TRƯỜNG ĐẠI HỌC GIAO THÔNG VẬN TẢI
PHÂN HIỆU TẠI TP. HỒ CHÍ MINH
BỘ MÔN CÔNG NGHỆ THÔNG TIN

BÁO CÁO BÀI TẬP LỚN

MÔN: KĨ THUẬT LẬP TRÌNH

Giảng viên hướng dẫn: ThS.TRẦN PHONG NHÃ

Sinh viên thực hiện: NGUYỄN THÀNH HUY

MSSV: 6551071034

Sinh viên thực hiện: NGUYỄN GIA KHANG

MSSV: 6551071042

Lớp : CQ.65.CNTT

Khoá : K65

Tp. Hồ Chí Minh, năm 2024

TRƯỜNG ĐẠI HỌC GIAO THÔNG VẬN TẢI
PHÂN HIỆU TẠI TP. HỒ CHÍ MINH
BỘ MÔN CÔNG NGHỆ THÔNG TIN

BÁO CÁO BÀI TẬP LỚN

MÔN: KĨ THUẬT LẬP TRÌNH

Giảng viên hướng dẫn: ThS.TRẦN PHONG NHÃ

Sinh viên thực hiện: NGUYỄN THÀNH HUY

MSSV: 6551071034

Sinh viên thực hiện: NGUYỄN GIA KHANG

MSSV: 6551071042

Lớp : CQ.65.CNTT

Khoá : K65

Tp. Hồ Chí Minh, năm 2024

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 2025

Giáo viên hướng dẫn

ThS.TRẦN PHONG NHÃ

MỤC LỤC

BẢNG BIỂU, SƠ ĐỒ, HÌNH VẼ

PHẦN A: LÝ THUYẾT

BÀI 1: HÀM

1. Định Nghĩa Hàm

Bạn đã từng sử dụng các hàm như sqrt, pow, abs... của các thư viện có sẵn trong C, tuy nhiên để giải quyết bài toán của bạn thì bạn cần phải tự xây dựng các hàm để giải quyết các chức năng nhỏ cho bài toán của mình.

Hàm (function) là một các khối lệnh có nhiệm vụ thực hiện một chức năng nào đó.

Hình 1.1:Lợi ích của hàm

Cú Pháp:

```
data_type function_name(type1 parameter1, type2 parameter2...){
    //code
}
```

Các thành phần của hàm :

- data_type : Kiểu trả về của hàm, có thể là các kiểu dữ liệu như int, long long, float, char, double, hoặc void (tương ứng với kiểu trả về là rỗng)
- function_name : Tên của hàm, cần tuân theo quy tắc như đặt tên biến

- parameter : Tham số của hàm, đây được coi như đầu vào của hàm. Bạn có thể xây dựng bao nhiêu tham số tùy ý và lựa chọn kiểu dữ liệu cho từng tham số.
- code : Các câu lệnh bên trong của hàm

Ví dụ 1: Hàm có kiểu trả về là int, có 3 tham số là a, b, c:

```
int tong(int a, int b, int c){
    int sum = a + b + c;
    return sum;
}
```

Ví dụ 2 : Hàm có kiểu trả về là void, có 3 tham số là a kiểu int, b kiểu long long, c kiểu double:

```
void display(int a, long long b, double c){
    printf("%d %lld %.2lf\n", a, b, c);
}
```

2. 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 bạn gọi hàm thì các câu lệnh trong hàm sẽ được thực hiện.

Hình 1.2:Lời gọi hàm

Ví dụ 1:

```c
#include <stdio.h>

void greet(){
    printf("Hello 28tech !\n");
    printf("blog.28tech.com.vn\n");
}

int main(){
    printf("Before\n");
    greet(); // Lời gọi hàm
    printf("After\n");
    return 0;
}
```

Output:

```
Before
Hello 28tech !
blog.28tech.com.vn
After
```

Giải thích :

1. Hàm main thực hiện câu lệnh đầu tiên in ra "Before"
2. Câu lệnh thứ 2 trong main là lời gọi hàm greet(), chương trình tiến hành nhảy vào bên trong hàm greet() và thực hiện lần lượt 2 câu lệnh in ra "Hello 28tech !" và "blog.28tech.com.vn"
3. Sau khi thực hiện xong 2 câu lệnh thì hàm greet() kết thúc tương đương câu lệnh thứ 2 trong main thực hiện xong
4. Hàm main tiếp tục thực hiện câu lệnh thứ 3 in ra "After" sau đó kết thúc chương trình

3. Đối Số Và Tham Số

Tham số (Parameter) hay tham số hình thức là các thành phần khi bạn xây dựng hàm, xem xét ví dụ dưới đây thì a, b, c sẽ được gọi là tham số

Đối số (Argument) hay tham số chính thức là các giá trị bạn truyền vào cho hàm khi gọi hàm, xem xét ví dụ dưới đây thì m, n, p được gọi là đối số

Khi bạn gọi hàm thì lần lượt giá trị của các đối số sẽ được gán cho tham số, trong ví dụ dưới thì m được gán cho a, n được gán cho b, p được gán cho c. Những gì thay đổi trên tham số sẽ không có ảnh hưởng gì tới đối số

Chú ý: Kiểu dữ liệu của đối số và tham số nên trùng nhau hoặc của tham số nên là kiểu dữ liệu lớn hơn kiểu dữ liệu của đối số. Ví dụ bạn xây dựng 1 hàm có tham số là long long thì nó có thể áp dụng với 1 số int nhưng ngược lại thì không.

Ví dụ 1:

```c
#include <stdio.h>

void display(int a, int b, int c){
    printf("%d %d %d\n", a, b, c);
}

int main(){
    int m = 100, n = 200, p = 300;
    display(m, n, p);
    return 0;
}
```

Output:

```
100 200 300
```

Ví dụ 2:

```c
#include <stdio.h>

void display(int a, int b, int c){
    printf("%d %d %d\n", b, a, c);
}

int main(){
    int x = 30;
    display(100.2, 200.3, x);
    return 0;
}
```

Output:

```
200 100 30
```

Giải thích :

1. a được gán giá trị là 100.2 nhưng do a có kiểu là int nên chỉ lưu được 100
2. b được gán giá trị là 200.3 nhưng do b có kiểu là int nên chỉ lưu được 200
3. c được gán giá trị của x tương tương với 30
4. Câu lệnh in ra b, c, a lần lượt là 200 100 30

Ví dụ 3 : Thay đổi tham số sẽ không ảnh hưởng gì tới đố số

```c
#include <stdio.h>

void thaydoi(int n){
    n += 28;
    printf("%d\n", n);
}

int main(){
    int a = 100;
    thaydoi(a);
    printf("%d\n", a);
    return 0;
}
```

Output:

```
128
100
```

Giải thích:

1. Tham số n được gán giá trị của đối số a nên n = 100
2. n += 28 => n = 128, câu lệnh printf bên trong hàm thay đổi in ra n sẽ là 128
3. Hàm kết thúc, câu lệnh in ra a thì a vẫn là 100

4. Câu Lệnh Return

Khi hàm của bạn thực hiện các chức năng tính toán và mong muốn trả về 1 giá trị cụ thể thì bạn cần câu lệnh return, trong các ví dụ trên thì hàm của mình đều không cần trả về giá trị nào nên mình để kiểu trả về là void.

Giả sử bạn cần viết hàm tính tổng của 3 số nguyên, khi đó hàm sẽ nhận vào tham số là 3 số nguyên và cần trả về tổng 3 số. Vậy bạn cần xác định tổng của 3 số đó sẽ có kiểu là gì để làm kiểu trả về cho hàm và bổ sung thêm câu lệnh return kèm giá trị bạn muốn trả về.

Có thể hiểu đơn giản giá trị đi kèm với return chính là kết quả mà hàm trả về cho bạn khi bạn gọi hàm.

Ví dụ 1 : Hàm tính tổng 3 số nguyên int

```c
#include <stdio.h>

int tong(int a, int b, int c){
    int sum = a + b + c;
    return sum;
}

int main(){
    printf("%d\n", tong(10, 20, 30));
    printf("%d\n", tong(28, 28, 28));
    return 0;
}
```

Output :

```
60
84
```

Ví dụ 2 : Hàm tính giai thừa của số n

```c
#include <stdio.h>

long long factorial(int n){
    long long gt = 1;
    for(int i = 1; i <= n; i++){
        gt *= i;
    }
    return gt;
}

int main(){
    printf("%lld\n", factorial(5));
    printf("%lld\n", factorial(10));
    return 0;
}
```

Output :

```
120
3628800
```

Chú ý :

1. Hàm của bạn sẽ kết thúc ngay khi gặp câu lệnh return và giá trị return đầu tiên đó sẽ được trả về cho hàm
2. Kiểu dữ liệu trả về của hàm với kiểu của biến mà bạn sử dụng trong câu lệnh return cần giống nhau. Ví dụ hàm factorial trả về long long thì khi trả về biến gt thì biến gt cũng nên có kiểu long long, nếu bạn sử dụng gt kiểu int có thể bị tràn dữ liệu, khi đó kiểu trả về là long long cũng không tự khôi phục cho bạn được kết quả đúng.

