**TRƯỜNG ĐẠI HỌC GIAO THÔNG VẬN TẢI**

**PHÂN HIỆU TẠI TP. HỒ CHÍ MINH**

**BỘ MÔN CÔNG NGHỆ THÔNG TIN**



**BÁO CÁO BÀI TẬP LỚN LẬP TRÌNH NÂNG CAO**

Giảng viên hướng dẫn: TRẦN THỊ DUNG

Sinh viên thực hiện: Võ Hoàng Minh Huy

Lớp: CQ.59.CNTT

**LỜI CẢM ƠN**

Em xin gửi lời cảm ơn chân thành đến quý thầy, cô bộ môn Công nghệ thông tin trường Đại học Giao thông Vận tải – Phân hiệu tại thành phố Hồ Chí Minh đã quan tâm hướng dẫn truyền đạt học những kiến thức và kinh nghiệm cho em trong suốt thời gian học tập. Em xin chúc quý thầy cô thật nhiều sức khỏe và luôn đạt được thành công trong cuộc sống. một thời gian nỗ lực thực hiện thì đề tài cũng đã hoàn thành. Nhưng không sao tránh khỏi những sai sót do em còn chưa có nhiều kinh nghiệm thực tế. Em kính mong nhận được sự góp ý và nhận xét từ quý thầy, cô để em có thể hoàn thiện và hoàn thành tốt hơn .

Lời sau cùng em một lần nữa kính chúc quý thầy, cô bộ môn Công nghệ thông tin Trường Đại học Giao thông Vận tải – Phân hiệu tại thành phố Hồ Chí Minh thật nhiều sức khỏe và thành công.

*Tp. Hồ Chí Minh, ngày tháng năm*

Sinh viên thực hiện

**Võ Hoàng Minh Huy**

**MỤC LỤC**

[CHƯƠNG I TỔNG QUAN VỀ ĐỀ TÀI 4](#_Toc44451687)

[1.1 Lý do chọn đề tài 4](#_Toc44451688)

[1.2 Mục tiêu nghiên cứu: 4](#_Toc44451689)

[1.3 Cách tiếp cận, phương pháp nguyên cứu: 4](#_Toc44451690)

[CHƯƠNG II CÁC NỘI DUNG LÝ THUYẾT 5](#_Toc44451691)

[2.1 Làm việc với tệp (FILE): 5](#_Toc44451692)

[2.1.1 Khái niệm về tệp tin 5](#_Toc44451693)

[2.1.2 Khai báo sử dụng tệp 7](#_Toc44451694)

[2.1.3 Mở tệp - hàm fopen 7](#_Toc44451695)

[2.1.4 Đóng tệp - hàm fclose 9](#_Toc44451696)

[2.1.5 Đóng tất cả các tệp đang mở- hàm fcloseall 10](#_Toc44451697)

[2.1.6 Chuyển con trỏ chỉ vị về đầu tệp - Hàm rewind 12](#_Toc44451698)

[2.1.7 Vị trí hiện tại của con trỏ chỉ vị - Hàm ftell 13](#_Toc44451699)

[2.1.8 Ghi các mẫu tin lên tệp - hàm fwrite 13](#_Toc44451700)

[2.1.9 Đọc các mẫu tin từ tệp - hàm fread 15](#_Toc44451701)

[2.1.10 Nhập xuất ký tự 16](#_Toc44451702)

[2.1.11 Xoá tệp - hàm unlink 19](#_Toc44451703)

[**2.2 Cấu trúc liên kết đơn** 20](#_Toc44451704)

[2.2.1 Khái niệm liên kết đơn 21](#_Toc44451705)

[2.2.2 Cài đặt danh sách liên kết đơn 22](#_Toc44451706)

[\_Khai báo linked list 22](#_Toc44451707)

[\_Tạo mới 1 Node 22](#_Toc44451708)

[\_Thêm Node vào danh sách liên kết 24](#_Toc44451709)

[2.2.3. Xóa Node khỏi danh sách liên kết 27](#_Toc44451710)

[2.2.4 Lấy giá trị ở vị trí bất kỳ 29](#_Toc44451711)

[2.2.5 Tìm kiếm trong danh sách liên kết 30](#_Toc44451712)

[2.2.6 Duyệt danh sách liên kết 31](#_Toc44451713)

[2.2.7 Một số hàm bổ trợ khác 32](#_Toc44451714)

[2.3 các thuật toán sắp xếp, tìm kiếm 33](#_Toc44451715)

[2.3.1 Thuật toán sắp xếp chèn (Insertion sort) 33](#_Toc44451716)

