C51 语言基础

一、标识符与关键字

变量名、函数名等都是标识符。标识符最大长度为 32 个字符。标识符只能由字母、数字、下划线构成，且第一个字符必须是字母或下划线。C51 语言对大小写字母敏感，如 max 与 MAX 是完全不同的两个标识符。

标识符的命名必须有实际意义，尽量做到通过其名称就知道它的类型及作用。通常用几个英文单词连在一起，每个单词的首字母大写，来表示变量名和函数名。在变量名前还要加上说明变量数据类型的前缀（见表一）。例如：ucKeyboardInputData、ChangeHexToAskii()。

关键字又称保留字，被 C51 编译器占用。我们自行定义的变量名和函数名不能与之相同。在 Keil 中编辑 C51 文件时，那些“蓝色”的单词就是关键字。

表一：

数据类型	表示方法	长度	表示数值的范围	前缀缩写
无符号字符型	unsigned char	1 字节	0～＋255	uc
有符号字符型	signed char	1 字节	-128～＋127	sc
无符号整型	unsigned int	2 字节	0～＋65535	ui
有符号整型	signed int	2 字节	-32768～＋32767	si
无符号长整型	unsigned long	4 字节	0～＋4294967295	ul
有符号长整型	signed long	4 字节	-2147483648～＋2147483647	sl
浮点型(小数型)	float	4 字节	±1.175494E-38 ～ ±3.402823E+38	f
位型	bit	1 位	0 或 1	b
指针形	*	1～3 字节	对象的地址	p

二、数据类型及变量

1、为什么要将变量定义成不同的数据类型呢？（数据类型见表一）

因为单片机处理数据类型小的变量比处理数据类型大的变量快，处理无符号变量比处理有符号变量快。为了提高程序执行效率及系统响应速度，应尽可能使用数据类型小的无符号变量。数据处理速度的决定因素有两个，数据类型是其中之一。

数据类型的关键问题有三点：

第一， 要知道实际数据的值是否超过该类型变量的表示范围。比如物料检测系统，需要定义一个变量对物料进行计数，物料可能的最大量为 5000，unsigned char 类型显然不能满足要求，该变量最合适的数据类型应该是 unsigned int。

第二， 数据类型转换。C51 能对不同的数据类型作隐式转换。例如，有几个不同数据类型的变量同时参加运算，C51 会自动先将那些低级类型的数据转换成高级类型，然后再运算处理。数据类型级别由低到高的顺序为：bit → char → int → long → float。

第三， 在多个变量参与数学运算的时候，运算结果的数据类型是参加运算的所有变量中级别最高的那种，可我们经常会遇到运算结果超出了级别最高的那种数据类型所能表示的数据范围。这就要求我们根据运算结果对变量类型进行临时性强制转换，以解决计算过程中发生“溢出”的问题。数据类型强制转换的书写格式为：(要转换成的数据类型)(变量)。

2、变量定义的格式如下：

变量数据类型说明 变量存储位置说明 变量名

变量一定是定义在 RAM 中。变量在 RAM 中的位置，即变量的存储位置是决定数据处理速度的另一个因素。单片机中变量的存储位置有 3 种，分别是 data 区、idata 区、xdata 区。在 data 区的变量处理最快，idata 区次之，xdata 区最慢。data 区对应单片机内部 RAM 的低 128 字节，idata 区对应单片机内部 RAM 的高 128 字节，xdata 区对应单片机外部扩展的 RAM。例：

```
unsigned char  xdata  abc1; //在 xdata 区定义一个名为 abc1 的无符号字符型变量。
       float  idata  abc3; //在 idata 区定义一个名为 abc3 的浮点型变量。
unsigned int   data  abc2; //在 data 区定义一个名为 abc2 的无符号整型变量，注意，
                           data 关键字可以省略。
```

3、变量的作用范围

变量按作用范围可分为局部变量和全局变量两种。

局部变量是指在函数体内部定义的变量，必须在函数体的最前面进行定义。局部变量只能在定义它们的函数内部使用，在该函数外部不能使用。

全局变量是指在所有函数体之外定义的变量。只有从全局变量定义位置之后书写的函数才能使用这些全局变量，之前的函数则不能使用。如果把全局变量写在程序的最前面，则所有函数就都能使用它们。

全局变量占用固定的 RAM 单元，其值永远不被释放。如果全局变量定义多了，则单片机程序的堆栈深度会降低，可能导致程序无法执行。另外，全局变量可以在多处被使用和修改，必须在程序的全局范围内考虑其值的变化，控制难度很高，容易出错。所以全局变量要少用和慎用。全局变量的作用相当巨大，尤其是在处理中断函数时。中断函数不允许有形参和返回值，所以只有通过全局变量才能建立主函数同中断函数的联系。

