UBND TỈNH **HẢI PHÒNG**
**TRƯỜNG CAO ĐẲNG CÔNG NGHIỆP HẢI PHÒNG**


# Giáo trình: Kỹ thuật vi xử lý

## Chuyên ngành: Kỹ thuật máy lạnh và điều hòa không khí

***(Lưu hành nội bộ)***

**HẢI PHÒNG**

# MỤC LỤC

# CHƯƠNG TRÌNH MÔ ĐUN

**Tên mô đun**: Kỹ thuật Vi xử lý

**Mã số mô đun** : MĐ 25

**Vị trí, ý nghĩa, vai trò mô đun**

* Vị trí của mô đun: Mô đun được bố trí dạy cuối chương trình sau khi học xong các mô đun cơ bản như Kỹ thuật Điện tử

* Tính chất của mô đun: Là mô đun bắt buộc.

**Mục tiêu của mô đun**:

Sau khi học xong mô đun này học viên có năng lực

- Vận hành được các thiết bị và dây chuyền sản xuất dùng vi xử lý
- Xác định được các nguyên nhân gây ra hư hỏng xảy ra trong thực tế.
- Kiểm tra và viết được các chương trình điều kiển.

* Về kiến thức:

- Trình bày được cấu trúc, ứng dụng cả vi xử lý trong công nghiệp
- Kiểm tra và viết được các chương trình điều kiển.

* Về kỹ năng:

- Vận hành được các thiết bị và dây chuyền sản xuất dùng vi xử lý
- Xác định được các nguyên nhân gây ra hư hỏng xảy ra trong thực tế.

* Về thái độ:

- Rèn luyện cho học sinh thái độ nghiêm túc, cẩn thận, chính xác trong học tập và thực hiện công việc

**Nội dung chính của mô đun**

Bài 1: Điều khiển I/O với LED đơn

Bài 2: Điều khiển với LED 7 đoạn

Bài 3: Điều khiển I/O sử dụng phím ấn

Bài 4: Điều khiển I/O với LCD

Bài 5: Điều khiển động cơ DC

# Bài 1: ĐIỀU KHIỂN I/O VỚI LED ĐƠN
# MÃ BÀI: MĐ25 – 01

## GIỚI THIỆU:

Lập trình I/O là lập trình đơn giản và cơ bản nhất, nhưng lại được sử dụng nhiều nhất, chúng ta điều khiển on/off bóng đèn, động cơ, hay 1 thiết bị nào đó cũng là 1 dạng của điều khiển I/O. Trong bài này gồm ngôn ngữ lập trình, phần mềm lập trình, các bài điều khiển I/O với LED đơn bằng ATMEGA

**Mục tiêu:**

Sau khi học xong bài học này người học có khả năng:

- Kiến thức: Trình bày được phương pháp kết nối điều khiển vào ra I/O với LED đơn
- Kỹ năng: Thực hiện sử dụng phần mềm và lập trình điều khiển khiển I/0 với LED đơn
- Thái độ: Nghiêm túc trong giờ học, có ý thức tự giác trong học tập.

**Nội dung chính:**

## 1. NGÔN NGỮ LẬP TRÌNH

### 1.1 Tập ký tự dùng trong ngôn ngữ C

Mọi ngôn ngữ lập trình đều được xây dựng từ một bộ ký tự nào đó. Các ký tự được nhóm lại theo nhiều cách khác nhau để tạo thành các từ, tiếp theo đó các từ lại được liên kết theo một quy tắc nào đó để tạo thành các câu lệnh. Một chương trình bao gồm nhiều câu lệnh biểu diễn một thuật toán để giải quyết một bài toán xác định. Ngôn ngữ C được xây dựng trên bộ ký tự sau:

- 26 chữ cái hoa: ABC..Z
- 26 chữ cái thường: abc..z
- 10 chữ số: 0..9
- Các ký hiệu toán học: + - * / = ()
- Ký tự gạch nối dưới : _
- Các ký hiệu đặc biệt khác: . , ; : [] {} ? ! \ & | % # $,...

Dấu cách (space) thực sự là một khoảng trống dùng để tách các từ. VD: HA NOI gồm 6 ký tự trong khi đó HANOI chỉ có 5 ký tự.

### 1.2 Từ khóa

Từ khóa là những từ có một ý nghĩa hoàn toàn xác định, thường được dùng để khai báo các kiểu dữ liệu, để viết các toán tử và các câu lệnh. Sau đây là các từ khóa của C:

| | | | |
|---|---|---|---|
| asm | break | case | cdecl |
| char | const | continue | default |
| do | double | else | enum |
| extern | far | float | for |
| goto | huge | if | int |
| interrupt | long | near | pascal |
| register | return | short | signed |
| sizeof | static | struct | switch |
| typedef | union | unsigned | void |

volatile while

Ý nghĩa và cách sử dụng chúng sẽ được bàn đến ở các phần sau, có hai điểm cần lưu ý, đó là:

- Không được dùng từ khóa để đặt tên cho các hằng, biến và hàm
- Từ khóa phải được viết bằng chữ thường

**1.3 Tên**

Tên là một khái niệm rất quan trọng, được dùng để xác định các đại lượng khác nhau trong một chương trình như tên hằng, tên biến, tên hàm…..

