TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN
KHOA CÔNG NGHỆ THÔNG TIN

GIÁO TRÌNH

TIN HỌC CƠ SỞ A

(Ngôn ngữ lập trình C)

~*~

ĐẶNG BÌNH PHƯƠNG

Lưu hành nội bộ

Mục lục

Chương 6.
GIỚI THIỆU NGÔN NGỮ LẬP TRÌNH C

Trong phần 1, chúng ta đã lần lượt tìm hiểu tổng quan về máy tính điện tử bao gồm các khái niệm về phần cứng, phần mềm, hệ điều hành, mạng máy tính, biểu diễn thông tin bên trong mạng máy tính và bước đầu làm quen với các khái niệm lập trình, biểu diễn thuật toán bằng sơ đồ khối… Bước sang phần 2, chúng ta sẽ đi sâu tìm hiểu cách viết chương trình máy tính bằng ngôn ngữ lập trình C để giải quyết một vấn đề, một bài toán nào đó.

Chương này giới thiệu về ngôn ngữ lập trình C, giúp chúng ta có cái nhìn tổng quan về ngôn ngữ lập trình này.

6.1 Giới thiệu

Ngôn ngữ C được thiết kế bởi nhà khoa học Dennis Ritchie tại phòng thí nghiệm Bell Telephone vào năm 1972. C được thiết kế để viết hệ điều hành UNIX và để hỗ trợ cho các nhà lập trình nhanh chóng hoàn thành công việc của mình. Về tên gọi, ngôn ngữ C được đặt tên như thế vì tiền thân của nó là ngôn ngữ B. Ngôn ngữ B được phát triển bởi nhà khoa học Ken Thompson, ông cũng làm việc tại phòng thí nghiệm Bell.

C là một ngôn ngữ lập trình rất mạnh và linh động do đó việc sử dụng nó nhanh chóng vượt qua khỏi giới hạn của phòng thí nghiệm Bell. Các nhà lập trình ở khắp bắt đầu sử dụng nó để viết đủ loại chương trình. Ngay sau đó, nhiều nhà sản xuất phần mềm bắt đầu tung ra các phiên bản C khác nhau, và việc phân

biệt hay sử dụng ngôn ngữ C bắt đầu làm cho các nhà lập trình bối rối. Để giải quyết vấn đề này, Viện Định Chuẩn Quốc Gia Mỹ (American National Standard Institute) gọi tắt là ANSI đã triệu một cuộc họp vào năm 1983 nhằm thiết lập các chuẩn mực cho ngôn ngữ C và được gọi là ANSI C.

Ngoài ngôn ngữ C còn có rất nhiều ngôn ngữ lập trình cấp cao khác như Pascal, BASIC… nhưng theo đánh giá của các chuyên gia điện toán, C có những ưu điểm nổi bật như sau :

- C là ngôn ngữ rất mạnh và linh động, có khả năng thể hiện bất kỳ ý tưởng nào của bạn. Nó có thể chạy trên các hệ điều hành khác nhau và có thể chen các đoạn lệnh C vào cách chương trình viết bằng ngôn ngữ khác.
- C là ngôn ngữ rất phổ biến, được sử dụng rộng rãi bởi các nhà lập trình chuyên nghiệp. Do đó có rất nhiều công cụ được phát triển giúp mọi người có thể lập trình bằng ngôn ngữ C nhanh chóng hơn.
- C là ngôn ngữ có tính khả chuyển. Tức là một chương trình C được viêt cho một hệ thống máy tính nào đó (ví dụ IBM PC) cũng có thể được biên dịch và chạy trên một hệ thống máy tính khác (ví dụ DEC VAX) với rất ít thay đổi hay hầu như không cần thay đổi gì cả.
- C là ngôn ngữ lập trình cô đọng, chỉ chứa các thuật ngữ gọi là từ khóa, là cơ sở để xây dựng chương trình.
- C là ngôn ngữ lập trình đơn thể. Mã lệnh trong C có thể được viết thành các hàm và có thể được sử dụng lại trong nhiều chương trình khác nhau.

Ngày nay có một số ngôn ngữ lập trình cấp cao khác như C++, C#, … Đây là các ngôn ngữ lập trình hướng đối tượng và có thể xem là ngôn ngữ C nâng cấp. Do đó, toàn bộ những gì bạn học được trong ngôn ngữ C đều có thể áp dụng cho các ngôn ngữ nâng cấp đó.

6.2 Bộ từ vựng của C

6.2.1 Các ký tự dùng trong C

C là ngôn ngữ lập trình phân biệt chữ In (Upcase) và chữ thường (Lowcase) trong câu lệnh, biến, hằng, kiểu. Các ký tự hợp lệ được liệt kê dưới đây:

- Bộ chữ cái 26 ký tự Latinh A, B, C… Z, a, b, c… z
- Bộ chữ số thập phân : 0, 1, 2, …, 9
- Các ký hiệu toán học thông dụng : + - * / = < > ( )
- Các ký tự đặc biệt : . , : ; [ ] % \ # $ ‘, gạch nối _, ‘ ’

6.2.2 Từ khóa (Keyword)

Từ khóa là các từ dành riêng trong ngôn ngữ, mỗi từ có chức năng nhất định và khi sử dụng phải viết đúng cú pháp. Từ khóa phân biệt chữ hoa chữ thường. Ngoài ra, không thể sử dụng từ khóa để đặt tên cho các biến, hàm, tên chương trình con. Từ khóa được chia thành các loại sau đây:

- **Các từ khóa dùng để khai báo**

```
const       enum        extern      register
signed      static      struct      typedef
union       unsigned    volatile
```

- **Các từ khóa về kiểu dữ liệu**

```
char        double      float       int
long        short       void
```

- **Các từ khóa điều khiển**

```
case        default     else        if
switch
```

- **Các từ khóa vòng lặp**

```
do          for         while
```

- **Các từ khóa điều khiển**

```
break       continue    goto        return
```

- **Các từ khóa khác**

```
asm         goto        sizeof
```

6.2.3 Tên hay định danh (Identifier)

Tên hay định danh là một dãy ký tự dùng để chỉ tên một hằng số, hằng ký tự, tên một biến, một kiểu dữ liệu, một hàm. Tên không được trùng với các từ khóa và được tạo thành từ các chữ cái và các chữ số nhưng bắt buộc chữ đầu phải là chữ cái hoặc dấu gạch dưới _. Có thể sử dụng ký tự _ chen trong tên nhưng không cho phép chen giữa các khoảng trắng. Tuy nhiên ta nên đặt tên ngắn gọn, dễ nhớ và đầy đủ ý nghĩa. Ví dụ:

Các tên hợp lệ: GiaiPhuongTrinh, Bai_Tap1, …

Các tên không hợp lệ: 1A, Giai Phuong Trinh, …

Ngôn ngữ C phân biệt chữ hoa chữ thường, do đó các tên sau đây có ý nghĩa khác nhau: BaiTap, baitap, BAITAP…

6.2.4 Dấu chấm phẩy (;)

Dấu chấm phẩy ; được sử dụng để phân cách các câu lệnh trong C. Ví dụ:

```
printf("Nhập a : ");
scanf("%d", &a);
```

6.2.5 Câu ghi chú

C cho phép người lập trình chèn vào chương trình các câu ghi chú ở bất kỳ vị trí nào nhằm mục đích làm sáng tỏ thêm cấu trúc của chương trình mà không làm ảnh hưởng đến các phần khác. Các câu ghi chú có thể được ghi trên một dòng hoặc trên các dòng khác nhau miễn là chúng được bao bởi dấu /* ở đầu và dấu */ ở cuối. Ví dụ:

/ Chương trình giải phương trình bậc nhất ax + b = 0 */*

/ Họ và Tên : Nguyễn Văn A*

*MSSV : 0712001 */*

Một số trình biên dịch sau này cho phép sử dụng hai dấu xổ chéo // thay cho việc dùng /* */ đối với các ghi chú trên 1 dòng.

Ví dụ:

// Chương trình giải phương trình bậc nhất ax + b = 0

6.2.6 Hằng ký tự và hằng chuỗi

Hằng ký tự có giá trị là các ký tự trong bảng mã ASCII, được biểu diễn trong cặp dấu nháy đơn.

Ví dụ: 'A', 'a', '0', ' ', …

Hằng chuỗi là dãy các ký tự, được biểu diễn trong cặp dấu nháy kép " "

Ví dụ : "Tin học cơ sở A", "Tôi tên là: Nguyễn Văn A"

6.3 Cấu trúc một chương trình C

Chương trình được viết bằng ngôn ngữ C gồm các phần chính sau đây:

- Phần khai báo chèn các tập tin tiêu đề (header file) vào chương trình. Đây là các tập tin chứa các định nghĩa cần thiết cho trình biên dịch. Có hai cách để xác định tập tin theo sau chỉ thị #include: bao tên tập tin bởi cặp dấu < > đối với tập tin thư viện như stdio.h, conio.h hoặc cặp dấu " " đối với các tập tin tiêu đề do người lập trình tạo ra.

- Phần khai báo các biến toàn cục hoặc các hàm được sử dụng trong chương trình. Thông thường người ta thường đặt các nguyên mẫu hàm (function prototype) ở đây còn phần mô tả hàm được đặt ở dưới cùng.

- Phần định nghĩa hàm chính. Hàm này có tên là main và là thành phần duy nhất luôn phải có trong một chương trình C. Thông thường chương trình sẽ bắt đầu bằng cách thực hiện các dòng lệnh trong hàm main này.

Ví dụ:

```
// Phan khai bao chen cac tap tin tieu de
#include <stdio.h>
#include <conio.h>

// Phan khai bao cac bien toan cuc, nguyen mau ham
int x, y;
void Nhap(int &);        // Prototype ham Nhap
int TinhTong(int, int); // Protytype ham TinhTong

// Phan dinh nghia ham main
void main()
{
      int a, b, tong;

      Nhap(a);
      Nhap(b);

      tong = TinhTong(a, b);

      printf("Tong cua a va b la %d.", tong);
}

// Phan mo ta cac ham
void Nhap(int &n)
{
      printf("Nhap mot so nguyen: ");
      scanf("%d", &n);
}

int TinhTong(int a, int b)
{
      return a + b;
}
```

Bài tập cuối chương

Lý thuyết

1. Tên (định danh) nào sau đây đặt không hợp lệ, tại sao?
    a. Tin hoc co SO A
    b. 1BaiTapKHO
    c. THucHaNH
    d. TinHOC_DaiCuonG
2. Câu ghi chú dùng để làm gì? Cách sử dụng ra sao? Cho ví dụ minh họa.
3. Trình bày cấu trúc của một chương trình C. Giải thích ý nghĩa của từng phần trong cấu trúc.

Thực hành

Gõ và chạy thử các chương trình sau. Xác định phần tiêu đề, phần khai báo và phần thân của chương trình.

4. Chương trình xuất một câu thông báo ra màn hình.

```
#include <stdio.h>

void main()
{
    printf("Tin hoc co so A");
}
```

5. Chương trình tính tổng, hiệu, tích, thương của 2 số nguyên a và b nhập từ bàn phím.

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

void main()
{
    int a, b, tong, hieu, tich;
    float thuong;

    // Nhap 2 so nguyen a va b
    printf("Nhap hai so nguyen: ");
```

```
    scanf("%d%d", &a, &b);

    // Tinh toan va xuat ket qua
    tong = a + b;
    hieu = a - b;
    tich = a * b;
    thuong = a * 1.0 / b;
    printf("%d + %d = %d", a, b, tong);
    printf("%d - %d = %d", a, b, hieu);
    printf("%d * %d = %d", a, b, tich);
    printf("%d / %d = %0.2f", a, b, thuong);
    getch();
}
```

6. Chương trình giải phương trình bậc nhất ax + b = 0 với a và b là 2 số nguyên nhập từ bàn phím.

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

void main()
{
    float a, b;
    // Nhap 2 so thuc a va b
    printf("Nhap so thu nhat: ");
    scanf("%f", &a);
    printf("Nhap so thu hai: ");
    scanf("%f", &b);

    // Xet cac dieu kien
    if (a == 0)
    {
        if (b == 0)
            printf("Phuong trinh VSN");
        else
            printf("Phuong trinh VN");
    }
    else
        printf("Nghiem x = %0.2f", -b/a);
}
```

Chương 7.
CÁC KIỂU DỮ LIỆU CƠ SỞ

Chương này trình bày các kiểu dữ liệu cơ sở trong C đồng thời giới thiệu một số hàm để thao tác trên các kiểu dữ liệu đó. Phần cuối của chương sẽ giới thiệu các lệnh đơn giản như khai báo biến, lệnh gán, nhập xuất giúp chúng ta có thể bắt đầu tự viết những chương trình tính toán đơn giản.

7.1 Các kiểu dữ liệu cơ sở

Trong C có các kiểu cơ sở như sau:

- Kiểu số nguyên.
- Kiểu số thực.
- Kiểu luận lý.
- Kiểu ký tự.

7.1.1 Kiểu số nguyên

Đây là các kiểu dữ liệu mà giá trị của nó là số nguyên. Dữ liệu kiểu số nguyên lại chia ra thành hai loại như sau:

- Các số nguyên có dấu (signed) để chứa các số nguyên âm hoặc dương.

Kiểu (Type)	***Độ lớn (Byte)***	***Miền giá trị (Range)***
char	1	–128 ... 127
int	2	–32.768 ... 32.768
short	2	–32.768 ... 32.768
long	4	–2.147.483.648 ... +2.147.483.647

- Các số nguyên không dấu (unsigned) để chứa các số nguyên dương (kể cả số 0).

Kiểu (Type)	*Độ lớn (Byte)*	*Miền giá trị (Range)*
unsigned char	1	0 ... 255
unsigned int	2	0 ... 65.535
unsigned short	2	0 ... 65.535
unsigned long	4	0 ... 4.294.967.295

7.1.2 Kiểu số thực

Đây là các kiểu dữ liệu mà giá trị của nó là số thực. Trong C định nghĩa các kiểu số thực chuẩn như sau:

Kiểu (Type)	*Độ lớn (Byte)*	*Miền giá trị (Range)*
float	4	$3.4*10^{-38}$ ... $3.4*10^{38}$
double	8	$1.7*10^{-308}$ ... $1.7*10^{308}$

Kiểu float là kiểu số thực có độ chính xác đơn (single-precision floating-point), kiểu double là kiểu số thực có độ chính xác kép (double-precision floating-point).

7.1.3 Kiểu luận lý

Trong C không hỗ trợ kiểu luận lý tường minh mà chỉ ngầm hiểu một cách không tường minh như sau:

- false (sai) là giá trị 0.
- true (đúng) là giá trị khác 0, thường là 1.

Các ngôn ngữ lập trình nâng cấp khác của C như C++ định nghĩ kiểu luận lý tường minh có tên là bool (2 giá trị false/true).

7.1.4 Kiểu ký tự

Đây chính là kiểu dữ liệu số nguyên char có độ lớn 1 byte và miền giá trị là 256 ký tự trong bảng mã ASCII.

7.2 Biến, hằng, câu lệnh và biểu thức

7.2.1 Biến và lệnh gán giá trị cho biến

Biến là một vùng nhớ có kích thước và có một địa chỉ nhất định nằm trong bộ nhớ RAM. Biến dùng để lưu giữ một dữ liệu đầu vào, đầu ra hoặc một kết quả trung gian trong quá trình làm việc. Dữ liệu được lưu trong biến nên cách tổ chức thông tin trong biến là kiểu của dữ liệu. Nội dung của biến có thể thay đổi trong quá trình làm việc.

