第二篇
C51语言程序设计指南

第 3 章　C51语言程序设计基础

C51语言既具有汇编语言对操作硬件的功能，又兼有高级编程语言的许多优点。在现代单片机程序设计中，C51语言得到了广泛的应用。本章主要介绍C51程序的基本结构以及C51程序设计的基础知识，包括标识符、关键字、数据类型、表达式和运算符等。

3.1　C51程序的基本结构

单片机C51语言继承了C语言的特点，其程序结构与一般C语言的程序结构没有差别。C51源程序文件的扩展名为“.c”，如Test.c、Function.c等。每个C51源程序中包含一个名为“main()”的主函数，C51程序的执行总是从main()函数开始的。当主函数中所有语句执行完毕，则程序执行结束。下面是一个典型的C51源程序的例子。

```
#include <reg52.h>                          //预处理命令,reg52.h是一个头文件

void Function1(void);                       //自定义函数Function1声明

void main(void)                             //主函数
{
     Function1();                           //调用自定义函数Function1
     unsigned char ch;                      //主函数中变量声明
     while(1)
     {
     printf("ch=%c\n",ch);                  //程序语句
     ch++;                                  //程序语句
}
}

void Function1(void)                        //自定义函数Function1
{
     unsigned char ps;                      //自定义函数内部变量声明
     ps=12;                                 //程序语句
     printf("ps=%d\n",ps);                  //程序语句
}
```

从上面的例子可以看出，一个典型的C51源程序包含预处理命令、自定义函数声明、main主函数和自定义函数。这几部分完全类似于C语言的程序结构，各个部分的功能如下。

- 预处理命令部分常用#include命令来包含一些程序中用到的头文件。这些头文件中包含了一些库函数以及其他函数的声明及定义。
- 自定义函数声明部分用来声明源程序中自定义的函数。
- main主函数是整个C51程序的入口。不论main()函数位于程序代码中的哪个位置，C51程序总是首先从main()函数开始执行的。
- 自定义函数部分是C51源程序中用到的自定义函数的函数体，其中实现了用户自定义的功能。

除了扩展名为“.c”的源程序文件外，C51程序还支持扩展名为“.h”的头文件以及扩展名为“.lib”的库文件等。在一般的编译系统中，通常以项目结构来管理复杂的C51程序文件。例如在Keil μ Vision3编译环境中，整个项目结构如图3.1所示。

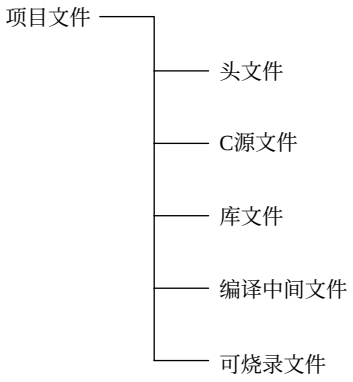

图3.1 Keil μ Vision3项目结构示意图

在这里，整个项目由项目文件来管理，项目文件的扩展名为“.Uv2”。整个工程项目中可以包含如下几类文件。

- 头文件用来包含一些库函数、系统变量声明以及将不同的C文件连接起来。
- C源文件是C51程序的主要部分，用来实现特定的功能。C源文件可以有一个，也可以按照不同的功能分成多个，但所有这些C源文件中有且仅有一个可以包含一个main()主函数。
- 库文件是实现特定功能的函数库，供C源文件进行调用。
- 编译中间文件是源程序在编译链接过程中生成的中间文件，其中包含了文件编译调试的信息。
- 可烧录文件是编译系统生成的可以烧录到单片机内部供执行的文件，类似于“.exe”可执行文件。在C51语言中，一般扩展名为“.hex”或者“.bin”等。

在这些文件中，C源文件是必需的，其他的文件可以根据用户的实际需要而选用。

3.2 C51编程规范及注意事项

在学习任何一种编程语言时，按照一定的规范培养良好的编程习惯很重要。良好的编程规范可以帮助开发人员理清思路、方便整理代码，同时也便于他人阅读、理解，以促进代码的交流。在进行C51语言程序设计时，应该注意以下几个方面的编程规范。

3.2.1 注释

任何编程语言都支持注释语句。注释语句只对代码起到功能描述的作用，在实际的编译链接过程中不起作用。在C51语言中可以通过下面两种方式来表示注释内容。

- 用“//”符号开头来注释一行，如“//变量声明”。
- 用“/*”符号开头，并以“*/”符号结束来对C51源程序中的一部分内容来进行注释，

如“ /*声明整型变量ch */ ”。

第一种方法简明、方便，“ // ”符号可以在一行的开始，这样整行都表示注释内容；“ // ”符号也可以放置在某行的执行语句后面，“ // ”符号后面的内容便是对该语句的注释内容。

第二种方法灵活多变，可以注释多行内容，示例如下。

```
/*****************************************************/
/*    Function.C:  使用C51编译器的自定义功能函数库            */
/*****************************************************/
```

用户也可以在程序语句的内部进行注释，示例如下。

```
printf("ch=%d\n",/*整型变量ch*/ch);
```

一个好的C51源程序应该添加必要的注释内容。这样，可以增加程序的可读性，方便日后修改或者与他人的代码交流。注释内容一般包括程序的功能、实现方式、自定义函数的功能描述及语句的功能等。

注意　注释不能嵌套，即一个注释中不能再包含另一个注释。

3.2.2　命名

在进行程序设计时，经常需要自定义一些函数或者变量。一般来说，只要符合C51命名规范即可通过编译。但是，为了便于源程序的理解和交流，在进行命名时应注意如下几点。

- 自定义函数或者变量的名称最好能反映该函数或变量的功能用途。因此，需要采用有意义的单词或者字母组合来表示。例如MAX表示最大值、MIN表示最小值等。
- 变量名通常加上表示数据类型的前缀，如“ ucSendData ”的前缀“ uc ”表示unsigned char。
- 在命名时不要和系统保留的标识符以及关键字产生冲突或者歧义。

经验　函数或变量名必须以字母a～z、A～Z或下划线“ _ ”开头，C51中的函数或变量名要区分字母的大小写。

3.2.3　格式

为了方便地阅读程序，在进行C51程序设计时，在程序结构以及语句书写格式方面应注意以下几点。

- 虽然C51语言对main()函数放置的位置没有限定，但为了程序阅读的方便，最好将其放置在所有自定义函数的最前面，即依次是头文件声明、自定义函数以及全局变量声明、main()函数、自定义函数。
- C51语句可以分一行或多行进行编写。为了便于理解程序，最好将每个语句单独写在一行，并加以注释。有时某几个相连的语句相近或者共同执行某个功能则可以放置在一行。
- 对于源程序文件不同结构部分之间要留有空行。例如，头文件声明、自定义函数声明、main()函数以及自定义函数之间均要空一行，来明显区分不同的结构。
- 对于if、while等块结构语句中的“ { ”和“ } ”要配对并且对齐，以便于程序阅读时能

