**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: QUẢN LÝ ĐẶT VÉ XEM PHIM CỦA KHÁCH HÀNG**

Giảng viên hướng dẫn: Th.S TRẦN THỊ DUNG

Nhóm sinh viên thực hiện: PHẠM VĂN QUÍ

NGUYỄN TẤN HÒA

NGUYỄN HỮU TÀI

Lớp : CQ.64.KTDTVT

Khoá :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: QUẢN LÝ ĐẶT VÉ XEM PHIM CỦA KHÁCH HÀNG**

Giảng viên hướng dẫn: Th.S TRẦN THỊ DUNG

Nhóm sinh viên thực hiện: PHẠM VĂN QUÍ

NGUYỄN TẤN HÒA

NGUYỄN HỮU TÀI

Lớp : CQ.64.KTDTVT

Khoá :64

Tp. Hồ Chí Minh, năm 2024

# LỜI CẢM ƠN

Lời đầu tiên nhóm 1 chúng em xin phép gửi đến Quý Thầy Cô trường Đại học Giao thông Vận tải - Phân hiệu tại Tp Hồ Chí Minh lời chúc sức khỏe và lời cảm ơn sâu sắc nhất. Thầy cô đã tận tình hướng dẫn, truyền đạt kiến thức, kinh nghiệm cho chúng em trong suốt quá trình học tập và rèn luyện ở tại trường.

Kế tiếp, chúng em xin cảm ơn nhà trường đã tạo điều kiện tốt nhất cho chúng em,

để chúng em có một môi trường học tập thật tốt. Đặc biệt cảm ơn đến các giáo viên

của Bộ môn Công nghệ thông tin đã truyền đạt và dạy dỗ chúng em rất nhiều kiến thức trong quãng thời gian học tập vừa qua.

Chúng em xin gửi lời cảm ơn đến cô Trần Thị Dung, cô đã tận tình hướng dẫn, chỉ bảo để nhóm chúng em có thể hoàn thành bài báo cáo của học phần Cấu Trúc Dữ Liệu & Giải Thuật này.

Cảm ơn đến toàn thể các bạn thuộc lớp Kỹ thuật Điện tử- Viễn thông K64 đã luôn giúp đỡ chúng mình trong thời gian học tập cùng nhau.

Mặc dù nhóm chúng em đã cố gắng hết sức trong quá trình thực hiện bài thuyết trình, nhưng vì thời gian có hạn và nhóm chúng em cũng chưa có kinh nghiệm làm việc nhóm, nên chắc chắn sẽ có nhiều thiếu sót. Vì vậy rất mong cô cùng tập thể lớp đưa ra nhận xét, góp ý để nhóm chúng em được hoàn thành bài báo cáo được tốt nhất.

Cuối cùng, chúng em chúc sức khỏe đến tất cả các thành viên trong gia đình, toàn thể giảng viên của trường, các bạn trong lớp Công nghệ Thông tin K64, cùng bạn bè khắp nơi. Nhóm 1 chúng em xin chân thành cảm ơn!

**Tp.HCM, ngày tháng năm 2024.**

**MỤC LỤC**

[LỜI CẢM ƠN 3](#_Toc182140511)

[DANH MỤC CHỮ VIẾT TẮT 6](#_Toc182140512)

[HÌNH VẼ 7](#_Toc182140513)

[CHƯƠNG 1: CƠ SỞ LÝ THUYẾT. 9](#_Toc182140514)

[1.1 Giới Thiệu về Cấu Trúc Dữ Liệu và Giải Thuật. 9](#_Toc182140515)

[1.1.1 Tầm Quan Trọng của Cấu Trúc Dữ Liệu Trong Quản Lý Danh Sách Khách Hàng. 9](#_Toc182140516)

[1.1.2 Khái Niệm Lập Trình C++ trong Quản Lý Dữ Liệu Động. 9](#_Toc182140517)

[1.1.3 Các Thao Tác Cơ Bản của Danh Sách Liên Kết, Ngăn Xếp và Hàng Đợi. 9](#_Toc182140518)

[1.1.3.1 Danh sách liên kết (Linked List): 9](#_Toc182140519)

[1.1.3.2 Ngăn xếp (Stack): 10](#_Toc182140520)

[1.1.3.3 Hàng đợi (Queue): 10](#_Toc182140521)

[1.2 Thiết Kế Cấu Trúc Dữ Liệu Cho Hệ Thống Đặt Chỗ Thực Tế 10](#_Toc182140522)

[1.2.1 Lựa Chọn Cấu Trúc Dữ Liệu Dựa Trên Đặc Điểm Của Hệ Thống Đặt Chỗ 10](#_Toc182140523)

[1.2.2 Ưu Nhược Điểm Của Các Cấu Trúc Dữ Liệu Thường Dùng 10](#_Toc182140524)

[1.2.2.1 Danh Sách Liên Kết (Linked List): 10](#_Toc182140525)

[1.2.2.2 Ngăn Xếp (Stack): 11](#_Toc182140526)

[1.2.2.3 Hàng Đợi (Queue): 11](#_Toc182140527)

[1.2.3 Thiết Kế Cấu Trúc Dữ Liệu Để Quản Lý Trạng Thái Chỗ Ngồi 11](#_Toc182140528)

[1.2.3.1 Sử dụng mảng 2 chiều (2D Array) 11](#_Toc182140529)

[1.2.3.2 Sử dụng Danh sách liên kết kết hợp với mảng. 12](#_Toc182140530)

[1.2.3.3 Xử lý trạng thái đặt chỗ của khách hàng. 13](#_Toc182140531)

[CHƯƠNG 2: XÂY DỰNG CHƯƠNG TRÌNH QUẢN LÍ GHẾ NGỒI CỦA MỘT PHÒNG CHIẾU. 14](#_Toc182140532)

[2.1 PHÂN TÍCH CHƯƠNG TRÌNH. 14](#_Toc182140533)

[2.1.1 Cấu trúc dữ liệu. 14](#_Toc182140534)

[2.1.2 Các chức năng của quản lý ghế ngồi của quản lý phòng chiếu. 14](#_Toc182140535)

[2.1.3 Menu điều khiển 15](#_Toc182140536)

[2.1.4 Chức năng nổi bật. 15](#_Toc182140537)

[2.2 THIẾT KẾ CHƯƠNG TRÌNH. 15](#_Toc182140538)

[2.2.1 Khai báo kiểu cấu trúc phòng chiếu. 15](#_Toc182140539)

[2.2.1.1 Struct khach. 15](#_Toc182140540)

[2.2.1.2 Hàm stack<khach\*> luuStack. 16](#_Toc182140541)

[2.2.1.3 Hàm struct list. 16](#_Toc182140542)

[2.2.1.4 Khởi tạo node hàm list::list( ) (Constructor). 17](#_Toc182140543)

[2.2.1.5 Hàm khach\* list::createNode() 18](#_Toc182140544)

[2.2.1.6 Tạo một nút khách hàng vào đầu danh sách. 20](#_Toc182140545)

[2.2.1.7 Tạo một nút vào cuối danh sách. 22](#_Toc182140546)

[2.2.1.8 Xóa một khách hàng đầu danh sách. 24](#_Toc182140547)

[2.2.1.9 Hàm tự cập nhập số thứ tự trong danh sách liên kết. 26](#_Toc182140548)

[2.2.1.10 Xóa khách hàng theo yêu cầu. 27](#_Toc182140549)

[**2.2.1.11 Hàm phục hồi vé.** 29](#_Toc182140550)

[2.2.1.12 Chèn một khách hàng tại một vị trí (vị trí pop). 31](#_Toc182140551)

[2.2.1.13 Hàm xóa tất cả khách hàng. 33](#_Toc182140552)

[2.2.1.14 Đặt ghế. 34](#_Toc182140553)

[2.2.1.15 Hiển thị bảng ghế. 35](#_Toc182140554)

[2.2.1.16 Hiển thị số lượng ghế. 36](#_Toc182140555)

[2.2.1.17 Xuất doanh thu phòng chiếu và xuất hóa đơn thông tin vé. 37](#_Toc182140556)

[2.2.1.18 Hiển thị bảng danh sách khách. 38](#_Toc182140557)

[2.2.2 Hiển thị menu và xử lý lựa chọn của người dùng trong vòng lặp do-while. 39](#_Toc182140558)

[2.3. Giao diện chương trình 41](#_Toc182140559)

[PHỤ LỤC 44](#_Toc182140560)

[Phụ lục 1: Hướng dẫn cài đặt 44](#_Toc182140561)

[1. Cài Đặt Trình Biên Dịch C++ 44](#_Toc182140562)

[2. Tải Mã Nguồn 44](#_Toc182140563)

[3. Biên Dịch Chương Trình 44](#_Toc182140564)

[4. Chạy Chương Trình 44](#_Toc182140565)

[Phụ lục 2: hướng dẫn sử dụng 44](#_Toc182140566)

[1. Các Tùy Chọn Trong Menu Chính 44](#_Toc182140567)

[2. Đặt Ghế và Xem Thông Tin 44](#_Toc182140568)

[3. Các Tùy Chọn Khác 44](#_Toc182140569)

[TÀI LIỆU THAM KHẢO 46](#_Toc182140570)

# DANH MỤC CHỮ VIẾT TẮT

|  |  |  |  |
| --- | --- | --- | --- |
| **STT** | **Mô tả** | **Ý nghĩa** | **Ghi chú** |
| 1 | STT | Số thứ tự |  |
| 2 | CTDL | Cấu trúc dữ liệu |  |
| 3 | DSLK | Danh sách liên kết |  |
| 4 | QL | Quản lý |  |
| 5 | LIFO | Last in First out |  |
| 6 | FIFO | First in First out |  |
| 7 | VND | Đơn vị tiền Việt Nam |  |

# HÌNH VẼ

[Hình 2.1.1: Khai báo struct khach trong hàm. 16](#_Toc182140458)

[Hình 2.1.2: Khai khai biến con trỏ trong danh sách và khai báo bảng ghế. 17](#_Toc182140459)

[Hình 2.1.3: Hàm khai báo bảng ghế . 18](#_Toc182140460)

[Hình 2.1.4: khi chạy bảng ghế trống. 18](#_Toc182140461)

[Hình 2.1.5: Code nhập tên khách hàng. 18](#_Toc182140462)

[Hình 2.1.6: Code nhập ghế hàng và ghi nhận ghế. 19](#_Toc182140463)

[Hình 2.1.7: Code cập nhập giá vé và hiện doanh thu. 19](#_Toc182140464)

[Hình 2.1.8: Code sử dụng để ghi thời gian vào rạp. 20](#_Toc182140465)

[Hình 2.1.9: Hàm tạo nút bằng createNode và kiểm tra nếu hàm dừng thì tạo thấp bại. 20](#_Toc182140466)

[Hình 2.1.10: Code liên kết danh sách và cập nhập nếu danh sách rỗng. 21](#_Toc182140467)

[Hình 2.1.11: Chương trình chạy khi chưa thêm addFirst 22](#_Toc182140468)

[Hình 2.1.12: Vd khi ta một danh sách liên kết sau khi ta addfirst và thêm khách C. 22](#_Toc182140469)

[Hình 2.1.13: Hàm tạo một nút và dừng lại khi tạo thất bại. 22](#_Toc182140470)

[Hình 2.1.15: Vd khi ta một danh sách liên kết sau khi ta addlast và thêm khách C. 24](#_Toc182140471)

[Hình 2.1.16: Hàm kiểu tra danh sách có rỗng không. 24](#_Toc182140472)

[Hình 2.1.17: Code lưu con trỏ xong giảm doanh thu và kiểm tra lại danh sách 24](#_Toc182140473)

[Hình 2.1.18: Code trên lưu phần tử đã xóa, cập nhập stt, kích thước xong thông báo. 25](#_Toc182140474)

[Hình 2.1.19: Vd khi cho danh sách liên kết trên và xóa khách A bằng hàm deleteFirst và giảm doanh thu 45000 khách A. 26](#_Toc182140475)

[Hình 2.1.20: Khai triển code tự cập nhập STT. 26](#_Toc182140476)

[Hình 2.1.21: Triển khai code kiểm tra danh sách rỗng hay không và xóa vị trí muốn xóa. 27](#_Toc182140477)

[Hình 2.1.22: Code duyệt danh sách khách hàng muốn xóa. 27](#_Toc182140478)

[Hình 2.1.23: Xử lý khách hàng muốn xóa và xóa đầu giữa và cuối tùy theo vị trị khách hàng. 28](#_Toc182140479)