Để phân biệt các biến với nhau, mỗi biến sẽ được đặt một tên theo quy tắc đặt định danh và được gọi là định danh biến (Variable Indentifier).

Ví dụ: i, j, dem1, TONG_CONG, KetQua…

Bất kỳ một biến nào khi sử dụng trong chương trình đều phải được khai báo như sau:

Cú pháp khai báo biến

```
<kiểu> <tên biến>;
<kiểu> <tên biến 1>, <tên biến 2>;
```

Các biến có cùng kiểu có thể khai báo chung và cách nhau bằng dấu phẩy. Các biến khác kiểu nhau được khai báo cách nhau bằng dấu chấm phẩy.

Ví dụ:

```
int i, j, k;
unsigned char dem;
float ketqua, delta;
```

Để xác định độ lớn của một biến (số byte mà biến chiếm giữ trong bộ nhớ) chúng ta sử dụng toán tử sau:

Toán tử lấy độ lớn của biến

```
int sizeof(<Biến>)
```

7.2.2 Hằng số

Hằng số cũng giống như biến nhưng nội dung của nó không thể thay đổi trong quá trình thực hiện chương trình. Trong C, ta có hai kiểu hằng số, đó là hằng thường (literal constant) và hằng ký hiệu (symbol constant).

7.2.2.1 Hằng thường

Cách khai báo hằng này thường được dùng để khởi tạo giá trị cho biến.

Cú pháp khai báo hằng thường

```
<kiểu> <tên hằng> = <giá trị>;
```

Những hằng số nguyên có thể được viết theo 3 dạng sau:

- Một hằng số bắt đầu với bất kỳ một ký số nào khác 0 sẽ được hiểu là một số nguyên ở hệ thập phân.
- Một hằng số bắt đầu với số 0 sẽ được hiểu là một số nguyên ở hệ bát phân.
- Một hằng số bắt đầu với 0x hoặc 0X sẽ được hiểu là một số nguyên ở hệ thập lục phân.

Ví dụ:

```
int a = 1506;           // 1506(10)
int b = 01506;          // 1506(8)
int c = 0x1506;         // 1506(16)
int d = 0X1506;         // 1506(16)
```

Những hằng số chấm động có thể viết theo ký hiệu hoa học.

Ví dụ:

```
float a = 1.76E2;          // = 1.76*10^2 = 176
float b = 29.12e2;         // = 29.12*10^2 = 2912
float c = 15.6e-3;         // = 15.6*10^-3 = 0.0156
```

7.2.2.2 Hằng ký hiệu

Hằng ký hiệu là một hằng được gán cho một cái tên trong chương trình. Khác với hằng thường, hằng ký hiệu không thể thay đổi giá trị trong quá trình thực hiện chương trình.

Có hai cách để khai báo hằng ký hiệu như sau:

- Sử dụng chỉ thị #define

Cú pháp khai báo hằng ký hiệu

```
#define <tên hằng> <giá trị>
```

Ví dụ:

```
#define MAX 100
#define PI 3.14
```

Dòng #define không kết thúc bằng dấu chấm phẩy vì nó chỉ là một chỉ thị tiền xử lý chứ không phải một câu lệnh. Cách làm này thực ra là thay thế cụm từ này bằng cụm từ kia. Chỉ thị #define có thể đặt ở bất cứ đâu trong chương trình nhưng thường được nhóm lại ở phần đầu chương trình.

- Sử dụng từ khóa const

Cách khai báo hằng này giống với cách khai báo hằng thường (tức là phải xác định kiểu của hằng).

Ví dụ:

```
const int max = 100;
const float PI = 3.14;
```

7.2.3 Câu lệnh

Một câu lệnh (statement) là một chỉ thị trực tiếp, hoàn chỉnh nhằm ra lệnh cho máy tính thực hiện một số tác vụ nhất định. Trong C, các câu lệnh có thể viết trên cùng một dòng. Tuy nhiên,

để cho rõ ràng và dễ kiểm tra lỗi, mỗi câu lệnh nên đặt trên những dòng khác nhau.

Trình biên dịch C sẽ bỏ qua các khoảng trắng (hay tab hoặc dòng trống) chen giữa lệnh. Ví dụ, các lệnh sau đây là tương đương nhau:

```
tong=a+b;

tong = a + b;

tong
=
a
+
b;
```

Tuy nhiên ta nên sử dụng cách thứ hai sẽ làm cho chương trình được trình bày đẹp và dễ đọc hơn.

Có một trường hợp trình biên dịch phải quan tâm đến khoảng trắng, đó là các khoảng trắng trong hằng chuỗi (chuỗi các ký tự).

Ví dụ, các hằng chuỗi trong các câu lệnh sau đây là khác nhau:

```
printf("Tin hoc co so A.");
printf("Tin  hoc   coso A.  ");
```

Nếu chuỗi trong câu lệnh quá dài ta có thể xuống dòng nhưng phải thêm ký hiệu \

Ví dụ:

```
printf("Tin hoc co so A.");     // Cách 1
printf("Tin hoc \               // Cách 2
co so A.");
```

Trong C có hai loại câu lệnh:

- Câu lệnh đơn: chỉ gồm một câu lệnh.
- Câu lệnh phức (còn gọi là khối lệnh): gồm nhiều câu lệnh đơn được nhóm và bao bởi cặp ngoặc nhọn { }.

Ví dụ:

```
printf("THCS A");          // Day la cau lenh don

// Duoi day la cau lenh phuc (hay khoi lenh)
{
    printf("THCS A");
    printf("\nChuong 7.");
}
```

7.2.4 Biểu thức

Biểu thức được tạo thành từ các toán tử (Operator) và các toán hạng (Operand) hay còn gọi là các phép tính. Toán tử tác động lên giá trị của các toán hạng cho kết quả là một giá trị có kiểu nhất định.

Toán hạng có thể là:

- Một hằng.
- Một biến.
- Một lời gọi hàm (Chương 10 – Hàm).
- Một tập các dữ liệu.
- Ép kiểu hay biến đổi kiểu (Type Casting).

Toán tử là các phép toán tác động lên các dữ liệu cùng kiểu, bao gồm các loại sau:

- Toán tử gán.
- Các toán tử số học.
- Các toán tử trên bit.
- Các toán tử quan hệ.
- Các toán tử luận lý.
- Toán tử điều kiện.
- Toán tử ,

7.2.4.1 Toán tử gán

Toán tử gán thường được sử dụng trong lập trình, có tác dụng gán giá trị cho biến.

Cú pháp lệnh gán

```
<biến> = <giá trị>;
<biến> = <biến>;
<biến> = <biểu thức>;
```

Ví dụ :

```
void main()
{
    int a, b, thuong;

    a = 10;          // Gan gia tri cho bien
    b = a;           // Gan bien cho bien
    thuong = a / b;  // Gan bieu thuc cho bien
}
```

Đặc biệt, có thể sử dụng liên tiếp các phép gán miễn là vế trái phải là biến. Ví dụ :

```
a = b = c = d = e = 156;
```

Câu lệnh trên tương đương với chuỗi câu lệnh gán sau:

```
e = 156;
d = e;
c = d;
b = c;
a = b;
```

7.2.4.2 Các toán tử toán học

Các toán tử toán học thực hiện các phép tính cộng, trừ, nhân, chia. C có 2 toán tử một ngôi và 5 toán tử hai ngôi.

- Các toán tử một ngôi

Ta gọi là các toán tử một ngôi vì nó chỉ có một toán hạng đi theo. Các toán tử này chỉ tác động lên toán hạng là biến, không tác động lên toán hạng là hằng.

Toán tử	*Ý nghĩa*	*Ví dụ*
++	Tăng toán hạng lên một đơn vị.	x++, ++x
– –	Giản toán hạng xuống một đơn vị	x– –,– –x

Toán tử một ngôi có thể đặt trước hoặc sau toán hạng. Cả hai cách này cho kết quả giống nhau nhưng cách hoạt động khác nhau. Khi đặt trước toán hạng, toán hạng sẽ được tăng (hoặc giảm) trước khi sử dụng và ngược lại.

Ví dụ sau gán y cho x rồi mới tăng x lên.

```
x = 10;
y = x++;            // y = 10, x = 11
```

Ví dụ sau tăng x lên rồi mới gán cho y.

```
x = 10;
y = x++;            // x = 11, y = 11
```

- Các toán tử hai ngôi

Toán tử	*Ý nghĩa*	*Ví dụ*
+	Cộng hai toán hạng với nhau	x + y
–	Trừ hai toán hạng với nhau	x – y
*	Nhân hai toán hạng với nhau	x * y
/	Chia hai toán hạng với nhau	x / y
%	Lấy số dư của phép chia nguyên	x % y

Toán tử % chỉ tác động lên hai toán hạng kiểu số nguyên.

Đặc biệt, nếu các toán tử này kết hợp với phép gán, ta có thể sử dụng các toán tử gộp +=, –=, *=, /=, %=.

Ví dụ:

```
x += y;             // Tương đương với x = x + y;
```

7.2.4.3 Các toán tử trên bit

Tác động lên các bit của toán hạng kiểu số nguyên.

Toán tử	*Ý nghĩa*	*Ví dụ*
&	And dãy bit của hai số nguyên	x & y
|	Or dãy bit của hai số nguyên	x | y
^	XOr dãy bit của hai số nguyên	x ^ y
>>	Dịch phải n bit	x >> n
<<	Dịch trái n bit	x << n

Tương tự các toán tử số học, các toán hạng này khi kết hợp với phép gán có thể sử dụng toán tử gộp &=, |=, ^=, >>=, <<=.

Ví dụ:

```
x &= y;              // Tương đương với x = x & y;
```

7.2.4.4 Các toán tử quan hệ

Các toán tử quan hệ được sử dụng để so sánh và sẽ cho kết quả là đúng hay true (giá trị 1) hoặc sai hay false (giá trị 0).

Toán tử	*Ý nghĩa*	*Ví dụ*
= =	Toán hạng 1 bằng Toán hạng 2 ?	x = = y
>	TH1 lớn hơn TH2 ?	x > y
<	TH1 bé hơn TH2 ?	x < y
>=	TH1 lớn hơn hay bằng TH2 ?	x >= y
<=	TH1 bé hơn hay bằng TH2 ?	x < y
!=	TH1 khác TH2 ?	x != y

Người lập trình thường hay nhầm lẫn giữa toán tử so sánh bằng là == (hai dấu = viết cạnh nhau) khác với toán tử gán = (một dấu =). Trong trường hợp nhầm lẫn đó, chương trình vẫn chạy bình thường nhưng thường sẽ cho kết quả sai.

7.2.4.5 Các toán tử luận lý

Các toán tử luận lý của C cho ta khả năng tổ hợp nhiều biểu thức quan hệ thành một biểu thức đơn và có thể xác định được tính đúng–sai (true-false) của biểu thức này.

Toán tử	*Ý nghĩa*	*Ví dụ*
&&	Và hai biểu thức	BT1 && BT2
||	Hoặc hai biểu thức	BT1 || BT2
!	Phủ định một biểu thức	!BT

Toán tử && chỉ cho kết quả true (1) nếu cả 2 biểu thức đều true. Toán tử || chỉ cho kết quả false (0) nếu cả 2 biểu thức đều false.

Ví dụ:

```
(1 == 1) && (1 != 2)        // Ket qua true (1)
(2 > 1) || (6 < 1)          // Ket qua true (1)
(2 == 1) && (2 == 2)        // Ket qua false (0)
!(1 == 2)                   // Ket qua true (0)
```

7.2.4.6 Toán tử điều kiện

Toán tử điều kiện là toán tử ba ngôi, tức là có 3 toán hạng.

Toán tử điều kiện
<biểu thức 1> ? <biểu thức 1> : <biểu thức 3>;

Nếu <biểu thức 1> là đúng hay true (khác 0) thì giá trị của toàn bộ biểu thức trên là giá trị của <biểu thức 2>. Ngược lại, nếu <biểu thức 1> là sai hay false (bằng 0) thì giá trị của toàn bộ biểu thức trên là giá trị của <biểu thức 3>.

Ví dụ:

```
x = (2 > 1) ? 3 : 4 ;       // x = 3 do 2 > 1 đung
y = (2 < 1) ? 3 : 4 ;       // y = 4 do 2 < 1 sai
```

7.2.4.7 Toán tử phẩy

Ta có thể tạo ra một biểu thức bằng cách đặt các biểu thức con cách nhau dấu phẩy. Biểu thức mới này sẽ được thực hiện như sau:

- Các biểu biểu thức con tạo nên biểu thức mới sẽ được tính, và biểu thức con bên trái sẽ được tính trước.
- Biểu thức mới nhận giá trị là giá trị của biểu thức bên phải.

Ví dụ:

```
x = (a++, b = b + 2);
```

Tương đương với chuỗi câu lệnh sau:

```
a++;
b = b + 2; x = b;
```

7.2.4.8 Độ ưu tiên của các toán tử

Các toán tử sẽ thực hiện theo thứ tự ưu tiên như sau:

Toán tử	*Thứ tự*
( ) [ ] –> . (toán tử chấm)	→
! ++ — – + (cast) * & sizeof	←
* / %	→
+ –	→
<< >>	→
< <= > >=	→
== !=	→
&	→
|	→
^	→
&&	→
||	→
? :	←
= += –= *= /= %= &= ^= |= <<= >>=	←
, (toán tử phẩy)	←

Ví dụ:

```
n1 = 2 + 3 * 5;
n2 = (2 + 3) * 5;            // Khac n1
n3 = 2 + (3 * 5);            // Bang n1
```

Phép nhân sẽ được ưu tiên hơn phép cộng nên n1 sẽ bằng n3.

Để tránh khó hiểu (có thể dẫn đến kết quả không như mong đợi) ta nên sử dụng các cặp ngoặc đơn ( ). Ví dụ:

```
Ketqua = a > 1 && b < 2;
```

Tuy phép so sánh thực hiện trước rồi mới tới phép && nhưng ta nên thêm các cặp ngoặc đơn để câu lệnh trở nên dễ hiểu.

```
Ketqua = (a > 1) && (b < 2);
```

7.3 Các lệnh nhập xuất

7.3.1 Xuất dữ liệu ra màn hình

Để xuất dữ liệu ra màn hình ta sử dụng hàm sau:

Hàm xuất dữ liệu ra màn hình (#include <stdio.h>)
`printf(<chuỗi định dạng>[, <đối số 1>, … ]);`

Chuỗi định dạng (được đặt giữa cặp nháy kép " ") là cách trình bày thông tin sẽ được xuất. Một chuỗi định dạng có ba thành phần:

- Văn bản thường (literal text) trong chuỗi định dạng sẽ được xuất y hệt lúc gõ. Ví dụ sau sẽ xuất chuỗi **Tin hoc co so A** và chuỗi **Chương 7** ra màn hình. Chuỗi sau xuất ngay sau chuỗi trước.

```
printf("Tin hoc co so A");
printf("Chương 7");
```

- Ký tự điều khiển (escape sequence) gồm dấu \ và một ký tự. Các ký tự điều khiển bao gồm:

Ký tự điều khiển	*Ý nghĩa*
\a	Tiếng chuông
\b	Lùi lại một bước
\n	Xuống dòng
\t	Dấu tab
\\	In dấu \
\?	In dấu ?
\“	In dấu “

Ví dụ:

```
printf("Tin hoc co so A");
printf("\nChuong 7");
```

Kết quả:
Tin hoc co so A
Chuong 7

- Đặc tả (conversion specifier) gồm dấu phần % và một ký tự. Phần này dùng để xác định kiểu của biến muốn xuất. Biến muốn xuất sẽ được đặt ở phần đối số. Nếu muốn xuất nhiều biến thì các biến sẽ được liệt kê cách nhau bằng dấu phẩy.

