ĐẠI HỌC GIAO THÔNG VẬN TẢI

– PHÂN HIỆU TẠI THÀNH PHỐ

HỒ CHÍ MINH

BÁO CÁO BÀI TẬP LỚN

NHÓM 17

ĐỀ TÀI: QUẢN LÝ THƯ VIỆN

Thành viên:

- Nguyễn Hoàng Phát : 6151071082 (Nhóm trưởng)
- Trần Trọng Nhân : 6151071016
- Lê Tô Nguyện : 6151071015

MỤC LỤC

3. **Hướng phát triển**
4. **Tài liệu tham khảo**
5. **Bảng phân công công việc mỗi thành viên**

NỘI DUNG

LỜI GIỚI THIỆU

Sau quá trình học tập và rèn luyện tại bộ môn Công nghệ thông tin trường Đại học Giao thông Vận tải – Phân hiệu tại thành phố Hồ Chí Minh em đã được trang bị các kiến thức cơ bản, các kỹ năng thực tế để có thể hoàn thành đề tài "Quản lí thư viện" của mình. Nhóm 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, và thực hiện bài tập lớn một cách tận tình và tâm huyết. Nhóm em xin chúc quý thầy cô thật nhiều sức khỏe và luôn đạt được thành công trong cuộc sống. Đặc biệt nhóm em xin cảm ơn cô Trần Thị Dung người đã hướng dẫn và chỉ bảo trong quá trình thực hiện bài tập này. Sau một thời gian nỗ lực thực hiện thì đề tài cũng đã hoàn thành, nhưng không sao tránh khỏi những sai sót do nhóm em còn chưa có nhiều kinh nghiệm. Nhóm em kính mong nhận được sự góp ý và nhận xét từ thầy để em có thể hoàn thiện và hoàn thành tốt hơn cho đề tài của mình. Lời sau cùng em một lần nữa kính chúc cô Trần Thị Dung 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.

CHƯƠNG I: TỔNG QUAN VỀ CHƯƠNG TRÌNH QUẢN LÝ THƯ VIỆN

Chúng ta không thể phủ nhận rằng hiện nay với tốc độ phát triển của internet thì việc tìm kiếm thông tin trở nên vô cùng dễ dàng. Nhưng cùng theo đó là thông tin thu thập được vô cùng hỗn tạp. Và chúng ta cần phải có sự chọn lọc thông tin một cách cẩn thận để thu thập được những thông tin chính xác và loại bỏ những thông tin nhiễu. Do vậy có rất nhiều người vẫn lựa chọn cách đến thư viện để tìm kiếm thông tin hoặc dành thời gian rảnh để đến thư viện đơn giản chỉ vì đó là niềm yêu thích của họ. Một thư viện với hàng trăm ngàn đầu sách thì chúng ta không thể quản lý thủ công bằng giấy tờ hay ghi chép. Cần có một chương trình quản lý chung về

mã sách, tên sách hay những thao tác sửa sách, xóa sách … để dễ dàng quản lý những vấn đề trên, nhóm của em đã viết một chương trình quản lý thư viện. Chương trình quản lý thư viện hỗ trợ đắc lực cho việc quản lý nguồn tài nguyên hiện có trong thư viện nhanh chóng và chính xác.

1. Lợi ích của chương trình quản lý thư viện

- Thông tin về mã sách được quản lý tập trung giúp dễ dàng tìm kiếm và phân loại.

2. Chức năng của chương trình quản lý thư viện

- Nhập thông tin về quyển sách muốn thêm
- Sắp xếp, thống kê và hiển thị thông tin chi tiết của từng quyển sách theo thể loại
- Sửa thông tin sách
- Xóa thông tin sách
- Tìm quyển sách theo thể loại
- Ghi vào tập tin nhị phân book.dat

CHƯƠNG II: CƠ SỞ LÝ THUYẾT

1. Mảng trong Ngôn ngữ C

Cấu trúc dữ liệu mảng:

Mảng (Array) là một trong các cấu trúc dữ liệu cũ và quan trọng nhất. Mảng có thể lưu giữ một số phần tử cố định và các phần tử này nền có cùng kiểu. Hầu hết các cấu trúc dữ liệu đều sử dụng mảng để triển khai giải thuật. Dưới đây là các khái niệm quan trọng liên quan tới Mảng.

- **Phần tử**: Mỗi mục được lưu giữ trong một mảng được gọi là một phần tử.
- **Chỉ mục (Index)**: Mỗi vị trí của một phần tử trong một mảng có một chỉ mục số được sử dụng để nhận diện phần tử.

Mảng gồm các bản ghi có kiểu giống nhau, có kích thước cố định, mỗi phần tử được xác định bởi chỉ số

Mảng là cấu trúc dữ liệu được cấp phát lien tục cơ bản

Ưu điểm của mảng :

Truy cập phàn tử vơi thời gian hằng số O(1)

Sử dụng bộ nhớ hiệu quả

Tính cục bộ về bộ nhớ

Nhược điểm

Không thể thay đổi kích thước của mảng khi chương trình dang thực hiện

Mảng động

Mảng động (dynamic aray) : cấp phát bộ nhớ cho mảng một cách động trong quá trình chạy chương trình trong C là malloc và calloc, trong C++ là new

Sử dụng mảng động ta bắt đầu với mảng có 1 phàn tử, khi số lượng phàn tử vượt qua khả năng của ảng thì ta gấp đôi kích thước mảng cuc và copy phàn tử mảng cũ vào nửa đầu của mảng mới

Ưu điểm : tránh lãng phí bộ nhớ khi phải khai báo mảng có kích thước lớn ngay từ đầu

Nhược điểm: + phải thực hiện them thao tác copy phần tử mỗi khi thay đổi kích thước.
+ một số thời gian thực hiện thao tác không còn là hằng số nữa

Bảng sau đây cung cấp một số thông tin các hàm trong việc cấp phát bộ nhớ:

STT	Hàm và Miêu tả
1	**void *calloc(int tongkichco, int kichco);** Hàm này cấp phát một mảng các phần tử có tổng kích thước là **tongkichco** mà kích cỡ của mỗi phần tử được tính bằng byte sẽ là **kichco**.
2	**void free(void *diachi);** Hàm này giải phóng một khối bộ nhớ được xác định bởi diachi.
3	**void *malloc(int tongkichco);** Hàm này cấp phát bộ nhớ động với kích thước **tongkichco**.
4	**void *realloc(void *diachi, int kichco_moi);**

	Hàm này để thay đổi kích cỡ bộ nhớ đã cấp phát thành kích cỡ mới **kichco_moi**.

Cấp phát bộ nhớ động trong C

Khi bạn lập trình, bạn phải nhận thức về độ lớn của một mảng, sau đó nó là dễ dàng cho việc định nghĩa mảng. Ví dụ, bạn lưu trữ một tên của người bất kỳ nào, nó có thể lên tới tối đa 100 ký tự vì thế bạn có thể định nghĩa như sau:

```
char ten_mang[100];
```

Bây giờ hãy xem xét trường hợp bạn không có một ý tưởng nào về độ lớn của mảng bạn dự định lưu trữ, ví dụ bạn muốn lưu trữ một miêu tả chi tiết về một chủ đề. Tại đây bạn cần định nghĩa một con trỏ tới ký tự mà không định nghĩa bao nhiêu bộ nhớ được yêu cầu và sau đó dựa vào yêu cầu chúng ta sẽ cấp phát bộ nhớ như ví dụ dưới đây:

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

int main()
{
   char tennhanvien[100];
   char *mieuta;

   strcpy(tennhanvien, "Tran Minh Chinh");

   /* Cap phat bo nho dong */
   mieuta = (char *) malloc(200);
   if( mieuta == NULL )
   {
      fprintf(stderr, "Error - khong the cap phat bo nho theo yeu cau\n");
   }
   else
   {
      strcpy( mieuta, "Chinh la nhan vien IT co nang luc chem gio tot!!!");
   }
   printf("Ten nhan vien la: %s\n", tennhanvien );
```

```
   printf("Mieu ta: %s\n", mieuta );


}
```

Chương trình như trên có thể được viết bởi sử dụng **calloc()**, thay cho malloc như sau:

```
mieuta = (char*)calloc(200, sizeof(char));
```

Như thế là bạn đã hoàn toàn điều khiển việc cấp phát bộ nhớ và bạn có thể truyền bất cứ giá trị kích cỡ nào trong khi cấp phát bộ nhớ, không giống như mảng có độ dài cố định không thể thay đổi được.

Thay đổi và giải phóng bộ nhớ trong C

Khi chương trình của bạn kết thúc, hệ điều hành sẽ tự động giải phóng bộ nhớ cấp phát cho chương trình, nhưng trong thực tế khi bạn không cần bộ nhớ nữa, bạn nên giải phóng bộ nhớ bằng cách sử dụng hàm **free()**.

Một cách khác, bạn có thể tăng hoặc giảm cỡ của khối bộ nhớ đã cấp phát bằng cách gọi hàm **realloc()**. Hãy kiểm tra chương trình trên lại một lần và sử dụng hàm realloc() và free():

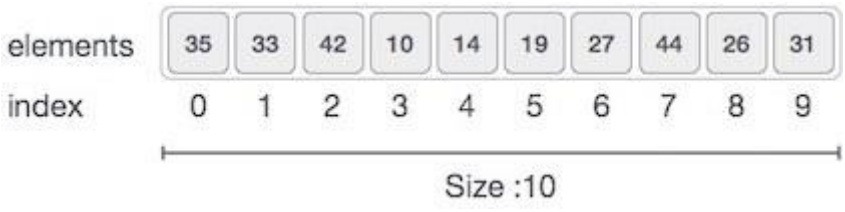
```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int main()
{
   char tennhanvien[100];
   char *mieuta;

   strcpy(tennhanvien, "Tran Minh Chinh");

   /* Cap phat bo nho dong */
   mieuta = (char *) malloc(100);
   if( mieuta == NULL )
   {
      fprintf(stderr, "Error - khong the cap phat bo nho theo yeu cau\n");
   }
   else
```

```
   {
      strcpy( mieuta, "Chinh la nhan vien IT co nang luc chem gio tot!!!");
   }
   /* Gia su ban muon luu tru mot mieuta nho hon */
   mieuta = (char*)calloc(50, sizeof(char));
   if( mieuta == NULL )
   {
      fprintf(stderr, "Error - khong the cap phat bo nho theo yeu cau\n");
   }
   else
   {
      strcat( mieuta, "Anh ta rat gioi!!!");
   }

   printf("Ten nhan vien: %s\n", tennhanvien );
   printf("Mieu ta: %s\n", mieuta );

