**Báo Cáo Bài Tập Lớn**

**Mục lục**

**Lời mở đầu.**

1. **Tổng quan về quản lý nhân viên.**
2. **Lợi ích của chương trình quản lý nhân viên.**
3. **Chức năng của chương trình.**
4. **Cơ sở lý thuyết.**
5. **Mảng trong c.**
6. **Ngôn ngữ hướng đối tượng “struct” trong c.**
7. **Đọc ghi file trong c.**
8. **Cấu trúc dữ liệu danh sách liên kết**
9. **Phân tích và xây dựng chủ đề.**
10. **Ý tưởng xây dựng chương trình**
11. **Giải thích code.**
12. **Hướng dẫn sử dụng chương trình**
13. **Kết luận.**

Lời mở đầu

* Lý do chọn đề tài : trong sự phát triển triển của khoa học kỹ thuật ,công nghê thông tin là nghành phát triển nhanh nhất. Công nghệ thông tin nước ta còn mới, song tốc độ phát triển của nó khá nhanh và mạnh.Mọi lĩnh vực đang được ứng dungjtin học hóa rất phổ biến ở nước ta là lĩnh vực quản lý.Việc ứng dụng công nghệ thông tin vào trong quản lý nhân viên là rất cần thiết đối với các công ty, xí nghiệp…Vì vậy nhóm mình chọn đề tài “quản lý nhân viên” làm đề tài nghiên cứu.
* Mục tiêu của đề tài : Xây dựng chương trình quản lý nhân viên bằng ngôn ngữ c.
* Phạm vi nghiên cứu : Phân tích ,xây dựng hệ thống quản lý nhân viên bằng ngôn ngữ c.
* Phương pháp nghiên cứu :

+ Đọc tham khảo các tài liệu, trên cơ sở đó tiến hành phân tích và thiết kế ra bài quản lý nhân viên.

+ Tham khảo , quản sát các phần mềm quản lý trong thực tế.

1. Tổng quan về quản lý nhân viên

1) Lợi ích của chương tình quản lý nhân viên .

- Thông tin về tên nhân viên , danh sách nhân viên , và tìm kiếm nhân viên theo tỉnh thành một cách nhanh chóng và hiệu quả.

2) Chức năng của chương trình .

* Nhập và xuất thông tin của nhiều nhân viên
* Thêm 1 nhân viên
* Tìm kiếm nhân viên theo tỉnh
* Sắp xếp và thống kê nhân viên theo tỉnh
* Ghi vào tập tin nhị phân employee.dat

1. Cơ sở lý thuyết

1)Mang trong c:

Ngôn ngữ lập trình C/C++ cung cấp cấu trúc dữ liệu gọi là **mảng**, được lưu trữ trong một tập hợp các dữ liệu cùng kiểu với độ dài cố định. Một mảng được sử dụng để lưu trữ tập hợp dữ liệu, nhưng nó rất hữu dụng nếu bạn nghĩ về một mảng các biến với cùng một kiểu.

Thay vì khai báo biến một cách rời rạc, như biến so0, so1,… và so99, bạn có thể khai báo một mảng các giá trị như so[0], so[1] và … so[99] để biểu diễn các giá trị riêng biệt. Một thành viên cụ thể của mảng có thể được truy cập qua index (chỉ số).

Tất cả mảng đều bao gồm các vị trí nhớ liền kề nhau. Địa chỉ thấp nhất tương ứng với thành viên đầu tiền và địa chỉ cao nhất tương ứng với thành viên cuối cùng của mảng.

## **Khai báo mảng trong C/C++**

Để khai báo một mảng trong ngôn ngữ C/C++, bạn xác định kiểu của biến và số lượng các phần tử được yêu cầu bởi biến đó như sau:

Kieu Ten\_mang [ Kich\_co\_mang ];

Đây là mảng một chiều. **Kich\_co\_mang** phải là một số nguyên lớn hơn 0 và **Kieu** phải hợp lệ trong ngôn ngữ C/C++. Ví dụ, khai báo một mảng 10 phần tử gọi là balance với kiểu double, sử dụng câu lệnh sau đây:

char sinhvien[10];

## **Khởi tạo mảng trong C/C++**

Bạn có thể khởi tạo mảng trong C/C++ hoặc từng phần tử một hoặc sử dụng một câu lệnh như dưới đây:

int hanghoa[5] = {45, 34, 29, 67, 49};

Số lượng các giá trị trong dấu ngoặc kép {} không được lớn hơn số lượng phần tử khai báo trong dấu ngoặc vuông [].

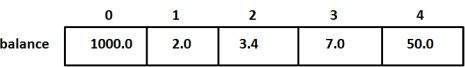
Nếu bạn bỏ sót kích cỡ mảng thì mảng đó đủ lớn để giữ các giá trị được khởi tạo: Bạn sẽ tạo chính xác một chuỗi có giá trị giống hệt chuỗi bên trên bằng cách gán từng phần tử một. Dưới đây là một ví dụ khi gán giá trị cho một phần tử của mảng:

int hanghoa[] = {45, 34, 29, 67, 49};

Bạn có thể tạo ra cùng một mảng giống như đã làm trong ví dụ trước.

hanghoa[4] = 50;

Câu lệnh bên trên gán giá trị thứ 5 của mảng giá trị 50.0. Tất cả các mảng đều có chỉ số (index) đầu tiên bằng 0, đây được gọi là chỉ số cơ bản và phần tử cuối cùng của mảng có chỉ số bằng độ lớn của mảng trừ đi 1. Dưới đây là cách biểu diễn hình họa cho chuỗi khai báo bên trên thông qua chỉ số:



## **Truy cập các phần tử mảng trong C/C++**

Một mảng được truy cập bởi cách đánh chỉ số trong tên của mảng. Dưới đây là một cách truy cập một giá trị của mảng:

int hocphi = hocphik60[55];

Câu lệnh trên lấy phần tử thứ 56 của mảng và gán giá trị này cho biến hocphi. Dưới đây là một ví dụ về việc sử dụng với tất cả mô tả bên trên:

#include <iostream>

using namespace std;

#include <iomanip>

using std::setw;

int main ()

{

int n[ 10 ]; // n la mot mang gom 10 so nguyen

// khoi tao gia tri cac phan tu cua mang n la 0

for ( int i = 0; i < 10; i++ )

{

n[ i ] = i + 100; // thiet lap phan tu tai vi tri i la i + 100

}

cout << "Phan tu thu:" << setw( 13 ) << "Gia tri la:" << endl;

// hien thi gia tri cua moi phan tu

for ( int j = 0; j < 10; j++ )

{

cout << setw( 7 )<< j << setw( 13 ) << n[ j ] << endl;

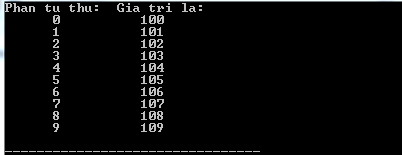
}

return 0;

}

Chương trình này sử dụng hàm **setw(so\_nguyen)** trong C/C++ để định dạng output. Tại đây, tham số **so\_nguyen** là một số chỉ độ rộng của kết quả mà bạn muốn hiển thị. Chẳng hạn, với so\_nguyen là 3 tức là bạn dành 3 vị trí để in kết quả, nếu kết quả cần hiển thị là thừa thì nó sẽ bị cắt bớt, nếu thiếu thì chèn thêm khoảng trống vào. Hàm setw() được dùng cho cả **cout và cin**.

Chạy chương trình C/C++ trên sẽ cho kết quả như hình sau:



## **Chi tiết về mảng trong C/C++**

Mảng là một phần rất quan trọng trong ngôn ngữ C/C++. Dưới đây là những định nghĩa quan trọng liên quan đến một mảng cụ thể mà được trình bày rõ ràng hơn cho các lập trình viên C/C++:

|  |  |
| --- | --- |
| **Khái niệm** | **Miêu tả** |
| [**Mảng đa chiều trong C/C++**](https://vietjack.com/cplusplus/mang_da_chieu_trong_cplusplus.jsp) | C/C++ hỗ trợ các mảng đa chiều. Mẫu đơn giản nhất của mảng này là mảng hai chiều |
| [**Con trỏ tới một mảng trong C/C++**](https://vietjack.com/cplusplus/con_tro_toi_mang_trong_cplusplus.jsp) | Bạn có thể trỏ tới phần tử đầu tiên của mảng một cách đơn giản chỉ bởi xác định tên mảng đó, chứ không phải một chỉ số |
| [**Truyền mảng tới hàm như là tham số trong C/C++**](https://vietjack.com/cplusplus/truyen_mang_toi_ham_trong_cplusplus.jsp) | Bạn có thể truyền tới hàm một điểm trỏ chỉ tới một mảng bởi xác định tên mảng chứ không phải là một chỉ số |
| [**Trả về mảng từ hàm trong C/C++**](https://vietjack.com/cplusplus/tra_ve_mang_tu_ham_trong_cplusplus.jsp) | C/C++ cho phép một hàm có thể trả về một mảng |

Cấu trúc dữ liệu mảng:

Mảng (Array) là một trong các cấu trúc dữ liệu 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 câp 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

## **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**

Output:

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

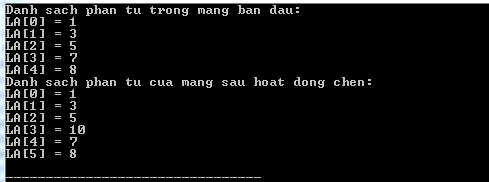
8. Kết thúc

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

[?](https://viettuts.vn/cau-truc-du-lieu-va-giai-thuat/cau-truc-du-lieu-mang)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27 | #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]);     }  } |

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**

Output:

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

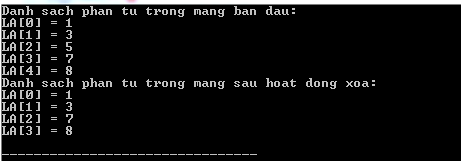
7. Kết thúc

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

[?](https://viettuts.vn/cau-truc-du-lieu-va-giai-thuat/cau-truc-du-lieu-mang)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27 | #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]);     }  } |

Kết quả:



## **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**

Output:

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

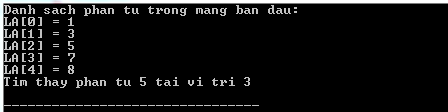
7. Kết thúc

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

[?](https://viettuts.vn/cau-truc-du-lieu-va-giai-thuat/cau-truc-du-lieu-mang)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23 | #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);  } |

Kết quả:



## **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.

Output:

1. Bắt đầu

2. Thiết lập LA[K-1] = ITEM

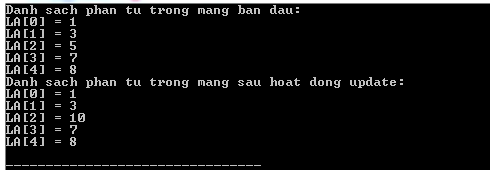
3. Kết thúc

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

[?](https://viettuts.vn/cau-truc-du-lieu-va-giai-thuat/cau-truc-du-lieu-mang)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18 | #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]);     }  } |

Kết quả:



## **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);

}

2)Ngôn ngữ hướng đối tượng “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);

