Bài 7: Những thao tác làm việc với tập tin

Những lỗi của các biến chỉ tồn tại trong RAM. Một khi bạn đã hoàn thành chương trình của mình, tất cả các biến của bạn sẽ bị xóa khỏi bộ nhớ và giá trị của chúng sẽ không thể khôi phục lại.

Vậy thì làm cách nào để chúng ta có thể lưu lại điểm số cao nhất trong game?

Làm sao để một chương trình soạn thảo văn bản hoạt động nếu tất cả những gì bạn viết ra sẽ bị xóa khi tắt chương trình?

May mắn thay, bạn có thể đọc và ghi dữ liệu của các tập tin trong ngôn ngữ C. Những tập tin này được lưu vào ổ cứng (hard drive) của máy tính: lợi ích là những tập tin sẽ được lưu lại tại đó, ngay cả khi bạn dừng chương trình hay tắt máy tính.

Để có thể đọc và ghi dữ liệu vào các tập tin chúng ta sẽ ứng dụng tất cả những gì đã học trước đây: con trỏ, cấu trúc, chuỗi ký tự …

Mở và đóng một tập tin:

Để đọc và ghi dữ liệu trong một tập tin chúng ta sẽ sử dụng các hàm (functions) có sẵn trong thư viện "stdio" mà chúng ta được học trước giờ.

Vâng và cũng chính nó là nơi chứa 2 function quen thuộc "printf" và "scanf". Nhưng nó không chỉ chứa duy nhất 2 function này mà còn có cả những function được tạo ra để làm việc với các tập tin.

 Tất cả những thư viện (libraries) mà chúng ta đã từng sử dụng cho tới nay (stdlib.h, stdio.h, math.h, string.h …) được gọi là các thư viện chuẩn. Chúng sẽ được IDE tự động nhận diện dù cho bạn có chạy chương trình trên bất kỳ hệ điều hành nào Windows, Linux, Mac hay một hệ điều hành nào đó.

Những thư viện chuẩn có số lượng giới hạn và hỗ trợ cho phép bạn thực hiện một số điều cơ bản như chúng ta đã từng thấy trong những bài trước. Đối với những chức năng cao cấp hơn, như là mở một cửa sổ chương trình, ta phải tải về và cài đặt một thư viện mới. Chúng ta sẽ học về chúng sau !!!

Để chắc cú thì bạn nên luôn luôn bắt đầu những dòng code của mình với việc khai báo các thư viện chuẩn như *stdio.h* và *stdlib.h* trên đầu tập tin "file.c" của bạn:

C code:

```
#include <stdlib.h>
#include <stdio.h>
```

Những thư viện này rất cơ bản, thực sự rất cơ bản, nhưng tôi vẫn khuyên bạn hãy luôn thêm nó vào tất cả những chương trình của bạn sau này, dù cho chúng có tác dụng gì đi nữa.

Okie! Bây giờ chúng ta sẽ xem những thư viện tuyệt vời đó có thể làm gì, chúng ta có thể giải quyết câu hỏi làm sao để mở hoặc đọc hoặc ghi dữ liệu vào một tập tin. Dưới đây là những gì luôn diễn ra khi bạn yêu cầu máy tính thực hiện các thao tác đó:

1. Chúng ta sẽ gọi một hàm mở tập tin , thứ sẽ trả về cho chúng ta một con trỏ đến tập tin.
2. Chúng ta sẽ kiểm tra xem thao tác mở tập tin có thành công không (tất nhiên là trước tiên tập tin đó phải tồn tại) bằng cách kiểm tra các giá trị con trỏ mà ta nhận được. Nếu con trỏ mang giá trị NULL, có nghĩa là thao tác mở tập tin đã thất bại, trong trường hợp này chúng ta không thể tiếp túc nữa (màn hình sẽ hiển thị thông báo lỗi).
3. Trong trường hợp thao tác mở tập tin thành công (con trỏ không mang giá trị NULL), chúng ta có thể thoải mái đọc, ghi dữ liệu lên tập tin thông qua chức năng của các hàm mà các bạn sắp được học.
4. Một khi bạn đã làm xong việc với các tập tin, hẳn là bạn phải nghĩ tới việc đóng nó lại đúng không? Và chúng ta có *fclose*.

Đầu tiên chúng ta sẽ tìm hiểu về 2 hàm *fopen* và *fclose*. Sau khi đã nắm rõ vấn đề, chúng ta sẽ bắt đầu học cách đọc nội dung của một tập tin và ghi dữ liệu vào nó.

fopen: hàm mở một tập tin:

Trong bài học về chuỗi ký tự (string), chúng ta đã biết cách sử dụng "nguyên mẫu hàm" (prototype) như một "hướng dẫn sử dụng" của hàm (function). Đó cũng là thói quen phổ biến của các lập trình viên, họ đọc các prototypes để hiểu về những functions có trong đoạn code.

Tuy nhiên tôi thấy rằng chúng ta vẫn cần giải thích thêm một số thứ:

C code:

```
FILE* fopen(const char* tenTaptin, const char* chedoMotaptin);
```

Hàm này có 2 tham số:

- Tên tập tin được mở.
- Chế độ mở tập tin, nhìn vào tham số này bạn có thể thấy được nó diễn tả những gì bạn muốn làm với tập tin: chẳng hạn như đọc hoặc ghi dữ liệu hoặc cả 2 chức năng cùng lúc.

Hàm này sẽ trả về một con trỏ tới FILE. Đây là con trỏ đến cấu trúc FILE. Cấu trúc này đã được định nghĩa sẵn trong thư viện *stdio.h*, bạn có thể mở file thư viện ra để tự mình kiểm tra xem

FILE có cái quái gì trong đó nhưng tôi nghĩ dù có chứa gì đi nữa cũng không ảnh hưởng nhiều đến công việc của chúng ta.

 Tại sao tên của cấu trúc (struct) FILE lại được viết hoa hết vậy? Tôi nhớ những cái tên được viết hoa là dành riêng cho các định nghĩa (define) và hằng (constant) cơ mà ???

Đây là quy tắc do tôi tự đặt ra (và rất nhiều lập trình viên khác cũng làm theo quy tắc tương tự). Bạn không bắt buộc phải làm theo những quy tắc của tôi và người tạo ra thư viện *stdio* chắc cũng có những quy tắc của riêng mình. Và lý do duy nhất tôi có thể giải thích cho bạn tại sao FILE lại được viết hoa trong thư viện *stdio*, là vì người tạo ra nó muốn như vậy ^^!

Đừng để một cái tên làm bạn mất tập trung. Rồi bạn sẽ thấy những thư viện chúng ta sắp học sau này cũng sử dụng quy tắc đặt tên rất giống với tôi, cụ thể là khi đặt tên cấu trúc thường chỉ cần viết hoa chữ cái đầu tiên.

