**TRƯỜNG ĐẠI HỌC GIAO THÔNG VẬN TẢI**

**PHÂN HIỆU TẠI TP. HỒ CHÍ MINH**

**BỘ MÔN CÔNG NGHỆ THÔNG TIN**



**BÁO CÁO MÔN HỌC**

**LẬP TRÌNH NÂNG CAO**

**ĐỀ TÀI: CHƯƠNG TRÌNH QUẢN LÝ SINH VIÊN**

**Sinh viên thực hiện:**

1. NGUYỄN MINH PHÁT – MSV: 6051071088

2. NGUYỄN BẢO TRUNG – MSV: 60510710

3. LÊ THANH DUẨN – MSV: 60510710

Lớp: CÔNG NGHỆ THÔNG TIN K60

Khoá:K60

Tp. Hồ Chí Minh, năm 2020

**Mục Lục:**

[**Lời mở đầu:** 1](#_Toc44357051)

[**PHẦN LÝ THUYẾT:** 2](#_Toc44357052)

[**I. Làm việc với tệp** 2](#_Toc44357053)

[**1. Tại sao sao chúng ta cần đến tệp (file)** 2](#_Toc44357054)

[**2. Các thao tác với file** 2](#_Toc44357055)

[**3. Thao tác đóng file** 3](#_Toc44357056)

[**4. Đọc/Ghi file văn bản trong C** 4](#_Toc44357057)

[**5. Đọc/ghi file nhị phân trong C** 5](#_Toc44357058)

[**II. Cấu trúc liên kết đơn** 7](#_Toc44357059)

[**1. Khai báo linked list** 9](#_Toc44357060)

[**2. Tạo mới một Node** 9](#_Toc44357061)

[**3. Thêm Node vào danh sách liên kết** 10](#_Toc44357062)

[**4. Xóa Node khỏi danh sách liên kết** 12](#_Toc44357063)

[**5. Lấy giá trị ở vị trí bất kỳ** 13](#_Toc44357064)

[**6. Tìm kiếm trong danh sách liên kết** 14](#_Toc44357065)

[**7. Duyệt danh sách liên kết** 14](#_Toc44357066)

[**8. Một số hàm bổ trợ khác** 15](#_Toc44357067)

[**III. Các thuật toán sắp xếp** 16](#_Toc44357068)

[**1. Sắp xếp chọn** 16](#_Toc44357069)

[**2. Sắp xếp chèn** 17](#_Toc44357070)

[**3. Sắp xếp nổi bọt (BubbleSort)** 18](#_Toc44357071)

[**IV. Các thuật toán tìm kiếm** 19](#_Toc44357072)

[**1. Thuật toán tìm kiếm tuyến tính** 20](#_Toc44357073)

[**2. Thuật toán tim kiếm nhị phân** 20](#_Toc44357074)

[**3. Thuật toán tìm kiếm nội suy** 22](#_Toc44357075)

[**B. CHƯƠNG TRÌNH QUẢN LÝ SINH VIÊN** 23](#_Toc44357076)

[**I. Chương trình quản lý sinh viên gồm:** 23](#_Toc44357077)

[**II. Kiến thức cần có:** 23](#_Toc44357078)

[**III. Giải thích code:** 23](#_Toc44357079)

[**1 .Khai báo Thư viện** 23](#_Toc44357080)

[**2. Câu lệnh Struct:** 23](#_Toc44357081)

[**3. Các hàm:** 24](#_Toc44357082)

[**a)** **Hàm nhập bao gồm:** 25](#_Toc44357083)

[**b)** **Nhập n sinh viên:** 26](#_Toc44357084)

[**c)** **Hàm xuất:** 26](#_Toc44357085)

[**d)** **Xuất n sinh viên** 27](#_Toc44357086)

[**e)** **Hàm tính điểm trung binh** 27](#_Toc44357087)

[**f)** **Hàm sắp xếp sinh viên có DTB tăng dần:** 28](#_Toc44357088)

[**g)** **Hàm xếp loại sinh viên** 29](#_Toc44357089)

[**h)** **Hàm tìm tên sinh viên** 29](#_Toc44357090)

[**i)** **Hàm xuất file** 30](#_Toc44357091)

[**j)** **Hàm main (Hàm chính)** 30](#_Toc44357092)

# **Lời mở đầu:**

Trong xã hội ngày càng phát triển hiện nay, khoa học công nghệ là thứ không thể thiếu đối với mỗi quốc gia, doanh nghiệp, trường học hay mỗi cá nhân, đặc biệt là công nghệ thông tin. Với sự phát triển nhanh một cách không ngừng nghĩ như vậy của công nghệ thông tin đã giúp giải quyết các công việc học tập, nguyên cứu, quản lý thông tin,… một cách dễ dàng và tiện lợi. Thấy được tiềm năng đó các quốc gia, doanh nghiệp, trường học, các cá nhân, … đã ứng dụng nó vào thực tiển cuộc sống để giải quyết công việc, học tập, giải trí.

Trong những năm gần đây nhu cầu về các chương trình quản lý càng nhiều hơn do nhu cầu ngày càng cao và thuận tiện cho người sử dụng. Công tác quản lí( kết quả học tập) của sinh viên đóng vai trò rất quan trọng đối với mọi hoạt động của một khoa trong các trường đại học và cao đẳng.

Qua việc học môn Lập trình nâng cao của cô Dung đã cung cấp cho chúng em rất nhiều kiến thức bổ ích và có thể ứng dụng trong đời sống. Nhóm em đã vận dụng những kiến thức đã học được ứng dụng viết chuơng trình nhỏ để quản lý sinh viên. Chương trình giúp quản lý sinh viên một cách dễ dàng và thuận tiện hơn.

# **PHẦN LÝ THUYẾT:**

1. **Làm việc với tệp**
2. **Tại sao sao chúng ta cần đến tệp (file)**

Dữ liệu được lưu ở biến của chương trình, và nó sẽ biến mất khi chương trình kết thúc. Sử dụng file để lưu trữ dữ liệu cần thiết để đảm bảo dữ liệu của chúng ta không bị mất ngay cả khi chương trình của chúng ta ngừng chạy.

Nếu chương trình của bạn có đầu vào(input) là lớn, bạn sẽ rất vất vả nếu phải nhập mỗi khi chạy. Thay vào đó, hãy lưu vào file và chương trình của bạn sẽ tự đọc mỗi lần khởi chạy

Dễ dàng sao chép, di chuyển dữ liệu giữa các thiết bị với nhau

1. **[File văn bản – text files](https://nguyenvanhieu.vn/doc-ghi-file-trong-c/" \l "1-file-van-ban-%e2%80%93-text-files" \o "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](https://nguyenvanhieu.vn/doc-ghi-file-trong-c/" \l "2-file-nhi-phan-%e2%80%93-binary-files" \o "2. 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**

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.

FILE \*fptr;

Để đọc ghi file trong C cũng như trong mọi ngôn ngữ lập trình, việc đầu tiên bạn cần làm là mở file mà bạn muốn làm việc. Trong ngôn ngữ lập trình C, chúng ta có thể mở file bằng cách sử dụng hàm  tfopen() rong thư viện stdio.h như sau:

fptr = fopen("fileopen","mode");

|  |  |  |
| --- | --- | --- |
| Mode | 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 | Mở để viết chế độ in 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 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. |

Trong đó mode là một tham số chúng ta cần chỉ định.

### **Thao tác đóng file**

Khi làm việc với tập tin hoàn tất, kể cả là file nhị phân hay file văn bản. Bạn cần đóng file sau khi làm việc với nó xong.

