**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**

**ĐỀ TÀI: DỰ ÁN QUẢN LÝ SÂN BÓNG**

Sinh viên thực hiện: TRẦN TRỌNG NHÂN

NGUYỄN TRẦN THÔNG THÁI

PHẠM LONG VỸ

Giảng viên hướng dẫn: TRẦN THỊ DUNG

Lớp : KỸ THUẬT ĐIỆN TỬ- VIỄN THÔNG

Khoá :KHÓA 64

Tp. Hồ Chí Minh, năm 2024

**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**

**ĐỀ TÀI: DỰ ÁN QUẢN LÝ SÂN BÓNG**

Sinh viên thực hiện: TRẦN TRỌNG NHÂN

NGUYỄN TRẦN THÔNG THÁI

PHẠM LONG VỸ

Giảng viên hướng dẫn: TRẦN THỊ DUNG

Lớp : KỸ THUẬT ĐIỆN TỬ- VIỄN THÔNG

Khoá :KHÓA 64

Tp. Hồ Chí Minh, năm 2024

**LỜI CẢM ƠN**

Qua quãng thời gian học tập và rèn luyện tại trường Trường Đại học Giao thông Vận tải phân hiệu tại TP HCM, đến nay chúng 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.

Chúng muốn bày tỏ lòng biết ơn đến tất cả các thầy cô giáo Bộ môn Công Nghệ Thông Tin, cũng như 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.

Chúng em nhận được sự truyền đạt kiến thức quý giá và nhờ sự quan tâm, giúp đỡ, chỉ bảo tận tình của cô Nguyễn Thị Dung trong môn học Cấu Trúc Dữ Liệu Và Giải Thuật, đã giúp chúng em không chỉ có thêm kiến thức quý giá phục vụ cho việc học và còn cả hành trang để chúng em vững bước vào đời.

Lần đầu chúng em thực hiện viết báo cáo nên có nhiều hạn chế và còn nhiều điều thiếu sót. Mong nhận được sự góp ý và đóng góp quý báu của cô để chúng em được hoàn thiện hơn đồng thời có thời gian bổ sung và nâng cao kỹ năng, kiến thức của mình.

Cuối cùng chúng em xin kính chúc cô dồi dào sức khỏe, thành công trong sự nghiệp cao quý của mình.

Chúng em xin chân thành cảm ơn!

MỤC LỤC

[CHƯƠNG 1. GIỚI THIỆU Ý TƯỞNG VÀ CƠ SỞ LÝ THUYẾT 1](#_Toc182119761)

[1.1: Ý tưởng thiết kế và nhu cầu người dùng 1](#_Toc182119762)

[1.2: Cơ sở lý thuyết về cấu trúc danh sách liên kết đơn 1](#_Toc182119763)

[1.2.1: Khái niệm 1](#_Toc182119764)

[1.2.2: Tính chất 2](#_Toc182119765)

[1.2.3: Cấu trúc node trong DSLK đơn. 2](#_Toc182119766)

[1.2.4: Thêm phần tử vào đầu danh sách ( addFirst ). 3](#_Toc182119767)

[1.2.5:Thêm phần tử vào cuối danh sách ( addLast ). 4](#_Toc182119768)

[1.2.6:Thêm vào sau node bất kỳ ( InsertAfter ). 5](#_Toc182119769)

[1.2.7: Xóa phần tử đầu khỏi danh sách ( removeFirst ). 6](#_Toc182119770)

[1.2.8: Xóa phần tử cuối khỏi danh sách ( removeLast ). 7](#_Toc182119771)

[1.2.9: Xóa ở sau node bất kì. 8](#_Toc182119772)

[1.3: Cơ sở lý thuyết về sắp xếp. 10](#_Toc182119773)

[1.3.1: Khái niệm 10](#_Toc182119774)

[1.3.2: Sắp xếp nổi bọt (Bubble Sort). 10](#_Toc182119775)

[1.4: Cơ sở lý thuyết về tìm kiếm. 13](#_Toc182119776)

[1.4.1: Khái niệm. 13](#_Toc182119777)

[1.4.2: Tìm kiếm tuần tự (Linear Search). 13](#_Toc182119778)

[CHƯƠNG 2. XÂY DỰNG CHƯƠNG TRÌNH 14](#_Toc182119779)

[2.1 Gới thiệu thuật toán. 15](#_Toc182119780)

[2.1.1: Các thư viện trong chương trình. 15](#_Toc182119781)

[2.2: Xây dựng chương trình. 15](#_Toc182119782)

[2.2.1: Khai báo cấu trúc node: 17](#_Toc182119783)

[2.2.2: Khai báo danh sách: 17](#_Toc182119784)

[2.2.3: Các hàm khai báo của cấu trúc List: 17](#_Toc182119785)

[2.2.4: Hàm khởi tạo: 18](#_Toc182119786)

[2.2.5: Hàm tạo node mới: 18](#_Toc182119787)

[2.2.6: Hàm thêm sân bóng đầu: 18](#_Toc182119788)

[2.2.7: Hàm thêm sân bóng cuối: 19](#_Toc182119789)

[2.2.8: Hàm thêm sân bóng bất kỳ: 19](#_Toc182119790)

[2.2.9: Hàm liệt kê số sân còn hoạt động: 21](#_Toc182119791)

[2.2.10: Hàm xóa sân bóng đầu danh sách: 21](#_Toc182119792)

[2.2.11: Hàm xóa sân bóng cuối danh sách: 22](#_Toc182119793)

[2.2.12: Hàm kiểm tra sân bóng có tồn tại hay không: 22](#_Toc182119794)

