|  |
| --- |
| **TRƯỜNG ĐẠI HỌC GIAO THÔNG VẬN TẢI PHÂN HIỆU THÀNH PHỐ HỒ CHÍ MINH**    **BÁO CÁO BÀO TẬP LỚN**  Môn: Lập trình nâng cao  Giảng viên: Cô Trần Thị Dung  Nhóm: Nguyễn Thành Tiến-Nguyễn Văn Tiến-Lê Quang Sơn  Lớp: CQ.60.CNTT  Đề tài: xây dựng chương trình quản lý sinh viên bằng ngôn ngữ C  THÀNH PHỐ HỒ CHÍ MINH 2020 |

ĐẠI HỌC GIAO THÔNG VẬN TẢI PHÂN HIỆU THÀNH PHỐ HỒ CHÍ MINH

BÁO CÁO BÀI TẬP LỚN

Môn: Lập trình nâng cao

THÀNH PHỐ HỒ CHÍ MINH 2020

**MỤC LỤC**

[**I.** **Danh sách liên kết đơn, cấp phát bộ nhớ và các thuật toán.** 4](#_Toc43048433)

[***1.1.*** ***Cấp phát bộ nhớ.*** 4](#_Toc43048434)

[***1.2.Cấu trúc (struct).*** 5](#_Toc43048435)

[***1.3.*** ***Danh sách liên kết đơn*** 6](#_Toc43048436)

[**1.3.1 Cài đặt danh sách.** 7](#_Toc43048437)

[**1.3.2 Khởi tạo danh sách rỗng.** 7](#_Toc43048438)

[**1.3.3 Kiểm tra danh sách có rỗng hay không.** 7](#_Toc43048439)

[**1.3.4 Tính độ dài danh sách.** 8](#_Toc43048440)

[**1.3.5 Tạo một node trong danh sách.** 8](#_Toc43048441)

[**1.3.6 Chèn Node p vào đầu danh sách.** 8](#_Toc43048442)

[**1.3.7 Chèn Node p vào vị trí k của danh sách.** 9](#_Toc43048443)

[**1.3.8 Tìm phần tử có giá trị x trong danh sách.** 9](#_Toc43048444)

[**1.3.9 Xóa phần ở vị trí đầu tiên.** 10](#_Toc43048445)

[**1.3.10 xóa phần tử ở vị trí k.** 10](#_Toc43048446)

[**1.3.11 Xóa phần tử có giá tri x.** 11](#_Toc43048447)

[**2. Các thuật toán sắp xếp.** 11](#_Toc43048448)

[**2.1 Sắp xếp nổi bọt (bubble sort)** 11](#_Toc43048449)

[**2.2 Sắp xếp chèn.** 12](#_Toc43048450)

[**2.3 Sắp xếp chọn.** 13](#_Toc43048451)

[**2.4 Sắp xếp đổi chỗ.** 14](#_Toc43048452)

[**2.5 Sắp xếp dãy số giảm dần** 15](#_Toc43048453)

[**2.6 Sắp xếp dãy dố tăng dần.** 16](#_Toc43048454)

[**3. Các thuật toán tìm kiếm.** 17](#_Toc43048455)

[**3.1 Tìm kiếm tuyến tính.** 17](#_Toc43048456)

[**3.2 Tìm kiếm nhị phân.** 17](#_Toc43048457)

[**4. Các thao tác với file làm việc với tệp.** 18](#_Toc43048458)

[**4.1.** **Làm việc với tệp.** 18](#_Toc43048459)

[**a.** **File văn bản – text files.** 18](#_Toc43048460)

[**b.** **File nhị phân – Binary files.** 18](#_Toc43048461)

[**4.2.** **Các thao tác với file** 19](#_Toc43048462)

[**4.2.1 Khai báo sử dụng FILE.** 19](#_Toc43048464)

[**4.2.2** **Thao tác mở file – hàm fopen.** 19](#_Toc43048465)

[**4.2.3** **Thao tác đóng file – hàm fclose.** 21](#_Toc43048466)

[**4.2.4** **Đóng tất cả các hàm đang mở - hàm fcloseall.** 22](#_Toc43048467)

[**4.2.5 Đọc/Ghi file trong văn bản C.** 23](#_Toc43048468)

[**4.2.6 Đọc/Ghi file nhị phân trong C.** 28](#_Toc43048469)

[**I.** **Lý do chọn đề tài.** 30](#_Toc43048470)

[**II.** **Giải thích chi tiết bài tập.** 30](#_Toc43048471)

[**1.** **Mô tả công việc cần làm.** 30](#_Toc43048472)

[**2.** **Thực hiện các công việc cần làm.** 30](#_Toc43048473)

[**a.** **Tạo cấu trúc sinh viên.** 30](#_Toc43048474)

[**b.** **Nhập tên sinh viên và in ra màn hình.** 31](#_Toc43048475)

[**c.** **Sắp xếp sinh viên theo tên.** 31](#_Toc43048476)

[**d.** **Sắp xếp điểm trung bình 3 môn học.** 32](#_Toc43048477)

[**e.** **Tìm kiếm và hiển thị sinh viên trong danh sách.** 32](#_Toc43048478)

[**f.** **Đọc/Ghi thông tin sinh viên vào tệp txt.** 33](#_Toc43048479)

**Phần 1: Cơ sở lý thuyết.**

1. **Danh sách liên kết đơn, cấp phát bộ nhớ và các thuật toán.**
   1. ***Cấp phát bộ nhớ.***

* Dùng để cấp phát vùng nhớ cho biến con trỏ trong ngôn ngữ C.
* Sử dụng các hàm mallo(), calloc().
* Sử dụng hàm free() để giải phóng bộ nhớ đã cấp phát cho biến con trỏ khi không cần sử dụng.
* Để thay đổi kích thước bộ nhớ đã cấp phát trong khi chạy chương trình t sử dụng hàm realloc().
* Cú pháp sử dụng các hàm:
* Hàm malloc(): là hàm thực hiện cấp phát bằng cách chỉ định số byte cần cấp phát, hàm trả về kiểu void cho phép có thể ép kiểu bất cứ kiểu dữ liệu nào.

Cú pháp: ptr = (castType\*)malloc(size);

Ví dụ: ptr = (int\*)malloc(100 \* sizeof(int));

* Hàm calloc(): là hàm cấp phát bộ nhớ thì vùng nhớ ko được khởi tạo lại giá trị ban đầu, thực hiện cấp phát và khởi tạo tất cả các ô nhớ có giá trị bằng 0.

Cú pháp: ptr = (castType\*)calloc(n,size);

Ví dụ: ptr = (int\*)calloc(100,sizeof(int));

* Hàm free(): là hàm dùng để giải phóng bộ nhớ cho hàm malloc() và calloc() vì chúng không thể tự giải phóng bộ nhớ.

Cú pháp: free(ptr);// ptr là con trỏ

* Hàm realloc(): là hàm thay đổi kích thước bộ nhớ đã được cấp phát trước đó khi bộ nhớ động không đủ hoặc cần nhiều hơn mức đã cấp phát.

Cú pháp: ptr = realloc(ptr, n);

Ví dụ:

|  |
| --- |
| #include <stdio.h>  #include <stdlib.h>  int main(){  int \*ptr, i, n1, n2;  printf(“nhap so luong phan tu: ”);  scanf(“%d”, &n1);  ptr = (int\*)malloc(n1 \* sizeof(int));  printf(“Dia chi cua vung nho vua cap phat: %u”, ptr);  printf(“nhap lai so luong phan tu: ”);  scanf(“%d”, &n2);  // phân bổ lại vùng nhớ  ptr = (int\*)realloc(ptr, n2 \* sizeof(int));  printf(“Dia chi cua vung nho duoc cap phat lai: %u”, ptr);  // giải phóng bộ nhớ  free(ptr);  return 0;  } |

## ***1.2.Cấu trúc (struct).***

* Cấu trúc là một tập hợp các biến, các mảng và được biểu thị bởi một tên duy nhất.
* Cấu trúc có thể xem như một sự mở rộng của các khái niệm biến và mảng, nó cho phép lưu trữ và sử lý các dạng thông tin phức tạp hơn.
* Cú pháp:

struct structurename

{

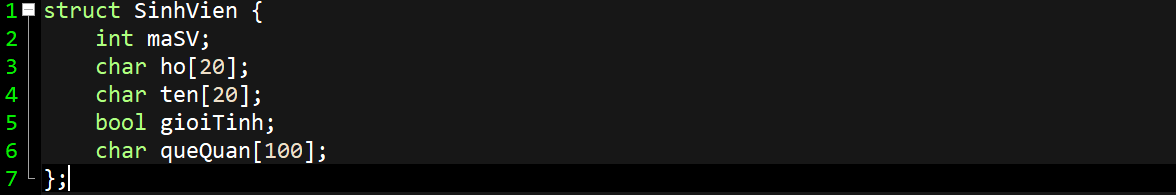
dataType member1;

dataType member2;

…

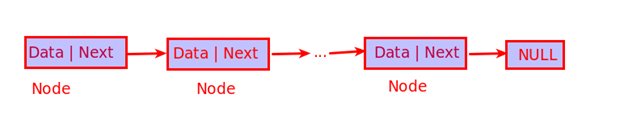
};

Ví dụ:



* 1. ***Danh sách liên kết đơn***

Danh sách liên kết đơn là một tập hợp các Node được phân bố động, được sắp xếp theo cách sao cho mỗi Node chứa “một giá trị”(Data) và “một con trỏ”(Next). Con trỏ sẽ trỏ đến phần tử kế tiếp của danh sách liên kết đó. Nếu con trỏ mà trỏ tới NULL, nghĩa là đó là phần tử cuối cùng của linked list.