[Hình 2.1.24: Code lưu thông tin khách hàng đã xóa cập nhập lại danh sách và thông báo nếu không có khách hàng. 28](#_Toc182140480)

[Hình 2.1.25: Code kiểm tra trong stack nếu có thông tin khách hàng thì khôi phục tại ghế cho khách và cập nhập lại doanh thu. 29](#_Toc182140481)

[Hình 2.1.26: Hàm trên them khách và cập nhập lại stt và thông báo nếu cập nhập thành công. 30](#_Toc182140482)

[Hình 2.1.27: Hàm kiểm tra vị trí chèn có hợp lệ hay không tạo node rồi chèn vào đầu danh sách. 31](#_Toc182140483)

[Hình 2.1.28: Code chèn vị trí cuối và giữa danh sách khi chèn xong tự động tăng kích thước và tăng stt. 32](#_Toc182140484)

[Hình 2.1.29: Triển khai hàm xóa tất cả. 33](#_Toc182140485)

[Hình 2.1.30: Hàm khai báo hàng ghế cốt ghế giá ghế cho từng hàng. 34](#_Toc182140486)

[Hình 2.1.31: Code kiểm tra vị trí đó hợp lệ hay không và đánh dấu X nếu đã đặt xong trả lại true. 34](#_Toc182140487)

[Hình 2.1.32: Code hiển thị bảng ghế với “X” đã đặt “–“ còn trống. 35](#_Toc182140488)

[Hình 2.1.33: Hàm khai báo sống lượng ghế trống. 36](#_Toc182140489)

[Hình 2.1.34: Hàm báo cáo doanh thu phòng vé và hóa đơn thông tin khách hàng. 37](#_Toc182140490)

[Hình 2.1.35: Code thể hiện bảng danh sách gồm những gì và nhập thông tin khách hàng. 38](#_Toc182140491)

[Hình 2.2.1: Menu lựa chọn để sử dụng chức năng. 39](#_Toc182140492)

[Hình 2.2.2: Vòng lặp switch case để xử lí các lựa chọn của người dùng. 40](#_Toc182140493)

[Hình 2.3.1 Giao diện khi bắt đầu chạy. 41](#_Toc182140494)

[Hình 2.3.2 Hiển thị danh sách khách hàng. 41](#_Toc182140495)

[Hình 2.3.3 Lựa chọn “2”, thêm khách vào đầu danh sách. 42](#_Toc182140496)

[Hình 2.3.4 Lựa chọn “4” xóa khách đầu tiên khỏi danh sách 42](#_Toc182140497)

[Hình 2.3.5 Lựa chọn “7” phục hồi khách vừa **xóa.** 42](#_Toc182140498)

[Hình 2.3.6 Lựa chọn “9” hiển thị bảng ghế. 42](#_Toc182140500)

[Hình 2.3.7 Lựa chọn “10” Hiển thị số ghế còn trống. 42](#_Toc182140501)

[Hình 2.3.8 Lựa chọn “11” Hiển thị doanh thu hiện tại. 42](#_Toc182140502)

[Hình 2.3.9 Lựa chọn “12” Xuất hóa đơn cho khách hàng. 43](#_Toc182140503)

# CHƯƠNG 1: CƠ SỞ LÝ THUYẾT.

## 1.1 Giới Thiệu về Cấu Trúc Dữ Liệu và Giải Thuật.

### 1.1.1 Tầm Quan Trọng của Cấu Trúc Dữ Liệu Trong Quản Lý Danh Sách Khách Hàng.

Trong các hệ thống quản lý khách hàng như hệ thống đặt chỗ ngồi, việc lưu trữ, truy cập và cập nhật dữ liệu nhanh chóng là vô cùng quan trọng. Sử dụng cấu trúc dữ liệu hợp lý giúp chương trình có thể thực hiện các thao tác này một cách hiệu quả và nhanh chóng. Đặc biệt, các CTDL như DSLK và ngăn xếp rất phù hợp cho việc QL danh sách khách hàng, vì chúng hỗ trợ linh hoạt trong việc thêm, xóa và truy xuất dữ liệu.

Danh sách liên kết (Linked List): DSLK giúp lưu trữ các phần tử (trong trường hợp này là khách hàng) theo một thứ tự nhất định. Nó cho phép chèn và xóa phần tử ở bất kỳ vị trí nào mà không cần phải di chuyển các phần tử khác trong danh sách, giúp tiết kiệm thời gian khi QL danh sách khách hàng lớn.

Ngăn xếp (Stack): Ngăn xếp có thể được sử dụng để lưu trữ các khách hàng vừa bị xóa để có thể khôi phục lại khi cần. Đây là cấu trúc dữ liệu theo nguyên tắc “LIFO” (Last In, First Out - phần tử thêm vào cuối sẽ được lấy ra đầu tiên), rất phù hợp cho các thao tác cần phục hồi dữ liệu gần nhất.

Hàng đợi(Queue): là một cấu trúc dữ liệu quan trọng trong lập trình quản lý danh sách khách hàng và các hệ thống đợi khác. Queue tuân theo nguyên tắc FIFO (First In, First Out), tức là phần tử nào được thêm vào trước sẽ được xử lý trước. Cấu trúc này phù hợp cho các ứng dụng cần xử lý dữ liệu theo thứ tự thời gian hoặc ưu tiên, như quản lý hàng đợi của khách hàng hoặc hệ thống xếp hàng trong các dịch vụ công.

### 1.1.2 Khái Niệm Lập Trình C++ trong Quản Lý Dữ Liệu Động.

Trong C++, khi làm việc với dữ liệu động, ta thường sử dụng con trỏ và quản lý bộ nhớ để tối ưu hóa việc sử dụng tài nguyên.

Con trỏ (Pointers): Là biến lưu trữ địa chỉ của một biến khác. Trong DSLK, con trỏ giúp kết nối các phần tử với nhau. Mỗi phần tử sẽ có một con trỏ trỏ tới phần tử tiếp theo, giúp tạo thành chuỗi kết nối. Điều này đặc biệt quan trọng trong danh sách liên kết vì các phần tử không được lưu trữ ở các vị trí liên tiếp trong bộ nhớ.

Quản lý bộ nhớ động (Dynamic Memory Management): Trong C++, khi làm việc với DSLK, bộ nhớ thường được cấp phát động qua các lệnh new và delete. Điều này giúp chương trình quản lý bộ nhớ hiệu quả và tránh lãng phí bộ nhớ khi số lượng phần tử thay đổi liên tục.

### 1.1.3 Các Thao Tác Cơ Bản của Danh Sách Liên Kết, Ngăn Xếp và Hàng Đợi.

#### 1.1.3.1 Danh sách liên kết (Linked List):

Thêm (Insert): Thêm một phần tử vào danh sách liên kết. Có thể thêm vào đầu, cuối hoặc bất kỳ vị trí nào trong danh sách, tùy thuộc vào nhu cầu.

Xóa (Delete): Xóa một phần tử khỏi danh sách. Quá trình này thường cần xác định vị trí của phần tử muốn xóa và thay đổi con trỏ của phần tử trước đó để kết nối tới phần tử tiếp theo, tránh để lại "lỗ hổng" trong danh sách.

#### 1.1.3.2 Ngăn xếp (Stack):

Đẩy (Push): Thêm một phần tử vào đầu ngăn xếp. Đây là thao tác thêm khách hàng vừa xóa vào ngăn xếp, giúp lưu trữ dữ liệu để có thể khôi phục.

Lấy ra (Pop): Xóa phần tử ở đầu ngăn xếp và trả về phần tử đó. Trong hệ thống đặt chỗ, thao tác này giúp khôi phục lại khách hàng vừa bị xóa.

#### 1.1.3.3 Hàng đợi (Queue):

**Thêm (Enqueue)**: Thêm một phần tử vào cuối hàng đợi. Đối với hệ thống quản lý khách hàng, thao tác này giúp thêm khách vào hàng chờ khi khách hàng yêu cầu một dịch vụ mới.

**Lấy ra (Dequeue)** : Xóa phần tử ở đầu hàng đợi và trả về phần tử đó. Thao tác này giúp đưa khách hàng từ đầu hàng đợi vào danh sách phục vụ, giữ cho việc phục vụ đúng thứ tự.

**Kiểm tra trạng thái hàng đợi**: Có thể bao gồm kiểm tra hàng đợi có trống không, lấy phần tử đầu tiên mà không xóa (peek), hoặc đếm số lượng phần tử.

**1.2 Thiết Kế Cấu Trúc Dữ Liệu Cho Hệ Thống Đặt Chỗ Thực Tế**

**1.2.1 Lựa Chọn Cấu Trúc Dữ Liệu Dựa Trên Đặc Điểm Của Hệ Thống Đặt Chỗ**

Đối với một hệ thống đặt chỗ như rạp chiếu phim, nhà hàng, hoặc khách sạn, yêu cầu về cấu trúc dữ liệu cần có tính linh hoạt để xử lý các thao tác như đặt chỗ, hủy chỗ, kiểm tra trạng thái ghế, và báo cáo doanh thu. Một số yếu tố cần cân nhắc khi lựa chọn cấu trúc dữ liệu bao gồm:

**Tần suất và cách thức cập nhật dữ liệu**: Nếu hệ thống liên tục thêm và xóa khách hàng, các cấu trúc dữ liệu động như danh sách liên kết hoặc hàng đợi sẽ phù hợp hơn so với mảng tĩnh.

**Khả năng truy cập dữ liệu**: Trong một số hệ thống đặt chỗ, cần tra cứu thông tin khách hàng nhanh chóng. Đối với điều này, hàng đợi và danh sách liên kết có thể là lựa chọn hợp lý tùy thuộc vào yêu cầu truy xuất dữ liệu.

**1.2.2 Ưu Nhược Điểm Của Các Cấu Trúc Dữ Liệu Thường Dùng**

**1.2.2.1 Danh Sách Liên Kết (Linked List):**

+ Ưu điểm: DSLK rất linh hoạt, dễ dàng thêm hoặc xóa khách mà không cần di chuyển dữ liệu. Điều này hữu ích trong các hệ thống có số lượng khách biến động.

+ Nhược điểm: Tốn bộ nhớ hơn so với mảng vì phải lưu thêm con trỏ. Ngoài ra, việc tìm kiếm khách hàng có thể chậm hơn so với một số cấu trúc khác do cần duyệt tuần tự.

**1.2.2.2 Ngăn Xếp (Stack):**

+ Ưu điểm: Ngăn xếp có cấu trúc LIFO, phù hợp cho các tác vụ hủy thao tác cuối cùng, như trong hệ thống hủy đặt chỗ cho khách hàng mới nhất.

+ Nhược điểm: Ngăn xếp chỉ xử lý dữ liệu theo thứ tự LIFO, không linh hoạt trong truy cập ngẫu nhiên, nên thường chỉ sử dụng cho một số thao tác cụ thể.

**1.2.2.3 Hàng Đợi (Queue):**

+ Ưu điểm: Hàng đợi có cấu trúc FIFO (First-In-First-Out), rất phù hợp trong quản lý thứ tự chờ của khách hàng.

+ Nhược điểm: Không cho phép truy cập ngẫu nhiên, chỉ thao tác với dữ liệu đầu và cuối của hàng đợi, do đó cần phối hợp với các cấu trúc khác khi cần truy cập ngẫu nhiên.

**1.2.3 Thiết Kế Cấu Trúc Dữ Liệu Để Quản Lý Trạng Thái Chỗ Ngồi**

Trong hệ thống đặt chỗ cho rạp chiếu phim hay nhà hàng, việc quản lý trạng thái chỗ ngồi (ví dụ, trống hoặc đã được đặt) là một yếu tố quan trọng, đảm bảo rằng khách hàng có thể xem và lựa chọn chỗ ngồi còn trống một cách chính xác và nhanh chóng. Các cấu trúc dữ liệu hiệu quả không chỉ giúp hệ thống dễ dàng truy cập và cập nhật trạng thái chỗ ngồi mà còn cải thiện hiệu suất tổng thể. Dưới đây là hai cách phổ biến để thiết kế cấu trúc dữ liệu quản lý chỗ ngồi trong các hệ thống đặt chỗ.

**1.2.3.1 Sử dụng mảng 2 chiều (2D Array)**

Mảng hai chiều là một cấu trúc dữ liệu trong đó mỗi phần tử của mảng là một chỗ ngồi, đại diện cho vị trí trong không gian hai chiều (ví dụ, hàng và cột của một rạp chiếu phim hoặc nhà hàng). Mỗi ô trong mảng có thể chứa một ký hiệu để biểu thị trạng thái của chỗ ngồi: 'X' có thể đại diện cho ghế đã đặt, trong khi '-' biểu thị ghế còn trống.

