**TRƯỜNG ĐẠI HỌC GIAO THÔNG VẬN TẢI**

**PHÂN HIỆU TẠI TP. HỒ CHÍ MINH**

**BỘ MÔN CÔNG NGHỆ THÔNG TIN**



**BÁO CÁO MÔN HỌC**

**LẬP TRÌNH NÂNG CAO**

**ĐỀ TÀI: XÂY DỰNG PHẦN MỀM NHẬP XUẤT THÀNH PHỐ THEO YÊU CẦU**

Sinh viên thực hiện:

LÊ MINH TUẦN – MSV: 6051071136

TRƯƠNG CÔNG THẠCH – MSV: 6051071105

TRẦN HOÀNG NAM – MSV: 605107174

Lớp : CÔNG NGHỆ THÔNG TIN K60

Khoá :K60

Tp. Hồ Chí Minh, năm 2020

**TRƯỜNG ĐẠI HỌC GIAO THÔNG VẬN TẢI**

**PHÂN HIỆU TẠI TP. HỒ CHÍ MINH**

**BỘ MÔN CÔNG NGHỆ THÔNG TIN**



**BÁO CÁO MÔN HỌC**

**LẬP TRÌNH NÂNG CAO**

**ĐỀ TÀI: XÂY DỰNG PHẦN MỀM QUẢN LÍ SINH VIÊN**

Sinh viên thực hiện:

LÊ MINH TUẦN – MSV: 6051071136

TRƯƠNG CÔNG THẠCH – MSV: 6051071105

TRẦN HOÀNG NAM – MSV: 605107174

Lớp : CÔNG NGHỆ THÔNG TIN K60

Khoá :K60

Tp. Hồ Chí Minh, năm 2020

|  |
| --- |
|  |

**LỜI NÓI ĐẦU**

Ngày nay, khoa học công nghệ không ngừng phát triển, cả thế giới đang dần bước vào thời đại công nghệ mới- thời đại công nghệ 4.0. Việc áp dụng các nghệ hiện đại vào các lĩnh vực trong đời sống sản xuất cũng như sinh hoạt còn còn xa lạ với chúng ta nữa. Chính vì thế, cùng với sự phát triển đó, nhóm chúng em đã xây dựng nên phần mềm nhập và xuất dữ liệu thông tin của các thành phố trên lãnh thổ. Từ đó, chúng ta có thể dễ dàng tìm kiếm, sắp xếp, thống kê và hiển thị thông tin của từng thành phố một cách nhanh nhất, chính xác nhất. Và đem lại nhiều lợi ích cho tất cả người dân trong nước cũng như với những nước bạn. Vì nó sẽ đem lại cái nhìn tổng thể về một thành phố nào đó.

Nội dung báo cáo gồm những chương sau:

CHƯƠNG 1: Tổng quan về đề tài

CHƯƠNG 2: Bài toán về nhập xuất thành phố theo yêu cầu

CHƯƠNG 3: Lý thuyết áp dụng

Những lợi ích từ việc sử dụng

Trong quá trình làm bài tập lớn và báo cáo sẽ không tránh khỏi những sai sót vì đây là lần đầu tiên nên mong cô có thể góp ý và chia sẻ cho nhóm chúng em để rút kinh nghiệm lần sau.

Xin chân thành cảm ơn cô !

**M** **ỤC L** **ỤC**

CHƯƠNG I. TỔNG QUAN VỀ ĐỀ TÀI 7

[Lý do chọn đề tài 7](#_Toc43935666)

[Mục tiêu nghiên cứu: 7](#_Toc43935667)

[Phạm vi nghiên cứu: 7](#_Toc43935668)

[Khảo sát thực tế: 7](#_Toc43935669)

[Cách tiếp cận, phương pháp nguyên cứu: 7](#_Toc43935670)

[Cấu trúc báo cáo bài tập tập chuyên môn: 7](#_Toc43935671)

[CHƯƠNG II. BÀI TOÁN VỀ NHẬP XUẤT THÀNH PHỐ THEO YÊU CẦU 8](#_Toc43935672)

[Giới thiệu bài toán: 8](#_Toc43935673)

[Mô tả và cài đặt giải thuật cho bài toán 8](#_Toc43935674)

[Hướng dẫn sử dụng 8](#_Toc43935675)

[CHƯƠNG III. LÝ THUYẾT ỨNG DỤNG 11](#_Toc43935676)

[1. Làm việc với tệp 11](#_Toc43935677)

[Các tham số của “mode” 11](#_Toc43935678)

[1.2 Thao tác đóng file: 13](#_Toc43935679)

[Ví dụ sử dụng fprintf() 13](#_Toc43935680)

[1.4 Đọc file sử dụng fscanf() 13](#_Toc43935681)

[1.5 Đọc/Ghi file nhị phân: 14](#_Toc43935682)

[Ví dụ đọc file nhị phân sử dụng fread() 15](#_Toc43935683)

[2. Danh sách liên kết 16](#_Toc43935684)

[2.1 Lý thuyết về danh sác liên kết : 16](#_Toc43935685)

[2.2 Danh sách liên kết là gì? 16](#_Toc43935686)

[2.3 Cài đặt danh sách liên kết đơn: 17](#_Toc43935687)

[*Khai báo và khởi tạo liên kết đơn:* 17](#_Toc43935688)

[**2.3.1 Khai báo linked list** 17](#_Toc43935689)

[**2.3.2 Tạo mới 1 Node** 18](#_Toc43935691)

[**2.3.3 Thêm Node vào danh sách liên kết** 18](#_Toc43935692)

[**2.3.4. Xóa Node khỏi danh sách liên kết** 21](#_Toc43935694)

[**2.3.5 Lấy giá trị ở vị trí bất kỳ** 23](#_Toc43935695)

[**2.3.6 Tìm kiếm trong danh sách liên kết** 23](#_Toc43935696)

[**2.3.7 Duyệt danh sách liên kết** 24](#_Toc43935698)

[**2.3.8 Một số hàm bổ trợ khác** 24](#_Toc43935699)

[3. Cấu trúc dữ liệu struct 31](#_Toc43935700)

[3.1 Cách định nghĩa struct 31](#_Toc43935701)

[3.2 Cú pháp định nghĩa struct 31](#_Toc43935702)

[3.3 Cách khai báo biến kiểu struct 31](#_Toc43935703)

[**3.4 Truy xuất các thuộc tính của struct** 32](#_Toc43935704)

[**3.6 Cấu trúc struct lồng nhau:** 33](#_Toc43935705)

[**3.7 Struct và con trỏ** 33](#_Toc43935706)

[Áp dụng vô bài tập lớn: 34](#_Toc43935707)

[3.8 Địa chỉ của biến 35](#_Toc43935708)

[4. Con trỏ trong C 36](#_Toc43935710)

[4.2 Cách khai báo con trỏ 36](#_Toc43935711)

[4.3 Gán giá trị cho con trỏ 37](#_Toc43935712)

[**4.4 Bản chất của con trỏ trong C** 38](#_Toc43935713)

[**4.5 Các lỗi thường gặp khi làm việc với con trỏ** 40](#_Toc43935715)

[Áp dụng vô bài tập lớn: 41](#_Toc43935716)

[5. Các giải thuật tìm kiếm , sắp xếp : 41](#_Toc43935717)

[5.1 Bài toán Tìm kiếm: 41](#_Toc43935717)

**[5.2 Bài toán Sắp xếp :](#_Toc43935718)** [42](#_Toc43935718)

[Tổng kết: 43](#_Toc43935719)

[1. Kết quả đạt được: 43](#_Toc43935720)

[2. Mục tiêu thực hiện đề tài này: 43](#_Toc43935721)

[Tài liệu tham khảo 43](#_Toc43935722)

**CHƯƠNG I. TỔNG QUAN VỀ ĐỀ TÀI**

## **Lý do chọn đề tài**

Trong xã hội ngày càng phát triển hiện nay, khoa học công nghệ là thứ không thể thiếu đối với mỗi quốc gia, doanh nghiệp, trường học hay mỗi cá nhân, đặc biệt là các phần mềm ứng dụng để quản lý một dữ liệu nào đó. Với sự phát triển nhanh một cách không ngừng nghĩ như vậy của công nghệ thông tin đã giúp giải quyết các công việc học tập, nguyên cứu, quản lý thông tin,… một cách dễ dàng và tiện lợi. Thấy được tiềm năng đó các quốc gia, doanh nghiệp, trường học, các cá nhân, … đã ứng dụng nền công nghệ tiên tiến của mình vào các hoạt động thuộc lĩnh vực kinh tế - xã hội, đối ngoại, quốc phòng, an ninh và các hoạt động khác nhằm nâng cao năng suất, chất lượng, hiệu quả của các hoạt động này..