Quay lại với hàm *fopen* của chúng ta. Nó trả về FILE*. Con trỏ này thật sự rất quan trọng, nó sẽ giúp ta có thể đọc hoặc ghi dữ liệu vào một tập tin.

Bây giờ chúng ta sẽ tạo một con trỏ của FILE để bắt đầu hàm của bạn (VD function main):

C code:

```
int main (int argc, char *argv[ ])
{
   FILE* taptin = NULL;

   return 0;
}
```

Con trỏ được khởi tạo giá trị NULL ngay từ đầu. Tôi nhắc lại cho bạn nhớ đây là một nguyên tắc cơ bản, hãy luôn khởi tạo giá trị NULL cho con trỏ ngay từ đầu nếu bạn không có giá trị nào khác để gán cho nó. Nếu không thực hiện điều này, rồi bạn sẽ tha hồ đọc thông báo lỗi trong tương lai nhé.

 Bạn sẽ thấy rằng việc viết struct FILE *taptin = NULL là không cần thiết. Người tạo ra *stdio* cũng đã tạo ra một *typedef* như tôi đã dạy bạn để ta có thể viết là FILE *taptin = NULL.

Lưu ý rằng cấu tạo của cấu trúc có thể thay đổi tùy theo từng hệ điều hành khác nhau (không nhất thiết lúc nào cũng phải có các biến thành phần giống nhau). Vì vậy, chúng ta sẽ không bao giờ trực tiếp thay đổi nội dung của FILE (vd như ta sẽ không sử dụng taptin.bienthanhphan1 để tác động vào biến thành phần của FILE). Nhưng bằng cách sử dụng các hàm chúng ta có thể làm việc được với FILE.

Nào bây giờ chúng ta sẽ gọi hàm *fopen* và lấy giá trị nó đã trả về trong con trỏ *taptin*. Nhưng trước khi làm điều đó, tôi phải giải thích cho các bạn về tham số thứ hai, tham số *chedoMotaptin*. Thật ra có một đoạn code đã được gửi cho máy tính để nói cho nó biết rằng tập tin này sẽ được mở trong chế độ "chỉ được đọc dữ liệu" (read-only) hoặc "chỉ được ghi dữ liệu" (write-only) hoặc cả 2 chế độ cùng một lúc.

Sau đây là những chế độ có thể áp dụng cho tập tin của bạn trong máy tính:

- "r" – read only – chế độ chỉ đọc: Bạn có thể đọc được nội dung của tập tin nhưng không thể viết thêm gì vào *(Tất nhiên là tập tin đã được tạo ra trước đó).*

- "w" – write only – chế độ chỉ viết: Bạn có thể viết thêm hoặc chỉnh sửa nội dung tập tin nhưng lại không đọc được nó *(Nếu chưa tồn tại thì tập tin sẽ được tạo ra ở chế độ này).*

- "a" – user addition – chế độ bổ sung: Bạn có thể viết thêm vào nội dung của một tập tin bắt đầu từ vị trí kết thúc nội dung tập tin đó (*Nếu chưa tồn tại thì tập tin sẽ được tạo ra ở chế độ này).*

- "r+" – read and write – chế độ đọc và viết: Bạn có thể đọc và viết trong nội dung của tập tin *(Tất nhiên là tập tin đã được tạo ra trước đó).*

- "w+" – read and write, with removal of content beforehand – chế độ đọc và viết đồng thời xóa sạch nội dung tập tin trước đó: Trước hết tập tin sẽ không chứa bất kỳ nội dung nào trước đó. Bạn có thể viết trong nội dung của tập tin và đọc nó sau *(Nếu chưa tồn tại thì tập tin sẽ được tạo ra ở chế độ này).*

- "a+" – add read/write at the end – chế độ đọc và viết ở vị trí cuối cùng: Bạn có thể viết và đọc nội dung của tập tin bắt đầu từ vị trí kết thúc của nội dung trong tập tin *(Nếu chưa tồn tại thì tập tin sẽ được tạo ra ở chế độ này).*

Để tôi nói thêm cho bạn một thông tin, tôi chỉ mới cho bạn thấy một phần các chế độ của thao tác mở tập tin. Thật ra còn một điều nữa, với mỗi chế độ mà bạn đang thấy ở trên, nếu bạn thêm ký tự "b" sau ký tự đầu tiên của mỗi chế độ ("rb", "wb", "ab", "rb+", "wb+", "ab+"), lúc này các tập tin của bạn sẽ được mở trong chế độ nhị phân (binary). Đây là một cách khá đặc biệt mà chúng ta sẽ không đề cập ở đây. Thật ra chế độ hiển thị văn bản (text mode) là để lưu trữ dữ liệu dạng văn bản vd như "tên hiển thị" (chỉ hiển thị các ký tự), trong khi chế độ hiển thị số nhị phân (binary mode) là để lưu trữ dữ liệu dạng số vd như "dung lượng bộ nhớ" (hầu như chỉ toàn là số). Đây là sự khác nhau duy nhất giữa 2 chế độ hiển thị.

Dù là chế độ nào thì chúng vẫn hoạt động theo một cách tương tự nhau như chúng ta thấy trong bài học này.

Cá nhân tôi thường sử dụng chế độ “r” (chỉ đọc), “w” (chỉ viết), “r+” (đọc và viết). Chế độ “w+” có một chút nguy hiểm vì nó sẽ xóa sạch nội dung tập tin của bạn mà chẳng cho bạn một thông báo nào. Bạn chỉ nên sử dụng chế độ này khi muốn làm mới (reset) lại tập tin của mình.

Về chế độ “a” có lẽ nó sẽ hữu dụng trong một số trường hợp khi mà bạn chỉ muốn bổ sung một số thông tin vào cuối tập tin.

 Nếu bạn chỉ có ý định đọc một tập tin, tôi khuyến khích các bạn nên chọn chế độ “r”, dĩ nhiên là chế độ “r+” cũng sẽ giúp bạn đọc được tập tin nhưng trong chế độ “r”, tập tin của bạn sẽ đảm bảo tính bảo mật, bạn sẽ không sợ người nào đó thay đổi nội dung của tập tin, đây cũng là một cách bảo vệ tập tin của mình.

Nếu bạn muốn viết một hàm loadLevel (để tải các cấp độ trò chơi) thì nên chọn chế độ “r”, còn nếu bạn viết một hàm saveLevel (để lưu lại các cấp độ trò chơi) thì chỉ cần chọn chế độ “w” là đủ.

