CHƯƠNG 3. CÁC CẤU TRÚC ĐIỀU KHIỂN

Một chương trình bao gồm nhiều câu lệnh. Thông thường các câu lệnh được thực hiện một cách lần lượt theo thứ tự mà chúng được viết ra. Các cấu trúc điều khiển cho phép thay đổi trật tự nói trên, do đó máy có thể nhảy thực hiện một câu lệnh khác ở một ví trí trước hoặc sau câu lệnh hiện thời.

Xét về mặt công dụng, có thể chia các cấu trúc điều khiển thành các nhóm chính:

- ✓ Nhảy không có điều kiện.
- ✓ Rẽ nhánh.
- ✓ Tổ chức chu trình.

Ngoài ra còn một số toán tử khác có chức năng bổ trợ như break, continue.

3.1. Cấu trúc rẽ nhánh

3.1.1. Cấu trúc if-else

Toán tử if cho phép lựa chọn chạy theo một trong hai nhánh tuỳ thuộc vào sự bằng không và khác không của biểu thức. Nó có hai cách viết sau:

if (biểu thức)
khối lệnh 1;
/* Dạng một */
	if (biểu thức)
	khối lệnh 1;
	else
	khối lệnh 2 ;
	/* Dạng hai */

Hoạt động của biểu thức dạng 1: Máy tính giá trị của biểu thức. Nếu biểu thức đúng (biểu thức có giá trị khác 0) máy sẽ thực hiện khối lệnh 1 và sau đó sẽ thực hiện các lệnh tiếp sau lệnh if trong chương trình. Nếu biểu thức sai (biểu thức có giá trị bằng 0) thì máy bỏ qua khối lệnh 1 mà thực hiện ngay các lệnh tiếp sau lệnh if trong chương trình.

Hoạt động của biểu thức dạng 2: Máy tính giá trị của biểu thức. Nếu biểu thức đúng (biểu thức có giá trị khác 0) máy sẽ thực hiện khối lệnh 1 và sau đó sẽ thực hiện các lệnh tiếp sau khối lệnh 2 trong chương trình. Nếu biểu thức sai (biểu thức có giá trị bằng 0) thì máy bỏ qua khối lệnh 1 mà thực hiện khối lệnh 2 sau đó thực hiện tiếp các lệnh tiếp sau khối lệnh 2 trong chương trình.

Ví dụ 3.1: Chương trình nhập vào hai số a và b, tìm max của hai số rồi in kết quả lên màn hình. Chương trình có thể viết bằng cả hai cách trên như sau:

Cách 1:

```
#include <stdio.h>
int main()
{
     float a, b, max;
     printf("\n Cho a = ");
     scanf("%f", &a);

     printf("\n Cho b = ");
     scanf("%f", &b);

     max = a;
     if (b>max) max = b;
     printf(" \n Max cua hai so a = %8.2f  va b = %8.2f la
Max = %8.2f", a, b, max);
}
```

Cách 2:

```
#include <stdio.h>
int main()
{
     float a, b, max;

     printf("\n Cho a = ");
```

```
     scanf("%f", &a);

     printf("\n Cho b = ");
     scanf("%f", &b);

     if (a>b) max = a;
     else max = b;
     printf(" \n Max cua hai so a = %8.2f  va b = %8.2f la
Max = %8.2f", a, b, max);
}
```

Sự lồng nhau của các toán tử if: C cho phép sử dụng các toán tử if lồng nhau có nghĩa là trong các khối lệnh (1 và 2) ở trên có thể chứa các toán tử if - else khác. Trong trường hợp này, nếu không sử dụng các dấu đóng mở ngoặc cho các khối thì sẽ có thể nhầm lẫn giữa các if-else.

Chú ý là máy sẽ gắn toán tử else với toán tử if không có else gần nhất. Chẳng hạn như đoạn chương trình ví dụ sau:

```
     if (n>0)  // if thứ nhất
          if (a>b)  // if thứ hai
               z = a;
          else
               z = b;
```

thì else ở đây sẽ đi với if thứ hai.

Đoạn chương trình trên tương đương với:

```
     if (n>0)       //if thứ nhất
     {
          if (a>b)  // if thứ hai
               z = a;
          else
               z = b;
```

```
}
```

Trường hợp ta muốn else đi với if thứ nhất ta viết như sau:

```
if (n>0) // if thứ nhất
{
     if (a>b) // if thứ hai
     z = a;
}
       else
     z = b;
```

3.1.2. Cấu trúc switch:

Là cấu trúc tạo nhiều nhánh đặc biệt. Nó căn cứ vào giá trị một biểu thức nguyên để để chọn một trong nhiều cách nhảy.

Cấu trúc tổng quát của nó là:

switch (biểu thức nguyên)
{
case n1
khối lệnh 1
case n2
khối lệnh 2
.....
case nk
khối lệnh k
[default
khối lệnh k+1]
}

Với ni là các số nguyên, hằng ký tự hoặc biểu thức hằng. Các ni cần có giá trị khác nhau. Đoạn chương trình nằm giữa các dấu { } gọi là thân của toán tử switch.

default là một thành phần không bắt buộc phải có trong thân của switch.

Sự hoạt động của toán tử switch phụ thuộc vào giá trị của biểu thức viết trong dấu ngoặc () như sau:

Khi giá trị của biểu thức này bằng ni, máy sẽ nhảy tới các câu lệnh có nhãn là case ni.

Khi giá trị biểu thức khác tất cả các ni thì cách làm việc của máy lại phụ thuộc vào sự có mặt hay không của lệnh default như sau:

Khi có ***default*** máy sẽ nhảy tới câu lệnh sau nhãn ***default***.

Khi không có ***default*** máy sẽ nhảy ra khỏi cấu trúc ***switch***.

Chú ý:

Máy sẽ nhảy ra khỏi toán tử switch khi nó gặp câu lệnh break hoặc dấu ngoặc nhọn đóng cuối cùng của thân switch. Ta cũng có thể dùng câu lệnh goto trong thân của toán tử switch để nhảy tới một câu lệnh bất kỳ bên ngoài switch.

Khi toán tử switch nằm trong thân một hàm nào đó thì ta có thể sử dụng câu lệnh return trong thân của switch để ra khỏi hàm này (lệnh return sẽ đề cập sau).

Khi máy nhảy tới một câu lệnh nào đó thì sự hoạt động tiếp theo của nó sẽ phụ thuộc vào các câu lệnh đứng sau câu lệnh này. Như vậy nếu máy nhảy tới câu lệnh có nhãn case ni thì nó có thể thực hiện tất cả các câu lệnh sau đó cho tới khi nào gặp câu lệnh break, goto hoặc return. Nói cách khác, máy có thể đi từ nhóm lệnh thuộc case ni sang nhóm lệnh thuộc case thứ ni+1. Nếu mỗi nhóm lệnh được kết thúc bằng break thì toán tử switch sẽ thực hiện chỉ một trong các nhóm lệnh này.

Ví dụ 3.2: Lập chương trình phân loại học sinh theo điểm sử dụng cấu trúc switch:

```
#include <stdio.h>
int main()
{
     int diem;
     printf("\nVao du lieu:");
     printf("\n Diem = ");
```

```
    scanf("%d", &diem);
    switch (diem)
    {
        case 0:
        case 1:
        case 2:
        case 3:
            printf("Kem\n");
            break;
        case 4:
            printf("Yeu\n");
            break;
        case 5:
        case 6:
            printf("Trung binh\n");
            break;
        case 7:
        case 8:
            printf("Kha\n");
            break;
        case 9:
        case 10:
            printf("Gioi\n");
            break;
        default:
            printf("Vao sai\n");
    }
}
```

3.2. Cấu trúc lặp

3.2.1. Cấu trúc lặp for

Toán tử for dùng để xây dựng cấu trúc lặp có dạng sau:

for (biểu thức 1; biểu thức 2; biểu thức 3)

Lệnh hoặc khối lệnh ;

Toán tử for gồm ba biểu thức và thân for. Thân for là một câu lệnh hoặc một khối lệnh viết sau từ khoá for. Bất kỳ biểu thức nào trong ba biểu thức trên có thể vắng mặt nhưng phải giữ dấu ; .

Thông thường biểu thức 1 là toán tử gán để tạo giá trị ban đầu cho biến điều khiển, biểu thức 2 là một quan hệ logic biểu thị điều kiện để tiếp tục chu trình, biểu thức ba là một toán tử gán dùng để thay đổi giá trị biến điều khiển.

Hoạt động của toán tử for:

Toán tử for hoạt động theo các bước sau:

Xác định biểu thức 1

Xác định biểu thức 2

Tuỳ thuộc vào tính đúng sai của biểu thức 2 để máy lựa chọn một trong hai nhánh:

- Nếu biểu thức hai có giá trị 0 (sai), máy sẽ ra khỏi for và chuyển tới câu lệnh sau thân for.

- Nếu biểu thức hai có giá trị khác 0 (đúng), máy sẽ thực hiện các câu lệnh trong thân for.

- Tính biểu thức 3, sau đó quay lại bước 2 để bắt đầu một vòng mới của chu trình.

Chú ý: Nếu biểu thức 2 vắng mặt thì nó luôn được xem là đúng. Trong trường hợp này việc ra khỏi chu trình for cần phải được thực hiện nhờ các lệnh break, goto hoặc return viết trong thân chu trình.

Trong dấu ngoặc tròn sau từ khoá for gồm ba biểu thức phân cách nhau bởi dấu ; . Trong mỗi biểu thức không những có thể viết một biểu thức mà có quyền viết một dãy biểu thức phân cách nhau bởi dấu phảy. Khi đó các biểu thức trong mỗi phần được xác định từ trái sang phải. Tính đúng sai của dãy biểu thức được tính là tính đúng sai của biểu thức cuối cùng trong dãy này.

Trong thân của for ta có thể dùng thêm các toán tử for khác, vì thế ta có thể xây dựng các toán tử for lồng nhau.

Khi gặp câu lệnh break trong thân for, máy ra sẽ ra khỏi toán tử for sâu nhất chứa câu lệnh này. Trong thân for cũng có thể sử dụng toán tử goto để nhảy đến một ví trí mong muốn bất kỳ.