够理解该结构的起始和结束。

- 源代码安排时可以通过适当的空格以及Tab键来实现代码对齐。

以上是一些常用的编程规范，读者可以参考借鉴。

3.3 C51的标识符与关键字

标识符和关键字是一种编程语言最基本的组成部分，C51语言同样支持自定义的标识符以及系统保留的关键字。在进行C51程序设计时，需要了解标识符和关键字的使用规则。

3.3.1 标识符

标识符常用来声明某个对象的名称，如变量和常量的声明、数组和结构的声明、自定义函数的声明以及数据类型的声明等。示例如下。

```
int count;
void Function1();
```

在上面的例子中，count为整型变量的标识符，Function1为自定义函数的标识符。

在C51语言中，标识符可以由字母、数字（0～9）和下划线“_”组成，最多可支持32个字符。C51标识符第一个字符必须是字母或者下划线“_”，例如“ut1”、“ch_1”等都是正确的，而“5count”则是错误的标识符。另外，C51的标识符区分大小写，例如“count1”和“COUNT1”代表两个不同的标识符。在C51语言中，使用标识符需要注意如下几点。

- 在命名C51标识符时，需要能够清楚地表达其功能含义，这样有助于阅读和理解源程序。
- C51的标识符原则上可以使用下划线开头，但有些编译系统的专用标识符或者预定义项是以下划线开头的。为了程序的兼容性和可移植性，所以建议一般不要以下划线开头来命名标识符。例如，Keil μ Vision3编译系统中，包含了一些预定义的函数用来对程序进行调试，这些大都采用下划线开头，如“_sleep_”、“__sin”等。
- 尽量不要使用过长的标识符，以便于使用和程序理解方便。
- 自定义的C51标识符不能使用C51语言保留的关键字，也不能和用户已使用的函数名或C51库函数同名。例如“char”是关键字，所以它不能作为标识符使用。

3.3.2 关键字

关键字是C51语言重要的组成部分，是C51编译器已定义保留的专用特殊标识符，有时也称为保留字。这些关键字通常有固定的名称和功能，如int、float、if、for、do、while、case等。C51语言中常用的关键字如表3.1所示。

表3.1 C51语言中的关键字

类　别	关 键 字	类　型	用 途 说 明
ANSIC标准关键字	auto	存储种类说明	常用于声明局部变量，默认值为此类型
	break	程序语句	无条件退出循环程序最内层循环
	case	程序语句	switch选择语句中的选择项
	char	数据类型说明	单字节整型数据或字符型数据

（续）

类　别	关 键 字	类　型	用 途 说 明
ANSIC标准关键字	const	存储类型说明	定义不可更改的常量值
	continue	程序语句	中断本次循环，并转向下一次循环
	default	程序语句	switch选择语句中的默认选择项
	do	程序语句	用于构成do...while循环结构
	double	数据类型说明	声明双精度浮点型数据
	else	程序语句	用于构成if...else选择结构
	enum	数据类型说明	枚举
	extern	存储种类说明	在其他程序模块中说明了的全局变量
	float	数据类型说明	定义单精度浮点型数据
	for	程序语句	构成for循环语句
	goto	程序语句	构成goto转移结构
	if	程序语句	用于构成if...else选择结构
	int	数据类型说明	声明基本整型数据
	long	数据类型说明	声明长整型数据
	register	存储种类说明	CPU内部寄存的变量
	return	程序语句	用于返回函数的返回值
	short	数据类型说明	声明短整型数据
	signed	数据类型说明	声明有符号数，二进制表示的最高位为符号位
	sizeof	运算符	计算表达式或数据类型的占用字节数
	static	存储种类说明	声明静态变量
	struct	数据类型说明	声明结构类型数据
	switch	程序语句	构成switch选择结构
	typedef	数据类型说明	重新定义数据类型
	union	数据类型说明	声明联合类型数据
	unsigned	数据类型说明	声明无符号数据
	void	数据类型说明	声明无类型数据
	volatile	数据类型说明	该变量在程序执行中可被隐含地改变
	while	程序语句	用于构成do...while或while循环结构
C51扩展关键字	bit	位标量声明	声明一个位变量以及位类型的函数
	sbit	位标量声明	声明一个可位寻址的变量
	sfr	特殊功能寄存器声明	声明一个8位的特殊功能寄存器
	sfr16	特殊功能寄存器声明	声明一个16位的特殊功能寄存器
	data	存储器类型说明	直接寻址的单片机片内数据存储器
	bdata	存储器类型说明	可位寻址的单片机片内数据存储器
	idata	存储器类型说明	间接寻址的单片机片内数据存储器
	pdata	存储器类型说明	分页寻址的单片机片内数据存储器
	xdata	存储器类型说明	单片机片外数据存储器
	code	存储器类型说明	单片机程序存储器
	interrupt	中断函数说明	定义一个中断服务函数
	reentrant	再入函数说明	定义一个再入函数
	using	寄存器组定义	定义单片机的工作寄存器

从该表中可以看出，单片机C51程序语言不仅继承了ANSIC标准定义的32个关键字，还根据C51语言以及单片机硬件的特点扩展了相关的关键字。在C51语言程序设计中，用户自定义的标识符不能和这些关键字相冲突，否则无法正确通过编译。

3.4 C51的变量类型

数据类型是C51语言最基本的组成部分。在C51中，每个变量在使用之前必须定义其数据类型。C51语言中的数据类型分为基本数据类型和聚合数据类型，这里首先介绍基本数据类型。

3.4.1 C51的数据类型

C51的基本数据类型有整型（int）、浮点型（float）、字符型（char）、无值型（void）。在基本数据类型中，除void类型外，其前面均可以有各种修饰符。修饰符用来改变基本类型的意义，以便更准确地适应各种情况的需求。

常用的修饰符有signed（有符号）、unsigned（无符号）、long（长型符）、short（短型符）。在C51语言中，所有数据类型的字长和取值范围如表3.2所示。

表3.2 C51语言的数据类型

类 型	字长（bit）	取值范围
char（字符型）	8	ASCII字符或0～255
unsigned char（无符号字符型）	8	0～255
signed char（有符号字符型）	8	-128～127
int（整型）	16	-32 768～32 767
unsigned int（无符号整型）	16	0～65 535
signed int（有符号整型）	16	同int
short（短整型）	16	同int
unsigned short int（无符号短整型）	16	0～65 535
signed short int（有符号短整型）	16	-32 768～32 767
long int（长整型）	32	-2 147 483 648~2 147 483 649
signed long int（有符号长整型）	32	-2 147 483 648~2 147 483 649
unsigned long int（无符号长整型）	32	0～4 294 967 295
float（单浮点型）	32	-1.175 494E-38 ~ 3.402823E+38
void（无值型）	0	无值

注意 表中各个数据类型组合的字长和取值范围是假定CPU的字长为16bit。

从表中可以看出，修饰符signed、short、long和unsigned可以修饰字符型和整型两种基本类型。此外，修饰符long还可用于float型。

说明 C51除了继承了标准C语言中基本的数据类型外，又兼具其本身的特点，如在C51语言中int和short，float和double具有相同的取值范围和含义。

一般来说，编译环境对整数类型的默认定义是有符号数，singed修饰符可以省略。为了使用方便，C51编译程序允许使用整型简写形式，如下所示。

- short int 简写为short
- long int 简写为long
- unsigned short int 简写为unsigned short
- unsigned int 简写为unsigned
- unsigned long int 简写为unsigned long