   /* giai phong bo nho voi ham free() */
   free(mieuta);
}
```

Biểu diễn Cấu trúc dữ liệu mảng

Mảng có thể được khai báo theo nhiều cách đa dạng trong các ngôn ngữ lập trình. Để minh họa, chúng ta sử dụng phép khai báo mảng trong ngôn ngữ C:

Hình minh họa phần tử và chỉ mục:



Dưới đây là một số điểm cần ghi nhớ về cấu trúc dữ liệu mảng:

- Chỉ mục bắt đầu với 0.
- Độ dài mảng là 10, nghĩa là mảng có thể lưu giữ 10 phần tử.
- Mỗi phần tử đều có thể được truy cập thông qua chỉ mục của phần tử đó. Ví dụ, chúng ta có thể lấy giá trị của phần tử tại chỉ mục 6 là 27.

Phép toán cơ bản được hỗ trợ bởi mảng

Dưới đây là các hoạt động cơ bản được hỗ trợ bởi một mảng:

- **Duyệt**: In tất cả các phần tử mảng theo cách in từng phần tử một.
- **Chèn**: Thêm một phần tử vào mảng tại chỉ mục đã cho.
- **Xóa**: Xóa một phần tử từ mảng tại chỉ mục đã cho.
- **Tìm kiếm**: Tìm kiếm một phần tử bởi sử dụng chỉ mục hay bởi giá trị.
- **Cập nhật**: Cập nhật giá trị một phần tử tại chỉ mục nào đó.

Trong ngôn ngữ C, khi một mảng được khởi tạo với kích cỡ ban đầu, thì nó gán các giá trị mặc định cho các phần tử của mảng theo thứ tự sau:

Kiểu dữ liệu	Giá trị mặc định
bool	false
char	0
int	0
float	0.0
double	0.0f
void	
wchar_t	0

Hoạt động chèn phần tử vào mảng

Hoạt động chèn là để chèn một hoặc nhiều phần tử dữ liệu vào trong một mảng. Tùy theo yêu cầu, phần tử mới có thể được chèn vào vị trí đầu, vị trí cuối hoặc bất kỳ vị trí chỉ mục đã cho nào của mảng.

Phần tiếp theo chúng ta sẽ cùng triển khai hoạt động chèn trong một ví dụ thực. Trong ví dụ này, chúng ta sẽ chèn dữ liệu vào cuối mảng.

Ví dụ

Giả sử LA là một mảng tuyến tính không có thứ tự có N phần tử và K là một số nguyên dương thỏa mãn K <= N. Dưới đây là giải thuật chèn phần tử A vào vị trí thứ K của mảng LA.

Giải thuật

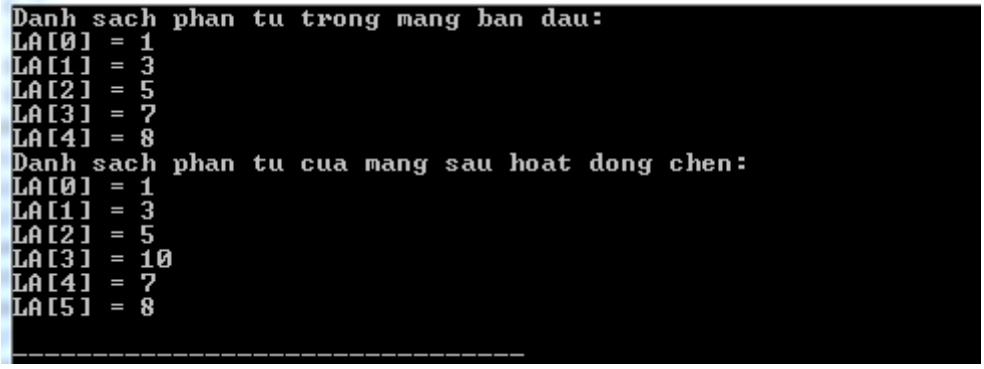
```
1. Bắt đầu
2. Gán J=N
3. Gán N = N+1
4. Lặp lại bước 5 và 6 khi J >= K
5. Gán LA[J+1] = LA[J]
6. Gán J = J-1
7. Gán LA[K] = ITEM
8. Kết thúc
```

Sau đây là code đầy đủ của giải thuật trên trong ngôn ngữ C:

```c
#include <stdio.h>

main() {

   int LA[] = {1,3,5,7,8};

   int item = 10, k = 3, n = 5;

   int i = 0, j = n;


   printf("Danh sach phan tu trong mang ban dau:\n");


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

      printf("LA[%d] = %d \n", i, LA[i]);

   }


   n = n + 1;


   while( j >= k){

      LA[j+1] = LA[j];

      j = j - 1;

   }


   LA[k] = item;


   printf("Danh sach phan tu cua mang sau hoat dong chen:\n");
```

```
   for(i = 0; i<n; i++) {
      printf("LA[%d] = %d \n", i, LA[i]);
   }
}
```

Biên dịch và chạy chương trình C trên sẽ cho kết quả:



Hoạt động xóa phần tử từ mảng

Hoạt động xóa là xóa một phần tử đang tồn tại từ một mảng và tổ chức lại các phần tử còn lại trong mảng đó.

Ví dụ

Giả sử LA là một mảng tuyến tính có N phần tử và K là số nguyên dương thỏa mãn K <= N. Dưới đây là thuật toán để xóa một phần tử có trong mảng LA tại vị trí K.

Giải thuật

```
1. Bắt đầu
2. Gán J=K
3. Lặp lại bước 4 và 5 trong khi J < N
4. Gán LA[J-1] = LA[J]
5. Gán J = J+1
6. Gán N = N-1
7. Kết thúc
```

Sau đây là code đầy đủ của giải thuật trên trong ngôn ngữ C:

```
#include <stdio.h>

main() {

   int LA[] = {1,3,5,7,8};

   int k = 3, n = 5;

   int i, j;


   printf("Danh sach phan tu trong mang ban dau:\n");
```

```
   for(i = 0; i<n; i++) {
      printf("LA[%d] = %d \n", i, LA[i]);
   }

   j = k;

   while( j < n){
      LA[j-1] = LA[j];
      j = j + 1;
   }

   n = n -1;

   printf("Danh sach phan tu trong mang sau hoat dong xoa:\n");

   for(i = 0; i<n; i++) {
      printf("LA[%d] = %d \n", i, LA[i]);
   }
}
```

Biên dịch và chạy chương trình C trên sẽ cho kết quả:

```
Danh sach phan tu trong mang ban dau:
LA[0] = 1
LA[1] = 3
LA[2] = 5
LA[3] = 7
LA[4] = 8
Danh sach phan tu trong mang sau hoat dong xoa:
LA[0] = 1
LA[1] = 3
LA[2] = 7
LA[3] = 8

--------------------------------
```

Hoạt động tìm kiếm

Bạn có thể thực hiện hoạt động tìm kiếm phần tử trong mảng dựa vào giá trị hay chỉ mục của phần tử đó.

Ví dụ

Giả sử LA là một mảng tuyến tính có N phần tử và K là số nguyên dương thỏa mãn K <= N. Dưới đây là giải thuật để tìm một phần tử ITEM bởi sử dụng phương pháp tìm kiếm tuần tự (hay tìm kiếm tuyến tính).

Giải thuật

```
1. Bắt đầu
2. Gán J=0
3. Lặp lại bước 4 và 5 khi J < N
4. Nếu LA[J] là bằng ITEM THÌ TỚI BƯỚC 6
5. Gán J = J +1
6. In giá trị J, ITEM
7. Kết thúc
```

Sau đây là code đầy đủ của giải thuật trên trong ngôn ngữ C:

```c
#include <stdio.h>

main() {

   int LA[] = {1,3,5,7,8};

   int item = 5, n = 5;

   int i = 0, j = 0;


   printf("Danh sach phan tu trong mang ban dau:\n");


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

      printf("LA[%d] = %d \n", i, LA[i]);

   }


   while( j < n){


      if( LA[j] == item ){

         break;

      }


      j = j + 1;

   }


   printf("Tim thay phan tu %d tai vi tri %d\n", item, j+1);

}
```

Biên dịch và chạy chương trình C trên sẽ cho kết quả:

```
Danh sach phan tu trong mang ban dau:
LA[0] = 1
LA[1] = 3
LA[2] = 5
LA[3] = 7
LA[4] = 8
Tim thay phan tu 5 tai vi tri 3

--------------------------------
```

Hoạt động cập nhật (Hoạt động update)

Hoạt động cập nhật là update giá trị của phần tử đang tồn tại trong mảng tại chỉ mục đã cho.

Giải thuật

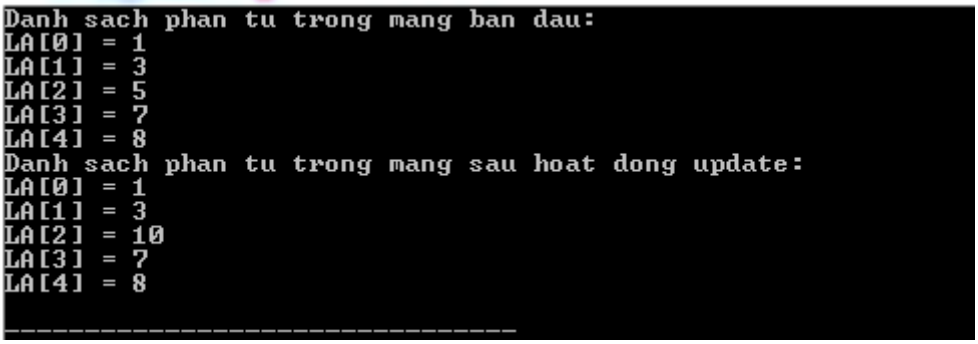
Giả sử LA là một mảng tuyến tính có N phần tử và K là số nguyên dương thỏa mãn K <= N. Dưới đây là giải thuật để update giá trị phần tử tại vị trí K của mảng LA.

