**BỘ GIÁO DỤC VÀ ĐÀO TẠO**

**TRƯỜNG ĐẠI HỌC GIAO THÔNG VẬN TẢI**

**PHÂN HIỆU TẠI THÀNH PHỐ HỒ CHÍ MINH**



|  |
| --- |
| **BÁO CÁO BÀI TẬP LỚN** |

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **ĐỀ TÀI: CHƯƠNG TRÌNH QUẢN LÝ NHÂN VIÊN**   |  |  | | --- | --- | | **Giảng viên hướng dẫn:**  **Lớp:** CNTT.K60 | Trần Thị Dung | | **Thành viên:** | **MSSV:** | | Đỗ Thành Chiến | 6051071011 | | Trương Được | 6051071033 | | Cao Lâm Bảo Khanh | 6051071056 | | **Mã học phần:** | CPM215.3 | |
|  |

|  |
| --- |
| Tp.Hồ Chí Minh, năm 2020 |

**LỜI CẢM ƠN**

Sau quá trình học tập và rèn luyện tại môn Lập trình nâng cao trường Đại học Giao thông Vận tải – Phân hiệu tại thành phố Hồ Chí Minh em đã được trang bị các kiến thức cơ bản, các kỹ năng thực tế để có thể hoàn thành đề tài bài tập lớn của chúng em.

Thực tế luôn cho thấy, sự thành công nào cũng đều gắn liền với những sự hỗ trợ, giúp đỡ của những người xung quanh dù cho sự giúp đỡ đó là ít hay nhiều, trực tiếp hay gián. Trong suốt thời gian từ khi bắt đầu làm tiểu luận đến nay, em đã nhận được sự quan tâm, chỉ bảo, giúp đỡ của thầy cô, gia đình và bạn bè xung quanh. Với tấm lòng biết ơn vô cùng sâu sắc, em xin gửi lời cảm ơn chân thành nhất từ đáy long đến quý Thầy Cô của trường…. đã cùng dùng những tri thức và tâm huyết của mình để có thể truyền đạt cho chúng em trong vốn kiến thức quý báu suốt thời gian học tập tại trường.

*Tp. Hồ Chí Minh, ngày tháng năm 2020*

**Mục lục**

[CHƯƠNG 1: TỔNG QUAN VỀ ĐỀ TÀI 1](#_Toc44075562)

[1. Lý do chọn đề tài 1](#_Toc44075563)

[2. Mục đích đề tài 1](#_Toc44075564)

[3. Giới thiệu đề tài 1](#_Toc44075565)

[CHƯƠNG 2: CÁC THUẬT TOÁN TRONG CHƯƠNG TRÌNH 2](#_Toc44075566)

[1. Danh sách liến kết đơn 2](#_Toc44075567)

[1.1 Danh sách liên kết đơn là gì? 2](#_Toc44075568)

[1.2 Khai báo linked list 2](#_Toc44075569)

[1.3 Tạo mới 1 Node 2](#_Toc44075570)

[1.4 Thêm Node vào danh sách liên kết 3](#_Toc44075571)

[1.5 Xóa Node khỏi danh sách liên kết 5](#_Toc44075572)

[2. Thuật toán sắp xếp 7](#_Toc44075573)

[2.1 Thuật toán sắp xếp nổi bọt. 7](#_Toc44075574)

[3. Thuật toán tìm kiếm 9](#_Toc44075575)

[3.1 Thuật toán tìm kiếm nhị phân 9](#_Toc44075576)

[3.2 Thuật toán tìm kiếm tuyến tính. 11](#_Toc44075577)

[4. Thao tác với file trong C 12](#_Toc44075578)

[4.1 Tạo file 12](#_Toc44075579)

[4.2 Thao tác mở file 12](#_Toc44075580)

[4.3 Thao tác đóng file 14](#_Toc44075581)

[4.4 Ghi file nhị phân 14](#_Toc44075582)

[4.5 Đọc file nhị phân 15](#_Toc44075583)

[CHƯƠNG 3: DEMO CODE VÀ HƯỚNG DẪN SỬ DỤNG CHƯƠNG TRÌNH 16](#_Toc44075584)

[1. Demo code 16](#_Toc44075585)

[Link source code: https://github.com/sieusaodienanh/BTBIG 16](#_Toc44075586)

[2. Phân tích và giải thích code 16](#_Toc44075587)

[2.1 Sơ lược về chương trình 16](#_Toc44075588)

[2.2 Nhập danh sách 16](#_Toc44075589)

[3. Hướng dẫn sử dụng chương trình 19](#_Toc44075590)

[CHƯƠNG 4: TÀI LIỆU THAM KHẢO 20](#_Toc44075591)