Tên được đặt theo cấu trúc như sau:

Tên là một dãy ký tự: Chữ, số và dấu gạch nối. Ký tự đầu của tên phải là chữ hoặc dấu gạch nối, độ dài mặc định của tên là 32, các ví dụ đúng về tên:

A_1 BETA x1 delta_7_x1

Các tên sau là sai

3XYZ_7 (ký tự đầu tiên là số)
r#3 (sử dụng ký tự #)
f(x) (sử dụng dấu ngoặc tròn)
case (trùng với từ khóa)
be ta (sử dụng khoảng trắng)
X-1 (sử dụng dấu gạch ngang)

Trong các tên có phân biệt chữ hoa và chữ thường, do đó tên AB khác tên ab. Trong C thường dùng chữ hoa để đặt tên cho các hằng và chữ thường cho hầu hết các đại lượng còn lại. Tuy nhiên, điều này không bắt buộc.

**1.4. Kiểu dữ liệu**

Trong C sử dụng các kiểu dữ liệu sau: Số nguyên (int), số thực hay số dấu phẩy động (float), số dấu phẩy động có độ chính xác kép (double) và ký tự (char). Hằng chính là một giá trị thông tin cụ thể, biến và mảng là các đại lượng mang tin. Mỗi loại biến có thể chứa một dạng thông tin nào đó VD biến kiểu int chứa các số nguyên, biến kiểu float chứa các số thực. Để có thể lưu trữ được thông tin biến phảI được cấp phát bộ nhớ, biến lại được chia thành biến tĩnh, biến tự động và biến ngoài.

Biến tự động chỉ tồn tại (được cấp phát bộ nhớ) khi chúng đang được sử dụng, biến ngoài và tĩnh tồn tại trong suốt thời gian hoạt động của chương trình. Cách tổ chức như vậy vừa tiết kiệm bộ nhớ vừa cho phép sử dụng cùng một tên cho các đối tượng khác nhau mà không gây ra nhầm lẫn.

*** Kiểu char**

Một giá trị kiểu char chiếm một byte bộ nhớ và biểu diễn được một ký tự thông qua bảng mã ASCII. Ví dụ:

| **Ký tự** | **Mã ASCII** |
|---|---|
| 0 | 048 |
| 1 | 049 |
| 2 | 050 |
| A | 065 |
| B | 066 |
| a | 097 |

b 098

Có hai kiểu char là: Signed char và unsigned char. Kiểu thứ nhất biểu diễn một số nguyên từ -128 đến +127, kiểu thứ hai có giá trị từ 0 đến 255

Có thể chia 256 ký tự thành 3 nhóm:

- Nhóm thứ nhất là các ký tự điều khiển có mã từ 0 đến 31, các ký tự trong nhóm này không hiển thị ra màn hình.
- Nhóm thứ hai là các ký tự văn bản có mã từ 32 đến 126, các ký tự này có thể đưa ra màn hình và máy in.
- Nhóm ba là các ký tự đồ họa có mã từ 127 đến 255 có thể đưa ra màn hình.

*** Kiểu nguyên**

Trong C cho phép sử dụng số nguyên (int), số nguyên dài (long) và số nguyên không dấu (únigned)

| *Kiểu* | *Phạm vi biểu diễn* | *Kích thước* |
|---|---|---|
| Int | -32768…+32767 | 2 byte |
| Unsigned int | 0...65535 | 2 byte |
| Long | -2147483648…2147483647 | 4 byte |
| Unsigned long | 0…4294967295 | 4 byte |

Các kiểu ký tự cũng có thể xem là một dạng của kiểu nguyên

*** Kiểu dấu phẩy động**

Trong C cho phép sử dụng ba loại giá trị dấu phẩy động đó là: Float, double và long double được trình bày như sau:

| Kiểu | Phạm vi biểu diễn | Kích thước |
|---|---|---|
| Float | 3.4E-38…3.4E+38 | 4 byte |
| Double | 1.7E-308...1.7E+308 | 8 byte |
| Long double | 3.4E-4932…1.1E+4932 | 10 byte |

## 1.5 BIẾN

Mọi biến cần phải được khai báo trước khi sử dụng, mẫu khai báo như sau

Type var_name; Ví dụ:

```
Main ( )
{
int     a, b, c; /* 3 biến kiểu integer */ char   d, e, f; /* 3 biến kiểu char */
float   x, y, z; /* 3 biến kiểu float */
……..
}
```

Biến kiểu int chỉ nhận các giá trị int, các biến kiểu khác cung có đặc tính tương tự, ví dụ biến kiểu char chỉ chứa một ký tự, để lưu trữ một xâu ký tự cần phải dùng một mảng kiểu char

Các khai báo biến phải đượ đặt ngay sau dấu { đầu tiên của thân hàm và phải đứng trước mọi câu lệnh khác.

