Ngôn Ngữ Lập Trình C Cơ Bản Cho 8051

1. Giới thiệu ngôn ngữ C

Trong kỹ thuật lập trình vi điều khiển nói chung, ngôn ngữ lập trình được sử dụng thường chia làm 2 loại: Ngôn ngữ bậc thấp và Ngôn ngữ bậc cao.

Ngôn ngữ bậc cao là các ngôn ngữ gần với ngôn ngữ con người hơn, do đó việc lập trình bằng các ngôn ngữ này trở nên dễ dàng và đơn giản hơn. Có thể kể đến một số ngôn ngữ lập trình bậc cao như C, Basic, Pascal… trong đó C là ngôn ngữ thông dụng hơn cả trong kỹ thuật vi điều khiển. Về bản chất, sử dụng các ngôn ngữ này thay cho ngôn ngữ bậc thấp là giảm tải cho lập trình viên trong việc nghiên cứu các tập lệnh và xây dựng các cấu trúc giải thuật. Chương trình viết bằng ngôn ngữ bậc cao cũng sẽ được một phần mềm trên máy tính gọi là trình biên dịch (Compiler) chuyển sang dạng hợp ngữ trước khi chuyển sang mã máy.

Khi sử dụng ngôn ngữ C người lập trình không cần hiểu sâu sắc về cấu trúc của bộ vi điều khiển. Có nghĩa là với một người chưa quen với một vi điều khiển cho trước sẽ xây dựng được chương trình một cách nhanh chóng hơn, do không phải mất thời gian tìm hiểu kiến trúc của vi điều khiển đó. Và việc sử dụng lại các chương trình đã xây dựng trước đó cũng dễ dàng hơn, có thể sử dụng toàn bộ hoặc sửa chữa một phần.

2. Ngôn ngữ C

2.1 Kiểu dữ liệu

2.1.1 Kiểu dữ liệu trong C

Kiểu	Số Byte	Khoảng giá trị
Char	1	-128 – +127
Unsigned char	1	0 – 255
Int	2	-32768 - +32767
Unsigned int	2	0 - 65535
Long	4	-2147483648 - +2147483647
Unsigned long	4	0 – 4294967295
Float	4	

*** Khai báo biến:**

- Cú pháp: Kiểu_dữ_liệu Vùng_nhớ Tên_biến _at_ Đia_chỉ;
Ví dụ: Unsigned char data x;
- Khi khai báo biến có thể gán luôn cho biến giá trị ban đầu.

Ví dụ: Thay vì: unsigned char x;
x = 0;
Ta chỉ cần: unsigned char x = 0;
- Có thể khai báo nhiều biến cùng một kiểu một lúc.

Ví dụ: Unsigned int x,y,z,t;

- Chỉ định vùng nhớ: từ khoá "Vùng_nhớ" cho phép người dùng có thể chỉ ra vùng nhớ sử dụng để lưu trữ các biến sử dụng trong chương trình. Các vùng nhớ có thể sử dụng là: CODE, DATA, DATAB, IDATA, PDATA, XDTA. Khi không khai báo vùng nhớ trình dịch Keil C sẽ mặc định đó là vùng nhớ DATA.

Vùng nhớ	Ý nghĩa
CODE	Bộ nhớ mã nguồn chương trình

DATA	Bộ nhớ dữ liệu gồm 128 Byte thấp của RAM trong vi điều khiển
BDATA	Bộ nhớ dữ liệu có thê định địa chỉ bit, nằm trong vùng nhớ DATA
IDATA	Bộ nhớ dữ liệu gồm 128 Byte cao của RAM trong vi điều khiển chỉ có ở một số dòng vi điều khiển sau này
PDATA	Bố nhớ dữ liệu ngoài gồm 256 Byte, được truy cập bởi địa chỉ đặt trên P0
XDATA	Bộ nhớ dữ liệu ngoài có dung lượng có thể lên đến 64 KB, được truy cập bởi địa chỉ đặt trên P0 và P2

*** Định nghĩa lại kiểu**

- Cú pháp: Typedef Kiểu_dữ_liệu Tên_biến;
- Tên_biến sau này sẽ được sử dụng như một kiểu dữ liệu mới và có thể dùng để khai báo các biến khác.