**+ Ưu điểm**:

**Truy cập nhanh và trực tiếp**: Sử dụng chỉ số hàng và cột giúp truy cập trạng thái của bất kỳ ghế nào trong hệ thống một cách nhanh chóng, chỉ với một phép tra cứu đơn giản. Điều này rất hữu ích khi hiển thị bảng trạng thái toàn bộ chỗ ngồi cho khách hàng, giúp tiết kiệm thời gian truy xuất.

**Dễ dàng hiển thị bảng ghế**: Cấu trúc mảng hai chiều dễ dàng biểu diễn trực quan, phù hợp cho việc in ra hoặc hiển thị bố trí chỗ ngồi như một bảng, cho phép khách hàng có cái nhìn tổng quan về tình trạng chỗ ngồi.

**Tối ưu cho hệ thống cố định**: Mảng hai chiều rất phù hợp cho các hệ thống mà số lượng chỗ ngồi không thay đổi (như rạp chiếu phim có số hàng ghế cố định). Điều này giúp đơn giản hóa việc quản lý, khi cấu trúc không cần động.

**+ Nhược điểm:**

**Tĩnh và kém linh hoạt**: Mảng hai chiều có kích thước cố định, khó thay đổi trong thời gian thực. Nếu rạp chiếu phim thay đổi số lượng hàng ghế, hoặc nhà hàng cần mở rộng thêm bàn, việc thay đổi cấu trúc mảng sẽ gây phức tạp, đòi hỏi phải tạo lại mảng và sao chép dữ liệu.

**Bộ nhớ thừa**: Trong trường hợp không phải tất cả các ghế đều được sử dụng hoặc nếu bố trí ghế không đồng đều (ví dụ: một số hàng có ít ghế hơn), mảng hai chiều có thể chiếm nhiều bộ nhớ hơn mức cần thiết vì phải lưu trữ trạng thái cho tất cả vị trí, bao gồm cả các ghế không tồn tại trong thực tế.

**1.2.3.2 Sử dụng Danh sách liên kết kết hợp với mảng.**

Cấu trúc này bao gồm một danh sách liên kết, trong đó mỗi nút chứa một mảng con, đại diện cho một hàng ghế. Mỗi mảng con lưu trữ trạng thái của các ghế trong một hàng cụ thể. Điều này cho phép linh hoạt trong việc thêm hoặc xóa các hàng mà không cần phải tạo lại toàn bộ cấu trúc như với mảng hai chiều.

**+ Ưu điểm**:

**Linh hoạt hơn**: Do các hàng được quản lý bằng danh sách liên kết, cấu trúc này dễ dàng mở rộng hoặc giảm số lượng hàng ghế mà không cần thay đổi kích thước toàn bộ. Hệ thống có thể thêm hoặc xóa một hàng bằng cách thêm hoặc gỡ bỏ nút trong danh sách liên kết mà không ảnh hưởng đến các hàng khác.

**Tiết kiệm bộ nhớ**: Trong các trường hợp mà mỗi hàng có số lượng ghế khác nhau hoặc không phải tất cả các hàng đều được sử dụng, danh sách liên kết giúp tiết kiệm bộ nhớ vì nó chỉ lưu trữ trạng thái cho các hàng và ghế có thực.

**Dễ dàng cập nhật trạng thái chỗ ngồi**: Nếu có yêu cầu về thay đổi cấu hình hàng ghế, chẳng hạn chuyển đổi số ghế trong một hàng, việc thao tác với một nút cụ thể trong danh sách giúp thực hiện dễ dàng mà không phải thao tác lại toàn bộ mảng.

**+ Nhược điểm**:

**Phức tạp trong quản lý**: Danh sách liên kết yêu cầu quản lý con trỏ, dễ gây lỗi trong các thao tác thêm, xóa, hay duyệt qua danh sách. Điều này làm tăng độ phức tạp lập trình và yêu cầu kiểm soát chặt chẽ hơn khi cập nhật trạng thái chỗ ngồi.

**Truy cập chậm hơn**: So với mảng hai chiều, truy cập vào danh sách liên kết chậm hơn vì phải duyệt từ đầu danh sách đến vị trí mong muốn, đặc biệt nếu cần kiểm tra các hàng ở gần cuối.

**Khó hiển thị toàn bộ bảng ghế**: Vì danh sách liên kết không có khả năng truy cập ngẫu nhiên, việc hiển thị trạng thái của tất cả ghế sẽ chậm hơn so với mảng hai chiều, đặc biệt với các hệ thống có quy mô lớn.

**1.2.3.3 Xử lý trạng thái đặt chỗ của khách hàng.**

Trong hệ thống đặt chỗ, việc quản lý và xử lý trạng thái đặt chỗ của khách hàng là rất quan trọng để đảm bảo rằng mỗi khách hàng đều nhận được dịch vụ tốt nhất và hệ thống có thể hoạt động trơn tru. Các trạng thái đặt chỗ giúp hệ thống hiểu rõ tình trạng của mỗi chỗ ngồi, từ đó đưa ra các quyết định kịp thời khi cần thiết, chẳng hạn cho phép đặt chỗ, thông báo với khách hết chỗ, hoặc giải phóng chỗ cho khách hàng mới (hủy chỗ ngồi).

**+ Trạng Thái Đã Đặt:**

Khi một chỗ ngồi được đặt thành công, hệ thống sẽ chuyển trạng thái của chỗ ngồi đó thành "Đã Đặt". Trạng thái này cho biết rằng chỗ ngồi không còn trống và đã được gán cho một khách hàng cụ thể. Trạng thái "Đã Đặt" là trạng thái quan trọng nhất vì nó đảm bảo rằng không có khách hàng nào khác có thể đặt chỗ đó. Khi một khách hàng đặt thành công, hệ thống sẽ lưu thông tin đặt chỗ của khách, bao gồm số chỗ ngồi, thông tin khách hàng, thời gian đặt, và bất kỳ chi tiết nào khác để phục vụ mục đích tra cứu hoặc kiểm tra.Hệ thống sẽ cập nhật thông tin chỗ ngồi từ "Trống" sang "Đã Đặt" khi đặt chỗ thành công.Cấu trúc dữ liệu như mảng hai chiều hoặc danh sách liên kết sẽ được cập nhật để ghi nhận trạng thái mới.Trạng thái này chỉ có thể thay đổi nếu khách hàng hủy hoặc nếu đặt chỗ bị hết hiệu lực (ví dụ, không đến đúng giờ trong các hệ thống có thời gian chờ nhất định).

**+ Trạng Thái Hủy:**

Khi một khách hàng quyết định hủy đặt chỗ, trạng thái chỗ ngồi sẽ chuyển từ "Đã Đặt" sang "Hủy". Trạng thái này cho biết rằng chỗ ngồi hiện đang trống và sẵn sàng để phục vụ khách hàng khác. Tuy nhiên, trong một số hệ thống, trạng thái "Hủy" không chuyển trực tiếp thành "Trống" mà được lưu trong một ngăn xếp (stack) để có thể hoàn tác nếu cần thiết. Trạng thái "Hủy" giúp hệ thống dễ dàng nhận diện các chỗ ngồi bị hủy để có thể tái sử dụng hoặc cung cấp cho các khách hàng trong hàng đợi. Việc sử dụng ngăn xếp cho trạng thái "Hủy" mang lại lợi ích đặc biệt trong trường hợp hệ thống cần hỗ trợ tính năng hoàn tác, cho phép khôi phục lại trạng thái đặt chỗ vừa bị hủy. Khi khách hàng hủy chỗ, hệ thống chuyển trạng thái chỗ ngồi thành "Hủy" và lưu vào một ngăn xếp. Việc sử dụng ngăn xếp đảm bảo rằng hệ thống có thể dễ dàng lấy lại trạng thái vừa bị hủy một cách nhanh chóng, đặc biệt trong các tình huống nhầm lẫn hoặc khi khách hàng muốn khôi phục đặt chỗ ngay lập tức. Nếu hệ thống không cần hoàn tác, trạng thái có thể được chuyển trực tiếp từ "Hủy" thành "Trống" và chỗ ngồi sẽ sẵn sàng để nhận khách mới. Ngăn xếp cũng có thể dùng để lưu lại các hành động trước đó, giúp hệ thống dễ dàng quản lý lịch sử trạng thái chỗ ngồi.

Để xử lý trạng thái "Hủy" một cách hiệu quả, hệ thống cần một ngăn xếp (stack) lưu các trạng thái chỗ ngồi bị hủy gần đây. Điều này cho phép thực hiện hoàn tác theo kiểu LIFO) – trạng thái hủy gần đây nhất có thể được khôi phục đầu tiên.

# CHƯƠNG 2: XÂY DỰNG CHƯƠNG TRÌNH QUẢN LÍ GHẾ NGỒI CỦA MỘT PHÒNG CHIẾU.

## 2.1 PHÂN TÍCH CHƯƠNG TRÌNH.

### 2.1.1 Cấu trúc dữ liệu.

* khach: Là cấu trúc dữ liệu lưu trữ thông tin về một khách hàng bao gồm:
  + Tên khách hàng.
  + Số thứ tự (STT).
  + Vị trí ghế trong rạp (VD: HA-C1).
  + Số tiền vé.
  + Giờ vào rạp.
  + Con trỏ next trỏ đến khách hàng tiếp theo trong danh sách.
* list: Là lớp quản lý danh sách liên kết của các khách hàng và trạng thái ghế ngồi trong rạp.
* Thuộc tính:
  + head, tail: Quản lý đầu và cuối danh sách liên kết.
  + size: Số lượng khách trong danh sách.
  + bangGhe: Ma trận ghế ngồi (kích thước 5x5) với trạng thái '-' là ghế trống, 'X' là ghế đã đặt.
  + doanhThu: Tổng doanh thu bán vé.

### 2.1.2 Các chức năng của quản lý ghế ngồi của quản lý phòng chiếu.

* Đặt vé và quản lý khách hàng**:**
* CreateNode: Nhập thông tin của khách hàng và xác nhận vị trí ghế muốn đặt. Vé sẽ được xác định giá dựa trên hàng ghế:
  + Hàng A, B: 45,000 VND.
  + Hàng C, D, E: 55,000 VND.
* Addfirst và addlast: Thêm khách hàng vào đầu hoặc cuối danh sách liên kết.
* Insert: Chèn khách hàng vào một vị trí cụ thể trong danh sách.
* DeleteFirst: Xóa khách hàng đầu tiên trong danh sách và lưu vào ngăn xếp (stack) để phục hồi khi cần.
* XoaKhachTheoYeuCau: Xóa khách hàng theo vị trí ghế yêu cầu.
* PhucHoiVe: Phục hồi thông tin khách hàng đã xóa từ ngăn xếp và cập nhật trạng thái ghế.
* Hiển thị thông tin:
* Hiển thị danh sách khách hàng hiện có cùng thông tin hienThiDanhSachKhach chi tiết.
* HienThiBangGhe: Hiển thị trạng thái các ghế ngồi trong rạp.
* HienThiSoLuongGhe: Hiển thị số lượng ghế trống còn lại.
* HienThiDoanhThu: Hiển thị tổng doanh thu từ việc bán vé.
* XuatHoaDon: Xuất hóa đơn chi tiết cho từng khách hàng.
* Quản lý danh sách:
* DeleteAll: Xóa toàn bộ danh sách khách hàng, đặt lại trạng thái ghế và doanh thu về 0.

### 2.1.3 Menu điều khiển

Chương trình cung cấp một menu tương tác cho người dùng lựa chọn các chức năng như:

* Thêm khách hàng.
* Xóa khách hàng.
* Phục hồi vé đã xóa.
* Hiển thị danh sách khách.
* Quản lý ghế ngồi và tính doanh thu.
* Thoát chương trình.

### 2.1.4 Chức năng nổi bật.

* + Quản lý trạng thái ghế ngồi: Theo dõi trạng thái ghế ngồi theo thời gian thực.
  + Phục hồi vé: Khả năng khôi phục khách hàng đã xóa thông qua cấu trúc ngăn xếp.
  + Tính toán doanh thu: Tự động cập nhật doanh thu dựa trên giá vé từng hàng ghế.

