Chương 4:
Con trỏ và mảng

Nguyễn Hồng Phương
Email: phuongnh-fit@mail.hut.edu.vn
Website: http://is.hut.edu.vn/~phuongnh
Bộ môn Hệ thống thông tin
Viện Công nghệ thông tin và Truyền thông
Đại học Bách Khoa Hà Nội

Nội dung chương này

Nội dung trình bày

4.1. Con trỏ và địa chỉ

- 4.1.1.Tổng quan về con trỏ
- 4.1.2. Các phép toán làm việc với con trỏ

4.1.1. Tổng quan về con trỏ

- a. Địa chỉ và giá trị của một biến
  - Bộ nhớ như một dãy các byte nhớ.
  - Các byte nhớ được xác định một cách duy nhất qua một *địa chỉ*.
  - Biến được lưu trong bộ nhớ.
  - Khi khai báo một biến
    - Chương trình dịch sẽ cấp phát cho biến đó một số ô nhớ liên tiếp đủ để chứa nội dung của biến. Ví dụ một biến số nguyên (int) được cấp phát 2 byte.
    - Địa chỉ của một biến chính là địa chỉ của byte đầu tiên trong số đó.

Địa chỉ và giá trị của một biến (tiếp)

- Một biến luôn có hai đặc tính:
  - Địa chỉ của biến.
  - Giá trị của biến.
- Ví dụ:
  - `int i, j;`
  - `i = 3;`
  - `j = i + 1;`

Biến	Địa chỉ	Giá trị
i	FFEC	3
j	FFEE	4

b. Khái niệm và khai báo con trỏ

- Con trỏ là một biến mà giá trị của nó là địa chỉ của một vùng nhớ.
- Khai báo con trỏ:
  - Cú pháp khai báo một con trỏ như sau:

```
Kieu_du_lieu *ten_bien_con_tro;
```

- Ví dụ
  - `int i = 3;`
  - `int *p;`
  - `p = &i;`

Biến	Địa chỉ	Giá trị
i	FFEC	3
p	FFEE	FFEC

- Một con trỏ chỉ có thể trỏ tới một đối tượng cùng kiểu.

Toán tử & và *

- Toán tử **&**: Trả về địa chỉ của biến.
- Toán tử *: Trả về giá trị chứa trong vùng nhớ được trỏ bởi con trỏ.
- Cả hai toán tử * và & có độ ưu tiên cao hơn tất cả các toán tử số học ngoại trừ toán tử đảo dấu.
- Ví dụ:

```
void main()
{
  int i = 3; int *p;
  p = &i;
  printf("*p = %d \n",*p);
  getch();
}
```

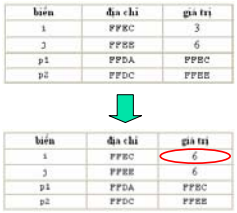
... p ... a ...

c. Sử dụng biến con trỏ:

- Một biến con trỏ có thể được gán bởi:
  - Địa chỉ của một biến khác:
    - `ten_bien_con_tro = &ten_bien;`
  - Giá trị của một con trỏ khác (tốt nhất là cùng kiểu):
    - `ten_bien_con_tro2 = ten_bien_con_tro1;`
  - Giá trị NULL (số 0):
    - `ten_bien_con_tro = 0;// hoặc là NULL`
- Gán giá trị cho biến con trỏ:
  - `*ten_bien_con_tro = 10;`

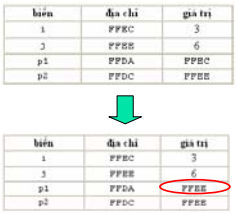
4.1.1. Tổng quan về con trỏ - ví dụ 1

```
main()
{
  int i = 3, j = 6;
  int *p1, *p2;
  p1 = &i;
  p2 = &j;
  *p1 = *p2;
}
```



4.1.1. Tổng quan về con trỏ - ví dụ 2

```
main()
{
  int i = 3, j = 6;
  int *p1, *p2;
  p1 = &i;
  p2 = &j;
  p1 = p2;
}
```



d. Con trỏ void

- `void *ten_bien_con_tro;`
- Con trỏ đặc biệt, không có kiểu,
- Có thể nhận giá trị là địa chỉ của một biến thuộc bất kỳ kiểu dữ liệu nào.
- Ví dụ:
  - `void *p, *q;`
  - `int x = 21;`
  - `float y = 34.34;`
  - `p = &x; q = &y;`