另外，某些编译环境允许将unsigned用于浮点型，如unsigned float。但这种数据类型并不通用，在程序移植时会出现问题，建议一般不要采用。

此外，C51语言还提供了几种聚合类型（aggregate types），包括数组、指针、结构、联合（共用体）、枚举和位域。这几种聚合类型将在后面章节中陆续介绍，这里首先介绍常用的基本类型。

3.4.2　整型变量

整型变量是整数类型的数据。整型变量是最常用的数据类型。整型变量的定义格式是“类型说明符 变量标识符，变量标识符...”。示例如下。

```
int             a, b;                    //定义a、b为短整型变量
long            c, d;                    //定义c、d为长整型变量
```

其中，类型说明符与变量标识符之间至少有一个空格。最后一个变量标识符必须以“；”结尾。整型变量的前面可以加上不同的修饰符，整型变量的类型如表3.3所示。

表3.3　整型变量的类型

类　　型	简 写 形 式	字长（bit）	取 值 范 围
unsigned short int（无符号短整型）	unsigned int	8	0～65 535
signed short int（有符号短整型）	short或int	8	-32 768～32 767
signed long int（有符号长整型）	unsigned long	32	-2 147 483 648～2 147 483 649
unsigned long int（无符号长整型）	long	32	0～4 294 967 296

> 说明　在有符号型数据的二进制表示中，字节最高位表示数据的符号，“0”表示正数，“1”表示负数。

另外，在C51中定义变量时，允许同时定义多个相同类型的变量，各变量间用逗号间隔。变量定义必须放在变量使用之前，一般位于函数的开头。整型变量使用的示例程序如下。

```
01:  #include <stdio.h>                 //头文件
02:
03:  void main( )                       //主函数
04:  {
05:       int    a,b,c,d;               //定义a,b,c,d为整型变量
06:       a=15; b= -30;                 //赋初值
07:       unsigned int x;               //定义x为无符号整型变量
08:       x=25;                         //赋初值
09:       c=a+x; d=b+x;                 //变量运算
```

```
10:         printf("a+x=%d, b+x=%d\n",c,d);          //打印输出结果
11:  }
```

该程序可以在Keil μ Vision3集成开发环境中运行，执行的结果如下。

```
a+x=40, b+x=-5
```

在本例中，int型数据与unsigned int型数据进行相加减的算术运算，由此可以看出，不同类型的整型数据之间可以进行算术运算。

技巧	C51输出函数是通过串行口工作的，因此，要使用printf函数必须对单片机的串行口进行设置和初始化。要在Keil μ Vision3中编译运行以上程序，可在第7行之后添加入代码TI= 1;//允许发送数据，并在第2行中包含“reg51.h”或“reg52.h”头文件。

3.4.3 浮点型变量

浮点型变量是用于表示包含小数点的数据类型。浮点型变量的一般定义格式是“类型说明符 变量标识符，变量标识符...”，示例如下。

```
float     a, b;                                  //定义a、b为单精度浮点型变量
double    c, d;                                  //定义c、d为双精度浮点型变量
```

同整型变量一样，浮点型变量也可以同时定义多个。C51支持3种浮点型变量类型，即float类型、double类型和long double类型。但是在C51中不具体区分这3种类型，它们都被当做float类型对待，因此，这3种浮点类型的精度和取值范围相同。浮点类型变量的字长为4个字节，共32位二进制数，取值范围为$3.4x10^{-38}$～$3.4\times10^{+38}$。浮点型变量的示例程序如下。

```
#include <stdio.h>                                //头文件
void main( )                                      //主函数
{
     float a;                                     //定义a为单精度浮点型变量
     double b;                                    //定义b为双精度浮点型变量
     a=8976.1538;                                 //赋值
     b=6950.2692;
     printf("a=%f\nb=%f\n",a,b);                  //打印输出结果
}
```

该程序可在Keil μ Vision3集成开发环境中运行，执行结果如下。

```
a=8976.154000
b=6950.269000
```

在本例中，a为单精度浮点型变量，有效位数是7位，因此最后四舍五入为8976.154。b为双精度浮点型变量，但在C51中仍按float型处理，所以有效位数也是7位，因此最后四舍五入为6 950.269。

注意	在C51中，浮点型数据均为有符号浮点数，而没有无符号浮点数，字节最高位表示数据的符号，所以浮点型变量的有效位数是7位。

3.4.4 字符型变量

在信息的表示和传递中，经常用到字符以及文字的表示。字符型变量就是用来存放单个字符的变量类型。字符型变量的定义格式是“ 类型说明符 变量标识符，变量标识符... ”。示例如下。

```
char            a;                  //定义a为有符号字符变量
unsigned char   b;                  //定义b为无符号字符变量
```

在C51中，可以定义两种类型的字符型变量为有符号字符变量char和无符号字符变量unsigned char。在C51中，字符型变量在操作时将按整型变量处理。字符是以ASCII码方式表示的，其字长为1个字节，所以有符号字符型变量的取值范围为 - 128～127，无符号字符型变量的取值范围是0～255。如果某个变量被定义成char，则表明该变量是有符号的，即它将转换成有符号的整型变量。字符型变量使用的示例程序如下。

```
01:  #include <stdio.h>                                          //头文件
02:
03:  void main()                                                  //主函数
04:  {
05:       char c1,c2,c3,c4;                                      //定义字符变量
06:       c1=65;                                                 //赋值
07:       c2=66;                                                 //赋值
08:       c3='A';
09:       c4='B';
10:       printf("c1=%c\nc2=%c\nc3=%c\nc4=%c",c1,c2,c3,c4);    //输出结果
11:  }
```

该程序可在Keil μ Vision3集成开发环境中运行，执行结果如下。

```
c1=A
c2=B
c3=A
c4=B
```

注意	如上例代码中的第8行和第9行所示，将字符号向字符变量赋值时，字符必须以半角状态的单引号括起来，不能用双引号。

程序中，c1、c2、c3、c4被定义为字符变量。其中，c3和c4分别赋值两个字符，因此输出结果为字符。c1和c2分别赋值十进制整数65和66，因为字符A和B的ASCII码值分别为65和66，因此程序中使用“ %c ”来表示输出时将输出其对应的字符。对c1和c2的赋值相当于以下的赋值语句。

```
c1 = 'A';
c2 = 'B' ;
```

在实际使用中，常会需要存放一个字符串，这时就需要定义一个字符型数组，将字符串存放到该数组中来表示。定义字符串数组的示例程序如下。

```
#include <stdio.h>                          //头文件

void main()                                 //主函数
```

```
{
      char str[5];                          //定义字符型数组
      str [0]='T';                          //赋值
      str [1]='e';                          //赋值
      str [2]='s';                          //赋值
      str [3]='t';                          //赋值
      printf("str=%s",str);                 //打印输出结果
}
```

在程序编译时，系统将留出str [0]到str [4]共5个字符的连续空间，其中只有前4个可以自由赋值，最后一个用来存放字符串终止符“ NULL ”，即“ \0 ”。