Ví dụ 3.3: Nhập một dãy số rồi đảo ngược thứ tự của nó.

```
#include <stdio.h>
float x[] = {1.3, 2.5, 7.98, 56.9, 7.23};
int n = sizeof(x)/sizeof(float);
int main()
{
     int i, j;
     float c;
     for (i = 0, j = n-1; i<j; ++i, --j)
     {
          c = x[i]; x[i] = x[j]; x[j] = c;
     }
     printf("\n Day so dao la \n\n");
     for (i = 0; i<n; ++i)
     printf("%8.2f", x[i]);
}
```

Ví dụ 3.4: Tính tích hai ma trận mxn và nxp.

```
#include <stdio.h>
float x[3][2], y[2][4], z[3][4], c;
int main()
{
     int i, j;
     printf("\n nhap gia tri cho ma tran X ");
     for (i = 0; i<= 2; ++i)
          for (j = 0; j<= 1; ++j)
```

```
          {
               printf("\n x[%d][%d] = ", i, j);
               scanf("%f", &c);
               x[i][j] = c;
          }
     printf("\n nhap gia tri cho ma tran Y ");
     for (i = 0; i<= 1; ++i)
          for (j = 0; j<= 3; ++j)
          {
               printf("\n y[%d][%d] = ", i, j);
               scanf("%f", &c);
               y[i][j] = c;
          }
}
```

3.2.2. Cấu trúc lặp while

Toán tử while dùng để xây dựng chu trình lặp dạng:

while (biểu thức)

Lệnh hoặc khối lệnh;

Như vậy toán tử while gồm một biểu thức và thân chu trình. Thân chu trình có thể là một lệnh hoặc một khối lệnh.

Hoạt động của chu trình như sau:

Máy xác định giá trị của biểu thức, tuỳ thuộc giá trị của nó máy sẽ chọn cách thực hiện như sau:

Nếu biểu thức có giá trị 0 (biểu thức sai), máy sẽ ra khỏi chu trình và chuyển tới thực hiện câu lệnh tiếp sau chu trình trong chương trình.

Nếu biểu thức có giá trị khác không (biểu thức đúng), máy sẽ thực hiện lệnh hoặc khối lệnh trong thân của while. Khi máy thực hiện xong khối lệnh này nó lại thực hiện xác định lại giá trị biểu thức rồi làm tiếp các bước như trên.

Chú ý:

Trong các dấu ngoặc () sau while chẳng những có thể đặt một biểu thức mà còn có thể đặt một dãy biểu thức phân cách nhau bởi dấu phảy. Tính đúng sai của dãy biểu thức được hiểu là tính đúng sai của biểu thức cuối cùng trong dãy.

Bên trong thân của một toán tử while lại có thể sử dụng các toán tử while khác. bằng cách đó ta đi xây dựng được các chu trình lồng nhau.

Khi gặp câu lệnh break trong thân while, máy sẽ ra khỏi toán tử while sâu nhất chứa câu lệnh này.

Trong thân while có thể sử dụng toán tử goto để nhảy ra khỏi chu trình đến một vị trí mong muốn bất kỳ. Ta cũng có thể sử dụng toán tử return trong thân while để ra khỏi một hàm nào đó.

Ví dụ 3.5: Chương trình tính tích vô hướng của hai véc tơ x và y:

```
#include <stdio.h>
float x[] = {2, 3.4, 4.6, 21}, y[] = {24, 12.3, 56.8,
32.9};

int main()
{
     float s = 0;
     int i = -1;

     while (++i<4)
          s += x[i]*y[i];
     printf("\n Tich vo huong hai vec to x va y la:%8.2f",
s);
}
```

3.2.3. Cấu trúc lặp do-while

Khác với các toán tử while và for, việc kiểm tra điều kiện kết thúc đặt ở đầu chu trình, trong chu trình do while việc kiểm tra điều kiện kết thúc đặt cuối chu trình. Như vậy thân của chu trình bao giờ cũng được thực hiện ít nhất một lần.

Chu trình do while có dạng sau:

do

Lệnh hoặc khối lệnh;

while (biểu thức);

Lệnh hoặc khối lệnh là thân của chu trình có thể là một lệnh riêng lẻ hoặc là một khối lệnh.

Hoạt động của chu trình như sau:

Máy thực hiện các lệnh trong thân chu trình.

Khi thực hiện xong tất cả các lệnh trong thân của chu trình, máy sẽ xác định giá trị của biểu thức sau từ khoá while rồi quyết định thực hiện như sau:

Nếu biểu thức đúng (khác 0) máy sẽ thực hiện lặp lại khối lệnh của chu trình lần thứ hai rồi thực hiện kiểm tra lại biểu thức như trên.

Nếu biểu thức sai (bằng 0) máy sẽ kết thúc chu trình và chuyển tới thực hiện lệnh đứng sau toán tử while.

Chú ý: Những điều lưu ý với toán tử while ở trên hoàn toàn đúng với do while.

Ví dụ 3.6: Đoạn chương trình xác định phần tử âm đầu tiên trong các phần tử của mảng x.

```
#include <stdio.h>
float x[5], c;

int main()
{
     int i = 0;
     printf("\n nhap gia tri cho ma tran x ");

     for (i = 0; i<= 4; ++i)
     {
          printf("\n x[%d] = ", i);
          scanf("%f", &c);
```

```
          x[i] = c;
     }
     do
          ++i;
     while (x[i] >= 0 && i<= 4);
     if (i<= 4)
          printf("\n Phan tu am dau tien = x[%d] = %8.2f",
i, x[i]);
     else
          printf("\n Mang khong có phan tu am ");
}
```

3.3. Câu lệnh break, continute

3.3.1. Câu lệnh break

Câu lệnh break cho phép ra khỏi các chu trình với các toán tử for, while và switch. Khi có nhiều chu trình lồng nhau, câu lệnh break sẽ đưa máy ra khỏi chu trình bên trong nhất chứa nó không cần điều kiện gì. Mọi câu lệnh break có thể thay bằng câu lệnh goto với nhãn thích hợp.

Ví dụ 3.7: Biết số nguyên dương n sẽ là số nguyên tố nếu nó không chia hết cho các số nguyên trong khoảng từ 2 đến căn bậc hai của n. Viết đoạn chương trình đọc vào số nguyên dương n, xem n có là số nguyên tố.

```
#include <stdio.h>
#include <math.h>
unsigned int n;
int main()
{
     int i, nt = 1;
     printf("\n cho n = ");
     scanf("%d", &n);
     for (i = 2; i<= sqrt(n); ++i)
          if ((n % i) == 0)
```

```
          {
               nt = 0;
               break;
          }
     if (nt)
          printf("\n %d la so nguyen to", n);
     else
          printf("\n %d khong la so nguyen to", n);
}
```

3.3.2. Câu lệnh continue

Trái với câu lệnh break, lệnh continue dùng để bắt đầu một vòng mới của chu trình chứa nó. Trong while và do while, lệnh continue chuyển điều khiển về thực hiện ngay phần kiểm tra, còn trong for điều khiển được chuyển về bước khởi đầu lại (tức là bước: tính biểu thức 3, sau đó quay lại bước 2 để bắt đầu một vòng mới của chu trình).

Chú ý: Lệnh continue chỉ áp dụng cho chu trình chứ không áp dụng cho switch.

Ví dụ 3.8: Viết chương trình để từ một nhập một ma trận a sau đó:

- Tính tổng các phần tử dương của a.

- Xác định số phần tử dương của a.

- Tìm cực đại trong các phần tử dương của a.

```
#include <stdio.h>
float a[3][4];
int main()
{
     int i, j, soptd = 0;
     float tongduong = 0, cucdai = 0, phu;
     for (i = 0; i<3; ++i)
     for (j = 0; i<4; ++j)
     {
          printf("\n a[%d][%d] = ", i, j);
```

```
            scanf("%f", &phu);
            a[i][j] = phu;
            if (a[i][j]<= 0) continue;
            tongduong +=a[i][j];
            if (cucdai<a[i][j]) cucdai = a[i][j];
            ++soptd;
        }
        printf("\n So phan tu duong la: %d", soptd);
        printf("\n  Tong  cac  phan  tu  duong  la:  %8.2f",
tongduong);
        printf("\n Cuc dai phan tu duong la: %8.2f", cucdai);
}
```

CHƯƠNG 4. HÀM VÀ TRUYỀN THAM SỐ

4.1. Định nghĩa hàm trong C

4.1.1. Khai báo hàm

Hàm là một khối lệnh được thực hiện khi nó được gọi từ một điểm khác của chương trình.

Cú pháp:

type <tên hàm> ([tham số 1], [tham số 2], ...)
<khối lệnh>;

Trong đó:

- *type* là kiểu dữ liệu được trả về của hàm.

- <*tên hàm*> là tên gọi của hàm.

- *[tham số i]* là các tham số (có nhiều bao nhiêu cũng được tuỳ theo nhu cầu).

Một tham số bao gồm tên kiểu dữ liệu sau đó là tên của tham số giống như khi khai báo biến (ví dụ int x) và đóng vai trò bên trong hàm như bất kì biến nào khác. Chúng dùng để truyền tham số cho hàm khi nó được gọi. Các tham số khác nhau được ngăn cách bởi các dấu phẩy.

- <*khối lệnh*> là thân của hàm. Nó có thể là một lệnh đơn hay một khối lệnh.

Ví dụ 4.1: Dưới đây là ví dụ đầu tiên về hàm

```
#include <stdio.h>
int addition(int a, int b)
{
     int r;
     r = a+b;
     return (r);
}
int main()
{
```

```
    int z;
    z = addition(5, 3);
    printf("\n Z = %d", z);
}
```

Kết quả: z = 8

Chúng ta có thể thấy hàm main bắt đầu bằng việc khai báo biến z kiểu int. Ngay sau đó là một lời gọi tới hàm addition. Nếu để ý chúng ta sẽ thấy sự tương tự giữa cấu trúc của lời gọi hàm với khai báo của hàm:

```
int addition (int a, int b)
                 ↑     ↑
z = addition (  5  ,   3  );
```

Các tham số có vai trò thật rõ ràng. Bên trong hàm main chúng ta gọi hàm addition và truyền hai giá trị: 5 và 3 tương ứng với hai tham số int a và int b được khai báo cho hàm addition.

Vào thời điểm hàm được gọi từ main, quyền điều khiển được chuyển sang cho hàm addition. Giá trị của c hai tham số (5 và 3) được copy sang hai biến cục bộ int a và int b bên trong hàm.

