**TRƯỜNG ĐẠI HỌC GIAO THÔNG VẬN TẢI**

**PHÂN HIỆU TẠI TP. HỒ CHÍ MINH**

**BỘ MÔN CÔNG NGHỆ THÔNG TIN**



**BÁO CÁO BÀI TẬP LỚN**

**THIẾT KẾ WEB**

**ĐỀ TÀI: TẠO WEB BÁN HÀNG ĐƠN GIẢN**

Giảng viên hướng dẫn: ThS. TRẦN THỊ DUNG

Sinh viên thực hiện: NGUYỄN NGUYÊN HUY

Lớp : CNTT

Khoá : K63

MSSV: 6351071030

Tp. Hồ Chí Minh, năm 2023

**TRƯỜNG ĐẠI HỌC GIAO THÔNG VẬN TẢI**

**PHÂN HIỆU TẠI TP. HỒ CHÍ MINH**

**BỘ MÔN CÔNG NGHỆ THÔNG TIN**



**BÁO CÁO BÀI TẬP LỚN**

**THIẾT KẾ WEB**

**ĐỀ TÀI: TẠO WEB BÁN HÀNG ĐƠN GIẢN**

Giảng viên hướng dẫn: ThS. TRẦN THỊ DUNG

Sinh viên thực hiện: NGUYỄN NGUYÊN HUY

Lớp : CNTT

Khoá :K63

MSSV: 6351071030

Tp. Hồ Chí Minh, năm 2023

**LỜI CẢM ƠN**

Qua thời gian học tập và rèn luyện tại trường Trường Đại học Giao thông Vận tải phân hiệu tại TP HCM, đến nay em đã được trang bị những kĩ năng, kiến thức cơ bản để có thể hoàn thành được bài tập lớn do giảng viên giao.

Cảm ơn tập thể các thầy cô giáo Bộ môn Công Nghệ Thông Tin và các thầy cô thỉnh giảng đã giảng dạy, luôn quan tâm và không ngần ngại dành thời gian để chỉ bày và giải đáp những thắc mắc của chúng em trong những tiết học và cả những lúc ngoài giờ.

Và cảm ơn thạc sĩ Trần Thị Dung đã luôn quan tâm nhiệt tình hướng dẫn, giúp đỡ chúng em trong quá trình triển khai và thực hiện bài tập lớn. Cô cũng luôn nhắc nhở, giúp đỡ mỗi khi chúng em gặp khó khăn, nhờ vậy mà em đã hoàn thành bài tập lớn của nhóm mình đúng thời hạn được giao. Nếu không có sự hướng dẫn nhiệt tình của cô thì có lẽ chúng em đã khó có thể thực hiện được bài tập đúng theo mong muốn của mình.

Nhóm chúng em đã bỏ ra nhiều thời gian để tìm hiểu và trang bị thêm kiến thức nhằm phục vụ cho việc thực hiện ý tưởng, nhưng chắc chắn rằng nhóm chúng em sẽ không thể tránh khỏi những sai sót không đáng có vì kiến thức còn hạn chế. Chúng em hi vọng rằng sẽ nhận được những lời góp ý quý báu của cô để chúng em có thể hoàn thiện ý tưởng của nhóm một cách tốt nhất có thể.

**NHẬN XÉT CỦA GIẢNG VIÊN HƯỚNG DẪN**

|  |
| --- |
| ***Tp. Hồ Chí Minh, ngày 07 tháng 11 năm 2023***  **Giảng viên hướng dẫn**  **Trần Thị Dung** |

**MỤC LỤC**

[LỜI CẢM ƠN 3](#_Toc9154)

[NHẬN XÉT CỦA GIẢNG VIÊN HƯỚNG DẪN 4](#_Toc16572)

[MỤC LỤC 5](#_Toc13955)

[PhÂN CHIA CÔNG VIỆC](#_Toc8826) 7

[PHẦN I: DỊCH SÁCH 8](#_Toc7262)

[DANH SÁCH 8](#_Toc17485)

[1. Khái niệm danh sách 8](#_Toc24285)

[2. Phương pháp cài đặt danh sách 8](#_Toc7811)

[3. Danh sách liên kết đơn 9](#_Toc7750)

[4. Các loại danh sách khác 11](#_Toc23756)

[CẤU TRÚC NGĂN XẾP(STACK) 12](#_Toc14844)

[1. Giới thiệu stack 12](#_Toc8595)

[2. Cách cài đặt stack 12](#_Toc20750)

[3. Ứng dụng 13](#_Toc9420)

[CẤU TRÚC HÀNG ĐỢI(QUEUE) 17](#_Toc1077)

[1. Giới thiệu 17](#_Toc4732)

[2. Cách cài đặt 18](#_Toc27856)

[3. Ứng dụng 19](#_Toc28237)

[CẤU TRÚC CÂY VÀ CÂY NHỊ PHÂN(TREE AND BINARY TREE) 20](#_Toc6295)

[1. Cây 20](#_Toc11495)

[1.1 Giới thiệu 20](#_Toc6178)

[1.2 Các thuật ngữ cơ bản trong cấu trúc dữ liệu cây: 21](#_Toc30947)

[2. Cây nhị phân 22](#_Toc24185)

[2.1 Các thuộc tính của cây nhị phân 23](#_Toc2200)

[2.2 Cài đặt cây nhị phân 23](#_Toc25157)

[3. Ứng dụng của cây 24](#_Toc13654)

[MỘT SỐ THUẬT TOÁN 25](#_Toc11679)

[1. Sắp xếp (Sorting) 25](#_Toc27393)

[1.1. Sắp xếp chọn (Selection sort) 25](#_Toc21211)

[1.2. Sắp xếp nổi bọt (Bubble Sort) 26](#_Toc9151)

[1.3. Sắp xếp chèn (Insertion sort) 27](#_Toc7787)

[1.4. Sắp xếp nhanh (Quick Sort) 27](#_Toc4681)

[1.5. Sắp xếp trộn (Merge sort) 30](#_Toc18237)

[1.6. Sắp xếp vun đống (Heap Sort) 32](#_Toc555)

[2. Tìm kiếm (Search) 33](#_Toc25008)

[2.1. Tìm kiếm tuyến tính (Linear Search) 33](#_Toc31018)

[2.2. Tìm kiếm nhị phân 34](#_Toc7120)

[PHẦN 2: CHƯƠNG TRÌNH 35](#_Toc5902)

[I. Lý do chọn đề tài quản lý 35](#_Toc24687)

[II. Mô tả bài toán 35](#_Toc25833)

[III. Giao diện chương trình 36](#_Toc16342)

[1. Giao diện menu chính của chương trình 36](#_Toc6003)

[2. Chức năng thêm sinh viên 37](#_Toc13089)

[3. Chức năng hiển thị sinh viên hiện có: 38](#_Toc9588)

[4. Chức năng cập nhật thông tin sinh viên 39](#_Toc14228)

[5. Chức năng tìm kiếm sinh viên 41](#_Toc23125)

[6. Chức năng sắp xếp sinh viên 43](#_Toc26977)

[7. Chức năng xóa sinh viên 44](#_Toc3655)

[8. Chức năng thống kê sinh viên 46](#_Toc16812)

[9. Chức năng đọc & ghi file 48](#_Toc14503)

[10. Chức năng so sánh thời gian chạy của các thuật toán 49](#_Toc31205)

[PHẦN 3. KẾT QUẢ VÀ KIẾN NGHỊ 53](#_Toc30348)

[1. Kết quả đạt được 53](#_Toc26278)

[2. Kiến nghị 54](#_Toc11928)

[PHỤ LỤC 55](#_Toc28338)

[TÀI LIỆU THAM KHẢO 56](#_Toc1158)

**PHÂN CHIA CÔNG VIỆC**

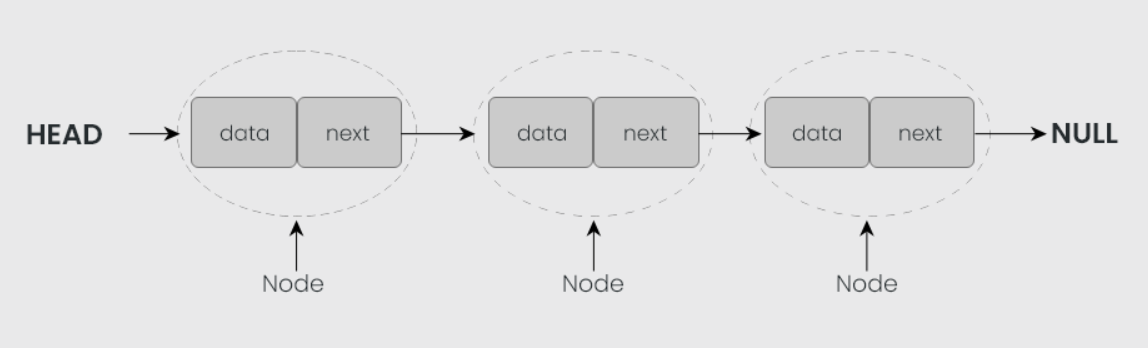
|  |  |
| --- | --- |
| **Sinh viên** | **Công việc** |
| Hà Văn Dũng  (50%) | * Thêm thông tin sinh viên * Cập nhập thông tin sinh viên * Sắp xếp sinh viên tăng dần * Xử lý đọc file, ghi file |
| Nguyễn Nguyên Huy  (50%) | * Xuất thông tin sinh viên * Xóa sinh viên * Sắp xếp sinh viên giàm dần * Tìm kiếm sinh viên * Thống kê sinh viên |

**PHẦN I: DỊCH SÁCH**

**DANH SÁCH**

1. **Khái niệm danh sách**

Danh sách liên kết là cấu trúc dữ liệu tuyến tính, trong đó các phần tử không được lưu trữ ở các vị trí bộ nhớ liền kề. Các phần tử trong danh sách liên kết được liên kết bằng cách sử dụng con trỏ.



1. **Phương pháp cài đặt danh sách**

Cài đặt danh sách bằng mảng tĩnh:

|  |
| --- |
| class ArrayList{ public:  static const int MAX\_SIZE = 100;  ArrayList(); // Khởi tạo danh sách  // … private:  int element[MAX\_SIZE]; // Khai báo mảng có tối đa 100 phần tử  int last; // Biến chỉ phần tử hiện tại }; |

Cài đặt danh sách bằng mảng động:

|  |
| --- |
| class ArrayList{ public:  ArrayList() {               size = 0;               element = nullptr;               last = -1;             }           // ... private:  int\* element; // Khai báo mảng có tối đa 100 phần tử  int last; // Biến chỉ phần tử hiện tại   int size // Biến kích thước của mảng  }; |

1. **Danh sách liên kết đơn**

Danh sách liên kết đơn là cấu trúc dữ liệu tuyến tính trong đó các phần tử không được lưu trữ ở các vị trí bộ nhớ liền kề và mỗi phần tử chỉ được kết nối với phần tử tiếp theo bằng con trỏ.

Danh sách liên kết đơn là một danh sách các Node, mỗi Node gồm 2 thành phần:

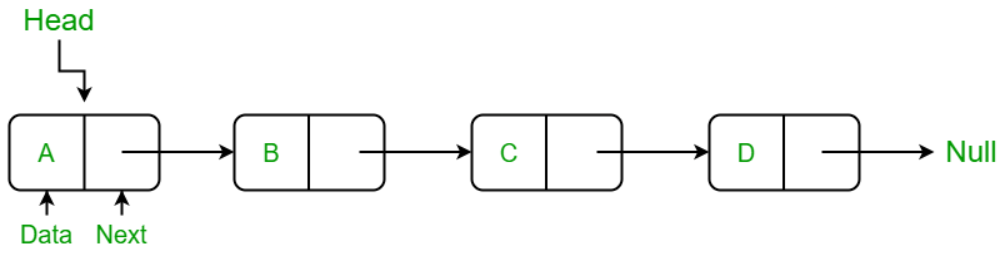
* Phần chứa dữ liệu – data
* Phần chỉ vị trí (địa chỉ) của phần tử tiếp theo trong danh sách – next. Next là con trỏ trỏ đến Node kế tiếp

|  |
| --- |
| struct Node{   Data;    Node\* next; } |

Tùy thuộc vào đề bài mà data có kiểu dữ liệu khác nhau, có thể là int, float, hoặc một kiểu dữ liệu được định nghĩa khác . Một node có thể có nhiều hơn một data.

Xác định danh sách liên kết đơn bằng địa chỉ của Node đầu tiên trong danh sách:

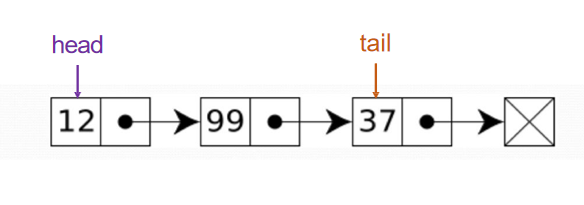
* Gọi biến lưu địa chỉ này là con trỏ đầu head
* Khởi tạo danh sách rỗng: Node \*head = NULL;



Có thể sử dụng thêm con trỏ cuối để làm việc với danh sách dễ dàng hơn:

* Khởi tạo danh sách rỗng: Node\* head = tail = NULL;

|  |
| --- |
| class LinkedList {   public:     LinkedList();     // ...   private:      Node\* head;     Node\* tail; }; |



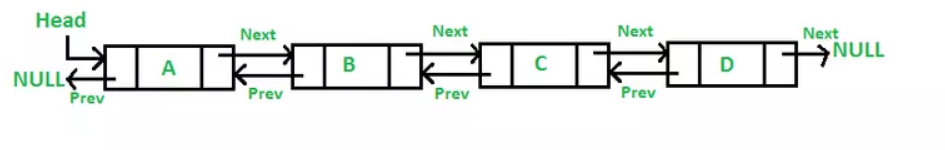
Các phép toán mẫu hàmvà cơ bản trên danh sách liên kết đơn:

* Tạo Node mới : creatNode(value)
* Thêm các Node vào danh sách
* Xóa Node ra khỏi danh sách
* ….

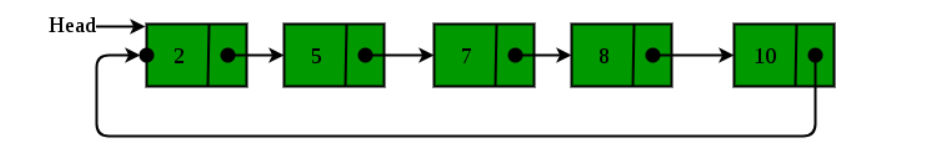
|  |
| --- |
| class LinkedList {    public:     LinkedList();       void append(int value); //Hàm thêm đầu       void prepend(int vlaue);// Hàm thêm cuối  void removeLast(); // Hàm xóa cuối  void insert(int value, int locus); // Hàm thêm vị trí bất kỳ       void remove(int locus); // Hàm xóa vị trí bất kỳ       void print(); // Hàm in ra các phần tử       void removeFist(); // Hàm xóa phần tử đầu   private:        Node\* head;  Node\* tail; }; |

1. **Các loại danh sách khác**

* Danh sách liên kết đôi:

****

* Danh sách liên kết vòng:

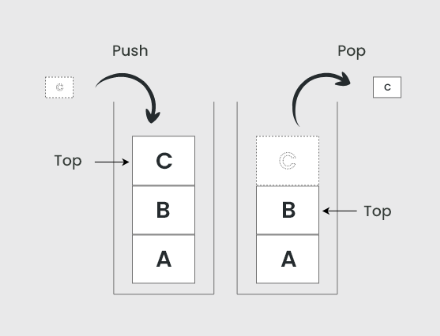
****

Ngoài ra còn một số loại danh sách liên kết khác như danh sách liên kết đôi tròn, danh sách liên kết tiêu đề. Nhưng thường sử dụng phổ biến danh sách liên kết đơn, danh sách liên kết đôi và danh sách liên kết vòng.

**CẤU TRÚC NGĂN XẾP(STACK)**

1. **Giới thiệu stack**

Ngăn xếp là một cấu trúc dữ liệu tuyến tính tuân theo một thứ tự cụ thể trong đó các hoạt động được thực hiện. Thứ tự có thể là LIFO(Last In First Out) hoặc FILO(First In Last Out). LIFO ngụ ý rằng phần tử được chèn cuối cùng sẽ xuất hiện trước và FILO ngụ ý rằng phần tử được chèn trước sẽ xuất hiện sau cùng.