Đoạn code sau đây sẽ mở một tập tin *test.txt* ở chế độ “r+” (đọc / ghi):

C code:

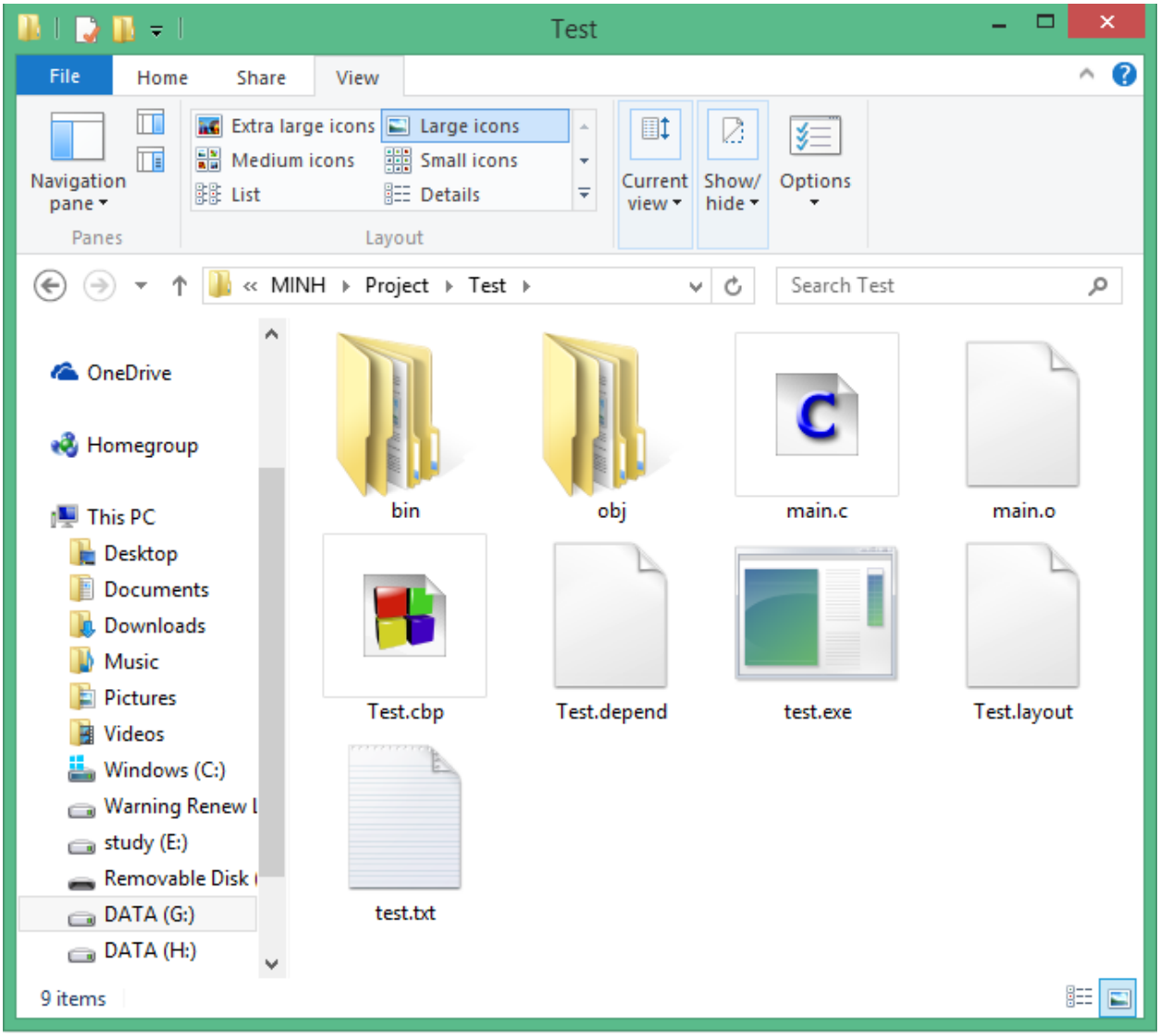
```
int main (int argc, char *argv[ ])
{
   FILE* taptin = NULL;

  taptin = fopen("test.txt", "r+");

   return 0;
}
```

Con trỏ *taptin* sau đó sẽ thành con trỏ hướng đến tập tin *test.txt*

 Nhưng tập tin *test.txt* được đặt ở đâu?

Nó nên được đặt trong cùng một thư mục với tập tin thực thi (executable) .exe của bạn.

Để tiện theo dõi bài học này, hãy tạo một tập tin *test.txt* trong thư mục như tôi đã làm trong hình dưới:



Như bạn thấy, tôi đang sử dụng IDE Code::Blocks, đây là câu trả lời cho câu hỏi tại sao lại tồn tại tập tin dự án (project) có đuôi .cbp (chứ không phải là tập tin dự án có đuôi chấm .sln như những người dùng Visual C++). Tạm thời, điều quan trọng cần để ý ở đây là tập tin *test.txt* được đặt trong cùng một thư mục với chương trình *test.exe*.

Vậy tập tin được tạo phải có đuôi *.txt* mới được sao?

Không. Đó là do bạn chọn phần định dạng của tập tin khi bạn mở tập tin. Bạn có thể sáng tạo ra kiểu định dạng của bạn VD như tạo một tập tin đuôi chấm *.level* để lưu lại cấp độ trò chơi của mình.

 Vậy tập tin này lúc nào cũng phải nằm cùng thư mục chứa chương trình thực thi .exe sao?

Cũng không luôn. Tập tin này có thể được đặt trong thư mục con:

C code:

```
taptin = fopen("folder/test.txt", "r+");
```

Bây giờ tập tin *test.txt* được đặt trong thư mục con tên là *folder*.Phương pháp này được gọi là "đường dẫn tương đối", và thường được sử dụng rộng rãi. Với cách này, chương trình của bạn sẽ ít bị gặp lỗi hơn.

Chúng ta cũng có thể mở một tập tin bất kỳ ở đâu đó trong ổ cứng máy tính. Trong trường hợp này bạn phải viết đường dẫn một cách chính xác và đầy đủ (ta có thể gọi là đường dẫn tuyệt đối):

C code:

```
taptin = fopen("C:\\Program Files\\Notepad++\\readme.txt", "r+");
```

Đoạn code này sẽ mở các tập tin *readme.txt* nằm trong C:\Program Files\Notepad++

 Tôi đã dùng 2 dấu \\ mỗi lần rẽ nhánh thư mục như bạn thấy. Nếu tôi chỉ dùng một dấu \ máy tính tính sẽ hiểu nhầm rằng bạn đang thêm vào một ký tự đặc biệt như \n hoặc \t. Để viết một dấu \ trong chuỗi ký tự, bạn phải viết nó 2 lần (viết là \\), lúc này máy tính sẽ hiểu rằng bạn muốn sử dụng một dấu \ này.

Có một nhược điểm là những đường dẫn tuyệt đối có thể chỉ hoạt động trên một hệ điều hành cụ thể. Đây không phải là một giải pháp linh động. Chẳng hạn như cũng là một đường dẫn nhưng trên Linux bạn phải viết như sau:

C code:

```
taptin = fopen("/home/Minh/folder/readme.txt", "r+");
```

Tôi khuyên bạn nên sử dụng đường dẫn tương đối thay vì đường dẫn tuyệt đối. Đừng sử dụng đường dẫn tuyệt đối nếu chương trình của bạn được viết riêng cho một hệ điều hành nào đó và các tập tin của bạn cũng nên được lưu ở một thư mục cụ thể trong ổ cứng của máy tính.