[2.2.13: Hàm sắp xếp theo giá sân. 24](#_Toc182119795)

[2.2.14: Sắp xếp theo loại sân. 26](#_Toc182119796)

[2.3: Giao diện chương trình. 27](#_Toc182119797)

[CHƯƠNG 3. KẾT QUẢ CHƯƠNG TRÌNH 28](#_Toc182119798)

DANH MỤC CHỮ VIẾT TẮT

|  |  |  |
| --- | --- | --- |
| STT | Mô tả | Ý nghĩa |
| 1 | CNTT | Công Nghệ Thông Tin |
| 2 | DSLK | Danh Sách Liên Kết |

BẢNG BIỂU, SƠ ĐỒ, HÌNH VẼ

[Hình 1: Hình ảnh minh họa về danh sách liên kết đơn. 2](#_Toc181949100)

[Hình 2: Mô hình danh sách liên kết đơn. 2](#_Toc181949101)

[Hình 3: Cấu trúc node trong DSLK đơn. 3](#_Toc181949102)

[Hình 4: Mô hình thêm phần tử vào đầu DSLK đơn. 3](#_Toc181949103)

[Hình 5: Cấu trúc thêm phần tử vào đầu DSLK đơn. 4](#_Toc181949104)

[Hình 6: Mô hình thêm phần tử vào cuối DSLK đơn. 4](#_Toc181949105)

[Hình 7: Cấu trúc thêm phần tử vào cuối DSLK đơn. 5](#_Toc181949106)

[Hình 8: Mô hình thêm phần tử vào sau một nút trong DSLK đơn. 5](#_Toc181949107)

[Hình 9: Cấu trúc thêm phần tử vào sau một node DSLK đơn. 6](#_Toc181949108)

[Hình 10: Mô hình xóa phần tử đầu trong DSLK đơn. 6](#_Toc181949109)

[Hình 11: Cấu trúc xóa phần tử đầu trong DSLK đơn. 7](#_Toc181949110)

[Hình 12: Mô hình xóa phần tử cuối cùng trong DSLK đơn. 7](#_Toc181949111)

[Hình 13: Cấu trúc xóa phần tử cuối cùng trong DSLK đơn. 8](#_Toc181949112)

[Hình 14: Mô hình xóa node bất kỳ trong DSLK đơn. 9](#_Toc181949113)

[Hình 15: Cấu trúc xóa phần tử ở sau node bất kì trong DSLK đơn. 9](#_Toc181949114)

[Hình 16: Cấu trúc sắp xếp nổi bọt (Bubble Sort). 13](#_Toc181949115)

[Hình 17: Cấu trúc nội dung. 16](#_Toc181949116)

[Hình 18: Menu chính. 27](#_Toc181949117)

[Hình 19: Danh sách sân bóng. 28](#_Toc181949118)

[Hình 20: Danh sách giờ và thành tiền. 28](#_Toc181949119)

[Hình 21: Thêm sân bóng đầu, cuối và sau vị trí 3. 28](#_Toc181949120)

[Hình 22: Xóa sân bóng đầu và cuối. 29](#_Toc181949121)

[Hình 23: Cập nhật danh sách đặt sân cho CN1. 29](#_Toc181949122)

[Hình 24: Sắp xếp theo giá sân tăng dần. 29](#_Toc181949123)

[Hình 25: Sắp xếp theo loại sân. 30](#_Toc181949124)

# 

# CHƯƠNG 1. GIỚI THIỆU Ý TƯỞNG VÀ CƠ SỞ LÝ THUYẾT

### 1.1: Ý tưởng thiết kế và nhu cầu người dùng

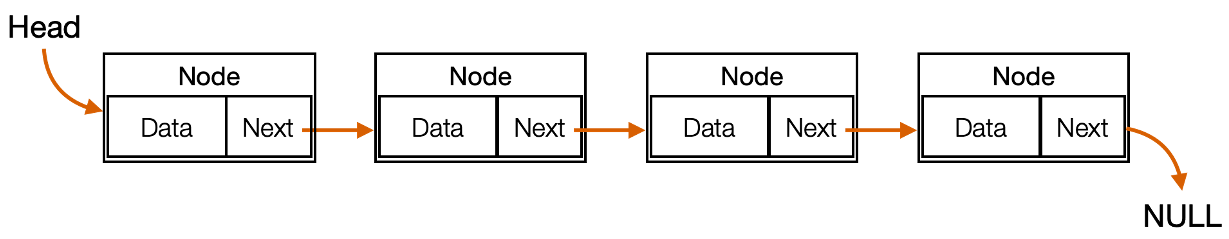
Trong thời đại số hiện nay, ứng dụng công nghệ thông tin (CNTT) trong quản lý đang trở thành một yếu tố quyết định cho sự phát triển bền vững của các tổ chức và doanh nghiệp. CNTT giúp tự động hóa quy trình làm việc, từ quản lý nhân sự, tài chính đến chuỗi cung ứng, giảm thiểu thời gian và chi phí. Các phần mềm quản lý thông minh cho phép phân tích dữ liệu lớn, giúp các nhà quản lý đưa ra quyết định chính xác hơn. Hơn nữa, công nghệ đám mây đã tạo điều kiện thuận lợi cho việc lưu trữ và chia sẻ thông tin, nâng cao khả năng phối hợp giữa các bộ phận trong tổ chức. Nhờ đó, việc theo dõi tiến độ công việc, đánh giá hiệu suất và phát hiện sớm các vấn đề trở nên dễ dàng hơn bao giờ hết, từ đó tối ưu hóa hoạt động và nâng cao hiệu quả quản lý.

