第 3 章　C51 程序设计

3.1　标准 C 语言与 C51 语言

标准 C 语言与 C51 语言在数据运算操作、程序流程控制语句及函数的使用上基本没有差别，但从适用于嵌入式计算机的角度出发，标准 C 语言与 C51 语言在以下几个方面不一样：

(1) 库函数进行了调整。C51 语言去掉了标准 C 语言中不适合嵌入式计算机的库函数，如图形函数，改进了部分函数，如标准输入输出函数 scanf()和 printf()，这两个函数在标准 C 语言中用于字符输入和屏幕输出，而在 C51 语言中用于串行口收发。

(2) 数据类型有所增加。C51 语言在标准 C 语言基础上增加了 4 种针对 51 单片机特有的数据类型，如位类型，以实现丰富的位操作，从而更好地完成控制功能。

(3) 存储模式不一样。C51 语言中变量的存储模式与 51 单片机的存储器紧密相关。

(4) 输入输出处理不一样。C51 语言中的输入输出是通过 51 单片机的串行口完成的，输入输出指令执行前必须对串行口进行初始化。

(5) C51 语言相对标准 C 语言设置了有利于嵌入式计算机控制的专门的中断函数。

3.2　C51 的数据类型

C51 的数据类型分为基本数据类型和构造数据类型，情况与标准 C 中的数据类型基本相同，其中 char 型与 short 型相同，float 型与 double 型相同。

C51 的基本数据类型有：**字符型 char、整型 int、长整型 long、浮点型 float、指针型 *、特殊功能寄存器型 sfr/sfr16、位型 bit/sbit**。构造数据类型有：数组、指针、结构、联合、枚举。C51 基本数据类型见表 3.1。表中后 4 种是 C51 相对标准 C 增加的特殊功能寄存器型和位型数据类型，不能使用指针存取。

表 3.1　C51 基本数据类型

基本数据类型	长度	取 值 范 围
unsigned char	1B	0～255 无符号字符型
signed char	1B	－128～＋127 有符号字符型
unsigned int	2B	0～65 535 无符号整型
signed int	2B	－32 768～＋32 767 有符号整型
unsigned long	4B	0～4 294 967 295 无符号长整型
signed long	4B	－2 147 483 648～＋2 147 483 647 有符号长整型

续表

基本数据类型	长度	取 值 范 围
float	4B	±1.175 494E-38～±3.402 823E+38 浮点型
*	1～3B	指针型
bit	1bit	0 或 1　位型变量
sbit	1bit	0 或 1　可位寻址的特殊功能寄存器的某位的绝对地址
sfr	1B	0～255 单字节特殊功能寄存器型
sfr16	2B	0～65 535 双字节特殊功能寄存器型

C51 程序在运算过程中可能出现各个运算量的数据类型不一致的情况。C51 允许运算中运算量的数据类型的隐式转换，隐式转换的优先级顺序如下：

```
bit→char→int→long→float
signed→unsigned
```

例如，当 char 型与 int 型运算量进行运算时，先自动将 char 型扩展为 int 型，然后与 int 型进行运算，运算结果为 int 型。C51 除了支持隐式类型转换外，还可以通过强制类型转换符“()”对数据类型进行人为的强制转换。如：

【例 3.1】 数据类型强制转换。

```
(float)x                    //将 x 强制转换成 float 型
(int)(x+y)                  //将 x+y 的和强制转换成 int 型
```

3.3　C51 的运算量

3.3.1　常量

常量是指在程序执行过程中其值不能改变的量。C51 支持整型常量、浮点型常量、字符型常量和字符串型常量。

1. 整型常量

整型常量就是整型常数，包括：

(1) 十进制整数，如 0、985、－51 等十进制数。

(2) 十六进制整数，以 0x 开头表示，如 0x16、0x1234 分别表示十六进制数 16H、1234H。

2. 浮点型常量

(1) 浮点型常量就是实型常数，包括：

(2) 十进制浮点数，如 0.123、56.78 等都是十进制数形式的浮点型常量。

指数浮点数，如 985.211e-2、－3.1415e3 等都是指数形式的浮点型常量。

3. 字符型常量

字符型常量是用单引号‘’引起来的字符，包括：

(1) 可以显示的 ASCII 字符,如‘A’、‘a’、‘3’等。

(2) 不可显示的控制字符。在其前面加上反斜杠“\”组成转义字符。利用它可以完成一些特殊功能和输出时的格式控制。常用的转义字符见表 3.2。

表 3.2 常用的转义字符

转义字符	含义	ASCII 码(十六进制数)
\0	空字符(NULL)	00H
\n	换行符(LF)	0AH
\r	回车符(CR)	0DH
\t	水平制表符(HT)	09H
\b	退格符(BS)	08H
\f	换页符(FF)	0CH
\\	反斜杠	5CH

4. 字符串型常量

字符串型常量是用双引号“ ”引起来的字符组。字符串常量与字符常量不一样。

用双引号“ ”括起来的一串字符称为字符串型常量,如“Sanxia”“5678”等。C 编译器会自动在字符串结尾加上转义字符“\0”作为字符串结束符。

用单引号‘ ’括起来的字符型常量,其值实际上是字符的 ASCII 码,而不是字符串,如‘A’表示 A 的 ASCII 码值为 49;“A”表示一个字符串,而不是一个字符。

