Thực tập KỸ THUẬT LẬP TRÌNH

Tuần 7-9: Thực hiện các chức năng sắp xếp

Yêu cầu:

Với dữ liệu sinh viên gồm các thông tin:

- Mã lớp
- Mã sinh viên
- Họ và tên
- Ngày sinh
- Điểm trung bình tích lũy

đã nhập và lưu trữ trên file.

Ví dụ đã nhập được danh sách sinh viên như sau:

Mã lớp	Mã sinh viên	Họ và tên	Ngày sinh	ĐTBTL
13A	2014000001	Nguyễn Đình Giang	02/12/1996	3.4
13C	2014000002	Trịnh Văn Anh	24/10/1996	2.8
13B	2014000003	Hoàng Xuân Đạt	15/06/1996	3.2

Hãy thực hiện các các yêu cầu sau:

- Sắp xếp danh sách sinh viên theo Mã lớp. Ví dụ danh sách trên sau khi xếp theo mã lớp ta được:

Mã lớp	Mã sinh viên	Họ và tên	Ngày sinh	ĐTBTL
13A	2014000001	Nguyễn Đình Giang	02/12/1996	3.4
13B	2014000003	Hoàng Xuân Đạt	15/06/1996	3.2
13C	2014000002	Trịnh Văn Anh	24/10/1996	2.8

- Sắp xếp danh sách sinh viên theo Mã sinh viên (như sanh sách đã cho).

- Sắp xếp danh sách sinh viên theo Họ và tên. Họ và tên sinh viên được so sánh theo tên rồi đến họ đệm. Ví dụ danh sách sinh viên ở trên đặc xếp lại theo họ tên là

Mã lớp	Mã sinh viên	Họ và tên	Ngày sinh	ĐTBTL
13C	2014000002	Trịnh Văn Anh	24/10/1996	2.8
13B	2014000003	Hoàng Xuân Đạt	15/06/1996	3.2
13A	2014000001	Nguyễn Đình Giang	02/12/1996	3.4

- Sắp xếp danh sách sinh viên theo Ngày sinh. Ngày sinh được so sánh theo năm, đến tháng, và cuối là đến ngày. Ví dụ danh sách trên sau khi sắp xếp ta được:

Mã lớp	Mã sinh viên	Họ và tên	Ngày sinh	ĐTBTL
13B	2014000003	Hoàng Xuân Đạt	15/06/1996	3.2
13C	2014000002	Trịnh Văn Anh	24/10/1996	2.8
13A	2014000001	Nguyễn Đình Giang	02/12/1996	3.4

- Sắp xếp danh sách sinh viên theo mã lớp, cùng mã lớp sắp xếp theo họ tên, cùng họ tên thì sắp xếp theo ngày sinh.

Kiến thức liên quan:

A. MỘT SỐ THUẬT TOÁN SẮP XẾP

- Bài toán sắp xếp thường được mô tả như sau:
- Cho danh sách *X* chứa *n* phần tử *X[1]*, *X[2]*, …, *X[n]*. Giả thiết là các phần tử của danh sách *X* có thể so sánh hơn kém với nhau theo một tiêu chí nào đó. Theo tiêu chí được chọn trước, cần sắp xếp lại mảng *X* theo thứ tự không giảm (hoặc không tăng).
- Trong mục này chúng ta xét trường hợp các phần tử của mảng *X* là số thực và yêu cầu sắp xếp *X* sao cho thành dãy không giảm. Có khá nhiều thuật toán sắp xếp để giải quyết bài toán đã nêu. Trong phần tiếp theo chúng ta xem xét một số thuật toán sắp xếp đơn giản như sắp xếp chọn (selection sort) , sắp xếp chèn (insertion sort) và sắp xếp nổi bọt (buble sort).

5.2.1. Sắp xếp chọn

Ý tưởng:

- Xét từng vị trí, từ *1* đến *n-1*, tại mỗi vị trí tìm phần tử thích hợp và chuyển đến. Như vậy, phần tử thích hợp với vị trí thứ nhất phải là phần tử bé nhất trong danh sách (với bài toán sắp xếp thành dãy không tăng thì phần tử đầu tiên phải là phần tử lớn nhất trong dãy). Ở vị trí thứ hai phải là phần tử bé nhất trong các phần tử còn lại (với bài toán sắp xếp thành dãy không tăng thì phần tử thứ hai phải là phần tử lớn nhất trong các phần tử còn lại của dãy).
- Cứ như vậy, giả sử đã chọn được các phần tử thích hợp cho các vị trí từ thứ nhất đến vị trí thứ *i-1*. Rõ ràng phần tử thích hợp với vị trí thứ i phải là phần tử bé nhất (hoặc phần tử lớn nhất đối với bài toán sắp xếp thành dãy không tăng) trong các phần tử còn lại *{X[i], X[i+1], …, X[n]}*. Việc chọn phần tử thứ *i* có thể mô tả như sau:

  Kí hiệu *X[k]* = min *{X[i], X[i+1], …, X[n]}*;
  Nếu *k > i* thì đổi chỗ hai phần tử *X[k]* và *X[i]*;

Do có thể xét vị trí i từ 1 nên có thể mô tả thuật toán như sau:

