

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



## **BÁO CÁO BÀO TẬP LỚN** **Môn: Lập trình nâng cao**

Đề tài: xây dựng chương trình quản lý sinh viên bằng ngôn ngữ C

Giảng viên: Cô Trần Thị Dung

Nhóm: Nguyễn Thành Tiến

Nguyễn Văn Tiến

Lê Quang Sơn

Lớp: CQ.60.CNTT

Khóa 60

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

<b>CHƯƠNG I: CƠ SỞ LÝ THUYẾT.....</b>	<b>6</b>
<b>I. Danh sách liên kết đơn, cấp phát bộ nhớ và các thuật toán. ....</b>	<b>6</b>
<b>1.1. Cấp phát bộ nhớ. ....</b>	<b>6</b>
<b>1.2. Cấu trúc (struct). ....</b>	<b>7</b>
<b>1.3. Danh sách liên kết đơn.....</b>	<b>8</b>
<b>1.3.1 Cài đặt danh sách.....</b>	<b>9</b>
<b>1.3.2 Khởi tạo danh sách rỗng. ....</b>	<b>9</b>
<b>1.3.3 Kiểm tra danh sách có rỗng hay không. ....</b>	<b>10</b>
<b>1.3.5 Tạo một node trong danh sách. ....</b>	<b>10</b>
<b>1.3.6 Chèn Node p vào đầu danh sách.....</b>	<b>10</b>
<b>1.3.7 Chèn Node p vào vị trí k của danh sách. ....</b>	<b>11</b>
<b>1.3.8 Tìm phần tử có giá trị x trong danh sách. ....</b>	<b>12</b>
<b>1.3.9 Xóa phần ở vị trí đầu tiên. ....</b>	<b>12</b>
<b>1.3.10 xóa phần tử ở vị trí k. ....</b>	<b>13</b>
<b>1.3.11 Xóa phần tử có giá trị x. ....</b>	<b>13</b>
<b>2. Các thuật toán sắp xếp. ....</b>	<b>13</b>
<b>2.1 Sắp xếp nổi bọt (bubble sort) ....</b>	<b>13</b>
<b>2.2 Sắp xếp chèn. ....</b>	<b>15</b>
<b>2.3 Sắp xếp chọn. ....</b>	<b>16</b>
<b>2.4 Sắp xếp đổi chỗ.....</b>	<b>17</b>
<b>2.5 Sắp xếp dãy số giảm dần ....</b>	<b>17</b>
<b>2.6 Sắp xếp dãy số tăng dần. ....</b>	<b>19</b>
<b>3. Các thuật toán tìm kiếm. ....</b>	<b>20</b>
<b>3.1 Tìm kiếm tuyến tính.....</b>	<b>20</b>
<b>3.2 Tìm kiếm nhị phân.....</b>	<b>20</b>
<b>4. Các thao tác với file làm việc với tệp.....</b>	<b>21</b>
<b>4.1. Làm việc với file.....</b>	<b>21</b>

4.1.1	File văn bản – text files.....	21
4.1.2	File nhị phân – Binary files.....	21
4.2.	Các thao tác với file.....	22
4.2.1	Khai báo sử dụng FILE.....	22
4.2.2	Thao tác mở file – hàm open.....	22
4.2.2	Thao tác đóng file – hàm fclose.....	24
4.2.3	Đóng tất cả các hàm đang mở - hàm fcloseall.....	24
4.2.5	Đọc/Ghi file trong văn bản C.....	26
4.2.6	Đọc/Ghi file nhị phân trong C. ....	31
<b>CHƯƠNG II: XÂY DỰNG PHẦN MỀM QUẢN LÝ SINH VIÊN .....</b>		<b>32</b>
I.	Lý do chọn đề tài.....	32
II.	Giải thích chi tiết bài tập.....	32
1.	Mô tả công việc cần làm.....	32
2.	Thực hiện các công việc cần làm.....	33
2.1	Tạo cấu trúc sinh viên.....	33
2.2	Nhập tên sinh viên và in ra màn hình. ....	33
3.	Sắp xếp sinh viên theo tên. ....	34
4.	Sắp xếp điểm trung bình 3 môn học.....	34
5.	Tìm kiếm và hiển thị sinh viên trong danh sách. ....	35
6.	Đọc/Ghi thông tin sinh viên vào tệp txt.....	35
<b>CHƯƠNG III. TỔNG KẾT .....</b>		<b>36</b>
I.	Kết quả cần đạt và hướng giải phát triển. ....	36
1.	Kết quả cần đạt. ....	36
2.	Hướng phát triển.....	36
II.	Các nguồn học tập tham khảo .....	37

# LỜI CẢM ƠN

Lời đầu tiên em muốn cảm ơn đến các thầy, cô trong Bộ môn Công nghệ thông tin – Phân hiệu trường đại học Giao thông vận tải tại Thành Phố Hồ Chí Minh. Những người đã mang đến kiến thức và hướng dẫn em trong công việc học Bộ môn Công nghệ thông tin này.