Vì vậy chúng em đã thực hiện và thiết kế dự án “ QUẢN LÝ SÂN BÓNG ”. Dự án mang đến giải phấp hiệu quả cho chủ sân, giúp theo dõi, quản lý lịch đặt sân, tính thu nhập, giúp tiết kiệm thời gian, dao diện dễ sử dụng, tối ưu hóa quy trình hoạt động, mang đến sự chuyên nghiệp và hiện đại.

### 1.2: Cơ sở lý thuyết về cấu trúc danh sách liên kết đơn

### 1.2.1: Khái niệm

Danh sách liên kết đơn dùng để chỉ một cấu trúc dữ liệu di động hay còn có thể hình dung như một danh sách mà trong đó mỗi phần tử đều liên kết với phần tử đứng sau nó.



Hình 1: Hình ảnh minh họa về danh sách liên kết đơn.

### 1.2.2: Tính chất

- Mở rộng và thu hẹp một cách linh hoạt

- Các phần tử trong DSLK gọi là note và được cấp phát động khi cần

- Số lượng phần tử trong DSLK phụ thuộc vào bộ nhớ heap

- Dễ dàng chèn và xóa phần tử

- Mỗi node trong DSLK cần có thêm một con trỏ để lưu liên kết.

Mỗi phần tử trong DSLK được gọi là một node, node này sẽ lưu thông tin dữ liệu ( ví dụ: một chuỗi ký tự, một số nguyên, một sinh viên,…) và ngoài ra cần có liên kết, liên kết này giúp các node có thể liên lạc với nhau. Mỗi node sẽ lưu thêm địa chỉ của node phía sau nó trong DSLK thông qua một thuộc tính con trỏ. Node cuối cùng trong danh sách liên kết thì phần liên kết của nó sẽ lưu con trỏ NULL.

A diagram of a data flow

Description automatically generated

Hình 2: Mô hình danh sách liên kết đơn.

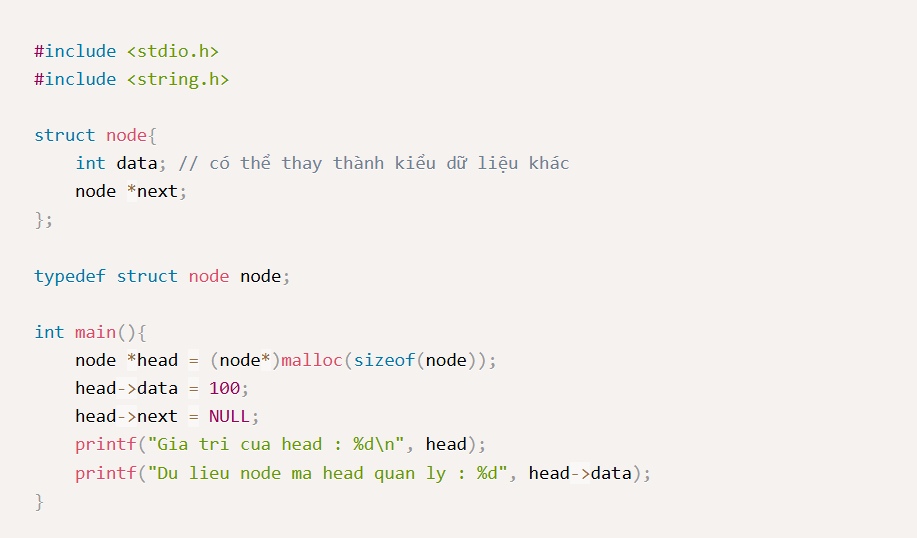
### 1.2.3: Cấu trúc node trong DSLK đơn.

Một danh sách được tạo lên từ nhiều node. Để xây dựng một node cho DSLK ta có thể dùng struct hoặc class, mỗi khi ta cần tạo ra một node mới trong DSLK thì ta cần cấp phát động, trong C bạn sử dụng malloc còn trong C++ ta sử dụng toán tử new.

Một node trong DSLK thường có 2 phần:

- Dữ liệu của node lưu thông tin của một node, có thể là kiểu dữ liệu như số, chuỗi, sinh viên, ...

- Phần liên kết – Đây là một con trỏ để lưu địa chỉ của node kế tiếp nó trong DSLK, thường đặt tên là next.



Hình 3: Cấu trúc hàm node trong DSLK đơn.

### 1.2.4: Thêm phần tử vào đầu danh sách ( addFirst ).

Đầu tiên ta cần xác định xem danh sách liên kết đơn này có rỗng hay không. Nếu danh sách đó rỗng, ta chỉ cần gán head và tail của danh sách bằng node đó. Ngược lại nếu danh

sách không rỗng, ta thực hiện trỏ thành phần liên kết vào head, sau đó gán lại head bằng node mới.

**A diagram of a diagram

Description automatically generated**

Hình 4: Mô hình thêm phần tử vào đầu DSLK đơn.

Như trong hình trên, chúng ta thêm node có data bằng 0 vào danh sách. Ta thực hiện trỏ next của node đó vào head của danh sách (chính là node đầu tiên của danh sách có data bằng 1), sau đó ta trỏ head vào node có data 0 vừa được thêm.



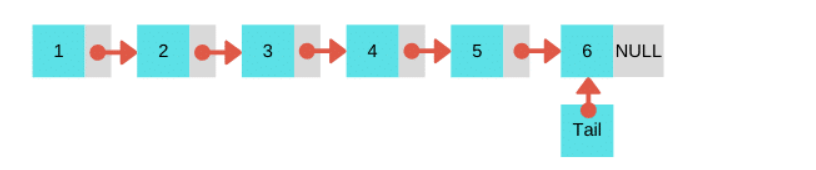
**A screen shot of a computer code

Description automatically generated**

Hình 5: Cấu trúc thêm phần tử vào đầu DSLK đơn.