Trong những năm gần đây nhu cầu về các phần mềm, ứng dụng học tập, giải trí càng nhiều hơn do nhu cầu sử dụng điện thoại các thiết bị gọn nhẹ ngày càng cao và thuận tiện cho người sử dụng. Các thiết bị thông minh đã và đang được sử dụng rộng rãi trên toàn thế giới. Giúp cho mọi người dễ dàng kết nối, liên lạc, đến gần nhau hơn.

Đặc biệt trong học tập và giải trí, sự tiện lợi của những phần mềm, ứng dụng trên máy tính hay điện thoại lại chứng minh ưu thế của mình. Người sử dụng có thể học tập và giải trí ở bất kì đâu mà họ muốn chỉ thông qua một chiếc máy tính có kết nối với Internet. Đã có rất nhiều ứng dụng đã ra đời và đem lại nhiều lợi ích cho người sử dụng, ta không cần phải đến thư viện để đọc sách hoặc tốn nhiều thời gian cho việc tìm kiếm thông tin, hoặc là không cần đến trường, trung tâm để học chỉ cần cài được ứng dụng bạn có thể gói gọn mọi thứ trong chiếc máy tính của bạn. Và ngay cả việc theo dõi tên, cũng như các thành phố mà bạn cần nghiên cứu, đều rất đơn giản. Chúng ta sẽ có được một cái nhìn tổng thể hơn về một thành phố nào đó nói riêng, quốc gia nói chung. Từ đó, sẽ có nhiều lựa chọn cho cá nhân mình.

## **Mục tiêu nghiên cứu:**

**-** Xây dựng ứng dụng nhập xuất dữ liệu các thành phố bằng ngôn ngữ C.

- Ứng dụng phải đáp ứng được giao diện thân thiện bắt mắt với người dùng, đầy đủ các chức năng,dễ sử dụng.

## **Phạm vi nghiên cứu:**

- Nghiên cứu về các thuật toán sắp xếp, các cấu trúc dữ liệu và các hàm, các dữ liệu nhập và xuất.

## **Khảo sát thực tế:**

- Theo dõi dữ liệu các thành phố 1 cách trực quan và dễ hiểu, dễ tìm kiếm theo sở thích cá nhân, dự định tham quan hoặc làm việc, sinh sống ở thành phố đó.

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

- Tìm hiểu các thành phố, các thông tin cần thiết, các hàm cần thiết, nhập xuất dữ liệu, và sắp xếp.

## **Cấu trúc báo cáo bài tập tập chuyên môn:**

- Cách tiếp cận và phương pháp nghiên cứu, lý thuyết, cách sử dụng.

# **CHƯƠNG II. BÀI TOÁN VỀ NHẬP XUẤT THÀNH PHỐ THEO YÊU CẦU**

## **Giới thiệu bài toán:**

Làm theo yêu cầu đã được giao, qua đó nhập vào và xuất ra, đồng thời sắp xếp các thành phố theo như ý muốn của đề bài.

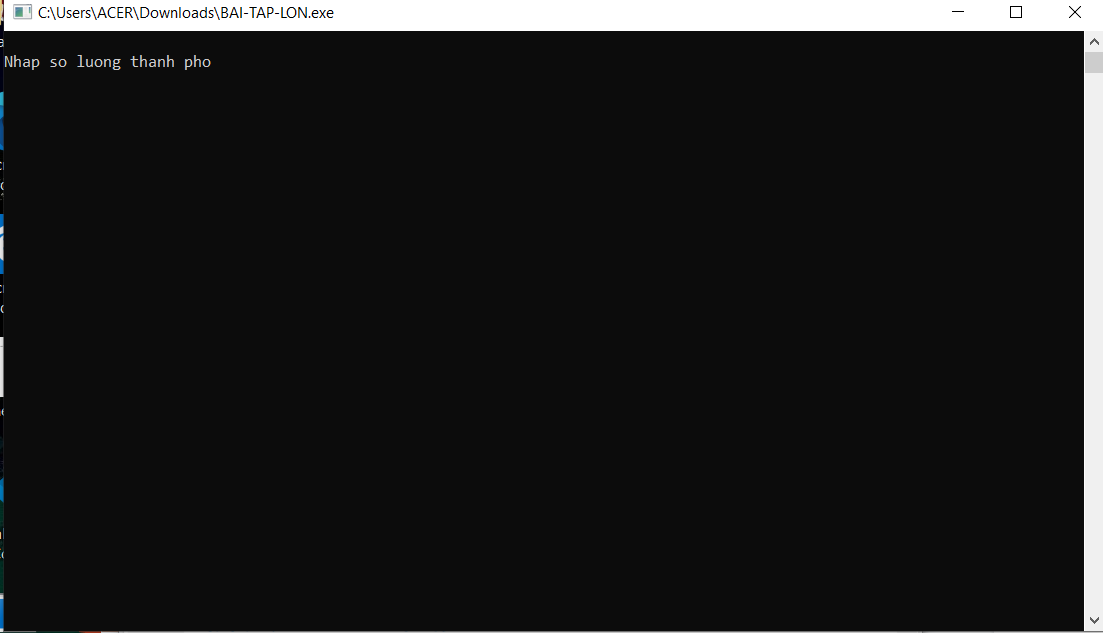
## **Mô tả và cài đặt giải thuật cho bài toán**

- Sử dụng hàm struct, từ đó khai báo và áp dụng vào lập danh sách các thành phố được nhập vào, xây dựng menu từ các case, và từ đó tiếp tục xây dựng các yêu cầu của bài toán.

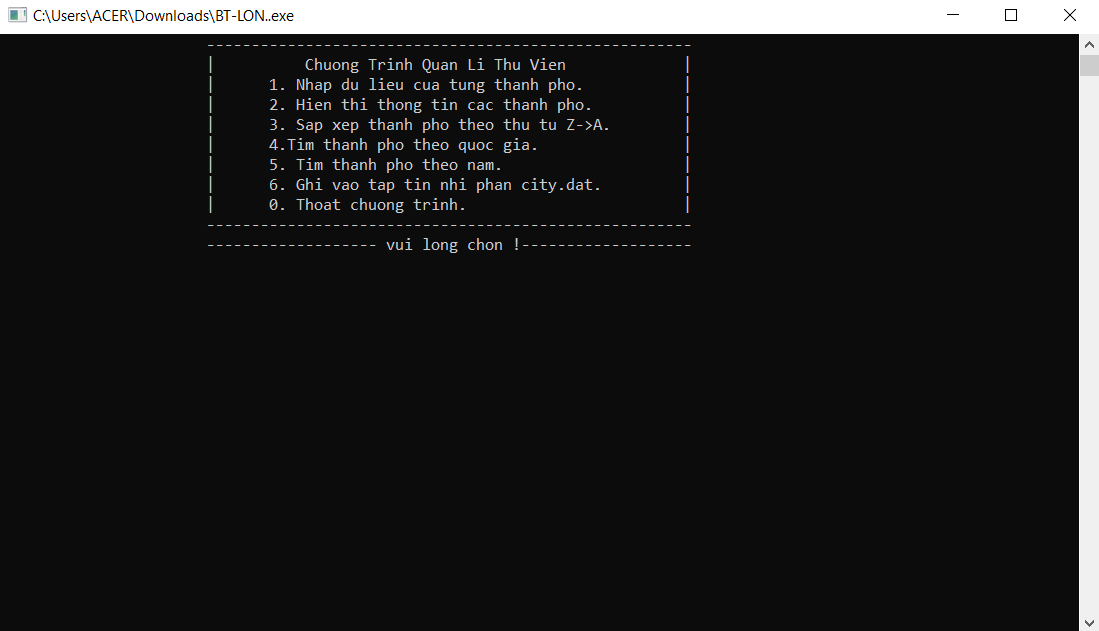
- Sử dụng void để nhập, xuất các thông tin của thành phố theo yêu cầu, đồng thời làm việc với tệp.

