TRƯỜNG ĐẠI HỌC GIAO THÔNG VẬN TẢI

PHÂN HIỆU TẠI TP. HỒ CHÍ MINH

BỘ MÔN CÔNG NGHỆ THÔNG TIN

BÁO CÁO MÔN HỌC

MÔN: KỸ THUẬT LẬP TRÌNH

Giảng viên hướng dẫn:	ThS. TRẦN PHONG NHÃ
Sinh viên thực hiện:	ĐINH HOÀNG PHÚC
	NGUYỄN MINH PHÚ
Mã số sinh viên:	6551071068
	6551071065
Lớp:	CQ.65.CNTT
Khóa:	65

Tp. Hồ Chí Minh, 24 tháng 05 năm 2025

TRƯỜNG ĐẠI HỌC GIAO THÔNG VẬN TẢI

PHÂN HIỆU TẠI TP. HỒ CHÍ MINH

BỘ MÔN CÔNG NGHỆ THÔNG TIN

BÁO CÁO MÔN HỌC

MÔN: KỸ THUẬT LẬP TRÌNH

Giảng viên hướng dẫn:	ThS. TRẦN PHONG NHÃ
Sinh viên thực hiện:	ĐINH HOÀNG PHÚC
	NGUYỄN MINH PHÚ
Mã số sinh viên:	6551071068
	6551071065
Lớp:	CQ.65.CNTT
Khóa:	65

Tp. Hồ Chí Minh, 24 tháng 05 năm 2025

LỜI CẢM ƠN

Lời nói đầu tiên,em xin gửi tới Quý Thầy Cô Bộ Môn Công Nghệ Thông Tin Trường Đại Học Phân Hiệu Giao Thông Vận Tải tại Thành Phố Hồ Chí Minh lời chúc sức khỏe và lời cảm ơn sâu sắc nhất. Trước tiên, em xin cảm ơn nhà trường đã tạo điều kiện cơ sở vật chất tốt nhất để em có thể hoàn thành tốt bài báo cáo này.

Đặc biệt, em xin gửi lời cảm ơn chân thành thầy Trần Phong Nhã đã nhiệt tình hướng dẫn, hỗ trợ và truyền đạt những kiến thức quý báo nhất trong quá trình thực hiện bài báo cáo này. Điều đó đã giúp cho em có thêm nhiều kiến thức bổ ích, trau dồi và rèn luyện thêm nhiều kĩ năng quan trọng . Nhờ vậy em có có cái nhìn sâu sắc hơn về môn học này.

Do những giới hạn về kiến thức, đồng thời bản thân em cũng thiếu kinh nghiệm trong việc thực hiện bao cáo vì vậy khó có thể tránh khỏi những thiếu sót. Em kính mong sự chỉ dẫn, nhận xét và đóng góp đến từ thầy để bài báo cáo này có thể hoàn thiện hơn nữa.

Lời cuối cùng,em xin kính chúc thầy cô bộ môn Công Nghệ Thông Tin Trường Đại Học Phân Hiệu Giao Thông Vận Tải tại Thành Phố Hồ Chí Minh và đặc biệt là thầy Trần Phong Nhã sẽ luôn có thật nhiều sức khỏe, hạnh phúc và thành công hơn nữa.

Em xin chân thành cảm ơn!.

NHẬN XÉT CỦA GIẢNG VIÊN HƯỚNG DẪN

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

Tp. Hồ Chí Minh, ngày … tháng … năm……

Giáo viên hướng dẫn

ThS. Trần Phong Nhã

MỤC LỤC

PHẦN A: LÝ THUYẾT

BÀI 1: HÀM

1.1 Khái niệm chung về hàm

Hàm (function) là một các khối lệnh có nhiệm vụ thực hiện một chức năng nào đó. Trong bất kỳ chương trình C nào, có một hoặc nhiều hàm được phân loại thành hàm thư viện và hàm do người dùng định nghĩa.

1.2 Tạo hàm do người dùng định nghĩa

Để khai báo hàm do người dùng định nghĩa gồm 3 bước:

- Khai báo hàm:

```
Cú pháp: data_type_function_name(type1 parameter1,type2 parameter2...);
```

- Định nghĩa hàm:

```
Cú pháp: data_type_function_name(type1 parameter1,type2 parameter2...){
//code}
```

- Lời gọi hàm:

Sau khi xây dựng hàm xong để hàm có thể thực thi bạn cần gọi nó trong hàm main và truyền cho nó tham số nếu cần. Khi bạn gọi hàm trong hàm main thì các câu lệnh bên trong hàm sẽ được thực thi, sau khi thực thi hết các câu lệnh thì hàm kết thúc và chương trình tiếp tục thực hiện các câu lệnh bên dưới hàm. Mỗi lần gọi hàm thì các câu lệnh trong hàm sẽ được thực hiện.

Ví dụ: Viết hàm tính tổng 2 số nguyên nhập từ bàn phím:

```
#include <stdio.h>
// Khai báo hàm
int tong(int a, int b);
        int main(){
        int x=5, y=10;
        int kq= tong(x,y); // Gọi hàm
    printf("Tong: %d\n",kq);
        return 0; }
```

```
// Định nghĩa hàm
int tong(int a,int b){
        return a+b; }
```

1.3 Các loại hàm trong ngôn ngữ lập trình C

1.3.1 Hàm có giá trị trả về. Hàm không có giá trị trả về

Hàm có giá trị trả về: Hàm có kiểu dữ liệu trả về (int, float, char,...). Sử dụng return để trả kết quả. thể tái sử dụng cho nhiều việc khác nhau.

Hàm không có giá trị trả về: Không trả về kết quả, chỉ thực hiện việc nào đó.

Ví dụ: Sử dụng hàm viết chương trình dùng hàm để: nhập vào mảng n phần tử nguyên, tính tổng mảng và xuất mảng đã nhập ra màn hình:

Ví dụ: Viết chương trình tính tích 2 số nguyên:

```
#include <stdio.h>
int tich1(int a, int b){
        return a*b; }
void tich2(int a, int b){
        printf("Tich a và b theo kieu tra ve void la: %d", a*b); }
int main(){      int a=6, b=8;
        printf("Tich a và b theo kieu tra ve int la: %d\n",tich1(a,b));
        tich2(a,b);
        return 0; }
```

1.3.2 Hàm đệ quy

Đệ quy là quá trình mà một hàm gọi chính nó. Ngôn ngữ C cho phép viết các hàm như vậy gọi chính nó để giải quyết các vấn đề phức tạp bằng cách chia nhỏ chúng thành các vấn đề đơn giản và dễ dàng. Các hàm này được gọi là hàm đệ quy.

Ví dụ : Ứng dụng phổ biến nhất của đệ quy là tính giai thừa:

```
#include <stdio.h>
int giaiThua(int i){
        if(i <= 1){
```

```
        return 1; } //cắm flag nếu hàm trả về 1 suy ra đối số truyền vào
không có giai thừa
        return i * giaiThua(i - 1); }
int main(){
        int a=6;
        int f = giaiThua(a);
        printf("Giai thua cua a la: %d", f);
        return 0; }
```

1.3.3 Hàm truyền tham số. Hàm không truyền tham số

Tham số (Parameter) hay tham số hình thức là các thành phần khi bạn xây dựng hàm. Đối số (Argument) hay tham số thực sự là các giá trị bạn truyền vào cho hàm khi gọi hàm. Có 2 cách truyền tham số cho hàm:

- Truyền theo tham trị (mặc định): giá trị của tham số thực sự (Argument) không bị thay đổi sau khi hàm kết thúc.

- Truyền theo tham chiếu: giá trị của tham số thực sự (Argument) có thể bị thay đổi khi hàm kết thúc. Khi xây dựng hàm cần đặt dấu & trước tham số hình thức.

Lưu ý: Đối với hàm có tham số là mảng thì giá trị của tham số thực sự (Argument) có thể bị thay đổi khi hàm kết thúc.

Ví dụ: Hàm doiCho() truyền tham số theo tham chiếu, đổi vị trí a và b:

```
#include <stdio.h>
void doiCho(int &a, int &b){
        int t;
        t = a; a = b; b = t;
        printf("a va b trong ham doiCho() la:\n a= %d\t b=%d\n", a, b);
}
int main(){
        int a=5, b=10;
        doiCho(a, b);
printf("a va b ngoai ham doiCho() la:\n a= %d\t b=%d\n", a, b);
        return 0; }
```

BÀI 2: CON TRỎ

2.1 Khái niệm

Con trỏ C là kiểu dữ liệu phái sinh được sử dụng để lưu trữ địa chỉ của một biến khác và cũng có thể được sử dụng để truy cập và thao tác dữ liệu của biến được lưu trữ tại vị trí đó. Con trỏ được coi là kiểu dữ liệu phái sinh.

2.2 Khai báo con trỏ

Để khai báo một con trỏ, hãy sử dụng toán tử hủy tham chiếu (*) theo sau là kiểu dữ liệu.

```
Cú pháp: type *var-name;
```

Ở đây, type là kiểu cơ sở của con trỏ; nó phải là kiêur dữ liệu C hợp lệ và var-name là tên của biến con trỏ. Dấu hoa thị * dùng để khai báo con trỏ giống với dấu hoa thị dùng cho phép nhân. Tuy nhiên, trong câu lệnh này, dấu hoa thị được dùng để chỉ định một biến là con trỏ.

Ví dụ: Khai báo con trỏ:

```
int *NamSinh;

char *GioiTinh;
```

Kiểu dữ liệu thực tế của giá trị của tất cả các con trỏ, dù là số nguyên, số thực, ký tự hay các loại khác, đều giống nhau, một số thập lục phân dài biểu diễn địa chỉ bộ nhớ. Sự khác biệt duy nhất giữa các con trỏ có kiểu dữ liệu khác nhau là kiểu dữ liệu của biến hoặc hằng số mà con trỏ trỏ tới.

