



合肥工业大学
HEFEI UNIVERSITY OF TECHNOLOGY



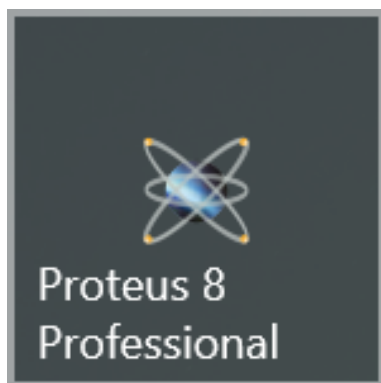
计算机与信息学院
SCHOOL OF COMPUTER SCIENCE AND INFORMATION ENGINEERING
人工智能学院
SCHOOL OF ARTIFICIAL INTELLIGENCE



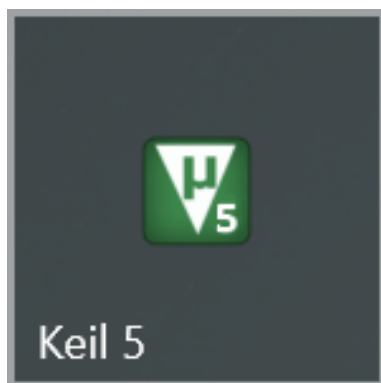
速通嵌入式C语言基础

第二次培训：2022-10-30

电子设计创新实验室 周布伟



电脑端



开发板



仿真工具

- 新建工程文件，添加51单片机；
- 添加所需的元器件，接线；
- 选择单片机，导入 .hex 文件，执行仿真。

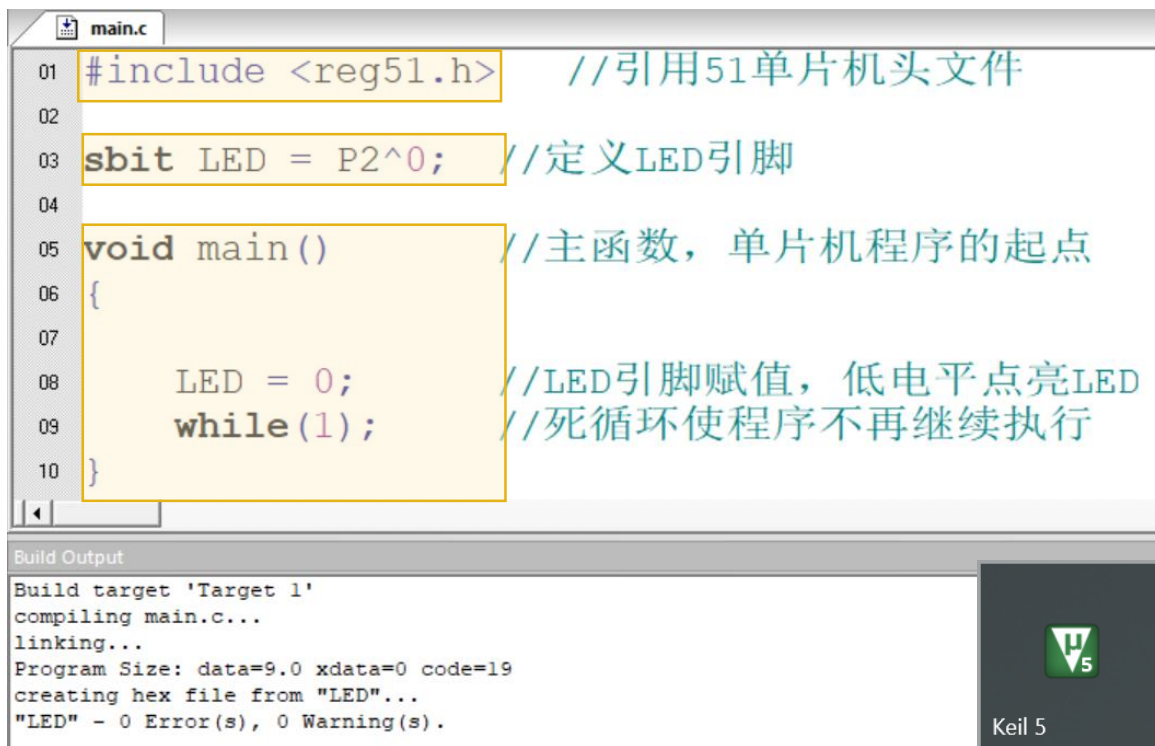
开发工具

- 新建工程文件，选择单片机型号；
- 新建C语言文件，将其添加至工程中；
- 编写代码，编译生成得到 .hex 文件。

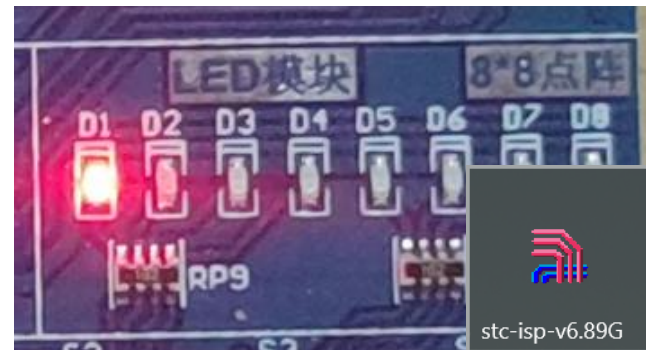
烧录工具

- 将开发板连接至电脑，安装驱动文件；
- 识别单片机，选择正确的单片机型号；
- 导入 .hex 文件，烧录至单片机中运行。

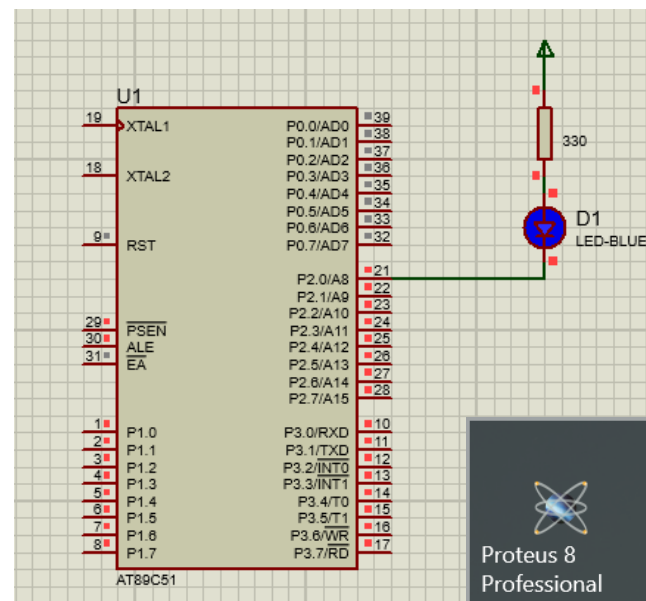
软件基础：代码框架



- 一般需要引入头文件，定义引脚，编写主函数；
- 嵌入式 C 语言和普通 C 语言存在一定的区别。

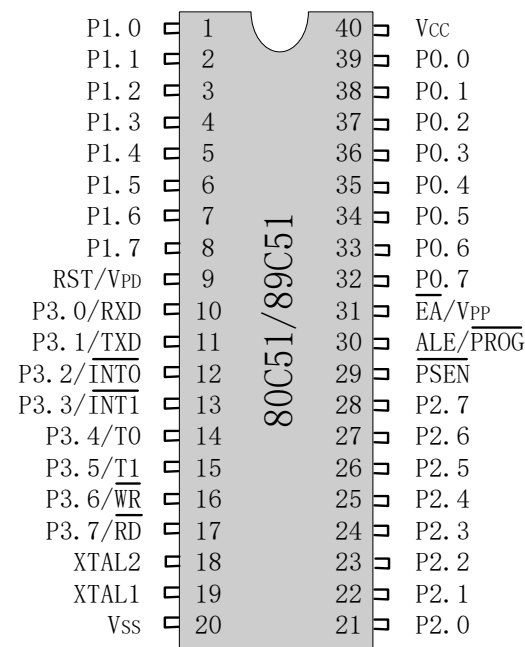
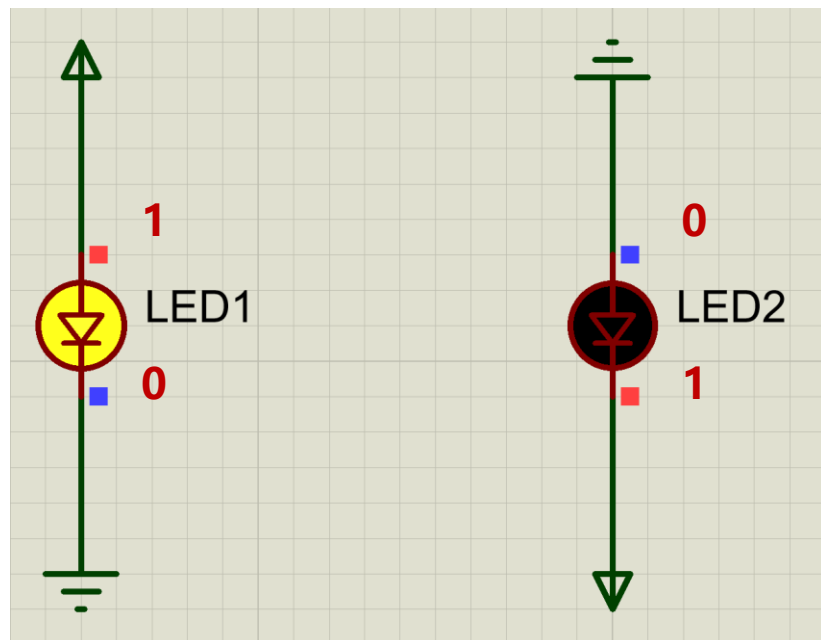


开发板现象



仿真现象

硬件基础：电平特性与引脚



○ 数字电路中只有高电平和低电平两种电平

○ 高电平：5V或者3.3V（取决单片机电源）；

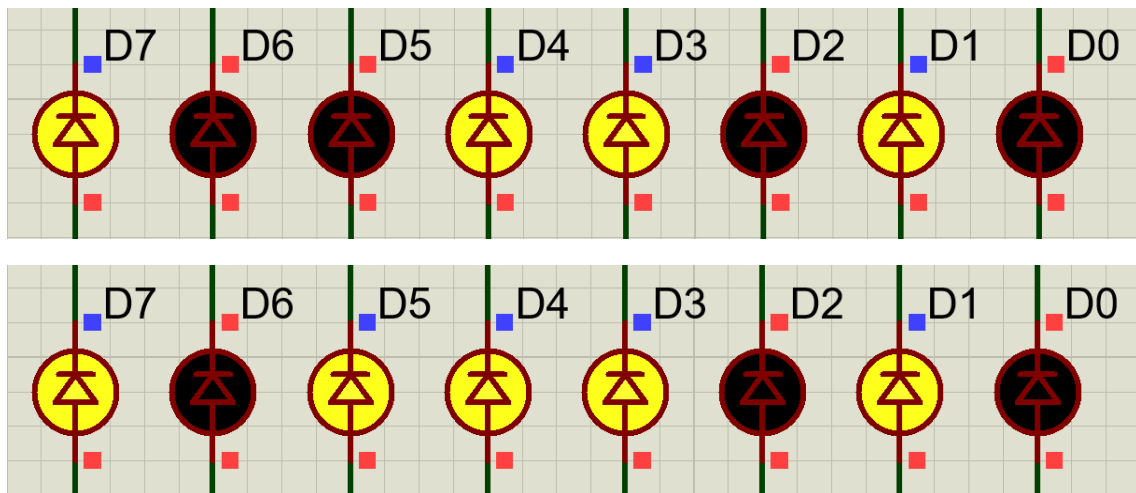
○ 低电平：0V。

○ 51单片机各引脚有着不同的功能

○ 部分有输入输出（I/O）作用；

○ 部分引脚进行了复用。

- 编程实现51单片机同时点亮多个LED
 - 同一时刻有一部分亮，有一部分灭（如亮灭灭亮亮灭亮灭）；
 - 在同一组的引脚，采用更快捷的方式；
 - 点亮和熄灭指定位置的LED（如第5位）。



一 从进制到基本数据类型

二 基本控制结构

三 函数与模块化设计

四 构造数据类型之数组

- 进制是数据的表示方法， x 进制表示数据满 x 后进一位；
- 计算机的数据是用二进制表示的；
- 51单片机端口的高低电平也使用二进制表示。
 - 每一个端口均有8位，每个引脚分别对应二进制数的一位；
 - 每个引脚只有高低电平两种状态，表示该位数只有1和0两种情况。

十进制	二进制	十六进制	十进制	二进制	十六进制
0	0000	0	8	1000	8
1	0001	1	9	1001	9
2	0010	2	10	1010	A
3	0011	3	11	1011	B
4	0100	4	12	1100	C
5	0101	5	13	1101	D
6	0110	6	14	1110	E
7	0111	7	15	1111	F

