**TRƯỜNG ĐẠI HỌC GIAO THÔNG VẬN TẢI**

**PHÂN HIỆU TẠI TP. HỒ CHÍ MINH**

**BỘ MÔN CÔNG NGHỆ THÔNG TIN**



**BÁO CÁO MÔN HỌC**

**LẬP TRÌNH NÂNG CAO**

**ĐỀ TÀI : XÂY DỰNG PHẦN MỀM QUẢN LÍ GIỎ HÀNG**

Sinh viên thực hiện:

NGUYỄN HOÀNG PHỤNG – MSSV 6051071095

NGÔ TÙNG HIỀN – MSSV 60510710

Lớp : CÔNG NGHỆ THÔNG TIN K60

Khóa : K60

Tp. Hồ Chí Minh năm 2020

**MỤC LỤC**

Contents

[**CHƯƠNG 1 TỔNG QUAN VỀ ĐỀ TÀI** 1](#_Toc43756732)

[1.1Lý do chọn đề tài 1](#_Toc43756733)

[1.3 Khảo sát thực tế 1](#_Toc43756734)

[1.4 Cách tiếp cận, phương pháp nghiên cứu 2](#_Toc43756735)

[**Chương 2 CƠ SỞ LÝ THUYẾT** 3](#_Toc43756736)

[2.1 Đọc ghi file trong C 3](#_Toc43756737)

[2.1.1 Các kiểu file 3](#_Toc43756738)

[2.1.2 Các thao tác với file 4](#_Toc43756739)

[2.1.3 Thao tác với file trên ngôn ngữ C 4](#_Toc43756740)

[2.1.6 Một số ví dụ về đọc ghi file trong C 11](#_Toc43756741)

[2.2 Lý thuyết về cấu trúc danh sách liên kết 13](#_Toc43756742)

[2.3Danh sách liên kết là gì? 14](#_Toc43756743)

[2.4 Cài đặt danh sách liên kết đơn 15](#_Toc43756744)

[2.4.1 Khai báo linked list 15](#_Toc43756745)

[2.4.2 Tạo mới 1 Node 16](#_Toc43756746)

[2.4.3 Thêm Node vào danh sách liên kết 16](#_Toc43756747)

[2.4.4 Xóa Node khỏi danh sách liên kết 20](#_Toc43756748)

[2.4.5 Lấy giá trị ở vị trí bất kỳ 22](#_Toc43756749)

[2.4.6 Tìm kiếm trong danh sách liên kết 23](#_Toc43756750)

[2.4.7 Duyệt danh sách liên kết 24](#_Toc43756751)

[3.1 Các thuật toán sắp xếp và tìm kiếm 24](#_Toc43756752)

[3.1.1 Các thuật toán tìm kiếm : 24](#_Toc43756753)

[3.1.2 Các thuật toán sắp xếp : 29](#_Toc43756754)

[**Phần 3: DEMO CODE ĐỀ TÀI XÂY DỰNG PHẦN MỀM** 36](#_Toc43756755)

[**QUẢN LÍ GIỎ HÀNG VÀ HƯỚNG DẪN SỬ DỤNG** 36](#_Toc43756756)

[3.1 Các hàm trong chương trình 36](#_Toc43756757)

[3.2 Hướng dẫn sử dụng 36](#_Toc43756758)

**LỜI CẢM ƠN**

Sau quá trình học tập và rèn luyện tại bộ môn Công nghệ thông tin trường Đại học

Giao thông Vận tải – Phân hiệu tại thành phố Hồ Chí Minh em đã được trang bị các kiến

thức cơ bản, các kỹ năng thực tế để có thể hoàn thành đề tài của mình.

Em xin gửi lời cảm ơn chân thành đến quý thầy, cô bộ môn Công nghệ thông tin trường

Đại học Giao thông Vận tải – Phân hiệu tại thành phố Hồ Chí Minh đã quan tâm hướng

dẫn truyền đạt học những kiến thức và kinh nghiệm cho em trong suốt thời gian học tập,

và thực hiện đồ án tốt nghiệp một cách tận tình và tâm huyết. Em xin chúc quý thầy cô

thật nhiều sức khỏe và luôn đạt được thành công trong cuộc sống. Đặc biệt em xin cảm

ơn cô Trần Thị Dung người đã trực tiếp hướng dẫn và chỉ bảo em trong quá trình đề tài

báo cáo môn học .Cô đã cùng em góp ý và xây dựng đề tài “XÂY DỰNG PHẦN MỀM

QUẢN LÍ GIỎ HÀNG.”

Sau một thời gian nỗ lực thực hiện thì đề tài cũng đã hoàn thành. Nhưng không sao tránh

khỏi những sai sót do em còn chưa có nhiều kinh nghiệm thực tế. Em kính mong nhận

được sự góp ý và nhận xét từ quý thầy, cô để em có thể hoàn thiện và hoàn thành tốt hơn

cho đề tài của mình.

Lời sau cùng em một lần nữa kính chúc quý thầy, cô bộ môn Công nghệ thông tin Trường

Đại học Giao thông Vận tải – Phân hiệu tại thành phố Hồ Chí Minh thật nhiều sức khỏe

và thành công.

Tp. Hồ Chí Minh, ngày tháng năm 2020

Sinh viên thực hiện

**Nguyễn Hoàng Phụng**

**Ngô Tùng Hiền**

# **CHƯƠNG 1 TỔNG QUAN VỀ ĐỀ TÀI**

## 1.1Lý do chọn đề tài

* Hiện nay công nghệ thông tin đang phát triển mạnh mẽ và ngày càng phổ biến, tác động đến mọi mặt của xã hội. Nắm bắt được xu thế này, các công ty, doanh nghiệp, tổ chức, cá nhân nói chung cũng như các cửa hàng bán lẻ nói riêng đã ứng dụng công nghệ thông tin vào công việc quản lý, kinh doanh của mình và đã đạt được một số lợi ích nhất định.
* Nhiều cửa hàng bán lẻ đã tiến hành cài đặt các phần mềm chuyên dụng để phục vụ cho việc quản lý hàng hóa, bán hàng, quản lý chứng từ, hóa đơn, giảm gánh nặng cho người quản lý cũng như người bán hàng. Bên cạnh đó, vẫn còn nhiều cửa hàng sử dụng cách thức quản lý và bán hàng thủ công thông qua sổ sách, giấy tờ. Hình thức quản lý này dễ gây nhầm lẫn, thất thoát hàng hóa, tốn kém tiền bạc, quá trình xử lý chậm, nhất là đối với những công việc có khối lượng xử lý lớn.
* Nhận thấy được những hạn chế của việc quản lý và bán hàng thủ công , nhóm chúng em đã thực hiện đề tài: “XÂY DỰNG PHẦN MỀM QUẢN LÍ GIỎ HÀNG” vào việc kinh doanh để mang lại lợi ích bền vững cho người sử dụng.