4.1.2. Các phép toán làm việc với con trỏ

- Cộng/trừ con trỏ với một số nguyên (`int, long`) → Kết quả là một con trỏ cùng kiểu
  - `ptr--;` //ptr trỏ đến vị trí của phần tử đứng trước nó.
- Trừ hai con trỏ cho nhau
  - Kết quả là một số nguyên
  - Kết quả này nói lên khoảng cách (số phần tử thuộc kiểu dữ liệu của con trỏ) ở giữa hai con trỏ.
- Các phép toán: Cộng, nhân, chia, lấy số dư trên con trỏ là không hợp lệ.
- Ví dụ: (p2 trỏ đến số nguyên nằm ngay sau x trong bộ nhớ)

```
int x, *p1, *p2;
p1= &x;
p2= p1+1;
```

13

Nội dung trình bày

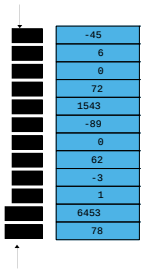
14

4.2. Mảng

15

4.2.1. Khái niệm mảng

- Mảng là một tập hợp **hữu hạn** các phần tử có **cùng kiểu dữ liệu** được lưu trữ **liên tiếp** nhau trong bộ nhớ.
- Các phần tử trong mảng có **cùng tên** (và cũng là tên mảng) nhưng phân biệt với nhau ở **chỉ số** cho biết vị trí của chúng trong mảng

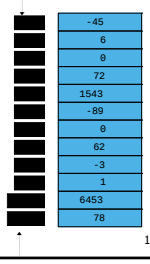


16

4.2.2. Khai báo và sử dụng mảng

a. Khai báo:

- Cú pháp:
  - `Kieu_du_lieu ten_mang[kich_thuoc_mang];`
- **Ví dụ:**
  - `char c[12];`
    - Phần tử đầu tiên có chỉ số `0`
    - n phần tử của mảng có tên `c`:
    - `c[ 0 ], c[ 1 ]...c[ n - 1 ]`
  - `int mang_nguyen[4];`
  - `float mang_thuc[6];`



17

4.2.2. Khai báo và sử dụng mảng (tiếp)

a. Khai báo (tiếp):

- Mảng nhiều chiều:
  - Mỗi phần tử của mảng cũng là một mảng khác → Giống vector trong toán học.
  - Ví dụ:
    - Mảng 2 chiều: `int a[6][5];`
    - Mảng 3 chiều: `int b[3][4][5];`

18

4.2.2. Khai báo và sử dụng mảng (tiếp)

- b. Sử dụng mảng:
  - Truy cập vào 1 phần tử của mảng thông qua tên mảng và chỉ số của phần tử đó.
  - Cú pháp: **`ten_mang[chi_so_cua_phan_tu]`**
  - Ví dụ 1: **`int mang_nguyen[3];`**
    - **`mang_nguyen[0]`**: Phần tử thứ 1.
    - **`mang_nguyen[1]`**: Phần tử thứ 2.
    - **`mang_nguyen[2]`**: Phần tử thứ 3.

19

4.2.2. Khai báo và sử dụng mảng (tiếp)

- b. Sử dụng mảng (tiếp):
  - Ví dụ 2: **`int a[6][5];`**
    - **`a[0]`** là phần tử đầu tiên của mảng, là 1 mảng
    → Phần tử đầu tiên của mảng **`a[0]`** là **`a[0][0]`**,…
    - …
    - **`a[2][3]`** sẽ là phần tử thứ 4 của phần tử thứ 3 của **`a`**.
    → **`a[i][j]`** sẽ là phần tử thứ **`j+1`** của **`a[i]`**, mà phần tử **`a[i]`** lại là phần tử thứ **`i+1`** của a.

20

4.2.3. Các thao tác cơ bản làm việc trên mảng

a. Nhập dữ liệu cho mảng:

- Nhập dữ liệu cho từng phần tử của mảng
- Ví dụ 1:

```
float a[10]; int i;
scanf("%f",&a[1]);
a[2] = a[1] + 5;
```

- Ví dụ 2:

```
int b[10];
int i;
//Nhap gia tri tu ban phim cho tat ca cac phan tu mang b
for(i = 0; i < 10; i++)
{
  printf("\n Nhap gia tri cho b[%d]", i);
  scanf("%d",&b[i]);
}
```

21

Nhập dữ liệu cho mảng (tiếp)