- 单片机内部的数据以二进制方式存储，通过十进制或十六进制表示；
- 嵌入式C语言中不支持二进制表示，默认为十进制表示，表示十六进制须在前面加上0x（如0xFF），不区分大小写；
- 进制转换可以借助电脑自带的计算器工具。

○ 二进制、十六进制 → 十进制

原则：按位权展开求和

$$101B = 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 = 5D$$

$$101AH = 1 \times 16^3 + 0 \times 16^2 + 1 \times 16^1 + 10 \times 16^0 = 4122D$$


○ 十进制 → 二进制、十六进制

原则：除基直到商为0，倒取余数（除基取余）

$$100D = 1100100B$$

$$100D = 64H$$

2	100	
2	50	0
2	25	0
2	12	1
2	6	0
2	3	0
2	1	1
	0	1



○ 二进制 → 十六进制

原则：从右向左四位一组，不足高位补零，写出四位二进制对应的十六进制符号

10011011100B



010011011100B

4 D C H

○ 十六进制 → 二进制

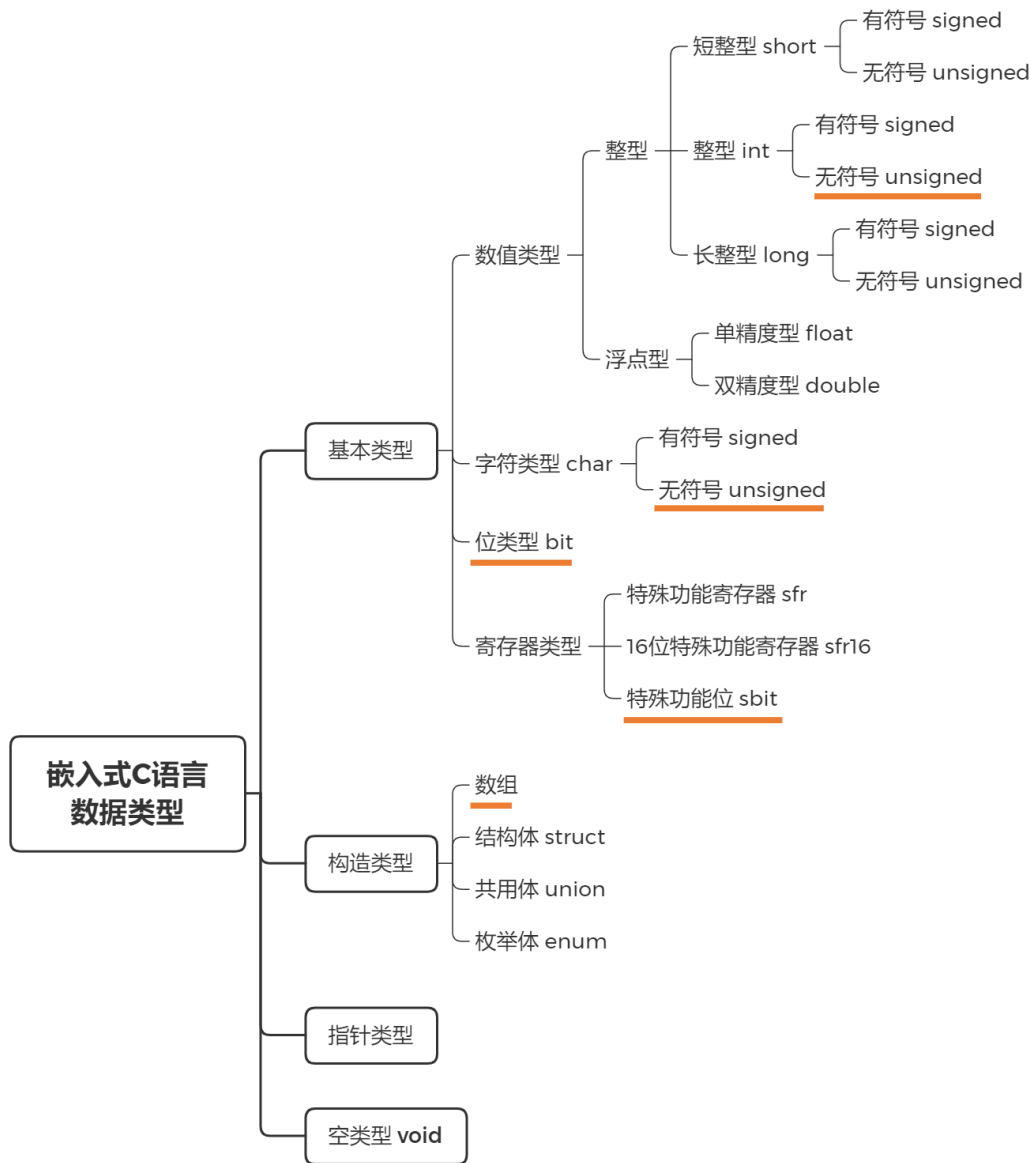
原则：按顺序写出一位十六进制对应的四位二进制序列

4FH



01001111B

- 数据是对客观事物特征抽象的符号化表示；
 - 客观事物不同，表示的方法也不同；
 - 客观事物不同，计算机的处理方法也不同。
- 计算机由于工程的限制，只能在有限精度和有限范围内在工程上近似地描述数据。根据程序处理的数据对象，应规定数据的类型。
 - 一位二进制称为一个比特（bit），八位二进制构成一个字节（Byte）；
 - 字节是存储的基本单元，是计算机对数据操作的最小单位。



- 程序中的数据可以分为常量（如整型常量、字符型常量、浮点型常量等，具有一定精度和范围）和变量，常量可以直接给出，可以通过`#define`定义，而变量需要命名并声明类型，且必须先声明后使用

```
#define PI 3.1415926
```

`#define` 宏名 字符串

- 通过 `typedef` 语句，可以定义自己命名的数据类型。实际上是给数据类型重新起个名字，如

```
typedef unsigned char uint8;
```

```
typedef unsigned int uint16;
```

`typedef` `type` 标识符; C语言的类型名

↑
用户定义类型名

变量类型	说明词	范围	bit
整型	int	-32768~32767	16
	short [int]	-32768~32767	16
	long [int]	$-2^{31} \sim +2^{31}-1$	32
	unsigned [int]	0 ~ 65535	16
	unsigned short [int]	0 ~ 65535	16
	unsigned long [int]	0 ~ $2^{32}-1$	32
字符类型	char	-128 ~ 127	8
	unsigned char	0 ~ 255	8
浮点类型	float	$10^{-38} \sim 10^{+38}$	32
	double	$10^{-308} \sim 10^{+308}$	64
位类型	bit	0 ~ 1	1

- 浮点类型的范围是大致的范围，精度分别为 7 位和 16 位；
- 寄存器数据类型直接用于对相关寄存器进行声明。

- 程序的核心是对数据按照算法进行处理（运算）。C 语言提供了强大的数据运算功能。数据通过运算符连接的式子称为表达式，表达式根据运算关系对数据运算，并得到一个值。

○ 赋值运算

格式:

$V = \text{expression};$ /* 将expression的值赋给变量V*/

$\text{int } a = 2;$ /*说明时赋值*/

$\text{int } b;$

$b = 3 + 2;$ /*说明后赋值*/

说明:

- (1) =是赋值号（动作）；
- (2) 赋值运算的左值只能是变量；
- (3) 赋值号两边类型应该一致，如不一致，以变量类型转换。

$a = a + 2;$

$\text{iCount} = \text{iCount} + 10;$ $a + 2 = 12;$ /*Error!*/

$\text{int } a = 2.5 ;$ /* 2→a */

○ 算术运算：双目算术运算（两个运算对象参加的运算）

运算	运算规则	示例
+	加	$2 + 3$
-	减	$a - b$
*	乘	$2 * (-a)$
/	除	$12f / 3.0f$
%	求余数	$15 \% 4$

说明：

- (1) 乘号不能省略；
- (2) 只能对整型或字符型数据运算。余数符号与被除数相同。

○ 算术运算：单目算术运算（一个运算对象参加的运算）

运算符	前置	后置	运算关系
++	$++a$	$a++$	$a=a+1$
--	$--a$	$a--$	$a=a-1$

说明：

- (1) 运算对象只能是一个变量。 $2++$; /* Error !*/ $x=++i$; /* $i=i+1$; $x=i$; */
- (2) 前置是先运算，后引用，而后置则是先引用，后运算。

```
int i, x;
i = 5;
x = i++; /* x=i; i=i+1; */
i = 5;
```

○ 算术运算：算术运算赋值

运算符	表达式示例	运算关系
<code>+=</code>	<code>a+=3</code>	<code>a=a+3</code>
<code>-=</code>	<code>b -= c</code>	<code>b=b-c</code>
<code>* =</code>	<code>a* =2</code>	<code>a=a*2</code>
<code>/=</code>	<code>s/=t</code>	<code>s=s/t</code>
<code>%=</code>	<code>a%=5</code>	<code>a=a%5</code>

说明：

- (1) 运算对象的左值只能是一个变量。 `(a+2)+=5;` /*Error !*/
- (2) `%=`运算的对象，必须是整型或字符型。

○ 算术运算的说明

- 运算符存在优先级关系，可以通过括号改变优先级；
- 不同数据类型在运算时会发生类型转换，有时会影响精度，尽量保证都是同种数据类型参与运算。

○ 逻辑运算

逻辑运算运算时判断对象真、假的运算。标准C语言没有提供逻辑类型。51单片机C语言中bit类型可近似看作逻辑类型。

表达式的值所对应的逻辑值	
表达式的值	表示的逻辑关系
非0	真true
0	假false

运算符	逻辑关系
&&	逻辑与
	逻辑或
!	逻辑非

运算对象非0代表逻辑真，是0代表逻辑假。也就是说任何类型的量都有逻辑值。运算对象之间可以通过逻辑运算符进行逻辑运算

逻辑运算的结果用整型值表示。

- ❖ 运算结果为真时，得到整型值1；
- ❖ 运算结果为假时，得到整型值0。

```
int a=2,b=0;
```

```
a
```

```
b
```

```
a+b
```

```
a && b
```

○ 关系运算

关系运算是比较两个表达式的数值相互关系的运算。

运算符	比较的关系	实例
>	大于	$a > b$
>=	大于等于	$a \geq b$
<	小于	$2 < 1$
<=	小于等于	$c \leq d$
==	等于	$1 == c$
!=	不等于	$1 != 3$

关系运算规则：参加运算的表达式的从左到右按关系运算符提供的关系进行比较，满足关系得到整型值1，不满足关系得到整型值0。



表示数值关系的原则：开放区间用或；闭合区间用与

○ 位运算：逻辑运算

运算符	运算规则	示例
~	按位取反	~b
&	按位与	3 & 7
	按位或	s 0x00ff
^	按位异或	x^(y+5)

说明：

- (1) 运算对象必须是整型或字符型；
- (2) ~为单目运算；
- (3) 运算按对应位，逐位进行。

○ 位运算：逻辑运算赋值

运算符	示例	运算规则
&=	a &= 7	a = a & 7
=	s = 0x00ff	s = s 0x00ff
^=	x ^= (y+5)	x = x ^ (y+5)

说明：

- (1) 运算对象必须是整型或字符型；
- (2) 左值只能是变量；
- (3) 运算赋值优先级与赋值相同。

○ 位运算：逻辑运算

位取反运算~ 真值表	
b1	~b1
0	1
1	0

作用：将所有位状态翻转

位或运算 真值表		
b1	b2	b1 b2
0	0	0
0	1	1
1	0	1
1	1	1

作用：将某些位置1，其余保持不变

位与运算 & 真值表		
b1	b2	b1&b2
0	0	0
0	1	0
1	0	0
1	1	1

作用：将某些位清零，其余保持不变

位异或运算 ^ 真值表		
b1	b2	b1^b2
0	0	0
0	1	1
1	0	1
1	1	0

作用：将某些位取反，其余保持不变

○ 位运算：移位运算

运算符	运算规则	示例
<<	左移	$b \ll 3$
>>	右移	$s \gg (n+3)$

说明：

- (1) 运算对象必须是整型或字符型；
- (2) 整体二进制位移动。

○ 位运算：移位运算赋值

运算符	示例	运算规则
<<=	$a \ll= 7$	$a = a \ll 7$
>>=	$s \gg= (n+3)$	$s = s \gg (n + 3)$

说明：

- (1) 运算对象必须是整型或字符型；
- (2) 左值只能是变量；
- (3) 运算赋值优先级与赋值相同。

○ 位运算：移位运算

○ 左移 <<

格式：a << n

操作：将a中的二进制位，左移n（无符号整数），高位移出低位补0。

```
unsigned short a;           000000000000000010
a = 2;                       000000000000100000
a = a << 4;
```