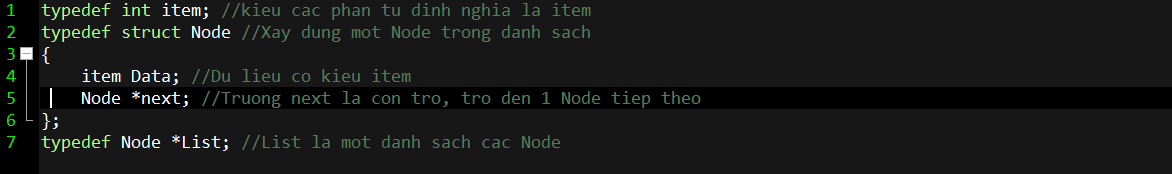
So sánh giứa mảng và danh sách liên kết:

|  |  |
| --- | --- |
| Mảng | Danh sách liên kết đơn |
| Vùng nhớ của các phần tử trong mảng được sắp xếp liên tục nhau. | Vùng nhớ của các phần tử trong danh sách liên kết được sắp xếp tùy ý (do hệ điều hành). Các phần tử lưu 1 con trỏ trỏ tới phần tử tiếp theo. |
| Truy cập tới phần tử trong mảng là truy cập trực tiếp dựa vào chỉ số (ví dụ: a[0], a[1], a[2],…, a[n]). | Cần phải duyệt tuần tự khi muốn truy cập tới phần tử trong danh sách liên kết. |
| * Kích thước của mảng là hằng số, không thay đổi khi chạy chương trình * Sử dụng mảng không tối ưu được bộ nhớ. Có thể thừa hoặc thiếu bộ nhớ khi xóa hoặc chèn phần tử vào mảng | * Kích thước của danh sách liên kết có thể thay đổi khi chạy chương trình. * Sử dụng danh sách liên kết tối ưu được bộ nhớ. Vùng nhớ được cấp phát thêm khi cần chèn thêm phần tử mới, vùng nhớ được free khi xóa phần tử. |

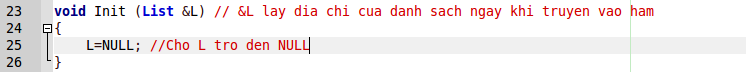
Các hoạt động cơ bản của danh sách liên kết đơn:

* Cài đặt danh sách (Khai báo)
* Khởi tạo danh sách rỗng
* Kiểm tra danh sách rỗng (danh sách đầy khi cài bằng mảng)
* Chèn phần tử vào đầu danh sách
* Chèn phần tử vào vị trí thứ k trong danh sách
* Nhập danh sách
* Xuất danh sách
* Tìm 1 phần tử trong danh sách
* Xóa phần tử đầu tiên trong danh sách
* Xóa phần tử thứ k trong danh sách
* Xóa phần tử có nội dung X trong danh sách

**1.3.1 Cài đặt danh sách.**

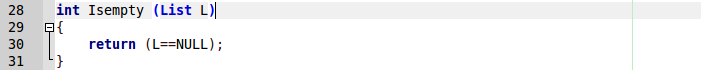


### **1.3.2 Khởi tạo danh sách rỗng.**

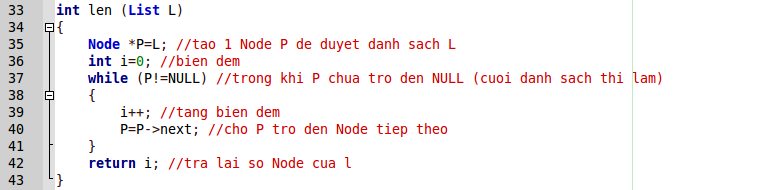


Trong các bài trước để có thể thay đổi được giá trị của đối mà ta truyền vào hàm ta thường dùng biến con trỏ (\*) và trong lời gọi hàm ta cần có & trước biến tuy nhiên khi chúng ta sử dụng cách truyền địa chỉ ngay khi khởi tạo hàm thì trong lời gọi hàm ta tiên hành truyền biến bình thường mà không phải lấy địa chỉ (thêm &) trước biến nữa.

### **1.3.3 Kiểm tra danh sách có rỗng hay không.**



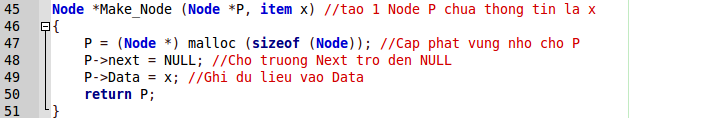
### **1.3.4 Tính độ dài danh sách.**



Ta dùng một node để duyêt từ đầu đến cuối, vừa duyệt vừa đếm.

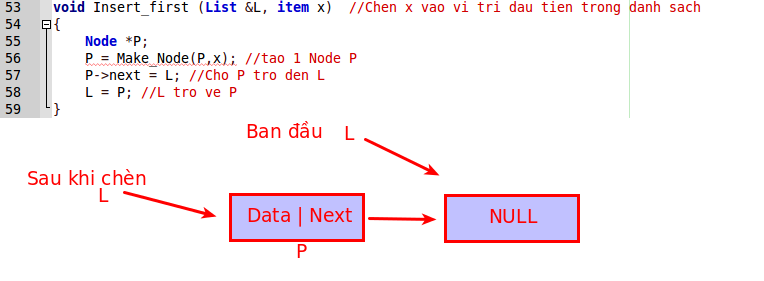
### **1.3.5 Tạo một node trong danh sách.**

Việc tạo 1 Node chứa thông tin trong danh sách sẽ giúp ta dễ dàng chèn, xóa và quản lý danh sách hơn. Trước tiên ta sẽ phải cấp phát vùng nhớ cho Node và sau đó gán Data vào là ok.



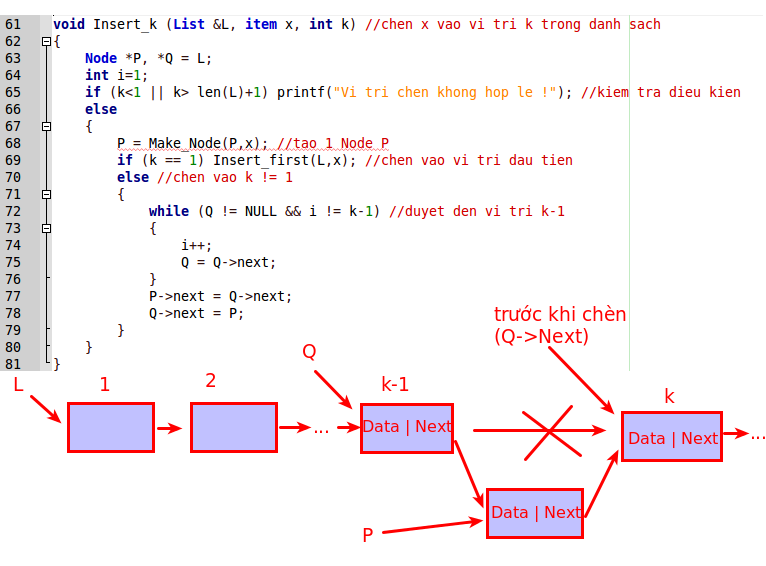
### **1.3.6 Chèn Node p vào đầu danh sách.**

Để chèn P vào đầu danh sách trước tiên ta cho P trỏ đến L, sau đó chỉ việc cho L trỏ lại về P là ok.