### 1.2.5:Thêm phần tử vào cuối danh sách ( addLast ).

Tương tự, để thêm node vào cuối danh sách, đầu tiên ta kiểm tra xem danh sách rỗng hay không, rỗng thì gán head và tail đều bằng node mới. Nếu không rỗng, ta thực hiện trỏ tail -> next vào node mới, sau đó gán lại tail bằng node mới (vì bây giờ node mới thêm chính là tail).



Hình 6: Mô hình thêm phần tử vào cuối DSLK đơn.

Trong hình trên, chúng ta thực hiện thêm node có data bằng 6 vào danh sách. Tail hiện tại là node có data 5, thực hiện gán tail->next bằng node mới để nối thêm nó vào đuôi danh sách, lúc này node mới trở thành phần tử cuối danh sách nên ta gán tail lại bằng node mới.

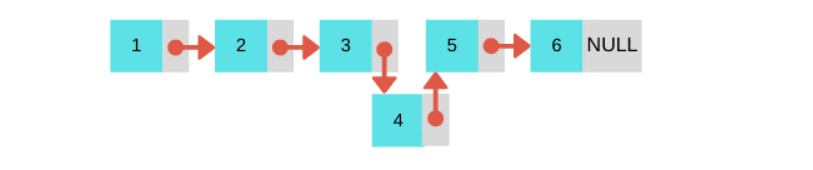
**A screenshot of a computer code

Description automatically generated**

Hình 7: Cấu trúc thêm phần tử vào cuối DSLK đơn.

### 1.2.6:Thêm vào sau node bất kỳ ( InsertAfter ).

Để thêm một node p vào sau node q bất kỳ, đầu tiên ta cần kiếm tra xem node q có NULL hay không, nếu node q là NULL tức là danh sách rỗng, vậy thì ta sẽ thêm vào đầu danh sách. Nếu node q không NULL, tức là tồn tại trong danh sách, ta thực hiện trỏ p->next = q->next, sau đó q->next = p. Tiếp theo chúng ta kiểm tra xem node q trước đó có phải là node cuối hay không, nếu node q là node cuối thì thêm p vào, p sẽ thành node cuối nên ta gán lại tail = p.



Hình 8: Mô hình thêm phần tử vào sau một nút trong DSLK đơn.

Trong hình trên, ta thêm node có data bằng 4 (node p) vào sau node có data bằng 3 (node q). Ta trỏ next của node p vào next của node q tức là node có data bằng 5, sau đó trỏ next của node q vào node p vậy là node p đã được thêm vào danh sách.

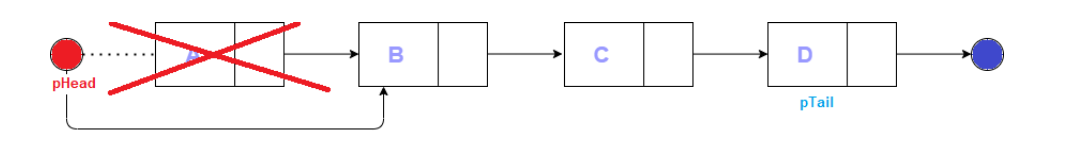
A screenshot of a computer program

Description automatically generated

Hình 9: Cấu trúc thêm phần tử vào sau một node DSLK đơn.

### 1.2.7: Xóa phần tử đầu khỏi danh sách ( removeFirst ).

Để xóa phần tử ở đầu danh sách, ta kiểm tra xem danh sách đó có rỗng hay không, nếu rỗng, ta không cần xóa, trả về kết quả là 0. Nếu danh sách không rỗng, ta thực hiện lưu node head lại, sau đó gán head bằng next của node head, sau đó xóa node head đi. Tiếp theo ta cần kiểm tra xem danh sách vừa bị xóa đi node head có rỗng hay không, nếu rỗng ta gán lại tail bằng NULL luôn sau đó trả về kết quả 1.



Hình 10: Mô hình xóa phần tử đầu trong DSLK đơn.

A screen shot of a computer code

Description automatically generated

Hình 11: Cấu trúc xóa phần tử đầu trong DSLK đơn.

### 1.2.8: Xóa phần tử cuối khỏi danh sách ( removeLast ).

Để xóa phần tử cuối cùng của danh sách, ta Kiểm tra nếu danh sách rỗng hay không, nếu rỗng, ta không cần xóa, trả về kết quả là 0. Nếu danh sách không rỗng, ta khởi tạo con trỏ a trỏ đến phần tử đầu tiên, sau đó ta duyệt đến phần tử trước phần tử cuối, cập nhật con trỏ next của phần tử trước cuối thành NULL, cập nhật tail để trỏ đến phần tử cuối, giảm kích thước danh sách.

A diagram of a flowchart

Description automatically generated

Hình 12: Mô hình xóa phần tử cuối cùng trong DSLK đơn.

**A screenshot of a computer program

Description automatically generated**

Hình 13: Cấu trúc xóa phần tử cuối cùng trong DSLK đơn.

