# Chương 5: Mảng

## 5.1 Đặt vấn đề

- Khi cần biểu diễn vài phần tử thì có thể dùng vài biến phân biệt (ví dụ để biểu diễn 3 hệ số của phương trình bậc 2 thì có thể dùng 3 biến a, b, c) nhưng khi số phần tử quá nhiều thì không thể khai báo kiểu liệt kê từng biến như vậy.

- Khi số phần tử hơi nhiều (chưa đến mức quá nhiều) và khai báo kiểu liệt kê từng biến được thì các thao tác trước mắt như nhập /xuất sẽ rất dài dòng bất tiện; các xử lý cơ bản như tìm kiếm, sắp xếp, ... sẽ là bất khả thi.

- Khi số lượng phần tử không thể biết trước là bao nhiêu thì hình thức biểu diễn chúng bằng cách liệt kê từng biến cũng không dùng được, và cho dù có được thì chương trình xử lý chúng cũng không thể viết được.

* Trên thực tế thì việc phải xử lý một danh sách nhiều phần tử (và số phần tử có thể không biết trước) là rất thường xuyên. Do đó cần phải có hình thức hỗ trợ để ta có thể biểu diễn và xử lý danh sách. Có 3 kiểu biểu diễn đáp ứng được việc này – đó là mảng, xâu và cây.

## 5.2 Khái niệm về Mảng (Array)

**- Là một kiểu dữ liệu có cấu trúc do người lập trình định nghĩa.**

**- Biểu diễn một dãy các biến có cùng kiểu. (dãy số, ký tự, …)**

**- Kích thước được xác định ngay khi khai báo & không thay đổi *(trừ trường hợp mảng động)***

- NNLT C luôn chỉ định một khối nhớ liên tục cho một biến kiểu mảng.

## 5.3 Cách khai báo Mảng

## * Cú pháp khai báo mảng một chiều:

**<kiểu phần tử> <tên biến mảng> [<số phần tử>];**

*- Phải xác định <số phần tử> cụ thể (hằng) khi khai báo.*

*- Chỉ số các phần tử đi từ 0 đến <tổng số phần tử>-1*

*- Bộ nhớ sử dụng = <tổng số phần tử>*sizeof(<kiểu phần tử>)*

*(Bộ nhớ sử dụng phải ít hơn 64KB )*

## Ví dụ:

```
int a [ 9 ];
int b[10];

a[9000] = 2;
b[0] = 3.4;
b[99] = a[0]*3;
```

khai báo tốt:

```
#define MAX 100
int fibo [MAX];
```

khai báo sai:

```
int N = 100;
int fibo [ N ];
```

## * Khai báo mảng đồng thời gán giá trị khởi tạo:

**<kiểu> <tên mảng> [<số phần tử>] = {<các trị khởi tạo>}**

*- Các trị khởi tạo được phân cách bởi dấu phẩy, và được gán tương ứng theo thứ tự từ phần tử 0 đến phần tử N-1 (N là số phần tử mảng)*

Vd: int a [ ];

*- Số giá trị khởi tạo có thể ít hơn số phần tử mảng, khi đó các phần tử 0 đến M-1 (M là số trị khởi tạo) sẽ được gán tương ứng.*

Vd: int a [5] = { 1, 7, 3 };  // a[0]=1; a[1]=7; a[2]=3;

*- Có thể không cần khai báo số phần tử mảng – khi này số phần tử của mảng sẽ bằng số giá trị khởi tạo.*

Vd: int a [ 5 ] = { 1, 3, 3, 2, 9 }; *// tương đương int a [5] = {1, 3, 3, 2,9};*

## * Cú pháp khai báo mảng nhiều chiều:

### - Khai báo mảng hai chiều:

**<kiểu> <tên mảng> [<số phtử chiều 1>][<số phtử chiều 2>];**

*- Giả sử N1 = <số phần tử chiều 1>, N2 = <số phần tử chiều 2>, mảng 2 chiều trên tương đương N1 mảng 1 chiều (mỗi mảng có N2 phần tử).*

### - Khai báo mảng k chiều:

**<kiểu> <tên mảng> [N1] [N2] .. [Nk];**

*- Mảng k chiều trên tương đương N1 mảng k-1 chiều. Tổng số phần tử lúc này là N1*N2*…*Nk. (Trên thực tế thường chỉ dùng mảng 1 chiều hoặc 2 chiều)*

## 5.4 Cách gán dữ liệu cho mảng và truyền mảng

### * Cú pháp truy xuất đến một phần tử

- Mảng 1 chiều:

**<tên mảng> [<chỉ số phần tử>]**

Ví dụ:

```
int a [5] = { 1, 7, 3 };
a[3] = 6;
a[4] = a[2] -3;
a[5] = 2; // SAI vì chỉ số hợp lệ chỉ từ 0 đến 4.
```

*- Chỉ số phần tử đi từ 0 đến <số phần tử>-1, nếu truy xuất các chỉ số khác thì chương trình có thể không bị báo lỗi cú pháp nhưng chạy khi đúng khi sai.*

- Mảng 2 chiều:

**<tên mảng> [<chỉ số dòng>] [<chỉ số cột>]**

Ví dụ:

```
int a [5] [9]; // ma trận 5 dòng 9 cột
a [3] [5] = 7;
a [4] [6] = 8;
a [5] [6] = 7; // SAI vì chỉ số dòng hợp lệ chỉ từ 0 đến 4.
```

*- Mảng 2 chiều có thể xem là 1 ma trận, trong đó chỉ số dòng là chỉ số chiều 1 và chỉ số cột là chỉ số chiều 2.*

## * Gán dữ liệu cho mảng

*- Ta phải gán từng phần tử một chứ không thể gán cùng lúc nhiều phần tử (ngoại trừ hình thức khởi tạo lúc khai báo mảng).*

Ví dụ:

```
int a [3] = {2, 6, 3};
int b [3];

b = a;
```

*// để gán mảng a vào mảng b không thể dùng lệnh b=a*
*// để gán mảng a vào mảng b phải dùng vòng lặp gán từng phần tử:*

```
for (int j=0; j<n; ++j)
    b [ n-1-j ] = a [ j ];
```

*- (Thật ra có thể dùng hàm copy bộ nhớ để gán cùng lúc nhiều phần tử nhưng đây là xử lý chuyên sâu và chưa nên dùng)*

## * Truyền mảng

Cú pháp khai báo:

*- Dạng 1:* **<kiểu phần tử> <tên mảng> [<số phần tử>]**

*- Dạng 2:* **<kiểu phần tử> <tên mảng> [ ]**

*- Dạng 3:* **<kiểu phần tử> * <tên mảng>**

Chức năng:

*- Cả 3 dạng trên là như nhau, và đều là dạng truyền tham biến cho hàm (tức mảng có thể thay đổi nội dung sau khi gọi hàm).*

*- Thực chất tham số mà hàm nhận được chính là địa chỉ của phần tử đầu tiên của mảng. Vì vậy khi truyền mảng cho hàm cần phải truyền thêm tham số số phần tử của mảng.*

## 5.5 Các thao tác trên mảng

Giả sử đã định nghĩa mảng như sau:

```
#define MAX 100
int a [ 100 ];
```

### 5.5.1 Nhập xuất mảng

* Nhập mảng

*Khi viết hàm nhập mảng, ngoài tham số hiển nhiên là mảng cần nhập, ta phải truyền thêm 1 tham số nữa – đó là số phần tử của mảng. Cả hai phải truyền dạng tham biến vì cần giữ lại giá trị sau khi gọi hàm.*

```
int NhapMang ( int a [ ] , int &n ) { // giá trị trả về =0 là không bị lỗi
    printf ("\n Ban co biet so phan tu? (C/K ");
    char ans = getchar();
    if (ans == 'C' || ans == 'c')
       NhapMang_TypeA (a, n);
    else
       NhapMang_TypeB (a, n);
}
```

```
int NhapMang_TypeA ( int a [ ] , int &n ) { // giá trị trả về =0 là không bị lỗi
    printf (“\n Cho biet so phan tu: “);
    scanf (“%d”, &n);
    if ( n<=0 || n>MAX ) // so phan tu khong hop le
      return -1;
    for (int j=0; j<n; j++) {
      printf (“\n Nhap phan tu a [%d]: “, j);
      scanf (“%d”, &a [ j ]);
    }
    return 0;
}

    or

void NhapMang_TypeB ( int a [ ] , int &n ) {
    n = 0;
    do {
      printf (“\n Nhap phan tu a [%d]: “, n);
      scanf (“%d”, & a [ n ]);
      n++;
      if (n==MAX) return;
      printf (“\n Ban co nhap them khong ?(C/K)“);
      char ans = getchar();
    }
    while (ans == 'C' || ans == 'c');
}
```