‘A’在内存中的存放为 49

“A”在内存中的存放为 49 0

其中 0 是 C 编译系统自动加上去的。

所以,字符常量‘A’只占 1B,字符串常量“A”实际占 2B。同理,字符串常量“ABCD”实际占 5B。

3.3.2 变量

变量是指在程序执行过程中其值可以改变的量。变量由两部分组成:变量名和变量值。

在 C51 中,变量在使用前必须对变量进行定义,指出变量的数据类型和存储模式,以便编译系统为它分配相应的存储单元。定义的格式如下:

```
[存储种类] 数据类型说明符 [存储器类型] 变量名 1[=初值],变量名 2[=初值]…;
```

1. 数据类型说明符

定义变量时,数据类型说明符是必需的。通过数据类型说明符指明变量的数据类型,指明变量在存储器中占用的字节数和取值范围。数据类型说明符可以是基本数据类型,也可以是构造数据类型。

为了增加 C51 程序的可读性,允许用户用 typedef 和 # define 为系统固有的基本数据类型说明符起别名,格式如下:

```
typedef  C51基本数据类型说明符  别名;
#define  别名  C51基本数据类型说明符;
```

定义别名后，就可以用别名代替数据类型说明符对变量进行定义。别名可以大写，也可以小写，为了区别，一般用大写字母表示。

【例 3.2】 typedef 和 # define 的使用。

```
typedef    unsigned char  INT8U;          //为无符号字符型取别名
#define    INT16U  unsigned int;          //为无符号整型取别名
INT8U      x1=0x12;
INT16U     x2=0x1234;
```

2. 变量名

变量名是 C51 区分不同变量，为不同变量取的名称，也是必需的。C51 中规定变量名可以由字母、数字和下画线三种字符组成，且第一个字母必须为字母或下画线。变量名有两种：普通变量名和指针变量名。它们的区别是指针变量名前面要带“*”号。

3. 存储种类

存储种类是指变量在程序执行过程中的作用范围。C51 变量的存储种类有四种，分别是自动(auto)、外部(extern)、静态(static)和寄存器(register)。

auto：使用 auto 定义的变量称为自动变量，其作用范围在定义它的函数体或复合语句内部，当定义它的函数体或复合语句执行时，C51 才为该变量分配内存空间，结束时占用的内存空间释放。自动变量一般分配在内存的堆栈空间中。定义变量时，如果缺省存储种类，则该变量默认为 auto 变量。

extern：使用 extern 定义的变量称为外部变量。在一个函数体内，要使用一个已在该函数体外或别的程序中定义过的外部变量时，该变量在该函数体内要用 extern 说明。外部变量被定义后分配固定的内存空间，在程序整个执行时间内都有效，直到程序结束才释放。

static：使用 static 定义的变量称为静态变量。它又分为内部静态变量和外部静态变量。在函数体内部定义的静态变量为内部静态变量，它在对应的函数体内有效，一直存在，但在函数体外不可见，这样不仅使变量在定义它的函数体外被保护，还可以实现当离开函数时值不被改变。外部静态变量(在函数外部定义的静态变量)在程序中一直存在，但在定义的范围之外是不可见的。例如，在多文件或多模块处理中，外部静态变量只在文件内部或模块内部有效。

register：使用 register 定义的变量称为寄存器变量。它定义的变量存放在 CPU 内部的寄存器中，处理速度快，但数目少。C51 编译器编译时能自动识别程序中使用频率最高的变量，并自动将其作为寄存器变量，用户无须专门声明。

4. 存储器类型

存储器类型用于指明变量处于单片机哪块存储器区域的情况。C51 编译器能识别的存储器类型有以下几种，见表 3.3。

定义变量时也可以缺省“存储器类型”，缺省时 C51 编译器将按编译存储模式默认存储器类型。C51 编译器支持三种编译存储模式：SMALL 模式、COMPACT 模式和 LARGE 模式。不同的编译存储模式对变量默认的存储器类型不一样。

表 3.3　C51 存储器种类

存储器类型	描　　述
data	片内 RAM 直接寻址区(00H～7FH),位于低 128B,直接访问,速度快
bdata	片内 RAM 位寻址区(20H～2FH),允许字节和位混合访问
idata	片内 RAM 间接寻址区,允许寄存器间接访问片内 RAM 全部的 256B
pdata	片外 RAM 低 256B,使用@Ri 间接访问
xdata	片外 RAM 全部 64KB,使用@DPTR 间接访问
code	程序存储器区 ROM 全部 64KB,使用 DPTR 访问

(1) **SMALL 模式(小编译模式)**。在 SMALL 模式下,编译时变量默认在片内 RAM 的低 128B 空间中,存储器类型为 data。

(2) **COMPACT 模式(紧凑编译模式)**。在 COMPACT 模式下,编译时变量默认在片外 RAM 的低 256B 空间中,存储器类型为 pdata。