1. **Cách cài đặt stack**

|  |
| --- |
| class Stack{ private:  Node \*top;  int size; public:  Stack();  // …  }; |

Stack có thể được định ngĩa thông qua một mảng hoặc danh sách liên kết.

Các loại ngăn xếp:

Ngăn xếp đăng ký : Loại ngăn xếp này cũng là một thành phần bộ nhớ có trong đơn vị bộ

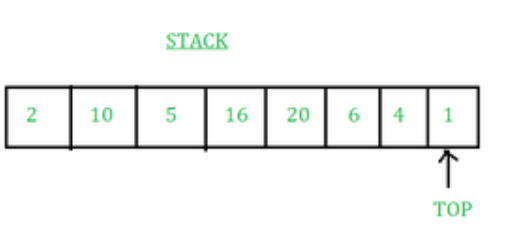
nhớ và chỉ có thể xử lý một lượng nhỏ dữ liệu. Chiều cao của ngăn xếp thanh ghi luôn bị

giới hạn vì kích thước của ngăn xếp thanh ghi rất nhỏ so với bộ nhớ.

Ngăn xếp bộ nhớ : Loại ngăn xếp này có thể xử lý một lượng lớn dữ liệu bộ nhớ. Chiều cao của ngăn xếp bộ nhớ rất linh hoạt vì nó chiếm một lượng lớn dữ liệu bộ nhớ.

**Top of the Stack nghĩa:**

Khi một phần tử mới được thêm vào ngăn xếp, nó sẽ được đặt lên trên các phần tử hiện có. Tương tự, khi một phần tử bị xóa khỏi ngăn xếp thì phần tử trên cùng sẽ bị xóa trước tiên. Phần trên cùng của ngăn xếp luôn là phần tử hiện có thể truy cập được để xem hoặc thao tác.Con trỏ qua đó các phần tử được truy cập, chèn và xóa trong ngăn xếp được gọi là đỉnh ngăn xếp . Nó là con trỏ tới phần tử trên cùng của ngăn xếp.



1. **Ứng dụng**

Chuyển biểu thức từ trung tố sang hậu tố và tính giá trị biểu thức hậu tố:

Chuyển biểu thức exp = A\*(B+C) sang dạng hậu tố

Khởi tạo: output = []

stack s = NULL

|  |  |  |  |
| --- | --- | --- | --- |
| **i** | **exp[i]** | **output** | **S** |
| 0 | A | A | NULL |
| 1 | \* | A | \* |
| 2 | ( | A | (\* |
| 3 | B | AB | (\* |
| 4 | + | AB | +(\* |
| 5 | C | ABC | +(\* |
| 6 | ) | ABC+ | \* |

output = ABC+\*

Chuyển biểu thức dạng trung tố sang tiền tố và tính giá trị biểu thức tiền tố:

Chuyển biểu thức exp = A\*B+C/D sang dạng tiền tố

output = []

stack s = NULL

rev = )C+B(\*A

rev = (C+B)\*A

|  |  |  |  |
| --- | --- | --- | --- |
| **i** | **exp[i]** | **output** | **S** |
| 0 | ( | [] | ( |
| 1 | C | C | ( |
| 2 | + | C | +( |
| 3 | B | CB | +( |
| 4 | ) | CB+ |  |
| 5 | \* | CB+ | \* |
| 6 | A | CB+A | \* |

output = \*A+BC

**Ứng dụng cấu trúc dữ liệu ngăn xếp:**

* Gọi hàm và đệ quy: Khi một hàm được gọi, trạng thái hiện tại của chương trình sẽ được đẩy lên ngăn xếp. Khi hàm trả về, trạng thái sẽ được lấy ra khỏi ngăn xếp để tiếp tục thực thi hàm trước đó.
* Hoạt động hoàn tác/làm lại: Tính năng hoàn tác làm lại trong các ứng dụng khác nhau sử dụng ngăn xếp để theo dõi các hành động trước đó. Mỗi khi một hành động được thực hiện, nó sẽ được đẩy vào ngăn xếp. Để hoàn tác hành động, phần tử trên cùng của ngăn xếp được lấy ra và thao tác ngược lại được thực hiện.
* Đánh giá biểu thức: Cấu trúc dữ liệu ngăn xếp được sử dụng để đánh giá các biểu thức trong các ký hiệu trung tố, hậu tố và tiền tố. Các toán tử và toán hạng được đẩy vào ngăn xếp và các thao tác được thực hiện dựa trên các phần tử trên cùng của ngăn xếp.
* Lịch sử trình duyệt: Trình duyệt web sử dụng ngăn xếp để theo dõi các trang web bạn truy cập. Mỗi lần bạn truy cập một trang mới, URL sẽ được đẩy lên ngăn xếp và khi bạn nhấn nút quay lại, URL trước đó sẽ được bật ra khỏi ngăn xếp.
* Dấu ngoặc đơn cân bằng: Cấu trúc dữ liệu ngăn xếp được sử dụng để kiểm tra xem dấu ngoặc đơn có cân bằng hay không. Dấu ngoặc đơn mở được đẩy vào ngăn xếp và dấu ngoặc đơn đóng được lấy ra khỏi ngăn xếp. Nếu ngăn xếp trống ở cuối biểu thức thì dấu ngoặc đơn được cân bằng.
* Thuật toán quay lui: Thuật toán quay lui sử dụng các ngăn xếp để theo dõi các trạng thái của quá trình giải quyết vấn đề. Trạng thái hiện tại được đẩy lên ngăn xếp và khi thuật toán quay lui, trạng thái trước đó sẽ được đưa ra khỏi ngăn xếp.

**Ứng dụng Stack trong đời sống:**

* Giá đỡ CD/DVD.
* Chồng sách trong một hiệu sách.
* Hệ thống trung tâm cuộc gọi.
* Cơ chế Undo và Redo trong soạn thảo văn bản.
* Lịch sử của trình duyệt web được lưu trữ dưới dạng ngăn xếp.
* Nhật ký cuộc gọi, E-mail và ảnh Google trong bất kỳ thư viện nào cũng được lưu trữ dưới dạng ngăn xếp.
* Các nội dung tải xuống và Thông báo trên YouTube cũng được hiển thị ở định dạng LIFO (thông báo mới nhất xuất hiện trước).
* Phân bổ bộ nhớ bởi hệ điều hành trong khi thực hiện một tiến trình.

**Ưu điểm của ngăn xếp:**

* Triển khai dễ dàng: Cấu trúc dữ liệu ngăn xếp dễ triển khai bằng cách sử dụng mảng hoặc danh sách liên kết, đồng thời các hoạt động của nó rất đơn giản để hiểu và thực hiện.
* Sử dụng bộ nhớ hiệu quả : Stack sử dụng một khối bộ nhớ liền kề, giúp việc sử dụng bộ nhớ hiệu quả hơn so với các cấu trúc dữ liệu khác.
* Thời gian truy cập nhanh: Cấu trúc dữ liệu ngăn xếp cung cấp thời gian truy cập

nhanh để thêm và xóa các phần tử khi các phần tử được thêm và xóa khỏi đầu ngăn

xếp.

* Trợ giúp trong các lệnh gọi hàm: Cấu trúc dữ liệu ngăn xếp được sử dụng để lưu trữ các lệnh gọi hàm và trạng thái của chúng, giúp thực hiện hiệu quả các lệnh gọi hàm đệ quy.
* Hỗ trợ quay lui: Cấu trúc dữ liệu ngăn xếp hỗ trợ các thuật toán quay lui, được sử dụng trong giải quyết vấn đề để khám phá tất cả các giải pháp có thể có bằng cách lưu trữ các trạng thái trước đó.
* Được sử dụng trong Thiết kế trình biên dịch: Cấu trúc dữ liệu ngăn xếp được sử dụng trong thiết kế trình biên dịch để phân tích cú pháp và phân tích cú pháp của các ngôn ngữ lập trình.
* Cho phép các hoạt động hoàn tác/làm lại : Cấu trúc dữ liệu ngăn xếp được sử dụng để cho phép các hoạt động hoàn tác và làm lại trong các ứng dụng khác nhau như trình soạn thảo văn bản, công cụ thiết kế đồ họa và môi trường phát triển phần mềm.

**Nhược điểm của ngăn xếp:**

* Dung lượng hạn chế: Cấu trúc dữ liệu ngăn xếp có dung lượng hạn chế vì nó chỉ có thể chứa một số phần tử cố định. Nếu ngăn xếp đầy, việc thêm phần tử mới có thể dẫn đến tràn ngăn xếp, dẫn đến mất dữ liệu.
* Không có quyền truy cập ngẫu nhiên: Cấu trúc dữ liệu ngăn xếp không cho phép truy cập ngẫu nhiên vào các phần tử của nó và nó chỉ cho phép thêm và xóa các phần tử khỏi đầu ngăn xếp. Để truy cập một phần tử ở giữa ngăn xếp, tất cả các phần tử phía trên nó phải được loại bỏ.
* Quản lý bộ nhớ: Cấu trúc dữ liệu ngăn xếp sử dụng một khối bộ nhớ liền kề, điều này có thể dẫn đến phân mảnh bộ nhớ nếu các phần tử được thêm và xóa thường xuyên.
* Không phù hợp với một số ứng dụng nhất định: Cấu trúc dữ liệu ngăn xếp không phù

hợp với các ứng dụng yêu cầu truy cập các phần tử ở giữa ngăn xếp, như thuật toán

tìm kiếm hoặc sắp xếp.

* Tràn ngăn xếp và tràn ngăn xếp : Cấu trúc dữ liệu ngăn xếp có thể dẫn đến tràn ngăn xếp nếu có quá nhiều phần tử được đẩy lên ngăn xếp và có thể dẫn đến tràn ngăn xếp nếu có quá nhiều phần tử được đưa ra khỏi ngăn xếp.
* Hạn chế của lệnh gọi hàm đệ quy: Mặc dù cấu trúc dữ liệu ngăn xếp hỗ trợ các lệnh gọi hàm đệ quy, nhưng quá nhiều lệnh gọi hàm đệ quy có thể dẫn đến tràn ngăn xếp, dẫn đến việc chấm dứt chương trình.

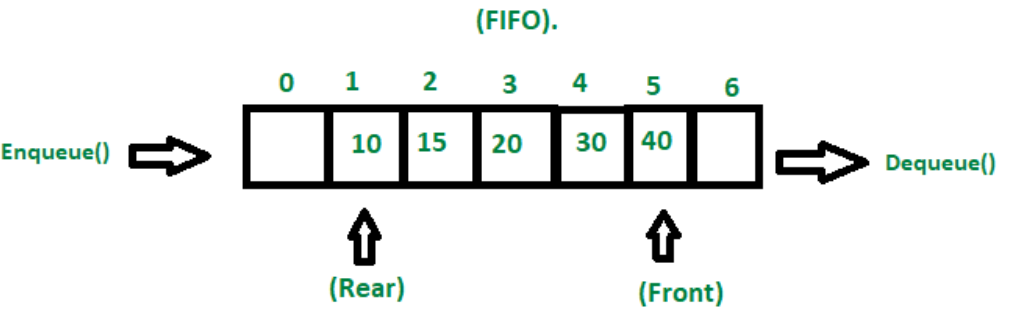
**CẤU TRÚC HÀNG ĐỢI(QUEUE)**

1. **Giới thiệu**

Hàng đợi là một cấu trúc dữ liệu tuyến tính mở ở cả hai đầu và các thao tác được thực hiện theo thứ tự Vào trước ra trước (FIFO). Là một danh sách trong đó tất cả các thao tác thêm vào danh sách được thực hiện ở một đầu và tất cả việc xóa khỏi danh sách được thực hiện ở đầu kia. Phần tử được đẩy vào thứ tự đầu tiên, thao tác xóa được thực hiện đầu tiên trên phần tử đó.

Nguyên tắc hàng đợi FIFO :

* Hàng đợi giống như hàng đợi mua vé, trong đó người đầu tiên xếp hàng là người được phục vụ đầu tiên. (tức là ai đến trước được phục vụ trước).
* Vị trí của mục trong hàng đợi sẵn sàng được phục vụ, nghĩa là mục đầu tiên sẽ bị xóa khỏi hàng đợi, được gọi là mặt trước của hàng đợi (đôi khi là đầu hàng đợi), tương tự như vậy, vị trí của mục cuối cùng trong hàng đợi, tức là hàng được thêm gần đây nhất, được gọi là phía sau (hoặc đuôi ) của hàng đợi. Xem hình dưới đây.



Đặc điểm của hàng đợi:

* Hàng đợi có thể xử lý nhiều dữ liệu.
* Chúng ta có thể truy cập cả hai đầu.
* Họ nhanh chóng và linh hoạt.

1. **Cách cài đặt**

Một hàng đợi cũng có thể được biểu diễn bằng các thực thể sau: danh sách liên kết, con trỏ và cấu trúc.

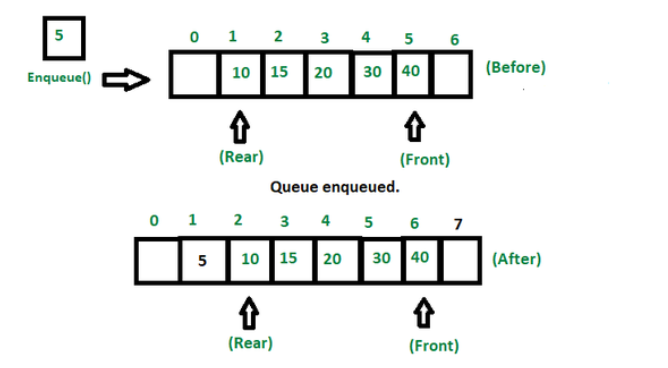
Giống như ngăn xếp, hàng đợi cũng có thể được biểu diễn trong một mảng: Trong cách biểu diễn này, hàng đợi được triển khai bằng cách sử dụng danh sách liên kết. Các biến được sử dụng trong trường hợp này là:

* Hàng đợi: tên của mảng lưu trữ các phần tử hàng đợi.
* Head : chỉ mục nơi phần tử đầu tiên được lưu trữ trong mảng đại diện cho hàng đợi.
* Tail: chỉ mục nơi phần tử cuối cùng được lưu trữ trong một mảng đại diện cho hàng đợi.
* Size: chỉ số lượng phần tử hiện có.

|  |
| --- |
| struct Node{ int data; Node \*next;  };  class Queue{ private:  Node \*head;  Node \*tail;  int size; public:  Queue();// khoi tao  bool isEmpty();// Kiểm tra queue có rỗng hay không  int Size();// Số phần tử của queue  void enqueue(int x);// Thêm cuối  void dequeue();// Xóa đầu  void show();// In queue  int front();// Trả về phần tử đầu  int back();// Trả về phần tử cuối  }; |

Ví dụ:

**enqueue()**: trong hàng đợi sẽ thêm (hoặc lưu trữ) một phần tử vào cuối hàng đợi



**isFull()**: Thao tác này trả về một giá trị boolean cho biết hàng đợi đã đầy hay chưa.

**isEmpty()**: Thao tác này trả về một giá trị boolean cho biết hàng đợi có trống hay không.

…

Có nhiều loại hàng đợi khác nhau:

* Hàng đợi hạn chế đầu vào: Đây là một hàng đợi đơn giản. Trong loại hàng đợi này, đầu vào chỉ có thể được lấy từ một đầu nhưng có thể xóa từ bất kỳ đầu nào.
* Hàng đợi hạn chế đầu ra: Đây cũng là một hàng đợi đơn giản. Trong loại hàng đợi này, đầu vào có thể được lấy từ cả hai đầu nhưng chỉ có thể xóa từ một đầu.
* Hàng đợi tròn : Đây là loại hàng đợi đặc biệt trong đó vị trí cuối cùng được kết nối trở lại vị trí đầu tiên. Ở đây các hoạt động cũng được thực hiện theo thứ tự FIFO. Để biết thêm hãy tham khảo điều này .
* Hàng đợi hai đầu (Dequeue) : Trong hàng đợi hai đầu, các thao tác chèn và xóa, cả hai đều có thể được thực hiện từ cả hai đầu. Để biết thêm hãy tham khảo điều này .
* Hàng đợi ưu tiên : Hàng đợi ưu tiên là hàng đợi đặc biệt trong đó các phần tử được truy cập dựa trên mức độ ưu tiên được gán cho chúng.

1. **Ứng dụng**

Việc áp dụng hàng đợi là phổ biến. Trong hệ thống máy tính, có thể có hàng đợi tác vụ

chờ máy in, truy cập vào bộ lưu trữ đĩa hoặc thậm chí trong hệ thống chia sẻ thời gian để sử dụng CPU. Trong một chương trình, có thể có nhiều yêu cầu được giữ trong hàng đợi hoặc một tác vụ có thể tạo ra các tác vụ khác, các tác vụ này phải được thực hiện lần lượt bằng cách giữ chúng trong hàng đợi.