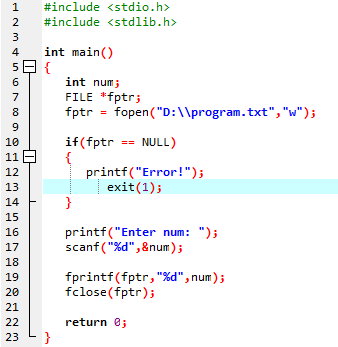
Việc đóng file đang mở có thể được thực hiện bằng cách dùng hàm fclose().

1. **Đọc/Ghi file văn bản trong C**

Chúng ta sẽ học cách đọc ghi file trong C với file văn bản trước. Với file nhị phân, bạn kéo xuống dưới để xem tiếp.

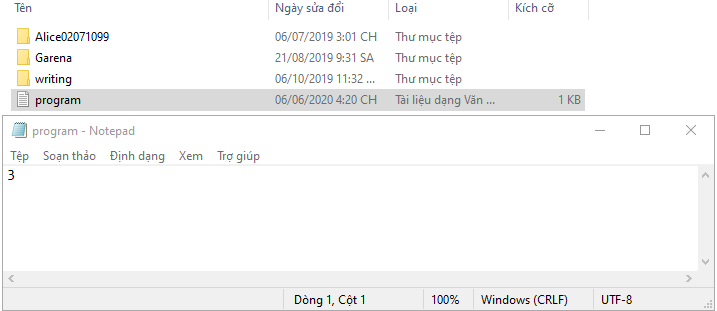
Để làm việc với file văn bản, chúng ta sẽ sử dụng fprintf() và fscanf().

VD1: Ghi file dùng fprintf().

****

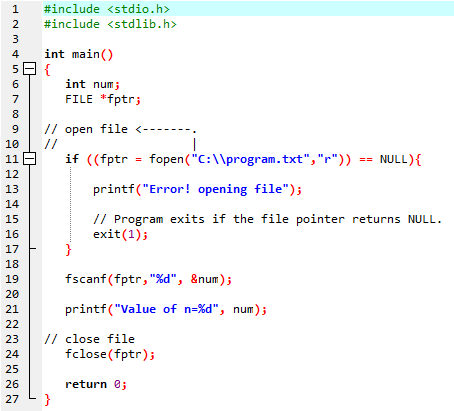
Chương trình nhận số num từ bàn phím và ghi vào file văn bản program.txt.

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.

****

Chương trình 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.

VD2: Đọc sử dụng fscanf().



Chương trình 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.

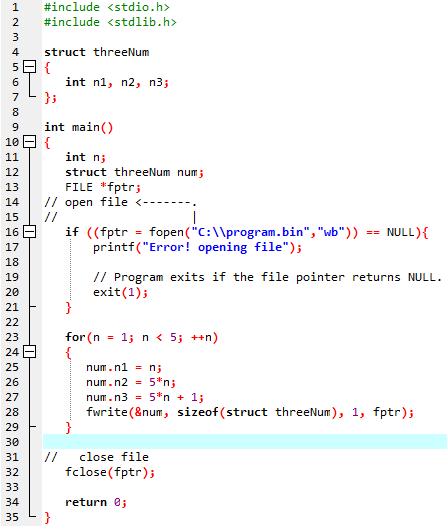
1. **Đọc/ghi file nhị phân trong C**

Các hàm fread() và fwrite() trong C được sử dụng để đọc và ghi file trong C ở dạng nhị phân.

*\* Ghi file nhị phân*

Để ghi file nhị phân, bạn cần sử dụng hàm fwrite(). Hàm này cần 4 tham số: địa chỉ của biến lưu dữ liệu cần ghi, kích thước của biến lưu dữ liệu đó, số lượng kiểu dữ liệu của biến đó và con trỏ FILE trỏ tới file bạn muốn ghi.

fwrite(address\_data,size\_data,numbers\_data,pointer\_to\_file);

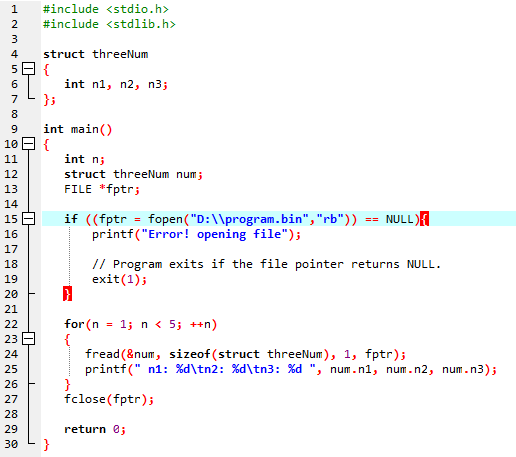


Trong VD3 này, chương trình sẽ tạo ra một file program.bin trên ổ đĩa C của bạn. Chương trình này đã khai báo 1 kiểu dữ liệu cấu trúc lưu 3 giá trị số n1, n2, n3; Và nó được sử dụng trong hàm main có tên biến là num.

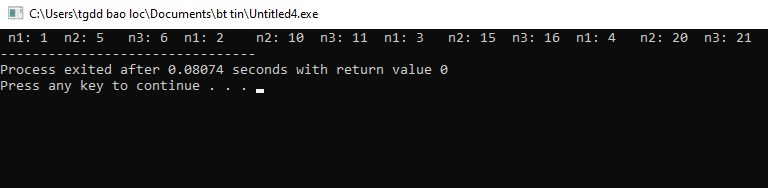
#### \* Đọc file nhị phân

Hàm fread() cũng có 4 tham số tương tự như hàm fwrite() phía trên.

fread(address\_data,size\_data,numbers\_data,pointer\_to\_file);



Trong ví dụ này, bạn đọc file program.bin và lặp qua từng dòng. Bạn sẽ nhận được các giá trị tương ứng khi bạn ghi vào trong VD3.



1. **Cấu trúc liên kết đơn**

Về bản chất, danh sách liên kết có chức năng như một mảng, có thể thêm và xóa các phần tử ở bất kỳ vị trí nào khi cần thiết. Một số sự khác nhau giữa danh sách liên kết và mảng



**Lưu ý:** Ở bảng phía trên, các phần in nghiêng thể hiện đó là ưu điểm so với đối thủ còn lại.

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.

Hình ảnh mô tả cho một Node trong danh sách liên kết đơn:



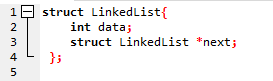
Và đây là hình ảnh mô phỏng một danh sách liên đơn kết đầy đủ:



Cài đặt danh sách liên kết đơn

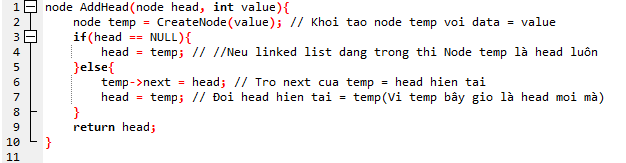
1. **Khai báo linked list**

Để đơn giản hóa, data của chúng ta sẽ là số nguyên(int). Bạn cũng có thể sử dụng các kiểu nguyên thủy khác(float, char,…) hay kiểu dữ liệu struct(SinhVien, CanBo,…) tự tạo.



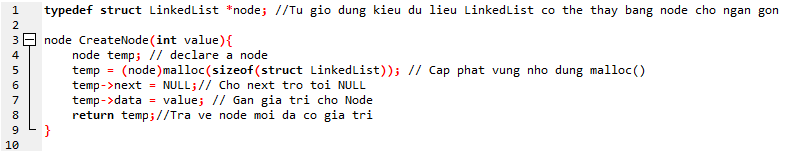
Khai báo trên sẽ được sử dụng cho mọi Node trong linked list. Trường data sẽ lưu giữa giá trị và next sẽ là con trỏ để trỏ đến thằng kế tiếp của nó.

