# BÀI 5
# KỸ THUẬT XỬ LÝ CHUỖI KÝ TỰ - CON TRỎ - TỆP TIN

## PHẦN I. CHUỖI KÝ TỰ

### I.1. Một số lưu ý khi sử dụng xâu ký tự

Chuỗi ký tự (hay xâu ký tự) là một dãy liên tiếp các ký tự. Như vậy, về bản chất xâu ký tự giống với một mảng một chiều mà mỗi phần tử của mảng là một ký tự.

Tuy nhiên, ngoài các đặc điểm như mảng, xâu ký tự còn có đặc thù riêng.

- **Khai báo biến kiểu xâu ký tự:** Có nhiều cách để khai báo biến xâu ký tự:
  - Khai báo như mảng các ký tự:

    **char <Tên biến xâu> [<số ký tự tối đa>];** **(1)**

  - Khai báo sử dụng kiểu string:

    **string <Tên biến xâu>;** **(2)**

Với cách khai báo (1), ta cần chỉ ra độ dài tối đa của xâu.

Với cách khai báo (2), độ dài xâu được xác định một cách tự động và biến xâu có thể chứa một lượng ký tự rất lớn.

**Ví dụ:** ta khai báo:

```
char S[30];
string T;
```

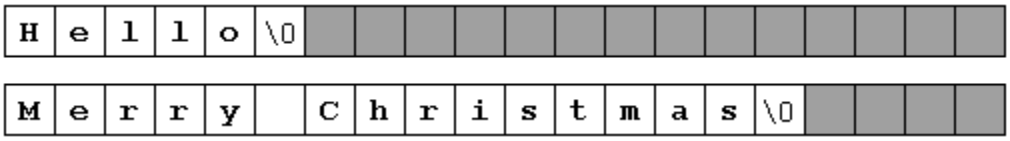
Ta thu được hai biến xâu. Biến xâu S chỉ chứa được các xâu có độ dài không quá 30 ký tự. Biến xâu T có thể chứa được các xâu ký tự có độ dài rất lớn.

**Chú ý:** Vì biến kiểu string có nhiều thao tác theo kiểu hướng đối tượng, nên trong phạm vi học phần, ta sử dụng biến xâu kiểu mảng ký tự.

Ký tự kết thúc xâu '\0' luôn được tự động thêm vào cuối xâu. Ví dụ ta có khai báo:

```
char S[20];
```

Biến S lúc này có thể chứa được tối đa 20 ký tự, kể cả ký tự kết thúc xâu. Giả sử biến S chứa xâu "Hello" hoặc xâu "Merry Christmas", nó được lưu trữ như sau:




- **Nhập xâu:**

Mặc dù xâu có bản chất giống mảng như việc nhập xâu không phức tạp như nhập mảng (không cần sử dụng vòng lặp). Thông thường người ta hay sử dụng lệnh gets cho việc nhập xâu:

**gets( <Biến xâu>);**

**Ví dụ:** để nhập xâu S, ta viết: **gets(S);**

Thông thường, ta hay bổ sung một thao tác xóa sạch bộ đệm bàn phím trước khi nhập xâu bằng lệnh gets() để tránh trường hợp lệnh gets() bị vô hiệu hóa nếu bộ đệm không được làm sạch (còn chứ ký tự enter). Lệnh xóa bộ đệm bàn phím: fflush(stdin);

**fflush(stdin); gets( <Biến xâu>);**

Ngoài ra, ta có thể sử dụng phương thức cin.getline() để nhập xâu. Cú pháp như sau:

**cin.getline(<Biến xâu>, <Số ký tự tối đa>);**

**Ví dụ:** lệnh **cin.getline(S, 30);** sẽ cho phép nhập tối đa 30 ký tự từ bàn phím, kể cả ký tự kết thúc xâu được tự động thêm vào.

**Xuất xâu:**

Tương tự nhập xâu, việc xuất xâu cũng trở nên rất đơn giản bằng cách sử dụng một trong các lệnh xuất puts, printf, cout.

Ví dụ: Xuất xâu S lên màn hình, ta viết: cout<<S; hoặc puts(S); ...

- **Duyệt xâu:**

Việc duyệt xâu cũng tương tự như duyệt mảng. Tuy nhiên, ta cần sử dụng hàm strlen(<Biến xâu>) trả về độ dài của xâu cần duyệt (hàm này có sẵn trong thư viện string.h). Giả sử duyệt xâu S, ta viết:

```
for(int i=0; i < strlen(S); i++)
{
        //Thăm ký tự S[i];
}
```

- **Phép gán xâu:**

Nếu a và b là hai biến kiểu số (int, float,...), ta dễ dàng gán a sang b và ngược lại bằng cách viết: a = b; hoặc b = a;

Tuy nhiên, nếu a và b là hai biến kiểu xâu ký tự, việc sử dụng phép gán trên là không hợp lệ. Để gán xâu b sang xâu a, ta phải sử dụng hàm copy xâu:

**strcpy(a, b);**

Một cách tổng quát, hàm copy xâu được viết như sau:

**strcpy(S1, S2);**

Trong đó, S1 là một biến xâu còn S2 có thể là một biến xâu hoặc một hằng xâu. Khi đó, xâu ký tự S2 sẽ được gán sang S1.