```
1. Bắt đầu
2. Thiết lập LA[K-1] = ITEM
3. Kết thúc
```

Sau đây là code đầy đủ của giải thuật trên trong ngôn ngữ C:

```c
#include <stdio.h>

main() {

   int LA[] = {1,3,5,7,8};

   int k = 3, n = 5, item = 10;

   int i, j;


   printf("Danh sach phan tu trong mang ban dau:\n");


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

      printf("LA[%d] = %d \n", i, LA[i]);

   }


   LA[k-1] = item;


   printf("Danh sach phan tu trong mang sau hoat dong update:\n");


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

      printf("LA[%d] = %d \n", i, LA[i]);

   }

}
```

Biên dịch và chạy chương trình C trên sẽ cho kết quả:



2. Cấu trúc (Struct) trong C

Các mảng trong C cho phép bạn định nghĩa một vài loại biến có thể giữ giá trị của một vài thành phần cùng kiểu dữ liêu. Nhưng **structure - cấu trúc** là một loại dữ liệu khác trong ngôn ngữ lập trình C, cho phép bạn kết hợp các dữ liệu khác kiểu nhau.

Cấu trúc được sử dụng để biểu diễn một bản ghi. Giả sử bạn muốn lưu trữ giá trị của một quyển sách trong thư viện của bạn. Bạn có thể lưu trữ các thuộc tính của sách sau đây:

- Tiêu đề
- Tác giả
- Chủ đề
- ID (giống như là mã số sinh viên của bạn)

Định nghĩa một cấu trúc trong C

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

```
struct [ten_cau_truc]
{
   phan dinh nghia thanh vien cau truc;
   phan dinh nghia thanh vien cau truc;
   ...
   phan dinh nghia thanh vien cau truc;
} [mot hoac nhieu bien cau truc];
```

Một **ten_cau_truc** có thể tùy chọn và một thành phần định nghĩa là các biến thường như int i, float j hoặc một định nghĩa biến khác …. Tại phần cuối cùng của định nghĩa cấu trúc, trước dấu chấm phẩy, bạn có thể xác định một hoặc nhiều biến cấu trúc (tùy chọn). Dưới đây là cách khai báo biến structure Book:

```
struct Books
{
   char  tieude[50];
   char  tacgia[50];
   char  chude[100];
   int   id;
} book;
```

Truy cập các thành phần của cấu trúc trong C

Để truy cập bất kỳ thành phần nào của cấu trúc, bạn sử dụng **toán tử truy cập phần tử**. Dưới đây là ví dụ cho cách sử dụng cấu trúc:

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

struct Books
{
   char  tieude[50];
   char  tacgia[50];
   char  chude[100];
   int   id;
};

int main( )
{
   struct Books Book1;        /* Khai bao Book1 la cua kieu Book */
   struct Books Book2;        /* Khai bao Book2 la cua kieu Book */

   /* thong tin chi tiet quyen sach thu nhat */
   strcpy( Book1.tieude, "Lap trinh C");
   strcpy( Book1.tacgia, "Pham Van At");
   strcpy( Book1.chude, "Ngon ngu lap trinh C");
   Book1.id = 1234567;

   /* thong tin chi tiet quyen sach thu hai */
   strcpy( Book2.tieude, "Toi thay hoa vang tren co xanh");
   strcpy( Book2.tacgia, "Nguyen Nhat Anh");
```

```
   strcpy( Book2.chude, "Van hoc");
   Book2.id = 6677028;

   /* hien thi thong tin Book1 */
   printf( "Tieu de cua Book1 la: %s\n", Book1.tieude);
   printf( "Tac gia cua Book1 la: %s\n", Book1.tacgia);
   printf( "Chu de cua Book1 la: %s\n", Book1.chude);
   printf( "ID cua Book1 la: %d\n", Book1.id);

   /* hien thi thong tin Book2 */
   printf( "Tieu de cua Book2 la: %s\n", Book2.tieude);
   printf( "Tac gia cua Book2 la: %s\n", Book2.tacgia);
   printf( "Chu de cua Book2 la: %s\n", Book2.chude);
   printf( "ID cua Book2 la: %d\n", Book2.id);

   printf("\n============================\n");
   return 0;
}
```

Các cấu trúc như các tham số hàm

Bạn có thể đặt cấu trúc như một tham số của hàm theo cách dễ dàng như các biến khác hay con trỏ. Truy cập biến cấu trúc như ví dụ dưới đây:

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

struct Books
{
   char  tieude[50];
   char  tacgia[50];
   char  chude[100];
   int   id;
};

/* khai bao ham */
void inthongtinsach( struct Books book );
int main( )
```

```
{
   struct Books Book1;        /* Khai bao Book1 la cua kieu Book */
   struct Books Book2;        /* Khai bao Book2 la cua kieu Book */

   /* thong tin chi tiet quyen sach thu nhat */
   strcpy( Book1.tieude, "Lap trinh C");
   strcpy( Book1.tacgia, "Pham Van At");
   strcpy( Book1.chude, "Ngon ngu lap trinh C");
   Book1.id = 1234567;

   /* thong tin chi tiet quyen sach thu hai */
   strcpy( Book2.tieude, "Toi thay hoa vang tren co xanh");
   strcpy( Book2.tacgia, "Nguyen Nhat Anh");
   strcpy( Book2.chude, "Van hoc");
   Book2.id = 6677028;

   /* hien thi thong tin Book1 */
   inthongtinsach( Book1 );

   /* hien thi thong tin Book2 */
   inthongtinsach( Book2 );
    return 0;
}
void inthongtinsach( struct Books book )
{
   printf( "Tieu de sach: %s\n", book.tieude);
   printf( "Tac gia: %s\n", book.tacgia);
   printf( "Chu de: %s\n", book.chude);
   printf( "Book ID: %d\n", book.id);
}
```

3. Đọc - Ghi File trong C

Một file biểu diễn một chuỗi các bytes, không kể đó là file văn bản hay file nhị phân. Ngôn ngữ lập trình C cung cấp các hàm truy cập mức độ cao cũng như thấp (mức hệ điều hành) để thao tác với file trên thiết bị lưu trữ. Chương này sẽ đưa bạn đến những cách gọi hàm quan trọng cho việc quản lý file.

Mở file trong C

Bạn có thể sử dụng hàm **fopen()** để tạo file mới hoặc để mở các file đã tồn tại. Cách gọi này sẽ khởi tạo đối tượng loại **FILE**, mà bao gồm thông tin cần thiết để điều khiển luồng. Dưới đây là một cách gọi hàm:

```
FILE *fopen( const char * ten_file, const char * che_do );
```

Ở đây, **ten_file** là một chuỗi, được coi như tên file và giá trị **che_do** truy cập có thể là những giá trị dưới đây:

Mode	Miêu tả
r	Mở các file đã tồn tại với mục đích đọc
w	Mở các file với mục đích ghi. Nếu các file này chưa tồn tại thi file mới được tạo ra. Ở đây, chương trình bắt đầu ghi nội dung từ phần mở đầu của file
a	Mở file văn bản cho việc ghi ở chế độ ghi tiếp theo vào cuối, nếu nó chưa tồn tại thì file mới được tạo. Đây là chương trình ghi nội dung với phần cuối của file đã tồn tại.
r+	Mở file văn bản với mục đích đọc và ghi.
w+	Mở một file văn bản cho chế độ đọc và ghi. Nó làm trắng file đã tồn tại nếu file này có và tạo mới nếu file này chưa có.
a+	Mở file đã tồn tại với mục đích đọc và ghi. Nó tạo file mới nếu không tồn tại. Việc đọc file sẽ bắt đầu đọc từ đầu nhưng ghi file sẽ chỉ ghi vào cuối file.

Nếu bạn thao tác với các file nhị phân, bạn có thể có các cách truy xuất thay cho các trường hợp trên như sau:

```
"rb", "wb", "ab", "rb+", "r+b", "wb+", "w+b", "ab+", "a+b"
```

Đóng file trong C

Để đóng 1 file bạn có thể sử dụng hàm **fclose()** dưới đây:

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

Hàm **fclose( )** trả về giá trị zero nếu thành công hoặc **EOF** nếu có lỗi trong quá trình đóng file. Hàm này thực tế xóa các dữ liệu trong bộ đệm đối với file, đóng file và giải phóng bộ nhớ được sử dụng với file. **EOF** là một hằng số được định nghĩa trong phần **stdio.h**.

Có nhiều hàm đa dạng được cung cấp bởi thư viện chuẩn của ngôn ngữ C để đọc và ghi từng ký tự và trong một dạng với số lượng ký tự cố định. Chúng ta sẽ xem xét trong ví dụ sau đây:

Ghi tới một file trong C

Dưới đây là hàm đơn giản nhất để thực hiện việc ghi các ký tự riêng tới một luồng:

```
int fputc( int c, FILE *fp );
```

Hàm **fputc()** ghi các ký tự với giá trị tham số c đến một luồng ra tham chiếu bởi con trỏ fp. Nó sẽ trả về ký tự được ghi nếu thành công hoặc **EOF** nếu có lỗi. Bạn có thể sử dụng hàm sau đây để ghi một chuỗi kết thúc bằng ký tự null đến một luồng:

```
int fputs( const char *s, FILE *fp );
```

Hàm **fputs()** ghi chuỗi **s** đến một luồng ra tham chiếu bởi fp. Nó trả về một giá trị không âm nếu thành công và trả về ký tự EOF nếu xảy ra một lỗi. Bạn có thể sử dụng hàm **int fprintf(FILE *fp,const char *format, ...)** để ghi một chuỗi ra file . Thử ví dụ dưới đây:

Bạn phải chắc chắn bạn có thư mục **/tmp**, nếu không có, bạn phải tạo thư mục này trên máy bạn.

```
#include <stdio.h>


main()
{
   FILE *fp;


   fp = fopen("vidu.txt", "w+");
   fprintf(fp, "Vi du kiem tra ham fprintf ...\n");
   fputs("Vi du kiem tra ham fputs ...\n", fp);
   fclose(fp);
}
```

Khi đoạn code trên được biên dịch và thực hiện, nó tạo file mới là **vidu.txt** và ghi vào đó 2 dòng của 2 hàm khác nhau. Cùng đọc file này ở phần tiếp theo.

Đọc file trong C

Dưới đây là hàm đơn giản nhất để đọc một ký tự riêng rẽ từ file:

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

Hàm **fgetc()** đọc một ký tự từ một file tham chiếu bởi con trỏ fp. Giá trị trả về là ký tự đọc được nếu thành công, và trong trường hợp lỗi trả về **EOF**. Hàm dưới đây cho phép bạn đọc chuỗi từ một luồng:

```
char *fgets( char *buf, int n, FILE *fp );
```

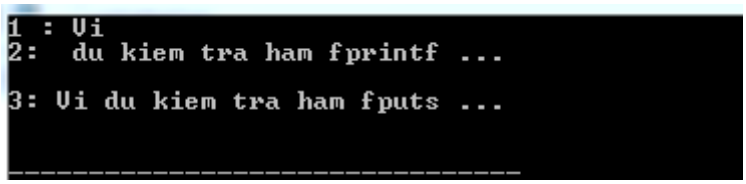
Hàm **fgets()** đọc n-1 ký tự từ một luồng vào tham chiếu bởi fp. Nó copy chuỗi đọc đến bộ đệm **buf**, gán ký tự **null** vào kết thúc chuỗi.

Nếu hàm gặp phải một ký tự newline (dòng mới) (xuống dòng) '\n' hoặc ký tự EOF trước khi đọc được số lượng tối đa các ký tự, nó sẽ chỉ trả về các ký tự cho đến ký tự xuống dòng và ký tự xuống dòng mới. Bạn có thể sử dụng hàm **int fscanf(FILE *fp, const char *format, ...)** để đọc chuỗi từ một file, nhưng dừng việc đọc ở khoảng trắng đầu tiên gặp phải:

```
#include <stdio.h>


main()
{
   FILE *fp;
   char buff[255];


   fp = fopen("vidu.txt", "r");
   fscanf(fp, "%s", buff);
   printf("1 : %s\n", buff );


   fgets(buff, 255, (FILE*)fp);
   printf("2: %s\n", buff );


   fgets(buff, 255, (FILE*)fp);
   printf("3: %s\n", buff );
   fclose(fp);
```

```
}
```

Biên dịch và chạy chương trình C trên, đầu tiên nó đọc từ file được tạo từ khu vực trước và in ra kết quả sau đây:



Cùng xem một chút chi tiết hơn về điều đã xảy ra tại đây. Đầu tiên **fscanf()** chỉ đọc **This** bởi vì sau đó nó gặp phải dấu cách, tiếp theo hàm **fgets()** trả về các dòng còn lại cho đến khi gặp ký tự cuối file. Cuối cùng nó gọi hàm **fgets()** để đọc hoàn toàn dòng thứ 2.

Hàm Nhập – Xuất nhị phân trong C

Dưới đây là hai hàm, có thể sử dụng cho việc input và output nhị phân:

```
size_t fread(void *ptr, size_t kich_co_cua_cac_phan_tu,
             size_t so_phan_tu, FILE *ten_file);


size_t fwrite(const void *ptr, size_t kich_co_cua_cac_phan_tu,
             size_t so_phan_tu, FILE *ten_file);
```

Cả 2 hàm trên được sử dụng để đọc và ghi các khối bộ nhớ, thường là các mảng hoặc cấu trúc.

4. Giải thuật sắp xếp nổi bọt (Bubble Sort)

Sắp xếp nổi bọt (Bubble Sort) là gì ?

Sắp xếp nổi bọt là một giải thuật sắp xếp đơn giản. Giải thuật sắp xếp này được tiến hành dựa trên việc so sánh cặp phần tử liền kề nhau và tráo đổi thứ tự nếu chúng không theo thứ tự.

Giải thuật này không thích hợp sử dụng với các tập dữ liệu lớn khi mà độ phức tạp trường hợp xấu nhất và trường hợp trung bình là $O(n^2)$ với n là số phần tử.

Giải thuật sắp xếp nổi bọt là giải thuật chậm nhất trong số các giải thuật sắp xếp cơ bản. Giải thuật này còn chậm hơn giải thuật đổi chỗ trực tiếp mặc dù số lần so sánh bằng nhau, nhưng do đổi chỗ hai phần tử kề nhau nên số lần đổi chỗ nhiều hơn.

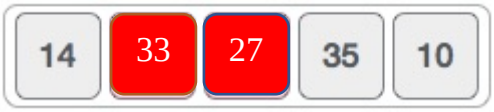
Cách giải thuật sắp xếp nổi bọt làm việc ?

Giả sử chúng ta có một mảng không có thứ tự gồm các phần tử như dưới đây. Bây giờ chúng ta sử dụng giải thuật sắp xếp nổi bọt để sắp xếp mảng này.

Giải thuật sắp xếp nổi bọt bắt đầu với hai phần tử đầu tiên, so sánh chúng để kiểm tra xem phần tử nào lớn hơn.

Trong trường hợp này, 33 lớn hơn 14, do đó hai phần tử này đã theo thứ tự. Tiếp đó chúng ta so sánh 33 và 27.

Chúng ta thấy rằng 33 lớn hơn 27, do đó hai giá trị này cần được tráo đổi thứ tự.



Mảng mới thu được sẽ như sau:

Tiếp đó chúng ta so sánh 33 và 35. Hai giá trị này đã theo thứ tự.

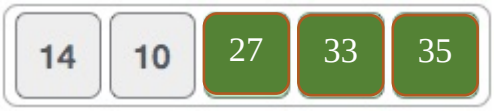
Sau đó chúng ta so sánh hai giá trị kế tiếp là 35 và 10.

Vì 10 nhỏ hơn 35 nên hai giá trị này chưa theo thứ tự.

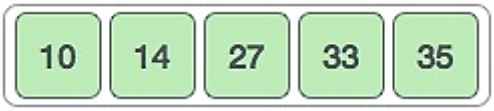
Tráo đổi thứ tự hai giá trị. Chúng ta đã tiến tới cuối mảng. Vậy là sau một vòng lặp, mảng sẽ trông như sau:

Để đơn giản, tiếp theo mình sẽ hiển thị hình ảnh của mảng sau từng vòng lặp. Sau lần lặp thứ hai, mảng sẽ trông giống như:

Sau mỗi vòng lặp, ít nhất một giá trị sẽ di chuyển tới vị trí cuối. Sau vòng lặp thứ 3, mảng sẽ trông giống như:



Và khi không cần tráo đổi thứ tự phần tử nào nữa, giải thuật sắp xếp nổi bọt thấy rằng mảng đã được sắp xếp xong.



Tiếp theo, chúng ta tìm hiểu thêm một số khía cạnh thực tế của giải thuật sắp xếp.

Giải thuật cho sắp xếp nổi bọt (Bubble Sort)

Giả sử list là một mảng có n phần tử. Tiếp đó giả sử hàm **swap** để tráo đổi giá trị của các phần tử trong mảng (đây là giả sử, tất nhiên là bạn có thể viết code riêng cho hàm swap này).

```
Bắt đầu giải thuật BubbleSort(list)

   for tất cả phần tử trong list
      if list[i] > list[i+1]
         swap(list[i], list[i+1])
      kết thúc if
   kết thúc for

   return list

Kết thúc BubbleSort
```

Giải thuật mẫu cho sắp xếp nổi bọt (Bubble Sort)

Chúng ta thấy rằng giải thuật sắp xếp nổi bọt so sánh mỗi cặp phần tử trong mảng trừ khi cả toàn bộ mảng đó đã hoàn toàn được sắp xếp theo thứ tự tăng dần. Điều này có thể làm tăng độ phức tạp, tức là tăng các thao tác so sánh và tráo đổi không cần thiết nếu như mảng này không cần sự tráo đổi nào nữa khi tất cả các phần tử đã được sắp xếp theo thứ tự tăng dần rồi.

Để tránh việc này xảy ra, chúng ta có thể sử dụng một biến **swapped** chẳng hạn để giúp chúng ta biết có cần thực hiện thao tác tráo đổi thứ tự hay không. Nếu không cần thiết thì thoát khỏi vòng lặp.

Bạn theo dõi phần giải thuật mẫu minh họa sau:

```
Bắt đầu hàm bubbleSort( list : mảng các phần tử )

   loop = list.count;

   for i = 0 tới loop-1 thực hiện:
      swapped = false

      for j = 0 tới loop-1 thực hiện:

         /* so sánh các phần tử cạnh nhau */
         if list[j] > list[j+1] then
            /* tráo đổi chúng */
            swap( list[j], list[j+1] )
            swapped = true
         kết thúc if

      kết thúc for

      /*Nếu không cần tráo đổi phần tử nào nữa thì
      tức là mảng đã được sắp xếp. Thoát khỏi vòng lặp.*/

      if(not swapped) then
         break
      kết thúc if

   kết thúc for
```

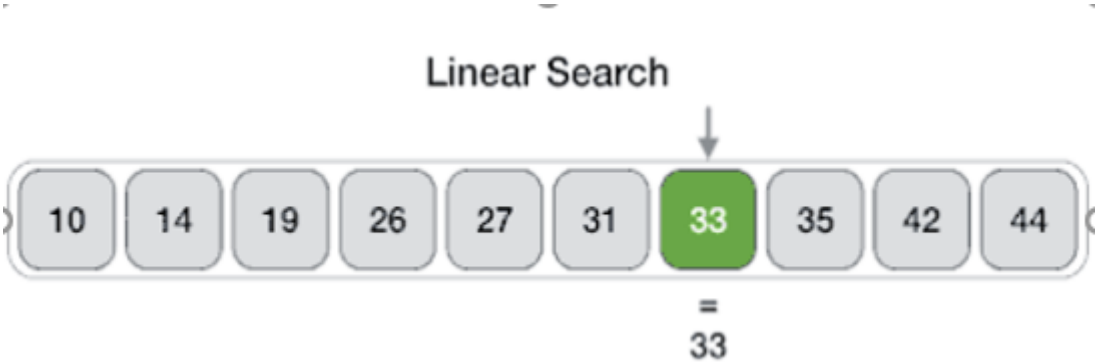
```
Kết thúc hàm return list
```

Triển khai giải thuật sắp xếp nổi bọt trong C

Một điều nữa mà chúng ta chưa nói tới trong 2 phần thiết kế giải thuật đó là cứ sau mỗi vòng lặp thì các giá trị lớn nhất sẽ xuất hiện ở vị trí cuối mảng (như trong hình minh họa: sau vòng lặp 1 là 35; sau vòng lặp 2 là 33 và 35; …). Do đó, vòng lặp tiếp theo sẽ không cần bao gồm cả các phần tử đã được sắp xếp này. Để thực hiện điều này, trong phần code chúng ta giới hạn vòng lặp lặp bên để tránh phải lặp lại các giá trị đã qua sắp xếp này.

5. Giải thuật tìm kiếm tuyến tính (Linear Search)

Linear Search là một giải thuật tìm kiếm rất cơ bản. Trong kiểu tìm kiếm này, một hoạt động tìm kiếm liên tiếp được diễn ra qua tất cả từng phần tử. Mỗi phần tử đều được kiểm tra và nếu tìm thấy bất kỳ kết nối nào thì phần tử cụ thể đó được trả về; nếu không tìm thấy thì quá trình tìm kiếm tiếp tục diễn ra cho tới khi tìm kiếm hết dữ liệu.



Giải thuật tìm kiếm tuyến tính

```
Giải thuật tìm kiếm tuyến tính ( Mảng A, Giá trị x)


Bước 1: Thiết lập i thành 1

Bước 2: Nếu i > n thì chuyển tới bước 7

Bước 3: Nếu A[i] = x thì chuyển tới bước 6

Bước 4: Thiết lập i thành i + 1

Bước 5: Tới bước 2

Bước 6: In phần tử x được tìm thấy tại chỉ mục i và tới bước 8

Bước 7: In phần tử không được tìm thấy

Bước 8: Thoát
```

Chương trình minh họa Tìm kiếm tuyến tính (Linear Search) trong C

```
#include <stdio.h>

#define MAX 20

// khai bao mang du lieu
int intArray[MAX] = {1,2,3,4,6,7,9,11,12,14,15,16,17,19,33,34,43,45,55,66};

void printline(int count){
   int i;

   for(i = 0;i <count-1;i++){
      printf("=");
   }

   printf("=\n");
}

// phuong thuc tim kiem tuyen tinh
int find(int data){

   int comparisons = 0;
   int index = -1;
   int i;

   // duyet qua tat ca phan tu
   for(i = 0;i<MAX;i++){

      // dem so phep tinh so sanh da thuc hien
      comparisons++;

      // neu tim thay du lieu, thoat khoi vong lap

      if(data == intArray[i]){
         index = i;
```

```
         break;
      }

   }

   printf("Tong so phep so sanh da thuc hien: %d", comparisons);
   return index;
}

void display(){
   int i;
   printf("[");