## **Hướng dẫn sử dụng**

- Sau khi chạy chương trình sẽ được yêu cầu nhập vào số thành phố theo yêu cầu:



- Sau đó sẽ có menu hiện ra như ảnh, qua đó nhập yêu cầu cần làm theo số ghi trên menu.



- Yêu cầu số 1 sẽ nhập vào thông tin chi tiết của từng thành phố.

- Yêu cầu số 2 sẽ hiển thị thông tin của từng thành phố đã nhập theo bảng dưới đây:

****

**-** Yêu cầu số 3 sẽ sắp xếp từng thành phố theo thứ tự Z -> A.

- Yêu cầu số 4 tìm thành phố theo quốc gia.

- Yêu cầu số 5 tìm thành phố theo năm.

- Yêu cầu số 6 sẽ ghi tất cả thông tin của thành phố thành file nhị phân và xuất ra.

- Yêu cầu số 0 để thoát chương trình.

# **CHƯƠNG III. LÝ THUYẾT ỨNG DỤNG**

## **1. Làm việc với tệp**

**1.1 Cách khai báo mở file**

Khi làm việc với file, cần khai báo 1 con trỏ kiểu FILE.

FILE \*fptr;

Để đọc ghi file trong C cũng như trong mọi ngôn ngữ lập trình, việc đầu tiên cần làm là mở file mà ta muốn làm việc. Trong ngôn ngữ lập trình C, chúng ta có thể mở file bằng cách sử dụng hàm fopen() trong thư viện stdio.h như sau:

fptr = fopen("fileopen","mode")

Trong đó mode là một tham số chúng ta cần chỉ định.

Ví dụ:

fptr = fopen("E:\\cprogram\\newprogram.txt","w");

 // hoặc

fptr = fopen("E:\\cprogram\\oldprogram.bin","rb");

Giả sử tập tin newprogram.txt chưa có trong thư mục E:\cprogram. Ví dụ đầu tiên với mode = "w" sẽ cho phép chương trình tự động tạo ra file newprogram.txt nếu nó chưa có. Và sau đó mở file này lên nhưng chương trình chỉ có thể ghi dữ liệu vào mà không thể đọc.

Mode là w chỉ cho phép chương trình ghi(nếu đã có dữ liệu thì ghi đè) nội dung của file.

Với ví dụ thứ 2, mode là rb cho phép chương trình mở 1 file nhị phân đã có sẵn oldprogram.bin. Với trường hợp này, chương trình chỉ có thể đọc file và không thể ghi nội dung vào file.

Các tham số của “mode”

Dưới đây là các giá trị có thể có của tham số mode  nói trên:

|  |  |  |
| --- | --- | --- |
| **Mode** | **Ý nghĩa** | **Nếu file không tồn tại** |
| r | Mở file chỉ cho phép đọc | Nếu file không tồn tại, fopen() trả về NULL. |
| rb | Mở file chỉ cho phép đọc dưới dạng nhị phân. | Nếu file không tồn tại, fopen() trả về NULL. |
| w | Mở file chỉ cho phép ghi. | Nếu file đã tồn tại, nội dung sẽ bị ghi đè. Nếu file không tồn tại, nó sẽ được tạo tự động. |
| wb | Open for writing in binary mode. | Nếu file đã tồn tại, nội dung sẽ bị ghi đè. Nếu file không tồn tại, nó sẽ được tạo tự động. |
| a | Mở file ở chế độ ghi “append”. Tức là sẽ ghi vào cuối của nội dung đã có. | Nếu file không tồn tại, nó sẽ được tạo tự động. |
| ab | Mở file ở chế độ ghi nhị phân “append”. Tức là sẽ ghi vào cuối của nội dung đã có. | Nếu file không tồn tại, nó sẽ được tạo tự động. |
| r+ | Mở file cho phép cả đọc và ghi. | Nếu file không tồn tại, fopen() trả về NULL. |
| rb+ | Mở file cho phép cả đọc và ghi ở dạng nhị phân. | Nếu file không tồn tại, fopen() trả về NULL. |
| w+ | Mở file cho phép cả đọc và ghi. | Nếu file đã tồn tại, nội dung sẽ bị ghi đè. Nếu file không tồn tại, nó sẽ được tạo tự động. |
| wb+ | Mở file cho phép cả đọc và ghi ở dạng nhị phân. | Nếu file đã tồn tại, nội dung sẽ bị ghi đè. Nếu file không tồn tại, nó sẽ được tạo tự động. |
| a+ | Mở file cho phép đọc và ghi “append”. | Nếu file không tồn tại, nó sẽ được tạo tự động. |
| ab+ | Mở file cho phép đọc và ghi “append” ở dạng nhị phân. | Nếu file không tồn tại, nó sẽ được tạo tự động. |

## **1.2 Thao tác đóng file:**

fclose(fptr);

**1.3 Đọc/ghi file văn bản:**

sử dụng fprintf() và fscanf().

### Ví dụ sử dụng fprintf()

int main()

{

   int num;

   FILE \*fptr;

   fptr = fopen("C:\\program.txt","w");

   if(fptr == NULL)

   {

      printf("Error!");

      exit(1);

   }

   printf("Enter num: ");

   scanf("%d",&num);

   fprintf(fptr,"%d",num);

   fclose(fptr);

   return 0;

}

## **1.4 Đọc file sử dụng fscanf()**

#include <stdio.h>

#include <stdlib.h>

int main()

{

   int num;

   FILE \*fptr;

   if ((fptr = fopen("C:\\program.txt","r")) == NULL){

       printf("Error! opening file");

       // Program exits if the file pointer returns NULL.

       exit(1);

   }

   fscanf(fptr,"%d", &num);

   printf("Value of n=%d", num);

   fclose(fptr);

   return 0;

}

## **1.5 Đọc/Ghi file nhị phân:**

Các hàm fread() và fwrite() trong C được sử dụng để đọc và ghi file trong C ở dạng nhị phân.

Ghi file nhị phân sử dụng fwrite()

#include <stdio.h>

#include <stdlib.h>

struct threeNum

{

   int n1, n2, n3;

};

int main()

{

   int n;

   struct threeNum num;

   FILE \*fptr;

   if ((fptr = fopen("C:\\program.bin","wb")) == NULL){

       printf("Error! opening file");

       // Program exits if the file pointer returns NULL.

       exit(1);

   }

   for(n = 1; n < 5; ++n)

   {

      num.n1 = n;

      num.n2 = 5\*n;

      num.n3 = 5\*n + 1;

      fwrite(&num, sizeof(struct threeNum), 1, fptr);

   }

   fclose(fptr);

   return 0;

}

### Ví dụ đọc file nhị phân sử dụng fread()

#include <stdio.h>

#include <stdlib.h>

struct threeNum

{

   int n1, n2, n3;

};

int main()

{

   int n;

   struct threeNum num;

   FILE \*fptr;

   if ((fptr = fopen("C:\\program.bin","rb")) == NULL){

       printf("Error! opening file");

       // Program exits if the file pointer returns NULL.

       exit(1);

   }

   for(n = 1; n < 5; ++n)

   {

      fread(&num, sizeof(struct threeNum), 1, fptr);

      printf("n1: %d\tn2: %d\tn3: %d", num.n1, num.n2, num.n3);

   }

   fclose(fptr);

return 0;

}

## **2. Danh sách liên kết**

## **2.1 Lý thuyết về danh sác liên kết :**

- Về bản chất, danh sách liên kết có chức năng như một mảng, có thể thêm và xóa các phần tử ở bất kỳ vị trí nào khi cần thiết. Một số sự khác nhau giữa danh sách liên kết và mảng:

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

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

## **2.2 Danh sách liên kết là gì?**

Danh sách liên kết đơn là một tập hợp các Node được phân bố động, được sắp xếp theo cách sao cho mỗi Node chứa “một giá trị”(Data) và “một con trỏ”(Next). Con trỏ sẽ trỏ đến phần tử kế tiếp của danh sách liên kết đó. Nếu con trỏ mà trỏ tới NULL, nghĩa là đó là phần tử cuối cùng của linked list.