(3) **LARGE 模式(大编译模式)**。在 LARGE 模式下,编译时变量默认在片外 RAM 的 64KB 空间中,存储器类型为 xdata。

在程序中,变量的编译存储模式的指定通过 # pragma 预处理命令实现。如果没有指定,则系统都默认为 SMALL 模式。随着内部 RAM 和 ROM 空间容量不断增大的增强型 51 单片机得到广泛应用,变量存储器类型的定义多默认为 data。

【例 3.3】 变量存储器类型的定义。

```
char data x1;            //片内 RAM 低 128B 空间用直接寻址方式访问的字符型变量 x1
int xdata x2;            //片外 RAM 64KB 空间用间接寻址方式访问的整型变量 x2
unsign char code x3;     //ROM 64KB 空间无符号字符型变量 x3
```

5. 特殊功能寄存器变量

51 单片机通过片内的特殊功能寄存器控制 I/O、中断、定时/计数器、串行口及其他功能部件,每个特殊功能寄存器在片内 RAM 中都对应一个或两个字节单元。

在 C51 中,允许用户对这些特殊功能寄存器进行访问,访问时须通过 sfr 或 sfr16 类型说明符进行定义,定义时须指明它们对应的片内 RAM 单元的地址。格式如下:

```
sfr 或 sfr16  特殊功能寄存器名 =地址;
```

sfr 用于对 51 单片机中单字节的特殊功能寄存器进行定义。sfr16 用于对双字节的特殊功能寄存器进行定义。特殊功能寄存器名一般用大写字母表示。地址一般用直接地址形式,具体的特殊功能寄存器见表 2.2。

【例 3.4】 特殊功能寄存器的定义。

```
sfr      P1=0x90;
sfr      SCON=0x98;
sfr16    DPTR=0x82;
sfr16    T1=0x8B;
```

6. 位变量

C51 位类型符有两个：bit 和 sbit。可以定义两种位变量。

bit 位类型符用于定义一般的可位处理的位变量。它的格式如下：

```
bit  位变量名;
```

位变量的存储器类型只能是 bdata、data、idata，只能是片内 RAM 的可位寻址区，严格来说只能是 bdata。

【例 3.5】 bit 型位变量的定义。

```
bit  data   x1;
bit  bdata  x2;
```

sbit 位类型符用于定义在可位寻址字节或特殊功能寄存器中的位，定义时须指明其位地址，也可以是位直接地址，可以是可位寻址变量带位号，还可以是特殊功能寄存器名带位号。格式如下：

```
sbit  位变量名 =位地址;
```

如位地址为位直接地址，则其取值范围为 0x00～0xff；如位地址是可位寻址变量带位号或特殊功能寄存器名带位号，则在它前面须对可位寻址变量或特殊功能寄存器进行定义。字节地址与位号之间、特殊功能寄存器与位号之间一般用“^”作间隔。

【例 3.6】 sbit 型位变量的定义。

```
sfr   P1=0x90;
sbit  P1_0=P1^0;
unsigned  char  bdata  reg;
sbit  reg_0=reg^0;
```

C51 编译器已经对 51 单片机常用的特殊功能寄存器和特殊位进行了定义，放在一个 reg51.h 或 reg52.h 的头文件中。reg52.h 头文件相对 reg51.h 只是增加了特殊功能寄存器 T2 和 DPTR1 的定义。使用时需用预处理命令把该头文件包含到程序中，然后就可以在程序中使用特殊功能寄存器名和特殊位名，其形式为

```
#include  <reg51.h>
```

3.4 C51 的运算符

3.4.1 算术运算符

C51 算术运算符及其说明见表 3.4。

表 3.4 C51 算术运算符及其说明

符　　号	说　　明	举例(设 x=7,y=3)
+	加法	z=x+y;　　//z=10

续表

符　　号	说　　明	举例(设 x=7,y=3)
－	减法	z=x-y;　　//z=4
*	乘法	z=x * y;　　//z=21
/	除法	z=x/y;　　//z=2
%	取余数	z=x%y;　　//z=1
++	自增 1	
--	自减 1	

C51 中自增和自减运算符是使变量自动加 1 或减 1。自增和自减运算符放在变量前和变量后是不同的,见表 3.5。

表 3.5　自增运算符与自减运算符及其说明

运　算　符	说　　明	举例(设 x=3)
x++	先用 x 的值,再让 x 加 1	y=x++;　　//y=3,x=4
++x	先让 x 加 1,再用 x 的值	y=++x;　　//y=4,x=4
x--	先用 x 的值,再让 x 减 1	y=x--;　　//y=3,x=2
--x	先让 x 减 1,再用 x 的值	y=--x;　　//y=2,x=2

3.4.2　关系运算符

关系运算符就是判断两个数的逻辑大小关系。关系运算的结果为逻辑量,成立为真(1),不成立为假(0)。另外,关系运算符中的等于“==”由两个“=”组成。

关系运算符及其说明见表 3.6。

表 3.6　关系运算符及其说明

符　　号	说　　明	举例(设 x=2,y=3)
>	大于	x>y;　　//返回值 0
<	小于	x<y;　　//返回值 1
>=	大于或等于	x>=y;　　//返回值 0
<=	小于或等于	x<=y;　　//返回值 1
==	等于	x==y;　　//返回值 0
!=	不等于	x! =y;　　//返回值 1