这段程序可在Keil μ Vision3集成开发环境中运行，执行结果如下。

```
str=Test
```

注意 终止符“ NULL ”是编译程序默认自动加上的，程序中绝对不能为最后一个赋值。如果对最后一个进行赋值，将导致程序的错误输出。

3.4.5 指针型变量

指针型变量是指向变量所存放地址的数据类型。在C51中，指针型变量的定义格式是“ 类型说明符 变量标识符，变量标识符... ”，示例如下。

```
int       *i；                          //定义整型指针变量
float     *f；                          //定义浮点型指针变量
char      *c；                          //定义字符型指针变量
struct    *stu；                        //定义结构型指针变量
union     *uni；                        //定义联合指针变量
```

指针型变量是一种特殊的数据类型，从上面可以看出，根据所指的变量类型不同，指针型变量可以分为整型指针、浮点型指针、字符型指针、结构型指针和联合指针。指针型变量的字长由所指向的变量类型决定。这里以字符型指针变量为例介绍指针型数据的使用，程序示例如下。

```
#include <stdio.h>                      //头文件

void main()                             //主函数
{
      char a1;                          //定义字符型变量
      char *p;                          //定义字符型指针
      a1='A';                           //变量赋值
      p=&a1;                            //将变量a1的地址赋给p
      printf("*p=%c",*p);               //输出地址p中所存的数据内容
}
```

这段程序可在Keil μ Vision3集成开发环境中运行，执行结果如下。

```
*p=A
```

在该程序中，首先给字符型变量a1赋值'A'，然后将a1的地址赋给字符型指针变量p。这样变量p所指向的地址内存放的数据是'A'。

3.4.6　无值型变量

无值型变量是一个特殊的类型，其字节长度为0。无值型变量的定义格式是“ 类型说明符 变量标识符，变量标识符... ”，示例如下。

```
void       *buf;                          //buf被定义为无值型指针
```

在C51中，无值型变量主要用于以下两个方面。

- ❑ 定义一个同一类型的指针，该指针可根据需要动态分配内存。
- ❑ 在自定义函数时，明确地表示这个函数不返回任何值。程序示例如下。

```
#include <stdio.h>                          //头文件

void Fun1(int a)                            //定义一个无返回值的函数
{
     printf("the number is %d",a);          //输出
}

void main()                                 //主函数
{
     int i;                                 //定义变量
     i=12;                                  //赋值
     Fun1(i);                               //调用函数
}
```

这段程序可在Keil μ Vision3集成开发环境中运行，执行结果如下。

```
the number is 12
```

在该程序中，先使用void自定义一个无返回值的函数，在主函数中调用该函数并输出结果。

3.5　C51的常量类型

常量也是C51中常用的数据形式。与变量的表示相比，常量的表示要相对简单，但是同样要注意表示的规范，否则会导致程序出错。C51中的常量有整型常量、浮点型常量、字符型常量和转义字符4种类型。下面分别对其进行介绍。

3.5.1　整型常量

整型数据包括整型变量和整型常量两种，整型变量的定义前面已经作了详细的介绍。整型常量及整型常数可以表示十进制、八进制、十六进制的整数值。根据表示的数的进制不同进行区分，整型常量的表示如表3.4所示。

表3.4　整型常量的表示

整型常量类型	表 示 形 式	示　　例
十进制数	以非0开始的数来表示	220，-560，45 900
八进制数	以0开始的数来表示	06，0 106，0 578
十六进制数	以0X或0x开始的数来表示	0X0D，0XFF，0x4e

说明	整型常量表示十六进制数时，其中的引导符0是必须有的，字母X既可用大写也可用小写，含义相同。

在整型常量后添加一个字母“L”或“l”时，表示该数为长整型数。例如23L、0713L、0Xfd4l等。整型常量在不加特别说明时总是正值。如果需要的是负值，则必须将负号“-”放置于常量表达式的最前面，例如-0x56、-09。整型常量在程序中的使用，示例如下。

```
#include <stdio.h>                                  //头文件

void main()                                         //主函数
{
     int a,b,c;                                     //定义变量
     a=12;                                          //十进制常量赋值给a
     b=012;                                         //八进制常量赋值给b
     c=0x12;                                        //十六进制常量赋值给c
     printf("a=%d,b=%d,c=%d\n",a,b,c);              //输出十进制数据
     printf("a=%o,b=%o,c=%o\n",a,b,c);              //输出八进制数据
     printf("a=%x,b=%x,c=%x\n",a,b,c);              //输出十六进制数据
}
```

这段程序可在Keil μ Vision3集成开发环境中运行，执行结果如下。

```
a=12,b=10,c=18
a=14,b=12,c=22
a=c,b=a,c=12
```

在该程序中，先声明了整型变量a、b和c。接着分别将十进制常量赋值给a，将八进制常量赋值给b，将十六进制常量赋值给c。最后，分别按照十进制、八进制和十六进制的格式输出。

提示	C51中的printf函数中通过格式控制符控制整型变量输出的进制，%d按十进制格式输出，%o按八进制格式输出，%x按十六进制格式输出。

3.5.2 浮点型常量

浮点型常量也称为实型常量，只可以用十进制来表示。一般来说，浮点型常量的值由整数部分、尾数部分和指数部分组成。在不加说明的情况下，浮点型常量通常为正值。如果需要表示负值，则在常量前使用负号，如18.47、-45.63、-4.2e-16、8.165。

在C51中，所有浮点常量都被默认为float型。对于绝对值小于1的浮点型常量，其小数点前面的零可以省略。例如0.68可写为.68，-0.0314E-4可写为-.0314E-4。

在编译环境中，默认的输出格式为浮点数时，最多只保留小数点后六位，不够的后面补零。在浮点型常量中不得出现任何空白符号。字母E或e之前必须有数字，其表示形式为“数字e±数字”，且浮点型常量中E或e后面的指数必须是整数，例如“e2.3”、“e-2.5”等都是不合法的指数形式。

浮点型常量在程序中的使用，示例如下：

```
#include <stdio.h>                                  //头文件

void main()                                         //主函数
```

```
{
     float a,b,c;                                    //浮点型变量
     a=1.2;                                          //赋值
     b=.27;
     c=1.7E-4;
     printf("a=%f,b=%f,c=%f\n",a,b,c);               //输出结果
     printf("%f",2.31);
     while(1);
}
```

这段程序可在Keil μ Vision3集成开发环境中运行，执行结果如下。

```
a=1.200000,b=0.270000,c=0.000170
2.310000
```

在该程序中，先声明了浮点型变量a、b和c。接着分别将浮点型常量赋值给a、b和c。最后输出各个浮点型数据，其中还使用了printf语句来直接输出浮点型常量。

3.5.3 字符型常量

字符型常量用来表示单个字符，其用一对单引号括起来。其中单引号只起定界作用，并不表示字符本身，例如'F'，'5'，'？'。在C51中，字符常量的单引号中的字符不能是单引号（'）或反斜杠（\）。如果需要使用这两个字符，则需要采用转义字符来表示，这将在后面的内容中进行介绍。

在C51中，字符是按其所对应的ASCII码值来存储的，一个字符占一个字节。因此也可用该字符的ASCII码值来表示该字符，例如十进制数65表示大写字母'A'，十六进制数0x5d表示符号']'，八进制数0110表示大写字母'B'等。