## 2.2 THIẾT KẾ CHƯƠNG TRÌNH.

### 2.2.1 Khai báo kiểu cấu trúc phòng chiếu.

#### 2.2.1.1 Struct khach.

Cấu trúc khach dùng để lưu thông tin của một khách hàng. Bao gồm các thành phần:

* char Ten[30]:
  + Mảng ký tự lưu tên khách hàng (tối đa 29 ký tự + 1 ký tự kết thúc chuỗi).
* int STT:
  + Số thứ tự của khách hàng trong danh sách.
* char ViTri[10]:
  + Mảng ký tự lưu vị trí ghế của khách hàng.
  + Vị trí ghế có dạng "Hàng-Cột" (VD: HA-C1).
* double SoTien:
  + Số tiền khách hàng phải thanh toán dựa trên vị trí ghế (VD: 45,000 hoặc 55,000 VND).
* char gioVao[10]:
  + Mảng ký tự lưu thời gian khách hàng vào rạp, định dạng hh:mm.
* khach\* next:
  + Con trỏ trỏ đến khách hàng tiếp theo trong danh sách liên kết (danh sách liên kết đơn).

A screen shot of a computer code

Description automatically generated

Hình 2.1.1: Khai báo struct khach trong hàm.

#### 2.2.1.2 Hàm stack<khach\*> luuStack.

* luuStack:
* Ngăn xếp lưu con trỏ trỏ đến các khách hàng đã bị xóa.
* Dùng để phục hồi vé cho khách nếu cần thiết (hàm phucHoiVe).
* Cơ chế LIFO (Last In, First Out): Khách hàng bị xóa gần nhất sẽ được phục hồi đầu tiên.

#### 2.2.1.3 Hàm struct list.

Cấu trúc list quản lý danh sách liên kết của các khách hàng và bảng ghế.

Thành phần của list:

* khach\* head:
  + Con trỏ trỏ đến khách hàng đầu tiên trong danh sách liên kết.
* khach\* tail:
  + Con trỏ trỏ đến khách hàng cuối cùng trong danh sách liên kết.
* int size:
  + Số lượng khách hàng hiện tại trong danh sách.
* char bangGhe[HANG][COT]:
  + Bảng trạng thái ghế ngồi trong rạp chiếu phim.
  + Kích thước cố định: HANG x COT (5x5 trong chương trình).
  + Mỗi ghế có thể ở một trong hai trạng thái:
    - '-': Ghế trống.
    - 'X': Ghế đã được đặt.
* double doanhThu:
  + Tổng doanh thu từ việc bán vé.
  + Mỗi lần thêm khách hàng, số tiền vé của khách sẽ được cộng vào doanhThu.
  + Nếu khách hàng bị xóa, số tiền này sẽ được trừ khỏi doanhThu.

A computer code with black text

Description automatically generated

Hình 2.1.2: Khai khai biến con trỏ trong danh sách và khai báo bảng ghế.

#### 2.2.1.4 Khởi tạo node hàm list::list( ) (Constructor).

* Khởi tạo danh sách liên kết rỗng (head, tail=NULL): head và tail đều được gán bằng NULL, nghĩa là chưa có khách hàng nào trong danh sách.
* Size 0, doanh thu = 0.0: khởi tạo số lượng, tổng doanh thu từ việc bán vé là 0 và 0.0 .
* Khơi tạo bảng ghế:

A close up of a number

Description automatically generated

Hình 2.1.3: Hàm khai báo bảng ghế .

* Hai vòng lặp lồng nhau để duyệt qua từng hàng (HANG) và từng cột (COT) của bảng ghế.
* Tất cả các ghế trong rạp được đánh dấu là trống ('-').

A white paper with yellow text

Description automatically generated with medium confidence

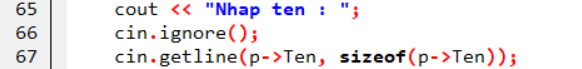
Hình 2.1.4: khi chạy bảng ghế trống.

* DeleteAll( ): (Destuctor) để xóa tất cả khách hàng trong danh sách khi đối tượng bị hủy.

#### 2.2.1.5 Hàm khach\* list::createNode()

Hàm createNode trong lớp list có chức năng tạo và khởi tạo một nút (node) kiểu khach. Mỗi nút đại diện cho thông tin của một khách hàng trong danh sách liên kết.

* khach \* p = new khach();
* Sử dụng **new** để cấp phát động bộ nhớ cho một đối tượng khach.
* Con trỏ **p** trỏ đến vùng nhớ vừa được cấp phát.
* Nhập thông tin khách hàng.



Hình 2.1.5: Code nhập tên khách hàng.

* cin.ignore(): Xóa bỏ ký tự thừa từ bộ đệm trước khi đọc chuỗi mới bằng getline.
* cin.getline(p->Ten, sizeof(p->Ten)): Nhập tên khách hàng (tối đa 30 ký tự).
* Đặt ghế ghế và ghi nhận ghế khách hàng.

A white background with text

Description automatically generated

Hình 2.1.6: Code nhập ghế hàng và ghi nhận ghế.

* Gọi hàm **datGhe** để đặt ghế.
  + Trả về true nếu ghế hợp lệ và được đặt thành công.
  + Nếu ghế không thể đặt (bị chiếm hoặc không hợp lệ), in thông báo lỗi và xóa nút p vừa tạo.
  + **return NULL;**: Trả về NULL để báo lỗi.
* Sử dụng **sprintf** để lưu vị trí ghế vào chuỗi **p->ViTri**.
  + - hang + 'A': Chuyển số hàng thành ký tự ('A', 'B', ...).
    - cot + 1: Chuyển chỉ số cột (0-based) thành số (1-based).
* Xác định giá vé và cập nhập doanh thu.

A screenshot of a computer

Description automatically generated

Hình 2.1.7: Code cập nhập giá vé và hiện doanh thu.

* Xác định giá vé.
  + Ghế ở **hàng A và B**: giá vé **45,000 VND** nếu code hang dưới 2.
  + Ghế ở **hàng C đến E**: giá vé **55,000 VND** với code if/else thì hàng trên 2.
* Cập nhập doanh thu.
* Cộng giá vé vào **doanh thu hiện tại** bằng cách sử dụng toán tử += là doanh thu+ p ra số tiền.
* Nhập giờ vào rạp rồi kiểm tra tính hợp lệ của giờ và khởi tạo con trỏ.

A computer code with colorful text

Description automatically generated with medium confidence

Hình 2.1.8: Code sử dụng để ghi thời gian vào rạp.

* Nhập giờ vào rạp.
* Nhập giờ vào rạp từ người dùng theo định dạng **hh:mm**.
* Kiểm tra tính hợp lệ của giờ vào.
* **sscanf**: Phân tích chuỗi **p->gioVao** thành hai số nguyên **hour** và **minute**.
* Kiểm tra giờ và phút hợp lệ (giờ từ 0 đến 23, phút từ 0 đến 59).
* Nếu không hợp lệ, yêu cầu nhập lại.
* Khởi tạo con trỏ(**next**) và trả về nút vừa mới tạo:
* Đặt con trỏ **next** của nút khách hàng thành **NULL** (vì đây là nút độc lập ban đầu).
* Trả về con trỏ đến nút khách hàng vừa khởi tạo.

#### 2.2.1.6 Tạo một nút khách hàng vào đầu danh sách.

* Tạo một nút khách hàng và kiểm tra:

A black and red text

Description automatically generated

Hình 2.1.9: Hàm tạo nút bằng createNode và kiểm tra nếu hàm dừng thì tạo thấp bại.

* Tạo một nút mới:
* Gọi hàm **createNode()** để tạo một nút khách hàng mới.
* **p** là con trỏ trỏ đến nút mới được tạo.
* Nếu việc tạo nút thất bại (trả về NULL), không tiếp tục thêm vào danh sách.
* Kiểm tra nút.
* Nếu con trỏ **p** là NULL, điều này có nghĩa là không thể tạo nút mới (ví dụ: lỗi đặt ghế, nhập thông tin không hợp lệ, hoặc không đủ bộ nhớ).
* Trong trường hợp này, hàm dừng lại mà không thêm gì vào danh sách.
* Liên kết nút mới với danh sách hiện tại và cập nhập sách nếu rỗng.

A computer screen shot of a computer code

Description automatically generated

Hình 2.1.10: Code liên kết danh sách và cập nhập nếu danh sách rỗng.

* Liên kết nút mới với danh sách hiện tại
* **p->next = head;**: Nút mới sẽ trỏ đến nút hiện tại ở đầu danh sách (hoặc NULL nếu danh sách rỗng).
* **head = p;**: Cập nhật con trỏ head để trỏ đến nút mới, tức là nút mới trở thành **nút đầu tiên** trong danh sách.
* Cập nhật tail nếu danh sách rỗng trước đó.
* Nếu **tail** là NULL, điều này có nghĩa là danh sách trước đó rỗng.
* Trong trường hợp này, cả head và tail đều trỏ đến nút mới.
* Tăng kích thước danh sách: Tăng số lượng phần tử trong danh sách thêm 1.
* Cách hoạt động của hàm addfirst
* Trước khi gọi addfirst:
* Danh sách có thể rỗng hoặc đã chứa các phần tử.
* Con trỏ head trỏ đến phần tử đầu tiên, và tail trỏ đến phần tử cuối cùng (nếu có).
* Sau khi gọi addfirst:
* Một nút mới được thêm vào đầu danh sách.
* Con trỏ head trỏ đến nút mới.
* Nếu danh sách trước đó rỗng, cả head và tail đều trỏ đến nút mới.

A screenshot of a computer

Description automatically generated

`Hình 2.1.11: Chương trình chạy khi chưa thêm addFirst

A screenshot of a computer screen

Description automatically generated

Hình 2.1.12: Vd khi ta một danh sách liên kết sau khi ta addfirst và thêm khách C.

#### 2.2.1.7 Tạo một nút vào cuối danh sách.

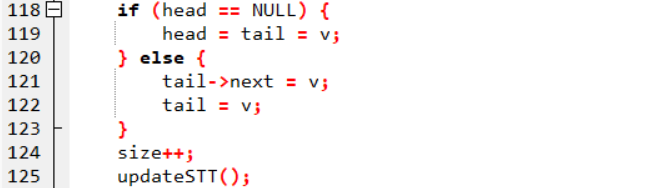
* Tạo một nút và kiểm tra.

**A close up of words

Description automatically generated**

Hình 2.1.13: Hàm tạo một nút và dừng lại khi tạo thất bại.

* Tạo một nút khách hàng mới.
* Hàm gọi **createNode()** để tạo một nút khách hàng mới.
* **p** là con trỏ trỏ đến nút mới được tạo.
* Nếu việc tạo nút thất bại (trả về NULL), không tiếp tục thêm vào danh sách.
* Kiểm tra nếu tạo nút thất bại.
* Nếu con trỏ **p** là NULL, điều này có nghĩa là không thể tạo nút mới (ví dụ: lỗi đặt ghế, nhập thông tin không hợp lệ, hoặc không đủ bộ nhớ).
* Trong trường hợp này, hàm dừng lại mà không thêm gì vào danh sách.
* Kiểm tra danh sách rỗng không nếu không thì them cuối rồi tăng kích thước.

Hình 2.1.14: Hàm kiểm tra nếu rỗng thì thêm cuối rồi tăng khích thước danh sách.

* Kiểm tra danh sách có rỗng không.
* Nếu **head == NULL**, điều này nghĩa là danh sách rỗng.
* Cả **head** và **tail** được cập nhật để trỏ đến nút mới.
* Nút mới trở thành phần tử đầu tiên và cũng là cuối cùng trong danh sách.
* Thêm vào cuối danh sách nếu không rỗng và tăng kích thước danh sách.
* **tail->next = p;**: Liên kết nút mới vào sau phần tử cuối hiện tại.
* **tail = p;**: Cập nhật **tail** để trỏ đến nút mới. Bây giờ, nút mới trở thành phần tử cuối cùng của danh sách.
* Size++: Tăng số lượng phần tử trong danh sách lên 1.
* Cách hoạt động của hàm addlast.
* **Trước khi gọi addlast:**
  + Danh sách có thể rỗng hoặc đã chứa các phần tử.
  + Con trỏ **head** trỏ đến phần tử đầu tiên và **tail** trỏ đến phần tử cuối cùng.
* **Sau khi gọi addlast:**
  + Một nút mới được thêm vào cuối danh sách.
  + Con trỏ **tail** trỏ đến nút mới.
  + Nếu danh sách trước đó rỗng, cả **head** và **tail** sẽ trỏ đến nút mới.