**Ví dụ:**

```
char S1[30], S2[30];
strcpy(S1, “Ha Noi”); //Gán “Ha Noi” vào S1
strcpy(S2, S1); //Gán S1 vào S2, S1 và S2 có cùng nội dung là Ha Noi
```

- **Phép so sánh xâu:**

Để so sánh 2 biến xâu, ta cũng không được phép sử dụng toán tử so sánh (==) mà sử dụng hàm so sánh xâu:

**strcmp(S1, S2);**

Hàm này sẽ trả về giá trị bằng 0 nếu hai xâu bằng nhau; giá trị dương nếu S1 > S2 và giá trị âm nếu S1 < S2.

**Ví dụ:**

```
char S1[30], S2[30];
gets(S1); fflush(stdin);
gets(S2);
if (strcmp(S1, S2)==0)
```

```
            cout<< " Xau S1 bang xau S2";
else        cout<< "Xau S1 khac xau S2";
```

Hai hàm strcpy() và strcmp() có sẵn trong thư viện string.h.

- **Lấy mã ASCII của một ký tự trong xâu:**

Một số bài toán ta cần lấy mã ASCII của các ký tự. Công việc này trong C++ được thực hiện một cách dễ dàng mà không cần sử dụng tới hàm lấy mã ASCII. Để lấy mã ASCII của một ký tự S[i], ta chỉ cần đặt toán tử ép kiểu (int) ngay trước ký tự đó:

**Ví dụ:** Với S là một xâu, đương nhiên S[i] là một ký tự của xâu. Hai câu lệnh sau sẽ cho kết quả khác nhau:

```
cout<<S[i];    (1)

cout<<(int) S[i];      (2)
```

Câu lệnh (1) in ra màn hình ký tự S[i], còn câu lệnh (2) chỉ in ra mã ASCII của ký tự đó.

- **Chuyển mã ASCII thành ký tự:**

Tương tự như trên, việc chuyển mã ASCII thành ký tự cũng được thực hiện dễ dàng bằng toán tử ép kiểu (char).

Giả sử muốn in ra màn hình ký tự có mã 65 (chữ A), ta chỉ cần viết:

```
cout<<(char) 65;
```

Khi đó ký tự 'A' sẽ được in ra màn hình do nó có mã ASCII là 65.

### III.2. Một số bài toán đặc thù trên xâu

Ngoài các bài toán tìm kiếm, sắp xếp như trên một mảng thông thường, các bài toán trên xâu còn được mở rộng do tính đặc thù của xâu. Sau đây là một số dạng phổ biến:

- **Thống kê trên xâu:** chẳng hạn thống kê số ký tự, số từ, số câu, số dấu chấm,….

- **Chuẩn hoá xâu:** Cắt các ký tự trống ở hai đầu xâu, xoá bớt dấu cách nếu có 2 dấu cách liền nhau trong thân xâu. Trước dấu chấm câu không có dấu cách, sau dấu chấm câu có 1 dấu cách; ký tự đầu câu viết hoa…

**Ví dụ 1:**

Nhập vào một xâu ký tự bất kỳ. Một từ trong xâu là một dãy liên tiếp, dài nhất các ký tự khác ký tự trống. Hãy cho biết xâu vừa nhập có bao nhiêu từ.

Dễ thấy với một xâu chưa chuẩn thì số từ không tỷ lệ thuận với số dấu cách. Do vậy việc đếm số dấu cách là không phù hợp.

Thay vào đó, ta đi đếm số lần bắt đầu của một từ, đó là số lần S[i] bằng dấu cách và S[i+1] khác dấu cách. Tuy nhiên, trong trường hợp S[0] khác dấu cách thì ta vẫn đếm thiếu 1 từ đầu tiên nên phải tăng biến đếm lên 1.

```
int main()
{
        char S[30];
        cout<<"S="; gets(S);
        int d=0;
        for(int i=0; i<strlen(S)-1; i++)
                if(S[i] == ' ' && S[i+1] != ' ') d++;

        if(S[0] != ' ') d++;
        cout<<"Xau co: "<<d<<" tu !";
}
```

**Ví dụ 2:** Nhập một xâu ký tự S bất kỳ. Hãy đếm số lần xuất hiện của từng chữ cái có trong S.

Nếu không phân biệt chữ hoa và chữ thường thì bảng mã ASCII có tổng cộng 26 chữ cái. Trường hợp tồi nhất là cả 26 chữ cái này đều xuất hiện trong S. Do vậy ta sử dụng 26 biến đếm (mảng d gồm 26 phần tử nguyên).

