Kỹ thuật lập trình(1):
Cơ bản về ngôn ngữ lập trình

Bộ môn Hệ thống thông tin
Khoa Công nghệ thông tin

Nội dung

- Giới thiệu chung
- Lệnh nhập/xuất
- Lệnh điều kiện
- Lệnh vòng lặp
- Hàm
- Kiểu mảng
- Xâu kí tự
- Kiểu cấu trúc (struct) và kiểu hợp (union)
- Làm việc với tệp

Giới thiệu chung

- Ngôn ngữ C ra đời năm 1972
- Phát triển thành C++ vào năm 1983
- Ngôn ngữ được sử dụng rất phổ biến
- Có nhiều trình biên dịch C khác nhau
  - Turbo C, Borland C
  - GCC
- Thực hành trên Turbo C
  - Cung cấp môi trường tích hợp cho phép soạn thảo và biên dịch

Giới thiệu chung

- Một số phím soạn thảo

Phím	Chức năng
←↑↓→	Di chuyển con trỏ sang trái, lên, xuống, sang phải
Home	Đưa con trỏ về đầu dòng
End	Đưa con trỏ về cuối dòng
PgUp	Đưa con trỏ về đầu một trang màn hình
PgDw	Đưa con trỏ về cuối một trang màn hình
Ctrl + →	Dịch con trỏ sang phải một chữ
Ctrl + ←	Dịch con trỏ sang trái một chữ

Giới thiệu chung

- Một số phím soạn thảo

Phím	Chức năng
Enter	Xuống dòng
Insert	Chuyển đổi chế độ chèn/đè
Delete	Xóa kí tự ngay sau vị trí con trỏ
Back space	Xóa kí tự ngay trước vị trí con trỏ
Ctrl + Y	Xóa dòng kí tự chứa con trỏ
Ctrl + Q + Y	Xóa các kí tự từ vị trí con trỏ đến cuối dòng

Giới thiệu chung

- Một số phím soạn thảo

Phím	Chức năng
Ctrl + K + C	Chép khối tới vị trí mới của con trỏ
Ctrl + K + V	Chuyển khối tới vị trí mới của con trỏ
Ctrl + K + Y	Xóa cả khối
Ctrl + K + W	Ghi một khối vào một tệp trên đĩa
Ctrl + K + R	Đọc một khối từ một tệp trên đĩa
Ctrl + Q + B	Dịch chuyển con trỏ về đầu khối
Ctrl + Q + K	Dịch chuyển con trỏ về cuối khối
Ctrl + Q + F	Tìm kiếm một cụm từ
Ctrl + Q + A	Tìm kiếm một cụm từ và sau đó thay thế bằng một cụm từ khác
Ctrl + Q + L	Lặp lại công việc Ctrl + Q + F hoặc Ctrl + Q + A cuối cùng

Giới thiệu chung

- Từ khóa
  - các từ dành riêng của ngôn ngữ C
  - từ khóa phải được sử dụng đúng cú pháp
  - một số từ khóa thông dụng

auto	**break**	**case**	**char**	**continue**	**default**
do	**double**	**else**	**extern**	**float**	**for**
goto	**if**	**int**	**long**	**register**	**return**
short	**sizeof**	**static**	**struct**	**switch**	**typedef**
union	**unsigned**	**void**	**volatile**	**while**	

Giới thiệu chung

- Tên (identifier)
  - Dùng để định danh các thành phần của chương trình
  - Tên biến, tên hàm, tên hằng, …
  - Tên là một dãy các kí tự gồm các chữ cái [a-z, A-Z, 0-9] và gạch nối "_"
  - Lưu ý:
    - tên không được chứa kí tự trống,
    - tên không được bắt đầu bằng một chữ số,
    - tên không được trùng với từ khóa
  - Nên đặt các tên gợi nhớ, có ý nghĩa
  - Tên chuẩn: một số tên có sẵn của trình biên dịch

Giới thiệu chung

- Hằng
  - là đại lượng có giá trị không thay đổi được trong chương trình
  - ví dụ
    - 111 hằng là một số
    - 'b' hằng là một kí tự
    - "lap trinh" hằng là một chuỗi kí tự
- *Biến*
  - là đại lượng có thể thay đổi được giá trị trong chương trình
- *Biểu thức*
  - là một công thức tính toán để có một giá trị theo một qui tắc toán học
  - ví dụ: x + y * z

Giới thiệu chung

- Mỗi một câu lệnh C đều phải kết thúc bởi một dấu ";"
- Lời chú thích được đặt giữa hai dấu "/*" và "*/"
  - Ví dụ

    /* Đây là một chú thích */
- Khi viết chương trình nên sử dụng các lời chú thích
- Trình biên dịch C phân biệt chữ in hoa và chữ in thường

Giới thiệu chung

- Các kiểu dữ liệu chuẩn
  - Kiểu kí tự
  - Kiểu số nguyên
  - Kiểu số thực

Giới thiệu chung

- Kiểu kí tự
  - Kiểu **char**
  - Chiếm một byte
  - Biểu diễn các kí tự trong bảng mã ASCII
  - Ví dụ
    - ‘a’ có giá trị mã ASCII là 65
    - ‘0’ có giá trị mã ASCII là 48
  - Kiểu kí tự đồng thời cũng là kiểu số nguyên
  - Có hai kiểu char: : **signed char** và **unsinged char**

Kiểu kí tự	Kích thước	Miền giá trị
signed char	1 byte	-128 -> 127
unsigned char	1 byte	0 -> 255

Giới thiệu chung

- Kiểu số nguyên
  - Có nhiều kiểu số nguyên

Kiểu số nguyên	Kích thước	Miền giá trị
int, short	2 byte	-32768 -> 32767
unsigned int, unsigned short	2 byte	0 -> 65535
long	4 byte	-2147483648 -> 2147483647
unsigned long	4 byte	0 -> 4294967295

Giới thiệu chung

- Kiểu số thực
  - Có nhiều kiểu số thực

Kiểu số thực	Kích thước	Miền giá trị
float	4 byte	3.4E-38 -> 3.4E+38
double	8 byte	1.7E-308 -> 1.7E+308
long double	10 byte	3.4E-4932 -> 1.1E+4932

Giới thiệu chung

- Kiểu số thực
  - Có hai cách biểu diễn số thực
    - Dạng thập phân: dùng dấu chấm để ngăn cách phần nguyên và phần thập phân
      - Ví dụ: -12.345672, 1203.8375
    - Dạng khoa học: gồm phần định trị và phần mũ của cơ số 10, hai phần cách nhau bởi chữ E hoặc e
      - Ví dụ: 6.123E+02

Giới thiệu chung

- Chuyển kiểu (casting)
  - Ngôn ngữ C cho phép chuyển kiểu: chuyển từ kiểu này sang kiểu khác
  - Cú pháp: **(kiểu_mới)biểu_thức**
  - Ví dụ

```
int i;
i = (int)10.45        /* i = 10 */
float x;
x = (float)1/3;       /* x = 1.0/3 = 0.3333 */
```

Giới thiệu chung

- Các phép toán
  - Các phép toán trên số nguyên
    - Cộng: +
    - Trừ: -
    - Nhân: *
    - Chia lấy phần nguyên: /
    - Chia lấy phần dư: %
  - Các phép toán trên số thực
    - Cộng: +
    - Trừ: -
    - Nhân: *
    - Chia: /

Giới thiệu chung

- Các phép toán
  - Các phép toán quan hệ (so sánh)
    - So sánh bằng nhau: ==
    - So sánh khác nhau: !=
    - So sánh lớn hơn: >
    - So sánh nhỏ hơn: <
    - So sánh lớn hơn hoặc bằng: >=
    - So sánh nhỏ hơn hoặc bằng : <=
  - Biểu thức chứa các phép toán quan hệ được gọi là biểu thức quan hệ
  - Biểu thức quan hệ có giá trị **đúng** hoặc **sai**

Giới thiệu chung

- Các phép toán
  - Các phép toán logic
    - Kiểu logic trong C không được định nghĩa một cách tường minh
    - Một giá trị khác 0 là đúng, một giá trị bằng 0 là sai

Phép toán	Kí hiệu	Ví dụ
Và (AND)	&&	2 && 0 = sai
Hoặc (OR)	||	10 || 5 = đúng
Phủ định (NOT)	!	!0 = đúng

Giới thiệu chung

- Các phép toán
  - Các phép toán trên bit
    - Phép OR từng bit: |
    - Phép AND từng bit: &
    - Phép XOR từng bit: ^
    - Phép đảo bit: ~
    - Phép dịch trái (nhân 2): <<
    - Phép dịch phải (chia 2): >>
  - Ví dụ
    - 3 & 5 = 1
    - a << n /* $a*(2^n)$ */
    - a >> n /* $a/(2^n)$ */

Giới thiệu chung

- Khái niệm hàm
  - Là đoạn chương trình viết ra một lần, được sử dụng nhiều lần
  - Mỗi lần sử dụng chỉ cần gọi tên hàm và cung cấp các tham số
- Cấu trúc chương trình

```
#include <...> /* Gọi các tệp tiêu đề trong chương trình */
#define ... /* Khai báo hằng số */
typedef /* Định nghĩa kiểu dữ liệu */
/* Nguyên mẫu các hàm: khai báo tên hằm và các tham số */
/* Khai báo các biến toàn cục */

main()
{
        /* Khai báo biến */
        /* Các câu lệnh */
}

/* Định nghĩa các hàm */
```

Giới thiệu chung

- Các khai báo
  - **#include**: dùng để gọi tệp tiêu đề
  - Khai báo biến: muốn sử dụng biến thì phải khai báo trước
    - Cú pháp: **kiểu_dữ liệu danh_sách_các_biến;**
    - Ví dụ
      - int x, y;
      - float a = 10.5, b; /* khai báo và khởi gán */
      - int a, b, c = 1;

Giới thiệu chung

- Các khai báo
  - Khai báo hằng
    - Có hai cách để khai báo hằng, hoặc sử dụng **#define** hoặc sử dụng từ khóa **const**
  - #define tên_hằng giá_trị_hằng
  - const kiểu_dữ_liệu tên_hằng = giá_trị_hằng;
    - Ví dụ
  - #define PI 3.14
    const float PI = 3.14;

Giới thiệu chung

- Phép gán
  - Gán giá trị cho một biến
  - Cú pháp: **tên_biến = biểu_thức;**
  - Ví dụ
    - x = 0;
    - y = z + 1;
  - Phép gán kép
    - x = y = z = 1;
    - x = y + (z = 2);

Giới thiệu chung

- Phép tăng 1 (++), giảm 1 (--)
  - Ngôn ngữ C cung cấp hai phép toán tăng 1 và giảm 1
  - Ví dụ
    - **x = x + 1;** sẽ được viết thành: **++x;** hoặc **x++;**
    - **y = y – 1;** sẽ được viết thành: **--y;** hoặc **y--;**
  - Sự khác nhau giữa khi toán tử ++ hoặc -- đứng trước hoặc sau biến là thể hiện trong phép gán: *biến = biểu_thức*
    - Nếu toán tử ++x (*--x*) xuất hiện trong *biểu_thức* thì x sẽ được tăng (giảm) 1 trước khi thực hiện phép gán
    - Nếu toán tử x++ (*x--*) xuất hiện trong *biểu_thức* thì thực hiện phép gán trước khi x được tăng (giảm) 1
  - Ví dụ
    - a = 5; b = ++a; kết quả ?
    - a = 5; b = a++; kết quả ?