* Xuất mảng

*Tương tự như hàm nhập mảng, ngoài tham số hiển nhiên là mảng cần xuất, ta phải truyền thêm 1 tham số số phần tử của mảng.*

*Tham số số phần tử chỉ cần truyền dạng tham trị vì nó không bị thay đổi giá trị khi hàm chạy.*

```
int XuatMang ( int a [ ], int n ) {
      printf ("\n Cac phan tu vua nhap: ");
      for (int j=0; j<n; j+=2)
        printf (" %d  ", a [ j ]);
      for (int j=1; j<n; j+=2)
        printf (" %d  ", a [ j ]);

}
```

## 5.5.2 Tìm phần tử trong mảng

*- Để tìm phần tử có tính chất nào đó trong mảng, ta phải quét mảng (duyệt từng phần tử trong mảng bằng 1 vòng lặp – thường là **for**). Mỗi lần duyệt 1 phần tử ta lấy giá trị của phần tử để kiểm tra có thỏa tính chất cần xét.*

*- Hàm phải trả về chỉ số (vị trí) của phần tử chứ không nên trả về giá trị phần tử (từ chỉ số ta xác định được phần tử và giá trị của phần tử, còn từ giá trị phần tử thì không xác định được phần tử cần tìm).*

* Hàm tìm phần tử có giá trị bằng với X

```
// Trường hợp mảng a (n phần tử) có nhiều phần tử bằng
// X thì hàm sẽ trả về phần tử có chỉ số nhỏ nhất thỏa điều kiện
// Nếu không tìm được phần tử nào thỏa hàm sẽ trả về -1
int TimPhanTuBang ( int a [ ], int n, int X ) { // tim vi tri ptu bang X,
  for (int j=0; j<n; j++)
```

```
        if ( a [ j ] == X )
            return j;
   return -1;
}
```

* Hàm tìm phần tử có giá trị đáp ứng một điều kiện nào đó:

*- Nếu điều kiện đơn giản thì có thể đưa thẳng vào biểu thức điều kiện của lệnh **if** ở hàm trên.*

*- Trường hợp điều kiện phức tạp thì có thể làm riêng một hàm kiểm tra rồi gọi đến hàm kiểm tra trong biểu thức điều kiện của **if**.*

*- Cụ thể, hàm kiểm tra sẽ nhận 1 tham số (dạng tham trị) có kiểu là kiểu phần tử mảng và trả về kiểu bool (int). Ví dụ, giả sử điều kiện là tổng các chữ số phải bằng với tích thì hàm kiểm tra có thể viết như sau:*

```
// kiểm tra tổng các chữ số của X có bằng tích ko và trả về 1 nếu thỏa
int Test ( int X ) {
  int chuso, tong = 0, tich = 1;
  if (X<0) X= - X; // chuyen thanh so duong neu X am
  while ( X > 0 ) {
     chuso = X % 10;
     tong += chuso;
     tich *= chuso;
     X /= 10;
  }
  return ( tong == tich );
}
// Trong trường hợp có nhiều phần tử đáp ứng thì hàm sẽ
// trả về phần tử có chỉ số nhỏ nhất thỏa điều kiện
// Nếu không tìm được phần tử nào thỏa hàm sẽ trả về -1

int TimPhanTuThoa ( int a [ ], int n ) {
   for (int j=0; j<n; j++)
```

```
        if ( Test ( a [ j ] )  )  return j;
   return -1;
}
```

* Hàm tìm phần tử có giá trị nhỏ nhất:

*- Trường hợp này phức tạp hơn so với các trường hợp trên. Đầu tiên ta giả định giá trị nhỏ nhất* ***min*** *=* $a_0$*, sau đó duyệt từng phần tử từ* $a_1$ *trở đi - nếu có tồn tại phần tử nào có giá trị nhỏ hơn* ***min*** *thì cập nhật lại* ***min*** *là giá trị đó. Như vậy khi duyệt hết mảng thì* ***min*** *sẽ là giá trị nhỏ nhất.*

*- Làm như trên thì ta mới xác định được giá trị nhỏ nhất chứ chưa xác định được chỉ số phần tử. Để xác định được phần tử ta cần có biến chỉ_số_min và cần cập nhật lại chỉ_số_min mỗi khi cập nhật giá trị min.*

```
// Trong trường hợp có nhiều phần tử có cùng giá  trị
// nhỏ nhất thì hàm trả về phần tử có chỉ số nhỏ nhất thỏa
int TimMin ( int a [ ], int n ) {
   int chi_so_min = 0;
   int min = a [chi_so_min] ;
   for (int j=1; j<n; j++)
      if ( a [ j ] < min ) {
         min = a [ j ];
         chi_so_min = j;
      }
   return chi_so_min;
}
```