2.3 Khởi tạo con trỏ

Sau khi khai báo một biến con trỏ, bạn cần khởi tạo nó bằng địa chỉ của một biến khác bằng cách sử dụng địa chỉ của toán tử (&) . Quá trình này được gọi là tham chiếu đến một con trỏ .

```
Cú Pháp: pointer variable = &variable;
```

Ví dụ : Khởi tạo con trỏ:

```
int x=2;

*n=&x;
```

Ở đây, x là một biến số nguyên, n là một con trỏ số nguyên. Con trỏ n đang được khởi tạo bằng x .

2.4 Con trỏ tham chiếu và hủy tham chiếu

Con trỏ tham chiếu đến một vị trí trong bộ nhớ. Việc lấy giá trị được lưu trữ tại vị trí đó được gọi là hủy tham chiếu con trỏ .

- Toán tử & − Còn được gọi là "Toán tử địa chỉ". Toán tử này được sử dụng để tham chiếu, tức là lấy địa chỉ của một biến hiện có (sử dụng &) để đặt một biến con trỏ.
- Toán tử * − Còn được gọi là "toán tử hủy tham chiếu". Hủy tham chiếu một con trỏ được thực hiện bằng toán tử * để lấy giá trị từ địa chỉ bộ nhớ mà con trỏ trỏ tới.

2.5 Ví dụ về con trỏ

Ví dụ : Dùng con trỏ cơ bản:

```
#include <stdio.h>
int main() {
    int x = 20;    // khai báo biến x
    int *n;        // khai báo biến con trỏ n
    n = &x;        // gán địa chỉ của x cho biến con trỏ n
    printf("Dia chi cua bien x: %p\n", &x);
    // địa chỉ được lưu trong biến con trỏ n
    printf("Dia chi luu trong bien n: %p\n", n);
    // truy cập giá trị thông qua con trỏ
    printf("Gia tri cua *n: %d\n", *n);
    return 0;}
```

Ví dụ : In giá trị và địa chỉ của một số nguyên:

```
#include <stdio.h>
int main() {
    int x = 100;
    printf("Bien: %d \t Dia chi: %p", x, &x);
    return 0; }
```

Ví dụ 3: Con trỏ trỏ tới con trỏ:

```
#include <stdio.h>

int main() {

    int x = 10;

    int *n = &x;

    int **k = &n;

    printf("x: %d \nDia chi cua x: %p \n\n", x, &x);

    printf("n: %p \nDia chi cua n: %p \n\n", n, &n);

    printf("x: %d \nGia tri tai n: %d \n\n", x, *n);

    printf("k: %p \nDia chi cua k: %p \n\n", k, &k);

    printf("n: %p \nGia tri tai k: %p \n\n", n, *k);

    printf("x: %d \n*n: %d \n**k: %d", x, *n, **k);

    return 0; }
```

Ví dụ 4: Con trỏ NULL:

```
#include <stdio.h>

int main() {

    int *n = NULL;

    printf("Gia tri cua n la: %p\n", n);

    return 0;}
```

BÀI 3: CON TRỎ MẢNG

3.1 Khái niệm

Con trỏ mảng là một con trỏ trỏ đến một mảng hay việc sử dụng con trỏ để tham chiếu đến các phần tử của mảng trong lập trình.

3.2 Khởi tạo và làm việc với con trỏ mảng

Trong ngôn ngữ lập trình C, một mảng thực chất là một con trỏ. Khi khai báo một mảng là cũng đang tạo một con trỏ trỏ tới địa chỉ của phần tử đầu tiên trong mảng, giúp quản lý nhiều mảng cùng lúc hoặc duyệt qua mảng một cách linh hoạt hơn. Truy cập các phần tử mảng theo dạng con trỏ mảng. Ta có các quy tắc sau:

&Tên_mảng[0] ~ Tên_mảng

&Tên_mảng[vị_trí] ~ Tên_mảng + vị_trí

Tên_mảng[vị_trí] ~ *(Tên_mảng + vị_trí)

Ví dụ: Viết chương trình c nhập mảng gồm n phần tử nhập từ bàn phím, xuất mảng vừa nhập và địa chỉ phần tử đầu tiên trong mảng:

```
#include <stdio.h>
#include <stdlib.h>
int main (){
        int n;
        printf("Nhap n: "); scanf("%d",&n);
//khai báo a là con trỏ mảng trỏ tới mảng arr
        int arr[n];
            int *a = arr;
//nhập mảng arr thông qua con trỏ a
        printf("Nhap mang arr: \n");
        for(int i=0; i<n; i++){
                printf("arr[%d]= ",i+1);
                       scanf("%d",a+i);  // a+i ~ arr[i]
        }
//xuất mảng
```

```
        printf("Mang da nhap la: \n");
        for(int i=0; i<n; i++){
                printf("%d\t",*(a+i));  //*(a+i) ~ arr[i]
        }
        printf("\nDia chi phan tu dau tien trong mang la: %d",a);  // a~
arr[0]
            return 0;
}
```

3.3 Ứng dụng của con trỏ mảng

- Truy xuất nhanh hơn và tối ưu hóa bộ nhớ

- Hỗ trợ việc làm việc với cấu trúc dữ liệu phức tạp như: mảng đa chiều, danh sách liên kết, cây hoặc đồ thị.

- Linh hoạt trong lập trình dữ liệu động như cấp phát động hàm malloc, calloc, rellloc,...

BÀI 4: MẢNG CON TRỎ

4.1 Khái niệm

Mảng con trỏ là một mảng trong đó mỗi phần tử không phải là một giá trị trực tiếp mà là một con trỏ. Mỗi con trỏ này thường trỏ đến một vùng nhớ khác, nơi chứa giá trị thực sự của các phần tử. Trong một mảng con trỏ, mỗi phần tử chứa con trỏ tới một kiểu dữ liệu cụ thể.

4.2 Khởi tạo và làm việc với mảng con trỏ

Để tạo một mảng con trỏ trong ngôn ngữ C, cần khai báo một mảng con trỏ theo cùng cách như khai báo con trỏ. Cú pháp để khai báo một mảng con trỏ trong ngôn ngữ lập trình C là:

```
Cú pháp: kiểu dữ liệu * Tên mảng con trỏ[kích thước mảng];
```

Ví dụ: Khai báo báo một mảng con trỏ kiểu int gồm 5 phần tử:

```
int *ptr[5];
```

Mảng con trỏ không giống như mảng thông thường. Mỗi phần tử của mảng con trỏ có thể trỏ đến một vùng nhớ khác nhau trong bộ nhớ, do đó chúng ta có thể sử dụng mảng con trỏ để lưu trữ địa chỉ của các biến khác nhau.

Ví dụ : Một mảng các con trỏ tới số nguyên:

```
#include <stdio.h>
int main (){
            int arr[]={1,2,3,4,5};
            int *a[5]; // khai báo mảng con trỏ gồm 5 phần tử
// cấp phát bộ nhớ cho các con trỏ trong mảng a bằng cách gán cho địa chỉ của mảng arr
        for(int i=0; i<5; i++){
                a[i]= &arr[i]; }
        printf("Mang co gia tri la: \n");
//duyệt mảng con trỏ
        for(int i=0; i<5; i++){
                printf("%d\t",*a[i]);
```

```
  }
    free(a);
    return 0; }
```

Ví dụ : Một mảng con trỏ tới các kí tự:

```
#include <stdio.h>
#include <string.h>
int main (){
        char *a[]={"Bai","bao","cao","duoc","10 diem."}; // khai báo mảng
con trỏ gồm 5 phần tử
//xuất mảng
        printf("Mang co gia tri la: \n");
        for(int i=0; i<5; i++){
                printf("%s\t",a[i]);
        }
    return 0;
    free(a);
}
```

Ví dụ : Một mảng trỏ tới các cấu trúc:

```
#include <string.h>
// tạo struct book
typedef struct {
        char ten[50];
        float gia;
} Book;
const int MAX = 3;
int main() {
        Book *book[MAX]; //khai báo mảng con trỏ
        for (int i = 0; i < MAX; i++) {
                book[i] = (Book*)malloc(sizeof(Book));
                snprintf(book[i]->ten, 50, "Book %d", i + 1);
```

```
                book[i]->gia = 100 + i;
        }
        for (int i = 0; i < MAX; i++) {
                printf("Ten: %s, Gia: %.2f000 VND\n", book[i]->ten,
book[i]->gia);
        }
//giải phóng bộ nhớ
        for (int i = 0; i < MAX; i++) {
                free(book[i]);
        }


  return 0;
}
```

4.3 Ứng dụng và nhưng lưu ý khi dùng mảng con trỏ

- Khi sử dụng mảng con trỏ, cần đảm bảo rằng mọi con trỏ được khởi tạo trước khi truy cập.

- Khi cấp phát động mảng con trỏ, luôn phải giải phóng bộ nhớ bằng free() để tránh rò rỉ.

- Mảng con trỏ rất hữu ích khi cần làm việc với dữ liệu động hoặc danh sách có độ dài không cố định.

- Lưu trữ dữ liệu phúc tạp như: chuỗi kí tự, mảng đa chiều,..

BÀI 5: CON TRỎ HÀM

5.1 Khái niệm:

Con trỏ trong C là một biến lưu trữ địa chỉ của một biến khác. Tương tự, một biến lưu trữ địa chỉ của một hàm được gọi là con trỏ hàm hoặc con trỏ đến một hàm. Con trỏ hàm có thể hữu ích khi bạn muốn gọi một hàm một cách động. Cơ chế của hàm gọi lại trong C phụ thuộc vào con trỏ hàm.

Con trỏ hàm trỏ đến mã như con trỏ thông thường. Trong con trỏ hàm, tên hàm có thể được sử dụng để lấy địa chỉ hàm. Một hàm cũng có thể được truyền dưới dạng đối số và có thể được trả về từ một hàm.

5.2 Khai báo trỏ hàm

Cú pháp: function_return_type(*Pointer_name)(function argument list)

Ví dụ: Khai báo và sử dụng con trỏ hàm để gọi một hàm:

```
#include <stdio.h>
// Định nghĩa hàm
void hello() {
    printf("Hello World");}
// Chương trình chính
int main() {
    void (*n)() = &hello; // Khai báo một con trỏ hàm
    (*n)(); // Gọi hàm bằng con trỏ hàm
    return 0; }
```

Lưu ý : Không giống như con trỏ thông thường là con trỏ dữ liệu , con trỏ hàm trỏ đến mã. Chúng ta có thể sử dụng tên hàm làm địa chỉ của nó (như trong trường hợp của một mảng). Do đó, con trỏ đến hàm hello() cũng có thể được khai báo như này : void (*n)()=hello**;**

5.3 Con trỏ hàm có đối số

Con trỏ hàm cũng có thể được khai báo cho hàm có đối số. Trong quá trình khai báo hàm, bạn cần cung cấp các kiểu dữ liệu cụ thể làm danh sách tham số.

