# CHƯƠNG X: XỬ LÝ FILE TRONG C

### File trong C là gì ?

Trong thế giới máy tính, file là **một loại “Tài liệu” nhằm lưu dữ dữ liệu và thông tin**. Có rất nhiều kiểu file khác nhau nhằm lưu dữ các kiểu dữ liệu và thông tin khác nhau, ví dụ như file text, file Excel, file Json, file XML, hay là file CSV v.v…

Để có thể thao tác với các loại file này, người dùng thông thường cần các phần mềm chuyên dùng để mở, đọc ghi và lưu chúng. Tuy nhiên thì với các lập trình viên như chúng ta thì bằng cách sử dụng sử dụng ngôn ngữ C, chúng ta cũng có thể dễ dàng thực hiện các thao tác tương tự với file như vậy.

Cũng giống như mảng, chuỗi hay biến thì file trong C là một loại dữ liệu trong chương trình, và một file khi nhập vào chương trình cũng sẽ được lưu giữ tại một vùng nào đó trên bộ nhớ máy tính. Tuy nhiên khác với các loại dữ liệu khác có thể gán vào một biến để xử lý, thì để thao tác với file, chúng ta cần tạo ra một thực thể của kiểu cấu trúc FILE để chứa thông tin của file cần thao tác, sau đó sử dụng tới một con trỏ để chỉ đến vị trí của thực thể FILE chứa thông tin file đó trên bộ nhớ, qua đó thực hiện các thao tác với file thông qua các hàm có sẵn, với các chức năng cụ thể như đóng, mở, ghi hay lưu file.

### Các kiểu 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

#### 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.

#### 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.

### Xử lý file trong C

Quy trình xử lý file trong C bao gồm các bước sau



#### Tạo con trỏ file

Mỗi file trong chương trình C được xử lý dưới dạng một thực thể của kiểu cấu trúc FILE - một kiểu cấu trúc được quy định sẵn trong ngôn ngữ C phục vụ cho việc xử lý file.

Để thao tác với file, trước tiên chúng ta cần phải tạo một con trỏ chỉ đến thực thể của kiểu cấu trúc FILE chứa thông tin của file đó trên bộ nhớ với cú pháp sau đây:

|  |
| --- |
| *FILE*  \*prt; |

Sau đó, bằng cách truy cập vào địa chỉ này thông qua con trỏ, chúng ta mới có thể thực hiện các thao tác với file.

#### Mở file trong C

Chúng ta có 2 phương pháp để mở file trong C như sau:

* Mở file bằng hàm fopen
* Mở file bằng hàm fopen\_s

##### Mở file trong C bằng hàm fopen

###### Hàm fopen trong C

Hàm fopen trong C là một hàm có sẵn trong thư viện chuẩn, có tác dụng mở file trong C. Tên hàm fopen được viết tắt bởi cụm từ file và open, được dịch theo tiếng Việt chính xác là hàm mở file.

Đây là một hàm không thể thiếu khi chúng ta muốn thao tác xử lý với file trong C.

Chúng ta sử dụng hàm fopen thông thường trong C với cú pháp sau đây:

|  |
| --- |
| fp = fopen(filepath, mode); |

Trong đó:

* fp là con trỏ file dùng để gán kiểu cấu trúc FILE được trả về từ hàm nếu mở file thành công
* filepath là đường dẫn tới file cần mở. Đường dẫn này có thể là đường dẫn tương đối, hoặc là đường dẫn tuyệt đối.
* mode là chế độ mở file.

Hàm fopen sẽ trả về một thực thể được tạo ra từ **kiểu cấu trúc FILE** chứa các thông tin của file đã mở để chúng ta có thể xử lý file. Chúng ta cũng gọi thực thể này là file handle - một đối tượng để xủ lý file trong C. Nếu mở file thất bại, một giá trị NULL sẽ được trả về.

Việc mở file và gán địa chỉ file vào con trỏ thường được tiến hành đồng thời. Ví dụ bạn có thể mở file sample.txt có sẵn bằng hàm fopen() với mode **r** và gán vào con trỏ file **fp** như sau:

|  |
| --- |
| #include <stdio.h  int main(void){      //Khai báo con trỏ file  *FILE* \* fp = NULL;      //Mở file và gán file handle vào con trỏ file      fp = fopen("sample.txt", "r");      return 0;  } |

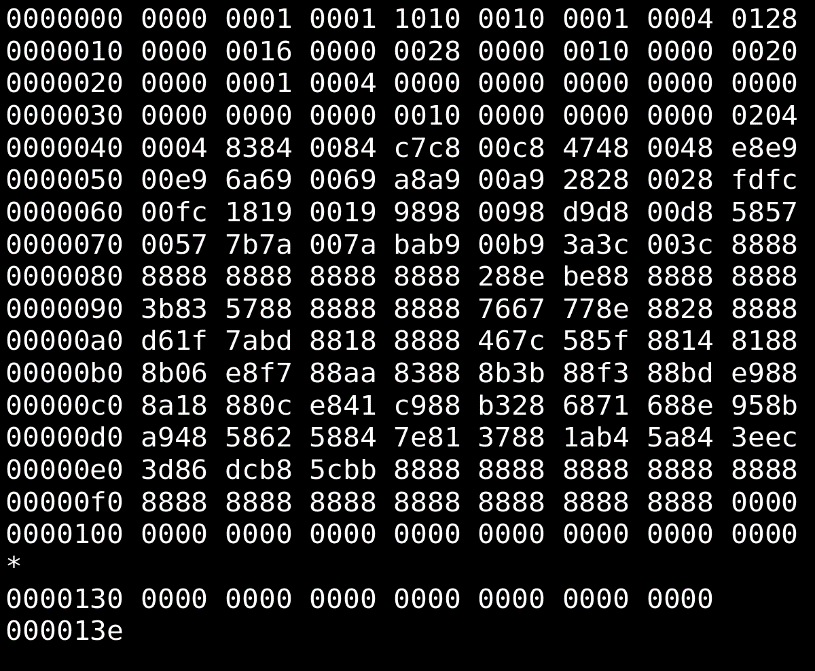
###### Các mode để mở file trong C

Các chế độ có thể sử dụng để mở file bằng hàm fopen trong C như sau:

|  |  |  |
| --- | --- | --- |
| **Mode** | **Xử lý** | **Chức năng** |
| r | Mở để đọc | Chỉ cho phép đọc file Nếu file không tồn tại thì trả về NULL |
| w | Mở để ghi đè | Xoá nội dung cũ và ghi đè nội dung mới Nếu file không tồn tại thì tạo file mới |
| a | Mở để ghi chèn | Ghi chèn nội dung mới vào cuối file Nếu file không tồn tại thì tạo file mới |
| r+ | Mở để đọc và ghi đè | Cho phép cả đọc và ghi đè Nếu file không tồn tại thì trả về NULL |
| w+ | Mở để đọc và ghi đè | Cho phép cả đọc và ghi đè Nếu file không tồn tại thì tạo file mới |
| a+ | Mở để đọc và ghi chèn | Cho phép cả đọc và ghi chèn Nếu file không tồn tại thì tạo file mới |

###### Mở file binary file trong C

Tiếp theo, bằng cách thêm ký tự **b** vào đằng trước tên mode thì chúng ta cũng có thể mở các **binary file** - tập tin nhị phân trong C. Khác với các tập tin văn bản mà chúng ta có thể đọc được các chữ cái, văn bản ghi trong đó, thì Binary File là một tập tin chứa nội dung là các chuỗi nhị phân như sau:



Các mode dùng để mở **binary file** trong C sẽ là:

|  |  |  |
| --- | --- | --- |
| **Mode** | **Xử lý** | **Chức năng** |
| rb | Đọc | Chế độ đọc file binary |
| wb | Ghi đè | Chế độ ghi đè file binary |
| ab | Ghi chèn | Chế độ ghi chèn file binary |

##### Mở file trong C bằng hàm fopen\_s

Hàm fopen\_s trong C là một phiên bản an toàn và bảo mật hơn của hàm fopen, và hàm này cũng giúp chúng ta mở file trong C.

Chúng ta sử dụng hàm fopen\_s trong C với cú pháp sau đây:

|  |
| --- |
| *errno\_t* err = fopen\_s(fp, filepath, mode); |

Trong đó:

* **errno\_t** là kiểu lỗi trong C, và **err** là tên biến để gán giá trị lỗi nếu mở file thất bại
* **fp** là con trỏ file dùng để gán kiểu cấu trúc FILE được trả về từ hàm nếu mở file thành công
* **filepath** là đường dẫn tới file cần mở. Đường dẫn này có thể là đường dẫn tương đối, hoặc là đường dẫn tuyệt đối.
* **mode** là chế độ mở file. Các mode có thể dùng tương tự như với hàm fopen mà chúng ta đã học ở trên.

Hàm fopen\_s sẽ trả về một số tự nhiên biểu thị số hiệu của lỗi khi mở file. Nếu số này bằng 0 thì việc mở file thành công, và nếu số này khác 0 thì chúng ta có thể tìm ra nội dung lỗi bằng cách đối chiếu số hiệu của lỗi với bảng lỗi.

Trong trường hợp không cần xác định tới kiểu lỗi nếu có khi mở file, chúng ta cũng có thể lược bỏ đi việc gán **errno\_t err** mà dùng hàm trực tiếp với cú pháp sau đây:

|  |
| --- |
| fopen\_s(fp, filepath, mode); |

Sau khi mở file thành công, thực thể được tạo ra từ **kiểu cấu trúc FILE** chứa các thông tin của file đã mở sẽ được gán vào con trỏ fp đã khai báo. Và chúng ta có thể thực hiện các thao tác với file thông qua con trỏ file này như bình thường.

Ví dụ cụ thể, chúng ta mở file bằng hàm fopen\_s như sau:

|  |
| --- |
| #include <stdio.h>  int main(void){      //Mở file bằng hàm fopen\_s mà không kiểm tra lỗi  *FILE* \* fp1 = NULL;      fopen\_s(&fp1, "sample.txt", "r");      fclose(fp1);      //Mở file bằng hàm fopen\_s có kiểm tra lỗi  *FILE* \* fp2 = NULL;  *errno\_t* no2;      no2 = fopen\_s(&fp2, "memo.txt", "w");      if (no2 != 0) printf("%s\n", "Mo file thanh cong");      fclose(fp2);  } |

##### Sự khác biệt giữa hàm fopen và fopen\_s

Hàm **fopen\_s** là một phiên bản an toàn và bảo mật hơn của hàm **fopen**. Do đó, hàm này cấm mở một file cùng lúc khi đang mở file đó ở chế độ ghi.

Bởi vậy, nếu chúng ta đã mở một file bằng hàm **fopen\_s** rồi thì chương trình tiếp sẽ kết thúc bất thường vào lần gọi thứ hai của hàm **fopen\_s**.

Sự khác biệt lớn nhất giữa giữa hàm **fopen** và **fopen\_s** cũng chính là điều này. Chúng ta có thể mở một file cùng lúc khi đang mở file đó ở chế độ ghi bằng hàm fopen, nhưng sẽ xảy ra lỗi nếu chúng ta làm việc này với hàm **fopen\_s.**

#### Đọc dữ liệu từ file

Chúng ta có 4 phương pháp để đọc file trong C như sau:

* Đọc từng ký tự trong file bằng hàm fgetc
* Đọc từng dòng file trong C bằng hàm fgets
* Đọc từng dòng file theo định dạng chỉ định bằng hàm fscanf
* Đọc từng dòng file theo định dạng chỉ định bằng hàm fgets kết hợp hàm sscanf

##### Đọc từng ký tự trong file bằng hàm fgetc

Hàm fgetc trong C là một hàm có sẵn trong thư viện chuẩn, có tác dụng đọc từng ký tự trong file chỉ định. Tên hàm fgetc được viết tắt bởi cụm từ file, get và character, được dịch theo tiếng Việt chính xác là hàm đọc từng ký tự trong file.

Chúng ta sử dụng hàm fgetc trong C với cú pháp sau đây:

|  |
| --- |
| int fgetc(*FILE* \* *fp*); |

Trong đó **fp** là con trỏ của file cần đọc, được tạo ra từ việc mở file ở phần trên.