### 1.2.9: Xóa ở sau node bất kì.

Để xóa một node p sau node q bất kỳ, ta kiểm tra xem node q có NULL hay không, nếu node q NULL thì không tồn tại trong danh sách, do đó trả về 0, không xóa. Nếu node q khác NULL nhưng next của q là NULL, tức là p bằng NULL thì không xóa, trả về 0 (do sau q không có node nào cả, q là tail). Nếu node p tồn tại, ta thực hiện kiểm tra xem node p có phải là tail hay không, nếu node p là tail thì gán lại tail là q, tức là node trước đó để xóa node p đi.

A diagram of a diagram

Description automatically generated

Hình 14**:** Mô hình xóa node bất kỳ trong DSLK đơn.

A screen shot of a computer program

Description automatically generated

Hình 15: Cấu trúc xóa phần tử ở sau node bất kì trong DSLK đơn.

### 1.3: Cơ sở lý thuyết về sắp xếp.

### 1.3.1: Khái niệm

- Sắp xếp là sắp xếp dữ liệu theo một định dạng cụ thể. Trong khoa học máy tính, giải thuật sắp xếp xác định cách để sắp xếp dữ liệu theo một thứ tự nào đó.

- Sắp xếp theo thứ tự ở đây là sắp xếp theo thứ tự dạng số hoặc thứ tự dạng chữ cái như trong từ điển.

- Tính quan trọng của việc sắp xếp dữ liệu nằm ở chỗ: việc tìm kiếm dữ liệu có thể được tối ưu nếu dữ liệu được sắp xếp theo một thứ tự nào đó (tăng hoặc giảm). Sắp xếp cũng được sử dụng để biểu diễn dữ liệu trong một định dạng dễ đọc hơn.

### 1.3.2: Sắp xếp nổi bọt (Bubble Sort).

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

- Giải thuật này không thích hợp sử dụng với các tập dữ liệu lớn khi mà độ phức tạp trường hợp xấu nhất và trường hợp trung bình là Ο(n2) với n là số phần tử.

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

**VD:**

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

A white squares with black numbers

Description automatically generated

Giải thuật sắp xếp nổi bọt bắt đầu với hai phần tử đầu tiên, so sánh chúng để kiểm tra xem phần tử nào lớn hơn. Trong trường hợp này, 33 lớn hơn 14, do đó hai phần tử này đã theo thứ tự.

A group of squares with numbers

Description automatically generated

Tiếp đó chúng ta so sánh 33 và 27.



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

A group of squares with numbers

Description automatically generated

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

A screenshot of a computer

Description automatically generated

Sau đó chúng ta so sánh hai giá trị kế tiếp là 35 và 10, 10 nhỏ hơn 35 nên ta lại đổi vị trí 2 giá trị này cho nhau

A group of squares with numbers

Description automatically generated

Vậy là sau một vòng lặp, mảng sẽ trông như sau

A white squares with black numbers

Description automatically generated

Chúng ta lặp lại từ đầu quá trình so sánh như vậy, sau lần lặp thứ hai, mảng sẽ trông giống như.

A group of squares with numbers

Description automatically generated

Lần thứ 3.

A group of squares with numbers

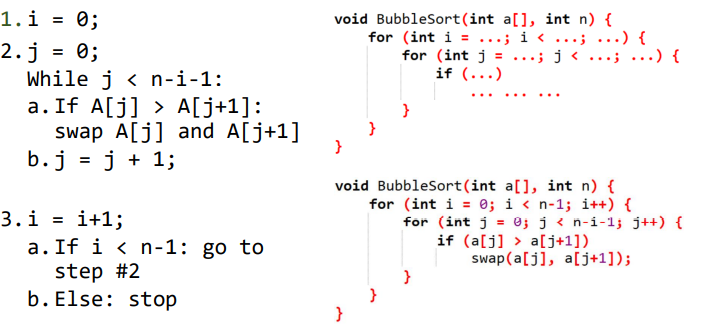
Description automatically generated

Lần thứ 4.

A green squares with black numbers

Description automatically generated

Kết thúc lần thứ 4 chúng ta thấy dãy số đã được sắp xếp đúng thứ tự, thuật toán kết thúc.



Hình 16: Cấu trúc sắp xếp nổi bọt (Bubble Sort).

### 1.4: Cơ sở lý thuyết về tìm kiếm.

### 1.4.1: Khái niệm.

Tìm kiếm là quá trình xác định vị trí của một phần tử cụ thể trong một tập hợp các phần tử, chẳng hạn như mảng, danh sách liên kết, cây, hoặc bảng băm. Mục tiêu của việc tìm kiếm là kiểm tra xem phần tử cần tìm có tồn tại trong tập hợp không, và nếu có thì trả về vị trí hoặc thông tin liên quan.

### 1.4.2: Tìm kiếm tuần tự (Linear Search).

Tìm kiếm tuần tự (Linear Search) là một giải thuật đơn giản, rất dễ cài đặt. Bắt đầu từ đối tượng a1,*a*1​, duyệt qua tất cả các đối tượng, cho tới khi tìm thấy đối tượng có khóa mong muốn, hoặc duyệt hết toàn bộ dãy mà không tìm thấy khóa đó.

**VD:**

Bắt đầu từ phần tử ngoài cùng bên trái của mảng và lần lượt so sánh phần tử chúng ta đang tìm kiếm với từng phần tử của mảng.

Nếu có sự trùng khớp giữa phần tử chúng ta đang tìm kiếm và một phần tử của mảng, ta sẽ trả về chỉ số của phần tử đó.