**Mục lục hình ảnh**

[Hình 1: Mô phỏng của danh sách liên kết đơn 2](#_Toc44075672)

[Hình 2: Thêm Node vào giữa danh sách liên kết 4](#_Toc44075673)

[Hình 3: Xóa node trong danh sách liên kết 6](#_Toc44075674)

[Hình 4: Nhập dữ liệu nhân viên 17](#_Toc44075675)

[Hình 5: Sắp xếp nhân viên theo tỉnh 17](#_Toc44075676)

[Hình 6: Tìm kiếm nhân viên theo tỉnh 18](#_Toc44075677)

[Hình 7: Tìm kiếm nhân viên theo tỉnh 19](#_Toc44075678)

CHƯƠNG 1: TỔNG QUAN VỀ ĐỀ TÀI

1. Lý do chọn đề tài

Quản lí nhân viên là một trong những bộ phận quan trọng trong công ty, đặc biệt là các công ty lớn trong nước và các công ty nước ngoài. Sự thành bài của công ty phụ thuộc vào cách thức tổ chức nhân viên có tốt hay không. Trong năm vừa qua quản lí nhân sự đang đần phát triển mạnh mẽ, không những các công ty nước ngoài mà các công ty tư nhân đang dần nhận thấy tầm quan trọng của cơ cấu tổ chức nhân sự trong công ty.

Dựa vào nhu cầu thực tế của xã hội đòi hỏi con người phải luôn năng động và sáng tạo để tạo ra nhiều sản phẩm cho xã hội. Các công ty luôn muốn phát triển, các hồ sơ tuyển vào sẽ nhiều lên vì vậy đòi hỏi chương trình quản lí hồ sơ nhân viên khi vào công ty.

Mỗi lần muốn tìm hồ sơ của một nhân viên nào đó trong công ty người quản lí nhân sự phải tìm lần lượt hồ sơ nhân viên đó nằm ở đâu. Như vậy mất rất nhiều thời gian, có khi không tìm ra do hồ sơ nhân viên quá nhiều.

2. Mục đích đề tài

Chương trình quản lí nhân viên sẽ là công cụ hỗ trợ đắc lực cho việc quản lí nhân sự. Đi sâu tìm hiểu về cách thức hoạt động của nó giúp chúng ta có thể hiểu rõ hơn về các thuật toán sắp xếp, thống kê, tìm kiếm, danh sách liên kết đơn trong C, và lưu trữ vào file nhị phân.

3. Giới thiệu đề tài

Để hiểu hiểu rõ về chương trình, chúng em xin nói về các chức năng của chương trình:

+ Nhập dữ liệu cho từng nhân viên.

+ Sắp xếp, thống kê và hiển thị thông tin của từng nhân viên theo tỉnh (Z->A).

+ Tìm nhân viên theo tỉnh

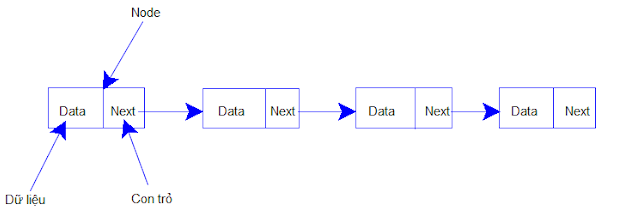
+ Ghi vào tệp tin employee.dat

CHƯƠNG 2: CÁC THUẬT TOÁN TRONG CHƯƠNG TRÌNH

1. Danh sách liến kết đơn

1.1 Danh sách liên kết đơn 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. Đến khi con trỏ mà trỏ tới NULL, đó là phần tử cuối cùng của linked list.



**Hình 1:** Mô phỏng của danh sách liên kết đơn

|  |
| --- |
| struct LinkedList{      int data;      struct LinkedList \*next;  }; |

1.2 Khai báo linked list

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

Khai báo trên sẽ được sử dụng cho mỗi Node trong linked list. Trường data sẽ lưu giữa giá trị và next sẽ là con trỏ để trỏ đến phần tử kế tiếp.

1.3 Tạo mới 1 Node

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

Mỗi một Node khi được khởi tạo chúng ta cần cấp phát bộ nhớ 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 next
* **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, chúng tachỉ cần lưu giữ Node đầu tiên(HEAD). Có head rồi chúng ta có thể đi tới bất cứ Node nào.