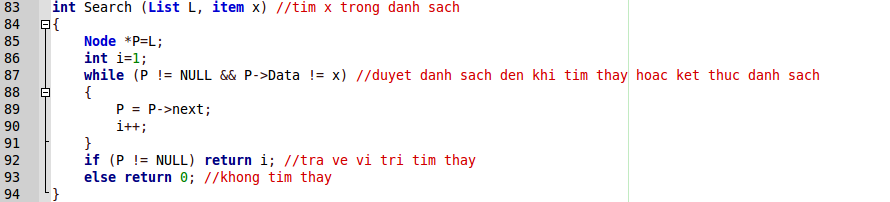
### **1.3.7 Chèn Node p vào vị trí k của danh sách.**

Trước tiên ta kiểm tra vị trí chèn có hợp lệ không, nếu hợp lệ kiểm tra tiếp chèn vào vị trí 1 hay k >1. Với k >1 ta thực hiện duyệt bằng Node Q đến vị trí k-1 sau đó cho P->Next trỏ đến Node Q->Next, tiếp đến cho Q->Next trỏ đến P.



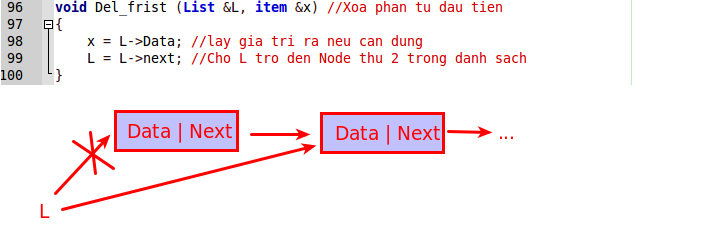
### **1.3.8 Tìm phần tử có giá trị x trong danh sách.**

Ta duyệt danh sách cho đến khi tìm thấy hoặc kết thúc và trả về vị trí nếu tìm thấy, ngược lại trả về 0.



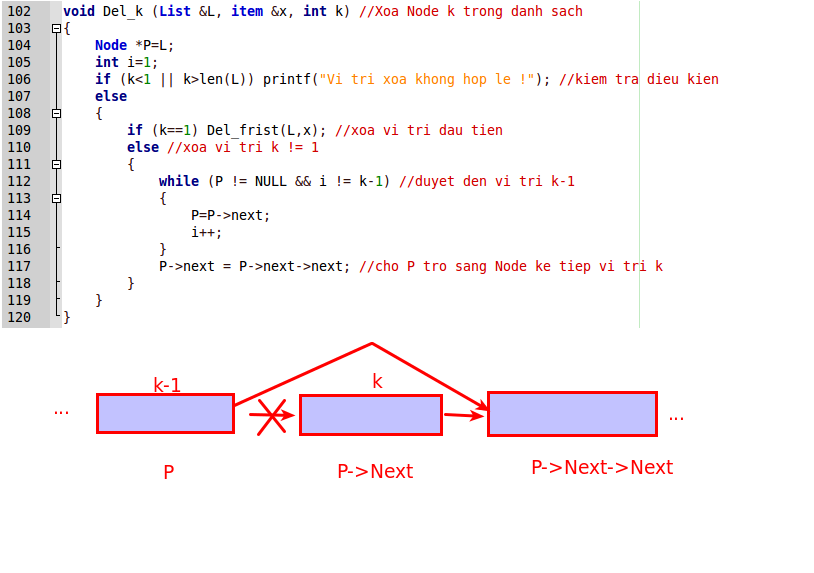
### **1.3.9 Xóa phần ở vị trí đầu tiên.**

Trước tiên ta lưu giá trị của phần tử đầu tiên vào biến x, sau đó tiền hành cho L trỏ đến L->Next.



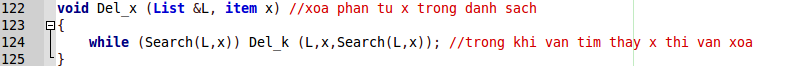
### **1.3.10 xóa phần tử ở vị trí k.**

Dùng P duyệt đến vị trí k-1 và tiến hành cho P->Next trỏ đến phần tư kế tiếp k mà bỏ qua k.



### **1.3.11 Xóa phần tử có giá tri x.**

 ta tìm x trong danh sách bằng hàm Search và xóa tại vị trí tìm thấy mà ta nhận được.

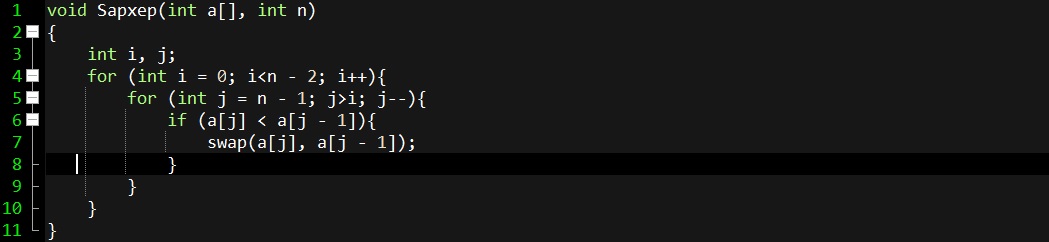


# **2. Các thuật toán sắp xếp.**

## **2.1 Sắp xếp nổi bọt (bubble sort)**

Thuật toán sắp xếp nổi bọt thực hiện sắp xếp dãy số bằng cách lặp lại công việc đổi chỗ 2 số liên tiếp nhau nếu chúng đứng sai thứ tự(số sau bé hơn số trước với trường hợp sắp xếp tăng dần) cho đến khi dãy số được sắp xếp.

Ví dụ:



Ví dụ minh họa:

Giả sử chúng ta cần sắp xếp dãy số [5 1 4 2 8] này tăng dần.  
**Lần lặp đầu tiên:**  
( **5** **1** 4 2 8 ) –> ( **1** **5** 4 2 8 ), Ở đây, thuật toán sẽ so sánh hai phần tử đầu tiên, và đổi chỗ cho nhau do 5 > 1.  
( 1 **5** **4** 2 8 ) –>  ( 1 **4** **5** 2 8 ), Đổi chỗ do 5 > 4  
( 1 4 **5** **2** 8 ) –>  ( 1 4 **2** **5** 8 ), Đổi chỗ do 5 > 2  
( 1 4 2 **5** **8** ) –> ( 1 4 2 **5** **8** ), Ở đây, hai phần tử đang xét đã đúng thứ tự (8 > 5), vậy ta không cần đổi chỗ.

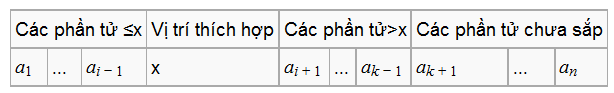
**Lần lặp thứ 2:**  
( **1** **4** 2 5 8 ) –> ( **1** **4** 2 5 8 )  
( 1 **4** **2** 5 8 ) –> ( 1 **2** **4** 5 8 ), Đổi chỗ do 4 > 2  
( 1 2 **4** **5** 8 ) –> ( 1 2 **4** **5** 8 )  
( 1 2 4 **5** **8** ) –>  ( 1 2 4 **5** **8** )  
Bây giờ, dãy số đã được sắp xếp, Nhưng thuật toán của chúng ta không nhận ra điều đó ngay được. Thuật toán sẽ cần thêm một lần lặp nữa để kết luận dãy đã sắp xếp khi và khi khi nó đi từ đầu tới cuối mà không có bất kỳ lần đổi chỗ nào được thực hiện.