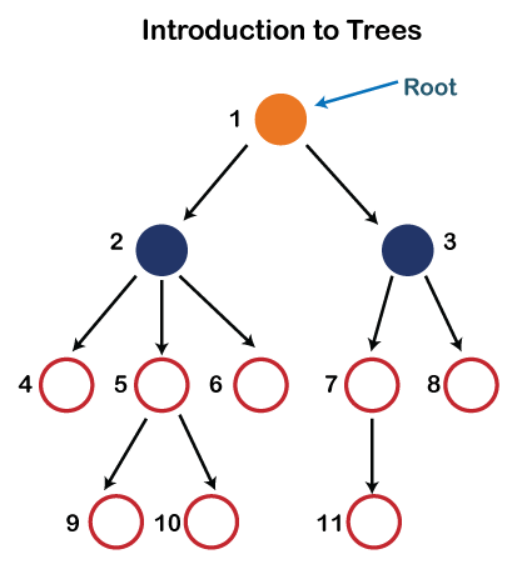
* Nó có một tài nguyên duy nhất và nhiều người tiêu dùng.
* Nó đồng bộ hóa giữa các thiết bị chậm và nhanh.
* Trong mạng, hàng đợi được sử dụng trong các thiết bị như bộ định tuyến/bộ chuyển mạch và hàng đợi thư.
* Các biến thể: dequeue, hàng đợi ưu tiên và hàng đợi ưu tiên hai đầu.

**CẤU TRÚC CÂY VÀ CÂY NHỊ PHÂN(TREE AND BINARY TREE)**

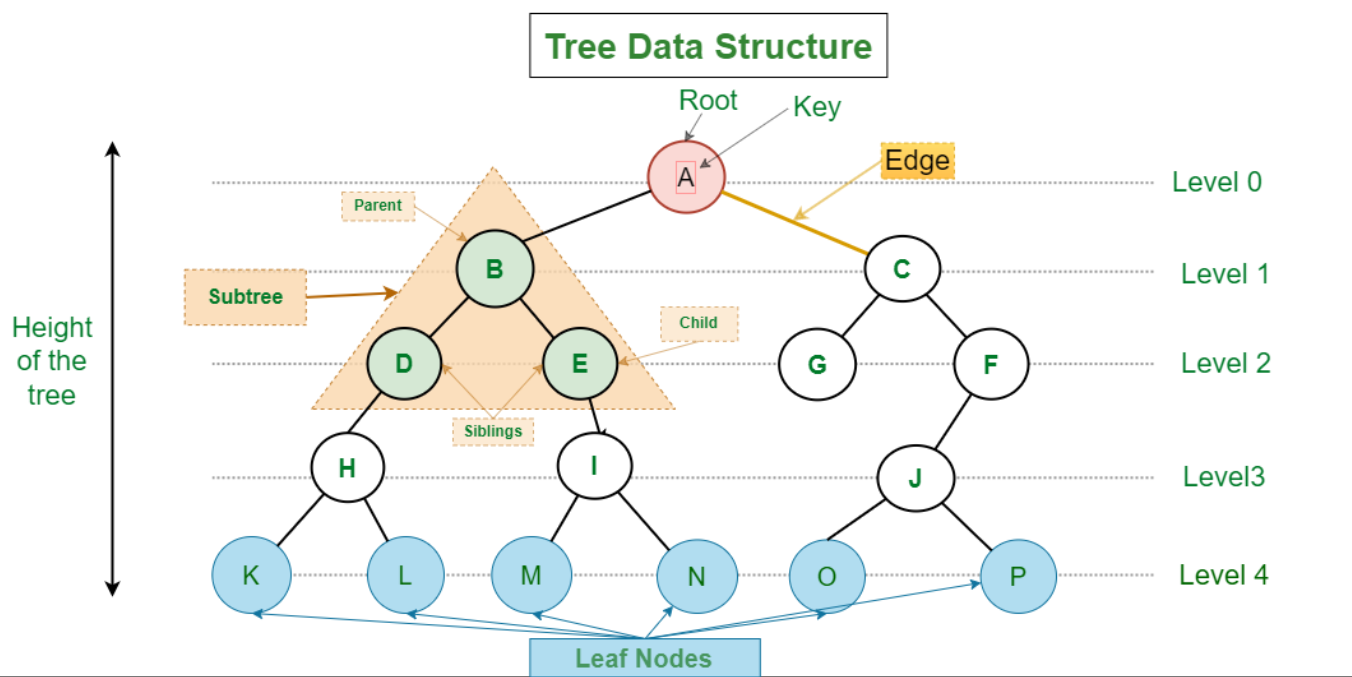
1. **Cây**
   1. **Giới thiệu**

Cấu trúc dữ liệu cây là cấu trúc phân cấp được sử dụng để biểu diễn và sắp xếp dữ liệu theo cách dễ dàng điều hướng và tìm kiếm. Nó là tập hợp các nút được kết nối bằng các cạnh và có mối quan hệ phân cấp giữa các nút. Nút trên cùng của cây được gọi là nút gốc, các nút bên dưới gọi là nút con. Mỗi nút có thể có nhiều nút con và các nút con này cũng có thể có các nút con riêng, tạo thành cấu trúc đệ quy.

Cấu trúc dữ liệu này là một phương pháp chuyên biệt để tổ chức và lưu trữ dữ liệu trong máy tính để sử dụng hiệu quả hơn. Nó bao gồm một nút trung tâm, các nút cấu trúc và các nút phụ, được kết nối thông qua các cạnh. Chúng ta cũng có thể nói rằng cấu trúc dữ liệu cây có rễ, nhánh và lá được kết nối với nhau.



* 1. **Các thuật ngữ cơ bản trong cấu trúc dữ liệu cây:**



Nút cha: Nút tiền thân của nút được gọi là nút cha của nút đó. {B} là nút cha của {D, E} .

Nút con: Nút kế thừa trực tiếp của nút được gọi là nút con của nút đó. Ví dụ: {D, E} là các nút con của {B}.

Nút gốc: Nút trên cùng của cây hoặc nút không có nút cha nào được gọi là nút gốc. {A } là nút gốc của cây. Cây không trống phải chứa chính xác một nút gốc và chính xác một đường dẫn từ gốc đến tất cả các nút khác của cây.

Nút lá hoặc Nút ngoài: Các nút không có nút con nào được gọi là nút lá. {K, L, M, N, O, P} là các nút lá của cây.

Tổ tiên của nút: Bất kỳ nút tiền thân nào trên đường dẫn gốc tới nút đó đều được gọi là Tổ tiên của nút đó. {A,B} là các nút tổ tiên của nút {E}

Hậu duệ: Bất kỳ nút kế thừa nào trên đường đi từ nút lá đến nút đó. {E,I} là hậu duệ của nút {B}.

Anh chị em: Con của cùng một nút cha được gọi là anh chị em. {D,E} được gọi là anh chị em.

Cấp độ của một nút: Số cạnh trên đường dẫn từ nút gốc đến nút đó. Nút gốc có cấp độ 0 .

Nút nội bộ: Một nút có ít nhất một nút con được gọi là Nút nội bộ.

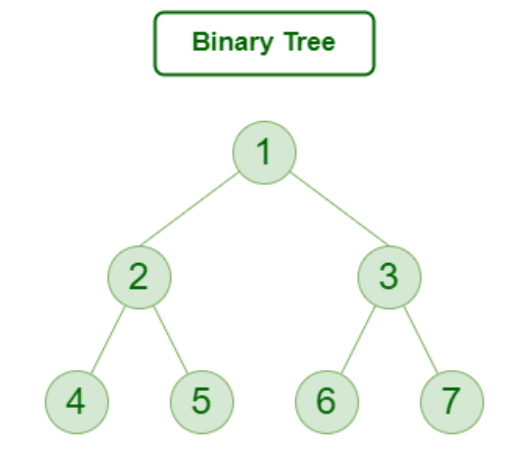
Hàng xóm của một nút: Các nút cha hoặc nút con của nút đó được gọi là hàng xóm của nút đó.

Cây con : Bất kỳ nút nào của cây cùng với con cháu của nó.

**2.** **Cây nhị phân**

Trong cây nhị phân, mỗi nút có thể có tối đa hai nút con được liên kết với nó. Một số loại

cây nhị phân phổ biến bao gồm cây nhị phân đầy đủ, cây nhị phân hoàn chỉnh, cây nhị phân cân bằng và cây nhị phân thoái hóa hoặc bệnh lý.



Các loại cây nhị phân:

* Cây nhị phân đầy đủ: là cây nhị phân mà mỗi Node chỉ có 0 hoặc 2 con.
* Cây nhị phân hoàn chỉnh: là cây nhị phân mà mỗi Node có ít hơn 2 con.
* Cây nhị phân hoàn hảo: là cây nhị phân mà mỗi Node có đúng 2 con.

**2.1 Các thuộc tính của cây nhị phân**

Số cạnh: Một cạnh có thể được định nghĩa là kết nối giữa hai nút. Nếu một cây có N nút

thì nó sẽ có (N-1) cạnh. Chỉ có một đường dẫn từ mỗi nút đến bất kỳ nút nào khác của cây.

Độ sâu của nút: Độ sâu của nút được định nghĩa là độ dài của đường dẫn từ gốc đến nút đó. Mỗi cạnh thêm 1 đơn vị chiều dài cho đường đi. Vì vậy, nó cũng có thể được định nghĩa là số cạnh trên đường đi từ gốc cây đến nút.

Chiều cao của nút: Chiều cao của nút có thể được định nghĩa là độ dài của đường đi dài nhất từ nút đến nút lá của cây.

Chiều cao của cây: Chiều cao của cây là độ dài đường đi dài nhất từ gốc cây đến nút lá của cây.

Cấp độ của một nút: Tổng số cây con gắn liền với nút đó được gọi là cấp độ của nút. Bậc của nút lá phải bằng 0 . Bậc của cây là bậc lớn nhất của một nút trong số tất cả các nút trong cây.

**2.2 Cài đặt cây nhị phân**

* Tạo cấu trúc cây:

|  |
| --- |
| struct Node {     DataType data;     Node \*left;     Node \*right; }; |

* Tạo node cho cây:

|  |
| --- |
| Node \*createNode(DataType v) {     Node \*newNode = new Node();     newNode->data = v;     newNode->left = NULL;     newNode->right = NULL;     return newNode; } |

* Thêm node vào bên trái:

|  |
| --- |
| void insertLeft(Node \*p, Item v) {     if (p == NULL)         cout << "Cannot insert to a NULL node. \n";     else if (p->left != NULL)         cout << "Left child existed. \n";     else         p->left = createNode(v); } |

* Thêm node vào bên phải:

|  |
| --- |
| void insertRight(Node \*p, Item v) {     if (p == NULL)         cout << "Cannot insert to a NULL node. \n";     else if (p->right != NULL)         cout << "Right child existed. \n";     else         p->right = createNode(v); } |

**3. Ứng dụng của cây**

*Hệ thống tệp:* Điều này cho phép điều hướng và sắp xếp tệp hiệu quả.

*Nén dữ liệu :* Mã hóa Huffman là một kỹ thuật phổ biến để nén dữ liệu liên quan đến việc xây dựng cây nhị phân trong đó các lá biểu thị các ký tự và tần suất xuất hiện của chúng. Cây kết quả được sử dụng để mã hóa dữ liệu theo cách giảm thiểu dung lượng lưu trữ cần thiết.

*Thiết kế trình biên dịch:* Trong thiết kế trình biên dịch, cây cú pháp được sử dụng để thể hiện cấu trúc của chương trình.

*Lập chỉ mục cơ sở dữ liệu:* Cây B và các cấu trúc cây khác được sử dụng trong lập chỉ mục cơ sở dữ liệu để tìm kiếm và truy xuất dữ liệu một cách hiệu quả.

\****Ưu điểm của cấu trúc dữ liệu cây:***

* Cung cấp cây Tìm kiếm hiệu quả Tùy thuộc vào loại cây, với thời gian tìm kiếm trung bình là O(log n) đối với các cây cân bằng như AVL.
* Cây cung cấp cách trình bày dữ liệu có thứ bậc, giúp dễ dàng tổ chức và điều hướng lượng lớn thông tin.
* Bản chất đệ quy của cây làm cho chúng dễ dàng duyệt qua và thao tác bằng các thuật toán đệ quy.

***\*Nhược điểm của cấu trúc dữ liệu cây:***

* Cây không cân bằng, nghĩa là chiều cao của cây bị lệch về một phía, điều này có thể dẫn đến thời gian tìm kiếm không hiệu quả.
* Cây yêu cầu nhiều không gian bộ nhớ hơn một số cấu trúc dữ liệu khác như mảng và danh sách liên kết, đặc biệt nếu cây rất lớn.
* Việc triển khai và thao tác với cây có thể phức tạp và đòi hỏi sự hiểu biết tốt về các thuật toán.

### **MỘT SỐ THUẬT TOÁN**

#### **Sắp xếp (Sorting)**

##### **1.1. Sắp xếp chọn (Selection sort)**

Sắp xếp chọn là một thuật toán sắp xếp hoạt động bằng cách chọn liên tục phần tử nhỏ nhất (hoặc lớn nhất) từ phần chưa được sắp xếp của danh sách và di chuyển nó đến phần được sắp xếp của danh sách.

Pseudocode:

* Duyệt từ đầu đến trước phần tử cuối cùng của mảng [i = 0, i = n - 2]
* Khởi tạo min\_pos = i để chọn ra phần tử đang xét là phần tử nhỏ nhất
* Với mỗi vòng lặp i thực hiện duyệt từ đầu đến cuối mảng [j = 0, j = n - 1], kiểm tra xem giá trị tại min\_pos có lớn hơn giá trị tại j hay không?

Đúng: min\_pos = j

Sai: tiếp tục lặp j

* Kết thúc vòng lặp j thực hiện đổi chỗ 2 giá trị tại min\_pos và tại j
* Quay lại bước 1

**Cài đặt:**

|  |
| --- |
| *void selectionSort(int arr[], int n){     for (int i = 0; i <= n - 2; i++){         int min\_pos = i;         for (int j = i; j <= n - 1; j++){             if (a[j] < a[min\_pos])                 min\_pos = j;         }         swap(a[min\_pos], a[i]);     } } // Độ phức tạp: O(n^2)* |

##### **1.2. Sắp xếp nổi bọt (Bubble Sort)**

Bubble Sort là thuật toán sắp xếp hoạt động bằng cách liên tục hoán đổi các phần tử liền kề nếu chúng sai thứ tự. Thuật toán này không phù hợp với các tập dữ liệu lớn vì độ phức tạp về thời gian trung bình và trường hợp xấu nhất của nó khá cao.

Pseudocode:

* Duyệt từ đầu đến trước phần tử cuối cùng của mảng [i = 0, i = n - 2]
* Mỗi vòng lặp i duyệt từ i đến trước phần tử cuối cùng của mảng [j = i, j = n - 2], kiểm tra xem arr[j] > arr[j + 1]?

Đúng: Thực hiện đổi chỗ arr[j] và arr[j + 1]

Sai: Tiếp tục vòng lặp j

* Quay lại B1

**Cài đặt:**

|  |
| --- |
| *void bubbleSort(int arr[], int n) {     for (int i = 0; i <= n - 2; i++)     {         for (int j = i; j <= n - 2; j++){             if (arr[j] > arr[j + 1])                 swap(arr[j], arr[j + 1]);         }     } } // Độ phức tạp: O(n^2)* |

##### **1.3. Sắp xếp chèn (Insertion sort)**

Sắp xếp chèn là một thuật toán sắp xếp hoạt động tương tự như cách sắp xếp các quân bài trên tay. Mảng hầu như được chia thành một phần được sắp xếp và một phần chưa được sắp xếp. Các giá trị từ phần chưa sắp xếp được chọn và đặt vào đúng vị trí trong phần đã sắp xếp.