Ví dụ : Con trỏ hàm có đối số:

```
#include <stdio.h>
// Hàm cộng hai số
int addition(int a, int b) {
    return a + b; }
int main() {
    int (*n)(int, int) = addition; // con trỏ hàm
    int x = 10, y = 20;
    int k = (*n)(x, y); // gọi hàm thông qua con trỏ
    printf("Cong x: %d va y: %d = %d", x, y, k);
    return 0; }
```

Ví dụ : Con trỏ đến hàm với các đối số con trỏ:

```
#include <stdio.h>
// Hàm hoán đổi giá trị của hai biến
void swap(int *a, int *b) {
    int k;
    k = *a;
    *a = *b;
    *b = k; }
int main() {
    void (*n)(int *, int *) = swap; // con trỏ hàm
    int x = 10, y = 20;
    printf("Gia tri x: %d va y: %d truoc khi swap\n", x, y);
    (*n)(&x, &y); // gọi hàm thông qua con trỏ
    printf("Gia tri x: %d va y: %d sau khi swap", x, y);
    return 0; }
```

5.4 Mảng các con trỏ hàm

Cú pháp: type (*ptr[])(args) = {fun1, fun2, ...};

Ví dụ: Mảng các con trỏ hàm:

```
#include <stdio.h>
```

```
// Hàm chia
float chia(int so1, int so2) {
    return (float)so1 / so2; }
// Hàm cộng
float cong(int so1, int so2) {
    return so1 + so2; }
// Hàm trừ
float tru(int so1, int so2) {
    return so1 - so2; }
// Hàm nhân
float nhan(int so1, int so2) {
    return so1 * so2; }
int main() {
    float (*danhSachHam[])(int, int) = {cong, tru, nhan, chia};
    int so1 = 15, so2 = 10;
// 1 là cộng, 2 là trừ, 3 là nhân, 4 là chia
    int phepToan = 3;
    if (phepToan > 4) return 0;
    printf("Ket qua: %.2f", (*danhSachHam[phepToan - 1])(so1, so2));
    return 0;}
```

BÀI 6: CẤP PHÁT ĐỘNG

6.1 Khái niệm

Cấp phát động (Dynamic memory allocation) là một kỹ thuật giúp ta có thể xin cấp phát một vùng nhớ phù hợp với nhu cầu của bài toán trong lúc thực thi thay vì phải khai báo cố định. Cấp phát động thường được sử dụng để cấp phát mảng động hoặc sử dụng trong các cấu trúc dữ liệu. Trong ngôn ngữ lập trình C cung cấp 4 hàm trong thư viện để ta có thể thao tác với việc cấp phát động vùng nhớ và giải phóng vùng nhớ sau khi sử dụng, bao gồm : malloc(), calloc(), free(), realloc().

6.2 Hàm malloc()

Hàm malloc() viết tắt của từ memory allocation tức là cấp phát động vùng nhớ, hàm này được sử dụng để xin cấp phát khối bộ nhớ theo kích thước byte mong muốn. Giá trị trả về của hàm là một con trỏ kiểu void, ta nên ép kiểu sang kiểu dữ liệu mà ta cần dùng. Các giá trị trong các ô nhớ được cấp phát là giá trị rác.

Cú pháp : ptr = (cast_type*)malloc(byte_size)

Trong đó : ptr là con trỏ lưu trữ ô nhớ đầu tiên của vùng nhớ được cấp phát; cast_type* là kiểu con trỏ mà bạn muốn ép kiểu sang; byte_size là kích thước theo byte bạn muốn cấp phát.

Ví dụ :

```
#include <stdio.h>
#include <stdlib.h>
int main(){
    //Cấp phát vùng nhớ tương đương mảng 100 phần tử int,sizeof(int) = 4
    int *a = (int*)malloc(100 * sizeof(int)); //Cấp phát vùng nhớ tương đương mảng 1000 phần tử char,sizeof(char) = 1
    char *c = (char*)malloc(1000 * sizeof(char));
    return 0; }
```

Trong ví dụ trên sau khi cấp phát động thì con trỏ a và c sẽ lưu giữ địa chỉ của ô nhớ đầu tiên trong các ô nhớ được cấp phát.

Ví dụ : Trong trường hợp không cấp phát đủ vùng nhớ thì hàm malloc sẽ trả về con trỏ NULL:

```
#include <stdio.h>
#include <stdlib.h>
int main(){
    int n = 10;
    int *a = (int*)malloc(n * sizeof(int));
    if(a == NULL){ printf("Cap phat khong thanh cong !\n");}
    else{   printf("Cap phat thanh cong !\n");
        for(int i = 0; i < n; i++){
            a[i] = 28 + i; }
        for(int i = 0; i < n; i++){
            printf("%d ", a[i]);        } }
    return 0; }
```

6.3 Hàm calloc()

Hàm calloc() viết tắt của contiguous allocation tương tự như malloc() sử dụng để cấp phát vùng nhớ động nhưng các giá trị của các vùng nhớ được cấp phát sẽ có giá trị mặc định là 0 thay vì giá trị rác như hàm malloc().

Cú pháp : ptr = (cast_type*) calloc(n, element_size)

Trong đó : ptr là con trỏ lưu trữ ô nhớ đầu tiên của vùng nhớ được cấp phát; cast_type* là kiểu con trỏ mà bạn muốn ép kiểu sang; n là số lượng phần tử bạn muốn cấp phát; element_size là kích thước theo byte của 1 phần tử.

Ví dụ :

```
#include <stdio.h>
#include <stdlib.h>
int main(){
    int n = 10;
    int *a = (int*)calloc(n, sizeof(int));
    if(a == NULL){
```

```
        printf("Cap phat khong thanh cong !\n"); }
    else{ printf("Cap phat thanh cong !\n");
        printf("Mang ban dau : ");
        for(int i = 0; i < n; i++){
            printf("%d ", a[i]); }
        for(int i = 0; i < n; i++){
            a[i] = 28 + i; }
        printf("\nMang sau khi thay doi : ");
        for(int i = 0; i < n; i++){
            printf("%d ", a[i]); } }
    return 0; }
```

6.4 Hàm free()

Hàm malloc() và calloc() xin cấp phát vùng nhớ nhưng lại không tự giải phóng vùng nhớ mà nó xin cấp phát, hàm free() có chắc năng giải phóng vùng nhớ mà malloc() hoặc calloc() đã xin cấp phát. Việc sử dụng free() sau khi sử dụng malloc() và calloc() là cần thiết để tránh lãng phí bộ nhớ.

Cú pháp : free(ptr)

Ví dụ :

```
#include <stdio.h>
#include <stdlib.h>
int main(){
    int n = 10;
    int *a = (int*)malloc(n * sizeof(int));
    if(a == NULL){  printf("Cap phat khong thanh cong !\n");    }
    else{  printf("Cap phat thanh cong !\n");
        for(int i = 0; i < n; i++){ a[i] = 28 + i; }
        for(int i = 0; i < n; i++){ printf("%d ", a[i]); }
        free(a);
        printf("\nGiai phong thanh cong !\n");    }
    return 0; }
```

6.5 Hàm realloc()

Hàm realloc() viết tắt của re-allocation tức là cấp phát lại, trong trường hợp sử dụng malloc() và calloc() nhưng cần bổ sung thêm ta sử dụng realloc(). realloc() giúp ta giữ lại các giá trị trên vùng nhớ cũ và bổ sung thêm vùng nhớ mới với các giá trị rác.

Ví dụ: Khi ta đang xin cấp phát 5 phần tử nhưng lại muốn cấp phát lại thành 10 phần tử và giữ nguyên giá trị của 5 phần tử trước đó. Nếu ta sử dụng malloc() hay calloc() thì 5 phần tử cũ sẽ không còn vì ta được cấp phát 1 vùng nhớ mới, calloc() thì chỉ bổ sung thêm vùng nhớ cho 5 phần tử mới còn 5 phần tử cũ thì vẫn giữ nguyên.

Cú pháp : ptr = (cast_type*)realloc(ptr, new_size)

Ví dụ :

```
#include <stdio.h>
#include <stdlib.h>
int main(){
    int n = 5;
    int *a = (int*)malloc(n * sizeof(int));
    if(a == NULL){ printf("Cap phat khong thanh cong !\n"); }
    else{   printf("Cap phat thanh cong !\n");
        for(int i = 0; i < n; i++){
            a[i] = 28 + i; }
        printf("Mang truoc khi cap phat lai : ");
        for(int i = 0; i < n; i++){
            printf("%d ", a[i]); }
        a = (int*)realloc(a, 10);
        printf("\nMang sau khi cap phat lai : ");
        for(int i = 0; i < 10; i++){
            printf("%d ", a[i]); }    }
     return 0; }
```

BÀI 7: XỬ LÍ TỆP

7.1 Khái niệm

Xử lý tệp trong C là quá trình xử lý các thao tác tệp như tạo, mở, đóng, ghi dữ liệu, đọc dữ liệu, đổi tên. Có hai loại tập tin: tập tin văn bảnv(.txt) và tập tin nhị phân (.bin, .png, .jpg,...).