Dòng lệnh sau:

return (r);

Kết thúc hàm addition, và trả lại quyền điều khiển cho hàm nào đã gọi nó (main) và tiếp tục chương trình ở cái điểm mà nó bị ngắt bởi lời gọi đến addition. Nhưng thêm vào đó, giá trị được dùng với lệnh return (r) chính là giá trị được trả về của hàm.

```
int addition (int a, int b)
 ↓8
z = addition (  5  ,   3  );
```

Giá trị trả về bởi một hàm chính là giá trị của hàm khi nó được tính toán. Vì vậy biến z sẽ có có giá trị được trả về bởi addition(5, 3), đó là 8.

4.1.2. Phạm vi hoạt động của các biến

Bạn cần nhớ rằng phạm vi hoạt động của các biến khai báo trong một hàm hay bất kì một khối lệnh nào khác chỉ là hàm đó hay khối lệnh đó và không thể sử dụng bên ngoài chúng. Trong chương trình ví dụ trên, bạn không thể sử dụng trực tiếp các biến a, b hay r trong hàm main vì chúng là các biến cục bộ của hàm addition. Thêm vào đó bạn cũng không thể sử dụng biến z trực tiếp bên trong hàm addition vì nó làm biến cục bộ của hàm main.

Tuy nhiên bạn có thể khai báo các biến toàn cục để có thể sử dụng chúng ở bất kì đâu, bên trong hay bên ngoài bất kì hàm nào. Để làm việc này bạn cần khai báo chúng bên ngoài mọi hàm hay các khối lệnh, có nghĩa là ngay trong thân chương trình.

Ví dụ 4.2: Đây là một ví dụ khác về hàm:

```
#include <stdio.h>
int subtraction(int a, int b)
{
     int r;
     r = a-b;
     return (r);
}
int main()
{
     int x = 5, y = 3, z;
     z = subtraction(7, 2);
     printf("\nKet qua 1: %d", z);
     printf("\nKet qua 2: %d", subtraction(7, 2));
     printf("\nKet qua 3: %d", subtraction(x, y));
     z = 4 + subtraction(x, y);
     printf("\nKet qua 4: %d", z);
}
```

Kết quả:

Ket qua 1: 5

Ket qua 2: 5

Ket qua 3: 2

Ket qua 4: 6

Trong trường hợp này chúng ta tạo ra hàm subtraction. Chức năng của hàm này là lấy hiệu của hai tham số rồi trả về kết quả.

Tuy nhiên, nếu phân tích hàm main các bạn sẽ thấy chương trình đã vài lần gọi đến hàm subtraction. Tôi đã sử dụng vài cách gọi khác nhau để các bạn thấy các cách khác nhau mà một hàm có thể được gọi.

Để có hiểu cặn kẽ ví dụ này bạn cần nhớ rằng một lời gọi đến một hàm có thể hoàn toàn được thay thế bởi giá trị của nó. Ví dụ trong lệnh gọi hàm đầu tiên:

```
z = subtraction(7, 2);

printf("Ket qua 1: %d", z);
```

Nếu chúng ta thay lời gọi hàm bằng giá trị của nó (đó là 5), chúng ta sẽ có:

```
z = 5;

printf("Ket qua 1: %d", z);
```

Tương tự như vậy

```
printf("Ket qua 2: %d", subtraction(7, 2));
```

Cũng cho kết quả giống như hai dòng lệnh trên nhưng trong trường hợp này chúng ta gọi hàm subtraction trực tiếp như là một tham số của printf. Chúng ta cũng có thể viết:

```
printf("Ket qua 2: %d", 5);
```

Vì 5 là kết quả của subtraction(7, 2). Còn với lệnh

```
printf("Ket qua 3: %d", subtraction(x, y));
```

Điều mới mẻ duy nhất ở đây là các tham số của subtraction là các biến thay vì các hằng. Điều này là hoàn toàn hợp lệ. Trong trường hợp này giá trị được truyền cho hàm subtraction là giá trị của x and y.

Trường hợp thứ tư cũng hoàn toàn tương tự. Thay vì viết

z = 4 + subtraction(x, y);

chúng ta có thể viết:

z = subtraction(x, y) + 4;

Cũng hoàn toàn cho kết quả tương đương.

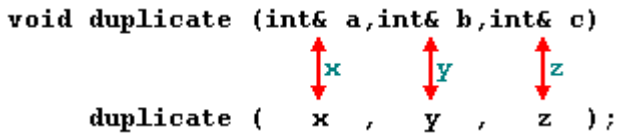
4.2. Truyền tham số cho hàm

Cho đến nay, trong tất cả các hàm chúng ta đã biết, tất cả các tham số truyền cho hàm đều được truyền theo giá trị. Điều này có nghĩa là khi chúng ta gọi hàm với các tham số, những gì chúng ta truyền cho hàm là các giá trị chứ không phải bản thân các biến. Ví dụ, giả sử chúng ta gọi hàm addition như sau:

int x = 5, y = 3, z;

z = addition(x, y);

Trong trường hợp này khi chúng ta gọi hàm addition thì các giá trị 5 and 3 được truyền cho hàm, không phải là bản thân các biến.



Đến đây các bạn có thể hỏi tôi: Như vậy thì sao, có ảnh hưởng gì đâu? Điều đáng nói ở đây là khi các bạn thay đổi giá trị của các biến a hay b bên trong hàm thì các biến x và y vẫn không thay đổi vì chúng đâu có được truyền cho hàm chỉ có giá trị của chúng được truyền mà thôi.

Hãy xét trường hợp bạn cần thao tác với một biến ngoài ở bên trong một hàm. Vì vậy bạn sẽ phải truyền tham số dưới dạng tham số biến như ở trong hàm duplicate trong ví dụ dưới đây:

Ví dụ 4.3:

```
#include <stdio.h>
void duplicate (int& a, int& b, int& c)
{
```

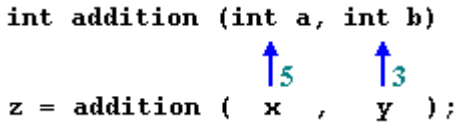
```
     a*= 2;
     b*= 2;
     c*= 2;
}
int main()
{
     int x = 1, y = 3, z = 7;
     duplicate (x, y, z);
     printf("x = %d, y = %d, z = %d", x, y, z);
}
```

Kết quả:

x = 2, y = 6, z = 14

Điều đầu tiên làm bạn chú ý là trong khai báo của duplicate theo sau tên kiểu của mỗi tham số đều là dấu và (&), để báo hiệu rằng các tham số này được truyền theo tham số biến chứ không phải tham số giá trị.

Khi truyền tham số dưới dạng tham số biến chúng ta đang truyền bản thân biến đó và bất kì sự thay đổi nào mà chúng ta thực hiện với tham số đó bên trong hàm sẽ ảnh hưởng trực tiếp đến biến đó.



Trong ví dụ trên, chúng ta đã liên kết a, b và c với các tham số khi gọi hàm (x, y và z) và mọi sự thay đổi với a bên trong hàm sẽ ảnh hưởng đến giá trị của x và hoàn toàn tương tự với b và y, c và z.

Kiểu khai báo tham số theo dạng tham số biến sử dụng dấu và (&) chỉ có trong C++. Trong ngôn ngữ C chúng ta phải sử dụng con trỏ để làm việc tương tự như thế.

Truyền tham số dưới dạng tham số biến cho phép một hàm trả về nhiều hơn một giá trị.

Ví dụ 4.4: Đây là một hàm trả về số liền trước và liền sau của tham số đầu tiên.

```
#include <stdio.h>
void prevnext (int x, int& prev, int& next)
{
     prev = x-1;
     next = x+1;
}
int main()
{
     int x = 100, y, z;
     prevnext (x, y, z);
     printf("Previous = %d, Next = %d", y, z);
}
```

Kết quả

Previous = 99, Next = 101

Giá trị mặc định của tham số

Khi định nghĩa một hàm chúng ta có thể chỉ định những giá trị mặc định sẽ được truyền cho các đối số trong trường hợp chúng bị bỏ qua khi hàm được gọi. Để làm việc này đơn giản chỉ cần gán một giá trị cho đối số khi khai báo hàm. Nếu giá trị của tham số đó vẫn được chỉ định khi gọi hàm thì giá trị mặc định sẽ bị bỏ qua.

Ví dụ 4.5: Giá trị mặc định trong hàm

```
#include <stdio.h>
int divide(int a, int b = 2)
{
     int r;
     r = a/b;
     return (r);
}
int main()
{
     printf("%d", divide(12));
```

```
     printf("\n");
     printf("%d", divide(20, 4));
}
```

Kết quả:

6

5

Nhưng chúng ta thấy trong thân chương trình, có hai lời gọi hàm divide. Trong lệnh đầu tiên:

divide(12)

Chúng ta chỉ dùng một tham số nhưng hàm divide cho phép đến hai. Bởi vậy hàm divide sẽ tự cho tham số thứ hai giá trị bằng 2 vì đó là giá trị mặc định của nó (chú ý phần khai báo hàm được kết thúc bởi int b = 2). Vì vậy kết quả sẽ là 6 (12/2).

Trong lệnh thứ hai:

divide(20, 4)

Có hai tham số, bởi vậy giá trị mặc định sẽ được bỏ qua. Kết quả của hàm sẽ là 5 (20/4).

4.3. Một số ví dụ minh họa

Ví dụ 4.6: Gọi hàm

```
#include <stdio.h>

/* Khai bao ham */
int max(int num1, int num2);

int main ()
{
   /* Khai bao bien cuc bo */
   int a = 100;
   int b = 200;
```

```
   int ret;

   /* Goi ham tra ve gia tri lon nhat */
   ret = max(a, b);

   printf( "Max value is : %d\n", ret );
   return 0;
}

/* Ham tra ve gia tri lon nhat cua hai so*/
int max(int num1, int num2)
{
   /* Khai bao bien cuc bo */
   int result;

   if (num1 > num2)
      result = num1;
   else
      result = num2;

   return result;
}
```

CHƯƠNG 5. CÁC KIỂU DỮ LIỆU CÓ CẤU TRÚC

5.1. Kiểu dữ liệu mảng

5.1.1. Mảng một chiều

Là tập hợp các phần tử có cùng dữ liệu. Giả sử bạn muốn lưu n số nguyên để tính trung bình, bạn không thể khai báo n biến để lưu n giá trị rồi sau đó tính trung bình.