Ví dụ: Typedef int m5[5];
Dùng tên m5 khai báo hai biến tên a và b có kiểu dữ liệu là mảng 1 chiều 5 phần tử: m5 a,b;

2.1.2 Kiểu dữ liệu trong Keil C

Kiểu	Số bit
Bit	1
Sbit	1
Sfr	8
Sfr16	16

- bit : dùng để khai báo các biến có giá trị 0 hoặc một hay các biến logic trên vùng RAM của vi điều khiển. Khi khai báo biến kiểu bit trình dịc Keil C sẽ mặc định vùng nhớ sử dụng là BDATA.
- sbit, sfr, sfr16: dùng để định nghĩa các cho các thanh ghi chức năng hoặc các cổng trên vi điều khiển dùng để truy nhập các đoạn dữ liệu 1 bit, 8 bit, 16 bit.

2.1.3 Mảng

Mảng là một tập hợp nhiều phần tử cùng một kiểu giá trị và chung một tên. Các phần tử của mảng phân biệt với nhau bởi chỉ số hay số thứ tự của phần tử trong dãy phần tử. Mỗi phần tử có vai trò như một biến và lưu trữ được một giá trị độc lập với các phần tử khác của mảng.
Mảng có thể là mảng một chiều hoặc mảng nhiều chiều.

Khai báo:

- Cú pháp: Tên_kiểu Vùng_nhớ Tên_mảng[số_phần_tử_mảng];
Khi bỏ trống số phần tử mảng ta sẽ có mảng có số phần tử bất kì.

Ví dụ: Unsigned int data a[5],b[2] [3];
Với khai báo trên ta sẽ có: mảng a là mảng một chiều 5 phần tử. Mảng b là mảng hai chiều, tổng số phần tử là 6.
Chỉ số của mảng bắt đầu từ số 0. Mảng có bao nhiêu chiều phải cung cấp đầy đủ bấy nhiêu chỉ số.

Ví du: Phần tử mảng 2 chiều:
b[0] [1] là đúng
Khi viết: b[0] là sai

2.1.4. Con trỏ

Khi ta khai báo một biến, biến đó sẽ được cấp phát một khoảng nhớ bao gồm một số byte nhất định dùng để lưu trữ giá trị. Địa chỉ đầu tiên của khoảng nhớ đó chính là địa chỉ của biến

được khai báo.

Con trỏ là một biến dùng để chứa địa chỉ mà không chứa giá trị, hay giá trị của con trỏ chính là địa chỉ khoảng nhớ mà nó trỏ tới.

Với các vùng nhớ cụ thể con trỏ tới vùng nhớ đó chiếm dung lượng phụ thuộc vào độ lớn của vùng nhớ đó. Con trỏ tổng quát khi không xác định trước vùng nhớ sẽ có dung lượng lớn nhất vì vậy tốt nhất nên sử dụng con trỏ cụ thể.

Loại con trỏ	Kích thước
Con trỏ tổng quát	3 byte
Con trỏ XDATA	2 byte
Con trỏ CODE	2 byte
Con trỏ DATA	1 byte
Con trỏ IDATA	1 byte
Con trỏ PDATA	1 byte

Khai báo biến con trỏ:

- Cú pháp: Kiểu_Dữ_liệu Vùng_nhớ *Tên_biến;

- Ví dụ:
```
int *int_ptr;
long data *long_ptr;
```

- khi không chỉ rõ vùng nhớ con trỏ sẽ được coi là con trỏ tổng quát.

2.1.5 Kiểu dữ liệu cấu trúc

Kiểu dữ liệu cấu trúc là một tập hợp các biến, các mảng và cả các kiểu cấu trúc khác được biểu thị bởi một tên duy nhất. kiểu dữ liệu cấu trúc dùng để lưu trữ các giá trị, thông tin có liên quan đến nhau.