1.4 Thêm Node vào danh sách liên kết

### Thêm vào đầu

Việc thêm vào đầu chính là việc cập nhật lại phần tử head. Chúng ta gọi Node mới(temp):

* Nếu head đang trỏ tới NULL, nghĩa là linked list đang trống, Node mới thêm vào sẽ làm phần tử head mới.
* Ngược lại, chúng ta phải thay thế phần tử 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      }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;  } |

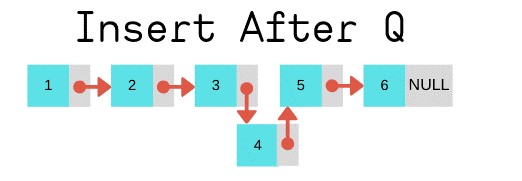
#### **Thêm vào cuối**

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 phần tử head mới
3. Ngược lại, ta sẽ duyệt tới Node cuối cùng(Node có next = NULL), và trỏ next của phần tử 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 phần tử cuối = temp. Khi đó temp sẽ là phần tử cuối(temp- >next = NULL mà)      }      return head;  } |

#### **Thêm vào vị trí bất kì**



**Hình 2:** Thêm Node vào giữa danh sách liên kết

Để làm được việc này, chúng ta phải duyệt từ đầu để tìm tới vị trí của Node cần chèn, giả sử là Node Q:

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

|  |
| --- |
| 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. Chúng 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, chúng ta sẽ mặc định chèn cuối              // Nếu bạn 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;  } |

1.5 Xóa Node khỏi danh sách liên kết

#### **Xóa đầu**

|  |
| --- |
| node DelHead(node head){      if(head == NULL){          printf("\nCha co gi de xoa het!");      }else{          head = head->next;      }      return head;  } |

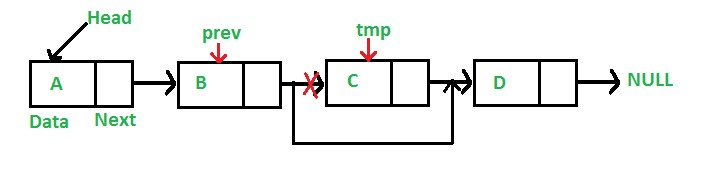
#### **Xóa cuối**

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 có p->next->next = NULL.

#### **Xóa ở vị trí bất kỳ**

Xóa ở vị trí bất kỳ cũng khá giống xóa ở cuối. Đơn giản là chúng ta bỏ qua một phần tử, như ảnh sau: 

**Hình 3:** Xóa node trong danh sách liên kết

|  |
| --- |
| node DelAt(node head, int position){      if(position == 0 || head == NULL || head->next == NULL){          head = DelHead(head); // Nếu vị trí chèn là 0 thì thêm vào đầu      }else{          // Bắt đầu tìm vị trí cần chèn. dùng k để đếm cho vị trí          int k = 1;          node p = head;          while(p->next->next != NULL && k != position){              p = p->next;              ++k;          } |

2. Thuật toán sắp xếp

2.1 Thuật toán sắp xếp nổi bọt.

Thuật toán sắp xếp nổi bọt thực hiện sắp xếp dãy số bằng cách lặp lại công việc đổi chỗ 2 số liên tiếp nhau nếu chúng đứng sai thứ tự(số sau bé hơn số trước với trường hợp sắp xếp tăng dần) cho đến khi dãy số đã được sắp xếp.

## **Ví dụ minh họa**

Cho dãy A gồm các phần tử: 4 5 0 11 8 6 9. Hãy dùng giải thuật sắp xếp nổi bọt để sắp xếp lại dãy đã cho.

Đầu tiên i=0 và j= 6.

Xét aj-1và aj tức là a6 và a5 . Do 6<9 nên không đổi chỗ và j=j-1;

Xét tiếp a5 và a5 . Do 8>6 nên đổi chỗ 6 và 8 được dãy 4 5 0 11 6 8 9. Sau đó j=j-1;

Xét tiếp a4 và a3. Do 11>6 nên đổi chỗ 11 và 6 được dãy 4 5 0 6 11 8 9. Sau đó tiếp tục giảm j=j-1;

Xét tiếp a3 và a2. Do 0<6 nên không đổi chỗ và Sau đó  giảm j=j-1;

Xét tiếp a2 và a1. Do 5>0 nên đổi chỗ 5 và 0 được dãy 4 0 5 6 11 8 9. Sau đó tiếp tục giảm j=j-1;

Xét tiếp a1 và a0. Do 4>0 nên đổi chỗ 4 và 0 được dãy 0 4 5 6 11 8 9. Sau đó tiếp tục giảm j=j-1;

Do j=i=0 nên tăng biến i lên i=i+1=1;

j=6.