A screenshot of a computer

Description automatically generated

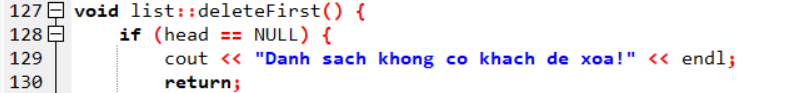
A screenshot of a computer

Description automatically generated

Hình 2.1.15: Vd khi ta một danh sách liên kết sau khi ta addlast và thêm khách C.

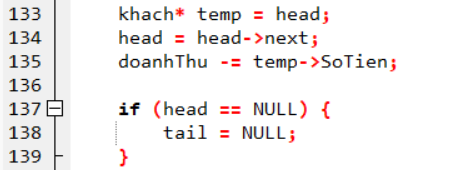
#### 2.2.1.8 Xóa một khách hàng đầu danh sách.

* Kiểm tra nếu danh sách rỗng.



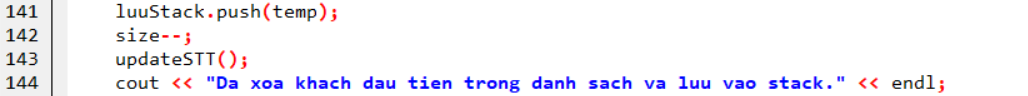
Hình 2.1.16: Hàm kiểu tra danh sách có rỗng không.

* Nếu **head** là NULL, danh sách không có phần tử nào.
* Thông báo danh sách rỗng và không thực hiện thêm hành động nào.
* Thoát khỏi hàm.
* Lưu con trỏ tạm thời xong giảm doanh thu và kiểm tra nếu danh sách trở nên rỗng.



Hình 2.1.17: Code lưu con trỏ xong giảm doanh thu và kiểm tra lại danh sách

* Lưu con trỏ tạm thời và cập nhật head.
* **temp** là con trỏ tạm thời trỏ đến phần tử đầu tiên (sẽ bị xóa).
* **head** được cập nhật để trỏ đến phần tử tiếp theo trong danh sách liên kết.
* Giảm doanh thu: Trừ số tiền của phần tử vừa bị xóa khỏi tổng doanh thu bằng toán tử -= là doanh thu trừ con trỏ tạm thời phần tử đầu bị xóa.
* Kiểm tra nếu danh sách trở nên rỗng: Sau khi xóa phần tử đầu tiên, nếu danh sách không còn phần tử nào (tức **head** là NULL), thì **tail** cũng được cập nhật về NULL.
* Lưu và cập nhập lại kích thước và thông báo.



Hình 2.1.18: Code trên lưu phần tử đã xóa, cập nhập stt, kích thước xong thông báo.

* Lưu phần tử bị xóa vào stack.
* **temp** (chứa phần tử bị xóa) được lưu vào **stack luuStack**.
* Điều này cho phép **khôi phục** lại phần tử này sau nếu cần.
* Cập nhật kích thước và số thứ tự.
* Giảm size (số lượng phần tử trong danh sách) đi 1.
* Gọi updateSTT() để cập nhật lại số thứ tự (STT) của các khách hàng còn lại.
* Thông báo kết quả: In thông báo rằng phần tử đầu tiên đã được xóa thành công và lưu vào **stack**.
* Cách hoạt động của hàm deleteFirst
* **Trước khi gọi deleteFirst:**
  + Danh sách chứa các phần tử với con trỏ **head** trỏ đến phần tử đầu tiên.
  + Con trỏ **tail** trỏ đến phần tử cuối cùng.
* **Sau khi gọi deleteFirst:**
  + Phần tử đầu tiên bị loại bỏ khỏi danh sách.
  + **head** trỏ đến phần tử thứ hai (nếu còn phần tử).
  + **tail** có thể cập nhật về NULL nếu danh sách trở nên rỗng.
  + Phần tử bị xóa được lưu vào **stack** để có thể khôi phục lại nếu cần.

A screenshot of a computer

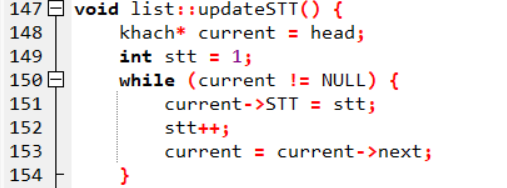
Description automatically generated

A screenshot of a computer

Description automatically generated

Hình 2.1.19: Vd khi cho danh sách liên kết trên và xóa khách A bằng hàm deleteFirst và giảm doanh thu 45000 khách A.

#### 2.2.1.9 Hàm tự cập nhập số thứ tự trong danh sách liên kết.

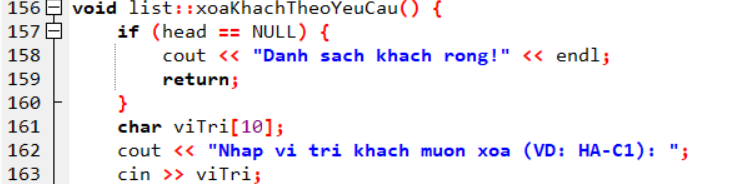
****

Hình 2.1.20: Khai triển code tự cập nhập STT.

* Kiểm tra danh sách có rỗng không.
* Nếu **head** là NULL, nghĩa là danh sách hiện không có phần tử nào.
* Không cần làm gì thêm và thoát khỏi hàm.
* Khởi tạo con trỏ current và biến đếm stt.
* **current**: Con trỏ dùng để duyệt qua các phần tử trong danh sách, ban đầu trỏ vào phần tử đầu tiên (**head**).
* **stt**: Biến đếm để cập nhật số thứ tự cho từng phần tử, khởi tạo từ **1**.
* Duyệt qua danh sách và cập nhật STT.
* **Vòng lặp while** duyệt qua từng phần tử trong danh sách từ **head** đến **tail**.
* Tại mỗi bước:
* **current->STT** được gán giá trị **stt**.
* **current** được di chuyển sang phần tử tiếp theo (**current->next**).
* Tăng biến **stt** để chuẩn bị gán STT cho phần tử kế tiếp.
* Cách thức hoạt động.
* Duyệt qua từng phần tử trong danh sách.
* Gán giá trị **STT** từ 1 đến hết.

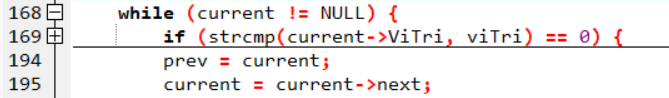
#### 2.2.1.10 Xóa khách hàng theo yêu cầu.

* Kiểm tra và nhập vị trí muốn xóa.

****

Hình 2.1.21: Triển khai code kiểm tra danh sách rỗng hay không và xóa vị trí muốn xóa.

* Kiểm tra danh sách có rỗng hay không.
* Nếu **head** là NULL, tức là danh sách liên kết trống, không có khách nào để xóa.
* In ra thông báo và thoát khỏi hàm.
* Nhập vị trí ghế cần xóa.
* Người dùng nhập vị trí ghế muốn xóa, ví dụ: **HA-C1**.
* Giá trị này được lưu vào **viTri** để so sánh với vị trí ghế của từng khách hàng trong danh sách.
* Duyệt qua danh sách để tìm khách hàng.



Hình 2.1.22: Code duyệt danh sách khách hàng muốn xóa.

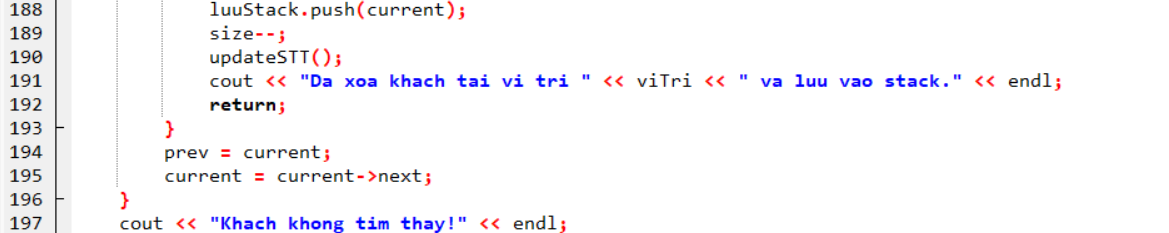
* **current**: Trỏ vào phần tử hiện tại trong danh sách, bắt đầu từ **head**.
* **prev**: Lưu trỏ của phần tử trước đó trong danh sách (khởi tạo là NULL).
* Duyệt qua danh sách liên kết:
* Nếu vị trí ghế của khách hàng (**current->ViTri**) trùng với vị trí ghế nhập vào (**viTri**), khách cần xóa đã được tìm thấy.
* Nếu không trùng, tiếp tục duyệt phần tử tiếp theo.
* Xử lý tìm kiếm khách hàng muốn xóa và xóa khách hàng.

A screenshot of a computer code

Description automatically generated

Hình 2.1.23: Xử lý khách hàng muốn xóa và xóa đầu giữa và cuối tùy theo vị trị khách hàng.

* Xử lý khi tìm thấy khách hàng cần xóa.
* **Hủy đặt chỗ ghế** bằng cách đặt lại **bangGhe** tại vị trí ghế thành **'-'**.
* Giảm **doanhThu** vì khách này đã rời đi.
* Trường hợp xóa phần tử đầu tiên.
* Nếu prev là NULL, nghĩa là đang xóa phần tử đầu danh sách (head).
* Cập nhật head sang phần tử tiếp theo.
* Nếu head trở thành NULL, danh sách trống nên cập nhật tail cũng thành NULL.
* Trường hợp xóa phần tử ở giữa hoặc cuối.
* Nếu xóa phần tử không phải đầu danh sách: **prev->next** trỏ đến phần tử sau **current**, bỏ qua **current**.
* Nếu phần tử bị xóa là **tail**, cập nhật **tail** về **prev**.
* Lưu thông tin cập nhập danh sách và trường hợp không có khách hàng.

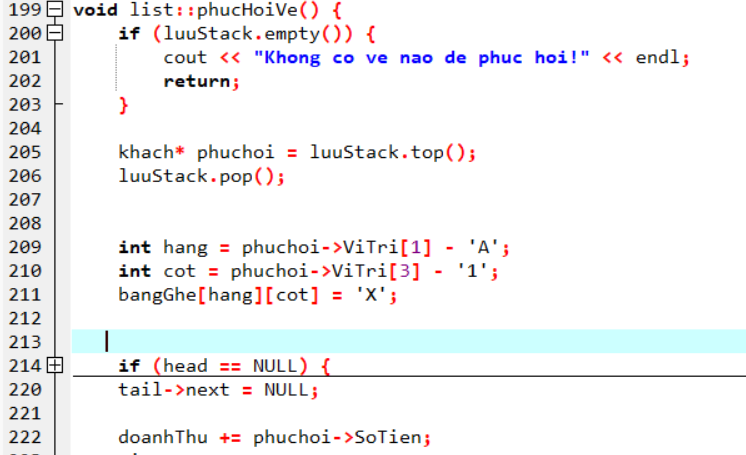


Hình 2.1.24: Code lưu thông tin khách hàng đã xóa cập nhập lại danh sách và thông báo nếu không có khách hàng.

* Lưu thông tin khách hàng đã xóa vào stack: Khách hàng đã xóa sẽ được lưu vào stack **luuStack** để có thể khôi phục sau này.
* Cập nhật lại danh sách.
* Giảm kích thước danh sách (**size**).
* Gọi **updateSTT()** để cập nhật lại số thứ tự cho các khách còn lại.
* Trường hợp không tìm thấy khách hàng: Nếu không tìm thấy khách hàng có vị trí ghế trùng khớp, thông báo lỗi và thoát khỏi hàm.

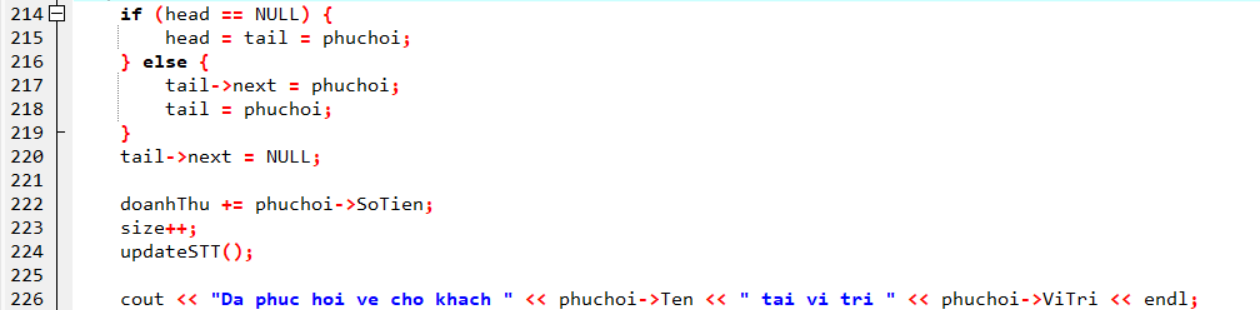
**2.2.1.11 Hàm phục hồi vé.**

* Kiểm tra và lấy thông tin tại stack và khôi phục tại ghế.