3.4.3　逻辑运算符

逻辑运算符用于求条件式的逻辑值,用逻辑运算符将关系表达式或逻辑量连接起来的式子就是逻辑表达式。逻辑运算的结果为真(1)或者假(0)。

逻辑运算符及其说明见表 3.7。

表 3.7　逻辑运算符及其说明

运　算　符	说　　明	举例(设 x=2,y=3)
&&	逻辑与	x&&y;　　//返回值为 1
||	逻辑或	x||y;　　//返回值为 1
!	逻辑非	! x　　//返回值为 0

3.4.4　位运算符

位运算符是按位对变量进行运算,但并不改变参与运算的变量的值。如果要求按位改变变量的值,则要利用相应的赋值运算。C51 中位运算符只能对整数进行操作,不能对浮点数进行操作。位运算符及其说明见表 3.8。

表 3.8　位运算符及其说明

符　　号	说　　明	举　　例
&	按位逻辑与	0x19&0x4d=0x09
|	按位逻辑或	0x19|0x4d=0x5d
^	按位逻辑异或	0x19^0x4d=0x54
~	按位取反	x=0x0f,则~x=0xf0
<<	按位左移(高位丢弃,低位补 0)	y=0x3a,若 y<<2,则 y=0xe8
>>	按位右移(高位补 0,低位丢弃)	w=0x0f,若 w>>2,则 w=0x03

3.4.5　赋值运算符

赋值运算符及其说明见表 3.9。

表 3.9　赋值运算符及其说明

符　　号	说　　明	举　　例
=	赋值	x=3;　　//x=3 x=3+2;　　//x=5 x=y=3;　　//x=3,y=3
+=	加法赋值	x+=1;　　//x=x+1
-=	减法赋值	x-=2;　　//x=x-2
*=	乘法赋值	x*=3;　　//x=x*3
/=	除法赋值	x/=4;　　//x=x/4
%=	取余赋值	x%=5;　　//x=x%5
&=	逻辑与赋值	x&=0x55;　　//x=x&0x55

续表

符　　号	说　　明	举　　例
|=	逻辑或赋值	x|=0x55;　　//x=x|0x55
^=	逻辑异或赋值	x^=0x55;　　//x=x^0x55
<<=	左移位赋值	x<<=2;　　//x=x<<2
>>=	右移位赋值	x>>=2;　　//x=x>>2

3.4.6　指针与地址运算符

C51 的指针变量用于存储某个变量的地址，C51 用“*”和“&”运算符提取变量的内容和变量的地址，见表 3.10。

表 3.10　指针与地址运算符及其说明

符　　号	说　　明
*	提取变量的内容
&	提取变量的地址

3.4.7　逗号与条件运算符

逗号与条件运算符及其说明见表 3.11。

表 3.11　逗号与条件运算符及其说明

符　　号	说　　明
,	逗号运算符表达式的值是最右边表达式的值。 如：z=(x=2,y=3,3*7); 则执行结果 z=21
?:	逻辑表达式? 表达式 1：表达式 2。 若逻辑表达式为真，则表达式 1 的值为整个条件表达式的值。 若逻辑表达式为假，则表达式 2 的值为整个条件表达式的值。 如：x=3;y=2;z=(x>=y? x：y); 则执行结果 z=3

3.5　C51 的流程控制语句

3.5.1　C51 的基本结构

1. 顺序结构

顺序结构就是从前向后依次执行语句。整体上看，所有程序的基本结构都是顺序结构，中间的某个过程可以是选择结构或循环结构。

2. 选择结构

选择结构的作用是根据指定的条件是否满足，决定从给定的两组操作选择其一。C51语言的选择结构通常用 if 语句、if else 语句和 switch/case 语句实现。

3. 循环结构

循环结构的作用是让某一段程序重复执行多次。C51 语言的循环结构通常用 while、do while 和 for 语句实现。

3.5.2 if 语句

if 语句是 C51 中的一个基本条件选择语句，通常有三种格式：

```
(1)if(表达式)
          {语句;}
(2)if(表达式)
          {语句 1;}
       else
          {语句 2;}
(3)if(表达式 1) {语句 1;}
       else  if(表达式 2){语句 2;}
       else  if(表达式 3){语句 3;}
       …
       else  if(表达式 n-1){语句 n-1;}
       else  {语句 n;}
```

说明如下：

(1) 当 if 后面小括号内的表达式成立(真)时，就执行大括号内的语句；当表达式不成立(假)时，则程序跳过 if 结构执行下一条语句。

(2) 当 if 后面小括号内的表达式成立(真)时，就执行大括号内的语句 1，然后跳过 else 大括号内的语句 2，执行下一条语句；当表达式不成立(假)时，则程序执行 else 后面大括号内的语句 2，然后执行下一条语句。