Dữ liệu vào: Dãy X[1], X[2], ..., X[*n*]
Dữ liệu ra: Dãy X không giảm: X[1] ≤ X[2] ≤ ....≤ X[*n*];

```
for (i = 1; i<n; i++)
{
    X[k] = min { X[i], ,..., X[n]};
    if (k>i)  swap (X[k], X[i]);
}
```

Kỹ thuật đổi chỗ *X[k]* và *X[i]*:

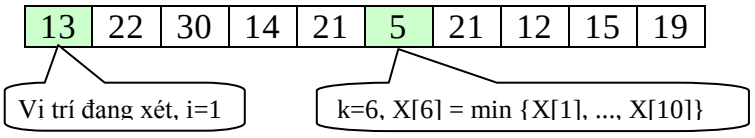
```
{
    tg = X[k];
    X[k]= X[i];
    X[i] = tg;
}
```

Ví dụ 5.1: Sắp xếp dãy số sau thành không giảm:

13, 22, 30, 14, 21, 5, 21, 12, 15, 19

Áp dụng thuật toán sắp xếp chọn đối với 3 vị trí đầu của dãy:

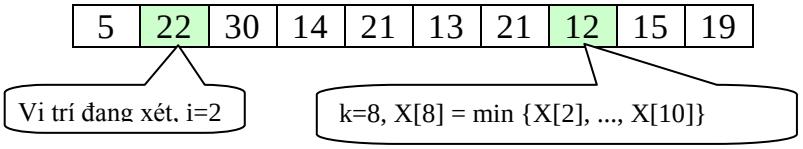
1) *i* = 1:



Đổi chỗ X[1], X[6]:

5	22	30	14	21	13	21	12	15	19

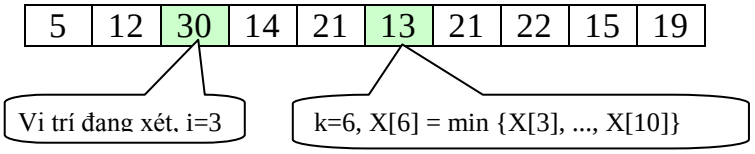
2) i = 2:



Đổi chỗ X[2], X[8]:

5	**12**	30	14	21	13	21	22	15	19

3) i = 3:



Đổi chỗ X[3], X[6]:

5	**12**	**13**	14	21	30	21	22	15	19

- Như vậy, sau khi thực hiện 3 bước của thuật toán, dãy đã có 3 phần tử đầu tiên đứng đúng vị trí (trong thứ tự không giảm).

Thuật toán chi tiết:

Dữ liệu vào: Dãy X[1], X[2], ..., X[*n*]

Dữ liệu ra: Dãy X không giảm: X[1] ≤ X[2] ≤ ....≤ X[n];

```
for     (i=1; i<n; i++)
{
    k = 1;
    for (j=k+1; j<=n; j++)  if   (X[j] <X[k])   k=j;
    if (k>i)  {tg = X[k]; X[k]= X[i];  X[i] = tg;}
}
```

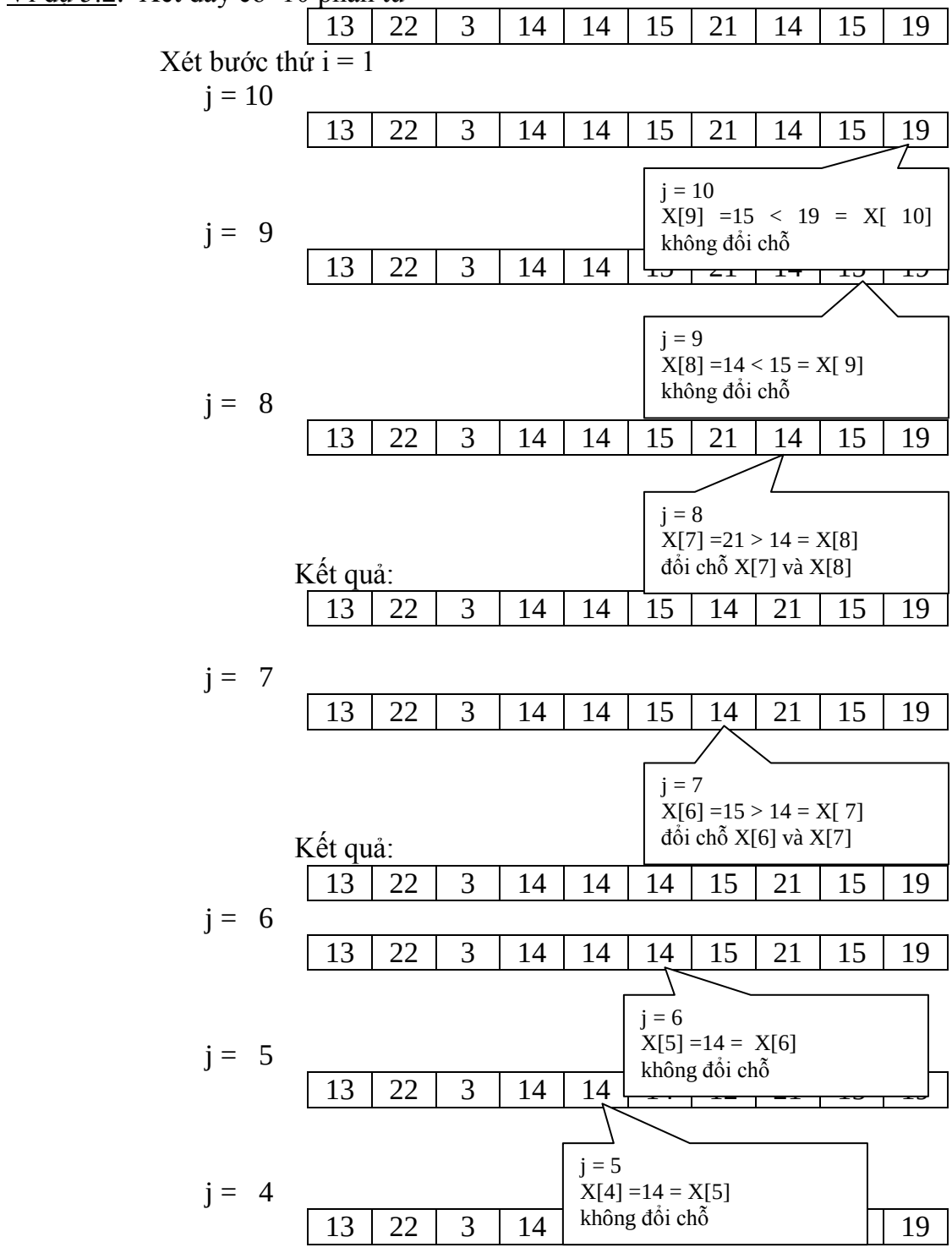
- Thuật toán có độ phức tạp $O(n^2)$, trong đó có *(n-1)(n+2)* phép so sánh và nhiều nhất là *3(n-1)* phép hoán đổi vị trí các phần tử.