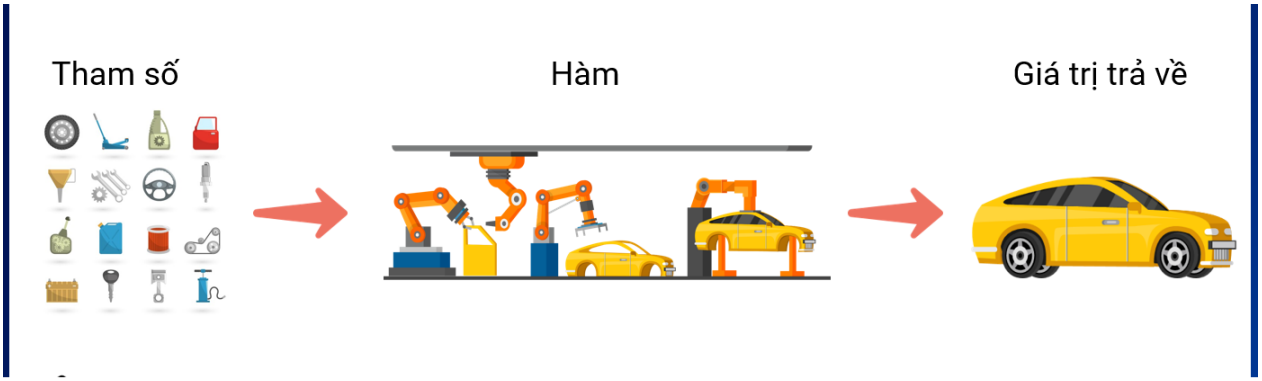
Ví dụ 3:

```c
#include <stdio.h>

int tong(int a, int b){
    int res = a + b;
    return 28; // kết thúc và trả về luôn giá trị 28
    return res; // không được thực hiện
}

int main(){
    printf("%d\n", tong(10, 20));
    printf("%d\n", tong(100, 200));
    return 0;
}
```

Output:

```
28
28
```

5. Chú Ý Khi Xây Dựng Hàm



Hình 1.3: Minh họa về Hàm

1. Hàm này có cần trả về giá trị không, nếu có thì trả về kiểu dữ liệu là gì ?
2. Hàm này có bao nhiêu tham số, các tham số có kiểu dữ liệu là gì ?
3. Hàm của bạn xây dựng đã đủ tổng quát chưa hay quá quá chi tiết và chỉ phù hợp cho 1 bài toán cụ thể

4. Bạn gọi hàm có đúng thứ tự tham số mà mình mong muốn hay không, kiểu dữ liệu của tham số hình thức và tham số chính thức có hợp lí hay không?
5. Nếu hàm của khác void, thì hàm sẽ kết thúc ngay lập tức khi gặp câu lệnh return một giá trị nào đó, còn nếu hàm void mà các bạn muốn kết thúc tại thời điểm nào đó các bạn có thể sử dụng câu lệnh return; là đủ.

Ví dụ bài toán: Tính tổng các ước của N và in ra màn hình

Cách 1:

```c
#include <stdio.h>

void xuly(){
    int n;
    scanf("%d", &n);
    int tong = 0;
    for(int i = 1; i <= n; i++){
        if(n % i == 0){
            tong += i;
        }
    }
    printf("%d\n", tong);
}

int main(){
    xuly();
    return 0;
}
```

Phân tích : Cách làm này không sai, code này sẽ tiến hành nhập, tính tổng và in ra luôn trong hàm xử lý. Tất cả các công việc đều được xử lý bởi hàm xuly(), tuy nhiên cách làm này không phù hợp vì hàm chưa đủ tổng quát.

Ví dụ nếu muốn tính tổng các ước của từng số từ 1 tới n thì hàm này không dùng được, hoặc chỉ muốn lấy ra tổng của 1 số để thực hiện các chức năng khác thì hàm này cũng không sử dụng được.

Cách 2:

```c
#include <stdio.h>

int tonguoc(int n){
    int tong = 0;
    for(int i = 1; i <= n; i++){
        if(n % i == 0){
            tong += i;
        }
    }
    return tong;
}

int main(){
    int n;
    scanf("%d", &n);
    printf("%d\n", tonguoc(n));
    return 0;
}
```

Phân tích : Cách làm này tổng quát hơn, hàm tính tổng ước của bạn có thể sử dụng bất cứ lúc nào bạn cần tính tổng ước của 1 số.

Các thao tác nhập và xuất bạn nên xử lý trong hàm main, hàm chỉ nên có chức năng xử lý nhiệm vụ nhất định.

BÀI 2: CON TRỎ

1. Con Trỏ Và Địa Chỉ

Con trỏ hay biến con trỏ cũng là một biến thông thường nhưng giá trị mà nó lưu lại là địa chỉ của 1 biến khác.

Ví dụ biến kiểu int N trong chương trình sẽ có địa chỉ nhất định trong bộ nhớ, để lưu trữ giá trị địa chỉ này ta cần biến con trỏ kiểu int

Khi khai báo biến con trỏ ta thêm dấu * vào trước tên biến.

Cú pháp khai báo: Kiểu_Dữ_Liệu *Tên_Biến_Con_Trỏ;

Ví dụ :

```c
#include <stdio.h>

int main(){
    //Dấu * thể hiện ptr là con trỏ
    int *ptr; // con trỏ kiểu int
    //Dấu * có thể đặt cạnh tên biến hoặc cạnh kiểu dữ liệu
    long long* ptr2; // con trỏ kiểu long long
    char *ptr3;
    return 0;
}
```

Mỗi biến trong chương trình đều được cấp phát vùng nhớ để lưu trữ giá trị của nó, ví dụ biến int sẽ được cấp phát 4 byte liên tiếp để lưu trữ và lấy địa chỉ của byte đầu tiên làm địa chỉ cho biến.

Để in ra địa chỉ của biến bạn dùng toán tử &, ví dụ &N sẽ cho bạn địa chỉ của biến N.

Code :

```c
#include <stdio.h>

int main(){
    int N = 28;
    long long M = 10000012828;
    printf("Dia chi cua N trong bo nho : %d\n", &N);
    printf("Dia chi cua M trong bo nho : %d\n", &M);
    return 0;
}
```

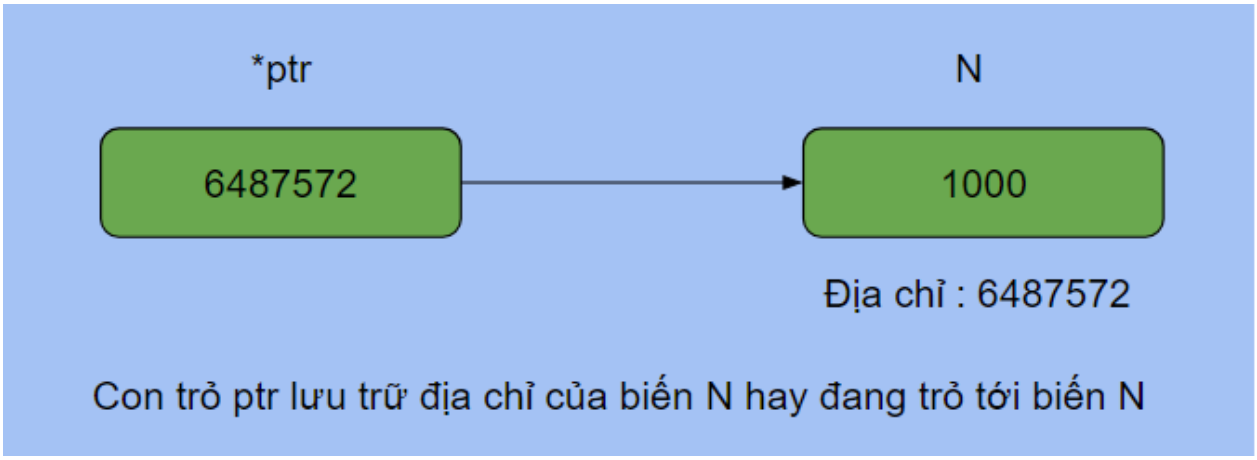
Output :

```
Dia chi cua N trong bo nho : 6487572
Dia chi cua M trong bo nho : 6488592
```

Chú ý : Khi bạn chạy code trên thì địa chỉ của N và M sẽ khác, và mỗi lần bạn chạy có thể sẽ có một địa chỉ khác nhau.