对于无符号数，左移一位相当于乘 2。

○ 右移 >>

格式：a >> n

操作：

对于无符号数，低位移出，高位补0。右移一位相当于除 2。

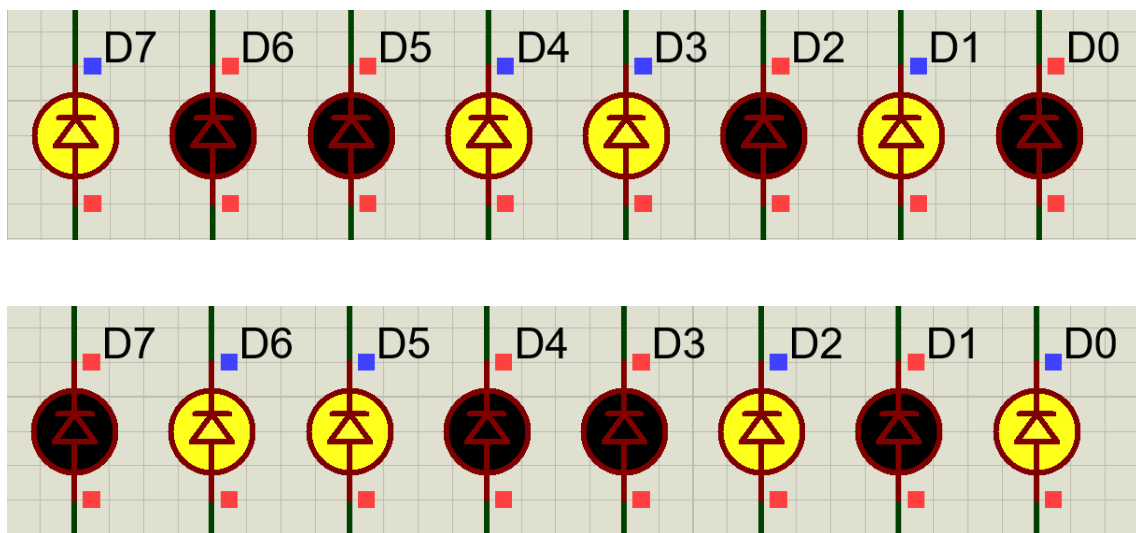
○ 运算优先级

○ 了解即可

○ 可通过括号
改变优先级

级别	运算符	结合顺序
1	() [] -> .	从左向右
2	! - ++ -- (type) sizeof * & ~	从右向左
3	* / %	从左向右
4	+ -	从左向右
5	<< >>	从左向右
6	< <= > >=	从左向右
7	== !=	从左向右
8	&	从左向右
9	^	从左向右
10		从左向右
11	&&	从左向右
12		从左向右
13	? :	从右向左
14	= op =	从右向左
15	,	从左向右

- 编程实现51单片机驱动多个LED产生亮-灭闪烁的效果
 - LED闪烁背后的原理
 - 让LED不断闪烁
 - 让LED闪烁一定次数（如5次）



一 从进制到基本数据类型

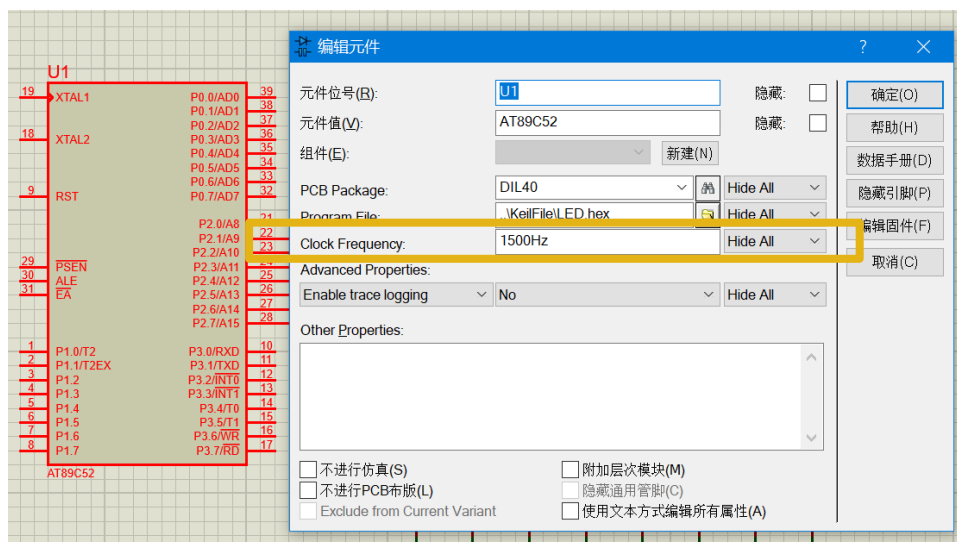
二 基本控制结构

三 函数与模块化设计

四 构造数据类型之数组

○ 语句的执行速度

- 一般情况下，单片机会以非常快的速度执行一条语句（严格上说应该是多条机器指令），肉眼无法察觉；
- 语句的执行速度和单片机的晶振频率有关，频率越高，语句执行的速度越快。不同语句的执行速度不同，对于整型，加法要快于乘法；
- 在 Proteus 仿真环境下，可以人为地改变单片机地晶振频率以调整语句的执行速度。如果要从定量的角度分析语句执行的速度，需要了解机器周期等概念。



○ 复合语句

C语言可以用{ }包括一系列的语句。一对{ }所包含的内容称为一个复合语句。其中可以含有0到多条C语言语句。凡是出现单一语句的地方都可以使用复合语句，复合语句可以嵌套。

复合语句可以作为分支和循环的块，作为标识符的作用域。

○ 基本概念

程序经常会重复执行某些相同的操作，如：

求： $s=1+2+3+4+\dots+100$

算法描述：

① $s=0; i=1;$ 初始化部分。

② $s+=i;$
 $i++;$ 循环体。含有使条件趋假的语句。

③ 判断 i 是否小于等于 100

如果 i 小于等于 100，重复②；
否则，结束。

循环的条件。

注意：循环应在有限次完成。

此类根据“条件”重复执行相同算法的结构，称为循环。

C 语言提供了三类实现循环的语句：

`while`， `do while`， `for`

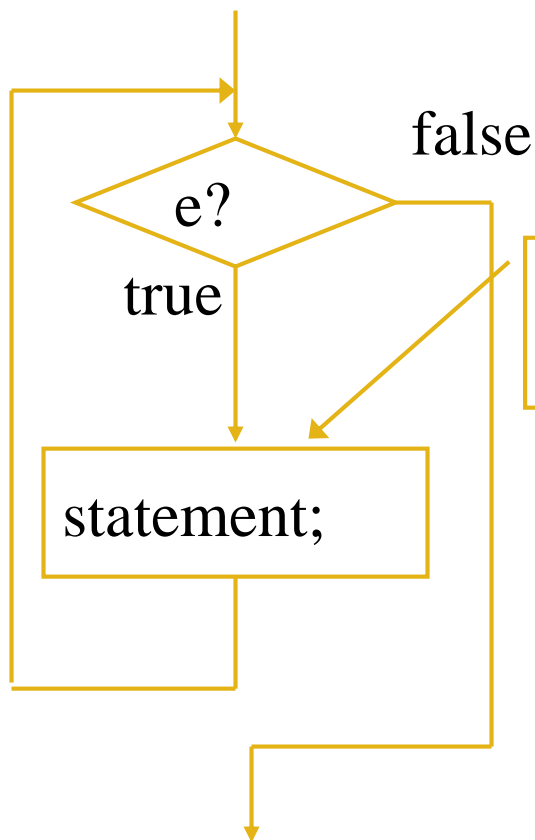
○ while循环（当型循环）

格式：

```
while(expression)  
statement;
```

表达式：值非0，表示满足条件；值为0代表不满足条件。

流程：



含有使条件趋假的语句。

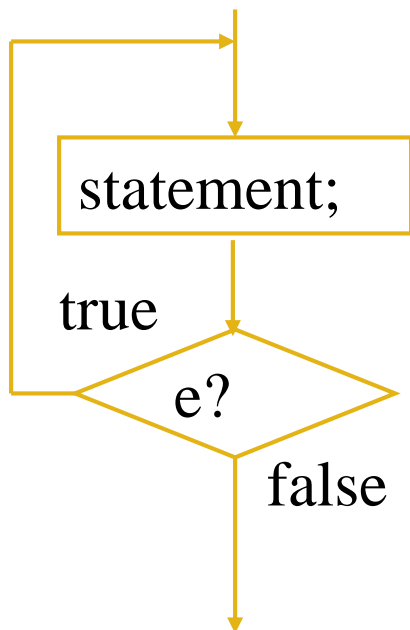
语句（复合语句），重复执行部分（循环体）。

○ do – while循环（直到型循环）

格式：

```
do {  
    statement;  
}while (expression );
```

流程：



while循环与do-while循环的区别：
while循环先判条件，后执行循环体；
do –while循环先执行循环体，后判条件。

含有使条件趋假的语句

○ for循环

格式:

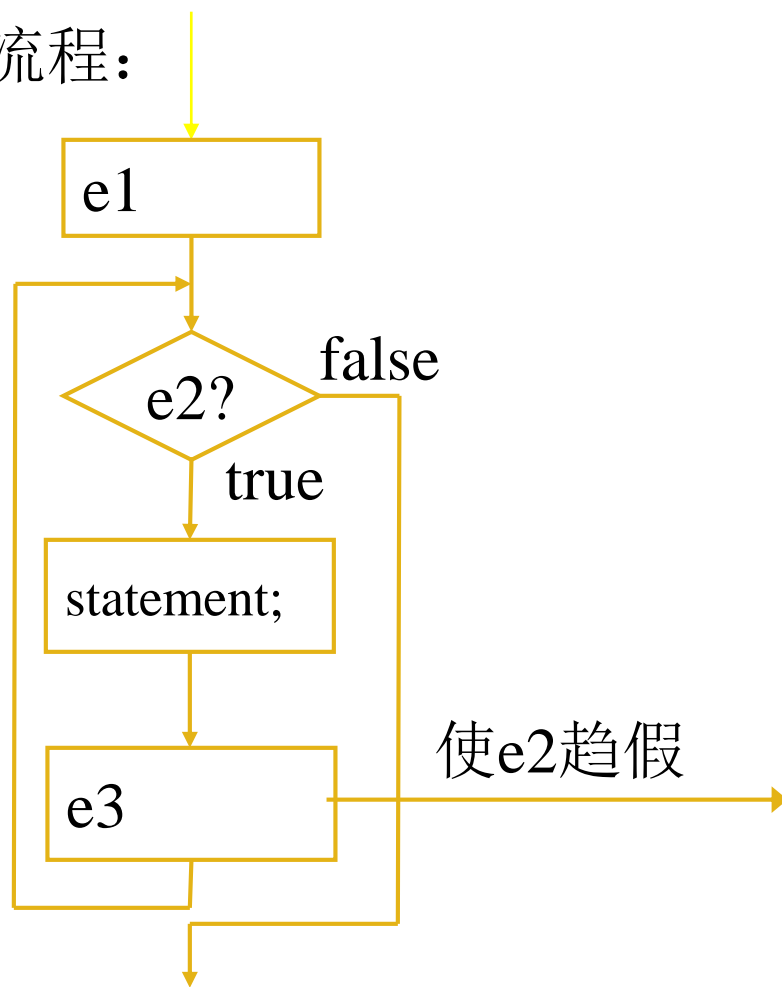
```
for (e1; e2; e3 )  
    statement;
```

初值表达式

测试表达式

增值表达式

流程:



在for循环中，e1、e2、e3都可以省略！

○ 无限循环和空循环

① 条件为恒真的循环——无限循环

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

```
do{
```

```
    ...}while(1);
```

```
for(;;){...}
```

通过条件控制的break语句退出循环。

② 循环体为空语句的循环——空循环

```
for (i=1 ;i<=MAX ; i++);
```

作用：程序延时。

—————→ { ; }

空语句

○ 循环的中断(break)和继续(continue)

○ 循环的中断：break语句

概念：循环体中可以加分支，判断是否继续执行循环，break语句可以提前结束循环。

○ 继续循环：continue语句

continue语句的作用是跳过本次循环剩余的循环体内容，执行下次循环。

- 循环嵌套
 - 内外层循环采用缩进形式;
 - while和do- while和for可以相互嵌套
 - 执行次数为内层循环次数和外层循环次数的乘积

○ 基本概念

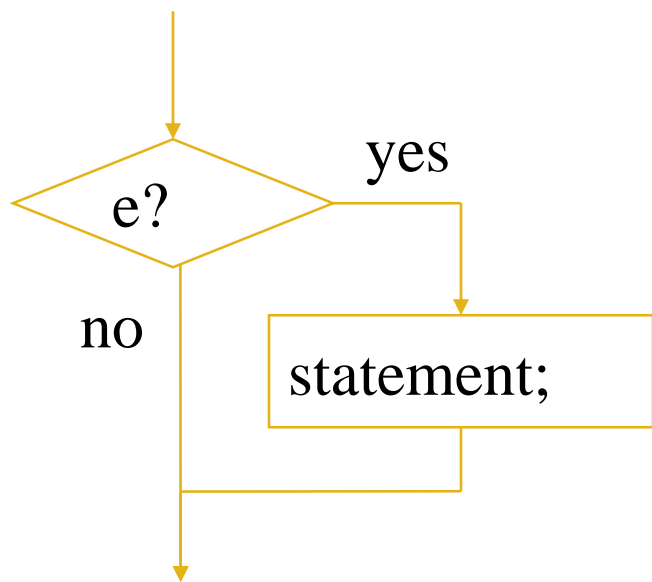
由标准的分支结构可以演化成单分支、多分支结构。C语言的分支语句有if、if – else、switch三种。