Giới thiệu chung

- Tóm lại
  - Các từ khóa, tên
  - Các kiểu dữ liệu chuẩn
  - Các phép toán
  - Cấu trúc chung một chương trình C
  - Các khai báo
  - Phép gán
  - Phép tăng 1, giảm 1

Nội dung

- Giới thiệu chung
- Lệnh nhập/xuất
- Lệnh điều kiện
- Lệnh vòng lặp
- Hàm
- Kiểu mảng
- Xâu kí tự
- Kiểu cấu trúc (struct) và kiểu hợp (union)
- Làm việc với tệp

Lệnh nhập/xuất

- Lệnh xuất / hiển thị ***printf***
  - Ví dụ

```
#include <stdio.h>
void main()
{
        printf("Chào các bạn.\n");
}
```

- Cú pháp

  printf(chuỗi_điều_khiển [, danh_sách_các_tham_số]);

  Chuỗi điều khiển dùng để định dạng dữ liệu cần hiển thị

- Ví dụ

```
printf("a = %f\n", a);
```

Lệnh nhập/xuất

- Chuỗi điều khiển bao gồm 3 loại kí tự
  - Các kí tự điều khiển
    - \n sang dòng mới
    - \f sang trang mới
    - \b xóa kí tự bên trái
    - \t dấu tab
  - Các kí tự để đưa ra màn hình
  - Các kí tự định dạng và khuôn in
    - Các kí tự định dạng theo sau kí tự %
    - Ví dụ
      - %f
      - %d

Lệnh nhập/xuất

- Các kí tự định dạng thường dùng

Kí tự định dạng	Ý nghĩa
c	In ra một kí tự kiểu **char**
d	In ra số nguyên kiểu **int**
u	In ra số nguyên không dấu kiểu **unsigned int**
ld	In ra số nguyên kiểu **long**
lu	In ra số nguyên kiểu **unsigned long**
f	In ra số thực dạng m...m.n..n với phần thập phân có 6 chữ số, áp dụng cho kiểu **float**, **double**
s	In ra xâu kí tự
x	In ra số nguyên dưới dạng cơ số 16 (hexa)
o	In ra số nguyên dưới dạng cơ số 8
e, E	In ra số thực dạng khoa học m...m.E[+ hoặc -]xx, áp dụng cho kiểu **float**, **double**
g, G	Dùng %e hoặc %f tùy thuộc loại nào ngắn hơn

Lệnh nhập/xuất

- Ví dụ

printf(“%c và %c có mã ASCII tương ứng là %d và %d\n”, ‘a’, ‘A’, ‘a’, ‘A’);

Kết quả: a và A có mã ASCII tương ứng là 97 và 65

printf(“%f”, x); /* phần thập phân được hiển thị ngầm định là 6 chữ số */

x = 4.2 kết quả: 4.200000

X = 4.2345678 kết quả: 4.234568 /*làm tròn*/

printf(“Ví dụ \nxoa\b kí\b tự\b trái\b\n”);

Kết quả:

Ví dụ

xo k t tra

Lệnh nhập/xuất

- Khuôn in
  - Qui định cách thức in ra dữ liệu và chỉ rỏ số chổ dữ liệu sẽ chiếm, canh lề trái hay phải
  - Khuôn in có dạng: **%m** hay **%m.n**
  - Đối với số nguyên, mẫu ghi là **%md**
    - m là số nguyên chỉ ra số vị trí mà số nguyên chiếm
    - Ví dụ: printf("x = %4d", x);
    - Kết quả: nếu x = 12 in ra ^^12
      nếu x = 12345 in ra 12345
  - Đối với số thực, mẫu ghi là **%m.nf**
    - m là tổng số chữ viết ra, n là số chữ số phần thập phân
    - Ví dụ: printf("x = %4.2f", x);
    - Kết quả: nếu x = 1.234 in ra ^1.23

Lệnh nhập/xuất

- In kí tự đặc biệt
  - \’ In ra dấu ’
  - \” In ra dấu ”
  - \\ In ra dấu \
- Các lệnh xuất dữ liệu khác
  - puts(chuỗi_kí_tự): hiển thị chuỗi kí tự
    - Ví dụ: puts(“Chào bạn”);
  - putchar(kí_tự): hiển thị một kí tự
    - Ví dụ: putchar(‘a’);

Lệnh nhập/xuất

- Lệnh nhập dữ liệu **scanf**
- Ví dụ

```
#include <stdio.h>
void main()
{
        float r, dien_tich;
        printf("Nhập vào bán kính: ");
        scanf("%f", &r);
        dien_tich = 3.14 * r * r;
        printf("Diện tích là: %f\n", dien_tich);
        getch();
}
```

- Cách sử dụng lệnh **scanf** gần giống với lệnh **printf**

Lệnh nhập/xuất

- Lệnh scanf
  - Cú pháp

    scanf(chuỗi_điều_khiển [, danh_sách_tham_số]);
    - chuỗi_điều_khiển cho phép định dạng dữ liệu nhập vào
    - danh_sách_tham_số là địa chỉ các biến cần nhập dữ liệu
  - Để lấy địa chỉ một biến, sử dụng toán tử &

Lệnh nhập/xuất

- Lệnh scanf

Kí tự định dạng	Ý nghĩa
c	Nhập vào một kí tự kiểu **char**
d	Nhập vào số nguyên kiểu **int**
u	Nhập vào số nguyên không dấu kiểu **unsigned int**
ld	Nhập vào số nguyên kiểu **long**
lu	Nhập vào số nguyên kiểu **unsigned long**
f	Nhập vào số thực dạng m...m.n..n với phần thập phân có 6 chữ số, áp dụng cho kiểu **float**, **double**
s	Nhập vào xâu kí tự, không chứa dấu cách (space)
x	Nhập vào số nguyên dưới dạng cơ số 16 (hexa)
o	Nhập vào nguyên dưới dạng cơ số 8

Lệnh nhập/xuất

- Một số lệnh nhập dữ liệu khác
  - gets(char *str): nhận chuỗi kí tự vào từ bàn phím cho dến khi gặp “\n”
  - getchar(): nhận kí tự nhập vào
    - Ví dụ: ch = getchar();
  - getch(): nhận kí tự nhập vào và không cho hiển thị kí tự đó trên màn hình
  - getche(): nhận kí tự nhập vào và cho hiển thị kí tự đó trên màn hình

Lệnh nhập/xuất

- Một số lệnh khác liên quan đến xuất/nhập
  - fflush(): xóa vùng đệm bàn phím
  - kbhit(): kiểm tra bộ đệm bàn phím, bộ đệm rỗng trả về giá trị 0, ngược lại trả về giá trị khác 0
  - clrscr(): xóa màn hình
  - gotoxy(int x, int y): di chuyển con trỏ màn hình đến vị trí cột x (1→80), và dòng y (1→25)

Lệnh nhập/xuất

- Bài tập
  - Nhập vào 3 số thực, tính tổng của chúng và in ra màn hình
  - Tính diện tích tam giác khi biết chiều cao và cạnh đáy

Tóm lại

- Lệnh nhập dữ liệu
  - printf
  - putchar
  - puts
- Lệnh xuất dữ liệu
  - scanf
  - getchar
  - gets
- Một số lệnh liên quan khác

Nội dung

- Giới thiệu chung
- Lệnh nhập/xuất
- Lệnh điều kiện
- Lệnh vòng lặp
- Hàm
- Kiểu mảng
- Xâu kí tự
- Kiểu cấu trúc (struct) và kiểu hợp (union)
- Làm việc với tệp

Lệnh điều kiện

- Lệnh
  - Một câu lệnh nhằm thực hiện một công việc nào đó
  - Câu lệnh kết thúc bởi dấu “;”
  - Ví dụ
    - printf(“một câu lệnh\n”);
    - i++;
- Khối lệnh
  - Là dãy các lệnh được đặt giữa cặp ngoặc nhọn “{“ và “}”
  - Khối lệnh thường được sử dụng khi muốn chúng thực hiện dưới một điều kiện nào đó

```
{
/* các lệnh */
}
```

Lệnh điều kiện

- Lệnh **if**
  - Thực hiện một trong hai khối lệnh tùy thuộc vào giá trị của biểu thức điều kiện
  - Lệnh **if** có hai dạng: dạng đầy đủ **if … else** và dạng chỉ có **if**
  - Cú pháp

    if (biểu thức điều kiện) *(dạng 1)*

    khốI lệnh 1;

    else

    khối lệnh 2;

    Hoặc

    if (biểu thức điều kiện) *(dạng 2)*

    khối lệnh 1;

Lệnh điều kiện

- Lệnh **if**
  - Ý nghĩa
    - Dạng 1: nếu biểu thức điều kiện có giá trị đúng (có giá trị khác không), khối lệnh 1 sẽ được thực hiện; nếu điều kiện là sai (có giá trị bằng không) thì khối lệnh 2 sẽ được thực hiện
    - Dạng 2: nếu biểu thức điều kiện là đúng (có giá trị khác không), khối lệnh 1 sẽ được thực hiện; nếu điều kiện là sai (có giá trị bằng không) thì thực hiện câu lệnh đứng sau khối lệnh 1
  - Mô tả hai dạng của lệnh if bằng sơ đồ khối
    - ???

Lệnh điều kiện

- Lệnh **if**
  - Ví dụ 1: tính giá trị nhỏ nhất của hai số

```
#include <stdio.h>
main()
{
        int a, b, min;
        printf("Nhập vào hai số nguyên a và b.\n");
        printf("a = ");
        scanf("%d", &a);
        printf("b = ");
        scanf("%d", &b);
        if (a < b)
                min = a;
        else
                min = b;
        printf("min = %d\n", min);
}
```

Lệnh điều kiện

- Lệnh **if**
  - Ví dụ 2: viết lại chương trình tìm giá trị nhỏ nhất của 2 số sử dụng dạng **if** không có **else**
  - Ví dụ 3: trường hợp sử dụng khối lệnh

```
if (a > b)
{
        max = a;
        min = b;
}
else
{
        max = b;
        min = a;
}
```

Lệnh điều kiện

- Lệnh **if**
  - Có thể sử dụng các toán tử “&&” và “||” để xây dựng các biểu thức điều kiện phức tạp hơn
  - Chẳng hạn
    - if ((đk1 && đk2) || đk3)
  - Ví dụ: viết biểu thức điều kiện kiểm tra 3 số thực là 3 cạnh tam giác

Lệnh điều kiện

- Một số lưu ý khi sử dụng lệnh **if**
  - Biểu thức điều kiện phải luôn đặt trong trong hai dấu “(“ và “)”
  - Biểu thức điều kiện là đúng, nếu nó có giá trị khác 0 và là sai nếu nó có giá trị bằng 0
  - Biểu thức điều kiện có thể là số nguyên hoặc thực
  - Nếu sau **if** hoặc **else** là một dãy các câu lệnh, thì các câu lệnh này phải được đặt trong cặp dấu ngoặc “{“ và “}”

Lệnh điều kiện

- Sử dụng lệnh **if** lồng nhau
  - Ví dụ: chương trình tính nghiệm phương trình ax+b=0

```
#include <stdio.h>
main()
{
        float a, b, x;
        printf("Nhap vao a va b:");  scanf("%f%f", &a, &b);
        if (a == 0){
                if (b == 0)
                        printf("Phuong trinh co vo so nghiem\n");
                else
                        printf("Phuong trinh vo nghiem\n");
        }
        else{
                x = -b/a;
                printf("Phuong trinh co nghiem x = %f\n", x);
        }
}
```