Bạn muốn tính trung bình 10 số nguyên nhập vào từ bàn phím, bạn sẽ khai báo 10 biến: a, b, c, d, e, f, g, h, i, j có kiểu int và lập thao tác nhập cho 10 biến này như sau:

printf("Nhap vao bien a: ");

scanf("%d", &a);

10 biến bạn sẽ thực hiện 2 lệnh trên 10 lần, sau đó tính trung bình:

(a + b + c + d + e + f + g + h + i + j)/10

Điều này chỉ phù hợp với n nhỏ, còn đối với n lớn thì khó có thể thực hiện được. Vì vậy khái niệm mảng được sử dụng

a) Cách khai báo mảng

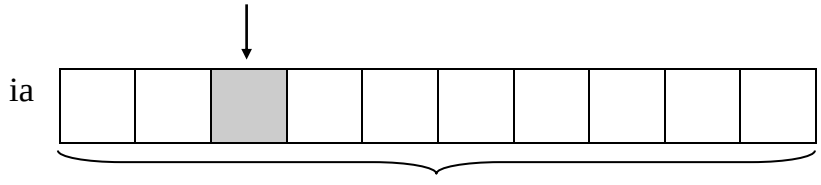
Ví dụ 5.1:

int ia[10];

với **int** là kiểu mảng, ia là tên mảng, 10 số phần tử mảng

Ý nghĩa: Khai báo một mảng số nguyên gồm 10 phần tử, mỗi phần tử có kiểu int.

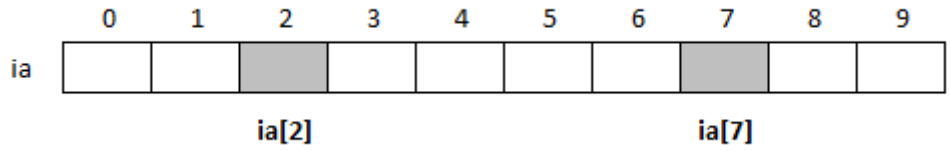
Mỗi phần tử trong mảng có kiểu **int**



10 phần tử

b) Tham chiếu đến từng phần tử mảng

Sau khi mảng được khai báo, mỗi phần tử trong mảng đều có chỉ số để tham chiếu. Chỉ số bắt đầu từ 0 đến n-1 (với n là kích thước mảng). Trong ví dụ trên, ta khai báo mảng 10 phần tử thì chỉ số bắt đầu từ 0 đến 9.



ia[2], ia[7]… là phần tử thứ 3, 8… trong mảng xem như là một biến kiểu **int**.

c) Nhập dữ liệu cho mảng

Ví dụ 5.2: vòng for có giá trị i chạy từ 0 đến 9

```
for (i = 0; i < 10; i++)
{
     printf("Nhap vao phan tu thu %d: ", i + 1);
     scanf("%d", &ia[i]);
}
```

d) Đọc dữ liệu từ mảng

```
for(i = 0; i < 10; i++)
  printf("%3d ", ia[i]);
```

Ví dụ 5.3: Viết chương trình nhập vào n số nguyên. Tính và in ra trung bình cộng

```
/* Tinh trung binh cong n so nguyen */
#include <stdio.h>
#include <conio.h>
int main()
{
     int ia[50], i, in, isum = 0;
     printf("Nhap vao gia tri n: ");
     scanf("%d", &in);
     //Nhap du lieu vao mang
```

```
     for(i = 0; i < in; i++)
     {
          printf("Nhap vao phan tu thu %d: ", i + 1);
          scanf("%d", &ia[i]);   //Nhap gia tri cho phan
tu thu i
     }
     //Tinh tong gia tri cac phan tu
     for(i = 0; i < in; i++)
          isum +=ia[i];          //cong don tung phan tu
vao isum
     printf("Trung binh cong: %.2f\n", (float) isum/in);
     getch();
}
```

Điều gì sẽ xảy ra cho đoạn chương trình trên nếu bạn nhập n > 50 trong khi bạn chỉ khai báo mảng ia tối đa là 50 phần tử. Bạn dùng lệnh if để ngăn chặn điều này trước khi vào thực hiện lệnh for. Thay dòng 9, 10 bằng đoạn lệnh sau:

```
do
{
     printf("Nhap vao gia tri n: ");
     scanf("%d", &in);
} while (in <= 0 || in > 50);    //chi chap nhan gia tri
nhap vao trong khoang 1..50
```

Chạy chương trình và nhập n với các giá trị -6, 0, 51, 6. Quan sát kết quả.

e) Khởi tạo mảng

Ví dụ 5.4: Có 4 loại tiền 1, 5, 10, 25 và 50 đồng. Hãy viết chương trình nhập vào số tiền sau đó cho biết số số tiền trên gồm mấy loại tiền, mỗi loại bao nhiêu tờ.

Phác họa lời giải: Số tiền là 246 đồng gồm 4 tờ 50 đồng, 1 tờ 25 đồng, 2 tờ 10 đồng, 0 tờ 5 đồng và 1 tờ 1 đồng, Nghĩa là bạn phải xét loại tiền lớn trước, nếu hết khả năng mới xét tiếp loại kế tiếp.

```
/* Nhap vao so tien va doi tien ra cac loai 50, 25, 10, 5, 1 */
#include <stdio.h>
#include <conio.h>
#define MAX   5
int main()
{
     int itien[MAX] = {50, 25, 10, 5, 1}; //Khai bao va khoi tao mang voi 5 phan tu
     int i , isotien, ito;
     printf("Nhap vao so tien: ");
     scanf("%d", &isotien);      //Nhap vao so tien
     for (i = 0; i < MAX; i++)
     {
          ito = isotien/itien[i];          //Tim so to cua loai tien thu i
          printf("%4d to %2d dong\n", ito, itien[i]);

          //So tien con lai sau khi da loai tru cac loai tien da co
          isotien = isotien%itien[i];
     }
     getch();
}
```

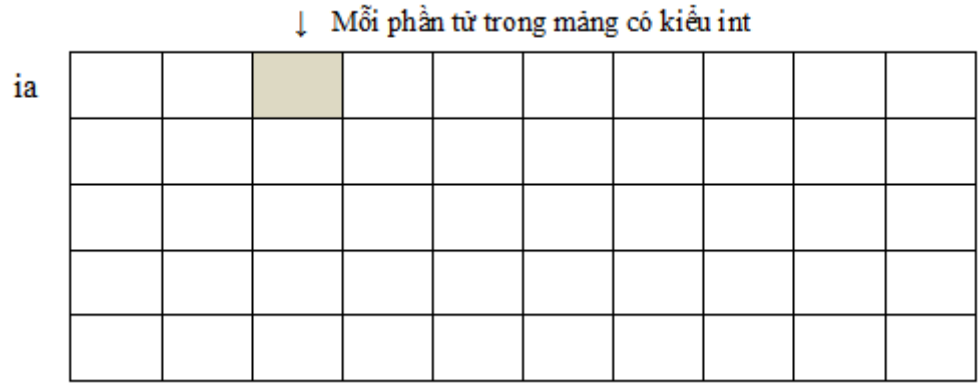
Điều gì sẽ xảy nếu số phần tử mảng lớn hơn số mục, số phần tử dôi ra không được khởi tạo sẽ điền vào số 0. Nếu số phần tử nhỏ hơn số mục khởi tạo trình biên dịch sẽ báo lỗi.

int itien[5] = {50, 25}, phần tử itien[0] sẽ có giá trị 50, itien[1] có giá trị 25, itien[2], itien[3], itien[4] có giá trị 0.

int itien[3] = {50, 25, 10, 5, 1} → trình biên dịch báo lỗi

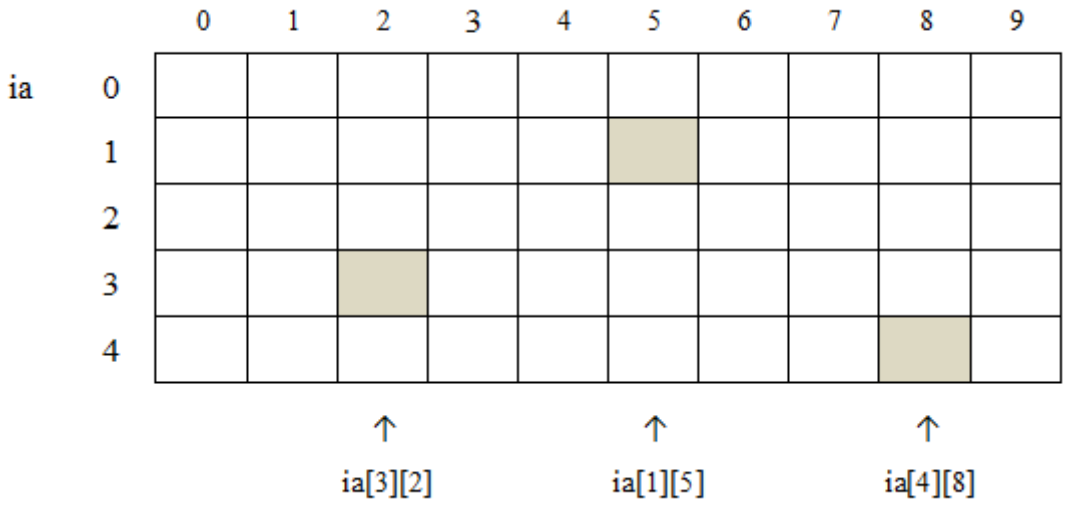
Khởi tạo mảng không bao hàm kích thước: Trong ví dụ trên giả sử ta khai báo int itien[] = {50, 25, 10, 5, 1}. Khi đó trình biên dịch sẽ đếm số mục trong danh sách khởi tạo và dùng con số đó làm kích thước mảng.