语句之间存在嵌套关系，书写采取缩进形式，便于区分。else与最近的if相匹配，从内到外。

○ if 结构

格式：
`if(expression)statement;`

流程图：
表达式，非0为yes，0为no。
语句，可以是复合语句。



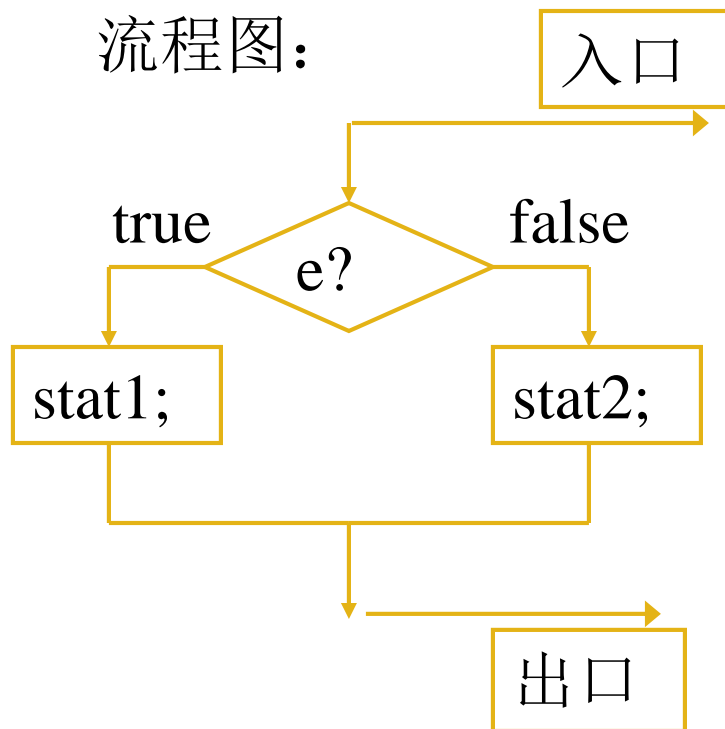
○ if – else 结构

格式：

```
if(expression)
    stat1;
else
    stat2;
```

语句或复合语句。

流程图：

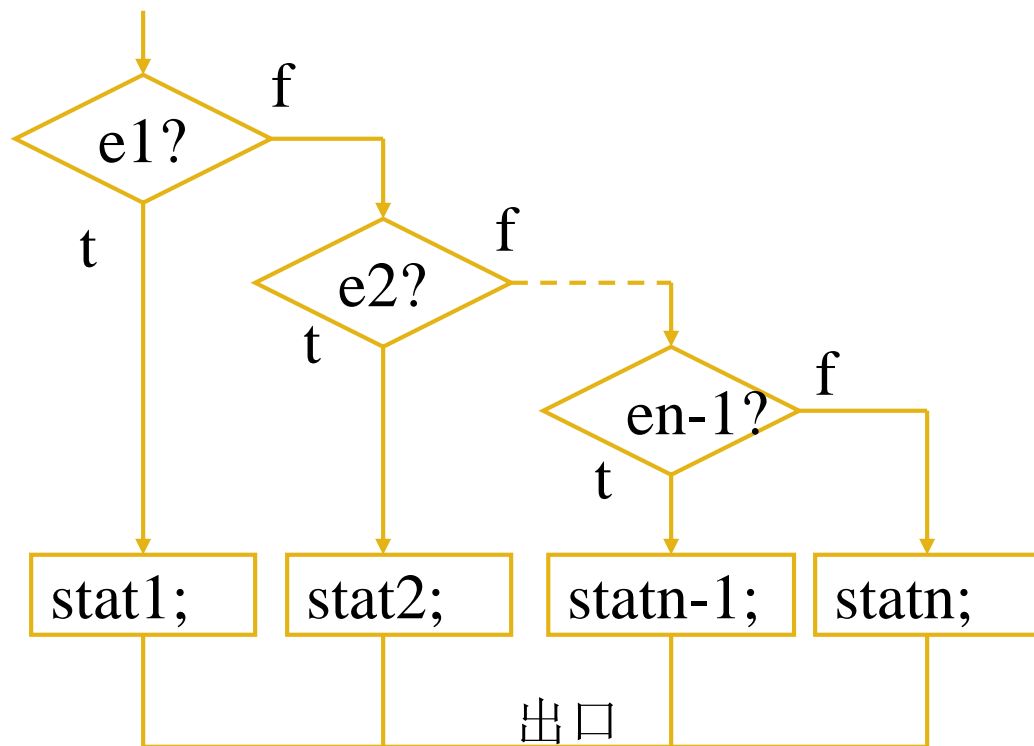


○ if-else if结构 (Multi-line)

格式:

```
if(e1)
    stat1;
else if(e2)
    stat2;
else if(e3)
    stat3;
...
else if(en-1)
    statn-1;
else
    statn;
```

框图:



n-1个条件，满足某个条件，执行对应的语句，然后到出口。

○ switch语句（多分支结构）

格式：

```
switch (expression)
```

```
{
```

```
    case 常量表达式1: statement 1;
```

```
    case 常量表达式2: statement 2;
```

```
    ...
```

```
    case 常量表达式n-1: statement n-1;
```

```
    default : statement n;
```

```
}
```

流程：

①先求expression的值。

②依次比较expression和各常量表达式的值。

③如果与第i个常量表达式相等，则执行第i条以后的语句。

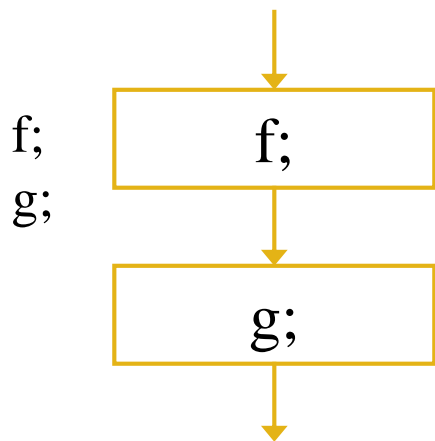
④如果都不相等，则执行default以后的语句。

只能是整型或字符型表达式。

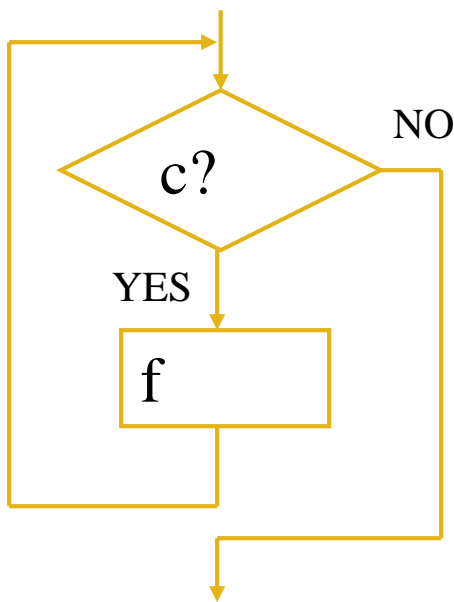
整型字符型常量表达式。
表达式的值要互不能相等！

○ 三种控制结构

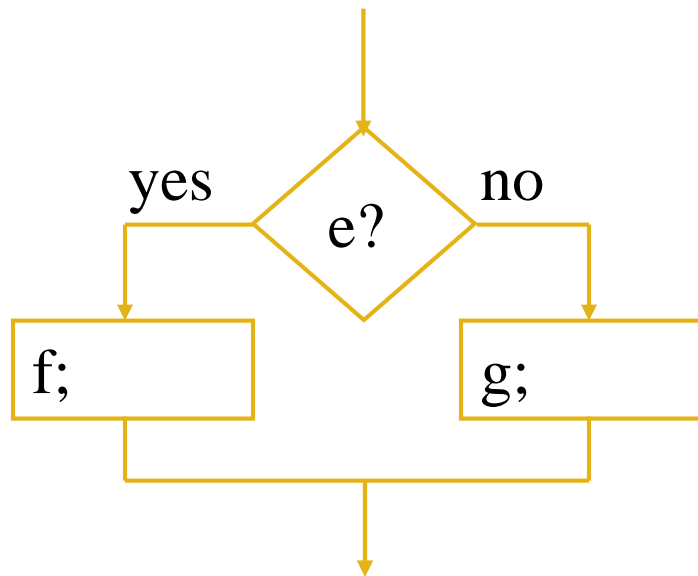
1966年BÖHM & Jacopini 证明：只要三种控制结构就能表达用一个入口和一个出口框图所能表达的任何程序逻辑。



while c is true
do f



if(e)
f;
else
g;



- 编程实现51单片机驱动多个LED产生流水灯的效果
 - 同一时刻只有一个LED亮
 - 多个方向流动
 - 指定流动的时间间隔（如500ms）



一 从进制到基本数据类型

二 基本控制结构

三 函数与模块化设计

四 构造数据类型之数组

○ 基本概念

- 函数是一组一起执行一个任务的语句。所有简单的程序都可以定义其他额外的函数。对于调用者而言相当于一个黑盒；
- 可以把代码划分到不同的函数中，划分通常是根据每个函数执行一个特定的任务来进行的。

○ 函数的引用

函数名(实参表)

