**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 BÁN HÀNG**

Giảng viên hướng dẫn: TRẦN THỊ DUNG

Sinh viên thực hiện: TỪ VĂN HIẾU

NGUYỄN THANH NAM

NGUYỄN MINH NGỌC

Lớp : KĨ THUẬT ĐIỆN TỬ -VIỄN THÔNG

Khoá : K64

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 BÁN HÀNG**

Giảng viên hướng dẫn: TRẦN THỊ DUNG

Sinh viên thực hiện: TỪ VĂN HIẾU

NGUYỄN THANH NAM

NGUYỄN MINH NGỌC

Lớp : KĨ THUẬT ĐIỆN TỬ -VIỄN THÔNG

Khoá : K64

Tp. Hồ Chí Minh, năm 2024

**LỜI CẢM ƠN**

Trước đó, chúng em xin bày tỏ lòng biết ơn sâu sắc đến cô Trần Thị Dung, người đã tận dụng tình giảng dạy, hướng dẫn và truyền đạt những kiến ​​thức quý báu trong suốt quá trình học tập. Những lời giảng dạy, chia sẻ của thầy/cô không chỉ là kim chỉ nam giúp chúng em hoàn thành bài tập tốt mà còn giúp chúng em phát triển tư duy, mở rộng tầm hiểu biết và tạo nền tảng cho các bước tiến xa hơn trong con đường học tập và sự nghiệp sau này. Chúng em nhận thức rõ rằng, những kiến ​​thức và kinh nghiệm mà cô đưa ra sẽ là hành động trang quý trong suốt quá trình hành động trước đó.

Em cũng xin gửi lời cảm ơn chân thành đến các bạn trong lớp, những người đã cùng chúng em đồng hành, chia sẻ kiến ​​thức và hỗ trợ nhau không chỉ trong quá trình thực hiện bài tập mà vẫn còn trong suốt thời gian học. Tinh thần đồng đội và những báu vật quý giá của các bạn đã giúp đỡ rất nhiều trong báo cáo hoàn thiện, đồng thời tạo ra một môi trường học tập và đầy ý nghĩa. Nhờ sự hỗ trợ và tập luyện của tất cả mọi người mà chúng em đã có thêm động lực và niềm vui trong học tập, vượt qua những khó khăn và thử thách.

Cuối cùng, chúng em xin dành lời cảm ơn sâu sắc đến gia đình thân yêu, những người luôn ở bên cạnh động viên và tạo ra mọi điều kiện thuận lợi nhất để chúng em có thể yên tâm học tập. Sự giúp đỡ và chăm sóc tận tình của gia đình là nguồn lực giúp họ vượt qua những thời điểm khó khăn, hoàn thành bài tập này và không ngừng nỗ lực trong quá trình học tập.

Chúng em hy vọng rằng bài tập này có thể đáp ứng được những yêu cầu được đặt ra và mang lại giá trị giá trị cao nhất. Dù còn nhiều hạn chế, chúng em sẽ nỗ lực tiếp tục học hỏi và hoàn thiện bản thân.

Chúng em xin chân thành cảm ơn!

Nhóm 6

**MỤC LỤC**

[CHƯƠNG 1:GIỚI THIỆU Ý TƯỞNG VÀ LÝ THUYẾT CƠ SỞ 6](#_Toc182155358)

[1.1 GIỚI THIỆU Ý TƯỞNG. 6](#_Toc182155359)

[**1.1.1** **LÝ DO VÀ ĐỘNG LỰC.** 6](#_Toc182155360)

[**1.1.2** **ĐỊNH NGHĨA BÀI TOÁN.** 6](#_Toc182155361)

[1.2 LÝ THUYẾT CƠ SỞ. 6](#_Toc182155362)

[**1.2.1** **CẤU TRÚC DỮ LIỆU VÀ GIẢI THUẬT LÀ GÌ?** 6](#_Toc182155363)

[**1.2.2** **DANH SÁCH LIÊN KẾT** 8](#_Toc182155364)

[**1.2.3** **MỘT SỐ THUẬT TOÁN SẮP XẾP CƠ BẢN** 13](#_Toc182155365)

[**1.2.4** **CÁC LOẠI THUẬT TOÁN TÌM KIẾM** 16](#_Toc182155366)

[**1.2.5** **BẢNG BĂM (HASH TABLE)** 17](#_Toc182155367)

[**1.2.6** **CÂY NHỊ PHÂN (BINARY TREE)** 19](#_Toc182155368)

[**1.2.7** **BỘ THƯ VIÊN CHUẨN** 21](#_Toc182155369)

[CHƯƠNG 2. XÂY DỰNG THUẬT TOÁN 23](#_Toc182155370)

[**2.2.2:KHAI BÁO DANH SÁCH:** 25](#_Toc182155371)

[**2.2.3:CÁC HÀM VOID:** 25](#_Toc182155372)

[**2.2.4 HÀM KHỞI TẠO:** 26](#_Toc182155373)

[**2.2.5:HÀM THÊM KHÁCH HÀNG:** 26](#_Toc182155374)

[**2.2.6:HÀM TÌM KIẾM KHÁCH HÀNG THEO ID:** 28](#_Toc182155375)

[**2.2.7:HÀM HIỂN THỊ KHÁCH HÀNG:** 30](#_Toc182155376)

[**2.2.9:XÓA KHÁCH HÀNG:** 35](#_Toc182155377)

[**2.2.11:THÊM SẢN PHẨM:** 38](#_Toc182155378)

[**2.2.12:TÌM KIẾM SẢN PHẨM THEO ID:** 39](#_Toc182155379)

[**2.2.13:HÀM HIỂN THỊ SẢN PHẨM:** 41](#_Toc182155380)

[**2.2.14:XÓA SẢN PHẨM:** 42](#_Toc182155381)

[**2.2.15:SẮP XẾP SẢN PHẨM:** 45](#_Toc182155382)

[Hình 2.20 45](#_Toc182155383)

[**2.2.16:CẬP NHẬT THÔNG TIN SẢN PHẨM:** 47](#_Toc182155384)

[**2.2.17:TÍNH TỔNG GIÁ:** 48](#_Toc182155385)

[**2.2.18:TÍNH TỔNG SỐ LƯỢNG:** 49](#_Toc182155386)

[**2.2.19:HÀM KIỂM TRA VÀ NHẬP SỐ LƯỢNG:** 50](#_Toc182155387)

[**2.2.20:KIỂM TRA ID KHÁCH HÀNG:** 52](#_Toc182155388)

[2.3:GIAO DIỆN CHƯƠNG TRÌNH: 54](#_Toc182155389)

[CHƯƠNG 3:KẾT QUẢ CHƯƠNG TRÌNH 55](#_Toc182155390)

# 

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

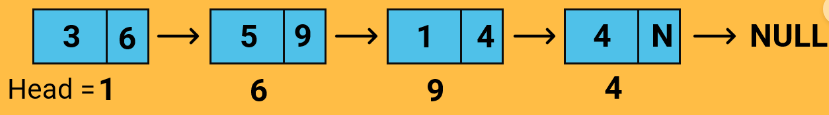
* 1. **GIỚI THIỆU Ý TƯỞNG.**
     1. **LÝ DO VÀ ĐỘNG LỰC.**
* Để đáp ứng yêu cầu môn học học cũng như mong muốn tìm hiểu, mở rộng kiến thức về ngôn ngữ lập trình nói chung và cấu trúc dữ liệu và giải thuật nói riêng, nhóm chúng em đã hướng tới đề tài với tiêu chí:
  + Hữu ích
  + Có tính thực tế
  + Áp dụng được kiến thức đã học
  + Học hỏi thêm kiến thức
* Với tiêu chí đó chúng em đã chọn đề tài “Quản lý bán hàng”.
  + 1. **ĐỊNH NGHĨA BÀI TOÁN.**
* Input:
* Khách hàng
* Sản Phẩm
* Output:
* Hiển thị khách hàng
* Thêm khách hàng
* Xóa khách hàng
* Tìm kiếm khách hàng
  1. **LÝ THUYẾT CƠ SỞ.**
     1. **CẤU TRÚC DỮ LIỆU VÀ GIẢI THUẬT LÀ GÌ?**
* **Cấu trúc dữ liệu** và **giải thuật** là hai khái niệm cốt lõi trong lĩnh vực khoa học máy tính và lập trình. Chúng không thể tách rời và thường xuyên được nghiên cứu và ứng dụng cùng nhau để giải quyết các bài toán trong lập trình.

Cấu trúc dữ liệu + Giải thuật = Chương trình

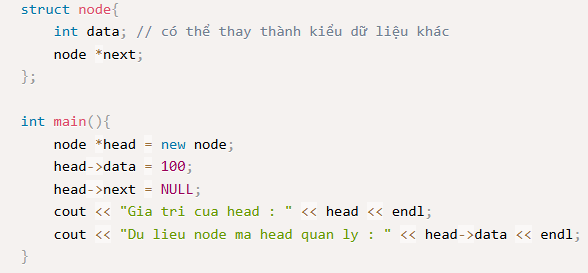
* + - 1. **CẤU TRÚC DỮ LIỆU.**
* **Cấu trúc dữ liệu (Data Structure)** là cách thức tổ chức và lưu trữ dữ liệu sao cho có thể truy cập và thay đổi một cách hiệu quả. Nó bao gồm các kiểu dữ liệu và phương pháp để tổ chức chúng sao cho dễ dàng xử lý các thao tác như tìm kiếm, chèn, xóa, cập nhật và duyệt qua dữ liệu.
* Các loại cấu trúc dữ liệu phổ biến bao gồm:
* **Mảng (Array)**: Một dãy các phần tử có cùng kiểu dữ liệu, được lưu trữ liên tiếp trong bộ nhớ. Mảng cho phép truy cập nhanh vào các phần tử bằng chỉ mục.
* **Danh sách liên kết (Linked List)**: Một chuỗi các phần tử (gọi là nút), mỗi phần tử chứa dữ liệu và một liên kết (con trỏ) đến phần tử tiếp theo. Danh sách liên kết không yêu cầu bộ nhớ liên tiếp, giúp dễ dàng thêm, xóa phần tử.
* **Ngăn xếp (Stack)**: Là một cấu trúc dữ liệu theo nguyên tắc "LIFO" (Last In, First Out) – phần tử được thêm vào sau cùng sẽ được lấy ra đầu tiên. Ví dụ: Lịch sử trình duyệt, Undo trong ứng dụng.
* **Hàng đợi (Queue)**: Là một cấu trúc dữ liệu theo nguyên tắc "FIFO" (First In, First Out) – phần tử được thêm vào đầu tiên sẽ được lấy ra đầu tiên. Ví dụ: Hệ thống in ấn, xử lý yêu cầu.
* **Cây (Tree)**: Là cấu trúc dữ liệu phân cấp, trong đó mỗi phần tử (gọi là nút) có thể có nhiều phần tử con. Cây được sử dụng trong các ứng dụng như hệ thống tệp, tìm kiếm và sắp xếp.
* **Bảng băm (Hash Table)**: Là một cấu trúc dữ liệu sử dụng một hàm băm (hash function) để ánh xạ một khóa đến một giá trị. Bảng băm cho phép tìm kiếm, chèn, xóa phần tử nhanh chóng.
  + - 1. :**GIẢI THUẬT.**

**Giải thuật** **(Algorithm)** là một tập hợp các bước đi theo một trình tự logic và hữu hạn nhằm giải quyết một vấn đề cụ thể. Giải thuật giúp chúng ta xử lý và thao tác