Lệnh điều kiện

- Sử dụng lệnh **if** lồng nhau
  - Khi sử dụng các lệnh **if** lồng nhau, nên sử dụng các dấu đóng mớ ngoặc "{}" để tránh gây ra sự hiểu nhầm **if** nào tương ứng với **else** nào
  - Ví dụ

```
if (a != 0)
        if (a > b)
                y = b/a;
        else
                y = -b/a;
```

```
if (a != 0)
{
        if (a > b)
                y = b/a;
        else
                y = -b/a;
}
```

Lệnh điều kiện

- Sử dụng **else if**
  - Khi muốn sử dụng một trong n quyết định, sử dụng dạng lệnh **if** như sau

```
if (điều kiện 1)
        khối lệnh 1;
else if (điều kiện 2)
        khối lệnh 2;
…
else if (biểu thức n-1)
        khối lệnh n-1;
else
        khối lệnh n;
```

Lệnh điều kiện

- Sử dụng **else if**
  - Ví dụ: Chương trình xếp loại kết quả học tập của một sinh viên

```
#include <stdio.h>
main()
{
        float diem;
        printf("Nhap diem vao");  scanf("%f", &diem);
        if (diem < 5)
                printf("Xep loai: kem");
        else if (diem < 7)
                printf("Xep loai: trung binh");
        else if (diem < 8)
                printf("Xep loai: kha");
        else if (diem < 9)
                printf("Xep loai: gioi");
        else
                printf("Xep loai: xuat sac");
}
```

Lệnh điều kiện

- Bài tập
  - Viết chương trình giải một phương trình bậc 2

Toán tử "?:"

- Có thể sử dụng toán tử "?:" thay cho lệnh **if**
- Cú pháp

  (điều kiện) ? lệnh 1 : lệnh 2;

  nếu điều kiện là đúng lệnh 1 sẽ được thực hiện, nếu không lệnh 2 sẽ được thực hiện
- Ví dụ

  (a > b) ? max = a : max = b;

  - Hoặc

    max = (a > b) ? a : b;

Lệnh switch … case

- Lệnh **if** chỉ cho phép chọn một trong hai phương án
- Lệnh **switch … case** cho phép chọn một trong nhiều phương án khác nhau
- Cú pháp

```
switch (biểu thức nguyên)
{
 case n1:
            Các câu lệnh;
 case n2:
            Các câu lệnh;
            ...
 case nk:
            Các câu lệnh;
 [default:  Các câu lệnh;]
}
```

Lệnh switch … case

- Ý nghĩa câu lệnh
  - Nếu biểu thức nguyên có giá trị bằng nhãn n_i thì máy sẽ nhảy đến thực hiện các lệnh của nhãn đó, nếu không thì máy sẽ nhảy đến thực hiện các lệnh trong thành phần tùy chọn **default**
  - Máy sẽ ra khỏi toán tử **switch** khi nó gặp câu lệnh **break**, **return** hoặc nó gặp dấu "}" của câu lệnh **switch**
  - Chú ý, khi máy nhảy tới nhãn n_i, nếu kết thúc dãy lệnh trong nhãn này không có câu lệnh **break** hoặc **return** thì máy sẽ tiếp tục thực hiện các lệnh trong nhãn n_{i+1}
  - Thường cuối mỗi dãy lệnh của một nhãn có một lệnh **break**

Lệnh switch … case

- Ví dụ

```
#include <stdio.h>
main()
{
  int n;
  printf(" Nhập vào một số nguyên từ 0 đến 2: ");
  scanf("%d", &n);
  switch(n)
  {
            case 0:    printf("Số không\n");
                       break;
            case 1:    printf("Số một\n");
                       break;
            case 2:    printf("Số hai\n");
                       break;
            default:   printf("Không đúng\n");
  }
  printf("Kết thúc\n");
}
```

Lệnh switch … case

- Ví dụ: thiếu lệnh **break**

```
#include <stdio.h>
main()
{
        int n;
        printf(" Nhập vào một số nguyên từ 0 đến 2: ");
        scanf("%d", &n);

        switch(n)
        {
                case 0: printf("Số không\n");
                case 1: printf("Số một\n");
                case 2: printf("Số hai\n");
        }
         printf("Kết thúc\n");
  }
```

Lệnh switch … case

- Bài tập
  - Viết chương trình nhập vào hai số thực *a*, *b* và một ký hiệu *op*, *op* là một trong các ký hiệu +, -, *, /. Hãy xuất kết quả của biểu thức *a op b* ra màn hình.

Nội dung

- Giới thiệu chung
- Lệnh nhập/xuất
- Lệnh điều kiện
- Lệnh vòng lặp
- Hàm
- Kiểu mảng
- Xâu kí tự
- Kiểu cấu trúc (struct) và kiểu hợp (union)
- Làm việc với tệp

Lệnh vòng lặp

- Thực hiện một công việc nào đó được lặp đi lặp lại nhiều lần
- Ví dụ
  - In ra màn hình các số từ 1 đến 10, mỗi số trên một dòng
    - Giải pháp đơn giản
      - printf(“1\n”);
      - printf(“2\n”);
      - …
      - printf(“10\n”);
    - Giải pháp tổng quát
      - Dùng vòng lặp

Lệnh vòng lặp

- Các lệnh vòng lặp
  - Lệnh **for**
  - Lệnh **while**
  - Lệnh **do** ... **while**

Lệnh vòng lặp

- Lệnh lặp **for**
  - Cú pháp

    for ([biểu thức 1]; [biểu thức 2]; [biểu thức 3])
    khối lệnh;
    - Các thành phần trong ngoặc "[" và "]" là tùy chọn, không bắt buộc
    - Các dấu ";" và cặp ngoặc "(" và ")" là bắt buộc phải có
  - Ý nghĩa câu lệnh: lệnh for hoạt động theo các bước
    1. Tính biểu thức 1.
    2. Tính biểu thức 2. Nếu biểu thức 2 có giá trị 0 (sai), máy sẽ ra khỏi **for** và chuyển tới câu lệnh sau thân **for**. Nếu biểu thức 2 có giá trị khác 0 (đúng), máy thực hiện các câu lệnh trong thân **for**, sau đó chuyển tới bước 3.
    3. Tính biểu thức 3, sau đó quay trở lại bước 2 để bắt đầu các bước lặp mới.

Lệnh vòng lặp

- Ví dụ

```
#include <stdio.h>
main()
{
        int i;
        for (i=1; i <=10; i++) printf("%d\n", i);
        getch();
}
```

- Có thể viết cách khác đoạn chương trình trên không ?
- Cách biểu diễn bằng sơ đồ khối lệnh **for** như thế nào ?

Lệnh vòng lặp

- Ví dụ: tính tổng n số tự nhiên đầu tiên

```
#include <stdio.h>
main()
{
        int n, s, i;
        /*nhập n*/
        printf("n = ");
        scanf("%d", &n);
        s = 0;
        for (i = 1; i <= n; i++)
                s = s + i;
        printf("Tổng của %d số tự nhiên đầu tiên là: %d\n", n, s);
}
```

Lệnh vòng lặp

- Nhận xét
  - Biểu thức 1 chỉ được tính một lần
  - Biểu thức 2, biểu thức 3 và khối lệnh trong thân lệnh **for** được lặp đi lặp lại nhiều lần
  - Khi biểu thức 2 vắng mặt thì nó được xem là đúng
    - Để thoát khỏi lệnh for trong trường hợp này phải dùng lệnh **break** hoặc **return**
  - Có thể sử dụng các lệnh **for** lồng nhau
- Câu lệnh sau làm gì ?
  - for(;;){}

Lệnh vòng lặp

- Lệnh lặp **while**
  - Cú pháp

    while (biểu thức)

    khối lệnh;
  - Ý nghĩa
    - Trong khi *biểu thức* có giá trị đúng, tức khác 0, thì còn phải thực hiện khối lệnh. Việc lặp dừng lại khi biểu thức có giá trị sai (bằng 0).
    - Lệnh **while** kiểm tra điều kiện trước khi thực hiện khối lệnh
  - Hãy vẽ sơ đồ khối biểu diễn lệnh **while**

Lệnh vòng lặp

- Ví dụ
  - Viết lại chương trình tính tổng n số tự nhiên đầu tiên sử dụng lệnh **while**

```
#include <stdio.h>
main()
{
        int n, s, i;
        printf("n = "); scanf("%d", &n);
        s = 0;
        i = 1;
        while (i <= n) {
                s = s + i;
                i++;
        }
        printf("Tổng của %d số tự nhiên đầu tiên là: %d\n", n, s);
}
```

Lệnh vòng lặp

- Nhận xét
  - Biểu thức điều kiện luôn dược đặt trong cặp dấu "(" và ")"
  - Biểu thức điều kiện sẽ được tính toán đầu tiên nên phải có giá trị xác định
- Câu lệnh sau làm gì ?
  - while(0) printf("nothing\n");
- Hãy chuyển lệnh **for** dạng tổng quát thành lệnh **while**

Lệnh vòng lặp

- Lệnh lặp **do … while**
  - Cú pháp

    do

    khối lệnh;

    while (biểu thức);
  - Ý nghĩa
    - Thực hiện khối lệnh trong khi biểu thức có giá trị đúng, tức là khác 0
    - Thực hiện khối lệnh trước khi kiểm tra biểu thức điều kiện
    - Khối lệnh được thực hiện ít nhất 1 lần
  - Hãy vẽ sơ đồ khối biểu diễn lệnh **do … while**

Lệnh vòng lặp

- Ví dụ
  - Viết chương trình nhập vào một số lớn hơn 10

```
#include <stdio.h>
main()
{
        int n;
        do
        {       printf(" Hãy cho một số > 10 :");
                scanf("%d", &n);
                printf(" Bạn đã đọc một số  %d\n", n);
        }while  (n <= 10);

        printf(" Đúng số lớn hơn 10 rồi.");
}
```

Lệnh vòng lặp

- Ví dụ (tiếp)
  - Nếu dùng lệnh while

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

        printf(" Hãy cho một số > 10 :");
        scanf("%d", &n);
        printf(" Bạn đã đọc một số  %d\n", n);
        while  (n <= 10)
        {       printf(" Hãy cho một số > 10 :");
                scanf("%d", &n);
                printf(" Bạn đã đọc một số  %d\n", n);
        }
        printf(" Đúng số lớn hơn 10 rồi.");
}
```

Lệnh vòng lặp

- Hãy chuyển lệnh **do … while** thành lệnh **while** tương ứng
- Hãy viết lại chương trình tính tổng n số tự nhiên đầu tiên dùng **do … while**
- Viết câu lệnh nhập vào các kí tự và dừng lại khi kí tự nhập vào là ‘@’

Lệnh break

- Lệnh **break** thường được sử dụng kết hợp lệnh lặp
- Lệnh **break** dùng để thoát khỏi vòng lặp
- Nếu có nhiều lệnh lặp lồng nhau thì lệnh **break** chỉ thoat vòng lặp trực tiếp chứa nó
- Lệnh **break** cũng dùng để thoát khỏi lệnh **switch … case**

Lệnh break

- Ví dụ

```
#include <stdio.h>
main()
{
int i;
for (i=1; i <=5; i++)
{
        printf("Bắt đầu vòng %d\n", i);
        printf("Chào bạn\n");
        if (i==3) break;
        printf("Kết thúc vòng %d\n", i);
}
printf("Hết vòng lặp.");
}
```

- Kết quả ?
- Hãy vẽ sơ đồ khối mô tả chương trình trên

Lệnh continue

- Lệnh **continue** dùng để quay trở lại từ đầu để thực hiện lần lặp mới mà không cần thực hiện phần còn lại
- Ví dụ

```
#include <stdio.h>
main()
{ int i;
  for (i=1; i <= 5; i++)
  {
            printf(" Bắt đầu %d\n", i);
            if (i<4) continue;
            printf(" Chào bạn\n");
  }
}
```

Lệnh continue

- Đoạn chương trình sau làm gì ?