Ở tại đây em được trang bị những kiến thức vô cùng bổ ích trên sách vở và ngoài ra em còn được trang bị những kỹ năng sống những kiến thức để ứng xử linh hoạt đối với những vấn đề ngoài xã hội sau khi rời khỏi băng ghế nhà trường phổ thông. Đặc biệt em muốn gửi lời cảm ơn đến cô Trần Thị Dung người đã tận tình hướng dẫn chúng em học tập trong môn Lập trình nâng cao. Một người đã sử dụng hết những gì mình có để có thể ôn lại cũng như hướng dẫn cho em trong quá trình học.

Và trong quá trình học tập và là các bài tập em không thể nào không mắc các sai lầm từ là lớn nhất hay là nhỏ nhất đi nữa. Và đây là lần đầu tiên em làm một bài tập lớn như thế này em mong rằng quý thầy cô có thể cho em những ý kiến chỉ ra những lỗi sai để em có thể khắc phục để không bị mắc phải cho những bài tập lớn cũng như đồ án sau này.

Và cuối cùng em xin chúc quý thầy cô trong Bộ môn Công nghệ thông tin cũng như quý thầy cô toàn trường có nhiều sức khỏe để có thể tiếp tục thực hiện công việc truyền đạt những kiến thức cho chúng em.

Em xin chân thành cảm ơn!

# CHƯƠNG I: CƠ SỞ LÝ THUYẾT

## I. Danh sách liên kết đơn, cấp phát bộ nhớ và các thuật toán.

### 1.1. Cấp phát bộ nhớ.

- Dùng để cấp phát vùng nhớ cho biến con trỏ trong ngôn ngữ C.
- Sử dụng các hàm malloc(), 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 không 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:

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  int main(){
4  int *ptr, i, n1, n2;
5  printf("nhap so luong phan tu: ");
6  scanf("%d", &n1);
7  ptr = (int*)malloc(n1 * sizeof(int));
8  printf("Dia chi cua vung nho vua cap phat: %u", ptr);
9
10 printf("nhap lai so luong phan tu: ");
11 scanf("%d", &n2);
12 // phân bo lai vung nho
13 ptr = (int*)realloc(ptr, n2 * sizeof(int));
14 printf("Dia chi cua vung nho duoc cap phat lai: %u", ptr);
15 // giải phóng bo nho
16 free(ptr);
17 return 0;
18 }
```

## 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 struct sinhvien{
2     int masv;
3     char ho[20];
4     char ten[20];
5     bool gioitinh;
6     char queQuan[100];
7 };
8

```

### 1.3. 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 giữa 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.



<ul style="list-style-type: none"> <li>▪ Kích thước của mảng là hằng số, không thay đổi khi chạy chương trình</li> <li>▪ Sử dụng mảng không tối ưu được bộ nhớ. Có thể thừa hoặc thiếu bộ nhớ khi xóa hoặc chèn phần tử vào mảng</li> </ul>	<ul style="list-style-type: none"> <li>▪ Kích thước của danh sách liên kết có thể thay đổi khi chạy chương trình.</li> <li>▪ 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.</li> </ul>
---	---

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
- 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 một 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  typedef int item; //kieu cac phan tu dinh nghĩa là item
2  typedef struct Node //xay dung mot Node trong danh sach
3  {
4      item Data;//du lieu co kieu item
5      Node *Next;//truong Next là một con trỏ, trỏ đến Node tiếp theo
6  };
7  typedef Node *List;//List là một danh sách các Node

```

### 1.3.2 Khởi tạo danh sách rỗng.

```

23 void Init (List &L) // &L lấy địa chỉ của danh sách ngay khi truyền vào hàm
24 {
25     L=NULL; //Cho L trỏ đến NULL
26 }

```

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.

```
28 int Isempthy (List L)|
29 {
30     return (L==NULL);
31 }
```

### 1.3.4 Tính độ dài danh sách.