Nếu ta gặp một ký tự nào đó trong S thì biến đếm tương ứng của nó sẽ được tăng lên 1. Bảng sau chỉ ra sự tương ứng giữa biến đếm và ký tự:

| **Chữ cái hoa (mã ASCII)** | **Biến đếm tương ứng** |
|---|---|
| A (65) | d[0] = d[65-65] |
| B (66) | d[1] = d[66-65] |
| C (67) | d[2] = d[67-65] |
| … | … |
| **S[i] ((int) S[i])** | **d[(int) S[i]-65]** |

| **Chữ cái thường (mã ASCII)** | **Biến đếm tương ứng** |
|---|---|
| a (97) | d[0] = d[97- 97] |
| b (98) | d[1] = d[98-97] |
| c (99) | d[2] = d[99-97] |
| … | … |
| **S[i] ((int) S[i])** | **d[(int) S[i]-97]** |

Như vậy ta cần chia hai trường hợp:

Trường hợp thứ nhất: nếu chữ cái S[i] có mã ASCII nhỏ hơn 97 thì S[i] sẽ là chữ cái hoa. Mã ASCII của nó là k = (int) S[i]. Khi đó ta cần tăng biến đếm d[k-65].

Trường hợp thứ 2: nếu chữ cái S[i] có mã ASCII lớn hơn hoặc bằng 97 thì S[i] sẽ là chữ cái thường. Mã ASCII của nó là k = (int) S[i]. Khi đó ta cần tăng biến đếm d[k-97].

Kết thúc quá trình đếm, ta duyệt lại mảng d. Nếu thấy d[i] khác 0 thì đó chính là số lần xuất hiện của chữ cái (char) (i+65) hoặc (char) (i+97).

```
int main()
{
        char S[30];
        cout<<"S="; gets(S);
        int d[26];
        for(int i=0; i<26; i++) d[i]=0;

        for(int i=0; i<strlen(S); i++)
        {
                if(S[i] < 97)           d[S[i]-65]++;
               else                     d[S[i]-97]++;
        }
        for(int i=0; i<26; i++)
        if(d[i] > 0)
        cout<<"ky tu "<< (char)(i+97) <<" xuat hien "<<d[i]<<" lan !"<<endl;
}
```

# PHẦN II. CON TRỎ

## II.1 Tổng quan về con trỏ

### II.1.1. Khái niệm và cách khai báo

- Mỗi byte trong bộ nhớ đều được đánh địa chỉ, là một con số hệ thập lục phân. Địa chỉ của biến là địa chỉ của byte đầu tiên trong vùng nhớ dành cho biến.

Thông thường, khi ta khai báo một biến, máy tính sẽ cấp phát một ô nhớ với kích thước tương ứng trong Data Segment, là vùng nhớ dành để cấp phát cho biến. Ô nhớ này có thể dùng để lưu trữ các giá trị khác nhau, gọi là "giá trị của biến". Bên cạnh đó, mỗi biến còn có một địa chỉ là một con số hệ thập lục phân.

Địa chỉ của biến là địa chỉ của byte đầu tiên trong ô nhớ dành cho biến.

**Con trỏ (hay biến con trỏ) là một biến đặc biệt dùng để chứa địa chỉ của các biến khác, cùng kiểu với nó.**

Như vậy, con trỏ cũng giống như biến thường (tức cũng là một ô nhớ trong bộ nhớ) nhưng điểm khác biệt là nó không thể chứa các giá trị thông thường mà chỉ dùng để chứa địa chỉ của biến.

Con trỏ cũng có nhiều kiểu (nguyên, thực, ký tự…). **Con trỏ thuộc kiểu nào chỉ chứa địa chỉ của biến thuộc kiểu đó.**

**Cú pháp khai báo:**

**<Kiểu con trỏ> * <Tên con trỏ>;**

Trong đó: **<Kiểu con trỏ>** có thể là một trong các kiểu chuẩn của C++ hoặc kiểu tự định nghĩa. **<Tên con trỏ>** được đặt tuỳ ý theo quy ước đặt tên trong C++.

**Ví dụ:** Dòng khai báo `int a, *p; float b, *q;` khai báo a và p kiểu nguyên, b và q kiểu thực. Trong đó, p và q là hai con trỏ. Khi đó, p có thể chứa địa chỉ của a và q có thể chứa địa chỉ của b.

### II.1.2. Một số thao tác cơ bản trên con trỏ

- **Lấy địa chỉ của biến đặt vào con trỏ:**

Giả sử a là một biến nguyên và p là một con trỏ cùng kiểu với a. Để lấy địa chỉ của a đặt vào p ta viết:

**p = &a;**

Toán tử & cho phép lấy địa chỉ của một biến bất kỳ. Khi đó, ta nói p đang trỏ tới a. Một cách tổng quát, để lấy địa chỉ của một biến đặt vào con trỏ cùng kiểu, ta viết:

**<Tên con trỏ> = & <Tên biến>;**

- **Phép gán con trỏ cho con trỏ:**

Nếu p và q là hai con trỏ cùng kiểu, ta có thể gán p sang q và ngược lại, ta viết: p = q; hoặc q = p; Khi đó, địa chỉ đang chứa trong con trỏ ở vế phải sẽ được đặt vào con trỏ ở vế trái và ta nói hai con trỏ cùng trỏ tới một biến.

**Ví dụ:**

```
int a, *p, *q;
p = &a; //cho p trỏ tới a
q = p;  //p và q cùng trỏ tới a
```

- **Sử dụng con trỏ trong biểu thức:**

Khi sử dụng biến con trỏ trong biểu thức thì địa chỉ đang chứa trong con trỏ sẽ được sử dụng để tính toán giá trị của biểu thức.