**Tại sao next lại là kiểu LinkedList của chính nó?**Bởi vì nó là con trỏ trỏ của chính bản thân nó, và nó trỏ tới một thằng Node kế tiếp cũng có kiểu LinkedList.



### **Tạo mới một Node**

Hãy tạo một kiểu dữ liệu của struct LinkedList để code clear hơn:



Mỗi một Node khi được khởi tạo, chúng ta cần cấp phát bộ nhớ cho nó, và mặc định cho con trỏ next trỏ tới NULL. Giá trị của Node sẽ được cung cấp khi thêm Node vào linked list.

* **typedef** được dùng để định nghĩa một kiểu dữ liệu trong C. VD: typeder long long LL;
* **malloc** là hàm cấp phát bộ nhớ của C. Với C++ chúng ta dùng new
* **sizeof** là hàm trả về kích thước của kiểu dữ liệu, dùng làm tham số cho hàm malloc

**Lưu ý:** Không giống với mảng, cần khai báo arr[size]. Trong linked list, vì mỗi Node sẽ có con trỏ liên kết đến Node tiếp theo. Do đó, với danh sách liên kết đơn, bạn chỉ cần lưu giữ Node đầu tiên(HEAD). Có head rồi bạn có thể đi tới bất cứ Node nào.

1. **Thêm Node vào danh sách liên kết**
2. **Thêm vào**

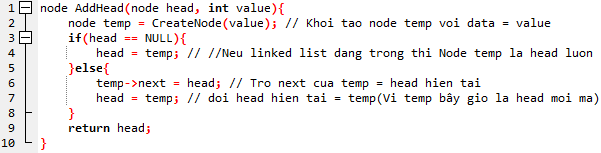
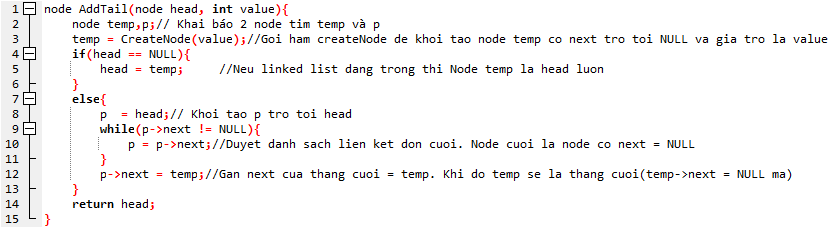
Việc thêm vào đầu chính là việc cập nhật lại thằng head. Ta gọi Node mới(temp), ta có:

* Nếu head đang trỏ tới NULL, nghĩa là linked list đang trống, Node mới thêm vào sẽ làm head luôn
* Ngược lại, ta phải thay thế thằng head cũ bằng head mới. Việc này phải làm theo thứ tự như sau:
  + Cho next của temp trỏ tới head hiện hành
  + Đặt temp làm head mới

#### **Thêm vào cuối**

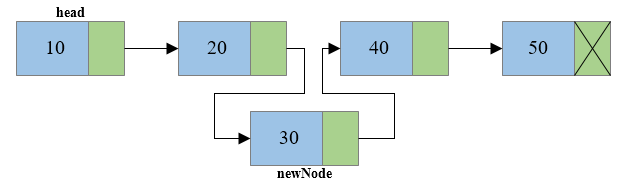
Chúng ta sẽ cần Node đầu tiên, và giá trị muốn thêm. Khi đó, ta sẽ:

1. Tạo một Node mới với giá trị value
2. Nếu head = NULL, tức là danh sách liên kết đang trống. Khi đó Node mới(temp) sẽ là head luôn.
3. Ngược lại, ta sẽ duyệt tới Node cuối cùng(Node có next = NULL), và trỏ next của thằng cuối tới Node mới(temp).



Tổng quan hơn, chúng ta sẽ sẽ viết hàm thêm một Node vào vị trí bất kỳ nhé.

1. **Thêm vào vị trí bất kì**

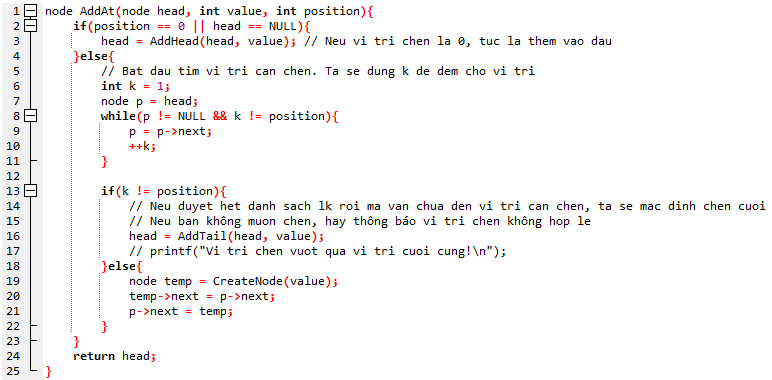


*Thêm Node vào giữa danh sách liên kết*

Để làm được việc này, ta phải duyệt từ đầu để tìm tới vị trí của Node cần chèn, giả sử là Node Q, khi đó ta cần làm theo thứ tự sau:

* Cho next của Node mới trỏ tới Node mà Q đang trỏ tới
* Cho Node Q trỏ tới Node mới

Lưu ý: Chỉ số chèn bắt đầu từ chỉ số 0 nhé các bạn

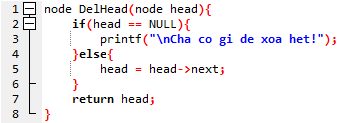


Lưu ý: Bạn phải làm theo thứ tự trên, nếu bạn cho p->next = temp trước. Khi đó, bạn sẽ không thể lấy lại phần sau của danh sách liên kết nữa(Vì next chỉ được được lưu trong p->next mà thay đổi p->next rồi thì còn đâu giá trị cũ).

### **4. Xóa Node khỏi danh sách liên kết**

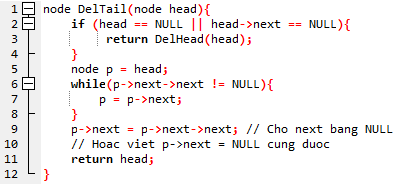
#### **Xóa đầu**

Xóa đầu đơn giản lắm, bây giờ chỉ cần cho thằng kế tiếp của head làm head là được thôi. Mà thằng kế tiếp của head chính là head->next.



1. **Xóa cuối**

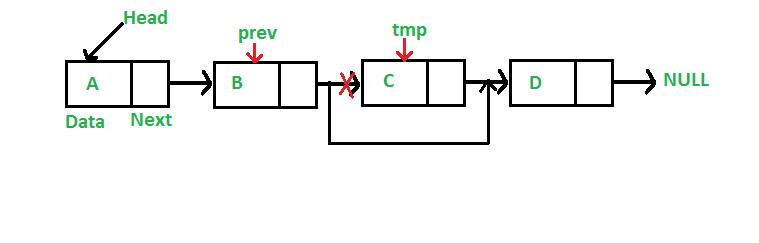
Xóa cuối mới nhọc nè, nhọc ở chỗ phải duyệt đến thằng cuối – 1, cho next của cuối – 1 đó bằng NULL.



Thằng Node cuối – 1 là thằng có p->next->next = NULL. Bạn cho next của nó bằng NULL là xong.