**Hàm fgetc** sẽ trả về mã ASCII của 1 ký tự được đọc ra từ file. Trong trường hợp vị trí đọc ký tự đã là cuối file, hoặc là việc đọc file thất bại thì giá trị EOF sẽ được trả về. Bằng việc in mã ASCII này sang dạng char, chúng ta có thể lấy xuất được ký tự đã đọc.

Ví dụ cụ thể, chúng ta có file **sample.txt** với nội dung sau đây:

**Hello**

**World**

Chúng ta sẽ dùng hàm fopen để mở file, sau đó đọc 1 ký tự trong file bằng hàm fgetc như sau:

|  |
| --- |
| #include <stdio.h>  int main(){  *FILE* \* fp = NULL;      //Mở file bằng hàm fopen      fp= fopen("C:\\Users\\LENOVO\\Desktop\\test.txt","r");      printf("%c\n", fgetc(fp));      printf("%c\n", fgetc(fp));  } |

**Kết quả**

|  |
| --- |
| **H**  **e** |

Bạn có thể thấy chúng ta chỉ có thể đọc từng ký tự từ file trong mỗi lần chạy hàm fgetc mà thôi.

Để có thể đọc tất cả các ký tự từ trong file, chúng ta sẽ cần tạo ra một vòng lặp để đọc từng ký tự từ đầu file cho tới cuối file, cho tới khi kết quả trả về là giá trị **EOF** như sau:

|  |
| --- |
| #include <stdio.h>  int main(void)  {  *FILE* \* fp = NULL;      //Mở file bằn hàm fopen      fp= fopen("C:\\Users\\LENOVO\\Desktop\\test.txt", "r");      char c;      //Đọc từng ký tự từ file cho tới khi gặp EOF      while ((c = fgetc(fp)) != EOF)      {          //Xuất từng ký tự ra màn hình          printf("%c", c);      }      fclose(fp);      return 0;  } |

**Kết quả**

|  |
| --- |
| **Hello**  **World** |

**Lưu ý**, các ký tự xuống dòng cũng được đọc từ file và do đó kết quả xuất ra màn hình cũng được xuống dòng như trên.

##### Đọc từng dòng file trong C bằng hàm fgets

###### Hàm fgets trong C

Hàm fgets trong C là một hàm có sẵn trong thư viện chuẩn, có tác dụng **đọc từng dòng trong file** chỉ định. Tên hàm fgets được viết tắt bởi cụm từ file, get và string, được dịch theo tiếng Việt chính xác là hàm đọc từng dòng trong file.

Chúng ta sử dụng hàm fgets trong C với cú pháp sau đây:

|  |
| --- |
| char \* fgets(char \* *buf*, int *size*, *FILE* \* *fp*); |

Trong đó:

* fp là con trỏ của file cần đọc, được tạo ra từ việc mở file
* buf là con trỏ tới nơi lưu trữ chuỗi đã đọc từ dòng trong file. Thông thường chúng ta chỉ định buf bằng một mảng.
* size là kích thước (số ký tự) lớn nhất có thể đọc từ dòng trong file.

Hàm fgets sẽ trả về con trỏ lưu địa chỉ trên bộ nhớ của chuỗi được đọc từ dòng trong file. Trong trường hợp vị trí đọc ký tự đã là cuối file, hoặc là việc đọc file thất bại thì con trỏ NULL sẽ được trả về.

Lại nữa, hàm fgets sẽ đọc một dòng từ file, và dòng đó được tính từ đầu dòng cho tới khi gặp ký tự xuống dòng **\n**.

###### Ví dụ dùng hàm fgets để đọc từng dòng file trong C

Ví dụ cụ thể, chúng ta có file test.txt với nội dung sau đây:

**1234567**

**89abc**

**def**

Chúng ta sẽ dùng hàm fopen để mở file, sau đó đọc từng dòng trong file bằng hàm fgets như sau:

|  |
| --- |
| #include <stdio.h>  int main(void)  {  *FILE* \* fp = NULL;      char arr[128];      //Mở file bằn hàm fopen      fp= fopen("C:\\Users\\LENOVO\\Desktop\\test.txt", "r");      //Đọc dòng 1      fgets(arr, 128, fp);      printf("%s", arr);        //Đọc dòng 2      fgets(arr, 128, fp);      printf("%s", arr);    } |

**Kết quả**

|  |
| --- |
| **1234567**  **89abc** |

Bạn có thể thấy chúng ta chỉ có thể đọc từng dòng từ file trong mỗi lần chạy hàm fgetc mà thôi.

Để có thể đọc tất cả các dòng từ trong file, chúng ta sẽ cần tạo ra một vòng lặp để đọc từng dòng từ đầu file cho tới cuối file, cho tới khi kết quả trả về là giá trị **NULL** như sau:

|  |
| --- |
| #include <stdio.h>  int main(void)  {  *FILE* \* fp = NULL;      char arr[128];      //Mở file bằn hàm fopen      fp= fopen("C:\\Users\\LENOVO\\Desktop\\test.txt", "r");      //Đọc từng dòng từ file cho tới khi gặp NULL      while (fgets(arr, 128, fp) != NULL)      {          //Xuất từng dòng ra màn hình          printf("%s", arr);      }      fclose(fp);      return 0;  } |

**Kết quả**

|  |
| --- |
| **1234567**  **89abc**  **def** |

###### Đọc từng dòng file khi đối số size nhỏ hơn số ký tự

Đối số size trong hàm **fgets** giúp chúng ta chỉ định số ký tự lớn nhất có thể được đọc từ một dòng trong file. Thông thường chúng ta chỉ định giá trị của size bằng với số phần tử của mảng chuẩn bị để lưu nội dung đọc từ dòng đó.

Lưu ý trong trường hợp giá trị của size nhỏ hơn số ký tự thực có trong dòng, thì chỉ có **size -1** ký tự thực được đọc ra từ dòng đó mà thôi. Chỗ trống của ký tự còn lại sẽ được tự động lấp chỗ bằng một ký tự kết thúc chuỗi **\0**.