Con trỏ được sinh ra để lưu địa chỉ của biến ở kiểu dữ liệu tương ứng với nó, ví dụ biến con trỏ kiểu int sẽ lưu được địa chỉ của biến int.

Chương trình sau sẽ gán địa chỉ của N cho biến con trỏ ptr, khi đó ta nói con trỏ ptr trỏ tới biến N



Hình 2.1: Ví dụ về con trỏ.

Code:

```c
#include <stdio.h>

int main(){
    int N = 1000;
    printf("Dia chi cua N : %d\n", &N);
    int *ptr;
    //Gán địa chỉ của N cho ptr
    ptr = &N;
    printf("Gia tri cua ptr : %d\n", ptr);
    return 0;
}
```

Output:

```
Dia chi cua N : 6487572
Gia tri cua ptr : 6487572
```

2. Tham Chiếu Và Giải Tham Chiếu

Khi con trỏ ptr trỏ tới hay tham chiếu (reference) tới biến N thì thông qua con trỏ ptr ta có thể truy xuất, thay đổi giá trị của biến N mà không cần dùng N.

Để truy xuất tới giá trị của biến mà con trỏ đang trỏ tới ta dùng toán tử giải tham chiếu * (dereference)

Sau khi con trỏ ptr trỏ tới biến N thì N và *ptr là một, đều truy xuất đến ô nhớ mà N đang chiếm để lấy giá trị tại ô nhớ đó.

Lưu ý là bạn cần phân biệt dấu * khi khai báo con trỏ ptr và dấu * khi giải tham chiếu con trỏ ptr. Dấu * khi khai báo thể hiện ptr là một con trỏ còn dấu * trước ptr ở những câu lệnh sau là toán tử giải tham chiếu.

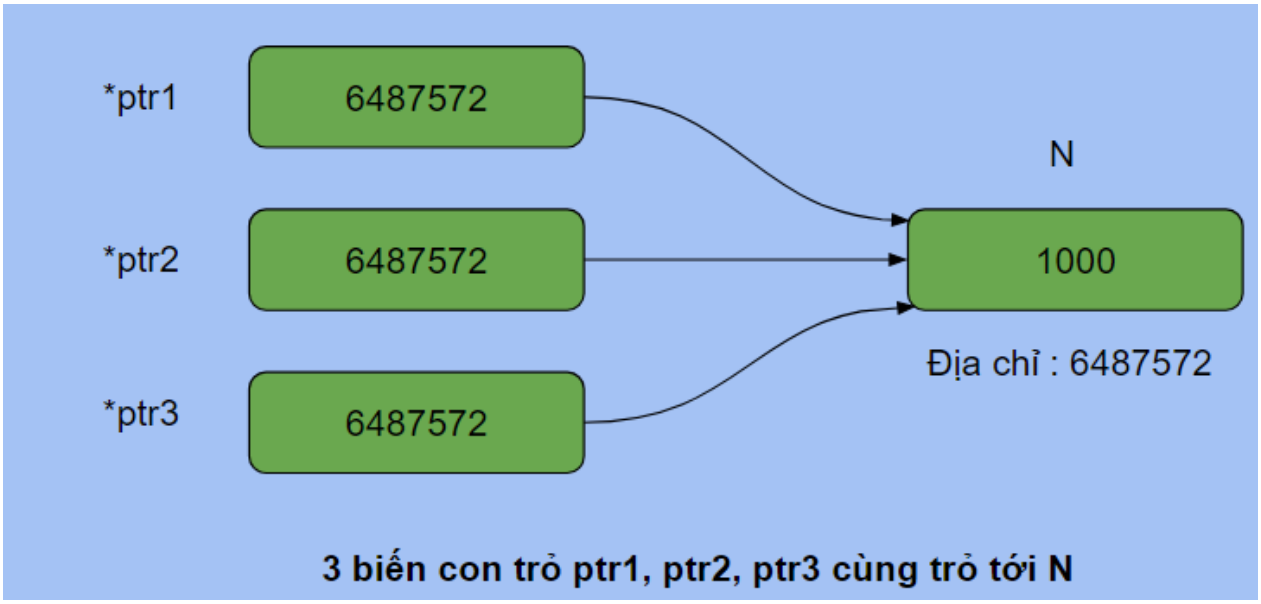
```c
#include <stdio.h>

int main(){
    int N = 1000;
    printf("Dia chi cua N : %d\n", &N);
    int *ptr = &N; // ptr trỏ tới N
    printf("Gia tri cua ptr : %d\n", ptr);
    //Toán tử giải tham chiếu
    printf("Gia tri cua bien ma con tro ptr tro toi : %d\n", *ptr);
    //Thay đổi N bằng ptr, *ptr và N là một
    *ptr = 280;
    printf("Gia tri cua N sau thay doi : %d %d\n", N, *ptr);
    return 0;
}
```

Output :

```
Dia chi cua N : 6487572
Gia tri cua ptr : 6487572
Gia tri cua bien ma con tro ptr tro toi : 1000
Gia tri cua N sau thay doi : 280 280
```

Một biến có thể được trỏ tới bởi nhiều con trỏ, khi đó bạn có thể thông qua bất cứ 1 con trỏ nào để thay đổi giá trị của biến mà nó đang trỏ tới.

Trong ví dụ dưới đây 3 biến con trỏ ptr1, ptr2, ptr3 đều trỏ tới N nên *ptr1, *ptr2, *ptr3 và N đều có giá trị giống nhau.



Hình 2.2: 3 biến con trỏ

Code :

```c
#include <stdio.h>

int main(){
    int N = 1000;
    int *ptr1 = &N; // ptr1 trỏ tới N
    int *ptr2 = &N; // ptr2 trỏ tới N
    int *ptr3 = ptr1; // Gán giá trị của ptr1 cho ptr3, tương tự gán &N cho ptr3
    printf("Gia tri cua 3 con tro : %d %d %d\n", ptr1, ptr2, ptr3);
    *ptr1 = 100; // N = 100
    printf("Gia tri cua N : %d\n", N);
    *ptr2 = 200; // N = 200
    printf("Gia tri cua N : %d\n", N);
    *ptr3 = 300; // N = 300
    printf("%d %d %d %d\n", *ptr1, *ptr2, *ptr3, N);
    return 0;
}
```

Output :

```
Gia tri cua 3 con tro : 6487572 6487572 6487572
Gia tri cua N : 100
Gia tri cua N : 200
300 300 300 300
```

3. Hàm Và Con Trỏ

Con trỏ làm tham số cho hàm.

Để thay đổi giá trị của 1 biến sau khi hàm kết thúc thì việc truyền giá trị là không hợp lý, thay vì đó bạn hãy sử dụng con trỏ với mục đích là thay đổi giá trị của biến thông qua con trỏ.

Khi hàm có tham số là một con trỏ thì khi gọi hàm bạn cần truyền vào một giá trị phù hợp, có thể là địa chỉ của 1 biến hoặc một con trỏ khác.

Ví dụ 1: Thay đổi giá trị của biến sau khi hàm kết thúc

Code :

```c
#include <stdio.h>

//*x ở đây là một con trỏ
void change(int *x){
    printf("Gia tri cua con tro x : %d\n", x);
    printf("Gia tri cua bien ma x đang tro toi : %d\n", *x);
    //Đây là tham chiếu tới giá trị của biến
    //mà con trỏ x đang trỏ tới để thay đổi nó thành 1000
    *x = 1000;
}

int main(){
    int N = 28;
    printf("Dia chi cua N : %d\n", &N);
    change(&N); // truyền địa chỉ của N vào
    printf("Gia tri cua N : %d\n", N);
    return 0;
}
```

Output :

```
Dia chi cua N : 6487580
Gia tri cua con tro x : 6487580
Gia tri cua bien ma x đang tro toi : 28
Gia tri cua N : 1000
```

Giải thích :

Hàm change có tham số là một con trỏ kiểu int có tên là x, trong main bạn gọi change và truyền địa chỉ của N vào.

Khi đó x sẽ được gán giá trị là địa chỉ của N, trong hàm change thì câu lệnh *x = 1000 sẽ truy xuất tới ô nhớ mà x đang trỏ tới và gán giá trị 1000. Mà x lại đang tham chiếu tới N nên giá trị của N đã bị thay đổi.

Sau khi hàm change kết thúc thì giá trị của N đã bị thay đổi thực sự.