[2.3.2 Thuật toán sắp xếp chọn (Selection sort) 35](#_Toc44451717)

[2.3.3 Thuật toán sắp xếp nổi bọt 35](#_Toc44451718)

[2.3.4 Counting Sort – Thuật toán sắp xếp đếm phân phối 36](#_Toc44451719)

[2.3.5 Merge Sort – Sắp xếp trộn 37](#_Toc44451720)

[2.3.6 Thuật toán Quick Sort – Sắp xếp nhanh 39](#_Toc44451721)

[III. LỜI KẾT 39](#_Toc44451722)

# CHƯƠNG I TỔNG QUAN VỀ ĐỀ TÀI

## 1.1 Lý do chọn đề tài

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.

Nhận thấy được vấn đề đó em đã tìm hiểu và thực hiện đề tài: **“Xây dựng chương trình quản lý sách bằng ngôn ngữ C”**. Nhằm giúp đở người sử dụng tiện lợi trong việc sắp xếp, quản lý các đầu sách của bản thân.

## 1.2 Mục tiêu nghiên cứu:

-Xây dựng chương trình quản lý sách bằng ngôn ngữ C

-Giao diện chương trình thân thiện bắt mắt dễ sử dụng

## 1.3 Cách tiếp cận, phương pháp nguyên cứu:

- Tìm hiểu các thông tin cần thiết, liên quan đến đề tài

- Tổng hợp tài liệu liên quan đến đề tài.

- Tham khảo trang chủ các ngôn ngữ lập trình C

- Tham khảo giáo trình và slide bài giảng

- Xây dựng phần mềm dựa theo phương pháp từ trên xuống dưới và sử dụng mô hình thác nước để xây dựng.

# CHƯƠNG II CÁC NỘI DUNG LÝ THUYẾT

## 2.1 Làm việc với tệp (FILE):

### 2.1.1 Khái niệm về tệp tin

* Tệp tin hay tệp dữ liệu là một tập hợp các dữ liệu có liên quan với nhau và có cùng một kiểu được nhóm lại với nhau thành một dãy. Chúng thường được chứa trong một thiết bị nhớ ngoài của mấy tính (đĩa mềm, đĩa cứng...) dưới một cái tên nào đó.
* Tên tiếng Anh của tệp là **file,**nó được dùng để chỉ ra một hộp đựng các phiếu hay thẻ ghi của thư viện. Một hình ảnh rõ nét giúp ta hình dung ra tệp là tủ phiếu của thư viện. Một hộp có nhiều phiếu giống nhau về hình thức và tổ chức, song lại khác nhau về nội dung. ở đây, tủ phiếu là tệp, các lá phiếu là các thành phần của tệp. Trong máy tính, một đĩa cứng hoặc một đĩa mềm đóng vai trò chiếc tủ (để chứa nhiều tệp).
* Tệp được chứa trong bộ nhớ ngoài, điều đó có nghĩa là tệp được lưu trữ để dùng nhiều lần và tồn tại ngay cả khi chương trình kết thúc hoặc mất điện. Chính vì lý do trên, chỉ những dữ liệu nào cần lưu trữ ( như hồ sơ chẳng hạn) thì ta nên dùng đến tệp.
* Tệp là một kiểu dữ liệu có cấu trúc. Định nghĩa tệp có phần nào giống mảng ở chỗ chúng đều là tập hợp của các phần tử dữ liệu cùng kiểu, song mảng thường có số phần tử cố định, số phần tử của tệp không được xác định trong định nghĩa.
* Trong C, các thao tác tệp được thực hiện nhờ các hàm thư viện. Các hàm này được chia làm hai nhóm : nhóm 1 và nhóm 2. Các hàm cấp 1 là các hàm nhập / xuất hệ thống, chúng thực hiện việc đọc ghi như DOS. Các hàm cấp 2 làm việc với tệp thông qua một biến con trỏ tệp.
* Do các hàm cấp 2 có nhiều kiểu truy xuất và dễ dùng hơn so với các hàm cấp 1 nên trong các chương trình viết trong C, các hàm cấp 2 hay được sử dụng hơn.
* Một tệp tin dù được xây dựng bằng cách nào đi nữa cũng chỉ đơn giản là một dãy các byte ghi trên đĩa (có giá trị từ 0 đến 255). Số byte của dãy chính là độ dài của tệp.
* Có hai kiểu nhập xuất dữ liệu lên tệp : Nhập xuất nhị phân và nhập xuất văn bản.