Ví dụ, chúng ta sẽ đọc lại file test.txt ở trên, nhưng với chỉ định số phần tử lớn nhất có thể đọc lại nhỏ hơn số ký tự thực trong dòng như sau:

|  |
| --- |
| #include <stdio.h>  int main(void)  {  *FILE* \* fp = NULL;      char arr[128];      //Mở file bằn hàm fopen      fp= fopen("C:\\Users\\LENOVO\\Desktop\\test.txt", "r");      //Đọc dòng 1      fgets(arr, 3, fp);      printf("%s\n", arr);      //Đọc dòng 2      fgets(arr, 4, fp);      printf("%s", arr);    } |

**Kết quả**

|  |
| --- |
| **12**  **345** |

##### Đọc từng dòng file theo định dạng chỉ định bằng hàm fscanf

###### Sự lợi hại của hàm fscanf trong C

Ở phần trên chúng ta đã biết cách đọc từng dòng trong file bằng hàm fgets rồi.

Tuy nhiên còn có một hàm lợi hại hơn nữa giúp chúng ta đọc từng dòng trong file với nội dung chọn lọc, ví dụ như khi chúng ta chỉ muốn **đọc chuỗi từ file trong c**, hoặc là khi chỉ muốn **đọc số từ file trong c.**

Đó chính là hàm fscanf giúp chúng ta đọc từng dòng file theo định dạng chỉ định trong C.

Hàm này đặc biệt hữu dụng khi cần đọc các file được viết theo định dạng cố định. Ví dụ như khi chúng ta cần **đọc file csv trong c** với các ô trong file được phân cách bởi dấu phẩy, hoặc là bằng dấu cách chẳng hạn.

###### Hàm fscanf trong C là gì

Hàm fscanf trong C là một hàm có sẵn trong thư viện chuẩn, có tác dụng **đọc từng dòng file theo định dạng chỉ định**. Hàm fscanf không đọc toàn bộ nội dung của dòng, mà sẽ phân tách nội dung dòng đó theo từng định dạng chỉ định, qua đó có thể truy xuất các thông tin cần thiết mà người dùng muốn lấy ra từ dòng đó, ví dụ như là chỉ lấy các số trong dòng, hoặc là chỉ lấy các chuỗi trong dòng được viết theo một quy luật (định dạng) cụ thể.

Các dữ liệu được truy xuất từ dòng theo định dạng chỉ định cũng sẽ được lưu trữ với định dạng tương ứng trong chương trình, giúp chúng ta đọc các dữ liệu từ file một cách có chọn lọc và nâng cao hiệu suất sử dụng chương trình.

Ví dụ cụ thể, chúng ta có thể chỉ **đọc số từ file trong c** hoặc là chỉ **đọc chuỗi từ file trong c** và bỏ qua các loại dữ liêu khác trong dòng.

Chúng ta sử dụng hàm fscanf trong C với cú pháp sau đây:

|  |
| --- |
| int fscanf(*FILE* \* *fp*, "fo1 fo2 fo3 ... ", add1, add2, add3 ...); |

Trong đó

* **fp** là con trỏ của file cần đọc, được tạo ra từ việc mở file ở phần trên.
* Các cặp**fo** và**add** tương ứng là định dạng (format) của dữ liệu cần đọc từ dòng, và địa chỉ của tên biến (con trỏ biến) dùng để lưu dữ liệu đó trong bộ nhớ.

Hàm **fscanf** sẽ trả về một số thuộc kiểu **int**, chính là số mục có thể đọc được từ dòng. Trong trường hợp dòng được đọc đã là dòng cuối file, hoặc là việc đọc file thất bại thì giá trị EOF sẽ được trả về.

**Lưu ý** là giống với **hàm fgets** thì **hàm fscanf** cũng chỉ có thể đọc lần lượt từng dòng trong file, do đó chúng ta cần phải sử dụng tới một vòng lặp để có thể đọc toàn bộ các dòng trong file.

Lại nữa, định dạng (format) của dữ liệu cần đọc cũng như kiểu của biến lưu dữ liệu được tóm tắt trong bảng dưới đây:

|  |  |  |
| --- | --- | --- |
| **Định dạng chuyển đổi** | **Kiểu biến** | **Chi tiết** |
| %hhd | char unsigned char | Chuyển về dạng thập phân và lưu trữ trong biến 1 byte |
| %hd | short unsigned short | Chuyển về dạng thập phân và lưu trữ trong biến 2 byte |
| %d | int unsigned int | Chuyển về dạng thập phân và lưu trữ trong biến kiểu int |
| %ld | long unsigned long | Chuyển về dạng thập phân và lưu trữ trong biến 4 byte |
| %hhx | char unsigned char | Chuyển về hệ thập lục phân và lưu trữ trong biến 1 byte |
| %hx | short unsigned short | Chuyển về hệ thập lục phân và lưu trữ trong biến 2 byte |
| %x | int unsigned int | Chuyển về hệ thập lục phân và lưu trữ trong biến kiểu int |
| %lx | long unsigned long | Chuyển về hệ thập lục phân và lưu trữ trong biến 4 byte |
| %f | float | Chuyển về số thực dấu phẩy động và lưu trữ trong biến kiểu float |
| %lf | double | Chuyển về số thực dấu phẩy động và lưu trữ trong biến kiểu double |
| %c | char | Chuyển về 1 ký tự và lưu trữ trong biến kiểu char |
| %s | char \* | Chuyển về chuỗi ký tự và lưu trữ trong biến kiểu mảng char |
| %p | void \* | Chuyển về địa chỉ và lưu trữ trong con trỏ |

###### Đọc số từ file trong C

Bằng cách chỉ định định dạng dữ liệu cần đọc dưới dạng số trong hàm fscanf, chúng ta có thể tiến hành **đọc số từ file trong c**.