Hình ảnh mô tả cho một Node trong danh sách liên kết đơn:

[](https://nguyenvanhieu.vn/wp-content/uploads/2018/12/data-node-danh-sach-lien-ket.png)

Và đây là hình ảnh mô phỏng một danh sách liên đơn kết đầy đủ:

[](https://nguyenvanhieu.vn/wp-content/uploads/2018/12/danh-sach-lien-ket-la-gi.jpg)

Danh sách các kiểu danh sách liên kết:

* Danh sách liên kết đơn(Single linked list): Chỉ có sự kết nối từ phần tử phía trước tới phần tử phía sau.
* Danh sách liên kết đôi(Double linked list): Có sự kết nối 2 chiều giữa phần tử phía trước với phần tử phía sau
* Danh sách liên kết vòng(Circular Linked List): Có thêm sự kết nối giữa 2 phần tử đầu tiên và phần tử cuối cùng để tạo thành vòng khép kín.

## **2.3 Cài đặt danh sách liên kết đơn:**

### *Khai báo và khởi tạo liên kết đơn:*

 struct city\_st

      {

          char city\_nm[30];

          char country[30];

          int year,Stt;

      };

### **2.3.1 Khai báo linked list**

Để đơn giản hóa, data của chúng ta sẽ là số nguyên(int). Ta cũng có thể sử dụng các kiểu nguyên thủy khác(float, char,…) hay kiểu dữ liệu struct(SinhVien, CanBo,…) tự tạo.

struct LinkedList{

    int data;

    struct LinkedList \*next;

 };

## Khai báo trên sẽ được sử dụng cho mọi Node trong linked list. Trường data sẽ lưu giữa giá trị và next sẽ là con trỏ để trỏ đến thằng kế tiếp của nó.

### **2.3.2 Tạo mới 1 Node**

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

typedef struct LinkedList \*node; //Từ giờ dùng kiểu dữ liệu LinkedList có thể thay bằng node cho ngắn gọn

node CreateNode(int value){

    node temp; // declare a node

    temp = (node)malloc(sizeof(struct LinkedList)); // Cấp phát vùng nhớ dùng malloc()

    temp->next = NULL;// Cho next trỏ tới NULL

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

    return temp;//Trả về node mới đã có giá trị

}

Mỗi một Node khi được khởi tạo, chúng ta cần cấp phát bộ nhớ cho nó, và mặc định cho con trỏ next trỏ tới NULL. Giá trị của Node sẽ được cung cấp khi thêm Node vào linked list.

* **typedef** được dùng để định nghĩa một kiểu dữ liệu trong C. VD: typeder long long LL;
* **malloc** là hàm cấp phát bộ nhớ của C. Với C++ chúng ta dùng new
* **sizeof** là hàm trả về kích thước của kiểu dữ liệu, dùng làm tham số cho hàm malloc

**Lưu ý:** Không giống với mảng, cần khai báo arr[size]. Trong linked list, vì mỗi Node sẽ có con trỏ liên kết đến Node tiếp theo. Do đó, với danh sách liên kết đơn, ta chỉ cần lưu giữ Node đầu tiên(HEAD). Có head rồi ta có thể đi tới bất cứ Node nào.

### **2.3.3 Thêm Node vào danh sách liên kết**

#### **Thêm vào đầu**

Việc thêm vào đầu chính là việc cập nhật lại head. Ta gọi Node mới(temp), ta có:

* Nếu head đang trỏ tới NULL, nghĩa là linked list đang trống, Node mới thêm vào sẽ làm head
* Ngược lại, ta phải thay thế head cũ bằng head mới. Việc này phải làm theo thứ tự như sau:
  + Cho next của temp trỏ tới head hiện hành
  + Đặt temp làm head mới

node AddHead(node head, int value){

    node temp = CreateNode(value); // Khởi tạo node temp với data = value

    if(head == NULL){

        head = temp; // //Nếu linked list đang trống thì Node  temp là head luôn

    }else{

        temp->next = head; // Trỏ next của temp = head hiện tại

        head = temp; // Đổi head hiện tại = temp(Vì temp bây giờ là head mới)

    }

    return head;

}

#### **Thêm vào cuối**

Chúng ta sẽ cần Node đầu tiên, và giá trị muốn thêm. Khi đó, ta sẽ:

1. Tạo một Node mới với giá trị value
2. Nếu head = NULL, tức là danh sách liên kết đang trống. Khi đó Node mới(temp) sẽ là head luôn.
3. Ngược lại, ta sẽ duyệt tới Node cuối cùng(Node có next = NULL), và trỏ next của thằng cuối tới Node mới(temp).node AddTail(node head, int value){

    node temp,p;// Khai báo 2 node tạm temp và p

    temp = CreateNode(value);//Gọi hàm createNode để khởi tạo node temp có next trỏ tới NULL và giá trị là value

    if(head == NULL){

        head = temp;     //Nếu linked list đang trống thì Node  temp là head

    }

    else{

        p  = head;// Khởi tạo p trỏ tới head

        while(p->next != NULL){

            p = p->next;//Duyệt danh sách liên kết đến cuối. Node cuối là node có  next = NULL

        }

        p->next = temp;//Gán next của cái cuối = temp. Khi đó temp sẽ là cái cuối(temp->next = NULL mà)

    }

    return head;

}

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

#### **Thêm vào vị trí bất kỳ**

## [Thêm Node vào giữa danh sách liên kết](https://nguyenvanhieu.vn/wp-content/uploads/2018/12/them-node-danh-sach-lien-ket.png)

Để làm được việc này, ta phải duyệt từ đầu để tìm tới vị trí của Node cần chèn, giả sử là Node Q, khi đó ta cần làm theo thứ tự sau:

* Cho next của Node mới trỏ tới Node mà Q đang trỏ tới
* Cho Node Q trỏ tới Node mới

Lưu ý: Chỉ số chèn bắt đầu từ chỉ số 0 nhé

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

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

        head = AddHead(head, value); // Nếu vị trí chèn là 0, tức là thêm vào đầu

    }else{

        // Bắt đầu tìm vị trí cần chèn. Ta sẽ dùng k để đếm cho vị trí

        int k = 1;

        node p = head;

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

            p = p->next;

            ++k;

        }

        if(k != position){

            // Nếu duyệt hết danh sách lk rồi mà vẫn chưa đến vị trí cần chèn, ta sẽ mặc định chèn cuối

            // Nếu không muốn chèn, hãy thông báo vị trí chèn  không hợp lệ

            head = AddTail(head, value);

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

        }else{

            node temp = CreateNode(value);

            temp->next = p->next;

            p->next = temp;

        }

    }

    return head;

}

Lưu ý: Chúng ta phải làm theo thứ tự trên, nếu bạn cho p->next = temp trước. Khi đó, ta sẽ không thể lấy lại phần sau của danh sách liên kết . (Vì next chỉ được được lưu trong p->next mà thay đổi p->next rồi thì sẽ không còn giá trị cũ).

### **2.3.4. Xóa Node khỏi danh sách liên kết**

#### **Xóa đầu**

Xóa đầu rất đơn giản, bây giờ chỉ cần sao cho kế tiếp của head làm head là được. Mà cái kế tiếp của head chính là head->next.