另外，格式控制符是无法在程序中显式地表示的。因此，这些格式控制字符不能用符号表示，但可以用ASCII码值来表示。例如，十进制数13表示回车符，八进制数033表示Esc，十六进制数0x0A表示换行符等。

由于C51语言中，字符常量和整型数据（short型）具有相同的取值范围，所以字符常量可以像整型数据一样在程序中进行相关的运算，示例如下。

```
'5'-5;                                               //执行结果53-5 = 48
'B'+32;                                              //执行结果66+32 = 98
'b'－32 ;                                            //执行结果98-32 = 66
```

注意	这里需要强调的是字符'5'和数字5的区别，前者是字符常量，后者是整型常量，它们的含义和在计算机中的存储方式都截然不同。

字符常量的表示以及字符常量的运算程序示例如下。

```
#include <stdio.h>                                   //头文件

void main()                                          //主函数
{
     char c1,c2;                                     //定义字符变量
```

```
    c1='A';                                    //赋值
    c2='B';
    c1=c1+32;                                  //转换大小写
    c2=c2+32;
    printf("c1=%c\nc2=%c",c1,c2);              //输出结果
}
```

这段程序可在Keil μ Vision3集成开发环境中运行，执行结果如下。

```
c1=a
c2=b
```

本例是将两个大写字母转换为小写字母。因为'a'的ASCII码为97，而'A'为65，'b'的ASCII码为98，'B'则为66。从ASCII代码表中可以看到每一个小写字母比大写字母的ASCII码大32，即'a'='A'+32。

3.5.4 转义字符

转义字符用于表示ASCII码字符集中的格式控制字符和特定功能字符，这些字符都是不可打印的。例如，用于表示字符常量的单引号（'），用于表示字符串常量的双引号（"）和反斜杠（\）等。转义字符是C51语言中一种特殊的字符表示形式，用反斜杠“\”后面加一个字符或一个八进制或十六进制数表示，转义字符的表示如表3.5所示。

表3.5 转 义 字 符

转义字符	含　义	ASCII码值(十进制)	转义字符	含　义	ASCII码值(十进制)
\a	响铃（BEL）	007	\\	反斜杠	092
\b	退格（BS）	008	\?	问号字符	063
\f	换页（FF）	012	\'	单引号字符	039
\n	换行（LF）	010	\"	双引号字符	034
\r	回车（CR）	013	\0	空字符（NULL）	000
\t	水平制表（HT）	009	\ddd	任意字符	三位八进制
\v	垂直制表(VT)	011	\xhh	任意字符	二位十六进制

如果需要在字符常量中使用单引号和反斜杠时，都必须使用转义字符表示，即在这些字符前加上反斜杠，例如“\'”、“\\”。

使用转义字符时需要注意以下几点。

- 在C51程序中，不可打印字符必须用转义字符来表示。
- 转义字符中只能使用小写字母来表示，每个转义字符只能看做一个字符。例如“\T”是错误的转义字符。
- “\v”垂直制表和“\f”换页符在屏幕显示时没有任何影响，但是在打印输出时，会影响到打印机的操作。
- 在C51中，可以使用转义字符“\ddd”表示任意字符。其中，“\ddd”为斜杠后面跟三位八进制数，该三位八进制数的值即为对应的八进制ASCII码值。
- 在C51中，可以使用转义字符“\xhh”表示任意字符。其中，“\x”后面跟两位十六进制数，该两位十六进制数为对应字符的十六进制ASCII码值。

技巧	通常使用转义字符表示非显示字符（不可见字符），不过，也可通过\ddd或\xhh的形式来表示可见字符。

3.6　变量作用域

变量作用域是程序中变量起作用的范围。由于C51中可以包含多个函数和程序文件，因此使用变量时，除要首先定义该变量外，还要注意变量的有效作用范围，即该变量的作用域。变量作用域即变量的作用范围，可以是作用于一个函数或一个程序文件，甚至整个工程里的所有文件都可用。一般而言，按照变量的存储类型，变量分为自动变量、全局变量、静态变量和寄存器变量4种类型。下面将分别介绍其作用域范围。

3.6.1　变量作用域的基本规则

在C51语言中，任何以花括号括起来的一段程序称为一个块结构，通常称为复合语句。最典型的块结构是函数或者for、if、do和while等语句。C51中规定，在块结构中进行定义的变量，其有效使用范围只在该块结构内部，示例如下。

```
#include <stdio.h>                          //头文件

void main()                                 //主函数
{
     int i;                                 //在main函数中定义整型变量
     i=15;                                  //变量赋值
     {                                      //块结构
          int t;                            //在块结构中定义整型变量
          t=2;                              //变量赋值
     }
     printf("i=%d",i);                      //输出变量i,正确
     printf("t=%d",t);                      //输出变量t,错误
}
```

其中，整型变量i定义在主函数的大括号内，其使用范围为整个主函数。因此，后面的打印输出i的语句是正确的。而整型变量t定义在块结构内，因此其只在该块结构中使用，出了该块结构便无效了。因此，最后的打印输出t的语句是错误的，在编译程序时，系统将会提示该变量未声明。

由于块结构内部的变量只在其内部有效，因此，即使块结构内定义的变量与块结构外定义的变量具有相同的变量名，它们之间也不会发生冲突。程序示例如下。

```
#include <stdio.h>                          //头文件

void main()                                 //主函数
{
     int i;                                 //定义整型变量
     i=15;                                  //变量赋值
     {                                      //块结构
          int i;                            //定义整型变量
```

```
            i=2;                                    //变量赋值
            printf("in block i=%d\n",i);            //输出变量i
      }
      printf("out block i=%d\n",i);                 //输出变量i
}
```

这段程序可在Keil μ Vision3集成开发环境中运行，程序输出结果如下。

```
in block i=2
out block i=15
```

该程序中，在主函数以及块结构中均定义了整型变量i，但各有不同的作用域。在块结构内部，其内部定义的变量i起作用，外部定义的变量被屏蔽，因此程序输出i为2。在块结构外部，是主函数定义变量的有效范围，块结构内部的变量将消失，因此程序输出i为15。

3.6.2 自动变量

自动变量一般是在函数的内部或者程序块中使用，是以关键字auto标识的变量类型。其定义格式为“ [auto]类型说明符 变量标识符，变量标识符... ”。自动变量的作用域范围是函数或者程序块的内部。

在编译C51程序时，自动变量根据变量类型动态分配存储空间。在程序执行到该函数时，根据变量类型为其自动分配存储空间，当该函数执行完毕后，立即取消该变量的存储空间，即该自动变量失效。这样在该函数内部定义的变量，就不能在该函数外引用。使用自动变量的程序示例如下。

```
#include <stdio.h>                              //头文件

void main()                                     //主函数
{
      int i=8;                                                  //定义整型变量
      int t=6;                                                  //定义整型变量
      {                                                         //程序块1开始
            int i=15;                                           //定义整型变量
            int t=25;                                           //定义整型变量
            {                                                   //程序块2开始
                  int i=20;                                     //定义整型变量
                  int t=30;                                     //定义整型变量
                  printf("i=%d,t=%d\n",i,t);                    //输出
            }                                                   //程序块2结束
            printf("i=%d,t=%d\n",i,t);                          //输出
      }                                                         //程序块1结束
      printf("i=%d,t=%d",i,t);                                  //输出
}
```