*- Hàm trên cũng có thể bỏ đi biến min và viết lại như sau:*

```
int TimMin ( int a [ ], int n ) {
   int chi_so_min = 0;

   for (int j=1; j<n; j++)
      if ( a [ j ] < a [chi_so_min ] )
         chi_so_min = j;
   return chi_so_min;
}
```

```
int TimMin ( int a [ ], int n ) {
   int min = a[0];

   for (int j=1; j<n; j++)
      if ( a [ j ] < min )
         min = a [ j ];
   return min;
}
```

* Hàm tìm phần tử có giá trị lớn nhất:

*- Tương tự, ta có biến chỉ_số_max lưu chỉ phần tử có giá trị lớn nhất đến vị trí hiện tại và mỗi lần duyệt đến vị trí mới nếu kiểm tra thấy giá trị tại đó lớn hơn thì cập nhật lại chỉ_số_max là vị trí đó.*

```
int TimMax ( int a [ ], int n ) {
   int chi_so_max = 0;

   for (int j=1; j<n; j++)
      if ( a [ j ] > a [chi_so_max ] )
         chi_so_max = j;
   return chi_so_max;
}
```

* Hàm tìm phần tử có giá trị gần bằng với X nhất:

*- Tương tự như hàm tìm phần tử nhỏ nhất, nhưng thay vì so sánh giá trị phần tử - ta sẽ so sánh với khoảng cách giữa X & giá trị phần tử.*

```
// Trong trường hợp có nhiều phần tử gần bằng X thì
// hàm sẽ trả về phần tử có chỉ số nhỏ nhất thỏa điều kiện

int TimPhanTuGanBang ( int a [ ], int n, int X ) {
   int kc_min, cs_min;

   cs_min = 0;   // chi so phan tu gan X nhat gia dinh la 0
   kc_min = abs (a[cs_min] - X); // kc_min la khoang cach tuong ung

   for (int j=1; j<n; j++)
      if ( abs (a[ j ] – X) < kc_min) {
         kc_min = abs (a[ j ] – X);
         cs_min = j;
      }
   return cs_min;
}
```

### 5.5.3 Kiểm tra tính chất của mảng

*- Để xác định mảng có tính chất nào đó không, ta cũng phải quét mảng bằng vòng lặp **for**. Mỗi lần duyệt qua 1 phần tử ta cũng lấy giá trị của phần tử ra để kiểm tra có thỏa tính chất cần xét.*

*- Giá trị trả về của hàm nên có kiểu int, trong đó mỗi giá trị tương ứng với 1 tính chất /trạng thái. Nếu chỉ có thể là 1 trong 2 trạng thái thì kiểu trả về có thể là bool.*

Ví dụ:

- Hàm xác định mảng chẵn (chứa toàn số chẵn) hay không

```
// hàm trả về TRUE nếu mảng chứa toàn số chẵn
// và FALSE nếu có tồn tại số lẻ

int MangChan ( int a [ ], int n ) {
   for (int j=0; j<n; j++)
        if ( a [ j ] %2 == 1 )  return FALSE;
   return TRUE;
}
```

- Hàm xác định mảng chẵn, lẻ hay không *(có cả số chẵn & số lẻ)*

```
// hàm trả về 0 nếu mảng a có chứa số chẵn và số lẻ,
// trả về 1 nếu mảng chứa toàn số lẻ,
// trả về 2 nếu mảng chứa toàn số chẵn

int KiemTraChanLe ( int a [ ], int n ) {
   int sosole = 0 ;

   for (int j=0; j<n; j++)
        if ( a [ j ] %2 == 1 )  sosole++;
   if (sosole==0) return 2;
   else if (sosole==n) return 1;
   return 0;
}
```

- Hàm kiểm tra mảng có thứ tự tăng (phần tử sau lớn hơn hoặc bằng phần tử trước) hay không

```
// hàm trả về TRUE nếu mảng a (n phần tử) tăng (FALSE nếu ko tăng)

int MangTang ( int a [ ], int n ) {
   for (int j=1; j<n; j++)
        if ( a [ j ] < a [ j-1] )  return FALSE;
   return TRUE;
}
```