**Nhập xuất nhị phân**

* Dữ liệu ghi lên tệp theo các byte nhị phân như bộ nhớ, trong quá trình nhập xuất, dữ liệu không bị biến đổi.
* Khi đọc tệp, nếu gặp cuối tệp thì ta nhận được mã kết thúc tệp EOF ( được định nghĩa trong stdio.h bằng -1) và hàm feof cho giá trị khác 0.

**Nhập xuất văn bản**

* Kiểu nhập xuất văn bản chỉ khác kiểu nhị phân khi xử lý ký tự chuyển dòng ( mã 10) và ký tự mã 26. Đối với các ký tự khác, hai kiểu đều đọc ghi như nhau.
* Mã chuyển dòng :

Khi ghi, một ký tự LF (mã 10) được chuyển thành 2 ký tự CR (mã 13) và LF

Khi đọc, 2 ký tự liên tiếp CR và LF trên tệp chỉ cho ta một ký tự LF

**Mã kết thúc tệp**

Trong khi đọc, nếu gặp ký tự có mã 26 hoặc cuối tệp thì ta nhận được mã kết thúc tệp EOF ( bằng -1) và hàm feof(fp) cho giá trị khác 0 ( bằng 1).

Khai báo sử dụng tệp - một số hàm thường dùng khi thao tác trên tệp

### 2.1.2 Khai báo sử dụng tệp

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 **,**).

Ví dụ

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

### 2.1.3 Mở tệp - hàm fopen

**Cấu trúc ngữ pháp của hàm**

FILE \*fopen(const char \*tên\_tệp, const char \*kiểu);

**Nguyên hàm trong : stdio.h .**

Trong đó :

Đối thứ nhất là tên tệp, đối thứ hai là kiểu truy nhập.

**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 sau chỉ ra các giá trị của kiểu:

|  |  |  |  |
| --- | --- | --- | --- |
| Các giá trị của kiểu | | | |
|  | Tên kiểu | ý nghĩa |  |
|  | "r" "rt" | Mở một tệp để đọc theo kiểu văn bản. Tệp cần đọc phải đã tồn tại, nếu không sẽ có lỗi |  |
|  | "w" "wt" | Mở một tệp để ghi theo kiểu văn bản. Nếu tệp đã tồn tại thì nó sẽ bị xoá. |  |
|  | "a" "at" | Mở một tệp để ghi bổ xung theo kiểu văn bản. Nếu tệp chưa tồn tại thì tạo tệp mới. |  |
|  | "rb" | Mở một tệp để đọc theo kiểu nhị phân. Tệp cần đọc phải đã tồn tại, nếu không sẽ có lỗi. |  |
|  | "wb" | Mở một tệp mới để ghi theo kiểu nhị phân. Nếu tệp đã tồn tại thì nó sẽ bị xoá. |  |
|  | "ab" | Mở một tệp để ghi bổ xung theo kiểu nhị phân. Nếu tệp chưa tồn tại thì tạo tệp mới. |  |
|  | "r+" "r+t" | Mở một tệp để đọc/ghi theo kiểu văn bản. Tệp cần đọc phải đã tồn tại, nếu không sẽ có lỗi |  |
|  | "w+" "w+t" | Mở một tệp để đọc/ghi theo kiểu văn bản. Nếu tệp đã tồn tại thì nó sẽ bị xoá. |  |
|  | "a+" "a+t" | Mở một tệp để đọc/ghi bổ xung theo kiểu văn bản. Nếu tệp chưa tồn tại thì tạo tệp mới. |  |
|  | "r+b" | Mở một tệp để đọc/ghi theo kiểu nhị phân. Tệp cần đọc phải đã tồn tại, nếu không sẽ có lỗi. |  |
|  | "w+b" | Mở một tệp mới để đọc/ghi theo kiểu nhị phân. Nếu tệp đã tồn tại thì nó sẽ bị xoá. |  |
|  | "a+b" | Mở một tệp để đọc/ghi bổ xung theo kiểu nhị phân. Nếu tệp chưa tồn tại thì tạo tệp mới. |  |

Trong các kiểu đọc ghi, ta nên làm sạch vùng đệm trước khi chuyển từ đọc sang ghi hoặc ngược lại. Ta sẽ đề cập đến các hàm với tính năng xoá sau này.

Ví dụ :

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

### 2.1.4 Đóng tệp - 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 xoá 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);

### 2.1.5 Đóng tất cả các tệp 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 lệnh thành công, hàm sẽ cho giá trị 0, trái lại nó cho hàm EOF.