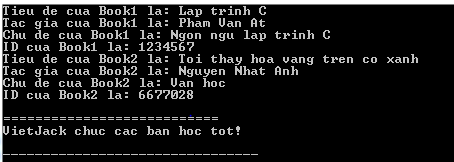
printf("\n===========================\n");

printf("VietJack chuc cac ban hoc tot! \n");

return 0;

}

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



## **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 );

printf("\n===========================\n");

printf("VietJack chuc cac ban hoc tot! \n");

return 0;

}

void inthongtinsach( struct Books book )

{

printf( "Tieu de sach: %s\n", book.tieude);

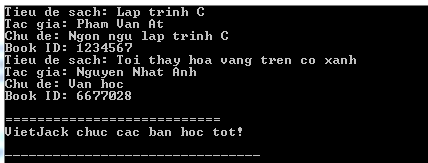
printf( "Tac gia: %s\n", book.tacgia);

printf( "Chu de: %s\n", book.chude);

printf( "Book ID: %d\n", book.id);

}

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



## **Con trỏ tới cấu trúc**

Bạn có thể định nghĩa con trỏ cấu trúc theo cách bạn định nghĩa các loại con trỏ khác như sau:

struct Books \*struct\_pointer;

Bây giờ bạn có thể lưu địa chỉ của biến cấu trúc trong biến con trỏ được định nghĩa ở trên. Để tìm địa chỉ của một biến cấu trúc, đặt toán tử & trước tên cấu trúc như sau:

struct\_pointer = &Book1;

Để truy cập vào thành phần của một structure sử dụng con trỏ tới structure đó, bạn phải sử dụng toán tử -> như sau:

struct\_pointer->tieude;

Bây giờ chúng ta viết lại ví dụ trên sử dụng con trỏ cấu trúc, hy vọng điều này sẽ dễ dàng cho bạn để hiểu khái niệm này:

#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;

/\* in thong tin Book1 bang cach truyen dia chi cua Book1 \*/

inthongtinsach( &Book1 );

/\* in thong tin Book2 bang cach truyen dia chi cua Book2 \*/

inthongtinsach( &Book2 );

printf("\n===========================\n");

printf("VietJack chuc cac ban hoc tot! \n");

return 0;

}

void inthongtinsach( struct Books \*book )

{

printf( "Tieu de sach: %s\n", book->tieude);

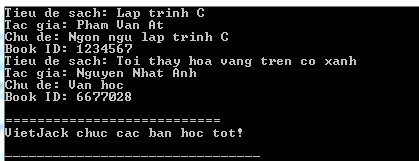
printf( "Tac gia: %s\n", book->tacgia);

printf( "Chu de: %s\n", book->chude);

printf( "Book ID: %d\n", book->id);

}

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



## **Các trường bit (Bit Fields) trong C**

Các **trường bit** cho phép đóng gói dữ liệu trong một cấu trúc. Nó giúp tối ưu hóa bộ nhớ.

C cho phép bạn thực hiện điều này trong một định nghĩa cấu trúc bởi việc đặt: độ dài bit sau biến. Ví dụ:

struct packed\_struct {

unsigned int f1:1;

unsigned int f2:1;

unsigned int f3:1;

unsigned int f4:1;

unsigned int type:4;

unsigned int my\_int:9;

} pack;

3) Đọc ghi file trong c:

* Dữ liệu được lưu ở biến của chương trình, và nó sẽ biến mất khi chương trình kết thúc. Sử dụng file để lưu trữ dữ liệu cần thiết để đảm bảo dữ liệu của chúng ta không bị mất ngay cả khi chương trình của chúng ta ngừng chạy.
* Nếu chương trình của bạn có đầu vào(input) là lớn, bạn sẽ rất vất vả nếu phải nhập mỗi khi chạy. Thay vào đó, hãy lưu vào file và chương trình của bạn sẽ tự đọc mỗi lần khởi chạy
* Dễ dàng sao chép, di chuyển dữ liệu giữa các thiết bị với nhau

## **Các kiểu file**

Trước khi bạn làm việc với file, bạn nên biết về 2 kiểu file khác nhau sau đây:

1. File văn bản – text files
2. File nhị phân – binary file

### **1. File văn bản – text files**

File văn bản là file thường có đuôi là .txt. Những file này bạn có thể dễ dàng tạo ra bằng cách dùng các text editer thông dụng như Notepad, Notepad++, Sublime Text,…

Khi bạn mở các file này bằng các text editer nói trên, bạn sẽ thấy được văn bản ngay và có thể dễ dàng thao tác sửa, xóa, thêm nội dung của file này.

Kiểu file này thuận tiện cho chúng ta trong việc sử dụng hàng ngày, nhưng nó sẽ kém bảo mật và cần nhiều bộ nhớ để lưu trữ hơn.

### **2. File nhị phân – Binary files**

File nhị phân thường có đuôi mở rộng là **.bin**

Thay vì lưu trữ dưới dạng văn bản thuần thúy, các file này được lưu dưới dạng nhị phân, chỉ bao gồm các số 0 và 1. Bạn cũng sẽ thấy các con số này nếu cố mở nó bằng 1 text editer kể trên.

Loại file này giúp lưu trữ được dữ liệu với kích thước lớn hơn, không thể đọc bằng các text editer thông thường và thông tin lưu trữ ở loại file được bảo mật hơn so với file văn bản.