Định nghĩa và khai báo biến cấu trúc:

- Định nghĩa:
```
Typedef struct {
                Khai báo các biến thành phần;
              } Tên_kiểu_cấu_trúc;
```
- Khai báo: Tên_kiểu_cấu_trúc Vùng_nhớ Tên_biến;

Ví dụ:
```
Typedef struct {
                char day;
               char month;
                int year;
              } Date_type;
Date_type date,date_arr[5];
```

2.2 Phép toán

Phép gán kí hiệu: "=".

- Cú pháp: Biến_1 = Biến_2;

Trong đó Biến_2 có thể là giá trị xác định cũng có thể là biến.

2.2.1 Phép toán số học

Phép toán	Ý nghĩa	Ví dụ
+	Phép cộng	**X = a+b;**
-	Phép trừ	**X = a-b;**
*****	Phép nhân	**X = a*b;**
/	Phép chia lấy phần nguyên	**X = a/b;** (a=9, b=2 → X=4)
%	Phép chia lấy phần dư	**X = a%b;** (a=9, b=2 → X=1)

2.2.2 Phép toán Logic

Chức năng	Phép toán
AND	&&
OR	||
NOT	!

2.2.3 Các phép toán so sánh:

Phép toán	ý nghĩa	Ví dụ
>	So sánh lớn hơn	a>b 4>5 sẽ trả ra giá trị 0
>=	So sánh lớn hơn hoặc bằng	a>=b 6>=2 sẽ trả ra giá trị 1
<	So sánh nhỏ hơn	a<b 6<7 sẽ trả ra giá trị 1
<=	So sánh nhỏ hơn hoặc bằng	a<=b 8<=5 sẽ trả ra giá trị 0
==	So sánh bằng nhau	a==b 6==6 sẽ trả ra giá trị 1
!=	So sánh khác nhau	a!=b 9!=9 sẽ trả ra giá trị 0

2.2.4 Phép toán thao tác Bit

Phép toán	Ý nghĩa	Ví dụ
&	Phép và (AND)	Bit_1 & Bit_2
|	Phép hoặc (OR)	Bit_1 | Bit_2
!	Phép đảo (NOT)	!Bit_1
^	Phép hoặc loại trừ (XOR)	Bit_1 ^ Bit_2
<<	Dịch trái	a<<3
>>	Dịch phải	a>>4
~	Lấy bù theo bit	~a

2.2.5 Phép toán kết hợp

Phép toán	Ví dụ
+=	a+=5 <=> a=a+5
-=	a-=5 <=> a=a-5
*=	a*=5 <=> a=a*5
/=	a/=5 <=> a=a/5
%=	a%=5 <=> a=a%5

2.3 Cấu trúc chương trình C

2.3.1 Cấu trúc chương trình

*** Cấu trúc:**

1. Khai báo chỉ thị tiền xử lý
2. Khai báo các biến toàn cục
3. Khai báo nguyên mẫu các hàm
4. Xây dựng các hàm và chương trình chính

*** Ví dụ:**

```
// Khai báo chỉ thị tiền xử lý:
        #include<regx51.h>
         #include<string.h>
         #define Led1 P1_0
//*********************************
// Khai báo biến toàn cục:
         Unsigned char code Led_arr[3];
         Unsigned char data dem;
         Unsigned int xdata X;
//*********************************
// Khai báo nguyên mẫu hàm
         Void delay(unsigned int n);
         bit kiemtra(unsigned int a);
//*********************************
// Xây dựng các hàm và chương trình chính:
          void delay(unsigned int n)
         {
                 Khai báo biến cục bộ;
              Mã chương trình trễ;
         }
        Void main()  // Chương trình chính
         {
                 Khai báo biến cụ bộ;
              Mã chương trình chính;
         }
         Bit kiemtra(unsigned int a)
         {
                 Khai báo biến cục bô;
              Mã chương trình kiểm tra biến a;
         }
```

Chú ý: Hàm không khai báo nguyên mẫu phải được xây dựng trước hàm có lời gọi hàm đó. Ở ví dụ trên do hàm "bit kiemtra(unsigned int a)" đã được khai báo nguyên mẫu hàm ở trên nên có thể xây dựng hàm ở bất kì vị trí nào trong chương trình.