(3) 当 if 后面小括号内的表达式 1 成立(真)时，就执行大括号内的语句 1，然后跳过之后所有 else 大括号内的语句，执行下一条语句；当表达式不成立(假)时，则程序执行 else 后面的语句，此时需要再次判断 if 后面小括号内的表达式 2，若成立(真)，则执行语句 2，然后跳过剩下所有的 else 语句执行下一条语句；若不成立(假)，则再次判断表达式 3，以此类推。

【例 3.7】 if 语句的用法。

(1) 输入 x 和 y，如果 x 等于 y，则输出 x 的值和 y 的值。串口调试观察结果。

```
#include  <reg51.h>                          //寄存器头文件
#include  <stdio.h>                          //基本输入输出头文件
void main(void)
{
    int x,y;
    SCON=0x52; TMOD=0x20; TH1=0xFD; TR1=1;   //串口初始化
    printf("please input x,y:\n");           //输出提示信息
```

```
    scanf("%d%d",&x,&y);                          //输入 x 和 y 的值
    if(x==y)                                      //如果 x 和 y 相等,则输出,否则不输出
    printf("x=%d,y=%d\n",x,y);
    while(1);                                     //循环
}
```

(2) 输入 x 和 y,如果 x 大于 y,则把 x 的值送给变量 max,如 x 不大于 y,则把 y 的值送给变量 max,输出 max 的值。

```
#include  <reg51.h>
#include  <stdio.h>
void main(void)
{
  int x,y,max;
  SCON=0x52; TMOD=0x20; TH1=0xFD; TR1=1;     //串口初始化
  printf("please input x,y:\n");
  scanf("%d%d",&x,&y);
    if(x>y)
      max=x;
    else
      max=y;
  printf("max=%d\n",max);
  while(1);
}
```

(3) 学生成绩划分为 A、B、C、D、E 五个等级。五名学生的分数分别为 90、80、70、60、50。输入其中某一学生的分数,打印出对应的等级。

```
#include  <reg51.h>
#include  <stdio.h>
void main(void)
{
    int x;
    SCON=0x52; TMOD=0x20; TH1=0xFD; TR1=1;   //串口初始化
    printf("please input x:\n");
    scanf("%d",&x);
if  (x==90)  printf("You are A\n");
    else  if  (x==80)  printf("You are B\n");
    else  if  (x==70)  printf("You are C\n");
    else  if  (x==60)  printf("You are D\n");
    else  if  (x==50)  printf("You are E\n");
    else  printf("You are wrong\n");
    while(1);
}
```

3.5.3 switch/case 语句

if 语句通过 if else if 嵌套可以实现多分支结构,但结构复杂。switch/case 是 C51 专门

处理多分支结构的选择语句，其格式如下：

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

说明如下：

(1) switch 后面小括号内的表达式必须是整型或字符型表达式，运算结果必须是常量。

(2) 大括号里面的每一个 case 后面的常量表达式的值必须不同。

(3) 每个 case 语句后面可以带一个语句，也可以带多个语句，还可以不带。语句可以用花括号括起，也可以不括。

(4) 多个 case 可以共用一组执行语句。

(5) 当表达式的值与大括号内某一 case 后面的常量表达式的值相同时，就执行该 case 后面的语句，然后遇到 break 语句即退出 switch 结构。若表达式的值与所有 case 后的常量表达式的值都不相同，则执行 default 后面的语句，然后退出 switch 结构。

(6) case 语句和 default 语句的出现次序对执行过程没有影响。

(7) 每个 case 语句后面可以有 break，也可以没有。有 break 语句，执行到 break 则退出 switch 结构，若没有，则会顺次执行后面的语句，直到遇到 break 或结束。

【例 3.8】 switch/case 语句的用法。

学生成绩划分为 A、B、C、D、E 五个等级。五名学生的分数分别为 90、80、70、60、50。输入其中某一学生的分数，打印出对应的等级。

```
#include  <reg51.h>
#include  <stdio.h>
void main(void)
{
    int x;
    SCON=0x52; TMOD=0x20; TH1=0xFD; TR1=1;   //串口初始化
    printf("please input x:\n");
    scanf("%d",&x);
    switch(x)
    {
      case 90:   printf("You are A\n"); break;
      case 80:   printf("You are B\n"); break;
      case 70:   printf("You are C\n"); break;
      case 60:   printf("You are D\n"); break;
      case 50:   printf("You are E\n"); break;
      default:   printf("You are wrong\n");
    }
```

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

3.5.4 while 语句

while 语句在 C51 中用于实现循环结构，格式如下：

```
while(表达式)
{  语句; }                                    //循环体
```

while 语句后面小括号里的表达式是能否循环的条件，大括号里的语句是循环体。

while 语句的执行过程如下：

先判断表达式，当表达式成立（真）时，就重复执行循环体内的语句；当表达式不成立（假）时，则中止 while 循环，程序将执行 while 循环结构之外的下一条语句。while 语句在执行时，如表达式第一次就不成立，则循环体一次也不执行。