5.1.2. Mảng hai chiều



a) Tham chiếu đến từng phần tử mảng 2 chiều

Sau khi được khai báo, mỗi phần tử trong mảng 2 chiều đều có 2 chỉ số để tham chiếu, chỉ số hàng và chỉ số cột. Chỉ số hàng bắt đầu từ 0 đến số hàng – 1 và chỉ số cột bắt đầu từ 0 đến số cột – 1. Tham chiếu đến một phần tử trong mảng 2 chiều ia: ia[chỉ số hàng][chỉ số cột]



ia[3][2] là phần tử tại hàng 3 cột 2 trong mảng 2 chiều xem như là một biến kiểu **int**.

b) Nhập dữ liệu cho mảng 2 chiều

Ví dụ 5.5: Nhập mảng hai chiều

```
for (i = 0; i < 5; i++)     //vòng for có giá trị i chạy từ 0 đến 4 cho hàng
     for (j = 0; j < 10; j++)    //vòng for có giá trị j chạy từ 0 đến 9 cho cột
     {
          printf("Nhap vao phan tu ia[%d][%d]: ", i + 1, j + 1);
          scanf("%d", &ia[i][j]);
     }
```

c) Đọc dữ liệu từ mảng 2 chiều

Ví dụ 5.6: in giá trị các phần tử mảng 2 chiều ra màn hình.

```
for (i = 0; i < 5; i++)     //vòng for có giá trị i chạy từ 0 đến 4 cho hàng
{
     for (j = 0; j < 10; j++)    //vòng for có giá trị j chạy từ 0 đến 9 cho cột
          printf("%3d ", ia[i][j]);
     printf("\n");      //xuống dòng để in hàng kế tiếp
}
```

Ví dụ 5.7: Viết chương trình nhập vào 1 ma trận số nguyên n x n. In ra ma trận vừa nhập vào và ma trận theo thứ tự ngược lại.

```
/* Tinh trung binh cong n so nguyen */
#include <stdio.h>
#include <conio.h>
#define MAX 50
int main()
{
     int ia[MAX][MAX], i, j, in;
```

```
    printf("Nhap vao cap ma tran: ");
    scanf("%d", &in);

    //Nhap du lieu vao ma tran
    for (i = 0; i < in; i++)    //vòng for có giá trị i
chạy từ 0 đến in-1 cho hàng
        for (j = 0; j < in; j++) //vòng for có giá trị j
chạy từ 0 đến in-1 cho cột
        {
            printf("Nhap vao phan tu ia[%d][%d]: ", i +
1, j + 1);
            scanf("%d", &ia[i][j]);
        }

    //In ma tran
    //Vòng for có giá trị i chạy từ 0 đến in-1 cho hàng
    for (i = 0; i < in; i++)
    {
        //vòng for có giá trị j chạy từ 0 đến in-1 cho
cột
        for (j = 0; j < in; j++)
            printf("%3d ", ia[i][j]);
        printf("\n");          //xuống dòng để in hàng
kế tiếp
    }
    printf("\n");          //Tao khoang cach giua 2 ma
tran

    //In ma tran theo thu tu nguoc
    for (i = in-1; i >= 0; i--) //vòng for có giá trị i
chạy từ in-1 đến 0 cho hàng
```

```
    {
        //vòng for có giá trị j chạy từ in-1 đến 0 cho
cột
        for (j = in-1; j >= 0; j--)
            printf("%3d ", ia[i][j]);
        printf("\n");      //xuống dòng để in hàng kế
tiếp
    }
    getch();
}
```

d) Khởi tạo mảng 2 chiều

Ví dụ 5.8: Khởi tạo mảng hai chiều

```
/* Chuong trinh ve chu H lon */
#include <stdio.h>
#include <conio.h>
#define MAX   5
int H[MAX][MAX]    =    {{1, 0, 0, 0, 1},
                         {1, 0, 0, 0, 1},
                         {1, 1, 1, 1, 1},
                         {1, 0, 0, 0, 1},
                         {1, 0, 0, 0, 1}};
int main()
{
    int i , j;
    for (i = 0; i < MAX; i++)
    {
        for (j = 0; j < MAX; j++)
            if (H[i][j])
                printf("!");
            else
```

```
                    printf(" ");
          printf("\n");
     }
     getch();
}
```

e) Dùng mảng 1 chiều làm tham số cho hàm

Ví dụ 5.9: Dùng mảng 1 hiều làm tham số cho hàm

```
/* Chuong trinh tim so lon nhat su dung ham */
#include <stdio.h>
#include <conio.h>
#define MAX   20
//Khai bao prototype
int max(int, int);
//Ham tim so lon nhat trong mang 1 chieu
int max(int ia[], int in)
{
     int i, imax;
     imax = ia[0];      //cho phan tu dau tien la max
     for (i = 1; i < in; i++)
          if (imax < ia[i])           //neu so dang xet >
max
               imax = ia[i];               //gan so nay cho
max
     return imax;                     //tra ve ket qua so
lon nhat
}
int main()
{
     int ia[MAX];
     int i = 0, inum;
```

```
     do
     {
          printf("Nhap vao mot so: ");
          scanf("%d", &ia[i]);
     } while (ia[i++] != 0);
     i--;
     inum = max(ia, i);
     printf("So lon nhat la: %d.\n", inum);
     getch();
}
```

Giải thích chương trình:

Chương trình ban đầu hàm max có hai tham số truyền vào và kết quả trả về là giá trị max có kiểu nguyên, một tham số là mảng 1 chiều kiểu int và một tham số có kiểu int. Với chương trình sau khi sửa hàm max chỉ còn một tham số truyền vào nhưng cho kết quả như nhau. Do sau khi sửa chương trình mảng a[MAX] được khai báo lại là biến toàn cục nên hàm max không cần truyền tham số mảng vào cũng có thể sử dụng được. Tuy vậy, khi lập trình bạn nên viết như chương trình ban đầu là truyền tham số mảng vào (dạng tổng quát) để hàm max có thể thực hiện được trên nhiều mảng khác nhau. Còn với chương trình sửa lại bạn chỉ sử dụng hàm max được với mảng a mà thôi.

Bạn khai báo các mảng sau ia[MAX], ib[MAX], ic[MAX]. Để tìm giá trị lớn nhất của từng mảng. Bạn chỉ cần gọi hàm

- imax_a = max(ia, i);

- imax_b = max(ib, i);

- imax_c = max(ic, i);

Với chương trình sửa lại bạn không thể tìm được số lớn nhất của mảng b và c.

Bạn lưu ý rằng khi truyền mảng sang hàm, không tạo bản sao mảng mới. Vì vậy mảng truyền sang hàm có dạng tham biến. Nghĩa là giá trị của các phần tử trong mảng sẽ bị ảnh hưởng nếu có sự thay đổi trên chúng.

Ví dụ 5.10: Tìm số lớn nhất của 3 mảng a, b, c

```
/* Chuong trinh tim so lon nhat su dung ham */
#include <stdio.h>
#include <conio.h>
#define MAX   20
//Khai bao prototype
int max(int, int);
int input(int);
//Ham tim phan tu lon nhat trong mang 1 chieu
int max(int ia[], int in)
{
     int i, imax;
     imax = ia[0];               //cho phan tu dau tien la
max
     for (i = 1; i < in; i++)
          if (imax < ia[i])  //neu so dang xet > max
               imax = ia[i];      //gan so nay cho max
     return imax;            //tra ve ket qua so lon nhat
}

//Ham nhap lieu vao mang 1 chieu
int input(int ia[])
{
     int i = 0;
     do
     {
          printf("Nhap vao mot so: ");
          scanf("%d", &ia[i]);
     } while (ia[i++] != 0);
     i--;
     return i;
```

```
}
int main()
{
     int ia[MAX], ib[MAX], ic[MAX];
     int inum1, inum2, inum3;
     printf("Nhap lieu cho mang a: \n");
     inum1 = max(ia, input(ia));
     printf("Nhap lieu cho mang b: \n");
     inum2 = max(ib, input(ib));
     printf("Nhap lieu cho mang c: \n");
     inum3 = max(ic, input(ic));
     printf("So lon nhat cua mang a: %d, b: %d, c: %d.\n",
inum1, inum2, inum3);
     getch();
}
```

Hàm input có kiểu trả về là int thông qua biến i (cho biết số lượng phần tử đã nhập vào) và 1 tham số là mảng 1 chiều kiểu int. Dòng 41, 43, 45 lần lượt gọi hàm input với các tham số là mảng a, b, c. Khi hàm input thực hiện việc nhập liệu thì các phần tử trong mảng cũng được cập nhật theo.

f) Dùng mảng 2 chiều làm tham số cho hàm

Ví dụ 5.11: Nhập vào 2 ma trận vuông cấp n số thập phân. Cộng 2 ma trận này lưu vào ma trận thứ 3 và tìm số lớn nhất trên ma trận thứ 3.

```
/* Cong ma tran */
#include <stdio.h>
#include <conio.h>
#define MAX   20

//Khai bao prototype
void input(float);
void output(float);
```

```
void add(float, float, float);
float max(float);
//Khai bao bien toan cuc
int in;
//Ham tim so lon nhat trong mang 2 chieu
float max(float fa[][MAX])
{
     float fmax;
     fmax = fa[0][0];        //cho phan tu dau tien la max
     for (int i = 0; i < in; i++)
          for (int j = 0; j < in; j++)
               if (fmax < fa[i][j])             //neu     so
dang xet > max
                    fmax = fa[i][j];        //gan so nay cho
max
     return fmax;                            //tra ve ket qua
so lon nhat
}

//Ham nhap lieu mang 2 chieu
void input(float fa[][MAX])
{
     float tg;
     for (int i = 0; i < in; i++)
          for (int j = 0; j < in; j++)
          {
               printf("Nhap vao ptu[%d][%d]: ", i, j);
               scanf("%f", &fa[i,j]);
          }
}
//Ham in mang 2 chieu ra man hinh
```

```
void output(float fa[][MAX])
{
     for (int i = 0; i < in; i++)
     {
          for (int j = 0; j < in; j++)
               printf("%5.2f", fa[i][j]);
          printf("\n");
     }
}

//Ham cong 2 mang 2 chieu
void   add(float   fa[][MAX],   float   fb[][MAX],   float
fc[][MAX])
{
     for (int i = 0; i < in; i++)
          for (int j = 0; j < in; j++)
               fc[i][j] = fa[i][j] + fb[i][j];
}

int main()
{
     float fa[MAX][MAX], fb[MAX][MAX], fc[MAX][MAX];
     printf("Nhap vao cap ma tran: ");
     scanf("%d", &in);
     printf("Nhap lieu ma tran a: \n");
     input(fa);
     printf("Nhap lieu ma tran b: \n");
     input(fb);
     printf("Nhap lieu ma tran c: \n");
     input(fc);
     add(fa, fb, fc);
```

```
     printf("Ma tran a: \n");
     output(fa);
     printf("Ma tran b: \n");
     output(fb);
     printf("Ma tran c: \n");
     output(fc);
     printf("So lon nhat cua ma tran c la: %5.2f.\n",
max(fc));
     getch();
}
```

Trong chương trình khai báo biến in toàn cục do biến này sử dụng trong suốt quá trình chạy chương trình. Tham số truyền vào hàm là mảng hai chiều dưới dạng a[][MAX] vì hàm không dành chỗ cho mảng, hàm chỉ cần biết số cột để tham khảo đến các phần tử.