Đặc tả	*Ý nghĩa*	*Kiểu dữ liệu phù hợp*
%c	Ký tự đơn	char
%d	Số nguyên có dấu	int, short, long
%f	Số thực	float, double
%s	Chuỗi ký tự	char[], char*
%u	Số nguyên không dấu	unsigned int/short/long

Ví dụ:

```
int a = 2912, b = 1706;
printf("%d cong %d bang %d", a, b, a + b);
```

Kết quả: *2912 cong 1706 bang 4618*

Lưu ý, thứ tự các đối số phải tương ứng thứ tự các đặc tả và phải chính xác (nghĩa là kiểu dữ liệu nào thì dùng đặc tả đó) nếu không kết quả xuất có thể sẽ không như ý muốn.

Thông thường, khi xuất các dữ liệu kiểu số (số nguyên, số thực) ta có nhu cầu định dạng các thể hiện thông tin ra màn hình do cách xuất thông thường không đẹp mắt và khó đọc.

Ví dụ:

```
#include <stdio.h>
void main()
{
    int a = 2912, b = 176;
    float c = 176.85;
    printf("%d", a);
    printf("%d", b);
    printf("%f", c);
}
```

Kết quả :

2912
176
1.7685000000

Cách xuất số thực trên rất khó chịu vì hệ thống tự động thêm các số 0 ở phần lẻ. Để có thể xác định số chữ số lẻ muốn xuất cũng như số ô trên màn hình để biểu diễn số, ta sử dụng cú pháp:

Định dạng số nguyên và số thực

```
%nd         Dùng n ô để in số nguyên.
%n.kf       Dùng n ô để in số thực và lấy k số lẻ.
            n = 0 hoặc bỏ nếu không quan tâm số ô)
```

Ví dụ:

```
#include <stdio.h>
void main()
{
    int a = 2912, b = 176;
    float c = 176.85;

    printf("%10d", a);
    printf("%10d", b);
    printf("%10.2f", c);
    printf("%.2f", c);
}
```

Kết quả (theo ô trên màn hình):

						2	9	1	2
							1	7	6
				1	7	6	.	8	5
1	7	6	.	8	5				

7.3.2 Nhập dữ liệu từ bàn phím

Để nhập dữ liệu ra màn hình ta sử thủ tục sau:

Hàm nhập dữ liệu từ bàn phím (#include <stdio.h>)
`scanf(<chuỗi định dạng>[, <đối số 1>, … ]);`

Đối số là tên biến được đặt trước dấu &. Chuỗi định dạng cũng giống như trong lệnh xuất printf nhưng không được có các văn bản thường.

Ví dụ:

```
#include <stdio.h>
void main()
{
    int a, b;
    scanf("%d%d", &a, &b);
}
```

Nhập các biến cách nhau khoảng trắng, tab hoặc xuống dòng.

Bài tập cuối chương

Lý thuyết

1. Trình bày tóm tắt các kiểu dữ liệu cơ sở trong C. Cho một số ví dụ kiểu dữ liệu cụ thể trong mỗi loại (tên kiểu, độ lớn theo byte, miền giá trị).
2. Trình bày khái niệm về biến và cách sử dụng lệnh gán.
3. Phân biệt hằng thường và hằng ký hiệu. Cho ví dụ.
4. Trình bày khái niệm về biểu thức. Tại sao nên sử dụng cặp ngoặc đơn ( ) trong biểu thức?
5. Trình bày cách định dạng khi xuất các số nguyên và số thực. Cho ví dụ minh họa.

Thực hành

Viết chương trình giải các bài tập 4 đến 9 trong chương 5 (các bài tập này đã biểu diễn thuật toán bằng sơ đồ khối).

Chương 8.
CÂU LỆNH ĐIỀU KIỆN VÀ RẼ NHÁNH

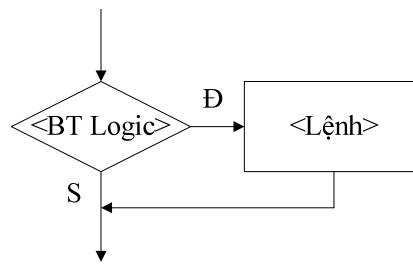
Chương này tập trung trình bày chi tiết câu lệnh điều kiện if...else và câu lệnh rẽ nhánh switch đồng thời so sánh đặc điểm giữa hai câu lệnh nhằm giúp lập trình viên có thể lựa chọn câu lệnh phù hợp với từng trường hợp.

8.1 Câu lệnh điều kiện if... else

8.1.1 Khái niệm

Câu lệnh if là câu lệnh làm cho chương trình thực hiện hay không thực hiện câu lệnh nào đó tùy vào điều kiện nêu ra.

8.1.2 Cú pháp

<table>
<tr><td colspan="2">Câu lệnh If (thiếu)
Thực hiện <Lệnh> nếu <BT Logic> đúng (true, khác 0).
Ngược lại, <BT Logic> sai (false, bằng 0) sẽ không làm gì cả.</td></tr>
<tr><td>Sơ đồ khối</td><td> 
</td></tr>
<tr><td>Cú pháp</td><td><pre>if (<BT Logic>)
    <Lệnh>;</pre></td></tr>
</table>

Trong đó:

- <BT Logic> cho kết quả đúng (true, khác 0) hoặc sai (false, bằng 0) và được đặt trong cặp dấu ngoặc đơn ( ).
- <Lệnh> là câu lệnh đơn hoặc khối lệnh (kẹp các câu lệnh đơn giữa { và }).

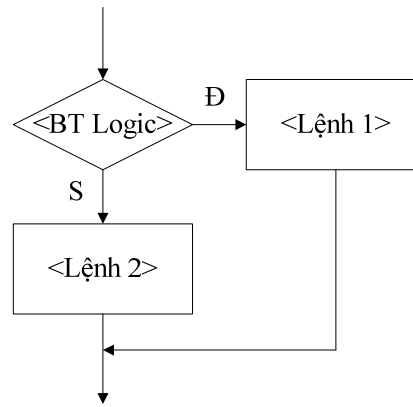
Ví dụ:

Nếu a khác không thì xuất thông báo "a khác 0".

```
if (a != 0)
    printf("a khac 0.");
```

Câu lệnh If (đủ)

Thực hiện <Lệnh 1> nếu <BT Logic> đúng (khác 0).
Ngược lại, <BT Logic> sai (bằng 0) sẽ thực hiện <Lệnh 2>.

Sơ đồ khối	

Cú pháp	`if (<BT Logic>)`
	`  <Lệnh 1>;`
	`else`
	`  <Lệnh 2>;`

Ví dụ: Tùy theo giá trị của a để xuất thông báo tương ứng.

```
if (a != 0)
    printf("a khac 0.");
else
    printf("a bang 0.");
```

8.1.3 Một số lưu ý

Về mặt cú pháp, câu lệnh if… else là một câu lệnh đơn. Do đó không cần phải phức hóa câu lệnh này bằng cách kẹp giữa { và }.

Các câu lệnh if có thể lồng vào nhau và theo nguyên tắc, phần else sẽ thuộc if gần nhất. Tuy nhiên, để cho rõ ràng và dễ hiểu chúng ta nên đặt vào đúng cấp đồng thời thêm { và }.

```
if (a != 0)
{
    if (b > 0)
        printf("a!=0 và b>0.");
    else
        printf("a!=0 và b<=0.");
}
else
    printf("a = 0.");
```

Không được thêm ; sau biểu thức điều kiện vì như thế câu phần theo sau if bây giờ không còn thuộc if nữa. Nói cách khác, cho dù biểu thức điều kiện là đúng hay sai thì các lệnh tiếp theo cũng sẽ được thực hiện. Ví dụ:

```
int a = 0;
if (a != 0);                    // Phai bo ;
    printf("a khac 0.");        // Luon thuc hien
```

8.2 Câu lệnh rẽ nhánh switch

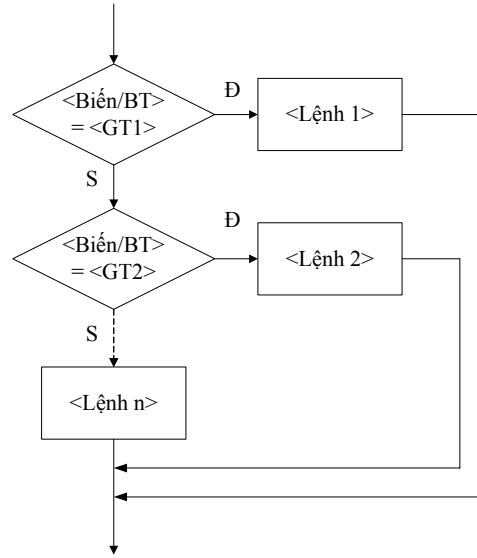
8.2.1 Khái niệm

Câu lệnh switch là câu lệnh làm chương trình thực hiện chọn lựa một trong nhiều hành động để thực hiện.

8.2.2 Cú pháp

Câu lệnh switch
<Biến/BT> bằng <GT> nào sẽ thực hiện <Lệnh> đó.
Nếu không tìm thấy <GT> tương ứng sẽ thực hiện <Lệnh n>.

Sơ đồ khối



Cú pháp

```
switch (<Biến/BT>)
{
     case <GT1> : <Lệnh 1>;[break;]
     case <GT2> : <Lệnh 2>;[break;]
     ...
     [default:
          <Lệnh n>;]
}
```

Trong đó:

- <Biến/BT> là biến hay biểu thức cho ra dữ liệu kiểu rời rạc, hữu hạn, đếm được như các kiểu số nguyên, ký tự, liệt kê và được đặt trong cặp dấu ngoặc đơn ( ). **Không được sử dụng kiểu số thực.**
- <Lệnh 1>, …, <Lệnh n> là một hay nhiều câu lệnh đơn.
- Cuối mỗi trường hợp (case) sẽ có hoặc không có lệnh break. Khi gặp lệnh này, lệnh switch sẽ kết thúc.
- Phần default có thể có hoặc không. Nếu không tìm thấy trường hợp nào phù hợp, phần này sẽ được thực hiện.

Ví dụ:

Lệnh switch sau sẽ xuất thông báo "Một" nếu a bằng 1, xuất thông báo "Hai" nếu a bằng 2, xuất thông báo "Ba" nếu a bằng 3. Nếu không sẽ xuất "a<1 hoặc a>3" và "Khong biet doc!".

```
switch (a)
{
    case 1 : printf("Mot"); break;
    case 2 : printf("Hai"); break;
    case 3 : printf("Ba"); break;
    default:
       printf("a<1 hoac a>6");
       printf("\nKhong biet doc!");
}
```

8.2.3 Một số lưu ý

Về mặt cú pháp, câu lệnh switch là một câu lệnh đơn. Các câu lệnh switch cũng có thể lồng vào nhau.

```
switch (a)
{
    case 1 : printf("Mot"); break;
    case 2 : switch (b)
             {
                case 1 : printf("Hai-Mot");break;
                case 2 : printf("Hai-Hai");break;
             } break;
```

```
    case 3 : printf("Ba"); break;
    default:
       printf("Khong biet doc!");
}
```

Lệnh break sau mỗi trường hợp (case) rất quan trọng. Nếu thiếu thì sau khi thực hiện trường hợp tương ứng nó sẽ thực hiện tiếp trường hợp liền sau nó đến khi gặp lệnh break hoặc kết thúc switch. Trường hợp (case) cuối cùng không cần lệnh break.

Ví dụ sau đây do thiếu break nên sẽ in **MotHaiBa** khi a = 1.

```
switch (a)
{
    case 1: printf("Mot");
    case 2: printf("Hai");
    case 3: printf("Ba");
}
```

Tuy nhiên, trong một số trường hợp, việc bỏ break cũng có lợi nếu như ta muốn nhóm các trường hợp cùng loại. Ví dụ, thông thường ta sẽ viết như sau nếu muốn xét các số chẵn lẻ.

```
switch (a)
{
    case 1: printf("So le"); break;
    case 2: printf("So chan"); break;
    case 3: printf("So le"); break;
    case 4: printf("So chan"); break;
    case 5: printf("So le"); break;
}
```

Ta viết lại như sau sẽ gọn hơn:

```
switch (a)
{
    case 1:
    case 3:
    case 5: printf("So le"); break;
    case 2:
    case 4: printf("So chan"); break;
}
```

8.3 Kinh nghiệm sử dụng câu lệnh điều kiện và rẽ nhánh

Câu lệnh if và câu lệnh switch đều rẽ nhánh để thi hành câu lệnh. Tuy vậy, hai câu lệnh này có một số điểm khác nhau cơ bản. Dưới đây là một số tổng kết quan trọng cần lưu ý:

- Câu lệnh if sẽ chọn một trong tối đa hai câu lệnh để thi hành trong khi câu lệnh switch sẽ chọn một trong nhiều câu lệnh để thi hành.
- Điều kiện rẽ nhánh của câu lệnh if là giá trị của biến hoặc biểu thức có đúng sai hoặc kiểu số (0 tương ứng với sai và khác 0 tương ứng với đúng) trong khi điều kiện rẽ nhánh của câu lệnh switch là giá trị của biến có kiểu rời rạc, hữu hạn, đếm được.
- Trong câu lệnh switch không có biểu thức điều kiện như <, =, >… hoặc một miền giá trị. Trong trường hợp này ta cũng phải sử dụng câu lệnh if thay thế.
- Những gì câu lệnh switch làm được thì câu lệnh if cũng sẽ làm được nhưng ngược lại câu lệnh switch có thể sẽ không thực hiện một số trường hợp của câu lệnh if (ví dụ có sử dụng biểu thức so sánh). Câu lệnh switch thường được sử dụng rất hiệu quả trong việc rút gọn một loạt liên tiếp các câu lệnh if.

Ví dụ:

Các câu lệnh if sau nên thay bằng một câu lệnh switch:

```
if (a == 1)
    printf("Mot");
if (a == 2)
    printf("Hai");
if (a == 3)
    printf("Ba");
```

Bài tập cuối chương

Lý thuyết

1. Nêu cú pháp và vẽ sơ đồ khối của câu lệnh if và câu lệnh switch. Chỉ rõ điều kiện của các thành phần trong cú pháp.
2. Cho các ví dụ tương ứng với các lưu ý trong phần *8.3 Kinh nghiệm sử dụng câu lệnh điều kiện và câu lệnh rẽ nhánh*.

Thực hành

3. Nhập một số nguyên bất kỳ. Hãy đọc giá trị của số nguyên đó nếu nó có giá trị từ 0 đến 9, ngược lại thông báo không đọc được.
4. Giải phương trình bậc nhất $ax + b = 0$ với a, b là các số thực nhập từ bàn phím.
5. Giải phương trình bậc hai $ax^2 + bx + c = 0$ với a, b, c là các số thực nhập từ bàn phím.
6. Nhập 4 số nguyên a, b, c và d từ bàn phím. Hãy tìm số có giá trị nhỏ nhất (hoặc lớn nhất).
7. Nhập 4 số nguyên a, b, c và d từ bàn phím. Hãy sắp xếp giá trị của 4 số nguyên này theo thứ tự tăng dần (hoặc giảm dần).