   // duyet qua tat ca phan tu
   for(i = 0;i<MAX;i++){
      printf("%d ",intArray[i]);
   }

   printf("]\n");
}

main(){

   printf("Mang du lieu dau vao: ");
   display();
   printline(50);

   //Tim kiem vi tri cua 55
   int location = find(55);

   // neu tim thay phan tu
   if(location != -1)
      printf("\nTim thay phan tu tai vi tri: %d" ,(location+1));
   else
      printf("Khong tim thay phan tu.");
}
```

Kết quả

Biên dịch và chạy chương trình C trên sẽ cho kết quả:

```
Mang du lieu dau vao: [1 2 3 4 6 7 9 11 12 14 15 16 17 19 33 34 43 45 55 66 ]
================================================
Tong so phep so sanh da thuc hien: 19
Tim thay phan tu tai vi tri: 19
--------------------------------
```

6. Cấu trúc dữ liệu danh sách liên kết (Linked List)

Danh sách liên kết (Linked List) là gì ?

Một Danh sách liên kết (Linked List) là một dãy các cấu trúc dữ liệu được kết nối với nhau thông qua các liên kết (link). Hiểu một cách đơn giản thì Danh sách liên kết là một cấu trúc dữ liệu bao gồm một nhóm các nút (node) tạo thành một chuỗi. Mỗi nút gồm dữ liệu ở nút đó và tham chiếu đến nút kế tiếp trong chuỗi.

Danh sách liên kết là cấu trúc dữ liệu được sử dụng phổ biến thứ hai sau mảng. Dưới đây là các khái niệm cơ bản liên quan tới Danh sách liên kết:

- **Link (liên kết)**: mỗi link của một Danh sách liên kết có thể lưu giữ một dữ liệu được gọi là một phần tử.
- **Next**: Mỗi liên kết của một Danh sách liên kết chứa một link tới next link được gọi là Next.
- **First**: một Danh sách liên kết bao gồm các link kết nối tới first link được gọi là First.

Biểu diễn Danh sách liên kết (Linked List)

Danh sách liên kết có thể được biểu diễn như là một chuỗi các nút (node). Mỗi nút sẽ trỏ tới nút kế tiếp.

Dưới đây là một số điểm cần nhớ về Danh sách liên kết:

- Danh sách liên kết chứa một phần tử link thì được gọi là First.

- Mỗi link mang một trường dữ liệu và một trường link được gọi là Next.
- Mỗi link được liên kết với link kế tiếp bởi sử dụng link kế tiếp của nó.
- Link cuối cùng mang một link là null để đánh dấu điểm cuối của danh sách.

Các loại Danh sách liên kết (Linked List)

Dưới đây là các loại Danh sách liên kết (Linked List) đa dạng:

- **Danh sách liên kết đơn (Simple Linked List)**: chỉ duyệt các phần tử theo chiều về trước.
- **Danh sách liên kết đôi (Doubly Linked List)**: các phần tử có thể được duyệt theo chiều về trước hoặc về sau.
- **Danh sách liên kết vòng (Circular Linked List)**: phần tử cuối cùng chứa link của phần tử đầu tiên như là next và phần tử đầu tiên có link tới phần tử cuối cùng như là prev.

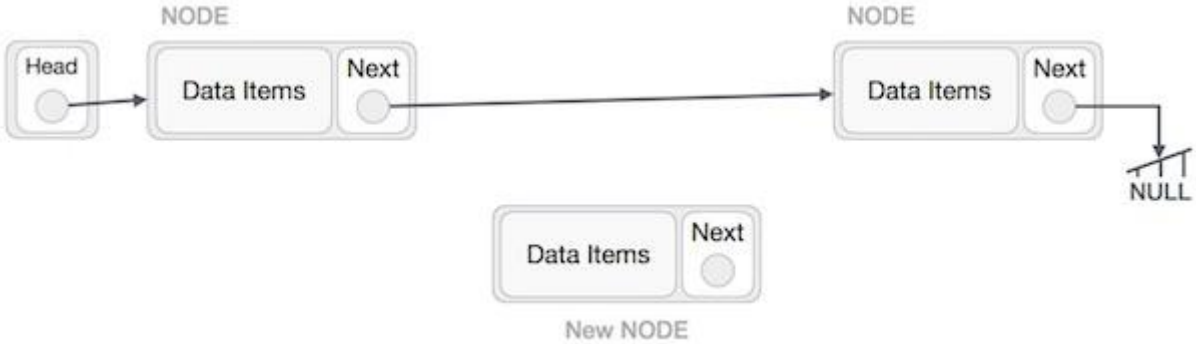
Các hoạt động cơ bản trên Danh sách liên kết

Dưới đây là một số hoạt động cơ bản có thể được thực hiện bởi một danh sách liên kết:

- **Hoạt động chèn**: thêm một phần tử vào đầu danh sách liên kết.
- **Hoạt động xóa (phần tử đầu)**: xóa một phần tử tại đầu danh sách liên kết.
- **Hiển thị**: hiển thị toàn bộ danh sách.
- **Hoạt động tìm kiếm**: tìm kiếm phần tử bởi sử dụng khóa (key) đã cung cấp.
- **Hoạt động xóa (bởi sử dụng khóa)**: xóa một phần tử bởi sử dụng khóa (key) đã cung cấp.

Hoạt động chèn trong Danh sách liên kết

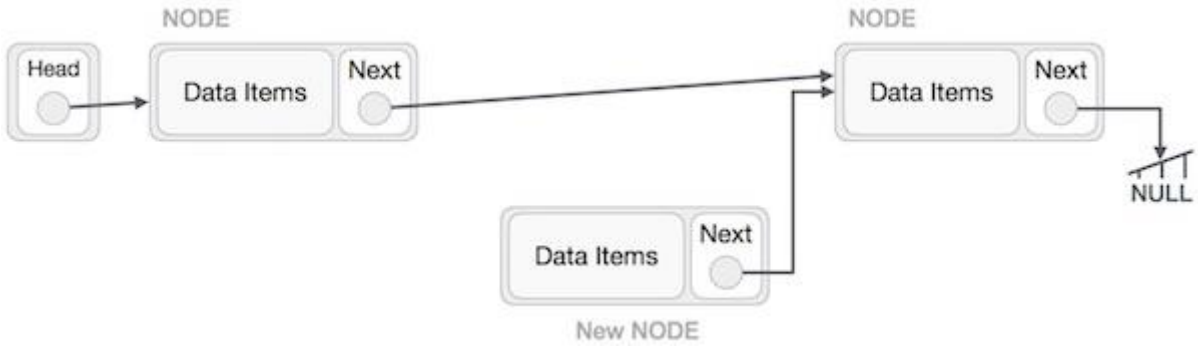
Việc thêm một nút mới vào trong danh sách liên kết không chỉ là một hoạt động thêm đơn giản như trong các cấu trúc dữ liệu khác (bởi vì chúng ta có dữ liệu và có link). Chúng ta sẽ tìm hiểu thông qua sơ đồ dưới đây. Đầu tiên, tạo một nút bởi sử dụng cùng cấu trúc và tìm vị trí để chèn nút này.



Giả sử chúng ta cần chèn một nút B vào giữa nút A (nút trái) và C (nút phải). Do đó: B.next trỏ tới C.

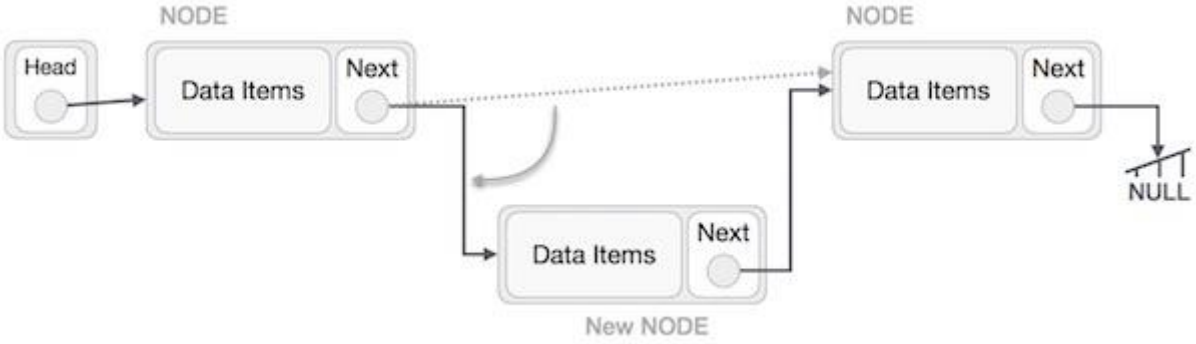
```
NewNode.next –> RightNode;
```

Hình minh họa như sau:



Bây giờ, next của nút bên trái sẽ trở tới nút mới.