```
for (i = 1; i <= 4; i++)
        for (j = 1; j <= 10; j++)
        {
                printf("%d", j);
                if (j != 10) continue;
                printf("\n");
        }
```

Tóm lại

- Lệnh điều kiện
  - Lệnh **if**
  - Toán tử "?:"
  - Lệnh **switch … case**
- Lệnh lặp
  - Lệnh **for**
  - Lệnh **while**
  - Lệnh **do … while**
- Các lệnh liên quan
  - Lệnh break
  - Lệnh continue

Kỹ thuật lập trình(5):
ngôn ngữ lập trình C

Tống Minh Đức
Khoa Công nghệ thông tin
Học viện Kỹ thuật Quân sự
100 – Hoàng Quốc Việt – Hà Nội

Nội dung

- Giới thiệu chung
- Lệnh nhập/xuất
- Lệnh điều kiện
- Lệnh vòng lặp
- Hàm
- Kiểu mảng
- Xâu kí tự
- Kiểu cấu trúc (struct) và kiểu hợp (union)
- Làm việc với tệp

Hàm

- Hàm (function) là một dãy các lệnh nhằm thực hiện một công việc nào đó, thường được sử dụng nhiều lần
- Ví dụ
  - Việc tính sin, cos, tan, … trong toán học
  - Xây dựng các hàm tính sin, cos, …
- Một chương trình C là một dãy các hàm, trong đó có một hàm chính, được đặt tên là **main**

Hàm

- Ví dụ

```
#include <stdio.h>
float max2so(float a, float b); /* Nguyên mẫu của hàm */
main() /* bắt đầu hàm chính */
{
        float x, y;
        printf("Nhập vào 2 số: ");
        scanf("%f%f", &x, &y);
        printf("Giá trị lớn nhất của %f và %f là %f\n", x, y, max2so(x, y));
} /* kết thúc hàm main */

/* định nghĩa hàm max2so */
float max2so(float a, float b)
{
        float max; /* khai báo biến cục bộ */
        max = a > b ? a : b;
        return max;
}
```

Hàm

- Các khái niệm
  - Tên hàm
  - Kiểu giá trị trả về của hàm
  - Đối hay tham số hình thức
  - Thân hàm
  - Nguyên mẫu hàm / khai báo hàm
  - Lời gọi hàm
  - Tham số thực

Hàm

- Định nghĩa hàm
  - Cú pháp

```
kiểu_dữ_liệu_trả_về_của_hàm  tên_hàm ([khai_báo_các_tham_số])
{
                    các khai báo dùng riêng bên trong hàm nếu có;
                    các lệnh bên trong hàm;
                    [return giá_trị_trả_về;]
  }
```

  - Định nghĩa hàm có thể đặt trước hoặc sau hàm main
    - Nếu định nghĩa hàm đặt sau hàm **main** thi phải khai báo nguyên mẫu hàm ở đầu chương trình
    - Nên định nghĩa hàm sau hàm **main** và khai báo nguyên mẫu hàm

Hàm

- Định nghĩa hàm
  - Kiểu dữ liệu trả về của hàm và kiểu dữ liệu tham số là kiểu dữ liệu chuẩn hoặc do người lập trình định nghĩa
  - Tên hàm và tên tham số đặt theo quy tắc tên biến
  - Câu lệnh **return** là tùy chọn
    - Nếu hàm không trả về giá trị, thì không cần có lệnh **return**
    - Nếu hàm trả về giá trị thì bắt buộc phải có lệnh **return**, trong trường hợp này giá trị trả về phải có cùng kiểu với kiểu dữ liệu trả về của hàm
  - Nếu hàm không trả về giá trị thì khai báo kiểu trả về của hàm là **void**
  - Nếu hàm không có tham số hình thức có thể sử dụng từ khóa **void**, hoặc không khai báo gì cả

Hàm

- Bài tập
  - Viết hàm kiểm tra 3 số thực có là 3 cạnh của tam giác
  - Mở rộng: nếu là 3 cạnh tam giác thì xác định đó là tam giác gì (cân, vuông, đều)

Hàm

- Lưu ý
  - Không cho phép định nghĩa một hàm bên trong hàm khác
  - Các tham số hình thức và các biến định nghĩa bên trong hàm (biến cục bộ) chỉ được sử dụng bên trong hàm đó

Hàm

- Lời gọi hàm
  - Hàm được sử dụng thông qua lời gọi hàm
  - Cú pháp: **tên_hàm ([danh sách các tham số thực]);**
  - Cần phân biệt
    - Tham số hình thức hay đối: xuất hiện trong định nghĩa hàm
    - Tham số thực: xuất hiện trong lời gọi hàm
  - Ví dụ
    - max2so(12, 341);
  - Lưu ý
    - Số tham số thực phải bằng số tham số hình thức
    - Kiểu các tham số thực phải phù hợp với kiểu của các tham số hình thức

Hàm

- Ví dụ: viết hàm tính n!

```
#include <stdio.h>
long giai_thua(int n); /* nguyên mẫu hàm */
main()
{
        int n;
        long gt;
        printf("\nn = "); scanf("%d", &n); /* Đọc số n */
        gt = giai_thua(n); /* gọi hàm tính giai thừa */
        printf("\n n! = %ld\n", gt); /* In ra kết quả */
}
long giai_thua(int n)
{
        int i;
        long gt = 1;
        if (n < 0) gt = 0;
        else
            for (i=2; i <=n; i++)  gt = gt * i;
        return (gt);
}
```

Hàm

- Biến toàn cục, biến cục bộ
  - Biến toàn cục: được khai báo bên ngoài thân hàm, thường ở đầu chương trình
  - Biến cục bộ: được khai báo bên trong thân hàm
  - Phạm vi hoạt động
    - Biến toàn cục được sử dụng kể từ vị trí khai báo đến cuối chương trình
    - Biến cục bộ chỉ được sử dụng bên trong hàm đó
  - Thời gian sống
    - Biến toàn cục kết thúc thời gian sống khi chương trình kết thúc
    - Sau khi hàm kết thúc hoạt động thì các tham số hình thức và các biến cục bộ cũng kết thúc thời gian sống của chúng

Hàm

- Ví dụ

```
#include <stdio.h>
int i;   /* Biến toàn cục */
void vi_du(void);

main()
{
        for (i=1; i <=5; i++) vi_du();
}

void vi_du(void)
{
        int m = 3;  /* Biến cục bộ */
        m++;
        printf(" %d %d\n", i, m);
}
```

Hàm

- Lưu ý
  - Biến toàn cục được sử dụng trong khắp chương trình
  - Việc thay đổi tùy tiện giá trị của biến toàn cục sẽ rất khó kiểm soát chương trình
    - Dễ sinh lỗi
  - **Hạn chế sử dụng biến toàn cục**

Hàm

- Ví dụ về phạm vi hoạt động biến

```
#include <stdio.h>
int a = 5; /* Biến toàn cục */

void ham_vi_du()
{
        int cuc_bo = 1;      /* Biến cục bộ */
        int a = 10;          /* Biến cục bộ */
        printf("Biến cục bộ trong hàm ví dụ:\n cuc_bo = %d\n a = %d\n",
                cuc_bo, a);
}
main()
{
        int cuc_bo = 100;    /* Biến cục bộ */
        ham_vi_du();         /* Gọi hàm ví dụ */
        printf("Biến cục bộ trong hàm main:\n cuc_bo = %d\n", cuc_bo);
        printf("Biến toàn cục:\n a = %d\n", a);
}
```

Hàm

- Biến cục bộ được chia làm hai loại
  - Biến cục bộ động
    - Biến được cấp phát bộ nhớ tự động mỗi khi có lời gọi hàm
    - Biến cục bộ động không lưu giữ giá trị mỗi khi hàm kết thúc (tức bị giải phóng khỏi bộ nhớ)
  - Biến cục bộ tĩnh
    - Biến cục tĩnh được khai báo bên trong thân hàm nhưng vẫn **tồn tại ngay cả khi hàm đã kết thúc hoạt động**
    - Biến cục bộ tĩnh được khai báo với từ khóa **static**

Hàm

- Ví dụ

```
#include <stdio.h>
void vi_du(void);
main()
{
        int n;
        for (n=1; n<=5; n++)
                vi_du();
}

void vi_du(void)
{
        static int i;
        i++;
        printf(" Goi lan thu %d\n", i);
}
```

Lần đầu tiên có lời gọi hàm giá trị biến i được khởi tạo giá trị 0

Hàm

- Sự giống nhau giữa biến cục bộ tĩnh và biến toàn cục
  - Cùng đều tồn tại trong suốt thời gian chương trình hoạt động
- Sự khác nhau giữa biến cục bộ tĩnh và biến toàn cục
  - Biến toàn cục được sử dụng kể từ vị trí nó khai báo đến cuối chương trình
  - Biến cục bộ tĩnh chỉ được sử dụng trong thân hàm nó được khai báo

Hàm

- Địa chỉ (address)
  - Với mỗi biến có các khái niệm
    - Tên biến, kiểu biến, giá trị biến
  - Ví dụ:
    - int i = 1;
    - Biến i kiểu số nguyên có giá trị là 1
    - Máy tính cấp phát một khoảng nhớ 2 byte liên tục để lưu trữ giá trị của biến i
  - **Địa chỉ biến** là số thứ tự của byte đầu tiên trong dãy các byte liên tục nhau máy dành để lưu trữ giá trị biến
  - Để lấy địa chỉ biến, sử dụng toán tử "**&**"
    - Ví dụ: &i
  - Lưu ý, máy tính phân biệt các kiểu địa chỉ: địa chỉ kiểu **int**, địa chỉ kiểu **float**, địa chỉ kiểu **long**, …

Hàm

- Con trỏ (pointer)
  - Là một biến dùng để chứa địa chỉ
  - Có nhiều loại con trỏ tương ứng với các kiểu địa chỉ khác nhau
    - Chẳng hạn, con trỏ kiểu **int** tương ứng địa chỉ kiểu **int**, …
  - Cú pháp khai báo con trỏ

    kiểu_dữ_liệu *tên_con_trỏ;
  - Ví dụ
    - int i, j, *pi, *pj;
    - pi = &i; /* pi là con trỏ chứa địa chỉ biến i */
    - pj = &j; /* pj là con trỏ chứa địa chỉ biến j */

Hàm

- Con trỏ
  - Giả sử có
    - px là con trỏ đến biến x, thì các cánh viết x và *px là tương đương nhau
  - Ví dụ