Ví dụ :

fflush(f);

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

**Cấu trúc ngữ pháp của hàm**

int fflushall(void);

**Nguyên hàm trong : stdio.h .**

**Công dụng**

Dùng làm sạch vùng đệm của tất cả các tệp đang mở. Nếu lệnh 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 0 nếu không có lỗi, trái lại hàm cho giá trị khác 0.

Kiểmtra 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 chuyên con trỏ chỉ vị

### 2.1.6 Chuyển con trỏ chỉ vị về đầu tệp - Hàm rewind

**Cấu trúc ngữ pháp**

void rewind(FILE \*fp);

**Nguyên hàm trong : stdio.h .**

Trong đó fp là con trỏ tệp.

**Công dụng**

Chuyển con trỏ chỉ vị của tệp fp về đầu tệp. Khi đó việc nhập xuất trên tệp fp được thực hiện từ đầu.

Ví dụ :

rewind(f);

Chuyển con trỏ chỉ vị trí cần thiết - Hàm fseek

**Cấu trúc ngữ pháp**

int fseek(FILE \*fp, long sb, int xp);

**Nguyên hàm trong : stdio.h .**

Trong đó

fp là con trỏ tệp.

sb là số byte cần di chuyển.

xp cho biết vị trí xuất phát mà việc dịch chuyển được bắt đầu từ đó.

xp có thể nhận các giá trị sau :

xp=SEEK\_SET hay 0 : Xuất phát từ đầu tệp.

xp=SEEK\_CUR hay 1: Xuất phát từ vị trí hiện tại của con trỏ chỉ vị.

xp=SEEK\_END hay 2 : Xuất phát từ cuối tệp.

**Công dụng**

Chuyển con trỏ chỉ vị của tệp fp về vị trí xác định bởi xp qua một số byte xác định bằng giá trị tuyệt đối của sb. Chiều di chuyển là về cuối tệp nếu sb dương, trái lại nó sẽ di chuyển về đầu tệp. Khi thành công, hàm trả về giá trị 0. Khi có lỗi hàm trả về giá trị khác không.

Không nên dùng fseek trên tệp tin văn bản, do sự chuyển đổi ký tự sẽ làm cho việc định vị thiếu chính xác.

Ví dụ :

fseek(stream, SEEK\_SET, 0);

### 2.1.7 Vị trí hiện tại của con trỏ chỉ vị - Hàm ftell

**Cấu trúc ngữ pháp**

int ftell(FILE \*fp);

**Nguyên hàm trong : stdio.h .**

Trong đó

fp là con trỏ tệp.

**Công dụng**

Hàm cho biết vị trí hiện tại của con trỏ chỉ vị (byte thứ mấy trên tệp **fp**) khi thành công. Số thứ tự tính từ 0. Trái lại hàm cho giá trị -1L.

Ví dụ

Sau lệnh fseek(fp,0,SEEK\_END);

ftell(fp) cho giá trị 3.

Sau lệnh fseek(fp,-1,SEEK\_END);

ftell(fp) cho giá trị 2.

### 2.1.8 Ghi các mẫu tin lên tệp - hàm fwrite

**Cấu trúc ngữ pháp của hàm**

int fwrite(void \*ptr, int size, int n, FILE \*fp);

**Nguyên hàm trong : stdio.h .**

Trong đó :

**ptr** là con trỏ trỏ tới vùng nhớ chứa dữ liệu cần ghi.

**size**là kích thước của mẫu tin theo byte

**n**là số mẫu tin cần ghi

**fp** là con trỏ tệp

**Công dụng**

Hàm ghi n mẫu tin kích thước **size**byte từ vùng nhớ **ptr**lên tệp **fp**.

Hàm sẽ trả về một giá trị bằng số mẫu tin thực sự ghi được.

Ví dụ :

#include "stdio.h"

struct mystruct

{

int i;

char ch;

};

main()

{

FILE \*stream;

struct mystruct s;

stream = fopen("TEST.TXT", "wb") /\* Mở tệp TEST.TXT \*/

s.i = 0;

s.ch = 'A';

fwrite(&s, sizeof(s), 1, stream); /\* Viết cấu trúc vào tệp \*/

fclose(stream); /\* Đóng tệp \*/

return 0;

}

### 2.1.9 Đọc các mẫu tin từ tệp - hàm fread

**Cấu trúc ngữ pháp của hàm**

int fread(void \*ptr, int size, int n, FILE \*fp);

**Nguyên hàm trong : stdio.h .**

Trong đó :

**ptr** là con trỏ trỏ tới vùng nhớ chứa dữ liệu cần ghi.

**size**là kích thước của mẫu tin theo byte

**n**là số mẫu tin cần ghi

**fp** là con trỏ tệp

**Công dụng**

Hàm đọc n mẫu tin kích thước **size**byte từ tệp **fp**lên lên vùng nhớ**ptr**.