* trên dữ liệu một cách hiệu quả. Một giải thuật tốt không chỉ cần đúng, mà còn cần phải có độ phức tạp thấp để xử lý vấn đề nhanh chóng.
* Các loại giải thuật phổ biến bao gồm:
* **Giải thuật sắp xếp (Sorting algorithms)**: Các giải thuật sắp xếp giúp sắp xếp một tập hợp các phần tử theo một thứ tự nhất định. Một số giải thuật sắp xếp thông dụng:
  + **Sắp xếp nổi bọt (Bubble Sort)**
  + **Sắp xếp chọn (Selection Sort)**
  + **Sắp xếp chèn (Insertion Sort)**
  + **Sắp xếp nhanh (Quick Sort)**
  + **Sắp xếp trộn (Merge Sort)**
* **Giải thuật tìm kiếm (Searching algorithms)**: Các giải thuật tìm kiếm giúp tìm kiếm một phần tử trong một tập hợp dữ liệu. Một số giải thuật tìm kiếm phổ biến:
  + **Tìm kiếm tuyến tính (Linear Search)**
  + **Tìm kiếm nhị phân (Binary Search)**
* **Giải thuật chia để trị (Divide and Conquer)**: Đây là phương pháp giải quyết bài toán bằng cách chia bài toán thành các phần nhỏ hơn, giải quyết từng phần, rồi kết hợp kết quả lại. Ví dụ: **Merge Sort**, **Quick Sort**.
  + 1. **DANH SÁCH LIÊN KẾT**
* **Danh sách liên kết (DSLK - Linked List)** là một cấu trúc dữ liệu tuyến tính trong đó các phần tử (gọi là nút) không được lưu trữ liên tiếp trong bộ nhớ mà thay vào đó mỗi phần tử chứa dữ liệu và một con trỏ (hoặc tham chiếu) trỏ tới phần tử tiếp theo.
* Cấu trúc này giúp việc chèn và xóa phần tử dễ dàng và hiệu quả hơn so với các cấu trúc dữ liệu như mảng, đặc biệt là khi cần thay đổi kích thước của danh sách. Tuy nhiên, việc truy cập các phần tử trong DSLK có thể chậm hơn vì không thể truy cập trực tiếp đến phần tử theo chỉ số.
* Các thành phần cơ bản trong danh sách liên kết:
* **Nút (Node):** Mỗi phần tử trong danh sách liên kết, chứa:
* **Dữ liệu (Data):** Thông tin mà nút lưu trữ.
* **Con trỏ** **(Pointer):** Một con trỏ (hoặc tham chiếu) đến nút tiếp theo trong danh sách.
* **Đầu danh sách** **(Head):** Con trỏ hoặc tham chiếu đến nút đầu tiên của danh sách. Nếu danh sách rỗng, đầu danh sách sẽ là null
* Các loại danh sách liên kết:
* **Danh sách liên kết đơn (Singly Linked List)**
* **Danh sách liên kết đôi (Doubly Linked List)**
* **Danh sách liên kết vòng (Circular Linked List)**
* Ưu điểm của danh sách liên kết:
* **Chèn và xóa phần tử nhanh**: Bạn có thể dễ dàng chèn hoặc xóa phần tử ở bất kỳ đâu trong danh sách mà không cần phải dịch chuyển các phần tử khác, như trong mảng.
* **Kích thước linh hoạt**: Không giống như mảng, DSLK không yêu cầu không gian bộ nhớ liên tiếp và có thể thay đổi kích thước linh hoạt mà không cần phải cấp phát lại bộ nhớ.
* Nhược điểm của danh sách liên kết:
* **Truy cập chậm**: Để truy cập vào một phần tử ở vị trí cụ thể, bạn phải duyệt qua danh sách từ đầu (hoặc từ cuối đối với danh sách liên kết đôi), do đó việc truy cập các phần tử có thể chậm hơn so với mảng.
* **Tiêu tốn bộ nhớ**: Mỗi nút trong DSLK cần một con trỏ để trỏ tới nút tiếp theo (và thêm một con trỏ trong danh sách liên kết đôi), điều này làm tăng việc tiêu thụ bộ nhớ so với mảng.
  + - 1. **DANH SÁCH LIÊN KẾT ĐƠN**
* **Danh sách liên kết đơn** (Singly Linked List - DSLK đơn) là một cấu trúc dữ liệu tuyến tính, trong đó mỗi phần tử (hay còn gọi là **nút**) trong danh sách lưu trữ một giá trị và một con trỏ (hoặc tham chiếu) trỏ đến phần tử kế tiếp trong danh sách.
* Node cuối cùng trong DSLK thì phần liên kết của nó sẽ lưu con trỏ NULL



Hình 1.1



Hình 1.2

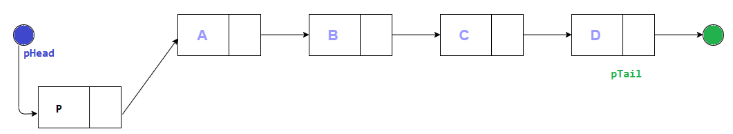
1.2.2.2 **THÊM PHẦN TỬ TRONG DSLK ĐƠN**

* Thêm vào đầu danh sách

+ Tạo một phần tử mới (node) chứa giá trị bạn muốn thêm.

+ Cập nhật con trỏ của phần tử mới để nó trỏ tới phần tử hiện tại đầu danh sách.

+ Cập nhật head (đầu danh sách) trỏ tới phần tử mới, vì nó trở thành phần tử đầu tiên của danh sách.



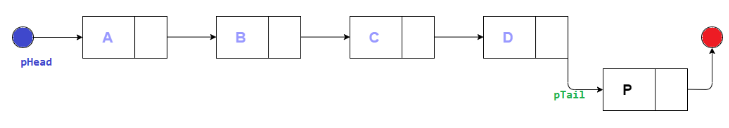
Hình 1.3

* Thêm vào cuối danh sách

+ Tạo một phần tử mới (node) chứa giá trị bạn muốn thêm.

+ Duyệt qua danh sách từ đầu đến cuối để tìm phần tử cuối cùng (phần tử có next là nullptr).

+ Cập nhật con trỏ của phần tử cuối cùng để nó trỏ đến phần tử mới.



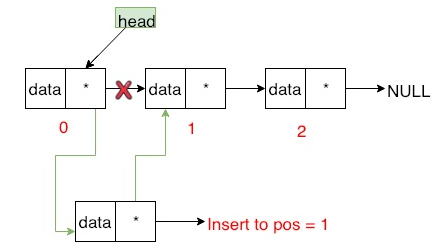
Hình 1.4

* Thêm vào vị trí bất kì

+ Xác định vị trí mà bạn muốn chèn phần tử mới.

+ Duyệt qua danh sách đến vị trí mong muốn.

+ Cập nhật con trỏ của phần tử trước đó để nó trỏ tới phần tử mới, và con trỏ của phần tử mới trỏ tới phần tử kế tiếp.



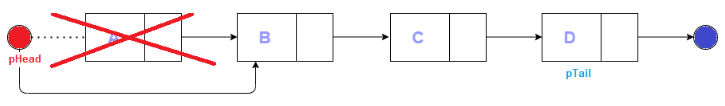
Hình 1.5

* + - 1. **XÓA PHẦN TỬ TRONG DSLK ĐƠN**
* Xóa phần tử đầu trong danh sách

+ Kiểm tra danh sách có rỗng không: Nếu danh sách trống, bạn không thể xóa được phần tử đầu.

+ Cập nhật con trỏ head: Cập nhật con trỏ head của danh sách trỏ đến phần tử tiếp theo.

+ Giải phóng bộ nhớ của phần tử đầu tiên.



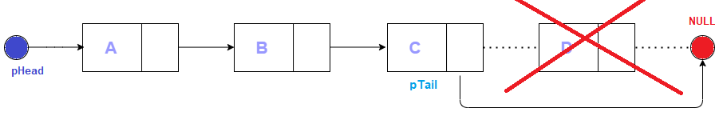
Hình 1.6

* Xóa phần tử cuối trong danh sách

+ Duyệt qua danh sách để tìm phần tử trước phần tử cuối (phần tử thứ hai cuối cùng).

+ Cập nhật con trỏ next của phần tử thứ hai cuối cùng thành nullptr.

+ Giải phóng bộ nhớ của phần tử cuối.



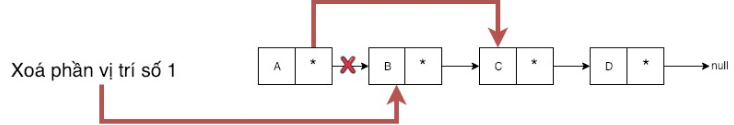
Hình 1.7

* Xóa phần tử bất kì trong danh sách

+ Duyệt qua danh sách để tìm phần tử tại vị trí cần xóa.

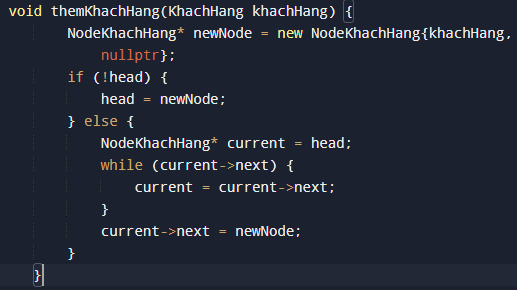
+ Cập nhật con trỏ của phần tử trước đó để nó trỏ đến phần tử tiếp theo của phần tử cần xóa.

+ Giải phóng bộ nhớ của phần tử bị xóa.



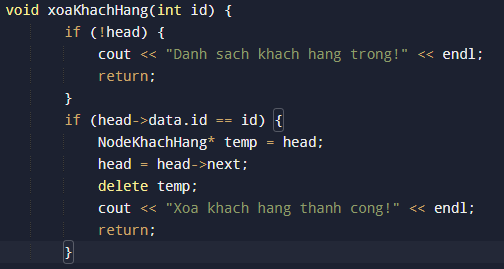
Hình 1.8

* + - 1. **VÍ DỤ VỀ DSLK ĐƠN**
* Thêm khách hàng



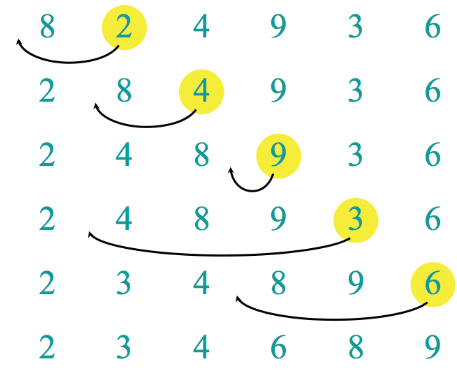
Hinh 1.9

* Xóa khách hàng



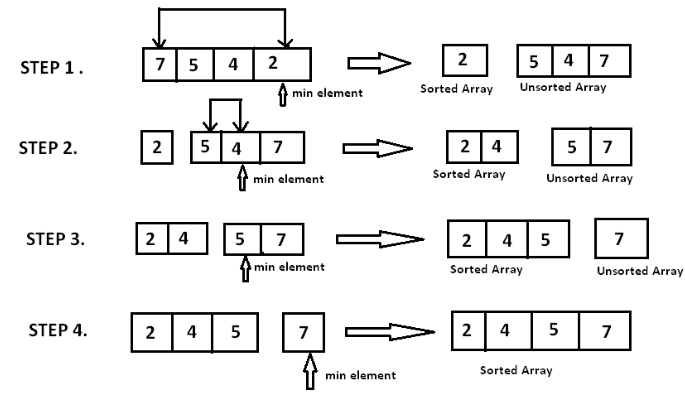
Hình 1.10

* + 1. **MỘT SỐ THUẬT TOÁN SẮP XẾP CƠ BẢN**
       1. **SẮP XẾP CHÈN (INSERTION SORT)**
* Sắp xếp chènhoạt động bằng cách lấy một phần tử từ danh sách chưa sắp xếp và "chèn" nó vào đúng vị trí trong phần danh sách đã sắp xếp.
* Với mỗi phần tử mới, ta sẽ tìm đúng vị trí mà nó phải chèn vào trong phần danh sách đã sắp xếp sao cho phần danh sách này vẫn giữ thứ tự tăng dần (hoặc giảm dần nếu sắp xếp ngược).