```
33 int len (List L)
34 {
35     Node *P=L; //tao 1 Node P de duyet danh sach L
36     int i=0; //bien dem
37     while (P!=NULL) //trong khi P chua tro den NULL (cuoi danh sach thi lam)
38     {
39         i++; //tang bien dem
40         P=P->next; //cho P tro den Node tiep theo
41     }
42     return i; //tra lai so Node cua l
43 }
```

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

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

```
45 Node *Make_Node (Node *P, item x) //tao 1 Node P chua thong tin la x
46 {
47     P = (Node *) malloc (sizeof (Node)); //Cap phat vung nho cho P
48     P->next = NULL; //Cho truong Next tro den NULL
49     P->Data = x; //Ghi du lieu vao Data
50     return P;
51 }
```

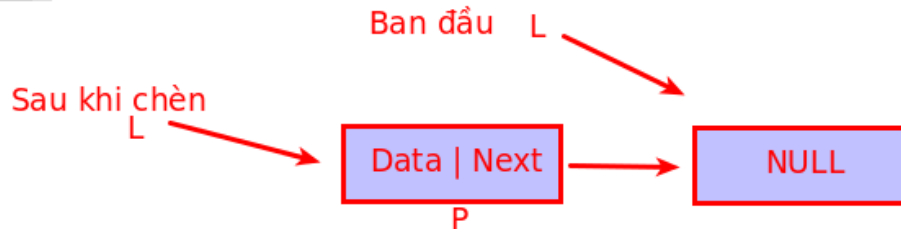
### 1.3.6 Chèn Node p vào đầu danh sách.

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

```

53 void Insert_first (List &L, item x) //Chen x vao vi tri dau tien trong danh sach
54 {
55     Node *P;
56     P = Make_Node(P,x); //tao 1 Node P
57     P->next = L; //Cho P tro den L
58     L = P; //L tro ve P
59 }

```



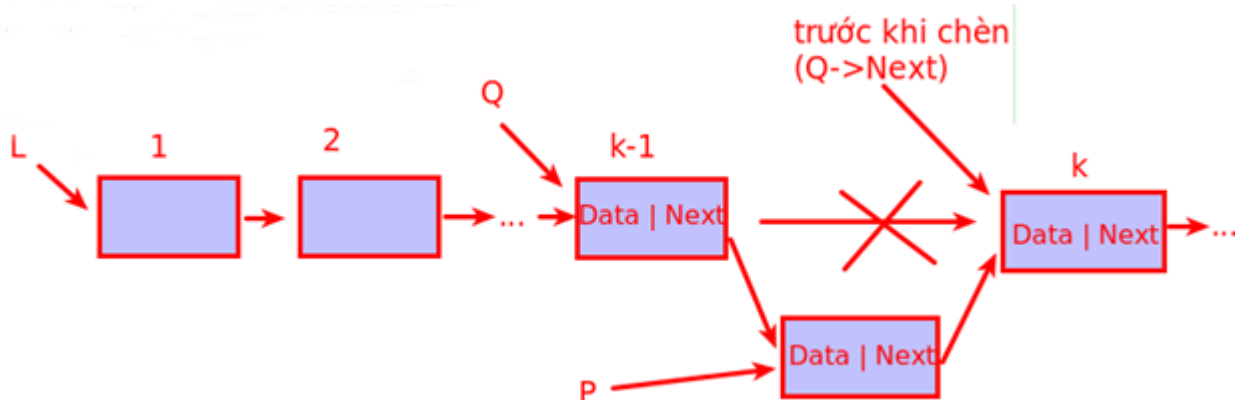
### 1.3.7 Chèn Node p vào vị trí k của danh sách.

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  $P \rightarrow \text{Next}$  trở đến Node  $Q \rightarrow \text{Next}$ , tiếp đến cho  $Q \rightarrow \text{Next}$  trở đến P.

```

61 void Insert_k (List &L, item x, int k) //chen x vao vi tri k trong danh sach
62 {
63     Node *P, *Q = L;
64     int i=1;
65     if (k<1 || k> len(L)+1) printf("Vi tri chen khong hop le !"); //kiem tra dieu kien
66     else
67     {
68         P = Make_Node(P,x); //tao 1 Node P
69         if (k == 1) Insert_first(L,x); //chen vao vi tri dau tien
70         else //chen vao k != 1
71         {
72             while (Q != NULL && i != k-1) //duyet den vi tri k-1
73             {
74                 i++;
75                 Q = Q->next;
76             }
77             P->next = Q->next;
78             Q->next = P;
79         }
80     }
81 }

```



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

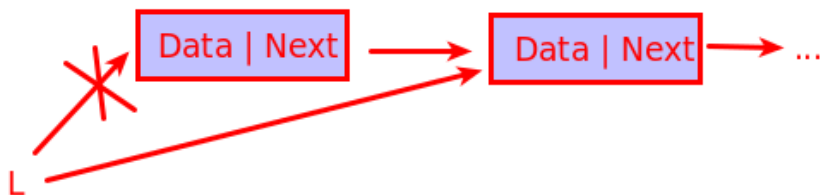
Ta duyệt danh sách cho đến khi tìm thấy hoặc kết thúc và trả về vị trí nếu tìm thấy, ngược lại trả về 0.