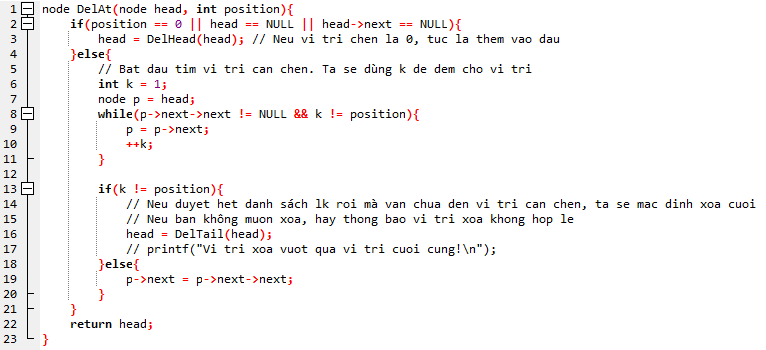
1. **Xóa vị trí bất kì**

Thằng Node cuối – 1 là thằng có p->next->next = NULL. Bạn cho next của nó bằng NULL là xong.



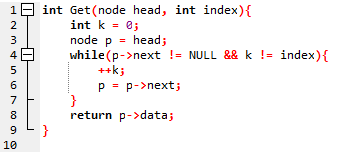
*Xóa node trong danh sách liên kết*

Lưu ý: Chỉ số xóa bắt đầu từ 0 nhé các bạn. Việc tìm vị trí càn xóa chỉ duyệt tới Node gần cuối thôi(cuối – 1). Sau đây là code xóa Node ở vị trí bất kỳ



### **5. Lấy giá trị ở vị trí bất kỳ**

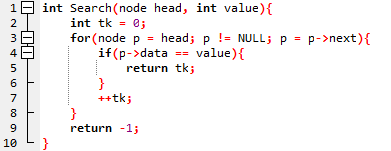
Chúng ta sẽ viết một hàm để truy xuất giá trị ở chỉ số bất kỳ nhé. Trong trường hợp chỉ số vượt quá chiều dài của linked list – 1, hàm này trả về vị trí cuối cùng. Do hạn chế là chúng ta không thể raise error khi chỉ số không hợp lệ. Tôi mặc định chỉ số bạn truyền vào phải là số nguyên không âm. Nếu bạn muốn kiểm tra chỉ số hợp lệ thì nên kiểm tra trước khi gọi hàm này.



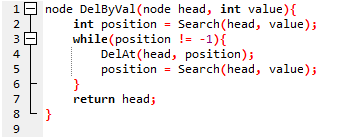
Lý do dùng p->next != NULL là vì chúng ta chỉ muốn đi qua các phần tử có value.

### **Tìm kiếm trong danh sách liên kết**

Hàm tìm kiếm này sẽ trả về chỉ số của Node đầu tiên có giá trị bằng với giá trị cần tìm. Nếu không tìm thấy, chúng ta trả về -1.

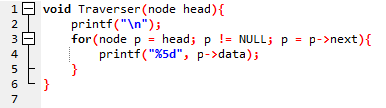


Chúng ta có thể sử dụng hàm này để xóa tất cả các Node trong danh sách liên kết có giá trị chỉ định như sau:



### **Duyệt danh sách liên kết**

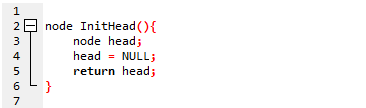
Việc duyệt danh sách liên kết cực đơn giản. Khởi tạo từ Node head, bạn cứ thế đi theo con trỏ next cho tới trước khi Node đó NULL.



### **Một số hàm bổ trợ khác**

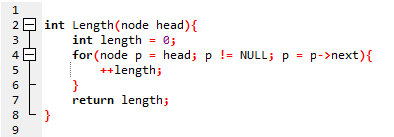
#### **Hàm khởi tạo Node head**

#### Đơn giản là cho con trỏ head = NULL thôi. Nếu bạn để ý, chúng ta vẫn check head = NULL để biết rằng danh sách liên kết chưa có phần tử nào ở các hàm phía trên.

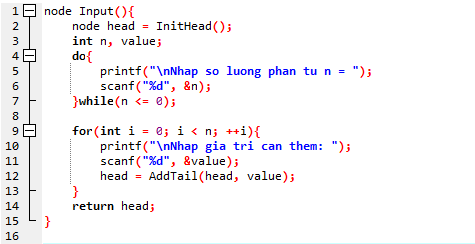


#### **Hàm lấy số phần tử của DSLK**

Duyệt và đếm chừng nào các Node chưa NULL. Sau cùng, trả về giá trị đếm được.



#### **Hàm nhập danh sách liên kết**



1. **Các thuật toán sắp xếp**
2. **Sắp xếp chọn**

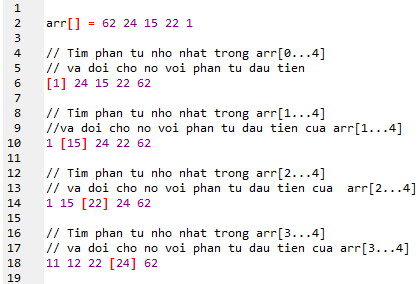
Thuật toán sắp xếp chọn sẽ sắp xếp một mảng bằng cách đi tìm phần tử có giá trị nhỏ nhất(giả sử với sắp xếp mảng tăng dần) trong đoạn đoạn chưa được sắp xếp và đổi cho phần tử nhỏ nhất đó với phần tử ở đầu đoạn chưa được sắp xếp(không phải đầu mảng). Thuật toán sẽ chia mảng làm 2 mảng con

- Một mảng con đã được sắp xếp

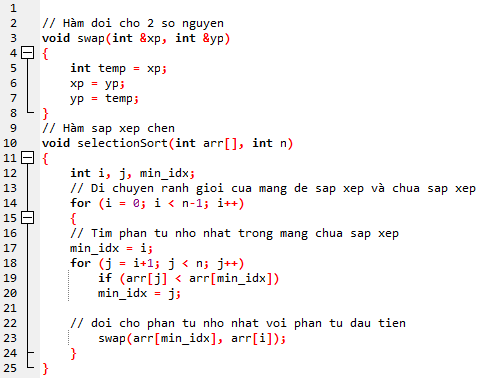
- Một mảng con chưa được sắp xếp

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 minh họa:*



**Code**

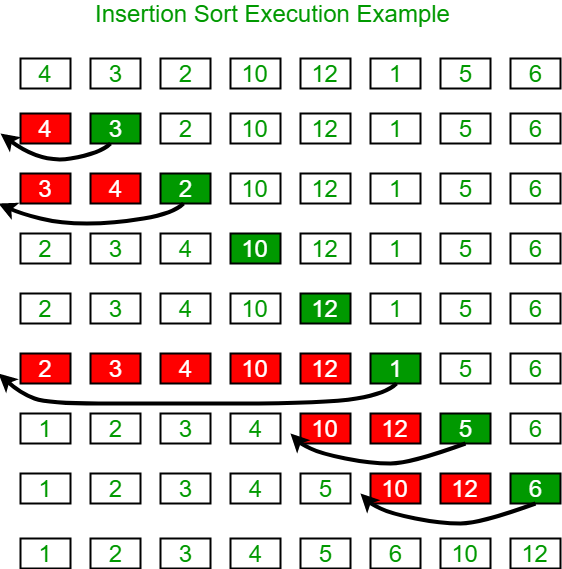


1. **Sắp xếp chèn**

Thuật toán sắp xếp chèn thực hiện sắp xếp dãy số theo cách duyệt từng phần tử và chèn từng phần tử đó vào đúng vị trí trong mảng con(dãy số từ đầu đến phần tử phía trước nó) đã sắp xếp sao cho dãy số trong mảng sắp đã xếp đó vẫn đảm bảo tính chất của một dãy số tăng dần.