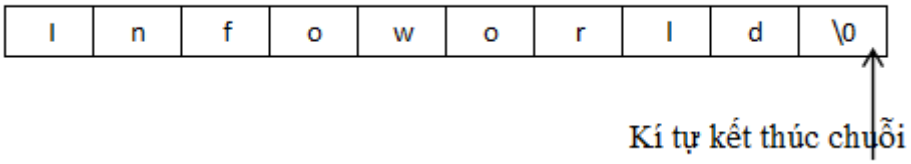
5.2. Chuỗi ký tự

5.2.1. Khái niệm

Chuỗi được xem như là một mảng 1 chiều gồm các phần tử có kiểu char như mẫu tự, con số và bất cứ ký tự đặc biệt như +, -, *, /, $, #…

Theo quy ước, một chuỗi sẽ được kết thúc bởi ký tự **null** ('\0': kí tự rỗng).

Ví dụ: chuỗi "Infoworld" được lưu trữ như sau:



Cách khai báo chuỗi:

Ví dụ khai báo chuỗi có tên cname:

char cname[30];

Ý nghĩa: ***Khai báo chuỗi cname có chiều dài 30 kí tự.*** Do chuỗi kết thúc bằng kí tự null, nên khi bạn khai báo chuỗi có chiều dài 30 kí tự chỉ có thể chứa 29 kí tự.

Ví dụ 5.12: Nhập vào in ra tên

```
/* Chuong trinh nhap va in ra ten*/
#include <stdio.h>
#include <conio.h>
int main()
{
     char cname[30];
     printf("Cho biet ten cua ban: ");
     scanf("%s", cname);
     printf("Chao ban %s\n", cname);
     getch();
}
```

Lưu ý: Không cần sử dụng toán tử địa chỉ & trong cname trong lệnh scanf("%s", cname), vì bản thân fname đã là địa chỉ.

Dùng hàm scanf để nhập chuỗi có hạn chế như sau: Khi bạn thử lại chương trình trên với dữ liệu nhập vào là Mai Lan, nhưng khi in ra bạn chỉ nhận được Mai. Vì hàm scanf nhận vào dữ liệu đến khi gặp khoảng trắng thì kết thúc.

5.2.2. Một số hàm thao tác trên chuỗi ký tự

Ví dụ 5.13: Sử dụng hàm gets, puts phải khai báo #include <stdio.h>

```
/* Chuong trinh nhap va in ra ten*/
#include <stdio.h>
int main()
{
     char cname[30];
     puts("Cho biet ten cua ban: ");
     gets(cname);
     puts("Chao ban ");
```

```
     puts(cname);
}
```

Đối với hàm puts kí tự kết thúc chuỗi null (\0) được thay thế bằng kí tự newline (\n). Hàm gets và puts chỉ có 1 đối số và không sử dụng dạng thức trong nhập liệu cũng như xuất ra màn hình.

Khởi tạo chuỗi:

Ví dụ 5.14: Chương trình nhập và in ra tên

```
#include <stdio.h>
int main()
{
     char cname[30];
     char chao[] = "Chao ban";
     printf("Cho biet ten cua ban: ");
     gets(cname);
     printf("%s %s.\n", chao, cname);
}
```

Chiều dài tối đa của chuỗi khởi tạo bằng số kí tự + 1 (kí tự null). Với chuỗi chao có chiều dài là 9.

Mảng chuỗi:

Ví dụ 5.15: Chơng trình nhập tháng (số) và in tháng (chữ) tương ứng

```
#include <stdio.h>
int main()
{
     char cthang[12][15]    =    {"January",    "February",
"March", "April",
                          "May",  "June",  "July",  "August",
"September",
                          "October",              "November",
"December"};
```

```
    int ithang;
    printf("Nhap vao thang (1-12): ");
    scanf("%d", &ithang);
    printf("%s.\n", cthang[ithang-1]);
}
```

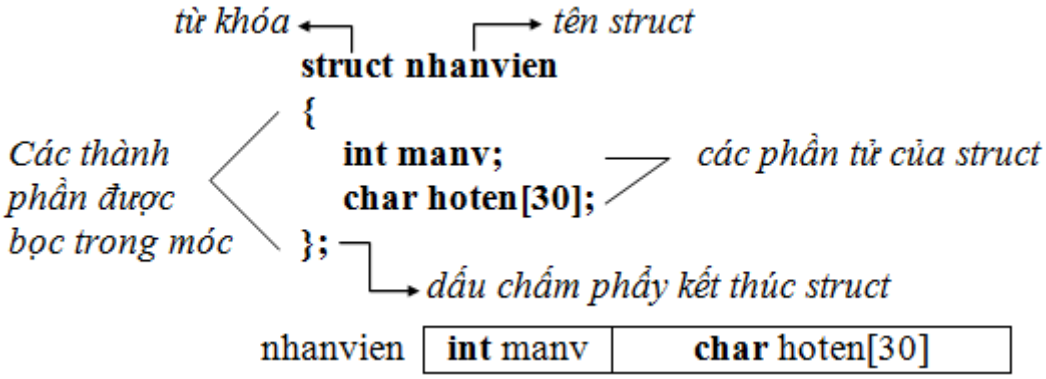
5.2.3. Một số ví dụ minh họa

5.3. Dữ liệu structure

Đối với mảng, chỉ có thể lưu nhiều thông tin có cùng kiểu dữ liệu. Nhưng với structure ta có thể lưu thông tin như một mảng có nhiều kiểu dữ liệu khác nhau.

5.3.1. Khái niệm

Ví dụ 5.16: khai báo một structure về thông tin nhân viên



Ví dụ trên định nghĩa kiểu dữ liệu mới có tên là **struct nhanvien**. Mỗi biến kiểu này gồm 2 phần tử: biến nguyên có tên là **manv** và biến chuỗi có tên **hoten**.

5.3.2. Truy xuất đến các thành phần kiểu cấu trúc

5.3.3. Khai báo kiểu cấu trúc

5.3.4. Cách khai báo biến có kiểu structure

struct nhanvien nv; hoặc nhanvien nv;

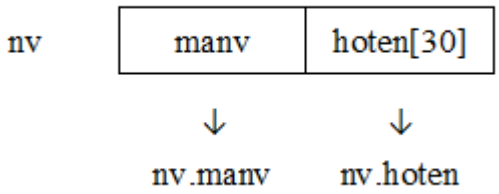
Khai báo biến nv có kiểu struct nhanvien

Vừa tạo structure nhanvien vừa khai báo biến nv

```
struct nhanvien
{
```

```
    int manv;
    char hoten[30];
} nv;
```

Tham chiếu các phần tử trong structure:



Để tham chiếu đến manv trong nv ta viết như sau: **nv.manv** (là biến có kiểu int)

Đối với biến khai báo kiểu con trỏ **nhanvien *nv** thì tham chiếu đến phần tử manv: **nv -> manv.**

Ví dụ 5.17: Nhập và in danh sách nhân viên.

```
/* Danh sach nhan vien */
#include <stdio.h>
#include <stdlib.h>
#define MAX 50
int main()
{
    struct nhanvien
    {
        int manv;
        char hoten[30];
    };
    nhanvien snv[MAX];
    char ctam[10];
    int i, in;
    printf("Nhap vao so nhan vien: ");
    gets(ctam);
    in = atoi(ctam);
```

```
     //Nhap danh sach nhan vien
     for(i = 0; i < in; i++)
     {
          printf("Nhap vao ma nhan vien thu %d: ", i + 1);
          gets(ctam);
          snv[i].manv = atoi(ctam);
          printf("Nhap vao ho ten: ");
          gets(snv[i].hoten);
     }
     //in danh sach nhan vien
     for(i = 0; i < in; i++)
          printf("%5d %s\n", snv[i].manv, snv[i].hoten);
}
```

g) Khởi tạo structure

Ví dụ 5.18: Nhập vào bảng số xe, cho biết xe đó đăng kí ở tỉnh nào

```
/* Xac dinh bien so xe */
#include <stdio.h>
#include <stdlib.h>
#define MAX 6
int main()
{
     struct Tinh
     {
          int ma;
          char *ten;
     };
     Tinh sds[MAX] = {{20, "Thai Nguyen"}, {29, "Ha Noi"},
{22, "Tuyen Quang"}};
     char ctam[10];
     int i, in;
```

```
     printf("Nhap vao bien so xe: ");
     gets(ctam);
     in = atoi(ctam);
     for(i = 0; i < MAX; i++)
          if (sds[i].ma == in)
               printf("Xe   dang   ki   o   tinh   %s.\n",
sds[i].ten);
}
```

h) Structure lồng nhau

Ví dụ 5.19: Danh sách nhân viên

```
#include <stdio.h>
#include <stdlib.h>
#define MAX 50
int main()
{
     struct giacanh
     {
          char vo_chong[30];
          char con;
     };
     struct nhanvien
     {
          int manv;
          char hoten[30];
          giacanh canhan;
     };
     nhanvien snv[MAX];
     char ctam[10];
     int i, in;
     printf("Nhap vao so nhan vien: ");
```

```
     gets(ctam);
     in = atoi(ctam);
     //Nhap danh sach nhan vien
     for(i = 0; i < in; i++)
     {
          printf("Nhap vao ma nhan vien thu %d: ", i + 1);
          gets(ctam);
          snv[i].manv = atoi(ctam);
          printf("Nhap vao ho ten: ");
          gets(snv[i].hoten);
          printf("Cho biet ten vo (hoac chong): ");
          gets(snv[i].canhan.vo_chong);
          printf("So con: ");
          gets(ctam);
     }
     //in danh sach nhan vien
     for(i = 0; i < in; i++)
     {
          printf("Ma so: %d\nHo ten: %s\n Ho ten vo (hoac
chong):  %s\nSo  con:  %d",  snv[i].manv,  snv[i].hoten,
snv[i].canhan.vo_chong, snv[i].canhan.con);
}
```

CHƯƠNG 6. TẬP TIN

6.1. Khái niệm về tệp tin

Tệp tin hay tệp dữ liệu là một tập hợp các dữ liệu có liên quan với nhau và có cùng một kiểu được nhóm lại với nhau thành một dãy. Chúng thường được chứa trong một thiết bị nhớ ngoài của máy tính (đĩa mềm, đĩa cứng...) dưới một cái tên nào đó.