Nếu muốn lấy *giá trị của biến mà con trỏ đang trỏ tới* để sử dụng trong biểu thức thì ta thêm dấu * vào đằng trước tên biến con trỏ.

**Ví dụ:**

```
int a=5, b=3, *p, *q;
p=&a;
q=&b;
int k = p + q;
int t = *(p) + *(q);
```

Khi đó, hai biến k và t có giá trị khác nhau. Trong biểu thức k, địa chỉ đang chứa trong con trỏ p và q sẽ được cộng lại và đặt vào k; ngược lại t sẽ có giá trị = a + b = 8.

**Chú ý:** *(p) chính là tên gọi khác của biến mà p đang trỏ tới. Như vậy, bằng cách sử dụng *(p), ta có thể truy cập và thao tác trên biến được trỏ bởi p, thậm chí có thể thay đổi giá trị trong biến được trỏ tới.

## II.2. Con trỏ và mảng

- **Tên mảng chính là con trỏ**

Khi khai báo mảng, ta được cấp phát một dãy các ô nhớ liên tiếp, cùng kiểu được gọi là các phần tử của mảng. Điều đặc biệt là tên mảng chính là một con trỏ trỏ tới phần tử đầu tiên của mảng. Như vậy tên mảng nắm giữ địa chỉ của ô nhớ đầu tiên trong mảng và do vậy, ta có thể sử dụng tên mảng để quản lý toàn bộ các phần tử của mảng theo cách của một con trỏ.

**Ví dụ:** giả sử ta khai báo: **int a[10];** Khi đó, 10 ô nhớ được cấp phát cho mảng a như sau:

Các phần tử lần lượt là a[0], a[1], a[2],….a[9]. Tuy nhiên, a là một ô nhớ riêng biệt và ô nhớ này đang chứa địa chỉ của a[0] (a trỏ tới a[0]):


Dễ thấy có sự tương ứng sau:

**a** **là địa chỉ của a[0]**
**a+1** **là địa chỉ của a[1]**
**a+2** **là địa chỉ của a[2]**
**…**
**a+i** **là địa chỉ của a[i]**

Như vậy thì

***a** **là a[0]**
***(a+1)** **là a[1]**
***(a+2)** **là a[2]**
**…**
***(a+i)** **là a[i]**

Vậy ta có thể sử dụng cách viết thứ hai cho các phần tử của mảng. Thay vì viết a[i], ta có thể viết *(a+i)

- **Con trỏ + Cấp phát bộ nhớ = Mảng**

Một con trỏ p bất kỳ, nếu ta cấp phát cho nó một dãy liên tiếp n ô nhớ, nó cũng tương đương với một mảng một chiều. Thật vậy, giả sử con trỏ p đang chứa địa chỉ của một ô nhớ đầu tiên trong dãy các ô nhớ được cấp. Khi đó ta có thể sử dụng p để quản lý toàn bộ dãy các ô nhớ đã được cấp.

Như vậy:

**p** **là địa chỉ của ô đầu tiên**

**p+1** **là địa chỉ của ô nhớ thứ 2**

**…**

**p+i** **là địa chỉ của ô nhớ thứ i**


Vậy, với p là con trỏ thì ta có thể coi nó như mảng một chiều và để truy cập tới phần tử thứ i trong vùng nhớ đã được cấp cho p ta có thể viết *(p+i) theo kiểu của con trỏ hoặc thậm chí viết p[i] theo kiểu của một mảng.

Để biết cách cấp phát bộ nhớ cho con trỏ, xin xem phần sau.

### II.3. Cấp phát và giải phóng bộ nhớ cho con trỏ

#### II.3.1. Cấp phát bộ nhớ động cho con trỏ

Khi ta khai báo một mảng theo cách thông thường (ví dụ int a[100];), việc cấp phát 100 ô nhớ cho mảng a được thực hiện tự động.

Nhưng nếu ta muốn "dành lấy" việc cấp phát, thu hồi bộ nhớ cho mảng thay vì làm tự động, ta gọi đó là cấp phát bộ nhớ động. Nói cách khác, cấp phát bộ nhớ động là việc cấp phát/ cấp phát lại/ thu hồi bộ nhớ cho con trỏ môt cách "thủ công" bởi lập trình viên. Việc này cho phép lập trình viên chủ động trong việc cân đối không gian nhớ.

Việc cấp phát bộ nhớ cho con trỏ sử dụng các hàm định vị bộ nhớ (allocation memory) của C++. Có rất nhiều hàm làm công việc này, tuy nhiên ta hay sử dụng toán tử **new.**

- **Toán tử new:**

  Trong C++ sử dụng toán tử cấp phát bộ nhớ new.

  Cấp phát một ô nhớ:

  **<Tên con trỏ> = new <Kiểu của con trỏ>;**

  Ví dụ: int *p = new int; hoặc float *q = new float;

  Cấp phát một khối nhớ (nhiều ô nhớ, memory block):

  **<Tên con trỏ> = new <Kiểu của con trỏ> [n];**

  Khi đó, con trỏ sẽ được cấp phát cho con trỏ n ô nhớ. Nó trở thành một mảng n phần tử. Cần lưu ý n là một hằng số nguyên hoặc một biến nguyên đang chứa một giá trị nguyên.