Ví dụ, chúng ta có file **test.txt** với nội dung sau đây:

**NgoXuanThuc 20 170.0 A**

**NguyenPhiHao 38 165.5 O**

Giả sử chúng ta chỉ muốn đọc dữ liệu số bao gồm cột tuổi và chiều cao trong từng dòng file trên, khi đó chúng ta dùng hàm fscanf như sau:

|  |
| --- |
| #include <stdio.h>  int main(void){  *FILE* \* fp = NULL;      char    name[32] = { 0 };    // Tên      int        age = 0;            // Tuổi      double    height = 0;            // Chiều cao      char    blood = 0;            // Nhóm máu      //Mở file bằng hàm fopen\_s      fopen\_s(&fp, "C:\\Users\\LENOVO\\Desktop\\test.txt","r");      //Đọc từng dòng trong file cho tới khi gặp EOF      while (fscanf(fp, "%s %d %lfcm %c", name, &age, &height, &blood) != EOF)      {          //Xuất các dữ liệu số cần đọc          printf("%d %3.2lf \n", age, height);      }      return 0;  } |

**Kết quả:**

|  |
| --- |
| **20 170.00**  **38 165.50** |

**Lưu ý** là khi đọc từng dòng trong file bằng hàm fscanf thì chúng ta cần đọc tất cả các mục có thể đọc ra, tuy nhiên khi cần xuất dữ liệu số thì chúng ta chỉ cần chỉ định các dữ liệu đó mà thôi.

###### Đọc chuỗi từ file trong C

Tương tự như trên thì bằng cách chỉ định định dạng dữ liệu cần đọc dưới dạng số trong hàm fscanf, chúng ta có thể tiến hành **đọc chuỗi từ file trong c**.

Giả sử chúng ta chỉ muốn đọc dữ liệu chuỗi bao gồm cột tên và nhóm máu trong từng dòng file trên, khi đó chúng ta dùng **hàm fscanf** như sau:

|  |
| --- |
| #include <stdio.h>  int main(void){  *FILE* \* fp = NULL;      char    name[32] = { 0 };    // Tên      int        age = 0;            // Tuổi      double    height = 0;            // Chiều cao      char    blood = 0;            // Nhóm máu      //Mở file bằng hàm fopen\_s      fopen\_s(&fp, "C:\\Users\\LENOVO\\Desktop\\test.txt","r");      //Đọc từng dòng trong file cho tới khi gặp EOF      while (fscanf(fp, "%s %d %lfcm %c", name, &age, &height, &blood) != EOF)      {          //Xuất các dữ liệu số cần đọc          printf("%s %c \n", name, blood);      }      return 0;  } |

**Kết quả:**

|  |
| --- |
| **NgoXuanThuc A**  **NguyenPhiHao O** |

**Lưu ý** là khi đọc từng dòng trong file bằng hàm fscanf thì chúng ta cần đọc tất cả các mục có thể đọc ra, tuy nhiên khi cần xuất dữ liệu chuỗi thì chúng ta chỉ cần chỉ định các dữ liệu đó mà thôi.

##### Đọc từng dòng file theo định dạng chỉ định bằng hàm sscanf

Hàm sscanf trong C là một hàm có sẵn trong thư viện chuẩn, có tác dụng **truy xuất thông tin theo định dạng chỉ định từ một chuỗi ký tự**. Bằng cách ứng dụng hàm sscanf, chúng ta có thể lấy ra các thông tin cần thiết từ chuỗi theo một định dạng chỉ định nào đó.

Cú pháp sử dụng hàm sscanf trong C như sau:

|  |
| --- |
| int sscanf (*buff*, "fo1 fo2 fo3 ...", *add1*, *add2*, *add3* ...); |

Trong đó

* buff là con trỏ tới chuỗi ký tự cần phân tích để lấy ra thông tin theo định dạng
* Các cặp fo và add tương ứng là định dạng (format) của dữ liệu cần đọc từ chuỗi ký tự, và địa chỉ của tên biến dùng để lưu thông tin được tách ra từ chuỗi trên bộ nhớ.

Và định dạng (format) sử dụng trong hàm sscanf thì cũng tương tự như với hàm fscanf

Ứng dụng hàm sscanf, chúng ta có thể đọc từng dòng trong file và lấy ra các thông tin theo một chỉ định cụ thể, bằng cách chỉ định chuỗi ký tự cần phân tích trong hàm sscanf chính là nội dung từng dòng trong file được đọc bằng hàm gets.

Nói một cách dễ hiểu thì cách sử dụng của nó rất giống với hàm fscanf ở trên, ngoại trừ việc nó sẽ xử lý nội dung dòng được đọc bởi hàm fgets mà thôi.

Ví dụ cụ thể, chúng ta có file test.txt với nội dung sau đây:

**test01 1.0 1.1 1.2 1.3 1.4**

**test02 2.0 2.1 2.2 2.3 2.4**

**test03 3.0 3.1 3.2 3.3 3.4**

Bằng việc sử dụng hàm fgets để đọc từng dòng trong file này, sau đó dùng hàm sscanf để lấy thông tin cần thiết theo định dạng từ dòng, mà chúng ta có thể đọc từng dòng trong file với định dạng chỉ định như sau:

|  |
| --- |
| #include <stdio.h>  #define N 256 //Chỉ định số ký tự lớn nhất có thể đọc từ một dòng  int main(void) {  *FILE* \*fp;      char fname[] = "C:\\Users\\LENOVO\\Desktop\\test.txt";      char line[N];      char str[16];      float f1, f2, f3, f4, f5;      //Mở file bằng hàm fopen, và trả về NULL nếu mở file thất bại.      fp = fopen(fname, "r");      if(fp == NULL) {          printf("%s file not open!\n", fname);          return -1;      }      //Đọc từng dòng trong file bằng hàm fgets      while(fgets(line, N, fp) != NULL) {          //Truy xuất thông tin cần thiết từ nội dung đọc được bằng hàm sscanf          sscanf(line, "%s %f %f %f %f %f", str, &f1, &f2, &f3, &f4, &f5);          printf("%s %.1f %.1f %.1f %.1f %.1f\n", str, f1, f2, f3, f4, f5);      }      fclose(fp); //Đóng file      return 0;  } |

**Kết quả:**

|  |
| --- |
| **test01 1.0 1.1 1.2 1.3 1.4**  **test02 2.0 2.1 2.2 2.3 2.4**  **test03 3.0 3.1 3.2 3.3 3.4** |

#### Ghi file trong C

Cùng tìm hiểu về cách ghi file trong C. Bạn sẽ học được cách ghi file trong C bằng 3 hàm có sẵn là fputc, fputs, và fprintf sau bài học này.  
Chúng ta có 3 phương pháp để ghi file trong C như sau:

* Ghi từng ký tự vào file bằng hàm fputc
* Ghi từng dòng file trong C bằng hàm fputs
* Ghi từng dòng file theo định dạng chỉ định bằng hàm fprintf

##### Trước khi ghi file trong C

Để ghi file trong C, bạn cần phải mở nó trước bằng một trong hai hàm ghi file là fopen hoặc fopen\_s

Lưu ý là tùy thuộc vào mục đích của việc ghi file mà mode dùng để mở file cũng sẽ khác nhau, do vậy chúng ta cần hết sức chú ý khi lựa chọn mode khi mở file.