Xét a6 và a5. Do 9>8 nên không đổi chỗ và giữ nguyên dãy 0 4 5 6 11 8 9. Sau đó giảm j=j-1;

Xét a5 và a4. Do 11>8 nên đổi chỗ 11 và 8 được dãy 0 4 5 6 8 11 9. Sau đó tiếp tục giảm j=j-1;

Tương tự như trên, giảm j dần về 1. Dãy vẫn giữ nguyên như vậy. Sau đó tăng i lên;

j=6.

Xét a6 và a5. Do 11>9 nên đổi chỗ 11 và 9 được dãy  0 4 5 6  8 9 11. Sau đó giảm j=j-1;

Do lúc này i>n nên dừng giải thuật và nhận dãy 0 4 5 6 8 9 11.

* **Hàm sắp xếp theo giải thuật 2 vòng lặp**

|  |
| --- |
| void Sapxep(int a[], int n)  {  int i, j;  for (int i = 0; i<n - 2; i++)  for (int j = n - 1; j>i; j--)  if (a[j] < a[j - 1])  swap(a[j], a[j - 1]);  } |

* **Hàm swap**

|  |
| --- |
| void swap(int &a, int &b)  {  int c = a;  a = b;  b = c;  } |

* **Hàm main**

|  |
| --- |
| int main()  {  int A[100]; int N;  NhapMang(A, N);  XuatMang(A, N);  Sapxep(A, N);  printf("\nMang sau khi sap xep la: ");  XuatMang(A, N);  getch();  } |

Ở đây, trong hàm bubbleSort chúng ta sử dụng thêm một biến haveSwap để kiểm tra tại lần lặp hiện hành có xảy ra việc đổi chỗ hai số không. Nếu không, chúng ta có thể kết luận mảng đã sắp xếp mà không cần phải thêm một lần lặp nữa.

3. Thuật toán tìm kiếm

3.1 Thuật toán tìm kiếm nhị phân

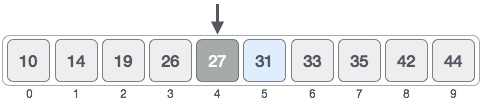
Giả sử chúng ta cần tìm vị trí của giá trị 31 trong một mảng bao gồm các giá trị như hình dưới đây bởi sử dụng Binary Search:



Đầu tiên, chúng ta chia mảng thành hai nửa theo phép toán sau:

chỉ-mục-giữa = ban-đầu + (cuối + ban-đầu)/ 2

Với ví dụ trên là 0 + (9 – 0)/ 2 = 4 (giá trị là 4.5). Do đó 4 là chỉ mục giữa của mảng.



Bây giờ chúng ta so sánh giá trị phần tử giữa với phần tử cần tìm. Giá trị phần tử giữa là 27 và phần tử cần tìm là 31, do đó là không kết nối. Bởi vì giá trị cần tìm là lớn hơn nên phần tử cần tìm sẽ nằm ở mảng con bên phải phần tử giữa.



Chúng ta thay đổi giá trị ban-đầu thành chỉ-mục-giữa + 1 và lại tiếp tục tìm kiếm giá trị chỉ-mục-giữa.

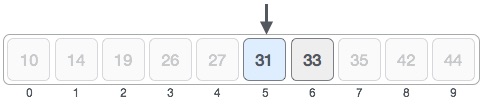
Bây giờ chỉ mục giữa của chúng ta là 7. Chúng ta so sánh giá trị tại chỉ mục này với giá trị cần tìm.



Giá trị tại chỉ mục 7 là không kết nối, và ngoài ra giá trị cần tìm là nhỏ hơn giá trị tại chỉ mục 7 do đó chúng ta cần tìm trong mảng con bên trái của chỉ mục giữa này.



Tiếp tục tìm chỉ-mục-giữa lần nữa. Lần này nó có giá trị là 5.



So sánh giá trị tại chỉ mục 5 với giá trị cần tìm và thấy rằng nó kết nối.



Do đó chúng ta kết luận rằng giá trị cần tìm 31 được lưu giữ tại vị trí chỉ mục 5.

Cho một mảng đã sắp xếp arr[] có n phần tử, viết một hàm tìm kiếm trả về chỉ số của phần tử có giá trị x trong arr[].