1.2 Mục tiêu nghiên cứu

* Xây dựng phần mền quản lí giỏ hàng trên nền tảng window bằng C
* Ứng dụng phải đáp ứng giao diện bắt mắt với người dùng, đầy đủ các chức năng và dễ sử dụng
* Thực hiện được nhiều chức năng như: chọn hàng , xem chi tiết giỏ hàng

## 1.3 Khảo sát thực tế

* Nhằm Phục vụ nhu cầu truyền thông ngày càng cao của con người, hoạt động kinh doanh điện thoại di động ngày càng phát triển mạnh mẽ trong xu hướng bán lẻ của các doanh nghiệp. Từ thực tế khảo sát các cửa hàng bán hàng trên thị trường cho thấy, đa số các cửa hàng bán hàng đều chưa có phần mềm quản lý, các phương pháp quản lý đều thực hiện một cách thủ công chủ yếu bằng ghi chép sổ sách... Nhằm giảm thao tác thủ công, mang lại tính chính xác và hiệu quả cao trong công tác quản lý hoạt động kinh doanh.
* Với mục tiêu nhằm khắc phục được những vấn đề này nên em sẽ đi sâu tìm hiểu về cách tổ chức Quản lý cửa hàng bán hàng hóa với những nội dung chính sau:

+Quản lý nhập xuất (nhập hàng, xuất hàng,nhóm hàng, xem chi tiết giỏ hàng).

+Quản lý nhân sự (khách hàng, nhà cung cấp,thêm nhân viên, xóa nhânviên)

+ Tìm kiếm (tìm hàng hóa , tìm khách hàng, tím số lượng)

+ Báo cáo (báo cáo nhập hàng, xuất hàng, tôn kho,công nợ, doanh thu )

## 1.4 Cách tiếp cận, phương pháp nghiên cứu

* Tìm hiểu các thông tin cần thiết liên quan đến đề tài như: các phương pháp quản lí trước đây của cữa hàng, thiết kế các chức năng phù hợp cho người dùng, các thức người dùng thường mua đồ trên mạng
* Tổng hợp tài liệu liên quan đến đề tài
* Tham khảo giáo trình và slide bài giảng thị giác về các lý thuyết xử lý ảnh, thị giác máy tính.
* Xây dựng phần mềm dựa theo phương pháp từ trên xuống dưới và sử dụng mô hình thác nước để xây dựng.
* Tham khảo trang chủ các ngôn ngữ lập trình C

# 

# **Chương 2 CƠ SỞ LÝ THUYẾT**

## 2.1 Đọc ghi file trong C

Tại sao chúng ta cần đến file?

* Dữ liệu được lưu ở biến của chương trình, và nó sẽ biến mất khi chương trình kết thúc. Sử dụng file để lưu trữ dữ liệu cần thiết để đảm bảo dữ liệu của chúng ta không bị mất ngay cả khi chương trình của chúng ta ngừng chạy.
* Nếu chương trình của bạn có đầu vào(input) là lớn, bạn sẽ rất vất vả nếu phải nhập mỗi khi chạy. Thay vào đó, hãy lưu vào file và chương trình của bạn sẽ tự đọc mỗi lần khởi chạy
* Dễ dàng sao chép, di chuyển dữ liệu giữa các thiết bị với nhaug ta cần đến file?

### 2.1.1 Các kiểu file

#### a. File văn bản – text files

* File văn bản là file thường có đuôi là .txt. Những file này bạn có thể dễ dàng tạo ra bằng cách dùng các text editer thông dụng như Notepad, Notepad++, Sublime Text,…
* Khi bạn mở các file này bằng các text editer nói trên, bạn sẽ thấy được văn bản ngay và có thể dễ dàng thao tác sửa, xóa, thêm nội dung của file này.
* Kiểu file này thuận tiện cho chúng ta trong việc sử dụng hàng ngày, nhưng nó sẽ kém bảo mật và cần nhiều bộ nhớ để lưu trữ hơn.

#### b. File nhị phân – Binary files

File nhị phân thường có đuôi mở rộng là .**bin**

* Thay vì lưu trữ dưới dạng văn bản thuần thúy, các file này được lưu dưới dạng nhị phân, chỉ bao gồm các số 0 và 1. Bạn cũng sẽ thấy các con số này nếu cố mở nó bằng 1 text editer kể trên.
* Loại file này giúp lưu trữ được dữ liệu với kích thước lớn hơn, không thể đọc bằng các text editer thông thường và thông tin lưu trữ ở loại file được bảo mật hơn so với file văn bản.

### 2.1.2 Các thao tác với file

Trong ngôn ngữ lập trình C, có một số thao tác chính khi làm việc với file, bao gồm cả file văn bản và file nhị phân:

1. Tạo mới một file

2. Mở một file đã có

3. Đóng file đang mở

4. Đọc thông tin từ file/ Ghi thông tin ra file

### 2.1.3 Thao tác với file trên ngôn ngữ C

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

FILE \*fptr;

#### a.Thao tác mở file

Để đọc ghi file trong C cũng như trong mọi ngôn ngữ lập trình, việc đầu tiên bạn cần làm là mở file mà bạn muốn làm việc. Trong ngôn ngữ lập trình C, chúng ta có thể mở file bằng cách sử dụng hàm fopen( ) trong thư viện stdio.h như sau:

fptr = fopen("fileopen","mode");

Trong đó mode là một tham số chúng ta cần chỉ định.

Ví dụ:

fptr = fopen("E:\\cprogram\\newprogram.txt","w");

// hoặc

fptr = fopen("E:\\cprogram\\oldprogram.bin","rb");

* Giả sử tập tin newprogram.txt chưa có trong thư mục E:\cprogram. Ví dụ đầu tiên với mode = "w" sẽ cho phép chương trình tự động tạo ra file newprogram.txt nếu nó chưa có. Và sau đó mở file này lên nhưng chương trình chỉ có thể ghi dữ liệu vào mà không thể đọc.
* Mode là w chỉ cho phép chương trình ghi(nếu đã có dữ liệu thì ghi đè) nội dung của file.
* Với ví dụ thứ 2, mode là rb cho phép chương trình mở 1 file nhị phân đã có sẵn oldprogram.bin. Với trường hợp này, chương trình của bạn chỉ có thể đọc file và không thể ghi nội dung vào file.