Ví dụ, nếu bạn chỉ muốn mở để ghi đè file, hãy dùng tới mode w như sau:

|  |
| --- |
| *FILE* \* fp = NULL;  fp = fopen("sample.txt", "w"); |

Hoặc nếu bạn muốn mở để ghi chèn file, hãy dùng tới mode a như sau:

|  |
| --- |
| *FILE* \* fp = NULL;  fp = fopen("sample.txt", "a"); |

Tuy nhiên nếu bạn muốn mở file và để vừa ghi và vừa ghi đè file đó, lúc này mode cần dùng để mở file không phải là mode w, mà sẽ là mode r+ chẳng hạn.

|  |
| --- |
| *FILE* \* fp = NULL;  fp = fopen("sample.txt", "r+"); |

Sau khi đã mở file thành công bằng một trong hai hàm trên, lúc này chúng ta đã có thể tiến hành **ghi dữ liệu vào file trong c** với các phương pháp sau đây.

##### Ghi kí tự vào file bằng hàm fputc

Hàm fputc trong C là một hàm có sẵn trong thư viện chuẩn, có tác dụng ghi từng ký tự vào file chỉ định. Tên hàm fputc được viết tắt bởi cụm từ file, put và character, được dịch theo tiếng Việt chính xác là **hàm ghi từng ký tự vào file.**

Chúng ta sử dụng hàm fputc trong C với cú pháp sau đây:

|  |
| --- |
| int fputc(int *char*, *FILE* \* *fp*); |

Trong đó:

* **fp** là con trỏ của file cần ghi, được tạo ra từ việc mở file ở phần trên.
* **char** là ký tự cần ghi vào file

Hàm fputc sẽ trả về ký tự đã được ghi vào file. Trong trường việc ghi file thất bại thì giá trị EOF sẽ được trả về.

Ví dụ cụ thể, chúng ta sẽ ghi dòng text Hello vào file trống sample.txt. Chúng ta sẽ dùng hàm fopen để mở file, sau đó ghi từng ký tự có trong dòng nội dung này vào file bằng hàm fputc như sau:

|  |
| --- |
| #include <stdio.h>  int main(void)  {  *FILE* \* fp = NULL;      fopen\_s(&fp, "C:\\Users\\LENOVO\\Desktop\\sample.txt", "w");      fputc('H', fp);      fputc('e', fp);      fputc('l', fp);      fputc('l', fp);      fputc('o', fp);      fputc('\n', fp);      fclose(fp);      return 0;  } |

Kết quả, các ký tự trong chuỗi sẽ được lần lượt ghi vào file sample.txt như sau:

|  |
| --- |
| **Hello** |

##### Ghi chuỗi vào file trong C bằng hàm fputs

###### Hàm fputs trong C

Việc ghi từng ký tự vào file bằng hàm fputc thật là vất vả phải không nào? Đó là lý do mà hàm fputs với chức năng ghi chuỗi vào file đã được ra đời.

Hàm fputs trong C là một hàm có sẵn trong thư viện chuẩn, có tác dụng ghi từng chuỗi vào file chỉ định. Tên hàm fputs được viết tắt bởi cụm từ file, put và string, được dịch theo tiếng Việt chính xác là hàm ghi chuỗi vào file.

Chúng ta sử dụng hàm fputs trong C với cú pháp sau đây:

|  |
| --- |
| int fputs(const char \* *str*, *FILE* \* *fp*); |

Trong đó:

* fp là con trỏ của file cần ghi, được tạo ra từ việc mở file
* str là con trỏ tới nơi lưu trữ chuỗi đã ghi từ dòng vào file. Thông thường chúng ta chỉ định str bằng một chuỗi.

Hàm fputs sẽ trả về 0 nếu việc ghi chuỗi vào file thành công. Trong trường hợp việc ghi file thất bại thì EOF sẽ được trả về.

**Lưu ý** là hàm fputs không tự động ghi ký tự xuống dòng \n sau khi đã ghi chuỗi, do đó khi muốn xuống dòng khi ghi chuỗi thì chúng ta phải tự điền ký tự \n vào vị trí cần xuống dòng trong chuỗi.

###### Ví dụ dùng hàm fputs để ghi chuỗi vào file trong C

Ví dụ cụ thể chúng ta sẽ ghi nội dung sau vào file trống Hello.txt.

**Hello World !**

**Toi la Ngo Xuan Thuc**

Chúng ta sẽ dùng hàm fopen để mở file, sau đó ghi các chuỗi này vào file bằng hàm fputs như sau:

|  |
| --- |
| #include <stdio.h>  int main(void)  {  *FILE* \* fp = NULL;      fopen\_s(&fp, "C:\\Users\\LENOVO\\Desktop\\Hello.txt", "w");      fputs("Hello World !\nToi la Ngo Xuan Thuc", fp);      fclose(fp);      return 0;  } |

Kết quả, chuỗi chỉ định sẽ được lần lượt ghi vào file Hello.txt như sau:

|  |
| --- |
| **Hello World !**  **Toi la Ngo Xuan Thuc** |

##### Ghi dữ liệu vào file trong C theo định dạng bằng hàm fprintf

###### Hàm fprintf trong C là gì

**Hàm fprintf** trong C là một hàm có sẵn trong thư viện chuẩn, có tác dụng ghi dữ liệu vào file theo định dạng chỉ định.

Bằng cách sử dụng hàm fprintf, chúng ta có thể định dạng các dữ liệu thuộc nhiều kiểu khác nhau, và sau đó ghi chúng vào các file có định dạng cố định như file CSV, JSON chẳng hạn.