Nếu cấp phát thành công, con trỏ sẽ trỏ tới ô nhớ đầu tiên trong khối nhớ được cấp. Nếu cấp phát không thành công, con trỏ sẽ trỏ tới NULL (không trỏ tới đâu).

**Ví dụ:**

```
int *p,*q;
//cấp phát cho con trỏ p một ô nhớ kiểu int
p = new int;
//cấp phát cho con trỏ q 5 ô nhớ liên tiếp kiểu int, q trỏ tới ô đầu tiên.
q = new int[5];
```

Ngoài sử dụng toán tử new theo kiểu của C++, chúng ta cũng có thể sử dụng các hàm cấp phát bộ nhớ của C như calloc(), malloc(), realloc().

- **Hàm calloc:**

Cú pháp:

**<con trỏ> = (<Kiểu con trỏ>*) calloc(<n>, <size>);**

Trong đó, <n> là số ô nhớ cần cấp phát (số phần tử của mảng); <size> là kích thước của một ô nhớ.

Hàm calloc nếu thực hiện thành công sẽ cấp phát một vùng nhớ có kích thước <n>*<size> Byte và <con trỏ> sẽ trỏ tới ô nhớ đầu tiên của vùng nhớ này. Ngược lại, nếu thực hiện không thành công (do không đủ bộ nhớ hoặc <n> hoặc <size> không hợp lệ) hàm sẽ trả về giá trị NULL (tức con trỏ trỏ tới NULL).

Giả sử p là một mảng nguyên (trong hệ điều hành 16 bits), khi đó kích thước mỗi ô nhớ là 2 Byte (tức <size> = 2). Nếu p là mảng thực thì <size> = 4 .v.v…Toán tử sizeof sẽ cho ta biết kích thước của mỗi ô nhớ thuộc một kiểu bất kỳ. Muốn vậy ta chỉ cần viết: sizeof(<kiểu>). Ví dụ sizeof(int) = 2; sizeof(float) = 4; .v.v…

Hàm calloc thuộc thư viện alloc.h.

**Ví dụ 1:** Nhập một mảng p gồm n phần tử nguyên, sử dụng hàm calloc cấp phát bộ nhớ động.

```
int *p, n;
cout<< "Nhập n="; cin>>n;
p = (int *) calloc(n, sizeof(int));
if(p==NULL)
        cout<< "Cap phat bo nho khong thanh cong";
```

```
else //Nhap mang
for(int i=0; i<n; i++)
{
        cout<< "p["<<i<< "]=";
        cin>>p[i];
}
```

- **Hàm malloc:**

Tương tự như hàm calloc, hàm malloc sẽ cấp phát một vùng nhớ cho con trỏ. Cú pháp như sau:

**<Con trỏ> = (<Kiểu con trỏ>*) malloc(<size>);**

Trong đó <size> là kích thước của ô nhớ cần cấp phát tính bằng Byte. Chẳng hạn ta cần cấp phát bộ nhớ cho một mảng a gồm 10 phần tử nguyên. Khi đó kích thước vùng nhớ cần cấp phát (hệ điều hành 16 bits) = 10 * sizeof(int) = 10*2=20 Byte, ta viết:

**a = (int*) malloc(20);** hoặc **a = (int*) malloc(10 * sizeof(int));**

**Ví dụ 2:** Nhập một mảng p gồm n phần tử nguyên, sử dụng hàm malloc cấp phát bộ nhớ động.

```
int *p, n;
        cout<< "Nhập n="; cin>>n;
p = (int *) malloc(n*sizeof(int));
if(p==NULL)
        cout<< "Cap phat bo nho khong thanh cong";
else //Nhap mang
for(int i=0; i<n; i++)
{
        cout<< "p["<<i<< "]=";
        cin>>p[i];
}
```

**Ví dụ 3.** Nhập một mảng a gồm n phần tử thực bằng cách sử dụng con trỏ và cấp phát bộ nhớ động. Tìm phần tử lớn nhất và lớn thứ nhì trong mảng.

```
int main()
{
	float *a;int n;
	cout<<"n="; cin>>n;
	a = (float*) calloc(n, sizeof(float));
	if(a==NULL)
		cout<<"cap phat bo nho that bai!";
	else
	{
		for(int i=0; i<n; i++)
		{
			cout<<"a["<<i<<"]=";
			cin>>*(a+i);
		}

		int Max1, Max2;
		//Tim phan tu lon nhat chua vao Max1
		Max1=a[0];
		for(i=0; i<n; i++)
			if(Max1<*(a+i)) Max1=*(a+i);
		//Tim phan tu lon thu nhi chua vao Max2
		i=0;
		while(a[i]==Max1) i++;
		Max2=a[i];
		for(i=0; i<n; i++)
			if(Max2<*(a+i) && *(a+i) !=Max1) Max2=*(a+i);
		//In ket qua ra man hinh
		cout<<"Phan tu lon nhat = "<<Max1<<endl;
		cout<<"Phan tu lon thu nhi = "<<Max2<<endl;
	}
}
```

Giải thuật trên sẽ không cho kết quả đúng khi tất cả các phần tử của mảng bằng nhau (không tồn tại số lớn thứ nhì). Ta có thể khắc phục điều đó bằng cách kiểm tra trước trường hợp này.

