Phụ lục 1

Ngôn ngữ C trong lập trình VĐK

Ngôn ngữ lập trình

- Ngôn ngữ lập trình gồm:
  - Ngôn ngữ bậc thấp (Mã máy và Assembly)
  - Ngôn ngữ bậc cao (Java, C/C++, Pascal, Basic…).

Ngôn ngữ C trong lập trình VĐK

- Để lập trình vi điều khiển thì sẽ có 2 ngôn ngữ lập trình là:
  - Ngôn ngữ C
  - Ngôn ngữ Assembly
- Ngôn ngữ C là ngôn ngữ thông dụng hơn trong lập trình vi điều khiển

Cấu trúc chương trình C

- 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

```
#include<pic.h>
#include<string.h>
#define Led1 P1_0
```

Khai báo chỉ thị tiền xử lý

```
Unsigned char code Led_arr[3];
Unsigned char data dem;
Unsigned int xdata X;
```

Khai báo biến toàn cục

```
Void delay(unsigned int n);
bit kiemtra(unsigned int a);
```

Khai báo nguyên mẫu hàm

Ví dụ

```
void delay(unsigned int n)
{
        Khai báo biến cục bộ;
        Mã chương trình trễ;
}
Void main()
{
        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;
}
```

Xây dựng các hàm và chương trình chính

Chú thích trong chương trình

- Mục đích: Giải thích ý nghĩa của câu lệnh, đoạn chương trình hoặc hàm hoạt động như thế nào và làm gì.
- Khai báo: Đặt sau dấu “//” nếu chú thích chỉ viết trên một dòng
- Hoặc trong cặp dấu “/*” và “*/” nếu chú thích nhiều dòng.

Kiểu dữ liệu

- Một số kiểu dữ liệu thông dụng

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 – 4294697295
Float	4	

Khai báo biến

- Cú pháp:

  Kiểu_dữ_liệu Tên_biến = Giá_trị_ban_đầu(Nếu có);

- VD: Unsigned int min = 1;
- Khi khai báo nhiều biến cùng một lúc:

  Kiểu_dữ_liệu Tên_biến_1, Tên_biến_2, Tên_biến_3...

- Định nghĩa lại kiểu:
- Cú pháp: typedef Kiểu_dữ_liệu Tên_biến;
- Tên_biến sau này sẽ được dụng như một kiểu dữ liệu mới và có thể được dùng để khai báo các biến khác.
- VD: typedef int m5[5]; m5 a,b;

//Dùng m5 để khai báo hai biến tên a và b có kiểu dữ liệu mảng 1 chiều 5 phần tử.

Một số kiểu dữ liệu thường dùng trong lập trình VĐK

Type	Definition	Description
int8_t	singed char	8 bits signed integer
int16_t	signed short	16 bits signed integer
int32_t	signed int	32 bits signed integer
uint8_t	unsigned char	8 bits unsigned integer
uint16_t	unsigned short	16 bits unsigned integer
uint32_t	unsigned int	32 bits unsigned integer

Mảng

- Mảng có thể là mảng một chiều hoặc mảng nhiều chiều
- Khai báo:

- Cú pháp: Tên_kiểu 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 a[5],b[2] [3];
- 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 b[0] [1] là đúng Khi viết b[0] là sai

Chuỗi ký tự trong C

- Chuỗi ký tự: Là một mảng một chiều gồm các phần tử có kiểu char như ký tự, con số và những ký tự đặc biệt như +, -, x, /, %, \, # ....
- Chuỗi được kết thúc bởi 1 ký tự null (‘\0’ ký tự rỗng)
- Khai báo: char Text[15]= “ Hello World”;
- Chuỗi ký tự Text gồm 15 phần tử, trong đó phần tử cuối là ký tự null.