Ví dụ 2 : Hoán đổi giá trị của 2 biến

```c
#include <stdio.h>

void swap(int *x, int *y){
    int tmp = *x;
    *x = *y;
    *y = tmp;
}

int main(){
    int a = 100, b = 200;
    swap(&a, &b);
    printf("%d %d\n", a, b);
    return 0;
}
```

Output :

```
200 100
```

Ví dụ 3 : Hàm trả về con trỏ

```c
#include <stdio.h>

int *convert(int *x, int *y){
    x = y;
    return x;
}

int main(){
    int N = 28, M = 56;
    int *ptr1 = &N;
    int *ptr2 = &M;
    printf("Truoc khi goi ham : \n");
    printf("Gia tri cua ptr1 : %d\n", ptr1);
    printf("Gia tri cua ptr2 : %d\n", ptr2);
    ptr1 = convert(ptr1, ptr2);
    printf("Sau khi goi ham : \n");
    printf("Gia tri cua ptr1 : %d\n", ptr1);
    printf("Gia tri cua ptr2 : %d\n", ptr2);
    return 0;
}
```

Output :

```
Truoc khi goi ham :
Gia tri cua ptr1 : 6487576
Gia tri cua ptr2 : 6487580
Sau khi goi ham :
Gia tri cua ptr1 : 6487580
Gia tri cua ptr2 : 6487580
```

BÀI 3: CON TRỎ MẢNG

1. Con Trỏ Và Mảng

Giá trị của mảng chính là địa chỉ phần tử đầu tiên trong mảng, ví dụ mảng A[] = {3, 8, 4, 2, 9} thì A[] có giá trị là địa chỉ của A[0]

Tên mảng chính là một hằng con trỏ và bạn không thể thay đổi được

Mảng 1 chiều A[] thì con trỏ trỏ tới phần tử A[i] là A + i, vậy (A + i) tương đương với &A[i] hay địa chỉ của A[i] và *(A + i) tương đương với A[i]

a	3	8	4	2	9
Chỉ số	0	1	2	3	4
Địa chỉ	1004	1008	1012	1016	1020
Con trỏ	a + 0	a + 1	a + 2	a + 3	a + 4

Ví dụ 1 : Sử dụng mảng thông qua con trỏ

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

int main(){
    int n = 5;
    int a[5] = {3, 8, 4, 2, 9};
    printf("Gia tri cua a : %d\n", a);
    for(int i = 0; i < n; i++){
        printf("Dia chi cua a[%d] : %d\n", i, a + i); // a + i <=> &a[i]
    }
    printf("Mang a : ");
    for(int i = 0; i < n; i++){
        printf("%d ", *(a + i));
    }
    return 0;
}
```

Output :

```
Gia tri cua a : 6487520
Dia chi cua a[0] : 6487520
Dia chi cua a[1] : 6487524
Dia chi cua a[2] : 6487528
Dia chi cua a[3] : 6487532
Dia chi cua a[4] : 6487536
Mang a : 3 8 4 2 9
```

Trong mảng 1 chiều thì bạn có thể sử dụng một con trỏ để trỏ tới các phần tử trong mảng và sử dụng các toán tử ++, -- hoặc +, - để di chuyển con trỏ qua lại các ô nhớ trong mảng.

Lưu ý giúp mình rằng bạn có thể sử dụng một con trỏ khác để di chuyển qua lại trong mảng nhưng không thể di chuyển hằng con trỏ mảng.

Ví dụ 2 :

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

int main(){
    int n = 5;
    int a[5] = {3, 8, 4, 2, 9};
    int *ptr = &a[0];
    for(int i = 0; i < n; i++){
        printf("%d ", *(ptr + i));
    }
    printf("\n");
    ++ptr; // => a[1]
    printf("%d\n", *ptr);
    ptr+= 2; // a[3]
    printf("%d\n", *ptr);
    --ptr; // a[2]
    printf("%d\n", *ptr);
    return 0;
}
```

Output :

```
3 8 4 2 9
8
2
4
```

BÀI 4: MẢNG CON TRỎ

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.

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 1: Một mảng các con trỏ tới số nguyên:

```
#include <stdio.h>

int main (){

            int arr[]={1,2,3,4,5};

            int *a[5]; // khai bao mang con tro gom 5 phan tu

// cap phat bo nho cho cac con trong trong mang a bang cach gan cho dia
chi cua mang arr

        for(int i=0; i<5; i++){

                a[i]= &arr[i];

        }

//duyet mang con tro

        printf("Mang co gia tri la: \n");

        for(int i=0; i<5; i++){

                printf("%d\t",*a[i]);

        }

    free(a);

    return 0;

}
```

Ví dụ 2: 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 bao mang
con tro gom 5 phan tu

//xuat mang

        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 3: Một mảng trỏ tới các cấu trúc:

```
#include <string.h>

// tao struct book

typedef struct {

        char ten[50];

        float gia;

} Book;

const int MAX = 3;

int main() {

        Book *book[MAX]; //khai bao mang con tro

        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);
        }
//giai phong bo nho
        for (int i = 0; i < MAX; i++) {
                free(book[i]);
        }

  return 0;
}
```

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

1. Khái niệm:

Con trỏ trong C là một biển 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.

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>

// Dinh nghia ham

void hello() {

    printf("Hello World");}

// Chuong trinh chinh

int main() {

    void (*n)() = &hello; // Khai bao mot con tro ham

    (*n)(); // Goi ham bang con tro ham

    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;

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 1 : Con trỏ hàm có đối số:

```
#include <stdio.h>

// Ham cong hai so

int addition(int a, int b) {

    return a + b; }

int main() {

    int (*n)(int, int) = addition; // con tro ham

    int x = 10, y = 20;

    int k = (*n)(x, y); // goi ham thong qua con tro

    printf("Cong x: %d va y: %d = %d", x, y, k);

    return 0; }
```

Ví dụ 2: Con trỏ đến hàm với các đối số con trỏ:

```
#include <stdio.h>

// Ham hoan doi gia tri cua hai bien

void swap(int *a, int *b) {

    int k;

    k = *a;

    *a = *b;

    *b = k; }

int main() {

    void (*n)(int *, int *) = swap; // con tro ham

    int x = 10, y = 20;

    printf("Gia tri x: %d va y: %d truoc khi swap\n", x, y);

    (*n)(&x, &y); // goi ham thong qua con tro

    printf("Gia tri x: %d va y: %d sau khi swap", x, y);

    return 0; }
```

4. Màng các con trỏ hàm

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

Ví dụ: Mảng các con trỏ hàm:

```
// Ham chia
float chia(int so1, int so2) {
    return (float)so1 / so2; }
// Ham cong
float cong(int so1, int so2) {
    return so1 + so2; }
// Ham tru
float tru(int so1, int so2) {
    return so1 - so2; }
// Ham nhan
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 la cong, 2 la tru, 3 la nhan, 4 la 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

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().

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 1:

```
#include <stdio.h>
#include <stdlib.h>
int main(){
    //Cap phat vung nho tuong duong mang 100 ptu int,sizeof(int) = 4
    int *a = (int*)malloc(100 * sizeof(int)); //Cap phat vung nho tuong duong mang 1000 phan tu 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 2: 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; }
```

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; }
```

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 :

```c
#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; }
```

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 :

```c
#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

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,...).

2. Khởi tạo tệp:

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;

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.

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.

2.4. Đổi tên một tệp 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)

3. Các thao tác khi làm việc với tệp:

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 bao con tro tep *file
    char buffer[100];
    file = fopen("example.txt", "w"); //Tao (mo) tep example.txt de ghi
    if (file == NULL) {
        printf("Khong the mo tep de ghi!\n");
return 1;
```

```
        }
    printf("Nhap noi dung: ");
        fgets(buffer, sizeof(buffer), stdin); // Nhap noi dung tu ban
phim vao tep
    fprintf(file, "%s", buffer); // Ghi noi dung vao tep voi ham
fprintf()
    fclose(file); // Dong tep
    file = fopen("example.txt", "r"); // Mo tep example.txt de doc
    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)) {// Doc noi dung tu tep
        printf("%s", buffer); //Hien thi noi dung ra man hinh
    }
    fclose(file); //Dong tep
    return 0;
}
```

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

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;
```

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.

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.

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.

5. Khởi tạo giá trị ban đầu cho cấu trúc:

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};
```