一般引用方式有三种：

语 句 形 式： `max(a,b);`

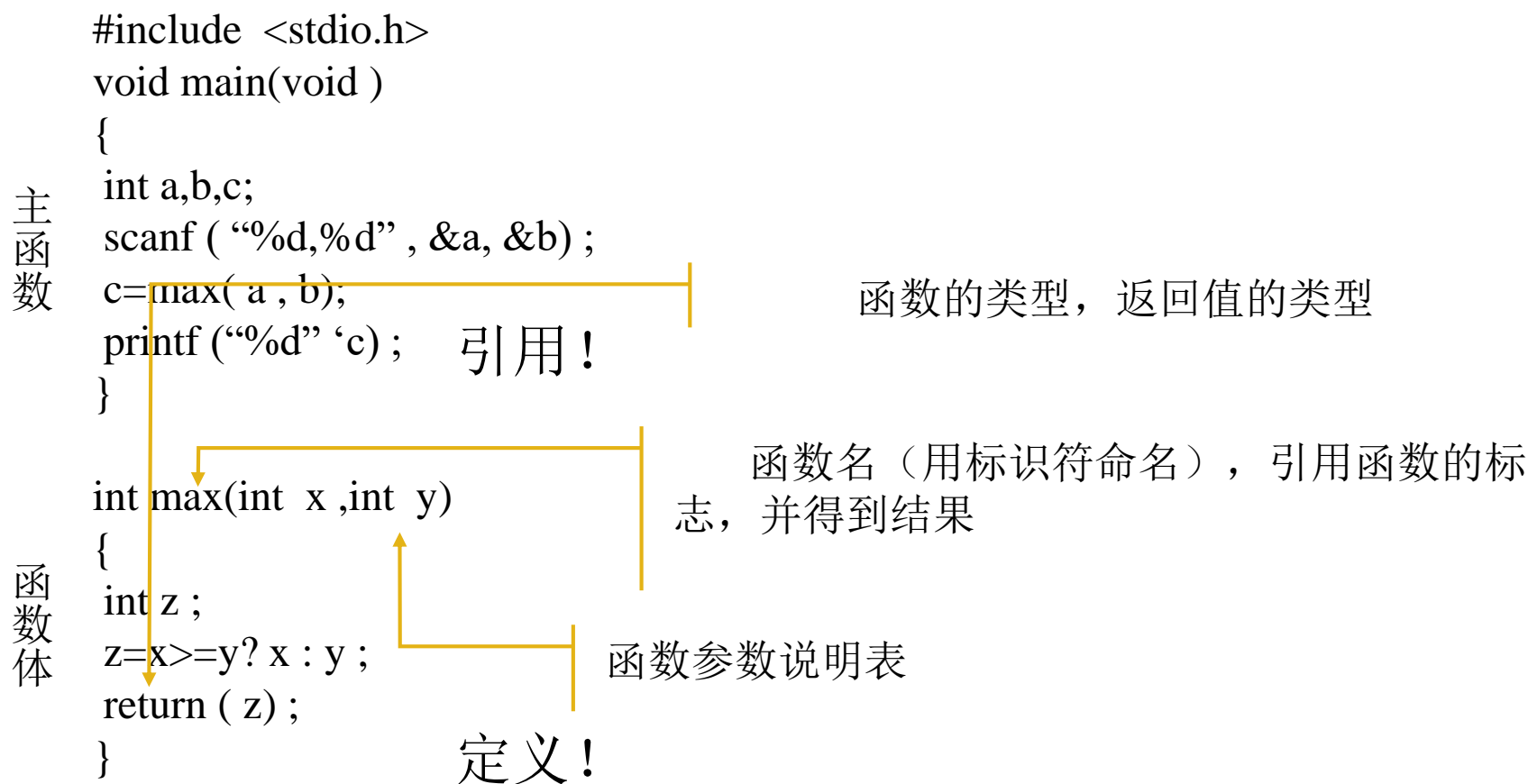
表达式形式： `c=max(a, b)*2 ;`

函 数 参 数： `c=max (a , max (b , d)) ;`

○ 函数定义时要确定如下四点：

函数的名称、函数的类型、函数的参数、函数的功能

○ 函数的结构



○ 函数的结构

格式:

```
type 函数名(参数说明表)
{
    内部说明语句 ;
    功能语句 ;
}
```

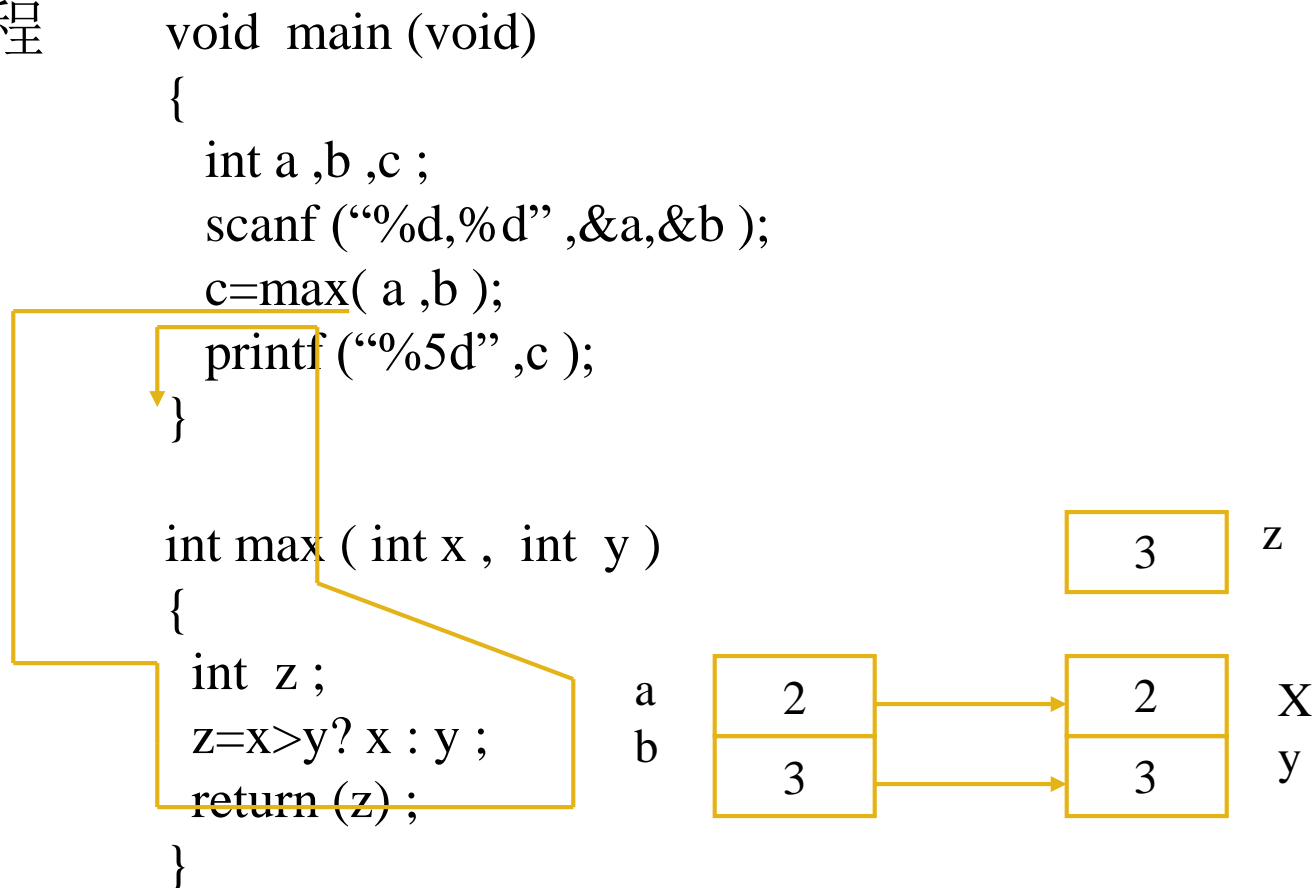
- (1) 函数不能嵌套定义，但可以嵌套引用，包括引用自己；
- (2) 函数若无返回值，`type` 应说明为空类型`void`。函数无参数应定义成`void`；
- (3) 如果函数有返回值，应含有`return`语句。

```
int max(int x,int y)
{
    ...
    int cx(int c,int d)
    {
        ...
    } /*Error !*/
    ...
}
```

○ 关于返回值的几点说明：

1. 函数可以通过一个return语句返回一个值，也可以不返回值，此时应在定义函数时用void类型加以说明；
2. 函数中可以出现多个return语句，遇到一个return 语句，则返回值，且返回调用函数，继续执行；
3. 为了确保参数和返回值类型正确，一般须在函数调用前对其类型和参数的类型加以说明，该说明称之为原型声明。

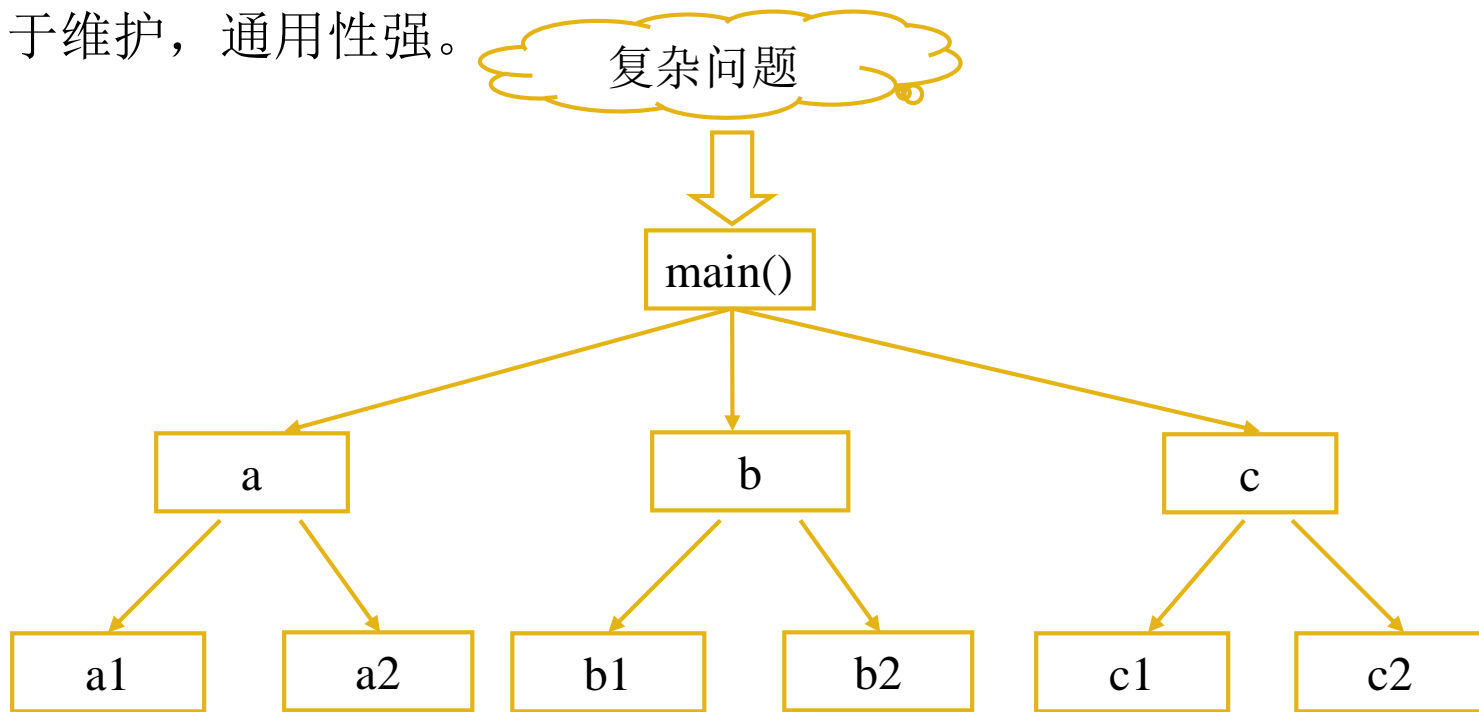
○ 函数的引用过程



- (1) 先计算实参的值，从右向左向函数传递调赋值给形参。
- (2) 转移在函数中运行，执行到一个return语句，将返回表达式的值。由函数名带回给调用函数。
- (3) 函数如果没有return语句，由最后一个}返回一个不确定的值！

○ 设计思想

- 在程序设计时，我们可以采用一种自顶向下的设计方法，也就是将复杂的系统划分为相对独立的，功能较为单一功能的子系统的组合。
- 每个子系统称为模块，在C语言中表现为函数。各模块之间的关系称之为接口。函数实现的功能单一完整，可以独立设计，单独调试。易于维护，通用性强。



- 编程实现51单片机驱动多个LED产生流水灯的效果
 - 同一时刻有多个LED亮（如亮灭灭亮亮灭亮灭），流水不影响数量和相对位置
 - 按顺序展示一串指定的数字序列（如20221030）



一 从进制到基本数据类型

二 基本控制结构

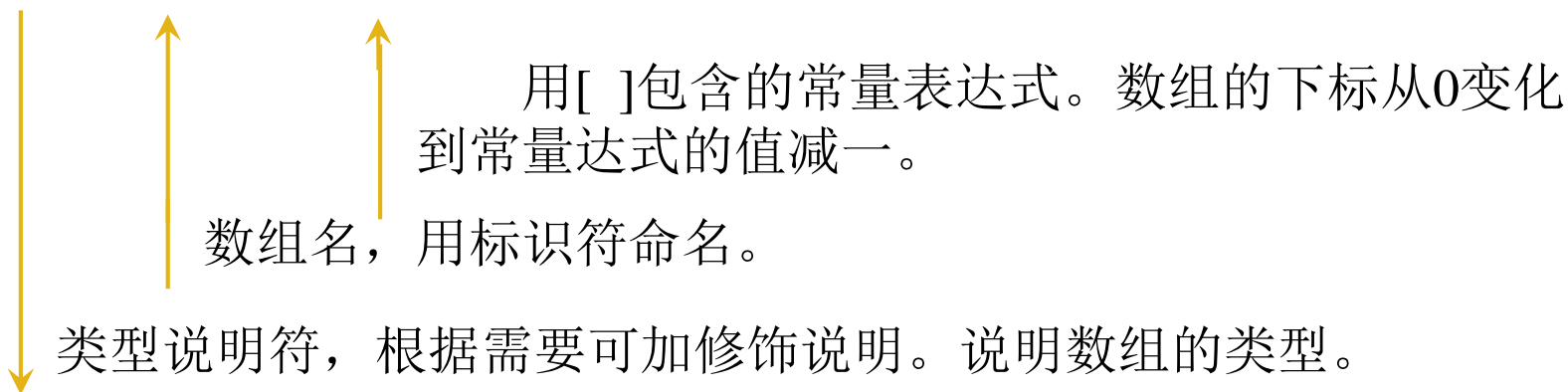
三 函数与模块化设计

四 构造数据类型之数组

- C 语言可以根据用户需要，用基本数据类型定义特殊性质的数据类型，称为构造类型。构造类型有：数组、结构、联合；
- 相同数据类型变量的有序集合称为数组。有序表现在数组元素在内存中连续存放。
- 数组用一个名字作为标识。为区分各元素，每个元素有一个用整型表示的序号，称之为下标。下标可以有多个，下标的个数称为数组的维数。
- 数组必须先说明后使用，说明数组的名字（标识）。说明数组的类型。说明数组的维数。确定各维下标的变化范围。
- 编译系统将根据说明，开辟内存单元按特有的顺序和相应的类型为各元素分配内存单元。

○ 一维数组的说明

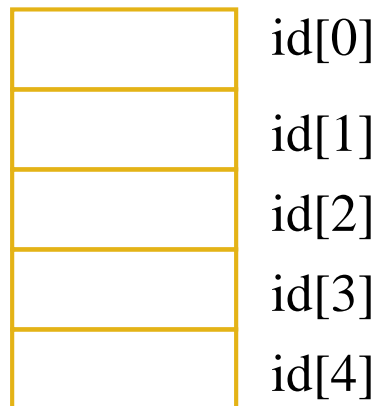
type array1[常量表达式], ..., arrayn[常量表达式];