- Trường hợp không biết mảng sẽ có bao nhiêu phần tử mà chỉ biết số phần tử tối đa có thể có của mảng. Ví dụ:

```
int a[100];//Khai bao mang, so phan tu toi da la 100
int n;  // Bien luu giu so phan tu thuc su cua mang
int i;
printf("\n Cho biet so phan tu cua mang: ");
scanf("%d",&n);
for(i = 0; i < n; i++)
{
  printf("\n a[%d] = ", i);
  scanf("%d",&a[i]);
}
```

22

Nhập dữ liệu cho mảng (tiếp)

- Mảng có thể được khởi tạo giá trị ngay khi khai báo

```
int a[4] = {4, 9, 22, 16};
float b[3] = {40.5, 20.1, 100};
char c[5] = {'h', 'e', 'l', 'l', 'o'};
```

→ Câu lệnh thứ nhất có tác dụng tương đương với 4 lệnh gán:

```
a[0] = 4; a[1] = 9; a[2] = 22; a[3] =
  16;
```

23

b. Xuất dữ liệu chứa trong mảng

```
#include <stdio.h>
#include <conio.h>
#define KT 6
void main()
{
  int a[KT];
  int i, k;
  //Nhap gia tri cho cac phan tu mang a tu ban phim
  for(i = 0; i < KT; i++)
  {
   printf("\n a[%d] = ", i);
   scanf("%d", &a[i]);
  }
  printf("\nBat dau hien thi gia tri cac phan tu\n");
  printf("\n a[3] = %d", a[3]);
  //Hien thi gia tri tat ca cac phan tu
  //Moi phan tu tren 1 dong
  for(i = 0; i < KT; i++)
     printf("\n%d", a[i]);
```

24

b. Xuất dữ liệu chứa trong mảng

```
    printf("\n"); // Xuong dong moi
    // Hien thi gia tri cua tat ca cac phan tu mang a
    // tren 1 dong, cac phan tu cach nhau 1 dau tab
    for(i = 0; i < KT; i++)
        printf("%d\t", a[i]);
    // Hien thi k phan tu tren mot dong
    printf("\n Cho biet gia tri cua k = ");
    scanf("%d",&k);
    for(i = 0; i < KT; i++)
    {
      printf("%d  ", a[i]);
      if((i+1)%k == 0)//xuong dong khi da hien thi k ptu
          printf("\n");
    }
}
```


Kết quả

```
a[0] = -2
a[1] = 4
a[2] = 1
a[3] = -11
a[4] = 3
a[5] = 8
Bat dau hien thi gia tri cac phan tu
a[3] = -11
-2
4
1
-11
3
8
-2      4       1       -11     3       8
 Cho biet gia tri cua k = 2
-2  4
1  -11
3  8
```


c. Tìm các phần tử có giá trị lớn nhất, nhỏ nhất

- `#define KT 100`
- `int a[KT];`
- `max = a[0];`
- **Dùng vòng lặp for so sánh max với phần tử có chỉ số từ 1, 2,..., n-1 của mảng a.**

c. Tìm các phần tử có giá trị lớn nhất, nhỏ nhất

```
int a[100]; int i, n; int max;
printf("\n Cho biet so phan tu cua mang: ");
scanf("%d",&n);
for(i = 0; i < n; i++)
{
  printf("\n a[%d] = ",i);
  scanf("%d",&a[i]);
}
max = a[0];// Ban dau gia su phan tu lon nhat la a[0]
//Lan luot so sanh voi cac phan tu con lai trong mang
for(i = 1; i < n; i++)
  if(max < a[i]) // gap phan tu co gia tri lon hon
   max = a[i];// coi phan tu nay la phan tu lon nhat
printf("\n Phan tu lon nhat trong mang la: %d", max);
```


4.2.3. Các thao tác cơ bản làm việc trên mảng (tiếp)

- Tìm kiếm trên mảng
- Sắp xếp mảng

Nội dung trình bày

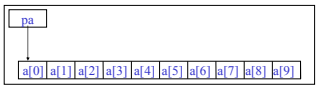
4.3.1. Con trỏ và mảng 1 chiều

- Xét câu lệnh:
  int a[10];
- Khai báo này hệ thống sẽ cấp cho mảng a 10 khối gồm 10 khối nhớ liên tiếp: a[0], a[1], a[2], a[3], ..., a[9](trong trường hợp này mỗi khối nhớ 2 byte).

a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]	a[7]	a[8]	a[9]