   8. Nhập điểm trung bình của một học sinh. Cho biết kết quả phân loại biết đtb ≥ 9.0 : xuất sắc; 8.0 ≤ đtb < 9.0 : giỏi; 6.5 ≤ đtb < 8.0 : khá; 5.0 ≤ đtb < 6.5 : trung bình; 3.5 ≤ đtb < 5.0 : yếu; đtb < 3.5 : kém;
9. Tính tiền đi taxi từ số km nhập vào. Biết:
   a. 1 km đầu giá 15000đ
   b. Từ km thứ 2 đến km thứ 5 giá 13500đ
   c. Từ km thứ 6 trở đi giá 11000đ
   d. Nếu đi hơn 120 km sẽ được giảm 10% trên tổng số tiền.

Chương 9.
CÂU LỆNH LẶP

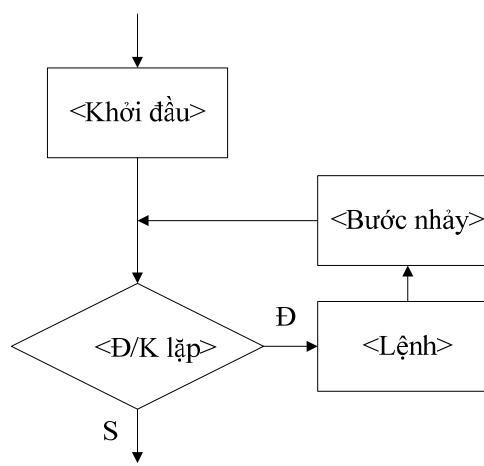
Chương này sẽ lần lượt trình bày 3 câu lệnh lặp với những tính năng khác nhau để có thể sử dụng linh hoạt trong từng trường hợp cụ thể.

9.1 Câu lệnh for

9.1.1 Khái niệm

Câu lệnh for là câu lệnh làm cho chương trình thực hiện lặp lại một hay nhiều hành động một số lần nhất định.

9.1.2 Cú pháp

<table>
<tr><th colspan="2">Câu lệnh For</th></tr>
<tr><td>Sơ đồ khối</td><td> 
</td></tr>
<tr><td>Cú pháp</td><td><pre>for (<Khởi đầu>;<Đ/K lặp>;<Bước nhảy>)
        <Lệnh>;</pre></td></tr>
</table>

Trong đó:

- <Khởi đầu>: là một biểu thức C bất kỳ, thường là một câu lệnh gán để đặt cho biến đếm một giá trị cụ thể.
- <Đ/K lặp>: là một biểu thức C bất kỳ, thường là một biểu thức quan hệ cho kết quả true-false. Nếu <Đ/K lặp> nhận giá trị false (bằng 0), câu lệnh for sẽ kết thúc. Ngược lại, <Lệnh> sẽ được thực hiện.
- <Bước nhảy>: là một biểu thức C bất kỳ, thường là một biểu thức nhằm tăng giá trị biến đếm trong biểu thức khởi đầu. Biểu thức này sẽ được thi hành sau khi <Lệnh> được hiện.
- <Lệnh>: là câu lệnh đơn hoặc khối lệnh.

Ví dụ:

Câu lệnh sau sẽ in các số từ 0 đến 9 ra màn hình.

```
int i;
for (i = 0; i < 10; i++)
    printf("%d\n", i);
```

Ta cũng có thể vừa khai báo và khởi tạo biến trong phần khởi đầu của for:

```
for (int i = 0; i < 10; i++)
    printf("%d\n", i);
```

Thực hiện nhiều câu lệnh đồng thời (khối lệnh):

```
for (int i = 0; i < 10; i++)
{
      printf("%d", i);
      printf("\n");
}
```

9.1.3 Một số lưu ý

Về mặt cú pháp, câu lệnh for là một câu lệnh đơn. Do đó không cần phải phức hóa câu lệnh này bằng cách kẹp cặp ngoặc nhọn { }. Ngoài ra, các câu lệnh for có thể lồng vào nhau.

Ví dụ, có thể bỏ { và } trong trường hợp này:

```
if (n < 10 && m < 20)
{
    for (int i = 0; i < n; i++)
    {
        for (int j = 0; j < m; j++)
            printf("%d\n", i + j);
    }
}
```

Trong câu lệnh for, có thể sẽ không có phần <Khởi đầu> do phần này được mang ra trước lệnh for.

```
int i = 0;                  // Day la khoi dau
for (; i < 10; i++)
    printf("%d\n", i);
```

Trong câu lệnh for, có thể sẽ không có phần <Bước nhảy> do phần này được mang vào trong thân lệnh for.

```
int i = 0;
for (; i < 10;)
{
    printf("%d\n", i);
    i++;                    // Day la buoc nhay
}
```

Trong câu lệnh for, cũng có thể không có phần <Đ/K lặp> nhưng lúc này lệnh for sẽ lập vô tận (loop) nếu không có lệnh break để thoát khỏi vòng lặp for.

```
int i = 0;
for (; ;)                    // Lap vo tan (loop)
{
    printf("%d\n", i);
    i++;
}
```

Không được thêm ; ngay sau câu lệnh for vì như thế câu phần theo sau for bây giờ không còn thuộc for nữa. Nói cách khác, mỗi lần lặp của for sẽ không có lệnh nào được thực hiện và những lệnh sau for sẽ được thực hiện sau khi for kết thúc.

```
for (int i = 0; i < 10; i++);     // Phai bo ;
    printf("%d\n", i);
```

Các thành phần <Khởi đầu>, <Đ/K lặp>, <Bước nhảy> cách nhau bằng dấu chấm phẩy. Nếu có nhiều phần trong mỗi phần thì các thành phần này sẽ cách nhau bằng dấu phẩy. Ví dụ :

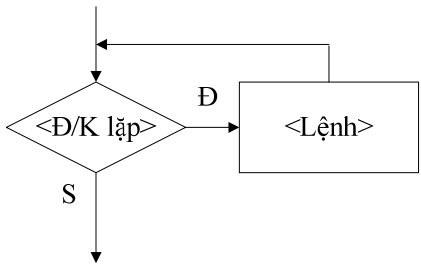
```
for (int i = 1, j = 2; i + j < 10; i++, j += 2)
      printf("%d\n", i + j);
```

9.2 Câu lệnh while

9.2.1 Khái niệm

Câu while là câu lệnh làm cho chương trình thực hiện lặp lại nhiều lần một số hành động cho trước trong khi vẫn còn thỏa một điều kiện để tiếp tục quá trình lặp.

9.2.2 Cú pháp

Câu lệnh while	
Sơ đồ khối	
Cú pháp	`while (<Đ/K lặp>)` `  <Lệnh>;`

Trong đó :

- <Đ/K lặp>: là một biểu thức C bất kỳ, thường là một biểu thức quan hệ cho kết quả true-false và được đặt trong cặp dấu ngoặc đơn ( ). Nếu <Đ/K lặp> nhận giá trị false (bằng 0), câu lệnh for sẽ kết thúc. Ngược lại, <Lệnh> sẽ được thực hiện.
- <Lệnh>: là câu lệnh đơn hoặc khối lệnh.

Ví dụ, vòng lặp sau sẽ xuất giá trị của n bắt đầu từ 0 cho đến 9.

```
int n = 0;
while (n < 10)
{
    printf("%d\n", n);
    n++;
}
```

9.2.3 Một số lưu ý

Về mặt cú pháp, câu lệnh while là một câu lệnh đơn. Do đó không cần phải phức hóa câu lệnh này bằng cách kẹp cặp dấu ngoặc nhọn { }. Ngoài ra, các câu lệnh while có thể lồng vào nhau.

Ví dụ, có thể bỏ { và } trong trường hợp này:

```
if (n < 10)
    while (n >= 1)
        printf("%d\n", n--);
```

Vòng lặp while có thể sẽ không thực hiện lần nào do ngay lần đầu tiên kiểm tra điều kiện lặp đã không thỏa.

```
int n = 1;
while (n > 10)
{
    printf("%d\n", n);
    n--;
}
```

Nếu không cẩn thận (trừ khi có chủ ý) vòng lặp while sẽ không thể thoát và lặp vô tận do điều kiện lặp luôn đúng.

Ví dụ:

Lặp vô tận do quên tăng giá trị biến n nên n < 10 luôn đúng.

```
int n = 1;
while (n < 10)
    printf("%d\n", n);
```

Tương tự vòng lặp for, không được thêm ; ngay sau điều kiện lặp của câu lệnh while vì như thế câu phần theo sau while bây giờ không còn thuộc while nữa.

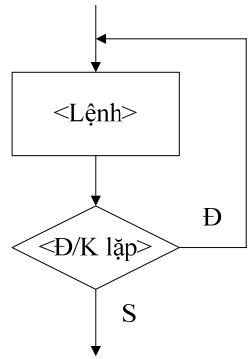
```
int n = 0;
while (n < 10);                 // Phai bo ;
{
    printf("%d\n", n);
    n++;
}
```

9.3 Câu lệnh do… while

9.3.1 Khái niệm

Câu lệnh do... while là câu lệnh làm cho chương trình thực hiện lặp lại nhiều lần một số hành động cho trước trong khi vẫn còn thỏa một điều kiện để tiếp tục quá trình lặp. Câu lệnh này ngược với câu lệnh while ở trên một chỗ là điều kiện lặp được kiểm tra sau khi các lệnh đã được thực hiện.

9.3.2 Cú pháp

Câu lệnh do... while	
Sơ đồ khối	

Cú pháp	`do`
	`  <Lệnh>;`
	`while (<Đ/K lặp>);`

<Đ/K lặp> và <Lệnh> giống như lệnh while ở trên.

Ví dụ, câu lệnh sau sẽ xuất các số từ 0 đến 9.

```
n = 0;
do
{
    printf("%d\n", n);
    n++;
}
while (n < 10);
```

Ứng dụng của câu lệnh này là tạo vòng lặp nhập dữ liệu cho một biến sao cho biến đó có giá trị giới hạn trong một phạm vi nào đó. Đoạn chương trình sau đây yêu cầu người sử dụng nhập vào giá trị cho biến n trong đoạn từ 1 đến 100.

```
int n;
do
{
     printf("Nhap n : ");
     scanf("%d", &n);
}
while (n < 1 || n > 100);
```

9.3.3 Một số lưu ý

Về mặt cú pháp, câu lệnh do… while là một câu lệnh đơn. Do đó không cần phải phức hóa câu lệnh này bằng cách kẹp giữa cặp ngoặc nhọn { }. Ngoài ra, các câu lệnh do… while có thể lồng vào nhau.

```
int a = 1, b;
do
{
    b = 1;
    do
    {
        printf("%d", a + b);
        b = b * 2;
     }
    while (b < 20);
    a++;
}
while (a < 10);
```

Vòng lặp do… while sẽ được thực hiện ít nhất một lần nên có thể sử dụng để kiểm tra giá trị nhập vào vì vòng lặp sẽ cho phép nhập ít nhất một lần rồi mới kiểm tra (ví dụ ở phần trên).

Vòng lặp do… while có khả năng bị lặp vô tận.

```
int n = 1;
do
    printf("%d", n);
while (n < 10);
```

9.4 Kinh nghiệm sử dụng các câu lệnh lặp

Các lệnh lặp trên đều là các câu lệnh đơn có chức năng lặp lại một hay nhiều hành động. Tuy nhiên, các lệnh này còn có một điểm khác nhau cơ bản là vòng lặp while và vòng lặp do… while thực hiện số lần lặp không xác định trong khi vòng lặp for thực hiện số lần lặp có thể xác định ngay trong cú pháp. Do đó khi biết trước số lần lặp ta sẽ sử dụng lệnh for, ngược lại sẽ sử dụng lệnh while hoặc lệnh do... while.

Điểm khác nhau cơ bản giữa vòng lặp while và vòng lặp do... while đó là vòng lặp while kiểm tra điều kiện rồi mới vào vòng lặp nên có khả năng không thực hiện lần nào trong khi đó vòng lặp do… while thực hiện vòng lặp rồi mới kiểm tra điều kiện do đó nó sẽ thực hiện ít nhất 1 lần. Do đó nếu xét một điều kiện có thể sẽ không thực hiện lần nào ta nên sử dụng vòng lặp while.

Bài tập cuối chương

Lý thuyết

1. Nêu cú pháp và vẽ sơ đồ khối của câu lệnh for, lệnh while và lệnh do… while. Chỉ rõ điều kiện của các thành phần trong cú pháp.
2. Cho các ví dụ tương ứng với các lưu ý trong phần 9.*4 Kinh nghiệm sử dụng các câu lệnh lặp*.

Thực hành

3. Nhập một số nguyên dương n. Hãy cho biết:
   a. Có phải là số đối xứng?
      Là số nghịch đảo bằng chính nó. Ví dụ: 121, ...
   b. Có phải là số chính phương?
      Là số bằng bình phương số khác. Ví dụ: 4, 9, ...
   c. Có phải là số nguyên tố?
      Là số lớn hơn 1 và chỉ có 2 ước số là 1 và nó.
      Ví dụ: 2, 3, 5, 7, 11, 13, ...
   d. Chữ số lớn nhất và nhỏ nhất?
      Ví dụ: số 1706, nhỏ nhất 0 và lớn nhất 7
   e. Các chữ số có tăng dần hay giảm dần không?
      Ví dụ: 12245, 156, 442, 941, ...
4. Nhập số nguyên n. Tính:
   a. $S = 1 + 2 + \ldots + n$
   b. $S = 1^2 + 2^2 + \ldots + n^2$
   c. $S = 1 + 1/2 + \ldots + 1/n$
   d. $S = 1! + 2! + \ldots + n!$
5. Nhập 3 số nguyên n, a, b (a, b < n).
   Tính tổng các số chia hết cho a nhưng không chia hết cho b và nhỏ hơn n.
6. Tính tổng các số nguyên tố nhỏ hơn n (0 < n < 50).
7. Nhập một số nguyên dương. Xuất ra số ngược lại.
8. Tìm và in lên màn hình tất cả các số nguyên trong phạm vi từ 10 đến 99 sao cho tích của 2 chữ số bằng 2 lần tổng của 2 chữ số đó.
9. Tìm các ước số chung nhỏ nhất của 2 số nguyên dương
10. In n số đầu tiên trong dãy Fibonacy.
    Dãy Fibonacy là dãy $a_0, a_1, \ldots, a_{n-2}, a_{n-1}, a_n$ với:
    a. $a_0 = a_1 = 1$
    b. $a_n = a_{n-1} + a_{n-2}$ $(n \geq 2)$
    Ví dụ: 1 1 2 3 5 8 13 21…

Chương 10.
HÀM

Trong thực tế, khi ta muốn giải quyết một công việc phức tạp nào đó, ta thường chia nhỏ công việc đó thành các công việc nhỏ hơn và tất nhiên những công việc nhỏ này lại thực hiện dễ dàng hơn rất nhiều. Thực vậy, trong lập trình ta cũng có nhu cầu chia nhỏ chương trình phức tạp thành những chương trình nhỏ hơn, đơn giản và dễ hiểu. Mỗi chương trình nhỏ đó được gọi là hàm.

Chương này trình bày các khái niệm cơ bản về hàm, cách viết và sử dụng hàm, đặc biệt là kỹ thuật đệ quy giúp chương trình ngắn gọn hơn rất nhiều.

10.1 Khái niệm

Hàm là một đoạn chương trình có tên và có chức năng giải quyết một số vấn đề chuyên biệt cho chương trình chính, nó có thể được gọi nhiều lần với các tham số khác nhau và trả lại một giá trị nào đó cho chương trình gọi nó.