**Lần lặp thứ 3:**  
( **1** **2** 4 5 8 ) –> ( **1** **2** 4 5 8 )  
( 1 **2** **4** 5 8 ) –> ( 1 **2** **4** 5 8 )  
( 1 2 **4** **5** 8 ) –> ( 1 2 **4** **5** 8 )  
( 1 2 4 **5** **8** ) –> ( 1 2 4 **5** **8** )

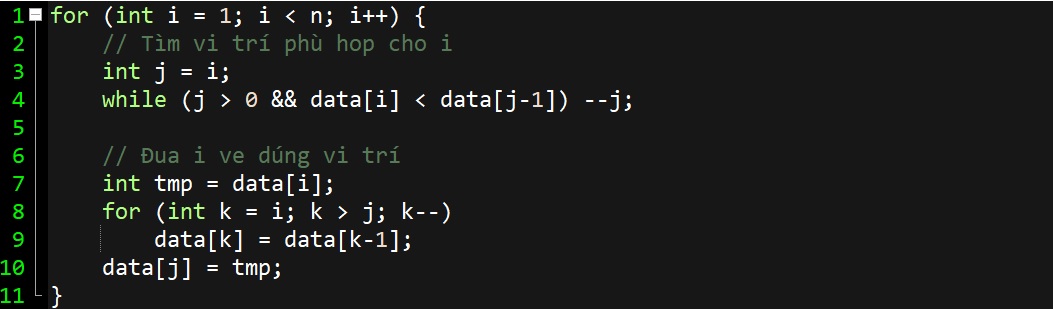
**2.2 Sắp xếp chèn.**

Sắp xếp chèn (insertion sort) là một thuật toán sắp xếp bắt chước cách sắp xếp quân bài của những người chơi bài. Muốn sắp một bộ bài theo trật tự người chơi bài rút lần lượt từ quân thứ 2, so với các quân đứng trước nó để chèn vào vị trí thích hợp.

Cơ sở lập luận của sắp xếp chèn có thể mô tả như sau: Xét danh sách con gồm k phần tử đầu a1,...,ak. Với k = 1, danh sách gồm một phần tử đã được sắp. Giả sử trong danh sách k-1 phần tử đầu a1,...,ak − 1 đã được sắp. Để sắp xếp phần tử ak = x ta tìm vị trí thích hợp của nó trong dãy a1,...,ak − 1. Vị trí thích hợp đó là đứng trước phần tử lớn hơn nó và sau phần tử nhỏ hơn hoặc bằng nó.



Ví dụ:



Ví dụ minh họa:

Cho danh sách

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 3 | 7 | - | 6 | 4 | 2 | 5 |

Danh sách con gồm 3 phần tử bên trái 1,3,7 đã được sắp. Để tiếp tục sắp xếp phần tử thứ tư a4 = 6 vào danh sách con đó, ta tìm vị trí thích hợp của nó là sau 3 và trước 7.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 3 | 6 | 7 | - | 4 | 2 | 5 |

Làm tiếp theo với a5 = 4 ta được

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 3 | 4 | 6 | 7 | - | 2 | 5 |

Làm tiếp theo với a6 = 2 ta được

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 2 | 3 | 4 | 6 | 7 | - | 5 |

Cuối cùng chèn a7 = 5

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | - |

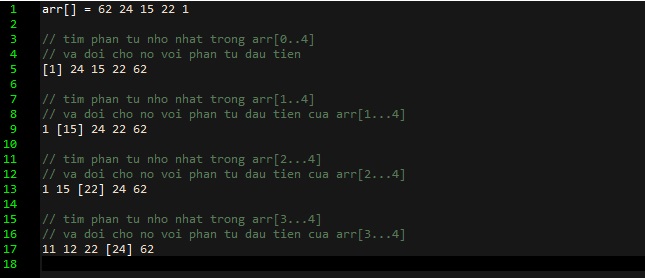
## **2.3 Sắp xếp chọn.**

Thuật toán selection sort sắp xếp một mảng bằng cách đi tìm phần tử có giá trị nhỏ nhất(giả sử với sắp xếp mảng tăng dần) trong đoạn đoạn chưa được sắp xếp và đổi cho phần tử nhỏ nhất đó với phần tử ở đầu đoạn chưa được sắp xếp(không phải đầu mảng). Thuật toán sẽ chia mảng làm 2 mảng con.

1. Một mảng con đã được sắp xếp.
2. Một mảng con chưa được sắp xếp.

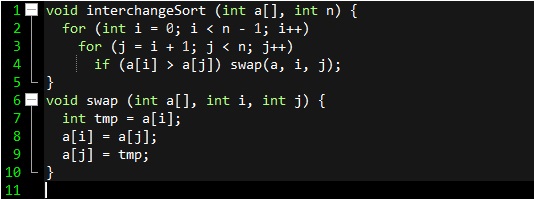
Tại mỗi bước lặp của thuật toán, phần tử nhỏ nhất ở mảng con chưa được sắp xếp sẽ được di chuyển về đoạn đã sắp xếp.

Ví dụ:



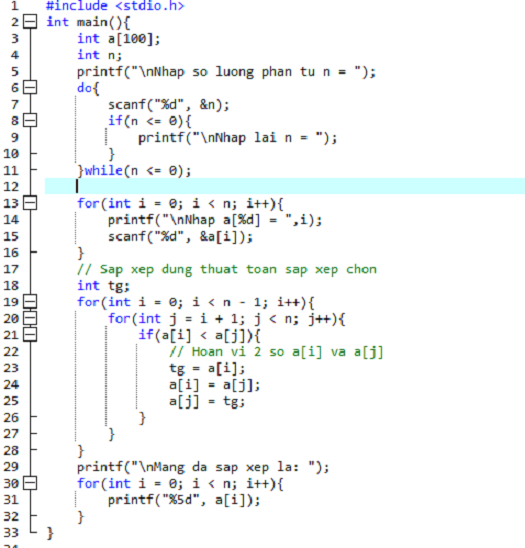
## **2.4 Sắp xếp đổi chỗ.**

Xuất phát từ đầu dãy, lần lượt so sánh phần tử đầu dãy với các phần tử còn lại, nếu thấy lớn hơn thì đổi chỗ cho nhau, mục đích là để sau khi quét một lượt, phần tử bé nhất sẽ về đầu dãy.

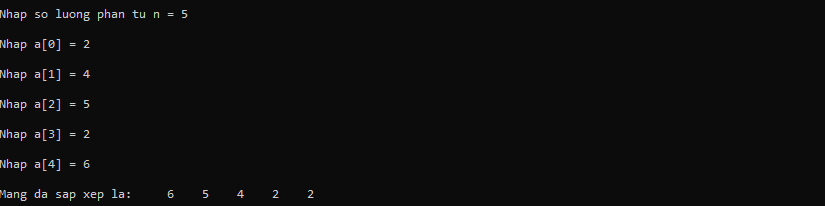


## **2.5 Sắp xếp dãy số giảm dần**

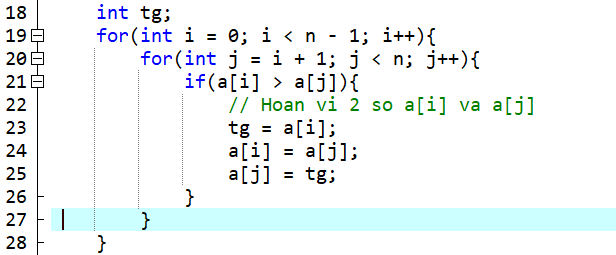
Việc sắp xếp dãy số giảm dần chỉ khác sắp xếp tăng dần duy nhất ở bước kiểm tra điều kiện để hoán vị.



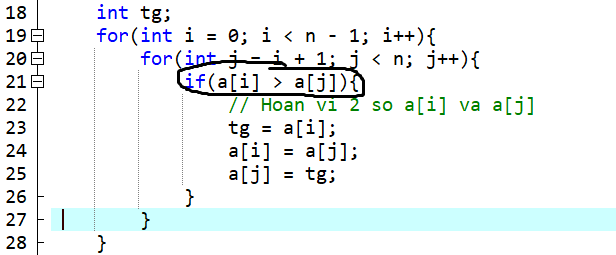
Kết quả chạy chương trình:



## **2.6 Sắp xếp dãy dố tăng dần.**



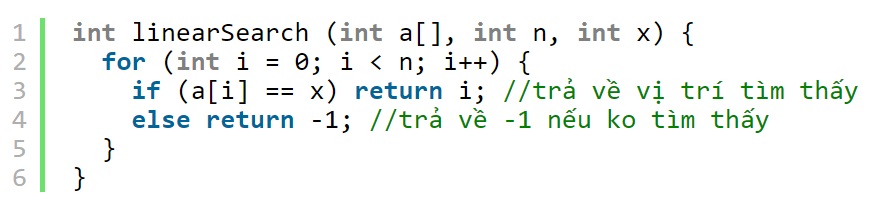
Giống như ở phần sắp xếp dãy số giảm dần ta chỉ việc thay a[i]<a[j] thành a[i]>a[j] như hình dưới là ta có ngay hàm sắp xếp giảm dần.