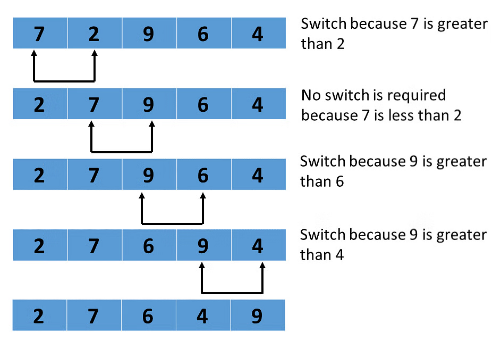
Hinh 1.11

* + - 1. **SẮP XẾP CHỌN (SELECTION SORT)**
* Duyệt qua danh sách: Tìm phần tử nhỏ nhất (hoặc lớn nhất) trong danh sách chưa được sắp xếp.
* Hoán đổi: Đặt phần tử tìm được vào đúng vị trí trong danh sách đã được sắp xếp (hoán đổi nó với phần tử đầu tiên của danh sách chưa sắp xếp).
* Lặp lại: Tiếp tục lặp lại bước 1 và 2 cho phần còn lại của danh sách.



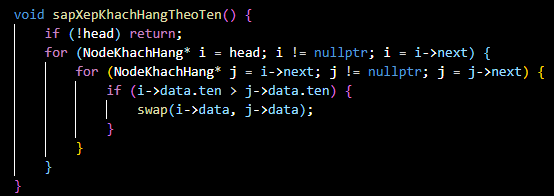
Hình 1.12

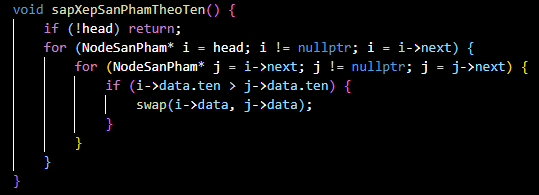
* + - 1. **SẮP XẾP NỔI BỌT (BUBBLE SORT)**
* So sánh hai phần tử liên tiếp trong danh sách.
* Hoán đổi chúng nếu phần tử trước lớn hơn phần tử sau (hoặc nhỏ hơn nếu sắp xếp giảm dần).
* Sau mỗi vòng lặp qua danh sách, phần tử lớn nhất (hoặc nhỏ nhất) sẽ được "nổi bọt" lên cuối danh sách.
* Lặp lại quá trình cho đến khi không còn cần phải hoán đổi nữa.



Hình 1.13

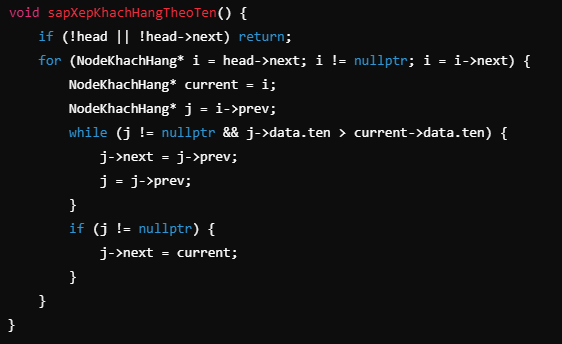
* + - 1. **VÍ DỤ VỀ THUẬT TOÁN SẮP XẾP**
* Hàm sắp xếp sử dụng bubble sort





Hình 1.14

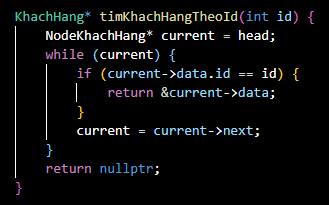
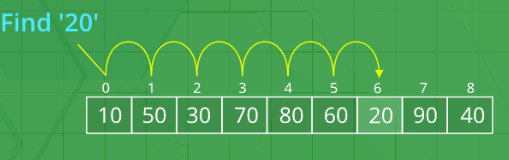
* Hàm sắp xếp sử dụng insertion sort



Hình 1.15

* + 1. **CÁC LOẠI THUẬT TOÁN TÌM KIẾM**

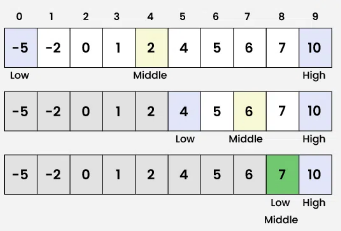
**1.2.4.1 TÌM KIẾM TUYẾN TÍNH (LINEAR SEARCH)**

* Thuật toán tìm kiếm tuyến tính kiểm tra từng phần tử trong mảng hoặc danh sách theo thứ tự để tìm phần tử cần tìm.

Hình 1.16 a Hình 1.16 b

1.2.4.2 **TÌM KIẾM NHỊ PHÂN (BINARY SEARCH)**

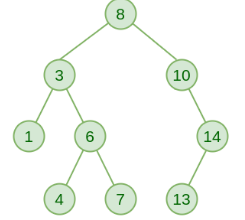
* Tìm kiếm nhị phân là thuật toán tìm kiếm hiệu quả trong mảng đã được sắp xếp. Thuật toán này chia đôi mảng, loại bỏ một nửa phần tử không cần thiết mỗi lần so sánh.
* Chỉ áp dụng với mảng đã sắp xếp



Hình 1.17

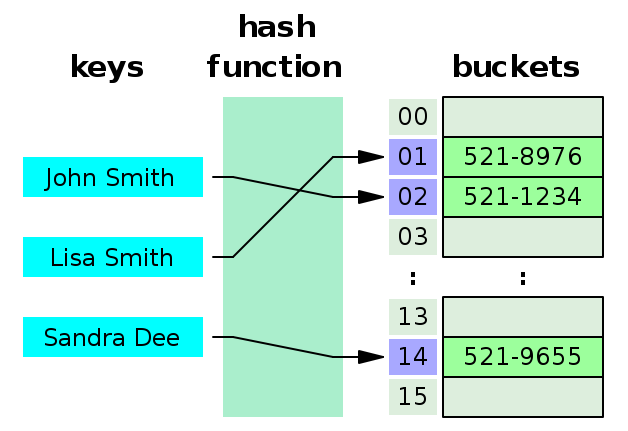
1.2.4.3 **TÌM KIẾM THEO CÂY NHỊ PHÂN TÌM KIẾM (BINARY SEARCH TREE - BST)**

* Tìm kiếm trong cây nhị phân tìm kiếm (BST), nơi mà giá trị con trái nhỏ hơn giá trị của nút gốc và giá trị con phải lớn hơn giá trị của nút gốc. Tìm kiếm sẽ đi qua các nút theo cách giảm dần kích thước của phạm vi tìm kiếm.



Hình 1.18

* + 1. **BẢNG BĂM (HASH TABLE)**
* Bảng băm là một cấu trúc dữ liệu cho phép lưu trữ và tìm kiếm các cặp khóa-giá trị một cách nhanh chóng. Thông qua một hàm băm (hash function), bảng băm ánh xạ khóa (key) vào một chỉ số (index) trong một mảng, từ đó giúp việc truy xuất giá trị (value) tương ứng trở nên rất nhanh.

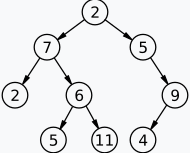


Hình 1.19

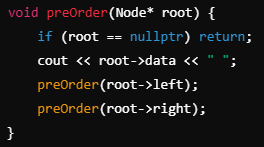
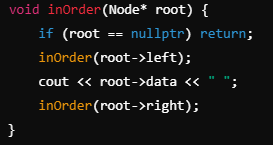
* Cấu trúc của bảng băm
* Mảng (Array): Bảng băm sử dụng một mảng để lưu trữ các phần tử.
* Khóa (Key): Mỗi phần tử trong bảng băm được xác định bởi một khóa duy nhất.
* Giá trị (Value): Mỗi khóa trong bảng băm sẽ được liên kết với một giá trị, gọi là giá trị của phần tử.
* Hàm băm (Hash Function): Hàm băm nhận vào một khóa và trả về một chỉ số trong mảng (index), nơi phần tử sẽ được lưu trữ.
* Nguyên lý hoạt động
* Khi thêm một phần tử vào bảng băm, hàm băm sẽ được sử dụng để xác định vị trí trong mảng nơi phần tử đó sẽ được lưu trữ. Để tìm kiếm hoặc lấy giá trị của một khóa, hàm băm cũng sẽ được sử dụng để tìm đúng vị trí trong mảng và lấy giá trị tương ứng.
* Hàm băm
* Hàm băm là bất kỳ hàm nào có thể được sử dụng để ánh xạ tập dữ liệu có kích thước tùy ý thành tập dữ liệu có kích thước cố định và đưa vào bảng băm. Các giá trị được trả về bởi hàm băm được gọi là giá trị băm.
  + 1. **CÂY NHỊ PHÂN (BINARY TREE)**
       1. **CÂY NHỊ PHÂN LÀ GÌ?**
* Cây nhị phân (Binary Tree) là một cấu trúc dữ liệu trong đó mỗi nút có tối đa hai con (con trái và con phải). Mỗi nút trong cây nhị phân chứa một giá trị và có thể có con trái và con phải (hoặc không có).
* Cấu trúc cây nhị phân được sử dụng rộng rãi trong các bài toán tìm kiếm, sắp xếp, lưu trữ dữ liệu, và nhiều ứng dụng khác.

1.2.6.2 **CÁC THÀNH PHẦN CỦA CÂY NHỊ PHÂN**

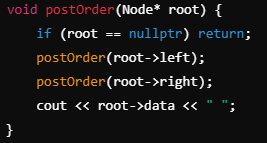
* Nút (Node): Mỗi phần tử trong cây nhị phân gọi là một nút, chứa một giá trị.
* Chồi (Child): Mỗi nút có thể có tối đa hai nút con (con trái và con phải).
* Nút gốc (Root): Nút đầu tiên của cây, là nút duy nhất không có nút cha.
* Nút lá (Leaf): Nút không có con.
* Nút cha (Parent): Nút có ít nhất một con.
* Cây con (Subtree): Một cây con là một cây con bắt đầu từ một nút bất kỳ trong cây ban đầu.

VD: Một cây nhị phân được gắn nhãn có kích thước là 9 và chiều cao là 3, với nút gốc có giá trị là 2. Cây trên không cân bằng và không được sắp xếp.

Hình 1.20

* + - 1. **CÁC LOẠI CÂY NHỊ PHÂN**
* Cây nhị phân đầy đủ (Full Binary Tree): Mỗi nút có thể có 0 hoặc 2 con.
* Cây nhị phân hoàn chỉnh (Complete Binary Tree): Mọi mức của cây đều đầy đủ trừ mức cuối cùng. Các nút ở mức cuối cùng phải được điền từ trái qua phải.
* Cây nhị phân cân bằng (Balanced Binary Tree): Chiều cao của cây con trái và cây con phải của mỗi nút không chênh lệch quá 1.
* Cây nhị phân tìm kiếm (Binary Search Tree - BST): Cây nhị phân có tính chất đặc biệt: mọi nút con trái có giá trị nhỏ hơn nút cha và mọi nút con phải có giá trị lớn hơn nút cha.
* Cây AVL (AVL Tree): Một loại cây nhị phân tìm kiếm tự cân bằng. Đảm bảo chiều cao của các cây con không chênh lệch quá 1.
  + - 1. **THUẬT TOÁN DUYỆT CÂY NHỊ PHÂN**
* Duyệt trước (Pre-order):
* Trước (Root), trái (Left), phải (Right).
* Thuật toán:
* Xử lý nút hiện tại.
* Duyệt cây con trái.
* Duyệt cây con phải. hình 1.21.a
* Duyệt giữa (In-order):
* Trái (Left), trước (Root), phải (Right).
* Thuật toán:
* Duyệt cây con trái.
* Xử lý nút hiện tại.
* Duyệt cây con phải.

Hình 1.21.b

* Duyệt sau (Post-order):
* Trái (Left), phải (Right), trước (Root).
* Thuật toán:
* Duyệt cây con trái.
* Duyệt cây con phải.
* Xử lý nút hiện tại. Hình 1.21.c
* + 1. **BỘ THƯ VIÊN CHUẨN**
* **bộ thư viện chuẩn (Standard Library)** cung cấp một tập hợp các hàm và lớp giúp lập trình viên giải quyết các vấn đề phổ biến trong lập trình như xử lý chuỗi, làm việc với dữ liệu đầu vào/đầu ra, thao tác với các cấu trúc dữ liệu như mảng, danh sách, bản đồ, v.v
* **#include <iostream>**
* Thư viện này cung cấp các đối tượng và hàm để thực hiện nhập xuất (I/O) trong C++. Nó hỗ trợ thao tác với các dòng dữ liệu (input/output) từ màn hình, file, hoặc các thiết bị khác.
* Các thành phần chính:

+ std::cin: Đối tượng dùng để nhập dữ liệu từ bàn phím.