```
83 int Search (List L, item x) //tim x trong danh sach
84 {
85     Node *P=L;
86     int i=1;
87     while (P != NULL && P->Data != x) //duyet danh sach den khi tim thay hoac ket thuc danh sach
88     {
89         P = P->next;
90         i++;
91     }
92     if (P != NULL) return i; //tra ve vi tri tim thay
93     else return 0; //khong tim thay
94 }
```

### 1.3.9 Xóa phần ở vị trí đầu tiên.

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

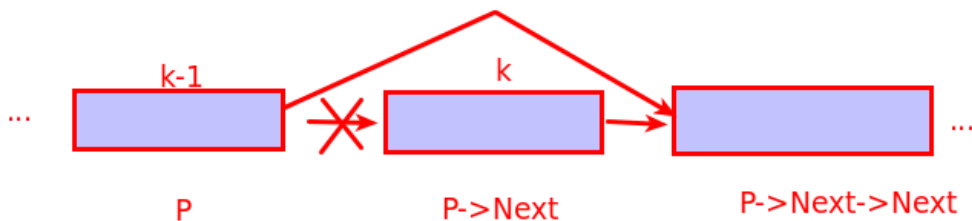
```
96 void Del_frist (List &L, item &x) //Xoa phan tu dau tien
97 {
98     x = L->Data; //lay gia tri ra neu can dung
99     L = L->next; //Cho L tro den Node thu 2 trong danh sach
100 }
```



### 1.3.10 xóa phần tử ở vị trí k.

Dùng P để duyệt tới vị trí k-1 và tiến hành cho P-Next trở đến phần tử kế tiếp k mà bỏ qua k.

```
102 void Del_k (List &L, item &x, int k) //Xoa Node k trong danh sach
103 {
104     Node *P=L;
105     int i=1;
106     if (k<1 || k>len(L)) printf("Vi tri xoa khong hop le !"); //kiem tra dieu kien
107     else
108     {
109         if (k==1) Del_frist(L,x); //xoa vi tri dau tien
110         else //xoa vi tri k != 1
111         {
112             while (P != NULL && i != k-1) //duyet den vi tri k-1
113             {
114                 P=P->next;
115                 i++;
116             }
117             P->next = P->next->next; //cho P tro sang Node ke tiep vi tri k
118         }
119     }
120 }
```



### 1.3.11 Xóa phần tử có giá trị x.

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

```
122 void Del_x (List &L, item x) //xoa phan tu x trong danh sach
123 {
124     while (Search(L,x)) Del_k (L,x,Search(L,x)); //trong khi van tim thay x thi van xoa
125 }
```

## 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:

```
1 void sapxep(int a[], int n){
2     for(int i=0; i<n-1; i++){
3         for(int j=n-1; j>i; j--){
4             if(a[j]<a[j-1]){
5                 swap(a[j],a[j-1]);
6             }
7         }
8     }
9 }
```

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 )  $\rightarrow$  ( **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 )  $\rightarrow$  ( 1 **4** 5 2 8 ), Đổi chỗ do  $5 > 4$

( 1 4 **5** 2 8 )  $\rightarrow$  ( 1 4 **2** 5 8 ), Đổi chỗ do  $5 > 2$

( 1 4 2 **5** 8 )  $\rightarrow$  ( 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 )  $\rightarrow$  ( **1** 4 2 5 8 )

( **1** 4 2 5 8 )  $\rightarrow$  ( **1** 2 4 5 8 ), Đổi chỗ do  $4 > 2$

( 1 2 **4** 5 8 )  $\rightarrow$  ( 1 2 **4** 5 8 )

( 1 2 4 **5** 8 )  $\rightarrow$  ( 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 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 )  $\rightarrow$  ( **1** 2 4 5 8 )

( 1 **2** 4 5 8 )  $\rightarrow$  ( 1 **2** 4 5 8 )

( 1 2 **4** 5 8 )  $\rightarrow$  ( 1 2 **4** 5 8 )

( 1 2 4 **5** 8 )  $\rightarrow$  ( 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  $a_1, \dots, a_k$ . Với  $k = 1$ , danh sách gồm một phần tử đã được sắp. Giả sử trong danh sách  $k-1$  phần tử đầu  $a_1, \dots, a_{k-1}$  đã được sắp. Để sắp xếp phần tử  $a_k = x$  ta tìm vị trí thích hợp của nó trong dãy  $a_1, \dots, a_{k-1}$ . Vị trí thích hợp đó là đứng trước phần tử lớn hơn nó và sau phần tử nhỏ hơn hoặc bằng nó.

Các phần tử $\leq x$			Vị trí thích hợp	Các phần tử $> x$			Các phần tử chưa sắp		
$a_1$	...	$a_{i-1}$	$x$	$a_{i+1}$	...	$a_{k-1}$	$a_{k+1}$	...	$a_n$

Ví dụ:

```
11 □ for(int i=1; i<n; i++){
12     //tim vi tri phu hop cho i
13     int j=i;
14     while(j>0 && data[i]<data[j-1])
15         j--;//dua i ve dung vi tri
16     int tmp=data[i];
17     for(int k=i; k>j; k--)
18         data[k]=data[k-1];
19     data[j]=tmp;
20 }
```

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  $a_4 = 6$  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  $a_5 = 4$  ta được

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

Làm tiếp theo với  $a_6 = 2$  ta được

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

Cuối cùng chèn  $a_7 = 5$

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 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:



```

22 arr[] = 62 24 15 22 1
23
24 // tìm phần tử nhỏ nhất trong arr[0...4]
25 // và đổi cho nó với phần tử đầu tiên
26 [1] 24 15 22 62
27
28 // tìm phần tử nhỏ nhất trong arr[1...4]
29 // và đổi cho nó với phần tử đầu tiên của arr[1...4]
30 1 [15] 24 22 62
31
32 // tìm phần tử nhỏ nhất trong arr[2...4]
33 // và đổi cho nó với phần tử đầu tiên của arr[2...4]
34 1 15 [22] 24 62
35
36 // tìm phần tử nhỏ nhất trong arr[3...4]
37 // và đổi cho nó với phần tử đầu tiên arr[3...4]
38 11 12 22 [24] 62

```

## 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.

```

40 void InterchangeSort(int a[], int n)
41 {
42     for (int i = 0; i < n-1; i++)
43         for (int j = i+1; j < n; j++)
44             if (a[i] > a[j])
45             {
46                 int temp = a[i];
47                 a[i] = a[j];
48                 a[j] = temp;
49             }
50 }

```

## 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.

```

1  #include <stdio.h>
2  int main(){
3      int a[100];
4      int n;
5      printf("\nNhap so luong phan tu n = ");
6      do{
7          scanf("%d", &n);
8          if(n <= 0){
9              printf("\nNhap lai n = ");
10             }
11         }while(n <= 0);
12
13         for(int i = 0; i < n; i++){
14             printf("\nNhap a[%d] = ", i);
15             scanf("%d", &a[i]);
16         }
17         // Sap xep dung thuat toan sap xep chon
18         int tg;
19         for(int i = 0; i < n - 1; i++){
20             for(int j = i + 1; j < n; j++){
21                 if(a[i] < a[j]){
22                     // Hoan vi 2 so a[i] va a[j]
23                     tg = a[i];
24                     a[i] = a[j];
25                     a[j] = tg;
26                 }
27             }
28         }
29         printf("\nMang da sap xep la: ");
30         for(int i = 0; i < n; i++){
31             printf("%5d", a[i]);
32         }
33     }

```

Kết quả chạy chương trình:

```

Nhap so luong phan tu n = 5

Nhap a[0] = 2

Nhap a[1] = 4

Nhap a[2] = 5

Nhap a[3] = 2

Nhap a[4] = 6

Mang da sap xep la:    6    5    4    2    2

```

## 2.6 Sắp xếp dãy số tăng dần.

```
18 | int tg;
19 | for(int i = 0; i < n - 1; i++){
20 |     for(int j = i + 1; j < n; j++){
21 |         if(a[i] > a[j]){
22 |             // Hoan vi 2 so a[i] va a[j]
23 |             tg = a[i];
24 |             a[i] = a[j];
25 |             a[j] = tg;
26 |         }
27 |     }
28 | }
```

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.

```
18 | int tg;
19 | for(int i = 0; i < n - 1; i++){
20 |     for(int j = i + 1; j < n; j++){
21 |         if(a[i] > a[j]){
22 |             // Hoan vi 2 so a[i] va a[j]
23 |             tg = a[i];
24 |             a[i] = a[j];
25 |             a[j] = tg;
26 |         }
27 |     }
28 | }
```

### 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.

```
1 | int linearSearch (int a[], int n, int x) {  
2 |     for (int i = 0; i < n; i++) {  
3 |         if (a[i] == x) return i; //trả về vị trí tìm thấy  
4 |         else return -1; //trả về -1 nếu ko tìm thấy  
5 |     }  
6 | }
```

#### 3.2 Tìm kiếm nhị phân.

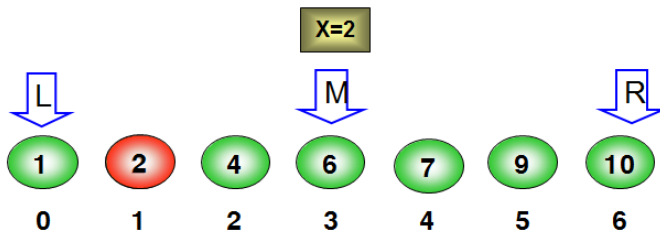
1. Xét đoạn mảng `art[left...right]` cần tìm kiếm phần tử `x`. Ta so sánh `x` với phần tử ở vị trí giữa của mảng (`mid = (left + right)/2`). Nếu:
2. Nếu phần tử `art[mid] = x`. Kết luận và thoát chương trình.
3. Nếu `art[mid] < x`. Chỉ thực hiện tìm kiếm trên đoạn `art[mid+1...right]` (nửa bên phải của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