Hàm sẽ trả về một giá trị bằng số mẫu tin thực sự đọc được.

Ví dụ

#include "string.h"

#include "stdio.h"

main()

{

FILE \*stream;

char msg[] = "Kiểm tra";

char buf[20];

stream = fopen("DUMMY.FIL", "w+");

/\* Viết vài dữ liệu lên tệp \*/

fwrite(msg, strlen(msg)+1, 1, stream);

/\* Tìm điểm đầu của file \*/

fseek(stream, SEEK\_SET, 0);

/\* Đọc số liệu và hiển thị \*/

fread(buf, strlen(msg)+1, 1, stream);

printf("%s\n", buf);

fclose(stream);

return 0;

}

### 2.1.10 Nhập xuất ký tự

Các hàm putc và fputc

**Cấu trúc ngữ pháp**

int putc(int ch, FILE \*fp);

int fputc(int ch, FILE \*fp);

**Nguyên hàm trong : stdio.h .**

Trong đó :

ch là một giá trị nguyên

fp là một con trỏ tệp.

**Công dụng**

Hàm ghi lên tệp fp một ký tự có mẫu bằng

m=ch % 256.

**ch** được xem là một giá trị nguyên không dấu. Nếu thành công hàm cho mã ký tự được ghi, trái lại cho EOF

Ví dụ :

#include "stdio.h"

main()

{

char msg[] = "Hello world\n";

int i = 0;

while (msg[i])

putc(msg[i++], stdout); /\* stdout thiết bị ra chuẩn - Màn hình\*/

return 0;

}

Các hàm getc và fgettc

**Cấu trúc ngữ pháp**

int gretc(FILE \*fp);

int fputc(FILE \*fp);

**Nguyên hàm trong : stdio.h .**

Trong đó :

fp là một con trỏ tệp.

**Công dụng**

Hàm đọc một ký tự từ tệp fp. Nếu thành công hàm sẽ cho mã đọc được ( có giá trị từ 0 đến 255). Nếu gặp cuối tệp hay có lỗi hàm sẽ trả về EOF.

Trong kiểu văn bản, hàm đọc một lượt cả hai mã 13, 10 và trả về giá trị 10. Khi gặp mã 26 hàm sẽ trả về EOF.

Ví dụ**:**

#include "string.h"

#include "stdio.h"

#include "conio.h"

main()

{

FILE \*stream;

char string[] = "Kiem tra";

char ch;

/\* Mở tệp để cập nhật\*/

stream = fopen("DUMMY.FIL", "w+");

/\*Viết một xâu ký tự vào tệp \*/

fwrite(string, strlen(string), 1, stream);

/\* Tìm vị trí đầu của tệp \*/

fseek(stream, 0, SEEK\_SET);

do

{

/\* Đọc một ký tự từ tệp \*/

ch = fgetc(stream);

/\* Hiển thị ký tự \*/

putch(ch);

} while (ch != EOF);

fclose(stream);

return 0;

}

### 2.1.11 Xoá tệp - hàm unlink

**Cấu trúc ngữ pháp**

int unlink(const char \*tên\_tệp)

**Nguyên hàm trong : dos.h, io.h, stdio.h**

Trong đó

**tên\_tệp**là tên của tệp cần xoá.

**Công dụng**

Dùng để xoá một tệp trên đĩa. Nếu thành công, hàm cho giá trị 0, trái lại hàm cho giá trị EOF.

Ví dụ :

#include <stdio.h>

#include <io.h>

int main(void)

{

FILE \*fp = fopen("junk.jnk","w");

int status;

fprintf(fp,"junk");

status = access("junk.jnk",0);

if (status == 0)

printf("Tệp tồn tại\n");

else

printf("Tệp không tồn tại\n");

fclose(fp);

unlink("junk.jnk");

status = access("junk.jnk",0);

if (status == 0)

printf("Tệp tồn tại\n");

else

printf("Tệp không tồn tại\n");

return 0;