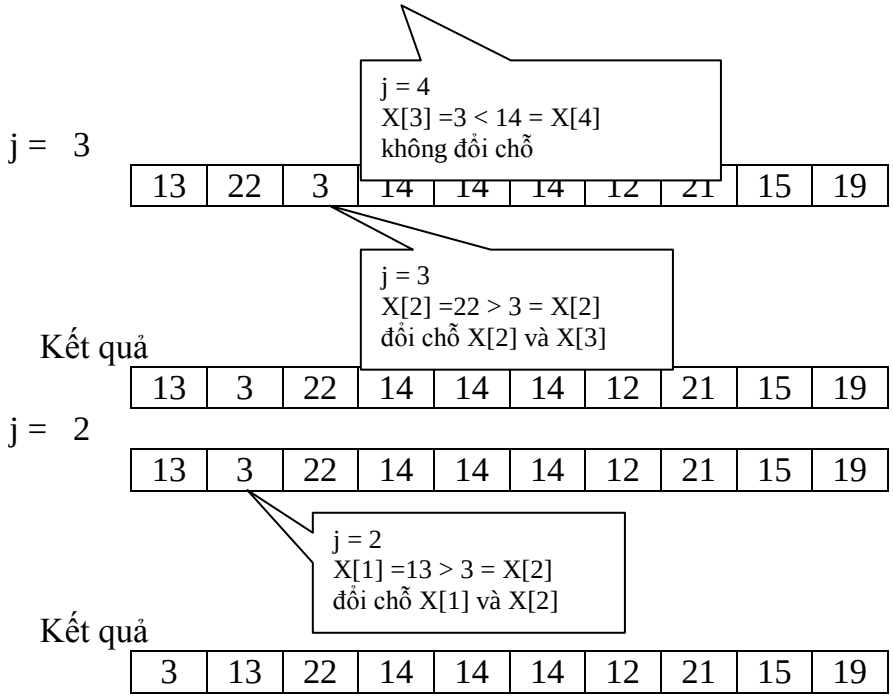
5.2.2. Sắp xếp nổi bọt

- Tại mỗi bước lớn của thuật toán sắp xếp nổi bọt một phần tử được đưa về đúng vị trí của nó. Như vậy, thuật toán sắp xếp nổi bọt cũng chỉ có n-1 bước lớn. Đây là điểm giống với thuật toán sắp xếp chọn.

- Điểm khác biệt là ở chỗ trong mỗi bước lặp của thuật toán sắp xếp chọn chỉ thông qua các phép toán so sánh để xác định vị trí hiện thời của phần tử thích hợp với vị trí đang được xét đến tại bước lặp, sau đó có thể thực hiện phép đổi chỗ để đưa phần tử về vị trí thích hợp; trong khi đó, với thuật toán sắp xếp nổi bọt, sẽ phải thực hiện liên tiếp một số phép toán so sánh và đổi chỗ để đưa được phần tử thích hợp về vị trí đang xét đến trong bước lặp của thuật toán. Chẳng hạn, xét bước thứ *i*, $1 \le i \le n$:
  a) Các phần tử ở vị trí trước i đều đã được chọn thích hợp;
  b) Xét lần lượt các cặp (X[j-1], X[j]), j = n, …, i+1, nếu X[j-1] > X[j] thì đổi chỗ X[j-1] và X[j].

Ví dụ 5.2: Xét dãy có 10 phần tử





- Trong ví dụ trên, sau khi kiểm tra từ chỉ số *j = 10* đến *j = 2* để đổi chỗ các cặp "nghịch biến", *X[j-1] > X[j]*, thuật toán đã chuyển được phần tử nhỏ nhất vào vị trí *i=1*. Không khó để chứng minh rằng phép toán b) ở bước thứ i nêu trên chuyển được phần tử nhỏ nhất trong số các phần tử *{X[i], ..., X[n]}* về vị trí thứ *i*.