这段程序可在Keil μ Vision3集成开发环境中运行，执行结果如下。

```
i=20,t=30
i=15,t=25
i=8,t=6
```

在该程序中，主函数声明了整型变量i和t，然后分别嵌套两个块结构的复合语句，其中分别定义并初始化变量i和t。根据前面的介绍，这里定义的各个变量都默认为自动变量，虽然变量名相同，其作用域仅限于函数内部和块结构内部，不会影响外部的变量。

> 说明　在C51中，函数或程序块内部定义的变量，一般都默认为自动型变量。因此，在不声明自动型变量时，关键字auto一般都可以省略。

3.6.3　全局变量

全局变量一般定义在所有函数的外部，即整个程序文件的最前面，也称为外部变量。全局变量的作用域是整个程序文件，即全局变量可以被该程序文件中的任何函数使用。

在编译C51程序时，全局变量根据变量类型被静态地分配适当的存储空间。在整个程序运行过程中，该变量一旦分配空间，便不会消失。这样全局变量对整个程序文件都有效。

全局变量是永久性的，因此全局变量可以作为不同函数间的参数进行传递和共享。全局变量的程序示例如下。

```
#include <stdio.h>                              //头文件

double PI=3.14159;                              //圆周率PI
int r;                                          //整型全局变量,表示半径

double FunctionS()                              //计算面积函数
{
     double S;
     S=r*r*PI;                                  //计算圆的面积
     return S;                                  //返回圆的面积
}
void main()                                             //主函数
{
     r=4;                                               //半径赋值
     printf("If r=%d, S=%f\n",r, FunctionS());          //输出圆的面积
}
```

这段程序可在Keil μ Vision3集成开发环境中运行，执行结果如下。

```
If r=4, S=50.265440
```

该程序在源文件的头部定义了浮点型的全局变量PI表示圆周率、整型的全局变量r表示圆的半径。整型变量r在主函数中初始化，然后在FunctionS()函数中调用，用来计算并返回圆的面积。

对于比较复杂的C51程序，其一般以工程项目的形式来组织多个程序文件。此时，全局变量允许在一个程序文件中定义，而同时可以在另一个程序文件中使用。为了方便使用，需要在文件的头部用关键字“ extern ”来对全局变量进行引用声明。这样该C51项目在编译的时候，可以自动在其他程序文件中寻找该全局变量，从而知道该全局变量的数据类型和值。

上面的程序可以采用这种方法来实现，将程序分为两个文件。FunctionS.c程序文件代码如下。

```
#include <stdio.h>                                              //头文件

double PI=3.14159;
```

```
int r;

double FunctionS()                                          //计算面积函数
{
     double S;
     S=r*r*PI;                                              //计算圆的面积
     return S;                                              //返回圆的面积
}
```

主程序main.c文件代码如下：

```
#include <stdio.h>                                          //头文件

extern PI;
extern r;

double FunctionS();

void main()                                                 //主函数
{
     r=4;                                                   //半径赋值
     printf("If r=%d, S=%f\n",r, FunctionS());              //输出圆的面积
}
```

这段程序可在Keil μ Vision3集成开发环境中运行，执行结果如下。

```
If r=4, S=50.265440
```

该程序实现的功能和前面是一样的，只不过分为两个程序文件来实现。在FunctionS.c程序文件中定义了浮点型的全局变量PI表示圆周率、整型的全局变量r表示圆的半径。在主程序文件main.c中使用这两个变量前，首先用extern关键字加以声明，这样便可以使用FunctionS.c文件中定义的全局变量PI和r了。

注意	一个全局变量的定义在一个程序中只能出现一次，而在函数中，说明对全局变量的引用可出现多次。

3.6.4 静态变量

静态变量即在编译C51程序时，根据数据类型静态地分配合适的存储空间，并在程序运行过程中始终占有该存储空间的变量。以关键字static定义，其定义格式为“ static 类型说明符变量标识符，变量标识符... ”，示例如下。

```
static   int   i;
static   char   c;
```

在C51语言中，根据变量声明位置的不同，静态变量可以分为如下两种。

- 内部静态变量，即在函数内部定义，其作用域只是定义该变量的函数内部，和自动变量类似。
- 外部静态变量，即在函数外部定义，其始终占有内存空间，和全局变量类似。

除了静态变量外，C51语言还允许将自定义函数定义为静态型，同样用static关键字来定义。这样，只有同一程序文件中的其他函数才能调用这个静态型函数，而工程项目中的其他程序文件则不能调用访问。使用静态型函数既有利于程序的模块化设计，又可以防止和其他文件中的函数发生重名的情况。

3.6.5　寄存器变量

寄存器变量被存储在CPU的寄存器中。寄存器变量以关键字register声明，声明格式为“ register 类型说明符 变量标识符，变量标识符... ”，示例如下。

```
register char   ch;
register double db;
```

由于寄存器变量被存储在CPU的寄存器中，因此其读写速度比较快。寄存器变量常用于某一变量名频繁使用的情况，这样做可以提高系统的运算速度。

在使用寄存器变量时需要注意如下两点。

- 由于单片机内部寄存器数量有限，不能定义多个寄存器变量。在实际程序设计中，应将最重要的变量设置为寄存器变量，以提高系统执行速度。
- 在C51中，只允许同时定义两个寄存器变量，如果多于两个，程序在编译时会自动地将两个以外的寄存器变量作为非寄存器变量来进行处理。

提示	使用register关键字，只是提示编译器将变量定义为寄存器变量，而不是“ 命令 ”编译器必须将指定变量作为寄存器变量。这是因为CPU的寄存器数量是有限的，当没有寄存器可使用时，编译器会将其作为自动变量来进行处理。

3.7　分隔符与const修饰符

分隔符和修饰符是计算机语言中重要的组成部分，下面分别介绍C51的分隔符和const修饰符。

3.7.1　C51分隔符

分隔符在C51程序语言中起辅助标识和分界的作用，如+、 - 、*、/、!、|、<、>、=、.、,、:、;、'、#等。这些简单分隔符是由除字母、数字、下划线以外的单个ASCII字符组成的符号。除此以外，C51中还有复合分隔符，例如<=、!=、>=、{、}、[、]、(、)等。复合分隔符是特定字符的某种组合。

C51的分隔符常用于数组、结构、指针以及语句等情况，下面举出常用的分隔符示例。

- “ ; ”分隔符：主要用于语句的末尾，起到隔离语句的作用，例如例如int r;。
- “ [ ] ”分隔符：主要用于对数组说明，例如char ch[ ]="Hello Everyone!"。
- “ ( ) ”分隔符：主要用于进行隔离，例如s=area(r)。
- “ , ”分隔符：主要用于进行隔离，例如char ch1,ch2。
- “ {} ”分隔符：主要用于进行复合语句的隔离，例如for {i=0;i<10;i++}。
- “ * ”分隔符：主要用于做指针，例如char *p1。
- “ # ”分隔符：主要用于做预处理伪指令，例如#define PI=3.14159。