Kiểm tra thao tác mở tập tin:

Tập tin chứa địa chỉ con trỏ *taptin* của cấu trúc FILE. Nó đã được cấp phát bộ nhớ bởi *fopen ( )*.

Bây giờ có 2 trường hợp có thể xảy ra:

- Mở tập tin thành công và bạn hoàn toàn có thể tiếp tục thao tác với tập tin (có thể là đọc, hoặc viết thêm vào tập tin).
- Mở tập tin thất bại do tập tin đó chưa tồn tại hoặc tập tin đang được sử dụng bởi chương trình khác. Trong trường hợp này bạn không thể làm gì với tập tin đó hết.

Sau khi thực hiện thao tác mở tập tin chúng ta có thể kiểm tra xem có thành công không. Cách kiểm tra rất đơn giản thôi: nếu con trỏ mang giá trị NULL, mở tập tin thất bại. Nếu con trỏ mang một giá trị bất kì nào khác NULL, mở tập tin thành công.

Chúng ta sẽ làm như sau để kiểm tra:

C code:

```
int main (int argc, char *argv[ ])
{
   FILE* taptin = NULL;

   taptin = fopen("test.txt", "r+");

   if (taptin != NULL)
   {
      // khac gia tri NULL thi ban co the doc va ghi du lieu vao tap tin roi
   }
   else
   {
      // Ban co the cho no hien thi thong bao loi neu thich
      printf ("Khong the mo tap tin test.txt");
   }

   return 0;
}
```

Hãy làm như trên mỗi khi muốn mở tập tin. Nếu bạn không làm hoặc tập tin không tồn tại, chương trình sẽ gặp lỗi.

fclose: đóng một tập tin

Nếu tập tin được mở thành công thì bạn có thể đọc và ghi thêm dữ liệu vào nội dung của nó (chút nữa thôi chúng ta sẽ thấy cách làm).

Một khi bạn đã xong việc với tập tin thì bạn phải đóng nó lại đúng không? Chúng ta sẽ thực hiện thao tác này bằng *fclose*, việc này có vai trò giúp giải phóng bộ nhớ cho máy tính, điều đó cũng có nghĩa là những tập tin được nạp vào RAM sẽ được xóa sạch.

Prototype của *fclose* là:

C code:

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

Hàm này chỉ có một tham số: đó là con trỏ của tập tin.

Nó trả về một giá trị kiểu *int* cho biết đã đóng được tập tin chưa, giá trị đó là:

- Giá trị bằng 0: Nếu tập tin được đóng thành công.
- Giá trị là EOF (End Of File): Nếu việc đóng tập tin thất bại. EOF được định nghĩa sẵn trong *stdio.h* tương ứng với một số đặc biệt, giá trị này có nhiệm vụ thông báo cho máy tính về một lỗi đã xảy ra.

Để đóng một tập tin chúng ta sẽ làm như sau:

C code:

```
fclose(taptin);
```

Tóm lại chúng ta sẽ làm theo cách sau để mở và đóng một tập tin trong project như sau:

C code:

```
int main (int argc, char *argv[ ])
{
   FILE* taptin = NULL;

   taptin = fopen("test.txt", "r+");

   if (taptin != NULL)
   {
      // Ban co the doc va ghi du lieu vao noi dung tap tin

      // ...

      fclose(taptin); // Dong tap tin da duoc mo
   }

   return 0;
}
```

Tôi đã không sử dụng *else* để hiển thị thông báo lỗi nếu thao tác mở tập tin thất bại. Nhưng nếu muốn, tôi tin là bạn biết cách làm mà đúng không.

Hãy luôn nhớ đóng tập tin lại mỗi khi hoàn thành công việc, điều này giúp giải phóng bộ nhớ của máy tính.

Nếu bạn không giải phóng bộ nhớ cho máy tính, sau khi hoàn thành, chương trình của bạn sẽ chiếm rất nhiều bộ nhớ và nó có thể không sử dụng được. Với những ví dụ nhỏ như trên có lẽ bạn sẽ không thấy được sự quan trọng của việc này nhưng trong một chương trình lớn hơn, đây thật sự là một vấn đề quan trọng.

Việc quên không giải phóng bộ nhớ trước sau gì cũng sẽ xảy ra và bạn sẽ gặp phải một sự cố mang tên "tràn bộ nhớ". Chương trình của bạn sẽ sử dụng nhiều bộ nhớ hơn cần thiết mà bạn không hiểu được lý do tại sao. Thường thì nguyên nhân đơn giản chỉ vì 1 hoặc 2 chi tiết nhỏ như vì quên dùng *fclose*.

Những phương pháp đọc/ghi dữ liệu trong tập tin:

Sau khi đã biết cách mở và đóng một tập tin, bây giờ chúng ta chỉ việc thêm vào một vài dòng code để đọc và ghi dữ liệu vào.

Chúng ta sẽ bắt đầu với việc học cách ghi dữ liệu vào một tập tin trước (cũng đơn giản thôi) và sau đó bạn sẽ học cách làm thế nào để đọc tập tin.

Ghi vào một tập tin:

Có một vài hàm có chức năng ghi dữ liệu vào tập tin, việc chọn ra cách nào thích hợp nhất là phụ thuộc vào bạn.

Đây là 3 hàm mà chúng ta sẽ học:

- *fputc:* viết một ký tự vào tập tin (duy nhất mỗi lần 1 ký tự).
- *fputs:* viết một chuỗi vào tập tin.
- *fprintf:* viết một chuỗi có định dạng vào tập tin, gần giống như hàm *printf*.

Hàm fputc:

Hàm này sẽ thêm vào tập tin mỗi lần 1 ký tự. Đây là prototype của nó:

C code:

```
int fputc (int kytu, FILE* taptin);
```

Hàm này có 2 tham số:

- Tham số 1: Biến đại diện cho ký tự được viết thêm vào (biến được khai báo kiểu int, như tôi đã từng nói với bạn nó cũng tương đương khi khai báo kiểu char, khác ở chỗ số ký tự có thể sử dụng ở đây nhiều hơn). VD bạn có thể viết trực tiếp ký tự 'A'.
- Tham số 2: Con trỏ đến tập tin để viết. theo như vd của chúng ta con trỏ tên là *taptin*. Lợi thế của việc gọi con trỏ *taptin* mỗi lần cần sử dụng là có thể mở nhiều tập tin cùng lúc và nhờ vậy bạn có thể đọc và viết thêm vào mỗi tập tin. Bạn không bị giới hạn phải mở 1 tập tin tại 1 thời điểm.

Hàm này trả về một giá trị *int*, tương đương với giá trị của ký tự được thêm vào. Giá trị *int* này sẽ là EOF thể hiện cho 1 lỗi, nếu hàm trả về 1 giá trị khác EOF thì mọi thứ vẫn bình thường.