# **3. Các thuật toán tìm kiếm.**

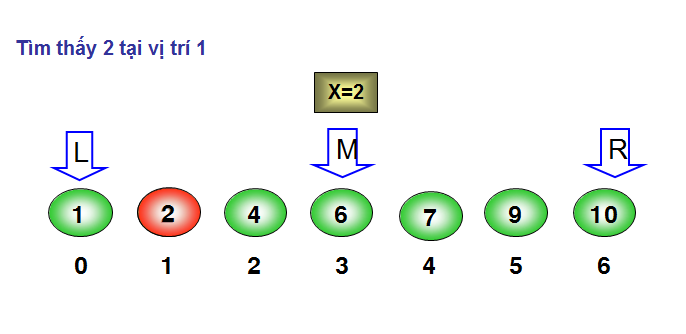
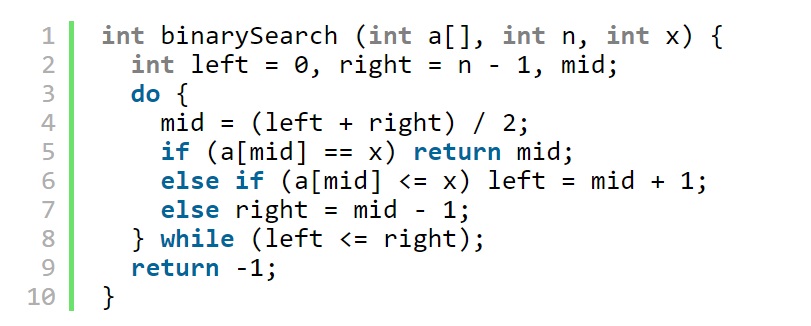
## **3.1 Tìm kiếm tuyến tính.**

Là kiểm tra tuần tự từng phần tử của mảng, đến khi nào giống thì thôi.



## **3.2 Tìm kiếm nhị phân.**

1. Xét đoạn mảng arr[left…right] cần tìm kiếm phần tử x. Ta so sánh x với phần tử ở vị trí giữa của mảng(mid = (left + right)/2). Nếu:
2. Nếu phần tử arr[mid] = x. Kết luận và thoát chương trình.
3. Nếu arr[mid] < x. Chỉ thực hiện tìm kiếm trên đoạn arr[mid+1…right] (nữa bên phải của dãy sô).
4. Nếu arr[mid] > x. Chỉ thực hiện tìm kiếm trên đoạn arr[left…mid-1] (nữa bên trái của dãy số).



# **4. Các thao tác với file làm việc với tệp.**

* 1. **Làm việc với tệp.**

Trước khi bạn làm việc với file, bạn nên biết về 2 kiểu file khác nhau sau đây:

* File văn bản – text files
* File nhị phân – binary file

1. **File văn bản – text files.**

File văn bản là file thường có đuôi là .txt. Những file này bạn có thể dễ dàng tạo ra bằng cách dùng các text editer thông dụng như Notepad, Notepad++, Sublime Text,....

Khi bạn mở các file này bằng các text editer nói trên, bạn sẽ thấy được văn bản ngay và có thể dễ dàng thao tác sửa, xóa, thêm nội dung của file này.

Kiểu file này thuận tiện cho chúng ta trong việc sử dụng hàng ngày, nhưng nó sẽ kém bảo mật và cần nhiều bộ nhớ để lưu trữ hơn.

1. **File nhị phân – Binary files.**

File nhị phân thường có đuôi mở rộng là **.bin**

Thay vì lưu trữ dưới dạng văn bản thuần thúy, các file này được lưu dưới dạng nhị phân, chỉ bao gồm các số 0 và 1. Bạn cũng sẽ thấy các con số này nếu cố mở nó bằng 1 text editer kể trên.

Loại file này giúp lưu trữ được dữ liệu với kích thước lớn hơn, không thể đọc bằng các text editer thông thường và thông tin lưu trữ ở loại file được bảo mật hơn so với file văn bản.

* 1. **Các thao tác với file**

Trong ngôn ngữ lập trình C, có một số thao tác chính khi làm việc với file, bao gồm cả file văn bản và file nhị phân:

* Tạo mới một file
* Mở một file đã có
* Đóng file đang mở
* Đọc thông tin từ file/ Ghi thông tin ra file

### **4.2.1 Khai báo sử dụng FILE.**

Khi làm việc với file, bạn cần khai báo 1 con trỏ kiểu FILE. Việc khai báo này là cần thiết để có sự kết nối giữa chương trình của bạn và tập tin mà bạn cần thao tác.

Để khai báo sử dụng tệp, ta dùng lệnh sau :

FILE biến\_con\_trỏ\_tệp;

Trong đó biến\_con\_trỏ\_tệp có thể là biến đơn hay một danh sách các biến phân cách nhau bởi dấu phảy ( dấu **,**).



* + 1. **Thao tác mở file – hàm fopen.**

**Cấu trúc ngữ pháp của hàm**



**Nguyên hàm trong: stdio.h**

Trong dó: đối thứ nhất là tên tệp, đối thứ hai là kiểu truy cập.

**Công dụng**

Hàm dùng để mở tệp. Nếu thành công hàm cho con trỏ kiểu FILE ứng với tệp vừa mở. Các hàm cấp hai sẽ làm việc với tệp thông qua con trỏ này. Nếu có lỗi hàm sẽ trả về giá trị NULL.

Bảng chỉ ra các giá trị của kiểu:

Các giá trị của kiểu

| **Kiểu** | **Ý nghĩa** | **Nếu file không tồn tại** |
| --- | --- | --- |
| r | Mở file chỉ cho phép đọc | Nếu file không tồn tại, fopen() trả về NULL. |
| rb | Mở file chỉ cho phép đọc dưới dạng nhị phân. | Nếu file không tồn tại, fopen() trả về NULL. |
| w | Mở file chỉ cho phép ghi. | Nếu file đã tồn tại, nội dung sẽ bị ghi đè. Nếu file không tồn tại, nó sẽ được tạo tự động. |
| wb | Open for writing in binary mode. | Nếu file đã tồn tại, nội dung sẽ bị ghi đè. Nếu file không tồn tại, nó sẽ được tạo tự động. |
| a | Mở file ở chế độ ghi “append”. Tức là sẽ ghi vào cuối của nội dung đã có. | Nếu file không tồn tại, nó sẽ được tạo tự động. |
| ab | Mở file ở chế độ ghi nhị phân “append”. Tức là sẽ ghi vào cuối của nội dung đã có. | Nếu file không tồn tại, nó sẽ được tạo tự động. |
| r+ | Mở file cho phép cả đọc và ghi. | Nếu file không tồn tại, fopen() trả về NULL. |
| rb+ | Mở file cho phép cả đọc và ghi ở dạng nhị phân. | Nếu file không tồn tại, fopen() trả về NULL. |
| w+ | Mở file cho phép cả đọc và ghi. | Nếu file đã tồn tại, nội dung sẽ bị ghi đè. Nếu file không tồn tại, nó sẽ được tạo tự động. |
| wb+ | Mở file cho phép cả đọc và ghi ở dạng nhị phân. | Nếu file đã tồn tại, nội dung sẽ bị ghi đè. Nếu file không tồn tại, nó sẽ được tạo tự động. |
| a+ | Mở file cho phép đọc và ghi “append”. | Nếu file không tồn tại, nó sẽ được tạo tự động. |
| ab+ | Mở file cho phép đọc và ghi “append” ở dạng nhị phân. | Nếu file không tồn tại, nó sẽ được tạo tự động. |



* + 1. **Thao tác đóng file – hàm fclose.**

**Cấu trúc ngữ pháp của hàm:**



**Nguyên hàm trong: stdio.h**

Trong đó: fp là con trỏ ứng với tệp cần đóng.

**Công dụng:**

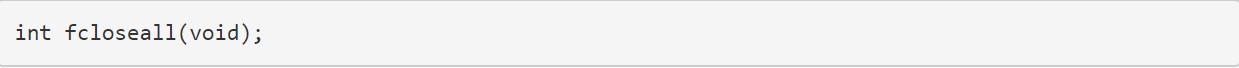
Hàm dùng để đóng tệp khi kết thúc các thao tác trên nó. Khi đóng tệp, máy thực hiện các công việc sau :

* Khi đang ghi dữ liệu thì máy sẽ đẩy dữ liệu còn trong vùng đệm lên đĩa
* Khi đang đọc dữ liệu thì máy sẽ xoá vùng đệm
* Giải phóng biến trỏ tệp.
* Nếu lệnh thành công, hàm sẽ cho giá trị 0, trái lại nó cho hàm EOF.