#### b. Các tham số của “mode”

Dưới đây là các giá trị có thể có của tham số mode nói trên:

|  |  |  |
| --- | --- | --- |
| **Mode** | **Ý nghĩa** | **Nếu file không tồn tại** |
| r | Mở file chỉ cho phép đọc | Nếu file không tồn tại, fopen() trả về NULL. |
| rb | Mở file chỉ cho phép đọc dưới dạng nhị phân. | Nếu file không tồn tại, fopen() trả về NULL. |
| w | Mở file chỉ cho phép ghi. | Nếu file đã tồn tại, nội dung sẽ bị ghi đè. Nếu file không tồn tại, nó sẽ được tạo tự động. |
| wb | Open for writing in binary mode. | Nếu file đã tồn tại, nội dung sẽ bị ghi đè. Nếu file không tồn tại, nó sẽ được tạo tự động. |
| a | Mở file ở chế độ ghi “append”. Tức là sẽ ghi vào cuối của nội dung đã có. | Nếu file không tồn tại, nó sẽ được tạo tự động. |
| ab | Mở file ở chế độ ghi nhị phân “append”. Tức là sẽ ghi vào cuối của nội dung đã có. | Nếu file không tồn tại, nó sẽ được tạo tự động. |
| r+ | Mở file cho phép cả đọc và ghi. | Nếu file không tồn tại, fopen() trả về NULL. |
| rb+ | Mở file cho phép cả đọc và ghi ở dạng nhị phân. | Nếu file không tồn tại, fopen() trả về NULL. |
| w+ | Mở file cho phép cả đọc và ghi. | Nếu file đã tồn tại, nội dung sẽ bị ghi đè. Nếu file không tồn tại, nó sẽ được tạo tự động. |
| wb+ | Mở file cho phép cả đọc và ghi ở dạng nhị phân. | Nếu file đã tồn tại, nội dung sẽ bị ghi đè. Nếu file không tồn tại, nó sẽ được tạo tự động. |
| a+ | Mở file cho phép đọc và ghi “append”. | Nếu file không tồn tại, nó sẽ được tạo tự động. |
| ab+ | Mở file cho phép đọc và ghi “append” ở dạng nhị phân. | Nếu file không tồn tại, nó sẽ được tạo tự động. |

Hình 2.1

c. Thao tác đóng file

Khi làm việc với tập tin hoàn tất, kể cả là file nhị phân hay file văn bản. Bạn cần đóng file sau khi làm việc với nó xong.

Việc đóng file đang mở có thể được thực hiện bằng cách dùng hàm fclose().

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

2.1.4 Đọc/Ghi file văn bản trong C

Chúng ta sẽ học cách đọc ghi file trong C với file văn bản trước. Với file nhị phân, bạn kéo xuống dưới để xem tiếp.

Để làm việc với file văn bản, chúng ta sẽ sử dụng fprintf() và fscanf() .

**VD1. Ghi file sử dụng fprintf()**

#include <stdio.h>

#include <stdlib.h>

int main()

{

   int num;

   FILE \*fptr;

   fptr = fopen("C:\\program.txt","w");

   if(fptr == NULL)

   {

      printf("Error!");

      exit(1);

   }

   printf("Enter num: ");

   scanf("%d",&num);

   fprintf(fptr,"%d",num);

   fclose(fptr);

   return 0;

}

Chương trình nhận số num từ bàn phím và ghi vào file văn bản program.txt.

Sau khi bạn chạy chương trình này, bạn sẽ thấy file văn bản program.txt được tạo mới trong ổ C trên máy tính bạn. Khi mở file này lên, bạn sẽ thấy số mà bạn vừa nhập cho biến num kia.

**VD2. Đọc file sử dụng fscanf()**

#include <stdio.h>

#include <stdlib.h>

int main()

{

   int num;

   FILE \*fptr;

   if ((fptr = fopen("C:\\program.txt","r")) == NULL){

       printf("Error! opening file");

       // Program exits if the file pointer returns NULL.

       exit(1);

   }

   fscanf(fptr,"%d", &num);

   printf("Value of n=%d", num);

   fclose(fptr);

   return 0;

}

Chương trình này sẽ đọc giá trị số được lưu trong file program.txt mà chương trình ở VD1 vừa tạo ra và in lên màn hình.

2.1.5 Đọc/Ghi file nhị phân trong C

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

#### a. Ghi file nhị phân

Để ghi file nhị phân, bạn cần sử dụng hàm fwrite(). Hàm này cần 4 tham số: địa chỉ của biến lưu dữ liệu cần ghi, kích thước của biến lưu dữ liệu đó, số lượng kiểu dữ liệu của biến đó và con trỏ FILE trỏ tới file bạn muốn ghi.

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

**VD3. Ghi file nhị phân sử dụng fwrite()**

#include <stdio.h>

#include <stdlib.h>

struct threeNum

{

   int n1, n2, n3;

};

int main()

{

   int n;

   struct threeNum num;

   FILE \*fptr;

   if ((fptr = fopen("C:\\program.bin","wb")) == NULL){

       printf("Error! opening file");

       // Program exits if the file pointer returns NULL.

       exit(1);

   }

for(n = 1; n < 5; ++n)

   {

      num.n1 = n;

      num.n2 = 5\*n;

      num.n3 = 5\*n + 1;

      fwrite(&num, sizeof(struct threeNum), 1, fptr);

   }

   fclose(fptr);

   return 0;

}

b.Đọc file nhị phân

Trong VD3 này, chương trình sẽ tạo ra một file program.bin trên ổ đĩa C của bạn. Chương trình này đã khai báo 1 kiểu dữ liệu cấu trúc lưu 3 giá trị số n1, n2, n3; Và nó được sử dụng trong hàm main có tên biến là num.