Giống như khi thao tác mở tập tin thành công, tôi không thường kiểm tra xem *fputc* có thực hiện tốt nhiệm vụ của nó không, nhưng nếu bạn muốn thì bạn cứ việc kiểm tra lại.

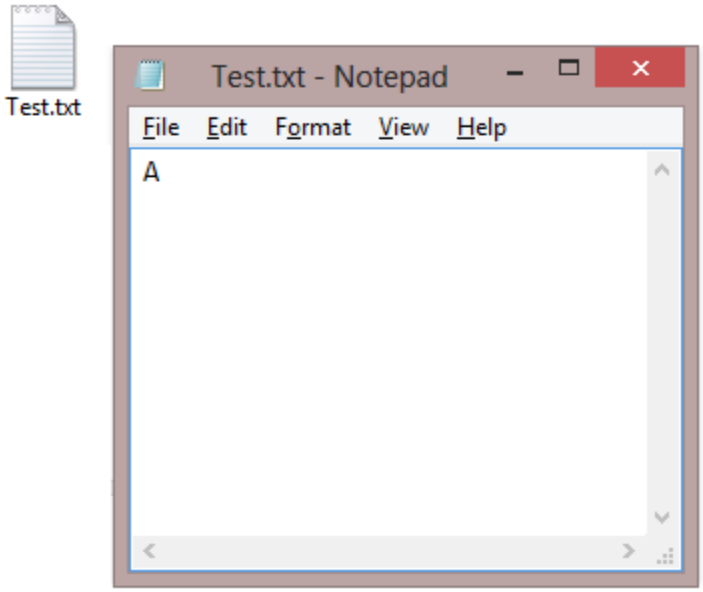
Đoạn code sau đây sẽ thêm ký tự 'A' trong *test.txt* (nếu tập tin đã tồn tại nó sẽ được thay thế, nếu chưa thì nó sẽ được tạo ra). Có đầy đủ mọi thứ trong đoạn code bên dưới, mở tập tin, ghi thêm dữ liệu và đóng tập tin:

C code:

```
int main (int argc, char *argv[ ])
{
   FILE* taptin = NULL;

   taptin = fopen("test.txt", "w");

   if (taptin != NULL)
   {
      fputc('A', taptin); // Ghi them vao tap tin ky tu A
      fclose(taptin);
   }

   return 0;
}
```

Thử mở tập tin *test.txt* của bạn ra để kiểm tra thử xem.

Thật kỳ diệu đúng không, bây giờ nó đã chứa ký tự A như hình dưới:



Hàm fputs:

Hàm này rất giống với hàm *fputc*, chỉ có một khác biệt đó là nó sẽ ghi vào tập tin một chuỗi (string), khác với *fputc* chỉ có thể thêm vào một ký tự.

Điều này không có nghĩa là hàm *fputc* trở nên vô dụng, bạn sẽ phải sử dụng *fputc* trong những trường hợp chương trình muốn yêu cầu người dùng điền một ký tự duy nhất (trả lời câu hỏi trắc nghiệm chẳng hạn).

Sau đây là prototype của hàm:

C code:

```
char* fputs(const char* chuoi, FILE* taptin);
```

Cả hai tham số của hàm này cũng rất đơn giản để hiểu:

Tham số 1: *chuoi*, đây là chuỗi được thêm vào tập tin. Để ý rằng tham số này có kiểu *const char**: bằng cách thêm từ *const* vào, hàm này muốn nói rằng chuỗi này sẽ được xem như một hằng số. Tóm lại, bạn sẽ không thể sửa nội dung của chuỗi, điều này sẽ giúp bạn hiểu là: hàm *fputs* chỉ đọc và thêm vào chuỗi của bạn chứ nó không hề thay đổi gì nội dung chuỗi, cũng có nghĩa là thông tin bạn muốn thêm vào sẽ được bảo vệ an toàn.

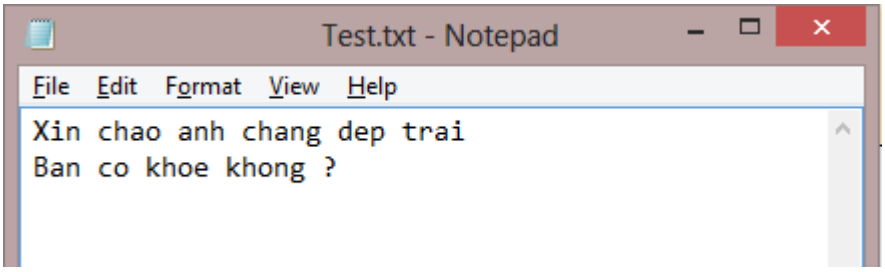
Tham số 2: *taptin*, tương tự như trong hàm *fputc*, nó là con trỏ FILE* của bạn để dẫn đề tập tin được đã được mở.

Nào bây giờ chúng ta sẽ thử thêm một chuỗi vào tập tin:

C code:

```
int main (int argc, char *argv[ ])
{
   FILE* taptin = NULL;

   taptin = fopen("test.txt", "w");

   if (taptin != NULL)
   {
      fputs("Xin chao anh chang dep trai\nBan co khoe khong ?", taptin);
      fclose(taptin);
   }

   return 0;
}
```

Tập tin sau khi chạy dòng code trên sẽ như sau:



Hàm fprintf:

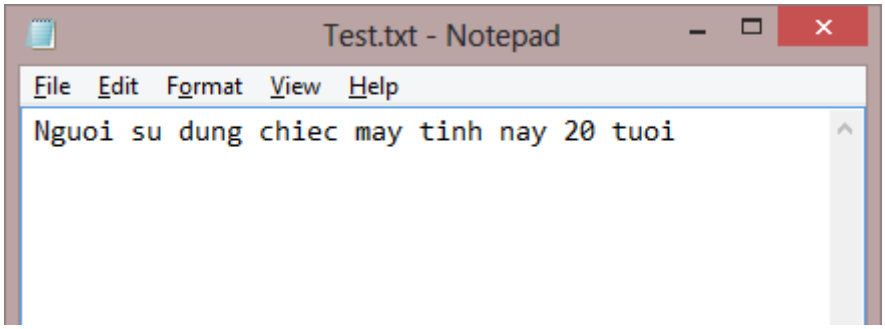
Sau đây là một vd khác cho hàm *printf*. Chúng ta có thể sử dụng cách này để ghi dữ liệu vào một tập tin. Cách sử dụng cũng gần giống như hàm *printf*, ngoại trừ việc bạn phải chỉ định một con trỏ ở vị trí tham số đầu tiên.