Thao tác với chuỗi

- int sprintf(char *str, const char *format, int/float/double Value) ; //Hàm biến đổi từ biến kiểu số thành kiểu chuỗi
- strcpy(s1, s2); //Sao chép chuỗi s2 cho chuỗi s1.
- strcat(s1, s2); //Nối chuỗi s2 vào cuối chuỗi s1.
- strlen(s1); //Trả về độ dài của chuỗi s1.
- strcmp(s1, s2); //Trả về 0 nếu s1 và s2 là như nhau; nhỏ hơn 0 nếu s1<s2; lớn hơn 0 nếu s1>s2.
- strchr(s1, ch); //Trả về con trỏ tới vị trí đầu tiên của ch trong s1.
- strstr(s1, s2); //Trả về con trỏ tới vị trí đầu tiên của chuỗi s2 trong chuỗi s1.

Con trỏ

- Con trỏ là một biến dùng để lưu trữ địa chỉ của biến mà không chứa giá trị, hay giá trị của con trỏ chính là địa chỉ khoảng nhớ mà nó trỏ tới.
- Khai báo: Kiểu_dữ_liệu *Tên_biến

VD int *int_st;

- Để lấy địa chỉ 1 biến sau đó gán địa chỉ đó cho biến con trỏ trên

```
int max=10;
int_st= &max; //Tên_biến_con_trỏ= &tên_biến_bất_kỳ
```

- Để lấy nội dung của biến mà con trỏ int_st đang trỏ đến

```
int min;
min= *int_st; //Biến_Bất_kỳ= *Tên_biến_con_trỏ
//min= 10 do int_st đang trỏ đến biến max
```

Dữ liệu kiểu cấu trúc trong C

- Kiểu Struct dùng để lưu nhiều thông tin có nhiều kiểu dữ liệu khác nhau.
- Cú pháp:

Cách 1.

```
struct StructName{
        kieu_bien1 TenBien1;
        kieu_bien2 TenBien2;
        ....
}
```

Cách 2. Sử dụng từ khóa typedef để định nghĩa kiểu

```
typedef struct{
        kieu_bien1 TenBien1;
        kieu_bien2 TenBien2;
        ....
} StructName;
```

Dữ liệu kiểu cấu trúc trong C

- Biến kiểu cấu trúc
- Với cách khai báo 1

```
struct Tên_cấu_trúc biến_1 biến_2 ... ;
```

- Với cách khai báo 2

```
Tên_cấu_trúc biến_1 biến_2 ...;
```

- VD

```
typedef struct
{
unsigned char Ngay;
unsigned char Thang;
unsigned int Nam;
} NgayThang;
typedef struct
{
    char MSSV[10];
    char HoTen[40];
    NgayThang NgaySinh;
    int Phai;
    char DiaChi[40];
} SinhVien;
Int main(){
    SinhVien SV, s;
    printf("Nhap MSSV: ");gets(SV.MSSV);
    printf("Nhap Ho va ten: ");gets(SV.HoTen);
    printf("Sinh ngay: ");
    scanf("%d",&SV.NgaySinh.Ngay);
}
```

Phép toán trong C

- Phép gán: “ = ”

Cú pháp: Biến_1= Biến_2; //Biến 2 có thể là giá trị xác định hoặc cũng có thể là biến.

- **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	a%b (a=9, b=2 → X=1)

Phép toán trong C

- **Phép toán Logic:**

  AND: && OR: | | NOT: !!

- **Phép toán logic theo các 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

Phép toán trong C

Thao tác với Bit trong vi điều khiển							
STT	Phép toán	Ký hiệu	Thanh ghi R1	Thanh ghi R2	Biểu thức	Kết quả trả về	Lưu ý
1	AND	&	0B0000 0001	0B0000 0011	R= R1&R2	0x0000 0001	
2	OR	|	0B0000 0001	0B0000 0011	R= R1|R2	0x0000 0011	
3	NOT	!	0B0000 0001	0B0000 0011	R= !R1	0	Kết quả trả về 0 hoặc 1
4	XOR	^	0B0000 0001	0B0000 0011	R= R1^R2	0x0000 0010	
5	Dịch trái	<<	0B0000 0001	0B0000 0011	R= R2<<2	0x0000 1100	
6	Dịch phải	>>	0B0000 0001	0B0000 0011	R= R2>>1	0x0000 0001	
7	Lấy bù	~	0B0000 0001	0B0000 0011	R= R1	0x1111 1110	
8	Set bit ở vị trí n	|(1<<n)	0B0000 0001	0B0000 0011	R1= R1|(1<<1) R1|= (1<<1)	0x0000 0011	
9	Xóa bit ở vị trí n	&~(1<<n)	0B0000 0001	0B0000 0011	R2= R2&~(1<<1) R2&= ~(1<<1)	0x0000 0001	

Phép toán trong C

- Thao tác khác.