Hàm thường được sử dụng khi:

- **Nhu cầu tái sử dụng:** có một số công việc được thực hiện ở nhiều nơi (cùng một chương trình hoặc ở nhiều chương trình khác nhau), bản chất không đổi nhưng giá trị các tham số cung cấp khác nhau ở từng trường hợp.
- **Nhu cầu sửa lỗi và cải tiến:** giúp phân đoạn chương trình để chương trình được trong sáng, dễ hiểu và do đó rất dễ dàng phát hiện lỗi cũng như cải tiến chương trình.

10.2 Cú pháp

10.2.1 Cú pháp

Hàm có cấu trúc tổng quát như sau:

```
<kiểu trả về> <tên hàm>([<danh sách tham số>])
{
    <các câu lệnh>
    [return <giá trị>;]
}
```

Trong đó:

- <kiểu trả về>: là bất kỳ kiểu dữ liệu nào của C như char, int, long, float hay double… Nếu hàm đơn thuần chỉ thực hiện một số câu lệnh mà không cần trả về cho chương trình gọi nó thì kiểu trả về này là void.
- <tên hàm>: là tên gọi của hàm và được đặt theo quy tắc đặt tên/định danh.
- <danh sách tham số>: xác định các đối số sẽ truyền cho hàm. Các tham số này giống như khai báo biến và cách nhau bằng dấu phẩy. Hàm có thể không có đối số nào.
- <các câu lệnh>: là các câu lệnh sẽ được thực hiện mỗi khi hàm được gọi.
- <giá trị>: là giá trị trả về cho hàm thông qua câu lệnh return.

Ví dụ:

Hàm sau đây có tên là Tong, nhận vào hai đối số kiểu nguyên và trả về tổng của hai số nguyên đó.

```
/* Ham ten Tong
Nhan vao hai so nguyen va tra ve mot so nguyen */
int Tong(int a, int b)
{
    return a + b;
}
```

Hàm sau đây có tên là Xuat, nhận vào một đối số kiểu nguyên và xuất số nguyên đó ra màn hình. Hàm này không trả về gì cả.

```
void Xuat(int n)
{
    printf("%d", n);
}
```

Hàm sau đây có tên là Nhap, không nhận đối số nào cả và trả về giá trị số nguyên người dùng nhập vào.

```
int Nhap()
{
    int n;
    printf("Nhap mot so nguyen: ");
    scanf("%d", &n);
    return n;
}
```

10.2.2 Một số lưu ý

Hàm phải được khai báo và định nghĩa trước khi sử dụng và thường đặt ở trên hàm chính (hàm main).

```
int Tong(int a, int b)
{
    return a + b;
}
void main()
{
    int a = 2912, b = 1706;
    int sum = Tong(a, b);   // Loi goi ham
}
```

Thông thường, trước hàm main ta chỉ xác định tên hàm, các tham số và giá trị trả về của hàm để thông báo cho các hàm bên dưới biết cách sử dụng của nó còn phần định nghĩa hàm sẽ được đưa xuống dưới cùng. Phần ở trên này được gọi là nguyên mẫu

hàm (function prototype). Nguyên mẫu hàm chính là tiêu đề hàm được kết thúc bằng dấu chấm phẩy.

```
int Tong(int a, int b);      // prototype ham Tong

void main()
{
    int a = 2912, b = 1706;
    int sum = Tong(a, b);    // Loi goi ham
}

int Tong(int a, int b)       // Mo ta ham Tong
{
    return a + b;
}
```

Trên thực tế, nguyên mẫu hàm không cần thiết phải giống tuyệt đối tiêu đề hàm. Tên tham số có thể khác hoặc bỏ luôn miễn là cùng kiểu. Tuy nhiên, không nên để chúng khác nhau vì như vậy sẽ gây rối cho chương trình.

Ví dụ sau cho thấy có thể bỏ hẳn tên tham số:

```
int Tong(int, int);          // prototype ham Tong
…
```

10.3 Tầm vực của biến và hàm

10.3.1 Các loại biến

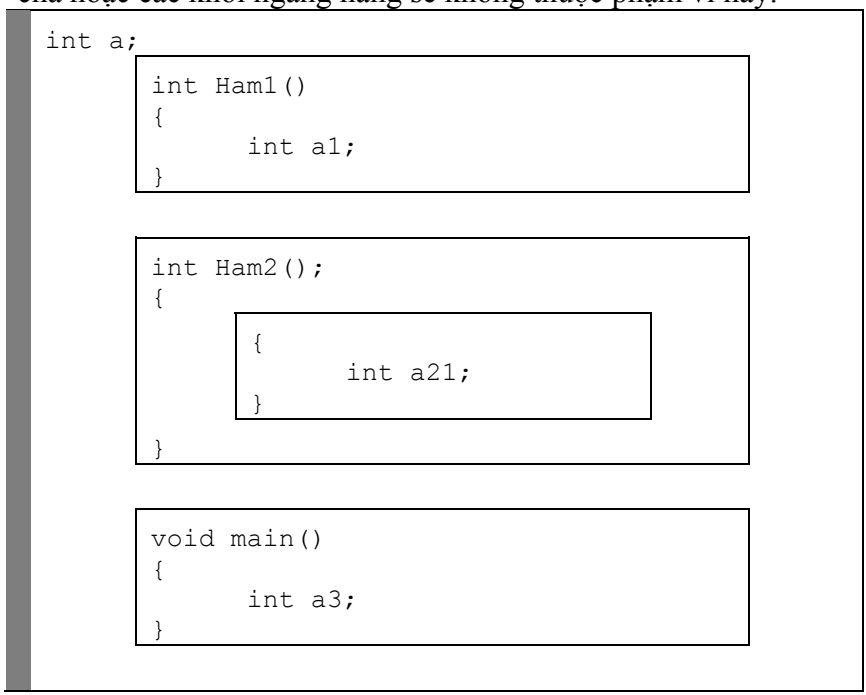
Trong C có hai loại biến được phân theo phạm vi hiệu quả của nó, đó là :

- **Biến toàn cục (Global Variable):** được khai báo ngoài tất cả các hàm và có tác dụng lên toàn bộ chương trình.
- **Biến cục bộ (Local Variable):** được khai báo trong hàm hoặc khối { } và chỉ có tác dụng trong bản thân hàm đó hoặc khối đó. Các biến cục bộ không có tác dụng đối với hàm và các khối khác ngoài hàm hoặc ngoài khối đã khai

báo nó. Biến cục bộ sẽ bị xóa bỏ khỏi bộ nhớ khi kết thúc hàm hoặc khối khai báo nó.

10.3.2 Tầm vực

Là phạm vi hiệu quả của biến hoặc hàm. Phạm vi này bao gồm bản thân khối đó và các khối con bên trong nó. Các khối cha hoặc các khối ngang hàng sẽ không thuộc phạm vi này.



Nhận xét :

- Các hình chữ nhật bao quanh tạo thành một khối. Một khối có thể chứa khối con trong nó.
- Biến khai báo trong khối nào thì chỉ có tác dụng trong khối đó và các khối con của nó, không có tác dụng với khối cùng cấp.

- Biến khai báo trong khối lớn nhất (chứa tất cả các khối khác) là biến toàn cục.
- Biến khai báo trong các hàm (hoặc khối) là cục bộ, sẽ bị mất khi kết thúc hàm (hoặc khối).
- Hàm cùng một khối (cùng cấp) có thể gọi lẫn nhau nhưng phải tuân theo thứ tự khai báo.
- Các biến cục bộ nên đặt khác tên với các biến ở khối cha để tránh nhầm lẫn. Trong trường hợp đặt trùng tên thì biến được ưu tiên là biến cục bộ của khối con.

Ở ví dụ trên, *a* là biến toàn cục, có thể sử dụng ở bất cứ đâu. *a1*, *a2*, *a21*, *a3* là các biến cục bộ do được khai báo trong hàm (hoặc khối). *a1* chỉ có tác dụng trong hàm *Ham1*; *a2* có tác dụng trong thủ tục *Ham2* và khối trong *Ham2*; *a21* chỉ có tác dụng trong khối mà nó khai báo; *a2* chỉ có tác dụng trong hàm *main.* Hàm *main* có thể gọi *Ham1*, *Ham2*. *Hàm 2* có thể gọi *Ham1*.

10.4 Tham số và lời gọi hàm

10.4.1 Các cách truyền tham số

Tham số trong hàm định kiểu đối số mà chương trình chính truyền khi gọi hàm. Có hai cách truyền tham số sau đây:

Truyền Giá trị (Call by Value)

- Truyền đối số cho hàm ở dạng giá trị.
- Được sử dụng khi ta không có nhu cầu thay đổi giá trị của tham số sau khi thực hiện hàm.

Truyền Địa chỉ (Call by Address)

- Truyền đối số cho hàm ở dạng địa chỉ (con trỏ).
- Không được truyền giá trị cho tham số này.
- Được sử dụng khi ta mong muốn sau khi thực hiện hàm thì giá trị của tham số đó sẽ thay đổi.

Trong C++ hỗ trợ thêm cách truyền sau có tác dụng như cách truyền địa chỉ ở trên.

Truyền Tham chiếu (Call by Reference)

- Truyền đối số cho hàm ở dạng địa chỉ (con trỏ).
- Tham chiếu được bắt đầu bằng ký hiệu & trong khai báo.

Ví dụ:

```
void XuLy(int n);          // Truyen gia trị
void XuLy(int *n);         // Truyen dia chi
void XuLy(int &n);         // Truyen tham chieu
```

10.4.2 Lời gọi chương trình con

Muốn hàm thực hiện thì trong hàm gọi nó phải thực hiện lời gọi hàm. Việc này được thực hiện bằng cách gọi tên của hàm đó đồng thời truyền biến hoặc trị cho các tham số mà chương trình con đã khai báo, các biến hoặc trị này cách nhau bằng dấu phẩy.

Ví dụ 1, chương trình sau định nghĩa thủ tục **XuatSo** cho phép xuất các số từ a đến b ra màn hình (a và b được truyền vào hàm). Các tham số đều là tham trị.

```
#include <stdio.h>

void XuatSo(int a, int b);

void main()
{
    XuatSo(5, 20);      // Goi ham lan 1
    int x = 5;
    XuatSo(x, x + 5);   // Goi ham lan 2
}

void XuatSo(int a, int b)
{
    for (int i = a; i <= b; i++)
        printf("%d\n", i);
}
```

Ví dụ 2, chương trình sau định nghĩa hàm **HoanVi** cho phép hoán vị giá trị của 2 số a và b truyền vào thủ tục. Các tham số đều là tham chiếu.

```
#include <stdio.h>

void HoanVi(int &x, int &y);
void main()
{
    int a, b;
    printf("Nhap 2 so a va b: ");
    scanf("%d%d", &a, &b);
    printf("Truoc: a = %d, b = %d", a, b);
    HoanVi(1, 5);
    HoanVi(a, b);
    printf("Sau: a = %d, b = %d", a, b);
}
void HoanVi(int &x, int &y)
{
    int tam = x;
    x = y;
    y = tam;
}
```

Nhập a = 1, b = 5. Sau khi gọi hàm HoanVi(a, b) giá trị của a = 5, b = 1. Cách gọi HoanVi(1, 5) sai do phải là biến.

10.5 Đệ quy

Một hàm có thể gọi một hàm khác vào làm việc, nhưng nếu hàm này gọi chính mình thì đó được gọi là sự đệ quy. Một hàm thực hiện đệ quy khi trong giải thuật có nhiều lần lặp lại chính mình và số lần lặp này phải có giới hạn.

Ví dụ:

Tính S(n) = n ! = 1 * 2 * … * (n-1) * n

Ta nhận thấy S(n) = S(n-1) * n

Vậy thay vì đi tính S(n) ta đi tính S(n-1) rồi nhân với n.

Việc đệ quy này sẽ dừng khi ta tính S(1). Lúc đó ta tính S(1) = S(0) * 1. Như ta đã biết S(0) = 1.

Nếu tiếp tục thực hiện đệ quy với S(0) thì ta sẽ nhận được kết quả sai do S(0) = S(-1) * 0 = 0.

Hàm tính giai thừa theo kiểu đệ quy được viết như sau:

```
int GiaiThua(int n)
{
    if (n == 0)
        return 1;
    return GiaiThua(n-1) * n;
}
```

Bài tập cuối chương

Lý thuyết

1. Hàm là gì? Tại sao phải sử dụng hàm?
2. Trình bày các thành phần của hàm.
3. Trình bày tóm tắt các cách truyền tham số? Phân biệt sự khác nhau giữa các cách này. Cho ví dụ minh họa.
4. Trình bày khái niệm đệ quy và cho ví dụ minh họa.

Thực hành

5. Cài đặt lại các bài tập ở những chương trước bằng chương trình con (thủ tục và hàm).
6. Sử dụng đệ quy để cài đặt các chương trình sau:
    a. Tính S = 1 + 2 + … + n
    b. Tính S = n!
    c. Tính tổng các chữ số của n.
    d. In n phần từ đầu tiên của dãy Fibonacy.

Chương 11.
DỮ LIỆU KIỂU MẢNG (ARRAY)

Trong một số bài toán tổng quát, tại thời điểm lập trình chúng ta chưa thể xác định số lượng biến cần khai báo. Ví dụ như những bài toán tìm số nhỏ nhất trong n số, hoặc sắp xếp n số theo thứ tự tăng dần với số lượng n này người sử dụng sẽ nhập khi chạy chương trình... Dữ liệu kiểu mảng (array) ra đời đã giải quyết được khó khăn trên. Chương này sẽ trình bày chi tiết cách sử dụng dữ liệu kiểu mảng và các vấn đề liên quan.

11.1 Khái niệm

Mảng là một tổ chức kiểu dữ liệu có cấu trúc bao gồm một số cố định các thành phần có cùng kiểu. Mỗi thành phần của mảng được truy xuất thông qua các chỉ số mô tả vị trí của thành phần đó trong mảng.

11.2 Khai báo

11.2.1 Khai báo kiểu mảng

Cú pháp khai báo kiểu mảng (1 chiều)

```
typedef <kiểu cơ sở> <tên kiểu mảng>[<số phần tử>];
```

Trong đó:

- <tên kiểu mảng> : tên kiểu mảng theo quy định đặt tên/định danh.
- <kiểu cơ sở> : kiểu dữ liệu của các thành phần trong mảng.
- <số phần tử> : số lượng phần tử của mảng.

Ví dụ sau khai báo kiểu dữ liệu mảng một chiều có tên MangInt (gồm 20 phần tử kiểu số nguyên) và MangFloat (gồm 30 phần tử kiểu số thực).

```
typedef int MangInt[20];
typedef float MangFloat[30];
```

Muốn khai báo kiểu mảng nhiều chiều, ta sử dụng cú pháp:

Cú pháp khai báo kiểu mảng (nhiều chiều)

```
typedef <kiểu cơ sở> <tên kiểu mảng>[<N1>]…[<Nn>];
```

Với <N1>, …, <Nn> là số phần tử của mỗi chiều.

Ví dụ:

```
typedef int MaTran[25][80];
typedef float MaTranVuong[10][10];
typedef char RuBik[3][3][3];
typedef float DiemHK[100][2];
```

Để dễ dàng thay đổi số phần tử mỗi chiều của mảng tùy nhu cầu sử dụng sau này, ta có thể khai báo chúng như các hằng ký hiệu bằng chỉ thị #define

Ví dụ:

```
#define MAX 100
typedef int MangInt[MAX];
typedef float MangFloat[MAX];
```

Sau này nếu có nhu cầu thay đổi số phần tử của các mảng liên quan đến MAX ta chỉ cần sửa lại duy nhất ở #define.