****

Hình 2.1.25: Code kiểm tra trong stack nếu có thông tin khách hàng thì khôi phục tại ghế cho khách và cập nhập lại doanh thu.

* Kiểm tra nếu stack rỗng: **luuStack.empty()**: Kiểm tra nếu stack **luuStack** rỗng.
* Nếu **luuStack** rỗng, tức là không có khách hàng nào để khôi phục.
* In ra thông báo và thoát khỏi hàm.
* Lấy thông tin khách hàng từ stack.
* **luuStack.top()**: Lấy con trỏ tới khách hàng trên đỉnh stack nhưng không xóa phần tử.**luuStack.**
* **pop()**:
* Xóa khách hàng vừa lấy khỏi stack.**kh** là con trỏ trỏ đến khách hàng sẽ được khôi phục.
* Khôi phục trạng thái ghế.
* **ViTri[1]**: Ký tự chỉ hàng ghế, chuyển thành chỉ số hàng.
* **ViTri[3]**: Ký tự chỉ cột ghế, chuyển thành chỉ số cột.
* **bangGhe[hang][cot] = 'X'**: Đánh dấu ghế này đã được đặt lại.
* **doanhThu += kh->SoTien**: Tăng doanh thu theo số tiền mà khách hàng đã trả.
* Thêm khách cập nhập STT và thông báo.

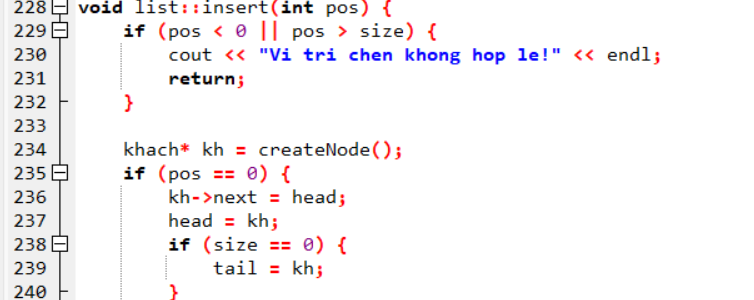


Hình 2.1.26: Hàm trên them khách và cập nhập lại stt và thông báo nếu cập nhập thành công.

* Thêm khách hàng vào danh sách liên kết.
* Trường hợp danh sách liên kết rỗng: Nếu **head == NULL**, tức là danh sách liên kết đang rỗng:
* Đặt **head** và **tail** trỏ đến khách hàng khôi phục.
* **kh->next = NULL**: Đảm bảo khách hàng không trỏ tới phần tử nào.
* Trường hợp danh sách liên kết không rỗng.
* **tail->next = kh**: Thêm khách hàng mới vào cuối danh sách.
* Cập nhật **tail** để trỏ tới khách hàng mới.
* **kh->next = NULL**: Đảm bảo tail không trỏ đến đâu.
* Cập nhật kích thước danh sách và STT
* **size++**: Tăng số lượng khách hàng trong danh sách.
* **updateSTT()**: Cập nhật lại **STT** cho tất cả khách hàng trong danh sách.
* Thông báo khôi phục thành công: In thông báo xác nhận khách hàng đã được khôi phục và hiển thị vị trí ghế của khách đó.
* Mục đích của hàm: **phucHoiVe()** cho phép khôi phục lại một khách hàng đã xóa trước đó. Điều này hữu ích trong các trường hợp xóa nhầm hoặc cần khôi phục thông tin đặt chỗ của khách hàng nhanh chóng.

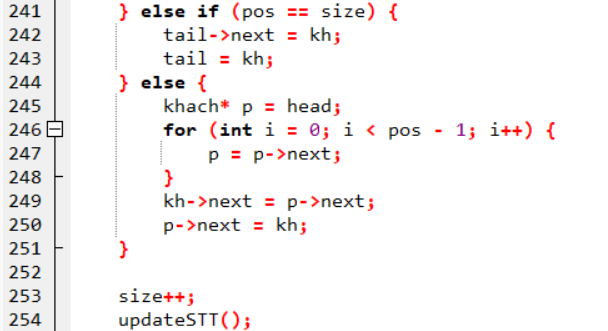
#### 2.2.1.12 Chèn một khách hàng tại một vị trí (vị trí pop).

* Đầu tiên kiểm tra danh sách rồi tạo một node mới rồi chèn vô.



Hình 2.1.27: Hàm kiểm tra vị trí chèn có hợp lệ hay không tạo node rồi chèn vào đầu danh sách.

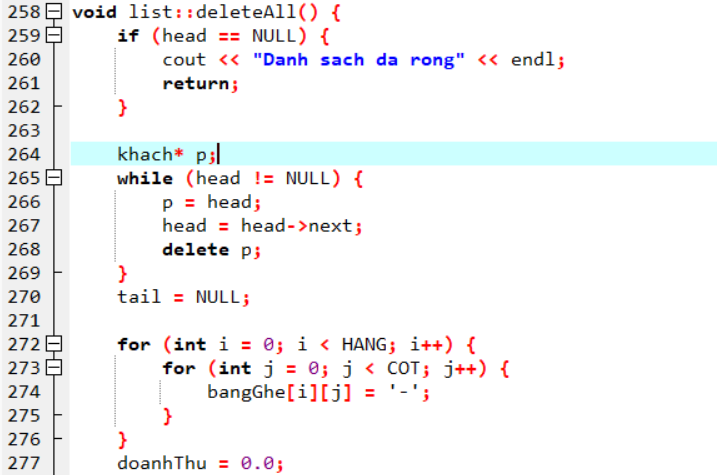
* Kiểm tra tính hợp lệ của vị trí.
* **pos** là vị trí được chỉ định để chèn khách hàng.
* Kiểm tra:
* Nếu **pos < 0**: Vị trí nằm ngoài danh sách (âm).
* Nếu **pos > size**: Vị trí vượt quá số lượng phần tử trong danh sách.
* Nếu vị trí không hợp lệ, thông báo và thoát khỏi hàm.
* Tạo một khách hàng mới.
* **createNode()**: Tạo một khách hàng mới với thông tin được nhập từ người dùng.
* **kh** là con trỏ trỏ đến khách hàng mới.
* Chèn khách hàng mới vào đầu danh sách.
* **kh->next = head**: Khách hàng mới trỏ đến phần tử đầu tiên hiện tại.
* **head = kh**: Cập nhật lại **head** để trỏ đến khách hàng mới.
* Nếu danh sách trước đó rỗng (**size ==0):** tail = kh: Tail cũng trỏ đến khách hàng mới, vì đây là phần tử duy nhất.
* Chèn cuối, giữa và cập nhập stt và tăng kích thước danh sách.



Hình 2.1.28: Code chèn vị trí cuối và giữa danh sách khi chèn xong tự động tăng kích thước và tăng stt.

* Chèn khách hàng mới vào cuối danh sách: Nếu **pos == size**, tức là chèn vào cuối danh sách:
* **tail->next = kh**: Tail hiện tại trỏ đến khách hàng mới.
* **tail = kh**: Cập nhật **tail** để trỏ đến khách hàng mới.
* Chèn khách hàng mới vào giữa danh sách: nếu pos nằm giữa:
* Dùng con trỏ tạm **p** để duyệt danh sách đến phần tử ngay trước vị trí cần chèn (**pos - 1**).
* **kh->next = p->next**: Khách hàng mới trỏ đến phần tử sau vị trí chèn.
* **p->next = kh**: Phần tử trước vị trí chèn trỏ đến khách hàng mới.
* Cập nhật kích thước và STT.
* **size++**: Tăng số lượng phần tử trong danh sách.
* **updateSTT()**: Cập nhật **STT** cho toàn bộ khách hàng trong danh sách để đảm bảo tính đúng đắn.
* **Mục đích của hàm**
* **insert(int pos)** cho phép chèn một khách hàng vào vị trí bất kỳ trong danh sách liên kết.
* Hàm này linh hoạt trong việc thêm mới khách hàng ở đầu, giữa, hoặc cuối danh sách, đồng thời đảm bảo tính chính xác của trạng thái danh sách và STT sau mỗi lần chèn.

#### 2.2.1.13 Hàm xóa tất cả khách hàng.

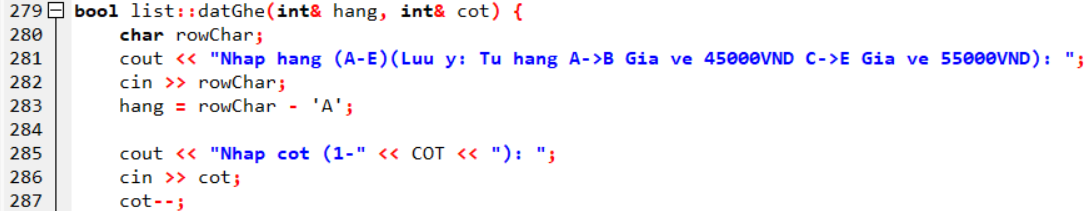
****

Hình 2.1.29: Triển khai hàm xóa tất cả.

* Lặp qua danh sách và xóa từng phần tử.
* **while (head != nullptr)**: Lặp qua toàn bộ danh sách liên kết cho đến khi danh sách rỗng (khi **head** trở thành **nullptr**).
  + **khach\* temp = head**: Con trỏ tạm **temp** trỏ đến nút hiện tại (**head**).
  + **head = head->next**: Di chuyển **head** đến nút tiếp theo trong danh sách.
  + **delete temp**: Giải phóng bộ nhớ của nút hiện tại (**temp**).
* Hành động này đảm bảo rằng bộ nhớ động của từng nút được giải phóng đúng cách, tránh rò rỉ bộ nhớ.
* Cập nhật lại các thuộc tính sau khi danh sách rỗng.
* **tail = nullptr**: Sau khi xóa hết các phần tử, **tail** (điểm cuối danh sách) cũng được đặt về **nullptr** để biểu thị rằng danh sách hiện tại rỗng.
* **size = 0**: Cập nhật số lượng phần tử trong danh sách về **0**, vì tất cả các phần tử đã bị xóa.
* Mục đích của hàm: Hàm **deleteAll** có vai trò chính:
* **Giải phóng toàn bộ bộ nhớ động** đã cấp phát cho các phần tử trong danh sách.
* Đảm bảo không còn nút nào tồn tại trong danh sách.
* Đặt danh sách trở về trạng thái ban đầu (rỗng) để sẵn sàng cho các thao tác mới.

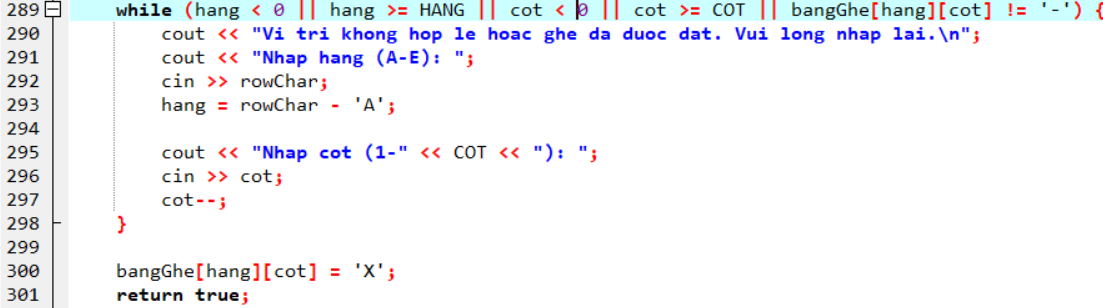
#### 2.2.1.14 Đặt ghế.

* Nhập thông tin vị trí ghế.



Hình 2.1.30: Hàm khai báo hàng ghế cốt ghế giá ghế cho từng hàng.