Tên tiếng Anh của tệp là **file,** nó được dùng để chỉ ra một hộp đựng các phiếu hay thẻ ghi của thư viện. Một hình ảnh rõ nét giúp ta hình dung ra tệp là tủ phiếu của thư viện. Một hộp có nhiều phiếu giống nhau về hình thức và tổ chức, song lại khác nhau về nội dung. ở đây, tủ phiếu là tệp, các lá phiếu là các thành phần của tệp. Trong máy tính, một đĩa cứng hoặc một đĩa mềm đóng vai trò chiếc tủ (để chứa nhiều tệp).

Tệp được chứa trong bộ nhớ ngoài, điều đó có nghĩa là tệp được lưu trữ để dùng nhiều lần và tồn tại ngay cả khi chương trình kết thúc hoặc mất điện. Chính vì lý do trên, chỉ những dữ liệu nào cần lưu trữ (như hồ sơ chẳng hạn) thì ta nên dùng đến tệp.

Tệp là một kiểu dữ liệu có cấu trúc. Định nghĩa tệp có phần nào giống mảng ở chỗ chúng đều là tập hợp của các phần tử dữ liệu cùng kiểu, song mảng thường có số phần tử cố định, số phần tử của tệp không được xác định trong định nghĩa.

Trong C, các thao tác tệp được thực hiện nhờ các hàm thư viện. Các hàm này được chia làm hai nhóm: nhóm 1 và nhóm 2. Các hàm cấp 1 là các hàm nhập / xuất hệ thống, chúng thực hiện việc đọc ghi như DOS. Các hàm cấp 2 làm việc với tệp thông qua một biến con trỏ tệp.

Do các hàm cấp 2 có nhiều kiểu truy xuất và dễ dùng hơn so với các hàm cấp 1 nên trong các chương trình viết trong C, các hàm cấp 2 hay được sử dụng hơn.

Một tệp tin dù được xây dựng bằng cách nào đi nữa cũng chỉ đơn giản là một dãy các byte ghi trên đĩa (có giá trị từ 0 đến 255). Số byte của dãy chính là độ dài của tệp.

Có hai kiểu nhập xuất dữ liệu lên tệp: Nhập xuất nhị phân và nhập xuất văn bản.

Nhập xuất nhị phân:

- Dữ liệu ghi lên tệp theo các byte nhị phân như bộ nhớ, trong quá trình nhập xuất, dữ liệu không bị biến đổi.

- Khi đọc tệp, nếu gặp cuối tệp thì ta nhận được mã kết thúc tệp EOF (được định nghĩa trong stdio.h bằng -1) và hàm feof cho giá trị khác 0.

Nhập xuất văn bản:

- Kiểu nhập xuất văn bản chỉ khác kiểu nhị phân khi xử lý ký tự chuyển dòng (mã 10) và ký tự mã 26. Đối với các ký tự khác, hai kiểu đều đọc ghi như nhau.

- Mã chuyển dòng:

Khi ghi, một ký tự LF (mã 10) được chuyển thành 2 ký tự CR (mã 13) và LF

Khi đọc, 2 ký tự liên tiếp CR và LF trên tệp chỉ cho ta một ký tự LF

Mã kết thúc tệp:

Trong khi đọc, nếu gặp ký tự có mã 26 hoặc cuối tệp thì ta nhận được mã kết thúc tệp EOF (bằng -1) và hàm feof(fp) cho giá trị khác 0 (bằng 1).

6.2. Một số hàm thường dùng khi thao tác trên tệp

6.2.1. Khai báo sử dụng tệp

Để khai báo sử dụng tệp, ta dùng lệnh sau:

FILE biến_con_trỏ_tệp;

Trong đó biến_con_trỏ_tệp có thể là biến đơn hay một danh sách các biến phân cách nhau bởi dấu phảy (dấu **,**).

Ví dụ 6.1:

```
FILE *vb, *np;        /* Khai báo hai biến con trỏ tệp */
```

6.2.2. Mở tệp - hàm fopen

Cấu trúc ngữ pháp của hàm:

FILE *fopen(const char *tên_tệp, const char *kiểu);

Nguyên hàm trong: stdio.h

Trong đó:

Đối thứ nhất là tên tệp, đối thứ hai là kiểu truy nhập.

Công dụng:

Hàm dùng để mở tệp. Nếu thành công hàm cho con trỏ kiểu FILE ứng với tệp vừa mở. Các hàm cấp hai sẽ làm việc với tệp thông qua con trỏ này. Nếu có lỗi hàm sẽ trả về giá trị NULL.

Bảng sau chỉ ra các giá trị của kiểu:

Tên kiểu	**ý nghĩa**
"r" "rt"	Mở một tệp để đọc theo kiểu văn bản. Tệp cần đọc phải đã tồn tại, nếu không sẽ có lỗi
"w" "wt"	Mở một tệp để ghi theo kiểu văn bản. Nếu tệp đã tồn tại thì nó sẽ bị xoá.
"a" "at"	Mở một tệp để ghi bổ xung theo kiểu văn bản. Nếu tệp chưa tồn tại thì tạo tệp mới.
"rb"	Mở một tệp để đọc theo kiểu nhị phân. Tệp cần đọc phải đã tồn tại, nếu không sẽ có lỗi.
"wb"	Mở một tệp mới để ghi theo kiểu nhị phân. Nếu tệp đã tồn tại thì nó sẽ bị xoá.
"ab"	Mở một tệp để ghi bổ xung theo kiểu nhị phân. Nếu tệp chưa tồn tại thì tạo tệp mới.
"r+" "r+t"	Mở một tệp để đọc/ghi theo kiểu văn bản. Tệp cần đọc phải đã tồn tại, nếu không sẽ có lỗi
"w+" "w+t"	Mở một tệp để đọc/ghi theo kiểu văn bản. Nếu tệp đã tồn tại thì nó sẽ bị xoá.
"a+" "a+t"	Mở một tệp để đọc/ghi bổ xung theo kiểu văn bản. Nếu tệp chưa tồn tại thì tạo tệp mới.

"r+b"	Mở một tệp để đọc/ghi theo kiểu nhị phân. Tệp cần đọc phải đã tồn tại, nếu không sẽ có lỗi.
"w+b"	Mở một tệp mới để đọc/ghi theo kiểu nhị phân. Nếu tệp đã tồn tại thì nó sẽ bị xoá.
"a+b"	Mở một tệp để đọc/ghi bổ xung theo kiểu nhị phân. Nếu tệp chưa tồn tại thì tạo tệp mới.

Chú ý: Trong các kiểu đọc ghi, ta nên lầm sạch vùng đệm trước khi chuyển từ đọc sang ghi hoặc ngược lại. Ta sẽ đề cập đến các hàm với tính năng xoá sau này.

Ví dụ 6.2:

f=fopen("TEPNP","wb");

6.2.3. Đóng tệp - hàm fclose

Cấu trúc ngữ pháp của hàm:

int fclose(FILE *fp);

Nguyên hàm trong: stdio.h

Trong đó: fp là con trỏ ứng với tệp cần đóng.

Công dụng: Hàm dùng để đóng tệp khi kết thúc các thao tác trên nó. Khi đóng tệp, máy thực hiện các công việc sau:

- ✓ Khi đang ghi dữ liệu thì máy sẽ đẩy dữ liệu còn trong vùng đệm lên đĩa
- ✓ Khi đang đọc dữ liệu thì máy sẽ xoá vùng đệm
- ✓ Giải phóng biến trỏ tệp.
- ✓ Nếu lệnh thành công, hàm sẽ cho giá trị 0, trái lại nó cho hàm EOF.

Ví dụ 6.3:

fclose(f);

6.2.4. Đóng tất cả các tệp đang mở- hàm fcloseall:

Cấu trúc ngữ pháp của hàm:

int fcloseall(void);

Nguyên hàm trong: stdio.h

Công dụng: Hàm dùng để đóng tất cả các tệp đang mở . Nếu lệnh thành công, hàm sẽ cho giá trị bằng số là số tệp được đóng, trái lại nó cho hàm EOF.

Ví dụ 6.4:

```
fcloseall();
```

6.2.5. Kiểmtra cuối tệp - hàm feof

Cấu trúc ngữ pháp của hàm:

```
int feof(FILE *fp);
```

Nguyên hàm trong: stdio.h

Trong đó fp là con trỏ tệp.

Công dụng: Hàm dùng để kiểm tra cuối tệp. Hàm cho giá trị khác 0 nếu gặp cuối tệp khi đọc, trái lại hàm cho giá trị 0.

6.2.6. Truy nhập ngẫu nhiên - các hàm di chuyên con trỏ chỉ vị:

6.2.6.1. Chuyển con trỏ chỉ vị về đầu tệp - Hàm rewind:

Cấu trúc ngữ pháp:

void rewind(FILE *fp);

Nguyên hàm trong: stdio.h

Trong đó fp là con trỏ tệp.

Công dụng: Chuyển con trỏ chỉ vị của tệp fp về đầu tệp. Khi đó việc nhập xuất trên tệp fp được thực hiện từ đầu.

Ví dụ 6.5:

```
rewind(f);
```

6.2.6.2. Chuyển con trỏ chỉ vị trí cần thiết - Hàm fseek

Cấu trúc ngữ pháp:

int fseek(FILE *fp, long sb, int xp);

Nguyên hàm trong: stdio.h

Trong đó

- ✓ fp là con trỏ tệp.
- ✓ sb là số byte cần di chuyển.
- ✓ xp cho biết vị trí xuất phát mà việc dịch chuyển được bắt đầu từ đó.
- ✓ xp có thể nhận các giá trị sau:

  xp=SEEK_SET hay 0: Xuất phát từ đầu tệp.

  xp=SEEK_CUR hay 1: Xuất phát từ vị trí hiện tại của con trỏ chỉ vị.

  xp=SEEK_END hay 2: Xuất phát từ cuối tệp.

Công dụng: Chuyển con trỏ chỉ vị của tệp fp về vị trí xác định bởi xp qua một số byte xác định bằng giá trị tuyệt đối của sb. Chiều di chuyển là về cuối tệp nếu sb dương, trái lại nó sẽ di chuyển về đầu tệp. Khi thành công, hàm trả về giá trị 0. Khi có lỗi hàm trả về giá trị khác không.