【例 3.9】 while 语句的用法。计算并输出 1～10 的累加和。

```
#include  <reg51.h>
#include  <stdio.h>
void main(void)
{
  int  i=1,sum=0;
  SCON=0x52; TMOD=0x20; TH1=0xFD; TR1=1;     //串口初始化
  while  (i<=10)
  {
      sum=sum+i;
      i++;
  }
  printf("1+2+3+4+5+6+7+8+9+10=%d\n",sum);
  while(1);
}
```

3.5.5 do while 语句

do while 语句在 C51 中也用于实现循环结构，格式如下：

```
    do
  {  语句; }                                  //循环体
while(表达式);
```

do 后面大括号里的语句是循环体，while 后面小括号里的表达式是能否循环的条件。

do while 语句的执行过程如下：

先执行一遍循环体中的语句，后判断表达式。如表达式成立（真），则再执行循环体，然后又判断表达式，直到表达式不成立（假）时退出循环，执行 do while 循环结构外的下一条语句。do while 语句在执行时，循环体内的语句至少会被执行一次。

【例 3.10】 do while 语句的用法。计算并输出 1～10 的累加和。

```
#include  <reg51.h>
#include  <stdio.h>
void main(void)
{
  int  i=1,sum=0;
  SCON=0x52; TMOD=0x20; TH1=0xFD; TR1=1;     //串口初始化
  do
  {
      sum=sum+i;
      i++;
  }
  while  (i<=10);
  printf("1+2+3+4+5+6+7+8+9+10=%d\n",sum);
  while(1);
}
```

3.5.6 for 语句

for 语句在 C51 中用于实现循环结构的功能最强大，可用于循环次数已经确定的情况，也可用于循环次数不确定的情况。其格式如下：

```
for(表达式 1; 表达式 2; 表达式 3)
{  语句; }                                    //循环体
```

for 后面小括号里包含 3 个表达式，大括号里的语句是循环体。在 for 循环中，一般表达式 1 为初值表达式，用于给循环变量赋初值；表达式 2 为条件表达式，对循环变量进行判断；表达式 3 为循环变量更新表达式，用于对循环变量的值进行更新，使循环变量在不满足条件时退出循环。

for 语句的执行过程如下：

(1) 求解表达式 1 的值。

(2) 求解表达式 2 的值。若表达式 2 的值为真，则执行循环体中的语句，执行完后再执行步骤(3)的操作；若表达式 2 的值为假，则退出 for 循环，执行 for 循环结构外的下一条语句。

(3) 若表达式 2 的值为真，则执行完循环体中的语句后，求解表达式 3，然后转到步骤(2)继续执行。

【例 3.11】 for 语句的用法。计算并输出 1～10 的累加和。

```
#include  <reg51.h>
#include  <stdio.h>
void main(void)
{
  int  i=1,sum=0;
  SCON=0x52; TMOD=0x20; TH1=0xFD; TR1=1;     //串口初始化
```

```
    for (i=1;i<=10;i++)
      sum=sum+i;
    printf("1+2+3+4+5+6+7+8+9+10=%d\n",sum);
    while(1);
}
```

3.5.7 break 语句和 continue 语句

break 语句和 continue 语句通常用于循环结构中，用来跳出循环结构，但是二者又有所不同。

1. break 语句

break 语句可以跳出 switch 结构，使程序继续执行 switch 结构后面的下一条语句。break 语句还可以从循环体中跳出该循环，提前结束该循环而接着执行循环结构下面的语句。break 语句不能用在除了 switch 语句和循环语句之外的任何其他语句中。

2. continue 语句

continue 语句用在循环结构中，用于结束本次循环，跳过循环体中 continue 下面尚未执行的语句，直接进行下一次是否继续执行循环的判定。

continue 语句和 break 语句的区别在于：continue 语句只是结束本次循环，而不是终止整个循环；break 语句则是结束循环，不再进行条件判断。

3.5.8 return 语句

return 语句一般放在函数的最后位置，用于终止函数的执行，并控制程序返回调用该函数时所处的位置。返回时还可以通过 return 语句带回返回值。return 语句的格式有两种：

(1) return；

(2) return (表达式)；

如果 return 语句后面带有表达式，则要计算表达式的值，并将表达式的值作为函数的返回值。若不带表达式，则函数返回时将返回一个不确定的值。通常用 return 语句把调用函数取得的值返回给主调用函数。

3.6 C51 的函数

3.6.1 函数的定义

函数定义的一般格式如下：

```
函数类型  函数名(形式参数表) [reentrant] [interrupt m] [using n]
{
    函数体
}
```

1. 函数类型

函数类型用于说明函数返回值的类型，为表 3.1 所述基本数据类型。如果函数没有返

回值,则函数类型一般定义为空类型 void。

2. 函数名

函数名是用户为自定义函数取的名字,以便调用函数时使用。

3. 形式参数表

形式参数表用于列出在主调用函数与被调用函数之间进行数据传递的形式参数。形式参数也需要进行数据类型说明。无形式参数时,括号内可以空着或者用 void。若包含多个形式参数,则各个形参之间用逗号隔开。