### III.2. Cấp phát lại hoặc giải phóng bộ nhớ cho con trỏ

Đôi khi, trong quá trình hoạt động, kích thước của mảng lại thay đổi. Nếu ta sử dụng cấp phát bộ nhớ động và muốn kích thước của mảng "vừa đủ dùng" thì khi kích thước này tăng hoặc giảm (khi chương trình thực thi mới phát sinh điều này) ta nên cấp phát lại bộ nhớ cho con trỏ.

Để làm điều đó, ta sử dụng hàm realloc. Hàm này có nhiệm vụ cấp phát một vùng nhớ với kích thước mới cho mảng (con trỏ) nhưng vẫn giữ nguyên các giá trị vốn có của mảng.

**Cú pháp:**

**<Con trỏ> = (<Kiểu con trỏ>*) realloc(<Con trỏ>, <Kích thước mới>);**

Trong đó, <Kích thước mới> được tính bằng Byte.

**Ví dụ:** Nhập vào một mảng a gồm n phần tử nguyên. Hãy sao chép các giá trị chẵn của mảng đặt vào cuối mảng.

Giả sử ta có mảng a ban đầu gồm các phần tử như sau:

| 1 | 4 | 3 | 2 | 6 | 5 |
|---|---|---|---|---|---|

Sau khi sao chép các phần tử chẵn đặt vào cuối mảng, mảng a có dạng:

| 1 | 4 | 3 | 2 | 6 | 5 | 4 | 2 | 6 |
|---|---|---|---|---|---|---|---|---|

Rõ ràng kích thước của mảng a bị thay đổi (tăng lên). Mỗi khi có một phần tử chẵn được đặt vào cuối mảng thì kích thước của mảng được tăng lên 1. Do đó cần cấp phát lại bộ nhớ cho a với kích thước tăng thêm 1 ô nhớ (2 Byte).

```
int main()
{
	int *a;int n;
	cout<<"n="; cin>>n;
	a = (int*) calloc(n, sizeof(int));
	if(a==NULL)
		cout<<"cap phat bo nho that bai!";
	else
	{
		for(int i=0; i<n; i++)
		{
			cout<<"a["<<i<<"]=";
```

```
                cin>>*(a+i);
            }
            int m=n;
            for(int i=0; i<m; i++)
            if(a[i]%2==0)
            {
                a = (int*) realloc(a,(n+1)*sizeof(int));
                a[n]=a[i];      n++;
            }
        }
        for(int i=0; i<n; i++)
            cout<<a[i]<<"  ";
}
```

- **Giải phóng bộ nhớ đang chiếm giữ bởi con trỏ**

Khi không sử dụng tới con trỏ nữa, nếu ta không giải phóng vùng nhớ đã cấp phát cho con trỏ thì hiển nhiên vùng nhớ này vẫn bị nó chiếm giữ và không thể cấp phát cho các con trỏ khác (nếu có). Đặc biệt trong các hàm có cấp phát bộ nhớ động cho con trỏ, khi mà việc gọi hàm xảy ra thường xuyên nhưng khi kết thúc hàm ta không giải phóng vùng nhớ đã cấp phát thì bộ nhớ sẽ bị chiếm dụng một cách nhanh chóng gây ra tình trạng gọi là "memory leak".

Giải phóng bộ nhớ đang chiếm giữ bởi con trỏ không làm thay đổi giá trị của con trỏ (là một địa chỉ). Nó chỉ đơn giản cho hệ thống quản lý bộ nhớ biết rằng bit bộ nhớ tại địa chỉ đó, địa chỉ đã được dự trữ trước đó, không còn được dự trữ nữa và có thể sử dụng lại.

- Giải phóng bộ nhớ đã cấp phát bằng toán tử **new**:

Trong C++ sử dụng toán tử delete với cú pháp:

**delete [] <Tên con trỏ>;**

**Ví dụ:** Giả sử con trỏ q đã được cấp phát bộ nhớ bằng toán tử new. Muốn giải phóng nó, ta viết:

```
delete[] q;
```

- Giải phóng bộ nhớ đã cấp phát bằng malloc, calloc:

**Cú pháp:**

**free(<Tên con trỏ muốn giải phóng>);**

**Ví dụ:** Giả sử con trỏ p đã được cấp phát bộ nhớ bởi malloc, calloc, hoặc được cấp phát lại bằng realloc. Muốn giải phóng nó, ta viết:

```
free(p);
```

## PHẦN III. TỆP TIN

Trong phần này, chung ta sẽ làm quen với các thao tác với tệp tin dạng văn bản trong C++. Để thực hiện các phương thức, ta sử dụng thư viện "fstream".