+ std::cout: Đối tượng dùng để xuất dữ liệu ra màn hình.

+ std::cerr: Đối tượng dùng để xuất lỗi ra màn hình (stderr).

+ std::clog: Đối tượng dùng để xuất thông tin log ra màn hình (stderr).

* **#include <string>**
* Thư viện này cung cấp các lớp và hàm để thao tác với chuỗi (strings) trong C++. std::string là lớp chính để làm việc với chuỗi văn bản. Thư viện này hỗ trợ các phép toán chuỗi như nối chuỗi, tìm kiếm trong chuỗi, so sánh chuỗi, v.v.
* Các thành phần chính:

+ std::string: Lớp chính đại diện cho chuỗi trong C++.

+ Các hàm thành viên của std::string: size(), length(), substr(), find(), replace(), c\_str(), v.v.

* **#include <iomanip>**
* Thư viện iomanip (input/output manipulators) cung cấp các công cụ để điều chỉnh định dạng đầu vào và đầu ra, chẳng hạn như căn lề, làm tròn số, và định dạng hiển thị số thập phân.
* Các thành phần chính:

+ std::setw(n): Đặt chiều rộng của cột cho dữ liệu đầu ra.

+ std::setprecision(n): Đặt số chữ số thập phân khi hiển thị số thực.

+ std::fixed và std::scientific: Chế độ hiển thị số thực theo dạng số cố định (fixed-point) hoặc khoa học (scientific).

+ std::left, std::right, std::internal: Điều chỉnh căn lề của đầu ra.

* **#include <algorithm>**
* Thư viện algorithm cung cấp một loạt các thuật toán chuẩn cho việc thao tác với các cấu trúc dữ liệu như mảng, vector, danh sách, v.v. Các thuật toán bao gồm tìm kiếm, sắp xếp, xoá, thay đổi, v.v.
* Các thành phần chính:

+ std::sort(): Sắp xếp dãy phần tử (mảng, vector).

+ std::find(): Tìm kiếm một phần tử trong dãy.

+ std::reverse(): Đảo ngược dãy phần tử.

+ std::max() và std::min(): Tìm giá trị lớn nhất hoặc nhỏ nhất trong dãy.

+ std::swap(): Hoán đổi hai giá trị.

# CHƯƠNG 2. XÂY DỰNG THUẬT TOÁN

**2.1 GIỚI 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.

**2.1.1:CÁC THƯ VIỆN TRONG CHƯƠNG TRÌNH**

**<iostream>**:Thư viện này cung cấp đầu vào luồng và tiêu chuẩn đầu ra.

**<string>**:Hỗ trợ các thao tác với chuỗi, như phân bổ giá trị, chuỗi kết nối, lấy độ dài và các thao tác khác trên chuỗi.

**<iomanip>**:Thư viện này cung cấp các hàm và đối tượng để định dạng đầu dữ liệu, hữu ích khi hiển thị dạng số, ngày tháng hoặc khoảng trắng ở đầu ra.

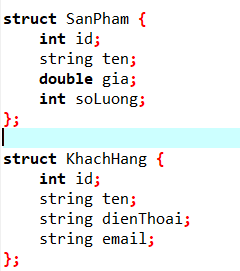
**<algorithm>:**thư viện chuẩn chứa các biến thuật toán phổ biến giúp thực thi các phép xử lý dữ liệu và thao tác trên vùng chứa

**<cctype>**:thư viện có tác dụng giúp bạn dễ dàng kiểm tra và thao tác với các ký tự trong C++. Cung cấp các ký tự xử lý chức năng, giúp chương trình dễ đọc, dễ bảo trì và giảm thiểu lỗi.

**<limits>:**là thư viện trong C++ cung cấp các giới hạn (giá trị lớn nhất và nhỏ nhất) của các nguyên dữ liệu kiểu (`int`, `long`, `float`, `double`, vv) trong ngôn ngữ. Thư viện này rất hữu ích khi bạn muốn biết khoảng giá trị mà một loại công cụ dữ liệu có thể biểu hiện hoặc khi cần thiết lập các giới hạn kiểm tra để tránh tràn số (tràn) hoặc suy giảm độ chính xác (tràn) .

**2.2:XÂY DỰNG CHƯƠNG TRÌNH.**

-Cấu trúc nội dung:



Id :mã số khách hàng/mã số sản phẩm

Ten:tên khách hàng/tên sản phẩm

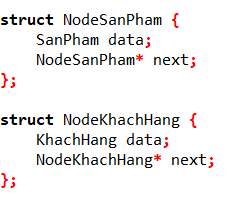
Gia:giá tiền sản phẩm

Soluong: số lượng sản phẩm

Dienthoai:số điện thoại khách hàng

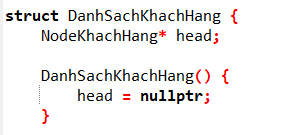
Email:email của khách hàng

**2.2.1:KHAI BÁO CẤU TRÚC NODE:**



Hình 2.1

**2.2.2:KHAI BÁO DANH SÁCH:**

****

Hình 2.2

**2.2.3:CÁC HÀM VOID:**

**1:**Trong danhsachkhachhang

**-void themKhachHang(KhachHang khachHang):** Thêm một khách hàng vào cuối danh sách khách hàng.

**-void hienThiKhachHang()**: Hiển thị danh sách tất cả các khách hàng. Nếu danh sách rỗng, in ra thông báo tương ứng.

**-void xoaKhachHang(int id):** Xóa khách hàng có id xác định. Nếu khách hàng không tồn tại, in ra thông báo.

**-void capNhatKhachHang(int id):** Cập nhật thông tin của khách hàng có id xác định (tên, số điện thoại và email).

**-void sapXepKhachHangTheoTen():** Sắp xếp danh sách khách hàng theo tên theo thứ tự bảng chữ cái.

**-void sapXepKhachHangTheoId():** Sắp xếp danh sách khách hàng theo id theo thứ tự tăng dần.

**2** Trong DanhSachSanPham

**-void themSanPham(SanPham sanPham):** Thêm một sản phẩm vào cuối danh sách sản phẩm.

**-void hienThiSanPham():** Hiển thị danh sách tất cả các sản phẩm. Nếu danh sách rỗng, in ra thông báo tương ứng.

**-void xoaSanPham(int id):** Xóa sản phẩm có id xác định. Nếu sản phẩm không tồn tại, in ra thông báo.

**-void capNhatSanPham(int id):** Cập nhật thông tin của sản phẩm có id xác định (tên, giá và số lượng).

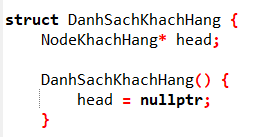
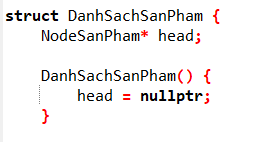
**-void sapXepSanPhamTheoGia():** Sắp xếp danh sách sản phẩm theo giá theo thứ tự tăng dần.

**-void sapXepSanPhamTheoTen():** Sắp xếp danh sách sản phẩm theo tên theo thứ tự bảng chữ cái.

**3**. Trong hàm menu

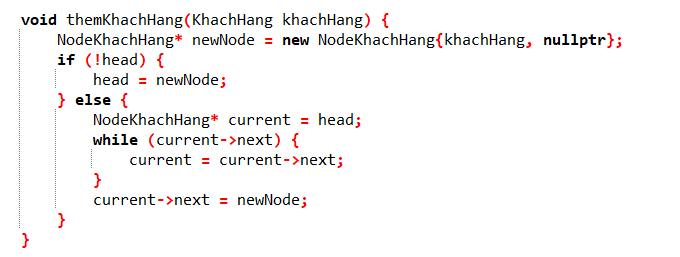
**-void menu(DanhSachKhachHang& dskh, DanhSachSanPham& dssp**): Hiển thị menu chính và xử lý các lựa chọn của người dùng. Menu cho phép người dùng thực hiện các chức năng như thêm, xóa, hiển thị, tìm kiếm, cập nhật, sắp xếp và tính tổng giá trị sản phẩm.rrr

**2.2.4 HÀM KHỞI TẠO:**



Hình 2.3.a hình 2.3.b

**2.2.5:HÀM THÊM KHÁCH HÀNG:**



Hình 2.3

***\*Giải thích :***

**1.NodeKhachHang\* newNode = new NodeKhachHang{khachHang, nullptr**};

Tạo một node mới (newNode) kiểu NodeKhachHang, trong đó:

-khachHang là thông tin khách hàng cần thêm, được truyền vào khi gọi hàm.

-nullptr là giá trị gán cho con trỏ next của node mới. Điều này chỉ ra rằng node này hiện tại không có node tiếp theo (vì đây sẽ là node cuối cùng nếu danh sách không rỗng).

**2. if (!head)**

-Kiểm tra xem danh sách khách hàng có rỗng không, tức là con trỏ head có trỏ đến nullptr không. Nếu danh sách rỗng (không có khách hàng nào), thì head chưa được gán giá trị.

**3. head = newNode;**

-Nếu danh sách rỗng, gán head bằng newNode, tức là node mới sẽ là khách hàng đầu tiên trong danh sách

**4. else**

-Nếu danh sách không rỗng, tức là đã có ít nhất một khách hàng trong danh sách.

**5. NodeKhachHang\* current = head;**

-Tạo con trỏ current để duyệt qua danh sách từ đầu (từ node đầu tiên head).

**6. while (current->next)**

-Duyệt qua từng node trong danh sách bằng cách di chuyển con trỏ current tới node tiếp theo cho đến khi current->next là nullptr. Điều này có nghĩa là con trỏ current sẽ dừng lại khi đến node cuối cùng trong danh sách.

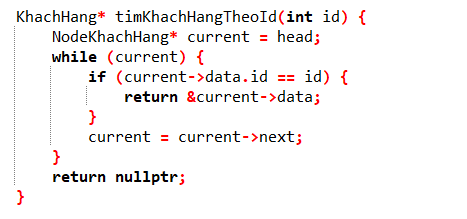
**7. current->next = newNode;**

-Khi tìm thấy node cuối cùng trong danh sách (khi current->next là nullptr), gán current->next bằng newNode, tức là nối node mới vào cuối danh sách.Giải thích chi tiết:

**=>Tóm lại:**

-Hàm themKhachHang: Thêm một khách hàng mới vào cuối danh sách. Nếu danh sách rỗng, node mới sẽ là node đầu tiên. Nếu danh sách không rỗng, node mới sẽ được thêm vào cuối danh sách.

**2.2.6:HÀM TÌM KIẾM KHÁCH HÀNG THEO ID:**

****

Hình 2.4

***\*Giải thích :***

**1. Khai báo con trỏ current: NodeKhachHang\* current = head;**

-Chúng ta khai báo một con trỏ current kiểu NodeKhachHang\* và gán nó trỏ đến head của danh sách. Con trỏ current sẽ giúp duyệt qua các nút trong danh sách liên kết.

**2. Vòng lặp while để duyệt qua danh sách: while (current)**

-Vòng lặp sẽ tiếp tục chạy miễn là current không phải là nullptr. Nếu current là nullptr, tức là chúng ta đã duyệt qua toàn bộ danh sách và không tìm thấy khách hàng có ID phù hợp.

**3. So sánh ID trong dữ liệu của nút với ID tìm kiếm: if (current->data.id == id)**

-Trong mỗi vòng lặp, chúng ta so sánh id của khách hàng trong nút hiện tại (current->data.id) với ID mà người dùng nhập vào (id).

-current->data: Đây là phần dữ liệu trong nút hiện tại (chứa thông tin của khách hàng).

Nếu ID của khách hàng trong nút hiện tại trùng khớp với ID tìm kiếm, nghĩa là chúng ta đã tìm thấy khách hàng mong muốn.