当说明数组后，编译时系统会根据定义的类型分配连续的一段内存单元给数组的各元素。

系统为数组分配的连续内存单元，每个单元占两个BYTE。首地址用数组名id表示。

short id[5], iyear[10];
float fScore[36];



○ 一维数组的初始化

数组的元素可以在说明数组时初始化。

```
int a[10]={1,2,3,4,5,6,7,8,9,10};  
/*说明数组，同时初始化全部元素。*/
```

```
float fValue[10]={1.0,2.0,3.0};  
/*说明数组，给部分元素初值，其余元素为0。*/
```

```
unsigned a[ ]={0x0000,0x0001,0x0002};  
/*当数组元素全部赋初值时，可以不指定长度*/
```

○ 一维数组的引用

原则：只能引用数组元素，而不能引用整个数组。

引用方式：数组名[整型表达式] /*下标变量*/

每个数组元素，可以出现在简单变量能够出现的任何地方。注意数组范围

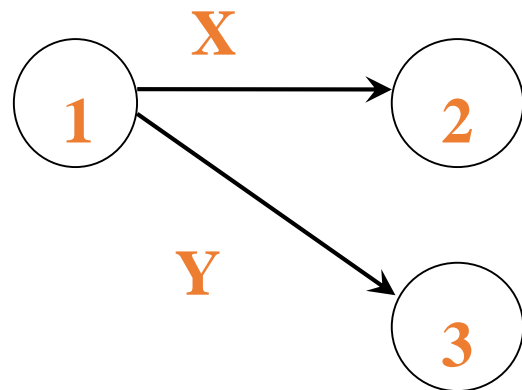
```
a[1]=12;  
s=a[2]+a[1]*20;
```

○ 状态的表示

结点代表状态，用圆圈表示。

状态之间用箭弧连结，箭弧上的标记(字符)代表状态转移的条件。

一张转换图只包含有限个状态，其中有一个为初态，至少要有有一个终态。



○ 状态机的实现

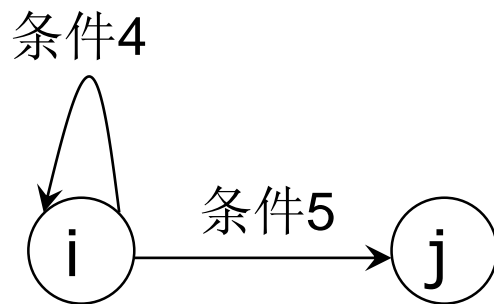
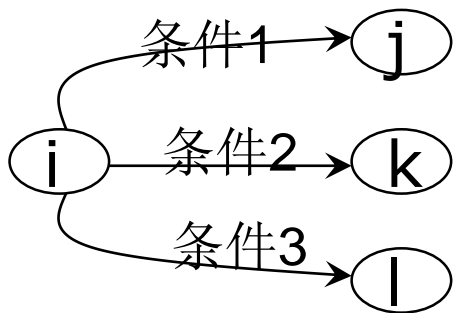
○ 思想：每个状态结对应一小段程序

○ 做法：

○ 对不含回路的分叉结，可用一个CASE语句或一组IF-THEN-ELSE语句实现；

○ 对含回路的状态结，可对应一段由WHILE结构和IF语句构成的程序；

○ 终态结表示算法结束。



1. (90分)编程题：通过P2端口（有开发板的以开发板定义为准）控制8个LED按照顺序实现以下功能。代码编写符合规范，贴合本次授课内容，须有适当的注释。
 - LED初始状态为“亮亮亮亮灭灭灭灭”，控制使其状态发生两次翻转，时间间隔分别为2s和1s（不要求精确延时）；
 - 分两次显示学号后四位，通过LED二进制表示，亮表示1，灭表示0状态切换的时间间隔自定；
 - 使LED第6位不断闪烁，闪烁时间自定。
2. (10分)简答题：结合自身代码的编程和调试过程，分享在51单片机学习过程中遇到的问题 and 解决的办法。

1. 第一题需要在程序文件(.c)开头通过注释介绍实现的功能；
2. 本次作业11月6日（星期日）23:59截止，作业以“学号-姓名-第四次作业”格式命名发送至413732041@qq.com，邮件标题与文件名相一致；
3. 根据题号创建文件夹，并统一打包，主要包含程序文件(.c) 运行文件(.hex) 仿真文件(.pdsprj)、运行视频、简答题（.doc/.docx/.txt/.pdf/图片格式）等，有实物的同学不需要提交仿真文件；
4. 视频要求清晰呈现实验现象，最好压缩到10M以内。

📁 > 2021217100-周百威-第四次作业

名称

自行命名代替

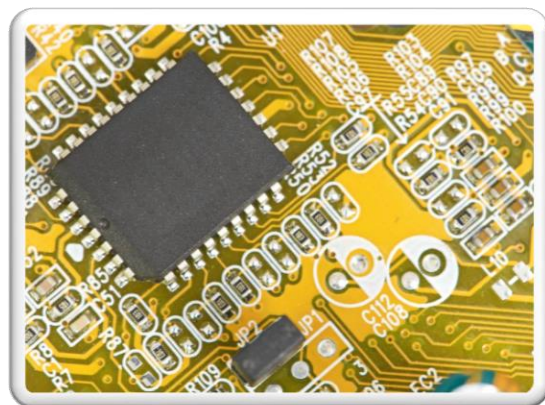
📁 1-外部中断实现1602清屏与流程图

📁 2-LED控制

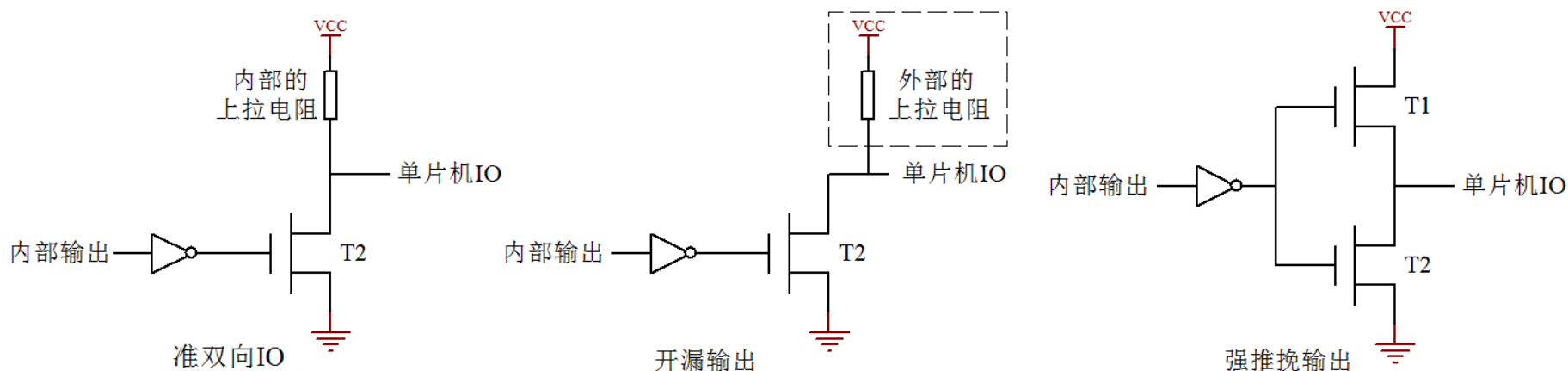
📁 3-简答题

📁 4-预习 (选做)

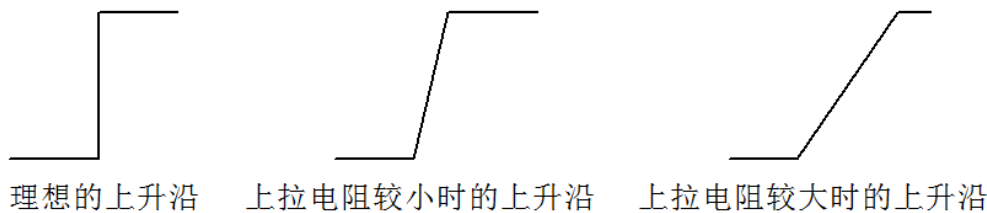
▲第四次培训作业文件命名规范(供参考)



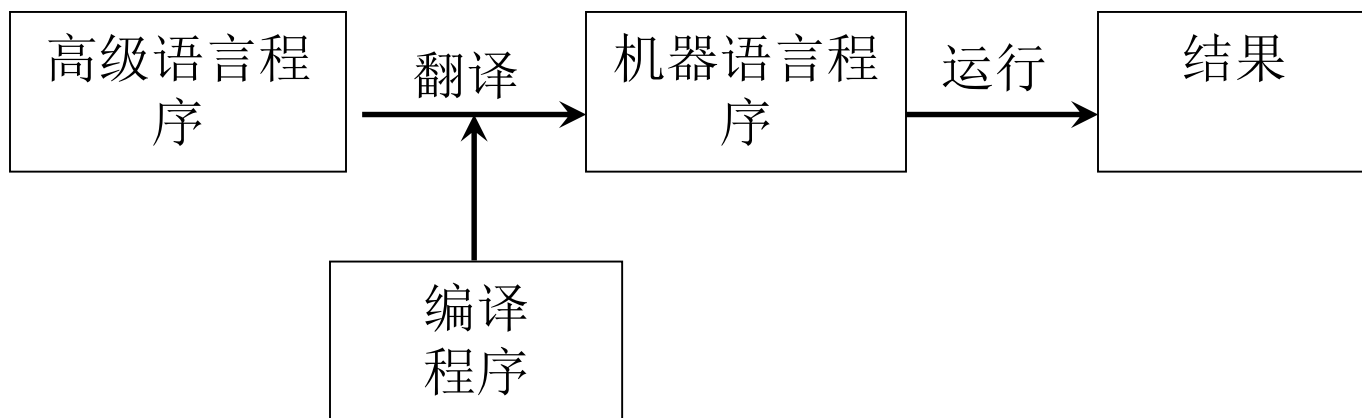
感谢倾听
The End



- 上拉电阻就是将不确定的信号通过一个电阻拉到高电平，同时此电阻起到一个限流的作用，下拉就是下拉到低电平。
- 上拉电阻有OC门要输出高电平，外部必须加上拉电阻。加大普通IO口的驱动能力。起到限流的作用。抵抗电磁干扰。
- 从降低功耗方面考虑应该足够大，因为电阻越大，电流越小。从确保足够的引脚驱动能力考虑应该足够小，电阻越小，电流才能越大。开漏输出时，过大的上拉电阻会导致信号上升沿变缓。



- C语言属于高级语言，接近于人类语言，方便理解，但是机器是无法直接识别C语言，它只能识别二进制序列（机器语言）；
- Keil 所作的一部分工作就是将我们能够理解的 C 语言转换为单片机所能理解的机器语言。把某一种高级语言程序（如 C 语言程序）等价地转换成另一种低级语言程序(如机器语言程序)的过程称为编译，完成这样的过程的工具被称为编译程序或编译器；
- 通常一条高级语言程序会被编译成多条机器指令。



- 振荡周期：也称时钟周期，是指为单片机提供时钟脉冲信号的振荡源（如晶振）的周期（周期为频率的倒数），通常开发板上为12MHz或11.0592MHz。
- 状态周期：每个状态周期为时钟周期的 2 倍，是振荡周期经二分频后得到的。
- 机器周期：一个机器周期包含 6 个状态周期S1~S6，也就是 12 个时钟周期。在一个机器周期内，CPU可以完成一个独立的操作。
- 指令周期：它是指CPU完成一条操作所需的全部时间。每条指令执行时间都是有一个或几个机器周期组成。51单片机系统中，有单周期指令、双周期指令和四周期指令。

○ a && b && c

只有a为真（非0）才需要判断b的值；
只有a和b都为真，才需要判断c的值。

```
int a,b,c,d;
a = 0;
b = 1;
c = 2;
d = a++ && b++ && --c;
```

因为a++是先判断a的值再自加，而a初始值为0，所以（a++）为假，由短路现象可知&&后面式子b++和--c就都不会执行；对于赋值语句，是先将a的值赋值给d，然后再自加，所以d的值为0，a最终为1。

```
root@zh:/home/book/code# ./a.out
a=1 b=1 c=2 d=0
```

○ a || b || c

只要a为真（非0）就不必判断b和c；
只有a为假，才需要判断b的值；只有a和b都为假，才有必要判断c的值。

```
a = 0;
b = 1;
c = 2;
d = a++ || b++ || --c;
```