Thuật toán chi tiết:

Dữ liệu vào: Dãy X[1], X[2], ..., X[*n*]

Dữ liệu ra: Dãy X không giảm: X[1] ≤ X[2] ≤ ....≤ X[*n*];

```
for      (i=1; i<n; i++)
{
    for (j=n; j>i; j- -)
    if  (X[j-1] >X[j])
    {tg = X[j-1]; X[j-1]= X[j];  X[j] = tg;}
}
```

- Thuật toán có độ phức tạp O(n^2), trong đó phải thực hiện *(n-1)(n+1)* phép so sánh và nhiều nhất là *3n(n-1)/2* phép hoán đổi vị trí các phần tử.

5.2.3. Sắp xếp chèn

Ý tưởng:

- Giả sử có phần đầu của mảng là *B(i) = { X[1],...,X[i] }* đã được sắp xếp không giảm:

  X[1] ≤ … ≤ X[i], (5.1)

- xét phần tử thứ i+1. Đưa (chèn) X[i+1] vào vị trí thích hợp trong dãy (5.1). Đặt tg = X[i+1]. Có ba khả năng xảy ra:

a) X[i] ≤ X[i+1]: Giữ nguyên X[i+1] ở vị trí thứ i+1.

b) Tồn tại chỉ số j, 1< j≤ i, sao cho X[j-1] ≤ tg < X[j]: Kéo toàn bộ các phần tử từ X[j],…, X[i] về phía sau. Chèn giá trị X[i+1] vào vị trí thứ j.

c) X[i+1] < X[1]. Kéo toàn bộ các phần tử từ X[1],…, X[i] về phía sau. Chèn giá trị X[i+1] vào vị trí thứ 1.

Trường hợp b) có thể mô tả như sau:

```
j = i+1;
tg = X[i+1];
```

```
while (X[j-1] > tg)
{   X[j] = X[j-1];
     j = j-1;
}
X[j] = tg;
```

- Trong đoạn mã trên, khi chỉ số j nhận giá trị 1 tương ứng với trường hợp c). Vì vậy, trường hợp b) và c) có thể được mô tả:

```
j = i+1;
tg = X[i+1];
while ((j>1) && (X[j-1] > tg))
{   X[j] = X[j-1];
    j = j-1;
}
X[j] = tg;
```

- Để ý rằng nếu *X[i] ≤ X[i+1]*, tức là *X[j-1] ≤ tg* ngay khi *j = i+1* thì rõ ràng là vòng lặp không được thực hiện bước nào, chỉ số *j* không thay đổi, nghĩa là phép gán *X[j] = tg* thực ra là gán *X[i+1]* cho chính nó, không làm thay đổi giá trị phần tử nào.
- Trong mọi trường hợp, luôn có thể coi *B(1) = { X[1] }* đã được sắp xếp không giảm. Như vậy, thuật toán sắp xếp chèn có thể được mô tả như sau:

Dữ liệu vào: Dãy X[1], X[2], ..., X[*n*]
Dữ liệu ra: Dãy X không giảm: X[1] ≤ X[2] ≤ ....≤ X[*n*];

```
for (i=2; i<=n;i++)
{   j = i+1;
     tg = X[i+1];
     while ((j>1) && (X[j-1] > tg))
     {       X[j] = X[j-1];
             j = j-1;
     }
     X[j] = tg;
}
```

Ví dụ 5.3: Xét dãy X:

X	13	22	30	14	5	21	21	12	15	19

Ta có đoạn B(3) = {X[1], X[2], X[3]} = {13, 22, 30} không giảm. Xét *i* = 4.
Đặt tg = 14, j= 4.

X	13	22	30	14	5	21	21	12	15	19
j				4						

X[j-1] = 30 > tg =14. Đặt X[j] = X[j-1], tức là X[4] = X[3], j =j -1= 3.

X	13	22	30	30	5	21	21	12	15	19
j			3							

X[j-1] = 22 > tg=14. Đặt X[j] = X[j-1], tức là X[3] = X[2], j =j -1= 2.

X	13	22	22	30	5	21	21	12	15	19
j		2								

X[j-1] = 13 < tg=14. Đặt X[j] = tg. Ta có dãy con B(4) được sắp không giảm.

X	**13**	**14**	**22**	**30**	5	21	21	12	15	19
j		2								

- Thuật toán sắp xếp chèn có độ phức tạp $O(n^2)$, trong đó, trong trường hợp xấu nhấtphải thực hiện *(n-1)(n+1)* phép so sánh và *n(n+1)/2* phép đổi chỗ.

B. MỘT SỐ KỸ THUẬT XỬ LÝ XÂU

- Trong thực tiễn việc xử lý chuỗi các ký tự xảy ra thường xuyên trên máy tính, vì thế nhu cầu định nghĩa kiểu dữ liệu đặc biệt cho chuỗi ký tự được đặt ra cho các ngôn ngữ lập trình. Trong ngôn ngữ lập trình C, chuỗi ký từ là mảng các ký tự với ký tự đặc biệt đánh dấu kết thúc – mã là 0. Do đặc trưng riêng và nhu cầu xử lý thường xuyên nên các ngôn ngữ lập trình xây dựng thư viện các hàm xử lý chuỗi ký tự.