4. reentrant 修饰符

reentrant 修饰符用于把 C51 的函数定义为可重入函数。所谓可重入函数,就是允许被递归调用的函数。函数的递归调用是指当一个函数正被调用尚未返回时,又直接或间接调用函数本身。一般的函数不允许递归调用,只有声明为可重入函数才允许递归调用。

5. interrupt m 修饰符

interrupt m 修饰符用于把 C51 的函数定义为中断函数。当函数定义时用了 interrupt m 修饰符,系统在编译时自动把对应函数转化为中断函数,自动加上程序头段和尾段,并按 51 单片机中断的处理方式自动把它安排在程序存储器中的相应位置。在该修饰符中,m 的取值为 0～31。

6. using n 修饰符

using n 修饰符用于指定本函数内部使用的工作寄存器组,其中 n 的取值为 0～3,表示寄存器组号。using n 通常与 interrupt m 联合使用。

3.6.2 函数的调用

函数调用的一般格式如下:

```
函数名(实际参数);
```

对于有参数的函数调用,若包含多个实际参数,则各个实参之间用逗号隔开。

按照函数调用在主调函数中出现的位置,函数调用方式有以下三种:

(1) 函数语句。把被调用函数作为主调用函数的一个语句。

(2) 函数表达式。函数被放在一个表达式中,以一个运算对象的方式出现,这时的被调用函数要求带有返回语句,以返回一个明确的数值参加表达式的运算。

(3) 函数参数。被调用函数作为另一个函数的参数。

【例 3.12】 函数的定义和调用。

```
#include  <reg51.h>
#include  <stdio.h>
int max(int  x,int  y)                          //取两个数中的最大值
{   int  z;
    z=(x>=y?x:y);                               //若 x>=y,则把 x 赋给 z,否则把 y 赋给 z
    return(z);                                  //返回 z 的值
}
void main(void)                                 //主函数
{
```

```
    int x,y;
    SCON=0x52; TMOD=0x20; TH1=0xFD; TR1=1;   //串口初始化
    printf("please input x,y:\n");
    scanf("%d%d",&x,&y);
    printf("max is:%d\n",max(x,y));
    while(1);
}
```

3.7 C51 的数组与指针

3.7.1 数组

1. 一维数组

1）一维数组的定义

一维数组只有一个下标，定义的格式如下：

```
数据类型说明符  数组名[常量表达式] [={初值,初值…}]
```

说明如下：

（1）“数据类型说明符”用于说明数组中各个元素存储的数据的类型，为前述基本数据类型，如 int 等。

（2）“数组名”是整个数组的标识符，取名方法与变量取名方法相同。同时，数组名还是整个数组第一个元素的首地址。

（3）“常量表达式”的取值为整型常量，必须用方括号“[ ]”括起来，用于说明该数组的长度，即该数组元素的个数。

（4）“初值”用于给数组元素赋初值，也称为初始化。对数组元素赋初值，可以在定义时赋值，也可以在定义之后赋值。在定义时赋值，后面须带等号，初值须用花括号括起来，括号内的初值互相之间用逗号间隔，可以对数组的全部元素赋值，也可以只对部分元素赋值。初值为 0 的元素可以只用逗号占位，而不写初值 0。

下面是一维数组定义的两个例子：

```
unsigned  char  a[5];
unsigned  int  b[3]={1,2,3};
```

第一句定义了一个无符号字符型数组，数组名为 a，数组中的元素个数为 5。

第二句定义了一个无符号整型数组，数组名为 b，数组中的元素个数为 3，在定义的同时给数组中的三个元素赋初值，初值分别为 1、2、3。

需要注意的是，C51 中数组的下标是从 0 开始的，因此上面第一句定义的 5 个元素分别是 a[0]、a[1]、a[2]、a[3]、a[4]。第二句定义的 3 个元素分别是 b[0]、b[1]、b[2]。赋值情况为 b[0]=1；b[1]=2；b[2]=3。

C51 规定，在引用数组时只能逐个引用数组中的各个元素，而不能一次引用整个数组。但如果是字符数组，则可以一次引用整个数组。

2）一维数组的初始化

对一维数组的初始化，可以用以下方法实现：

（1）在定义数组时对数组的全部元素赋初值。例如：

```
int  a[3]={1,2,3};
```

经过上面的定义与初始化后，a[0]＝1，a[1]＝2，a[2]＝3。

（2）只对数组的一部分元素赋值。例如：

```
int  a[3]={1,2};
```

定义数组 a 有 3 个元素，但花括号内只提供 2 个初值，初始化后，有 a[0]＝1，a[1]＝2，最后 1 个元素的值默认为 0，即 a[2]＝0。

（3）对数组的全部元素赋值时，也可以不指定数组长度。例如：

```
int  a[3]={1,2,3};
```

可以写成：

```
int  a[ ]={1,2,3};
```

由于这种写法花括号里面有 3 个数，因此系统自身定义 a 数组元素个数为 3，并将这 3 个初值分配给 3 个数组元素。注意，如果只对一部分元素赋值，就不能省略掉表示数组长度的常量表达式。

2. 二维数组

1）二维数组的定义

二维数组有两个下标，定义的格式如下：

```
数据类型说明符  数组名[常量表达式][常量表达式] [={初值,初值…}]
```

例如：

```
int  a[2][3];                                    //定义 a 为 2 行 3 列的数组
```