#### **Xóa cuối**

Xóa cuối sẽ khó hơn, vì ta phải giải quyết ở chỗ duyệt đến thằng cuối – 1, cho next của cuối – 1 đó bằng NULL.

node DelTail(node head){

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

         return DelHead(head);

    }

    node p = head;

    while(p->next->next != NULL){

        p = p->next;

    }

    p->next = p->next->next; // Cho next bằng NULL

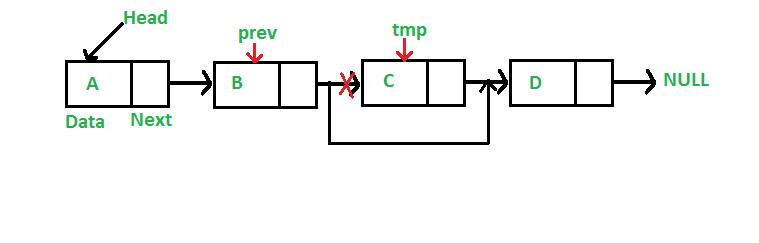
    // Hoặc viết p->next = NULL cũng được

    return head;

}

Node cuối – 1 là cái có p->next->next = NULL. Ta chỉ cần cho next của của cái trên bằng NULL là xong.

#### **Xóa ở vị trí bất kỳ**

[](https://nguyenvanhieu.vn/wp-content/uploads/2018/12/xoa-node-danh-sach-lien-ket.png)Việc xóa ở vị trí bất kỳ cũng khá giống xóa ở cuối phía trên. Đơn giản là chúng ta bỏ qua một phần tử, như ảnh sau:

Lưu ý: Chỉ số xóa bắt đầu từ 0. Việc tìm vị trí cần xóa chỉ duyệt tới Node gần cuối. (cuối – 1). Sau đây là code xóa Node ở vị trí bất kỳ

node DelAt(node head, int position){

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

        head = DelHead(head); // Nếu vị trí chèn là 0, tức là thêm vào đầu

    }else{

        // Bắt đầu tìm vị trí cần chèn. Ta sẽ dùng k để đếm cho vị trí

        int k = 1;

        node p = head;

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

            p = p->next;

            ++k;

        }

        if(k != position){

            // Nếu duyệt hết danh sách lk rồi mà vẫn chưa đến vị trí cần chèn, ta sẽ mặc định xóa cuối

            // Nếu không muốn xóa, hãy thông báo vị trí xóa không hợp lệ

            head = DelTail(head);

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

        }else{

            p->next = p->next->next;

        }

    }

    return head;

}

### **2.3.5 Lấy giá trị ở vị trí bất kỳ**

Chúng ta sẽ viết một hàm để truy xuất giá trị ở chỉ số bất kỳ nhé. Trong trường hợp chỉ số vượt quá chiều dài của linked list – 1, hàm này trả về vị trí cuối cùng. Do hạn chế là chúng ta không thể raise error khi chỉ số không hợp lệ. Phải mặc định chỉ số truyền vào phải là số nguyên không âm. Nếu muốn kiểm tra chỉ số hợp lệ thì nên kiểm tra trước khi gọi hàm này.

int Get(node head, int index){

    int k = 0;

    node p = head;

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

        ++k;

        p = p->next;

    }

    return p->data;

}

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

### **2.3.6 Tìm kiếm trong danh sách liên kết**

Hàm tìm kiếm này sẽ trả về chỉ số của Node đầu tiên có giá trị bằng với giá trị cần tìm. Nếu không tìm thấy, ta trả về -1.

int Search(node head, int value){

    int position = 0;

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

        if(p->data == value){

            return position;

        }

        ++position;

    }

    return -1;

}

## Chúng ta có thể sử dụng hàm này để xóa tất cả các Node trong danh sách liên kết có giá trị chỉ định như sau:

node DelByVal(node head, int value){

    int position = Search(head, value);

    while(position != -1){

        DelAt(head, position);

        position = Search(head, value);

    }

    return head;

}

### **2.3.7 Duyệt danh sách liên kết**

Việc duyệt danh sách liên kết cực đơn giản. Khởi tạo từ Node head, cứ thế đi theo con trỏ next cho tới trước khi Node đó NULL.

void Traverser(node head){

    printf("\n");

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

        printf("%5d", p->data);

    }

}

### **2.3.8 Một số hàm bổ trợ khác**

#### **Hàm khởi tạo Node head**

Đơn giản là cho con trỏ head = NULL. Nếu để ý, chúng ta vẫn check head = NULL để biết rằng danh sách liên kết chưa có phần tử nào ở các hàm phía trên.

node InitHead(){

    node head;

    head = NULL;

    return head;

}

#### **Hàm lấy số phần tử của DSLK**

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

int Length(node head){

    int length = 0;

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

        ++length;

    }

    return length;

}

#### **Hàm nhập danh sách liên kết**

node Input(){

    node head = InitHead();

    int n, value;

    do{

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

        scanf("%d", &n);

    }while(n <= 0);

    for(int i = 0; i < n; ++i){

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

        scanf("%d", &value);

        head = AddTail(head, value);

    }

    return head;

}

**2.4 Code hoàn chỉnh của danh sách liên kết đơn**

#include<stdio.h>

#include<stdlib.h>

struct LinkedList{

    int data;

    struct LinkedList \*next;

 };

typedef struct LinkedList \*node; //Từ giờ dùng kiểu dữ liệu LinkedList có thể thay bằng node cho ngắn gọn

node CreateNode(int value){

    node temp; // declare a node

    temp = (node)malloc(sizeof(struct LinkedList)); // Cấp phát vùng nhớ dùng malloc()

    temp->next = NULL;// Cho next trỏ tới NULL

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

    return temp;//Trả về node mới đã có giá trị

}

node AddTail(node head, int value){

    node temp,p;// Khai báo 2 node tạm temp và p

    temp = CreateNode(value);//Gọi hàm createNode để khởi tạo node temp có next trỏ tới NULL và giá trị là value

    if(head == NULL){

        head = temp;     //Nếu linked list đang trống thì Node temp là head luôn

    }

    else{

        p  = head;// Khởi tạo p trỏ tới head

        while(p->next != NULL){

            p = p->next;//Duyệt danh sách liên kết đến cuối. Node cuối là node có next = NULL

        }

        p->next = temp;//Gán next của thằng cuối = temp. Khi đó temp sẽ là thằng cuối(temp->next = NULL mà)

    }

    return head;

}

node AddHead(node head, int value){

    node temp = CreateNode(value); // Khởi tạo node temp với data = value

    if(head == NULL){

        head = temp; // //Nếu linked list đang trống thì Node temp là head luôn

    }else{

        temp->next = head; // Trỏ next của temp = head hiện tại

        head = temp; // Đổi head hiện tại = temp(Vì temp bây giờ là head mới mà)

    }

    return head;

}

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

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

        head = AddHead(head, value); // Nếu vị trí chèn là 0, tức là thêm vào đầu

    }else{

        // Bắt đầu tìm vị trí cần chèn. Ta sẽ dùng k để đếm cho vị trí

        int k = 1;

        node p = head;

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

            p = p->next;

            ++k;

        }

        if(k != position){

            // Nếu duyệt hết danh sách lk rồi mà vẫn chưa đến vị trí cần chèn, ta sẽ mặc định chèn cuối

            // Nếu không muốn chèn, hãy thông báo vị trí chèn không hợp lệ

            head = AddTail(head, value);

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

        }else{

            node temp = CreateNode(value);

            temp->next = p->next;

            p->next = temp;

        }

    }

    return head;

}

node DelHead(node head){

    if(head == NULL){

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

    }else{

        head = head->next;

    }

    return head;

}

node DelTail(node head){

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

         return DelHead(head);

    }

    node p = head;

    while(p->next->next != NULL){

        p = p->next;

    }

    p->next = p->next->next; // Cho next bằng NULL

    // Hoặc viết p->next = NULL cũng được

    return head;

}

node DelAt(node head, int position){

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

        head = DelHead(head); // Nếu vị trí chèn là 0, tức là thêm vào đầu

    }else{

        // Bắt đầu tìm vị trí cần chèn. Ta sẽ dùng k để đếm cho vị trí

        int k = 1;

        node p = head;

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

            p = p->next;

            ++k;