## **Các thao tác với file**

Trong ngôn ngữ lập trình C, có một số thao tác chính khi làm việc với file, bao gồm cả file văn bản và file nhị phân:

1. Tạo mới một file
2. Mở một file đã có
3. Đóng file đang mở
4. Đọc thông tin từ file/ Ghi thông tin ra file

## **Thao tác với file trên ngôn ngữ C**

Khi làm việc với file, bạn cần khai báo 1 con trỏ kiểu FILE. Việc khai báo này là cần thiết để có sự kết nối giữa chương trình của bạn và tập tin mà bạn cần thao tác.

|  |  |
| --- | --- |
| 0  1  2 | FILE \*fptr; |

### **Thao tác mở file**

Để đọc ghi file trong C cũng như trong mọi ngôn ngữ lập trình, việc đầu tiên bạn cần làm là mở file mà bạn 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:

|  |  |
| --- | --- |
| 0  1  2 | fptr = fopen("fileopen","mode") |

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

Ví dụ:

|  |  |
| --- | --- |
| 0  1  2  3  4  5  6 | 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 của bạn 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. |

### **Thao tác đóng file**

Khi làm việc với tập tin hoàn tất, kể cả là file nhị phân hay file văn bản. Bạn cần đóng file sau khi làm việc với nó xong.

Việc đóng file đang mở có thể được thực hiện bằng cách dùng hàm fclose().

|  |  |
| --- | --- |
| 0  1  2 | fclose(fptr); //Con trỏ FILE trỏ tới file cần được đóng. |

## **Đọc/Ghi file văn bản trong C**

Chúng ta sẽ học cách đọc ghi file trong C với file văn bản trước. Với file nhị phân, bạn kéo xuống dưới để xem tiếp.

Để làm việc với file văn bản, chúng ta sẽ sử dụng fprintf() và fscanf().

#### **VD1. Ghi file sử dụng fprintf()**

|  |  |
| --- | --- |
| 0  1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24 | #include <stdio.h>  #include <stdlib.h>    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;  } |

Chương trình nhận số num từ bàn phím và ghi vào file văn bản program.txt.

Sau khi bạn chạy chương trình này, bạn sẽ thấy file văn bản program.txt được tạo mới trong ổ C trên máy tính bạn. Khi mở file này lên, bạn sẽ thấy số mà bạn vừa nhập cho biến num kia.

#### **VD2. Đọc file sử dụng fscanf()**

|  |  |
| --- | --- |
| 0  1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23 | #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;  } |

Chương trình này sẽ đọc giá trị số được lưu trong file program.txt mà chương trình ở VD1 vừa tạo ra và in lên màn hình.

## **Đọc/Ghi file nhị phân trong C**

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**

Để ghi file nhị phân, bạn cần sử dụng hàm fwrite(). Hàm này cần 4 tham số: địa chỉ của biến lưu dữ liệu cần ghi, kích thước của biến lưu dữ liệu đó, số lượng kiểu dữ liệu của biến đó và con trỏ FILE trỏ tới file bạn muốn ghi.

|  |  |
| --- | --- |
| 0  1  2 | fwrite(address\_data,size\_data,numbers\_data,pointer\_to\_file); |

#### **VD3. Ghi file nhị phân sử dụng fwrite()**

|  |  |
| --- | --- |
| 0  1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33 | #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;  } |

Trong VD3 này, chương trình sẽ tạo ra một file program.bin trên ổ đĩa C của bạn. Chương trình này đã khai báo 1 kiểu dữ liệu cấu trúc lưu 3 giá trị số n1, n2, n3; Và nó được sử dụng trong hàm main có tên biến là num.

Trong vòng lặp, các số được ghi vào file sử dụng hàm fwrite(). Các tham số gồm:

* Tham số đầu tiên là địa chỉ của biến num
* Tham số thứ 2 là kích thước của biến num
* Tham số thứ 3 là số lượng kiểu dữ liệu – ở đây là 1.
* Tham số thứ 4 là con trỏ FILE trỏ tới tệp tin program.bin

Cuối cùng, chúng ta đóng file sử dụng fclose().

#### **Đọc file nhị phân**

Hàm fread() cũng có 4 tham số tương tự như hàm fwrite() phía trên.

|  |  |
| --- | --- |
| 0  1  2 | fread(address\_data,size\_data,numbers\_data,pointer\_to\_file); |

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

|  |  |
| --- | --- |
| 0  1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31 | #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;  } |

Trong ví dụ này, bạn đọc file program.bin và lặp qua từng dòng. Bạn sẽ nhận được các giá trị tương ứng khi bạn ghi vào trong VD3.

## **Một số ví dụ về đọc ghi file trong C**

Trong phần này, mình sẽ trình bày 2 ví dụ về đọc ghi file trong C, bao gồm các bài tập đọc ghi file sau:

* Ghi văn bản vào file trong C
* Đọc dữ liệu văn bản từ file trong C

#### **Ghi vào file một câu văn bản**

|  |  |
| --- | --- |
| 0  1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23 | #include <stdio.h>  #include <stdlib.h>  /\* For exit() function \*/  int main()  {     char sentence[1000];     FILE \*fptr;       fptr = fopen("program.txt", "w");     if(fptr == NULL)     {        printf("Error!");        exit(1);     }       printf("Enter a sentence:\n");     gets(sentence);       fprintf(fptr,"%s", sentence);     fclose(fptr);       return 0;  } |

Chạy thử:

|  |  |
| --- | --- |
| 0  1  2  3 | Enter sentence:  I am awesome and so are files. |

#### **Đọc dữ liệu văn bản từ file**

|  |  |
| --- | --- |
| 0  1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23 | #include <stdio.h>  #include <stdlib.h> // For exit() function  int main()  {      char c[1000];      FILE \*fptr;        if ((fptr = fopen("program.txt", "r")) == NULL)      {          printf("Error! opening file");          // Program exits if file pointer returns NULL.          exit(1);      }        // reads text until newline      fscanf(fptr,"%[^\n]", c);        printf("Data from the file:\n%s", c);      fclose(fptr);        return 0;  } |

Giả sử file văn bản program.txt có nội dung như sau:

|  |  |
| --- | --- |
| 0  1  2  3  4 | C programming is awesome.  I love C programming.  How are you doing? |

Chạy thử:

|  |  |
| --- | --- |
| 0  1  2 | Data from the file: C programming is awesome. |

d) Cấu trúc dữ liệu danh sách liên kết :

Một danh sách liên kết (Linked List) là một dãy các cấu trúc dữ liệu bao gồm một nhóm các nút (node) được kết nối với nhau thông qua các liên kết (link) 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.