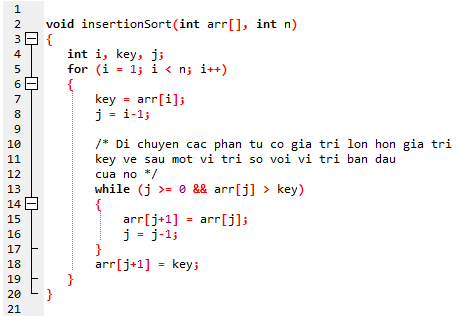
1. Khởi tạo mảng với dãy con đã sắp xếp có k = 1 phần tử(phần tử đầu tiên, phần tử có chỉ số 0)
2. Duyệt từng phần tử từ phần tử thứ 2, tại mỗi lần duyệt phần tử ở chỉ số i thì đặt phần tử đó vào một vị trí nào đó trong đoạn từ [0…i] sao cho dãy số từ [0…i] vẫn đảm bảo tính chất dãy số tăng dần. Sau mỗi lần duyệt, số phần tử đã được sắp xếp k trong mảng tăng thêm 1 phần tử.
3. Lặp cho tới khi duyệt hết tất cả các phần tử của mảng.

*Ví dụ minh họa:*

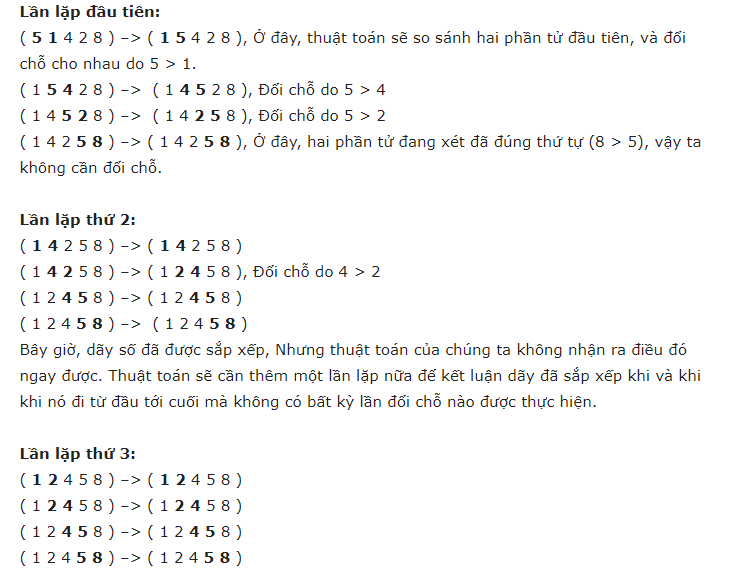


Hàng đầu tiên mô phỏng trạng thái ban đầu của mảng(dãy số chưa sắp xếp). Từ hàng thứ 2 trở đi, ta tìm chèn số đang xét vào vị trí thích hợp để đảm bảo dãy số vẫn tăng dần. Và khi lặp hết tất cả các số trong mảng, ta có trạng thái đã sắp xếp ở hàng cuối cùng.

**Code**

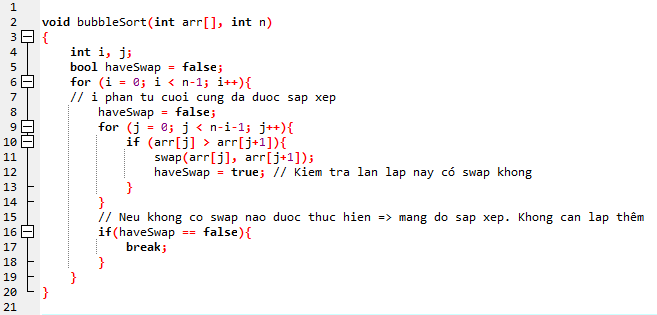


1. **Sắp xếp nổi bọt (BubbleSort)**



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.

**Code**

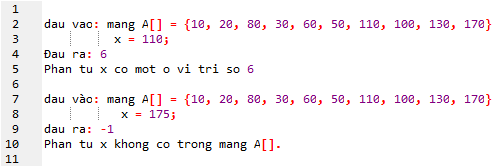


Ở đây, trong hàm bubbleSort tôi sử dụng thêm một biến haveSwap để kiểm tra tại lần lặp hiện hành có xảy ra việc đổi chỗ hai số không. Nếu không, ta có thể kết luận mảng đã sắp xếp mà không cần phải thêm một lần lặp nữa.

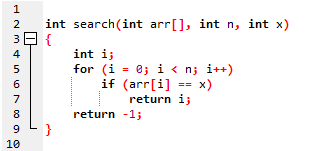
1. **Các thuật toán tìm kiếm**
2. **Thuật toán tìm kiếm tuyến tính**

Thuật toán **tìm kiếm tuyến tính** (linear search) hay còn gọi là thuật toán **tìm kiếm tuần tự** (Sequential search)  là một phương pháp tìm kiếm một phần tử cho trước trong một danh sách bằng cách duyệt lần lượt từng phần tử của danh sách đó cho đến lúc tìm thấy giá trị mong muốn hay đã duyệt qua toàn bộ danh sách.

**Tìm kiếm tuyến tính** là một giải thuật rất đơn giản khi hiện thực. Giải thuật này tỏ ra khá hiệu quả khi cần tìm kiếm trên một danh sách đủ nhỏ hoặc một danh sách chưa sắp thứ tự đơn giản. Trong trường hợp cần tìm kiếm nhiều lần, dữ liệu thường được xử lý một lần trước khi tìm kiếm: có thể được sắp xếp theo thứ tự, hoặc được xây dựng theo một cấu trúc dữ liệu đặc trưng cho giải thuật hiệu quả hơn,…



**Code trong c**



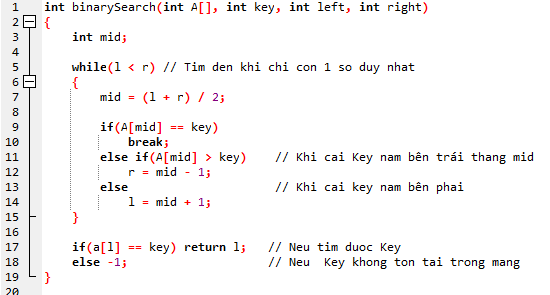
1. **Thuật toán tim kiếm nhị phân**

Thuật toán Tìm kiếm nhị phân (Binary Search) là một thuật toán cao cấp tìm kiếm tuyến tính hơn với thời gian chạy là O(log n). Đối với các danh sách lớn, thuật toán này tốt hơn hẳn tìm[kiếm tuyến tính](https://dnmtechs.com/vi/thuat-toan-tim-kiem-tuyen-tinh-tim-kiem-tuan-tu/), nhưng nó đòi hỏi danh sách phải được sắp xếp từ trước ([xem phân loại thuật toán sắp xếp)](https://dnmtechs.com/vi/thuat-toan-sap-xep/) và đòi hỏi khả năng truy nhập ngẫu nhiên (random access).

Thuật toán tìm kiếm nhị phân dùng để tìm kiếm phần tử trong một danh sách đã được [sắp xếp](https://dnmtechs.com/vi/thuat-toan-sap-xep/), ví dụ như trong một danh bạ điện thoại sắp xếp theo tên, có thể tìm kiếm số điện thoại của một người theo tên người đó.