```
int x, y, *px, *py;
px = &x;
py = &y;
x = 3; /* tương đương với *px = 3 */
y = 5; /* tương đương với *py = 5 */
/* Các câu lệnh dưới đây là tương đương: */
x = 10 * y;
*px = 10 * y;
x = 10 * (*py);
*px = 10 * (*py);
```

Hàm

- Hàm có tham số là con trỏ
  - Xét chương trình

```
#include <stdio.h>
void hoan_vi(int a, int b); /* nguyên mẫu hàm, prototype */
main()
{ int n=10, p=20;
  printf(" Trước khi gọi hàm : %d %d\n", n, p);
  hoan_vi(n, p);
  printf(" Sau khi gọi hàm   : %d %d\n", n, p);
}

void hoan_vi(int a, int b)
{ int t;
  printf(" Trước khi hoán vị : %d %d\n", a, b);
  t=a;
  a=b;
  b=t;
  printf(" Sau khi hoán vị   : %d %d\n", a, b);
}
```

Hàm

- Hàm có tham số là con trỏ
  - Chương trình trên cho kết quả không đúng
  - Tại sao ?
  - Do cơ chế biến cục bộ hay tham số hình thức bị giải phóng bộ nhớ khi hàm kết thúc
  - Truyền tham số thực cho hàm là địa chỉ biến thay vì truyền giá trị biến
    - **Sử dụng tham số là con trỏ**

Hàm

- Ví dụ

```
#include <stdio.h>
void hoan_vi(int *a, int *b);
main()
{           int n = 10, p = 20;
            printf(" Trước khi gọi hàm : %d %d\n", n, p);
            hoan_vi(&n, &p);
            printf(" Sau khi gọi hàm   : %d %d\n", n, p);
}

void hoan_vi(int *a, int *b)
// a và b bây giờ là 2 địa chỉ
{           int t;
            printf(" Trước khi hoán vị : %d %d\n", *a, *b);
            t = *a;    /* t nhận giá trị chứa trong địa chỉ a */
            *a = *b;
            *b = t;
            printf(" Sau khi hoán vị   : %d %d\n", *a, *b);
}
```

Hàm

- Khi nào thì dùng tham số là con trỏ ?
  - Cần phân biệt hai loại tham số hình thức
    - Tham số hình thức chỉ nhận giá trị truyền vào để hàm thao tác, trường hợp có thể gọi là tham số vào
    - Tham số hình thức dùng để chứa kết quả của hàm, trường hợp này có thể gọi là tham số ra
  - Đối với tham số ra ta phải sử dụng kiểu con trỏ

- Bài tập
  - Giải thích tham số của lệnh **scanf**
  - Viết hàm giải phương trình bậc hai

Hàm

- Hàm đệ qui
  - Là hàm mà từ trong thân hàm có lời gọi tới chính hàm đó
  - Hàm đệ qui được xây dựng dựa trên định nghĩa đệ qui trong toán học
  - Ví dụ: định nghĩa giai thừa của n (n!)

    n! = 1.2.3…n
    - Hoặc

      n! = 1 khi n = 0

      n.(n-1)! khi n >= 1

Hàm

- Hàm đệ qui
  - Viết hàm đệ qui tính n!

```
long giai_thua (int n)
{
        if (n==0)
                return(1);
        else
                return (n * giai_thua(n-1));
}
```

  - Sử dụng hàm đệ qui cần một bộ nhớ xếp chồng LIFO (Last In, First Out stack) để lưu trữ các giá trị trung gian
  - Giải thích cơ chế hoạt động hàm giai_thua với lời gọi hàm *giai_thua(3)*

Hàm

- Hàm đệ qui
  - Điều gì xảy ra nếu có lời gọi hàm sau
    - *k = giai_thua (-1);*
  - Khắc phục ?
  - Hạn chế của hàm đệ qui
    - Dùng nhiều bộ nhớ
  - Hãy viết lại hàm *giai_thua* sử dụng vòng lặp
  - So sánh hai cách viết đệ qui và lặp

Hàm

- Hàm đệ qui
  - Hàm đệ qui thường phù hợp để giải quyết các bài toán có đặc trưng
    - Bài toán dễ dàng giải quyết trong một số trường hợp riêng, đó chính là **điều kiện dừng đệ qui**
    - Trong trường hợp tổng quát, bài toán suy về cùng dạng nhưng giá trị tham số bị thay đổi

Hàm

- Hàm đệ qui
  - Ví dụ: tìm ước số chung lớn nhất của hai số nguyên dương
    - Ước số chung lớn nhất của hai số nguyên dương được định nghĩa như sau
      - nếu $x = y$ thì $usc(x, y) = x$
      - nếu $x > y$ thì $usc(x, y) = usc(x-y, y)$
      - nếu $x < y$ thì $usc(x, y) = usc(x, y-x)$

Hàm

- Hàm đệ qui
  - Ví dụ: tìm ước số chung lớn nhất của hai số nguyên dương

```
int usc(int x, int y)
{
        if (x == y)
                return (x);
        else if (x > y)
                return usc(x-y, y);
        else
                return usc(x, y-x);
}
```

Hàm

- Bài tập
  - Viết lại hàm *usc* dùng vòng lặp
  - Hãy viết chương trình sử dụng hàm đệ qui để tạo dãy số Fibonacci

    Dãy số Fibonacci là dãy số F1, F2, F3, ... Fn được tạo ra với công thức:

    Fn = Fn-1 + Fn-2

    Với F1=1, F2=1

    Ví dụ: 1, 1, 2, 3, 5, 8, 13, 21, ...

Hàm

- Hàm chuẩn
  - Là các đã được định nghĩa sẵn
  - printf, scanf, puts, gets, … (tệp tiêu đề stdio.h)
  - clrscr, getch, getche, … (tệp tiêu đề conio.h)
  - rand, randomize, … (tệp tiêu đề stdlib.h)
  - abs, fabs, sqrt, sin, cos, tan, … (tệp tiêu đề math.h)
  - …

Tóm lại

- Khái niệm hàm
  - Định nghĩa hàm
  - Sử dụng hàm
- Địa chỉ
- Con trỏ
- Tham số là con trỏ
- Hàm đệ qui
- Hàm chuẩn

Kỹ thuật lập trình(6):
ngôn ngữ lập trình C

Tống Minh Đức
Khoa Công nghệ thông tin
Học viện Kỹ thuật Quân sự
100-Hoàng Quốc Việt – Hà Nội

Nội dung

- Giới thiệu chung
- Lệnh nhập/xuất
- Lệnh điều kiện
- Lệnh vòng lặp
- Hàm
- Kiểu mảng
- Xâu kí tự
- Kiểu cấu trúc (struct) và kiểu hợp (union)
- Làm việc với tệp

Kiểu mảng

- Khi làm việc với các cấu trúc dữ liệu dạng dãy hay danh sách các phần tử, ta sử dụng kiểu mảng (array)
  - Mảng 1 chiều: một vec-tơ các phần tử
  - Mảng nhiều chiều: một bảng các phần tử

- Mảng một chiều
  - Dãy các phần tử có cùng kiểu dữ liệu
  - Các phần tử được sắp xếp theo trật tự nhất định

Kiểu mảng

- Cú pháp khai báo mảng một chiều

  kiểu_dữ_liệu tên_mảng[số_phần_tử_của_mảng];
- Ví dụ
  - int ai[10];
  - float af[100];
- Số phần tử mảng được xác định khi khai báo
- Sử dụng toán tử [] để truy cập phân tử của mảng
  - Ví dụ: ai[2], af[10], …
- Chỉ số các phần tử mảng được **đánh số từ 0**

Kiểu mảng

- Ví dụ
  - Nhập danh sách các giá trị nguyên vào một mảng, sau đó tìm phần tử có giá trị nhỏ nhất trong mảng

```
#include <stdio.h>
#define N 10
main()
{
   int x[N], min;
    int i;
    for (i=0; i <= N-1; i++){
            printf(" x[%d]= ", i);
            scanf("%d", &x[i]);
    }
   min = x[0];
    for (i=1; i < N; i++)
            if (min > x[i]) min = x[i];
   printf("\n min= %d", min);
}
```

Kiểu mảng

- Khởi tạo mảng
  - Mảng có thể được khởi tạo giá trị ngay khi khai báo
  - Cú pháp

    kiểu_dữ_liệu tên_mảng[số_phần_tử_của_mảng] = {danh_sách_các_giá_trị_khởi_tạo};
  - Khi khai báo mảng có khởi tạo giá trị thì có thể không cần chỉ ra số phần tử mảng
  - Ví dụ

    int ai[3] = {2, 4, 5};
    - Hoặc

      int ai[] = {2, 4, 5}; /*không khai báo số phần tử mảng*/

Kiểu mảng

- Định nghĩa kiểu mới – từ khóa **typedef**
  - Có thể sử dụng từ khóa **typedef** để định nghĩa các kiểu dữ liệu mới
  - Kiểu dữ liệu mới sẽ được sử dụng để khai báo dữ liệu
  - Ví dụ
    - typedef int kieunguyen;
    - typedef float mangthuc10[10];

    sử dụng
    - kieunguyen x, a[100];
    - mangthuc10 x, y;

Kiểu mảng

- Mảng và địa chỉ
  - Toán tử & dùng để lấy địa chỉ một biến
  - Toán tử & cũng được dùng để lấy địa chỉ của một phần tử mảng
  - Các phần tử trong mảng được bố trí các ô nhớ liên tiếp nhau trên bộ nhớ
    - Nếu biết được địa chỉ phần tử thú i sẽ xác định được địa chỉ phần tử thú i+1
    - Địa chỉ phần tử đầu tiên là địa chỉ của mảng
  - Tên mảng mang địa chỉ của mảng đó

Kiểu mảng

- Mảng và địa chỉ
  - Ví dụ

    float a[100];

    float *pa;

    - Các cách viết sau là tương đương:

      a ⇔ &a[0]

      a + i ⇔ &a[i]

      *(a + i) ⇔ a[i]

    - Các phép gán hợp lệ

      pa = a;

      pa = &a[0];

Kiểu mảng

- Mảng là tham số của hàm
  - Khi sử dụng mảng là tham số của hàm, ta có thể khai báo, chẳng hạn:

    int a[]
    - Hoặc

      int *a
  - Như thế, hai cách sau là tương đương:

    f(int a[]) { … }

    f(int *a) { … }
  - Khi sử dụng, có thể gọi:

    f(a);

    Hoặc

    f(&a[0]);

Kiểu mảng

- Mảng là tham số của hàm
  - Ví dụ

```
void nhap_mang(int *x, int n)
{
        int i;
        /* Đọc các giá trị mảng */
        for (i=0; i <= n-1; i++)
        {
                printf(" x[%d]= ", i);
                scanf("%d", &x[i]);
        }
}
```

Kiểu mảng

- Mảng là tham số của hàm
  - Ví dụ

```
void xuat_mang(int *x, int n)
{
        int i;
        /* In các giá trị mảng */
        for (i=0; i <= n-1; i++)
                printf(" x[%d]= %d\n", i, x[i]);
}
```

Kiểu mảng

- Sắp xếp mảng
  - Sắp xếp các phần tử của mảng sao cho giá trị chúng theo thứ tự tăng dần hay giảm dần
  - Vấn đề thường gặp trong tin lập trình
  - Có nhiều cách sắp xếp khác nhau
    - Sắp xếp lựa chọn
    - Sắp xếp nổi bọt
    - Sắp xếp nhanh
    - Sắp xếp vun đống
    - …
  - Giả sử các phần tử của mảng có kiểu nguyên hoặc thực

Kiểu mảng

- Sắp xếp lựa chọn
  - Lấy phần tử đầu so sánh với các phần tử còn lại, nếu nó lớn hơn (nhỏ hơn) thì đổi chỗ giá trị của phần tử đầu tiên với phần tử đang so sánh. Kết quả sau lượt đầu, phần tử đầu tiên sẽ giữ giá trị nhỏ nhất.
  - Tiếp tục lượt hai, lấy phần tử thứ hai so sánh với các phần tử tiếp theo, nếu nó lớn hơn thì đổi chỗ giá trị của phần tử thứ hai với phần tử đang so sánh.
  - Việc này được tiến hành cho đến khi ta gặp phần tử cuối cùng.

Kiểu mảng

- Sắp xếp lựa chọn

```
#define N 50

…

int x[N];
int i, j, tam;

/* Đọc các giá trị mảng */

/* Sắp xếp mảng theo chiều tăng dần */
for (i=0; i < N-1; i++)
  for (j=i+1; j < N; j++)
  {
          if (x[i] > x[j])
          {
                  tam=x[i];/* Hoán đổi giá trị 2 biến */
                  x[i]=x[j];
                  x[j]=tam;
          }
  }
```

Kiểu mảng

- Sắp xếp lựa chọn
  - Cải tiến: ở một lượt i nào đó, thay vì đổi chỗ liên tục phần tử thứ i với phần tử có giá trị nhỏ hơn, thì ta chỉ thực hiện việc đổi chỗ phần tử nhỏ nhất ở lượt i với phần tử thứ i.