二维数组的存取顺序是：按行存取，先存取第 1 行第 0 列，1 列，2 列，直到第 1 行的最后一列。然后转到第 2 行开始，再取第 2 行第 0 列，1 列，2 列，直到第 2 行最后一列。以此类推，直到最后一行的最后一列。

2）二维数组的初始化

对二维数组的初始化，可以用以下方法实现：

（1）在定义数组时分行对数组的全部元素赋初值。例如：

```
int  a[2][3]={{1,2,3},{4,5,6}};
```

赋值后数组元素如下：

```
1   2   3
4   5   6
```

可见，经过上面的定义与初始化后，把第 1 个花括号内的值赋给第 1 行元素，把第 2 个花括号内的值赋给第 2 行元素。也可以将所有数据写在一个花括号内，按数组的排列顺序

对各元素赋初值。其初始化结果与上面一样。例如：

```
int  a[2][3]={1,2,3,4,5,6};
```

(2) 可以只给一部分元素赋值。例如：

```
int  a[2][3]={1,2};
```

赋值后数组元素如下：

```
1   2   0
0   0   0
int  a[2][3]={ {1},{2} }
```

赋值后数组元素如下：

```
1   0   0
2   0   0
```

3. 字符数组

用来存放字符数据的数组称为字符数组，它是 C 语言中常用的一种数组。字符数组中的每一个元素都用来存放一个字符，也可用字符数组存放字符串。字符数组的定义与一般数组相同，只是在定义时把数据类型定义为 char 型。

1) 字符数组的定义

一维字符数组有一个下标，定义的格式如下：

```
char   数组名[常量表达式] [={初值,初值…}]
```

二维字符数组有两个下标，定义的格式如下：

```
char   数组名[常量表达式][常量表达式] [={初值,初值…}]
```

例如：

```
char  a[10];
char  a[3][20];
```

上面第一句定义了一个一维字符数组，包含 10 个字符元素。第二句定义了一个二维字符数组，包含 60 个字符元素。

在 C51 语言中，字符数组用于存放一组字符或字符串，字符串以“\0”作为结束符，只存放一般字符的字符数组的赋值与使用和一般的数组完全相同。对于存放字符串的字符数组，既可以对字符数组的元素逐个进行访问，也可以对整个数组按字符串的方式进行处理。

2) 字符数组的初始化

将字符数组初始化的最直接方法是将各个字符逐个赋给数组中的元素。例如：

```
char  a[10]={'N','a','n','k','a','i',' ',' ',' '};
```

上面定义了一个字符型数组 a[10]，一共有 10 个元素。

C51 还允许用字符串直接给字符数组赋初值，有如下两种形式：

```
char  a[ ]={"Nankai"};
```

```
char  a[ ]="Nankai";
```

上面例子的[ ]中的常量表达式没有规定元素个数,由初始化的字符串长度决定。

二维字符数组由若干个字符串组成,也可称之为字符串数组。二维字符数组的第 1 个下标是定义字符串的个数,第 2 个下标是定义每个字符串的长度。该长度应当比这批字符串中最长字符的个数多一个字符,用于装入字符串的结束符"\0"。

例如:

```
unsigned  char  a[3][20]=
{   {"Hello Qinghua!"},
    {"Byebye Beida!"},
    {"This is a Joke!"},
};
```

上面定义了一个二维字符数组 a[3][20],数组名为 a,可以容纳 3 个字符串,每个字符串最多能够存放 20 个字符。其中第一个下标可以省略,如 a[ ][20],由初始化的字符串个数决定。如本例中省略第 1 个下标,那么其值是 3,因为是 3 个字符串。第 2 个下标必须给定,因为它不能从数据表中得到。

【例 3.13】 数组的定义和输出。

```
#include  <reg51.h>
#include  <stdio.h>
void main(void)
{
    int i;
    int array[8]={1,2,3,4,5,6,7};
    char  string[20]="please show array:\n";
    SCON=0x52; TMOD=0x20; TH1=0xFD; TR1=1;  //串口初始化
    printf("%s",string);
    for(i=0;i<8;i++)
        printf("%2d", array[i]);
    while(1);
}
```

3.7.2 指针

在 C 语言中,数据存放在内存单元中,内存单元按字节进行组织管理。内存单元前面的编号字节叫作内存单元的**地址**;内存单元里存放的数据叫作内存单元的**内容**。C 语言可以直接访问内存单元的数据,也可以通过地址方式访问内存单元的数据。作为一种高级程序设计语言,C 语言中的数据通常是以变量的形式进行存放和访问的。

变量在使用时需要分清两个概念:变量名和变量的值。前一个是数据的标识,后一个是数据的内容。**变量名**相当于内存单元的**地址**,**变量的值**相当于内存单元的**内容**。

对于内存单元的数据有两种访问方式,对于变量也有两种访问方式:直接访问方式和间接访问方式。

直接访问方式。对于变量的访问是直接给出变量名。例如:printf("%d",x),直接给