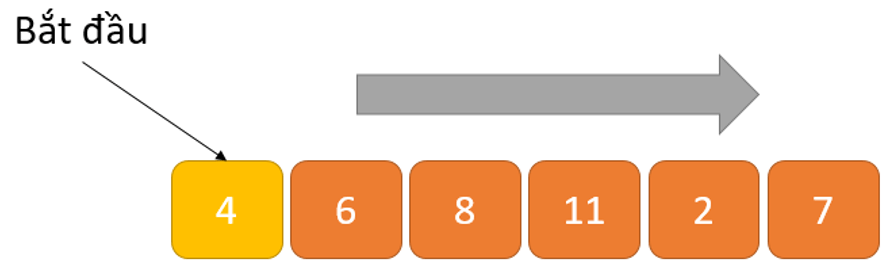
Nếu không có kết quả phù hợp nào giữa phần tử chúng ta đang tìm kiếm và một phần tử của mảng, trả về giá trị là -1 hoặc bất kỳ giá trị nào mà không nằm trong chỉ số của danh sách.

Ví dụ, các bước sau được thực hiện để tìm kiếm phần tử k = 2 trong danh sách dưới đây.

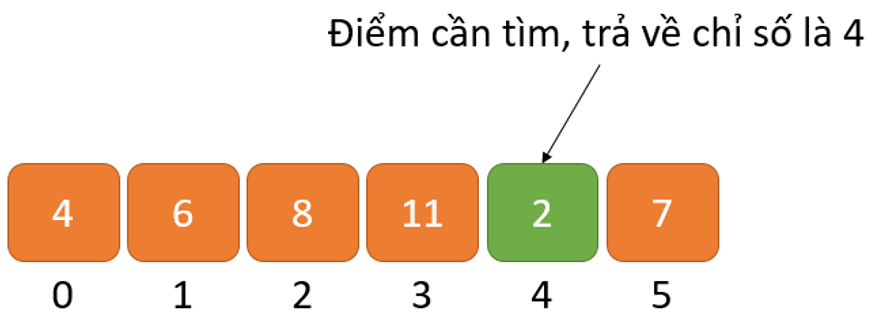
Giả sử ta có mảng như sau:



Bắt đầu từ phần tử đầu tiên, so sánh k với mỗi phần tử x.



Nếu x == k, ta sẽ trả về chỉ số của phần tử đó.



Nếu không, trả về thông báo là không tìm thấy phần tử trong danh sách. Trong mảng trên, giá trị k cần tìm có chỉ số là 4 trong mảng.

# CHƯƠNG 2. XÂY DỰNG CHƯƠNG TRÌNH

### 2.1 Gới thiệu thuật toán.

- Chương trình sử dụng danh sách liên kết đơn (Linklist) để lưu trữ và quản lý các phần tử, sử dụng danh sách liên kết đơn giúp cho việc thêm và xoá các phần tử được linh hoạt mà không cần phải cấp phát trước kích thước của danh sách, tiết kiệm bộ nhớ.

- Sử dụng cấu trúc (struct) để định nghĩa kiểu dữ liệu cho các đối tượng.

- Ứng dụng thuật toán sắp xếp nổi bọt (Bubble Sort) cho chức năng sắp xếp.

- Ứng dụng thuật toán tìm tuyến tuyến tính (Linear Search) cho chức năng tìm kiếm.

### 2.1.1: Các thư viện trong chương trình.

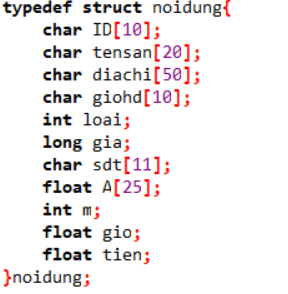
**-** stdio.h: cung cấp các chức năng để nhập xuất dữ liệu, xử lý các tệp và thực hiện các thao tác với đầu vào, đầu ra

**-** string.h: Thư viện chứa các lớp và hàm xử lý chuỗi, cho phép làm việc linh hoạt với chuỗi ký tự.

**-** math.h: cung cấp một tập hợp các hàm toán học cơ bản để thực hiện các phép toán số học, hàm số học và các phép tính với số thực (float, double, long double). Các hàm trong math.h rất hữu ích cho các ứng dụng yêu cầu tính toán số học nâng cao như tính toán góc, logarit, hàm mũ, căn bậc hai, và nhiều phép toán khác.

### 2.2: Xây dựng chương trình.

- Cấu trúc nội dung:



Hình 17: Cấu trúc nội dung.

ID: Số của sân bóng (chuỗi ký tự).

tensan: Tên sân bóng (chuỗi ký tự).

diachi: Địa chỉ sân bóng (chuỗi ký tự).

giohd: Giờ hoạt động của sân bóng (chuỗi ký tự).

loai: Loại sân (số nguyên).

gia: Giá thuê sân theo giờ (kiểu long).

sdt: Số điện thoại liên lạc của sân bóng (chuỗi ký tư).

A[25]: Mảng chứa thông tin về thời gian đặt sân (dữ liệu giờ bắt đầu và giờ kết thúc).

m: Số người đặt sân.

gio: Tổng số giờ sử dụng sân (kiểu float).

tien: Tổng tiền thuê sân (kiểu float).