```
/* Sắp xếp mảng theo chiều tăng dần */
for (i=0; i < kich_thuoc - 1; i++)
{
   m = i;
            for (j = i+1; j < kich_thuoc; j++)
            {
                        if (x[m] > x[j]) m = j;
            }
            if (m != i)
            {
                        tam=x[m];
                        x[m]=x[i];
                        x[i]=tam;
            }
}
```

Kiểu mảng

- Sắp xếp nổi bọt
  - Duyệt các phần tử của mảng từ cuối mảng lên đến đầu mảng
  - Gặp hai phần tử kế cận ngược thứ tự thì đổi chỗ cho nhau
  - Như thế, lượt đầu sẽ chuyển phần tử nhỏ nhất lên đầu mảng phần tử
  - Tiếp tục, lượt thứ hai phần tử nhỏ thứ hai sẽ được chuyển đến vị trí thứ hai
  - …
  - Hình dung mảng được xếp thẳng đứng thì sau từng lượt các phần tử nhỏ dần sẽ được nổi lên như “bọt nổi lên trong nồi nước đang sôi”

Kiểu mảng

- Sắp xếp nổi bọt

```
/* Sắp xếp nổi bọt */
for (i = 0; i < kich_thuoc - 1; i++)
{
for (j = kich_thuoc - 1; j > i + 1; j--)
{
        if (x[j] < x[j-1])
        {
                tam=x[j];
                x[j]=x[j-1];
                x[j-1]=tam;
        }
}
}
```

Kiểu mảng

- Sắp xếp nhanh (quicksort)
  - Chọn một phần tử làm “chốt”
  - So sánh các phần tử còn lại với chốt và thực hiện hoán đổi sao cho các phần tử nhỏ hơn chốt được xếp trước chốt, các phần tử nhỏ hơn chốt được xếp sau chốt
  - Sau bước này mảng gồm
    - Phân đoạn các phần tử nhỏ hơn chốt
    - Chốt (cũng là vị trí thực của chốt sau khi mảng đã sắp xếp)
    - Phân đoạn các phần tử lớn hơn chốt
  - Thực hiện lại các bước trên cho hai phân đoạn trước và sau chốt, cho đến khi phân đoạn chỉ gồm một phần tử thì dừng lại

Kiểu mảng

- Sắp xếp nhanh (quicksort)

```
void quicksort(int a[], int l, int r)
{
        int i, j, chot;
        if (l < r){
                i = l+1;
                j = r;
                chot = a[l];
                while (i < j){
                        while(a[i] < chot) i++;
                        while(a[j] > chot) j--;
                        if (i < j) hoanvi(&a[i], &a[j]);
                }
                hoanvi(&a[j], &a[l]);
                quicksort(a, l, j-1);
                quicksort(a, j+1, r);
        }
}
```

Kiểu mảng

- Tìm kiếm phần tử trong mảng
  - Tìm sự xuất hiện của một phần tử trong mảng
  - Hai phương pháp cơ bản
    - Tìm kiếm tuần tự
    - Tìm kiếm nhị phân

Kiểu mảng

- Tìm kiếm tuần tự
  - Duyệt từ đầu mảng đến cuối mảng để tìm sự xuất hiện của một phần tử

```
int timkiem_tuantu(int a[], int n, int x)
{
        int i;
        i = 0;
        while ((i < n) && (a[i] != x)) i++;
        /* nếu i == n thì không tìm thấy x */
        return(i);
}
```

Kiểu mảng

- Tìm kiếm nhị phân
  - Áp dụng đối với mảng đã được sắp xếp
  - Ý tưởng
    - Giả sử mảng đã được sắp xếp tăng dần
    - Lấy phần tử cần tìm so sánh với phần tử giữa mảng (gọi là *g*), có các khả năng xảy ra:
      - Nếu phần tử cần tìm lớn hơn *g*, thì chỉ tìm nữa cuối mảng
      - Nếu phần tử cần tìm nhỏ hơn *g*, thì chỉ tìm nữa đầu mảng
      - Nếu không, *g* chính là phần tử cần tìm

Kiểu mảng

- Tìm kiếm nhị phân

```
int timkiem_nhiphan(int a[], int n, int x)
{
        int t, p, g;
        t = 0;
        p = n-1;
        while (t <= p)
        {
                g = (t + p)/2;
                if (x < a[g]) p = g - 1;
                else if (x > a[g]) t = g + 1;
                        else return(g);
        }
        return(n); /* trường hợp không tìm thấy x */
}
```

Kiểu mảng

- Mảng nhiều chiều
  - Ví dụ, khai báo mảng hai chiều

    int a[4][10];

    là mảng có 4 hàng, 10 cột
  - Truy cập các phần tử của mảng

    a[0][0], a[0][1], a[i][j]…
  - Ví dụ khác

    float arr[3][4][5];

    char arrc[4][4];

Kiểu mảng

- Mảng nhiều chiều
  - Ví dụ

```
/* Hàm nhập mảng các số nguyên */
void nhap_ma_tran(int a[][Max_Cot], int m, int n)
{
    int i, j;
    int x;
    for (i=0; i < m; i++){
        printf("\n Nhap hang thu %2d\n", i);
        for (j=0; j < n; ++j){
            printf("pt[%d][%d]", i, j);
            scanf("%d", &x);
            a[i][j] = x;
        }
  }
}
```

Kiểu mảng

- Mảng nhiều chiều
  - Ví dụ

```
/* Hiển thị các phần tử của mảng */
void in_ma_tran (int a[][Max_Cot], int m, int n);
{
  int i, j;
  for (i=0; i < m; i++){
            for (j=0; j < n; ++j)
                        printf("%4d", a[i][j]);
            printf("\n");
  }
}
```

Kiểu mảng

- Mảng nhiều chiều
  - Ví dụ

```
/* Tính tổng 2 mảng các số nguyên */
void tinh_tong(int a[][Max_Cot], int b[][Max_Cot],
               int c[][Max_Cot], int m, int n);
{
  int i, j;
  for (i=0; i < m; i++)
          for (j=0; j < n; ++j)
                  c[i][j]= a[i][j] + b[i][j];
}
```

  - Bài tập ?

Nội dung

- Giới thiệu chung
- Lệnh nhập/xuất
- Lệnh điều kiện
- Lệnh vòng lặp
- Hàm
- Kiểu mảng
- Xâu kí tự
- Kiểu cấu trúc (struct) và kiểu hợp (union)
- Làm việc với tệp

Xâu kí tự

- Xâu kí tự là một mảng các phần tử kiểu **char**, kết thúc bởi kí tự '\0' hay NULL
- Thao trên xâu kí tương tự nhue thao tác trên mảng một chiều
- Khai báo xâu kí tự

  char xau[100]; /* xâu chứa tối đa 100 */

  Hoặc

  char *xau; /* khai báo con trỏ đến xâu kí tự */
- Khởi gán
  - char xau[100] = {'a', 'b', 'c', '\0'};
  - char xau[100] = "abc";
  - char xau[] = "abc";
  - char *xau = "abc";

Xâu kí tự

- Ví dụ

```
/* Đếm độ dài xâu kí tự */
int strlen1(char *s)
{
        int n = 0;
        while (s[n] != '\0') n++;
        return n;
}
```

```
/* Đếm độ dài xâu kí tự */
int strlen2(char *s)
{
        int n;
        for (n = 0; *s != '\0'; s++) n++;
        return n;
}
```

Xâu kí tự

- Một số hàm thao tác trên xâu kí tự (được định nghĩa trong tệp **string.h**)
  - Xác định độ dài xâu

    *int strlen(char *str)*
  - Chép xâu kí tự

    *char * strcpy(char *str1, char *str2)*
    - Không sử dụng phép gán (=) để gán xâu kí tự cho con trỏ xâu kí tự
    - Phải sử dụng hàm *strcpy*
  - Hàm ghép hai xâu kí tự

    *char *strcat(char *dest, char *source)*

Xâu kí tự

- Một số hàm thao tác trên xâu kí tự (được định nghĩa trong tệp **string.h**)
  - So sánh hai xâu kí tự

    *int strcmp(char *str1, char *str2)*
    - Hàm trả về
      - 0 nếu str1 == str2
      - > 0 nếu str1 > str2
      - < 0 nếu str1 < str2
  - Tìm một xâu trong một xâu khác

    *char *strstr(char *str1, char *str2)*
    - Hàm trả về con trỏ trỏ đến vị trí xâu str2 trong str1 nếu tìm thây, nếu không sẽ trả về NULL

Xâu kí tự

- Một số hàm so sánh xâu khác
  - *int strcmpi(char *str1, char *str2)*: so sánh không phân biệt chữ hoa thường
  - *int stricmp(char *str1, char *str2)*: so sánh có phân biệt hoa thường
  - …

Xâu kí tự

- Một số hàm chuyển đổi giữa chuỗi và số
  - Các hàm được định nghĩa trong tệp **stdlib.h**
    - *double atof (const char *str)*: chuyển chuỗi *str* thành số thực kiểu *double*
    - *int atoi (const char *str)*: chuyển chuỗi *str* thành số nguyên kiểu *int*
    - *long int atol (const char *str)*: chuyển chuỗi *str* thành số nguyên kiểu *long*
    - Các hàm trên trả về 0, nếu việc chuyển không thành công

Xâu kí tự

- Một số hàm chuyển đổi giữa chuỗi và số
  - Hàm định nghĩa trong tệp **stdio.h**
    - int sprintf(char *s, const char *format, ...): có thể dùng để chuyển đổi số (nguyên hay thực) sang chuỗi chứa trong con trỏ *str*
    - Ví dụ
      - sprintf(str, “%d”, 100);
      - sprintf(str, “%f”, 12.25);

Xâu kí tự

- Mảng các xâu kí tự
  - Ví dụ
    - Khai báo

      char list_str[MAX][30];
    - Nhập

      for(i = 0; i < MAX; i++) gets(list_str[i]);
    - Xuất

      for(i = 0; i < MAX; i++)puts(list_str[i]);

Xâu kí tự

- Mảng các xâu kí tự
  - Sắp xếp