2.2.1. Khai báo và nhập xuất dữ liệu

1. Khai báo

- *Cú pháp*:

  char Tên_biến_1[Số_phần_tử_1], … ,Tên_biến_n[Số_phần_tử_n];
- *Trong đó*:
- char: là từ khóa kiểu dữ liệu cho chuỗi.
- Tên_biến_1, …, Tên_biến_n: Tên biến do người sử dụng tự định nghĩa, lưu ý rằng tên biến phải tuân theo qui tắc đặt tên qui định bởi ngôn ngữ lập trình.
- Số_phần_tử_1,…, Số_phần_tử_n: Hằng số nguyên dương xác định số phần tử của của chuỗi. Tùy theo hỗ trợ của hệ điều hành và trình biên dịch mà số lượng phần tử của chuỗi là khác nhau.

Ví dụ 2.1: Khai báo chuỗi

```
char x[10];
```

- Khai báo chuỗi *x* có nhiều nhất là 10 ký tự, chỉ số để truy xuất đến các ký tự trong chuỗi x là từ 0 đến 9.
- Chú ý: Bản chất chuỗi là một mảng các ký tự với ký tự kết thúc là *'\0'* vì thế các quan niệm trên mảng vẫn đúng với chuỗi ký tự.
- Khai báo chuỗi với giá trị khởi tạo theo cú pháp sau.
- *Cú pháp*:

  char Tên_biến_1[Số_phần_tử_1] = {chuỗi_khởi_tạo_1}, …
  Tên_biến_n[Số_phần_tử_n] = {chuỗi_khởi_tạo_n};
- *Trong đó*:
- Tên_biến_1, …, Tên_biến_n, Số_phần_tử_1, …, Số_phần_tử_n được hiểu như trên;
- chuỗi_khởi_tạo_1, …, chuỗi_khởi_tạo_n là chuỗi khởi tạo cho các giá trị biến.
- Trong trường hợp khai báo có khởi tạo các giá trị, độ dài của chuỗi có thể được để trống, lúc đó hệ thống sẽ xác định độ dài của chuỗi theo độ dài chuỗi khởi tạo.
- Ví dụ 2.2: Khai báo mảng có khởi tạo giá trị

```
char s[100]= "Xin chao den voi ky thua lap trinh";
char a[]="Ky thuat lap trinh co ban";
```

- Khai báo chuỗi *s* có độ dài tối đa là 100 ký tự, được khởi tạo chuỗi ban đầu là *"*Xin chao den voi ky thua lap trinh*"*. Chuỗi a được khởi tạo là *"*Ky thuat lap trinh co ban*"* với độ dài tối đa của chuỗi chính là độ dài của chuỗi khởi tạo.
- Chú ý: Cách khai báo chuỗi ký tự trên hoàn toàn khác với khai báo

  ```
  char *b="Con tro den hang so chuoi";
  ```

  - trường hợp này khai báo một con trỏ kiểu ký tự và khởi tạo của con trỏ được trỏ đến địa chỉ chứa hằng số là chuỗi trong chương trình. Vì là hằng số chuỗi nên mọi thay đổi trên dữ liệu trỏ đến bởi con trỏ b sẽ nảy sinh lỗi.

2. Truy xuất phần tử

- Truy xuất phần tử của chuỗi thông qua cặp ngoặc [,] như truy xuất phần tử của

mảng. Mỗi phần tử của chuỗi là một ký tự.

- Ví dụ 2.3: Ví dụ truy xuất phần tử của chuỗi

```
char s[100]= "Xin chao den voi ky thua lap trinh";
s[0]='x';
```

3. Nhập xuất dữ liệu xâu

- Chuỗi là kiểu mảng đặc biệt được xem như là kiểu dữ liệu nên trong ngôn ngữ lập trình C hỗ trợ các hàm nhập xuất xâu.
  - **a) Nhập xâu**
- Sử dụng hàm scanf() với tham số %s
- Ví dụ 2.4: Nhập xâu bằng hàm scanf()

```
char s[100];
scanf("%s",s);
```

- Sử dụng hàm gets()
- *Cú pháp*:

```
char *gets(char *s);
```

  - Tham số đầu vào s là con trỏ của xâu. Giá trị trả về: con trỏ đến xâu đã được nhập vào.
- Ví dụ 2.5: Nhập xâu bằng hàm gets()

```
char a[100];
char *b;
b=gets(a);
```

  - **b) Xuất chuỗi**
- Xuất xâu lên màn hình bằng hàm printf() với tham số %s.
- Ví dụ 2.6: Xuất xâu bằng hàm printf()

```
char s[100]= "Xin chao den voi ky thua lap trinh";
printf("Loi chao: %s",s);
```

- Xuất xâu lên màn hình sử dụng hàm puts()

Cú pháp:

```
int puts(const char *s);
```

  - Trong đó s là xâu cần được đưa lên màn hình, sẽ đưa các ký tự lên màn hình đến khi gặp ký tự kết thúc có mã ASCII là 0. Giá trị trả về của hàm trên là số ký tự đã xuất lên màn hình.
- Ví dụ 2.7: Xuất xâu bằng hàm puts()

```
char s[100]= "Xin chao den voi ky thua lap trinh";
int c;
c=puts(s);
```