Đoạn code sau đây sẽ hỏi tuổi của một người và ghi kết quả nhận được vào tập tin:

C code:

```c
int main (int argc, char *argv[ ])
{
   FILE* taptin = NULL;
   int tuoi = 0;

   taptin = fopen("test.txt", "w");

   if (taptin != NULL)
   {
      // Sau đây chúng ta sẽ hỏi tuổi
      printf (" Ban bao nhieu tuoi ? ");
      scanf ("%d", &tuoi);

      // Và bây giờ là ghi dữ liệu vào tập tin
      fprintf (taptin, "Nguoi dang su dung chiec may tinh nay %d tuoi", tuoi);
      fclose(taptin);
   }

   return 0;
}
```

Và kết quả khi bạn mở tập tin *test.txt* sẽ như hình sau:



Bạn có thể vận dụng lại những kiến thức đã học về hàm *printf* để áp dụng cho việc ghi dữ liệu vào tập tin đối với hàm *fprintf*. Đây cũng là lý do tại sao tôi rất hay sử dụng *fprintf* mỗi lần muốn thêm dữ liệu vào tập tin.

Đọc một tập tin:

Chúng ta sẽ sử dụng lại hầu hết các hàm dùng để ghi dữ liệu ở trên, chỉ có một chút thay đổi trong tên của chúng:

1. *fgetc:* Đọc một ký tự
2. *fgets:* Đọc một chuỗi
3. *fscanf:* Đọc một chuỗi có định dạng

Lần này tôi sẽ đi nhanh hơn một chút trong việc giải thích về những hàm này. Nếu bạn đã hiểu những gì tôi nói ở trên về việc ghi dữ liệu vào tập tin thì những kiến thức này chỉ là chuyện nhỏ.

Hàm fgetc:

Prototype là:

C code:

```
int fgetc(FILE* taptin);
```

Hàm sẽ trả về một giá trị *int,* có nghĩa là ký tự đó đã được đọc. Ngược lại nếu không thể đọc được, hàm sẽ trả về EOF.

 Nhưng làm sao ta biết được mình sẽ đọc ký tự nào? Chẳng hạn như bạn muốn đọc ký tự thứ 3 hoặc thứ 10 thì làm sao để đọc?

Thật ra trong thực tế, khi bạn đọc một tập tin, sẽ xuất hiện một "dấu nháy ảo". Dấu nháy này sẽ không hiển thị lên màn hình cho bạn thấy. Bạn có thể tưởng tượng nó giống như dấu nháy khi bạn chỉnh sửa nội dung văn bản trên Notepad hoặc Microsoft Word. Nó có nhiệm vụ chỉ ra bạn đang ở đâu trong tập tin.

Chúng ta sẽ thấy ngay sau đây dấu nháy ảo nằm ở vị trí nào trong tập tin và làm thế nào để thay đổi vị trí dấu nháy đó.(chuyển nó về đầu tập tin hoặc một vị trí cụ thể nào đó của ký tự, vd như vị trí của ký tự thứ 10).

Mỗi lần bạn dùng *fgetc* để đọc một ký tự thì hàm sẽ gửi "dấu nháy ảo" vào đó. Nếu bạn gọi hàm này lại một lần nữa, thì hàm sẽ tiếp tục từ vị trí đó đọc tiếp ký tự thứ 2, thứ 3 và cứ thế đọc tiếp. Bạn có thể tạo ra một vòng lặp (loop) để lần lượt đọc từng ký tự trong tập tin.

Chúng ta sẽ viết một đoạn code để lần lượt đọc tất cả các ký tự trong tập tin, đồng thời in ra màn hình những ký tự đọc được. Vòng lặp sẽ ngừng lại khi hàm *fgetc* trả về giá trị EOF (End Of File, nhằm thông báo cho máy tính là "kết thúc một tập tin"):

C Code:

```
int main (int argc, char *argv[ ])
{
   FILE* taptin = NULL;
   int kytuHientai = 0;

   taptin = fopen("test.txt", "r");

   if (taptin != NULL)
   {
      // Vong lap lan luot doc tung ky tu
      do
      {
         kytuHientai = fgetc(taptin); // Doc ky tu
         printf ("%c", kytuHientai); // In ky tu do ra man hinh
      } while (kytuHientai != EOF); // fgetc tiep tuc duoc goi lai vi bien kytuHientai khac EOF
      fclose(taptin);
   }

   return 0;
}
```

Màn hình console sẽ hiển thị toàn bộ nội dung tập tin, VD:

Console:

```
Xin chao, day la noi dung cua tap tin test.txt !
```

Hàm fgets:

Hàm này sẽ đọc một chuỗi trong tập tin. Nó sẽ giúp bạn tiết kiệm thời gian thay vì phải đọc từng ký tự một. Hàm sẽ đọc một chuỗi trên một dòng (nó sẽ dừng lại khi gặp ký tự xuống dòng **\n).** Nếu bạn muốn đọc nhiều dòng thì hãy tạo ra một vòng lặp.

Sau đây là prototype của hàm *fgets:*

C code:

```
char* fgets(char* chuoi, int soKytuDuocdoc, FILE* taptin);
```

Hàm này có chứa một tham số hơi kỳ quặc, nhưng nó thực sự sẽ rất hữu ích: *soKytuDuocdoc*. Nó sẽ thông báo cho hàm *fgets* ngừng đọc dòng nội dung nếu vượt quá số lượng ký tự tương ứng.

Lợi ích: Nó sẽ bảo đảm rằng chúng ta sẽ không rơi vào tình trạng "tràn bộ nhớ". Thật vậy, nếu một dòng của bạn quá lớn so với chuỗi, hàm có thể sẽ phải đọc nhiều ký tự hơn. Điều này có thể gây ra lỗi cho chương trình.

Đầu tiên chúng ta sẽ xem làm thế nào để đọc nội dung trên một dòng với hàm *fgets* (chúng ta cũng sẽ biết cách để đọc toàn bộ tập tin).

Để thực hiện điều này, chúng ta sẽ tạo ra một chuỗi đủ lớn để chứa nội dung của dòng mà ta sẽ đọc (ít nhất là vậy, vì ta không thể chắc chắn được 100% nội dung dòng đó là bao nhiêu). Bạn sẽ thấy lợi ích của việc sử dụng định nghĩa (define) để xác định trước kích thước của mảng (array):

C code:

```
#define SO_KY_TU_TOI_DA 1000 // Kich thuoc cua mang la 1000

int main (int argc, char *argv[ ])
{
   FILE* taptin = NULL;
   char chuoi[SO_KY_TU_TOI_DA] = ""; // Chuoi co kich thuoc bang SO_KY_TU_TOI_DA

   taptin = fopen("test.txt", "r");

   if (taptin != NULL)
   {
      fgets (chuoi, SO_KY_TU_TOI_DA, taptin); /* Co toi da SO_KY_TU_TOI_DA trong tap
tin duoc doc, chung duoc luu tru vao "chuoi" */
      printf ("%s", chuoi); // Hien thi chuoi len man hinh

      fclose (taptin);
   }

   return 0;
}
```

Kết quả vẫn giống như code của hàm *fgetc* mà chúng ta đã thấy trước đó, đây là những nội dung hiển thị trên console:

Console:

```
Xin chao, day la noi dung cua tap tin test.txt !
```

Sự khác biệt ở đây là chúng ta không sử dụng vòng lặp. Nó hiển thị toàn bộ chuỗi trong một lần.