### III.1. Mở tệp tin

- **Mở tệp để ghi dữ liệu**

  **Bước 1: Khai báo dòng xuất tệp (biến tệp)**

  **ofstream** ⟨Tên_biến_tệp⟩;

  **Bước 2: Mở tệp để ghi dữ liệu vào tệp**

  ⟨Tên_biến_tệp⟩.**open**(⟨Tên_tệp⟩, ⟨Kiểu_mở⟩);

  Trong đó, ⟨Tên_biến_tệp⟩ do ta đặt và được gọi là một dòng xuất tệp. ⟨Tên_tệp⟩ là một chuỗi ký tự biểu thị tên của tệp, có thể bao gồm cả đường dẫn. ⟨Kiểu_mở⟩ thường sử dụng chỉ thị **ios::out** nếu muốn ghi đè (dữ liệu ghi vào tệp sẽ đè lên dữ liệu cũ của tệp nếu có), ngược lại, chỉ thị **ios::app** sẽ cho phép ghi bổ sung dữ liệu mới vào cuối tệp.

  Ví dụ ta khai báo một biến tệp *f* để ghi dữ liệu và sử dụng nó để mở tệp VANBAN.TXT trên ổ đĩa D, sử dụng chế độ ghi đè:

  ```
  ofstream f;
  f.open("D:/VANBAN.TXT", ios::out);
  ```

  Nếu tệp VANBAN.TXT chưa tồn tại trên ổ đĩa D, chương trình sẽ tạo ra một tệp như vậy để sẵn sàng cho việc ghi dữ liệu. Ngược lại, nếu tệp đã tồn tại, dữ liệu cũ sẽ bị ghi đè do ta sử dụng chỉ thị ios::out.

  Ta cũng có thể thực hiện gộp hai dòng lệnh trên thành một như sau:

  ```
  ofstream f("D:/VANBAN.TXT", ios::out);
  ```

- **Mở tệp để đọc dữ liệu**

  **Bước 1: Khai báo dòng nhập từ tệp (biến tệp)**

  **ifstream** ⟨Tên_biến_tệp⟩;

  **Bước 2: Mở tệp để đọc**

  ⟨Tên_biến_tệp⟩.**open**(⟨Tên_tệp⟩, ios::in);

Trong đó, ⟨Tên_biến_tệp⟩ do ta đặt và được gọi là một dòng nhập từ tệp. ⟨Tên_tệp⟩ là một chuỗi ký tự biểu thị tên của tệp, có thể bao gồm cả đường dẫn. Ví dụ để khai báo một biến tệp *f* và mở tệp "VANBAN.TXT" trên ổ đĩa D để đọc dữ liệu:

```
ifstream f;
f.open("D:/VANBAN.TXT", ios::in);
```

Ta cũng có thể thực hiện gộp hai dòng lệnh trên thành một như sau:

```
ifstream f("D:/VANBAN.TXT", ios::in);
```

Để sử dụng một biến tệp mà vừa có thể ghi dữ liệu vào tệp, vừa có thể đọc dữ liệu từ tệp, ta có thể sử dụng kiểu fstream thay cho ofstream hoặc ifstream. Ví dụ:

```
fstream f("D:/VANBAN.TXT", ios::in); hoặc
fstream f("D:/VANBAN.TXT", ios::out);
```

### III.2. Ghi dữ liệu vào tệp

C++ sử dụng cout như một dòng xuất dữ liệu gắn với màn hình. Nói cách khác, cout là dòng xuất gắn với thiết bị xuất là màn hình. Để xuất dữ liệu vào tệp, ta chỉ đơn giản là khai báo một dòng xuất tệp, mở một tệp và dùng nó thay cho cout. Ví dụ ta khai báo dòng xuất tệp có tên f và mở tệp "D:/vidu.txt" như sau:

```
ofstream f("D:/vidu.txt", ios::out);
```

Sau đó ta có thể sử dụng fout như đối tượng cout:

```
f<< "Hello"<<endl;
```

Tương tự cout, với dòng xuất tệp f ta cũng có thể sử dụng endl để xuống dòng, sử dụng setw(n), setprecision(n)... để định dạng dữ liệu xuất... Kết thúc việc xuất dữ liệu, ta cần đóng tệp lại bằng phương thức close(), ví dụ: f.close();.

**Ví dụ 1:** Nhập vào một mảng *a* gồm *n* phần tử thực, xuất dữ liệu trong mảng *a* vào tệp "Mangthuc.txt" với định dạng:

- Dòng đầu tiên: Ghi số *n*.
- Dòng thứ 2: Ghi các phần tử của mảng, mỗi phần tử cách nhau bởi dấu cách.

Hàm Ghidulieu() sau sẽ thực hiện khai báo một dòng xuất tệp fout với tên tệp đang chứa trong đối tentep để ghi dữ liệu theo chế độ ghi đè (dòng 3), sau đó xuất dữ liệu vào tệp (dòng 4, 5, 6) và không quên đóng tệp (dòng 7).

```
1   void Ghidulieu(int*a, int n, char* tentep)
2   {
3      ofstream fout(tentep, ios::out);
4      fout<<n<<endl;
5      for(int i=0; i<n; i++)
6         fout<<a[i]<<" ";
7      fout.close();
8   }
9   int main()
10  {
11     int n;
12     cout<<"n="; cin>>n;
13     float *a = new float[n];
14     for(int i=0; i<n; i++)
15        cin>>a[i];
16     Ghidulieu(a, n, “D:/Mangthuc.txt”);
17  }
```

### III.3. Đọc dữ liệu từ tệp

Việc đọc dữ liệu từ tệp sẽ có chút phức tạp hơn là việc ghi dữ liệu vào tệp. Ta có hai kiểu đọc dữ liệu: Đọc từng dòng và đọc từng cụm. Hơn nữa, ta cần sử dụng một số thao tác bổ trợ cho việc đọc này.