```
LeftNode.next –> NewNode;
```



Quá trình trên sẽ đặt nút mới vào giữa hai nút. Khi đó danh sách mới sẽ trông như sau:

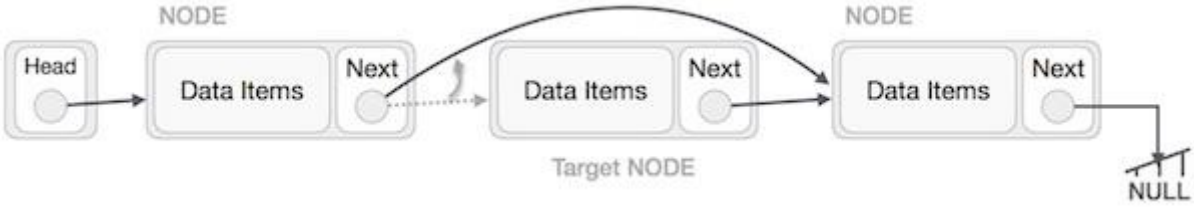
Các bước tương tự sẽ được thực hiện nếu chèn nút vào đầu danh sách liên kết. Trong khi đặt một nút vào vị trí cuối của danh sách, thìnút thứ hai tính từ nút cuối cùng của danh sách sẽ trỏ tới nút mới và nút mới sẽ trỏ tới NULL..

Hoạt động xóa trong Danh sách liên kết

Hoạt động xóa trong Danh sách liên kết cũng phức tạp hơn trong cấu trúc dữ liệu khác. Đầu tiên chúng ta cần định vị nút cần xóa bởi sử dụng các giải thuật tìm kiếm.

Bây giờ, nút bên trái (prev) của nút cần xóa nên trỏ tới nút kế tiếp (next) của nút cần xóa.

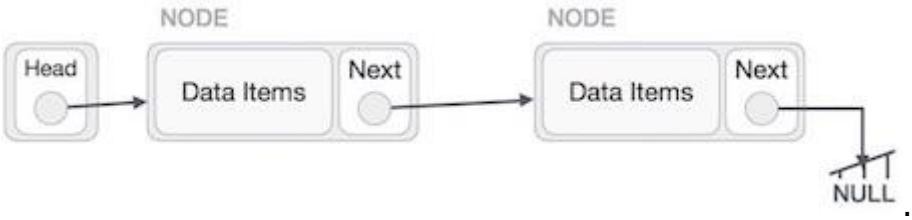
```
LeftNode.next -> TargetNode.next;
```



Quá trình này sẽ xóa link trỏ tới nút cần xóa. Bây giờ chúng ta sẽ xóa những gì mà nút cần xóa đang trỏ tới.

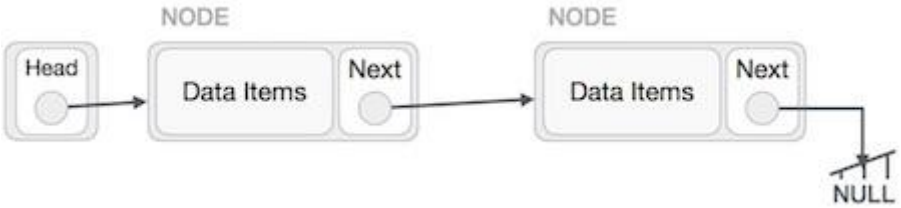
```
TargetNode.next -> NULL;
```

Nếu bạn cần sử dụng nút đã bị xóa này thì bạn có thể giữ chúng trong bộ nhớ, nếu không bạn có thể xóa hoàn toàn hẳn nó khỏi bộ nhớ.

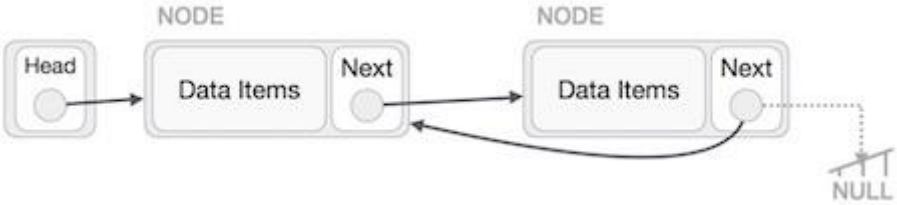


Hoạt động đảo ngược Danh sách liên kết

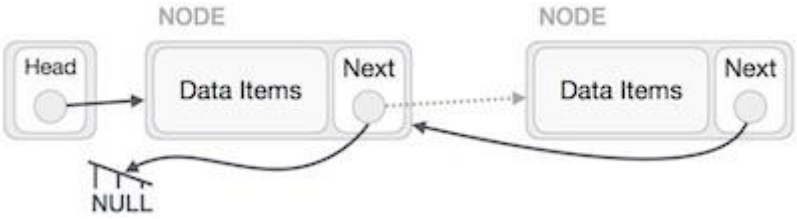
Với hoạt động này, bạn cần phải cẩn thận. Chúng ta cần làm cho nút đầu (head) trỏ tới nút cuối cùng và đảo ngược toàn bộ danh sách liên kết.



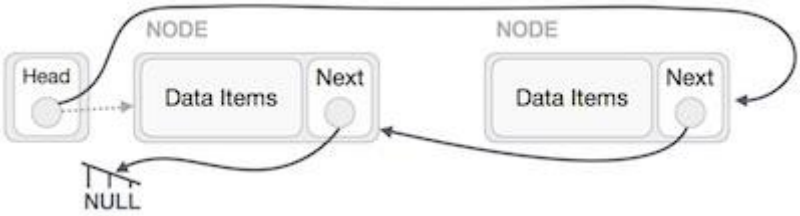
Đầu tiên, chúng ta duyệt tới phần cuối của danh sách. Nút này sẽ trỏ tới NULL. Bây giờ điều cần làm là làm cho nút cuối này trỏ tới nút phía trước của nó.



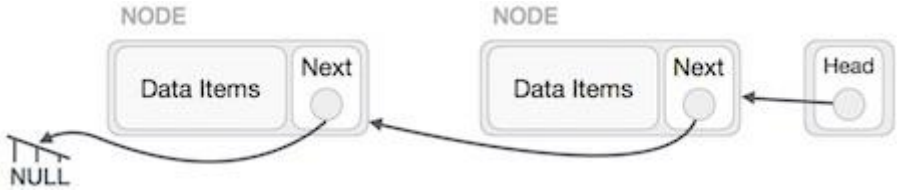
Chúng ta phải đảm bảo rằng nút cuối cùng này sẽ không bị thất lạc, do đó chúng ta sẽ sử dụng một số nút tạm (temp node – giống như các biến tạm trung gian để lưu giữ giá trị). Tiếp theo, chúng ta sẽ làm cho từng nút bên trái sẽ trỏ tới nút trái của chúng.



Sau đó, nút đầu tiên sau nút head sẽ trỏ tới NULL.



Chúng ta sẽ làm cho nút head trỏ tới nút đầu tiên mới bởi sử dụng các nút tạm.



Bây giờ Danh sách liên kết đã bị đảo ngược.

CHƯƠNG III: PHÂN TÍCH VÀ XÂY DỰNG ĐỀ TÀI

1. Ý tưởng làm đề tài

Muốn làm phần mềm quản lý thư viện, trước tiên chúng ta phải sử dụng cấu trúc struct để lưu trữ một đối tượng có nhiều thuộc tính.

Sau đó chúng ta sử dụng cấu trúc lặp: for, do while, while, để thực hiện các câu lệnh tiếp theo. Ngoài ra chúng ta phải sử dụng cấu trúc điều khiển và rẽ nhánh: if else, switch case.

Cuối cùng chúng ta nhập xuất file để lưu trữ dữ liệu, dễ dàng sao chép, di chuyển dữ liệu giữa các thiết bị với nhau.

2. Giải thích code

- **Chức năng sắp xếp sách theo thể loại từ Z - A**

```
void arrangeBook(book_st *output, int numberBooks)
{
    //sap xep sach
    book_st temp;
    for (int i = 0; i < numberBooks - 1; i++)
    {
        for (int j = i + 1; j < numberBooks; j++)
        {
            if (strcmp((output + i)->type, (output + j)->type) < 0)
            {
                temp = *(output + i);
                *(output + i) = *(output + j);
                *(output + j) = temp;
            }
        }
    }
}
```

Biến temp kiểu struct (book_st) dùng làm biến trung gian để sắp xếp thông tin các quyển sách theo thể loại. Hàm

`if (strcmp((output + i)->type, (output + j)->type) < 0)` dùng để so sánh chuỗi, nếu thể loại i < thể loại j thì thông tin của quyển sách j sẽ được sắp xếp ở trước thông tin của quyển sách i. Tương tự các cuốn sau.

- **Chức năng đếm sách theo thể loại**

```
int countBooksByType(book_st *input, int numberBooks, char search[30])
{
    //dem sach theo the loai
    int count = 0;
    for (int i = 0; i < numberBooks; i++)
    {
        if (strcmp((input + i)->type, search) == 0)
        {
```

```
            count++; //neu cung loai thì dem
        }
    }
    return count;
}
void printTypeBooks(book_st *output, int numberBooks)
{
    //in so the loai
    if (numberBooks != 0)
    {
        printf("\nTruyen tranh co %d quyen sach",
countBooksByType(output, numberBooks, "truyen tranh"));
        printf("\nTai lieu co %d quyen sach",
countBooksByType(output, numberBooks, " tai lieu"));
        printf("\nGiao trinh co %d quyen sach\n",
countBooksByType(output, numberBooks, "giao trinh"));
    }
}
```

Hàm countBookByType(…) dùng để trả về 1 số khi đếm theo thể loại sách. Điều kiện `if (strcmp((input + i)->type, search) == 0)` trong vòng lặp là so sánh thể loại sách ban đầu với biến search là các chuỗi được ghi phía dưới, nếu đúng thì count sẽ tăng một đơn vị. Và hàm printTypeBooks sẽ in ra count theo từng thể loại, ở đây có 3 thể loại chính của thư viện đó là: truyện tranh, tài liệu và giáo trình. 3 chuỗi này chính là biến search để so sánh ở hàm trên.

- **Chức năng tìm kiếm sách theo thể loại**

```
void addBook(book_st *input, int &numberBooks, const book_st book)
{
    //them sach
    numberBooks++;
    input = (book_st *)realloc(input, numberBooks * sizeof(book_st));
    *(input + numberBooks - 1) = book;
}
void enterType(char search[30])
{
    printf("\nNhap the loai: ");
    fflush(stdin);
    gets(search);
}
book_st *findBookByType(book_st *input, int numberBooks, int totalBooks, ch
ar search[30])
{
    //tim sach theo the loai
    book_st *result;
    book_st temp;
    totalBooks = 0;
```

```
    result = (book_st *)calloc(totalBooks, sizeof(book_st));
    for (int index = 0; index < numberBooks; index++)
    {
        if (strcmp((input + index)->type, search) == 0)
        {
            temp = *(input + index);
            addBook(result, totalBooks, temp);
        }
    }
    return result;
}
```

. Đầu tiên là hàm enterType(…), người dùng sẽ nhập thể loại cần tìm kiếm.

. Hàm findBookByType(…) sẽ trả về một mảng động để in ra màn hình ở hàm print. Như vậy sẽ phải cấp phát bộ nhớ cho result để sử dụng. Điều kiện ở vòng lặp cũng tương tự như hàm đếm sách theo thể loại nhưng thay thế bằng biến search là người dùng nhập vào. Lúc này temp sẽ được gán bằng thông tin i, hàm addBook sẽ thực hiện.

. Hàm addBook truyền vào result = input, totalBooks = numberBooks và temp = book. numberBooks tăng 1 đơn vị và sẽ cấp phát cho input theo numberBooks. Cuối cùng trả về phần tử đầu tiên chính là temp lúc đầu. Tiếp tục vòng lặp cho các phần tử thứ 2, thứ 3 … Nếu hết thì kết thúc vòng lặp và trả về result. Và hàm print sẽ in ra màn hình các thông tin của quyển sách theo thể loại của người nhập.

- **Chức năng xóa sách**

```
void removeBook(book_st *output, int &id_need_to_find, int &numberBooks)
{
    //ham xoa sach
    bool has_book = false;
    printf("Nhap id quyen sach: ");
    scanf("%d", &id_need_to_find);
    for (int index = 0; index < numberBooks; index++)
    {
        if ((output + index)->id == id_need_to_find)
        {
            for(int j = index; j < numberBooks - 1; j++){
                *(output + j) = *(output + j + 1);
                has_book = true;
            }
            numberBooks--;
        }
    }
    if (has_book)
    {
```

```
        printf("Xoa sach thanh cong !\n");
    }
    else
    {
        printf("Khong co quyen sach nay !\n");
    }
}
```

. Biến has_book để kiểm tra xem có tồn tại id quyển sách đó không.

. Ở vòng lặp, nếu id của sách trùng với id người dùng nhập vào thì sẽ dịch vị trí qua bên phải 1 đơn vị và số quyển sách cũng giảm 1 đơn vị, có nghĩa là bỏ qua quyển sách hiện tại = xóa sách đó. Và biến has_book theo kiểu bool sẽ trả về Đúng, in ra Xóa sách thành công. Nếu id nhập vào không tồn tại thì biến has_book sẽ Sai và in ra Không có quyển sách này.

- **Chức năng sửa sách**