* **Hàm tìm kiếm nhị phân**

|  |
| --- |
| int TimkiemNhiphan(int arr[], int a){  int trai = 0;  int n = sizeof(arr);  int phai = (n);  while(trai <= phai){  int giua = (trai+phai)/2;  if(arr[giua]==a)  return giua;  else if(arr[giua]<a)  trai = giua +1;  else  phai = giua -1;  }  return -1;  } |

* **Hàm main**

|  |
| --- |
| int main(){  int arr[]={1,3,4,7,8};  int a,k;  printf("Nhap so can tim ne:");  scanf("%d",&a);  k=TimkiemNhiphan(arr,a);  if(k==-1)  printf ("khong tim thay kkk");  else  printf("\nSo %d o vi tri %d",a,k+1);  } |

3.2 Thuật toán tìm kiếm tuyến tính.

Giải thuật chính của tìm kiếm tuyến tính chính là: So sánh phần tử cần tìm với tất cả các phần tử có trong mảng hoặc danh sách cần tìm. Chạy từ phần tử đầu đến cuối và so sánh từng đôi một, nếu bằng thì thông báo có, ngược lại nếu đã đi hết dãy mà vẫn chưa có phần tử nào thõa mãn thì cho kết quả là không tìm thấy.

**Giải thuật:**

*Bước 1:* Khởi tạo biến i và gán biến i bằng 0.

*Bước 2:* So sánh a[i] với giá trị cầm tìm.

+ Nếu tìm được giá trị a[i] bằng giá trị cần tìm thì dừng lại và dừng. Ngược lại

+  Nếu a[i] khác giá trị cần tìm thì sang bước 3

*Bước 3:*Tăng i lên một đơn vị, nếu i bằng số phần tử trừ 1 của mảng thì dừng lại và cho kết quả là không tìm thấy. Ngược lại quay lại bước 2.

**Ví dụ**: Cho dãy A gồm các phần tử: 11 4 3 9 8 0 2 45. Dùng giải thuật tìm kiếm tuyến tính để tìm xem có phần tử 8 nằm ở trong mảng hay không.

*Bước 1:* gán i=0.

*Bước 2:* so sánh a[0]= 11 != 8. Tăng i lên một đơn vị.

*Bước 3:* i=1 < n-1 (n = 8). Quay lại *Bước 2.*

So sánh a[1] = 4 !=8. Tăng i lên một đơn vị .

Lặp lại *Bước 3:* i=2 < n-1 (n = 8). Quay lại *Bước 2*.

So sánh a[2] = 3 !=8. Tăng i lên một đơn vị .

Lặp lại *Bước 3:* i=3 < n-1 (n = 8). Quay lại *Bước 2*.

So sánh a[3] = 9 !=8. Tăng i lên một đơn vị .

Lặp lại *Bước 3:* i=4 < n-1 (n = 8). Quay lại *Bước 2.*

So sánh a[4] = 8 = 8. Nên kết thúc và tìm được x ở vị trí số a[4].

4. Thao tác với file trong C

4.1 Tạo file

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

FILE \*fptr;

4.2 Thao tác mở file

Để đọc ghi file trong C cũng như trong mọi ngôn ngữ lập trình, việc đầu tiên chúng ta cần làm là mở file 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.

|  |
| --- |
| fptr = fopen("E:\\cprogram\\newprogram.txt","w");    // hoặc    fptr = fopen("E:\\cprogram\\oldprogram.bin","rb"); |

Ví dụ:

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.txt. Với trường hợp này, chương trình chỉ có thể đọc file và không thể ghi nội dung vào file.

* **Giá trị có thể có của tham số 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. |

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

4.3 Thao tác đóng file

Khi làm việc với tập tin hoàn tất, kể cả là file nhị phân hay file văn bản. Chúng ta 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()

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

4.4 Ghi file nhị phân

Để ghi file nhị phân, chúng ta 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 chúng ta muốn ghi.

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

#### **VD3. 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;  } |

Trong VD3 này, chương trình sẽ tạo ra một file program.bin trên ổ đĩa C. Chương trình này đã khai báo 1 kiểu dữ liệu cấu trúc lưu 3 giá trị số n1, n2, n3; 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().

4.5 Đọc file nhị phân

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

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

#### **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;  } |

CHƯƠNG 3: DEMO CODE VÀ HƯỚNG DẪN SỬ DỤNG CHƯƠNG TRÌNH

1. Demo code

Link source code: <https://github.com/sieusaodienanh/BTBIG>

2. Phân tích và giải thích code

2.1 Sơ lược về chương trình

- Chương trình có tên : BAITAPBIG.

- Chương trình sử dụng cấu trúc struct employee\_st để lưu trữ các phần tử liên quan đến nhân viên bao gồm: name, province, year.