- **Đọc từng dòng trong tệp**

Khi đọc từng dòng trong tệp, mỗi dòng được xem là một chuỗi ký tự. Do vậy, nếu tệp chứa dữ liệu kiểu số, cách đọc này sẽ có thể không phù hợp và cần có thao tác tách từng phần tử số trong dòng đọc được, chuyển đổi kiểu dữ liệu,... Do vậy, cách này thường áp dụng để đọc tệp mà nội dung tệp là thuần văn bản.

Lệnh getline sau đây cho phép đọc từng dòng dữ liệu trong tệp đang được mở bởi dòng đọc tệp f ra một biến kiểu chuỗi ký tự str:

f.**getline**(str, ⟨p⟩);

Trong đó, ⟨p⟩ là một số nguyên đủ lớn, là số ký tự tối đa sẽ đọc. Để đọc toàn bộ nội dung tệp, ta sử dụng phương thức eof() để kiểm tra xem đã đọc hết tệp hay chưa f.eof() sẽ trả về giá trị true nếu ta đã đọc hết dữ liệu trong tệp f (con trỏ tệp trỏ tới cuối tệp), ngược lại nó trả về giá trị false. Hàm ReadLineFile() sau đây sẽ đọc toàn bộ nội dung tệp với tên tệp chứa trong đối số tentep và in chúng ra màn hình:

```
1   void ReadLineFile(char* tentep)
2   {
3      ifstream f(tentep, ios::in);
4      char str[255];
5      while(!f.eof())
6      {
7         f.getline(str, 255);
8         cout<<str<<endl;
9      }
10     f.close();
11  }
```

- **Đọc từng cụm trong tệp**

Với dữ liệu kiểu số (ví dụ một mảng số), việc đọc từng dòng dữ liệu trong tệp sẽ có thể gây khó khăn khi phải xử lý tách các phần tử trong dòng đó. Trong trường hợp này, ta sử dụng kỹ thuật đọc từng cụm.

Một cụm ký tự được coi là tập các ký tự liên tiếp ngăn cách với cụm khác bởi một hoặc nhiều dấu cách.

Toán tử nhập (>>) sẽ thực hiện việc đọc từng cụm dữ liệu trong tệp (mỗi lần đọc một cụm). Câu lệnh sau sẽ đọc 1 cụm ký tự trong tệp đang được mở bởi dòng đọc tệp f và chứa vào một biến *a*:

f>>a;

Khi đó biến a có thể là kiểu chuỗi ký tự, có thể là kiểu số. Nếu a là biến kiểu số, cần đảm bảo dữ liệu trong tệp cũng có dạng số.

Ta cũng có thể đọc nhiều cụm liên tiếp bằng toán tử nhập. Câu lệnh sau đọc ba cụm liên tiếp của tệp f và chứa vào ba biến nguyên *a*, *b*, *c*:

```
f>>a>>b>>c;
```

Thông thường, biến a, b, c có kiểu chuỗi ký tự. Tuy nhiên, tùy theo dữ liệu có kiểu gì, ta có thể sử dụng biến a, b, c có kiểu đó mà không cần thực hiện thao tác chuyển đổi kiểu.

**Ví dụ 2.** Cho một tệp "MANGSO.txt" trong ổ đĩa D. Dòng đầu tiên chứa số phần tử của một mảng số nguyên. Dòng tiếp theo chứa các phần tử của mảng với mỗi phần tử cách nhau bởi dấu cách, ví dụ như sau:

```
5  12  24  28   11  17
```

Hãy đọc dữ liệu của tệp lên một biến mảng *a*, in mảng đọc được ra màn hình.

Hàm ReadFile sau trả về giá trị true nếu đọc tệp thành công, ngược lại trả về false. Hàm sử dụng phương thức **good**() để kiểm tra việc tệp có tồn tại hay không. Nếu tệp tồn tại, **good**() trả về giá trị **true**, ngược lại, nó trả về giá trị **false**.

```
1   bool ReadFile(char* tentep, int*&a, int&n)
2   {
3      ifstream f(tentep, ios::in);
4      if(f.good())
5      {
6         f>>n;
7         a = new int[n];
8         for(int i=0; i<n; i++)
9            f>>a[i];
10        f.close();
11        return true;
12     }
```

```
13      else
14      {
15          cout<<"File not found !";
16          return false;
17      }
18  }
19  int main()
20  {
21      int*a; int n;
22      bool fExists = ReadFile("mang.txt",a, n);
23      if(fExists)
24      {
25          cout<<n<<endl;
26          for(int i=0; i<n; i++)
27              cout<<a[i]<<" ";
28      }
29  }
```