Tuy nhiên chúng ta nên khai báo nguyên mẫu hàm trước hàm main, và xây dựng các hàm phụ ở sau hàm main. Như thế sẽ tạo thói quen lập trình gọn gàng hơn, và cũng tạo thuận lợi hơn cho việc xem lại code, tìm kiếm và sửa lỗi sau này.

2.3.2 Chỉ thị tiền xử lý

Các chỉ thị tiền sử lý không phải là các lệnh của ngôn ngữ C mà là các lệnh giúp cho việc soạn thảo chương trình nguồn C trước khi biên dịch. Khi dịch một chương trình C thì không phải chính bản chương trình nguồn mà ta soạn thảo được dịch. Trước khi dịch, các lệnh tiền xử lý sẽ chỉnh lý bản gốc, sau đó bản chỉnh lý này sẽ được dịch. Có ba cách chỉnh lý được dùng là:

+ Phép thay thế **#Define**
+ Phép chèn tệp **#Include**
+ Phép lựa chọn biên dịch **#Ifdef**

Các chỉ thị tiền xử lý giúp ta viết chương trình ngắn gọn hơn và tổ chức biên dịch, gỡ rối chương trình linh hoạt, hiệu quả hơn.

*** Chỉ thị #Define:** Chỉ thị #define cho phép tạo các macro thay thế đơn giản.

- Cú pháp: #Define Tên_thay_thế dãy_kí_tự

Một Tên_thay_thế có thể được định nghĩa lại nhiều lần, nhưng trước khi định nghĩa lại phải giải phóng định nghĩa bằng chỉ thị:

#Undef Tên_thay_thế

- Ví dụ: #define N 100

*** Chỉ thị #Include:** Chỉ thị #include báo cho trình biên dịch nhận nội dung của tệp khác và chèn vào tệp chương trình nguồn mà ta soạn thảo.

- Cú pháp:

Cách 1: **#include<tên_tệp>**
Cách 2: **#include"tên_tệp"**

- Ví dụ:

Cách 1: #include<regx51.h>
Ở cách này tệp regx51.h sẽ được tìm trong thư mục INC để chèn vào chương trình nguồn.

Cách 2: #include"regx51.h"
Ở cách này tệp regx51.h sẽ được tìm trong thư mục chứa chương trình nguồn nếu không có mới tìm trong thư mục INC.

Khi muốn chèn tệp ngoài thư viện hoặc ngoài thư mục chứa chương trình nguồn thìtên_tệp sẽ bao gồm cả đường dẫn thư mục chứa tệp.

*** Chỉ thị #Ifdef:** Chỉ thị #ifdef này thường dùng để biên dịch các tệp thư viện.

- Cú pháp:

```
Cách 1:   #Ifdef ten_macro
                //Đoạn chương trình
          #endif
Cách 2:   #ifdef ten_macro
                //Đoạn chương trình 1
          #else
                //Đoạn chương trình 2
          #endif
```

Ở cách 1: nếu tên_macro đã được định nghĩa thì "Đoạn chương trình" sẽ được dịch, ngược lại thì "Đoạn chương trình" sẽ bị bỏ qua.

*** Chỉ thị #Ifndef:** Chỉ thị #ifndef này thường dùng để biên dịch các tệp thư viện.

- Cú pháp:

```
Cách 1:   #ifndef ten_macro
              //Đoạn chương trình
          #endif
Cách 2:   #ifndef ten_macro
                //Đoạn chương trình 1
          #else
                //Đoạn chương trình 2
          #endif
```

Ở cách 1: nếu tên_macro chưa được định nghĩa thì "Đoạn chương trình" sẽ được dịch, ngược lại thì "Đoạn chương trình" sẽ bị bỏ qua.