记住，一定不要定义同名的全局变量和局部变量！

4、将变量定义在绝对地址上

如果不指定变量在外部 RAM 中的绝对地址，则变量从 0x0000 地址开始定义。很多时候外部 RAM 的地址范围并不从 0x0000 开始，如何将变量定义到指定的 xdata 地址区域呢？使用外部 RAM 的绝对地址访问说明“_at_”。例如：

```
unsigned char xdata abc1       _at_ 0x1b24;
unsigned int  xdata abc2[100] _at_ 0x1b25;
unsigned char xdata abc3       _at_ 0x1bed;
```

三、宏定义和常量

宏定义的格式为：**#define 宏名 宏内容**

通常宏内容都是一些不便书写或意思模糊的内容，而宏名则简练直观。在需要书写宏内容的地方用宏名进行书写替代，编译程序时，编译器会自动用宏内容替换所有的宏名。使用宏定义的目的就是让程序的书写清晰简练。例如：

```
#define  PI   3.1415926
#define  CLOSE  0
#define  UCHAR  unsigned char
#define    MAX_DATA  8
```

常量分数字常量、字符常量、字符串常量三种。123、-456、0、0x89、2048L、0.314、

1.414 等是数字常量。‘a’、‘M’、‘7’等是字符常量，字符常量是用单引号括起来的单个字符。“ABcd”、“$1234”等是字符串常量，字符常量是用双单引号括起来的一个或几个字符。

C51 将字符串常量作为一个字符数组来处理，在存储字符串常量时，会在字符串的尾部自动加一个转义字符\0 作为该字符串常量的结束符。\0 表示空字符 NULL，其值就是 0。字符串在内存中占有的字节数为实际字符数加 1,即使是空字符串“”,它在内存中也占 1 个字节。只能用字符数组来存储字符串。

字符常量在数值上等于其 ASKII 码值，如 i=‘A’、i=0x41、i=65 这三个语句是等价的。

在串口数据传输、液晶显示等操作时，经常会用到字符常量和字符串常量。注意，‘A’是字符常量，“A”是字符串常量，二者是完全不同的。我们通常将变量定义在程序存储区做常量使用，例如：char code CHAR_ARRAY[]={“ABCDE”}。

宏和常量的命名规则是：单词的字母全部大写，各单词之间用下划线隔开。

四、特殊功能寄存器的 C51 定义 —— sfr、sbit

对 89S52 单片机进行 C51 编程时，必须将 REG52.H 这个“头文件”包含进来。REG52.H 头文件里面没有任何函数,该头文件只是用 sfr 和 sbit 声明了单片机的所有特殊功能寄存器及可以位操作的特殊功能寄存器位。例如：

```
sfr TCON  = 0x88;
sfr TMOD  = 0x89;
sbit TR0  = TCON^4;
```

REG52.H 声明的特殊功能寄存器的名称与单片机教材里的完全一样，所以只要看懂教材就能轻松设置单片机的特殊功能寄存器。例如：

```
TMOD = 0x21;
TR0 = 1;
```

注意，P0、P1、P2、P3 这 4 个端口状态特殊功能寄存器虽然也能位操作，但 REG52.H 头文件并没有对这 32 位进行定义，使用时要自己定义，如：

```
sbit  LED_OUT = P1^2;
sbit  KEY_IN1 = P2^7;
```

定义之后，就可以通过 LED_OUT、KEY_IN1 这些端口位变量来控制单片机管脚输出高低电平或读取管脚的输入电平状态。用 sbit 对端口位进行定义,是操作单片机端口位的唯一办法。

五、运算符

1、赋值运算符： =

2、算术运算符：加 +、减 -、乘 *、除 /、取余 %

如果两个整数相除，结果为整数，小数部分被舍弃。

取余%是两数相除取余数，注意，这两个数必须都是整数。

3、增量、减量运算符：自加 + +、自减 - -

如 i++、i--、++i、--i 等。由于++(--)的前后位置不同，其运算过程也不相同。++i（或--i）是先执行 i=i+1(或 i=i-1)然后再使用 i 的值；i++（或 i--）是先使用 i 的值然后再执行 i=i+1(或 i=i-1)。例如：

```
i = 2; j = i++ * 3; //此时 j = 6
```

```
i = 2; j = ++i * 3; //此时 j = 9
```