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

Phép toán kết hợp trong C

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

STATEMENT	EXAMPLE

if (expr) **stmt; [**else **stmt;]**	`if (x==25)`
	`  x=1;`
	`else`
	`  x=x+1;`

while **(expr) stmt;**	`while (get_rtcc()!=0)` `  putc('n');`

do **stmt** while **(expr);**	`do {`
	`  putc(c=getc());`
	`} while (c!=0);`

for **(expr1;expr2;expr3) stmt;**	`for (i=1;i<=10;++i)` `  printf("%u\r\n",i);`

switch (expr) {
case **cexpr: stmt; //one or more case**
[default:**stmt]**
... }
	`switch (cmd) {`
	`  case 0:  printf("cmd 0");`
	`  break;`
	`  case 1: printf("cmd 1");`
	`  break;`
	`  default: printf("bad cmd");`
	`  break; }`

return **[expr];**	`return (5);`
goto **label;**	`goto loop;`
label: stmt;	`loop: I++;`
break;	`break;`
continue;	`continue;`
expr;	`i=1;`
;	`;`
{[stmt**]}** **Zero or more**	`{a=1;` `  b=1;}`

Các lệnh cơ bản trong C

Câu lệnh rẽ nhánh if:

- *Cấu trúc:* *if(dieu_kien)* {

```
// Đoạn chương trình
}
```

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

- Cấu trúc:

```
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 x ử l ý "Đoạn chương trình 1"

Các lệnh cơ bản trong C

Câu lệnh lựa chọn:

Cấu trúc:

```
switch(bien)                    {
        case gia_tri_1: {//các câu lệnh  break;}
        case gia_tri_2: {//các câu lệnh  break;}
        case gia_tri_3: {//các câu lệnh  break;}
        ……..
        case gia_tri_n: {//các câu lệnh  break;}