Output:

NewNode.next −> RightNode;

Hình minh họa như sau:



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

Output:

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.

Output:

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.

Output:

TargetNode.next −> NULL;

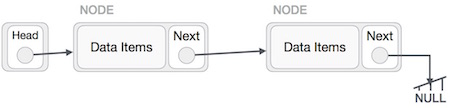


Nếu bạn cần sử dụng nút đã bị xóa này thì bạn có thể giữ chúng trong bộ nhớ, nếu không bạn có thể xóa hoàn toàn hẳn nó khỏi bộ nhớ.

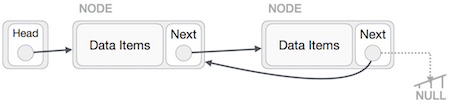


## **Hoạt động đảo ngược Danh sách liên kết**

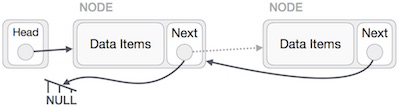
Với hoạt động này, bạn cần phải cẩn thận. Chúng ta cần làm cho nút đầu (head) trỏ tới nút cuối cùng và đảo ngược toàn bộ danh sách liên kết.



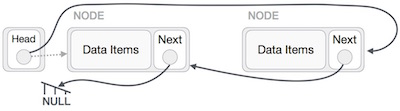
Đầu tiên, chúng ta duyệt tới phần cuối của danh sách. Nút này sẽ trỏ tới NULL. Bây giờ điều cần làm là làm cho nút cuối này trỏ tới nút phía trước của nó.



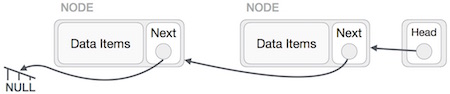
Chúng ta phải đảm bảo rằng nút cuối cùng này sẽ không bị thất lạc, do đó chúng ta sẽ sử dụng một số nút tạm (temp node – giống như các biến tạm trung gian để lưu giữ giá trị). Tiếp theo, chúng ta sẽ làm cho từng nút bên trái sẽ trỏ tới nút trái của chúng.



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.

1. Phân tích và xây dựng chủ đề
2. Ý tưởng để xây dựng chương trình.

### **Kiến thức cần có**

1. Kiến thức lập trình cơ bản
2. Kiến thức về hàm con
3. [Struct trong C](https://nguyenvanhieu.vn/struct-trong-c/)
4. Cách sử dụng cấu trúc lặp: for, do while, while
5. Cách sử dụng cấu trúc điều khiển & rẽ nhánh: if else, switch case
6. Nhập xuất file

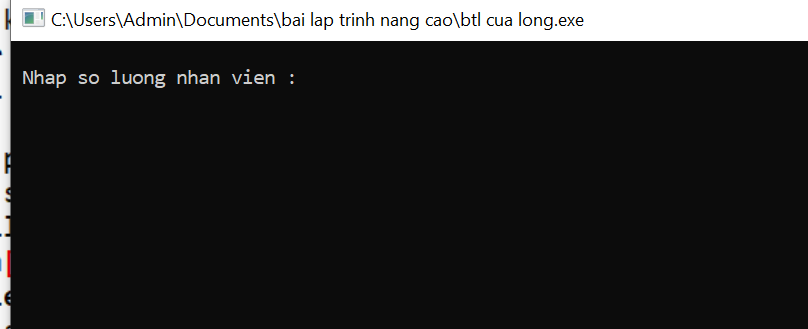
### **Đề bài chương trình quản lý sinh viên**

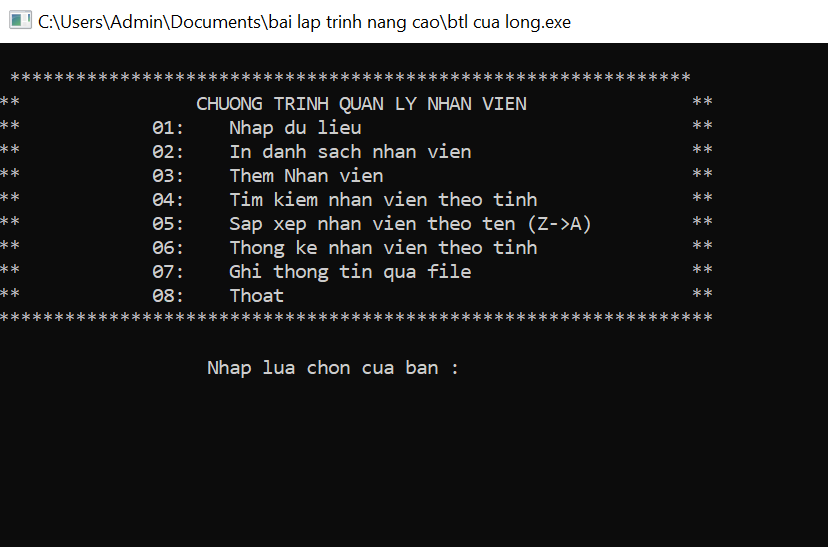
1. Khai báo kiểu dữ liệu SinhVien có các trường họ tên, tỉnh, tuổi, mã số sinh viên.
2. Nhập vào danh sách N sinh viên
3. Xuất danh sách N sinh viên
4. Chèn thêm sinh viên
5. Tìm kiếm nhân viên
6. Sắp xếp và thông kê nhân viên
7. Xuất danh sách N sinh viên ra file
8. Viết chương trình dạng menu cho phép sử dụng các tính năng trên