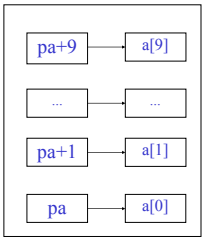
4.3.1. Con trỏ và mảng 1 chiều (tiếp)

- Có một điều cần nói ở đây là tên mảng là một hằng địa chỉ, nó chính là địa chỉ của phần tử đầu tiên của mảng; điều này có nghĩa là:
  a tương đương với &a[0]
- Nếu pa là một con trỏ kiểu nguyên,
  int *pa;
- Khi đó phép gán
  pa = &a[0];
- Đem pa trỏ đến phần tử thứ nhất (có chỉ số là 0) của a; nghĩa là pa chứa địa chỉ của a[0].



4.3.1. Con trỏ và mảng 1 chiều (tiếp)

- pa +i là địa chỉ của a[i] và *(pa+i) là nội dung của a[i]



4.3.1. Con trỏ và mảng 1 chiều (tiếp)

```
#define N 5
int tab[N] = {1, 2, 6, 0, 7};
main()
{
   int i;  int *p;
   p = tab;
   for (i = 0; i < N; i++)
   {
      printf(" %d \n",*p);
      p++;
   }
}
```


4.3.1. Con trỏ và mảng 1 chiều (tiếp)

- Tóm lại, một biểu thức chứa một mảng và một chỉ số tương đương với một cách viết khác sử dụng một con trỏ cùng một độ lệch.
- Có sự khác nhau giữa tên của mảng và con trỏ. Một con trỏ là một biến và vì thế, các câu lệnh pa=a và pa++ là hợp lệ. Nhưng một tên mảng không phải là một con trỏ, do đó các câu lệnh kiểu như a=pa;a++ là không hợp lệ. (*Về thực chất, tên mảng là một hằng con trỏ do đó chúng ta không thể thay đổi giá trị của nó được*).
- Để minh hoạ cho các ý tưởng vừa trình bày ta xét một ví dụ sau: viết chương trình thực hiện các công việc sau:
  - Đọc từ bàn phím các phần tử của một máng.
  - Tính tổng của chúng

4.3.1. Con trỏ và mảng 1 chiều (tiếp)

```
#include <stdio.h>
main()
 {
  float a[5],s;
  int i;
  for(i=0;i<5;i++)
        {
                printf("\na[%d]
= ",i) ;

        scanf("%f",&a[i]);
        }
        s=0;
  for(i=0;i<5;i++)
        s+=a[i];
  printf("\n tong la: %10.2f",s);
 }
```

```
#include <stdio.h>
main()
 {
  float a[5],s;
  int i;
  for(i=0;i<5;i++)
        {
                printf("\na[%d]
= ",i) ;
                scanf("%f",a+i);
        }
        s=0;
  for(i=0;i<5;i++)
        s+=a[i];
  printf("\n tong la:
%10.2f",s);
 }
```

4.3.1. Con trỏ và mảng 1 chiều (tiếp)

```
#include <stdio.h>
main()
 {
  float a[5],s, *p;
  int i; p=a;
  for(i=0;i<5;i++)
        {
                printf("\na[%d]
= ",i) ;

        scanf("%f",&p[i]);
        }
        s=0;
   for(i=0;i<5;i++)
        s+=p[i];
   printf("\n tong la: %10.2f",s);
 }
```

```
#include <stdio.h>
main()
 {
  float a[5],s,*p;
  int i; p=a;
  for(i=0;i<5;i++)
        {
                printf("\na[%d]
= ",i) ;
                scanf("%f",p+i);
        }
        s=0;
   for(i=0;i<5;i++)
        s+=*(p+i);
   printf("\n tong la: %10.2f",s);
 }
```


4.3.2. Con trỏ và mảng nhiều chiều

- Phép toán lấy địa chỉ
- Phép cộng địa chỉ trong mảng hai chiều
- Con trỏ và mảng hai chiều

4.3.2. Con trỏ.... - Phép toán lấy địa chỉ

- Phép toán lấy địa chỉ nói chung không dùng được đối với các thành phần của mảng nhiều chiều (trừ trường hợp mảng hai chiều các số nguyên).
- Xét chương trình sau với ý định nhập số liệu cho ma trận thực.

```
#include <stdio.h>
main()
   {
   float a[10][20];
   int i,j,n,m;
   printf("Nhap vao kich thuoc ma tran m,n=");
   scanf("%d%d",&m,&n); /*Chuong trinh chay dung cho den
   day*/
   for(i=0;i<m;i++)
    for(j=0;j<n;j++)
          {
          printf("a[%d][%d] = ",i,j);
          scanf("%f",&a[i][j]);
          }
   }
```