2.3.3 Chú thích trong chương trình

Việc viết chú thích trong trình nhằm mục đích giải thích ý nghĩa của câu lệnh, đoạn chương trình hoặc hàm hoạt động như thế nào và làm gì. Viết chú thích sẽ giúp cho người đọc có thể hiểu được chương trình dễ dàng và nhanh chóng hơn, sửa lỗi đơn giản hơn hoặc giúp cho ta xem lại chương trình cũ mà ta đã làm trở lên nhanh hơn.

Chú thích trong chương trình sẽ không ảnh hưởng đến chương trình mà ta soạn thảo vì trình dịch sẽ bỏ qua tất cả lời chú thích khi biên dịch chương trình sang mã máy.

Lời giải thích được đặt sau dấu "//" nếu chú thích chỉ viết trên một dòng hoặc trong cặp dấu "*" và "*\".

3. Các lệnh cơ bản trong C

+ Câu lệnh rẽ nhánh - if:

- Cấu trúc 1:

```
if(dieu_kien)
{
        // Đoạn chương trình
}
```

Giải thích: nếu dieu_kien đúng thì xử lí các câu lệnh bên trong còn sai thì nhảy qua.

- Cấu trúc 2:

```
if(dieu_kien)
{
// Đoạn chương trình 1
}
else
{
  // Đoạn chương trình 2
}
```

Giải thích: nếu dieu_kien đúng thì xử lí "Đoạn chương trình 1" bên trong còn sai thì xử lý "Đoạn chương trình 2".

+ Câu lệnh lựa chọn - Switch:

Cấu trúc:

```
switch(biến)
{
        case gia_tri_1:
          //các câu lệnh
          break;
  case gia_tri_2:
          //các câu lệnh
          break;
  ……..
  case gia_tri_n:
          //các câu lệnh
          break;
    Default:
            //các câu lệnh
}
```

Giải thích: Tuỳ vào biến có giá trị bằng giá trị của Case nào thì thực hiện các câu lệnh tương ứng trong Case đó, sau đó thoát khỏi cấu trúc nhờ câu lệnh "break;". Nếu không có Case nào phù hợp thì thực hiện các câu lệnh trong default.

+ Vòng lặp xác định - For:

Cấu trúc:

```
for( x=n ; điều_kiện ; phép_toán )
{
        // các câu lệnh xử lí
}
```

Giải thích: x là biến, n là giá trị xác định. Trước tiên vòng lặp sẽ gán giá trị ban đầu cho biến: x=n, rồi kiểm tra nếu điều_kiện đúng thì thực hiện các câu lệnh xử lý, sau đó thực hiện Phép_toán nhằm tác động đến điều kiện. Sau đó lại kiểm tra lại điều_kiện, nếu còn đúng thì thực hiện tiếp, nếu sai sẽ thoát khỏi vòng lặp.

Các thành phần trong vòng for có thể không cần khai báo,for sẽ bỏ qua phần đó, nhưng vẫn phải có đủ 2 dấu ";".

+ Vòng lặp không xác định - while:

Cấu trúc:

```
while(dieu_kien)
{
        // các câu lệnh
}
```

Giả thích: Trước tiên chương trình sẽ kiểm tra điều_kiện, nếu đúng thì thực hiện các câu lệnh, sau đó quay lại kiểm tra điều_kiện. Còn nếu điều_kiện sai thì thoát khỏi vòng lặp ngay.

+ Vòng lặp không xác định - do while:

Cấu trúc:

```
do
 {
        // các câu lệnh
 } while(dieu_kien);
```

Giả thích: Trước tiên đoạn chương trình thực hiện các câu lệnh sau đó kiểm tradiều_kiện nếu đúng thì lặp lại thực hiện các câu lệnh tiếp, nếu sai thì thoát khỏi vòng lặp.

Để hiểu rõ hơn, và để **làm chủ ngôn ngữ C**, các bạn hãy tham khảo loạt bài **Tự học Lập trình C** trong mục **Lập Trình.**

Comments

You do not have permission to add comments.