7.2 Khởi tạo tệp

7.2.1 Con trỏ tệp

Khi làm việc với xử lý tệp, cần một con trỏ tệp để lưu trữ tham chiếu của cấu trúc FILE được trả về bởi hàm fopen(). Con trỏ tệp là bắt buộc đối với tất cả các hoạt động xử lý tệp. Khai báo con trỏ tệp:

```
Cú pháp: FILE *file_pointer;
```

7.2.2 Mở (tạo) tệp

Một tệp phải được mở để thực hiện bất kỳ thao tác nào. Hàm fopen() được sử dụng để tạo tệp mới hoặc mở tệp hiện có. Với filename: tên của tệp cần mở, mode: chế độ mở của tệp.

```
Cú pháp: FILE *fopen(const char * filename,const char * mode);
```

Một số chế độ mở tệp chính:

- " r ": Mở file để đọc.
- " w ": Mở file đẻ ghi.
- " a ": Mở file text lên để ghi tiếp vào cuối file mà không xóa nội dung cũ trong file.
- " rb ": Mở file theo kiểu file nhị phân.

7.2.3 Đóng tệp

Mỗi tệp phải được đóng sau khi thực hiện các thao tác trên tệp đó. Hàm close() đóng một tệp đã mở.

```
Cú pháp: int fclose(FILE *fp);
```

Hàm fclose() trả về số không nếu thành công hoặc trả về EOF (EOF là hằng số được định nghĩa trong tệp tiêu đề stdio.h) nếu có lỗi khi đóng tệp.

7.2.4 Đổi tên một tập tin

Hàm rename() được sử dụng để đổi tên một tệp hiện có từ tên tệp cũ thành tên tệp mới.

Cú pháp: int rename(const char *old_filename,const char *new_filename)

7.3 Các thao tác khi làm việc với tệp

7.3.1 Đọc và ghi vào 1 tệp văn bản

Các hàm để ghi dữ liệu vào tệp được mở ở chế độ có thể ghi:

- fputc() : Ghi một ký tự đơn vào một tệp.
- fputs() : Ghi một chuỗi vào tệp.
- fprintf() : Ghi một chuỗi được định dạng (dữ liệu) vào một tệp.

Các hàm thư viện để đọc dữ liệu từ một tệp được mở ở chế độ đọc:

- fgetc() : Đọc một ký tự đơn lẻ từ một tệp.
- fgets() : Đọc một chuỗi từ một tệp.
- fscanf() : Đọc một chuỗi được định dạng từ một tệp.

Ví dụ: Viết chương trình ghi nội dung bất kì từ bàn phím và đọc nội dung đó để hiện thị lên màn hình:

```
#include <stdlib.h>
#include <stdio.h>
int main() {
    FILE *file; //Khai báo con trỏ tệp *file
    char buffer[100];
    file = fopen("example.txt", "w"); //Tạo (mở) tệp example.txt để ghi
    if (file == NULL) {
        printf("Khong the mo tep de ghi!\n");
return 1;
```

```
        }
    printf("Nhap noi dung: ");
        fgets(buffer, sizeof(buffer), stdin); // Nhập nội dung từ bàn
phím vào tệp
    fprintf(file, "%s", buffer); // Ghi nội dung vào tệp với hàm
fprintf()
    fclose(file); // Đóng tệp
    file = fopen("example.txt", "r"); // Mở tệp example.txt để đọc
    if (file == NULL) {
        printf("Khong the mo de doc!\n");
        return 1;
    }
    printf("\nNoi dung trong tep la:\n");
    while (fgets(buffer, sizeof(buffer), file)) {// Đọc nội dung từ tệp
        printf("%s", buffer); //Hiển thị nội dung ra màn hình
    }
    fclose(file); //Đóng tệp
    return 0;
}
```

7.3.2 Đọc và ghi vào 1 tệp nhị phân

Các hoạt động đọc/ghi được thực hiện ở dạng nhị phân trong trường hợp tệp nhị phân. Bạn cần đưa ký tự " b " vào chế độ truy cập (" wb " để ghi tệp nhị phân, " rb " để đọc tệp nhị phân). Có hai hàm có thể được sử dụng cho đầu vào và đầu ra nhị phân:

- Hàm fread() đọc một khối byte được chỉ định từ một tệp.
- Hàm fwrite() ghi một khối byte được chỉ định từ bộ đệm vào một tệp.

BÀI 8: KIỂU CẤU TRÚC

8.1 Khai báo cấu trúc

Trong C++, cấu trúc do người dùng định nghĩa bằng từ khoá struct:

```
Cú Pháp: struct <Tên cấu trúc> {
   <Kiểu dữ liệu 1> <Tên thuộc tính 1>;
   <Kiểu dữ liệu 2> <Tên thuộc tính 2>;
   ... };
```

Trong đó:"struct" là từ khóa khai báo cấu trúc;"Tên cấu trúc" là do người dùng đặt, giống tên biến; "Thuộc tính" là khai báo giống biến, kết thúc bằng dấu";" .

Ví dụ:

```
struct NhanVien {
    char ten[50];
    int tuoi;
    float luong;
};
```

Lưu ý: Cấu trúc chỉ cần định nghĩa một lần trong chương trình và có thể được khai báo biến cấu trúc nhiều lần. Khi cấu trúc đã được định nghĩa, việc khai báo biến ở lần khác trong chương trình được thực hiện như khai báo biến thông thường:

```
Cú Pháp: <Tên cấu trúc> <Tên biến 1>, <Tên biến 2>;
```

Ví dụ:

```
NhanVien nv1, nv2;
```

8.2 Cấu trúc lồng nhau

Các cấu trúc có thể được định nghĩa lồng nhau khi một thuộc tính của một cấu trúc cũng cần có kiểu là một cấu trúc khác. Khi đó, việc định nghĩa cấu trúc cha được thực hiện như một cấu trúc bình thường, với khai báo về thuộc tính đó là một cấu trúc con:

```
Cú pháp: struct <Tên cấu trúc lớn >{
   <Kiểu dữ liệu 1> <Tên thuộc tính 1>;
```

```
   // Có kiểu cấu trúc
   <Kiểu cấu trúc nhỏ > <Tên thuộc tính 2>;
   …
   <Kiểu dữ liệu n> <Tên thuộc tính n>;
};
```

Ví dụ:

```
struct Ngay {
    int ngay,thang,nam; };
struct NhanVien {
    char ten[50];
    Ngay ngaySinh;
    float luong; };
```

Lưu ý: Cấu trúc con phải được định nghĩa trước cấu trúc cha.

8.3 Định nghĩa cấu trúc với từ khóa typedef

Để tránh phải dùng từ khoá struct mỗi khi khai báo biến cấu trúc, ta có thể dùng từ khóa typedef khi định nghĩa cấu trúc:

```
typedef struct {
    <Kiểu dữ liệu 1> <Tên thuộc tính 1>;
    <Kiểu dữ liệu 2> <Tên thuộc tính 2>;
    …
    <Kiểu dữ liệu n> <Tên thuộc tính n>;
} <Tên kiểu dữ liệu cấu trúc>;
```

Trong đó: Tên kiểu dữ liệu cấu trúc: là tên kiểu dữ liệu của cấu trúc vừa định nghĩa. Tên này sẽ được dùng như một kiểu dữ liệu thông thường khi khai báo biến cấu trúc.

Ví dụ:

```
typedef struct {
    int ngay, thang, nam; } Ngay;
typedef struct {
```

```
    char ten[50];
    Ngay ngaySinh;
    float luong; } NhanVien;
```

Khi đó, muốn có hai biến là nv1 và nv2 có kiểu cấu trúc NhanVien, ta chỉ cần khai báo như sau mà không cần từ khoá struct: NhanVien nv1, nv2;

Lưu ý: Không thể khai báo biến ngay trong typedef; tên ở dòng cuối mới là tên kiểu dữ liệu dùng khi khai báo.

8.4 Thao tác trên cấu trúc

Các thao tác trên cấu trúc bao gồm: Khai báo và khởi tạo giá trị ban đầu cho biến cấu trúc và truy nhập đến các thuộc tính của cấu trúc.

8.5 Khởi tạo giá trị ban đầu cho cấu trúc

8.5.1 Khởi tạo biến có cấu trúc đơn

Cú pháp: <Tên kiểu dữ liệu cấu trúc> <Tên biến>;

Ngoài ra, ta có thể khởi tạo các giá trị cho các thuộc tính của cấu trúc ngay khi khai báo bằng các cú pháp sau:

```
Cú pháp:  <Tên kiểu dữ liệu cấu trúc> <tên biến> = {
   <giá trị thuộc tính 1>,
   <giá trị thuộc tính 2>,
   …
   <giá trị thuộc tính n>
};
```

Trong đó: Giá trị thuộc tính: là giá trị khởi đầu cho mỗi thuộc tính, có kiểu phù hợp với kiểu dữ liệu của thuộc tính. Mỗi giá trị của thuộc tính được phân cách bằng“,”. Ví dụ:

```
NhanVien nv = {
    "Nguyen Van A",
    27,
    300f};
```

8.5.2 Khởi tạo các biến có cấu trúc lồng nhau

Trong trường hợp các cấu trúc lồng nhau, phép khởi tạo cũng thực hiện như thông thường với phép khởi tạo cho tất cả các cấu trúc con.

Ví dụ :

```
NhanVien nv = {
    "Nguyen Van A",
    {15, 05, 1980},
    300f};
```

8.6 Truy nhập đến thuộc tính của cấu trúc

Có hai cách để truy cập vào các thành viên cấu trúc: Dùng dấu”.” (thành viên hoặc toán tử chấm) và “->”(toán tử con trỏ cấu trúc).

Ví dụ: Dùng dấu chấm:

```
cout << nv.ten;
nv.tuoi += 1;
```

Ví dụ: Cấu trúc lồng nhau:

```
nv.ngaySinh.ngay = 16;
nv.ngaySinh.thang = 07;
```

Đối với kiểu cấu trúc lồng nhau, phép truy nhập đến thuộc tính được thực hiện lần lượt từ cấu trúc cha đến cấu trúc con.