- Xây dựng xâu từ hàm sprintf()
- *Cú pháp*:

```
int sprintf (char *buffer, const char *format [, argument, ...]);
```

  - Trong đó buffer là xâu nhận kết quả. Hàm sprintf() tương tự hàm printf() nhưng kết quả thay về đưa lên màn hình sẽ đưa vào trong xâu buffer.
- Ví dụ 2.8: Xuất xâu bằng hàm sprintf()

```
#include <stdio.h>
#include <conio.h>
int main()
{
   char s[100];
   int a=5, b=7;
```

```
        sprintf(s,"Tich cua %d va %d la %d",a,b,a*b);
        puts(s);
        getch();
        return 0;
    }
```

2.2.2. Các hàm xử lý dữ liệu xâu

- Trong ngôn ngữ lập trình C các hàm xử lý xâu được định nghĩa trong thư viện string.h.
- **a) Các hàm chuyển đổi kiểu chữ, chuyển đổi thứ tự**
- Chuyển ký tự thành chữ thường
- *Cú pháp*:

```
char *strlwr(char *s);
```

- *Trong đó*:

s là xâu đầu vào, hàm chuyển các ký tự trong xâu s thành chữ thường.
Giá trị trả về của hàm là con trỏ chứa xâu đã được chuyển thành chữ thường.

- Chuyển ký tự thành chữ hoa
- *Cú pháp*:

```
char *strupr(char *s);
```

- *Trong đó*:

Tham số s là xâu đầu vào, hàm chuyển các ký tự trong xâu s thành chữ hoa.
Giá trị trả về của hàm là con trỏ chứa xâu đã chuyển thành chữ hoa.

- Đảo các ký tự trong xâu
- *Cú pháp*:

```
char *strrev(char *s);
```

- *Trong đó*:

Tham số s là xâu đầu vào, hàm sẽ đảo ngược thứ tự xuất hiện các ký tự.
Giá trị trả về của hàm là con trỏ đến xâu đã được đảo ngược.

- Ví dụ 2.9: Hàm chuyển đổi ký tự

```
#include <conio.h>
#include <stdio.h>
#include <string.h>
int main(void)
{
    char a[]="Chuyen Thanh Chu Hoa";
    char b[]="Chuyen Thanh Chu Thuong";
    char c[]="Dao nguoc thu tu";
    strupr(a);
    strlwr(b);
    strrev(c);
    puts(a);
    puts(b);
    puts(c);
    getch();
    getch();
    return 0;
}
```

- **b) Hàm xác định chiều dài xâu**
- Xác định chiều dài của xâu văn bản. Chiều dài của xâu văn bản kết thúc bởi ký tự

có mã 0.

- *Cú pháp*:

size_t strlen(const char *s);

- *Trong đó*:
- Tham số s là xâu cần kiểm tra độ dài.
- Giá trị trả về của hàm là độ dài của xâu

- **c) Hàm sao chép, ghép xâu**

- Sao chép xâu

- *Cú pháp*:

char *strcpy(char *dest, const char *src);

- *Trong đó*:

Tham số dest là địa chỉ xâu sẽ nhận giá trị sao chép.
Tham số src là địa chỉ xâu nguồn được sao chép.
Giá trị trả về của hàm là con trỏ xâu được sao chép.

- Hàm strcpy tiến hành sao chép các ký tự từ xâu src sang xâu dest đến khi gặp ký tự có mã là 0 (ký tự kết thúc xâu). Hàm sao chép cả ký tự kết thúc xâu.

- Nối xâu

- *Cú pháp*:

char *strcat(char *dest, const char *src);

- *Trong đó*:

Tham số dest là xâu sẽ được cộng thêm xâu src vào cuối.
Tham số src là xâu cộng thêm vào xâu dest.
Giá trị trả về của hàm là con trỏ chỉ đến vùng nhớ xâu được cộng vào.

- Hàm strcat thực hiện nối xâu src vào xâu dest.

- Tạo bản sao xâu mới

- *Cú pháp*:

char *strdup(const char *s);

- *Trong đó*:

Tham số s là xâu sẽ được tạo bản sao.
Giá trị trả về của hàm là con trỏ trỏ đến vùng bộ nhớ mới được cấp phát để lưu trữ xâu.

- Hàm strdup thực hiện việc cấp pháp một vùng nhớ mới có độ lớn bằng độ dài xâu, và tiến hành bản sao sang vùng mới này. Vì hàm này cấp phát bộ nhớ nên sau khi sử dụng phải sử dụng hàm free() để giải phóng vùng nhớ.

- Chú ý: Các hàm strcpy(), strcat() không giới hạn số lượng ký tự sẽ đưa vào xâu dest vì thế trong trường hợp xâu src lớn sẽ tạo ra hiện tượng tràn bộ đệm (vùng nhớ lưu trữ không đủ) vì thế trong trường hợp không giới hạn được xâu đầu vào nên sử dụng các hàm strncpy(), strncat() thay thế.