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};
```

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:

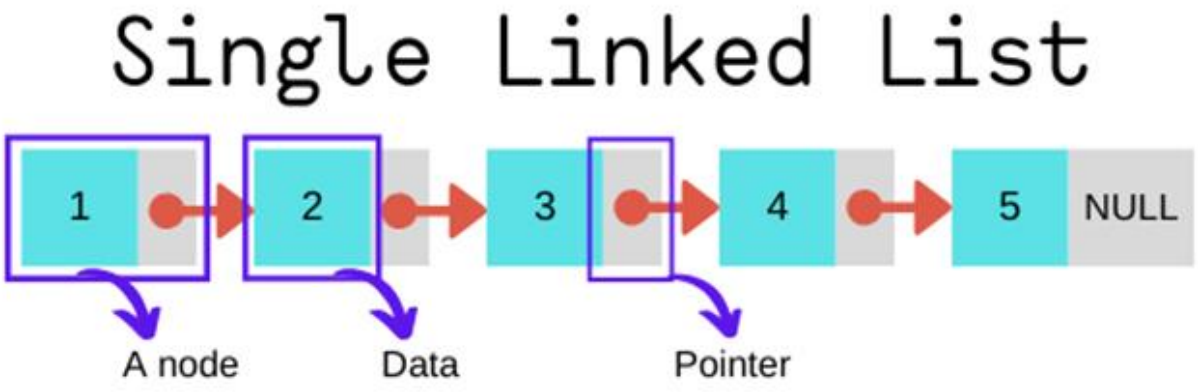
```
nv.ngaySinh.ngay = 16;
nv.ngaySinh.thang = 07;
```

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.

BÀI 9: DANH SÁCH LIÊN KẾT

1. Khái niệm:

Khái niệm danh sách liên kết đơn đóng vai trò quan trọng trong các thao tác lập trình. Ứng dụng nó mang lại là vô cùng lớn. Vì thế bạn đọc nếu muốn tìm hiểu ngành này thì không thể bỏ qua các kiến thức về danh sách liên kết đơn. Bài viết sau từ Teky sẽ giúp bạn đọc tìm hiểu thêm thông tin về chủ đề này như định nghĩa, đặc điểm và cách cài đặt danh sách liên kết đơn.



Hình 9.1: Ví dụ về danh sách liên kết.

2. Đặc điểm của danh sách liên kết đơn:

2.1. Tính cấp phát dữ liệu động:

Trong khi chạy chương trình, Single link list C++ sẽ được cấp phát bộ nhớ. Các phần tử được lưu trữ một cách ngẫu nhiên trong RAM. Khi thêm hoặc bớt phần tử, kích thước của danh sách cũng sẽ thay đổi. Kích thước tối đa của Single linked list trong c++ phụ thuộc vào bộ nhớ khả dụng của RAM.

Ví dụ: Cấp phát động cho từng node khi cần:

```
Node* head = new Node{10, NULL};
head->next = new Node{20, NULL};
```

2.2. Tính liên kết của phần tử đầu và phần tử đứng sau:

Vì có sự liên kết giữa hai phần đứng trước đứng sau nên chỉ cần nắm được thông tin của phần tử đầu tiên và phần tử cuối cùng là người dùng có thể dễ dàng quản lý được cả danh sách. Tuy nhiên nếu muốn truy cập đến một vị trí bất kỳ thì phải tiến hành duyệt từ

đầu đến phần tử đó. Ngoài ra, trong danh sách liên kết đơn C++ cũng chỉ cho phép người dùng tìm kiếm tuyến tính duy nhất 1 phân tử.

3. Cách cài đặt danh sách liên kết đơn:

3.1. Tạo Node:

Một danh sách được tạo lên từ nhiều node. Do vậy ta sẽ đi từ bước tạo node trước. Như đã nói ở trên, một node bao gồm 2 phần là thành phần liên kết và thành phần dữ liệu. Đối với thành phần dữ liệu, bạn có thể tự tạo lên dữ liệu theo ý muốn (class) hoặc sử dụng dữ liệu có sẵn (struct). Còn phần liên kết thì đương nhiên sẽ là con trỏ. Con trỏ này trỏ từ node trước đến node liền kề phía sau.

Với phần ví dụ tạo node này, ta sẽ sử dụng int cho phần dữ liệu như sau:

```
struct Node
{
int data;
Node* next;
};
```

Để tạo thêm 1 node mới, ta sẽ tiến hành khởi tạo giá trị ban đầu và trả địa chỉ về cho node được cấp phát.

```
Node* CreateNode(int init_data)
{
Node* node = new Node;
node->data = init_data;
node->next = NULL;
```

Node vừa được tạo chưa thêm vào danh sách nên chưa liên kết với phần tử nào cả. Do đó nên phần liên kết gán bằng NULL.

PHẦN B: ỨNG DỤNG XÂY DỰNG HỆ THỐNG QUẢN LÝ SINH VIÊN.

1. **Các kiến thức áp dụng:**
   1. Hàm.
   2. Con trỏ.
   3. Cấp phát động.
   4. Xử lí tệp.
   5. Kiểu cấu trúc.
   6. Danh sách liên kết đơn.
2. **Mô tả hệ thống quản lý sinh viên:**

2.1. Các thư viện dã sử dụng:

Tên thư viện	Mô tả ngắn gọn	Mục đích sử dụng
stdio.h	- Cung cấp các hàm nhập xuất chuẩn như: printf(), scanf(), fgets(), ...	- Dùng để in thông tin ra màn hình hoặc đọc dữ liệu từ bàn phím/tệp.
stdlib.h	- Cung cấp các hàm quản lý bộ nhớ và xử lý hệ thống như: malloc(), free(), exit(), ...	- Dùng để cấp phát động cho node (malloc()), giải phóng bộ nhớ (free()).
string.h	- Cung cấp các hàm xử lý chuỗi như: strlen(), strcmp(), strcpy(), ...	- Dùng để so sánh, sao chép, đo độ dài, loại bỏ ký tự xuống dòng ('\n') trong chuỗi.
ctype.h	- Cung cấp các hàm xử lý ký tự như: toupper(), tolower(), isalpha(), ...	- Dùng toupper() để viết hoa chữ cái đầu tên sinh viên, phục vụ sắp xếp theo Alphabet.

2.2. Chức năng:

1.Thêm sinh viên.

- Thêm N sinh viên. (N nhập từ bàn phím)
- Nhập thông tin sinh viên. (gồm: MSSV, họ và tên, điểm tổng kết)

2.Hiển thị toàn bộ sinh viên ra màn hình.

3.Tìm kiếm sinh viên theo mã số sinh viên.

4.Xóa sinh viên theo mã số sinh viên.

5.Tìm sinh viên có điểm trung bình cao nhất.

6.Tách danh sách sinh viên Đạt (điểm>= 5) và Không Đạt (điểm < 5).

7.Tìm sinh viên theo điểm sinh viên từ N đến M (N,M nhập từ bàn phím)

8.Cập nhật sinh viên theo mã số sinh viên.

- Thay đổi họ và tên của sinh viên và điểm của sinh viên.

9.Sắp xếp sinh viên theo điểm giảm dần.

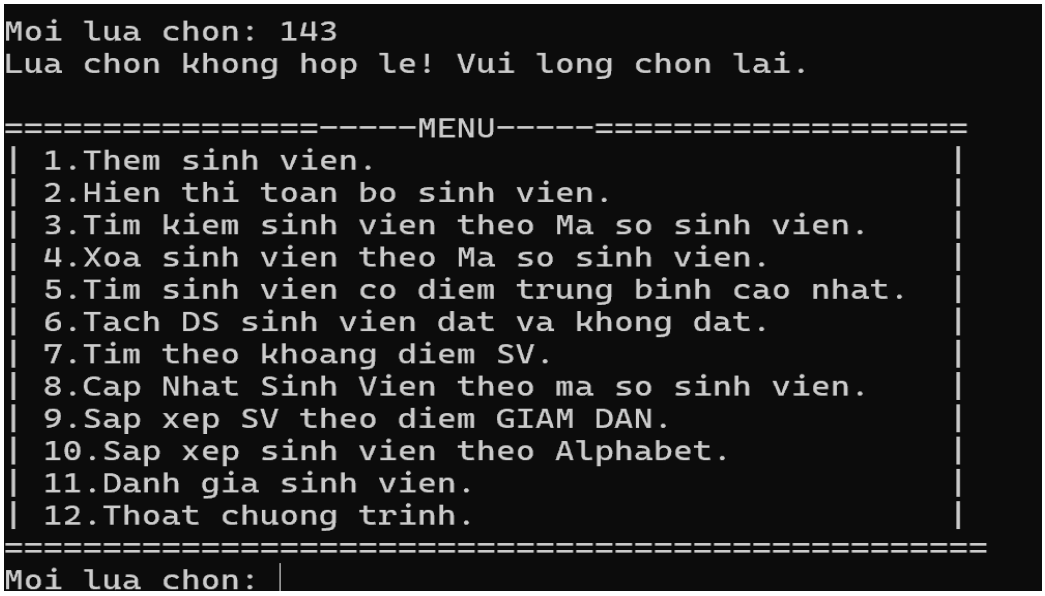
10.Sắp xếp sinh viên theo bảng chữ cái Alphabet.

11.Danh gia sinh vien.

12.Thoát khỏi chương trình.

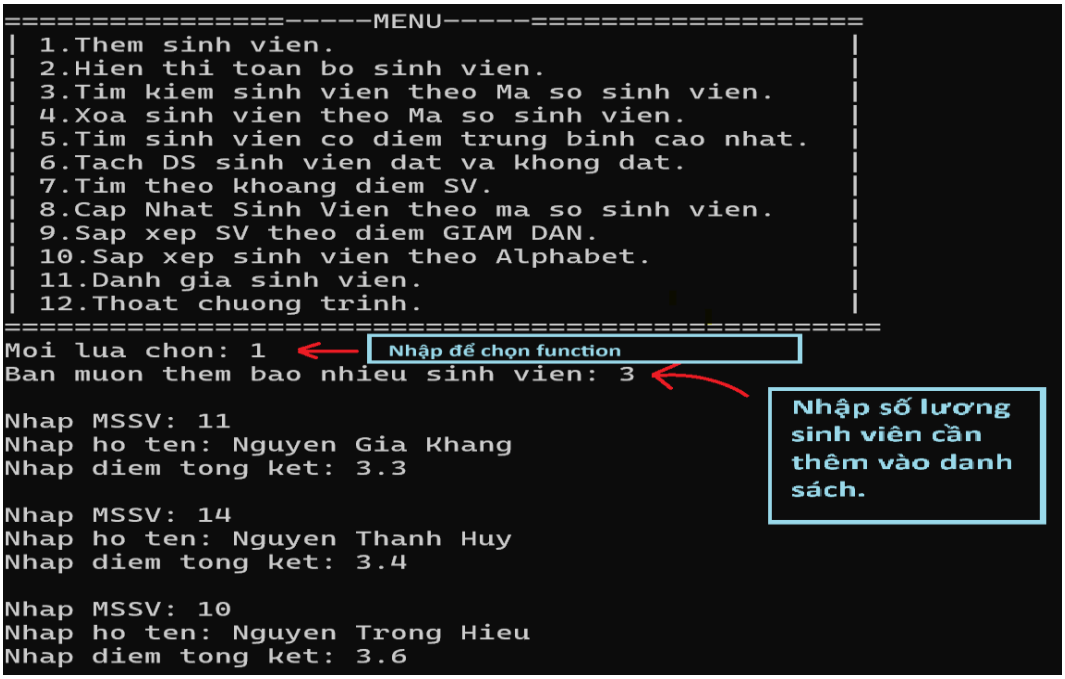
2.3. phân tích chức năng:

Khi chương trình bắt đầu, máy sẽ hỏi người dùng muốn chọn chức năng nào để là việc, người dùng sẽ phải nhập các số từ 1 đến 12 như trong MENU. Nếu như nhập vào một số khác các số thứ tự đó thì sẽ được yêu cầu nhập lại như hình:



2.3.1 Thêm sinh viên:

Chức năng thêm sinh viên là một chức năng quan trọng và được dùng đầu tiên nếu người dùng muốn nhập thông tin các sinh viên để quản lí. Dưới đây là phân tích các thành phần chính, chức năng và mục đích của chứa năng này:



Mục đích:

- Ở đây là hình ảnh khi ta chọn thêm sinh viên vào danh sách quản lí, sau đó nhập các thông tin cần thiết của mỗi sinh viên, bao gồm: họ và tên, MSSV, điểm tổng kết.

Chức năng:

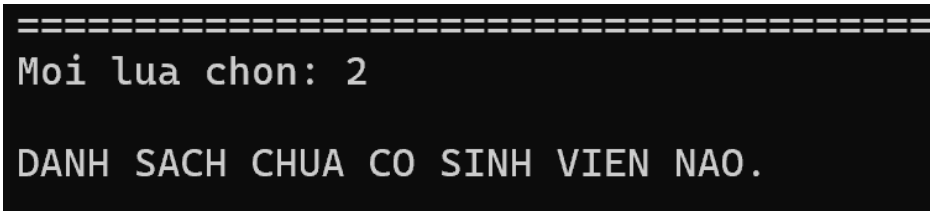
- Thêm sinh viên vào cuối danh sách quản lí.