```
void sapxep(char list_str[][30], int row)
{
   int i, j;
   char tmp[30];
   for(i = 0; i < row-1; i++)
          for(j = i+1; j < row; j++)
                 if (stricmp(list_str[i], list_str[j])>0)
                 {
                        strcpy(tmp, list_str[i]) ;
                        strcpy(list_str[i], list_str[j]);
                        strcpy(list_str[j], tmp);
                 }
}
```

Kỹ thuật lập trình(7):
ngôn ngữ lập trình C

Tống Minh Đức
Khoa Công nghệ thông tin
Học viện Kỹ thuật Quân sự
100-Hoàng Quốc Việt – Hà Nội

Nội dung

- Giới thiệu chung
- Lệnh nhập/xuất
- Lệnh điều kiện
- Lệnh vòng lặp
- Hàm
- Kiểu mảng
- Xâu kí tự
- Kiểu cấu trúc (struct) và kiểu hợp (union)
- Làm việc với tệp

Kiểu cấu trúc

- Kiểu cấu trúc cho phép tạo ra kiểu dữ liệu mới gồm các phần tử dữ liệu có kiểu khác nhau nhưng liên kết với nhau
- Kiểu cấu trúc (**struct**ure) hay còn được gọi là kiểu bản ghi (record)
- Kiểu cấu trúc gồm nhiều phần tử dữ liệu khác nhau
- Các phần tử dữ liệu được gọi là các trường (field)
- Dùng từ khóa **struct** để định nghĩa kiểu cấu trúc

Kiểu cấu trúc

- Ví dụ: dùng kiểu cấu trúc mô tả dữ liệu là địa chỉ
  - Địa chỉ gồm các thông tin: số nhà, tên đường, tên thành phố

```
struct dia_chi
{
        int so_nha;
        char duong[40];
        char thanh_pho[30];
}ong_A, ba_B;
```

Kiểu cấu trúc

- Hoặc có thể khai báo các biến cấu trúc trực tiếp không cần khai báo tên cấu trúc

```
struct
{
        int so_nha;
        char duong[40];
        char thanh_pho[30];
}ong_A, ba_B;
```

Kiểu cấu trúc

- Hoặc chỉ khai báo kiểu cấu trúc

```
struct dia_chi
{
        int so_nha;
        char duong[40];
        char thanh_pho[30];
};
```

- Sau đó khai báo các biến
  - struct dia_chi ong_A, ba_B;

Kiểu cấu trúc

- Khai báo kiểu cấu trúc lồng nhau

```
typedef struct
{
        char ho_ten[40];
        struct dia_chi noi_o;
        char gioi_tinh;
} nhan_su;
```

- Khai báo biến
  - nhan_su p;

Kiểu cấu trúc

- Truy cập phần tử của cấu trúc
  - **tên_biến_cấu_trúc.tên_trường**
- Ví dụ
  - p.ho_ten
  - p.o_tai.so_nha
  - p.o_tai.duong
  - p.o_tai.thanh_pho
  - p.gioi_tinh

  - puts(p.ho_ten);

Kiểu cấu trúc

- Gán cấu trúc: 2 cách
  - Gán hai biến cấu trúc cho nhau
  - Gán các thành phần (trường) tương ứng của hai cấu trúc

  - Ví dụ

```
struct dia_chi d1, d2;

d1 = d2;
```

Hoặc

```
d1.so_nha = d2.so_nha;
strcpy(d1.duong, d2.duong);
strcpy(d1.thanh_pho, d2.thanh_pho);
```

Kiểu cấu trúc

- Mảng cấu trúc
  - Khai báo mảng gồm các phần tử có kiểu cấu trúc
  - Ví dụ

    ```
    nhan_su mang_nhan_su[100];
    ```

  - Sử dụng

    ```
    for (i = 0; i < 100; i++)
            puts(mang_nhan_su[i].ho_ten);
    ```

Kiểu cấu trúc

- Hàm có tham số kiểu cấu trúc
  - Ví dụ

```
void nhap(nhan_su *ns)
{
        nhan_su n;
        printf("Ho ten:");
        scanf("%s", n.hoten); /*gets(n.hoten)*/
        *ns = n;
}

void xuat(nhan_su ns)
{
        printf("%30s\n", ns.hoten);
}
```

Kiểu cấu trúc

- Hàm có tham số kiểu cấu trúc
  - Ví dụ

```
void hoanvi(nhan_su *p1, nhan_su *p2)
{
   nhan_su tmp;
   tmp = *p1; *p1 = *p2; *p2 = tmp;
}
void sapxep(nhan_su *p, int n)
{
   int i, j;
          for (i = 0; i < n-1; i++)
             for(j = i+1; j < n; j++)
                    if(strcmp(p[i].hoten, p[j].hoten) > 0)
                           hoanvi(&p[i], &p[j]);
}
```

Kiểu hợp

- Kiểu hợp (union) cho phép chia sẽ cùng một vùng bộ nhớ cho các biến khác nhau
- Nhằm tiết kiệm bộ nhớ
- Sử dụng từ khóa **union** để định nghĩa kiểu hợp
- Ví dụ

```
union union_type
{
    int i;
    char ch;
};
```

Kiểu hợp

- Máy dành 2 byte để lưu trữ khai báo trên
- Cả hai phần tử *i* và *ch* dùng chung vùng nhớ 2 byte
- Khai báo biến kiểu hợp

```
union union_type x;
```

- Tại mỗi thời điểm chỉ một trong hai biến *i* và *ch* được sử dụng
- Truy cập các phần tử kiểu hợp như kiểu cấu trúc

```
x.i
x.ch
```

Nội dung

- Giới thiệu chung
- Lệnh nhập/xuất
- Lệnh điều kiện
- Lệnh vòng lặp
- Hàm
- Kiểu mảng
- Xâu kí tự
- Kiểu cấu trúc (struct) và kiểu hợp (union)
- Làm việc với tệp

Làm việc với tệp

- Tìm hiểu các thao tác trên tệp
  - Mở tệp, Đóng tệp, Đọc, Ghi, …
- Ngôn ngữ C định nghĩa (trong tệp *stdio.h*)
  - cấu trúc kiểu tệp **FILE**
  - mã kết thúc tệp **EOF** (-1)
  - các hàm thao tác trên tệp

- Khai báo con trỏ tệp
  - `FILE *pf;`

Làm việc với tệp

- Cấu trúc chung của một tệp tin trên đĩa
  - Một tệp tin là một dãy các byte có giá trị từ 0 đến 255
  - Số byte là kích thước (size) của tệp
  - Khi đọc cuối tệp thì ta nhận được mã kết thúc tệp EOF
- Tệp tin chia làm hai loại
  - Tệp tin văn bản
  - Tệp tin nhị phân

Làm việc với tệp

- Dòng chảy (stream)
  - Trước khi một tệp tin được đọc hay ghi, một cấu trúc dữ liệu được gọi là *dòng chảy* phải được liên kết với nó
  - Một dòng chảy mà một con trỏ đến một cấu trúc
  - Có 3 dòng chảy được mở ra cho bất kỳ một chương trình C nào
    - **stdin** (standard input): được nối với bàn phím để đọc
    - **stdout** (standard output), **stderr** (standard error):được nối với màn hình để ghi

Làm việc với tệp

- Dòng chảy là gì ?
  - Dòng chảy tạo ra một vùng đệm (buffer) giữa chương trình đang chạy và tệp tin trên đĩa
  - Làm giảm việc chương trình truy cập trực tiếp thiết bị phần cứng (vd. đĩa)

Làm việc với tệp văn bản

- Mở tệp
  - Muốn thao tác trên tệp trước hết phải mở tệp
  - Mở tệp với hàm *fopen*
    - FILE *fopen(const char *name, const char *mode)
      - Hàm trả về con trỏ đến cấu trúc tệp hay dòng chảy tương ứng, nếu không thành công trả về NULL
      - *name* tên tệp tin cần mở
      - *mode* kiểu mở

        “w”: mở để ghi

        “r”: mở để đọc

        “a”: mở để ghi vào cuối tệp

Làm việc với tệp văn bản

- Mở tệp
  - Ví dụ

```
#include <stdio.h>
int main(void)
{
        FILE *in, *out, *append;

        in = fopen("dulieu.txt","r");
        out = fopen("dulieu.txt", "w");
        append = fopen("dulieu.txt", "a");
...
```

Làm việc với tệp văn bản

- Mở tệp

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

  if ((in = fopen("dulieu.txt","r")) == NULL)
     {
          fprintf(stderr,"Không thể mở tệp dulieu.txt\n");
          exit(1);
     }
...
```

Làm việc với tệp văn bản

- Đóng tệp
  - Phải đóng tệp khi không làm việc với nó nữa
  - Dùng hàm *fclose*
    - int fclose(FILE *fp)
      - fp là dòng chảy hay con trỏ tệp cần đóng
      - Hàm trả về 0 nếu thành công, ngược lại trả về EOF
    - Ví dụ
      - fclose(in);

Làm việc với tệp văn bản

- Báo lỗi hệ thống
  - Dùng hàm *perror*
    - void perror(const char *str)
      - trả về thông báo lỗi của hệ thống
  - Ví dụ

```
FILE *in;
if ((in = fopen("dulieu.txt","r")) == NULL)
  {
        fprintf(stderr,"Không thể mở tệp dulieu.txt\n");
        perror("lý do: ");
        exit(1);
  }
```

lý do: no such file or directory

Làm việc với tệp văn bản

- Đọc kí tự từ tệp
  - C cung cấp hai hàm *getc* và *fgetc*
    - int getc(FILE *fp)
    - int fgetc(FILE *fp)
      - Hai hàm có chức năng như nhau, đọc kí tự từ tệp tin ứng với dòng chảy *fp*, trả về mã ASCII kí tự được đọc nếu thành công, ngược lại trả về EOF

Làm việc với tệp văn bản

- Ví dụ: đọc nội dung tệp tin

```
#include <stdio.h>
main()
{ int c;
  FILE *fp;
  char name[80];
  printf("Nhập tên tệp cần đọc: ");
  scanf("%s", name);
  if ((fp = fopen(name,"r")) == NULL){
          fprintf(stderr,"Không thể mở tệp: %s\n", name);
          perror("Lý do:");
          exit(1);
  }
  while ((c = fgetc(fp)) != EOF)
          putchar(c);
  fclose(fp);
}
```

Làm việc với tệp văn bản

- Ghi kí tự vào tệp
  - C cung cấp hai hàm *putc* và *fputc*
    - int putc(int ch, FILE *fp)
    - int fputc(int ch, FILE *fp)
      - Hai hàm có chức năng như nhau, ghi kí tự có mã ASCII là *ch % 256* lên tệp tin ứng với dòng chảy *fp*, trả về mã ASCII kí tự được ghi nếu thành công, ngược lại trả về EOF