8.7 Ví dụ về kiểu cấu trúc

```
   #include <stdio.h>
// Khai báo kiểu cấu trúc SinhVien
struct SinhVien {
    char ten[50];
    int tuoi;
    float diem;
};
```

```
int main() {
    struct SinhVien sv1;
    // Nhập thông tin sinh viên
    printf("Nhap ten sinh vien: ");
    fgets(sv1.ten, sizeof(sv1.ten), stdin);
    printf("Nhap tuoi: ");
    scanf("%d", &sv1.tuoi);
    printf("Nhap diem: ");
    scanf("%f", &sv1.diem);
    // In thông tin sinh viên
    printf("\n=== Thong tin sinh vien ===\n");
    printf("Ten: %s", sv1.ten);
    printf("Tuoi: %d\n", sv1.tuoi);
    printf("Diem: %.2f\n", sv1.diem);
    return 0;
}
```

BÀI 9: DANH SÁCH LIÊN KẾT

9.1 Khái niệm

Danh sách liên kết là một cấu trúc dữ liệu được sử dụng để lưu trữ một tập hợp các phần tử có thể được truy cập theo thứ tự. Nó bao gồm các nút (hay node) được liên kết với nhau theo một chiều hoặc hai chiều. Bạn có thể hình dung danh sách liên kết nó như một sợi xích vậy. Danh sách liên kết được chia làm ba loại:

- Danh sách liên kết đơn (Singly Linked List).
- Danh sách liên kết đôi (Doubly Linked List).
- Danh sách liên kết vòng (Circular Linked List).

9.2 Danh sách liên kết đơn

9.2.1 Khởi tạo danh sách liên kết đơn

Mỗi nút trong danh sách liên kết đơn chứa hai phần: một phần dữ liệu để lưu trữ giá trị của phần tử và phần để lưu trữ con trỏ trỏ đến phần tử tiếp theo trong danh sách. Node cuối cùng trong danh sách liên kết thì phần để lưu trữ con trỏ là con trỏ NULL.

Ví dụ: Viết chương trình khởi tạo một node và xuất ra màn hình giá trị của node đó:

```
#include <stdio.h>
#include <stdlib.h>
//Định nghĩa node
struct node{
    int data; //Có thể dùng kiểu dữ liệu khác để phù hợp với nhu cầu của chương trình       struct node *next;   //Cấu trúc tự trỏ: con trỏ trỏ tới node tiếp theo.
};
typedef struct node node;
int main(){
  // khởi tạo node *head
    node *head = (node*)malloc(sizeof(node));
    head->data = 100;
```

```
    head->next = NULL;
    printf("Gia tri cua head : %d\n", head);
    printf("Du lieu node ma head quan ly : %d\n", head->data);
    if(head->next == NULL){
        printf("Phan con tro cua head la NULL"); }}
```

9.2.2 Các thao tác cơ bản với danh sách liên kết đơn

Thêm phần tử vào Linked List

- Chèn vào đầu danh sách: Cần tạo một node mới và thay đổi con trỏ head để trỏ đến node mới này. Node mới sẽ trỏ đến node cũ.
- Chèn vào cuối danh sách: Cần duyệt qua toàn bộ danh sách để tìm node cuối cùng và thay đổi con trỏ của nó trỏ đến node mới.

Ví dụ: Hàm chèn vào cuối danh sách (độ phức tạp O(N)):

```
void insert (Node** head, int data) {
   // Tạo một nút mới và khởi tạo giá trị dữ liệu và con trỏ next của nó
Node* newNode = (Node*)malloc(sizeof(Node));
    newNode->data = data;
    newNode->next = NULL;
     // Nếu danh sách liên kết rỗng, thì đặt nút mới làm head
    if (*head == NULL) {
        *head = newNode;    }
else {         // Tìm nút cuối cùng trong danh sách liên kết
        Node* lastNode = *head;
        while (lastNode->next != NULL) {
            lastNode = lastNode->next;        }
 // Chèn nút mới vào cuối danh sách liên kết
        lastNode->next = newNode;}}
```

Xóa phần tử trong Linked List

- Xóa node đầu: Cần thay đổi con trỏ của node đầu tiên (head) trỏ đến node tiếp theo và giải phóng bộ nhớ của node cũ.
- Xóa node cuối: Cần duyệt qua danh sách để tìm node cuối cùng và node trước nó, sau đó thay đổi con trỏ của node trước để trỏ đến NULL.

Ví dụ: Hàm xóa vào cuối danh sách (độ phức tạp O(N)):

```
void deleteAtEnd(struct Node** head) {
    if (*head == NULL) return;
    struct Node* temp = *head;
    if (temp->next == NULL) {
        free(temp);
        *head = NULL;
        return;    }
    while (temp->next->next != NULL) {
        temp = temp->next; }
    free(temp->next);
    temp->next = NULL; }
```

Duyệt qua Linked List: Cần bắt đầu từ node đầu tiên (head) và di chuyển qua từng node bằng cách truy cập vào con trỏ "next" của mỗi node. Quá trình này tiếp tục cho đến khi gặp NULL (node cuối cùng).

Ví dụ: Hàm duyệt danh sách (độ phức tạp O(N)):

```
void duyet(Node* head) {
    Node* tam = head; // Con trỏ tam trỏ đến nút đầu tiên của danh sách liên kết đơn
    while (tam != NULL) { // Vòng lặp sẽ thực hiện cho đến khi tam trỏ đến NULL (cuối danh sách liên kết đơn)
        printf("%d",tam->data); // In ra giá trị của nút hiện tại
        tam = tam->next; // Con trỏ tam được di chuyển đến nút tiếp theo
    }
}
```

9.3 Ưu điểm và nhược điểm

9.3.1 Ưu điểm của danh sách liên kết

- Bộ nhớ được sử dụng linh hoạt, không cần cấp phát trước.
- Cho phép chèn và xóa phần tử một cách nhanh chóng mà không cần dịch chuyển dữ liệu.
- Hỗ trợ nhiều kiểu danh sách khác nhau phù hợp với từng ứng dụng.

9.3.2 Nhược điểm của danh sách liên kết

- Cần nhiều bộ nhớ hơn so với mảng do phải lưu trữ con trỏ.
- Truy xuất phần tử chậm hơn mảng vì không có chỉ mục trực tiếp.
- Khó khăn trong việc truy cập ngẫu nhiên.
- Phức tạp hơn khi triển khai so với mảng.

9.4 Ứng dụng của danh sách liên kết

Danh sách liên kết được sử dụng trong nhiều lĩnh vực, bao gồm:

- Quản lý bộ nhớ động, danh sách động.
- Thực hiện các thao tác trong cấu trúc dữ liệu khác: Stack (ngăn xếp), Queue (hàng đợi), hoặc thậm chí các loại cây nhị phân (Binary Trees).
- Xử lý các bài toán với dữ liệu chuỗi như: phân tích cú pháp (parsing), chuyển đổi biểu thức, hoặc quản lý các chuỗi ký tự lớn.
- Giải quyết các bài toán đệ quy phức tạp như trong các thuật toán đệ quy xử lý đồ thị hoặc cây.
- Cấu trúc dữ liệu cho hệ thống đa luồng: đặc biệt trong việc quản lý các tác vụ (task management) hoặc hàng đợi các tác vụ (task queue) trong các ứng dụng yêu cầu xử lý đồng thời.
- Giải quyết bài toán vòng lặp hoặc tuần hoàn như trong các ứng dụng game, quản lý tài nguyên vòng, hoặc xử lý các hàng đợi tuần hoàn.
- Xử lý dữ liệu phức tạp và đồ thị: đặc biệt hữu ích trong các thuật toán tìm kiếm và tối ưu hóa, như thuật toán Dijkstra để tìm đường đi ngắn nhất trong đồ thị.

PHẦN B: ỨNG DỤNG

XÂY DỰNG ỨNG DỤNG CHO VIỆC QUẢN LÍ THỰC ĐƠN VÀ DẶT MÓN THEO BÀN

1. Các tính năng của chương trình

- Người dùng có thể thêm món ăn vào thực đơn.
- Người dùng có thể cập nhật giá món ăn trong thực đơn .
- Người dùng có thể xóa món ăn khỏi thực đơn.
- Thực đơn tối đa có thể chứa 100 món ăn.
- In ra danh sách các món ăn trong thực đơn theo định dạng bảng với cột tên món ăn và giá.
- Quán ăn có tối đa 10 bàn.
- Người dùng có thể chọn một bàn cụ thể (từ 1 đến 10) và đặt món cho bàn đó.
- Người dùng có thể nhập đơn hàng gồm: tên món ăn và số lượng món ăn cần đặt.
- Khi hoàn thành đơn hàng, in ra tổng số tiền của các món ăn đã đặt cho bàn đó.
- In hóa đơn gồm: danh sách các món ăn đã đặt, tổng số tiền, … của bàn được chọn.
- Người dùng có thể hủy món ăn đã đặt theo bàn
- Thoát khỏi chương trình quản lý quán ăn.

2. Mục đích tạo ra chương trình

Chương trình quản lý quán ăn bằng ngôn ngữ C giúp đơn giản hóa quy trình vận hành, hỗ trợ quản lý thực đơn và đơn hàng theo bàn một cách hiệu quả. Hệ thống này có thể trở thành một ứng dụng đầy đủ chức năng, phù hợp cho các quán ăn nhỏ và vừa. Chương trình được phát triển nhằm đạt được các mục tiêu chính sau:

- Tự động hóa quy trình quản lý thực đơn: Giúp quán ăn lưu trữ và hiển thị danh sách món ăn một cách hiệu quả.
- Hỗ trợ đặt món theo bàn: Cho phép khách hàng chọn bàn và đặt món dễ dàng.
- Quản lý đơn hàng: Theo dõi đơn hàng của từng bàn để giúp nhân viên phục vụ nhanh chóng nắm bắt thông tin.

- Tính toán tổng chi phí: Hỗ trợ tính toán hóa đơn cho từng bàn, giúp giảm thiểu sai sót trong thanh toán.