**4. Trả về con trỏ đến khách hàng tìm được: return &current->data;**

-Nếu tìm thấy khách hàng có ID trùng khớp, chúng ta trả về con trỏ tới phần dữ liệu của khách hàng đó (current->data).

-Lưu ý rằng current->data là đối tượng kiểu KhachHang, và khi sử dụng &, chúng ta trả về con trỏ đến đối tượng này.

**5. Di chuyển con trỏ đến nút tiếp theo: current = current->next**;

-Nếu chưa tìm thấy khách hàng có ID trùng khớp, chúng ta di chuyển con trỏ current đến nút tiếp theo trong danh sách (current->next), và tiếp tục kiểm tra với nút đó.

**6. Trả về nullptr nếu không tìm thấy khách hàng: return nullptr**

-Nếu vòng lặp kết thúc (tức là current trở thành nullptr), điều này có nghĩa là chúng ta đã duyệt hết danh sách mà không tìm thấy khách hàng có ID trùng khớp.

-Trong trường hợp này, hàm trả về nullptr, chỉ ra rằng không có khách hàng nào có ID tương ứng.

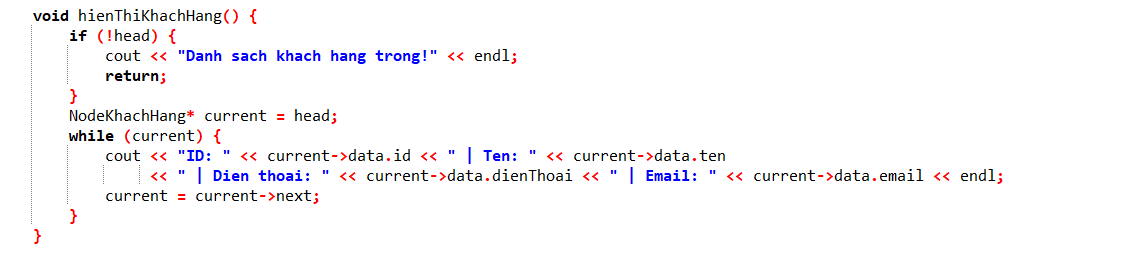
**=>Tóm lại:**

-Đầu vào: Hàm nhận vào một tham số id, đây là ID của khách hàng mà chúng ta muốn tìm.

-Đầu ra: Nếu tìm thấy khách hàng có ID trùng với id nhập vào, hàm trả về con trỏ tới đối tượng KhachHang chứa thông tin của khách hàng đó. Nếu không tìm thấy, hàm trả về nullptr.

-Hàm này sẽ hữu ích khi bạn cần tra cứu thông tin của một khách hàng dựa trên ID của họ trong danh sách.

**2.2.7:HÀM HIỂN THỊ KHÁCH HÀNG:**

****

Hình 2.5

***\*Giải thích :***

**1. Kiểm tra danh sách rỗng:**

**if (!head) {**

**cout << "Danh sach khach hang trong!" << endl;**

**return;**

**}**

-Đầu tiên, chúng ta kiểm tra xem danh sách khách hàng có trống hay không. Nếu head là nullptr, có nghĩa là danh sách không có khách hàng nào. Khi đó, hàm sẽ in ra thông báo "Danh sach khach hang trong!" và dừng thực hiện hàm bằng lệnh return.

**2. Khởi tạo con trỏ current: NodeKhachHang\* current = head;**

-Nếu danh sách không trống, chúng ta khởi tạo một con trỏ current trỏ đến head của danh sách. Con trỏ current sẽ dùng để duyệt qua các phần tử trong danh sách.

**3. Duyệt qua danh sách khách hàng: while (current)**

-Vòng lặp while sẽ tiếp tục chạy miễn là current không phải là nullptr. Mỗi lần vòng lặp chạy, current trỏ tới một nút trong danh sách.

**4. Hiển thị thông tin khách hàng tại nút hiện tại:**

**cout << "ID: " <<**

**current->data.id << " | Ten: "**

**<< current->data.ten**

**<< " | Dien thoai: " <<**

**current->data.dienThoai << " |**

**Email: " << current->data.email**

**<< endl;**

-Trong mỗi vòng lặp, hàm sẽ in ra thông tin của khách hàng tại nút hiện tại. Dữ liệu khách hàng có trong current->data, bao gồm:

-current->data.id: ID của khách hàng

-current->data.ten: Tên khách hàng

-current->data.dienThoai: Số điện thoại khách hàng

-current->data.email: Email khách hàng

-Dữ liệu được in ra theo định dạng "ID: ... | Ten: ... | Dien thoai: ... | Email: ...".

**5. Di chuyển con trỏ đến nút tiếp theo: current = current->next;**

Sau khi in thông tin của khách hàng tại nút hiện tại, con trỏ current được di chuyển tới nút tiếp theo trong danh sách bằng cách gán current = current->next.

**6. Kết thúc khi đã duyệt hết danh sách:**

-Khi current trở thành nullptr, vòng lặp sẽ kết thúc, và hàm hienThiKhachHang() sẽ kết thúc.

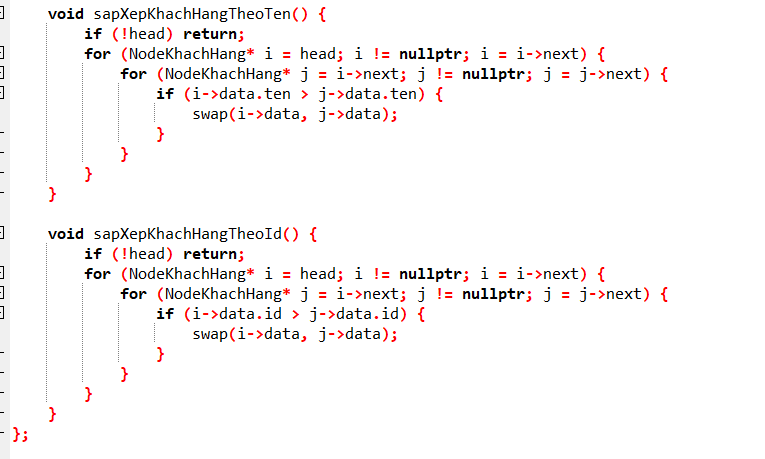
**=>Tóm lại:**

-Đầu vào: Hàm không nhận tham số đầu vào.

-Đầu ra: Hàm in ra danh sách thông tin các khách hàng (ID, tên, số điện thoại, email) nếu có. Nếu danh sách trống, hàm in ra thông báo rằng danh sách khách hàng trống.

-Mục đích: Duyệt qua danh sách khách hàng và in thông tin từng khách hàng.

**2.2.8:SẮP XẾP DANH SÁCH KHÁCH HÀNG:**

****

Hình 2.9

**1. Sắp xếp theo tên: sapXepKhachHangTheoTen()**

-Hàm này sử dụng thuật toán sắp xếp nổi bọt (Bubble Sort) để sắp xếp danh sách khách hàng theo tên (ten).

***\*Giải thích:***

-Vòng lặp đầu tiên **(for (NodeKhachHang\* i = head; i != nullptr; i = i->next))** duyệt qua từng nút trong danh sách khách hàng.

-Vòng lặp bên trong **(for (NodeKhachHang\* j = i->next; j != nullptr; j = j->next))** so sánh tên của nút i và j. Nếu tên của i lớn hơn tên của j (theo thứ tự chữ cái), chúng ta hoán đổi dữ liệu giữa i và j để sắp xếp theo thứ tự tăng dần.

=>Quá trình này tiếp tục cho đến khi danh sách được sắp xếp hoàn toàn.

**2. Sắp xếp khách hàng theo ID (sapXepKhachHangTheoId)**

-Hàm này cũng sử dụng thuật toán sắp xếp nổi bọt (Bubble Sort) nhưng sẽ sắp xếp khách hàng theo id thay vì theo tên.

***\*Giải thích***:

-Vòng lặp đầu tiên **(for (NodeKhachHang\* i = head; i != nullptr; i = i->next))** duyệt qua tất cả các nút trong danh sách khách hàng.

-Vòng lặp bên trong **(for (NodeKhachHang\* j = i->next; j != nullptr; j = j->next))** so sánh ID của i và j. Nếu ID của i lớn hơn ID của j, chúng ta hoán đổi dữ liệu giữa i và j để sắp xếp theo thứ tự tăng dần.

=>Quá trình này tiếp tục cho đến khi danh sách được sắp xếp theo ID.

**=>Tóm lại:**

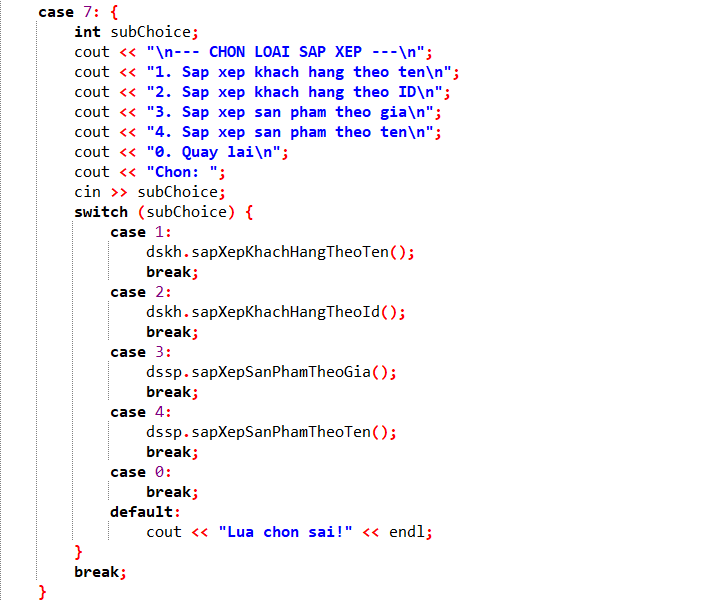
-Sắp xếp theo tên (sapXepKhachHangTheoTen): Sử dụng thuật toán Bubble Sort, so sánh và hoán đổi dữ liệu khách hàng theo tên.

-Sắp xếp theo ID (sapXepKhachHangTheoId): Tương tự như sắp xếp theo tên, nhưng so sánh và hoán đổi dữ liệu khách hàng theo ID.

**@Cách sử dụng trong menu:**

-Khi người dùng chọn sắp xếp khách hàng trong menu của chương trình, bạn có thể gọi các hàm này để sắp xếp danh sách khách hàng theo tên hoặc theo ID.

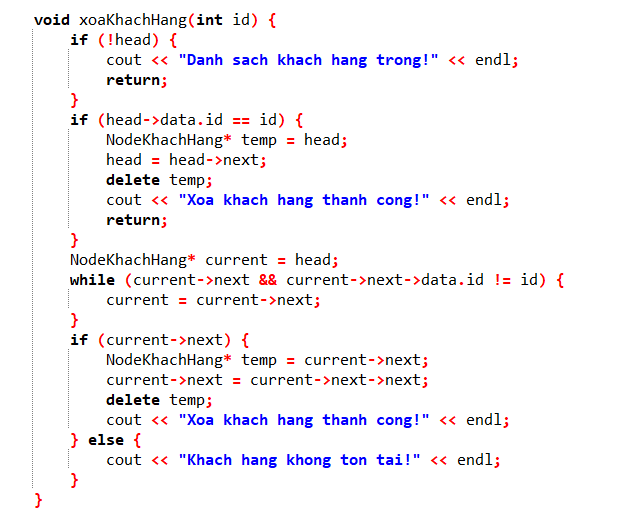
-Ví dụ trong menu của bạn:



Hình 2.10

-Trong đoạn mã trên, khi người dùng chọn số 1 hoặc 2, các hàm tương ứng **(sapXepKhachHangTheoTen hoặc sapXepKhachHangTheoId**) sẽ được gọi để sắp xếp danh sách khách hàng.

**2.2.9:XÓA KHÁCH HÀNG:**

****

Hình 2.11

***\*Giải Thích:***

**1. Kiểm tra danh sách trống:**

-Nếu head là nullptr, tức là danh sách rỗng, hàm sẽ thông báo rằng danh sách khách hàng trống và không làm gì thêm.