Các biến có thể được khởi gán ngay khi khai báo bằng cách đặt thêm dấu = kèm theo một giá trị Ví dụ:

```
Int     a, b = 20, c, d = 40;
```

Kết quả này cũng tương đương bằng các câu lệnh gán sau:

Int a, b, c, d;
b = 20;
d = 40;

Mỗi biến được cung cấp một vùng nhớ gồm nhiều byte liên tiếp, địa chỉ byte đầu tiên chính là địa chỉ của biến, địa chỉ biến nhận đươc qua phép toán & var_name

## 1.6 Mảng

Mỗi biến chỉ có thể biểu diễn được một giá trị. Để biểu diễn một dãy số hoặc một bảng số có thể dùng nhiều biến nhưng cách này không thuận tiện khi số giá trị quá nhiều, cách tốt nhất là sử dụng mảng trong những trường hợp như thế này.

Mảng là một tập hợp gồm nhiều phần tử có cùng kiểu dữ liệu và có chung một tên, mỗi phần tử mảng biểu diễn được một giá trị. Có bao nhiêu kiểu dữ liệu thì cũng có bấy nhiêu kiểu mảng, mảng cần được khai báo về:

- Kiểu mảng (int, char,…)
- Tên mảng
- Số chiều và kích thước mỗi chiều

Kiểu mảng và tên mảng cũng giống như kiểu biến và tên biến. Ví dụ các khai báo sau đây:

Int a[10], b[4][2];
Float x[5], y[3][3];

Sẽ xác định 4 mảng: a, b, x và y như sau:

- Mảng thứ nhất có kiểu là int, tên là a, số chiều là 1, kích thước là 10. Mảng gồm 10 phần tử có số thứ tự là: a[0], a[1], a[2],…,a[9]. Mỗi một phần tử a[i] chứa một giá trị int và mảng này biểu diễn một dãy gồm 10 số nguyên.

- Mảng thứ hai có kiểu là int, tên là b, số chiều là 2, kích thước các chiều là 4 và 2. Mảng gồm 8 phần tử được đánh số và sắp xếp như sau:

b[0][0] b[0][1]
b[1][0] b[1][1]
b[2][0] b[2][1]
b[3][0] b[3][1]

Mỗi phần tử b[i][j] chứa được một giá trị kiểu int, và mảng này biểu diễn được một bảng số nguyên gồm 4 hàng và 2 cột

- Mảng thứ ba có kiểu là float, tên là x, số chiều là 1, kích thước là 5. Mảng gồm 5 phần tử được đánh số như sau: x[0], x[1], x[2], x[3], x[4]. Mỗi phần tử x[i] chứa được một giá trị kiểu float và mảng này biểu diễn được một dãy gồm 5 số thực

- Mảng thứ tư có kiểu là float, tên là y, số chiều là 2, kích thước các chiều là 3, mảng gồm 9 phần tử được đánh số và sắp xếp như sau:

y[0][0] y[0][1] y[0][2]
y[1][0] y[1][1] y[1][2]
y[2][0] y[2][1] y[2][2]

Mỗi phần tử y[i][j] chứa được một giá trị kiểu float và mảng y biểu diễn

một bảng số thực 3 hàng 3 cột

*Chú ý:*

- Các phần tử của mảng được cấp phát các khoảng nhớ liên tiếp nhau có nghĩa là địa chỉ của chúng cũng liên tiếp nhau
- Trong mảng hai chiều các phần tử của mảng được sắp xếp theo hàng.

Một phần tử của mảng được xác định nhờ chỉ số của nó, chỉ số của mảng phải có giá trị int và không vượt quá kích thước của chiều tương ứng. Số chỉ số phải bằng số chiều của mảng.

Giả sử a, b, x, y đã được khai báo như trên và giả sử I, j là 2 biến nguyên. Trong đó I = 2; j = 1 thì

| | |
|---|---|
| a[j+i-1] | là a[2] |
| b[j+i][2-i] | là b[3][0] |
| x[j/i] | là x[0] |
| y[i][j] | là [2][1] |

Và các cách viết sau đây là sai

y[j] vì y là mảng 2 chiều cần có 2 chỉ số

b[i][j][1] vì b cũng là mảng 2 chiều chỉ cần 2 chỉ số

Tuy nhiên, biểu thức dùng làm chỉ số có thể là số thực, khi đó phần nguyên sẽ là chỉ số mảng. VD: a[2.4] là a[2], a[1.9] là a[1]. Khi chỉ số vượt ra ngoài kích thước mảng máy vẫn không báo lỗi nhưng sẽ truy cập đến vùng nhớ ngoài mảng có dữ liệu không biết trước.