❑ “^”分隔符：主要用于标识特殊寄存器的位，例如sbit P10=P1^0。

3.7.2 const修饰符

在C51语言中，用关键字const修饰的是一类特殊的常量，一般称为符号常量或const变量。const修饰符主要用来定义常量或变量。其定义格式为“const <类型说明符> <常量名> =<常量值>;”，示例如下。

```
const double PI=3.14159;                          //定义浮点型的const变量
```

基本数据类型的变量一旦加上const修饰符，程序在编译时，将其视为一个常量，而不再为其分配内存。当在程序中遇到该const变量时，将用其定义时的初值来代替，不能在程序中修改const修饰的变量的值。所以在声明const变量时，必须对其进行初始化赋值，除非该变量是用extern修饰的全局变量。

在C51程序中使用const变量，具有如下两点好处。

❑ 可以防止程序运行时该值被意外修改。

❑ 可以方便对于程序中经常使用的值进行统一修改，便于调试程序。

const变量的程序示例如下。

```
#include <stdio.h>                                        //头文件

const double PI=3.14159;                                  //定义圆周率为const变量

void main()                                               //主函数
{
     int r;                                               //定义半径
     double area,length;                                  //定义圆面积和周长
     r=2;                                                 //半径赋值
     area=r*r*PI;                                         //计算面积
     length=2*PI*r;                                       //计算周长
     printf("If r=%d,the area=%lf,length=%lf\n",r,area,length);   //输出结果
}
```

这段程序可在Keil μ Vision3集成开发环境中运行，执行结果如下。

```
If r=2,the area=12.566360,length=12.566360
```

该程序中，将圆周率PI定义为double型的const变量。主程序编译和执行时，如果遇到该变量则自动以double型的值3.14159来代替。

在C51语言中，宏定义和const变量有些类似。宏定义在程序预处理时只对上下文进行简单的文本代替，并不进行具体的语法检查，示例如下。

```
#define ch 12;
```

对于上面的宏定义语句，C51程序在编译时，如果遇到字符串ch，则将其全部替换为字符串12。至于ch的数据类型则没有指明，因此在使用时会出现一些问题，例如不知道该变量是整型数据还是浮点型数据，就很容易引起混乱。如果使用const变量来定义这个常量，可以进行如下定义。

```
const int       ch=12;                          //定义ch为整型const变量
const double    ch=12;                          //定义ch为浮点型const变量
```

这样，在定义时便指明了其数据类型。因此，在这种情况下使用const变量要比宏定义简单明了。使用const变量可以完全代替无参数的宏。

说明　宏定义在后面的章节中会具体进行介绍，这里仅指出了其和const变量的区别。

3.8　C51的运算符

运算符是表示特定的算术或逻辑操作的符号，也称为操作符。例如，“ * ”号表示了一个乘法运算符；“ && ”号表示了一个逻辑与的运算符。在C51语言中，需要进行运算的各个量（常量或变量）通过运算符连接起来便构成了一个表达式。本节首先介绍运算符，表达式将在3.9节进行介绍。

C51语言中主要有算术运算符、关系运算符、逻辑运算符和位运算符这四类运算符，还有些用于辅助完成复杂功能的特殊运算符，如“ , ”运算符、“ ? ” 运算符、地址操作运算符、联合操作运算符、“ sizeof ”运算符、类型转换运算符等。使用特殊运算符可以起到简化程序的作用。下面对各种运算符的含义和用法分别进行介绍。

3.8.1　算术运算符

算术运算符是用来进行算术运算的操作符。C51语言中的算术运算符继承了其他高级计算机语言的特点，用法也基本一致。C51语言中的算术运算符有如下所示的几种。

- “ - ”运算符：进行减法或取负的运算。
- “ + ”运算符：进行加法运算。
- “ * ”运算符：进行乘法运算。
- “ / ”运算符：进行除法运算。
- “ % ”运算符：进行模运算。
- “ - - ”运算符：进行自减（减1）运算。
- “ ++ ”运算符：进行自增（增1）运算。

1. 普通算术运算符

普通算术运算符是指加“ + ”、减“ - ”、乘“ * ”、除“ / ”以及模运算“ % ”，示例如下。

```
3+23=26;
17-5=12;
4*5=20;
6/3=2;
43%2=1;
```

这些普通算术运算符的运算操作和其他高级语言的运算相类似，需要注意的是以下几个运算符在操作中的不同之处。

- 除法运算符“ / ”的运算结果是取除法结果的整数部分。例如，“ 10/4=2 ”，结果取商2.5的整数部分，值为2。

- 取模运算符“ % ”的运算结果是取除法结果的余数部分。该运算符不能用于浮点型数据的运算操作。例如，“ 9%4=1 ”，结果取商9 - 4*2=1的余数部分，值为1。
- 减法运算符“ - ”除进行减法运算外，还可以用来进行取负运算操作。例如，“ - sz ”是取变量sz的负操作。

普通算术运算符的使用程序示例如下。

```
#include <stdio.h>                                          //头文件

void main()                                                 //主函数
{
     int x,y,z;                                             //定义整型变量
     x=30;y=12;                                             //赋初值
     z=x+y;                                                 //加法运算
     printf("x+y=%d\n",z);                                  //输出结果
     z=x-y;                                                 //减法运算
     printf("x-y=%d\n",z);                                  //输出结果
     z=x*y;                                                 //乘法运算
     printf("x*y=%d\n",z);                                  //输出结果
     z=x/y;                                                 //除法运算
     printf("x/y=%d\n",z);                                  //输出结果
     z=x%y;                                                 //取模运算
     printf("x%%y=%d\n",z);                                 //输出结果
     z=-x;                                                  //取负运算
     printf("-x =%d\n",z);                                  //输出结果

}
```

该程序可在Keil μ Vision3集成开发环境中运行，其执行结果如下。

```
x+y=42
x-y=18
x*y=360
x/y=2
x%y=6
-x=-30
```

2. 自增和自减运算

自增运算符“ ++ ”表示操作数加1，即x++等同于x=x+1；自减运算符“ - - ”表示操作数减1，即x - -等同于x=x - 1。这两个很常用的运算符是沿用了C语言的特点。

自增和自减运算符既可放在操作数之前，也可放在其后。例如x=x+1，可写成++x，也可以写成x++，但在表达式中这两种用法是有区别的。自增或自减运算符放在操作数之前时，C51语言在引用操作数之前就先执行加1或减1操作；运算符在操作数之后时，C51语言就先引用操作数的值，而后再进行加1或减1操作，示例如下。

```
x=++m;                                                      //m先增加1,然后赋值给x
x=m++;                                                      //m先赋值给x,然后再增加1
```

> 说明 在大多数的编译环境中，采用自增和自减操作符所生成的程序代码比等价的赋值语句所生成的代码执行起来要快得多，因此推荐采用自增和自减运算符。

自增和自减运算符的程序示例如下。

```
#include <stdio.h>                              //头文件

void main()                                     //主函数
{
int x,y,z1,z2;                                  //定义整型变量
x=10;y=21;                                      //赋初值
z1=(x++)+(x++);
printf("x=%d,z1=%d\n",x,z1);                    //输出结果
z2=(++y)+(++y);
printf("y=%d,z2=%d",y,z2);                      //输出结果
}
```