```
61  //Ham nhap thong tin sinh vien vao DSLK:
62  void NhapSV(SinhVien **a,int n){
```

Gồm các hàm con: Them().

2.3.2 Hiện thị toàn bộ sinh viên:

Hiển thị ra màn hình toàn bộ sinh viên đã nhập thông tin. Nếu trong danh sách quản lí không có sinh viên nào thì chương trình báo như hình 2.2.2:



Mục đích: Cho phép người dùng xem danh sách sinh viên sau khi thực hiện các thao tác quản lí khác.

Chức năng: In ra các sinh viên có trong danh sách ở thời điểm hiện tại.

```
void inDanhSachSinhVien(SinhVien* head)
```

Gồm các hàm con: in1SinhVien().

2.3.3 Tìm kiếm sinh viên theo mã số sinh viên

Mục đích: Giúp người dùng tìm ra sinh viên có mã số được nhập như bàn phím rồi sau đó in ra mà hình thông tin của sinh viên đó.

Chức năng: In ra màn hình sinh viên có mã đó, trường hợp MSSV nhập vào không có trong danh sách thì chương trình sẽ thông báo nhập lại và nhập cho đến khi có sinh viên có MSSV hợp lệ thì in ra.

```
//Ham tim sinh vien theo ma sinh vien:
void TimSV(SinhVien *a){
```

```
while (tmp != NULL) {
      if (strcmp(tim,tmp->mssv)==0) {
      in1SinhVien(tmp);
      return;
      }
      tmp = tmp->next;
  }
  printf("Khong co sinh vien nay.\n");
```

-Duyệt qua từng node trong danh sách đến khi tìm được sinh viên có mã số đó. Nếu không có sinh viên đó thì in ra dòng cuối cùng.

Gồm các hàm con: in1SinhVien().

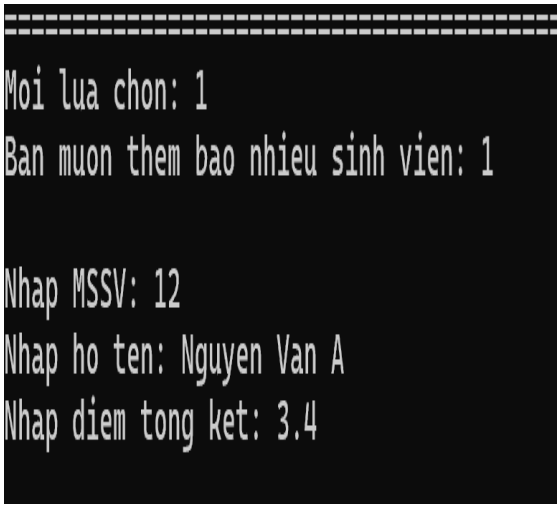
2.3.4 Xóa sinh viên theo mã số sinh viên

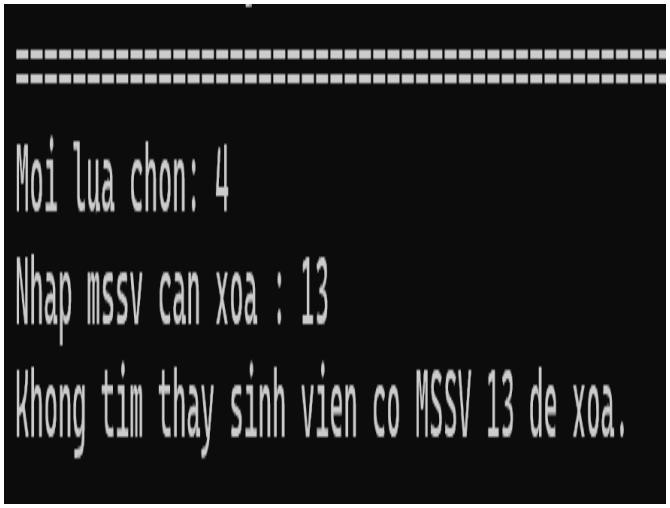
Mục đích: Dùng để xóa sinh viên khỏi danh sách đang quản lí.

Chức năng: Dò đến sinh viên có mã cần thiết để xóa, nếu người dùng nhập vào một MSSV không tồn tại trong danh sách thì chương trình sẽ thông báo như hình 2.2.4.1:

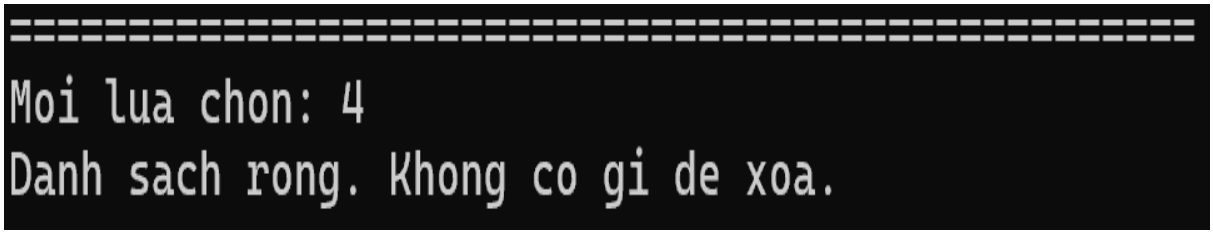
```
while (tmp != NULL && strcmp(tmp->mssv, xoa) != 0) {
        truoc = tmp;
        tmp = tmp->next;
    }