Pseudocode:

* Duyệt từ phần tử thứ i = 2 đến phần tử cuối cùng của mảng [2, n - 1]
* Khởi tạo j = i - 1 và key = arr[i]
* Lặp j >= 0 và arr[j] > key

arr[j + 1] = arr[j]

j--

* Kết thúc vòng lặp bây giờ j = -1 hoặc arr[j] <= key. Thực hiện cho

arr[j + 1] = key

**Cài đặt:**

|  |
| --- |
| *void insertionSort(int arr[], int n) {     for(int i = 1; i <= n - 1; i++){         int j = i - 1, key = arr[i];         while(j >= 0 && arr[j] > key)         {             arr[j + 1] = arr[j];             j--;         }         arr[j + 1] = key;     } }* |

##### **1.4. Sắp xếp nhanh (Quick Sort)**

QuickSort là một thuật toán sắp xếp dựa trên thuật toán chia để trị, chọn một phần tử làm trục và phân chia mảng đã cho xung quanh trục đã chọn bằng cách đặt trục vào đúng vị trí của nó trong mảng đã sắp xếp.

Quá trình chính trong quickSort là phân hoạch. Mục tiêu của các phân hoạch là đặt trục (bất kỳ phần tử nào có thể được chọn làm trục) vào đúng vị trí của nó trong mảng đã sắp xếp và đặt tất cả các phần tử nhỏ hơn ở bên trái trục và tất cả các phần tử lớn hơn ở bên phải trục.

**Hai cách phân hoạch:**

*\*Phân hoạch Lomuto*

Pseudocode:

|  |
| --- |
| * Lomuto-partition(A, l, r) * x = A[r] * i = l - 1 * for j = l đến r - 1   if (A[l] <= x)  ++i  Hoán đổi A[i] và A[j]   * ++i * Hoán đổi A[i] và A[r] * return i |

**Cài đặt:**

|  |
| --- |
| *int partitionLomuto(int a[], int l, int r){     int pivot = a[r];// Chọn pt cuối làm chốt     int i = l - 1;     for (int j = l; j < r; j++){         if (a[j] <= pivot){             ++i;             swap(a[i], a[j]);         }         // a[j] > pivot -> bỏ qua     }     ++i;     swap(a[i], a[r]); // Đưa chốt về chính giữa     return i; }* |

\**Phân hoạch Hoare*

Pseudocode:

|  |
| --- |
| 1. Hoare-Partition(A, l, r) 2. i = l - 1, j = r - 1, x = A[r] 3. While TRUE   do  j = j - 1  while (A[j] > x)  do  i = i + 1  while (A[i] < x)  if i < j  Hoán đổi A[i] và A[j]  else return j |

**Cài đặt:**

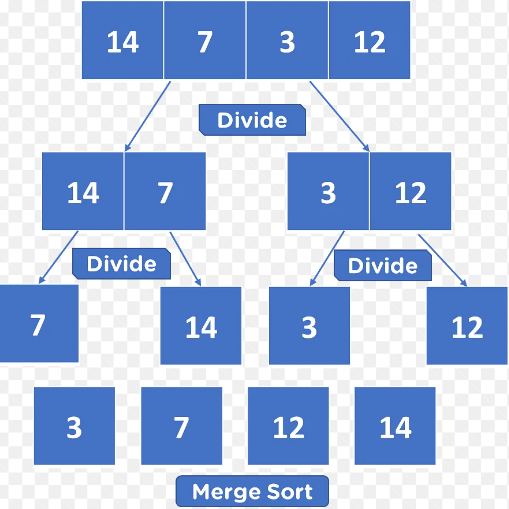
|  |
| --- |
| *int partitionHoare(int a[], int l, int r){     int i = l - 1, j = r + 1, pivot = a[r];     while (true){         do             ++i;         while (a[i] < pivot);          do             --j;         while (a[j] > pivot);          if (i < j)             swap(a[i], a[j]);         else             return j;     } }* |

Giải thuật quick sort với 2 phân hoạch trên:

|  |
| --- |
| *// Quick sort với phân hoạch Lomuto  void quickSortLomuto(int a[], int l, int r) {     if (l >= r)         return;     int m = partitionLomuto(a, l, r);     quickSortLomuto(a, l, m - 1);     quickSortLomuto(a, m + 1, r); }  // Quick sort với phân hoạch Hoare  void quickSortHoare(int a[], int l, int r) {     if (l >= r)         return;     int m = partitionHoare(a, l, r);     quickSortHoare(a, l, m - 1);     quickSortHoare(a, m, r); }* |

##### **1.5. Sắp xếp trộn (Merge sort)**

Sắp xếp trộn được định nghĩa là một thuật toán sắp xếp hoạt động bằng cách chia một mảng thành các mảng con nhỏ hơn, sắp xếp từng mảng con và sau đó hợp nhất các mảng con đã sắp xếp lại với nhau để tạo thành mảng được sắp xếp cuối cùng.



**Cài đặt:**

|  |
| --- |
| *// Merge Sort void mergeSort(int a[], int l, int r){     if (l >= r)         return;     int m = (l + r) / 2;     // Lấy chỉ số giữa     mergeSort(a, l, m);      // Sắp xếp bên trái     mergeSort(a, m + 1, r);  // Sắp xếp bên phải     // Trộn 2 bên thành được mảng sắp xếp     merge(a, l, m, r); }* |

|  |
| --- |
| *// Thao tác merge của thuật toán merge Sort => merge mảng L[] và R[] => arr[] void merge(int arr[], int l, int m, int r){     int n1 = m - l + 1;     int n2 = r - m;     int L[n1], R[n2];      for (int i = 0; i < n1; i++)         L[i] = arr[l + i];     for (int j = 0; j < n2; j++)         R[j] = arr[m + 1 + j];      int i = 0, j = 0;     while (i < n1 && j < n2){         if (L[i] <= R[j])             arr[l++] = L[i++];         else             arr[l++] = R[j++];     }      while (i < n1)         arr[l++] = L[i++];      while (j < n2)         arr[l++] = R[j++]; }* |

##### **1.6. Sắp xếp vun đống (Heap Sort)**

Heap Sort là một thuật toán sắp xếp dựa trên cấu trúc dữ liệu heap. Nó kết hợp cả tính chất của heap và quy tắc sắp xếp của thuật toán sắp xếp chọn (Selection Sort).

Quá trình Heap Sort gồm các bước:

- Xây dựng Max-Heap: Đầu tiên, chuyển đổi mảng ban đầu thành một Max-Heap bằng cách áp dụng phép heapify cho từng nút từ cuối cùng của mảng đến nút gốc (nút đầu tiên).

- Sắp xếp mảng: Sau khi xây dựng Max-Heap, phần tử lớn nhất sẽ nằm ở vị trí nút gốc (nút đầu tiên) của heap. Thực hiện các bước sau cho đến khi heap chỉ còn một phần tử:

+ Hoán đổi giữa nút gốc và phần tử cuối cùng của heap để đưa phần tử lớn nhất vào vị trí cuối cùng của mảng.

+ Giảm kích thước của heap đi 1 (loại bỏ phần tử cuối cùng) để không xét đến phần tử đã được sắp xếp.

+ Thực hiện phép heapify cho nút gốc để đảm bảo thuộc tính heap vẫn được duy trì.

- Kết quả: Sau khi hoàn thành quá trình sắp xếp, mảng sẽ chứa các phần tử theo thứ tự tăng dần.

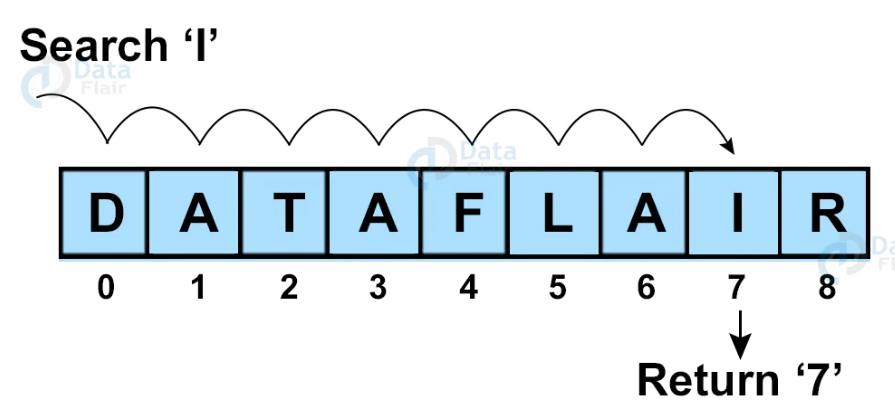
|  |
| --- |
| *// Thao tác heapify để xây dựng max-heap void heapify(int arr[], int n, int i) {     int largest = i;     int left = 2 \* i + 1;     int right = 2 \* i + 2;     if (left < n && arr[left] > arr[largest])         largest = left;     if (right < n && arr[right] > arr[largest])         largest = right;     if (largest != i)     {         swap(arr[i], arr[largest]);         heapify(arr, n, largest);     } }* |

|  |
| --- |
| *// Thuật toán heap sort void heapSort(int arr[], int n) {     // Xây dựng max - heap     for (int i = n / 2 - 1; i >= 0; i--)         heapify(arr, n, i);     for (int i = n - 1; i >= 0; i--)     {         swap(arr[i], arr[0]);         heapify(arr, i, 0);     } }* |

#### **2. Tìm kiếm (Search)**

##### **2.1. Tìm kiếm tuyến tính (Linear Search)**

Tìm kiếm tuyến tính được định nghĩa là thuật toán tìm kiếm tuần tự bắt đầu ở một đầu và đi qua từng phần tử của danh sách cho đến khi tìm thấy phần tử mong muốn, nếu không thì việc tìm kiếm sẽ tiếp tục cho đến hết tập dữ liệu.



*Cách hoạt động:*

* Duyệt qua tất cả các phần tử
* Nếu tìm thấy bất kỳ phần tử nào bằng khóa thì tìm kiếm thành công.
* Nếu không tìm thấy phần tử nào bằng khóa, kết quả tìm kiếm sẽ là “Không tìm thấy kết quả khớp”.

|  |
| --- |
| *// Thuật toán trả về true nếu key được tìm thấy trong mảng a[], ngược lại false  bool linearSearch(int arr[], int n, int key){     for (int i = 0; i < n; i++){         if (arr[i] == key)             return true;     }     return false; } // Trường hợp tốt nhất: O(1) // Trường hợp xấu nhất: O(N) // Trường hợp trung bình: O(N)* |

##### **2.2. Tìm kiếm nhị phân**

Tìm kiếm nhị phân là một thuật toán tìm kiếm được sử dụng trong một mảng được sắp xếp bằng cách liên tục chia khoảng thời gian tìm kiếm thành một nửa.

Ý tưởng của tìm kiếm nhị phân là sử dụng mảng đã được sắp xếp và giảm độ phức tạp về thời gian xuống O(log N).

Pseudocode

|  |
| --- |
| * BinarySearch(arr, n, key) * low = 0, high = n -1 * while low <= high   mid = (low + high) / 2  if arr[mid] = key  return true  else if(arr[mid] < key)  low = mid + 1  else  high = mid - 1   * return false |

**Cài đặt:**

|  |
| --- |
| *// Thuật toán trả về true nếu key được tìm thấy trong mảng arr[], ngược lại false bool binarySearch(int arr[], int n, int key){     int low = 0, high = n - 1;     while (low <= high){         int mid = (low + high) / 2;         if (arr[mid] == key)             return true;         else if (arr[mid] < key)                          low = mid + 1;         else                          high = mid - 1;     }     return false; } // Trường hợp tốt nhất: O(1) // Trường hợp xấu nhất: O(logN) // Trường hợp trung bình: O(logN)* |

## **PHẦN 2: CHƯƠNG TRÌNH**

### **Lý do chọn đề tài quản lý**

*Nhu cầu thực tiễn:* Trong hầu hết các tổ chức giáo dục, quản lý thông tin của sinh viên là một nhiệm vụ quan trọng. Các thông tin bao gồm danh tính, học phí, lịch học, điểm số, và nhiều thông tin khác. Một hệ thống quản lý sinh viên giúp tổ chức quản lý thông tin này hiệu quả hơn.

*Tích hợp dữ liệu:* Quản lý sinh viên liên quan đến nhiều dữ liệu khác nhau, từ thông tin cá nhân đến tình trạng học tập và tài chính. Cần một hệ thống để tổng hợp, lưu trữ và cung cấp dữ liệu này dễ dàng truy cập và quản lý.

*Tăng cường hiệu suất:* Một hệ thống quản lý sinh viên có thể giúp tổ chức giáo dục tăng cường hiệu suất. Nó có thể tự động hóa các quy trình quản lý, giảm thời gian và công sức của người làm việc và giảm thiểu sai sót.

### **Mô tả bài toán**

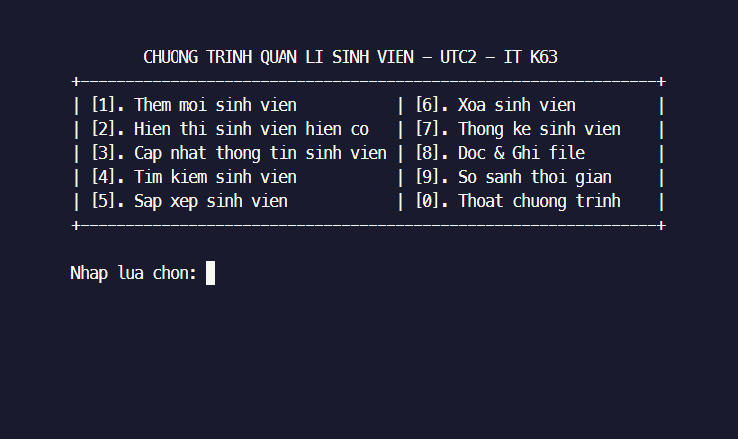
Chương trình quản lý sinh viên là một chương trình dành để quản lý các sinh viên trong trong một trường đại học, cao đẳng,.. mục đích để tránh sự thống kê không chính xác và đưa ra các số liệu sinh viên theo nhiều tiêu chí khác nhau: thống kê điểm trung bình môn, thống kê số lượng nam/nữ, thống kê số lượng,..., muốn làm được chương trình này trước tiên chúng ta phải sử dụng cấu trúc class/struct để lưu trữ đối tượng nhiều thuộc tính, sau đó chúng ta sử dụng các cấu trúc dữ liệu, các thuật toán, cấu trúc lặp,... để quản lý chương trình.

Chương trình bọn em gồm có 9 chức năng chính, trong đó có 1 số chức năng nổi bật như là: thêm, hiển thị, xóa, chỉnh sửa, tìm kiếm, sắp xếp, so sánh thời gian thực hiện các thuật toán,... các cấu trúc dữ liệu được sử dụng là danh sách liên kết đôi cùng với 2 thuật toán nổi bật sắp xếp nhanh và sắp xếp trộn, tìm kiếm nhị phân,..

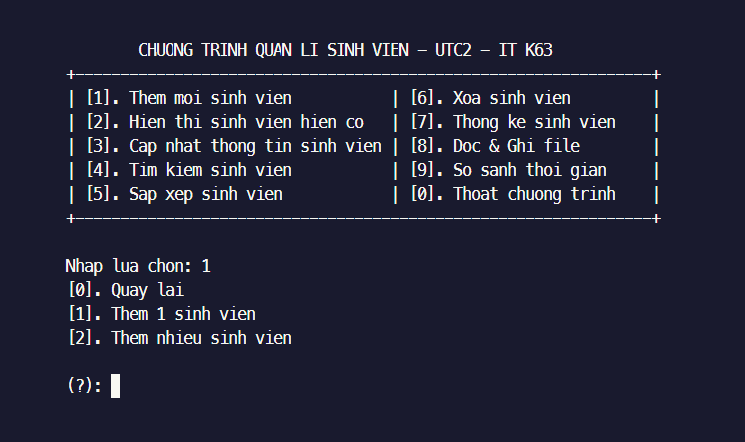
Lý do bọn em sử dụng cấu trúc dữ liệu danh sách liên kết đôi, các thuật toán sắp xếp nhanh, sắp xếp trộn, tìm kiếm nhị phân là thứ nhất là bọn em muốn ứng dụng từ những kiến thức đã học được ở môn cấu trúc dữ liệu và giải thuật, thứ hai là nó thích hợp với chương trình quản lý bọn em hướng tới, độ phức tạp thấp, dễ quản lý, dễ sử dụng.