Trong vòng lặp, các số được ghi vào file sử dụng hàm fwrite(). Các tham số gồm:

* Tham số đầu tiên là địa chỉ của biến num
* Tham số thứ 2 là kích thước của biến num
* Tham số thứ 3 là số lượng kiểu dữ liệu – ở đây là 1.
* Tham số thứ 4 là con trỏ FILE trỏ tới tệp tin program.bin

Cuối cùng, chúng ta đóng file sử dụng fclose().

c. Đọc file nhị phân

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

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

Ví dụ đọc file nhị phân sử dụng fread()

#include <stdio.h>

#include <stdlib.h>

struct threeNum

{

   int n1, n2, n3;

};

int main()

{

   int n;

   struct threeNum num;

   FILE \*fptr;

   if ((fptr = fopen("C:\\program.bin","rb")) == NULL){

       printf("Error! opening file");

        // Program exits if the file pointer returns NULL.

       exit(1);

   }

   for(n = 1; n < 5; ++n)

   {

      fread(&num, sizeof(struct threeNum), 1, fptr);

      printf("n1: %d\tn2: %d\tn3: %d", num.n1, num.n2, num.n3);

   }

   fclose(fptr);

   return 0;

}

Trong ví dụ này, bạn đọc file program.bin và lặp qua từng dòng. Bạn sẽ nhận được các giá trị tương ứng khi bạn ghi vào trong VD3.

### 2.1.6 Một số ví dụ về đọc ghi file trong C

Trong phần này, mình sẽ trình bày 2 ví dụ về đọc ghi file trong C, bao gồm các bài tập đọc ghi file sau:

* Ghi văn bản vào file trong C
* Đọc dữ liệu văn bản từ file trong C

Ghi vào file một câu văn bản

#include <stdio.h>

#include <stdlib.h>  /\* For exit() function \*/

int main()

{

   char sentence[1000];

   FILE \*fptr;

   fptr = fopen("program.txt", "w");

   if(fptr == NULL)

   {

      printf("Error!");

      exit(1);

   }

   printf("Enter a sentence:\n");

   gets(sentence);

   fprintf(fptr,"%s", sentence);

   fclose(fptr);

   return 0;

}

Chạy thử:



Hình 2.2

**Đọc dữ liệu văn bản từ file**

#include <stdio.h>

#include <stdlib.h> // For exit() function

int main()

{

    char c[1000];

    FILE \*fptr;

    if ((fptr = fopen("program.txt", "r")) == NULL)

    {

        printf("Error! opening file");

        // Program exits if file pointer returns NULL.

        exit(1);

    }

    // reads text until newline

    fscanf(fptr,"%[^\n]", c);

    printf("Data from the file:\n%s", c);

    fclose(fptr);

    return 0;

}

Giả sử file văn bản program.txt có nội dung như sau:

## 2.2 Lý thuyết về cấu trúc danh sách liên kết

Về bản chất, danh sách liên kết có chức năng như một mảng, có thể thêm và xóa các phần tử ở bất kỳ vị trí nào khi cần thiết. Một số sự khác nhau giữa danh sách liên kết và mảng:

|  |  |  |
| --- | --- | --- |
| **Nội dung** | **Mảng** | **Danh sách liên kết** |
| Kích thước | * Kích thước cố định * Cần chỉ rõ kích thước trong khi khai báo | * *Kích thước thay đổi trong quá trình thêm/ xóa phần tử* * *Kích thước tối đa phụ thuộc vào bộ nhớ* |
| Cấp phát bộ nhớ | * Tĩnh: Bộ nhớ được cấp phát trong quá trình biên dịch | * *Động: Bộ nhớ được cấp phát trong quá trình chạy* |
| Thứ tự & sắp xếp | * Được lưu trữ trên một dãy ô nhớ liên tục | * *Được lưu trữ trên các ô nhớ ngẫu nhiên* |
| Truy cập | * *Truy cập tới phần tử ngẫu nhiên trực tiếp bằng cách sử dụng chỉ số mảng: O(1)* | * Truy cập tới phần tử ngẫu nhiên cần phải duyệt từ đầu/cuối đến phần tử đó: O(n) |
| Tìm kiếm | * *Tìm kiếm tuyến tính hoặc tìm kiếm nhị phân* | * Chỉ có thể tìm kiếm tuyến tính |

Hình 2.3

**Lưu ý**: Ở bảng phía trên, các phần in nghiêng thể hiện đó là ưu điểm so với đối thủ còn lại.

## 2.3Danh sách liên kết là gì?

Danh sách liên kết đơn là một tập hợp các Node được phân bố động, được sắp xếp theo

cách sao cho mỗi Node chứa “một giá trị”(Data) và “một con trỏ”(Next). Con trỏ sẽ trỏ

đến phần tử kế tiếp của danh sách liên kết đó. Nếu con trỏ mà trỏ tới NULL, nghĩa là đó

là phần tử cuối cùng của linked list.

Hình ảnh mô tả cho một Node trong danh sách liên kết đơn:



Hình 2.4

Và đây là hình ảnh mô phỏng một danh sách liên đơn kết đầy đủ:



Hình 2.5

*Mô phỏng của danh sách liên kết đơn*

Danh sách các kiểu danh sách liên kết:

* Danh sách liên kết đơn(Single linked list): Chỉ có sự kết nối từ phần tử phía trước

tới phần tử phía sau.

* Danh sách liên kết đôi(Double linked list): Có sự kết nối 2 chiều giữa phần tử phía

trước với phần tử phía sau

* Danh sách liên kết vòng(Circular Linked List): Có thêm sự kết nối giữa 2 phần tử

đầu tiên và phần tử cuối cùng để tạo thành vòng khép kín.

## 2.4 Cài đặt danh sách liên kết đơn

### 2.4.1 Khai báo linked list

Để đơn giản hóa, data của chúng ta sẽ là số nguyên(int). Bạn cũng có thể sử dụng các

kiểu nguyên thủy khác(float, char,…) hay kiểu dữ liệu struct(SinhVien, CanBo,…) tự tạo.

struct LinkedList{

    int data;

    struct LinkedList \*next;

 };

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

**Tại sao next lại là kiểu LinkedList của chính nó?** Bởi vì nó là con trỏ trỏ của chính bản thân nó, và nó trỏ tới một thằng Node kế tiếp cũng có kiểu LinkedList.