- Chương trình sẽ cho phép xử lý các chức năng sau: nhập dữ liệu nhân viên, sắp xếp nhân viên theo tỉnh từ z->a, tìm kiếm nhân viên theo tỉnh, ghi thông tin ra file và cuối cùng là thoát chương trình

2.2 Nhập danh sách

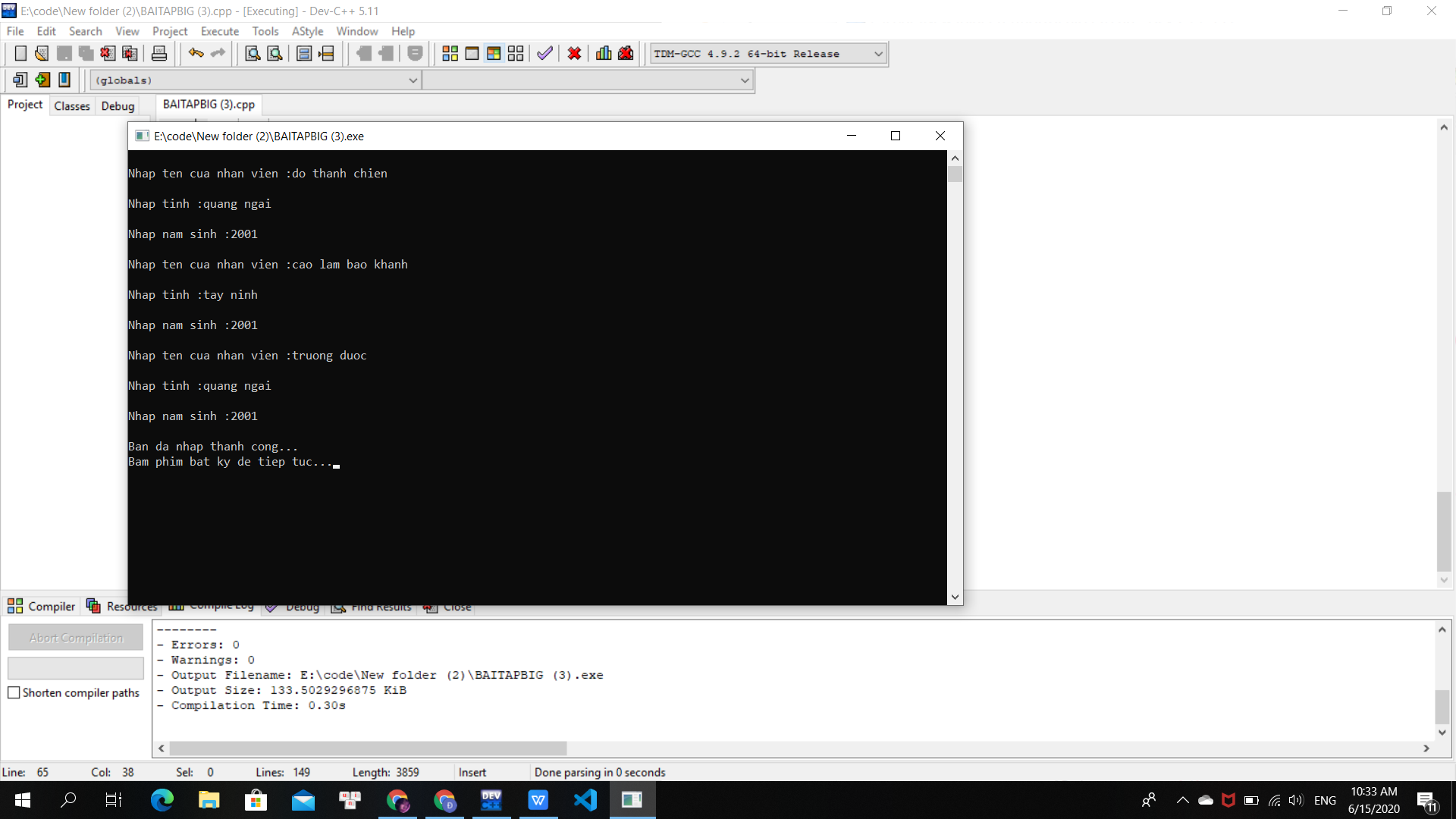
Cho phép người sử dụng nhập vào các thông tin của nhân viên gồm:

- Họ và tên của nhân viên được khai báo kiểu char có tối đa 30 kí tự.

- Tỉnh được khai báo kiểu char có tối đa 30 kí tự

- Năm sinh được khai báo kiểu integer

|  |
| --- |
| **typedef struct** employee\_st{  **char** name[30];  **char** province[30];  **int** year;  }ST; |