1. Giải thích code

### 

3) Hướng dẫn sử dụng code

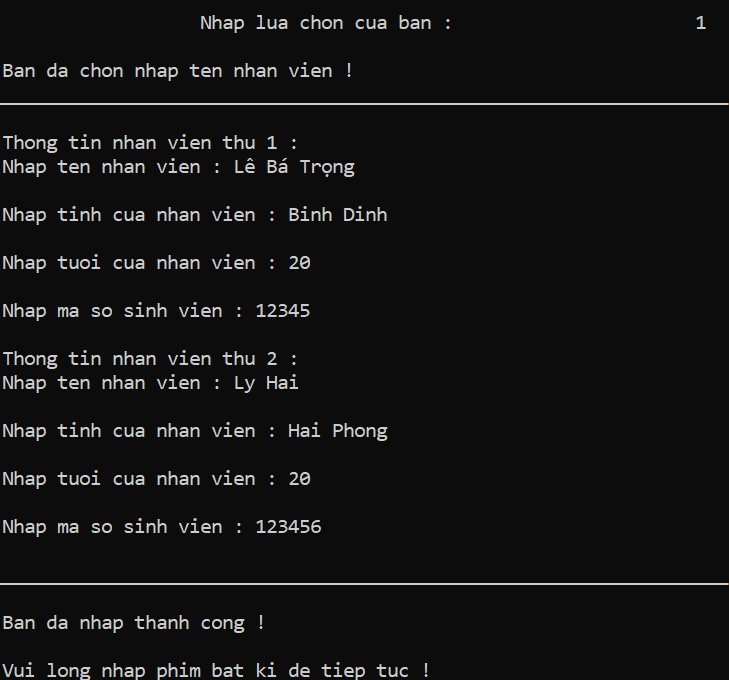
 -Đầu tiên chúng ta cần nhập số lượng nhân viên

hf

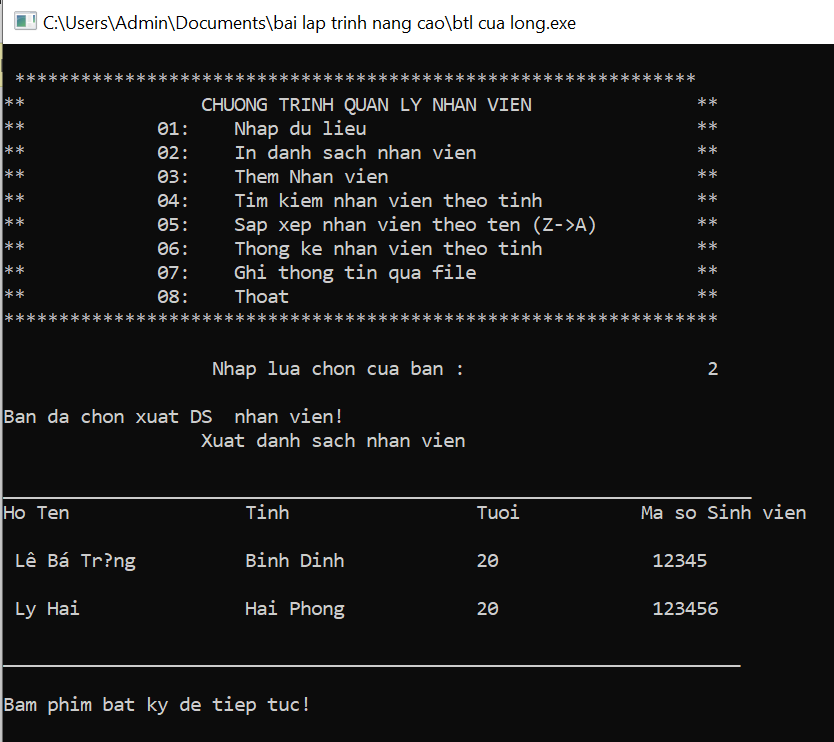
- Sau khi nhập sô lượng nhân viên các chức năng của chương trình đã xuất hiện. sau khi nhập số lượng sinh viên chung ta cần nhập dữ liệu sinh viên, vì nếu không có dữ liệu thì chương trình không thể tiếp tục được, và thông báo không có chức năng này.

- Vì vậy điều đầu tiên chúng ta nên nhập dữu liệu cho chương trình để chuong trình được thực hiên.

- Đầu tiên ta nhập tên nhân viên ,tỉnh ,tuổi và mã số của nhân viên (vd nhập số lượng 2 nhân viên ):