```
void editBook(book_st *output, int &id_need_to_find, int numberBooks)
{
    //ham sua sach
    int has_book = false;
    printf("Nhap id quyen sach: ");
    scanf("%d", &id_need_to_find);
    for (int index = 0; index < numberBooks; index++)
    {
        if (id_need_to_find == (output + index)->id)
        {
            printf("\n======== Sua thong tin sach ========");
            fflush(stdin);
            printf("\nTen: ");
            gets((output + index)->name);
            printf("******** Nhap thong tin tac gia ********");
            (output + index)->author = (Author *)malloc(sizeof(Author));
            enter((output + index)->author);
            fflush(stdin);
            printf("\nThe loai: ");
            gets((output + index)->type);
            printf("Gia tien: ");
            scanf("%d", &(output + index)->price);
            has_book = true;
        }
    }
    if (has_book)
    {
        printf("Sua sach thanh cong!\n");
    }
    else
```

```
    {
        printf("Khong co quyen sach nay !\n");
    }
}
```

Tương tự như hàm xóa sách nhưng trong điều kiện sẽ là nhập lại thông tin của quyển sách đó.

- **Chức năng xuất ra file nhị phân**

```
void exportBook(FILE *file, char *path, book_st *output, int numberBooks)
{
    //xuat sach ra file
    char *mode = "ab";
    file = fopen(path, mode);
    fprintf(file, "\nId ||Ten\t\t\t\t\t||Tac gia\t\t\t\t\t\t\t\t||The loai\
t\t\t\t||Gia tien");
    for (int index = 0; index < numberBooks; index++)
    {
        fprintf(file, "\n%-3d||%-20s\t||%-24s(%d/ %d/ %d) ||%-
17s\t\t||%d", (output + index)->id, (output + index)-
>name, (output + index)->author->name, (output + index)->author->birthday-
>day, (output + index)->author->birthday->month, (output + index)->author-
>birthday->year, (output + index)->type, (output + index)->price);
    }
    fclose(file);
}
```

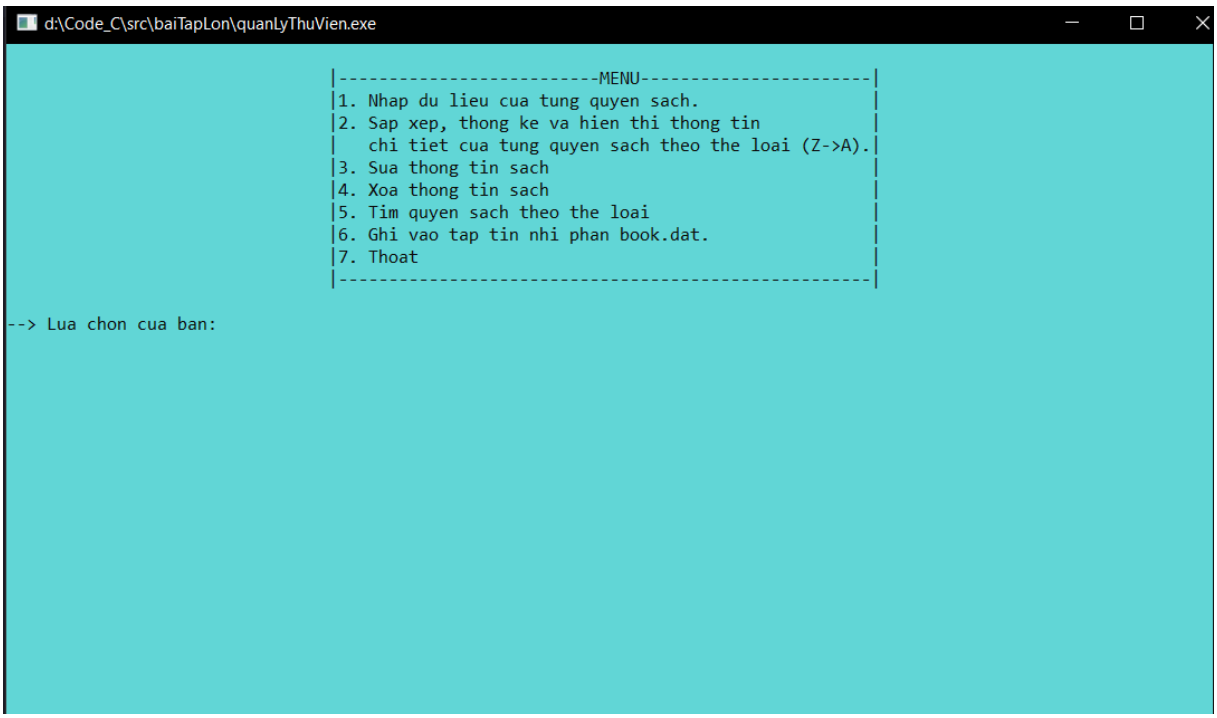
Ở đây chọn mode “ab” có nghĩa là mở file ở chế độ ghi nhị phân, “append” là sẽ ghi vào cuối của nội dung đã có. Nếu file không tồn tại, nó sẽ được tạo tự động.

. Để xuất ra file thì đầu tiên phải mở file trước bằng hàm fopen(), sau đó ghi vào file và đóng file bằng hàm fclose().

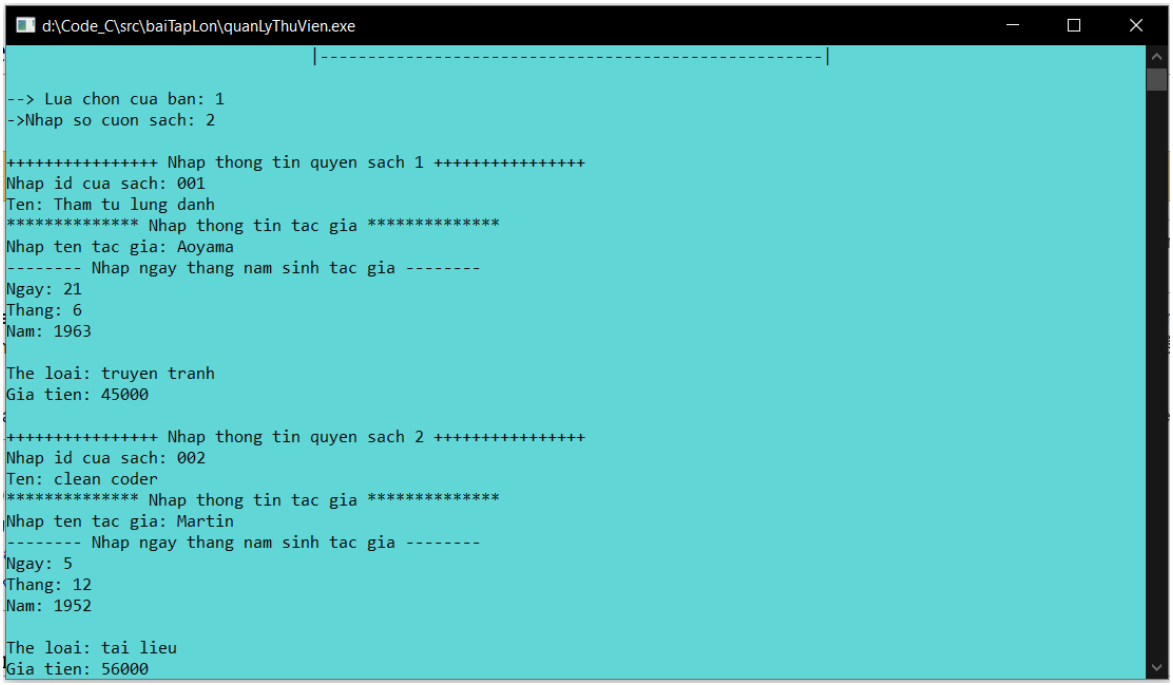
3. Hướng dẫn sử dụng

Xin chào, sau đây mình mình xin hướng dẫn mọi người cách sử dụng chương trình của nhóm chúng mình.

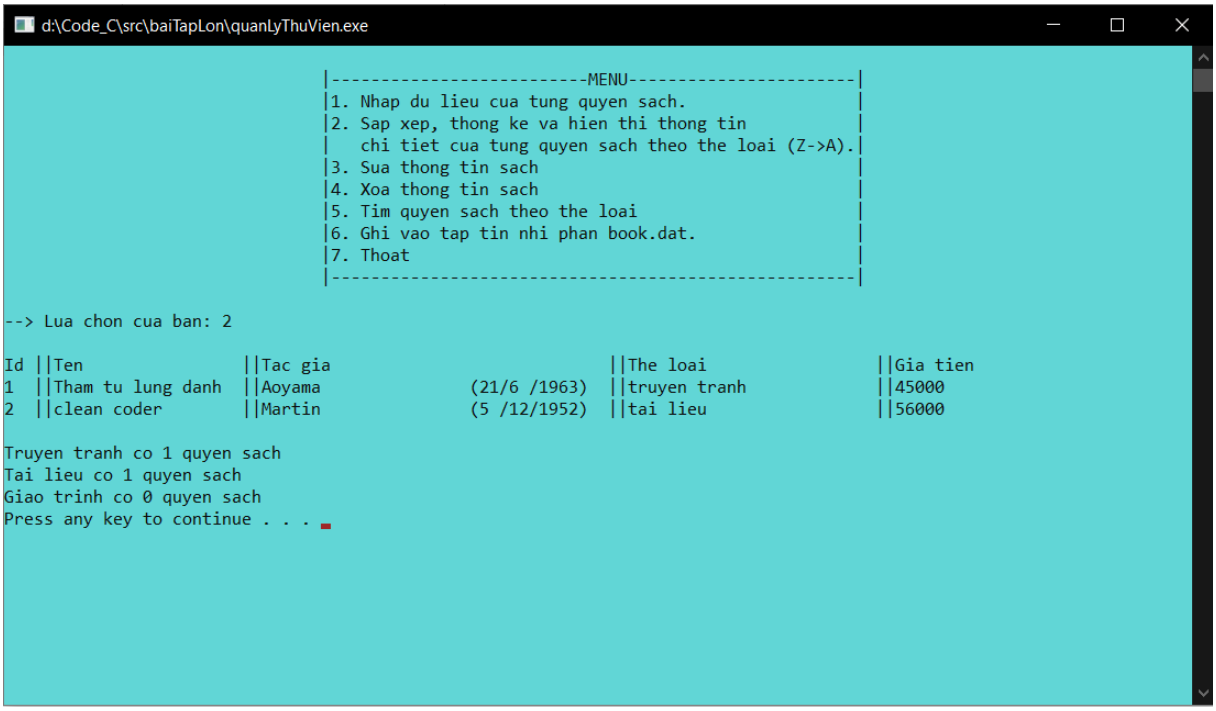
- Đầu tiên màn hình sẽ hiện ra các lựa chọn sau



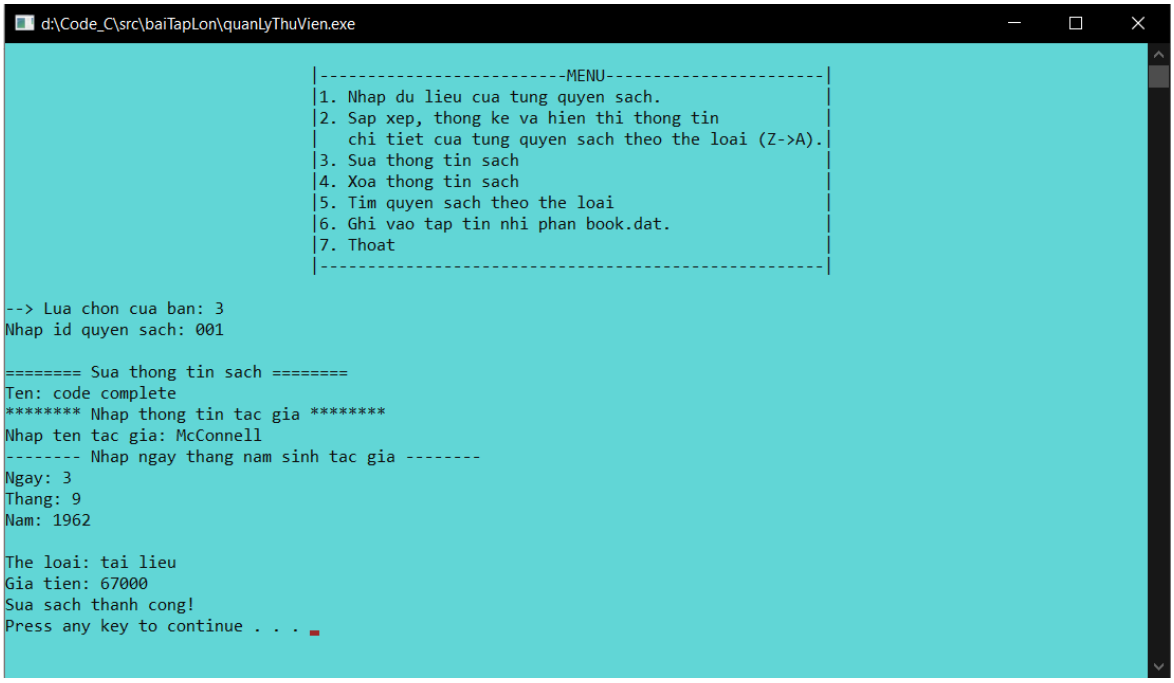
- Trước khi chọn các sự lựa chọn phía dưới thì việc đầu tiên ta cần làm là nhập dữ liệu trước, nếu không có dữ liệu các lựa chọn phía dưới sẽ không chạy và nó sẽ yêu cầu bạn nhập danh sách sinh viên trước.
- Vì vậy chúng ta sẽ bắt buộc phải chọn lựa chọn thứ nhất để thực hiện việc quản lý thư viện.
- Đầu tiên nhập số lượng sách, sau đó lần lượt nhập thông tin của sách bao gồm id sách, tên sách, tên tác giả, ngày tháng năm sinh tác giả, thể loại sách, giá tiền.