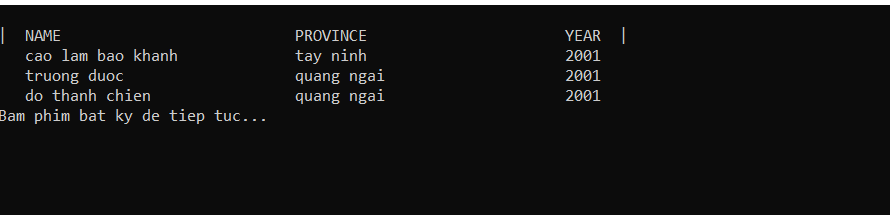


**Hình 4:** Nhập dữ liệu nhân viên

1. **Sắp xếp nhân viên theo tỉnh (z->a).**

|  |
| --- |
| **void** sapxep(ST \*a, **int** n){  system("cls");  **for**(**int** i=0; i<n-1; i++){  **for**(**int** j=i+1; j<n; j++){  **if**(strcmp((a+i)->province,(a+j)->province)<1){  ST k=\*(a+i);  \*(a+i)=\*(a+j);  \*(a+j)=k;  }  }  }  } |

Đặt câu lệnh **if**(strcmp((a+i)->province,(a+j)->province)<1) trong vòng lặp for. Dùng strcmp để so sánh 2 chuỗi kí tự. Ví dụ a là chuỗi 1 , b là chuỗi 2 , và c là chuỗi 3 sau khi chạy chương trình thì nó sẻ sắp xếp lại như sau:c b a.



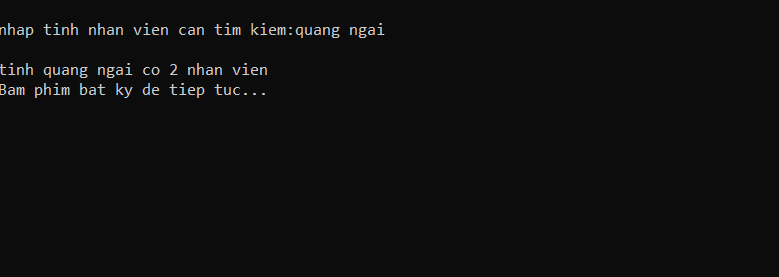
**Hình 5:** Sắp xếp nhân viên theo tỉnh

1. **Tìm kiếm nhân viên theo tỉnh**

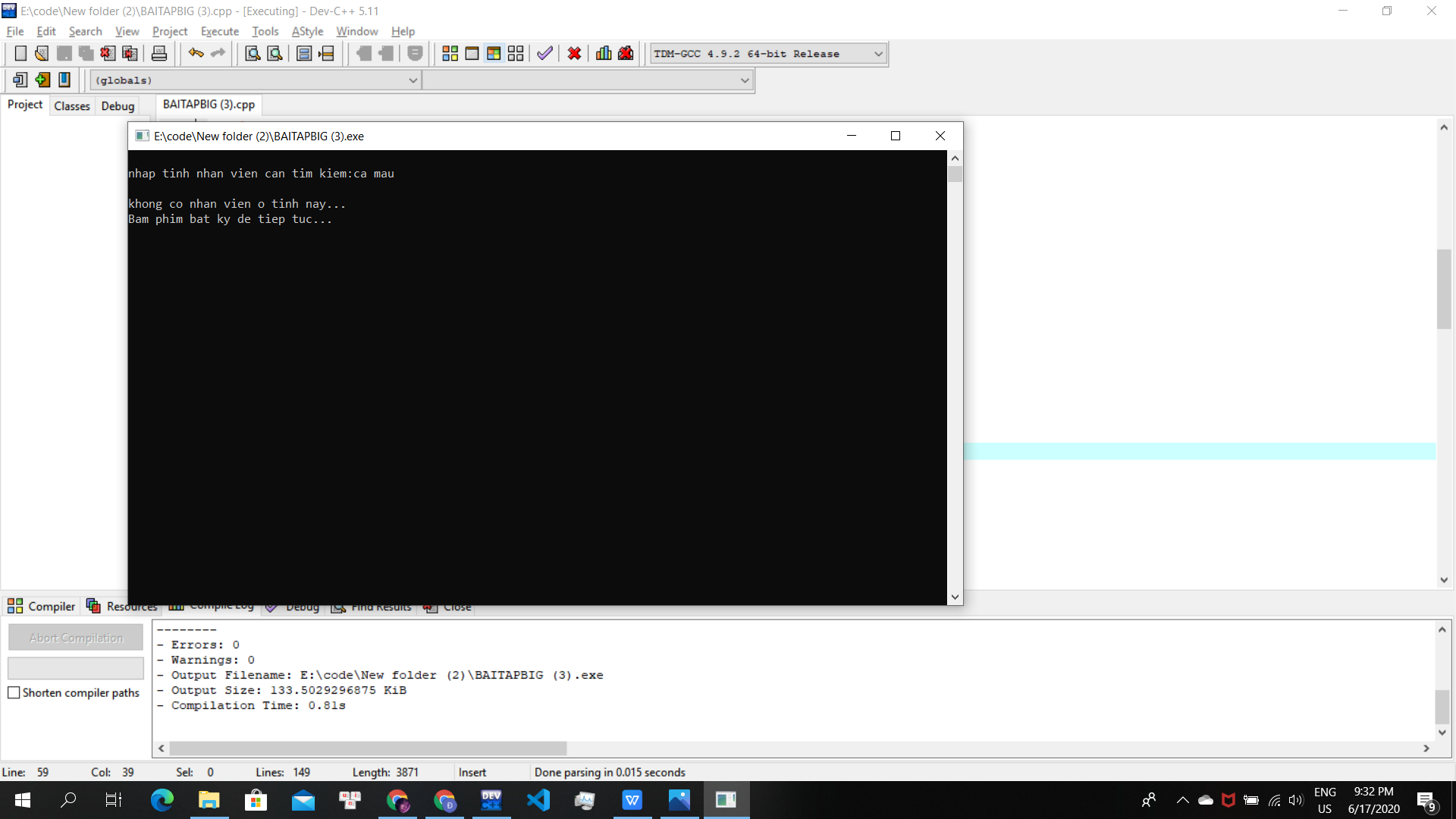
|  |
| --- |
| **int** timkiem(ST \*a, **int** n){  system("cls");  **char** d[20];  **int** dem=0,b=0;  printf("\nnhap tinh nhan vien can tim kiem:");  fflush(stdin);  gets(d);  **for**(**int** i=0; i<n; i++){  **if**(strcmp(d,(a+i)->province)==0){  dem++;  b++;  }  }  **if**(b++){  printf("\ntinh %s co %d nhan vien",d,dem);  }  **else**{  printf("\nkhong co nhan vien o tinh nay...");  }  } |