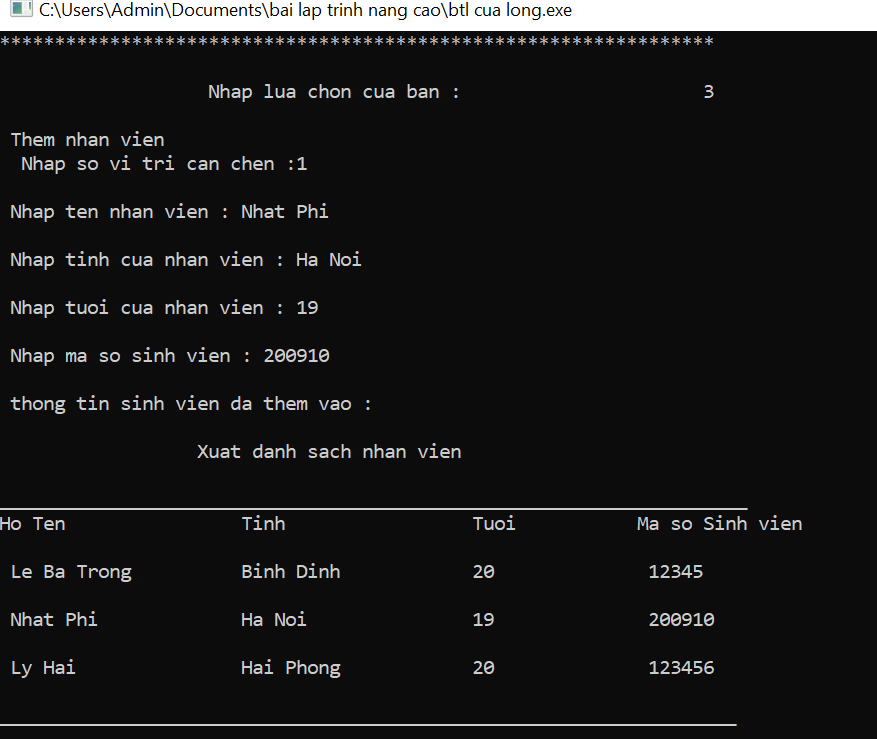


-Sau khi bạn nhập thông tin nó sẽ thông báo thành công và nhấn phím bất khì để tiếp tục.

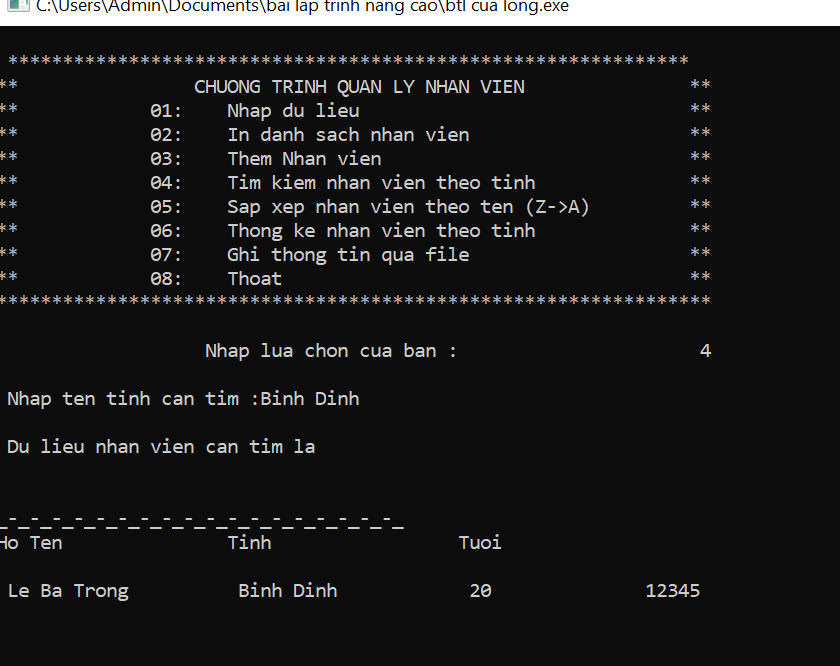


-Sau khi chọn 2 thông tin nhân viên vừa nhập được hiện ra.

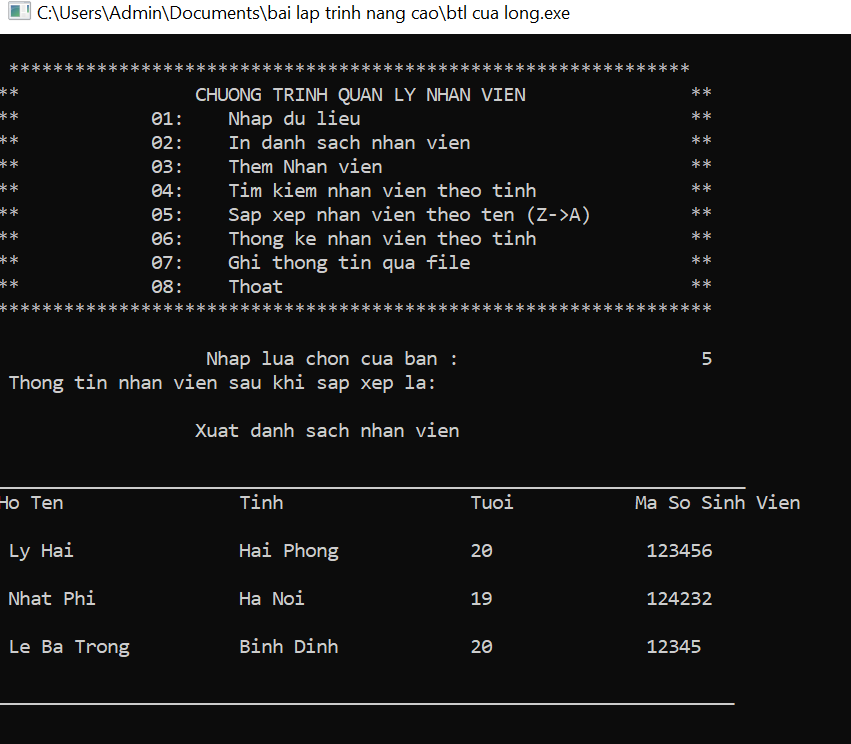
-Chọn 3 : nhập vị trí chèn nhân viên(vị trí cần chèn phải nhỏ hơn số lượng nhân viên vừa nhập), sau đó nó sẽ xuất ra thông tin nhân viên.