}

**2.2 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:

|  |  |  |
| --- | --- | --- |
| **Nội dung** | **Mảng** | **Danh sách liên kết** |
| Kích thước | * Kích thước cố định * Cần chỉ rõ kích thước trong khi khai báo | * *Kích thước thay đổi trong quá trình thêm/ xóa phần tử* * *Kích thước tối đa phụ thuộc vào bộ nhớ* |
| Cấp phát bộ nhớ | * Tĩnh: Bộ nhớ được cấp phát trong quá trình biên dịch | * *Động: Bộ nhớ được cấp phát trong quá trình chạy* |
| Thứ tự & sắp xếp | * Được lưu trữ trên một dãy ô nhớ liên tục | * *Được lưu trữ trên các ô nhớ ngẫu nhiên* |
| Truy cập | * *Truy cập tới phần tử ngẫu nhiên trực tiếp bằng cách sử dụng chỉ số mảng: O(1)* | * Truy cập tới phần tử ngẫu nhiên cần phải duyệt từ đầu/cuối đến phần tử đó: O(n) |
| Tìm kiếm | * *Tìm kiếm tuyến tính hoặc tìm kiếm nhị phân* | * Chỉ có thể tìm kiếm tuyến tính |

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

### 2.2.1 Khái niệm 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.

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:



### 2.2.2 Cài đặt danh sách liên kết đơn

### \_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.

struct LinkedList{

int data;

struct LinkedList \*next;

};

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 1 Node

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

typedef struct LinkedList \*node; //T? gi? dùng ki?u d? li?u LinkedList có th? thay b?ng node cho ng?n g?n

node CreateNode(int value){

node temp; // declare a node

temp = (node)malloc(sizeof(struct LinkedList)); // C?p phát vùng nh? dùng malloc()

temp->next = NULL;// Cho next tr? t?i NULL

temp->data = value; // Gán giá tr? cho Node

return temp;//Tr? v? node m?i dã có giá tr?

}

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.

### \_Thêm Node vào danh sách liên kết

#### Thêm vào đầu

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

node AddHead(node head, int value){

node temp = CreateNode(value); // Kh?i t?o node temp v?i data = value

if(head == NULL){

head = temp; // //N?u linked list dang tr?ng thì Node temp là head luôn

}else{

temp->next = head; // Tr? next c?a temp = head hi?n t?i

head = temp; // Ð?i head hi?n t?i = temp(Vì temp bây gi? là head m?i mà)

}

return head;

}

#### 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).

node AddTail(node head, int value){

node temp,p;// Khai báo 2 node t?m temp và p

temp = CreateNode(value);//G?i hàm createNode d? kh?i t?o node temp có next tr? t?i NULL và giá tr? là value

if(head == NULL){

head = temp; //N?u linked list dang tr?ng thì Node temp là head luôn

}

else{

p = head;// Kh?i t?o p tr? t?i head

while(p->next != NULL){

p = p->next;//Duy?t danh sách liên k?t d?n cu?i. Node cu?i là node có next = NULL

}

p->next = temp;//Gán next c?a th?ng cu?i = temp. Khi dó temp s? là th?ng cu?i(temp->next = NULL mà)

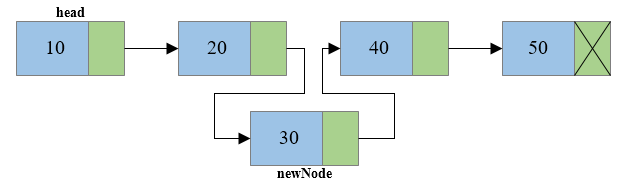
}

return head;

}

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

#### Thêm vào vị trí bất kỳ



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

node AddAt(node head, int value, int position){

if(position == 0 || head == NULL){

head = AddHead(head, value); // N?u v? trí chèn là 0, t?c là thêm vào d?u

}else{

// B?t d?u tìm v? trí c?n chèn. Ta s? dùng k d? d?m cho v? trí

int k = 1;

node p = head;

while(p != NULL && k != position){

p = p->next;

++k;

}

if(k != position){

// N?u duy?t h?t danh sách lk r?i mà v?n chua d?n v? trí c?n chèn, ta s? m?c d?nh chèn cu?i

// N?u b?n không mu?n chèn, hãy thông báo v? trí chèn không h?p l?

head = AddTail(head, value);

// printf("Vi tri chen vuot qua vi tri cuoi cung!\n");

}else{

node temp = CreateNode(value);

temp->next = p->next;

p->next = temp;

}

}

return head;

}

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).