Thuật toán tìm kiếm nhị phân chạy nhanh hơn tìm kiếm tuần tự nhưng cũng có một số nhược điểm. Tìm kiếm nhị phân có thể chậm hơn bảng băm. Nếu nội dung danh sách bị thay đổi thì danh sách phải được sắp xếp lại trước khi sử dụng tìm kiếm nhị phân. Thao tác này thường tốn nhiều thời gian.

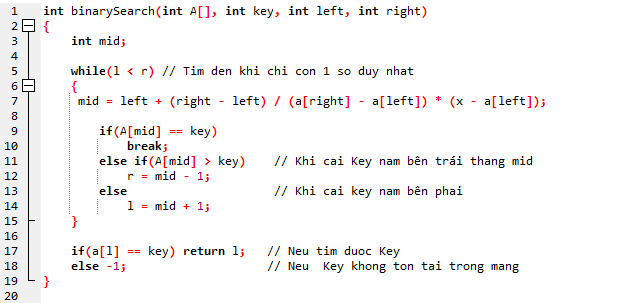
**Code trong c**



1. **Thuật toán tìm kiếm nội suy**

Là cải tiến của thuật toán tìm kiếm nhị phân. Thay vì chia đôi, thuật toán này chia theo phép tính *mid = left + (right - left) / (a[right] - a[left]) \* (x - a[left])* giúp thu gọn khoảng tìm kiếm hơn. Chỉ cần thay biểu thức tính mid này vào code mẫu của thuật toán tìm kiếm nhị phân là được.

**Code trong c**



# **B. CHƯƠNG TRÌNH QUẢN LÝ SINH VIÊN**

## **I. Chương trình quản lý sinh viên gồm:**

* Khai báo kiểu dữ liệu SinhVien có các trường họ tên, giới tính, tuổi, điểm toán – lý – hóa và điểm trung bình.
* Nhập vào danh sách N sinh viên
* Xuất danh sách N sinh viên
* Tính điểm trung bình cho N sinh viên
* Sắp xếp N sinh viên theo thứ tự tăng dần của điểm trung bình
* Xếp loại N sinh viên
* Xuất danh sách N sinh viên ra file
* Viết chương trình dạng menu cho phép sử dụng các tính năng trên

## **II. Kiến thức cần có:**

1. Kiến thức lập trình cơ bản
2. Kiến thức về hàm con
3. Struct trong C
4. Cách sử dụng cấu trúc lặp: for, do while, while
5. Cách sử dụng cấu trúc điều khiển & rẽ nhánh: if else, switch case
6. Nhập xuất file

## **III. Giải thích code:**

**1 .Khai báo Thư viện**

****

**2. Câu lệnh Struct:**

Câu lệnh struct định nghĩa một kiểu dữ liệu mới, với hơn một thành phần trong chương trình của bạn. Dạng tổng quát của câu lệnh struct như sau đây:

struct [ten\_cau\_truc]

{

phan dinh nghia thanh vien cau truc;

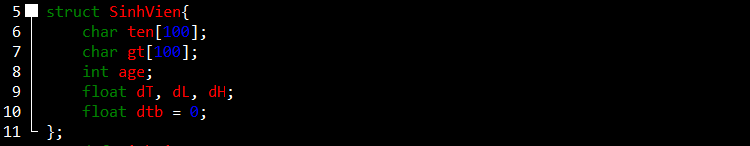
phan dinh nghia thanh vien cau truc;

...

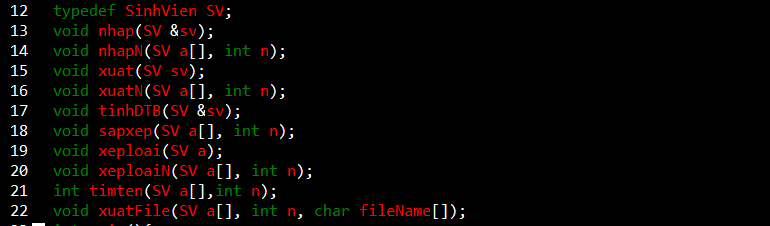
phan dinh nghia thanh vien cau truc;

} [mot hoac nhieu bien cau truc];

**ten\_cau\_truc** có thể tùy chọn và một thành phần định nghĩa là các biến thường như int i, float j hoặc một định nghĩa biến khác …. Tại phần cuối cùng của định nghĩa cấu trúc, trước dấu chấm phẩy, bạn có thể xác định một hoặc nhiều biến cấu trúc (tùy chọn).



**3. Các hàm:**

- Các hàm cần có trong chương trình:

Mẫu chung của định nghĩa hàm trong Ngôn ngữ C như sau:

Một định nghĩa hàm trong ngôn ngữ C bao gồm *đầu hàm* và một *thân hàm*. Dưới đây là các phần của một hàm:

**Kieu\_tra\_ve** Ten\_ham( Danh sach tham so )

{

Than ham

}

* **Kiểu trả về**: Một hàm có thể trả về một giá trị. **Kieu\_tra\_ve** là dạng dữ liệu của giá trị mà hàm trả về. Vài hàm cung cấp các hoạt động và không trả về giá trị nào cả. Đó là hàm **void**.
* **Tên hàm:** Đây là tên thực sự của hàm. Tên hàm và danh sách tham số cấu tạo nên dấu hiệu hàm.
* **Danh sách tham số:** Khi hàm được gọi, bạn phải truyền vào danh sách các tham số. Một giá trị hướng đến một tham số thực tế. Danh sách tham số có các kiểu, thứ tự và số lượng các tham số của hàm. Các tham số trong hàm là tùy chọn, nghĩa là một hàm có thể không có tham số.
* **Thân hàm:** Phần thân của một hàm bao gồm tập hợp các lệnh xác định những gì mà hàm thực hiện.

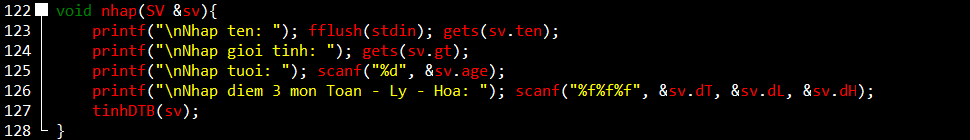
1. **Hàm nhập bao gồm:**

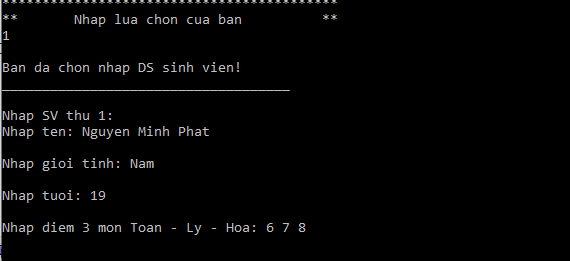
* Nhập tên:

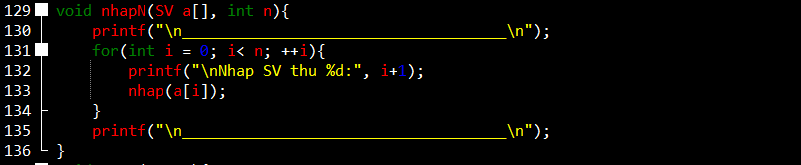
Hàm gets: dùng để nhạp xâu kí tự.