Có thể lấy địa chỉ của phần tử mảng một chiều và thường không lấy được địa chỉ của phần tử mảng nhiều chiều, phép tính sau lấy địa chỉ phần tử mảng 1 chiều

& a[i]

nhưng máy không chấp nhận phép tính

& y[i][j]

Một lưu ý quan trọng là tên mảng chính là địa chỉ đầu của mảng

a = & a[0]

**1.7 Toán tử**

Toán hạng (operand) có thể xem là một đại lượng có một giá trị nào đó ví dụ: Hằng, biến, mảng và hàm. Toán là là các phép toán tác động lên các toán hạng. Trong C toán hạng được chia thành các nhóm chính như sau:

- Toán tử số học (arithmatic operator)
- Toán tử quan hệ (relative operator)
- Toán tử logic (logical operator)
- Toán tử xử lý bít (bitwise operator)

Ngoài ra, C còn giới thiệu thêm 2 toán tử mối đó là:

- Toán tử tăng (incrementing operator)
- Toán tử giảm (decrementing operator)

*** Toán tử gán**

Theo C ký hiệu = không có nghĩa là bằng, ký hiệu này được gọi là toán tử gán (assignment operator), toán tử gán được gọi là gán đơn (single assignment) để phân biệt với gán phức hợp (compound assignment)

lulu = 2002 ;

Gán giá trị nguyên 2002 cho biến tên là lulu, toán tử gán cho phép gán một trị mới cho một biên. Dạng đơn giản nhất là:

biến = biểu thức ;

Nghĩa là biểu thức ở bên phải dấu = sẽ được định trị và kết quả định trị sẽ được gán cho biến ở phía trái dấu =. Nói cách khác ở bên trái dấu = phải là biến chứ không được là hằng còn ở bên phải là trị của biến, xem câu lệnh sau

I = I + 1 ;

Về mặt biểu diễn toán học thì không đúng, nhưng đối với máy tính thì hoàn toàn hợp lệ, giá trị của biến I được cộng với 1 rồi đem kết quả gán trở lại vào I, nhưng câu lệnh sau

2002 = lulu ;

Thì hoàn toàn sai vì 2002 là 1 hằng, không thể lấy giá trị biến lulu gán lên 1 hằng số

Ví dụ:

toc_do = khoang_cach / thoi_gian ;

gia_ban = gia_mua + chi_phi + lai ;

*** Toán tử cộng**

Toán tử cộng (+) sẽ cộng hai trị ở hai bên dấu + lại với nhau

thu_nhap = luong + tien_tham_nhung ;

Máy sẽ cộng giá trị hai bên dấu + và sau đó gán vào biến thu_nhap, toán tử + là loại toán tử 2 ngôi (binary operator) vì nó cần đến 2 toán hạng)

*** Toán tử trừ**

Toán tử trừ lấy giá trị trước dấu – trừ đi giá trị sau dấu trừ

tien_dem_ve = thu_nhap – tien_hoi_lo

Dấu trừ còn được dùng để thay dấu đại số của một giá trị.

Ví dụ:

toto = -12 ;

lili = - toto ;

Dấu trừ được dùng theo cách này được gọi là toán tử một ngôi vì nó chỉ cần một toán hạng

*** Toán tử nhân**

Toán tử nhân dùng dấu *, ví dụ:

luong_thang = luong_ngay * 30 ;

*** Toán tử chia**

C dùng ký hiệu / làm toán tử chia. Số bị chia là trị ở bên trái và số chia là trị ở bên phải

Ngay_trong_tuan = 30 / 4 ;

Cách chia số nguyên khác với cách chia số thực, chia theo số thực sẽ cho ra kết quả là một số thực. Khi chia hai số nguyên với nhau kết quả sẽ là phần nguyên còn phần thập phân sẽ bị cắt bỏ. Nếu trong phép chia có số nguyên và số thực lẫn lộn thì kết quả sẽ là số thực vì số nguyên sẽ được chuyển thành số thực trước khi chia.

*** Toán tử modulus**

Toán tử modulus ký hiệu là % sẽ cho ra số dư của phép chia, ví dụ:

13 % 5

Đọc là 13 modulus 5 sẽ cho ra kết quả là 3, đây chính là số dư của phép chia 13 cho 5. Toán tử này chỉ làm việc với số nguyên

*** Toán tử tăng / giảm**

Toán tử tăng (++) tăng giá trị của toán hạng lên 1 còn toán tử giảm (--) thì làm giảm giá trị của toán hạng xuống 1

toto++ ;

kiki -- ;

Tương đương với hai lệnh sau toto = toto + 1 ;

kiki = kiki – 1 ;

Các toán tử này có thể viết theo 2 cách prefix mode hoặc postfix mode có nghĩa là có thể viết phía trước hay sau toán hạng

++toto hoặc toto++

--kiki hoặc kiki—

Nếu viết ++ trước thì giá trị của toán hạng sẽ được tăng 1 trước (preincrement) sau đó kết quả này mới được dùng cho các thao tác khác, còn nếu viết sau thì giá trị của toán hạng sẽ được dùng trước rồi mới tăng 1 sau (postincrement), tương tự đối với toán tử giảm –

*** Toán tử gán phức hợp**

C còn cung cấp nhiều phép gán phức hợp để rút ngắn cách viết các biểu thức. Ví dụ:

```
ctr = ctr + 3 ;
ctr = ctr * 3 ;
ctr = ctr – 3 ;
ctr = ctr / 3 ;
ctr = ctr % 3 ;
```

Nhận xét là ở hai bên dấu gán đều có cùng một tên biến, để rút ngắn có thể viết như sau:

```
ctr += 3 ;
ctr *= 3 ;
ctr -= 3 ;
ctr /= 3 ;
```

## 1.8 Hàm trong C

Một chương trình C gồm một hoặc nhiều đơn thể chương trình được gọi là hàm (function) ghép lại, thường hàm trả về cho chương trình một trị sau khi được thực hiện.