**2. Xóa khách hàng đầu tiên:**

-Nếu khách hàng đầu tiên trong danh sách có id khớp với id cần xóa, ta cập nhật head để trỏ đến nút kế tiếp và giải phóng bộ nhớ của nút cũ (bằng cách delete temp).

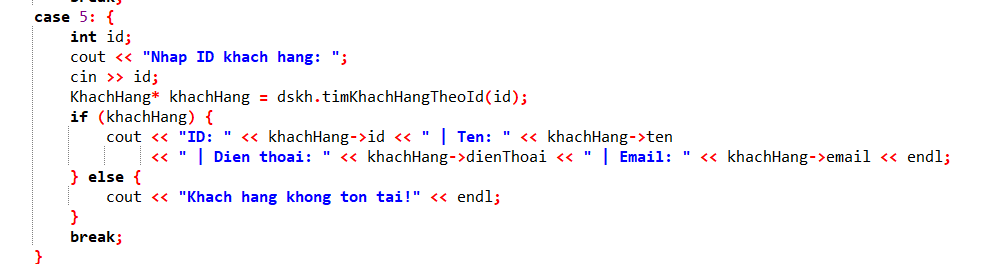
**3. Xóa khách hàng không phải đầu tiên:**

-Nếu khách hàng cần xóa không phải là phần tử đầu tiên, ta duyệt qua danh sách bằng vòng lặp. Khi tìm được nút có id trùng khớp, ta cập nhật con trỏ next của nút trước đó để bỏ qua nút cần xóa và giải phóng bộ nhớ của nó.

**4. Khách hàng không tìm thấy:**

-Nếu không tìm thấy khách hàng có id cần xóa, hàm sẽ thông báo rằng khách hàng đó không tồn tại trong danh sách.

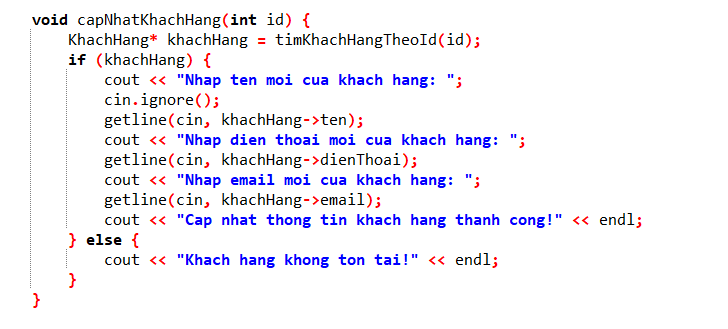
-Khi người dùng chọn xóa khách hàng từ menu, bạn có thể gọi hàm này như sau:



Hình 2.12

-Ở đây, sau khi người dùng nhập id, hàm xoaKhachHang(id) sẽ được gọi để xóa khách hàng tương ứng.

**2.2.10:CẬP NHẬT THÔNG TIN KHÁCH HÀNG:**



Hình 2.14

***\*Giải thích:***

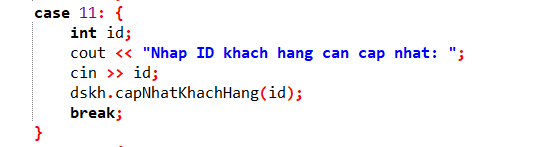
**1. Tìm khách hàng theo ID**: Hàm timKhachHangTheoId sẽ được gọi để tìm khách hàng có id tương ứng. Nếu tìm thấy, bạn sẽ nhận được con trỏ tới khách hàng đó.

**2. Nhập thông tin mới:** Sau khi tìm được khách hàng, chương trình yêu cầu người dùng nhập các thông tin mới (tên, điện thoại, email) và sử dụng getline() để lấy chuỗi nhập vào.

**3. Cập nhật thông tin:** Sau khi người dùng nhập thông tin mới, các trường thông tin của khách hàng sẽ được cập nhật.

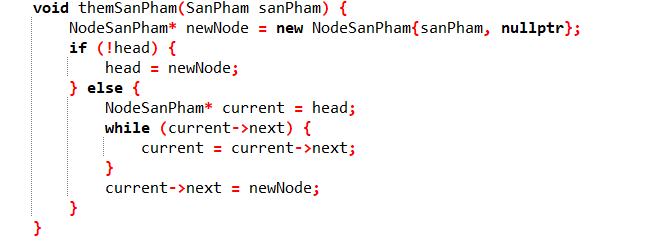
**4. Thông báo:** Chương trình sẽ thông báo rằng thông tin khách hàng đã được cập nhật thành công.

=>Gọi hàm trong menu:



Hình 2.15

**2.2.11:THÊM SẢN PHẨM:**

****

Hình 2.16

***\*Giải thích:***

**1. Tạo đối tượng sản phẩm mới:**

-Đầu tiên, chương trình yêu cầu người dùng nhập các thông tin về sản phẩm như ID, tên, giá và số lượng.

-Sau khi người dùng nhập các thông tin, một đối tượng SanPham mới (sanPhamMoi) sẽ được tạo và chứa thông tin mà người dùng đã nhập.

**2. Tạo một nút mới (NodeSanPham):**

-Một nút mới (nodeMoi) được tạo ra, với thông tin sản phẩm vừa nhập. Nút này có trường sanPham chứa sản phẩm và trường next được gán giá trị nullptr để đánh dấu đây là nút cuối cùng trong danh sách.

**3. Thêm vào danh sách liên kết:**

-Nếu danh sách liên kết (dau) hiện tại là rỗng (nghĩa là không có sản phẩm nào trong danh sách), thì dau sẽ trỏ đến nút mới.

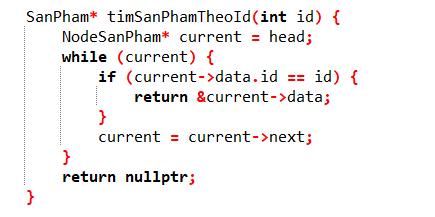
-Nếu danh sách không rỗng, hệ thống sẽ duyệt qua danh sách và thêm nút mới vào cuối danh sách.

**4. Thông báo cho người dùng:**

-Sau khi thêm sản phẩm thành công vào danh sách, hệ thống sẽ in ra thông báo "Sản phẩm đã được thêm vào danh sách!".

**=>Tóm lại:**Hàm **themSanPham()** giúp người dùng thêm một sản phẩm mới vào danh sách sản phẩm, đồng thời đảm bảo rằng sản phẩm sẽ được thêm vào cuối danh sách liên kết.

**2.2.12:TÌM KIẾM SẢN PHẨM THEO ID:**

****

Hình 2.17

***\*Giải thích :***

**1. Khởi tạo biến con trỏ current:**current được khởi tạo với giá trị là head, tức là bắt đầu từ đầu danh sách sản phẩm.

**2. Vòng lặp while (current):**

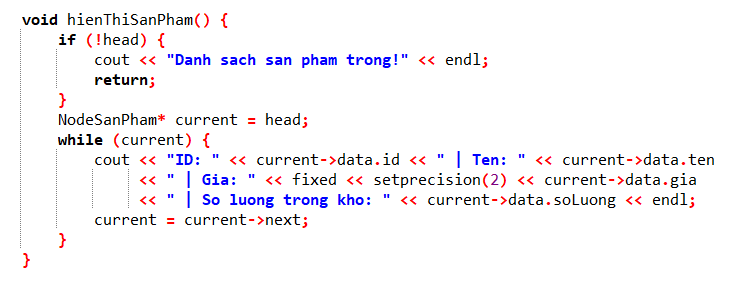
-Vòng lặp sẽ tiếp tục chạy miễn là current không phải là nullptr (tức là còn sản phẩm trong danh sách).

-Trong mỗi vòng lặp, hàm kiểm tra xem ID của sản phẩm tại nút current có khớp với id người dùng nhập vào không. Nếu có, nó trả về con trỏ đến sản phẩm đó.

**3. Trả về nullptr nếu không tìm thấy:**Nếu không tìm thấy sản phẩm nào có ID khớp sau khi duyệt hết danh sách, hàm trả về nullptr, chỉ ra rằng sản phẩm không tồn tại trong danh sách.

**=>Tóm tắt:**Hàm timSanPhamTheoId() giúp tìm một sản phẩm trong danh sách theo ID. Nếu tìm thấy, nó trả về con trỏ tới sản phẩm; nếu không tìm thấy, trả về nullptr.

**2.2.13:HÀM HIỂN THỊ SẢN PHẨM:**

****

Hình 2.18

***\*Giải thích :***

**1. Kiểm tra danh sách rỗng:**

**if (!head) {**

**cout << "Danh sach san pham trong!" << endl;**

**return;**

**}**

-Hàm đầu tiên kiểm tra xem danh sách sản phẩm có rỗng không bằng cách kiểm tra con trỏ head. Nếu head là nullptr, tức là danh sách không có sản phẩm, nó sẽ in thông báo "Danh sach san pham trong!" và thoát khỏi hàm.

**2. Duyệt qua danh sách sản phẩm:**

**NodeSanPham\* current = head;**

**while (current) {**

**cout << "ID: " << current->data.id << " | Ten: " << current->data.ten**

**<< " | Gia: " << fixed << setprecision(2) << current->data.gia**

**<< " | So luong trong kho: " << current->data.soLuong << endl;**

**current = current->next;**

**}**

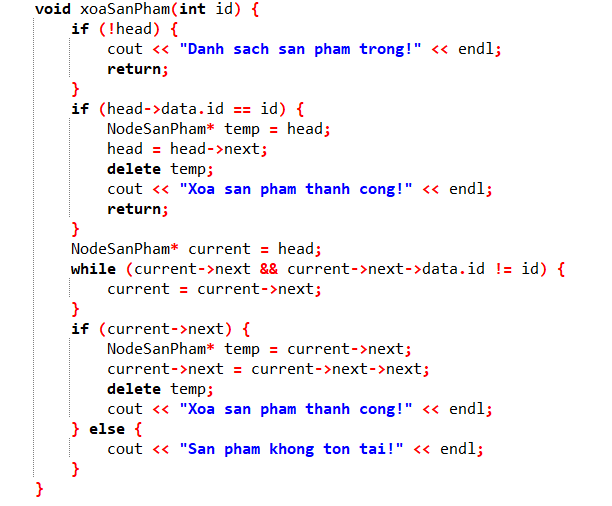
-Nếu danh sách không rỗng, hàm khởi tạo con trỏ current trỏ đến head và bắt đầu duyệt qua danh sách. Trong mỗi vòng lặp, hàm sẽ in thông tin của sản phẩm tại nút hiện tại.

-Dòng fixed << setprecision(2) giúp in giá sản phẩm với 2 chữ số thập phân.

-Cuối cùng, con trỏ current được cập nhật để trỏ đến nút tiếp theo (current = current->next).

**=>Tóm lại:**Hàm hienThiSanPham() duyệt qua danh sách sản phẩm và hiển thị thông tin của từng sản phẩm như ID, tên, giá (với 2 chữ số thập phân), và số lượng trong kho. Nếu danh sách rỗng, nó sẽ thông báo rằng danh sách sản phẩm trống.

**2.2.14:XÓA SẢN PHẨM:**

****

Hình 2.19

***\*Giải thích:***

**1. Kiểm tra danh sách rỗng:**

**if (!head) {**

**cout << "Danh sach san pham trong!" << endl;**

**return;**

**}**

-Nếu danh sách sản phẩm là rỗng (head == nullptr), hàm sẽ in thông báo "Danh sach san pham trong!" và thoát khỏi hàm.

**2. Xóa sản phẩm đầu tiên:**

**if (head->data.id == id) {**

**NodeSanPham\* temp = head;**

**head = head->next;**

**delete temp;**

**cout << "Xoa san pham thanh cong!" << endl;**

**return;**

**}**

-Nếu sản phẩm cần xóa là sản phẩm đầu tiên trong danh sách (head->data.id == id), hàm sẽ cập nhật head để trỏ đến nút tiếp theo của danh sách (head = head->next). Sau đó, bộ nhớ của nút đầu tiên sẽ được giải phóng với delete.