3. Mô tả chương trình

Chương trình Quản lý Quán ăn với chức năng Đặt món theo Bàn được xây dựng bằng ngôn ngữ lập trình C, sử dụng tổng hợp các kiến thức cơ bản và nâng cao trong phần lý thuyết như: mảng, con trỏ, con trỏ mảng, mảng con trỏ, con trỏ hàm, cấp phát động, xử lý tệp, kiểu cấu trúc (struct) và danh sách liên kết đơn. Cụ thể:

- Dùng danh sách liên kết đơn để tạo và quản lý: thực đơn món ăn (mỗi món có tên và giá), đơn hàng của từng bàn (tên món, số lượng và tổng tiền), danh sách bàn ăn (mỗi bàn gắn liền với một danh sách đơn hàng riêng).
- Cấp phát động được sử dụng để quản lý bộ nhớ linh hoạt cho thực đơn và đơn hàng.
- Struct được dùng để định nghĩa các thực thể MonAn, DonHang và BanAn.
- Mảng con trỏ (BanAn **) dùng để quản lý nhiều bàn ăn một cách hiệu quả.
- Chương trình hướng tới mô hình menu điều khiển với đầy đủ các tính năng sau:
    - Cập nhật thực đơn: thêm món ăn vào thực đơn, cập nhật giá món ăn trong thực đơn, xóa món ăn khỏi thực đơn, hiển thị danh sách các món ăn trong thực đơn.
    - Đặt món (đặt đơn hàng) cho bàn (từ bàn số 1 đến bàn số 10): nhập đơn hàng (tên món ăn và số lượng món ăn), hiển thị tổng số tiền của các món ăn đã đặt, hủy món ăn đã đặt, in hóa đơn.
    - Thoát chương trình.

4. Source Code

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
typedef struct MonAn{ // Đinh nghĩa kiểu cấu trúc MonAn
    char ten[50];
    float gia;
```

```
    struct MonAn *next; //Cấu trúc tự trỏ:con trỏ trỏ tới MonAn tiếp theo
}MonAn;
typedef struct DonHang{// Đinh nghĩa kiểu cấu trúc DonHang
    char ten[50];
    int soLuong;
    float tongTien;
    struct DonHang *next; // Con trỏ đến đơn hàng tiếp theo
}DonHang;
typedef struct BanAn{// Đinh nghĩa kiểu cấu trúc BanAn
    int soBan;
    DonHang *donHang; // Danh sách đơn hàng của bàn
}BanAn;
int n=10; // Biến toàn cục chỉ quán có 10 bàn
//Hàm tạo món ăn mới
MonAn* taoMonAn(){
        MonAn *MonMoi=(MonAn*)malloc(sizeof(MonAn)); //Cấp phát động
MonMoi
        if(MonMoi==NULL){ // Kiểm tra lỗi bộ nhớ
                printf("Loi cap phat bo nho!\n");
                return NULL;
    }
        printf("Nhap ten mon an: ");
        fgets(MonMoi->ten,sizeof(MonMoi->ten),stdin); // Đọc tên MonMoi
từ bàn phím
        MonMoi->ten[strcspn(MonMoi->ten,"\n")]=0; // Thay kí tự '\n'
thành '\0'
        printf("Nhap gia mon an: ");
        while(scanf("%f",&MonMoi->gia)!=1){ // Nhập lại nếu nhập kí tự
khác số thực
                printf("Gia mon an vua nhap khong phu hop dinh dang.\nVui
long nhap lai: ");
                while(getchar()!='\n'); // Xóa kí tự xuống dòng
```

```
        }
        while(getchar()!='\n'); // Xóa kí tự xuống dòng khi kết thúc
        MonMoi->next=NULL;
        return MonMoi;
}
// Hàm thêm món mới vào thực đơn
void themMonAn(MonAn **thucDon) {
    MonAn *MonMoi=taoMonAn(); // Khởi tạo món mới
    if (*thucDon==NULL) { //Kiểm tra nếu thực đơn trống
        *thucDon=MonMoi; //Gán món mới là món đầu tiên  trong thực đơn
        printf("Mon an da duoc them vao thuc don.\n");
         return; // Thoát hàm
    }
    MonAn *temp=*thucDon; // Khởi tạo con trỏ chạy tạm thay thực đơn
    while(temp->next!=NULL){ // Duyệt thực đơn đến món cuối cùng
        temp=temp->next;
        }
    temp->next=MonMoi; // Gán món mới vào món cuối cùng trong thực đơn
    printf("Mon an da duoc them vao thuc don.\n");
}
// Hàm xác nhận người dùng có muốn tiếp tục thao tác
int xacNhan(){
        char yn;
        printf("Nhap 'Y' de dong y.\nNhap 'N' de huy bo.\nLua chon cua
ban la: ");
        while(1){ // Lặp vô hạn
                scanf("%c",&yn); // Nhập kí tự
                while(getchar()!='\n'); // Xóa kí tự xuống dòng
                if(yn=='y' || yn=='Y') return 1; // Nếu nhập y/Y thì
thoát và hàm trả về 1
```

```
                else if (yn=='n' || yn=='N') return 0; // Nếu nhập y/Y
thì thoát và hàm trả về 0
                else printf("Lua chon cua ban khong hop le.\nVui long
nhap lai: ");
        }
}
// Hàm tìm món ăn trong thực đơn, trả về số lượng món ăn tìm được
int timMonAn(MonAn *thucDon,const char *a){
    int dem=0; // Khởi tạo biến đếm bằng 0
    MonAn *temp=thucDon; // Khởi tạo con trỏ tạm thay thực đơn
    while(temp!=NULL){ // Duyệt thực đơn đến món ăn cuối cùng
        if(strcmp(temp->ten, a)==0){ // Kiểm tra món ăn trong thực đơn có
giống với món ăn cần tìm
            dem++; // Biến tăng 1
        }
        temp=temp->next; // Chuyển đến món tiếp theo trong thực đơn
    }
    return dem; // Trả về giá trị biến đếm và thoát hàm
}
// Hàm cập nhật giá món ăn trong thực đơn
void capNhatGiaMonAn(MonAn *thucDon,const char *tenMon){
    int dem=timMonAn(thucDon,tenMon); // Khởi tạo dem là số món ăn tìm
được
    if(dem==0){ // Kiểm tra không tìm thấy món ăn cần cập nhật giá
        printf("Mon an %s khong co trong thuc don.\n",tenMon);
        return; // Thoát
    }
    float giaMoi; // Khởi tạo giá mới
    printf("Nhap gia moi cho mon %s: ",tenMon);
    while(scanf("%f",&giaMoi)!=1 || giaMoi<=0){ // Nhập lại giá mới nếu
sai định dạng
        printf("Gia nhap khong hop le. Vui long nhap lai: ");
```

```
        while(getchar()!='\n'); // Xóa kí tự xuống dòng
    }
    while(getchar()!='\n'); // Xóa kí tự xuống dòng khi kết thúc vòng lặp
    printf("Xac nhan cap nhat gia %d mon %s?\n",dem,tenMon);
    if(!xacNhan()){ // Kiểm tra có muốn dừng thao tác cập nhật giá
        printf("Da huy thao tac cap nhat gia.\n");
        return; // Thoát hàm
    }
    MonAn *temp=thucDon; // Khởi tạo con trỏ tạm thay thực đơn
    while(temp!=NULL){ // Duyệt thực đơn đến món ăn cuối cùng
        if(strcmp(temp->ten,tenMon)==0){ // Kiểm tra nếu là món ăn cần
cập nhật
            temp->gia=giaMoi; // Gán giá trong thực đơn bằng giá mới
        }
        temp=temp->next; // Di chuyển đến món tiếp theo
    }
    printf("Da cap nhat gia mon an %s thanh %.0f VND cho %d mon.\n",
tenMon, giaMoi, dem);
}
// Hàm xóa món ăn trong thực đơn
void xoaMonAn(MonAn **thucDon,const char *a) {
    MonAn *temp=*thucDon; // Khởi tạo con trỏ tạm thay thực đơn
    MonAn *truoc=NULL; //Khởi tạo con trỏ trước thay món trước món bị xóa
    int dem=timMonAn(*thucDon,a); // Tạo dem bằng số lượng món cần xóa
tìm được
        if(dem==0){ // Kiểm tra nếu không tìm được món cần xóa
                printf("Mon an %s khong co trong thuc don.\n",a);
                return; // Thoát hàm
        }
        printf("Xac nhan xoa %d mon an %s trong thuc don?\n",dem,a);
        if(!xacNhan()){// Kiểm có muốn dừng thao tác xóa món
```

```
                printf("Da xac nhan huy thao tac xoa.\n");
                return; // Thoát hàm
        }
    while(temp!=NULL){  // Duyệt thực đơn đến món ăn cuối cùng
        if(strcmp(temp->ten,a)==0){ // Kiểm tra nếu là món cần xóa
            MonAn *diaChi=temp;//Khởi tạo con trỏ lưu địa chỉ món cần xóa
            if(truoc==NULL){ // Kiểm tra nếu món cần xóa là món đầu thực
đơn
                *thucDon=temp->next;//Gán thực đơn bắt đầu từ món kế tiếp
            }else{
                truoc->next=temp->next;//Duyệt thực đơn bỏ qua món bị xóa
            }
            free(diaChi); // Giải phóng địa chỉ món bị xóa
// Kiểm tra nếu xóa món đầu thì temp thành món đầu danh sách, ngược lại
thì tiếp tục
            temp=(truoc==NULL) ? *thucDon : truoc->next;
            continue; // Nếu có món bị xóa thì bắt đầu lại vòng lặp
        }
        truoc=temp; // Cập nhật lại con trỏ trước
        temp=temp->next; // Di chuyển đến món tiếp theo
    }
    printf("Da xoa tat ca mon %s khoi thuc don.\n",a);
}
// Hàm duyệt các món ăn trong thực đơn
void duyetThucDon(MonAn *thucDon){
        if(thucDon==NULL){ // Kiểm tra nếu thực đơn trống
                printf("Thuc don trong. Vui long them mon an vao thuc
don!\n");
                return; // Thoát hàm
        }
        printf("\n<================== THUC DON
==================>\n");
```