Hàm cho phép chia một chương trình lớn thành những chương trình nhỏ nhằm dể quản lý và do đó làm đơn giản hóa công việc lập trình. Ngoài ra, một hàm hoạt động tốt ở chương trình này có thể được ghép vào một chương trình khác mà không cần phải viết lại từ đầu nên giúp giảm thời gian phát triển chương trình. Hàm được chia làm 2 loại:

- Hàm do người dùng viết gọi là user defined function (UDF). Đây là những hàm thực hiện các công việc cụ thể nào đó tại nhiều nơi trong chương trình

- Hàm thư viện chuẩn của C như hàm printf ( ). C cung cấp một số hàm thư viện chuẩn khá phong phú để tính toán, xử lý chuỗi, xử lý bít, xuất nhập dữ liệu…

*** Cấu trúc của hàm**

Đặc điểm chung của hàm:

1. Được đặt chung trong cùng một file chương trình nguồn hoặc ghép từ một file nguồn khác nhờ chỉ thị #include hoặc được biên dịch riêng rẽ sau đó liên kết lại để tạo thành file thực thi được.

2. Được gọi từ chương trình chính main ( ) hoặc từ một hàm khác hoặc từ chính nó (đệ quy)

3. Có hoặc không có đối muc (argument)

4. Có hoặc không có trị trả về ( return value)

5. Mỗi hàm chỉ có một điểm nhập chính là lệnh đầu tiên của hàm.

6. Một hàm có thể có nhiều điểm thoát thông qua lệnh return hoặc khi gặp lệnh cuối cùng của hàm

Một số lưu ý đối với hàm trong C:

1. Không cho phép các hàm lồng nhau có nghĩa là không được định nghĩa một hàm trong một hàm khác

2. Các đối mục thực sự của hàm chỉ có thể chuyển bằng trị có nghĩa là trị của đối mục được sao chép vào một biến tạm được gọi là đối mục hình thức và hàm thao tác trên biến tạm đó

Dạng tổng quát của một hàm như sau:

```
        Kiểu Tên_hàm (các đối mục)
{
thân hàm
}
```

• ***Kiểu***: Là kiểu của trị mà hàm sẽ trả về thông qua lệnh return. Trị mà hàm trả về phải có cùng kiểu với kiểu đặt trước tên hàm, nếu không xác định trước kiểu trả về thì máy sẽ mặc định là kiểu int.

• ***Tên_hàm***: Do người dùng tự đặt để nhận diện, không được dùng cùng một tên cho 2 hàm khác nhau

• ***Các đối mục***: Là danh sách các biến được phân cách bởi dấu phẩy. Nếu hàm không có đối mục thì danh sách đối mục là rỗng nhưng vẫn phải có 2 dấu móc đơn. VD hàm main ( )

• ***Thân hàm***: Là phần nằm trong cặp dấu ngoặc nhọn { } gồm một loạt các khai báo biến và các câu lệnh kết thúc bởi dấu chấm phẩy. Phần thân hàm có thể rỗng

Ví dụ hàm power ( ) được định nghĩa như sau:

```
Int  power (int x, n)
{
int  j, p ;
p = 1 ;
for (j = 1; j <= n ; j++)
p = p*x ;
```

return (p) ;
}

Dòng đầu tiên mô tả tên hàm (power) và danh sách các đối mục (x và n). Các đối mục này gọi là các đối mục hình thức (formal argument hoặc formal parameter). Đây là các biến trung gian tiếp nhận trị của các thông số cung cấp từ chương trình triệu gọi hàm.

VD trong main ( )

```
main ( )
{
power (2, i) ;
}
```

Khi gọi hàm power (2, i) để tính $2^i$ thì 2 và I gọi là các đối mục thực sự (actual argument). Như vậy khi gọi hàm power ( ) với các thông số 2 và I thì máy tính sẽ biến x thành 2 và n thành i. Quá trình sao chép các trị từ các đối mục thực sự của chương trình triệu gọi sang các đối mục hình thức của hàm được gọi là truyền đối mục (parameter passing).

Vì hàm có thể trả về một trị cho chương trình triệu gọi như là kết quả thực hiện của hàm. Do đó kiểu của trị trả về phải được chỉ rỏ ở trước tên hàm, nếu không chỉ rỏ thì kiểu trả về mặc nhiên xem như là int.

Các đối mục hình thức phải cùng kiểu với các đối mục thực sự khi gọi hàm. Các đối mục hình thức và các biến khai báo bên trong hàm được xem như là các biến cục bộ nghĩa là chúng chỉ tồn tại và có ý nghĩa bên trong thân hàm.

Câu lệnh return trả về trị do power ( ) tính được cho main ( ). Bất cứ trị nào của một biến, hằng hay một biểu thức đều có thể xuất hiện bên trong dấu ngoặc đơn của return với điều kiện là kiểu của đại lượng đó phải thích hợp với kiểu của trị trả về đã được khai báo trên dòng mô tả tên hàm.