Làm việc với tệp văn bản

- Ví dụ: chép tệp tin

```
#include <stdio.h>
main()
{ int c;
  FILE *in, *out;
  char in_name[80], out_name[80];
  printf("Nhập tên tệp nguồn: "); scanf("%s", in_name);
  if ((in = fopen(in_name,"r")) == NULL){
            fprintf(stderr,"Không thể mở tệp: %s\n", in_name);
            perror("Lý do:"); exit(1);
  }
  printf("Nhập tên tệp đích: "); scanf("%s", out_name);
  if ((out = fopen(out_name, "w")) == NULL){
            fprintf(stderr,"Không thể mở tệp: %s\n", out_name);
            perror("Lý do:"); exit(1);
  }
  while ((c = fgetc(in)) != EOF) fputc(c, out);
  fclose(in); fclose(out);
}
```

Làm việc với tệp văn bản

- Đọc/Ghi chuỗi kí tự trên tệp
  - Đọc chuỗi kí tự *fgets*
    - char* fgets(char *s, int n, FILE *fp)
      - Hàm đọc từng chuỗi kí tự có độ dài lớn nhất là *n* trên tệp trỏ bởi *fp* vào chuỗi *s*
      - Hàm trả về con trỏ đến vùng nhớ chứa chuỗi kí tự được đọc nếu thành công, ngược lại trả về NULL
  - Ghi chuỗi kí tự *fputs*
    - int fputs(const char *s, FILE *fp)
      - Ghi chuỗi kí tự *s* lên tệp được trỏ bởi *fp*
      - Nếu thành công trả về mã kí tự cuối cùng được ghi, ngược lại trả về EOF

Làm việc với tệp văn bản

- Ví dụ: chép tệp tin

```
#include <stdio.h>
main()
{ int c;
  FILE *in, *out;
  char in_name[80], out_name[80], str[80];
  printf("Nhập tên tệp nguồn: "); scanf("%s", in_name);
  if ((in = fopen(in_name,"r")) == NULL){
            fprintf(stderr,"Không thể mở tệp: %s\n", in_name);
            perror("Lý do:"); exit(1);
  }
  printf("Nhập tên tệp đích: "); scanf("%s", out_name);
  if ((out = fopen(out_name, "w")) == NULL){
            fprintf(stderr,"Không thể mở tệp: %s\n", out_name);
            perror("Lý do:"); exit(1);
  }
  while (fgets(str, 80, in) != NULL) fputs(str, out);
  fclose(in); fclose(out);
}
```

Làm việc với tệp văn bản

- Ví dụ: chép tệp tin
  - Chúng ta muốn sử dụng:

    mycopy source dest↵
  - Sử dụng đọc tham số từ dòng lệnh
  - Dòng lệnh có thể được đọc bởi các tham số của hàm *main*, theo qui ước các tham số này được gọi "argc" và "argcv"

    int main(int argc, char *argv[])
  - Tham số "argc" chứa số từ trên dòng lệnh, kể cả tên chương trình
  - Tham số "argv" chứa danh sách con trỏ đến các từ trên dòng lệnh

Làm việc với tệp văn bản

- Ví dụ: chép tệp tin

```
#include <stdio.h>
int main(int argc, char *argv[])
{ int c; FILE *in, *out;
  if (argc != 3){
            fprintf(stderr,"Cú pháp: 'copy source dest'\n");
            return 1;
  }
  if ((in = fopen(argv[1],"r")) == NULL){
            fprintf(stderr,"Không thể mở tệp: %s\n", argv[1]);
            perror("Lý do:"); return 1;
  }
  if ((out = fopen(argv[2], "w")) == NULL){
            fprintf(stderr,"Không thể mở tệp: %s\n", argv[2]);
            perror("Lý do:"); return 1;
  }
  while ((c = fgetc(in)) != EOF) fputc(c, out);
  fclose(in); fclose(out);
  return 0;
}
```

Làm việc với tệp văn bản

- Đọc/Ghi dữ liệu trên tệp theo định dạng
  - Đọc dữ liệu theo định dạng fscanf

    int fscanf(FILE *fp, const char *chuỗi_điều_khiển, danh_sách_đối)
    - Đọc dữ liệu từ tệp trỏ bởi *fp* theo định dạng chuỗi điều khiển vào danh cách các đối, sử dụng tương tự hàm *scanf*
  - Ghi dữ liệu theo định dạng fprintf

    int fprintf(FILE *fp, const char *chuỗi_điều_khiển, danh_sách_đối)
    - Ghi dữ liệu vào tệp trỏ bởi *fp* theo định dạng chuỗi điều khiển và từ danh cách các đối, sử dụng tương tự hàm *printf*

Làm việc với tệp văn bản

- Ví dụ

```
…
FILE *in, *out;
…
int i, j, k;
float f;
…
fscanf(in, "%d|%d|%d|%f", &i, &j, &k, &f);
fprintf(out, "%d:%d:%d:%f", i, j, k, f);
…
```

Làm việc với tệp văn bản

- Ngoài các hàm được trình bày ở trên, C còn cung cấp nhiều hàm khác
  - Tự tìm hiểu
  - fcloseall, ferror, feof, unlink, remove, fseek, …

Làm việc với tệp nhị phân

- C còn cho phép thao tác trên các tệp nhị phân
  - Truy cập tệp một cách ngẫu nhiên dễ dàng
  - Dữ liệu có thể đọc ghi từng khối (blocs)
  - Tệp nhị phân và tệp văn bản có sự khác nhau khi xử lí mã chuyển dòng (newline) và mã kết thúc tệp (end of file)
  - Hầu hết các hàm dùng cho tệp văn bản đều được sử dụng cho tệp nhị phân, ngoại trừ các hàm *fgets*, *fputs*
  - Khi sử dụng hàm *fopen* sử dụng thêm tùy chọn “b” để mở tệp nhị phân
  - Ngoài ra, C cung cấp thêm một số hàm đọc ghi riêng cho tệp nhị phân

Làm việc với tệp nhị phân

- Ví dụ

```
#include <stdio.h>
main()
{ FILE *out, *in;
  int i = 11, j = 12; char ch = 'a'; char str[80] = "end.";
  if ((out = fopen("bifile.dat", "wb")) == NULL){
          fprintf(stderr,"impossible to open: bifile.dat\n");
          perror("Because:"); exit(1);
  }
  fputc(ch, out); fprintf(out, "\n%i:%i\n%s", i, j, str);
  fclose(out);
  if ((in = fopen("bifile.dat", "rb")) == NULL){
          fprintf(stderr,"impossible to open: bifile.dat\n");
          perror("Because:"); exit(1);
  }
  ch = fgetc(in); fscanf(in, "%i:%i%s", &i, &j, str);
  fprintf(stdout, "%c\n%i:%i\n%s\n", ch, i, j, str);
  fclose(in);
}
```

Làm việc với tệp nhị phân

- Vấn đề với mã kết thúc tệp
  - Mã kết thúc tệp đối với kiểu văn bản là 26 (Control-Z)
  - Khi đọc các kí tự của tệp trong kiểu văn bản, nếu gặp kí tự này thì giá trị EOF được trả về và kết thúc việc đọc
  - Kiểu nhị phân không không coi mã kết thúc tệp là 26
  - Để đọc tất cả các kí tự của tệp, nên đọc trong kiểu nhị phân

Làm việc với tệp nhị phân

- Ví dụ

```
#include <stdio.h>
main()
{ FILE *textfile, *binaryfile;
   if ((textfile = fopen("textfile.dat", "w")) == NULL){
            fprintf(stderr,"impossible to open: textfile.dat\n");
            perror("Because:"); exit(1);
  }
  if ((binaryfile = fopen(" binaryfile.dat", "wb")) == NULL){
            fprintf(stderr,"impossible to open: binaryfile.dat\n");
            perror("Because:"); exit(1);
  }
  fputc('A', textfile); fputc(26, textfile); fputc('B', textfile);
  fputc('A', binaryfile); fputc(26, binaryfile); fputc('B', binaryfile);
  fcloseall();
}
```

Điều gì xảy ra khi đọc các tệp trên trong kiểu văn bản ?

Làm việc với tệp nhị phân

- Vấn đề với mã chuyển dòng (newline)
  - Đối với kiểu văn bản
    - Khi ghi vào tệp mã chuyển dòng ‘\n’, thì hai kí tự được ghi vào tệp là ‘\r’ và ‘\n’ (kí tự ‘\r’ chuyển về cột đầu tiên và ‘\n’ chuyển sang dòng mới)
    - Khi đọc hai kí tự ‘\r’ và ‘\n’ thì được nhận biết là kí tự ‘\n’
  - Đối với kiểu nhị phân
    - Khi ghi vào tệp ‘\n’, thì chỉ kí tự ‘\n’ được ghi vào tệp

Làm việc với tệp nhị phân

- Ví dụ

```
…
FILE *bf, *tf;

…

tf = fopen(“txtfile”, “w”);
fprintf(“hi\n”);

…

bf = fopen(“binfile”, “wb”);
fprintf(“hi\n”);

…
```

4 kí tự được ghi vào tệp:
‘h’, ‘i’, ‘\r’, ‘\n’

3 kí tự được ghi vào tệp:
‘h’, ‘i’, ‘\n’

Làm việc với tệp nhị phân

- Các hàm chỉ đọc/ghi theo kiểu nhị phân
  - *int putw(int n, FILE *fp)* dùng để ghi một số nguyên (2 bytes) lên tệp
  - *int getw(FILE *fp)* dùng để đọc một số nguyên (2 bytes) từ tệp
  - *int fwrite(void *ptr, int size, int n, FILE *fp)* dùng để ghi *n* mẫu tin kích thước *size* từ vùng nhớ trỏ bởi *ptr* lên tệp *fp*, hàm trả về số mẫu tin thực sự ghi
  - *int fread(void *ptr, int size, int n, FILE *fp)* dùng để đọc *n* mẫu tin kích thước *size* từ tệp *fp* lên vùng nhớ trỏ bởi *ptr*, hàm trả về số mẫu tin thực sự được đọc
  - Các hàm *fread* và *fwrite* thường được dùng để đọc/ghi các mẫu tin là cấu trúc, số thực, …

Làm việc với tệp nhị phân

- Ví dụ

```
#include <stdio.h>
main()
{ FILE *out, *in;
  int i;
  if ((out = fopen("bifile.dat", "wb")) == NULL){
            fprintf(stderr,"impossible to open: bifile.dat\n");
            perror("Because:"); exit(1);
  }
  for(i = 0; i <10; i++) putw(i, out);
  fclose(out);
  if ((in = fopen("bifile.dat", "rb")) == NULL){
            fprintf(stderr,"impossible to open: bifile.dat\n");
            perror("Because:"); exit(1);
  }
  while((i = getw(in)) != EOF) printf("%d\n", i);
  fclose(in);
}
```

Làm việc với tệp nhị phân

- Ví dụ: sao chép tệp tin

```
#include <stdio.h>
#define N 1000
int main(int argc, char *argv[])
{ int n; FILE *in, *out; char str[N];
  if (argc != 3){
            fprintf(stderr,"Cú pháp: 'copy source dest'\n");
            return 1;
  }
  if ((in = fopen(argv[1],"rb")) == NULL){
            fprintf(stderr,"Không thể mở tệp: %s\n", argv[1]);
            perror("Lý do:"); return 1;
  }
  if ((out = fopen(argv[2], "wb")) == NULL){
            fprintf(stderr,"Không thể mở tệp: %s\n", argv[2]);
            perror("Lý do:"); return 1;
  }
  while ((n = fread(str, 1, N, in)) > 0) fwrite(str, 1, n, out);
  fclose(in); fclose(out);
  return 0;
}
```