- Nhập tỉnh cần tìm.

- Đặt câu lệnh **if**(strcmp(d,(a+i)->province)==0) trong vòng lặp for. Dùng lệnh strcmp để so sánh 2 chuỗi. Nếu tỉnh vừa nhập là d,(a+i)==0 thì lệnh kết thúc in kết quả ra màng hình. Nếu như tỉnh vừa nhập là d,(a+i)!=0 thì khi kết thúc chương trình sẻ in ra màng hình không có nhân viên ở tỉnh này.



**Hình 6:** Tìm kiếm nhân viên theo tỉnh



**Hình 7:** Tìm kiếm nhân viên theo tỉnh

1. **Ghi file dạng văn bản**

Bước 1: Khai báo con trỏ kiểu file và dùng hàm fopen() để mở file. Việc khai báo này là cần thiết để có sự kết nối giữa chương trình bạn với tập tin mà bạn thao tác.

Mở file ở chế độ ghi “append”. Tức là nếu file đã tồn tại thì nó sẻ ghi vào cuối nội dung đã có. Nếu file không tồn tại nó sẻ tạo file tự động .

Bước 2 : Gọi lại hàm nhập để nhập thông tin nhân viên vào file employee.dat.

Bước 3 : Đóng chương trình bằng lệnh fclose().

|  |
| --- |
| **void** luufile(ST \*a, **int** n){  system("cls");  **FILE** \* fp;  fp = **fopen** ("employee.dat","a");  fprintf(fp, "\n%-20s%-20s%-20s", "Ho Ten", "Tinh", "Nam sinh");  for(**int** i = 0;i < n;i++){  fprintf(fp,"\n%-20s%-20s%-20d",(a+i)->name,(a+i)->province,(a+i)->year);  }  **fclose** (fp);  } |

3. Hướng dẫn sử dụng chương trình

- Nhập số lượng nhân viên.

- Nhập dữ liệu cho nhân viên nhấn phím 1.

- Sắp xếp nhân viên theo tỉnh nhấn phím 2.

- Tìm kiếm nhân viên theo tỉnh nhấn phím 3.

- Ghi file vào tệp employee.dat nhấn phím 4.

- Thoát chương trình nhấn phím 5.

CHƯƠNG 4: TÀI LIỆU THAM KHẢO

<https://sharecode.vn/source-code/chuong-trinh-quan-ly-nhan-vien-bang-c-17119.htm>

<https://sinhvientot.net/giai-thuat-tim-kiem-tuyen-tinh/>

<https://www.slideshare.net/tuoitrecomvn/slide-5-23855078>

<https://freetuts.net/thuat-toan-sap-xep-noi-bot-trong-php-13.html>

<https://nguyenvanhieu.vn/sap-xep-day-so-tang-dan/>

<https://www.stdio.vn/articles/danh-sach-lien-ket-don-106>

<https://vietjack.com/lap_trinh_c/file_io_trong_c.jsp>