Chắc hẳn là bạn đã thấy những lợi ích của #define trong những dòng code trên nhằm xác định kích thước cho mảng, VD như SO_KY_TU_TOI_DA đã được sử dụng 2 lần trong đoạn code:

- Một lần để xác định kích thước cho mảng khi được khởi tạo.
- Và lần thứ hai trong hàm *fgets* nhằm giới hạn số ký tự tối đa sẽ được đọc.

Lợi ích của việc này là nếu bạn thấy rằng chuỗi chưa đủ lớn để đọc hết nội dung của một dòng trong tập tin, bạn chỉ cần thay đổi giá trị trên dòng định nghĩa (dòng chứa #define) và sau đó biên dịch lại chương trình. Điều này giúp bạn tiết kiệm thời gian vì không cần phải đọc lại toàn bộ code để tìm chỗ thay đổi kích thước của mảng. Tiền xử lý (preprocessor) sẽ thay thế toàn bộ giá trị cũ của SO_KY_TU_TOI_DA bằng giá trị mới mà bạn muốn.

Như tôi đã nói, hàm *fgets* đọc nội dung trên một dòng. Nó sẽ ngừng đọc dòng đó nếu vượt quá số ký tự tối đa được đọc do bạn quy định.

Nhưng câu hỏi là: Bây giờ nó chỉ đọc được mỗi lần một dòng vậy thì làm thế quái nào để ta có thể đọc được toàn bộ nội dung của tập tin? Câu trả lời vô cùng đơn giản: Vòng lặp (loop)

Hàm *fgets* sẽ trả về giá trị NULL nếu không đọc được nội dung mà bạn yêu cầu.

Các vòng lặp sẽ ngừng trước khi *fgets* trả về giá trị NULL.

Cần thêm vào một thứ để *fgets* không trả về NULL:

C code:

```
#define SO_KY_TU_TOI_DA 1000

int main (int argc, char *argv[ ])
{
   FILE* taptin = NULL;
   char chuoi[SO_KY_TU_TOI_DA] = "";

   taptin = fopen("test.txt", "r");

   if (taptin != NULL)
   {
      while (fgets(chuoi, SO_KY_TU_TOI_DA, taptin) != NULL); /* Cu viec doc noi dung tap
tin mien sao khong xuat hien loi (NULL)*/
      printf ("%s", chuoi); // Hien thi noi dung doc duoc len man hinh

      fclose(taptin);
   }

   return 0;
}
```

Đoạn code trên sẽ đọc toàn bộ nội dung của tập tin, từng dòng một.

Đây là những dòng thú vị nhất của đoạn code, dòng code có sử dụng vòng lặp while:

C code:

```
while (fgets (chuoi, SO_KY_TU_TOI_DA, taptin) != NULL);
```

Dòng code trên thực hiện 2 việc: Nó sẽ đọc nội dung của một dòng trong tập tin và kiểm tra xem *fgets* có trả về giá trị NULL hay không. Có thể hiểu nội dung của dòng code trên là "đọc nội dung của một dòng và tiếp tục đọc dòng tiếp theo cho tới khi kết thúc tập tin".

Hàm *fscanf*:

Vẫn là một nguyên tắc hoạt động tương tự với hàm *scanf* mà chúng ta đã từng học.

Hàm này đọc nội dung của tập tin nhưng nó phải được viết một cách chính xác.

Giả sử tập tin của bạn chứa 3 số được phân cách bằng khoảng trắng, VD đó là những điểm số cao nhất của một trò chơi 15 20 30.

Và bạn muốn lấy từng số đó dưới dạng biến kiểu int.

Hàm *fscanf* sẽ giúp bạn làm được điều đó một cách dễ dàng:

C code:

```
int main (int argc, char *argv[ ])
{
   FILE* taptin = NULL;
   int diemso[3] = {0}; // Mang chua 3 gia tri diem so cao nhat

   taptin = fopen("test.txt", "r");

   if (taptin != NULL)
   {
      fscanf (taptin, "%d %d %d", &diemso[0], &diemso[1], &diemso[2]);
      printf ("Cac diem so cao nhat la: %d, %d va %d", diemso[0], diemso[1], diemso[2]);

      fclose (taptin);
   }

   return 0;
}
```

Console:

```
Cac diem so cao nhat la: 15, 20 va 30
```

Như bạn sẽ thấy, hàm *fscanf* sẽ nhận biết 3 giá trị được phân cách nhau bằng những khoảng trắng ("%d %d %d"). Nó sẽ đưa vào mảng của chúng ta 3 thành phần. Sau đó bạn có thể dùng *printf* để hiển thị mỗi giá trị nhận được.

 Bạn có để ý rằng trước đây tôi chỉ đặt một **%d** trong dấu ngoặc kép của hàm *scanf* thì trong lần này với hàm *fscanf* chúng ta có thể đặt một lúc nhiều **%d** nhập giá trị. Nếu tập tin của bạn được viết theo một quy chuẩn rõ ràng thì việc thu thập các giá trị sẽ được tiến hành dễ dàng hơn.

Di chuyển một tập tin:

Khi nãy, tôi đã nói với bạn về một "dấu nháy ảo" đúng không? Bây giờ chúng ta sẽ tìm hiểu chi tiết hơn để thấy hết công dụng của nó.

Mỗi lần bạn mở một tập tin, sẽ có một "dấu nháy ảo" xuất hiện và chỉ ra vị trí hiện tại của bạn trong tập tin. Bạn có thể tưởng tượng nó tương tự như dấu nháy trong các trình soạn thảo văn bản (Notepad hoặc Microsoft Word), nó chỉ ra vị trí của bạn trong tập tin và bạn sẽ bắt đầu đọc hoặc ghi thêm dữ liệu từ vị trí đó.

Tóm lại, "dấu nháy ảo" cho phép bạn đọc hoặc ghi thêm dữ liệu vào một tập tin từ một vị trí cụ thể.

Có 3 hàm chúng ta cần phải biết:

ftel: cho biết vị trí hiện tại của bạn trong tập tin.

fseek: chỉ định vị trí của "dấu nháy ảo" tại một khu vực cụ thể.

rewind: đưa "dấu nháy ảo" về vị trí bắt đầu của tập tin (tương tự, chúng ta cũng có thể sử dụng *fseek* để chỉ định vị trí "dấu nháy ảo" về vị trí bắt đầu của tập tin).

Hàm *ftell*: Vị trí hiện tại trong tập tin

Cách sử dụng hàm này rất đơn giản. Nó trả về giá trị của "dấu nháy ảo" như một biến kiểu *long*.