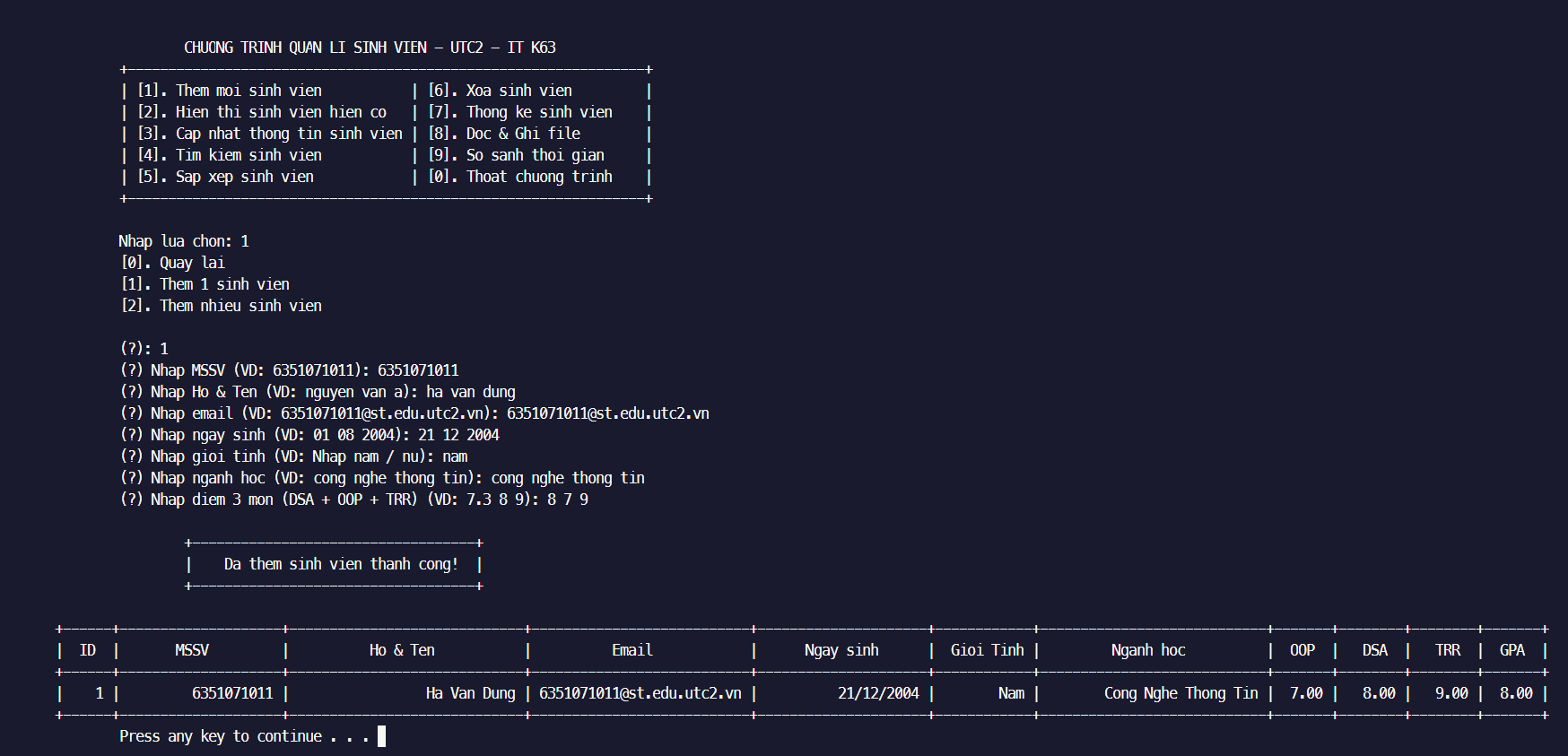
### **Giao diện chương trình**

## Giao diện menu chính của chương trình

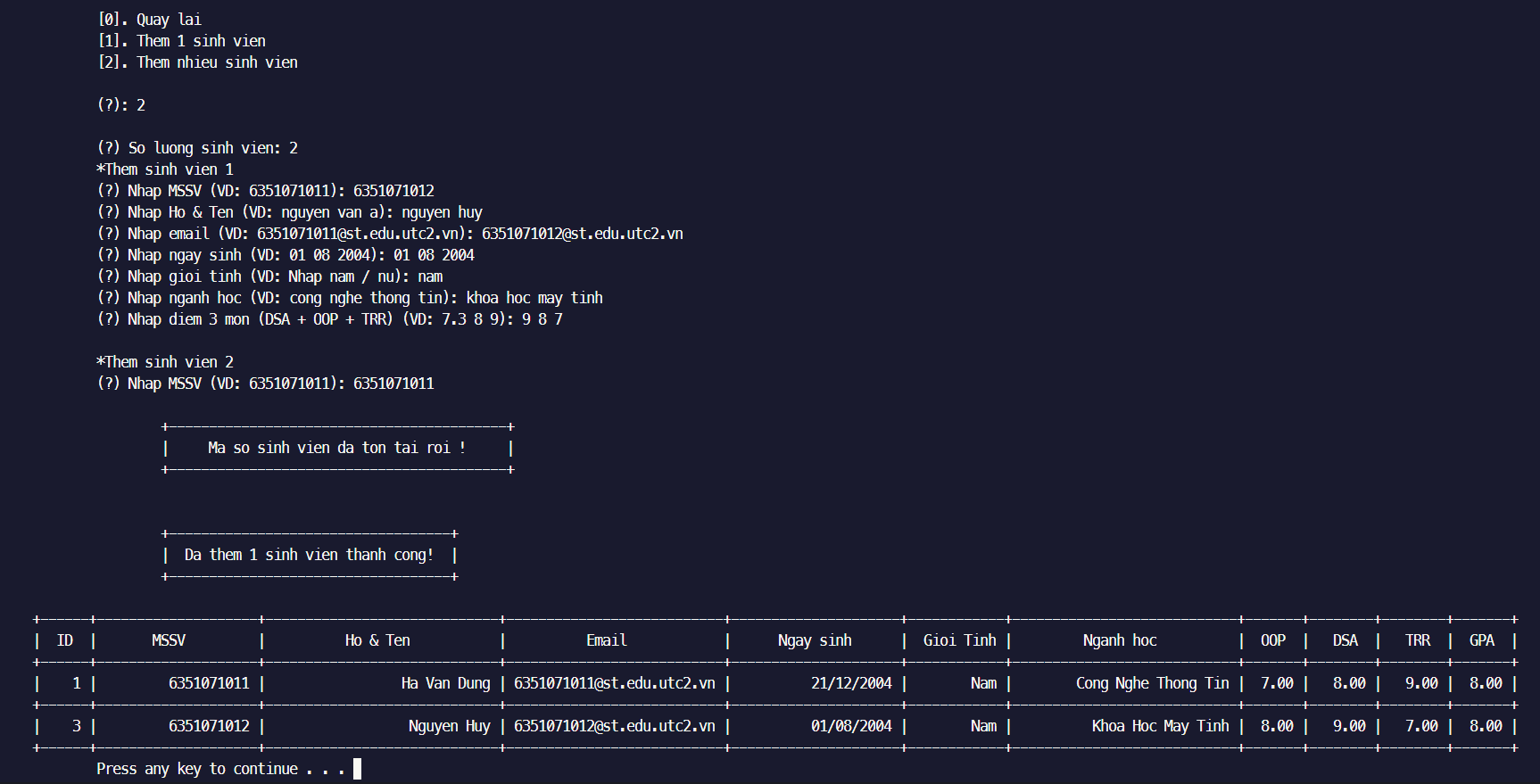


## Chức năng thêm sinh viên

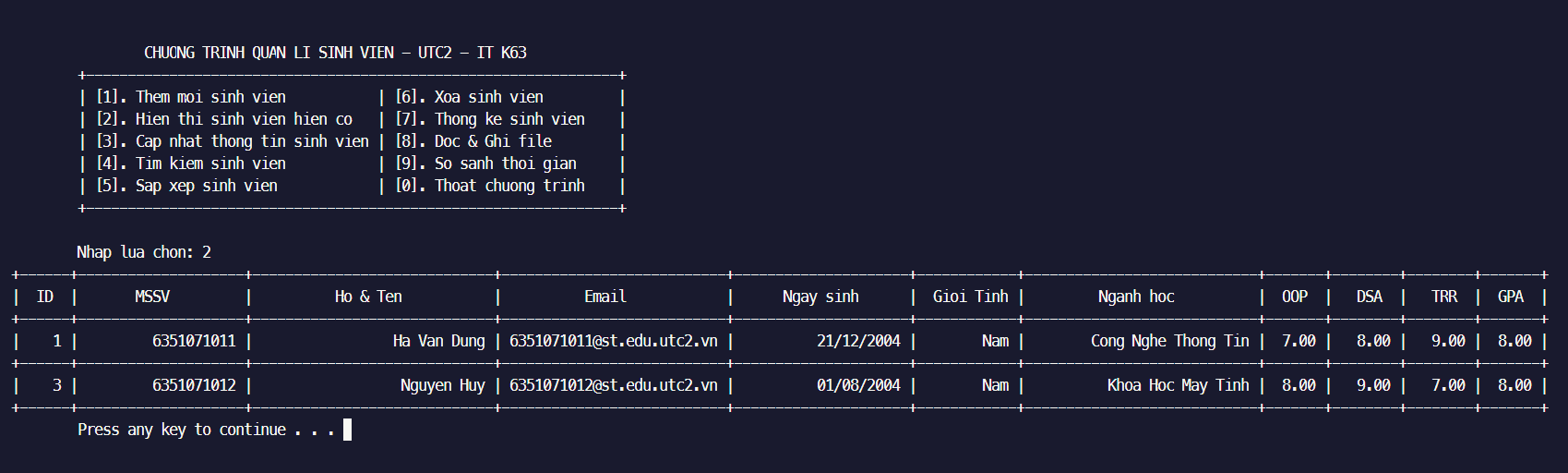




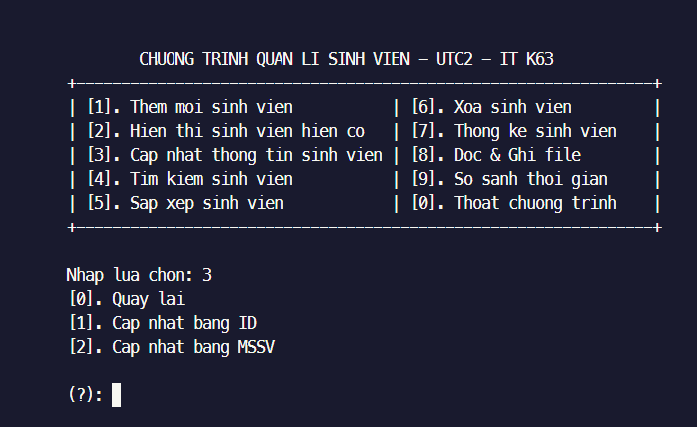
*-Thêm 1 sinh viên-*

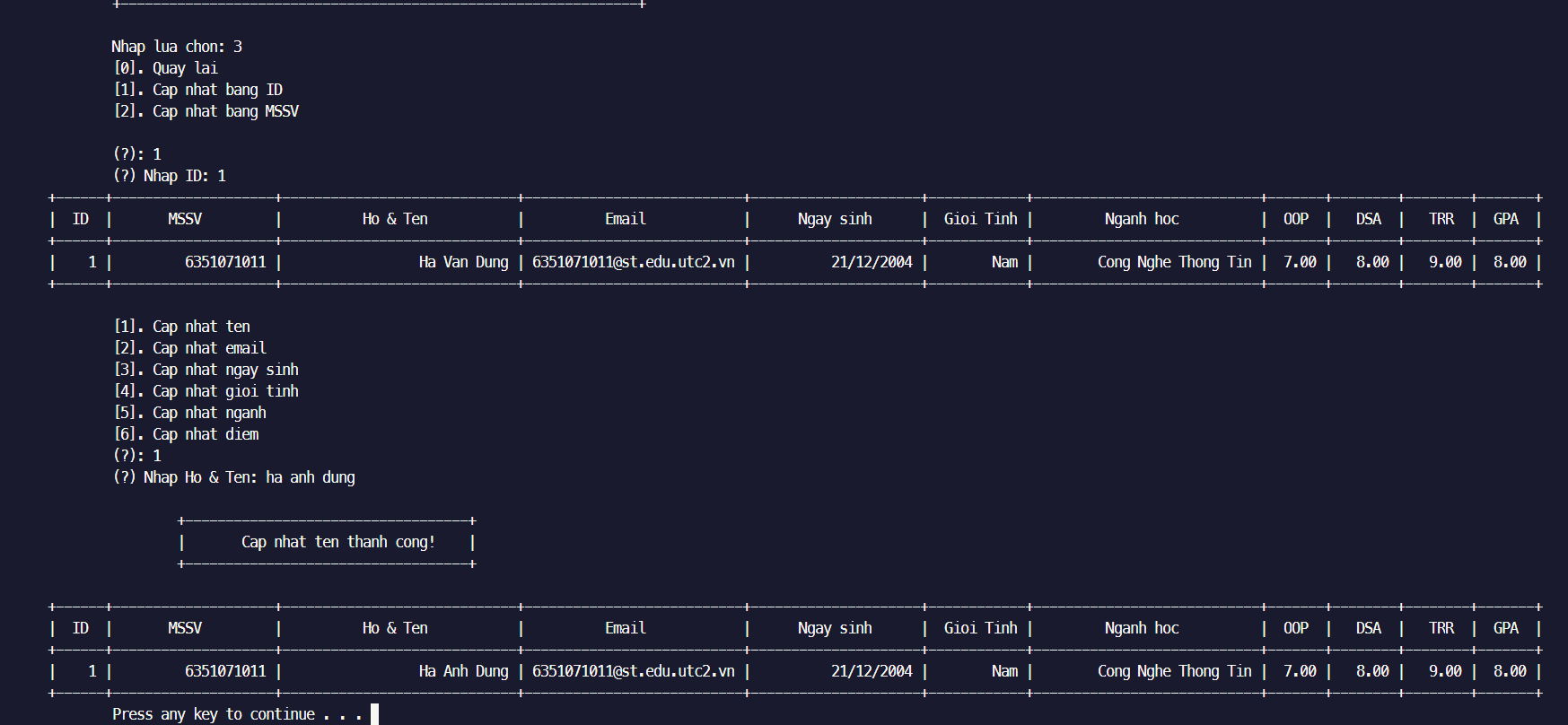
  *-Thêm nhiều sinh viên-*

## Chức năng hiển thị sinh viên hiện có:

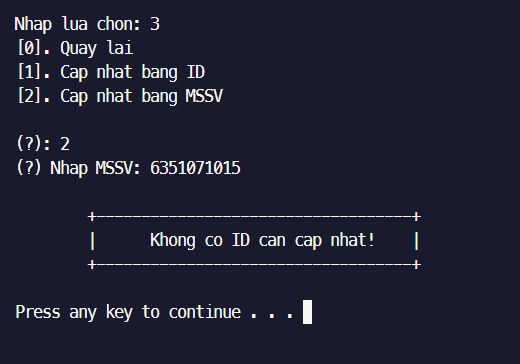


## Chức năng cập nhật thông tin sinh viên

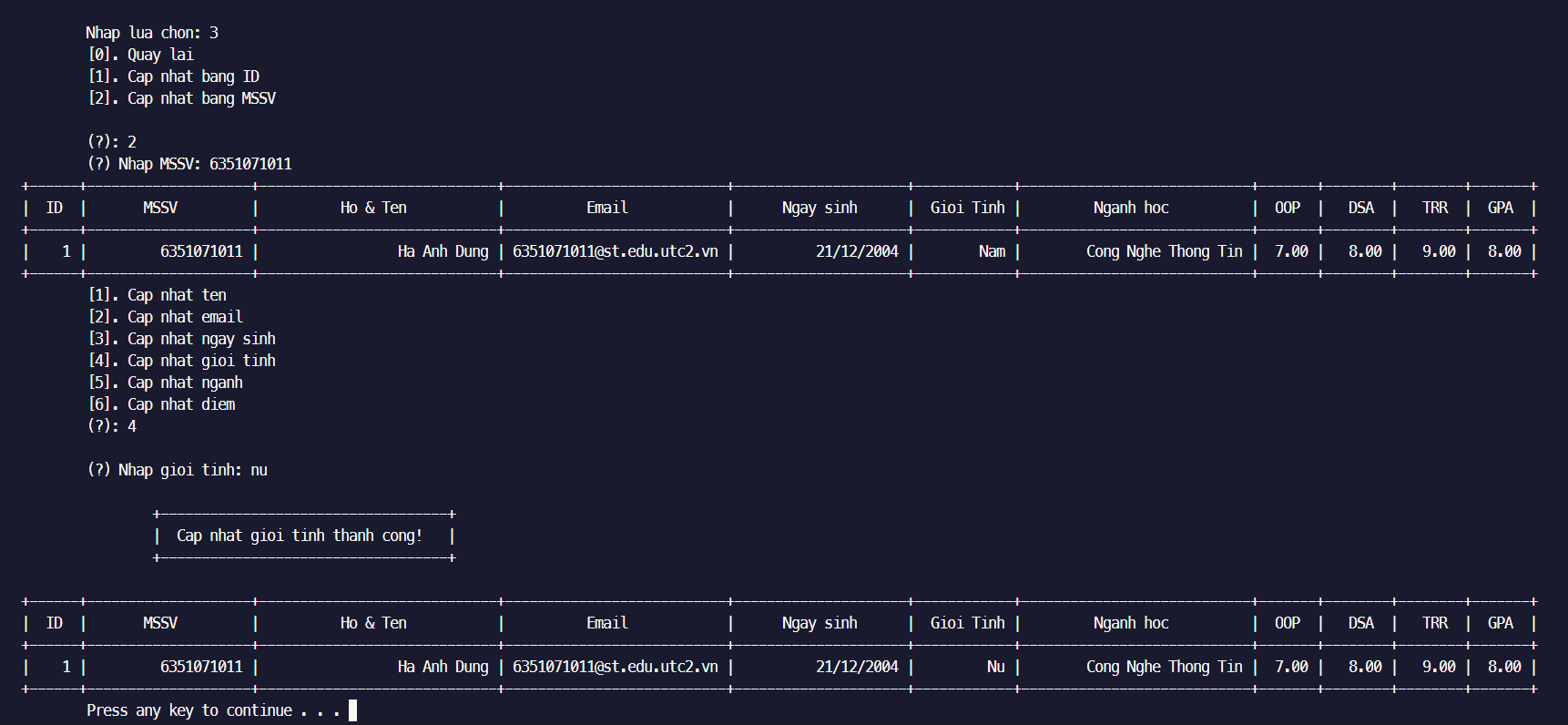




*-Cập nhật tên cho sinh viên có ID là 1 thành công-*

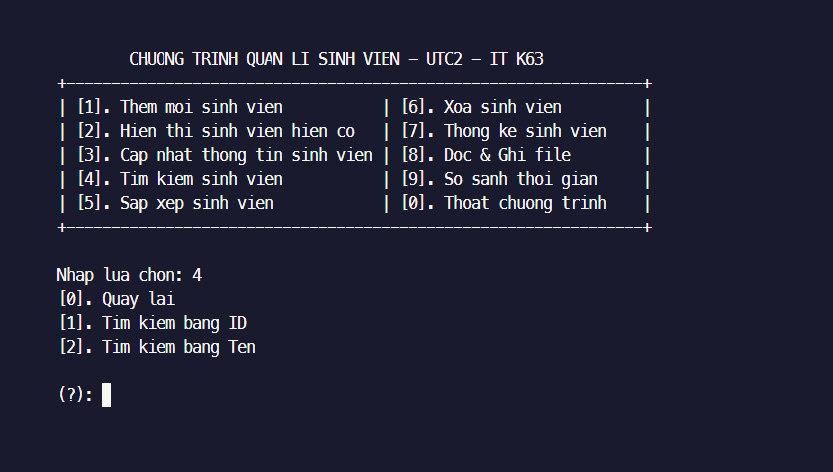