因为a++是先判断a的值再自加，而a初始值为0，所以（a++）为假，由短路现象可知，还需要继续判断||后面的表达式b++,b++要先判断b的值，b为1，所以b++为真，由短路现象可知，后面的式子--c就不在执行；对于赋值语句，不再是将a的值赋值给d，而是将b先赋值给d然后a和b再自加，所以d的值为1，a最终为1，b最终为2。

```
root@zh:/home/book/code# ./a.out
a=1 b=2 c=2 d=1
```

○ 内存、地址、指针

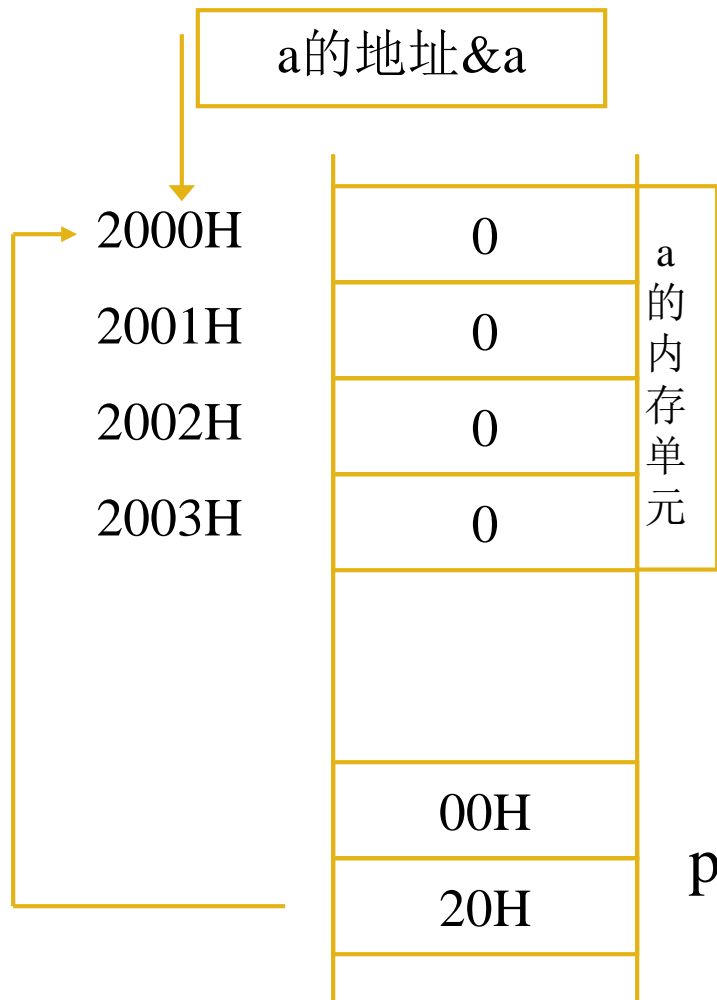
内存存放了计算机正在运行的程序和程序正在使用的数据。内存的基本单元是字节(Byte)。

为了访问内存单元，给每个内存单元一个编号，该编号称为该内存单元的地址。

变量是程序中可以改变的量，当说明变量时，系统将为其在内存中开辟相应得内存单元。由此确定变量的地址及内存中的表示方式。

```
int a=0;
```

如果有一变量p，其内容存放了a的地址&a，通过p也可实现对a的访问，p称为指针，并指向a。



○ 指针与一维数组

数组是同类型的变量的集合，各元素按下标的特定顺序占据一段连续的内存，各元素的地址也连续，指针对数组元素非常方便。

通过指针引用数组元素可以分以下三个步骤：

- (1)说明指针和数组 `int *p,a[10];`
- (2)指针指向数组 `p=a; /*指向数组的首地址*/`
 `p=&a[0]; /*指向数组的首地址*/`
- (3)通过指针引用数组元素

当指针指向数组的首地址时，则下标为*i*的元素地址为：

$p+i$ 或 $a+i$

引用数组元素可以有三种方法：

- 下标法： `a[i]` 注意：数组名是地址常量，不能改变！
- 指针法： `*(p+i)`
- 数组名法： `*(a+i)` `a=p; /*Error!*/`

○ 结构体的概念

在数据中，经常有一些既有联系，类型又不同的数据，并且它们之间又有一定的相关性，需要一起处理。

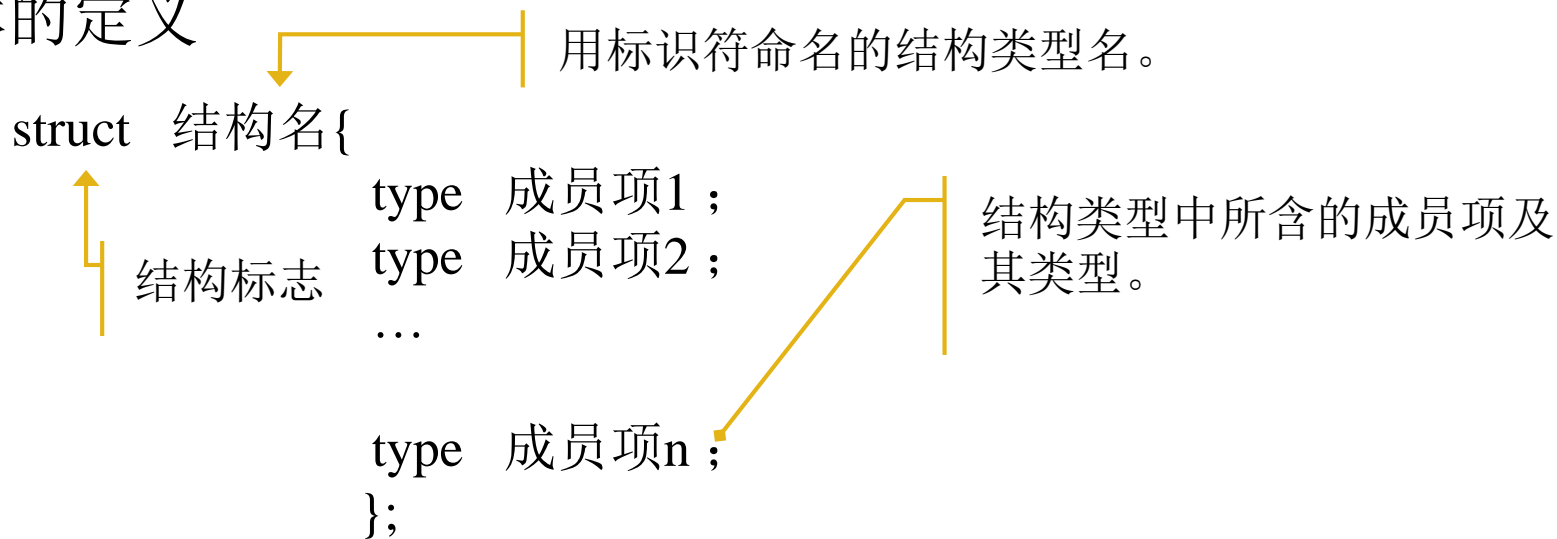
如：学生基本档案的数据

字段： 学号 姓名 性别 地址 分数

类型： long char char char float

C语言允许用户按自己的需要将不同的基本类型构造成一种特殊类型，即结构。

○ 结构体的定义



○ 结构变量的引用

结构变量都是以成员项作为引用单位,引用方式:

结构变量名.成员项名

说明:

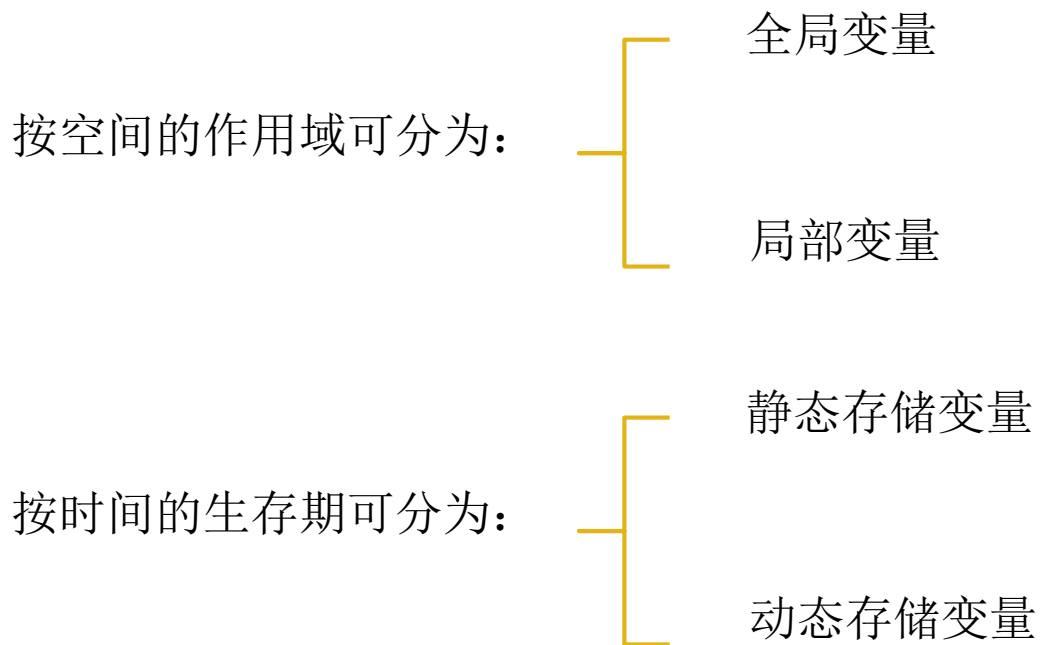
结构变量的成员项与普通变量有相同的性质。

结构变量可以相互赋值，如： `zhang = wang;`
则wang的所有成员项的值赋给了zhang的对应成员项。

```
struct student{                                wang.score=100;
    long num ;
    char name[20];
    int age ;
    float score ;
    char addr [30] ;
} wang={99010101 ,“王五” , 20 , 90.5 ,“上海” } ,zhang;
```

○ 基本概念

变量是内存数据的抽象，即将内存地址、数据表示抽象成一个符号。此外，变量还有存储类型，存储类型确定了变量在时间上的生存期和空间上的作用域。

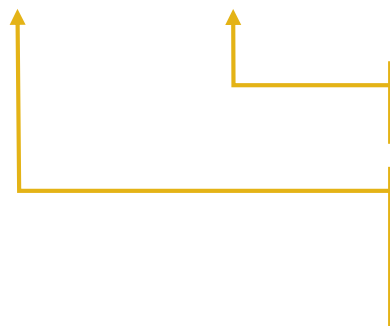


○ 四种存储类型

- (1)自动类型 `auto`
- (2)寄存器类型 `register`
- (3)静态类型 `static`
- (4)外部类型 `extern`

变量的说明格式：

存储类型 类型 变量名表；



确定变量在内存中的表示方法。

确定变量的生存期和作用域。
该项省略表示`auto`存储类型。

○ 四种存储类型：自动类型与寄存器类型（局部变量）

定义在复合语句的开始处。块内生存、块内有效。

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

```
void main(void)
```

```
{
```

```
    auto int a,b;
```

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

```
    if(b>a)
```

```
    {
```

```
        int iTemp;
```

```
        iTemp=a;
```

```
        a=b;
```

```
        b=iTemp;
```

```
    }
```

```
    printf("Max=%d",a);
```

```
}
```

iTemp的作用域

a
b
的
作
用
域

生存期：

执行到复合语句建立内存变量。执行出复合语句变量消亡。

register存储类型(局部变量)

作用域和生存期和auto相同，差别在于，如果CPU内部的寄存器空闲，则使用寄存器作为变量的存储单元，以提高速度。主要用于循环变量，且应该是整型和字符型。

○ 四种存储类型：局部static(静态)存储类型

作用域：在说明的复合语句内引用，出了复合语句不可见。

生存期：从程序开始运行直到程序结束，执行出 { } 时，原值并不消失，只是不能引用。

生存期从编译开始到程序结束。

的作用域

```
void row (void) ;  
void main ( void )  
{  
    int b ;  
    for (b=1 ; b<=9 ; b++ )  
        row ( ) ;  
}  
void row (void )  
{  
    static int a=1 ;  
    int b ;  
    for (b=1 ; b<=9 ; b++)  
        printf ( “%5d” , a*b ) ;  
    printf ( “\ n ” ) ;  
    a++ ;  
}
```

说明静态变量。

○ 四种存储类型：外部变量（extern存储类型、全局变量）

外部变量是定义在任何模块之外的变量。也称为全局变量。

作用域：从说明变量开始直到文件结束。

生存期：在程序的整个执行过程中。任何函数对外部变量的修改都会影响其他函数对外部变量引用时的值。

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

全局变量

作用域

○ 四种存储类型：外部变量（extern存储类型、全局变量）

(1) 全局变量可以通过说明改变其作用域。

```
void main()  
{  
  extern i;  
  ...  
}  
int i; /* i为全局变量*/  
void max(int a,int b)  
{  
  ...  
}
```

新的作用域

作用域

(2) 全局变量可以被不同的文件共享。

file1.c
static int a ;

file2.c
extern int a ;