C code:

```
long ftell (FILE* taptin);
```

Giá trị được trả về cho biết vị trí hiện tại của "dấu nháy ảo" trong tập tin.

Hàm *fseek*:

Prototype của hàm *fseek* là:

C code:

```
int fseek(FILE* taptin, long vitri_chuyenden, int vitri_hientai);
```

Hàm *fseek* sẽ di chuyển "dấu nháy ảo" từ vị trí gốc (được chỉ định bởi biến *vitri_hientai*) đến vị trí của một ký tự trong tập tin (được chỉ định theo giá trị của biến *vitri_chuyenden*).

- Giá trị của *vitri_chuyenden* có thể là một số dương (để dấu nháy di chuyển tiến lên), đứng im không di chuyên (giá trị bằng 0), và số âm (để di chuyển lùi lại).
- Giá trị khởi tạo có thể là một trong ba hằng số (constant) sau (khai báo #define nhé), xem nào:
    1. SEEK_SET: chỉ ra vị trí bắt đầu của tập tin
    2. SEEK_CUR: chỉ ra vị trí hiện tại của "dấu nháy ảo".
    3. SEEK_END: chỉ ra vị trí kết thúc của tập tin.

Sau đây là một vài ví dụ để chúng ta biết cách làm việc với những biến *vitri_hientai* và *vitri_hientai*.

- Đoạn code sau đây sẽ đặt "dấu nháy ảo" vào vị trí của ký tự thứ 2 ***sau*** vị trí bắt đầu tập tin:

C code:

```
fseek (taptin, 2, SEEK_SET);
```

- Đoạn code này sẽ đặt "dấu nháy ảo" vào vị trí của ký tự thứ 4 ***trước*** vị trí hiện tại của dấu nháy:

C code:

```
fseek (taptin, -4, SEEK_CUR);
```

Lưu ý là với giá trị âm như trên, dấu nháy sẽ di chuyển ngược về trước.

- Đoạn code sau đây sẽ đặt "dấu nháy ảo" về vị trí cuối tập tin:

C code:

```
fseek (taptin, 0, SEEK_END);
```

Nếu bạn ghi thêm dữ liệu vào sau vị trí kết thúc của tập tin, máy tính sẽ bổ sung thêm dữ liệu đó cho tập tin của bạn (lần sau khi mở lại tập tin này bạn sẽ thấy những thông tin được bổ sung thêm ở vị trí cuối cùng).

Nhưng nếu bạn đặt "dấu nháy ảo" ở đầu tập tin và bắt đầu ghi thêm dữ liệu vào thì lúc này, những dữ liệu cũ sẽ bị ghi đè lên. Chúng ta không thể "chèn" thêm dữ liệu vào tập tin trừ khi ta dùng một hàm để lưu lại những dữ liệu đứng sau trước khi chúng bị ghi đè lên.


Nhưng làm thế éo nào biết được vị trí nào là ở đâu để tôi tìm đến mà đọc hoặc ghi dữ liệu?

Điều này tùy thuộc vào cách sắp xếp nội dung tập tin của bạn. Nếu tập tin này là do bạn viết ra, thì chắc hẳn là bạn phải biết rõ nó có cấu trúc như thế nào. Bởi vậy bạn sẽ biết những thông tin mà bạn cần nằm ở đâu. VD: bạn sắp xếp những điểm số người chơi ở vị trí 0, tên của người chơi cuối cùng nằm ở vị trí thứ 50…

Sau này chúng ta sẽ làm việc với những tập tin khổng lồ, và nếu như bạn không có một quy tắc sắp xếp nội dung tập tin riêng của mình, bạn sẽ không biết phải làm thế nào để lấy những thông tin mà mình cần ở đâu. Hãy nhớ rằng bạn chính là người sắp xếp tất cả những nội dung này trong tập tin, tất cả đều tùy thuộc vào bạn. Chẳng hạn như bạn quy định:"tôi để điểm của người chơi thứ nhất tại dòng 1, điểm của người chơi thứ 2 tại dòng 2…"

Hàm *fseek* có thể sẽ không hoạt động tốt khi dùng nó để mở những tập tin chứa nội dung dạng văn bản. Nói chung người ta thường dùng nó để mở các tập tin chứa nội dung dạng nhị phân.

 Khi một người đọc hoặc ghi dữ liệu trong tập tin dạng văn bản, thường thì ký tự sẽ được thay thế bằng ký tự. Điều duy nhất hữu dụng trong chế độ tập tin văn bản khi sử dụng hàm *fseek* là nó giúp đưa bạn về vị trí đầu hoặc cuối tập tin.

Hàm rewind: quay về vị trí ban đầu.

Cách này tương tự như việc bạn sử dụng *fseek* để đưa "dấu nháy ảo" về vị trí 0 của tập tin.

C code:

```
void rewind (FILE* taptin);
```

Cách viết hàm của nó giống như nguyên mẫu ở trên, không có gì để giải thích thêm.

Đổi tên và xóa tập tin:

Chúng ta sẽ kết thúc bài học kỳ này với 2 hàm đơn giản:

- *rename*: đổi tên tập tin.
- *remove*: xóa tập tin.

Điểm khác biệt ở những hàm này là chúng không yêu cầu bạn sử dụng con trỏ *taptin.* Những hàm này chỉ cần xác định tên của tập tin để xóa hoặc đổi tên tập tin đó.

Hàm rename: Đổi tên tập tin.

Prototype của hàm là:

C code:

```
int rename(const char* teCu, const char* tenMoi);
```

Hàm sẽ trả về giá trị 0 nếu đổi tên thành công. Nếu không thành công nó sẽ trả về một giá trị khác 0. Và sau đây là một ví dụ:

C code:

```
int main (int argc, char *argv[ ])
{
   rename("test.txt", "test_rename.txt");

   return 0;
}
```

Hàm remove: Xóa tập tin.

Hàm này sẽ xóa tập tin ngay và luôn mà không cần hỏi ý kiến hay thông báo cho bạn.

C code:

```
int remove(const char* tentaptinMuonXoa);
```

 Bạn phải cẩn thận khi sử dụng hàm này. Nó sẽ xóa tập tin của bạn mà không cần xác nhận lại. Tập tin sẽ không bị đưa vào thùng rác (recycle bin) mà nó sẽ bị xóa hoàn toàn khỏi máy tính của bạn. Sẽ không có cách nào khôi phục lại tập tin đó (trừ khi bạn dùng thủ thuật đặc biệt để khôi phục lại tập tin, nhưng thường thì rất khó thành công).

Và hàm *remove* này cũng sẽ kết thúc bài học đúng theo công năng của nó. Chúng ta đã tạo ra tập tin *test.txt* và bây giờ tự tay mình sẽ xóa nó:

C code:

```
int main (int argc, char *argv[ ])
{
   remove("test.txt");

   return 0;
}
```

Kết thúc bài học!