- Ví dụ 2.10: Copy và nối xâu

```
#include <conio.h>
#include <stdio.h>
#include <string.h>
int main(void)
{
    char t[100];
    char a[10];
    char b[20]= "Thu";
    char c[10]= "123456789";
    char d[]="Chuoi dai hon vung dem";
```

```
    strcpy(a,b);
    puts(a);
    strcat(b,c);
    puts(b);
    strncpy(a,d,10);
    a[9]=0;
    puts(a);
    strncat(b,d,20);
    b[19]=0;
    puts(b);
    //loi tran bo dem
    printf("\nLoi\n");
    strcpy(a,d);
    strcat(b,d);
    puts(a);
    puts(b);
    getch();
    return 0;
}
```

- **d) Hàm so sánh hai xâu phân biệt chữ hoa chữ thường**

- *Cú pháp:*

```
int strcmp(const char *s1, const char*s2);
```

- *Trong đó*:
- Tham số s1 xâu cần so sánh thứ nhất.
- Tham số s2 xâu cần so sánh thứ hai.
- Giá trị trả về của hàm bé hơn 0 nếu xâu s1 bé hơn xâu s2, bằng 0 nếu xâu s1 bằng xâu s2, lớn hơn 0 nếu xâu s1 lớn hơn xâu s2.
- Hàm so sánh từ trái sang phải theo mã của các ký tự. Hai xâu bằng nhau khi xâu có độ dài bằng nhau, các giá trị giống nhau. Nếu từ trái sang phải gặp ký tự khác nhau đầu tiên nếu ký tự thuộc xâu s1 có mã lớn ký tự từ xâu s2 thì chuỗi s1 lớn hơn xâu s2 hoặc ngược lại.
    - Ví dụ 2.11: So sánh xâu

```
char a[]="xin chao";
char b[]="xin chao";
char c[]="xin don mung cac ban";
int i1, i2;
i1=strcmp(a,b);
i2=strcmp(a,c);
```

- Sau khi thực hiện: i1 có giá trị là 0 (xâu a bàng xâu b), i2 có giá trị bé hơn 0 (xâu a bé hơn xâu c - xét từ trái sang khi đến ký tự thứ 4 thì ký tự 'd' sẽ có mã lớn hơn ký tự 'c' nên xâu c lớn hơn xâu a).
- Ngoài ra, để so sánh hai xâu không phân biệt chữ hoa, chữ thường có thể sử dụng cú pháp sau đây.

- *Cú pháp:*

```
int stricmp(const char *s1, const char *s2);
```

- Hàm này cơ bản giống hàm strcmp nhưng khi so sánh thì không biệt chữ hoa và chữ thường có nghĩa hai ký tự 'A' và 'a' là như nhau về thứ tự so sánh.

- **e) Hàm tìm sự xuất hiện đầu tiên của ký tự trong xâu**

- *Cú pháp:*
  char *strchr(const char *s, int c);
- *Trong đó*:
- Tham số s là xâu sẽ tìm kiếm ký tự trên đó.
- Tham số c là ký tự cần tìm kiếm.
- Nếu xuất hiện ký tự cần tìm kiếm (tính từ trái qua) thì hàm trả về con trỏ trỏ đến vị trí của ký tự xuất hiện đầu tiên; nếu không xuất hiện ký tự cần tìm kiếm thì con trỏ trỏ đến NULL.
- Ngoài ra, để tìm ký tự xuất hiện đầu tiên bên phải có thể sử dụng cú pháp sau đây.
- *Cú pháp:*
  char *strrchr(const char *s, int c);
- Hàm làm việc tương tự hàm strchr*()* hướng tìm kiếm là từ phải sang.

- **f) Tìm kiếm xâu con xuất hiện trong xâu khác**

- *Cú pháp:*
  char *strstr(const char *s1, const char *s2);
- *Trong đó*:
- Tham số s1 là xâu sẽ được tìm kiếm trên đó.
- Tham số s2 là xâu con cần tìm kiếm trên xâu s1.
- Nếu tìm thấy sự xuất hiện của xâu s2 trên xâu s1 thì hàm sẽ trả về con trỏ trỏ đến ký tự đầu tiên xuất hiện của xâu con; ngược lại trả lại giá trị NULL.

- **g) Các hàm biến đổi kiểu xâu thành số**

- Chuyển xâu thành số nguyên
- *Cú pháp:*
  int atoi(const char *s);
  long atol(const char *s);
- *Trong đó*:
- Tham số s là xâu có định dạng kiểu số.
- Giá trị trả về của hàm là số tương ứng với xâu. Nếu hàm atoi() trả về kiểu int, atol() trả về kiểu long. Nếu lỗi sẽ trả lại giá trị 0.
- Chuyển đổi xâu thành số thực
- *Cú pháp:*
  double atof(const char *s);
- *Trong đó*:
- Tham số s là xâu có định dạng kiểu số.
- Giá trị trả về của hàm là số. Nếu có lỗi sẽ trả lại giá trị 0.

2.2.3. Một số ví dụ

1. Loại bỏ các ký tự trắng (spacebar) cạnh nhau, và đầu xâu, cuối xâu

- *Bài toán*: Cho một xâu là một đoạn văn bản hãy loại bỏ các dấu cách cạnh nhau (nếu có nhiều dấu cách cạnh nhau chỉ để lại một dấu cách), loại bỏ các dấu cách ở đầu, và cuối của văn bản.
- *Ý tưởng*: Tìm ký tự đầu tiên khác ký tự trắng là st. Tìm vị trí cuối cùng của xâu là khác ký tự trắng là en. Chuyển các ký tự từ khoảng st đến en sang xâu mới nếu ký tự đó không đồng thời là ký tự trắng và ký tự trước đó cũng là ký tự trắng.