```
        printf("--------------------~~~o~oOo~o~~~----------------------\n");
        printf("|STT| Ten mon an                      | Gia        |\n");
        printf("-------------------------------------------------\n");
        MonAn *temp=thucDon; // Khởi tạo con trỏ tạm thay thực đơn
        int i=1; // Khởi tạo biến i là số thứ tự món ăn trong thực đơn
        while(temp!=NULL){
                printf("|%-3d| %-30s | %-10.0f |\n",i,temp->ten,temp-
>gia);
                temp= temp->next; // Di chuyển đến món tiếp theo
                i++; // Tăng i
        }
        printf("-------------------------------------------------\n");
}
// Hàm tạo danh sách bàn ăn
BanAn* taoBanAn(){
    BanAn *danhSach=(BanAn*)malloc(n*sizeof(BanAn)); // Cấp phát bộ nhớ
    if(!danhSach){ // Kiểm tra nếu cấp phát thất bại
        printf("\nKhong the cap phat bo nho cho danh sach ban\n");
        return NULL; // Trả về NULL nếu không cấp phát được
    }
    for(int i=0;i<n;i++) { // Khởi tạo thông tin cho từng bàn
        danhSach[i].soBan=i+1; // Đánh số bàn từ 1 đến n
        danhSach[i].donHang=NULL; // Chưa có đơn hàng nào
    }
    return danhSach; // Trả về con trỏ tới mảng bàn ăn
}
// Hàm đặt món ăn cho một bàn
int DatMonAn(BanAn **danhSachBan,MonAn *thucDon){
    if(thucDon==NULL){ // Kiểm tra nếu thực đơn trống thì không đặt món
                printf("Thuc don trong. Vui long them mon an vao thuc
don!\n");
```

```
                return 0; // Không thể đặt món
        }
        int soBan;
    printf("Nhap so ban muon dat trong pham vi tu 1-%d ban: ",n);
    // Nhập và kiểm tra đầu vào cho số bàn
    while(scanf("%d",&soBan)!=1 || soBan<=0 || soBan>n){
                printf("So ban vua nhap khong phu hop dinh dang.\nVui
long nhap lai: ");
                while(getchar()!='\n'); // Xóa kí tự xuống dòng
        }
        while(getchar()!='\n'); // Xóa ký tự xuống dòng khi kết thúc
    BanAn *ban=danhSachBan[soBan-1]; // Lấy con trỏ đến bàn được chọn
    char tenMon[50];
    int soLuong;
    MonAn *mon;
    do{ // Vòng lặp để nhập tên món có trong thực đơn
        printf("Nhap ten mon an: ");
        fgets(tenMon,sizeof(tenMon),stdin); // Nhập chuỗi có khoảng trắng
        tenMon[strcspn(tenMon,"\n")]=0; // Xóa ký tự newline cuối chuỗi
        mon=thucDon; // Tìm món ăn trong thực đơn theo tên
        while(mon && strcmp(mon->ten,tenMon)!=0){
            mon=mon->next; // Duyệt tiếp nếu tên không khớp
        }
        if(!mon){
            printf("Mon an khong ton tai trong thuc don. Moi nhap
lai\n");
        }
    }while(!mon); // Lặp lại nếu món ăn không tìm thấy
    printf("Nhap so luong: ");// Nhập số lượng món
    while(scanf("%d",&soLuong)!=1 || soLuong<=0){
```

```
                printf("So luong vua nhap khong phu hop dinh dang.\nVui
long nhap lai: ");
                while(getchar()!='\n'); // Xóa kí tự xuống dòng
        }
        while(getchar()!='\n'); // Xóa kí tự xuống dòng khi kết thúc
    DonHang *don=(DonHang*)malloc(sizeof(DonHang)); // Cấp phát bộ nhớ
    if(!don){
        printf("Khong the cap phat bo nho cho don hang\n");
        return 0;
    }
    // Gán thông tin cho đơn hàng
    strcpy(don->ten,tenMon); // Sao chép tên món vào đơn hàng
    don->soLuong=soLuong; // Gán số lượng
    don->tongTien=mon->gia*soLuong; // Tính tổng tiền = giá * số lượng
    don->next=NULL; // Đơn hàng mới nằm cuối, nên trỏ next = NULL
    if(!ban->donHang){ // Thêm đơn hàng vào cuối danh sách đơn hàng của
bàn
        ban->donHang=don; // Nếu bàn chưa có đơn hàng nào
    }
        else{ // Thêm đơn hàng vào cuối danh sách đơn hàng của bàn
        DonHang *TongDuyet=ban->donHang;
        while(TongDuyet->next){
            TongDuyet=TongDuyet->next; // Duyệt tới cuối danh sách
        }
        TongDuyet->next=don; // Gắn đơn hàng mới vào cuối danh sách
    }
    printf("Dat thanh cong %d %s cho ban %d\n",soLuong,tenMon,soBan);
    return 1; // Trả về 1 để báo đặt món thành công
}
// Hàm tính tổng tiền các món đã đặt của một bàn ăn
float tinhTongTien(BanAn *ban) {
```

```
    if(!ban){ // Kiểm tra nếu con trỏ bàn là NULL
        printf("\nBan nay khong ton tai");
        return 0; // Trả về 0 nếu bàn không hợp lệ
    }
    if(!ban->donHang){ // Kiểm tra nếu bàn chưa đặt món nào
        printf("\nBan %d chua dat mon nao",ban->soBan);
        return 0; // Trả về 0 nếu không có đơn hàng
    }
    float tongTien=0;
    DonHang *TongDuyet=ban->donHang; // Con trỏ duyệt danh sách đơn hàng
của bàn
    while(TongDuyet){ // Duyệt qua danh sách đơn hàng và tính tổng tiền
        tongTien+=TongDuyet->tongTien; // Cộng từng đơn hàng vào tổng
        TongDuyet=TongDuyet->next; // Chuyển sang đơn hàng tiếp theo
    }
    return tongTien; // Trả về tổng tiền tất cả các đơn hàng của bàn
}
// Hàm in hóa đơn cho một bàn ăn
void inHoaDon(BanAn *ban){
    if(!ban){ // Kiểm tra nếu con trỏ bàn là NULL
        printf("\nBan nay khong ton tai");
        return; // Không in hóa đơn nếu bàn không hợp lệ
    }
    if(!ban->donHang) { // Kiểm tra nếu bàn chưa gọi món nào
        printf("\nBan %d chua dat mon nao",ban->soBan);
        return; // Không in hóa đơn nếu không có đơn hàng
    }
    printf("\n                    HOA DON CUA BAN %d
\n",ban->soBan);
    printf("-------------------------~~~o~oOo~o~~~-----------------------
\n");
```

```
    printf("%-30s | %-10s | %-10s|\n","Ten Mon","So Luong","Thanh Tien");
    printf("-------------------------------------------------------
\n");
    // Duyệt danh sách đơn hàng và in từng món
    DonHang *TongDuyet=ban->donHang;
    while(TongDuyet){
        printf("%-30s | %-10d | %-10.0f|\n",TongDuyet->ten,TongDuyet-
>soLuong,TongDuyet->tongTien);
        TongDuyet=TongDuyet->next; // Chuyển đến đơn hàng tiếp theo
    }
    printf("-------------------------------------------------------
\n");
    float tong=tinhTongTien(ban); // Tính và in tổng tiền
    printf("\nTong so tien can thanh toan la: %.0f VND\n",tong);
}
// Hàm xóa một món ăn khỏi đơn hàng của một bàn cụ thể
void xoaDonHang(BanAn **danhSachBan,int n) {
    int soBan;
    printf("\nNhap so ban muon xoa don hang trong pham vi tu 1-%d ban:
",n); // Nhập và kiểm tra số bàn
    while(scanf("%d",&soBan)!=1 || soBan<=0 || soBan>n){
                printf("So ban vua nhap khong phu hop dinh dang.\nVui
long nhap lai: ");
                while(getchar()!='\n'); // Xóa kí tự xuống dòng
        }
        while(getchar()!='\n'); // Xóa kí tự xuống dòng khi kết thúc
    BanAn *ban=danhSachBan[soBan-1]; // Lấy con trỏ đến bàn cần xóa đơn
    if(!ban->donHang){ // Kiểm tra xem bàn có đơn hàng nào không
        printf("Ban %d chua co don hang nao\n",soBan);
        return; // Nếu không có đơn hàng thì thoát hàm
    }
    char tenMon[50];
```

```
    printf("Nhap ten mon an muon xoa: "); // Nhập tên món ăn cần xóa
    fgets(tenMon,sizeof(tenMon),stdin);
    tenMon[strcspn(tenMon, "\n")]=0; // Xóa ký tự newline cuối chuỗi
    printf("Xac nhan xoa mon %s khoi don hang ban so %d?\n", tenMon,
soBan); // Xác nhận trước khi xóa
    if (!xacNhan()) { // Giả định bạn có hàm xacNhan()trả về 1 nếu đồng ý
        printf("Da huy thao tac xoa mon an.\n");
        return; // Nếu không đồng ý thì thoát hàm
    }
    // Khởi tạo con trỏ để duyệt danh sách đơn hàng
    DonHang *truoc=NULL;
    DonHang *hienTai=ban->donHang;
    while(hienTai){ // Tìm món ăn cần xóa trong danh sách
        if(strcmp(hienTai->ten,tenMon)==0){
            if(truoc==NULL){ // Nếu là phần tử đầu danh sách
                ban->donHang=hienTai->next; // Cập nhật con trỏ đầu danh
sách
            }
                    else{
                truoc->next=hienTai->next; // Bỏ qua phần tử hiện tại
trong danh sách
            }
            free(hienTai); // Giải phóng bộ nhớ cho đơn hàng bị xóa
            printf("Da xoa mon %s khoi don hang cua ban
%d\n",tenMon,soBan);
            return; // Kết thúc hàm sau khi xóa
        }
        truoc=hienTai; // Lưu lại nút trước đó
        hienTai=hienTai->next; // Di chuyển đến nút tiếp theo
    }
    printf("Khong tim thay mon %s trong don hang cua ban
%d\n",tenMon,soBan); // Nếu không tìm thấy món ăn
```