    if (tmp == NULL) {
        printf("Khong tim thay sinh vien co MSSV %s de xoa.\n", xoa);
        return;
    }

    truoc->next = tmp->next;
    free(tmp);
```

-Duyệt qua danh sách, lưu lại node cũ(temp hiện tại). Quá trình này được lặp lại cho đến khi gặp được node cần xóa. “truoc” sẽ lưu node trước node cần xóa. Sau đó cập nhật phần link và giải phóng “tmp” bằng hàm free().





Hoặc nếu như danh sách lúc đó rỗng thì sẽ có thông báo như:



2.3.5 Tìm sinh viên có điểm trung bình cao nhất

Mục đích: Giúp người dùng tìm ra sinh viên có điểm cao nhất trong danh sách.

Chức năng: Duyệt qua toàn bộ danh sách, so sánh điểm và lưu lại sinh viên có điểm cao nhất.

```
//Ham tim sinh vien co diem cao nhat:
void timSinhVienDiemCaoNhat(SinhVien* a) {
```

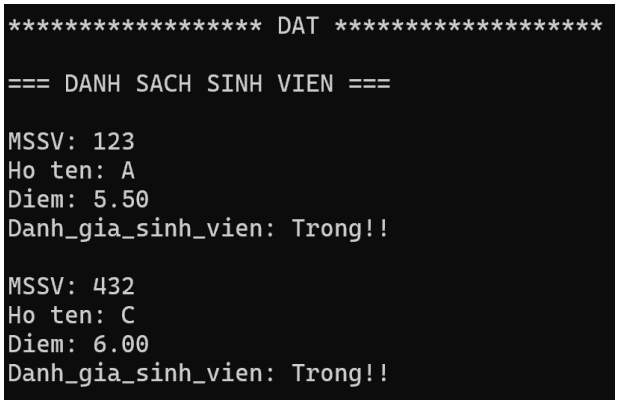
Gồm các hàm con: in1SinhVien().

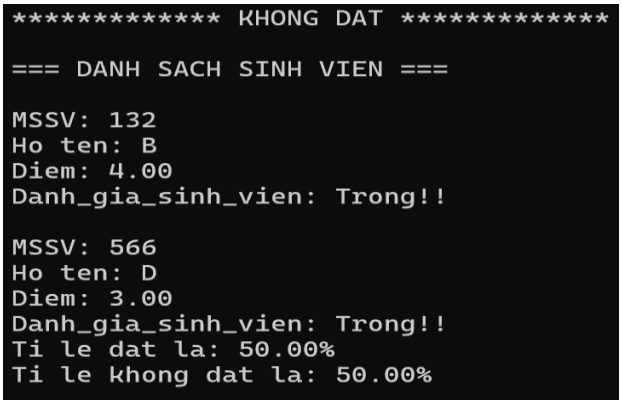
2.3.6 Tách danh sách sinh viên Đạt và Không Đạt

Mục đích: Tách danh sách ban đầu thành hai danh sách con theo điều kiện điểm.

Chức năng: Các sinh viên có điểm >= 5 sẽ vào danh sách 'Đạt', còn lại vào danh sách 'Không Đạt'. Sau đó in ra danh sách đạt, chưa đạt và tỉ lệ đạt và chưa đạt ra màn hình như

```
void tachDanhSach(SinhVien* head, SinhVien** dat, SinhVien** khongdat)
```





Gồm các hàm con: demSinhVien(), inDanhSachSinhVien().

2.3.7 Tìm sinh viên theo khoảng điểm

Mục đích: Cho phép người dùng tìm các sinh viên có điểm trong khoảng từ N đến M (do người dùng nhập vào) để tìm sinh viên trên khoảng điểm đó.

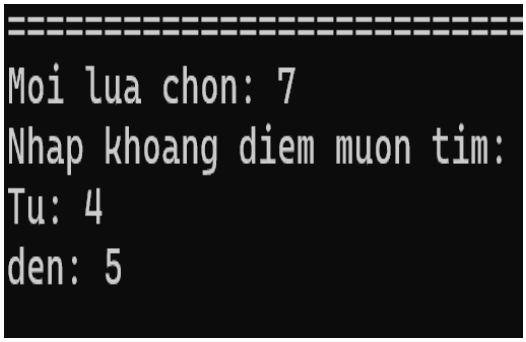
Chức năng: Duyệt toàn bộ danh sách và kiểm tra điều kiện điểm nằm trong khoảng.

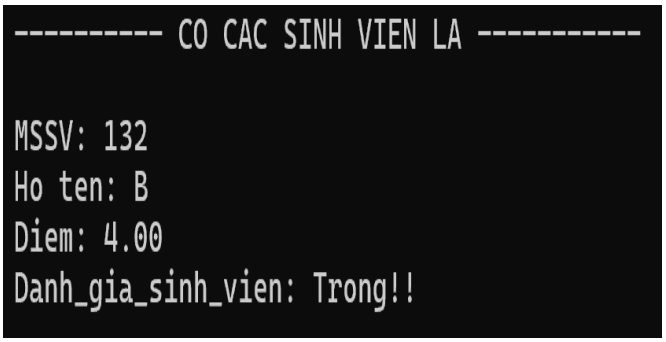
```
//Ham tim sinh vien trong khoang diem tu c den d:
void timTheoDiemSV(SinhVien *a){
```

```
if ( minMax(c,d,1) < minDiem(a) || minMax(c,d,0) > maxDiem(a) )
            dem = 0;

              if (dem){
                printf("\n---------- CO CAC SINH VIEN LA ----------\n");
                while (a != NULL){
               if (a->diem >= minMax(c,d,0) && a->diem <= minMax(c,d,1))
               in1SinhVien(a);
               a = a->next;
                 }
                    } else printf("KHONG CO SINH VIEN NAO CA");
```

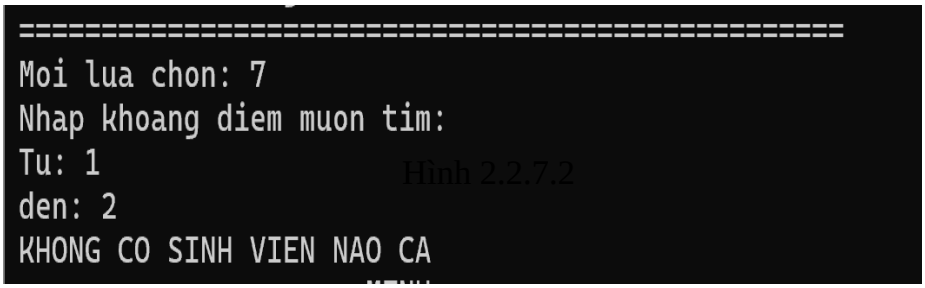
-Sau khi người dùng nhập vào khoảng điểm cần tìm, sẽ kiểm tra điều kiện if đầu tiên. Nếu có điểm của sinh viên nào tồn tại thì chuyển đến thực hiện khối lệnh if thứ 2.

-Các hàm như: minDiem(), maxDiem(), minMax() lần lượt là các hàm tìm điểm trung bình nhỏ nhất, lớn nhất trong danh sách sinh viên và hàm trả về giá trị min hoặc max khi so sánh hai số với nhau.





Hoặc nếu không có sinh viên nào thì sẽ thông báo như thế này ra màn hình:



Hình 2.2.7.2

Gồm các hàm con: timTheoDiemSV(), minMax(), minDiem(), maxDiem().

2.3.8 Cập nhật sinh viên theo MSSV

Mục đích: Thay đổi họ tên và điểm tổng kết của sinh viên có MSSV được nhập vào.

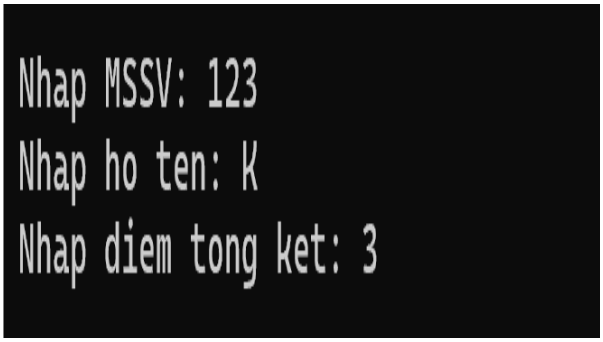
Chức năng: Tìm sinh viên theo MSSV và cập nhật thông tin mới.

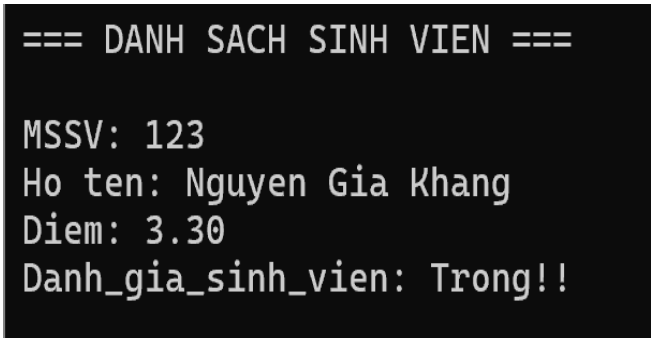
```
//Ham cap nhat sinh vien theo MSSV:
void capNhatSinhVien(SinhVien* head)
```