        }

        if(k != position){

            // Nếu duyệt hết danh sách lk rồi mà vẫn chưa đến vị trí cần chèn, ta sẽ mặc định xóa cuối

            // Nếu không muốn xóa, hãy thông báo vị trí xóa không hợp lệ

            head = DelTail(head);

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

        }else{

            p->next = p->next->next;

        }

    }

    return head;

}

int Get(node head, int index){

    int k = 0;

    node p = head;

    while(p != NULL && k != index){

        p = p->next;

    }

    return p->data;

}

int Search(node head, int value){

    int position = 0;

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

        if(p->data == value){

            return position;

        }

        ++position;

    }

    return -1;

}

node DelByVal(node head, int value){

    int position = Search(head, value);

    while(position != -1){

        DelAt(head, position);

        position = Search(head, value);

    }

    return head;

}

void Traverser(node head){

    printf("\n");

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

        printf("%5d", p->data);

    }

}

node InitHead(){

    node head;

    head = NULL;

    return head;

}

int Length(node head){

    int length = 0;

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

        ++length;

    }

    return length;

}

node Input(){

    node head = InitHead();

    int n, value;

    do{

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

        scanf("%d", &n);

    }while(n <= 0);

    for(int i = 0; i < n; ++i){

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

        scanf("%d", &value);

        head = AddTail(head, value);

    }

    return head;

}

int main(){

    printf("\n==Tao 1 danh sach lien ket==");

    node head = Input();

    Traverser(head);

    printf("\n==Thu them 1 phan tu vao linked list==");

    head = AddAt(head, 100, 2);

    Traverser(head);

    printf("\n==Thu xoa 1 phan tu khoi linked list==");

    head = DelAt(head, 1);

    Traverser(head);

    printf("\n==Thu tim kiem 1 phan tu trong linked list==");

    int index = Search(head, 5);

    printf("\nTim thay tai chi so %d", index);

    printf("\nBan co the thu them nhe!");

}

Kết quả chạy thử:

==Tao 1 danh sach lien ket==

Nhap so luong phan tu n = 5

Nhap gia tri can them: 1

Nhap gia tri can them: 2

Nhap gia tri can them: 3

Nhap gia tri can them: 4

Nhap gia tri can them: 5

    1    2    3    4    5

==Thu them 1 phan tu vao linked list==

    1    2  100    3    4    5

==Thu xoa 1 phan tu khoi linked list==

    1  100    3    4    5

==Thu tim kiem 1 phan tu trong linked list==

Tim thay tai chi so 4

Ban co the thu them nhe!

Process returned 0 (0x0)   execution time : 4.166 s

Press any key to continue.

## **3. Cấu trúc dữ liệu struct**

## **3.1 Cách định nghĩa struct**

Trước khi chúng ta có thể khai báo biến với struct, cần định nghĩa lại – Đó cũng là lý do tại sao struct được gọi là kiểu dữ liệu người dùng định nghĩa.

Khi nào cần phải tự định nghĩa 1 kiểu cấu trúc? Khi ta cần lưu trữ một đối tượng có nhiều thuộc tính. Ví dụ, đối tượng SinhVien có các thuộc tính (Mã sinh viên, họ, tên, giới tính, quê quán,…) hay đối tượng LopHoc có các thuộc tính (Mã lớp, tên lớp, giáo viên chủ nhiệm, sĩ số,…). Khi đó chúng ta nên dùng struct để quản lý chương trình.

**3.2 Cú pháp định nghĩa struct**

 struct city\_st

      {

          char city\_nm[30];

          char country[30];

          int year,Stt;

      };

Như vậy, kiểu dữ liệu city\_st đã được định nghĩa. Từ đây chúng ta có thể khai báo các biến với kiểu dữ liệu này.

node DelHead(node head){

    if(head == NULL){

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

    }else{

        head = head->next;

    }

    return head;

}

## **3.3 Cách khai báo biến kiểu struct**

Việc khai báo biến với struct cũng giống như cách khai báo biến thông thường, trong đó kiểu dữ liệu là kiểu struct trong C mà ta vừa định nghĩa. Xem ví dụ dưới đây:

struct SinhVien

{

    int maSV;

    char ho[20];

    char ten[20];

    bool gioiTinh;

    char queQuan[100];

};

int main(){

    // Khai báo 2 biến sv1 và sv2 có kiểu SinhVien

    SinhVien sv1, sv2;

    // Ta nên thêm từ khóa struct ở đầu,

    // để phân biệt được biến này là biến của kiểu dữ liệu tự  định nghĩa

    struct SinhVien sv3, sv4;

    // Khai báo mảng

    struct SinhVien sv[100];

}

### **3.4 Truy xuất các thuộc tính của struct**

Chúng ta có 2 toán tử dùng để truy xuất tới các biến thành viên của kiểu struct trong C.

* Sử dụng . => Toán tử truy xuất tới thành viên khi khai báo biến bình thương.
* Sử dụng -> => Toán tử truy xuất tới thành viên khi biến là con trỏ.

Giả sử trong ví dụ trên, bạn muốn truy xuất gioiTinh của đối tượng sinh viên, bạn làm như sau:

SinhVien sv;

// to do

printf("Gioi tinh: %s", sv.gioiTinh);

**3.5 Từ khóa typedef**

Có thể sử dụng từ khóa typedef để tạo ra một tên thay thế cho kiểu dữ liệu đã có. Thường sử dụng kiểu struct để đơn giản hóa cú pháp khai báo biến. Nhưng cũng có thể sử dụng với các kiểu dữ liệu nguyên thủy.

struct Distance{

    int feet;

    float inch;

};

int main() {

    structure Distance d1, d2;

}

Code trên tương đương với:

typedef struct SinhVien{

    int feet;

    float inch;

} distances;

int main() {

    distances d1, d2;

}

Hoặc:

struct PhanSo{

    int tu;

    int mau;

};

typedef struct PhanSo PS;

### **3.6 Cấu trúc struct lồng nhau:**

Giả sử ta muốn xây dựng kiểu dữ liệu để lưu trữ đối tượng Tam giác, khi đó chúng ta có thể xây dựng struct mô tả tọa độ của 1 điểm, khi đó đối tượng tam giác sẽ là 3 đối tượng điểm. Cụ thể:

struct Point{

    int x; // hoành độ

    int y; // tung độ

};

struct Triangle{

    Point a; // đỉnh thứ 1

    Point b; // đỉnh thứ 2

    Point c; // đỉnh thứ 3

}

int main(){

    Triangle tg;

    // truy xuất hoành độ của điểm thứ nhất

    tg.a.x = 5;

}

### **3.7 Struct và con trỏ**

Tương tự như khai báo con trỏ với các kiểu dữ liệu có sẵn trong C. Chúng ta cũng có thể khai báo biến con trỏ, cấp phát động cho biến con trỏ kiểu struct.

Sau đây là cách chúng ta khai báo biến con trỏ kiểu struct trong C:

struct name {

    member1;

    member2;

    .

    .

};

int main()

{

    struct name \*ptr, Harry;

}

Khi đó ptr là con trỏ kiểu name, còn Harry là biến kiểu name.