* + 1. **Đóng tất cả các hàm đang mở - hàm fcloseall.**

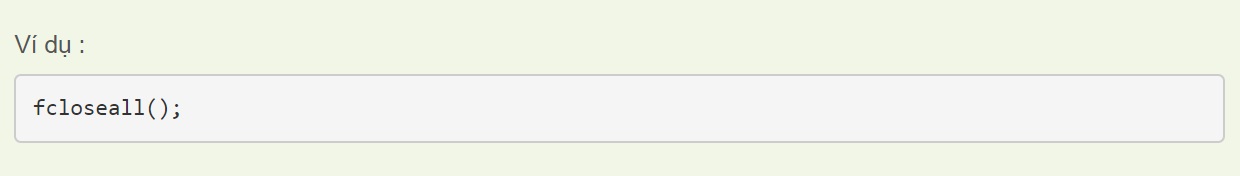
**Cấu trúc ngữ pháp của hàm**



**Nguyên hàm trong: stdio.h**

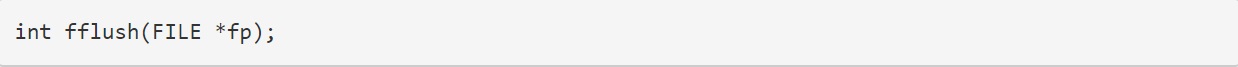
**Công dụng**

Hàm dùng để đóng tất cả các tệp đang mở. Nếu lệnh thành công, hàm sẽ cho giá trị bằng số là số tệp được đóng, trái lại nó cho hàm EOF.



*Làm sạch vùng đệm – hàm fflush*

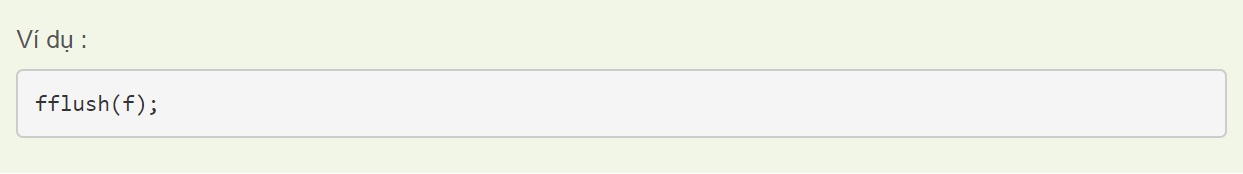
**Cấu trúc ngữ pháp của hàm**



**Nguyên hàm trong: stdio.h**

**Công dụng**

Dùng làm sạch vùng đệm của tệp fp. Nếu thành công, hàm sẽ trả về giá trị 0, trái lại nó cho hàm EOF.



*Làm sạch vùng đệm của các tệp đang mở - hàm fflushall.*

**Cấu trúc ngữ pháp của hàm**



**Nguyên hàm trong: stdio.h**

**Công dụng**

Làm sạch vùng đệm của tất cả các tệp đang mở. Nếu thành công, hàm sẽ cho giá trị bằng số các tệp đang mở, trái lại nó cho hàm EOF.

Ví dụ: fflushall();

*Kiểm tra lỗi file – hàm ferror*

**Cấu trúc ngữ pháp của hàm**



**Nguyên hàm trong: stdio.h**

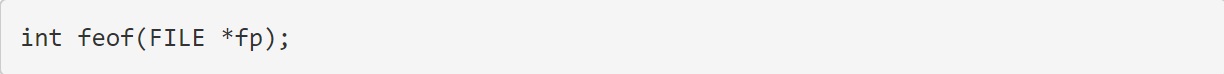
Trong đó fp là con trỏ tệp.

**Công dụng**

Hàm dùng để kiểm tra lỗi khi thao tác trên tệp fp. Hàm cho giá trị không nếu không có lỗi, trái lại hàm cho ra giá trị khác 0.

*Kiểm tra cuối tệp – hàm feof*

**Cấu trúc ngữ pháp của hàm**



**Nguyên hàm trong : stdio.h .**

Trong đó fp là con trỏ tệp.

**Công dụng**

Hàm dùng để kiểm tra cuối tệp. Hàm cho giá trị khác 0 nếu gặp cuối tệp khi đọc, trái lại hàm cho giá trị 0.

Truy nhập ngẫu nhiên - các hàm di chuyên con trỏ chỉ vị

**4.2.5 Đọc/Ghi file trong văn bản C.**

Các hàm chỉ dùng cho tệp văn bản:

fprintf: dùng để ghi dữ liệu theo khuông dạng lên tệp.

fscanf: dùng để đọc dữ liệu từ tệp theo khuông dạng.

fputs: dùng để ghi một chuỗi ký tự lên tệp.

fgets: dùng để đọc một dãy ký tự từ tệp.

**Hàm fprintf**

* Dạng hàm:

int fprintf(FILE \*fp, const char \*dk, bt);

* Công dụng: Giá trị các biểu thức bt được ghi lên tệp fp theo khuông dạng xác định trong mỗi chuỗi điểu khiển dk. Nếu thành công, hàm tra về một giá trị nguyên bằng số byte ghi lên tệp. Khi có lỗi hàm cho EOF. Hàm làm việc giống như printf.
* Đối:

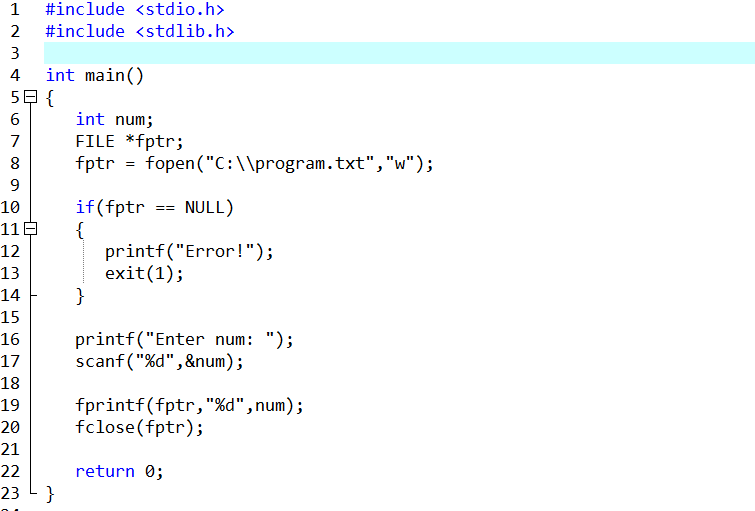
fp là con trỏ tệp,

dk chưa địa chỉ của chuỗi điều khiển,

bt là danh sách các biểu thức mà giá trị của chúng cần ghi lên tệp.

Ví dụ:

Ghi file sử dụng fprintf()



Chương trình nhận số **num** từ bàn phím và ghi vào file văn bản *program.txt.*

Sau khi bạn chạy chương trình này, bạn sẽ thấy file văn bản *program.txt* được tạo mới trong ổ C trên máy tính bạn. Khi mở file này lên, bạn sẽ thấy số mà bạn vừa nhập cho biến **num** kia.

**Hàm fscanf**

* Dạng hàm:

int fscanf(FILE \*fp, const char \*dk,…);

* Công dụng: Đọc dữ liệu từ tệp fp, biến đổi theo khuôn dạng (đặc tả) trong dk và lưu kết quả vào các đối. Hàm làm việc giống như scanf. Hàm trả về một giá trị bằng sô trường được đọc.
* Đối:

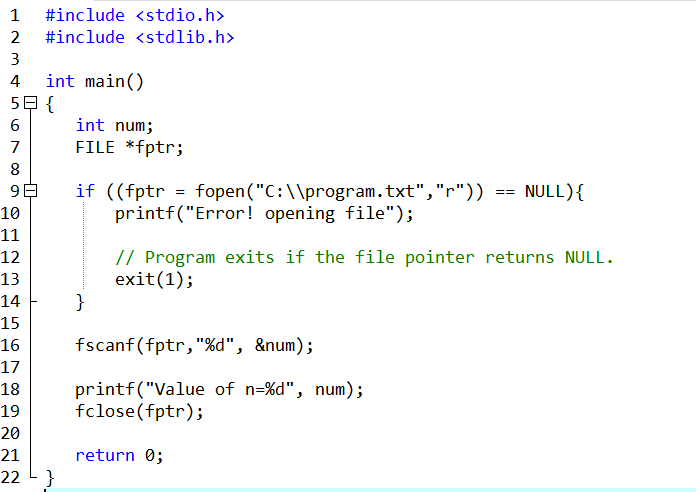
fp là con trỏ,

dk chứa địa chỉ của chuỗi điều khiển,

… là danh sách địa chỉ các đối chứa kết quả đọc được từ tệp.