Chúng ta sử dụng hàm fprintf trong C với cú pháp sau đây:

|  |
| --- |
| int fprintf(*fp*, “fo1 fo2 fo3 “, var1, var2, var3 ); |

Trong đó

* fp là con trỏ của file cần ghi, được tạo ra từ việc mở file ở phần trên.
* Các cặp fo và var tương ứng là định dạng (format) ghi vào file của dữ liệu, và tên biến chứa dữ liệu đó.

Cơ chế hoạt động của hàm fprintf là lấy dữ liệu từ biến var, định dạng nó theo định dạng ghi trong fo để tạo ra một chuỗi. Sau đó thì ghi chuỗi vừa tạo này vào file mà con trỏ fp chỉ đến.

Hàm fprintf sẽ trả về một số thuộc kiểu int, chính là số ký tự có trong chuỗi được tạo. Trong trường hợp việc ghi file thất bại thì giá trị EOF sẽ được trả về.

###### Định dạng dữ liệu sử dụng trong hàm fprintf

|  |  |  |
| --- | --- | --- |
| **Định dạng chuyển đổi** | **Kiểu biến** | **Chi tiết** |
| %hhd | char unsigned char | Chuyển về dạng thập phân và lưu trữ trong biến 1 byte |
| %hd | short unsigned short | Chuyển về dạng thập phân và lưu trữ trong biến 2 byte |
| %d | int unsigned int | Chuyển về dạng thập phân và lưu trữ trong biến kiểu int |
| %ld | long unsigned long | Chuyển về dạng thập phân và lưu trữ trong biến 4 byte |
| %hhx | char unsigned char | Chuyển về hệ thập lục phân và lưu trữ trong biến 1 byte |
| %hx | short unsigned short | Chuyển về hệ thập lục phân và lưu trữ trong biến 2 byte |
| %x | int unsigned int | Chuyển về hệ thập lục phân và lưu trữ trong biến kiểu int |
| %lx | long unsigned long | Chuyển về hệ thập lục phân và lưu trữ trong biến 4 byte |
| %f | float | Chuyển về số thực dấu phẩy động và lưu trữ trong biến kiểu float |
| %lf | double | Chuyển về số thực dấu phẩy động và lưu trữ trong biến kiểu double |
| %c | char | Chuyển về 1 ký tự và lưu trữ trong biến kiểu char |
| %s | char \* | Chuyển về chuỗi ký tự và lưu trữ trong biến kiểu mảng char |
| %p | void \* | Chuyển về địa chỉ và lưu trữ trong con trỏ |

###### Ví dụ ghi dữ liệu vào file trong C

Giả sử chúng ta có các kiểu thông tin như [chuỗi], [ký tự] và [số] được lưu vào các biến tương ứng. Bằng cách sử dụng hàm fprintf, chúng ta có thể định dạng các thông tin này và ghi vào file như chương trình sau đây:

|  |
| --- |
| #include <stdio.h>  int main(void)  {  *FILE* \* fp = NULL;      char ten[] = "NgoXuanThuc";      char gioitinh = 'M';      int  tuoi = 20;      fopen\_s(&fp, "C:\\Users\\LENOVO\\Desktop\\Hello.txt", "w");      // Ghi dữ liệu theo định dạng chỉ định vào file      fprintf(fp, "%s %c %d\n", ten, gioitinh, tuoi);      fprintf(fp, "%s,%c,%d\n", ten, gioitinh, tuoi);      fprintf(fp, "Ten: %s, GioiTinh: %c, Tuoi: %d\n", ten, gioitinh, tuoi);      fclose(fp);      return 0;  } |

Kết quả, các dữ liệu với định dạng chỉ định sẽ được vào file Hello.txt như sau:

|  |
| --- |
| **NgoXuanThuc M 20**  **NgoXuanThuc,M,20**  **Ten: NgoXuanThuc, GioiTinh: M, Tuoi: 20** |

Giống như thế, các kiểu thông tin như [chuỗi], [ký tự] và [số] sẽ được ghi theo định dạng chỉ định vào file.

##### Ghi mảng vào file trong C

Ứng dụng hàm fprintf, chúng ta có thể thực hiện ghi mảng vào file trong c.

Do các phần tử trong mảng C đều có cùng kiểu dữ liệu, nên chúng ta có thể chỉ định cùng kiểu định dạng để ghi các phần tử từ mảng vào file bằng hàm hàm fprintf. Ví dụ cụ thể:

|  |
| --- |
| #include <stdio.h>  int main(void)  {  *FILE* \* fp = NULL;      int num[] = {10, 20, 44, 60, 82};      fopen\_s(&fp, "C:\\Users\\LENOVO\\Desktop\\Hello.txt", "w");      // Ghi dữ liệu theo định dạng chỉ định vào file      fprintf(fp, "%d %d %d %d %d", num[0], num[1], num[2], num[3], num[4]);      fclose(fp);      return 0;  } |

**Kết quả ghi vào file như sau**

|  |
| --- |
| **10 20 44 60 82** |

Một cách tương tự thì chúng ta cũng có thể ghi các mảng 2 chiều vào file trong C. Khi đó mỗi mảng con 1 chiều trong mảng 2 chiều sẽ được coi như một dòng để ghi vào file, và chúng ta cần phải thêm ký tự xuống dòng **\n** khi ghi từng mảng dòng vào file như sau:

|  |
| --- |
| //"C:\\Users\\LENOVO\\Desktop\\Hello.txt"  #include <stdio.h>  int main(void)  {  *FILE* \* fp = NULL;      int bangdiem[][3] = {      {7, 9, 8} ,      {8, 6, 7} ,      {5, 7, 6} ,      {4, 9, 5} ,      {5, 8, 7} ,      {6, 9, 3}  } ;      //Lấy độ dài của mảng 2 chiều      int y = sizeof(bangdiem) / sizeof(bangdiem[0]);      int x = sizeof(bangdiem[0]) / sizeof(bangdiem[0][0]);      fopen\_s(&fp, "C:\\Users\\LENOVO\\Desktop\\bangdiem.txt", "w");      // Ghi từng mảng 1 chiều trong mảng 2 chiều vào file      for (int i = 0; i < y; i++){          fprintf(fp, "Toan: %d Van: %d Anh: %d\n", bangdiem[i][0], bangdiem[i][1], bangdiem[i][2]);      }      fclose(fp);      return 0;  } |

Kết quả, từng dòng sẽ được ghi vào file như sau:

|  |
| --- |
| **Toan: 7 Van: 9 Anh: 8**  **Toan: 8 Van: 6 Anh: 7**  **Toan: 5 Van: 7 Anh: 6**  **Toan: 4 Van: 9 Anh: 5**  **Toan: 5 Van: 8 Anh: 7**  **Toan: 6 Van: 9 Anh: 3** |

##### Ghi cấu trúc vào trong file

Ứng dụng hàm fprintf, chúng ta có thể thực hiện **ghi cấu trúc vào file trong c**.