### 2.2.3. 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.

node DelHead(node head){

if(head == NULL){

printf("\nCha co gi de xoa het!");

}else{

head = head->next;

}

return head;

}

#### Xóa cuối

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

*node DelTail(node head){*

*if (head == NULL || head->next == NULL){*

*return DelHead(head);*

*}*

*node p = head;*

*while(p->next->next != NULL){*

*p = p->next;*

*}*

*p->next = p->next->next; // Cho next b?ng NULL*

*// Ho?c vi?t p->next = NULL cung du?c*

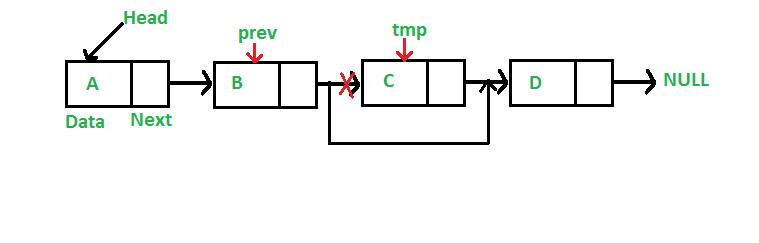
*return head;*

*}*

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

#### Xóa ở vị trí bất kỳ

Việc xóa ở vị trí bất kỳ cũng khá giống xóa ở cuối kia. Đơn giản là chúng ta bỏ qua một phần tử, như ảnh sau:



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

node DelAt(node head, int position){

if(position == 0 || head == NULL || head->next == NULL){

head = DelHead(head); // N?u v? trí chèn là 0, t?c là thêm vào d?u

}else{

// B?t d?u tìm v? trí c?n chèn. Ta s? dùng k d? d?m cho v? trí

int k = 1;

node p = head;

while(p->next->next != NULL && k != position){

p = p->next;

++k;

}

if(k != position){

// N?u duy?t h?t danh sách lk r?i mà v?n chua d?n v? trí c?n chèn, ta s? m?c d?nh xóa cu?i

// N?u b?n không mu?n xóa, hãy thông báo v? trí xóa không h?p l?

head = DelTail(head);

// printf("Vi tri xoa vuot qua vi tri cuoi cung!\n");

}else{

p->next = p->next->next;

}

}

return head;

}

### 2.2.4 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.

int Get(node head, int index){

int k = 0;

node p = head;

while(p->next != NULL && k != index){

++k;

p = p->next;

}

return p->data;

}

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

### 2.2.5 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.

int Search(node head, int value){

int position = 0;

for(node p = head; p != NULL; p = p->next){

if(p->data == value){

return position;

}

++position;

}

return -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:

node DelByVal(node head, int value){

int position = Search(head, value);

while(position != -1){

DelAt(head, position);

position = Search(head, value);

}

return head;

}

### 2.2.6 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.

void Traverser(node head){

printf("\n");

for(node p = head; p != NULL; p = p->next){

printf("%5d", p->data);

}

}

### 2.2.7 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.

node InitHead(){

node head;

head = NULL;

return head;

}

#### Hàm lấy số phần tử của DSLK

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

int Length(node head){

int length = 0;

for(node p = head; p != NULL; p = p->next){

++length;

}

return length;

}

#### Hàm nhập danh sách liên kết

node Input(){

node head = InitHead();

int n, value;

do{

printf("\nNhap so luong phan tu n = ");

scanf("%d", &n);

}while(n <= 0);

for(int i = 0; i < n; ++i){

printf("\nNhap gia tri can them: ");

scanf("%d", &value);

head = AddTail(head, value);

}

return head;

}

## 2.3 các thuật toán sắp xếp, tìm kiếm

### 2.3.1 Thuật toán sắp xếp chèn (Insertion sort)

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.Description: A calculator next to a keyboard

Description automatically generated

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.

### 2.3.2 Thuật toán sắp xếp chọn (Selection sort)

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

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.

A screenshot of a cell phone

Description automatically generated

### 2.3.3 Thuật toán sắp xếp nổi bọt

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.

Description: A screenshot of a social media post

Description automatically generated

### 2.3.4 Counting Sort – Thuật toán sắp xếp đếm phân phối

Counting sort là một [thuật toán sắp xếp](https://nguyenvanhieu.vn/tag/thuat-toan-sap-xep/) cực nhanh một mảng các phần tử mà mỗi phần tử là các số nguyên không âm; Hoặc là một danh sách các ký tự được ánh xạ về dạng số để sort theo bảng chữ cái. Counting sort là một thuật toán sắp xếp các con số nguyên không âm, không dựa vào so sánh. Trong khi các thuật toán sắp xếp tối ưu sử dụng so sánh có độ phức tạp O(nlogn) thì Counting sort chỉ cần O(n) nếu độ dài của danh sách không quá nhỏ so với phần tử có giá trị lớn nhất.