Không nhất thiết hàm nào cũng phải trả về một trị, câu lệnh return không có đối mục trong dấu ngoặc sẽ không trả về trị nào cả mà chỉ đơn thuần là trả quyền điều khiển về cho chương trình triệu gọi để chương trình này tiếp tục thực hiện các công việc khác.

*** Chương trình gồm nhiều hàm**

Như đã nói ở trên chương trình C gồm một hoặc nhiều hàm, và đương nhiên một trong các hàm đó phải là hàm main ( ). Hàm cung cấp một phương tiện hữu hiệu để đóng gói một công việc cụ thể nào đó. Khi hàm được triệu gọi thì công việc này sẽ được thi hành. Nói chung, có thể xem hàm như một cái hộp đen mang một cái tên và có thể thực hiện một công việc cụ thể nào đó

Trong chương trình C chỉ có một hàm duy nhất tên là main ( ) và không có đối mục. Chương trình được bắt đầu thi hành từ hàm main ( ) trở đi, hàm có thể triệu gọi các hàm khác tạo thành lớp hàm phân thành đẳng cấp hàm nối kết nhau

*** Hàm có kiểu void**

Trong các phiên bản về sau, C có đưa ra một loại hàm đặc biệt không có trị trả về được gọi là void, void có nghĩa là không có hiệu lực. VD:

```
void  tên_hàm ( )
{
```

}

## 1.9 Cấu trúc điều khiển

### *Lệnh if

Cú pháp như sau:

```
If (biểu thức điều kiện)
  Lệnh 1 ;
elsse
  Lệnh 2 ;
```

Nếu (biểu thức điều kiện) cho ra một trị khác không (true) thì lệnh 1 sẽ được thực hiện và chương trình sẽ tiếp tục với lệnh theo sau lệnh if, ngược lại nếu (biểu thức điều kiện) cho ra trị là 0 (false) thì lệnh 2 được thực hiện.

Ví dụ: Tráo biến

```
if (a < b)
  printf (“nguyên thủy a nhỏ hơn b\n”) ;
else {
  tam = a ;
  a = b ;
  a = tam ;
  printf (“tráo a và b\n”) ;
```

Cấu trúc lồng nhau của lệnh if.

Ví dụ: chương trình phân loại điểm như sau

| Điểm | Loại |
|---|---|
| 80-100 | A |
| 70-79 | B |
| 60-69 | C |
| 50-59 | D |
| 40-49 | E |
| 0-39 | F |

```
if (điểm >= 80)
  loai = ‘A’ ;
else if (điểm >= 70)
  loai = ‘B’ ;
else if (điểm >= 60)
  loai = ‘C’ ;
else if (điểm >= 50)
  loai = ‘D’ ;
else if ( điểm >= 40)
  loại = ‘E’ ;
else
loại = ‘F’ ;
```

### * Lệnh switch

Như trong ví dụ trên cho thấy việc só sánh trị của một biến với các trị khác nhau được xảy ra nhiều lần, để thuận tiện C đưa ra một lệnh đặc biệt cho mục đích này đó là lệnh switch, switch có nghĩa là bật chuyển qua lại,

switch được hình dung như là một công tắc xoay nhiều vị trí. Cú pháp như sau:

```
switch (biểu thức)
{
case biểu_thức_hằng_1:
lệnh 1a ;
lệnh 1b ;
case biểu_thức_hằng_2:
lệnh 2a ;
lệnh 2b ;
……
…….
case biểu_thức_hằng_n:
lệnh na ;
lệnh nb ;
default:
lệnh da ;
lệnh db ;
}
```

Biểu thức điều kiện trong dấu ngoặc sẽ được định trị, kết quả định trị phải là kiểu nguyên kể cả kiểu char. Các biểu thức hằng ở từng trường hợp sau từ khóa case (thường được gọi là case labels) cũng phải là một số nguyên và không thể có hai số nguyên trùng nhau.

Khi thực hiện lệnh switch thì trước tiên biểu thức nằm sau từ switch được định trị. Trị kết quả sau đó được đem so sánh với từng case label, nếu bằng nhau thì quyền điều khiển sẽ được trao cho lệnh đầu tiên trong nhóm case label này và tất cả các lệnh kể từ đây cho đến cuối switch sẽ được thi hành. Ví dụ:

```
n = 2 ;
switch (n)
{
case 1: printf (“một\n”) ;
case 2: printf (“hai\n”) ;
case 3: printf (“ba\n”) ;
case 4: printf (“bốn\n”) ;
default: printf (“default\n”) ;
}
printf (“cuối switch\n”) ;
```

Biểu thức điều kiện ở đây là trị của n = 2. Kết quả chương trình như sau:

```
hai ba bốn
default
cuối switch
```

Lệnh trong case label có thể là một lệnh rỗng. Nếu không có biểu thức hằng của case label nào bằng với trị của biểu thức điều kiện thì các lệnh của

default sẽ được thi hành. Trong ví dụ trên nếu n = 7 thì kết quả sẽ như sau
default
cuối switch
Từ default và các lệnh liên quan là tùy chọn có nghĩa là không có cũng được. Nếu trị của biểu thức điều kiện không bằng với bất cúa biểu thức hằng nào và trong trường hợp này cũng không có default thì các lệnh trong switch sẽ không được thi hành.