11.2.2 Khái báo biến mảng

Biến mảng được khai báo trực tiếp (tường minh) như sau:

Cú pháp khai báo mảng 1 chiều (tường minh)

```
<kiểu cơ sở> <tên biến mảng>[<số phần tử>];
```

Cú pháp khai báo biến mảng nhiều chiều (tường minh)

```
<kiểu cơ sở> <tên biến mảng>[<N1>]…[<Nn>];
```

Về nguyên tắc ta có thể khai báo một mảng 255 chiều. Tuy nhiên do sự hạn chế của bộ nhớ nên chúng ta khai báo mảng 2 hoặc 3 chiều và độ lớn không được quá 65.536 Bytes (64KB).

Ví dụ một mảng 3 chiều như sau :

```
int ManHinh[25][80][15];
```

Kích thước của mảng này là 25 * 80 * 16 * 2 byte = 64.000 Bytes. Biến mảng này chiếm giữ trong vùng nhớ DATA SEGMENT từ lúc khởi động chương trình cho đến khi chấm dứt chương trình làm hạn chế việc khai báo các biến khác. Vì vậy chúng ta chỉ nên sử dụng biến kiểu mảng trong trường hợp số thành phần nhỏ và cố định và không nên khai báo nhiều hơn mục đích sử dụng.

Trong thực tế, để tránh việc khai báo lại nhiều lần các biến mảng có cùng kiểu người ta thường sử dụng cách khai báo biến không tường minh (thông qua khai báo kiểu) như sau:

Cú pháp khai báo biến mảng (không tường minh)

```
typedef <kiểu> <tên kiểu mảng>[<N1>][<N2>]…[<Nn>];
<tên kiểu mảng> <tên biến mảng>;
```

Ví dụ:

```
typedef int Mang20[20];
Mang20 a, b, c;
```

11.2.3 Khởi tạo giá trị cho mảng lúc khai báo

Ta có thể khởi tạo giá trị cho mảng một chiều thông qua các cách sau:

- Khởi tạo giá trị cho mọi phần tử của mảng.

```
int a[4] = {2912, 1706, 1506, 1904};
```

Mảng a gồm 4 phần tử 2912, 1706, 1506, 1904

- Khởi tạo giá trị cho một số phần tử đầu tiên của mảng. Các phần tử còn lại sẽ có giá trị 0.

```
int a[4] = {2912, 1706};
```

Mảng a gồm 4 phần tử 2912, 1706, 0, 0

- Khởi tạo giá trị 0 cho tất cả các phần tử của mảng, ta chỉ cần khởi tạo phần tử đầu bằng 0.

```
int a[100] = {0};
```

- Trình biên dịch tự xác định số phần tử của mảng thông qua danh sách khởi tạo nếu số phần tử không được chỉ rõ.

```
int a[] = {2912, 1706, 1506, 1904};
```

Mảng a sẽ gồm 4 phần tử 2912, 1706, 1506, 1904.

Ta có thể xem mảng 2 chiều là mảng một chiều mà mỗi phần tử là mảng 1 chiều khác. Ví dụ mảng hai chiều a[3][4] có thể được xem là mảng một chiều gồm 3 phần tử, mỗi phần tử là mảng một chiều gồm 4 phần tử. Lúc này ta sẽ khởi tạo từng mảng một chiều 4 phần tử theo các cách mô tả ở trên.

```
int a[3][4] = {{1, 2, 3, 4}, {1, 2}, {0}};
```

Mảng a sẽ được khởi tạo như sau:

1	2	3	4
1	2	0	0
0	0	0	0

11.3 Truy xuất dữ liệu kiểu mảng

Truy xuất đến từng phần tử của mảng thông qua cú pháp sau:

Cú pháp truy xuất dữ liệu kiểu mảng

```
<biến mảng>[<gt cs1>][<gt cs2>]…[<gt csn>]
```

Trong đó <gt cs1>, <gt cs1>, ..., <gt csn> : là các giá trị cụ thể của phần tử trong mảng muốn truy xuất.

Lưu ý, mảng a có n phần tử thì chỉ số của mảng sẽ từ 0 đến n–1. Tức mà các phần tử của mảng là a[0], a[1], …, a[n-1].

Ví dụ, cho hai mảng sau:

```
int MangSo[100];
float DiemHK[100][2];
```

Lúc này, phần tử thứ 3 của mảng có chỉ số là 2: MangSo[2], điểm thi môn thứ nhất của sinh viên thứ 2: DiemHK[1, 0]…

Không thể nhập xuất trực tiếp biến kiểu mảng mà phải thông qua từng thành phần của mảng đó. Ví dụ:

```
printf("%d", MangSo);          // Sai
printf("%d", MangSo[2]);       // Dung
printf("%f", DiemHK);          // Sai
printf("%f", DiemHK[1][0]);    // Dung
```

Không thể dùng lệnh gán thông thường để gán dữ liệu giữa hai kiểu mảng cùng loại. Thay vào đó, ta sẽ gán dữ liệu trực tiếp giữa từng phần tử tương ứng.

Gán trực tiếp từng phần tử

```
<biến mảng đích>[<chỉ số thứ i>] =
                    <biến mảng nguồn>[<chỉ số thứ i>];
```

Ví dụ:

```
typedef int MangSo[3];
MangSo a, b = {1, 2, 3};
a = b;                          // Sai
for (int i=0; i<3; i++)         // Đung
    a[i] = b[i];
```

11.4 Truyền mảng cho hàm

Tham số kiểu mảng trong khai báo hàm giống khai báo mảng.

```
void SapXepTang(int a[100]);
```

Tham số kiểu mảng được truyền cho hàm chính là địa chỉ của phần tử đầu tiên của mảng do đó số lượng phần tử trong tham số mảng có thể bỏ trống hoặc ta khai báo dạng con trỏ.

```
void SapXepTang(int a[]);       // Cach 1
void SapXepTang(int *a);        // Cach 2
```

Do ta chỉ truyền địa chỉ của phần tử đầu tiên của mảng nên số thành phần thực sự được sử dụng phải được truyền cho hàm thông qua một tham số khác.

```
// n la so phan tu thuc su duoc su dung
void SapXepTang(int a[], int n);
void SapXepTang(int *a, int n);
```

Do biến mảng chính là địa chỉ của phần tử đầu tiên của mảng đó nên khi gọi hàm, ta chỉ việc truyền tên biến mảng cho hàm. Lưu ý, nội dung của mảng có thể thay đổi khi kết thúc hàm.

```
void NhapMang(int a[], &n);
void SapXepTang(int a[], int n);
void XuatMang(int a[], n);

void main()
{
     int a[100];
     int n;

     NhapMang(a, n);        // Goi ham NhapMang
     SapXepTang(a, n);      // Goi ham SapXepTang
     XuatMang(a, n);        // Goi ham XuatMang
}
```

11.5 Một số bài toán trên mảng một chiều

11.5.1 Một số quy ước

Mảng được xét là mảng các số nguyên và số phần tử tối đa của mảng là MAX = 100.

```
#define MAX 100
```

Hai hàm dưới đây sẽ được sử dụng trong các bài toán phần sau.

Hàm kiểm tra số nguyên tố
Đầu vào *: số nguyên n cần kiểm tra*
Đầu ra *: 0 nếu sai hoặc 1 nếu đúng*

```
int LaSNT(int n)
{
     // Khoi tao dem de luu so uoc so cua n
     int dem = 0;
     // Dem so uoc so cua n
     for (int i = 1; i <= n; i++)
          if (n % i == 0)
               dem++;
     // Dua tren gia tri cua dem de ket luan
     if (dem == 2)
          return 1;
     return 0;
}
```

Thủ tục hoán vị giá trị hai số nguyên
Đầu vào *: số nguyên x và y cần hoán vị*
Đầu ra *: số nguyên x và t đã hoán vị (tham chiếu)*

```
void HoanVi(int &x, int &y)
{
     int tam = x;
     x  = y;
     y  = tam;
}
```

11.5.2 Nhập mảng

Yêu cầu

Viết thủ tục nhập mảng cho phép người sử dụng nhập số lượng phần tử n thực tế của mảng a và lần lượt nhập vào giá trị cho từng phần tử trong mảng này.

Ý tưởng

- Nhập số lượng phần tử thực tế n của mảng.
- Lần lượt nhập giá trị cho n phần tử của mảng từ chỉ số 0 đến chỉ số n – 1.

Cài đặt

Thủ tục nhập mảng
Đầu vào *: mảng a, số lượng phần tử n (tham chiếu)*
Đầu ra *: không có*

```
void NhapMang(int a[], int &n)
{
    // Nhap so luong phan tu n cua mang
    printf("Nhap so phan tu n: ");
    scanf("%d", &n);

    // Nhap gia tri cho n phan tu cua mang
    for (int i = 0; i < n; i++)
    {
        printf("Nhap phan phan tu thu %d: ", i);
        scanf("%d", &a[i]);
    }
}
```

Lưu ý

- Tham số n (số lượng phần tử) phải là tham chiếu (có dấu &) vì nội dung sẽ thay đổi sau khi thực hàm.
- Vì số lượng phần tử lớn nhất là MAX nên khi cần có thể kiểm tra xem n có vượt quá MAX hay không trước khi cho nhập mảng.

11.5.3 Xuất mảng

Yêu cầu

Viết thủ tục xuất mảng cho phép người sử dụng xuất nội dung mảng a cho trước với số lượng phần tử là n.

Ý tưởng

- Lần lượt xuất giá trị của n phần tử trong mảng từ chỉ số 0 đến chỉ số n – 1.

Cài đặt

Thủ tục xuất mảng
Đầu vào *: mảng a, số lượng phần tử n*
Đầu ra *: không có*

```
void XuatMang(int a[], int n)
{
    // Xuat gia tri cua N phan tu trong mang
    printf("Noi dung cua mang la: ");
    for (int i = 0; i < n; i++)
        printf("%d ", a[i]);
    printf("\n");
}
```

Lưu ý

- Tham số n không cần thiết phải là tham chiếu (không cần &) vì nội dung sẽ không thay đổi sau khi thực hiện hàm.

11.5.4 Kiểm tra một phần tử có thuộc mảng hay không

Yêu cầu

Cho trước mảng a, số lượng phần tử n. Hãy viết hàm kiểm tra xem phần tử x cho trước có thuộc mảng a hay không. Nếu có hãy trả lại vị trí của x trong mảng a. Ngược lại trả về 0.

Ý tưởng

Duyệt mảng để so sánh từng phần tử của mảng a với giá trị x. Nếu tìm thấy thì thoát ra và thông báo vị trí tìm được này. Ngược lại trả về –1.

Cài đặt

Hàm tìm kiếm một số cho trước có thuộc mảng hay không
Đầu vào *: mảng a, số lượng phần tử n, số nguyên x*
Đầu ra *: chỉ số của x trong a nếu x trong a, ngược lại –1.*

```
int TimKiem(int a[], int n, int x)
{
    // Bat dau tim tu vi tri dau tien
    int vt = -1;

    // Lan luot kiem tra cac vi tri
    while (vt < n && a[vt] != x)
        vt++;
    // Kiem tra xem vong lap thoat khi nao
    if (vt < n)
        return vt;
    return -1;
}
```

Lưu ý

Thay vì sử dụng vòng lặp while, ta cũng có thể sử dụng vòng lặp for để duyệt hết mảng và return ngay khi gặp phần tử thỏa yêu cầu (bài tập).

11.5.5 Kiểm tra tính chất của mảng

Yêu cầu

Viết hàm cho phép kiểm tra xem mảng a, số lượng phần tử n có phải là mảng toàn các số nguyên tố hay không?

Ý tưởng

Để giải quyết bài toán dạng trên, có 3 cách như sau:

- **Cách 1:** đếm số lượng các phần tử thỏa yêu cầu. Nếu số lượng này bằng đúng số lượng phần tử n của mảng thì mảng đã cho toàn các phần tử thỏa yêu cầu.
- **Cách 2:** giống cách 1 nhưng ta sẽ kiểm tra xem số phần tử không thỏa yêu cầu có bằng 0 hay không. Nếu bằng 0 thì mảng toàn những phần tử thỏa yêu cầu.
- **Cách 3:** chỉ cần phát hiện 1 phần tử không thỏa yêu cầu ta kết luận ngay mảng đã cho không toàn các phần tử thỏa yêu cầu.

Cài đặt

Hàm kiểm tra mảng có toàn nguyên tố hay không (cách 1)
Đầu vào *: mảng a, số lượng phần tử n*
Đầu ra *: 1 (toàn số nguyên tố) hoặc 0 (ngược lại).*

```
int KiemTra_C1(int a[], int n)
{
    // Khoi tao bien dem
    int dem = 0;

    // Dem so phan tu thoa man yeu cau
    for (int i = 0; i < n; i++)
        if (LaSNT(a[i]))     // a[i] la SNT?
            dem++;

    // Ket luan
    if (dem == n)
        return 1;
    return 0;
}
```

Hàm kiểm tra mảng có toàn nguyên tố hay không (cách 3)
Đầu vào *: mảng a, số lượng phần tử n*
Đầu ra *: 1 (toàn số nguyên tố) hoặc 0 (ngược lại).*

```
int KiemTra_C3(int a[], int n)
{
     // Duyet tung phan tu cua mang
     // neu phat hien phan tu khong thoa
     // thi return 0 ngay
     for (int i = 0; i < n; i++)
          if (!LaSNT(a[i]))
               return 0;

     // Duyet xong cac phan tu cua mang
     // nhung khong tim thay phan tu nao vi pham
     return 1;
}
```

11.5.6 Tính toán trên mảng

Yêu cầu

Viết hàm cho phép tính tổng các số nguyên tố có trong mảng a, số lượng phần tử là n cho trước.

Ý tưởng

Duyệt từng phần tử của mảng a. Nếu phần tử nào thỏa yêu cầu (là số nguyên tố) thì ta cộng dồn vào tổng trước đó.

Cài đặt

Hàm tính tổng các số nguyên tổ có trong mảng
Đầu vào *: mảng a, số lượng phần tử n*
Đầu ra *: tổng các số nguyên tố có trong mảng.*

```
int TongSNT(int a[], int n)
{
     // Khai bao va khoi tao bien tong
     int tong = 0;
     // Tong so phan tu thoa man yeu cau
     for (int i = 0; i < n; i++)
          if (LaSNT(a[i]))     // a[i] la SNT?
               tong = tong + a[i];
     // Ket luan
     return tong;
}
```

11.5.7 Tách mảng

Yêu cầu

Cho trước mảng a, số lượng phần tử na. Viết thủ tục tách các các số nguyên tố có trong mảng a và đưa sang mảng b, số lượng phần tử nb.

Ý tưởng

- Số lượng phần tử nb trong mảng b ban đầu là 0 (rỗng).
- Duyệt từng phần tử của mảng a và sao chép các phần tử là số nguyên tố trong mảng a vào mảng b đồng thời tăng số lượng phần tử nb của mảng n lên 1 đơn vị.

Cài đặt

Hàm tách các số nguyên tố từ mảng này vào mảng khác
Đầu vào : *mảng a, số lượng phần tử n*
mảng b, số lượng phần tử nb (tham chiếu)
Đầu ra : *không có*

```
void TachSNT( MyArray a, int na,
              MyArray b, int &nb)
{
    // Khoi tao so luong nb cua mang b
    nb = 0;

    // Duyet tung phan tu cua mang a
    for (int i = 0; i < na; i++)
         if (LaSNT(a[i]))     // a[i] la SNT?
         {
              b[nb] = a[i];   // Them a[i] vao b
              nb++;
         }
}
```

Lưu ý

- Tham số nb (số lượng phần tử của mảng b) là tham chiếu do chúng có thể thay đổi khi kết thúc hàm.
- Phải khởi tạo số lượng phần tử nb bằng 0 để cho biết mảng b chưa có phần tử nào.