```

1  int binarySearch (int a[], int n, int x) {
2      int left = 0, right = n - 1, mid;
3      do {
4          mid = (left + right) / 2;
5          if (a[mid] == x) return mid;
6          else if (a[mid] <= x) left = mid + 1;
7          else right = mid - 1;
8      } while (left <= right);
9      return -1;
10 }

```

Tìm thấy 2 tại vị trí 1



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

### 4.1. Làm việc với file.

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

#### 4.1.1 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 editor thông dụng như Notepad, Notepad++, Sublime Text, ....

Khi bạn mở các file này bằng các text editor 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.

#### 4.1.2 File nhị phân – Binary files.

File nhị phân thường có đuôi mở rộng là .bin

Thay vì lưu trữ dưới dạng văn bản thuần túy, 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 editor kể trên.

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

## 4.2. 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 từ 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 phẩy (dấu ,).

Ví dụ

```
FILE *vb, *np; /* Khai báo hai biến con trỏ tệp */
```

### 4.2.2 Thao tác mở file – hàm open.

#### Cấu trúc ngữ pháp của hàm

```
FILE *vb, *np; /* Khai báo hai biến con trỏ tệp */
```

#### Nguyên hàm trong: stdio.h

Trong đó: 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, open() 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.

Ví dụ :

```
f=fopen("TEPNP", "wb");
```

## 4.2.2 Thao tác đóng file – hàm fclose.

**Cấu trúc ngữ pháp của hàm:**

```
int fclose(FILE *fp);
```

**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 xóa vùng đệm
- Giải phóng biến trỏ tệp.
- Nếu lệnh thành công, hàm sẽ cho giá trị 0, trái lại nó cho hàm EOF.

Ví dụ

```
fclose(f);
```

## 4.2.3 Đóng tất cả các hàm đang mở - hàm fcloseall.

**Cấu trúc ngữ pháp của hàm**



```
int fcloseall(void);
```

## 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.

Ví dụ :

```
fcloseall();
```

*Làm sạch vùng đệm – hàm fflush*

### Cấu trúc ngữ pháp của hàm

```
int fflush(FILE *fp);
```

## 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.

Ví dụ :

```
fflush(f);
```

```
int fflushall(void);
```

## 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

```
int ferror(FILE *fp);
```

## 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

```
int feof(FILE *fp);
```

### 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 chuyển 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 khuôn dạng lên tệp.

fscanf: dùng để đọc dữ liệu từ tệp theo khuôn 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 khuôn dạng xác định trong mỗi chuỗi điều khiển dk. Nếu thành công, hàm trả 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 chứa 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()

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  int main()
5  {
6      int num;
7      FILE *fptr;
8      fptr = fopen("C:\\program.txt", "w");
9
10     if(fptr == NULL)
11     {
12         printf("Error!");
13         exit(1);
14     }
15
16     printf("Enter num: ");
17     scanf("%d", &num);
18
19     fprintf(fptr, "%d", num);
20     fclose(fptr);
21
22     return 0;
23 }
```

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 số 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:

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  int main()
5  {
6      int num;
7      FILE *fptr;
8
9      if ((fptr = fopen("C:\\program.txt", "r")) == NULL){
10         printf("Error! opening file");
11
12         // Program exits if the file pointer returns NULL.
13         exit(1);
14     }
15
16     fscanf(fptr, "%d", &num);
17
18     printf("Value of n=%d", num);
19     fclose(fptr);
20
21     return 0;
22 }
```

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:

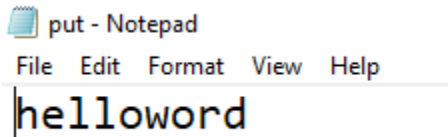
s là con trỏ tới địa chỉ đầu của một chuỗi ký tự kết thúc bằng dấu '\0'

fp là con trỏ tệp.

ví dụ:

```
1  #include <stdio.h>
2
3  int main()
4  {
5      FILE *fp;
6      fp=fopen("put.txt","w+");
7      fputs("helloworld",fp);
8      fclose(fp);
9      return 0;
10 }
```

Và đây là kết quả ra như vậy:



## 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 giá trị NULL.

+ Đối:

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

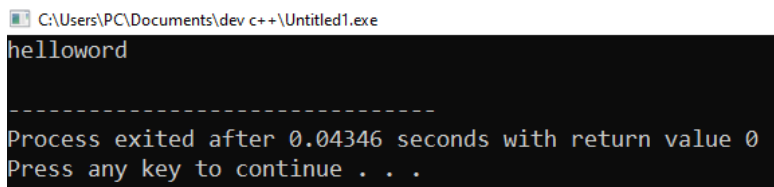
n là số nguyên xác định độ dài của dãy cần đọc.

fp là con trỏ tệp.

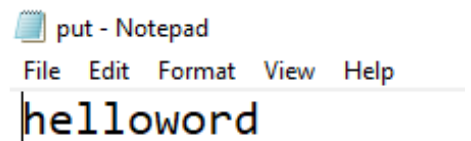
ví dụ:

```
1  #include <stdio.h>
2
3  int main()
4  {
5      FILE *fp;
6      char str[60];
7      /*mô file de doc*/
8      fp=fopen("put.txt","r");
9      if(fp==NULL){
10         perror("xay ra loi trong khi doc file");
11         return (-1);
12     }
13     if(fgets(str,60,fp)!=NULL){
14         /*ghi noi dung stdout*/
15         puts(str);
16     }
17     fclose(fp);
18     return 0;
19 }
```

Và kết quả như thế này:



```
C:\Users\PC\Documents\dev c++\Untitled1.exe
helloworld
-----
Process exited after 0.04346 seconds with return value 0
Press any key to continue . . .
```



```
put - Notepad
File Edit Format View Help
helloworld
```

## 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 tệp tin.

n: số phần tử sẽ ghi lên tệp tin.

size: kích thước của mỗi phần tử.

f: con trỏ tệp tin đã được mở.

```
1  #include <stdio.h>
2
3  int main(){
4      FILE *f;
5      float s=12.34;
6      if((f=fopen("bai.txt","wb"))==NULL)
7      {
8          printf("khong mo duoc tep\n");
9          return 1;
10     }
11     fwrite(&s, sizeof(float), 1, f);
12     fclose(f);
13     return 0;
14 }
```

### 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 tệp tin.

n: số phần tử được đọc từ tệp tin.

size: kích thước của mỗi phần tử.

f: con trỏ tệp tin đã được

## CHƯƠNG II: XÂY DỰNG PHẦN MỀM QUẢN LÝ SINH VIÊN

### I. 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.

### II. 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
  - Hiện 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 hiện 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

## 2. Thực hiện các công việc cần làm.

### 2.1 Tạo cấu trúc sinh viên.

```

9 struct HoTen {
10     char ho[20];
11     char dem[21];
12     char ten[20];
13 };
14
15 struct DiemMH {
16     float mon1;
17     float mon2;
18     float mon3;
19     float tbc;
20 };
21
22 struct SinhVien {
23     int ma;
24     struct HoTen hoVaTen;
25     int tuoi;
26     char gioiTinh[10];
27     struct DiemMH diem;
28 };

```

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

### 2.2 Nhập tên sinh viên và in ra màn hình.

```

50 struct SinhVien nhapSV() {
51     struct SinhVien sv;
52     printf("Nhap ma: ");
53     scanf("%d", &sv.ma);
54     nhapHoTen(&sv.hoVaTen);
55     printf("Tuoi: ");
56     scanf("%d", &sv.tuoi);
57     printf("Gioi tinh: ");
58     scanf("%s", sv.gioiTinh);
59     nhapDiem(&sv.diem);
60     return sv;
61 }
62
63 void hienThiTTSV(struct SinhVien sv) {
64     printf("%-10d %-10s %-20s %-10s %-10d %-10s %-10.2f %-10.2f %-10.2f %-10.2f\n",
65         sv.ma, sv.hoVaTen.ho, sv.hoVaTen.dem, sv.hoVaTen.ten, sv.tuoi, sv.gioiTinh,
66         sv.diem.mon1, sv.diem.mon2, sv.diem.mon3, sv.diem.tbc);
67 }

```

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.

### 3. Sắp xếp sinh viên theo tên.

```
--  
69 void sapXepTheoTen(struct SinhVien* ds, int slsv) {  
70     int i, j;  
71     for(i = 0; i < slsv - 1; i++) {  
72         for(j = slsv - 1; j > i; j --) {  
73             if(strcmp(ds[j].hoVaTen.ten, ds[j-1].hoVaTen.ten) < 0) {  
74                 struct SinhVien sv = ds[j];  
75                 ds[j] = ds[j - 1];  
76                 ds[j - 1] = sv;  
77             }  
78         }  
79     }  
80 }
```

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.

### 4. Sắp xếp điểm trung bình 3 môn học.