### 2.4.2 Tạo mới 1 Node

Hãy tạo một kiểu dữ liệu của struct LinkedList để code clear hơn:

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

node CreateNode(int value){

    node temp; // declare a node

    temp = (node)malloc(sizeof(struct LinkedList)); // Cấp phát vùng nhớ dùng malloc()

    temp->next = NULL;// Cho next trỏ tới NULL

    temp->data = value; // Gán giá trị cho Node

    return temp;//Trả về node mới đã có giá trị

}

Mỗi một Node khi được khởi tạo, chúng ta cần cấp phát bộ nhớ cho nó, và mặc định cho con trỏ next trỏ tới NULL. Giá trị của Node sẽ được cung cấp khi thêm Node vào linked list.

* **typedef** được dùng để định nghĩa một kiểu dữ liệu trong C. VD: typeder long long LL;
* **malloc** là hàm cấp phát bộ nhớ của C. Với C++ chúng ta dùng new
* **sizeof** là hàm trả về kích thước của kiểu dữ liệu, dùng làm tham số cho hàm malloc

**Lưu ý**: Không giống với mảng, cần khai báo arr[size]. Trong linked list, vì mỗi Node sẽ có con trỏ liên kết đến Node tiếp theo. Do đó, với danh sách liên kết đơn, bạn chỉ cần lưu giữ Node đầu tiên(HEAD). Có head rồi bạn có thể đi tới bất cứ Node nào.

### 2.4.3 Thêm Node vào danh sách liên kết

Thêm vào đầu

Việc thêm vào đầu chính là việc cập nhật lại thằng head. Ta gọi Node mới(temp), ta có:

* Nếu head đang trỏ tới NULL, nghĩa là linked list đang trống, Node mới thêm vào sẽ làm head luôn
* Ngược lại, ta phải thay thế thằng head cũ bằng head mới. Việc này phải làm theo thứ tự như sau:

+ Cho next của temp trỏ tới head hiện hành

+ Đặt temp làm head mới

node AddHead(node head, int value){

    node temp = CreateNode(value); // Khởi tạo node temp với data = value

    if(head == NULL){

        head = temp;

}else{

        temp->next = head;

        head = temp;

    }

    return head;

}

Thêm vào cuối

Chúng ta sẽ cần Node đầu tiên, và giá trị muốn thêm. Khi đó, ta sẽ:

1. Tạo một Node mới với giá trị value
2. Nếu head = NULL, tức là danh sách liên kết đang trống. Khi đó Node

mới(temp) sẽ là head luôn.

1. Ngược lại, ta sẽ duyệt tới Node cuối cùng(Node có next = NULL), và trỏ

next của thằng cuối tới Node mới(temp).

node AddTail(node head, int value){

    node temp,p;

    temp = CreateNode(value

    if(head == NULL){

        head = temp;

    }

    else{

        p  = head;// Khởi tạo p trỏ tới head

        while(p->next != NULL){

            p = p->next;

        }

        p->next = temp;

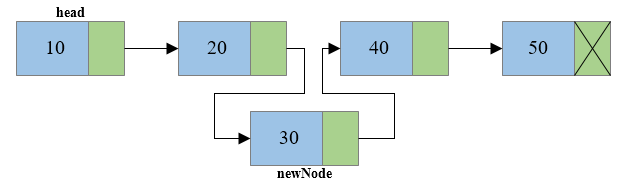
    }

    return head;

}

Tổng quan hơn, chúng ta sẽ sẽ viết hàm thêm một Node vào vị trí bất kỳ nhé.

**Thêm vào vị trí bất kỳ**



Hình 2.6

Để làm được việc này, ta phải duyệt từ đầu để tìm tới vị trí của Node cần chèn, giả sử là Node Q, khi đó ta cần làm theo thứ tự sau:

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

Lưu ý: Chỉ số chèn bắt đầu từ chỉ số 0 nhé các bạn

chèn là 0, tức là thêm vào đầu

    }else{

        // Bắt đầu tìm vị trí cần chèn. Ta sẽ dùng k để đếm cho vị trí

        int k = 1;

        node p = head;

        while(p != NULL && k != position){

            p = p->next;

            ++k;

        }

        if(k != position){

            // Nếu duyệt hết danh sách lk rồi mà vẫn chưa đến vị trí cần chèn, ta sẽ mặc định chèn cuối

            // Nếu bạn không muốn chèn, hãy thông báo vị trí chèn không hợp lệ

            head = AddTail(head, value);

            // printf("Vi tri chen vuot qua vi tri cuoi cung!\n");

        }else{

            node temp = CreateNode(value);

            temp->next = p->next;

            p->next = temp;

        }

    }

    return head;

}

Lưu ý: Bạn phải làm theo thứ tự trên, nếu bạn cho p->next = temp trước. Khi đó, bạn sẽ không thể lấy lại phần sau của danh sách liên kết nữa(Vì next chỉ được được lưu trong p->next mà thay đổi p->next rồi thì còn đâu giá trị cũ).

### 2.4.4 Xóa Node khỏi danh sách liên kết

**Xóa đầu**

Xóa đầu đơn giản lắm, bây giờ chỉ cần cho thằng kế tiếp của head làm head là được thôi. Mà thằng kế tiếp của head chính là head->next.

node DelHead(node head){

if(head == NULL){

        printf("\nCha co gi de xoa het!");

    }else{

        head = head->next;

    }

    return head;}

**Xóa cuối**

Xóa cuối mới nhọc nè, nhọc ở chỗ phải duyệt đến thằng cuối – 1, cho next của cuối – 1 đó bằng NULL.

node DelTail(node head){

    if (head == NULL || head->next == NULL){

         return DelHead(head);

    }

    node p = head;

while(p->next->next != NULL){

        p = p->next;

    }

    p->next = p->next->next; // Cho next bằng NULL

    // Hoặc viết p->next = NULL cũng được

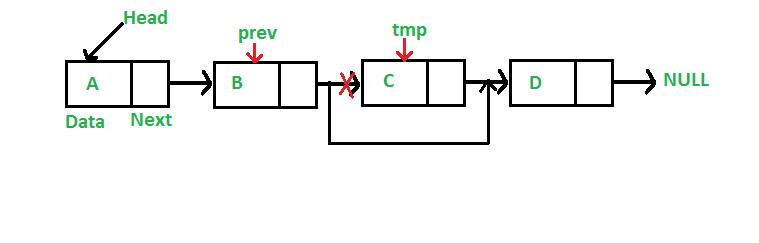
    return head;

}

Thằng Node cuối – 1 là thằng có p->next->next = NULL. Bạn cho next của nó bằng NULL là xong.