**3. Duyệt qua danh sách để tìm sản phẩm cần xóa:**

**NodeSanPham\* current = head;**

**while (current->next && current->next->data.id != id) {**

**current = current->next;**

**}**

**-**Nếu sản phẩm cần xóa không phải là sản phẩm đầu tiên, hàm sẽ duyệt qua các nút tiếp theo trong danh sách để tìm sản phẩm có ID khớp với ID cần xóa.

**4. Xóa sản phẩm tìm thấy:**

**if (current->next) {**

**NodeSanPham\* temp = current->next;**

**current->next = current->next->next;**

**delete temp;**

**cout << "Xoa san pham thanh cong!" << endl;**

**} else {**

**cout << "San pham khong ton tai!" << endl;**

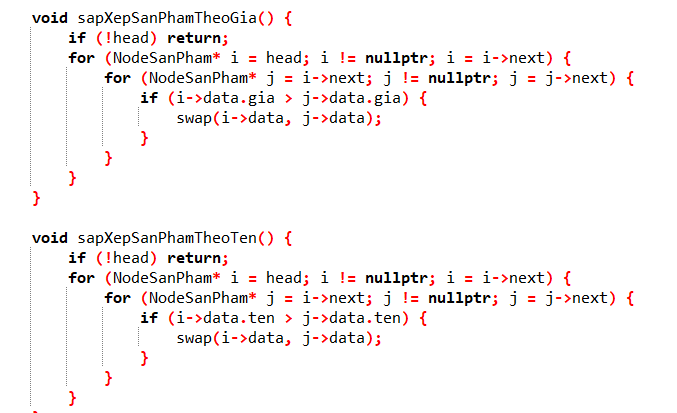
**}**

-Nếu sản phẩm được tìm thấy (tức là current->next không phải là nullptr và có ID khớp), hàm sẽ thực hiện việc xóa sản phẩm bằng cách thay đổi liên kết của nút trước đó (current->next = current->next->next) và giải phóng bộ nhớ của nút bị xóa (delete temp).

-Nếu không tìm thấy sản phẩm với ID tương ứng, hàm sẽ thông báo rằng sản phẩm không tồn tại.

**=>Tóm tắt:**Hàm xoaSanPham() tìm và xóa sản phẩm trong danh sách sản phẩm dựa trên ID. Nó xử lý cả trường hợp danh sách rỗng, sản phẩm cần xóa là sản phẩm đầu tiên, và sản phẩm cần xóa nằm ở giữa danh sách. Sau khi xóa, nó sẽ thông báo thành công hoặc lỗi nếu không tìm thấy sản phẩm.

**2.2.15:SẮP XẾP SẢN PHẨM:**

****

# Hình 2.20

**1. Sắp xếp sản phẩm theo giá:**

***\*Giải thích***

**1.1 Kiểm tra danh sách rỗng hoặc chỉ có một phần tử:**

**if (!head || !head->next) {**

**cout << "Danh sach san pham khong du de sap xep!" << endl;**

**return;**

**}**

-Nếu danh sách sản phẩm rỗng hoặc chỉ có một sản phẩm, không cần phải sắp xếp, vì vậy chúng ta in ra thông báo và kết thúc hàm.

**1.2. Sắp xếp bằng Bubble Sort:**

**for (NodeSanPham\* i = head; i != nullptr; i = i->next) {**

**for (NodeSanPham\* j = i->next; j != nullptr; j = j->next) {**

**if (i->data.price > j->data.price) {**

**SanPham temp = i->data;**

**i->data = j->data;**

**j->data = temp;**

**}**

**}**

**}**

-Thuật toán Bubble Sort hoạt động bằng cách lặp qua từng cặp sản phẩm (i và j), so sánh giá của chúng **(i->data.price > j->data.price)** và hoán đổi chúng nếu giá trị của sản phẩm ở i lớn hơn ở j.

-Vòng lặp bên **ngoài (for (NodeSanPham\* i = head; i != nullptr; i = i->next))** đi qua tất cả các sản phẩm trong danh sách, và vòng lặp bên trong (for **(NodeSanPham\* j =>next; j != nullptr; j = j->next))** so sánh mỗi sản phẩm với các sản phẩm tiếp theo trong danh sách.

**1.3. Thông báo sau khi sắp xếp: cout << "Sap xep san pham theo gia thanh cong!" << endl**;

-Sau khi sắp xếp xong, hàm sẽ thông báo rằng quá trình sắp xếp đã thành công.

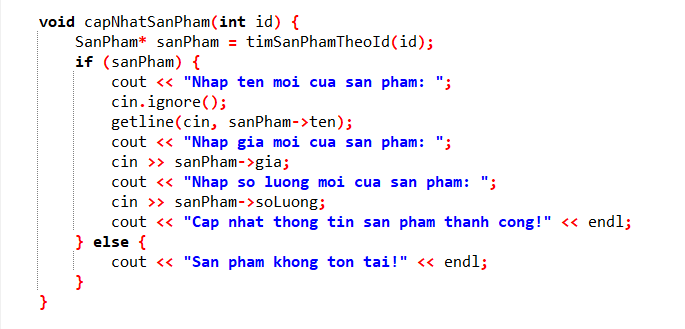
**2. Sắp xếp sản phẩm theo tên:**

**-Giải thích:**Thay vì so sánh price, chúng ta so sánh name của sản phẩm (i->data.name > j->data.name). Nếu tên của sản phẩm ở nút i lớn hơn tên của sản phẩm ở nút j, chúng ta hoán đổi chúng.

**-Kết quả:**Sau khi gọi hàm sapXepTheoGia() hoặc sapXepTheoTen(), danh sách sản phẩm sẽ được sắp xếp theo giá hoặc tên.Nếu bạn muốn kiểm tra kết quả, có thể in ra danh sách sản phẩm sau khi sắp xếp để xác nhận.

**-Lưu ý:**Thuật toán Bubble Sort có độ phức tạp là O(n²), vì vậy nếu danh sách của bạn có rất nhiều sản phẩm, thuật toán này có thể không phải là lựa chọn tối ưu. Nếu bạn cần tối ưu hóa, có thể sử dụng các thuật toán sắp xếp khác như Quick Sort hoặc Merge Sort.

### **2.2.16:CẬP NHẬT THÔNG TIN SẢN PHẨM:**

****

Hình 2.21

***\*Giải thích :***

**1. Tìm kiếm sản phẩm theo id:**

-Chúng ta sẽ duyệt qua danh sách sản phẩm (head), kiểm tra mỗi sản phẩm xem có id trùng với id được cung cấp trong tham số hay không.

-Nếu tìm thấy sản phẩm có id trùng khớp, ta cập nhật các thông tin của sản phẩm đó (tên, giá, số lượng).

**2. Cập nhật thông tin:**

-Cập nhật tên sản phẩm bằng **temp->data.name = ten**.

-Cập nhật giá sản phẩm bằng **temp->data.price = gia.**

-Cập nhật số lượng sản phẩm bằng **temp->data.quantity = soLuong.**

**3. Thông báo kết quả:**

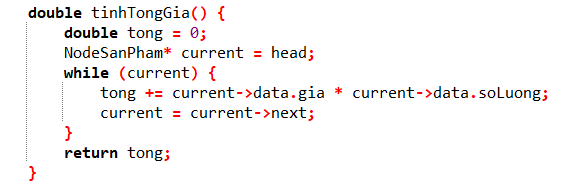
-Nếu tìm thấy sản phẩm và cập nhật thành công, hàm in ra thông báo "Cập nhật sản phẩm thành công".

-Nếu không tìm thấy sản phẩm với id được yêu cầu, hàm in ra thông báo "Sản phẩm không tồn tại".

**-Lưu ý:**Nếu danh sách sản phẩm có nhiều sản phẩm với cùng một id (dù là trường hợp bất thường), hàm này chỉ cập nhật sản phẩm đầu tiên tìm thấy. Bạn có thể thêm logic để xử lý trường hợp nhiều sản phẩm có cùng id, nếu cần.

Hãy đảm bảo rằng id của sản phẩm là duy nhất để tránh việc cập nhật sai sản phẩm.

### **2.2.17:TÍNH TỔNG GIÁ:**



Hình 2.22

***\*Giải thích:***

**1. double tong = 0;:**

-Đây là biến dùng để lưu trữ tổng giá trị của tất cả các sản phẩm. Ban đầu, giá trị của tong được khởi tạo là 0 (tức là chưa có sản phẩm nào tính vào).

**2. NodeSanPham\* current = head;:**

-current là một con trỏ dùng để duyệt qua danh sách các sản phẩm. Nó được gán bằng head (đầu danh sách), nghĩa là bắt đầu từ sản phẩm đầu tiên trong danh sách.

**3. while (current):**

Vòng lặp này sẽ tiếp tục chạy cho đến khi current trở thành nullptr, tức là khi hết danh sách sản phẩm (trường hợp danh sách trống hoặc đã duyệt hết các sản phẩm).

**4. tong += current->data.gia \* current->data.soLuong;:**

-Tại mỗi vòng lặp, ta sẽ tính giá trị của sản phẩm hiện tại bằng cách nhân giá (gia) của sản phẩm với số lượng (soLuong) của sản phẩm đó.Kết quả này sẽ được cộng vào tổng (tong), tức là tổng giá trị của tất cả các sản phẩm đã duyệt qua

**5. current = current->next;:**

-Sau khi tính toán giá trị của sản phẩm hiện tại, con trỏ current sẽ được di chuyển đến sản phẩm tiếp theo trong danh sách, thông qua trường next của NodeSanPham.

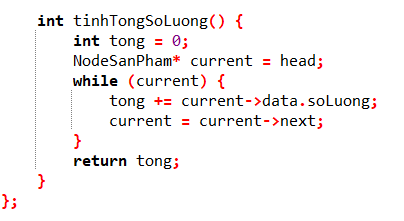
-Nếu còn sản phẩm tiếp theo, vòng lặp sẽ tiếp tục; nếu không, vòng lặp sẽ dừng

**6. return tong;**

Sau khi đã duyệt qua tất cả các sản phẩm trong danh sách và cộng dồn giá trị, hàm sẽ trả về giá trị của tong, tức là tổng giá trị của tất cả các sản phẩm trong danh sách.

**=>Tóm tắt**:Hàm này tính tổng giá trị của tất cả các sản phẩm trong kho bằng cách duyệt qua danh sách sản phẩm và tính giá trị của từng sản phẩm (giá x số lượng).Cuối cùng, tổng giá trị của tất cả các sản phẩm sẽ được trả về.

### **2.2.18:TÍNH TỔNG SỐ LƯỢNG:**

****

Hình 2.23

***\*Giải thích:***

**1. int tong = 0;:**

**-**Biến tong được sử dụng để lưu tổng số lượng của tất cả các sản phẩm trong danh sách. Ban đầu, tong được khởi tạo là 0 (chưa có sản phẩm nào tính vào).

**2. NodeSanPham\* current = head;:**

**-**current là con trỏ dùng để duyệt qua danh sách các sản phẩm. Nó được khởi tạo bằng head (đầu danh sách), nghĩa là bắt đầu từ sản phẩm đầu tiên trong danh sách.

**3. while (current):**

-Vòng lặp này sẽ tiếp tục chạy miễn là current không phải là nullptr, tức là danh sách chưa hết. Khi current là nullptr, có nghĩa là đã duyệt hết các sản phẩm trong danh sách.

**4. tong += current->data.soLuong;:**

**-**Trong mỗi vòng lặp, ta cộng số lượng của sản phẩm hiện tại vào biến tong. current->data.soLuong là số lượng của sản phẩm mà current đang trỏ tới.

-Dòng mã này làm việc như sau: với mỗi sản phẩm trong danh sách, ta cộng số lượng của sản phẩm đó vào tổng số lượng.

**5. current = current->next;:**

-Sau khi tính toán số lượng cho sản phẩm hiện tại, con trỏ current được di chuyển tới sản phẩm tiếp theo trong danh sách, thông qua trường next của NodeSanPham.

-Nếu có sản phẩm tiếp theo, vòng lặp sẽ tiếp tục; nếu không (danh sách đã duyệt hết), vòng lặp sẽ kết thúc.