```
}
// Hàm menu lựa chọn
int menu() {
    int luaChon;
    printf("\n------------------------------------------------------------
\n");
    printf("CAP NHAT THUC DON:\n");
    printf("   [1]. Them mon an vao thuc don.\n");
    printf("   [2]. Cap nhat gia mon an trong thuc don.\n");
    printf("   [3]. Xoa mon an khoi thuc don.\n");
    printf("   [4]. Hien thi danh sach mon an trong thuc don.\n");
    printf("DAT MON CHO BAN AN:\n");
    printf("   [5]. Nhap don hang.\n");
    printf("   [6]. Xoa don hang.\n");
    printf("   [7]. In hoa don.\n");
    printf("[0]. THOAT CHUONG TRINH.\n");
    printf("------------------------------------------------------------
\n");
    printf("Vui long nhap lua chon: ");
    while(scanf("%d",&luaChon)!=1){
                printf("So ban vua nhap khong phu hop dinh dang.\nVui
long nhap lai: ");
                while(getchar()!='\n'); // Xóa kí tự xuống dòng
        }
        while(getchar()!='\n'); // Xóa kí tự xuống dòng khi kết thúc
    return luaChon;
}
// Hàm giải phóng thực đơn
void giaiPhongThucDon(MonAn **thucDon){
    if(!thucDon || !(*thucDon)) return;//Kiểm tra thực đơn rỗng thì thoát
    MonAn *temp; // Khởi tạo con trỏ tạm
```

```
    while(*thucDon){ // Lặp đến khi thực đơn trống
        temp=*thucDon; // Gán con trỏ tạm thay thực đơn
        *thucDon=(*thucDon)->next; // Di chuyển đến món tiếp theo
        free(temp); // Giải phóng con trỏ tạm
    }
}
// Hàm giải phóng toàn bộ danh sách đơn hàng (DonHang) của một bàn
void giaiPhongDonHang(DonHang **donHang){
    if(!donHang || !(*donHang)) return; // Nếu con trỏ NULL hoặc danh sách rỗng thì thoát
    DonHang *temp;
    while(*donHang){ // Duyệt từng nút trong danh sách và giải phóng
        temp=*donHang; // Lưu địa chỉ nút hiện tại
        *donHang=(*donHang)->next; // Di chuyển sang nút tiếp theo
        free(temp); // Giải phóng nút hiện tại
    }
}
// Hàm giải phóng toàn bộ danh sách bàn ăn và các đơn hàng tương ứng
void giaiPhongDanhSachBan(BanAn **danhSachBan, int n){
    if (!danhSachBan) return; // Nếu danh sách không tồn tại thì thoát
    for (int i=0;i<n;i++){ // Duyệt qua từng bàn ăn trong mảng
        if (danhSachBan[i]){ // Giải phóng danh sách đơn hàng của bàn
            giaiPhongDonHang(&(danhSachBan[i]->donHang));
            free(danhSachBan[i]); // Giải phóng vùng nhớ của bàn ăn
        }
    }
    free(danhSachBan); // Giải phóng mảng chứa con trỏ đến các bàn
}
int main() {
    int luaChon;
    MonAn *thucDon=NULL; // Khởi tạo con trỏ thực đơn rỗng
```

```
    BanAn **danhSachBan=NULL; // Khởi tạo mảng con trỏ danh sách bàn ăn
    // Cấp phát bộ nhớ cho mảng con trỏ danh sách bàn ăn (gồm n bàn)
    danhSachBan=(BanAn**)malloc(n*sizeof(BanAn*));
    if(!danhSachBan){
        printf("\nLoi cap phat bo nho cho danh sach ban\n");
        return 1; // Thoát chương trình nếu cấp phát thất bại
    }
    BanAn *banKhoiTao=taoBanAn(); //Khởi tạo bàn ăn dùng con trỏ đơn
    // Cấp phát bộ nhớ từng bàn và gán số bàn, khởi tạo đơn hàng = NULL
    for(int i=0;i<n;i++) {
        danhSachBan[i] = (BanAn*)malloc(sizeof(BanAn)); // Cấp phát bộ
nhớ cho từng bàn
        danhSachBan[i]->soBan = i + 1; // Gán số bàn từ 1 đến n
        danhSachBan[i]->donHang = NULL;//Khởi tạo danh sách đơn hàng rỗng
    }
    do{
        luaChon=menu();// Gọi hàm menu và nhận lựa chọn
        DonHang *don=NULL; // Khởi tạo con trỏ đơn hàng
        switch(luaChon){
            case 1: // Lựa chọn 1
                int soLuongMon;
                printf("Nhap so luong mon an can them: ");
                // Nhập lại nếu nhập số lượng món ăn sai định dạng
                while(scanf("%d",&soLuongMon)!=1 || soLuongMon<=0){
                    printf("So luong mon an ban nhap khong dung dinh
dang.\nVui long nhap lai: ");
                    while(getchar()!='\n'); // Xóa kí tự xuống dòng
                }
                while(getchar()!='\n'); // Xóa kí tự xuống dòng khi kết
thúc vòng lặp
                for(int i=0;i<soLuongMon;i++){
```

```
                themMonAn(&thucDon); // Gọi hàm thêm món ăn vào thực
đơn
            }
            break;
        case 2: // Lựa chọn 2
            if (thucDon==NULL) { // Kiểm tra nếu thực đơn trống
                printf("Thuc don trong, khong co mon an de cap
nhat!\n");
                break; // Dừng case 2
            }
            int n2;
            printf("Nhap so luong mon an can cap nhat gia: ");
            // Nhập lại nếu nhập số lượng món ăn sai định dạng
            while(scanf("%d",&n2)!= 1 || n2<=0){
                printf("So luong khong hop le.Vui long nhap lai: ");
                while(getchar()!='\n'); // Xóa kí tự xuống dòng
            }
            while(getchar()!='\n'); // Xóa kí tự xuống dòng khi kết
thúc vòng lặp
            {   // Cấp phát động mảng con trỏ tên món ăn
                char **a2=(char**)malloc(n2*sizeof(char*));
                printf("Nhap ten cac mon an can cap nhat gia:\n");
                for (int i=0;i<n2;i++) {
                    a2[i]=(char*)malloc(30*sizeof(char));
                    printf("Mon so %d: ",i+1);
                    fgets(a2[i],50,stdin);
                    a2[i][strcspn(a2[i],"\n")]= 0; // Xóa kí tự xuống
dòng
                    capNhatGiaMonAn(thucDon,a2[i]); // Cập nhật giá
món ăn
                    free(a2[i]); // Giải phóng con trỏ tên món ăn
                }
```

```
                    free(a2); // Giải phóng mảng con trỏ tên món ăn
                }
                break;
            case 3: // Lựa chọn 3
                if(thucDon==NULL){ // Kiểm tra thực đơn rỗng
                    printf("Thuc don trong, khong co mon an de xoa!\n");
                    break; // Dừng case 3
                }
                    int n3;
                    printf("Nhap so luong mon an can xoa: ");
                    // Nhập lại nếu nhập số lượng món ăn sai định dạng
                    while(scanf("%d", &n3)!=1 || n3<=0){
                        printf("So luong mon an can xoa vua nhap khong
dung dinh dang.\nVui long nhap lai: ");
                        while(getchar()!='\n'); // Xóa kí tự xuống dòng
                    }
                    while(getchar()!='\n'); // Xóa kí tự xuống dòng khi
kết thúc lặp
                    { // Cấp phát động mảng con trỏ tên món ăn cần xóa
                    char **a3=(char**)malloc(n3*sizeof(char*));
                    printf("Nhap ten cac mon an can xoa:\n");
                    for(int i=0;i<n3;i++){
                        a3[i]=(char*)malloc(30 * sizeof(char));
                        printf("Mon so %d: ",i+1);
                        fgets(a3[i],50,stdin);
                        a3[i][strcspn(a3[i],"\n")]=0; // Xóa kí tự xuống
dòng
                        xoaMonAn(&thucDon,a3[i]); // Gọi hàm xóa món ăn
                        free(a3[i]); // Giải phóng con trỏ tên món ăn
                    }
                    free(a3); // Giải phóng mảng con trỏ tên món ăn
```

```
                }
                break; // Dừng case 3
            case 4: // Lựa chọn 4
                duyetThucDon(thucDon); // Duyệt và in thực đơn
                break;
            case 5: // Lựa chọn 5
                DatMonAn(danhSachBan,thucDon); // Đặt món ăn cho bàn
                break;
            case 6: // Lựa chọn 6
                xoaDonHang(danhSachBan,n); // Xóa món ăn khỏi đơn hàng đã
đặt
                break;
            case 7: // Lựa chọn 7
                // In hóa đơn cho bàn cụ thể
                int soBan;
                printf("\nNhap so ban can in hoa don: ");
                // Kiểm tra định dạng và giới hạn số bàn
                while(scanf("%d",&soBan)!=1 || soBan<=0 || soBan>n){
                    printf("So ban vua nhap khong phu hop dinh dang.\nVui
long nhap lai: ");
                    while(getchar()!='\n'); // Xóa kí tự xuống dòng
                }
                while(getchar()!='\n'); // Xóa kí tự xuống dòng khi kết
thúc lặp
                inHoaDon(danhSachBan[soBan-1]); // Gọi hàm in hóa đơn cho
bàn đã chọn
                break;
            case 0: // Lựa chọn 0
                // Thoát chương trình, giải phóng bộ nhớ
                giaiPhongThucDon(&thucDon); // Giải phóng danh sách thực
đơn
                free(banKhoiTao); // Giải phóng con trỏ bàn khởi tạo
```

```
                giaiPhongDanhSachBan(danhSachBan, n); // Giải phóng danh
sách các bàn ăn
                printf("Thoat chuong trinh thanh cong\n"); // Thông báo
thoát
                break;
            default: // Xử lý lựa chọn không hợp lệ
                printf("Lua chon khong hop le.Vui long nhap lai\n");
        }
    } while(luaChon!=0); // Lặp lại cho đến khi người dùng chọn 0
    return 0; // Kết thúc chương trình
}
```

TÀI LIỆU THAM KHẢO

[1]. “C Tutorial – Learning C Programming”,[Online]. Available: C Tutorial (w3schools.com). [Accessed 16/04/2025].

[2]. “Lập trình C cơ bản – Giới thiệu ngôn ngữ C”,[Online]. Available: Lập trình C cơ bản - Giới thiệu ngôn ngữ C (200lab.io). [Accessed 16/04/2025].

[3]. “Stack Overflow”, [Online]. Available: Stack Overflow - Where Developers Learn, Share, & Build Careers. [Accessed 16/04/2025].

[4]. “GitHub”, [Online]. Available: GitHub (https://github.com). [Accessed 19/04/2025].