11.5.8 Gộp mảng

Yêu cầu

Cho trước mảng a và b số lượng phần tử na và nb.

Viết thủ tục gộp 2 mảng a và b theo thứ tự đó thành mảng c, số lượng phần tử nc = na + nb.

Ý tưởng

- Số lượng phần tử nc của mảng c ban đầu bằng 0.
- Chuyển các phần tử từ mảng a sang mảng c đồng thời tăng số lượng phần tử của mảng c lên 1.
- Tiếp tục chuyển các phần tử từ mảng b sang mảng c đồng thời tăng số lượng phần tử của mảng c lên 1.

Cài đặt

Thủ tục gộp hai mảng thành một mảng
Đầu vào *: mảng a và b số lượng phần tử na và nb*
mảng c, số lượng phần tử nc (tham chiếu)
Đầu ra *: không có*

```
void GopMang( int a[], int na, int b[], int nb,
              int c[], int &nc)
{
     nc = 0;   // Khoi tao so luong nc cua mang c

     // Dua tung phan tu cua mang a vao mang C
     for (int i = 0; i < na; i++)
     {
          c[nc] = a[i];
          nc++;
     }

     // Dua tung phan tu cua mang B vao mang C
     for (int i = 0; i < nb; i++)
     {
          c[nc] = b[i];
          nc++;
     }
}
```

11.5.9 Tìm số nhỏ nhất/lớn nhất

Yêu cầu

Viết hàm trả về số nhỏ nhất trong mảng a, số lượng phần tử n cho trước.

Ý tưởng

- Giả sử số nhỏ nhất hiện tại là giá trị phần tử đầu tiên.
- Duyệt các phần tử tiếp theo của mảng. Nếu có phần tử nào khác nhỏ hơn số nhỏ nhất hiện tại thì cập nhật giá trị số nhỏ nhất hiện tại này lại (bằng giá trị của phần tử nhỏ hơn vừa tìm được)

Cài đặt

Hàm tìm số nhỏ nhất trong mảng
Đầu vào *: mảng a số lượng phần tử n*
Đầu ra *: số nhỏ nhất trong mảng*

```
int TimMin(int a[], int n)
{
    // Khoi tao bien min hien tai
    int min = a[0];

    // Kiem tra cac phan tu con lai
    for (int i = 1; i < n; i++)
        if (a[i] < min)
            min = a[i];

    // Ket luan
    return min;
}
```

Lưu ý

- Giá trị khởi tạo của min (phần tử nhỏ nhất hiện tại) là giá trị a[0] chứ không phải 0 (chỉ số 0).
- So sánh các phần tử a[i] còn lại với min chứ không phải với a[0].
- Nên chạy vòng lặp từ 1 vì phần tử đầu đã được kiểm tra.

11.5.10 Tìm số nhỏ nhất/lớn nhất thỏa yêu cầu cho trước

Yêu cầu

Viết hàm trả về số nguyên tố nhỏ nhất trong mảng a, số lượng phần tử n cho trước. Nếu không có thì trả về 0.

Ý tưởng

Để giải quyết bài toán dạng trên, có 3 cách như sau:

- **Cách 1 :** Tách các phần tử là số nguyên tố có trong mảng a sang mảng b (nếu có) rồi áp dụng hàm TimMin trên mảng b để tìm số nhỏ nhất có trong mảng b này. Số tìm được chính là số nguyên tố nhỏ nhất.
- **Cách 2 :** Cách này giống ý tưởng tìm số nhỏ nhất bình thường. Số nguyên tố nhỏ nhất hiện tại chính là số nguyên tố đầu tiên tìm được (nếu có). Tìm các số nguyên tố khác trong mảng a và so sánh với số nguyên tố nhỏ nhất hiện tại này.
- **Cách 3 :** Cách này giống cách viết chương trình tìm số nhỏ nhất bình thường. Ta làm như sau :
    - Giả sử số min hiện tại là phần tử đầu tiên.
    - Duyệt các phần tử tiếp theo của mảng. Nếu phần tử nào là số nguyên tố, lúc đó sẽ xảy ra 2 trường hợp :
        - min hiện tại không phải là số nguyên tố thì ta cập nhật min hiện tại chính là số nguyên tố đang xét.
        - min hiện tại là số nguyên tố nhưng số nguyên tố đang xét nhỏ hơn min hiện tại thì ta cập nhật min hiện tại chính bằng số nguyên tố đang xét.
    - Kiểm tra xem min hiện tại có thật sự là số nguyên tố hay không. Nếu nó là số nguyên tố thì trả về

chính nó. Nếu không (nghĩa là mảng không có số nguyên tố nào) thì trả về 0 theo yêu cầu.

Cài đặt

Hàm tìm số nguyên tố nhỏ nhất trong mảng (Cách 1)
Đầu vào : *mảng a số lượng phần tử n*
Đầu ra : *số nguyên tố nhỏ nhất trong mảng (0 nếu không có)*

```
int TimMinSNT_C1(int a[], int n)
{
    int b[MAX];
    int nb;

    // Tach cac so nguyen to trong a dua vao B
    TachSNT(a, n, b, nb);

    // Kiem tra mang B
    if (nb != 0)
        return TimMin(b, nb); // Tim min

    return 0;
}
```

Hàm tìm số nguyên tố nhỏ nhất trong mảng (Cách 2)
Đầu vào : *mảng a số lượng phần tử n*
Đầu ra : *số nguyên tố nhỏ nhất trong mảng (0 nếu không có)*

```
int TimMinSNT_C2(int a[], int n)
{
    int vt = 0;       // Tim tu vi tri dau tien

    // Trong khi chua kiem tra het mang
    // va vi tri hien tai khong phai la snt
    // thi duyet tiep phan tu tiep theo
    while (vt < n && !LaSNT(a[vt]))
        vt++;
```

```
    if (vt < n)              // Tim thay so nt
    {
         int minSNT = a[vt]; // min = so tim duoc
         // Duyet cac phan tu con lai
         for (int i = vt + 1; i < n; i++)
              if (a[i] < minSNT && LaSNT(a[i]))
                   minSNT = a[i];

         return minSNT;      // Ket luan
    }
    return 0;     // Khong tim thay so nt nao
}
```

Hàm tìm số nguyên tố nhỏ nhất trong mảng (Cách 3)
Đầu vào : *mảng a số lượng phần tử n*
Đầu ra : *số nguyên tố nhỏ nhất trong mảng (0 nếu không có)*

```
int TimMinSNT_C3(int a[], int n)
{
    // Khoi tao bien min hien tai
    int minSNT = a[0];

    // Duyet cac phan tu con lai
    for (int i = 1; i < n; i++)
         if (LaSNT(a[i]))
              if  (!LaSNT(minSNT) ||
              (LaSNT(minSNT) && a[i] < minSNT))
                   minSNT = a[i];

    // Kiem tra lai minSNT va ket luan
    if (LaSNT(minSNT))
         return minSNT;

    return 0;
}
```

Lưu ý

- Cách 1 và cách 2 dễ hiểu nhất vì ý tưởng khá đơn giản nhưng phải qua rất nhiều bước thực hiện. Cách 3 hơi khó hiểu nhưng bù lại cài đặt rất nhanh và mang tính tổng quát cao.
- Nên biết ý tưởng các cách làm nhưng chỉ chọn một cách làm cảm thấy dễ hiểu nhất để cài đặt và luôn sử dụng cách này.

11.5.11 Sắp xếp mảng tăng/giảm

Yêu cầu

Cho trước một mảng a, số lượng phần tử n. Hãy viết thủ tục sắp xếp mảng tăng dần theo trị số.

Ý tưởng

Duyệt tất cả các các cặp phần tử của mảng a để triệt tiêu các nghịch thế. Nghịch thế là các cặp không đúng thứ tự yêu cầu.

Cài đặt

Thủ tục sắp xếp mảng tăng dần
Đầu vào *: mảng a, số lượng phần tử n*
Đầu ra *: không có*

```
void SapXepTang(int a[], int n)
{
     int i, j;
     // Duyet tat ca cap phan tu trong A
     for (i = 0; i < n- 1; i++)
          for (j = i + 1; j < n; j++)
               // Neu (a[i], a[j]) nghich the
               if (a[i] > a[j]) Then
                    HoanVi(a[i], a[j]);
}
```

Lưu ý

- Chú ý các chỉ số chạy của hai biến i và j. Cách trên tối ưu nhất do một phần tử sẽ bắt cặp với các phần tử sau nó.

- Còn rất nhiều thuật toán sắp xếp khác nhanh hơn nhưng đây là thuật toán đơn giản và dễ hiểu nhất.

11.5.12 Sắp xếp mảng tăng/giảm thỏa yêu cầu cho trước

Yêu cầu

Cho trước một mảng a, số lượng phần tử n. Hãy sắp xếp mảng mảng a sao cho các số nguyên tố tăng dần đưa về đầu mảng và các số còn lại giảm dần đưa về cuối mảng.

Ý tưởng

- **Cách 1 :**
  - Chuyển các số nguyên tố từ mảng a sang mảng b và sắp xếp tăng dần mảng b này.
  - Chuyển các số còn lại (không phải số nguyên tố) từ mảng a sang mảng c và sắp xếp tăng dần mảng c này.
  - Gộp mảng b và mảng c thành mảng a.
- **Cách 2 :** thực hiện tương tự như cách sắp xếp đơn giản ở phần trên, tức là ta sẽ duyệt hết tất cả các cặp phần tử của mảng a và triệt tiêu các cặp nghịch thế. Các cặp (x, y) với x đứng trước y sẽ thuộc 4 trường hợp sau :
  - x, y đều là số nguyên tố nhưng x > y : đây là cặp nghịch thế do các số nguyên tố phải tăng dần.
  - x, y đều không là số nguyên tố nhưng x < y : đây là cặp nghịch thế do các số không phải số nguyên tố phải giảm dần.
  - x là không là số nguyên tố và y là số nguyên tố : đây là cặp nghịch thế do các số nguyên tố phải đứng trước các số không là nguyên tố.
  - x là số nguyên tố và y không là số nguyên tố : đây không phải là cặp nghịch thế do số nguyên tố đứng trước số không phải nguyên tố.

Cài đặt

Thủ tục sắp xếp mảng : số nguyên tố đứng trước tăng dần
(Cách 1) số còn lại đứng sau giảm dần
***Đầu vào** : mảng a, số lượng phần tử n*
***Đầu ra** : không có*

```
void SapXepSNT(int a[], int n)
{
    int b[MAX], c[MAX];
    int nb, nc;
    // Chuyen cac so nguyen to trong a vao b
    TachSNT(a, n, b, nb);

    // Chuyen cac so khong la snt trong a vào c
    TachSKNT(a, n, c, nc);

    // Sap xep mang b tang dan
    SapXepTang(b, nb);

    // Sap xep mang c giam dan
    SapXepGiam(c, nc);

    // Gop mang b va c thanh mang a
    GopMang(b, nb, c, nc, a, n);
}
```

Lưu ý

- Thủ tục SapXepTang, SapXepGiam đã trình bày ở trên.
- Thủ tục TachSNT và TachSKNT đã trình bày ở phần trên để tách các số nguyên tố và các số không phải nguyên tố từ một mảng ra mảng khác. Ngoài ra ta có thể viết một hàm khác để cùng lúc chia mảng a ra 2 mảng b (chứa các số nguyên tố) và c (chứa các số không phải nguyên tố). (bài tập)

Thủ tục sắp xếp mảng : số nguyên tố đứng trước tăng dần
(Cách 2) số còn lại đứng sau giảm dần
***Đầu vào** : mảng a, số lượng phần tử n*
***Đầu ra** : không có*

```
void SapXepSNT(int a[], int n)
{
     int i, j;
     // Duyet tat ca cac cap phan tu trong a
     for (i = 0; i < n - 1; i++)
          for (j = i + 1; j < n; j++)
               // Neu (a[i], a[j]) nghich the
               if (NghichThe(a[i], a[j]))
                    HoanVi(a[i], a[j]);
}
```

Hàm **NghichThe** sử dụng ở trên được viết như sau:

Hàm kiểm tra một cặp số có phải nghịch thế hay không
3 trường hợp nghịch thế :
x nguyên tố, y nguyên tố và x > y
x không nguyên tố, y không nguyên tố và x < y
x không nguyên tố, y nguyên tố
***Đầu vào** : cặp số nguyên x, y*
***Đầu ra** : 1 (nghịch thế), 0 (không nghịch thế)*

```
int NghichThe(int x, int y)
{
     // De tranh viec kiem tra nhieu lan
     int xLaSNT = LaSNT(x);
     int yLaSNT = LaSNT(y);

     // 3 truong hop la nghich the
     if  ((xLaSNT && yLaSNT && x > y) ||
         (!xLaSNT && !yLaSNT && x < y) ||
         (!xLaSNT && yLaSNT))
          return 1;
     return 0;
}
```

Lưu ý

- Có thể viết trực tiếp các trường hợp nghịch thế vào trong thủ tục sắp xếp thay vì gọi hàm NghichThe nhưng như vậy sẽ làm rối chương trình.
- Nên kiểm tra trước x và y có phải là số nguyên tố không rồi gán vào biến tạm để tránh việc tính đi tính lại nhiều lần.

11.5.13 Sửa/Thêm/Xóa mảng

Yêu cầu

Cho trước một mảng a, số lượng phần tử n. Hãy viết các hàm thực hiện các yêu cầu sau:

- Sửa một phần tử tại vị trí vt trong mảng a thành giá trị x.
- Thêm một phần tử x vào mảng a tại vị trí vt.
- Xóa một phần tử tại vị trí vt trong mảng a.

Ý tưởng

- **Sửa:**
  - Kiểm tra vt có hợp lệ hay không ($0 \le vt < n$).
  - Cập nhật giá trị của phần tử tại vt thành giá trị x.
- **Thêm:**
  - Kiểm tra vt có hợp lệ hay không ($0 \le vt \le n$).
  - "Đẩy giá trị" các phần tử từ vị trí vt sang phải 1 phần tử rồi "chèn" phần tử x vào vị trí vt.
  - Tăng số lượng phần tử n của mảng a lên 1.
- **Xóa:**
  - Kiểm tra vt có hợp lệ hay không ($0 \le vt < n$).
  - "Kéo giá trị" các phần tử bên phải vị trí vt sang trái 1 phần tử.
  - Giảm số lượng phần tử n của mảng a xuống 1.

Cài đặt

Thủ tục sửa một số trong mảng bằng một giá trị cho trước
Đầu vào *: mảng a, số lượng phần tử n, vị trí vt, số x*
Đầu ra *: không có*

```
void Sua(int a[], int n, int vt, int x)
{
     // Kiem tra tinh hop le cua vt
     if (vt >= 0 && vt < n)
          a[vt] = x;
}
```

Thủ tục thêm một phần tử vào mảng tại một ví trí nào đó
Đầu vào *: mảng a, số lượng phần tử n, vị trí vt, số x*
Đầu ra *: không có*

```
void Them(int a[], int &n, int vt, int x)
{
     // Kiem tra tinh hop le cua vt
     if (vt >= 0 && vt <= n)
     {
          // Day tu vi tri vt sang phai
          for (int i = n; i > vt; i--)
               a[i] = a[i - 1];

          // Chen x vao vi tri vt
          a[vt] = x;

          // Tang so luong phan tu n len 1
          n++;
     }
}
```

Thủ tục xóa một phần tử trong mảng tại một ví trí nào đó
Đầu vào *: mảng a, số lượng phần tử n, vị trí vt*
Đầu ra *: không có*

```
void Xoa(MyArray a, int &n, int vt)
{
     // Kiem tra tinh hop le cua vt
     if (vt >=0 && vt < n)
     {
          // Keo tu vi tri vt + 1 sang trai
          for (int i = vt; i < n – 1; i++)
               a[i] = a[i + 1];

          // Giam so luong phan tu n xuong 1
          n--;
     }
}
```

Lưu ý

- Nếu cẩn thận có thể kiểm tra số lượng phần tử hiện tại có tối đa chưa, nếu tối đa rồi thì không thể thêm vào nữa.
- Cẩn thận với các giá trị của biến chạy vòng lặp for.