文件1定义的外部变量



文件2通过说明使用文件1的外部变量


如果只希望在本文件中使用，可以加static说明

○ 四种存储类型：外部变量（extern存储类型、全局变量）

(3) 模块设计的原则：内聚性强，耦合性弱。全局变量的使用占用内存资源且增加模块的耦合性，因此，应尽量不使用全局变量。

(4) 当模块中出现和全局变量同名的局部变量时，局部变量在模块中优先。

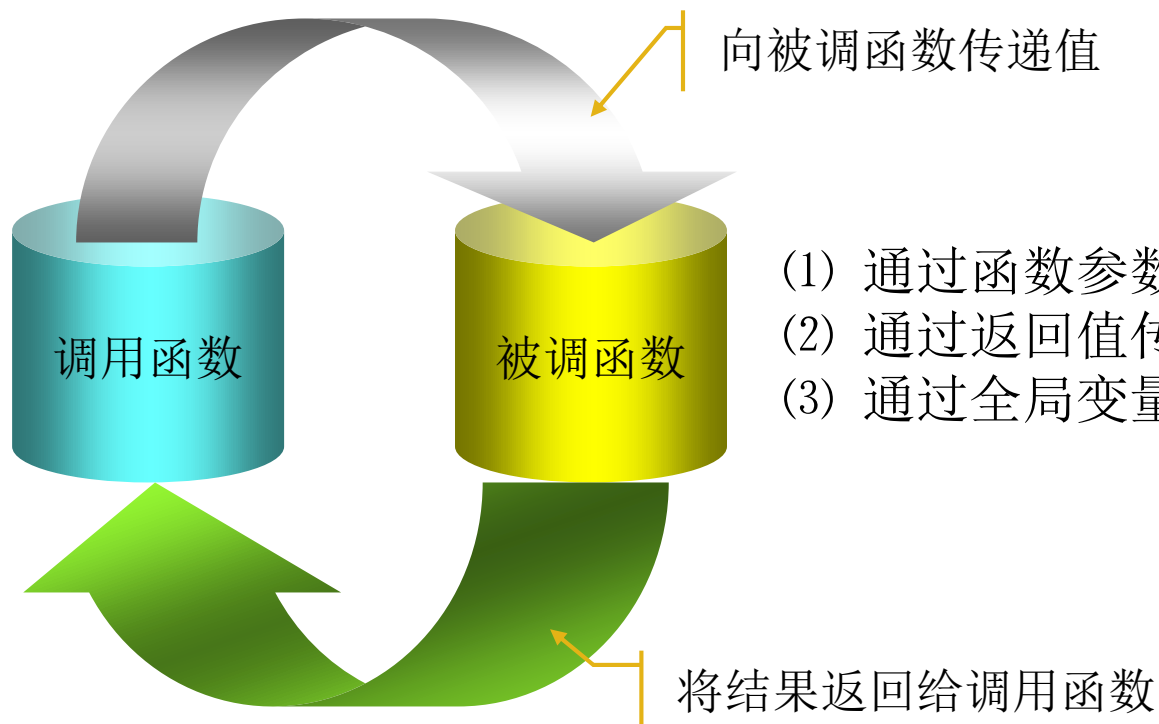
```
int a;  
void main(void)  
{  
    a=5;  
    ...  
}  
void fun(...)  
{  
    int a;  
    a=6;  
}
```



○ 基本概念

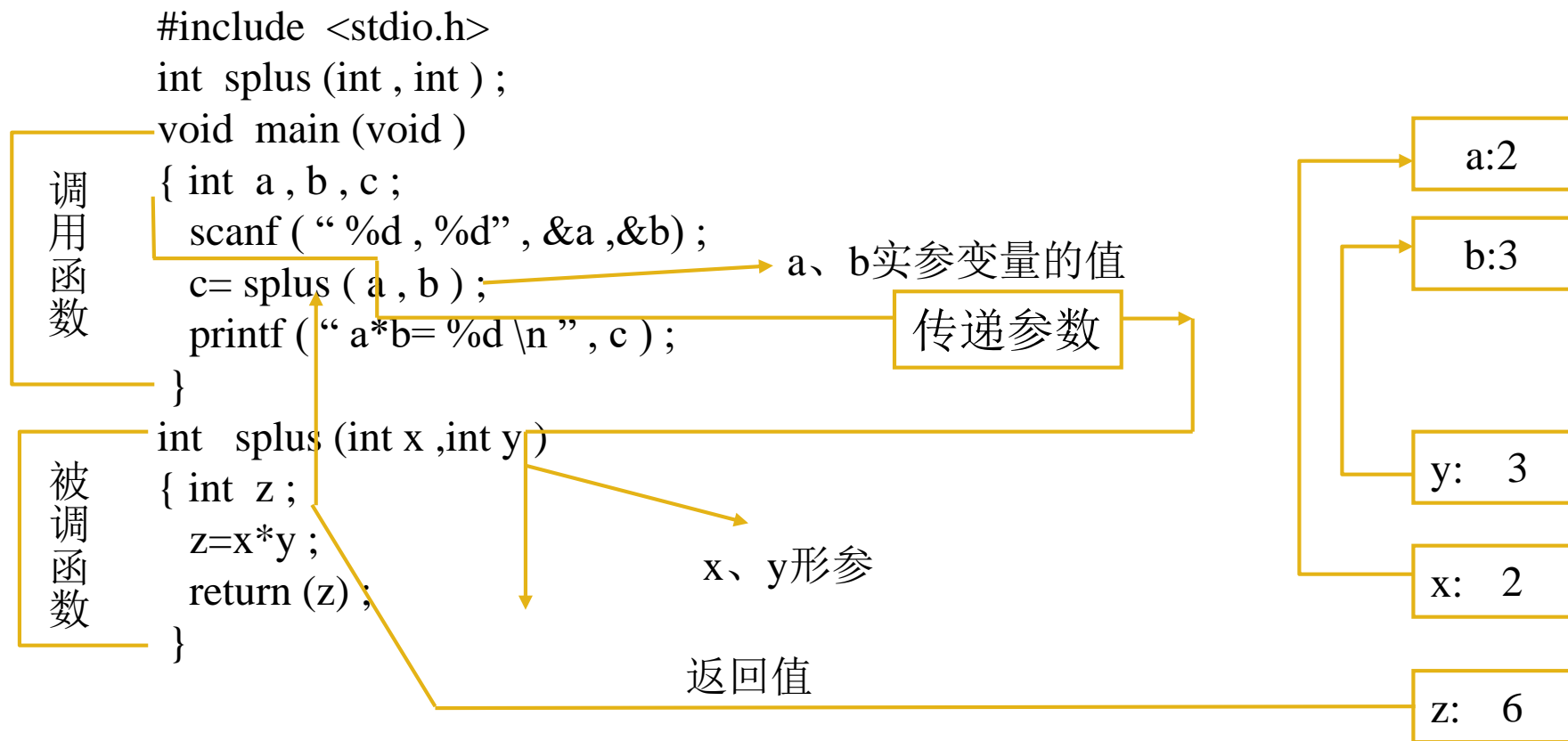
函数在调用的过程中，调用函数和被调函数存在数据的相互传递。数据的传递包括两个方面：

- (1)将值传递给被调函数；
- (2)将被调函数的结果返回给调用函数。



○ 使用参数传递数据：传值方式

通过实参与形参的结合，将数据值传递给形参，形参的改变不影响实参。

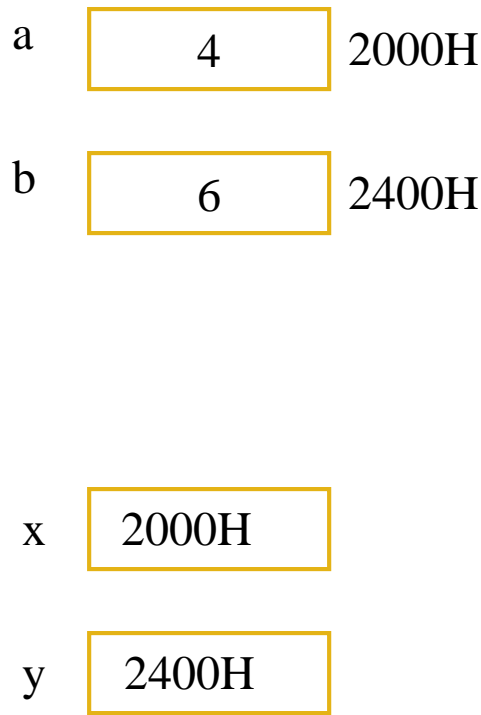


○ 使用参数传递数据：传地址方式

形参定义为指针，实参为变量的地址，被调函数通过地址可以修改地址对应的变量。

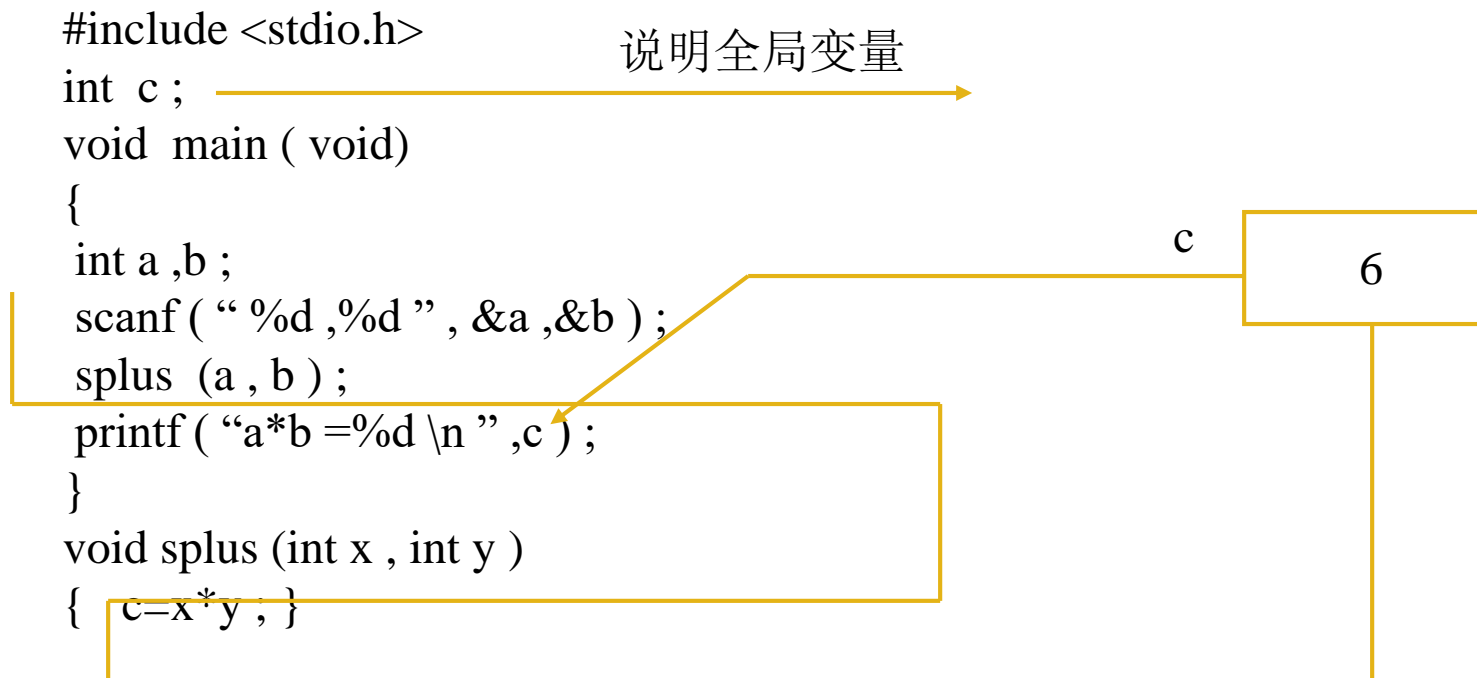
```
#include <stdio.h>
void swap ( int * , int * );
void main (void )
{
    int a , b ;
    scanf ( “%d , %d ” , &a , &b );
    if (a<b) swap ( &a , &b );
    printf ( “\n %d , %d \n ” , a,b );
}
void swap ( int *x , int *y )
{
    int t ;
    t=*x ; *x=*y ; *y=t ;
}
```

用函数实现两个变量a、b值的交换



○ 全局变量传递方式

全局变量可以被调用函数和被调函数共享，任何函数对全局变量的修改都会影响到其他函数所见的全局变量的值。



说明：应尽量少用全局变量，应使函数内部的内聚性强，函数之间的偶合性弱。

- 雷炜轩. 单片机C语言基础[EB/OL]. [2020-11-08].
- 宣善立. 程序设计基础[EB/OL]. [2020-11].
- 普中科技. 普中51单片机开发攻略[EB/OL]. [2019-09].
- 郭天祥. 新概念51 单片机 C语言教程:入门提高开发拓展全攻略 (第2版) [M]. 电子工业出版社, 2018.