        default: {//lenh mac dinh  break;}
       }
```

Giải thích: tuỳ vào biến có gia_tri_1 thì thực hiện các câu lệnh tương ứng rồi sau đó thoát khỏi cấu trúc nhờ câu lệnh break.

Biến có gia_tri_2 thì thực hiện câu lệnh tương ứng rồi thoát.

…….

Biến có gia_tri_n thì thực hiện các câu lệnh tương ứng rồi thoát.

Các lệnh cơ bản trong C

Vòng lặp xác định:

Cấu trúc: for(i=m;i<k;i++) {

// các câu lệnh xử lí

}

Giải thích:

m,k là giá trị(m>k), còn n là biến.

thực hiện lặp các câu lệnh (k-m) lần.

+ *Vòng lặp không xác định while:*

Cấu trúc: while(dieu_kien) {

// các câu lệnh

}

Giả thích: thực hiện lặp các câu lệnh khi điều kiện đúng nếu điều kiện sai thì thoát khỏi vòng lặp.

Các lệnh cơ bản trong C

+ *Vòng lặp không xác định do while:*

Cấu trúc: do

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

Giải thích: thực hiện lặp các câu lệnh sau đó kiểm tra điều kiện nếu đúng, nếu sai thì thoát khỏi vòng lặp.

Arithmetic Operators

Operator	Description	Example
+ (Addition)	Adds values on either side of the operator.	A + B will give 30
- (Subtraction)	Subtracts right-hand operand from left-hand operand.	A - B will give -10
* (Multiplication)	Multiplies values on either side of the operator.	A * B will give 200
/ (Division)	Divides left-hand operand by right-hand operand.	B / A will give 2
% (Modulus)	Divides left-hand operand by right-hand operand and returns remainder.	B % A will give 0
++ (Increment)	Increases the value of operand by 1.	B++ gives 21
-- (Decrement)	Decreases the value of operand by 1.	B-- gives 19

Relational Operators

Operator	Description	Example
== (equal to)	Checks if the values of two operands are equal or not, if yes then condition becomes true.	(A == B) is not true.
!= (not equal to)	Checks if the values of two operands are equal or not, if values are not equal then condition becomes true.	(A != B) is true.
> (greater than)	Checks if the value of left operand is greater than the value of right operand, if yes then condition becomes true.	(A > B) is not true.
< (less than)	Checks if the value of left operand is less than the value of right operand, if yes then condition becomes true.	(A < B) is true.
>= (greater than or equal to)	Checks if the value of left operand is greater than or equal to the value of right operand, if yes then condition becomes true.	(A >= B) is not true.
<= (less than or equal to)	Checks if the value of left operand is less than or equal to the value of right operand, if yes then condition becomes true.	(A <= B) is true.

Bitwise Operators

Operator	Description	Example
& (bitwise and)	Binary AND Operator copies a bit to the result if it exists in both operands.	(A & B) will give 12 which is 0000 1100
| (bitwise or)	Binary OR Operator copies a bit if it exists in either operand.	(A | B) will give 61 which is 0011 1101
^ (bitwise XOR)	Binary XOR Operator copies the bit if it is set in one operand but not both.	(A ^ B) will give 49 which is 0011 0001
~ (bitwise compliment)	Binary Ones Complement Operator is unary and has the effect of 'flipping' bits.	(~A) will give -61 which is 1100 0011 in 2's complement form due to a signed binary number.
<< (left shift)	Binary Left Shift Operator. The left operands value is moved left by the number of bits specified by the right operand.	A << 2 will give 240 which is 1111 0000
>> (right shift)	Binary Right Shift Operator. The left operands value is moved right by the number of bits specified by the right operand.	A >> 2 will give 15 which is 1111
>>> (zero fill right shift)	Shift right zero fill operator. The left operands value is moved right by the number of bits specified by the right operand and shifted values are filled up with zeros.	A >>>2 will give 15 which is 0000 1111

Logical Operators

Operator	Description	Example
&& (logical and)	Called Logical AND operator. If both the operands are non-zero, then the condition becomes true.	(A && B) is false
|| (logical or)	Called Logical OR Operator. If any of the two operands are non-zero, then the condition becomes true.	(A || B) is true
! (logical not)	Called Logical NOT Operator. Use to reverses the logical state of its operand. If a condition is true then Logical NOT operator will make false.	!(A && B) is true

The Assignment Operators

Operator	Description	Example
=	Simple assignment operator. Assigns values from right side operands to left side operand.	C = A + B will assign value of A + B into C
+=	Add AND assignment operator. It adds right operand to the left operand and assign the result to left operand.	C += A is equivalent to C = C + A
-=	Subtract AND assignment operator. It subtracts right operand from the left operand and assign the result to left operand.	C -= A is equivalent to C = C – A
*=	Multiply AND assignment operator. It multiplies right operand with the left operand and assign the result to left operand.	C *= A is equivalent to C = C * A
/=	Divide AND assignment operator. It divides left operand with the right operand and assign the result to left operand.	C /= A is equivalent to C = C / A
%=	Modulus AND assignment operator. It takes modulus using two operands and assign the result to left operand.	C %= A is equivalent to C = C % A
<<=	Left shift AND assignment operator.	C <<= 2 is same as C = C << 2
>>=	Right shift AND assignment operator.	C >>= 2 is same as C = C >> 2
&=	Bitwise AND assignment operator.	C &= 2 is same as C = C & 2
^=	bitwise exclusive OR and assignment operator.	C ^= 2 is same as C = C ^ 2
|=	bitwise inclusive OR and assignment operator.	C |= 2 is same as C = C | 2

Misc Operators (Java)