Để truy cập vào các biến thành viên sử dụng biến con trỏ của struct trong C, dùng ->, ví dụ:

#include <stdio.h>

struct person

{

   int age;

   float weight;

};

int main()

{

    struct person \*personPtr, person1;

    personPtr = &person1;

    printf("Enter age: ");

    scanf("%d", &personPtr->age);

    printf("Enter weight: ");

    scanf("%f", &personPtr->weight);

    printf("Displaying:\n");

    printf("Age: %d\n", personPtr->age);

    printf("weight: %f", personPtr->weight);

    return 0;

}

**Áp dụng vô bài tập lớn:**

#include<stdio.h>

#include<conio.h>

#include<stdlib.h>

#include<string.h>

#define MAX 10

struct city\_st;

void hienthiDSSS(struct city\_st\*, int n);

void hienthitheoCot();

    struct city\_st

      {

          char city\_nm[30];

          char country[30];

          int year,Stt;

      };

     typedef struct city\_st cs;

        void NhapN(city\_st fc[],int n )

      {

         for(int i=0;i<n;i++)

           {

               printf("\n\tNhap thong tin  thanh pho thu:%d",i+1);

               printf("\nNhap so thu tu cho thanh pho thu %d :",i+1);

               scanf("%d",&fc[i].Stt);

               printf("\n\tNhap ten thanh pho:");

               fflush(stdin);

               gets(fc[i].city\_nm);

               printf("\n\tNhap quoc gia :");

               fflush(stdin);

               gets(fc[i].country);

               printf("\n\tNhap nam thanh lap :");

               scanf("%d",&fc[i].year);

           }

      }

## **3.8 Địa chỉ của biến**

Để hiểu và sử dụng được con trỏ trong C, trước tiên ta cần hiểu về khái niệm địa chỉ ở trong C.

int number;

printf("\nNhap number = ");

scanf("%d", &number);

printf("\nnumber = %d", number);

Hãy nhìn ví dụ trên, tại sao khi dùng hàm scanf chúng ta cần truyền vào &number, còn hàm printf lại không có dấu & kia? Bởi vì nếu muốn nhập giá trị cho biến, hàm scanf cần biết địa chỉ của biến đó ở trong bộ nhớ.

Mỗi biến mà ta khai báo đều có địa chỉ riêng của nó và giá trị mà nó đang lưu trữ. Để xem được địa chỉ của biến, ta thêm dấu & vào trước tên biến. Xem xét ví dụ dưới đây:

#include <stdio.h>

int main()

{

  int number = 5;

  printf("Gia tri cua number = %d", number);

  // truy xuất địa chỉ bằng cách thêm & trước tên biến

  printf("\nDia chi cua number = %d", &number);

  return 0;

}

Kết quả khi chạy chương trình:

Gia tri cua number = 5

Dia chi cua number = 6487580

**Chú ý:**

* Ta có thể sẽ nhận được các địa chỉ khác nhau mỗi khi chạy code trên.
* Để nhận giá trị địa chỉ là hexa như ảnh ở đâu bài, bạn thay %d bằng %x là được.

## **4. Con trỏ trong C**

**4.1 Con trỏ là gì?**Con trỏ trong C cũng chỉ là là biến, cũng có thể khai báo, khởi tạo và lưu trữ giá trị và có địa chỉ của riêng. Nhưng biến con trỏ không lưu giá trị bình thường, là biến trỏ tới 1 địa chỉ khác, tức mang giá trị là 1 **địa chỉ**.

Chúng ta cùng thống nhất 1 số khái niệm khi làm việc với con trỏ :

* Giá trị của con trỏ: địa chỉ mà con trỏ trỏ đến.
* Địa chỉ của con trỏ: địa chỉ của bản thân biến con trỏ đó.
* Giá trị của biến nơi con trỏ đang trỏ tới.
* Địa chỉ của biến nơi con trỏ đang trỏ tới = giá trị của con trỏ.

Chính vì con trỏ mang địa chỉ, nó là 1 biến đặc biệt có thêm những quyền năng mà biến bình thường không có. Nhờ việc nó mang địa chỉ, nó có thể trỏ lung tung trong bộ nhớ. Đây là 1 điểm mạnh nếu ta khai thác tốt nhưng nếu quản lý không tốt thì lại là 1 tai hại.

## **4.2 Cách khai báo con trỏ**

Con trỏ trong C cũng có thể khai báo giống như biến bình thường, tên biến là một định danh hợp lệ. Cú pháp như sau:

<kiểu dữ liệu> \* <tên biến>

Trong đó:

* Kiểu dữ liệu có thể là: void, int, float, double,…
* Dấu \* trước tên biến là ký hiệu báo cho trình biên dịch biết ta đang khai báo con trỏ.
* int \*p\_i; // khai báo con trỏ để trỏ tới biến kiểu nguyên
* int \*p, val; // khai báo con trỏ p kiểu int, biến val (không phải con trỏ) kiểu int
* float \*p\_f; // khai báo con trỏ để trỏ tới biến kiểu thực
* char \*p\_char; // khai báo con trỏ để trỏ tới biến kiểu ký tự
* void \*p\_v; // con trỏ kiểu void (không kiểu)

## **4.3 Gán giá trị cho con trỏ**

Sau khi khai báo con trỏ, ta cần khởi tạo giá trị cho nó. Nếu con trỏ được sử dụng mà không được khởi tạo, giá trị của nó sẽ là giá trị rác, điều này sẽ làm chương trình của ta chạy không đúng, thậm chí là nguy hiểm nếu giá trị rác đó chẳng may lại chính là địa chỉ của 1 biến nào đó bạn đang dùng.

int \*p, value;

value = 5;

p = &value;// khởi tạo giá trị cho con trỏ p là địa chỉ của value

Hoặc cũng có thể khai báo và khởi tạo đồng thời:

int value = 5;

int \*p = &value; // khai báo con trỏ p và khởi tạo giá trị cho con trỏ là địa chỉ của value

**Lưu ý:**

* Con trỏ khi khai báo nên được khởi tạo giá trị ngay.
* Con trỏ kiểu void là loại biến con trỏ tổng quát, nó có thể nhận địa chỉ của biến bất kỳ ở bất cứ kiểu dữ liệu nào.

#include <stdio.h>

int main()

{

  int number = 5;

  float \*p\_int = &number;

}

// Ouput:

PS G:\c\_cources\day\_63> g++ .\Pointer.cpp

.\Pointer.cpp: In function 'int main()':

.\Pointer.cpp:5:19: error: cannot convert 'int\*' to 'float\*' in initialization

   float \*p\_int = &number;

Khởi tạo con trỏ bằng địa chỉ NULL nếu chưa cần dùng theo cách sau: int \*p = NULL. Khi đó con trỏ NULL luôn có giá trị 0.

#include <stdio.h>

int main()

{

  void \*p\_int = NULL;

  printf("Gia tri cua con tro la %d", p\_int);

}

// Output

// Gia tri cua con tro la 0

### **4.4 Bản chất của con trỏ trong C**

## Ta sẽ hiểu rõ hơn quyền năng của con trỏ qua ví dụ sau:

#include <stdio.h>

int main()

{

  // Khai báo + khởi tạo biến value = 10

  int value = 10;

  // Lấy giá trị của biến value

  printf("\nGia tri cua `value` = %d", value);

  // Lấy địa chỉ của biến value

  printf("\nDia tri cua `value` = %d", &value);

  printf("\n-------------------\n");

  /\*

  Khai báo + khởi tạo biến con trỏ p

  có giá trị là địa chỉ của biến value

  \*/

  int \*p = &value;

  // Lấy giá trị của con trỏ p

  printf("\nGia tri cua con tro `p` = %d", p);

  // Lấy địa chỉ của con trỏ p

  printf("\nDia tri cua con tro `p` = %d", &p);

  // Lấy giá trị của biến ma con trỏ p đang trỏ tới dùng toán tử \*

  printf("\nGia tri cua bien ma con tro `p` dang tro toi = %d", \*p);

  printf("\n-------------------\n");

  /\*

  Thay đổi giá trị của biến value thông qua con trỏ p

  Giống như hàm scanf() có thể thay đổi giá trị của biến khi nhận vào địa chỉ,

  con trỏ khi có địa chỉ của 1 biến hoàn toàn có thể thay đổi giá trị của

  biến đó theo cách dưới đây:

  \*/

  // Lấy giá trị của biến value

  printf("\nGia tri cua `value` = %d", value);

  // Thay đổi giá trị của biến value thông qua `p`

  \*p = 100;

  // Lấy giá trị của biến value

  printf("\nGia tri cua `value` = %d", value);

  // Lấy giá trị của biến ma con trỏ p đang trỏ tới dùng toán tử \*

  printf("\nGia tri cua bien ma con tro `p` dang tro toi = %d", \*p);

  printf("\n-------------------\n");

  /\*

  Việc lấy giá trị của biến thông qua con trỏ

  chỉ là 1 cách khác để lấy được giá trị của biến đó.