### 2.2.1: Khai báo cấu trúc node:

typedef struct Node{

noidung \*data;

Node \*next;

}Node;

### 2.2.2: Khai báo danh sách:

typedef struct List{

Node \*head;

Node \*tail;

int size;

List();

### 2.2.3: Các hàm khai báo của cấu trúc List:

Node \*createNode(noidung \*nd): Tạo một node mới với dữ liệu là một đối tượng noidung.

bool isEmpty(): Kiểm tra xem danh sách sân có rỗng hay không.

void addLast(noidung \*nd): Thêm một sân vào cuối danh sách.

void addFirst(noidung \*nd): Thêm một sân vào đầu danh sách.

void insert(noidung \*nd, int pos): Chèn một sân vào danh sách tại vị trí pos.

void deleteFirst(): Xóa sân đầu tiên của danh sách.

void deleteLast(): Xóa sân cuối cùng của danh sách.

void SOSANCONHD(): Kiểm tra số sân còn hoạt động.

void print(): In ra toàn bộ danh sách sân.

void print1(): In ra danh sách giờ và thành tiền.

void print2(): In ra danh sách đặt sân.

void Sapxep(): Sắp xếp theo giá sân.

void Sapxep1(): Sắp xếp theo loại sân.

void Loc(): Lọc sân còn trống.

int checkID(): Kiểm tra xem ID của sân có hợp lệ hay không.

void saveFile(int tam): Lưu dữ liệu danh sách sân vào một file.

int openFile1(): Mở file và đọc dữ liệu từ file vào danh sách sân.

void nhapdsdatsan(int dem): Nhập danh sách các sân bóng hoặc dữ liệu liên quan từ người dùng, với dem có thể là số lượng hoặc một điều kiện nhập liệu**.**

### 2.2.4: Hàm khởi tạo:

List::List(){

head=tail=NULL;

size =0;

}

### 2.2.5: Hàm tạo node mới:

Node\* List::createNode(noidung \*nd){

Node \*p= new Node();

p->data=nd;

p->next=NULL;

return p;

}

### 2.2.6: Hàm thêm sân bóng đầu:

void List::addFirst(noidung \*nd){

Node \*p = createNode(nd);

if(head==NULL){

head=tail=p;

}

else{

p->next=head;

head=p;

}

size++;

}

### 2.2.7: Hàm thêm sân bóng cuối:

void List::addLast(noidung \*nd){

Node \*p=createNode(nd);

if(head==NULL){

head=tail=p;

}

else{

tail->next=p;

tail=p;

}

size++;

}

### 2.2.8: Hàm thêm sân bóng bất kỳ:

void List::insert(noidung \*nd, int pos){

Node \*p=createNode(nd);

if(head==NULL)

head=tail=p;

else{

if(pos>size+1)

printf("Vuot danh sach");

else if(pos==1)

addFirst(nd);

else if(pos==size+1)

addLast(nd);

else{

Node \*a=head;

for(int i=1;i<pos-1;i++){

a=a->next;

}

p->next=a->next;

a->next=p;

size++;

}

}

}

Thêm một đối tượng noidung (thông tin sân bóng) vào một vị trí cụ thể trong danh sách. Cụ thể, phương thức insert(noidung \*nd, int pos) thực hiện việc chèn một sân bóng vào danh sách với các bước như sau:

- Kiểm tra tính hợp lệ của vị trí: Nếu vị trí pos vượt quá kích thước hiện tại của danh sách, phương thức sẽ thông báo "Vượt danh sách". Nếu vị trí hợp lệ, phương thức sẽ tiếp tục.

- Thêm vào đầu hoặc cuối danh sách: Nếu vị trí là đầu danh sách (pos == 1), sân bóng được thêm vào đầu bằng phương thức addFirst. Nếu vị trí là cuối danh sách (pos == size + 1), sân bóng được thêm vào cuối bằng phương thức addLast.

- Chèn vào giữa danh sách: Nếu vị trí là một vị trí ở giữa, phương thức duyệt danh sách đến node đứng trước vị trí cần chèn. Sau đó, nó điều chỉnh các con trỏ để chèn sân bóng vào đúng vị trí, đảm bảo các liên kết trong danh sách vẫn nguyên vẹn.

### 2.2.9: Hàm liệt kê số sân còn hoạt động:

void List::SOSANCONHD(){

printf("SO SAN BONG CON HOAT DONG LA: %d\n",size);

printf("DANH SACH SAN CON HOAT DONG:\n");

}

### 2.2.10: Hàm xóa sân bóng đầu danh sách:

void List::deleteFirst(){

if(head==NULL)

printf("Danh sach dang rong");

else{

head=head->next;

size--;

}

}

### 2.2.11: Hàm xóa sân bóng cuối danh sách:

void List::deleteLast(){

if(head==NULL)

printf("Danh sach dang rong");

else{

Node \*a=head;

for(int i=1;i<size-1;i++){

a=a->next;

}

a->next=NULL;

tail=a;

size--;

}

}

### 2.2.12: Hàm kiểm tra sân bóng có tồn tại hay không:

int List::checkID(){

char id[10];

int stt=0;

int temp=0;

scanf("%s",&id);

for( Node \*i=head;i!=NULL;i=i->next){

char \*result = strstr(i->data->ID, id);

if(result){

temp=temp+1;

stt=stt+1;

break;

}

stt=stt+1;

}

if(temp==0){

return 0;

}

else

return stt;

}

- Khai báo phương thức và các biến cần thiết.

- Trong vòng lặp, chương trình kiểm tra xem ID người dùng nhập có tồn tại trong trường ID của node hiện tại hay không bằng cách sử dụng hàm strstr(). Hàm strstr() tìm kiếm chuỗi con trong một chuỗi lớn và trả về con trỏ đến vị trí đầu tiên của chuỗi con trong chuỗi lớn (hoặc NULL nếu không tìm thấy).

- Nếu strstr() trả về một con trỏ khác NULL, điều đó có nghĩa là ID của người dùng tồn tại trong danh sách sân bóng tại vị trí hiện tại, và chương trình sẽ:

- Cập nhật biến temp thành 1 (đã tìm thấy).