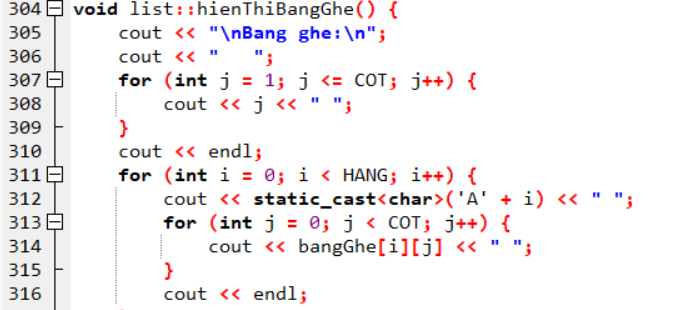
* **rowChar**: Ký tự hàng được nhập (A, B, C, ...).
* **hang = rowChar - 'A'**: Chuyển ký tự hàng thành chỉ số số nguyên từ 0 (A -> 0, B -> 1, ...).
* **cot**: Cột nhập vào là giá trị từ 1 đến **COT**
* **cot--**: Chuyển giá trị cột về dạng 0
* Kiểm tra tính hợp lệ đánh dấu và trả lại kết quả.



Hình 2.1.31: Code kiểm tra vị trí đó hợp lệ hay không và đánh dấu X nếu đã đặt xong trả lại true.

* **hang < 0 || hang >= HANG**: Kiểm tra xem chỉ số hàng có nằm ngoài phạm vi hợp lệ hay không (phạm vi hàng là từ A đến E, tương ứng từ 0 đến HANG - 1).
* **cot < 0 || cot >= COT**: Kiểm tra xem chỉ số cột có nằm ngoài phạm vi hợp lệ hay không (phạm vi cột từ 1 đến **COT**, chuyển thành 0-based từ 0 đến COT - 1).
* **bangGhe[hang][cot] != '-'**: Kiểm tra xem ghế tại vị trí chỉ định có trống không. Nếu không trống (đã đặt), yêu cầu nhập lại vị trí.
* Nếu bất kỳ điều kiện nào trong các điều kiện trên vi phạm, hệ thống yêu cầu người dùng nhập lại cho đến khi tìm được vị trí hợp lệ.
* Đánh dấu ghế đã đặt: Khi ghế hợp lệ và trống (**bangGhe[hang][cot] == '-'**), ghế đó sẽ được đánh dấu là đã đặt bằng ký hiệu 'X'.
* Trả về kết quả: Hàm trả về **true** nếu việc đặt ghế thành công.
* **Mục đích của hàm.**
* Đảm bảo rằng người dùng chỉ có thể đặt ghế tại các vị trí hợp lệ.
* Tránh đặt trùng ghế đã có khách khác chọn.
* Cập nhật trạng thái ghế sau khi đặt thành công.

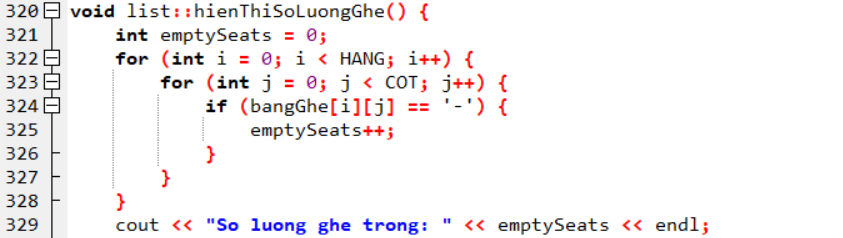
#### 2.2.1.15 Hiển thị bảng ghế.

****

Hình 2.1.32: Code hiển thị bảng ghế với “X” đã đặt “–“ còn trống.

* In tiêu đề bảng ghế.
* **cout << "Bang ghe:"**: In tiêu đề "Bảng ghế".
* **cout << " "**: In khoảng cách đầu dòng để căn chỉnh tiêu đề cột.
* **for (int j = 1; j <= COT; j++):**
* Duyệt qua các cột từ 1 đến COT (số cột của bảng ghế).
* In số thứ tự của từng cột (1, 2, 3, ...) để người dùng dễ theo dõi vị trí.
* In từng ghế ra.
* for (int i = 0; i < HANG; i++):
* Duyệt qua từng hàng từ **0** đến **HANG-1**.
* Mỗi hàng được in bắt đầu bằng ký tự tương ứng (A, B, C, ...).
* cout << static\_cast<char>('A' + i): Chuyển chỉ số hàng (i) thành ký tự (A, B, C, ...) dựa trên vị trí tương ứng (A là hàng 0).
* for (int j = 0; j < COT; j++): Duyệt qua từng ghế trong hàng hiện tại.
* cout << bangGhe[i][j]: In trạng thái của ghế hiện tại:
* 'X': Ghế đã được đặt.
* '-': Ghế còn trống.
* Mục đích của hàm.
* Cung cấp giao diện trực quan để hiển thị trạng thái ghế.
* Giúp người dùng dễ dàng kiểm tra ghế trống và ghế đã được đặt.

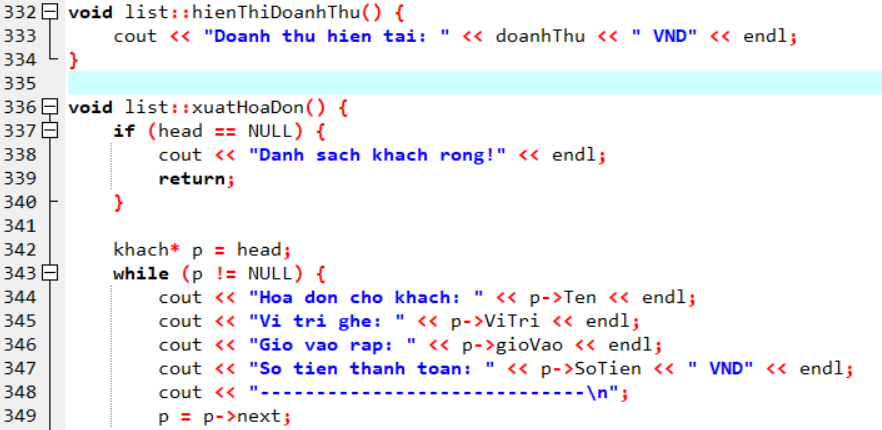
#### 2.2.1.16 Hiển thị số lượng ghế.



Hình 2.1.33: Hàm khai báo sống lượng ghế trống.

* Khai báo biến : Biến emptySeatsđược khởi tạo bằng 0. Biến này sẽ lưu số lượng ghế hiện đang trống.
* Vòng lặp for kép:
* Vòng lặp bên ngoài forlặp qua các hàng ( HANG), và forvòng lặp bên trong lặp qua các cột ( COT) trong sơ đồ chỗ ngồi.
* bangGhelà mảng 2D biểu diễn sơ đồ chỗ ngồi, với các hàng được lập chỉ mục theo ký tự i, các cột được lập chỉ mục theo ký tự j.
* Mỗi phần tử bangGhe[i][j]biểu thị trạng thái của một ghế:
* '-'chỉ ra một chỗ ngồi trống.
* 'X'chỉ ra một đặt phòng.
* Kiểm tra tình trạng: Đối với mỗi chỗ ngồi trong biểu đồ, hàm sẽ kiểm tra xem chỗ đó có trống không ( '-').
* Cuối cùng thông báo kết quả.

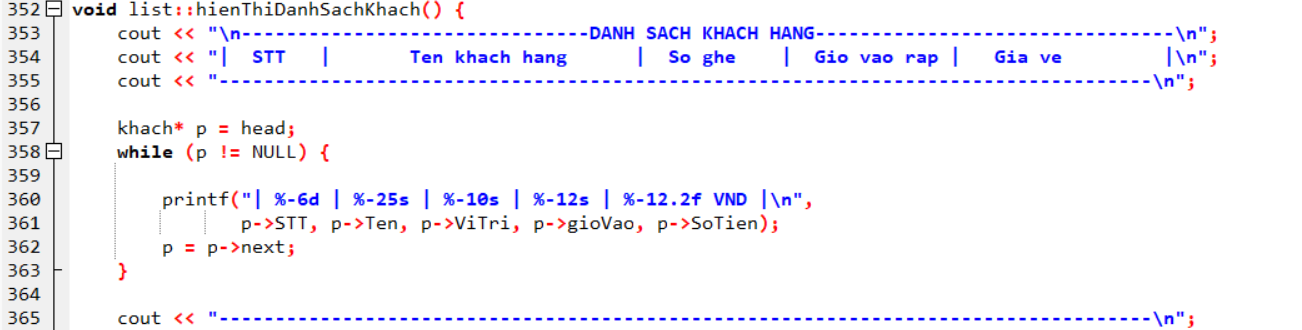
#### 2.2.1.17 Xuất doanh thu phòng chiếu và xuất hóa đơn thông tin vé.

****

Hình 2.1.34: Hàm báo cáo doanh thu phòng vé và hóa đơn thông tin khách hàng.

* Void list::hienthiDoanhThu(): hàm sẽ hiển thị doanh thu của phòng chiếu với đơn vị tiền tệ là VND.
* Hàm void list::xuatHoaDon().
* Kiểm tra danh sách trống: Trước khi bắt đầu in hóa đơn, hàm kiểm tra xem danh sách khách hàng có rỗng không. Nếu head == NULL (tức là không có khách hàng trong danh sách), hàm sẽ thông báo và dừng lại mà không làm gì thêm.
* Khởi tạo con trỏ p: Khởi tạo con trỏ p trỏ đến đầu danh sách khách hàng (head). Con trỏ này sẽ được sử dụng để duyệt qua tất cả các phần tử trong danh sách liên kết.
* Duyệt qua tất cả các khách hàng trong danh sách:
* Dùng một vòng lặp while để duyệt qua danh sách liên kết. Điều kiện lặp là p != NULL, tức là vòng lặp sẽ tiếp tục cho đến khi con trỏ p chỉ vào NULL, tức là khi đã duyệt hết danh sách.
* Sau mỗi lần in thông tin của một khách hàng, con trỏ p được di chuyển đến khách hàng tiếp theo (p = p->next).
* In thông tin hóa đơn:
* p->Ten: Tên khách hàng.
* p->ViTri: Vị trí ghế của khách hàng (ví dụ: HA-C1).
* p->gioVao: Giờ vào rạp của khách hàng (ví dụ: 14:30).
* p->SoTien: Số tiền khách hàng phải thanh toán (tính theo giá vé).
* In dấu phân cách giữa các hóa đơn: Sau khi in thông tin của mỗi khách hàng, một dòng phân cách sẽ được in để dễ dàng nhận biết các hóa đơn của từng khách hàng.

#### 2.2.1.18 Hiển thị bảng danh sách khách.

****

Hình 2.1.35: Code thể hiện bảng danh sách gồm những gì và nhập thông tin khách hàng.

* Kiểm tra danh sách trống: Trước khi bắt đầu in hóa đơn, hàm kiểm tra xem danh sách khách hàng có rỗng không. Nếu head == NULL (tức là không có khách hàng trong danh sách), hàm sẽ thông báo và dừng lại mà không làm gì thêm.
* Khởi tạo con trỏ p: Khởi tạo con trỏ p trỏ đến đầu danh sách khách hàng (head). Con trỏ này sẽ được sử dụng để duyệt qua tất cả các phần tử trong danh sách liên kết.
* Duyệt qua tất cả các khách hàng trong danh sách:
* Dùng một vòng lặp while để duyệt qua danh sách liên kết. Điều kiện lặp là p != NULL, tức là vòng lặp sẽ tiếp tục cho đến khi con trỏ p chỉ vào NULL, tức là khi đã duyệt hết danh sách.
* Sau mỗi lần in thông tin của một khách hàng, con trỏ p được di chuyển đến khách hàng tiếp theo (p = p->next).
* In thông tin hóa đơn**:** Trong mỗi lần lặp, thông tin hóa đơn của khách hàng được in ra:
* p->Ten: Tên khách hàng.
* p->ViTri: Vị trí ghế của khách hàng (ví dụ: HA-C1).
* p->gioVao: Giờ vào rạp của khách hàng (ví dụ: 14:30).
* p->SoTien: Số tiền khách hàng phải thanh toán (tính theo giá vé).
* In dấu phân cách giữa các hóa đơn: Sau khi in thông tin của mỗi khách hàng, một dòng phân cách sẽ được in để dễ dàng nhận biết các hóa đơn của từng khách hàng.

### 2.2.2 Hiển thị menu và xử lý lựa chọn của người dùng trong vòng lặp do-while.

* **Menu gồm các lựa chọn.**

**A screenshot of a computer code

Description automatically generated**

Hình 2.2.1: Menu lựa chọn để sử dụng chức năng.