4、关系运算符：大于> 、小于< 、大于等于>= 、小于等于<= 、等于== 、不等于!=

如 i>=8 、(i-1)!=0 、i==3 、i<=j 、(i+2)>(j*3) 等都是关系表达式。关系表达式的结果只有 0 或 1 两种值。当所指定的条件满足时，结果为 1，条件不满足时，结果为 0。

注意，在编程时不要将关系运算符“==”与赋值运算符“=”弄混！

5、逻辑运算符：逻辑或 || 、逻辑与 && 、逻辑非 ！

条件为非零就表示条件真，条件为 0 表示条件假。逻辑表达式的结果只有 0 或 1 两种值。逻辑运算的一般形式为：

- 逻辑与：条件 1 && 条件 2 …… && 条件 n 。只有当所有条件都为“真”时，逻辑与的结果为 1，有任何一个条件为“假”时，逻辑与的结果为 0。
- 逻辑或：条件 1 || 条件 2 …… || 条件 n 。有任何一个条件为 “真”时，逻辑或的结果为 1，只有当所有条件都为“假”时，逻辑或的结果才为 0。
- 逻辑非：!条件

6、位操作运算符：

位操作运算并不改变参与运算的变量的原值，如果想保留运算结果，则应利用赋值运算。不能对浮点型数据进行位操作。

- << 左移：i=0x36; i=i<<2; //此时 i 的值为 0xd8
- >> 右移：i=0x36; i=i>>1; //此时 i 的值为 0x1b
- & 按位与：i=0x47; j=0x58; z=i&j; //此时 z 的值为 0x40
- | 按位或：i=0x47; j=0x58; z=i|j; //此时 z 的值为 0x5f
- ~ 按位取反：i=0x36; i=~i; //此时 i 的值为 0xc9

7、复合赋值运算符：+= 、-= 、*= 、/= 、%= 、<<= 、>>= 、&= 、|= 、!=

```
i += 8   等价于 i = i+8
i *= j+5 等价于 i = i*(j+5)
```

当一个表达式中使用了多种运算符时，为了实现编程者的目的有必要用括号直接规定先做什么再做什么，而不是依靠运算符自身的优先级和结合性！

六、基本语法

在表达式后面加上一个分号就构成一条语句。用{}扩起来的多条语句叫复合语句。只由一个分号构成的语句叫“空语句”。

1、 条件语句

- **if(条件表达式) 语句**

含义：只有 “条件表达式”为真（非 0 值）才执行后面的语句，否则不执行。“语句”可以是一条，也可以是复合语句。后面遇到的“语句”都是这个意思。

- **if(条件表达式) 语句 1**

  else 语句 2

含义：如果“条件表达式”为真（非 0 值）执行语句 1，否则执行语句 2。

- if(条件表达式 1) 语句 1
  else if(条件表达式 2) 语句 2
  …
  else if(条件表达式 m) 语句 m
  …
  else if(条件表达式 n) 语句 n
  else 语句 p

含义：从“条件表达式 1”开始顺次向下判断，当遇到为真的那个条件表达式，如“条件表达式 m”，执行语句 m，之后不再判断余下的条件表达式，程序直接跳转到“语句 p”后面。如果所有的条件表达式没有一个为真，则执行“语句 p”。

2、开关语句

```
switch (表达式)
    {
      case  常量 1：语句 1
                  break;
      case  常量 2：语句 2
                  break;
      …
      case  常量 m：语句 m
                  break;
      …
      case  常量 n：语句 n
                  break;
      default：语句 p
    }
```

含义：用表达式的值同“常量 1”到“常量 n”逐个比较，如果表达式的值与某个常量相等，假设与“常量 m”相等，则执行“语句 m”，然后由 break 语句控制直接跳出 switch 开关。如果没有一个常量与表达式相等，则执行“default：语句 p”，然后结束 switch 开关。

3、循环语句

- while(条件表达式) 语句

含义：如果“条件表达式”为真，就重复执行后面的语句，直到“条件表达式”为假，则退出循环。

- for(变量赋初值；循环条件；变量更新) 语句

含义：常用 for 循环构建规定次数的循环。例如 for(i=0；i<10；i++) {语句}，“语句”将执行 10 遍。

4、其它语句

- 绝对跳转语句：goto

```
标号:
语句 1
…
语句 n
…
goto  标号;
```

- **返回语句：return**

return(表达式或变量);

含义：它是函数体的最后一条语句，控制函数结束。return 后面的括号中的表达式的值或变量就是函数的返回值。如果函数无返回值，则直接写“return;”即可。“return;”可以省略，在编译时，编译器会为该函数自动加上。

- **退出语句：break 和 continue**

含义：在循环体中如果执行了 break 语句，则直接跳出循环体，如果执行了 continue 语句，则 continue 后面的语句被全部跳过，循环体又重新从第一条语句开始执行。