### 2.3.5 Merge Sort – Sắp xếp trộn

Giống như Quick sort, Merge sort là một thuật toán chia để trị. Thuật toán này chia mảng cần sắp xếp thành 2 nửa. Tiếp tục lặp lại việc này ở các nửa mảng đã chia. Sau cùng gộp các nửa đó thành mảng đã sắp xếp. Hàm merge() được sử dụng để gộp hai nửa mảng. Hàm merge(arr, l, m, r) là tiến trình quan trọng nhất sẽ gộp hai nửa mảng thành 1 mảng sắp xếp, các nửa mảng là arr[l…m] và arr[m+1…r] sau khi gộp sẽ thành một mảng duy nhất đã sắp xếp.

Hãy xem ý tưởng triển khai code dưới đây để hiểu hơn

mergeSort(arr[], l, r)

If r > l

1. Tìm chỉ số nằm giữa mảng để chia mảng thành 2 nửa:

middle m = (l+r)/2

2. Gọi đệ quy hàm mergeSort cho nửa đầu tiên:

mergeSort(arr, l, m)

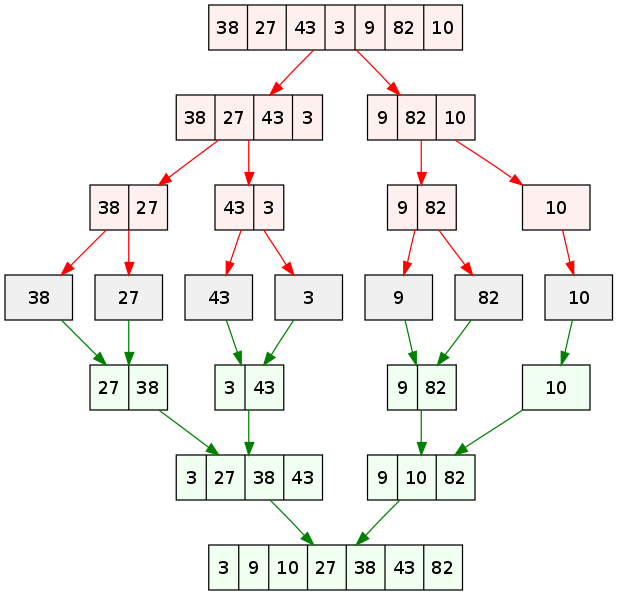
3. Gọi đệ quy hàm mergeSort cho nửa thứ hai:

mergeSort(arr, m+1, r)

4. Gộp 2 nửa mảng đã sắp xếp ở (2) và (3):

merge(arr, l, m, r)

Hình ảnh dưới đây từ wikipedia sẽ hiển thị cho bạn toàn bộ sơ đồ tiến trình của thuật toán merge sort cho mảng {38, 27, 43, 3, 9, 82, 10}. Nếu nhìn kỹ hơn vào sơ đồ này, chúng ta có thể thấy mảng ban đầu được lặp lại hành động chia cho tới khi kích thước các mảng sau chia là 1. Khi kích thước các mảng con là 1, tiến trình gộp sẽ bắt đầu thực hiện gộp lại các mảng này cho tới khi hoàn thành và chỉ còn một mảng đã sắp xếp.

[](https://nguyenvanhieu.vn/wp-content/uploads/2018/07/thuat-toan-sap-xep-merge-sort-minh-hoa-code-su-dung-c.png)

### 2.3.6 Thuật toán Quick Sort – Sắp xếp nhanh

**A close up of a map

Description automatically generated**Giống như Merge sort, thuật toán sắp xếp quick sort là một thuật toán chia để trị( Divide and Conquer algorithm). Nó chọn một phần tử trong mảng làm điểm đánh dấu(pivot). Thuật toán sẽ thực hiện chia mảng thành các mảng con dựa vào pivot đã chọn. Việc lựa chọn pivot ảnh hưởng rất nhiều tới tốc độ sắp xếp. Nhưng máy tính lại không thể biết khi nào thì nên chọn theo cách nào. Dưới đây là một số cách để chọn pivot thường được sử dụng:

1. Luôn chọn phần tử đầu tiên của mảng.
2. Luôn chọn phần tử cuối cùng của mảng. (Được sử dụng trong bài viết này)
3. Chọn một phần tử random.
4. Chọn một phần tử có giá trị nằm giữa mảng(median element).

# III. LỜI KẾT

Do thời gian nguyên cứu, tìm hiểu, phân tích thiết kế và xây dựng ứng tương đối hạn chế nên không thể chánh khỏi những sai xót. Nếu có thời gian và điều kiện sau này em muốn phát triển thêm cho đề tài để hoàn thiện một cách tốt nhất. Xin cảm ơn quý thầy cô đã theo dõi.