```
//Cap nhat CHUcaiDAU sau khi doi ten
        temp->CHUcaiDAU = toupper(chuCaiDau(temp->hoten));
```

-Tương tự như các hàm tìm sinh viên theo MSSV hay tìm sinh viên có MSSV nào đó để xóa ra khỏi danh sách. Tuy nhiên, sau khi chúng em cập nhật lại tên, nếu không cập nhật lại chữ các đầu của tên sinh viên thì nó sẽ ảnh hưởng đến việc sắp xếp theo Alphabet sau

này. Vậy nên nhóm đã chú ý thêm vào hàm đoạn code này. toupper() là hàm có sẵn trong thư viện “ctype.h”





2.3.9 Sắp xếp sinh viên theo điểm giảm dần

Mục đích: Sắp xếp danh sách sinh viên theo thứ tự điểm từ cao xuống thấp.

Chức năng: So sánh và hoán đổi thông tin giữa các sinh viên để sắp xếp. Bên trái là danh sách sinh viên trước khi sắp xếp còn bên phải là sau khi đã được sắp xếp giảm dần theo điểm trung bình.

```
//Ham sap xep theo diem giam dan:
void sapXepTheoDiemGiamDan(SinhVien* head)
```

```
for (i = head; i != NULL; i = i->next) {
      for (j = i->next; j != NULL; j = j->next) {
         if (i->diem < j->diem) {
            swapSinhVien(i,j);
         }
      }
   }
```

Ở đây, chúng em sử dụng vòng lặp lồng nhau với hai “for”. Đoạn code này áp dụng thuật toán sắp xếp nổi bọt (Bubble Sort) để sắp xếp.

Gồm các hàm con: sapXepTheoDiemGiamDan(), swapSinhVien().

2.3.10 Sắp xếp sinh viên theo bảng chữ cái Alphabet

Mục đích: Sắp xếp sinh viên theo thứ tự bang chu cái bằng chữ cái.

Chức năng: Lấy chữ cái đầu tiên từ tên sinh viên, sắp xếp tăng dần theo bảng chữ cái.

Bên trái là lúc chưa sắp xếp còn bên phải là ảnh của danh sách khi đã sắp xếp.

```
//Ham sap xep sinh vien theo Alphabet:
void sapXepAlphabet(SinhVien** head){
```

```
=== DANH SACH SINH VIEN ===

MSSV: 123
Ho ten: M
Diem: 345.00
Danh_gia_sinh_vien: Trong!!

MSSV: 364
Ho ten: Nguyen Bao
Diem: 3.30
Danh_gia_sinh_vien: Trong!!

MSSV: 456
Ho ten: Bui Thao Vi
Diem: 6.50
Danh_gia_sinh_vien: Trong!!
```

```
=== DANH SACH SINH VIEN ===

MSSV: 364
Ho ten: Nguyen Bao
Diem: 3.30
Danh_gia_sinh_vien: Trong!!

MSSV: 123
Ho ten: M
Diem: 345.00
Danh_gia_sinh_vien: Trong!!

MSSV: 456
Ho ten: Bui Thao Vi
Diem: 6.50
Danh_gia_sinh_vien: Trong!!
```

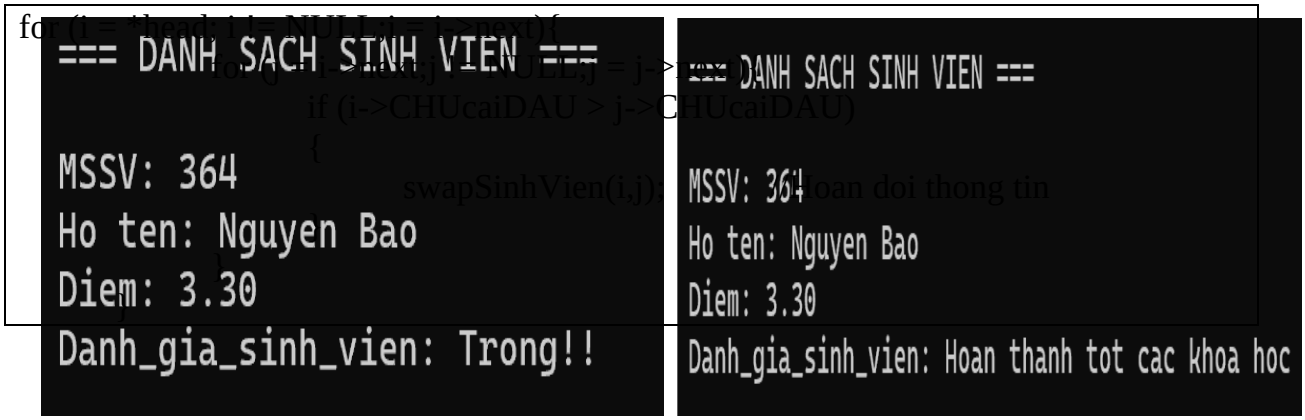
Gồm các hàm con: chuCaiDau(), swapSinhVien().

2.3.11 Đánh giá sinh viên

Mục đích: Cho phép người dùng nhập đánh giá thủ công cho từng sinh viên theo MSSV.

Chức năng: Tìm sinh viên theo MSSV, nhập chuỗi đánh giá và lưu vào thuộc tính 'Danh_Gia'. Bên trái là lúc chưa nhập đánh gai sinh viên nên ở mục đó hiển thị "Trong!!". Còn bên phải là sau khi ta đã cập nhật đánh giá cho sinh viên.

```
//Ham danh gia sinh vien:
void DGsinhvien(SinhVien** head)
```



2.3.12 Thoát chương trình

Mục đích: Kết thúc chương trình và thoát khỏi vòng lặp menu khi người dùng không muốn dừng các thao tác với

Chức năng: Không thực hiện thao tác nào khác, giải phóng bộ nhớ và thoát chương trình.

3.Tổng kết:

3.1 Kết quả đạt được

- Có được kỹ năng cần thiết về nghiệp vụ phân tích.
- Hiểu biết và nắm chắc được những kiến thức về môn học.
- Có thể tạo được một sản phẩm hoàn thiện ứng dụng được những kiến thức đã học.
- Nâng cao kỹ năng làm việc nhóm và kỹ năng sử dụng các công cụ hỗ trợ lập trình.
- Sản phẩm hầu như đáp ứng được các yêu cầu mà nhóm đề ra.
- Tính ổn định và khả năng phát triển thêm đáp ứng tốt yêu cầu đề ra.

3.2 Hướng mở rộng chương trình

- Bổ sung giao diện người dùng bằng thư viện ncurses hoặc chuyển sang C++ dùng Qt.
- Thêm tính năng lưu/đọc file nhị phân.
- Thêm chức năng thống kê: trung bình điểm, đếm theo học lực,...
- Cho phép lọc sinh viên theo năm sinh, giới tính (nếu mở rộng struct).

PHỤ LỤC

[1]. Link github: https://github.com/ThanhHuy1301/BAITAPLON.git

TÀI LIỆU THAM KHẢO

[1]. https://blog.28tech.com.vn/
[2]. https://www.tutorialspoint.com/cprogramming/index.htm