- Hàm xét tình trạng thứ tự của mảng

```
// hàm xác định tình trạng thứ tự của mảng a (n phần tử)
// Qui định giá trị trả về của hàm: 0: không có thứ tự / -1: giảm /
```

```
// 1: tăng / 2: các phần tử bằng nhau / 3: chỉ có 0 hoặc 1 phần tử
int XetTinhThuTu ( int a [ ], int n ) {
   if ( n <= 1) return 3; // mang chi co 0 hoac 1 phan tu
   for (int j=1; j<n; j++)
         if ( a [ j ] != a [ j-1] )  break;
   if ( j == n )  return 2;  // cac phan tu deu bang nhau
   for (int j=1; j<n; j++)
         if ( a [ j ] < a [ j-1] )  break;
   if ( j == n )  return 1;  // phan tu sau luon >= phan tu truoc
   for (int j=1; j<n; j++)
         if ( a [ j ] > a [ j-1] )  break;
   if ( j == n )  return -1;  // phan tu sau luon <= phan tu truoc
   return 0;
}
```

## 5.5.4 Các thao tác khiến số phần tử của mảng thay đổi

* Xóa một phần tử ra khỏi mảng

*Sau khi xóa một phần tử thì số phần tử của mảng sẽ giảm đi 1, và các phần tử ở sau phần tử bị xóa sẽ dịch chuyển lên trước 1 vị trí – tức ta phải làm thao tác dồn mảng.*

*Ví dụ:*

*Mảng trước khi xóa phần tử ở vị trí 4 (mang giá trị 7):*

| **3** | **2** | **5** | **1** | **7** | **4** | **8** | **3** | **0** | **6** |
|---|---|---|---|---|---|---|---|---|---|
| *0* | *1* | *2* | *3* | *4* | *5* | *6* | *7* | *8* | *9* |

*Mảng sau khi xóa phần tử ở vị trí 4 (mang giá trị 7):*

| **3** | **2** | **5** | **1** | **4** | **8** | **3** | **0** | **6** |
|---|---|---|---|---|---|---|---|---|
| *0* | *1* | *2* | *3* | *4* | *5* | *6* | *7* | *8* |

// Hàm xóa phần tử ở vị trí k của mảng a (n phần tử)

```
void XoaPhanTu ( int a [ ], int &n, int k ) {

   for (int j=k; j<n-1; j++)
      a [ j ] = a [ j+1];  // don mang
   n --;
   return;
}
```

## * Thêm một phần tử vào mảng

*Sau khi thêm một phần tử vào mảng thì số phần tử của mảng sẽ tăng thêm 1, và các phần tử ở sau vị trí thêm sẽ dịch chuyển ra sau 1 vị trí – tức ta cũng phải làm thao tác dồn mảng.*

*Ví dụ:*

*Mảng trước khi thêm phần tử 9 vào vị trí 3:*

| **3** | **2** | **5** | **1** | **4** | **8** | **3** | **0** | **6** |
|---|---|---|---|---|---|---|---|---|
| *0* | *1* | *2* | *3* | *4* | *5* | *6* | *7* | *8* |

*Mảng sau khi thêm phần tử 9 vào vị trí 3:*

| **3** | **2** | **5** | **1** | **4** | **8** | **3** | **3** | **0** | **6** |
|---|---|---|---|---|---|---|---|---|---|
| *0* | *1* | *2* | *3* | *4* | *5* | *6* | *7* | *8* | *9* |

// Hàm thêm phần tử X vào vị trí k của mảng a (n phần tử)

```
void ThemPhanTu ( int a [ ], int &n, int X, int k ) {

   for (int j=n; j>k; --j)
      a [ j ] = a [ j-1 ];  // don mang
   a [ k ] = X;
   n ++;
   return;
}
```

## * Ghép nối hai mảng

```
// Hàm gộp 2 mảng a, b (có na & nb phtử) thành mảng c có nc phtử
// giá trị trả về của hàm =0: không gộp được / =1: gộp được

int GopMang ( int a [ ], int na, int b [ ], int nb, int c [ ], int &nc ) {
   if ( na + nb > MAX ) return 0;
   int j;

   for ( j=0; j<na; j++)
      c [ j ] = a [ j ];
   nc = na;

   for ( j=0; j<nb; j++)
      c [ nc++ ] = b [ j ];
   return 1;
}

// Hàm ghép mảng b có nb phần tử vào sau mảng a có na phần tử
// giá trị trả về của hàm =0: không ghép được / =1: ghép được

int GhepMang ( int a [ ], int &na, int b [ ], int nb ) {
   if ( na + nb > MAX ) return 0;

   for (int j=0; j<nb; j++)
      a [ na++ ] = b [ j ];
   return 1;
}

// Hàm chèn mảng b (nb phần tử) vào vị trí k của mảng a (na phần tử)
// giá trị trả về của hàm =0: không chèn được / =1: chèn được

int ChenMang ( SV a [ ], int &na, SV b [ ], int nb, int k ) {

   for (int j=n+nb-1; j>=k+nb; j--)
      a [ j ] = a [ j-nb ];// don mang

   for ( j=0; j<nb; j++) {
     a [ k+j ] = b [ j ];
```

```
   na += nb;
   return;
}
```

```
int ChenMang ( int a [ ], int &na, int b [ ], int nb, int k ) {
   if ( (na+nb>MAX) || (k>na) ) return 0;
   for (int j=0; j<nb; j++) {
      a [ k+nb+j ] = a [ k+j ];
   }
   na += nb;
   return 1;
}
```