Xóa ở vị trí bất kỳ

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



Hình 2.7

*Xóa node trong danh sách liên kết*

Lưu ý: Chỉ số xóa bắt đầu từ 0 nhé các bạn. Việc tìm vị trí càn xóa chỉ duyệt tới Node gần cuối thôi(cuối – 1). Sau đây là code xóa Node ở vị trí bất kỳ

node DelAt(node head, int position){

    if(position == 0 || head == NULL || head->next == NULL){

        head = DelHead(head); // Nếu vị trí chèn là 0, tức là thêm vào đầu

    }else{

        // Bắt đầu tìm vị trí cần chèn. Ta sẽ dùng k để đếm cho vị trí

        int k = 1;

        node p = head;

        while(p->next->next != NULL && k != position){

            p = p->next;

            ++k;

        }

        if(k != position){

            // Nếu duyệt hết danh sách lk rồi mà vẫn chưa đến vị trí cần chèn, ta sẽ mặc định xóa cuối

            // Nếu bạn không muốn xóa, hãy thông báo vị trí xóa không hợp lệ

            head = DelTail(head);

            // printf("Vi tri xoa vuot qua vi tri cuoi cung!\n");

        }else{

            p->next = p->next->next;

        }}

        return head;}

### 2.4.5 Lấy giá trị ở vị trí bất kỳ

Chúng ta sẽ viết một hàm để truy xuất giá trị ở chỉ số bất kỳ nhé. Trong trường hợp chỉ số vượt quá chiều dài của linked list – 1, hàm này trả về vị trí cuối cùng. Do hạn chế là chúng ta không thể raise error khi chỉ số không hợp lệ. Tôi mặc định chỉ số bạn truyền vào phải là số nguyên không âm. Nếu bạn muốn kiểm tra chỉ số hợp lệ thì nên kiểm tra trước khi gọi hàm này.

int Get(node head, int index){

    int k = 0;

    node p = head;

    while(p->next != NULL && k != index){

        ++k;

        p = p->next;

    }

    return p->data;

}

Lý do dùng p->next != NULL là vì chúng ta chỉ muốn đi qua các phần tử có value.

### 2.4.6 Tìm kiếm trong danh sách liên kết

Hàm tìm kiếm này sẽ trả về chỉ số của Node đầu tiên có giá trị bằng với giá trị cần tìm. Nếu không tìm thấy, chúng ta trả về -1.

ta trả về -1.

int Search(node head, int value){

    int position = 0;

    for(node p = head; p != NULL; p = p->next){

        if(p->data == value){

            return position;

        }

        ++position;

    }

    return -1;

}

Chúng ta có thể sử dụng hàm này để xóa tất cả các Node trong danh sách liên kết có giá trị chỉ định như sau:

node DelByVal(node head, int value){

    int position = Search(head, value);

    while(position != -1){

        DelAt(head, position);

        position = Search(head, value);

    }

    return head;

}

### 2.4.7 Duyệt danh sách liên kết

Việc duyệt danh sách liên kết cực đơn giản. Khởi tạo từ Node head, bạn cứ thế đi theo con trỏ next cho tới trước khi Node đó NULL.

void Traverser(node head){

    printf("\n");

    for(node p = head; p != NULL; p = p->next){

        printf("%5d", p->data);

    }

}

## 3.1 Các thuật toán sắp xếp và tìm kiếm

### 3.1.1 Các thuật toán tìm kiếm :

#### a.Thuật toán tìm kiếm nhị phân

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

Giải thuật đơn giản nhất cho bài toán này là sử dụng linear search(tìm kiếm tuyến tính). Tức là bạn sẽ phải đi qua từng phần tử của mảng để đối chiếu với x cần tìm. Thuật toán này trong trường hợp xấu nhất cho độ phức tạp là O(n). Mình cũng sẽ minh họa code của thuật toán này dưới đây

Đây là code C/C++ sử dụng linear search.

#include<stdio.h>

#define Max\_Size 100

void nhapMang(int a[],int &n);

void xuatMang(int a[],int n);

int timkiem(int a[],int n,int x);

int main()

{

    int a[Max\_Size];

    int n,x;

printf("\n nhap n ");

   scanf("%d",&n);

   nhapMang(a,n);

   xuatMang(a,n);

   printf("\n nhap so can tim kiem= ");

   scanf("%d",&x);

   int tk=timkiem(a,n,x);

   if(tk != 0)

   {

       printf(" %d xuat hien tai chi so %d",x,tk);

   }

   else

   {

       printf(" %d khong có trong mang",x);

   }

    return 0;

}

void nhapMang(int a[],int &n)

{

    for(int i=0;i<n;i++){

    printf("\n nhap phan tu thu a[%d] ",i);

    scanf("%d",&a[i]);

    }

}

void xuatMang(int a[],int n)

{

    for(int i=0;i<n;i++)

    {

        printf("a[%d]=%d",i,a[i]);

    }

}

int timkiem(int a[],int n,int x)

{

    for(int i=0;i<n;i++)

    {

        if(a[i]== x)

        {

            return i;

        }

        return 0;

    }

}

Minh họa code cho thuật toán tìm kiếm nhị phân

Code minh họa C/C++ sử dụng đệ quy

#include <stdio.h>

// Hàm tìm kiếm nhị phân sử dụng giải thuật đệ quy

int binarySearch(int arr[], int l, int r, int x) {

  if (r >= l) {

    int mid = l + (r - l) / 2; // Tương đương (l+r)/2 nhưng ưu điểm tránh tràn số khi l,r lớn

    // Nếu arr[mid] = x, trả về chỉ số và kết thúc.

    if (arr[mid] == x)

      return mid;

    // Nếu arr[mid] > x, thực hiện tìm kiếm nửa trái của mảng

    if (arr[mid] > x)

      return binarySearch(arr, l, mid - 1, x);

    // Nếu arr[mid] < x, thực hiện tìm kiếm nửa phải của mảng

    return binarySearch(arr, mid + 1, r, x);

  }

  // Nếu không tìm thấy

  return -1;

}

int main(void) {

  int arr[] = {2, 3, 4, 10, 40};

  int n = sizeof(arr) / sizeof(arr[0]);

  int x = 10;

  int result = binarySearch(arr, 0, n - 1, x);

  if (result == -1)

    printf("%d xuat hien tai chi so %d", x, result);

  else

    printf("%d xuat hien tai chi so %d", x, result);

  return 0;

}

b.Thuật toán tìm kiếm tuyến tính.