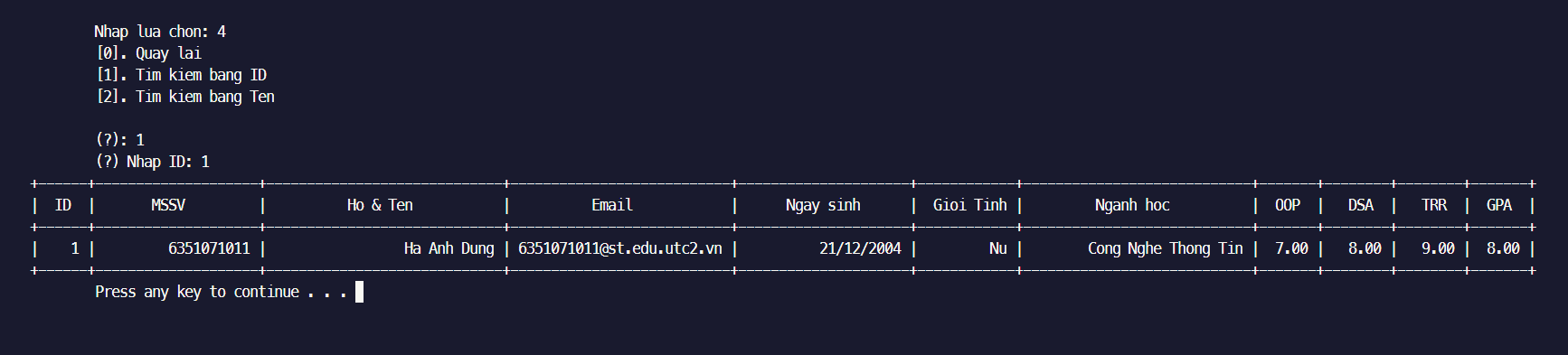
*-Cập nhật MSSV không tồn tại trong hệ thống-*



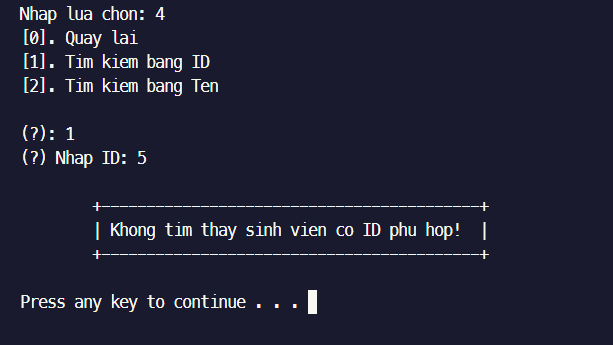
*-Cập nhật giới tính cho sinh viên có MSSV là 6351071011 thành công-*

## Chức năng tìm kiếm sinh viên

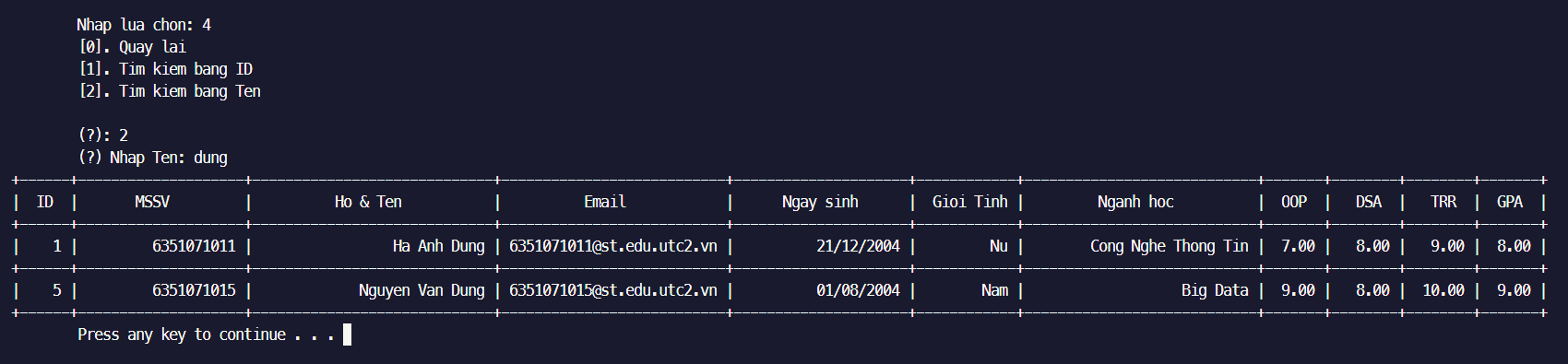




*-Tìm kiếm sinh viên có ID là 1 thành công-*



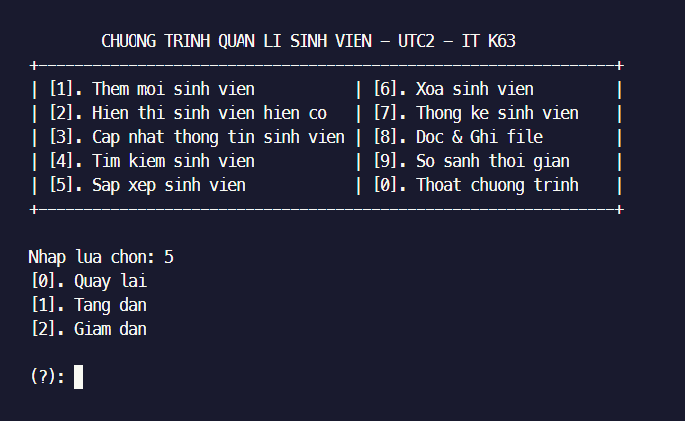
*-Tìm kiếm sinh viên có ID là 5 không thành công-*

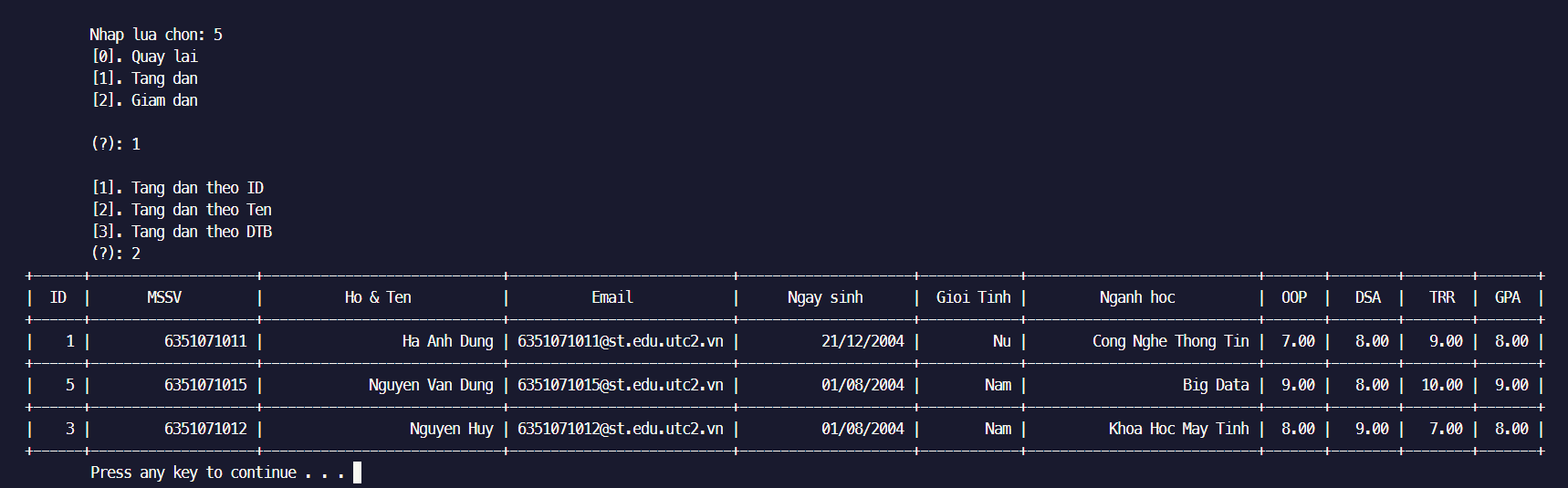


*-Tìm kiếm sinh viên có tên “dung” thành công-*

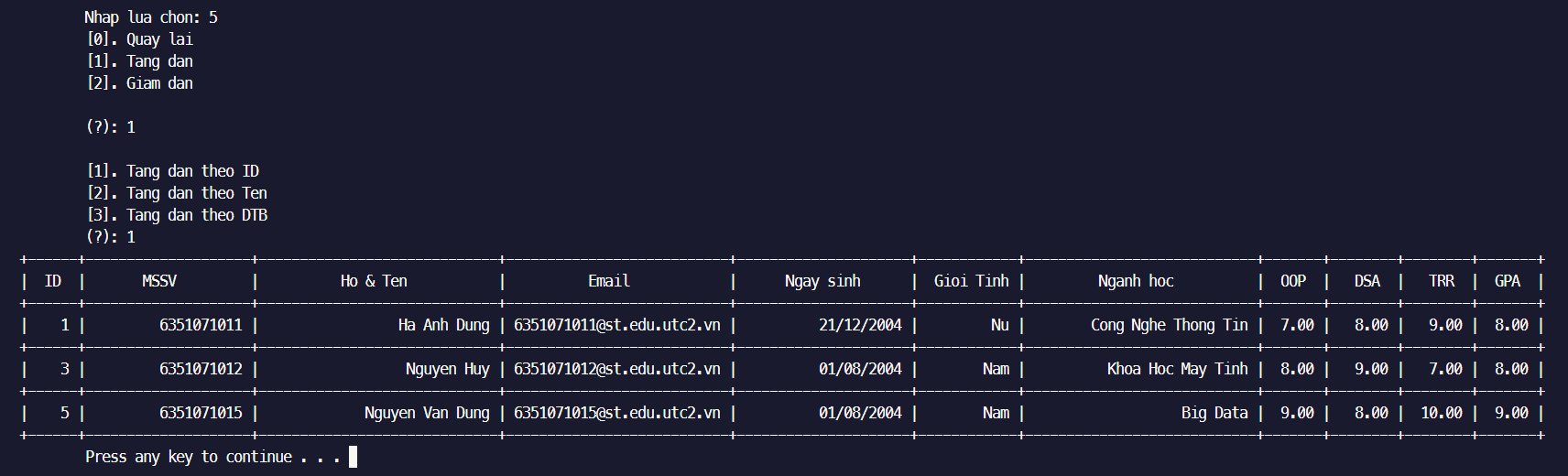
*Tìm kiếm sinh viên có tên “nhat” không thành công*