- Lưu vị trí của sân bóng trong stt.

- Thoát khỏi vòng lặp bằng break vì không cần phải tìm thêm.

- Nếu sau khi duyệt hết danh sách mà không tìm thấy ID (temp == 0), hàm trả về 0, báo hiệu rằng ID không tồn tại trong danh sách.

- Nếu tìm thấy ID, hàm trả về vị trí của sân bóng trong danh sách (stt).

### 2.2.13: Hàm sắp xếp theo giá sân.

void List::Sapxep(){

int g;

printf("GIA TANG DAN NHAN PHIM 1\n");

printf("GIA GIAM DAN NHAN PHIM 2\n");

scanf("%d",&g);

Node \*temp=new Node;

if(g==1){

for( Node \*a=head;a!=NULL;a=a->next){

for( Node \*b=a->next;b!=NULL;b=b->next){

if((a->data->gia)>(b->data->gia)){

temp->data = a->data;

a->data = b->data;

b->data = temp->data;

}

}

}

}

else if(g==2){

for( Node \*a=head;a!=NULL;a=a->next){

for( Node \*b=a->next;b!=NULL;b=b->next){

if((a->data->gia)<(b->data->gia)){

temp->data = a->data;

a->data = b->data;

b->data = temp->data;

}

}

}

}

else

printf("khong co lua chon %d vui long chon lai\n",g);

}

- Đầu tiên nhập g (từ bàn phím).

+ Nếu g == 1: sắp xếp theo giá tăng dần.

* Duyệt qua tất cả các node trong danh sách (for (Node \*a = head; a != NULL; a = a->next)).
* Duyệt qua các node phía sau node a (for (Node \*b = a->next; b != NULL; b = b->next)).
* So sánh giá của 2 sân bóng tại node a và node b. Nếu giá tại node a lớn hơn tại node b, hoán đổi dữ liệu của chúng (thực hiện hoán đổi bằng cách sử dụng node temp).

+ Nếu g == 2: sắp xếp theo giá giảm dần.

* Nếu người dùng chọn g == 2, phương thức sẽ thực hiện sắp xếp danh sách theo giá giảm dần, tương tự như khi sắp xếp theo giá tăng dần, nhưng lần này, điều kiện hoán đổi sẽ là:
* Nếu giá của sân tại node a nhỏ hơn giá tại node b, thì hoán đổi chúng.

### 2.2.14: Sắp xếp theo loại sân.

void List::Sapxep1(){

Node \*temp=new Node;

for( Node \*a=head;a!=NULL;a=a->next){

for( Node \*b=a->next;b!=NULL;b=b->next){

if((a->data->loai)>(b->data->loai)){

temp->data = a->data;

a->data = b->data;

b->data = temp->data;

}

}

}

}

Dùng hai vòng lặp for để duyệt qua tất cả các node trong danh sách liên kết:

* Vòng lặp ngoài sẽ lấy từng node một (gọi là node a).
* Vòng lặp trong sẽ so sánh node a với tất cả các node phía sau nó (gọi là node b).

Nếu điều kiện a->data->loai > b->data->loai được thỏa mãn (tức là loại sân trong node a phải được xếp xuống sau loại sân trong node b), ta sẽ thực hiện hoán đổi dữ liệu của hai node a và b để đảm bảo loại sân của node a không vượt qua node b.

### 2.3: Giao diện chương trình.

A black screen with white text

Description automatically generated

Hình 18: Menu chính.

# CHƯƠNG 3. KẾT QUẢ CHƯƠNG TRÌNH

Sau quá trình lên ý tưởng và xây dựng. Chương trình đã được hoàn thành với các chức năng cơ bản của một chương trình quản lí. Dưới đây là một số hình ảnh thu được khi chạy thử chương trình:

A black and white screen with white text

Description automatically generated

Hình 19: Danh sách sân bóng.

A screen shot of a computer

Description automatically generated

Hình 20: Danh sách giờ và thành tiền.

A screen shot of a computer

Description automatically generated

Hình 21: Thêm sân bóng đầu, cuối và sau vị trí 3.

**A black and white screen with white text

Description automatically generated**

Hình 22: Xóa sân bóng đầu và cuối.

**A screen shot of a computer

Description automatically generated**

Hình 23: Cập nhật danh sách đặt sân cho CN1.

A computer screen with white text

Description automatically generated

Hình 24: Sắp xếp theo giá sân tăng dần.

A black and white screen with white text

Description automatically generated

Hình 25: Sắp xếp theo loại sân.

**TÀI LIỆU THAM KHẢO**

* [**https://topdev.vn/blog/danh-sach-lien-ket-don-trong-c/#them-phan-tu-vao-danh-sach**](https://topdev.vn/blog/danh-sach-lien-ket-don-trong-c/#them-phan-tu-vao-danh-sach)**.**
* [**https://blog.28tech.com.vn/dslk-don-danh-sach-lien-ket-don-singly-linked-list**](https://blog.28tech.com.vn/dslk-don-danh-sach-lien-ket-don-singly-linked-list)**.**
* [**https://viblo.asia/p/thuat-toan-sap-xep-noi-bot-bubble-sort-m68Z0exQlkG**](https://viblo.asia/p/thuat-toan-sap-xep-noi-bot-bubble-sort-m68Z0exQlkG)**.**
* [**https://blog.28tech.com.vn/c-thuat-toan-tim-kiem-tuyen-tinh**](https://blog.28tech.com.vn/c-thuat-toan-tim-kiem-tuyen-tinh)**.**
* Cấu trúc dữ liệu và giải thuật – Nguyễn Trung Trực.