**6. return tong;:**

**-**Sau khi đã duyệt qua tất cả các sản phẩm trong danh sách và cộng dồn số lượng, hàm sẽ trả về giá trị của tong, tức là tổng số lượng của tất cả các sản phẩm trong kho.

**=>Tóm lại:**

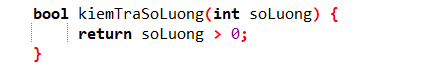
-Hàm này tính tổng số lượng của tất cả các sản phẩm trong danh sách.

-Nó duyệt qua tất cả các sản phẩm, cộng số lượng của từng sản phẩm vào tổng số lượng.

-Cuối cùng, hàm trả về tổng số lượng của các sản phẩm trong danh sách.

### **2.2.19:HÀM KIỂM TRA VÀ NHẬP SỐ LƯỢNG:**

1. Hàm kiemTraSoLuong:



Hình 2.24.a

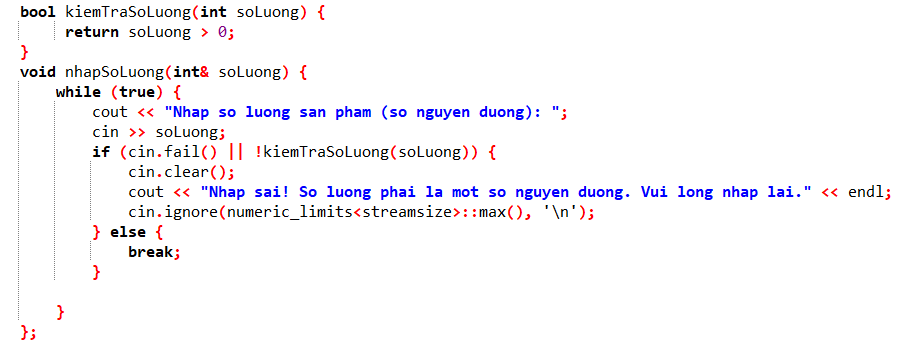
***\*Giải thích:***

-Hàm này kiểm tra xem số lượng soLuong có hợp lệ không, cụ thể là có lớn hơn 0 hay không.

-Nếu soLuong lớn hơn 0, hàm trả về true, ngược lại trả về false.

-Hàm này được dùng để đảm bảo rằng giá trị của soLuong là một số nguyên dương.

2.Hàm nhapSoLuong:



Hình 2.25

***\*Giải thích:***

**1.Yêu cầu người dùng nhập soLuong.**

**2.Nếu người dùng nhập sai (do nhập không phải số hoặc giá trị không hợp lệ):**

+**cin.fail()** sẽ trả về true nếu người dùng nhập không phải số.

+!**kiemTraSoLuong(soLuong)** sẽ trả về true nếu soLuong không phải số nguyên dương.

**3.Nếu có lỗi**:

+**cin.clear()** để xóa trạng thái lỗi của cin.

+**cin.ignore(numeric\_limits<streamsize>::max(), '\n')** để bỏ qua các ký tự sai còn lại trong dòng nhập.

+Hiển thị thông báo yêu cầu người dùng nhập lại.

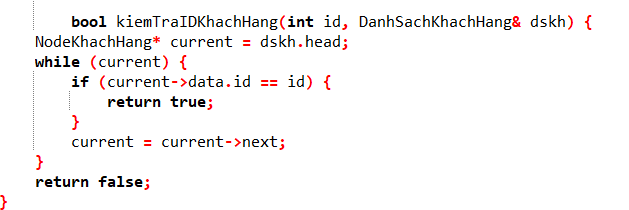
**4.Nếu không có lỗi, thoát khỏi vòng lặp while và chấp nhận giá trị của soLuong**.

**=>Tóm lại:**

**-kiemTraSoLuong** giúp kiểm tra soLuong có hợp lệ hay không (phải lớn hơn 0).

**-nhapSoLuong** sử dụng kiemTraSoLuong để đảm bảo rằng giá trị nhập vào hợp lệ và yêu cầu nhập lại nếu cần thiết.

### **2.2.20:KIỂM TRA ID KHÁCH HÀNG:**



Hình 2.26

***\*Giải thích:***

**1. Tham số và Giá trị Trả về:**

-Tham số: int id - ID của khách hàng mà bạn muốn kiểm tra trong danh sách.

-Giá trị trả về: bool - true nếu tìm thấy ID trong danh sách; false nếu không tìm thấy.

**2. Các biến và cấu trúc dữ liệu:NodeKhachHang\* current = head;:**

-current là con trỏ để duyệt qua các nút trong danh sách.

-head là con trỏ trỏ đến đầu danh sách liên kết (có thể là biến toàn cục hoặc thuộc tính của lớp quản lý danh sách khách hàng).

**3. Vòng lặp while:**

**-while (current)** lặp lại cho đến khi current là nullptr, tức là đã duyệt hết danh sách.

-Trong mỗi lần lặp:

+Kiểm tra **if (current->data.id == id)** để xem ID của nút hiện tại (current->data.id) có trùng với id cần tìm không.

+Nếu tìm thấy ID phù hợp, trả về true ngay lập tức, kết thúc hàm.

+Nếu ID không khớp, current = current->next chuyển đến nút kế tiếp trong danh sách.

**4. Kết thúc vòng lặp:**

-Nếu hàm duyệt hết danh sách mà không tìm thấy ID, thì thoát khỏi vòng lặp while và trả về false.

**=>Tóm lại:**

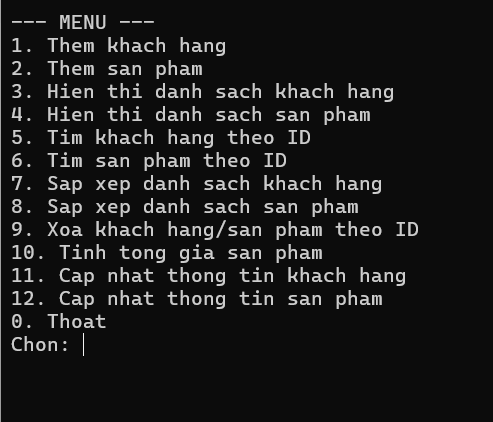
-Hàm kiemTraIDKhachHang sẽ duyệt qua từng nút trong danh sách liên kết của khách hàng và kiểm tra từng ID:

-Nếu tìm thấy ID trùng khớp với id cần kiểm tra, hàm trả về true.

-Nếu không tìm thấy ID nào phù hợp, hàm trả về false.

-Hàm này rất hữu ích để đảm bảo tính duy nhất của ID khi thêm khách hàng mới vào danh sách.

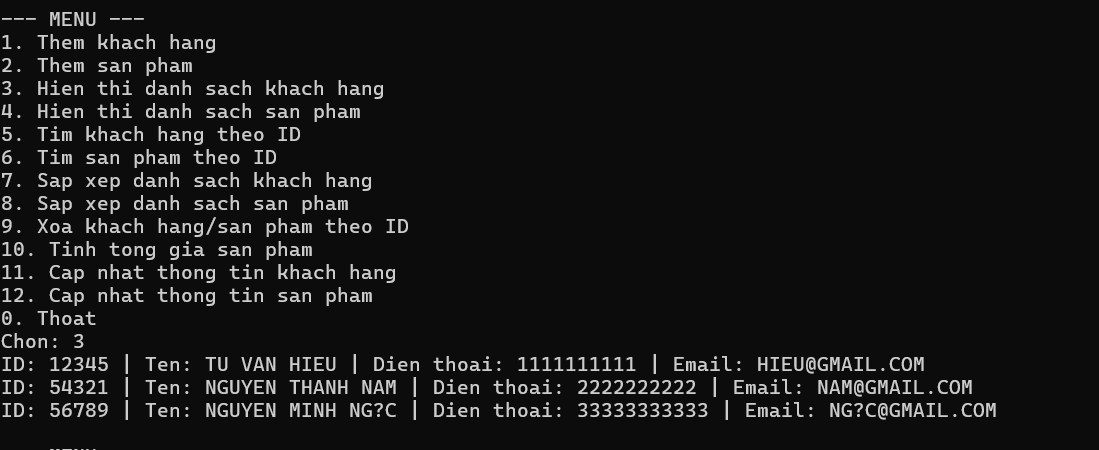
## **2.3:GIAO DIỆN CHƯƠNG TRÌNH:**



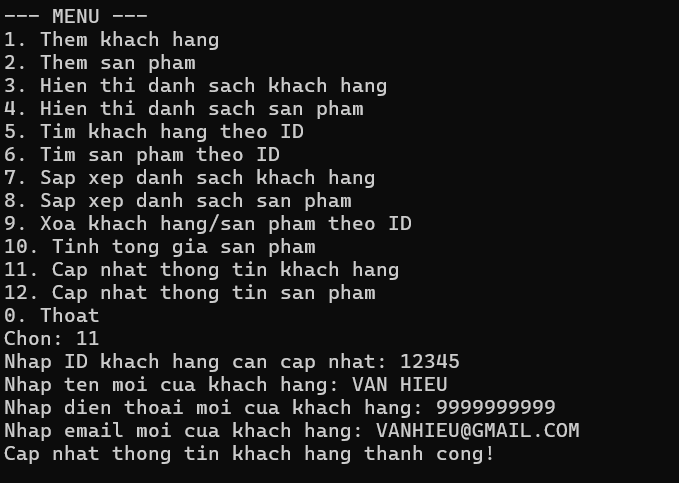
Hình 2.27: kết quả sau khi chạy

# CHƯƠNG 3:KẾT QUẢ CHƯƠNG TRÌNH

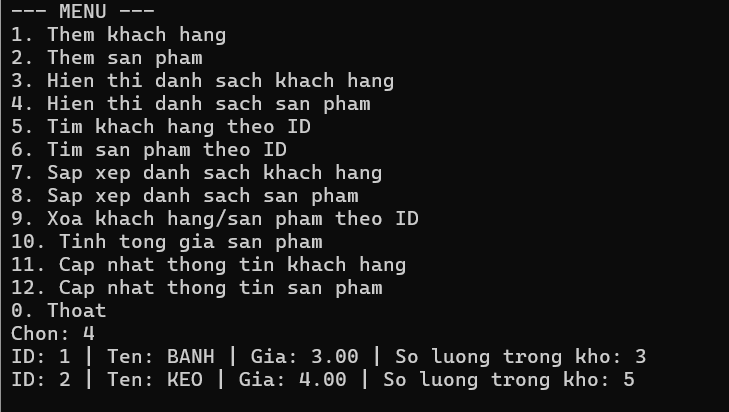
-Sau quá trình lên ý tưởng và xây dựng. Chương trình đã được hoàn thành với các chức năng cơ bản của một chương trình quản lí. Dưới đây là một số hình ảnh thu được khi chạy thử chương trình:

****

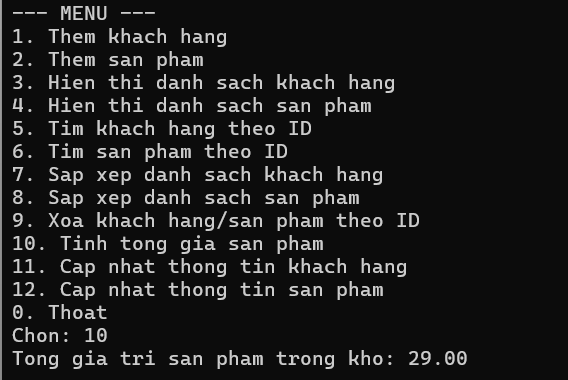
Hình 3.1:Danh sách khách hàng

****

Hình 3.2:cập nhập thông tin khách



Hình 3.3:cập nhập sản phẩm



Hình 3.4:tính tổng sản phẩm trong kho

**TÀI LIỆU THAM KHẢO**

1. [**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)**.**
2. [**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)**.**
3. [**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)
4. Cấu trúc dữ liệu và giải thuật – Nguyễn Trung Trực.
5. [**https://techacademy.edu.vn/danh-sach-lien-ket-don-c/**](https://techacademy.edu.vn/danh-sach-lien-ket-don-c/)
6. [**https://www.icantech.vn/kham-pha/danh-sach-lien-ket-don**](https://www.icantech.vn/kham-pha/danh-sach-lien-ket-don)