## Chức năng sắp xếp sinh viên

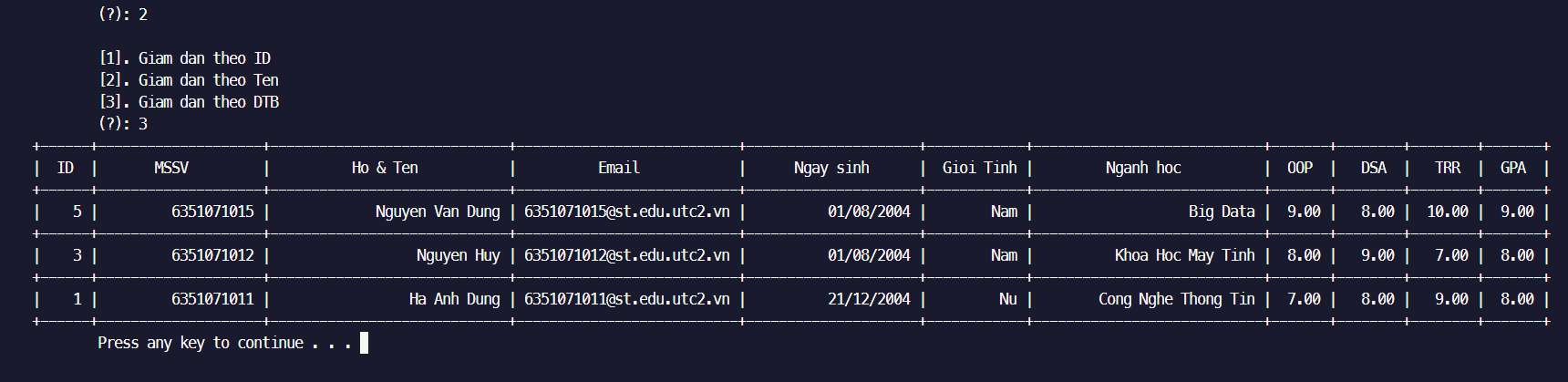




*-Sắp xếp sinh viên tăng dần theo tên-*

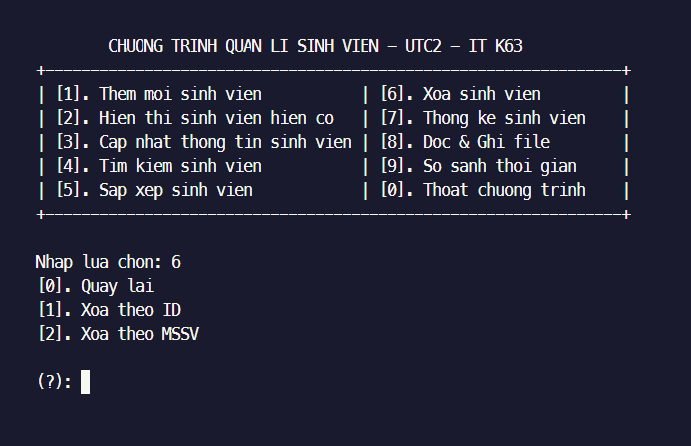


*-Sắp xếp sinh viên tăng dần theo ID-*



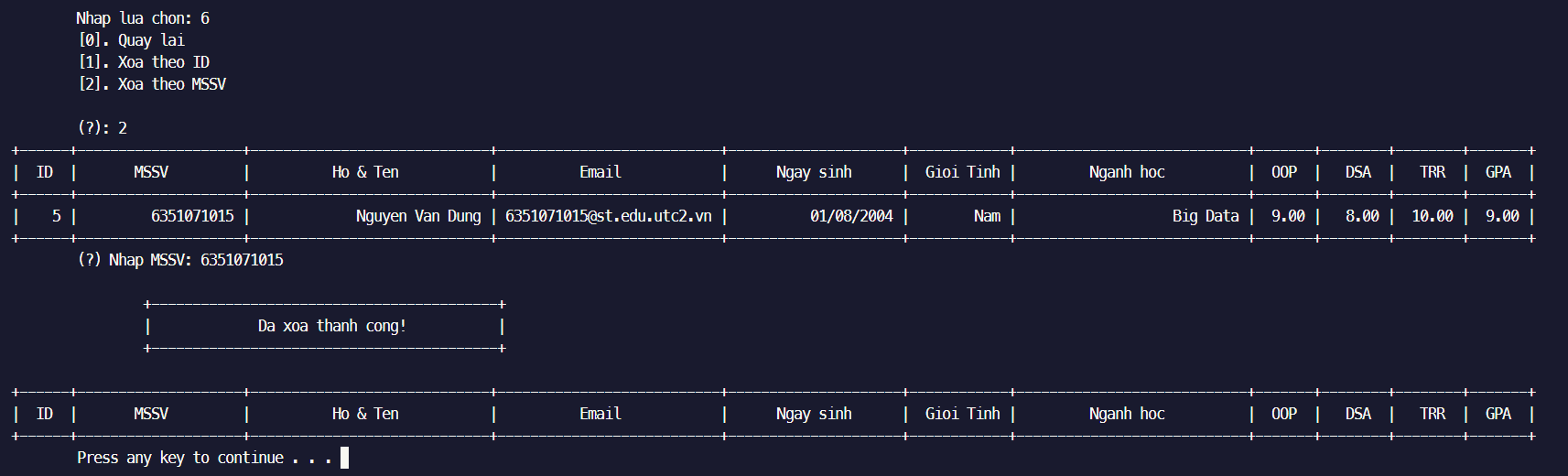
*-Sắp xếp sinh viên giảm dần theo điểm trung bình-*

## Chức năng xóa sinh viên



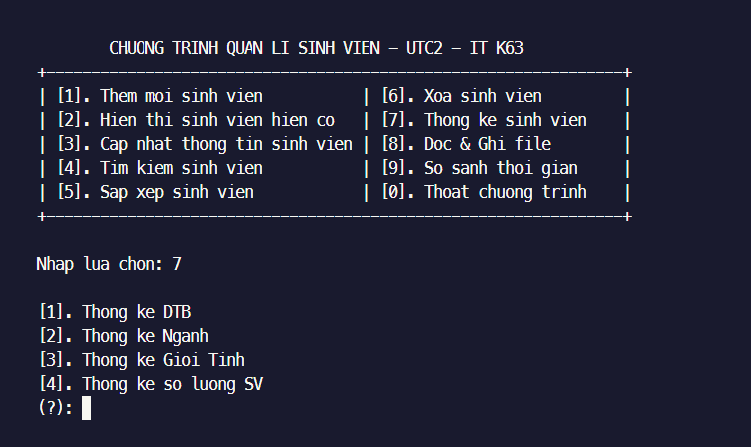


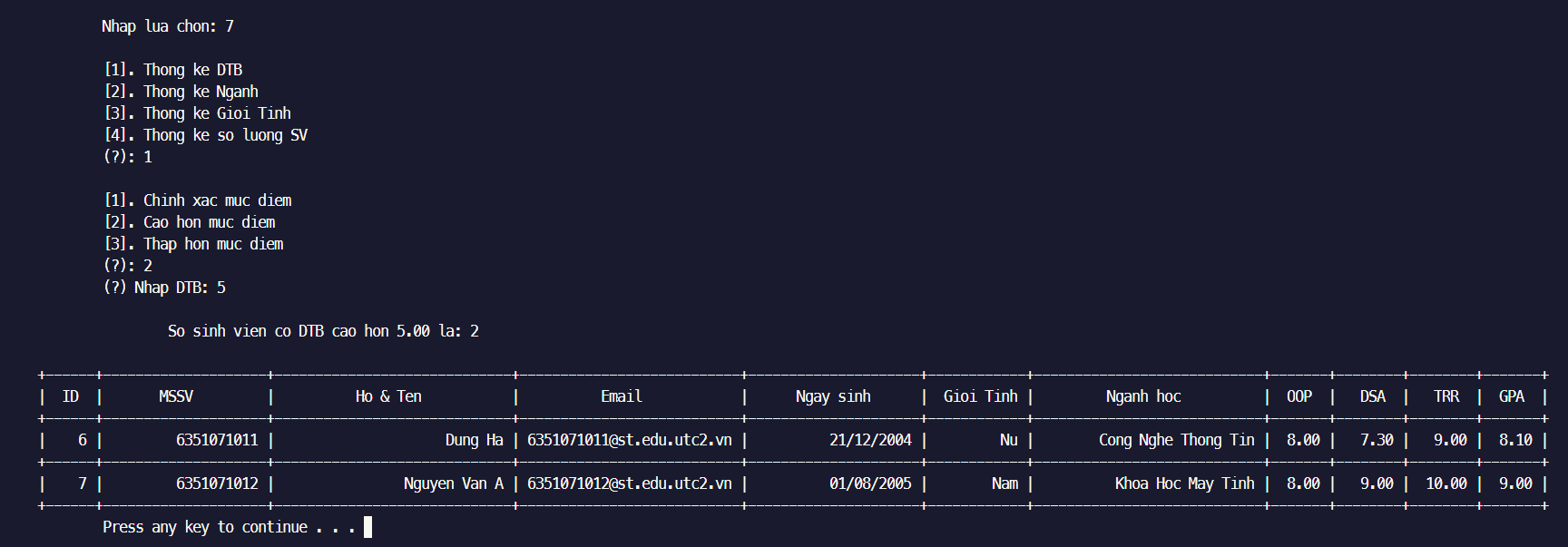
*-Xóa 2 sinh viên có ID là 1, 3 thành công-*



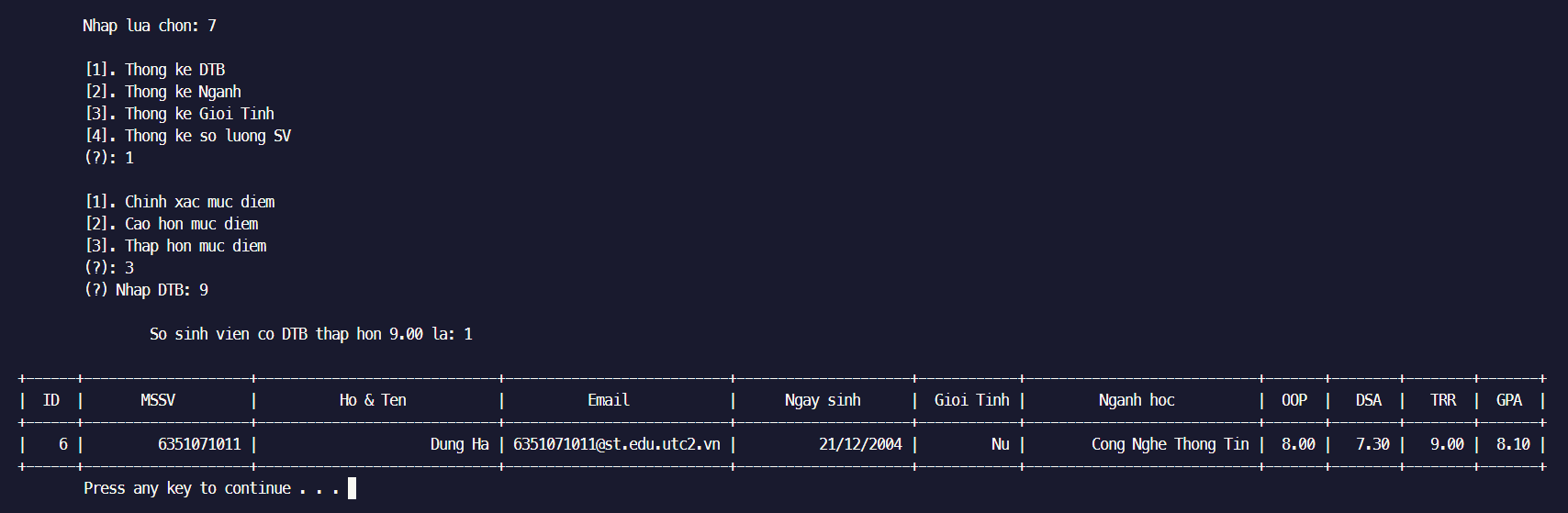
*-Xóa 1 sinh viên có MSSV là 6351071015 thành công-*

## Chức năng thống kê sinh viên

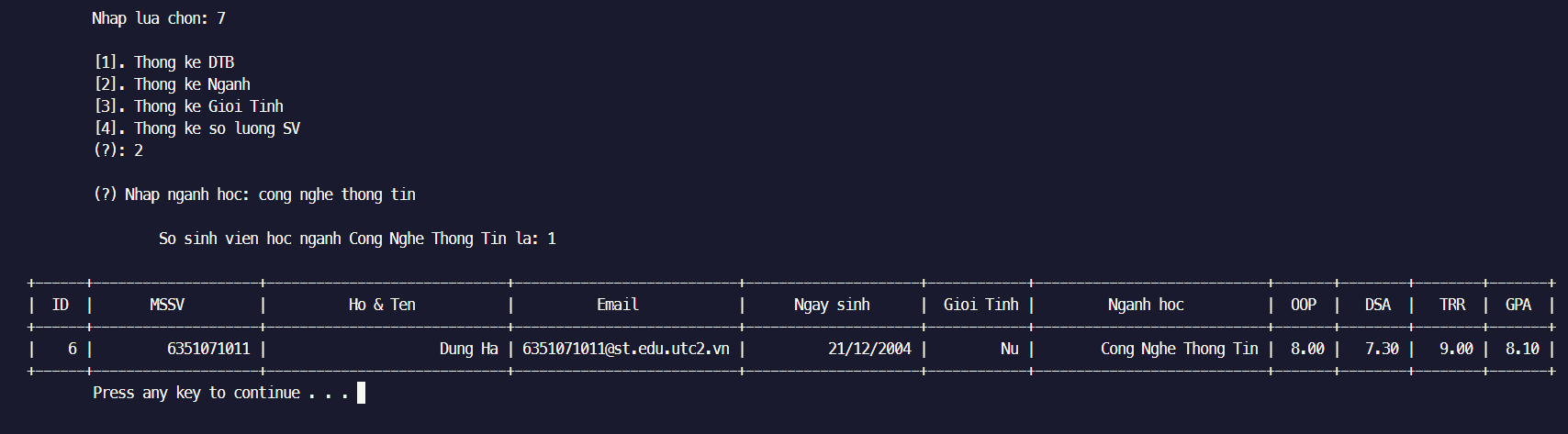




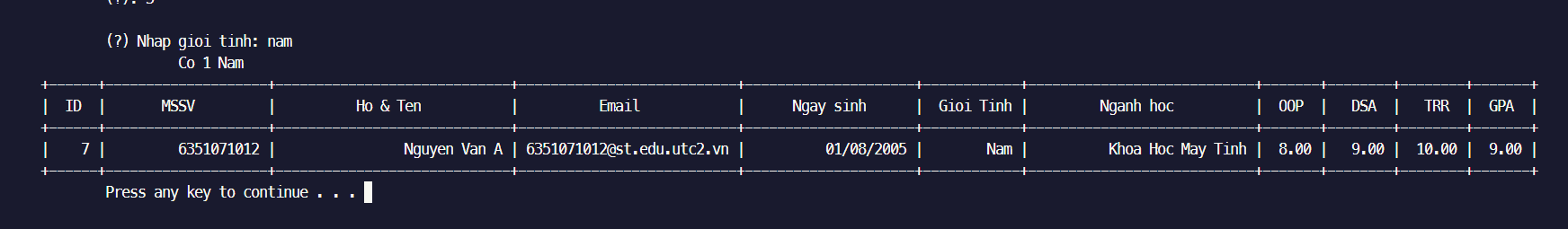
*-Thống kê các sinh viên có điểm trung bình lớn hơn 5-*



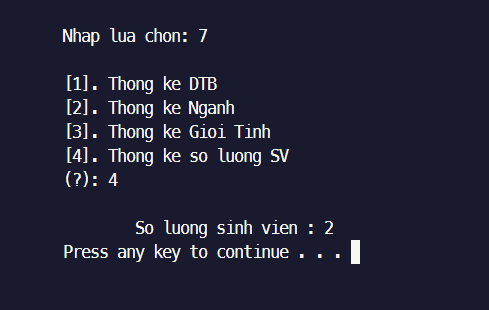
*-Thống kê các sinh viên có điểm trung bình thấp hơn 9-*