Để tránh lỗi về bộ nhớ đệm bàn phím của phím enter ta thêm fflush(stdin);

=> Xóa bộ nhớ đệm

* Nhập giới tính
* Nhập tuổi
* Nhập điểm 3 môn Toán Lý Hóa



1. **Nhập n sinh viên:**

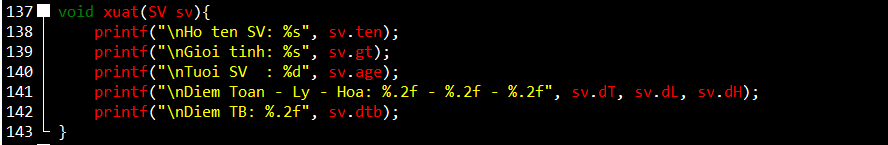


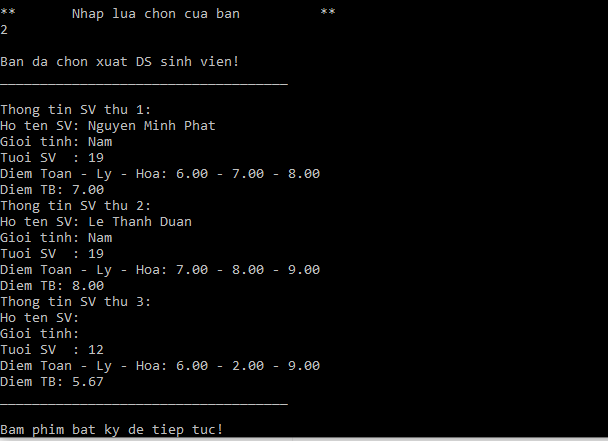
1. **Hàm xuất:**

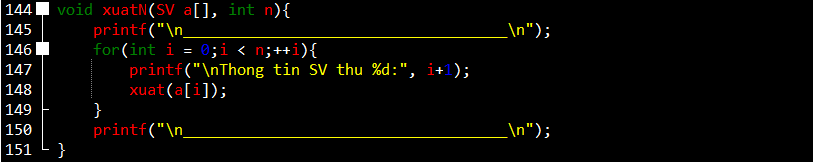
In ra màn hình dữ liệu đã nhập.

Để truy nhập đến thành phần của cấu trúc ta sử dụng toán tử chấm (.).

**TenBienCauTruc.TenThanhPhan;**

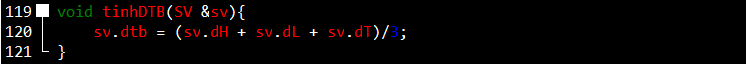
Như VD trên ta truy cập như sau:  
sv1.hoten; sv1.toan; // truy xuất tới họ tên, điểm toán  
sv1.ns.ngay; sv1.ns.thang; // truy xuất tới ngày sinh và tháng sinh.



1. **Xuất n sinh viên**

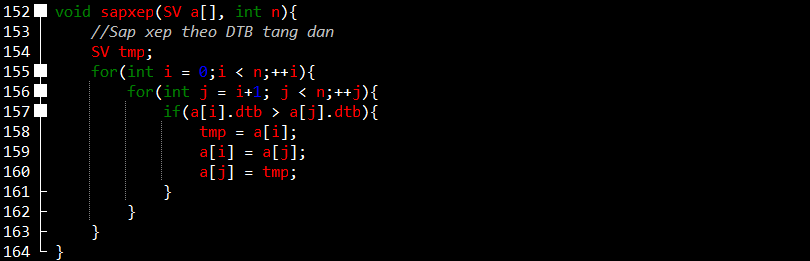
Xuất từng sinh viên.

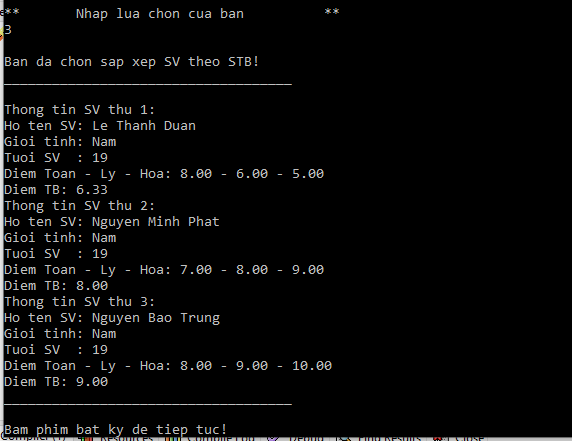
1. **Hàm tính điểm trung binh**

Lấy tổng điểm 3 môn chia 3

1. **Hàm sắp xếp sinh viên có DTB tăng dần:**

Sử dụng 2 vòng lặp for:

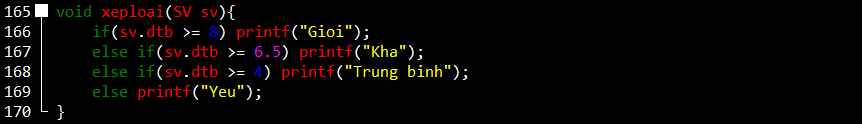
Sinh viên nào có DTB cao hơn sẽ được hoán vị sang bên phải. 

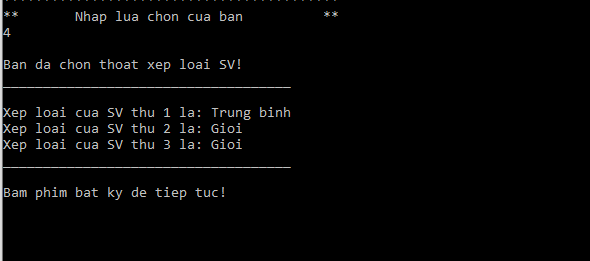


1. **Hàm xếp loại sinh viên**

- xeploai(N) - xep học lực của tất cả SV trong a[]

- for (từng sinh viên trong danh sách) rồi gọi hàm xeploai(a[i]) để print học lực cho sinh viên a[i] đang xét.

****

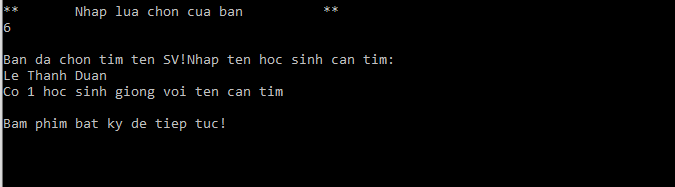
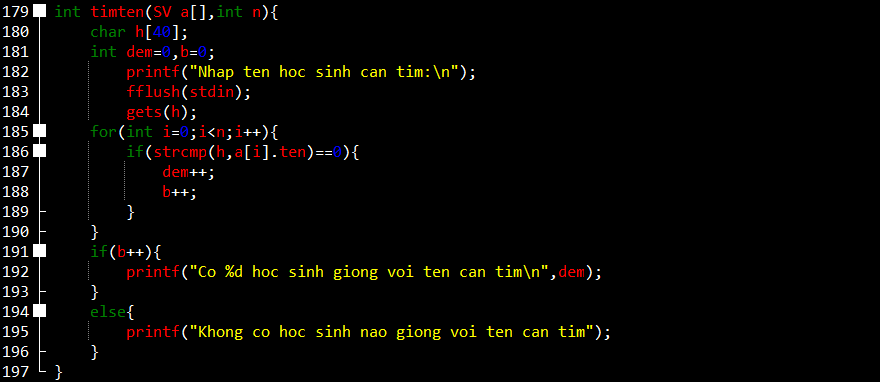


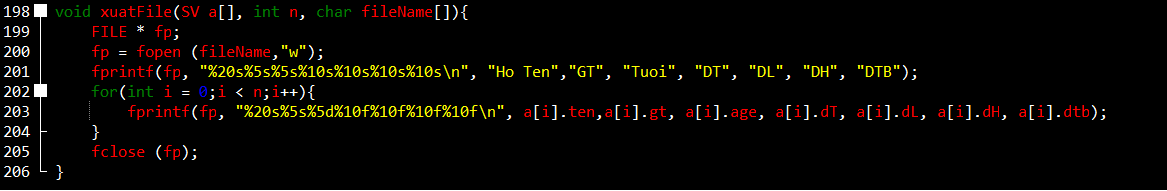
1. **Hàm tìm tên sinh viên**

- for (từng sinh viên trong danh sách)

- strcmp so sanh 2 chuỗi, giống thì trả về 0

- Xét từng tên trong danh sách SV a[] có tên nào giống thì **Số** sinh viên có tên giống = ( đếm +1)

- if(b++) --> nếu b có giá trị 0 và b tăng (có sinh viên có tên giống thì b mới tăng).