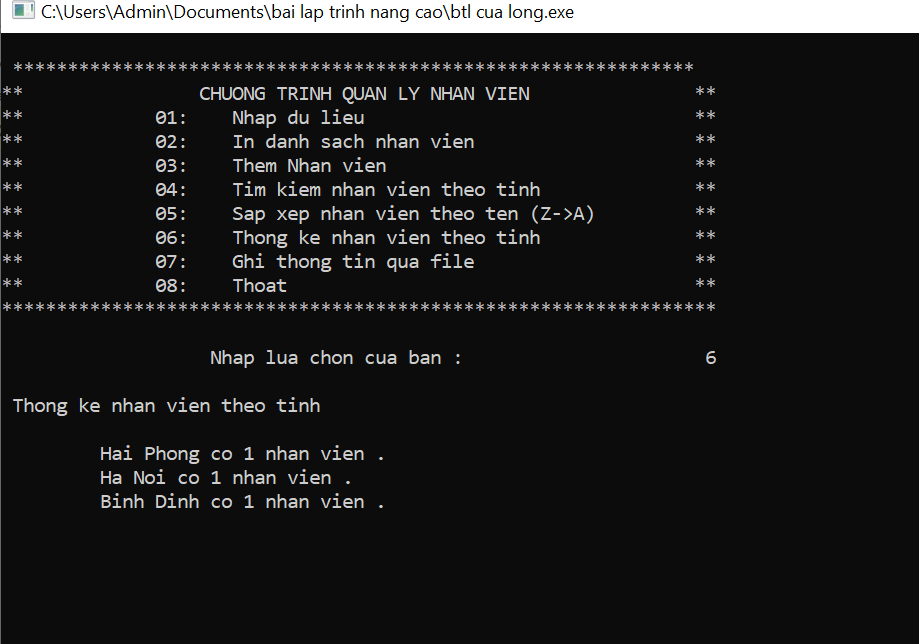
- Nếu muốn tìm kiếm thông tin của nhân viên theo tỉnh nào đó ta chọn số 4 và chương trình sẽ cho ta kết quả thông tin mà ta muốn tìm.



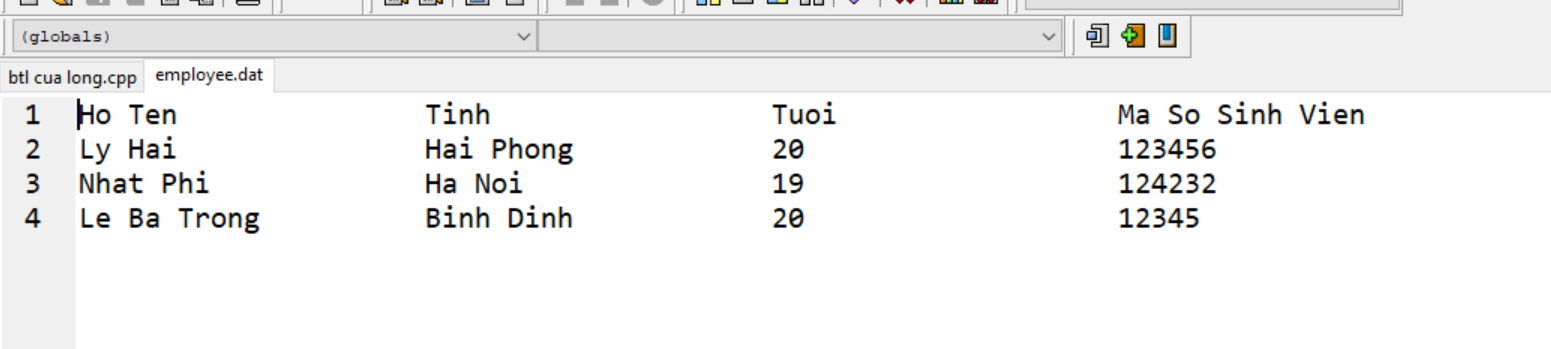
-Chọn 5 để ta sắp xếp nhân viên theo tên.



-Chọn 6: Ta thống kê nhân viên theo tỉnh.



- Chọn 7 : để lưu thông tin nhân viên qua file employ.dat



- Cuối cùng bạn muốn thoát vui lòng nhấn 8.

1. Kết luận

## Kết quả đạt được

* Nhóm chúng em đã hoàn thành xong bai quan lý nhân viên với các chức năng như:

+In ra thông tin nhân viên.

+Chèn thêm nhân viên

+Tìm kiếm nhân viên theo tỉnh

+ Sắp xếp và thống kê nhân viên

+Xuất thông tin qua file

## Nhược điểm

* Chưa vận dụng được danh sách liên kết đơn vào bài tập lớn.
* Không có nhiều ý tưởng còn thiếu xót và khó kiểm soát, phân ông công việc trong quá trình làm bài tập.
* Chưa thực sự sát ý tưởng với thực tế.

## Hướng phát triển

* Nhóm em sẽ cố gắng học tập và trau dồi, kỹ năng làm việc nhóm, phân chia công việc , và học hỏi nhiều hơn nữa.
* Học tập và tìm kiếm thông tin thực tế để giúp bài hoàn thiện hơn.

## Tài liệu tham khảo

* Daynhayhoc.com, Codelearn.io, laptrinhkhongkho.com, topdev.vn, giáo trình tiếng anh Advanced C…

1. Tóm lại

Sau khi học bộ môn CNTT Trường Đại Học Giao Thông Vận Tải Phân Hiệu tại TP Hồ Chí Minh do cô Trần Thị Dung giản dạy nhóm em đã có 1 vài kiến thức cơ bản để làm bài tập lớn.Sau một khoảng thời gian cuối cung nhóm em cũng đã làm xong bài tập lớn, và tụi em cũng chỉ là những người mới không tránh khỏi những sai sót.Kính mong thầy cô cho những nhận xét để hoàn thiện bài hơn. Lời sau cung em xin chúc các thầy cô bộ môn thật nhiều sức khỏe và thành công trên con đường giảng dạy.