Do các thành viên trong cấu trúc C có thể có kiểu dữ liệu khác nhau, nên chúng ta cần chỉ định kiểu định dạng phù hợp cho từng thành viên để ghi giá trị của chúng từ mảng vào file bằng hàm hàm fprintf.

Ví dụ chúng ta ghi dữ liệu từ mảng cấu trúc vào file trong C bằng hàm fprintf như sau:

|  |
| --- |
| #include <stdio.h>  #define PERSON\_NUM 5  //Khai báo kiểu struct  typedef struct {      char name[20];      char sex;      int age;      double height;      double weight;  } *person\_t*;  int main(void)  {      //Khởi tạo thực thể struct là một mảng struct  *person\_t* p[PERSON\_NUM] = {{"Bob",      'M', 19, 165.4, 72.5},                                {"Alice",    'F', 19, 161.7, 44.2},                                {"Tom",      'M', 20, 175.2, 66.3},                                {"Stefany",  'F', 18, 159.3, 48.5},                                {"Leonardo", 'M', 19, 172.8, 67.2}};      //Tạo con trỏ file  *FILE* \* fp = NULL;      //Mở file      fopen\_s(&fp, "C:\\Users\\LENOVO\\Desktop\\people.txt", "w");      //Ghi header vào file      fprintf(fp, "%s %s %s %s %s\n", "name", "sex", "age", "height", "weight");      // Ghi từng thực thể của cấu trúc như một dòng vào file      for (short i = 0; i < PERSON\_NUM; i++) {              fprintf(fp, "%s %c %d %.2f %.2f \n",p[i].name, p[i].sex,p[i].age,p[i].height,p[i].weight);      }      fclose(fp);      return 0;  } |

**Kết quả được ghi vào file**

|  |
| --- |
| **name sex age height weight**  **Bob M 19 165.40 72.50**  **Alice F 19 161.70 44.20**  **Tom M 20 175.20 66.30**  **Stefany F 18 159.30 48.50**  **Leonardo M 19 172.80 67.20** |

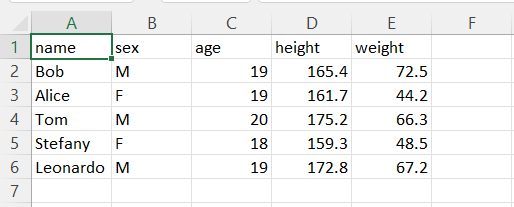
##### Ghi file CSV trong C

Ở phần trên, chúng ta đã tiến hành ghi nội dung các kiểu dữ liệu phức tạp như mảng hay cấu trúc vào file dưới dạng file txt rồi. Tuy nhiên chúng ta cũng có thể ghi dữ liệu trên vào file CSV trong C, đơn giản bằng cách thay đổi đuôi file từ *.txt* sang *.csv* là xong.

Ví dụ cụ thể, chúng ta ghi dữ liệu từ mảng cấu trúc vào file CSV trong C bằng hàm fprintf như sau. Lưu ý là chúng ta cần viết định dạng các dữ liệu cách nhau bởi dấu phẩy để có thể ghi từng dữ liệu vào từng ô trong file CSV.

|  |
| --- |
| #include <stdio.h>  #define PERSON\_NUM 5  typedef struct {      char name[20];      char sex;      int age;      double height;      double weight;  } *person\_t*;  int main(void)  {  *person\_t* p[PERSON\_NUM] = {{"Bob",      'M', 19, 165.4, 72.5},                                {"Alice",    'F', 19, 161.7, 44.2},                                {"Tom",      'M', 20, 175.2, 66.3},                                {"Stefany",  'F', 18, 159.3, 48.5},                                {"Leonardo", 'M', 19, 172.8, 67.2}};  *FILE* \* fp = NULL;      int num[] = {10, 20, 44, 60, 82};      fopen\_s(&fp, "C:\\Users\\LENOVO\\Desktop\\test.csv", "w");      // Ghi từng mảng 1 chiều trong mảng 2 chiều vào file      fprintf(fp, "%s,%s,%s,%s,%s\n", "name", "sex", "age", "height", "weight");      for (short i = 0; i < PERSON\_NUM; i++) {              fprintf(fp, "%s,%c,%d,%.2f,%.2f\n",p[i].name, p[i].sex,p[i].age,p[i].height,p[i].weight);      }        fclose(fp);      return 0;  } |

Kết quả, nội dung mảng cấu trúc sẽ được ghi vào file test.csv như sau:



#### Đóng file trong C

Sau khi đã xử lý xong file trong chương trình, chúng ta cần phải đóng file đó lại. Việc đóng file sẽ giúp kết thúc phiên làm việc với file, và giải phóng bộ nhớ.

Nếu không đóng file thì file đó vẫn tồn tại trên bộ nhớ, dẫn đến xảy ra các sự cố về bộ nhớ trong chương trình.

Để đóng một file trong C, chúng ta cần dùng đến hàm **fclose()** với cú pháp sau đây:

|  |
| --- |
| fclose(*fp*); |

Trong đó **fp** là con trỏ dùng để mở file.

Ví dụ cụ thể, sau khi mở file **sample.txt** ở trên và hoàn thành các xử lý với file này, chúng ta có thể đóng file và giải phóng bộ nhớ như sau:

|  |
| --- |
| fp = fopen("sample.txt", "r");  fprintf(*fp*, "%s", "Hello Vietnam!");  fclose(*fp*) |