*** Lệnh break**

Như đã trình bày ở trên khi quyền điều khiển trao về cho trường hợp case label bằng với trị của biểu thức điều khiển, thì việc thi hành sẽ bắt đầu từ đó cho đến cuối switch. Tuy nhiên, thông thường lệnh switch được sử dụng như là một phương thức lựa chọn nhiều chiều. Một khi quyền điều khiển được trao cho một case label thì chỉ những lệnh thuộc case label này được thi hành rồi nhảy về cuối switch bỏ qua những lệnh của các case label khác. Việc này được thực hiện bằng cách đặt tiếp theo sau lệnh cuối trong case label một lệnh break. Ví dụ:

```
n = 3 ;
switch (n)
{
case 1: printf (“một\n”) ; break ;
case 2: ;
case 3: printf (“hai hoặc ba\n”) ; break ;
case 4: printf (“bốn\n”) ; break ;
}
printf (“cuối switch\n”) ;
```

Sẽ cho kết quả như sau:

```
hai hoặc ba cuối switch
```

Thực ra trong case 4 không cần lệnh break

*** Lệnh Lệnh For :**

Cú pháp :

```
for ( bt1; bt2 ; bt3)
        { lenh 1;
          lenh 2;
             …..
          lenhn;
        }
```

- Giải thích :

+ bt1: là toán tử gán để tạo giá trị ban đầu cho biến điều khiển.
+ bt2: biểu thức điều kiện để thực hiện vòng lặp.
+ bt3: biểu thức tăng giá trị của biến điều khiển của vòng lặp.

* Ví dụ:

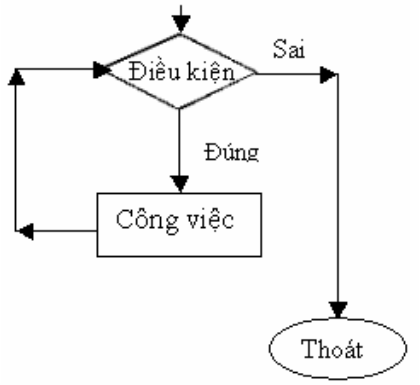
```
void tre(x)
{
int i,j;
```

```
for(i=0;i<x;i++)
for(j=0;j<1000;j++)
      {      }
}
```

*** Câu lệnh while**

Câu lệnh vòng lặp  while: Cú pháp:

```
while( Điều kiện)
{   //Các câu lệnh   }
```




*Giải thích:

Thực hiện lặp các câu lệnh khi điều kiện đúng, nếu điều kiện sai thi thoát khỏi vòng lặp

Đặc biệt:

Tạo vòng lặp mãi mãi, rất hay đùng trong lập trình VXL. Chương trình chính sẽ được viết trong dấu ngoặc.

```
while(1) {     }
```

### 1.10 Khai báo thư viện

Là nơi chứa thông tin về cấu hình cứng của loại vi điều khiển

Ví dụ:  #include <mega128.h>

Thư viện có thể thực hiện một số chức năng cụ thể và có thể sử dụng như một chương trình con

Ví dụ:  #include <delay.h>

### 1.11 Khai báo cổng vào ra

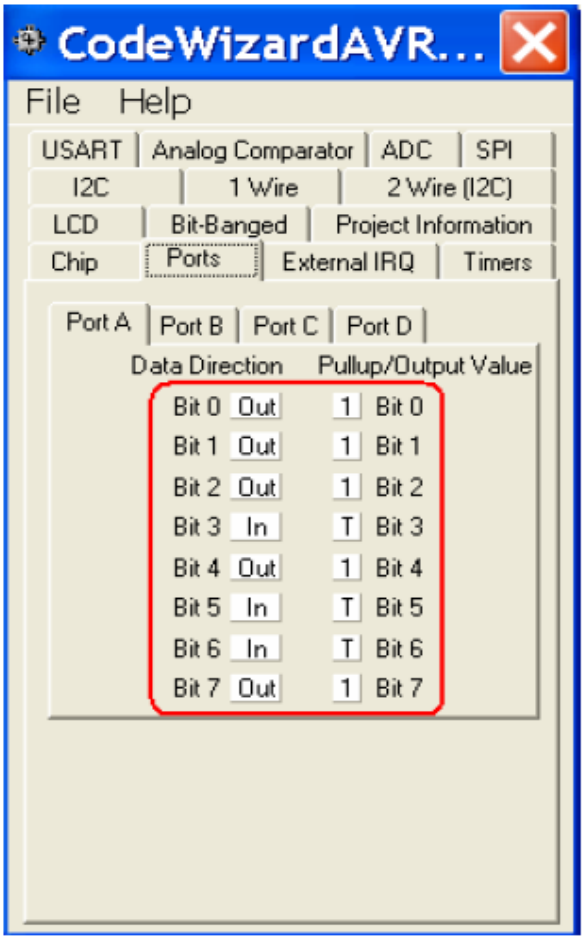
Thiết lập cổng vào ra:

Khi xem xét đến các cổng I/O của AVR thì ta phải xét tới 3 thanh ghi bit ***DDxn,PORTxn,PINxn*.**

-Các bit *DDxn* để truy cập cho địa chỉ xuất nhập *DDRx*. Bit *DDxn* trong thanh ghi *DDRx* dùng để điều khiển hướng dữ liệu của các chân của cổng này.Khi ghi giá trị logic ‘0’ vào bất kì bit nào của thanh ghi này thì nó sẽ trở thành lối vào,còn ghi ‘1’ vào bit đó thì nó trở thành lối ra.