Bài tập cuối chương

Lý thuyết

1. Mảng là gì? Trường hợp nào thì sử dụng mảng?
2. Tình bày các cách khai báo kiểu, biến, hằng mảng và cho ví dụ minh họa.
3. Trình bày các thao tác truy xuất trên mảng. Cho ví dụ minh họa từng trường hợp.

Thực hành

Thực hiện các thao tác sau trên mảng một chiều a gồm các số nguyên, số lượng phần tử phần tử là n.

4. Các thao tác nhập xuất
    a. Nhập mảng.
    b. Xuất mảng.
5. Các thao tác kiểm tra
    a. Mảng có phải là mảng toàn số chẵn?
    b. Mảng có phải là mảng tăng dần?
6. Các thao tác tính toán
    a. Có bao nhiêu số chia hết cho 4 nhưng không chia hết cho 5.
    b. Tính tổng các số nguyên tố có trong mảng.
7. Các thao tác tìm kiếm
    a. Tìm vị trí số nguyên tố đầu tiên trong mảng nếu có.
    b. Tìm số nhỏ nhất trong mảng.
    c. Tìm số dương nhỏ nhất trong mảng.
8. Các thao tác xử lý
    a. Tách mảng trên thành hai mảng b và c, mảng b chứa các số nguyên dương, mảng c chứa các số còn lại.
    b. Sắp xếp mảng giảm dần.
    c. Sắp xếp mảng sao cho các số dương đứng đầu mảng giảm dần, kế đến là các số âm tăng dần, cuối cùng là các số 0.
    d. Sửa các số nguyên tố có trong mảng thành số 0.
    e. Chèn (thêm) số 0 đằng sau các số nguyên tố trong mảng.
    f. Xóa tất cả số nguyên tố có trong mảng.

Chương 12. MỘT SỐ KIỂU DỮ LIỆU NÂNG CAO

Chúng ta đã tìm hiểu các kiểu dữ liệu cơ sở của C bao gồm kiểu số nguyên, kiểu số thực, kiểu luận lý, kiểu ký tự (ở chương 7) và kiểu dữ liệu mảng (ở chương 11). Chương này sẽ trình bày các kiểu dữ liệu nâng cao khác bao gồm kiểu chuỗi ký tự, kiểu cấu trúc (lưu trữ và xử lý dữ liệu phức tạp), kiểu tập tin (lưu trữ và xử lý thông tin trên đĩa) và kiểu con trỏ (quản lý bộ nhớ).

12.1 Kiểu chuỗi ký tự (string)

12.1.1 Khái niệm

Như ta đã biết, dữ liệu kiểu char chỉ chứa được một ký tự, để có thể lưu trữ một chuỗi (nhiều ký tự) ta sử dụng mảng ký tự (Chương 11 – Dữ liệu kiểu mảng).

12.1.2 Khai báo

Kiểu dữ liệu chuỗi được khai báo theo cú pháp khai báo mảng một chiều với mỗi phần tử có kiểu char (ký tự).

Ví dụ:

```
char s[10];
```

Biến s được khai báo như trên có thể chứa được một chuỗi có độ dài tối đa là 9 ký tự (không phải là 10) do trong C chuỗi được được xem là một loạt các ký tự kết thúc bởi ký tự rỗng (NULL), ký hiệu là \0.

Giống như cách khởi tạo mảng thông thường, để khởi tạo cho chuỗi s giá trị "THCS A" ta thực hiện như sau:

```
char s[10] = {'T', 'H', 'C', 'S', ' ', 'A', '\0'};
```

Tuy nhiên, ta có thể khởi tạo cho chuỗi ngắn gọn hơn bằng cách sử dụng chuỗi thường (chuỗi được bao bởi cặp dấu " "). Lúc này trình biên dịch tự động thêm vào ký tự kết thúc chuỗi ('\0').

```
char s[10] = "THCS A";
char s[] = "THCS A";        // Tu xac dinh do dai
```

12.1.3 Nhập xuất chuỗi

Để xuất chuỗi, ta có thể sử dụng hàm printf với đặc tả "%s" hoặc hàm puts như sau:

Hàm xuất dữ liệu ra màn hình (#include <stdio.h>)
`int puts(const char *s);`
Hàm puts xuất chuỗi s ra màn hình và tự động xuống dòng. Nếu thành công sẽ trả về giá trị không âm, ngược lại sẽ trả về EOF.

Ví dụ:

```
char ten[] = "THCS A";
printf("%s", ten);          // Khong xuong dong
puts(ten);                  // Tu dong xuong dong
```

Để nhập chuỗi, ta có thể sử dụng hàm scanf( ) với đặc tả "%s" nhưng hàm này chỉ đọc các ký tự từ bàn phím đến khi gặp ký tự khoảng trắng.

Ví dụ:

```
char ten[100;
scanf("%s", &ten);
puts(ten);
```

Nếu nhập vào "THCS A" thì chỉ nhận được chuỗi "THCS"

Để nhập được một chuỗi đầy đủ, ta sử dụng hàm sau:

Hàm nhập dữ liệu từ bàn phím (#include <stdio.h>)
`char *gets(char *s);`
Hàm gets đọc tất cả các ký tự nhập từ bàn phím đến khi gặp ký tự sang dòng mới (khi ta nhấn enter). Hàm gets sẽ loại bỏ ký tự enter và thêm vào chuỗi ký tự kết thúc chuỗi '\0'. Hàm gets trả về địa chỉ của chuỗi nhận được.

Ví dụ:

```
char ten[100];
gets(ten);
puts(ten);
```

12.1.4 Các thao tác trên dữ liệu kiểu chuỗi

Chuỗi chính là mảng ký tự, do đó không thể gán dữ liệu giữa hai kiểu chuỗi. Thay vào đó ta sử dụng lệnh strcpy được định nghĩa trong string.h.

Hàm sao chép chuỗi (#include <string.h>)
`char *strcpy(char *dest, const char *src);`
Sao chép chuỗi src sang chuỗi dest, kết thúc khi gặp ký tự kết thúc chuỗi. Hàm trả về địa chỉ chuỗi dest.

```
char s[100];
s = "Tin hoc co so A";              // Sai
strcpy(s, "Tin hoc co so A");       // Dung
```

Trong string.h còn cung cấp một số hàm khác thao tác trên dữ liệu kiểu chuỗi ký tự. Một vài hàm quan trọng như sau:

Hàm chuyển chuỗi thành chuỗi thường (#include <string.h>)
`char *strlwr(char *s);`
Hàm strlwr chuyển chuỗi s thành chuỗi thường (các ký tự 'A', 'B', …, 'Z' thành 'a', 'b', … 'z') và trả về địa chỉ của chuỗi s.

Hàm chuyển chuỗi thành hoa (#include <string.h>)

```
char *strupr(char *s);
```

Hàm strupr chuyển chuỗi s thành chuỗi hoa (các ký tự 'a', 'b', …, 'z' thành 'A', 'B', … 'Z') và trả về địa chỉ của chuỗi s.

Hàm đảo ngược chuỗi (#include <string.h>)

```
char *strrev(char *s);
```

Hàm đảo ngược thứ tự các ký tự trong chuỗi (trừ ký tự kết thúc chuỗi) và trả về địa chỉ của chuỗi kết quả.

Hàm so sánh hai chuỗi (#include <string.h>)

```
int *strcmp(const char *s1, const char *s2);
```

Hàm trả về < 0 nếu s1 < s2; = 0 nếu s1 = s2, > 0 nếu s1 > s2.

Hàm nối hai chuỗi (#include <string.h>)

```
char *strcat(char *dest, const char *src);
```

Hàm nối chuỗi src và chuỗi dest và trả về địa chỉ của chuỗi đã được nối.

Hàm tính độ dài chuỗi (#include <string.h>)

```
size_t *strlen(const char *s);
```

Hàm trả về độ dài chuỗi.

Hàm tìm chuỗi này trong chuỗi kia (#include <string.h>)

```
char *strstr(const char *s1, const char *s2);
```

Hàm tìm vị trí xuất hiện đầu tiên của chuỗi s2 trong chuỗi s1. Hàm trả về địa chỉ của thành phần đầu tiên trong chuỗi s1 tìm được. Nếu không tìm được thì trả về null.

12.2 Kiểu cấu trúc (struct)

12.2.1 Khái niệm

Cấu trúc (struct) là một dạng dữ liệu gồm nhiều thành phần khác kiểu. Mỗi thành phần được truy xuất bằng một định danh gọi là trường (field).

12.2.2 Khai báo

Kiểu dữ liệu cấu trúc được khai báo theo cú pháp như sau:

Cú pháp khai báo kiểu cấu trúc

```
struct <tên kiểu cấu trúc>
{
     <kiểu 1> <tên trường 1>;
     <kiểu 2> <tên trường 2>;
     …
     <kiểu n> <tên trường n>;
};
```

Ví dụ:

```
struct SinhVien
{
     char hoten[30];
     float toan, ly, hoa;
};
```

Khai báo biến có thể dựa trên khai báo kiểu (không tường minh) như sau:

Cú pháp khai báo biến cấu trúc (không tường minh)

```
struct <tên kiểu cấu trúc>
{
     <kiểu 1> <tên trường 1>;
     <kiểu 2> <tên trường 2>;
     …
     <kiểu n> <tên trường n>;
};

struct <tên kiểu cấu trúc> <tên biến cấu trúc>;
```

Hoặc khai báo biến trực tiếp (tường minh) như sau:

Cú pháp khai báo biến cấu trúc (tường minh)

```
struct <tên kiểu cấu trúc>
{
     <kiểu 1> <tên trường 1>;
     <kiểu 2> <tên trường 2>;
     …
     <kiểu n> <tên trường n>;
} <tên biến cấu trúc>;
```

Ví dụ:

```
struct SinhVien
{
    char hoten[30];
    float toan, ly, hoa;
} sv1, sv2, sv3;          // Tuong minh

struct SinhVien sv4, sv5;  // Khong tuong minh
```

Cách khai báo dạng tường minh cho ta dễ nhận thấy mối quan hệ giữa biến và trường. Tuy nhiên trong cấu trúc chương trình của C thường đòi hỏi khai báo tham số bằng định danh kiểu nên việc khai báo tường minh như trên ít được sử dụng.

Để khởi tạo giá trị cho biến cấu trúc, ta sử dụng cách sau:

Cú pháp khai báo biến bản ghi (tường minh)

```
struct <tên kiểu cấu trúc>
{
     <kiểu 1> <tên trường 1>;
     <kiểu 2> <tên trường 2>;
     …
     <kiểu n> <tên trường n>;
}<tên biến cấu trúc> = {<giá trị 1>,…,<giá trị n>};
```

Ví dụ:

```
struct SinhVien
{
    char hoten[30];
    float toan, ly, hoa;
} sv1 = {"Nguyen Van A", 10, 7.5, 9}, sv2, td3;
```

12.2.3 Truy xuất dữ liệu kiểu bản ghi

Chúng ta không thể nhập xuất trực tiếp dữ liệu kiểu cấu trúc mà chỉ có thể nhập xuất thông qua các trường của cấu trúc đó.

Để truy xuất đến trường của cấu trúc ta sử dụng toán tử chấm.

Cú pháp truy xuất đến các trường của cấu trúc

```
<tên biến cấu trúc>.<tên trường>
```

Ví dụ

```
struct SinhVien sv;

strcpy(sv.hoten, "Nguyen Van A");
sv.toan = 10;
sv.ly = 7.5;
sv.hoa = 9;

printf("Ho ten sinh vien: %s\n", sv.hoten);
printf("DTB: %.2f", (sv.toan + sv.ly + sv.hoa)/3);
```

12.2.4 Gán dữ liệu kiểu bản ghi

Đối với 2 dữ liệu bản ghi có cùng kiểu, chúng ta có thể thực hiện phép gán theo các cách sau đây:

Thông qua các trường (tường minh)

```
<biến cấu trúc đích>.<trường 1> :=
                    <biến cấu trúc nguồn>.<trường 1>;
<biến cấu trúc đích>.<trường 2> :=
                    <biến cấu trúc nguồn>.<trường 2>;
…
```

Sử dụng lệnh gán

```
<biến cấu trúc đích> = <biến cấu trúc nguồn>;
```

Ví dụ:

```
struct SinhVien
{
    char hoten[30];
    float toan, ly, hoa;
} sv1 = {"Nguyen Van A", 10, 7.5, 9}, sv2;

sv2 = sv1;                          // Cach 1

strcpy(sv2.hoten, sv1.hoten);    // Cach 2
sv2.toan = sv1.toan;
sv2.ly = sv1.ly;
sv2.hoa = sv1.hoa;
```

12.3 Kiểu tập tin (file) và kiểu con trỏ (pointer)

Các kiểu dữ liệu ta đã khảo sát đều hiện diện trong bộ nhớ RAM khi khởi động chương trình, nhưng khi chấm dứt chương trình các dữ liệu trên bị xóa mất, vì vậy việc lưu trữ dữ liệu lâu dài hoặc sử dụng lại nhiều lần không thể thực hiện được. C đã tạo một kiểu cho phép ta lưu trữ dữ liệu lâu dài trên đĩa mềm hoặc đĩa cứng gọi là dữ liệu kiểu file. Có hai loại tập tin là tập tin văn bản (text file) và tập tin nhị phân (binary file).

Kiểu con trỏ (pointer) là một loại dữ liệu có kích thước 2 byte, không dùng để chứa dữ liệu mà là chứa địa chỉ Segment và Offset của một biến khác hay nói cách khác biến con trỏ đang trỏ đến biến mà nó đang chứa địa chỉ. Biến con trỏ được sử dụng trong cách bài toán quy hoạch động…

Trong khuôn khổ chương trình Tin học cơ sở A sẽ không bàn đến hai kiểu này. Sinh viên có thể tham khảo các tài liệu chương trình Tin học cơ sở A2 để tìm hiểu thêm.

Bài tập cuối chương

Lý thuyết

1. Trình bày khái niệm kiểu chuỗi? Mô tả cách khai báo và sử dụng nó.
2. Trình bày khái niệm kiểu cấu trúc? Mô tả cách khai báo kiểu, biến, hằng bản ghi. Cho ví dụ minh họa.

Thực hành

3. Tập sử dụng các hàm trong thư viện string.h
4. Để biết một thí sinh đậu hay rớt trong một kỳ tuyển sinh, ta cần lưu các thông tin như Họ tên thí sinh, khu vực (1, 2, 3), nhóm (1, 2, 3), tổng điểm 3 môn.
    a. Nhập thông tin n thí sinh.
    b. Lập danh sách các thí sinh đậu với điểm chuẩn như bảng sau:

	Khu vực 1	*Khu vực 2*	*Khu vực 3*
Nhóm 1	18	17	16
Nhóm 2	17.5	16.5	15.5
Nhóm 3	17	16	15

    c. In danh sách đã sắp xếp theo điểm giảm dần.