七、函数

1、函数的定义

函数定义格式如下：

```
返回值的数据类型  函数名（形参变量 1 说明， 。。。 ，形参变量 n 说明）
    { 局部变量定义
      函数体语句
      返回语句
    }
```

一个函数包括如下几个要素：返回值，函数名，形式参数变量，函数体。形式参数变量就是该函数的局部变量，在该函数被调用时，主调者会将具体数据传递给形式参数变量，也就是说没发生函数调用之前，这个函数的形式参数变量没有实际意义。形式参数变量的声明同定义普通变量一样，格式为：**形参变量数据类型　形参变量存储位置　形参变量名称**。

在函数体里定义的变量叫局部变量，局部变量只能在定义它们的函数体里使用，局部变量必须紧跟在函数体开始大括号“{”之后定义。函数体的最后一句是 return,用来处理函数的返回值。一个函数可以没有形式参数变量和返回值，定义格式如下：

```
void  函数名（void）
    {
      函数体语句
    }
```

2、主函数

用户程序里必须且只能有一个名字叫 main 的主函数，main 函数不能有形参和返回值，main 函数不能被其它函数调用。main 函数的函数体是设计者根据自己的需要编写的主程序，在主程序里可以调用其它子函数（不包括中断函数）。在单片机上电后，用户程序总是从 main 函数体的第一条语句开始执行。主函数的书写格式为：

```
void  main (void)
```

```
{
 语句
}
```

3、中断函数

如果使用单片机的中断功能，就需要编写中断函数。中断函数同样不能有形参和返回值，也不能被其它函数调用。中断函数可以调用其它函数，因为涉及再入问题，使用时要十分小心，尽最大可能不在中断函数里调用其它函数。中断函数和其它子函数的名字可以任意起，如何区别中断函数和子函数呢？用 interrupt 关键字。例：

```
void  T0_interrupt (void)  interrupt 1
```

T0_interrupt 是中断函数名（虽然可以任意起，但最好具有说明性），interrupt 关键字后面的数字“1”决定了该函数是 T0 定时器中断函数，这个“1”叫中断号，中断号与中断函数的对应关系见下表。如果没有“interrupt 1”，则为普通函数。

中断源名称	中断号
外部中断 INT0	0
定时器/计数器 T0 中断	1
外部中断 INT1	2
定时器/计数器 T1 中断	3
串行口中断 UART	4
定时器/计数器 T2 中断 T	5

4、自行定义的子函数

除 main 和中断函数之外的其它函数叫子函数，用户程序中子函数的数量没有限制。

函数之间的调用是否允许与书写顺序有关，即书写顺序在前的函数可以被写在它后面的函数调用。子函数可以被 main 函数和中断函数调用，子函数之间也可以相互调用。当子函数很多时，如何快速整理子函数顺序呢？不用整理。只要在程序最前面对所有的子函数说明一下就可以了，而函数说明是没有先后顺序要求的。

子函数最主要的任务是将功能模块化。比如 main 函数中多处需要将单个数字 0～9 转换成对应的 ASKII 码，那么我们就需要编写一个 ASKII 码转换程序，在需要的地方调用。这样写使得程序简练、清晰，便于分析主程序的逻辑。

5、函数调用

在调用函数的时候，主调者将实际的数据（实参）赋给形式参数变量，被调函数的函数体对形式参数变量进行规定的运算，最后将计算的结果作为返回值提供给主调者。

参数的传递可分为值传递和地址传递。值传递，就是形参为普通变量；地址传递，就是形参为数组或指针。下面我们通过程序说明值传递：

```
int TwoDataAdd(char FirstData, char SecondData)
    {
      int Result ;
      Result = FirstData + SecondData ;
      return (Result) ;
    }
```

在 main 函数里调用 TwoDataAdd 函数：

```
Void  main(void)
    {
      char a,b;
      int c;
      a = 38;
      b = 99;
      c = TwoDataAdd (a, b);
    }
```

变量 a、b 叫实参，调用 TwoDataAdd ()函数时，实参 a、b 的值分别传递给形参 FirstData 和 SecondData，用变量 c 接收函数的返回值。值传递的点是，在函数调用结束后，实参仍旧保持原来的值，即 a = 38、b = 99 ，而形参被释放。我们再通过程序说明地址传递：

```
void DataCompositor(char Array[], char Number)
    {
      char TemporaryData;
      char i, j;
      for(i=0; i< Number; i++)
          { for(j=i+1; j< Number; j++)
                { if(Array[i]< Array[j])
                      { TemporaryData = Array[i];
                        Array[i] = Array[j];
                        Array[j] = TemporaryData;
                      }
                }
          }
    }
```