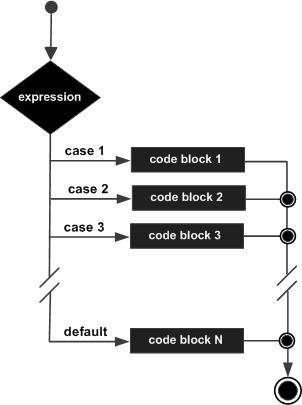
1. **Hàm xuất file**
2. **Hàm main (Hàm chính)**

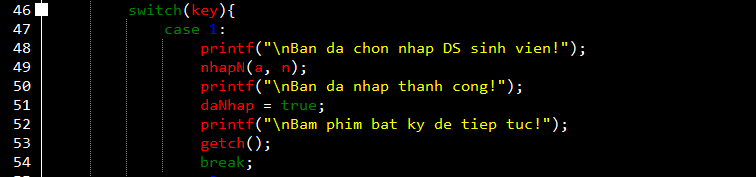
Một lệnh **switch** trong C/C++ cho một biến được kiểm tra một cách bình đẳng trong danh sách các giá trị. Mỗi giá trị được gọi là một **case - trường hợp** và biến được chuyển tới được kiểm tra cho mỗi trường hợp switch.

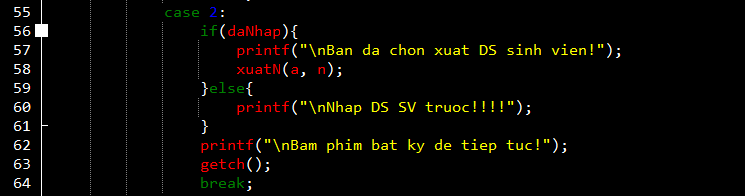
Các quy tắc sau được áp dụng tới một lệnh **switch**:

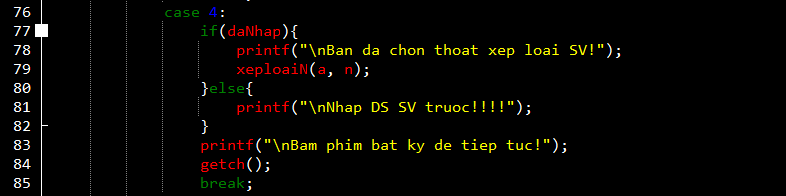
* Biểu thức **bieu\_thuc** được sử dụng trong một lệnh switch phải có kiểu là integer hoặc liệt kê, hoặc là một trong các kiểu lớp trong đó lớp có một hàm biến đổi đơn tới một kiểu integer hoặc kiểu liệt kê.
* Bạn có thể có bất kỳ số lệnh case nào trong một switch. Mỗi case được theo sau bởi giá trị để được so sánh và một dấu hai chấm.
* **bieu\_thuc\_hang**, là biểu thức hằng, cho một case phải cùng kiểu dữ liệu với biến trong switch, và nó phải là hằng số.
* Khi biến được chuyển tới là cân bằng với một case, lệnh theo sau case đó sẽ thực thi tới khi gặp lệnh **break**.
* Khi gặp lệnh **break**, switch kết thúc, và dòng điều khiển nhảy tới dòng lệnh tiếp theo của lệnh switch đó.
* Không phải mỗi case cần chứa một lệnh **break**. Nếu không có lệnh **break** nào xuất hiện, dòng điều khiển sẽ **không tới được** case tiếp theo cho tới khi bắt gặp một lệnh break.
* Một lệnh **switch** có thể có một case **mặc định** tùy chọn, mà phải xuất hiện ở cuối cùng của switch. Case mặc định này có thể được sử dụng để thực hiện một nhiệm vụ khi không có case nào true. Trong trường hợp case mặc định này thì không cần lệnh **break**.

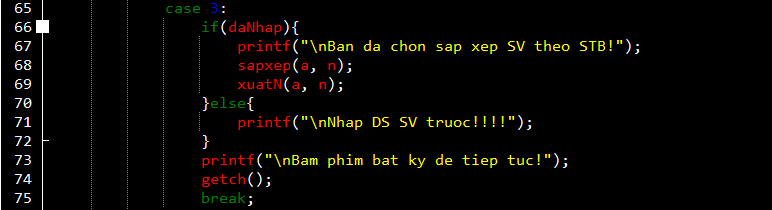
**Sơ đồ**

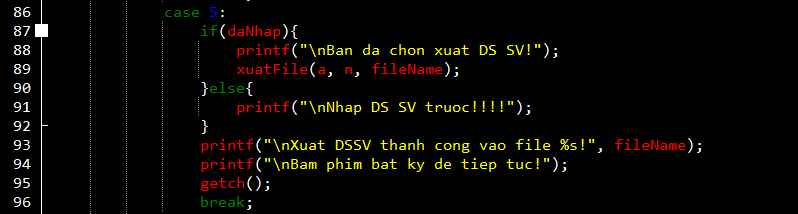


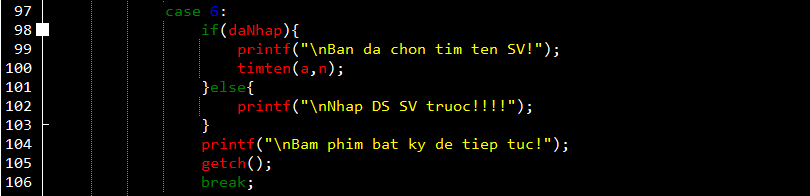


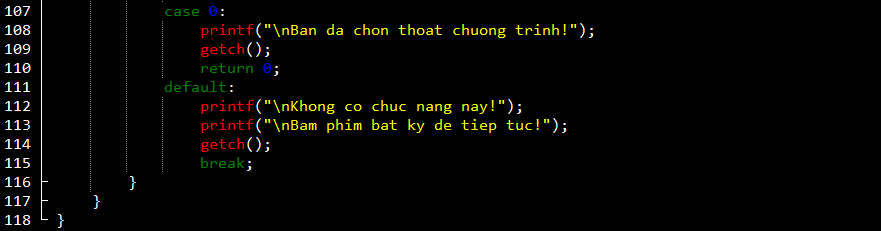












NGUỒN THAM KHẢO:

* 1. <https://nguyenvanhieu.vn/doc-ghi-file-trong-c/>
  2. <https://cunglaptrinh.blogspot.com/2015/07/danh-sach-lien-ket-don-va-cac-thao-tac.html>
  3. <https://nguyenvanhieu.vn/danh-sach-lien-ket-don/>
  4. <https://vietjack.com/bai-tap-c/cac-thuat-toan-sap-xep-trong-c.jsp>
  5. <https://nguyenvanhieu.vn/thuat-toan-tim-kiem-nhi-phan/>