4.3.2. Con trỏ.... - Phép cộng địa chỉ trong mảng 2 chiều

- Xét khai báo
  float a[2][3];
- Với khai báo này hệ thống cấp sáu phần tử liên tiếp trong bộ nhớ theo thứ tự sau:
  a[0][0] a[0][1] a[0][2] a[1][0] a[1][1] a[1][2]
- Cũng như mảng một chiều tên của mảng hai chiều được hiểu như địa chỉ của phần tử đầu tiên của nó; phép cộng địa chỉ phải hiểu như sau: C cho rằng mảng hai chiều là mảng của mảng; như vậy với khai báo trên thì a là mảng mà mỗi phần tử của nó là một dãy gồm 3 số thực. Vì vậy
  - a trỏ tới đầu hàng thứ nhất (phần tử a[0][0]
  - a+1 trỏ tới đầu hàng thứ hai (phần tử a[1][0]

4.3.2. Con trỏ.... - Con trỏ và mảng 2 chiều

- Để truy xuất vào các phần tử của mảng hai chiều ta vẫn có thể dùng con trỏ theo cách sau:
  float *p, a[2][3];
  p=(float*)a;
- Khi đó
  p trỏ tới a[0][0]
  p+1 trỏ tới a[0][1]
  p +2 trỏ tới a[0][2]
  p +3 trỏ tới a[1][0]
  p +4 trỏ tới a[1][1]
  p +5 trỏ tới a[1][2]

Minh họa cách đọc dữ liệu cho mảng 2 chiều

```
#include <stdio.h>
main()
  {
  float a[2][3],*p;
  int i,j,m,n;
  p=(float*)a;
  for(i=0;i<2;i++)
        for(j=0;j<3;j++)
                {
                printf("a[%d][%d] = ",i,j);
                scanf("%f",p+i*3+j);
                }
                 for(i=0;i<2;i++)
        for(j=0;j<3;j++)
                {
                 printf("%6.2f",a[i][j]);
                   if(j==2) printf("\n");

                }
  }
```

Con trỏ và mảng 2 chiều

- Các bạn để ý rằng, a là một hằng con trỏ trỏ đến các dòng của một ma trân hai chiều, vì vậy
  a trỏ đến dòng thứ nhất
  a+1 trỏ đến dòng thứ hai
- Để tính toán được địa chỉ của phần tử ở dòng i cột j chúng ta phải dùng phép chuyển đổi kiểu bắt buộc đối với a: (float *)a, đây là con trỏ trỏ đến thành phần a[0][0] của ma trận. Và vì vậy thành phần a[i][j] sẽ có địa chỉ là (float *a) +i*n+j (n là số cột).
- Một cách tổng quát, nếu mảng có kiểu type và có kích thước các chiều tương ứng là n1,n2,..,nk (Giả sử mảng a có k chiều). Địa chỉ cuả thành phần a[0]..[0] (k chỉ số 0) là (type *)a, và địa chỉ của a[i1][i2]...[ik] được tính như sau

$$(type*)a + \sum_{j=1}^{k-1} i_j \prod_{l=j+1}^{k} n_l + i_k$$

4.3.3. Mảng các con trỏ

- Trước tiên chúng ta có thể khai báo một mảng các con trỏ bằng câu lệnh sau:
  `type *pointer_array[size];`
- ví dụ câu lệnh,
  `char *ma[10];`
  Sẽ khai báo một mảng 10 con trỏ char có thể được dùng để khai báo một mảng để lưu trữ địa chỉ của mười chuỗi ký tự nào đó.
- Nếu các con trỏ được chuẩn bị để chỉ đến một biến nào đó đã có, thì như vậy, chúng ta có thể truy xuất được các biến này thông qua một mảng mà không cần đến vị trí thực sự của các biến đó có liên tiếp hay không.

4.3.3. Mảng các con trỏ

- Ví dụ:
  - Xem xét một mảng các con trỏ ptr_array được gán các địa chỉ của các biến int có giá trị và vị trí bất kỳ.
  - Chúng ta sẽ dùng một hàm để sắp xếp lại các địa chỉ này trong mảng để sao cho các địa chỉ của các số bé được xếp trước địa chỉ của các số lớn hơn. Lúc đó dù chúng ta không làm thay đổi vị trí hoặc thay đổi các giá trị của các biến nhưng mảng vẫn có vẻ như là một mảng chỉ đến các giá trị đã sắp xếp có thứ tự.