Chuỗi điều khiển và danh sách đối có cùng ý nghĩa như nhau trong hàm scanf.

Ví dụ:



Chương trình ở ví dụ này sẽ đọc giá trị số được lưu trong file *program.txt* mà chương trình ở VD1 vừa tạo ra và in lên màn hình.

**Hàm fputs**

* Dạng hàm:

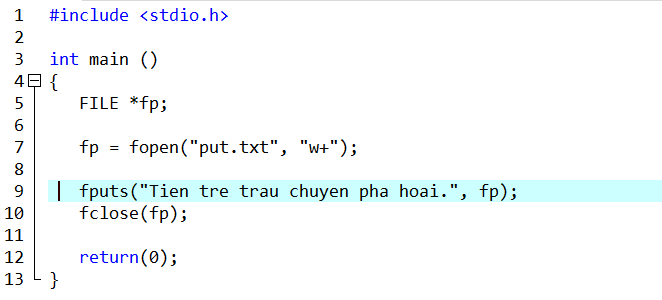
int fputs(const char \*s,FILE \*fp);

* Công dụng: Ghi chuỗi s lên tệp fp (dấu ‘\0’ không ghi lên tệp). Khi thành công, hàm trả về ký tự cuối cùng được ghi lên tệp. Khi có lỗi hàm cho EOF.
* Đối:

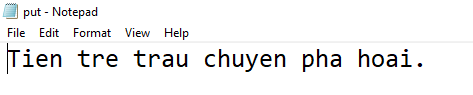
s là con trỏ tới địa chỉ đầu của một chuỗi ký tự kết thúc bằng dấu’\0’

fp là con trỏ tệp.

ví dụ:



Và đây là kết quả ra như vầy:



**Hàm fgets**

* Dạng hàm:

char \*fgets(char \*s, int n, FILE \*fp);

* Công dụng: Đọc mỗi dãy ký tự từ tệp fp chứa vào vùng nhớ s. việc đọc kết thúc khi:
* Hoặc đã đọc n-1 ký tự.
* Hoặc gặp dấu xuống dòng (cặp mã 13 10). Khi đó được đưa vào xâu kết quả.
* Hoặc kết thúc tệp.

Xâu kết quả sẽ được bổ sung thêm dấu hiệu kết thúc chuỗi ‘\0’. Khi thành công hàm trả địa chỉ vùng nhận kết quả. Khi có lỗi hoặc gặp cuối tệp, hàm cho gái trị NULL.

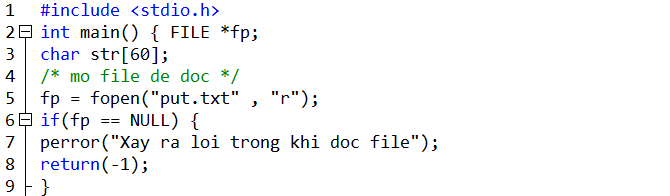
* Đối:

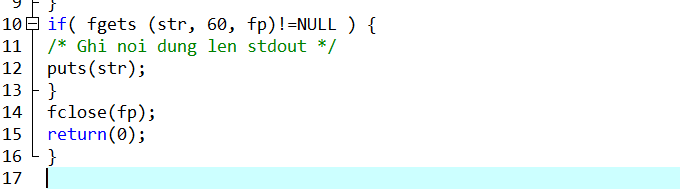
s là con trỏ (kiểu char) trỏ tới một vùng nhớ đủ lớn chứa chuỗi ký tự đọc từ tệp.

n là số nguyên xác định độ dài của dãy cần đọc.

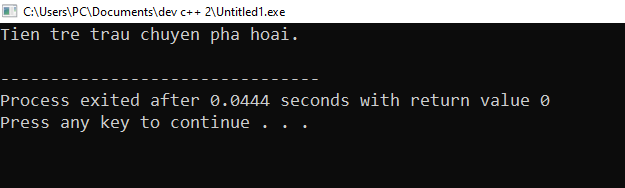
fp là con trỏ tệp.

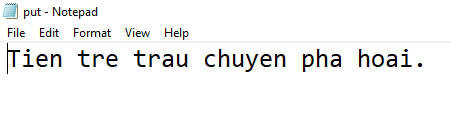
ví dụ:





Và kết quả như thế này:





**4.2.6 Đọc/Ghi file nhị phân trong C.**

**Hàm fwrite**

* Dạng hàm:

 size\_t fwrite(const void \*ptr, size\_t size, size\_t n, FILE \*f)

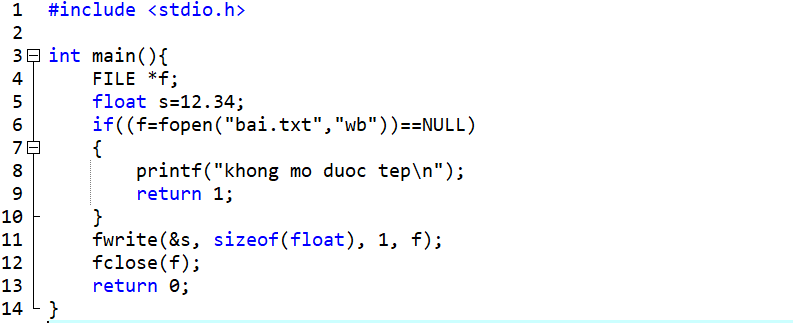
* Công dụng: Dùng để ghi một số mẫu tin lên tệp.
* Đối:

ptr: con trỏ chỉ đến vùng nhớ chứa thông tin cần ghi lên tập tin.

n: số phần tử sẽ ghi lên tập tin.

size: kích thước của mỗi phần tử.

f: con trỏ tập tin đã được mở.



**Hàm fread**

* Dạng hàm:

size\_t fread(const void \*ptr, size\_t size, size\_t n, FILE \*f)

* Công dụng: Dùng để đọc một số mẫu tin từ tệp.
* Đối:

ptr: con trỏ chỉ đến vùng nhớ sẽ nhận dữ liệu từ tập tin.

n: số phần tử được đọc từ tập tin.

size: kích thước của mỗi phần tử.

f: con trỏ tập tin đã được

**Phần 2: Giải thích bài tập lớn.**

* 1. **Lý do chọn đề tài.**

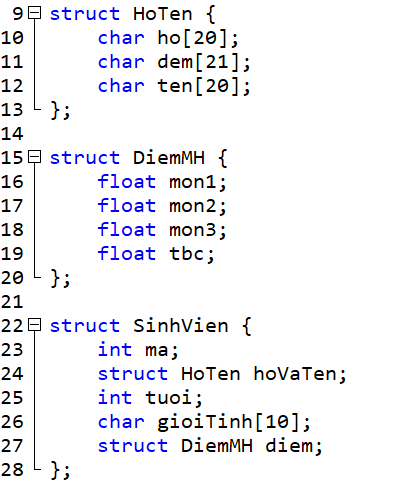
Nhằm giúp tiết kiệm thời gian trong việc quản lý sinh viên. Giúp người quản lý có thể theo dõi và quản lý sinh viên dễ dàng xử lý số liệu chính xác nhanh gọn. Do đó nhóm em lên ý tưởng làm chương trình quản lý sinh viên.