* Tách mảng

```
// Hàm lấy các phần tử từ mảng a (na phần tử) thỏa điều kiện
// <Test (aj) = OK> đưa vào mảng b (nb phần tử)
void TaoMang ( int a [ ], int na, int b [ ], int &nb ) {
   nb = 0;
   for (int j=0; j<na; j++)
      if ( Test ( a [ j ] ) == OK )
        b [ nb++ ] = a [ j ];
   }
}
```

```
// Hàm tách mảng a (na phtử) thành 2 mảng b, c (có nb, nc phtử)
// với aj thỏa <Test(aj)=OK> thì đưa vào b (ngược lại đưa vào c)
void TachMang ( int a [ ], int na, int b [ ], int &nb, int c [ ], int &nc )
{
   nb = nc = 0;
   for (int j=0; j<na; j++)
```

```
      if ( Test ( a [ j ] ) == OK )
         b [ nb++ ] = a [ j ];
      else
         c [ nc++ ] = a [ j ];
}
```

## 5.5.5 Sắp xếp mảng

Sắp xếp lúc nhập mảng

* Nhập mảng

```
void NhapMang ( int a [ ], int &n ) {
      printf (“\n Ban co biet so phan tu ko? (C/K) “);
      char ans = getchar();

      if (ans=='C' || ans=='c')
      {
          printf (“\n Cho biet so phan tu: “);
          scanf (“%d”, &n);
          for (int j=0; j<n; j++) {
             printf (“\n Nhap phan tu a [%d]: “, j);
             scanf (“%d”, &X ]);
             int k = TimPhanTuLonHon ( a, j, X );
             ThemPhanTu (a, j, X, k );
          }
       }
       else
       { ... }
}
int TimPhanTuLonHon ( int a [ ], int n, int X ) {
   for (int j=0; j<n; j++)
         if ( a [ j ] > X )  return j;
   return n;
}
```

// Hàm thêm phần tử X vào vị trí k của mảng a (n phần tử)

```
void ThemPhanTu ( int a [ ], int n, int X, int k ) {

   for (int j=n; j>=k+1; j--)
      a [ j ] = a [ j-1 ];  // don mang
   a [ k ] = X;
   n ++;
   return;
}
```

Trộn 2 mảng được sắp

Sắp xếp lại 1 mảng không thứ tự

Hàm **qsort**

## Bài tập:

**Viết hàm xác định số phần tử chỉ chứa toàn các chữ số lẻ và số phần tử chỉ chứa toàn các chữ số chẵn trong 1 mảng nguyên.**

## Viết hàm:

1. Đếm số phần tử có 2 chữ số và số phần tử có 3 chữ số trong 1 mảng nguyên
2. Loại các phần tử trùng nhau ra khỏi mảng – chỉ giữ lại 1 phần tử.
3. Xóa tất cả các phần tử có giá trị lẻ trong 1 mảng nguyên

## 4. Thêm phần tử vào mảng có thứ tự (để mảng mới vẫn có thứ tự).

```
// hàm tra ve vi tri phan tu le dau tien trong mang a hoac -1 neu ko co
int TimPhanTuLe (int a [ ], int n) {
  for (int j=0; j<n; j++)
        if ( a [ j ] % 2 ) return j;
  return -1;
}
// hàm xóa tất cả các phần tử có giá trị lẻ trong 1 mảng nguyên
int XoaPhanTuLe (int a[ ], int &n) {
   while (1) {
        j = TimPhanTuLe (a, n);
        if ( j == -1) break;
        for (int k = j; k<n; k++)
              a[k] = a[k+1];
        n--;
    }
}
```