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

**Giải thuật:**

Bước 1: Khởi tạo biến i và gán biến i bằng 0;

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

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

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

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

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

Bước 1: gán i=0.

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

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

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

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

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

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

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

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

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

Ngoài việc tìm kiếm trong mảng số chúng ta còn có thể áp dụng giải thuật tìm kiếm cho mảng, danh sách với cấu trúc dưới dạng struct

**Code ví dụ 1.**

void nhapMang(int a[],int &n)

{

    for(int i=0;i<n;i++){

    printf("\n nhap phan tu thu a[%d] ",i);

    scanf("%d",&a[i]);

    }

}

void xuatMang(int a[],int n)

{

    for(int i=0;i<n;i++)

    {

        printf("a[%d]=%d",i,a[i]);

    }

}

int timkiem(int a[],int n,int x)

{

    for(int i=0;i<n;i++)

    {

        if(a[i]== x)

        {

            return 1;

        }

        return 0;

    }

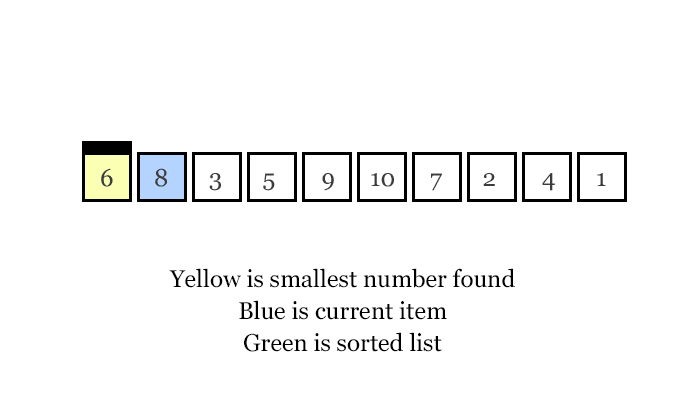
}

### 3.1.2 Các thuật toán sắp xếp :

#### a. Sắp xếp dãy số giảm dần

Trong code mà mình cung cấp dưới đây thì mình sẽ dùng thuật toán sắp xếp chọn – một thuật toán sắp xếp dễ hiểu và dễ cài đặt nhất.

Xem hình dưới đây để hiểu ý tưởng sắp xếp:



Hình 2.8

Code sắp xếp mảng/ dãy số giảm dần với C/C++:

#include <stdio.h>

int main(){

    int a[100];

    int n;

    printf("\nNhap so luong phan tu n = ");

    do{

        scanf("%d", &n);

        if(n <= 0){

            printf("\nNhap lai n = ");

        }

    }while(n <= 0);

    for(int i = 0; i < n; i++){

        printf("\nNhap a[%d] = ",i);

        scanf("%d", &a[i]);

    }

    // Sap xep dung thuat toan sap xep chon

    int tg;

    for(int i = 0; i < n - 1; i++){

        for(int j = i + 1; j < n; j++){

            if(a[i] < a[j]){

                // Hoan vi 2 so a[i] va a[j]

                tg = a[i];

                a[i] = a[j];

                a[j] = tg;

            }

        }

    }

    printf("\nMang da sap xep la: ");

    for(int i = 0; i < n; i++){

        printf("%5d", a[i]);

    }

}

#### b. Sắp xếp dãy số tăng dần

Việc sắp xếp dãy số tăng dần chỉ khác sắp xếp giảm dần duy nhât ở bước kiểm tra điều kiện để hoán vị.

#include <stdio.h>

void NhapMang(int a[], int n){

    for(int i = 0; i < n; i++){

        printf("\nNhap a[%d] = ",i);

        scanf("%d", &a[i]);

    }

}

void XuatMang(int a[], int n){

    for(int i = 0; i < n; i++){

        printf("%5d", a[i]);

    }

}

void TangDan(int a[], int n){

    int tg;

    for(int i = 0; i < n - 1; i++){

        for(int j = i + 1; j < n; j++){

            if(a[i] > a[j]){

                // Hoan vi 2 so a[i] va a[j]

                tg = a[i];

                a[i] = a[j];

                a[j] = tg;

            }

        }

    }

}

#### **c. Thuật toán sắp xếp nổi bọt.**

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

**Ví dụ minh họa**

Giả sử chúng ta cần sắp xếp dãy số [5 1 4 2 8] này tăng dần.  
**Lần lặp đầu tiên:**  
( **5** **1** 4 2 8 ) –> ( **1** **5** 4 2 8 ), Ở đây, thuật toán sẽ so sánh hai phần tử đầu tiên, và đổi chỗ cho nhau do 5 > 1.  
( 1 **5** **4** 2 8 ) –>  ( 1 **4** **5** 2 8 ), Đổi chỗ do 5 > 4  
( 1 4 **5** **2** 8 ) –>  ( 1 4 **2** **5** 8 ), Đổi chỗ do 5 > 2  
( 1 4 2 **5** **8** ) –> ( 1 4 2 **5** **8** ), Ở đây, hai phần tử đang xét đã đúng thứ tự (8 > 5), vậy ta không cần đổi chỗ.

**Lần lặp thứ 2:**  
( **1** **4** 2 5 8 ) –> ( **1** **4** 2 5 8 )  
( 1 **4** **2** 5 8 ) –> ( 1 **2** **4** 5 8 ), Đổi chỗ do 4 > 2  
( 1 2 **4** **5** 8 ) –> ( 1 2 **4** **5** 8 )  
( 1 2 4 **5** **8** ) –>  ( 1 2 4 **5** **8** )  
Bây giờ, dãy số đã được sắp xếp, Nhưng thuật toán của chúng ta không nhận ra điều đó ngay được. Thuật toán sẽ cần thêm một lần lặp nữa để kết luận dãy đã sắp xếp khi và khi khi nó đi từ đầu tới cuối mà không có bất kỳ lần đổi chỗ nào được thực hiện.