* 1. **Giải thích chi tiết bài tập.**

1. **Mô tả công việc cần làm.**

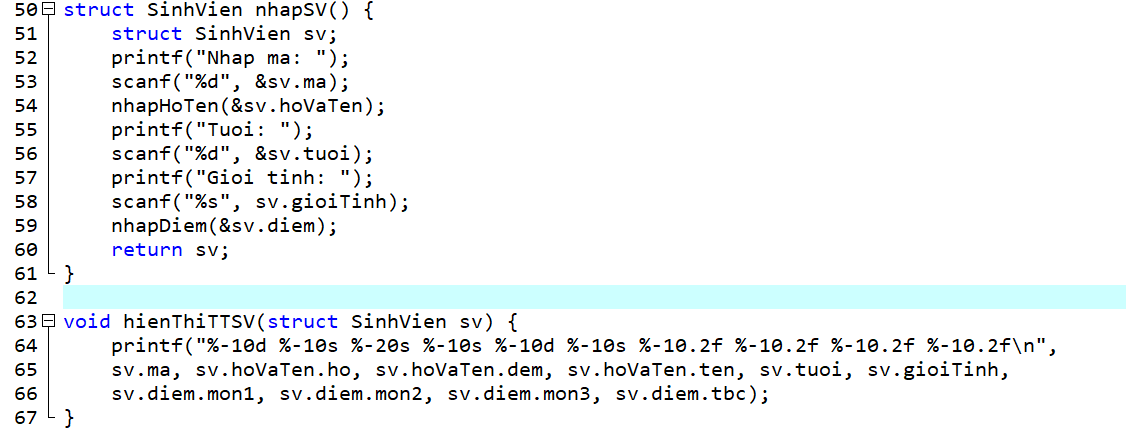
* Ghi thông tin sinh viên có đầy đủ thông tin: mã sinh viên, học và tên, tuổi, giới tính, điểm, ….
* Xét điểm 3 môn môn 1, môn 2, môn 3.
* Sử dụng cấu trúc struct để tạo chương trình.
* Chương trình bao gồm các phần:
* Thêm sinh viên mới vào chương trình theo tên a->z
* Hiển thị danh sách vừa tạo
* Sắp xếp sinh viên theo tên theo thứ tự từ a->z
* Sắp xếp điểm trung bình 3 môn giảm dần
* Tìm và hiển thị sinh viên trong danh sách vừa nhập
* Ghi thông tin sinh viên vào file .txt
* Tạo MENU để thực hiện các chức năng trên

1. **Thực hiện các công việc cần làm.**
2. **Tạo cấu trúc sinh viên.**



Tạo một cấu trúc sinh viên để khai báo các phần tử cần dùng trong bài.

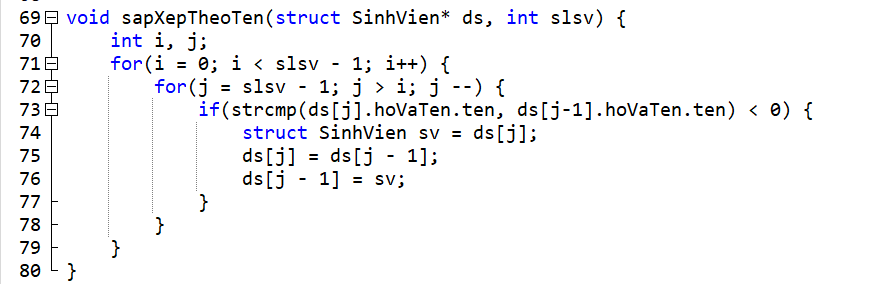
1. **Nhập tên sinh viên và in ra màn hình.**



Trong hàm nhập và xuất ta thực hiện như bình thường. Ta sử dụng toán tử “.” để chỉ tới địa chỉ biến cần dùng.

VD: &sv.tuoi → ta sử dụng biến sv để thực hiện công việc chỉ địa chỉ của “tuoi” để nhập giá trị tuổi của sinh viên.

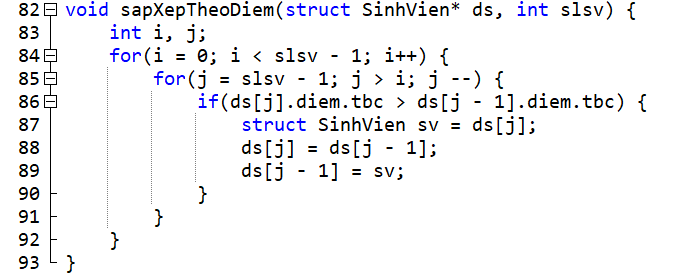
1. **Sắp xếp sinh viên theo tên.**



Với hàm này ta sử dụng hàm sắp xếp đổi chỗ để sắp xếp sinh viên theo tên.

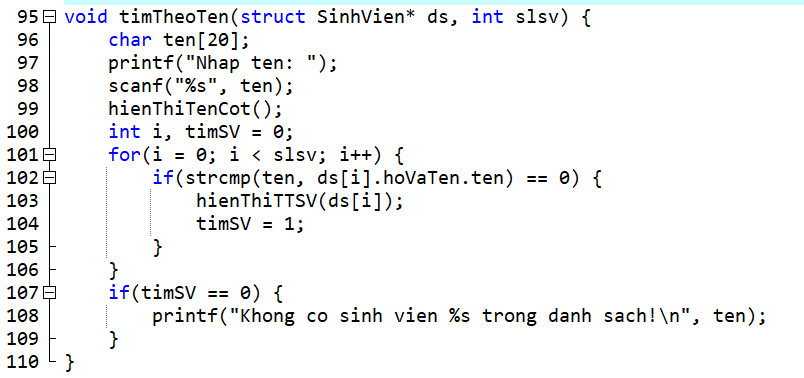
Ta sử dụng hàm strcmp() để so sánh tên sinh viên.

1. **Sắp xếp điểm trung bình 3 môn học.**



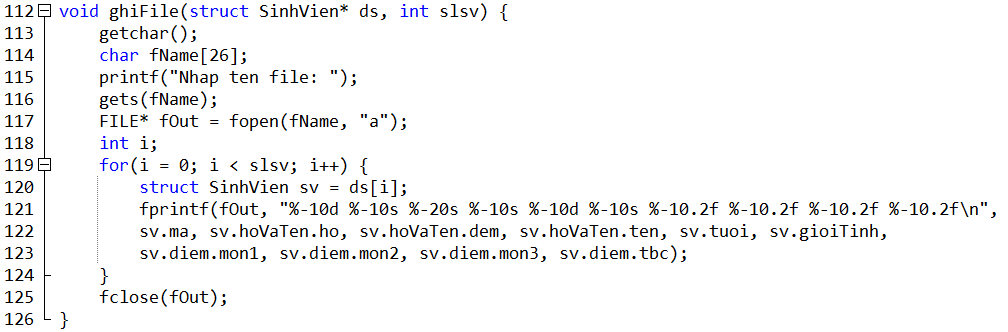
Ta tiếp tục sử dụng hàm sắp xếp như ở hàm sắp xếp theo tên sinh viên.

1. **Tìm kiếm và hiển thị sinh viên trong danh sách.**

Ta sử dụng hàm strcmp() để so sánh chuỗi mà được trỏ tới điển hình là tên sinh viên. Nếu giá trị mà strcmp trả về bằng 0 có nghĩa là tên sinh viên nhập vào bằng với sinh viên có trong danh sách, nếu trả về giá trị khac không thì tên không tồn tại trong danh sách.

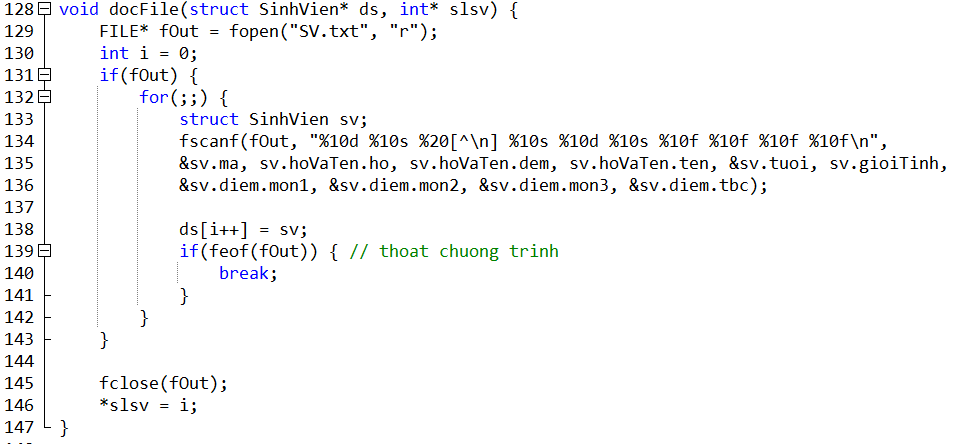
1. **Đọc/Ghi thông tin sinh viên vào tệp txt.**

Ghi file:



Đối với hàm ghi file ta sử dụng câu lệnh fopen(fName, “a”); mở tệp cần phải ghi sau đó ta sử dụng hàm fprintf() để có thể ghi thông tin trong danh sách vào file vừa tạo.

Đọc file:



Hàm đọc file này có chức năng lấy các giá trị trong danh sách vừa tạo để xuất ra màn hình khi vừa bắt đầu chạy chương trình.

Và nếu file vừa ghi không phải file SV như trong hàm đọc file đã ghi thì nó sẽ tự động hủy không xuất ra màn hình những thông tin trong file SV vừa nhập.

**~~~~~~~ HẾT ~~~~~~~**