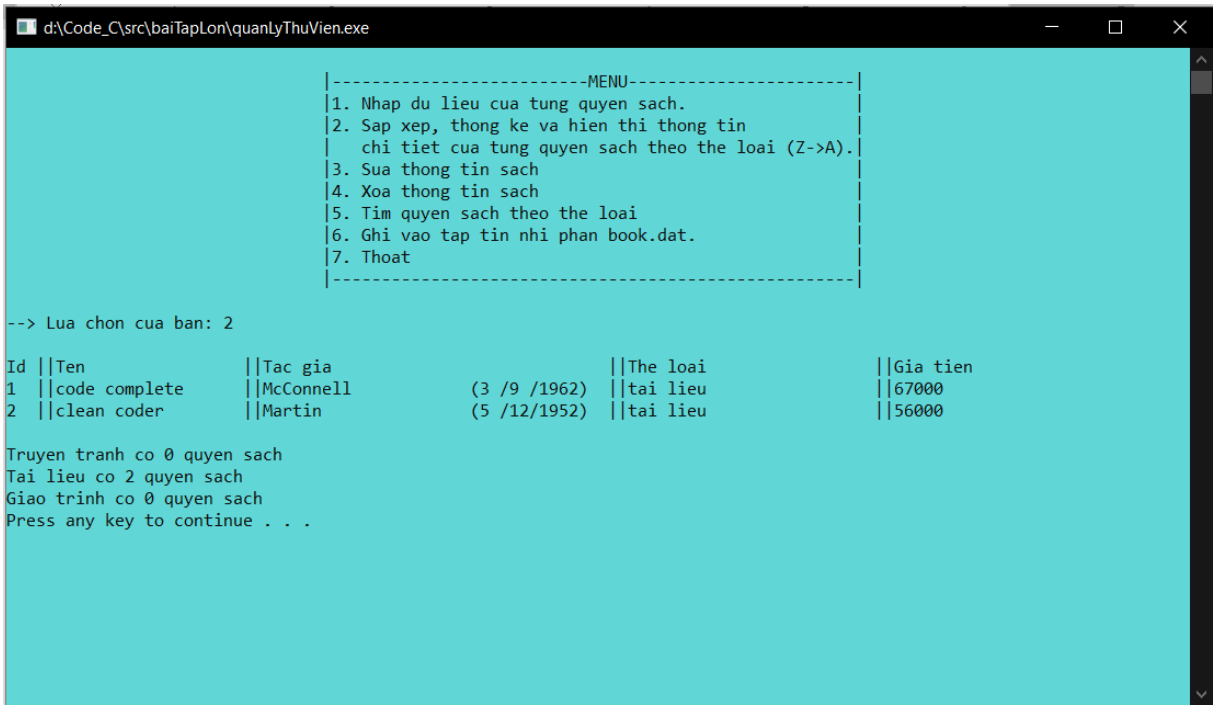
- Sau khi nhập hoàn tất, nếu bạn chọn lựa chọn thứ hai thì chương trình sẽ xuất danh sách bạn vừa nhập ra màn hình, như hình dưới:



- Tiếp theo nếu bạn muốn chỉnh sửa thông tin của một quyển sách, bạn chọn 3. Bạn sẽ nhập id sách cần sửa, nếu nhập sai màn hình sẽ hiện ra “Không có quyển sách này”, nếu đúng bạn sẽ nhập lại thông tin của quyển sách cần sửa và khi xong màn hình sẽ hiện ra “Sửa sách thành công”



+ Sau đó bạn chọn lại số 2 để kiểm tra sách đã được sửa

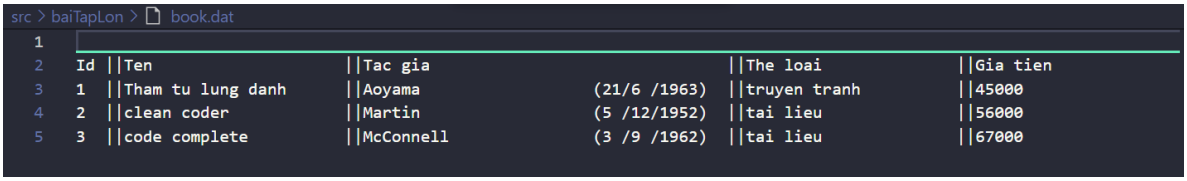


- Nếu bạn muốn xóa sách, bạn hãy chọn số 4. Tương tự như sửa sách, bạn sẽ nhập id của quyển sách bạn muốn xóa, sau đó màn hình sẽ hiện ra “Xóa sách thành công” hoặc “Không có quyển sách này” nếu bạn nhập id sai.

+ Sau đó chọn 2 để kiểm tra

- Bạn muốn tìm kiếm các quyển sách theo một thể loại thì bạn hãy chọn số 5. Bạn sẽ nhập thể loại cần tìm, sau đó màn hình sẽ hiện ra các quyển sách có thể loại bạn muốn tìm.

- Nếu bạn muốn xuất thông tin các quyển sách ra tệp nhị phân, bạn chọn số 6. Thông tin sẽ được xuất ra file book.dat.



- Cuối cùng bạn muốn thoát khỏi chương trình, bạn chọn số 7, bạn có thể chọn nó bất kì lúc nào.

CHƯƠNG IV: KẾT LUẬN

1. Kết quả đạt được

- Nhóm chúng em đã hoàn thành xong chương trình quản lý thư viện với các chức năng cơ bản khác nhau gồm: In danh sách các quyển sách ra file "book.dat", sắp xếp sách theo thể loại từ Z - A, tìm kiếm sách theo thể loại, chỉnh sửa thông tin sách…

2. Nhược điểm

- Chưa vận dụng được danh sách liên kết đơn vào bài tập lớn.
- Không có nhiều ý tưởng về các chức năng mới mẻ cho chương trình quản lý thư viện.

3. Hướng phát triển

- Nhóm chúng em sẽ cố gắng học hỏi và phát triển chương trình quản lý thư viện với nhiều chức năng hơn, ứng dụng vào thực tế nhiều hơn, thoả mãn nhu cầu của người sử dụng trong xã hội.

4. Tài liệu tham khảo

- Daynhayhoc.com, Codelearn.io, laptrinhkhongkho.com, topdev.vn, …

5. Bảng phân công công việc mỗi thành viên

Thành Viên	Nhiệm Vụ	Ghi chú
Nguyễn Hoàng Phát (Nhóm Trưởng) (MSV: 6151071082)	-Code(50%) -Làm báo cáo(43%) -Thuyết trình -PowerPoint	-Code: Hàm nhập ngày tháng, hàm nhập nhiều quyển sách, hàm đếm sách theo thể loại, hàm tìm kiếm sách theo thể loại, hàm xuất sách, hàm sắp xếp sách từ Z-A, hàm xuất thông tin sách ra file. -Bài báo cáo: Phân tích và xây dựng đề tài. -Chuẩn bị slide và thuyết trình
Trần Trọng Nhân (MSV: 6151071016)	-Code(40%) -Làm báo cáo(52%)	-Code: Hàm nhập tác giả, hàm nhập một quyển sách, hàm sửa sách, hàm xóa sách, hàm menu. -Bài báo cáo: Cơ sở lý thuyết.
Lê Tô Nguyện (MSV: 6151071015)	-Code(10%) -Làm báo cáo(5%)	-Code: Khai báo struct, hàm kiểm tra ngày tháng. -Bài báo cáo: Lời mở đầu

Link mã nguồn bài tập:

https://github.com/PhatNguyen-K61/BaiTapLon/blob/main/quanLyThuVien.cpp