Chú ý: Không nên dùng fseek trên tệp tin văn bản, do sự chuyển đổi ký tự sẽ làm cho việc định vị thiếu chính xác.

Ví dụ 6.6:

```
fseek(stream, SEEK_SET, 0);
```

6.2.6.3. Vị trí hiện tại của con trỏ chỉ vị - Hàm ftell:

Cấu trúc ngữ pháp:

int ftell(FILE *fp);

Nguyên hàm trong: stdio.h.

Trong đó fp là con trỏ tệp.

Công dụng: Hàm cho biết vị trí hiện tại của con trỏ chỉ vị (byte thứ mấy trên tệp **fp**) khi thành công. Số thứ tự tính từ 0. Trái lại hàm cho giá trị -1L.

Ví dụ 6.7:

Sau lệnh fseek(fp,0,SEEK_END);

ftell(fp) cho giá trị 3.

Sau lệnh fseek(fp,-1,SEEK_END);

ftell(fp) cho giá trị 2.

6.2.7. Ghi các mẫu tin lên tệp - hàm fwrite

Cấu trúc ngữ pháp của hàm:

int fwrite(void *ptr, int size, int n, FILE *fp);

Nguyên hàm trong: stdio.h

Trong đó:

- ✓ **ptr** là con trỏ trỏ tới vùng nhớ chứa dữ liệu cần ghi.
- ✓ **size** là kích thước của mẫu tin theo byte
- ✓ **n** là số mẫu tin cần ghi
- ✓ **fp** là con trỏ tệp

Công dụng: Hàm ghi n mẫu tin kích thước **size** byte từ vùng nhớ **ptr** lên tệp **fp**. Hàm sẽ trả về một giá trị bằng số mẫu tin thực sự ghi được.

Ví dụ 6.8:

```
#include <stdio.h>
struct mystruct
{
  int i;
  char ch;
};
int main()
{
```

```
   FILE *stream;
   struct mystruct s;
   stream = fopen("TEST.TXT", "wb") /* Mở tệp TEST.TXT */
   s.i = 0;
   s.ch = 'A';
   fwrite(&s, sizeof(s), 1, stream); /* Viết cấu trúc vào
tệp */
   fclose(stream); /* Đóng tệp */
   return 0;
}
```

6.2.8. Đọc các mẫu tin từ tệp - hàm fread

Cấu trúc ngữ pháp của hàm:

int fread(void *ptr, int size, int n, FILE *fp);

Nguyên hàm trong: stdio.h.

Trong đó:

- ✓ **ptr** là con trỏ trỏ tới vùng nhớ chứa dữ liệu cần ghi.
- ✓ **size** là kích thước của mẫu tin theo byte
- ✓ **n** là số mẫu tin cần ghi
- ✓ **fp** là con trỏ tệp

Công dụng: Hàm đọc n mẫu tin kích thước **size** byte từ tệp **fp** lên lên vùng nhớ **ptr**. Hàm sẽ trả về một giá trị bằng số mẫu tin thực sự đọc được.

Ví dụ 6.9:

```
#include <string.h>
#include <stdio.h>
int main()
{
   FILE *stream;
```

```
   char msg[] = "Kiểm tra";
   char buf[20];
 stream = fopen("DUMMY.FIL", "w+");
   /* Viết vài dữ liệu lên tệp */
   fwrite(msg, strlen(msg)+1, 1, stream);
  /* Tìm điểm đầu của file */
   fseek(stream, SEEK_SET, 0);
   /* Đọc số liệu và hiển thị */
   fread(buf, strlen(msg)+1, 1, stream);
   printf("%s\n", buf);
   fclose(stream);
   return 0;
}
```

6.2.9. Nhập xuất ký tự

6.2.9.1. Các hàm putc và fputc

Cấu trúc ngữ pháp:

int putc(int ch, FILE *fp);

int fputc(int ch, FILE *fp);

Nguyên hàm trong: stdio.h.

Trong đó: ch là một giá trị nguyên, fp là một con trỏ tệp.

Công dụng: Hàm ghi lên tệp fp một ký tự có mã bằng

m=ch % 256.

ch được xem là một giá trị nguyên không dấu. Nếu thành công hàm cho mã ký tự được ghi, trái lại cho EOF

Ví dụ 6.10:

```
#include <stdio.h>
int main()
{
```

```
   char msg[] = "Hello world\n";
   int i = 0;
   while (msg[i])
      putc(msg[i++], stdout);  /* stdout thiết bị ra chuẩn
- Màn hình*/
   return 0;
}
```

6.2.9.2. Các hàm getc và fgettc

Cấu trúc ngữ pháp:

int gretc(FILE *fp);

int fputc(FILE *fp);

Nguyên hàm trong: stdio.h.

Trong đó: fp là một con trỏ tệp.

Công dụng: Hàm đọc một ký tự từ tệp fp. Nếu thành công hàm sẽ cho mã đọc được (có giá trị từ 0 đến 255). Nếu gặp cuối tệp hay có lỗi hàm sẽ trả về EOF.

Trong kiểu văn bản, hàm đọc một lượt cả hai mã 13, 10 và trả về giá trị 10. Khi gặp mã 26 hàm sẽ trả về EOF.

Ví dụ 6.11:

```
#include <string.h>
#include <stdio.h>
#include <conio.h>
int main()
{
   FILE *stream;
   char string[] = "Kiem tra";
   char ch;
   /* Mở tệp để cập nhật*/
   stream = fopen("DUMMY.FIL", "w+");
   /*Viết một xâu ký tự vào tệp */
```

```
    fwrite(string, strlen(string), 1, stream);
    /* Tìm vị trí đầu của tệp */
    fseek(stream, 0, SEEK_SET);
    do
    {
        /* Đọc một ký tự từ tệp */
        ch = fgetc(stream);
        /* Hiển thị ký tự */
        putch(ch);
    } while (ch != EOF);
    fclose(stream);
    return 0;
}
```

6.2.10. Xoá tệp - hàm unlink:

Cấu trúc ngữ pháp:

int unlink(const char *tên_tệp)

Nguyên hàm trong: dos.h, io.h, stdio.h.

Trong đó **tên_tệp** là tên của tệp cần xoá.

Công dụng: Dùng để xoá một tệp trên đĩa. Nếu thành công, hàm cho giá trị 0, trái lại hàm cho giá trị EOF.

Ví dụ 6.12:

```
#include <stdio.h>
#include <io.h>
int main(void)
{
    FILE *fp = fopen("junk.jnk","w");
    int status;
    fprintf(fp,"junk");
    status = access("junk.jnk",0);
```

```
   if (status == 0)
      printf("Tệp tồn tại\n");
   else
      printf("Tệp không tồn tại\n");
   fclose(fp);
   unlink("junk.jnk");
   status = access("junk.jnk",0);
   if (status == 0)
      printf("Tệp tồn tại\n");
   else
      printf("Tệp không tồn tại\n");
   return 0;
}
```

6.3. Một số ví dụ

6.3.1. Ghi, đọc mảng

Ví dụ 6.13: Ghi n số nguyên vào file và độc ra từ file

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

#define MAX 5

int main()
{
     FILE *f;
     int i, ia[MAX], ib[MAX];
     for (i = 0; i < 10; i++)
     {
          printf("Nhap vao mot so: ");
          scanf("%d", &ia[i]);
```

```
     }

     if((f = fopen("array.dat", "wb")) == NULL)
     {
          printf("Khong the mo file!\n");
          exit(0);
     }
     fwrite(ia, sizeof(ia), 1, f);        //ghi  mang  vao
file
     fclose(f);

     f = fopen("array.dat", "rb");
     fread(ib, sizeof(ib), 1, f);         //doc  mang  tu
file
     for (i = 0; i < 10; i++)
          printf("%d ", ib[i]);
     fclose(f);
     getch();
}
```

6.3.2. Ghi, đọc structure

Ví dụ 6.14: Danh sách sinh viên

```
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
#define MAX 50
int main()
{
     FILE *f;
     struct nhanvien
     {
```

```
          int manv;
          char hoten[30];
     };
     nhanvien snv[MAX], snv1[MAX];
     char ctam[10];
     int i, in;
     printf("Nhap vao so nhan vien: ");
     gets(ctam);
     in = atoi(ctam);
     //Nhap danh sach nhan vien va ghi vao file
     if((f = fopen("struct.dat", "wb")) == NULL)
     {
          printf("Khong the mo file!\n");
          exit(0);
     }
     fwrite(&in, sizeof(int), 1, f);
          //ghi so nhan vien vao file
     for(i = 0; i < in; i++)
     {
          printf("Nhap vao ma nhan vien thu %d: ", i + 1);
          gets(ctam);
          snv[i].manv = atoi(ctam);
          printf("Nhap vao ho ten: ");
          gets(snv[i].hoten);
          fwrite(&snv[i], sizeof(nhanvien), 1, f);  //ghi
tung nhan vien vao file
     }
     fclose(f);
     //doc danh sach nhan vien tu file va in ra
     f = fopen("struct.dat", "rb");
```

```
     fread(&in, sizeof(int), 1, f);            //doc     so
nhan vien
     for(i = 0; i < in; i++)
     {
          //doc tung nhan vien in ra man hinh
          fread(&snv1[i], sizeof(nhanvien, 1, f);
          printf("%5d %s\n", snv[i].manv, snv[i].hoten);
     }
     getch();
}
```

Tài liệu tham khảo

[1]. Lê Đăng Hưng, Trần Việt Linh, Lê Đức Trung, Nguyễn Thanh Thủy, *Ngôn ngữ lập trình C*, NXB Giáo dục, 1996.

[2]. Phạm Văn Ất, *Giáo trình kỹ thuật lập trình C - Căn bản và nâng cao*, NXB Hồng Đức, 2009.

[3]. Phạm Văn Ất, *Giáo trình C++ và lập trình hướng đối*, NXB Hồng Đức, 2009.

[4]. http://vi.wikipedia.org/wiki/C%2B%2B

[5]. https://developers.google.com/edu/c++

[6]. http://www.tutorialspoint.com/cplusplus/index.htm

[7]. http://www.tutorialspoint.com/cprogramming

[8]. http://www.cplusplus.com

[9]. http://www.cppreference.com