Thuật toán:
Dữ liệu vào: Xâu cần loại bỏ ký tự trắng dư thừa
Dữ liệu ra: Xâu đã được loại bỏ ký tự trắng.
len=strlen(s);
st=0;

```
en=len-1;
while(st<len && s[st]==' ')
    st++;
while(en>st && s[en]==' ')
    en--;
mo=1;
s[0]=s[st];
    for(i=st+1 -> en)
            if(!(s[i]==' ' && s[i-1]==' '))
            {
                    s[mo]=s[i];
                    mo++;
            }
    s[mo]=0;
```

- *Cài đặt chương trình*:

```
#include <conio.h>
#include <stdio.h>
#include <string.h>
int main(void)
{
    char s[]="  chuoi   co   du nhieu   dau trang      ";
    int st=0, len, en, mo=1,i;
    len=strlen(s);
    en=len;
    while(st<len && s[st]==' ')
            st++;
    while(en>st && s[en]==' ')
            en--;
    s[0]=s[st];
    for(i=st+1;i<en;i++)
    {
            if(!(s[i]==' ' && s[i-1]==' '))
            {
                    s[mo]=s[i];
                    mo++;
            }
    }
    s[mo]=0;
    puts(s);
    getch();
    return 0;
}
```

2. Chuẩn hóa tên

- *Bài toán*: Tên đảm bảo qui tắc chữ đầu tên là viết hoa, các chữ cái khác viết thường.
- *Ý tưởng*: Chuyển đổi các ký tự trong xâu thành chữ thường. Chuyển đổi chữ cái đầu tiên thành chữ hoa. Tìm các chữ cái đứng sau dấu cách chuyển thành chữ hoa.

Thuật toán:

Dữ liệu vào: Xâu tên
Dữ liệu ra: Xâu tên đã được chuẩn hóa.

```
s=strlwr(s);
s[0]=toupper(s[0]);
for(i=1 -> N)
        if(s[i-1]==' ')
                s[i]=toupper(s[i]);
```

- *Cài đặt chương trình*:

```
#include <conio.h>
#include <stdio.h>
#include <string.h>
#include <ctype.h>
int main(void)
{
    char s[]="nguyen LAN ANH";
    int i;
    strlwr(s);
    s[0]=toupper(s[0]);
    for(i=1;i<strlen(s);i++)
    {
            if(s[i-1]==' ')
                    s[i]=toupper(s[i]);
    }
    puts(s);
    getch();
    return 0;
}
```

3. Phép toán cộng cho hai xâu số lớn

- *Bài toán*: Lưu số lớn trong hai xâu số, thực hiện phép cộng cho hai số lớn trên.
- *Ý tưởng*: Tìm độ dài của hai lớn nhất của hai xâu số. Số nhớ ban đầu bằng 0. Duyệt theo độ dài của hai số lấy từ phải sang nếu không thuộc xâu sẽ nhận giá trị là 0. Cộng hai số lấy được cùng với số nhớ hiện tại. Số hiện tại của xâu là số dư của tổng chia cho 10. Số nhớ tiếp theo là tổng chia cho 10. Trường hợp sau khi cộng xong vẫn còn số dư thì tính tiếp cho số tiếp theo. Do thứ tự hiển thị xâu là ngược tiến hành đảo ngược xâu.

Thuật toán:

Dữ liệu vào: Xâu số thức nhất s1 và xâu số thứ 2 s2.
Dữ liệu ra: Xâu số tổng.

```
cl1=strlen(s1);
cl2=strlen(s2);
clm=timmax(cl1,cl2);
sd=0;
for(i=0 ->cml)
{
    i1=0;
    i2=0;
    if(i<cl1)
            i1=chuyensangso(s1[cl1-i-1];
```

```
    if(i<cl2)
            i2=chuyensangso(s2[cl2-i-1];
    t=sd+i1+i2;
    s[i]=chuyensangkytu(t%10);
    sd=t/10;
    }
    if(sd>0)
    {
            s[i]=chuyensangkytu(sd);
            i++;
    }
    s[i]=0;
    s=strrev(s);
```

- *Cài đặt chương trình*:

```
#include <conio.h>
#include <stdio.h>
#include <string.h>
int main(void)
{
    char s1[]="12345678";
    char s2[]="3624354445345344678";
    char s[100],sd=0,t,i1,i2;
    int i, cl1, cl2, cml;
    cl1=strlen(s1);
    cl2=strlen(s2);
    cml=cl1;
    if(cml<cl2)
            cml=cl2;
    for(i=0;i<cml;i++)
    {
            i1=0;
            i2=0;
            if(i<cl1)
                    i1=s1[cl1-i-1]-'0';
            if(i<cl2)
                    i2=s2[cl2-i-1]-'0';
            t=sd+i1+i2;
            s[i]='0'+t%10;
            sd=t/10;
    }
    if(sd>0)
    {
            s[i]='0'+sd;
            i++;
    }
    s[i]=0;
    strrev(s);
    puts(s);
    getch();
```

```
    return 0;
}
```

Kết quả: Chương trình **QLSV** chạy được theo các yêu cầu đặt ra.