```
82 void sapXepTheoDiem(struct SinhVien* ds, int slsv) {  
83     int i, j;  
84     for(i = 0; i < slsv - 1; i++) {  
85         for(j = slsv - 1; j > i; j --) {  
86             if(ds[j].diem.tbc > ds[j - 1].diem.tbc) {  
87                 struct SinhVien sv = ds[j];  
88                 ds[j] = ds[j - 1];  
89                 ds[j - 1] = sv;  
90             }  
91         }  
92     }  
93 }
```

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

## 5. Tìm kiếm và hiển thị sinh viên trong danh sách.

```
95 void timTheoTen(struct SinhVien* ds, int slsv) {
96     char ten[20];
97     printf("Nhap ten: ");
98     scanf("%s", ten);
99     hienThiTenCot();
100    int i, timSV = 0;
101    for(i = 0; i < slsv; i++) {
102        if(strcmp(ten, ds[i].hoVaTen.ten) == 0) {
103            hienThiTTSV(ds[i]);
104            timSV = 1;
105        }
106    }
107    if(timSV == 0) {
108        printf("Khong co sinh vien %s trong danh sach!\n", ten);
109    }
110 }
```

Ta sử dụng hàm strcmp() để so sánh chuỗi mà được trả 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 khác không thì tên không tồn tại trong danh sách.

## 6. Đọc/Ghi thông tin sinh viên vào tệp txt.

**Ghi file:**

```
112 void ghiFile(struct SinhVien* ds, int slsv) {
113     getchar();
114     char fName[26];
115     printf("Nhap ten file: ");
116     gets(fName);
117     FILE* fOut = fopen(fName, "a");
118     int i;
119     for(i = 0; i < slsv; i++) {
120         struct SinhVien sv = ds[i];
121         fprintf(fOut, "%-10d %-10s %-20s %-10s %-10d %-10s %-10.2f %-10.2f %-10.2f %-10.2f\n",
122             sv.ma, sv.hoVaTen.ho, sv.hoVaTen.dem, sv.hoVaTen.ten, sv.tuoi, sv.gioiTinh,
123             sv.diem.mon1, sv.diem.mon2, sv.diem.mon3, sv.diem.tbc);
124     }
125     fclose(fOut);
126 }
```

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:

```
128 void docFile(struct SinhVien* ds, int* slsv) {
129     FILE* fOut = fopen("SV.txt", "r");
130     int i = 0;
131     if(fOut) {
132         for(;;) {
133             struct SinhVien sv;
134             fscanf(fOut, "%10d %10s %20[^\n] %10s %10d %10s %10f %10f %10f %10f\n",
135                 &sv.ma, sv.hoVaTen.ho, sv.hoVaTen demu, sv.hoVaTen.ten, &sv.tuoi, sv.gioiTinh,
136                 &sv.diem.mon1, &sv.diem.mon2, &sv.diem.mon3, &sv.diem.tbc);
137
138             ds[i++] = sv;
139             if(feof(fOut)) { // thoat chuong trinh
140                 break;
141             }
142         }
143     }
144
145     fclose(fOut);
146     *slsv = i;
147 }
```

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.

# CHƯƠNG III. TỔNG KẾT

## I. Kết quả cần đạt và hướng giải phát triển.

### 1. Kết quả cần đạt.

- Hiểu được danh sách liên kết đơn là gì? và cách sử dụng của danh sách liên kết đơn
- Hiểu được các thuật toán cơ bản và cách sử dụng
- Biết được file là gì? và các thao tác làm việc với file

Từ những ý trên sẽ hỗ trợ cho việc hiểu cách vận hành và cách sử dụng của chương trình quản lý sinh viên.

### 2. Hướng phát triển.

Sau khi báo cáo về bài tập lớn thì nhóm sẽ cố gắng cải tiến thêm vào chương trình nhiều tính năng mới hơn nhằm đáp ứng nhu cầu quản lý và sẽ tạo ra nhiều chương

trình quản lý hơn không chỉ có mình quản lý sinh viên để có thể đáp ứng mọi nhu cầu khi cần một chương trình quản lý nào đó.

## **II. Các nguồn học tập tham khảo**

- Giáo trình lập trình nâng cao
- [sinhvientot.net](http://sinhvientot.net)
- [cunglaptrinh.blogspot.com](http://cunglaptrinh.blogspot.com)
- [voer.edu.vn](http://voer.edu.vn)
- [nguyenvanhieu.vn](http://nguyenvanhieu.vn)
- [vietjack.com](http://vietjack.com)
- [nguyenvanquan7826.wordpress.com](http://nguyenvanquan7826.wordpress.com)
- [sites.google.com](http://sites.google.com)
- [cpp.daynhauhoc.com](http://cpp.daynhauhoc.com)