  \*/

  value = 1000;

  // Lấy giá trị của biến value

  printf("\nGia tri cua `value` = %d", value);

  // Lấy giá trị của biến ma con trỏ p đang trỏ tới dùng toán tử \*

  printf("\nGia tri cua bien ma con tro `p` dang tro toi = %d", \*p);

}

Kết quả chạy:

Gia tri cua `value` = 10

Dia tri cua `value` = 6487580

-------------------

Gia tri cua con tro `p` = 6487580

Dia tri cua con tro `p` = 6487568

Gia tri cua bien ma con tro `p` dang tro toi = 10

-------------------

Gia tri cua `value` = 10

Gia tri cua `value` = 100

Gia tri cua bien ma con tro `p` dang tro toi = 100

-------------------

Gia tri cua `value` = 1000

Gia tri cua bien ma con tro `p` dang tro toi = 1000

Qua ví dụ này, ta có thể thấy rõ sự đúng đắn của các kết luận sau đây về con trỏ:

* Địa chỉ của biến value chính là giá trị của con trỏ p, đều là 6487580. Lưu ý mỗi lần chạy thì giá trị địa chỉ này có thể khác nhau nhé.
* Con trỏ có thể lấy giá trị của biến mà nó đang trỏ tới bằng toán tử \*: printf("\nDia tri cua con tro p = %d", \*p);
* Con trỏ có thể thay đổi giá trị của biến mà nó đang trỏ tới. Do nó mang địa chỉ của biến, khi đó nó hoàn toàn có quyền thay đổi giá trị của biến đó. Như ở ví dụ trên ta thay đổi giá trị từ 10 lên 100.

### **4.5 Các lỗi thường gặp khi làm việc với con trỏ**

Giả sử ta muốn khởi tạo giá trị của con trỏ p trỏ tới địa chỉ của biến value, khi đó

int value, \*p;

// Sai! p cần địa chỉ cơ,

// value không phải là cái địa chỉ đó.

p = value;

// Sai! \*p là giá trị của biến mà con trỏ đang trỏ tới,

// &value là địa chỉ.

\*p = &value;

// Đúng rồi! p cần 1 địa chỉ,

// &value là địa chỉ của biến value.

p = &value;

// Đúng! \*p là giá trị của biến mà con trỏ đang trỏ tới, và

// c cũng là giá trị (không phải địa chỉ).

\*p = value;

Khi mới học con trỏ ta sẽ mông lung về dấu \* ở phần khai báo và khi lấy giá trị của biến mà con trỏ đang trỏ tới:

#include <stdio.h>

int main()

{

    int c = 5;

    // Dấu \* ở đây để chúng ta biết chúng ta đang khai báo con trỏ.

    // Không phải lấy giá trị của nó nhé

    int \*p = &c;

    // Khai báo trên tương đương

    // int \*p;

    // p = &c;

    // Nếu bạn muốn phân biệt 2 thằng này, khi khai báo có thể viết như sau:

    // int\* p = &c;

    // Lấy giá chỉ của biến mà con trỏ đang trỏ tới, chính là giá trị của c

    printf("%d", \*p); // 5

}

## **Áp dụng vô bài tập lớn:**

void sapxeptheoten(struct city\_st \*fc,int n)

              {

                int i,j;

                for(i=0;i<n-1;i++)

                    {

                        for(j=n-1;j>i;j--)

                           {

                               if(strcmp(fc[j].city\_nm,fc[j-1].city\_nm)>0)

                               {

                                struct city\_st a=fc[j];

                                fc[j]=fc[j-1];

                                fc[j-1]=a;

                                }

                           }

                    }

                }

## **5. Các giải thuật tìm kiếm , sắp xếp :** **5.1 Bài toán Tìm kiếm:**

Ví dụ cho dãy n số nguyên a0,a1,a2,..a(n-1) và một số nguyên x.Hãy tìm xem x có thuộc vào dãy số trên hay không. Nếu tìm được ở vị trí i thì xuất kết quả là I ; ngược lại nếu không tìm thấy thì xuất kết quả là -1(chú ý dãy bắt đầu từ chỉ số 0, nếu dãy có nhiều số bằng x thì xuất vị trí I nhỏ nhất nếu cần )

Sau đây là thuật toán tìm kiếm thường được sử dụng nhất có tên gọi là tìm kiếm tuyến tính .Ý tưởng chính của thuật toán tìm kiếm như sau : Bắt đầu từ phần tử thứ a[0] ,ta lần lượt so sánh x với các giá trị a[i].Nếu có a[i] bằng x thì I chính là kết quả cần tìm và kết thúc thuật toán .Nếu trong dãy không có số a[i] nào bằng x thì xuất kết quả là -1 và cũng kết thúc thuật toán.

Đoạn code thể hiện ý tưởng tìm kiếm tuyến tính như hàm sau :

       int timkiem(int a[], int n ,int x)

        {

          int i=0;

          while(i<n &&a[i]!=x)

        i++;

if(i==n)

return -1;//tìm kiếm nhưng không có x

return i;//tìm thấy a[i] là phần tử có khóa ( giá trị )x

   }

### **5.2 Bài toán Sắp xếp :**

- Trong các thuật toán sắp xếp , thuật toán đơn gỉn nhất đó là phương pháp sắp xếp đổi chỗ trực tiếp .

- Xét một mảng n số a0 ,a1,a(n-1). Nếu i<j và ai>aj,thì ta gọi đó là một nghịch thế .Mảng chưa sắp xếp sẽ có nghịch thế và ngược lại mảng đã có thứ tự sẽ không còn nghịch thế . Để sắp xếp một mảng ,ta có thể tìm cách làm giảm số các nghịch thế trong mảng này bằng cách hoán vị các phần tử ai,aj nếu có i<j,ai>aj (theo thuật toán hoán vị )

+ Đoạn code thê hiện ý tưởng sắp xếp đổi chỗ trực tiếp như hàm sau :

//đầu tiên cho viết hàm hoán vị

void hoanvi(int &a,int &b)

{

    int temp=0;

    a=b;

    b=temp;

}

void sapxep(int a[],int n)

{

   for(int i=0;i<n;i++)

   for(int j=i+1;j<n;j++)

   if(a[i]>a[j] hoanvi(a[i],a[j])

}

+Trong hàm hoán vị này các biến a,b phải được truyền theo kiểu tham biến .Hàm sắp xếp trên sẽ sắp xếp các phần tử theo chìu tăng dần , nếu cần sắp xếp giảm dần thì phép so sánh trong câu lệnh if(a[i]>a[j] sẽ được đổi lại là if(a[i]<a[j]).

## **Tổng kết:**

## **1. Kết quả đạt được:**

**-** Trong quá trình làm bài tập lớn cũng như sau khi làm xong bài tập lớn, chúng em đã rút ra được nhiều bài học cho mình, đó là:

+ Hiểu rõ hơn về bản chất cũng như cách sử dụng hàm, khai báo, kiểm tra, xuất hàm, logic,.. và các thuật toán khác, giúp tụi em có hướng đi rõ ràng hơn, định hướng chính xác hơn đối với môn học này.

+ Cách làm việc nhóm hiệu quả, cùng nhau xây dựng đề tài, từ đó có thể trao đổi, tranh luận, đưa ra những ý kiến để có thể hoàn thành bài tập một cách chỉn chu nhất.

## **2. Mục tiêu thực hiện đề tài này:**

- Nhằm tiết kiệm thời gian trong việc tra cứu, tìm kiếm, sắp xếp thông tin của các tỉnh, thành phố.

- Giúp cho mọi người có thể dễ dàng thấy được thông tin của các tỉnh thành. Từ đó, có thể đưa ra những chính sách điều chỉnh về nhiều mặt để phát triển đất nước, cải thiện đời sống của người dân.

## **Tài liệu tham khảo**

Nguyenvanhieu.vn// Lý thuyết danh sách liên kết đơn, cấu trúc, con trỏ.

Daynhauhoc.com// Bổ sung thêm các kiến thức, ví dụ minh họa.

Vietjack.com// Tham khảo tài liệu, ví dụ của các lý thuyết.

Quantrimang,com// Tham khảo tài liệu.