*-Thống kê các sinh viên học ngành “Cong Nghe Thong Tin”-*

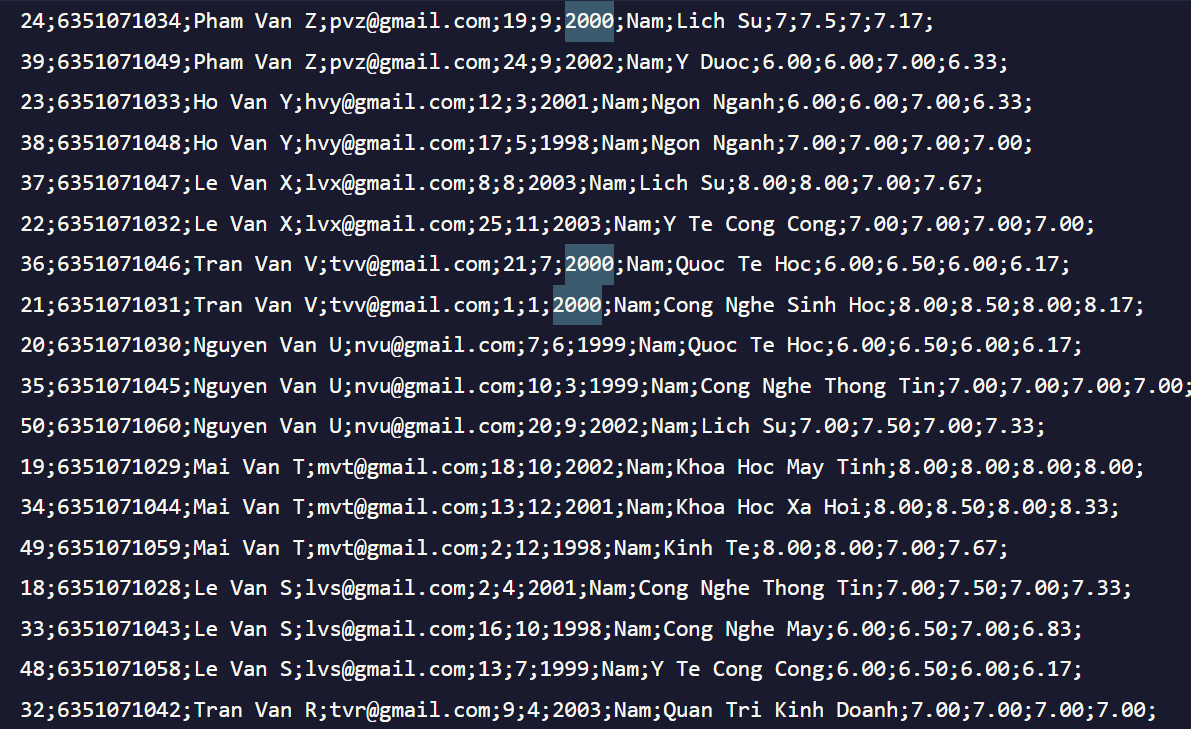


*-Thống kê sinh viên có giới tính là Nam-*

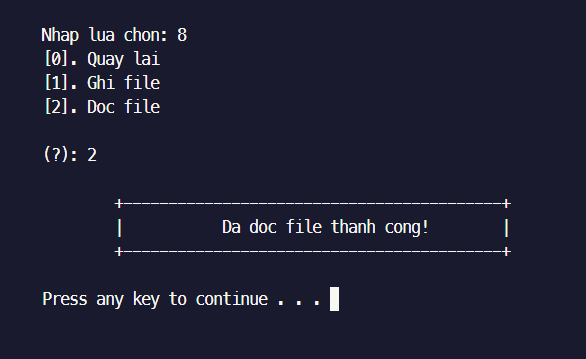


*-Số lượng sinh viên có trong hệ thống-*

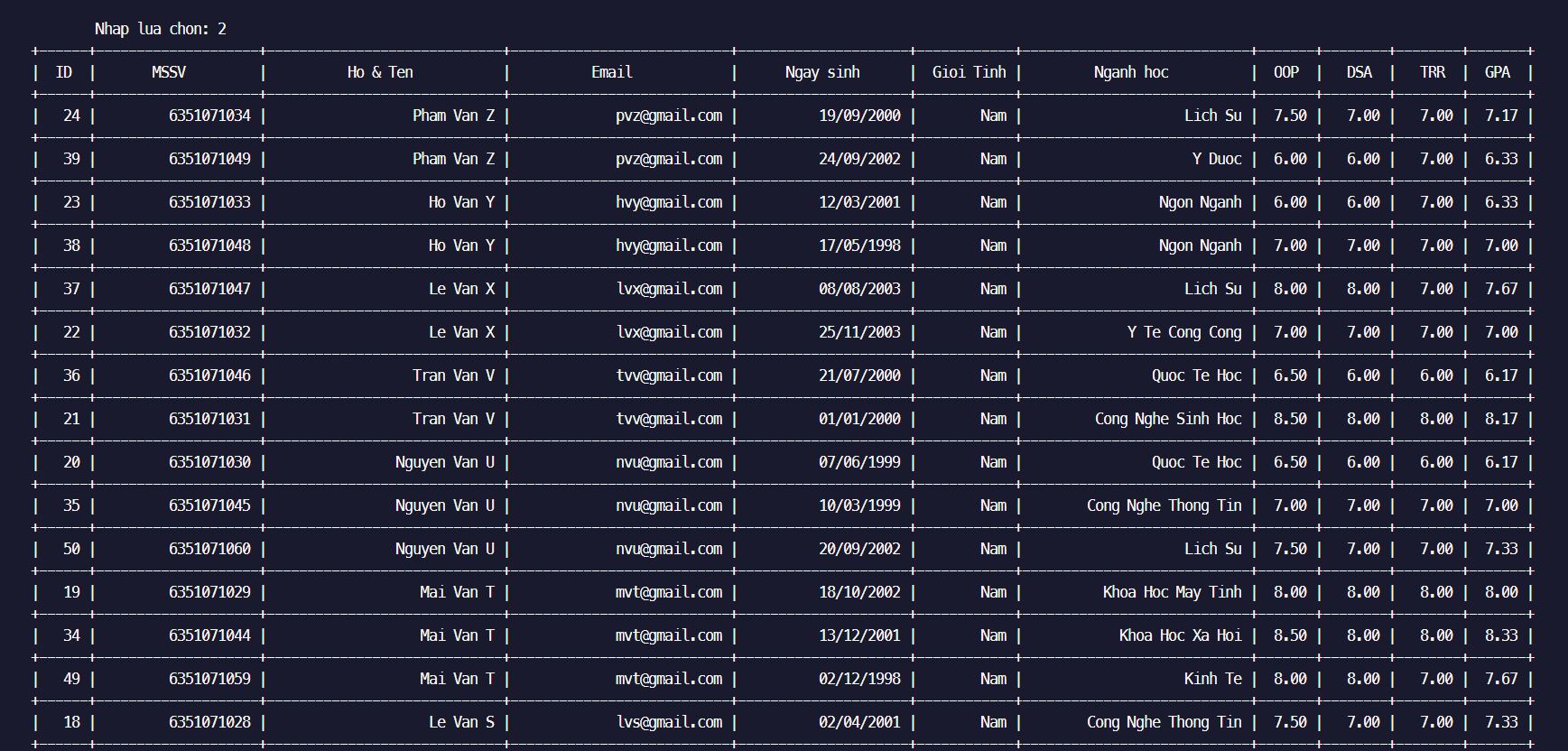
## Chức năng đọc & ghi file



*-File dữ liệu-*



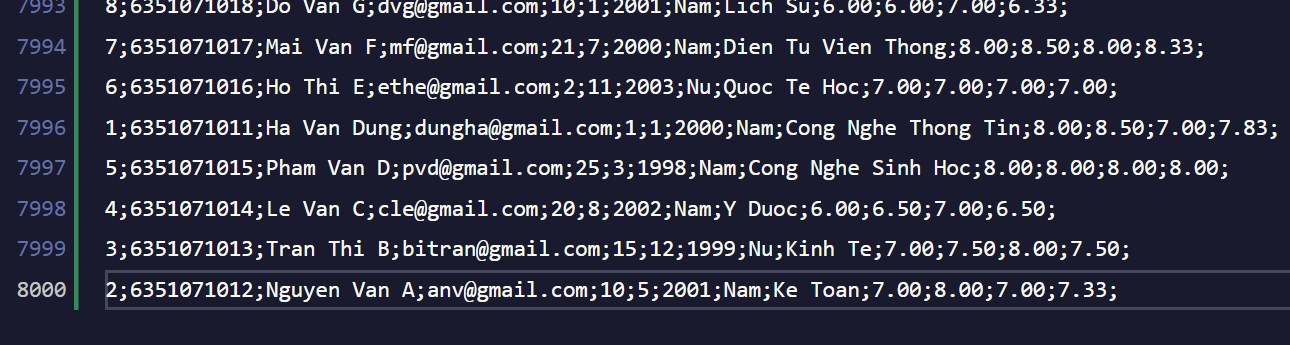
*-Đọc file thành công-*



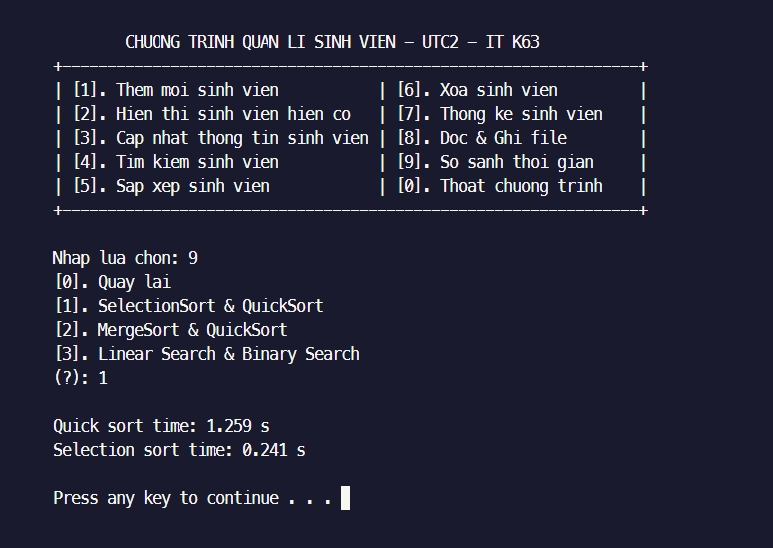
*-Sau khi đọc file thành công vào in ra màn hình kết quả-*

## Chức năng so sánh thời gian chạy của các thuật toán

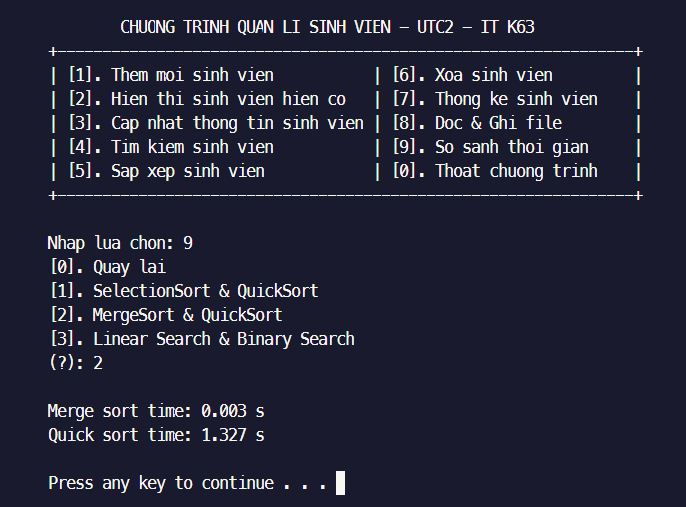
[Link data](https://github.com/VinCentProgrammer/Project_DataStructure_Algorithms/blob/master/Source/data/origin-data.txt)



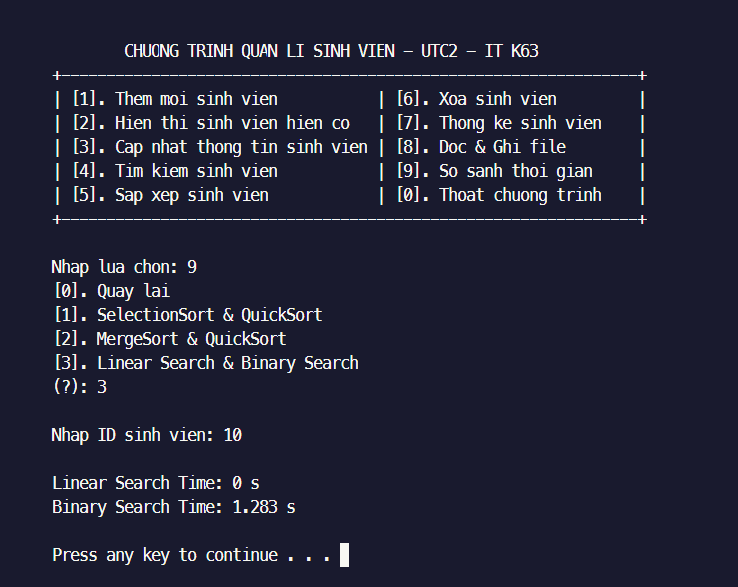
*-Dữ liệu ban đầu-*

**

*-Selection sort & Quick sort-*

**

*-Merge sort & Quicksort-*

**

*-Linear Search & Binary Search-*

*\*Quicksort:*

* Thời gian trung bình: O(n \* log(n))
* Thời gian tốt nhất (khi danh sách đã sắp xếp hoặc gần sắp xếp): O(n \* log(n))
* Thời gian xấu nhất (khi danh sách là một dãy tăng hoặc giảm): O(n^2)

Quicksort là một thuật toán sắp xếp hiệu quả với độ phức tạp trung bình là O(n \* log(n)). Nó hoạt động tốt trong hầu hết các trường hợp, đặc biệt là khi danh sách lớn.

*\*Selection Sort:*

* Thời gian trung bình: O(n^2)
* Thời gian tốt nhất: O(n^2)
* Thời gian xấu nhất: O(n^2)

Selection Sort là một thuật toán sắp xếp đơn giản, nhưng có độ phức tạp thời gian là O(n^2) trong tất cả các trường hợp. Nó thường hiệu quả hơn khi danh sách nhỏ.

*\*Merge sort*

* Thời gian chạy của thuật toán merge sort là O(n log n). Merge sort là một thuật toán sắp xếp ổn định, chia để trị, và tách thành các phần con rồi ghép lại để sắp xếp dãy số. Thời gian chạy O(n log n) là tốt nhất có thể đạt được với một thuật toán sắp xếp so sánh, và nó không bị ảnh hưởng bởi sự sắp xếp ban đầu của dãy số.
* Thuật toán merge sort làm việc theo cách chia dãy số thành hai phần bằng cách chia đôi dãy ban đầu, sau đó đệ quy sắp xếp từng phần con. Cuối cùng, nó ghép hai phần con đã sắp xếp lại thành một dãy số đã sắp xếp. Quá trình này lặp lại cho đến khi toàn bộ dãy số đã được sắp xếp.

Do đó, thời gian chạy của merge sort luôn là O(n log n) bất kể dãy số ban đầu là gì, và nó là một trong những thuật toán sắp xếp hiệu quả trong nhiều trường hợp.

**PHẦN 3. KẾT QUẢ VÀ KIẾN NGHỊ**

1. Kết quả đạt được

*Quản lý thông tin sinh viên:* Chương trình cho phép thêm, sửa, xóa và tìm kiếm thông tin của các sinh viên bao gồm tên, mã số sinh viên, ngày tháng năm sinh, địa chỉ, số điện thoại, email, v.v.

*Quản lý thông tin khóa học:* Chương trình cho phép thêm, sửa, xóa và tìm kiếm thông tin về các khóa học, bao gồm tên khóa học, mô tả, ngày bắt đầu và kết thúc, giảng viên, số lượng sinh viên đăng ký, v.v.

Đăng ký và quản lý lịch học: Sinh viên có thể đăng ký cho các khóa học, và chương trình sẽ theo dõi lịch học của họ. Học viên có thể xem lịch học của mình và thông báo về thay đổi lịch trình (nếu có).

*Quản lý điểm số:* Chương trình cho phép nhập và quản lý điểm số cho từng khóa học. Sinh viên và giảng viên có thể xem điểm số của họ.

Tạo báo cáo và thống kê: Chương trình cung cấp khả năng tạo báo cáo và thống kê về tiến trình học tập của sinh viên, điểm số, và thông tin khóa học.

*Tích hợp hệ thống:* Kết quả đạt được có thể bao gồm tích hợp với các hệ thống khác như hệ thống tài chính để quản lý học phí, hệ thống email để gửi thông báo, v.v.

*Bảo mật dữ liệu:* Đảm bảo rằng thông tin cá nhân của sinh viên và dữ liệu quản lý không bị đánh cắp hoặc truy cập trái phép.

2. Kiến nghị

*Xác định rõ mục tiêu:* Trước khi bắt tay vào phát triển, hãy xác định rõ mục tiêu và yêu cầu cụ thể cho chương trình. Điều này sẽ giúp bạn xác định rõ các tính năng và chức năng cần phát triển.

*Tùy chỉnh theo nhu cầu:* Đảm bảo rằng chương trình được tùy chỉnh theo nhu cầu cụ thể của tổ chức hoặc trường học. Không cần phải tích hợp các tính năng không cần thiết hoặc quá phức tạp.

*Thiết kế giao diện dễ sử dụng:* Hãy đảm bảo rằng giao diện người dùng (UI) của chương trình dễ sử dụng và thân thiện. Điều này sẽ giúp người dùng dễ dàng tương tác với hệ thống.

*Đảm bảo bảo mật dữ liệu:* Dữ liệu của sinh viên là quan trọng và cần phải được bảo vệ chặt chẽ. Đảm bảo rằng hệ thống có các biện pháp bảo mật đủ mạnh để ngăn chặn truy cập trái phép.

*Hỗ trợ đa nền tảng:* Đảm bảo rằng chương trình có thể hoạt động trên nhiều nền tảng, bao gồm máy tính cá nhân, điện thoại di động và máy tính bảng.

*Duyệt kỹ thuật:* Sử dụng các công nghệ hiện đại và tiến bộ để xây dựng chương trình. Điều này giúp cải thiện hiệu suất và độ tin cậy.

*Kiểm tra và triển khai thử nghiệm:* Trước khi đưa chương trình vào sử dụng thực tế, hãy tiến hành kiểm tra kỹ thuật và triển khai thử nghiệm để đảm bảo tính ổn định và hoạt động tốt.

*Đào tạo người dùng:* Cung cấp đào tạo cho người dùng cuối, bao gồm sinh viên, giảng viên và nhân viên quản lý, để họ có thể sử dụng chương trình một cách hiệu quả.

*Liên tục cập nhật và bảo trì:* Đảm bảo rằng chương trình được liên tục cập nhật và bảo trì để đáp ứng các yêu cầu và thay đổi mới.

**PHỤ LỤC**

Phụ lục 1: Hướng dẫn cài đặt

Phụ lục 2: Hướng dẫn sử dụng

**TÀI LIỆU THAM KHẢO**

1. Davis, Stephen R. "C++ cho người mới bắt đầu." For Dummies, 2014.
2. Stroustrup, Bjarne. "The C++ Programming Language." Addison-Wesley, 2013.
3. <http://www.cplusplus.com/> mọi vấn đề trong C++ được tổng hợp trong trang web.