-Các bit *PORTxn* để truy cập tại địa chỉ xuất nhập *PORTx*. Khi *PORTx* được ghi giá trị 1 khi các chân có cấu tạo như cổng ra thì điện trở kéo là chủ động(được nối với cổng).Ngắt điện trở kéo ra, *PORTx* được ghi giá trị 0 hoặc các chân có dạng như cổng ra.Các chân của cổng là 3 trạng thái khi 1 điều kiện reset là tích cực thậm chí xung đồng hồ không hoạt

động.
-Các bit *PINxn* để truy cập tại địa chỉ xuất nhập *PINx*. *PINx* là các cổng chỉ để đọc,các cổng này có thể đọc trạng thái logic của *PORTx*.*PINx* không phải là thanh ghi,việc đọc *PINx* cho phép ta đọc giá trị logic trên các chân của *PORTx*.chú ý *PINx* không phải là thanh ghi,việc đọc *PINx* cho phép ta đọc giá trị logic trên các chân của *PORTx*.
Nếu *PORTxn* được ghi giá trị logic '1' khi các chân của cổng có dạng như chân ra ,các chân có giá trị '1'.Nếu *PORTxn* ghi giá trị '0' khi các chân của cổng có dạng như chân ra thì các chân đó có giá trị '0'.
Các cổng của AVR đều có thể đọc,ghi. Để thiết lập 1 cổng là cổng vào ,ra thì ta tác động tới các bit *DDxn*, *PORTxn*,*PINxn*.ta có thể thiết lập để từng bit làm cổng vào,ra cứ không chỉ với cổng,như vậy ta có thể sử lí tới từng bit,đây chính là điểm mạnh của các dòng Vi điều khiển 8 bit.
- Các bit DDxn để truy cập cho địa chỉ xuất nhập DDRx. Bit DDxn trong thanh ghi DDRx dùng để điều khiển hướng dữ liệu của các chân của cổng này.Khi ghi giá trị logic '0' vào bất kì bit nào của thanh ghi này thì nó sẽ trở thành lối vào,còn ghi '1' vào bit đó thì nó trở thành lối ra.
- Các bit PORTxn để truy cập tại địa chỉ xuất nhập PORTx. Khi PORTx được ghi giá trị 1 khi các chân có cấu tạo như cổng ra thì điện trở kéo là chủ động (được nối với cổng). Ngắt điện trở kéo ra, PORTx được ghi giá trị 0 hoặc các chân có dạng như cổng ra.Các chân của cổng là 3 trạng thái khi 1 điều kiện reset là tích cực thậm chí xung đồng hồ không hoạt động.



Ví dụ như trên hình:các bit 0,1,2,4,7 của PORTA làm chân ra có trở kéo,còn các bit còn lại làm chân vào. Khi đã thiết lập xong thì các bit 0,1,2,4,7 sẽ có thể xuất dữ liệu ra còn các bit còn lại có thể nhận dữ liệu vào.

**Ví dụ** :

Ta muốn ghi dữ liệu giá trị logic ’0’ ra PORTA.0 để bật tắt một Led thì:

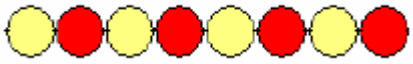
PORTA.0=1;

Ta muốn đọc dữ liệu là một bit từ chân 3 của PORTA:

Bit x;

x=PINA.3;

Cũng như vậy khi ta thiết lập PORTA làm cổng ra thì ta có thể xuất dữ liệu ra từ PORTA:

PORTA=0xAA;  **PORTA**

Còn nếu ta thiết lập PORTA làm cổng vào và giá trị hiện thời của PORTA**:**

 **PORTA**

Thì sau câu lệnh đọc giá trị từ PORTA: x=PORTA thì x=0x55. Khi thiết lập PORTA làm cổng ra thì khi reset giá trị của PORTA là PORTA=0xFF;

 **PORTA**

Khi thiết lập PORTA làm cổng vào thì khi reset giá trị của PORTA là PORTA=0x00; **PORTA**

Việc thiết lập cổng vào ra là một việc quan trọng vì tùy theo mục đích sử dụng các cổng nào làm cổng vào ra,thì ta phải thiết lập đúng thì mới có thể sử dụng được, động tác này khác với họ vi điều khiển 8051- AT8951.

## 2. SƠ ĐỒ ĐIỀU KHIỂN I/O VỚI LED ĐƠN