在 main 函数里调用 DataCompositor()排序函数：

```
void main(void)
{
  char FourDataArray[4]={4, 7, 1, 2};
  DataCompositor (FourDataArray, 4);
}
```

上面的子程序用数组做形参,在实际调用时只需将实参数组名传递给形参数组,“数组名”就是该数组第一个元素的存储地址，地址传递给形参后，形参数组也指向了实参数组的实际存储地址，子函数对形参数组排序实际上就是对实参数组进行排序。当然我们也可以这样定义子函数：`void  DataCompositor(char *Array, char Number)`，执行结果是一样的。

函数通过 return 语句只能返回一个数据,通过数组的地址传递方式可以修改一系列同类型的数据，如果我们要操作且返回多个不同类型的变量怎么办呢？使用全局变量。全局变量使用起来是最方便的，只不过全局变量可以在多处被修改，使用时要相当细心。

6、库函数

Keil C51 编译软件包含 10 个标准函数库，如果我们要使用某个函数，就必须把该函数所在的库包含进来。这 10 个函数库如下表：

函数库	对应的头文件	功能
字符函数	**CTYPE.H**	与 ASKII 码表相关
一般 I/O 函数	**STDIO.H**	与 UART 相关
字符串函数	**STRING.H**	字符串的截取、查找、比较等
标准函数	**STDLIB.H**	字符串与数字之间的转换
数学函数	**MATH.H**	求绝对值、平方开方、三角函数
绝对地址访问	**ABSACC.H**	绝对地址访问
内部函数	**INTRINS.H**	只有_NOP_()函数有用，相当于汇编里的 NOP
变量参数表	**STDARG.H**	不用
全程跳转	**SETJMP.H**	不用
SFR 访问	**REG51/52.H**	特殊功能寄存器声明

我们需要自己浏览一下每个函数库中各种函数的功能，当实际工作中需要编写某功能函数时要知道是否有现成的，避免重复劳动。

八、数组和指针

1、 数组

定义一个数组：int idata abc[8];

表示在 idata 区定义一个长度为 8 的整型数组，这个数组具有 8 个元素，abc[0]～abc[7]，每个数组元素可存放整型数，且每个单元在 RAM 中连续排列，数组占用 16 字节 RAM 单元。注意，数组的下标总是从[0]开始的。数组定义中的常数“8”表示数组长度，定义时不能用变量或表达式来表示数组长度。如果在定义数组的同时对数组进行初始化则长度数字可以省略，如下定义的数组其长度是 5 字节：char abc[] = {1,0,-5,8,2};

数据类型为字符型的数组叫字符数组，通常用字符数组来存储和处理字符串。只有在字符数组定义时可以用字符串一次性的对其赋值：

usigned char CharacterArray[]={“ABCDEF”};

在除此之外的其它任何地方，只能通过下标来访问字符数组中的各个元素。

数组的意义在于它为批量处理同一类型的数据提供了方便。例如，用 AD 连续采样 20 次然后做均值算法。字符数组在处理串口接收和发送时相当有用，驱动液晶显示也会用到字符数组。

2、 指针

指针是存储数据地址的变量，指针只能存放地址。指针的定义为：

指针指向数据的数据类型 指针指向数据的位置 * 指针变量名

例如：char code * abc1; //定义一个指向 code 区的 char 型数据的指针 abc1。

int xdata * abc2; //定义一个指向 xdata 区的 int 型数据的指针 abc2。

如何使用指针呢？例如：

```
char data *abc3; //定义指向内部 RAM 中的 char 类型数据的指针 abc3
char i;          //定义一个 char 类型变量 i
abc3 = & i;      //取变量 i 的地址赋给指针
*abc3 = 0x89;    //通过指针操作数据，等价于 i=0x89
```

3、数组和指针的关系

指针和数组关系密切，可以用指针来操作数组。数组名表示数组第一个元素的地址，即

数组的起始地址。可以将数组名赋给指针，然后用指针来操作数组。例如：

```
char abc1[5]；//定义数组
char *abc2；  //定义指针
abc2 = abc1； //将数组起始地址赋给指针，数组名就是数组的起始地址
abc2[3] = 98；//指针变量可以作为数组使用，将第四个数组元素赋值为 98，等价于
                abc1[3] = 98；
abc2 += 4；   //可以对指针变量进行加减运算来移动指针指向的位置，不能对数组名
                进行自加或自减运算。
*abc2 = 36；  //等价于 abc1[4] = 36；
```

沈阳单片机开发网