=0))
      return a<b; //nếu cùng chẵn hoặc cùng lẻ thì sắp a trước.
   if(a%2==0)
      return 1; //a chẵn sắp trước.
   else
      return 0; //ngược lại b trước.
}


int main()
{
   int n;
   int a[1001];
   cin >> n;
   //Nhập dữ liệu cho mảng.
   for(int i=0;i<n;i++)
      cin >> a[i];
   //Gọi hàm sắp dữ liệu của thư viện <algorithm> gói trong stdc++.h.
   sort(a,a+n,sosanh);
   //In dữ liệu sau sắp.
   for(int i=0;i<n;i++)
      cout << a[i] << " ";

return 0;
}
```

5 Bài tập

Các bạn hãy thực hiện thực hành trên các bài toán sau: $PT037, PT038, \ldots, PT064$ trên mục http://oj.hueuni.edu.vn/practice/tags/ky-thuat-lap-trinh/13