4.3.3. Mảng các con trỏ - ví dụ (tiếp)

```
#include <stdio.h>
main()  {
  int i,j,*x;
  int d=10,e=3,f=7;
  int a=12,b=2,c=6;
  int *ptr_array[6];
  /*Gán các thành phần của mảng*/
  ptr_array[0]=&a;
  ptr_array[1]=&b;
  ptr_array[2]=&c;
  ptr_array[3]=&d;
  ptr_array[4]=&e;
  ptr_array[5]=&f;
```


4.3.3. Mảng các con trỏ - ví dụ (tiếp)

```
/*Sắp xếp lại dãy số*/
  for(i=0;i<5;i++)
   for(j=i+1;j<6;j++)
   if(*ptr_array[i]>*ptr_array[j])
        {
        x=ptr_array[i];
        ptr_array[i]=ptr_array[j];
        ptr_array[j]=x;
        }
  /*In kết quả sau khi sắp xếp*/
  for(i=0;i<6;i++)
   printf(" %d \n",*ptr_array[i]);
  getch();
  }
```


4.3.3. Mảng các con trỏ - ví dụ (tiếp)

- Kết quả chạy chương trình

2
3
6
7
10
12

4.3.3. Mảng các con trỏ - nhận xét

- việc sử dụng một mảng các con trỏ có nhiều ý niệm gần giống như sử dụng một mảng hai chiều. Ví dụ, nếu các biến n và m được khai báo là:

  int m[10][9];

  int *n[10];

  thì cách viết để truy xuất được các phần tử của các mảng này có thể tương tự nhau, chẳng hạn: m[6][5] và n[6][5] đều cho ta một kết quả là một số int.
- Khai báo:

  char slist[10][100];

  char *plist[10];

  đều khiến cho C hiểu rằng slist[i] và plist[i] là các con trỏ chỉ đến một đối tượng là char, hoặc là mảng char

4.3.3. Mảng các con trỏ - nhận xét

- Tuy vậy, giữa mảng nhiều chiều và mảng các con trỏ cũng tồn tại nhiều điểm khác nhau:
  - Mảng nhiều chiều thực sự là mảng có khai báo, do đó có chỗ đầy đủ cho tất cả các phần tử của nó. Còn mảng các con trỏ chỉ mới có chỗ cho các biến con trỏ mà thôi.
- Bên cạnh đó, việc sử dụng mảng các con trỏ có hai ưu điểm, đó là
  - Việc truy xuất đến các phần tử là truy xuất gián tiếp thông qua các con trỏ và như vậy, vị trí của các mảng con này có thể là bất kỳ, và chúng có thể là những mảng đã có bằng cách xin cấp phát chỗ động hay bằng khai báo biến mảng bình thường, tùy ý.
  - Các mảng con của nó được chỉ đến bởi các con trỏ, có thể có độ dài tùy ý, hoặc có thể không có (nếu con trỏ đó không được chuẩn bị, hoặc được gán bằng NULL).

4.3.4. Con trỏ trỏ đến con trỏ

- *Ví dụ*

```
char *monthname[20] =
  {
     "January","February","March","April",
     "May","June","July","August","September",
                "October","November","December"
  };
char **pp;
pp=monthname; /*Tên mảng là địa chỉ của phần tử đầu
  tiên*/
```

  Khi đó *pp sẽ chỉ đến con trỏ đầu tiên của mảng, con trỏ này lại chỉ đến chuỗi "January" như đã khởi đầu cho mảng.
- Nếu tăng pp lên 1 thì khi đó pp sẽ chỉ đến phần tử kế tiếp của mảng có giá trị là con trỏ đến xâu ký tự "February".v.v.

Bài tập

- Bài 1: Nhập vào một dãy số thực, sắp xếp tăng dần (sử dụng mảng một chiều).
- Bài 2: Nhập vào một dãy gồm 6 số nguyên, sắp xếp giảm dần (không sử dụng mảng).

Lời hay ý đẹp

"Con người ta có ba điều lầm lỗi dễ mắc phải là: chưa đến lượt đã vội nói, điều đáng nói lại không nói và không nhìn vẻ mặt người khác mà đã nói"

Khổng Tử