**Lần lặp thứ 3:**  
( **1** **2** 4 5 8 ) –> ( **1** **2** 4 5 8 )  
( 1 **2** **4** 5 8 ) –> ( 1 **2** **4** 5 8 )  
( 1 2 **4** **5** 8 ) –> ( 1 2 **4** **5** 8 )  
( 1 2 4 **5** **8** ) –> ( 1 2 4 **5** **8** )

**Code sắp xếp nổi bọt trong C/C++**

#include <stdio.h>

void swap(int &x, int &y)

{

    int temp = x;

    x = y;

    y = temp;

}

// Hàm sắp xếp bubble sort

void bubbleSort(int arr[], int n)

{

    int i, j;

    bool haveSwap = false;

    for (i = 0; i < n-1; i++){

    // i phần tử cuối cùng đã được sắp xếp

        haveSwap = false;

        for (j = 0; j < n-i-1; j++){

            if (arr[j] > arr[j+1]){

                swap(arr[j], arr[j+1]);

                haveSwap = true; // Kiểm tra lần lặp này có swap không

            }

        }

        // Nếu không có swap nào được thực hiện => mảng đã sắp xếp. Không cần lặp thêm

        if(haveSwap == false){

            break;

        }

    }

}

/\* Hàm xuất mảng \*/

void printArray(int arr[], int size)

{

    int i;

    for (i=0; i < size; i++)

        printf("%d ", arr[i]);

    printf("n");

}

// Driver program to test above functions

int main()

{

int arr[] = {64, 34, 25, 12, 22, 11, 90};

    int n = sizeof(arr)/sizeof(arr[0]);

    bubbleSort(arr, n);

    printf("Sorted array: n");

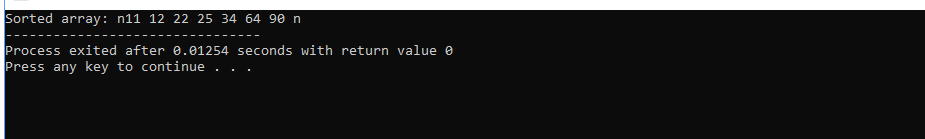
    printArray(arr, n);

    return 0;

}

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

Kiểm tra kết quả:



Hình 2.9

# 

# **Phần 3: DEMO CODE ĐỀ TÀI XÂY DỰNG PHẦN MỀM**

# **QUẢN LÍ GIỎ HÀNG VÀ HƯỚNG DẪN SỬ DỤNG**

## 3.1 Các hàm trong chương trình

- void taoGioHang(giohang\_st \*a){ là hàm dùng để nhập vào tên khách hàng , và địa chỉ

- void chonHang(giohang\_st \*a){ là hàm dùng để nhập vào tên sản phẩm , số lượng và giá

- void sapXepGioHang(giohang\_st \*a){ là hàm dùng để sắp xếp sản phẩm theo tên

- void hienThiGioHang(giohang\_st a){ là hàm dùng để hiển thị thông tin giỏ hàng

- void loaiBoMotMatHang(giohang\_st \*a){ là hàm dùng để xóa 1 mặt hàng theo lệnh

- void ghiFileNhiPhan(giohang\_st a, FILE \*p){ là hàng dùng để ghi vào nhị phân

- void docFileNhiPhan(giohang\_st \*a, FILE \*p){ là hàm dùng để đọc nhị phân

Hàm main

- int main(){ là hàm dùng để hiển thị menu và nhập các lệnh

## 3.2 Hướng dẫn sử dụng

Lưu ý

Bạn nên thực hiện từ bước 1 tới bước 2 đầu tiên nếu bạn nhập 3,4,5,6 trước hàm sẽ thoát

Chọn một lựa chọn trong menu

- Khi nhập 1 , chương trình sẽ yêu cầu bạn nhập vào tên khách hàng , và địa chỉ

- Khi nhập 2 , chương trình sẽ yêu cầu bạn nhập vào tên sản phẩm, số lượng , và giá

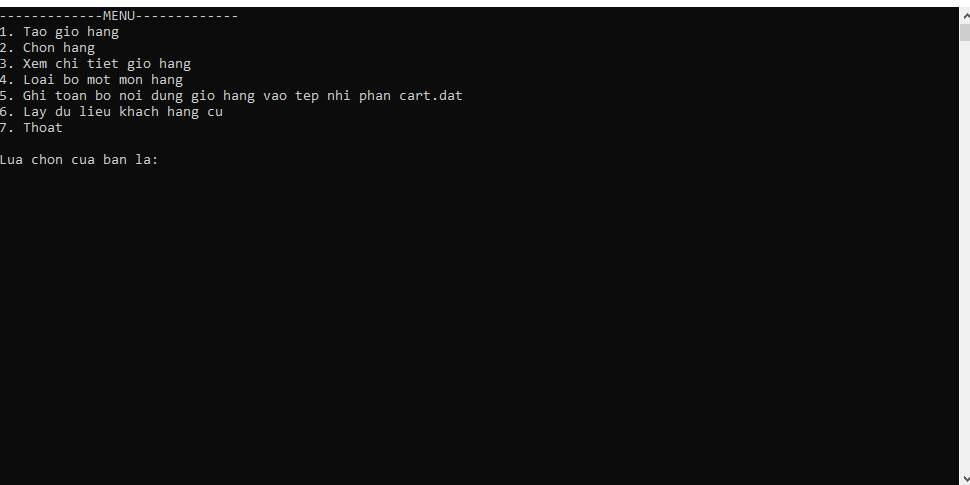
- Khi nhập 3 , chương trình sẽ hiển thị giỏ hàng từ những thông tin mà bạn nhập trước đó

- Khi nhập 4 , chương trình sẽ yêu cầu bạn nhập vào số thứ tự của thông tin bạn bần số nếu nó đúng thì chương trình sẽ xóa sản phẩm còn nếu sai chương trình sẽ yêu cầu nhập đúng số thứ tự cần được loại bỏ

- Khi nhập 5 , chương trình sẽ lưu toàn bộ dữ liệu vào tệp nhị phân cart.dat

- Khi nhập 6 , chương trình sẽ hiển thị thông tin từ những dữ liệu cũ bạn đã nhập vào

Hình ảnh đầu của chương trình



Hình 3.1

**Tài Liệu Tham Khảo**

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

<https://nguyenvanhieu.vn/thuat-toan-tim-kiem-nhi-phan/>

<https://nguyenvanhieu.vn/thuat-toan-sap-xep-chon/>

<https://nguyenvanhieu.vn/thuat-toan-sap-xep-noi-bot/>

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

<https://nguyenvanhieu.vn/danh-sach-lien-ket-don/>

<https://nguyenvanhieu.vn/doc-ghi-file-trong-c/#cac-kieu-file>

<https://vietjack.com/bai-tap-c/sap-xep-struct-trong-c.jsp>