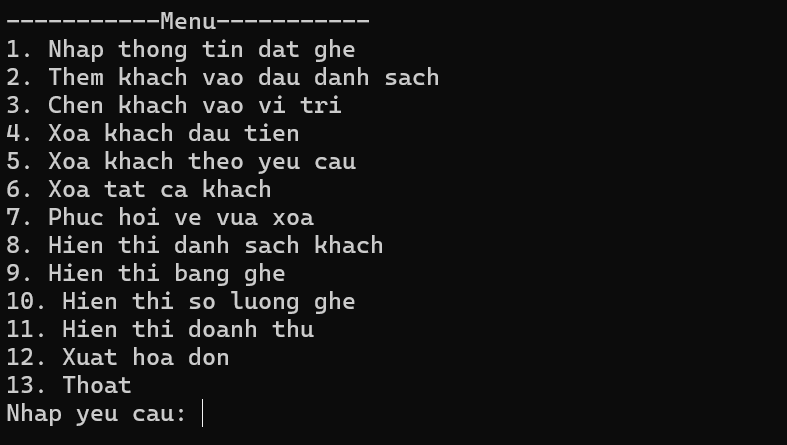
* Xử lý lựa chọn của người dùng bằng vòng lặp do-while

A screenshot of a computer program

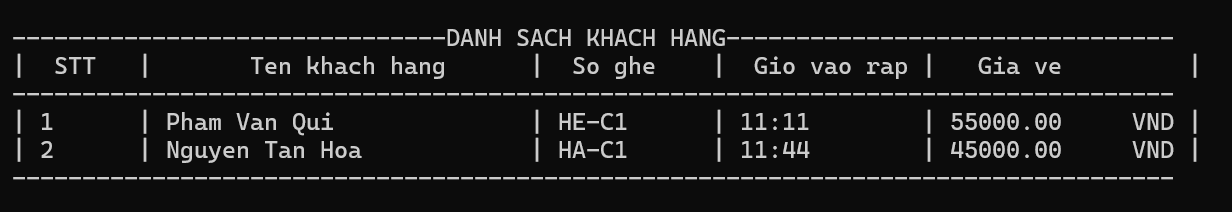
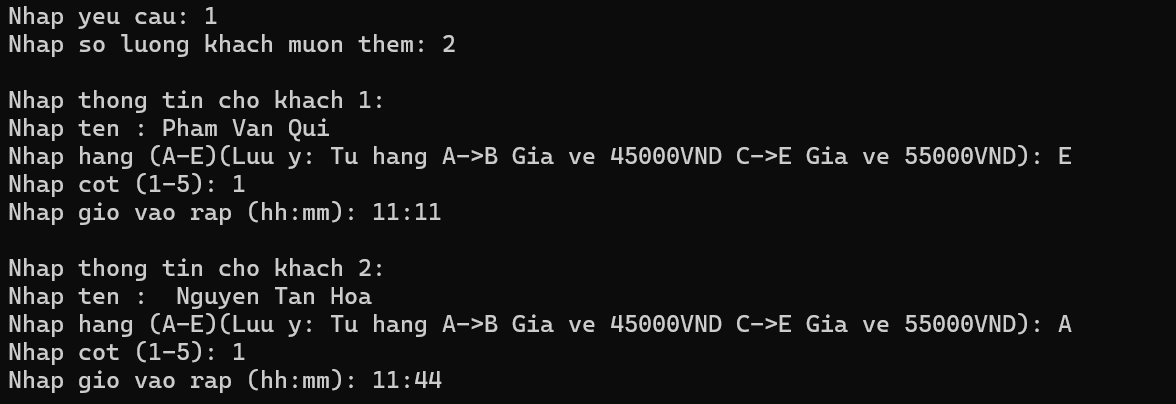
Description automatically generated

Hình 2.2.2: Vòng lặp switch case để xử lí các lựa chọn của người dùng.

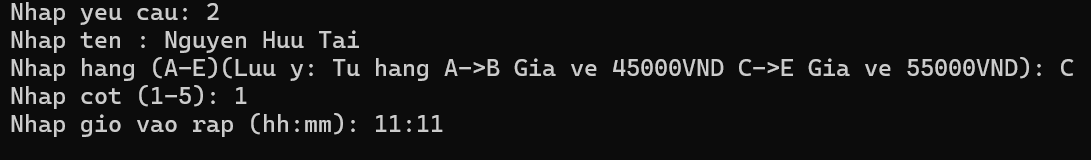
## 2.3. Giao diện chương trình

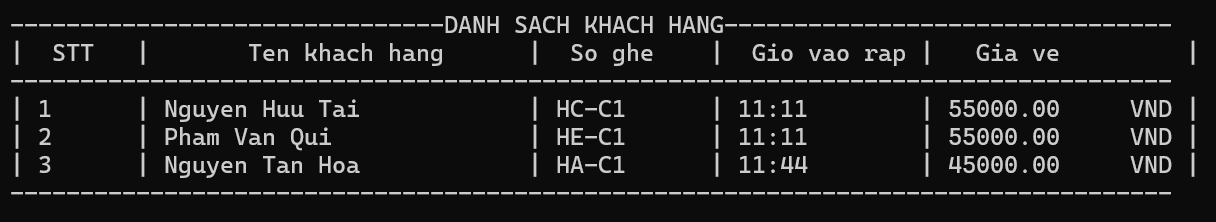
**

Hình 2.3.1 Giao diện khi bắt đầu chạy.

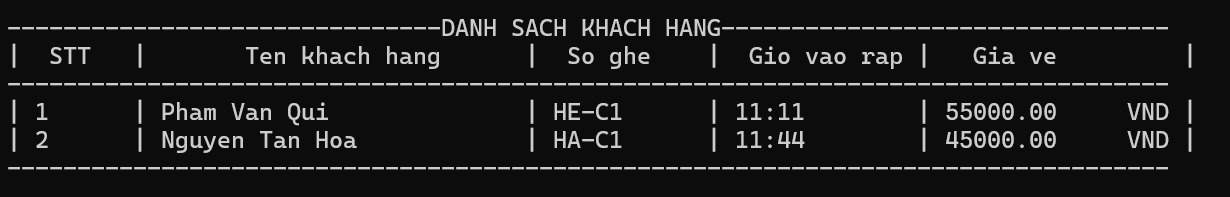
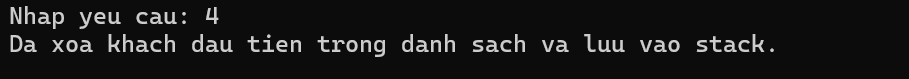
**

Hình 2.3.2 Hiển thị danh sách khách hàng.

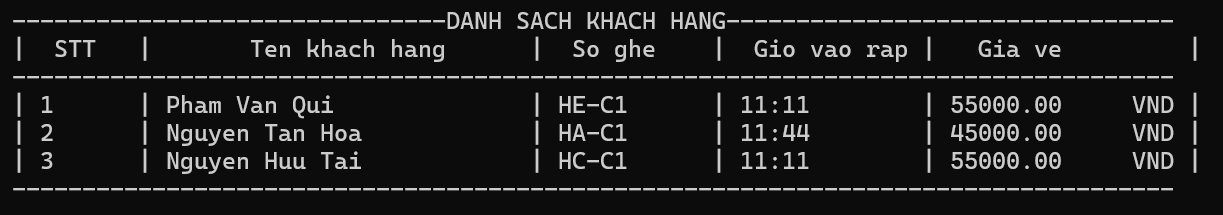
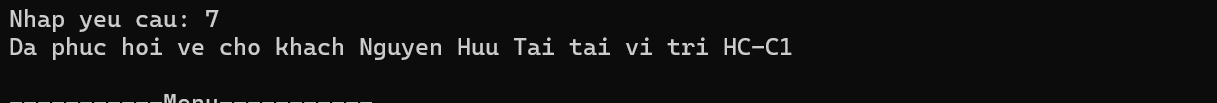
**

**

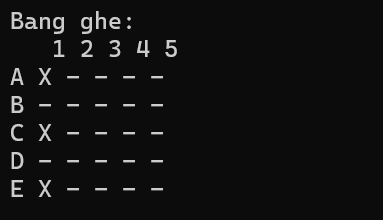
Hình 2.3.3 Lựa chọn “2”, thêm khách vào đầu danh sách.

**

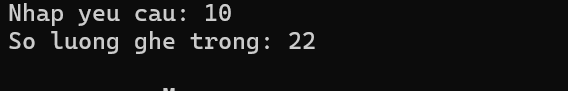
Hình 2.3.4 Lựa chọn “4” xóa khách đầu tiên khỏi danh sách*.*

**

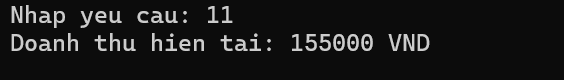
Hình 2.3.5 Lựa chọn “7” phục hồi khách vừa **xóa.**



Hình 2.3.6 Lựa chọn “9” hiển thị bảng ghế.

**

Hình 2.3.7 Lựa chọn “10” Hiển thị số ghế còn trống.

**

Hình 2.3.8 Lựa chọn “11” Hiển thị doanh thu hiện tại.

**

Hình 2.3.9 Lựa chọn “12” Xuất hóa đơn cho khách hàng.

# PHỤ LỤC

## Phụ lục 1: Hướng dẫn cài đặt

### 1. Cài Đặt Trình Biên Dịch C++

* + Đảm bảo máy tính đã có trình biên dịch C (trên Windows).

### 2. Tải Mã Nguồn

* + Sao chép mã nguồn cung cấp hoặc lưu nó dưới dạng tệp .cpp.

### 3. Biên Dịch Chương Trình

* + Mở terminal hoặc command prompt, điều hướng đến thư mục chứa tệp .cpp đã lưu, và biên dịch mã nguồn bằng lệnh
  + Sao chép mã

### 4. Chạy Chương Trình

* + Sau khi biên dịch thành công, bạn có thể chạy chương trình bằng lệnh vừa sao chép . Điều này sẽ khởi động ứng dụng đặt vé rạp chiếu phim.

## Phụ lục 2: hướng dẫn sử dụng

Ứng dụng này quản lý việc đặt chỗ trong rạp chiếu phim. Hướng dẫn dưới đây giải thích cách sử dụng từng chức năng trong chương trình**.**

### 1. Các Tùy Chọn Trong Menu Chính

* + Sau khi khởi động, chương trình sẽ hiển thị menu chính với các tùy chọn cho các hành động khác nhau.

### 2. Đặt Ghế và Xem Thông Tin

* + Tùy chọn 1: Nhập số lượng khách, sau đó cung cấp thông tin cho từng khách (tên, vị trí ghế, giờ vào). Ghế của khách sẽ được thêm vào danh sách.
  + Tùy chọn 2: Thêm một khách mới vào đầu danh sách.
  + Tùy chọn 3: Cho phép chèn một khách vào vị trí cụ thể trong danh sách.
  + Tùy chọn 4: Xóa khách đầu tiên trong danh sách.
  + Tùy chọn 5: Xóa một khách cụ thể dựa trên vị trí ghế.
  + Tùy chọn 6: Xóa toàn bộ dữ liệu khách hàng trong danh sách.

### 3. Các Tùy Chọn Khác

* + Tùy chọn 7: Phục hồi lại vé vừa xóa gần nhất từ ngăn xếp.
  + Tùy chọn 8: Hiển thị danh sách khách hiện có.
  + Tùy chọn 9: Hiển thị sơ đồ chỗ ngồi trong rạp.
  + Tùy chọn 10: Hiển thị số lượng ghế trống.
  + Tùy chọn 11: Hiển thị doanh thu hiện tại từ việc bán vé.
  + Tùy chọn 12: Xuất hóa đơn cho từng khách trong danh sách.
  + Tùy chọn 13: Thoát chương trình**.**

# TÀI LIỆU THAM KHẢO

[1]. Tài liệu giảng dạy, giáo trình Cấu Trúc Dữ Liệu và Giải Thuật NXB Giao Thông Vận Tải-2006. [Accessed 10/11/2024]

[2]. [Github quản lí rạp chiếu phim](https://github.com/luon-dinh/QuanLyRapChieuPhim). [Accessed 10/11/2024]

[3]. [Danh sách liên kết đơn.](https://nguyenvanhieu.vn/danh-sach-lien-ket-don/#google_vignette) [Accessed 10/11/2024]

[4]. [CodeLearn Cơ bản nhưng quan trọng Stack và Queue](https://codelearn.io/sharing/stack-va-queue-trong-cpp). [Accessed 10/11/2024]