这段程序可在Keil μ Vision3集成开发环境中运行，执行结果如下。

```
x=12,z1=23
y=23,z2=45
```

在该程序中，计算z1时，对于第一个括号先取x的值，然后x增1，此后x变为11；对于第二个括号同样先取x的值11，然后再增1，此后x变为12，因此最后x为12，z1为11+12=23。计算z2时，对于第一个括号y值，首先增1，然后取y的值，此后y变为22；对于第二个括号，同样y值先增1，然后取y的值，此后y变为23，因此最后y为23，z2为22+23=45。

> 说明 在C51程序中，一般按照从左向右的运算顺序，关于运算符的运算优先级别的内容将在后面章节中作详细的介绍。

3.8.2 逻辑运算符

逻辑运算符是进行逻辑运算的操作符。C51语言中的逻辑运算符如下所示。

- “！”运算符：进行逻辑非运算。
- “||”运算符：进行逻辑或运算。
- “&&”运算符：进行逻辑与运算。

逻辑运算符的操作对象可以是整型数据、浮点型数据以及字符型数据。如果逻辑运算符的操作结果是真，则运算结果为1；如果是假，则运算结果为0。逻辑运算符的逻辑真值如表3.6所示。

表3.6 逻辑运算符的真值表

A	B	A&&B	A||B	!A
0	0	0	0	1
0	1	0	1	1
1	0	0	1	0
1	1	1	1	0

逻辑运算符运算的程序示例如下。

```
#include <stdio.h>                                  //头文件

void main()                                         //主函数
{
     int a,b,c,d,e;                                 //定义整型变量,并存放逻辑运算结果
     a=!0;                                          //逻辑非运算
     b=15&&22;                                      //逻辑与运算
     c=35&&0;                                       //逻辑与运算
     d=17.3||0;                                     //逻辑或运算
     e=17.3||2.6;                                   //逻辑或运算
     printf("a=%d,b=%d,c=%d,d=%d,e=%d\n",a,b,c,d,e); //输出结果
}
```

该程序可在Keil μ Vision3集成开发环境中运行，执行结果如下。

```
a=1,b=1,c=0,d=1,e=1
```

说明　在C51语言中规定，非零的操作数都被视为是真，为零的操作数都被视为是假。

3.8.3　关系运算符

关系运算符主要用于比较操作数的大小关系，和一般的C语言相类似。常用的关系运算符如下所示。

- “ > ”运算符：判断是否大于。
- “ >= ”运算符：判断是否大于等于。
- “ < ”运算符：判断是否小于。
- “ <= ”运算符：判断是否小于等于。
- “ == ”运算符：判断是否等于。
- “ != ”运算符：判断是否不等于。

关系运算符和逻辑运算符在程序运算中常常在一起联合使用。如果关系运算符的操作结果是真，则运算结果为1；如果是假，则运算结果为0。关系运算符运算的程序示例如下。

```
#include <stdio.h>                                  //头文件

void main()                                         //主函数
{
     int a,b,c,d;                                   //定义整数变量,存储结果
     a=-2.3>=0;                                     //比较运算
     b=71==32;                                      //比较运算
     c=7!=0;                                        //比较运算
     d=-12<=0;                                      //比较运算
     printf("a=%d,b=%d,c=%d,d=%d\n",a,b,c,d);       //输出结果
}
```

该程序可在Keil μ Vision3集成开发环境中运行，执行结果如下。

```
a=0,b=0,c=1,d=1
```

说明	逻辑运算符和关系运算符的返回值都是True（真）和False（假）。在C51语言中规定，非0的值为True，0值为False。使用关系或逻辑运算符的表达式时，若表达式为True（真）则返回值为1；若表达式为False（假）则返回值为0。

3.8.4　位运算符

位运算符是对字节或字的二进制位（bit）进行逐位逻辑处理或移位的运算符。C51语言中的位运算符如下所示。

- “ & ”运算符：进行逻辑与（AND）运算。
- “ | ”运算符：进行逻辑或（OR）运算。
- “ ^ ”运算符：进行逻辑异或（XOR）运算。
- “ ~ ”运算符：进行按位取补（NOT）运算。
- “ >> ”运算符：进行右移运算。
- “ << ”运算符：进行左移运算。

位运算符的操作对象整型和字符型数据的字节或字，位操作不能用于float、double、long double、void或其他聚合类型。支持全部的位运算符，表明C51可以进行汇编语言所具有的位运算，因此C51语言既具有高级语言的特点，也具有低级语言的功能。

位运算中的AND、OR和NOT（1的补码）的真值表与逻辑运算等价，唯一不同的是，位操作是首先将操作数分解为二进制，然后逐位进行运算的。下面分别介绍各种位运算符的用法。

1. 按位与运算符

按位与运算符是将两个操作数按二进制展开，然后将对应位进行逻辑与运算。按位与运算符“ & ”是二目运算符，即要求有两个源操作数。如果操作数对应的二进制位均为1，则逻辑与的结果为1；否则，则逻辑与的结果为0。

例如，将a=56和b=37进行按位与运算。将56展开为二进制是00111000，将37展开为二进制是00100101。按位与运算的结果为a&b=32（00100000）。

注意	这里需要区分位运算符“ & ”和关系逻辑运算符“ && ”，关系逻辑运算符“ && ”的结果不是0就是1。而位运算符“ & ”的结果通过逐位处理，结果可为0和1之外的其他值。例如a=56和b=37，a&&b=1，而a&b=32。

2. 按位或运算符

按位或运算符是将两个操作数按二进制展开，然后将对应位进行逻辑或运算。按位或运算符“ | ”也是二目运算符，即要求有两个源操作数。如果对应的二进制位均为0，则逻辑或的结果为0；否则，则逻辑或的结果为1。

例如，将a=56和b=37进行按位或运算。将56展开为二进制是00111000，将37展开为二进制是00100101。按位或运算的结果a|b=61（00111101）。

3. 按位非运算符

按位非运算符是将操作数按二进制展开，然后将每一位进行取反操作，即将0变为1，将1

变为0。按位非运算符“~”是一目运算符，只需要一个源操作数。

例如，将a=56进行按位取反运算。56的二进制表示为00111000，结果~a=199（11000111）。

说明	按位非运算得到的结果和源操作数是逐位相反的，因此这两者进行按位或运算的结果每个二进制位均为1，即对应十进制255。这样可以通过255减去源操作数得到按位非运算的结果。例如56按位非的结果为255 - 56=199，和上面的结果相同。

4. 按位异或运算符

按位异或运算符是将两个操作数按二进制展开，然后将对应位进行逻辑异或运算。按位异或运算符“^”也是二目运算符，需要两个源操作数。如果对应的二进制位均为0或均为1，则逻辑或的结果为0；否则，只要对应位不相同，则逻辑异或的结果为1。异或运算的逻辑真值如表3.7所示。

表3.7 异或的真值表

A	B	A^B
0	0	0
0	1	1
1	0	1
1	1	0

例如，将a=56和b=37进行按位异或运算。将56展开为二进制是00111000，将37展开为二进制是00100101。按位异或运算的结果a|b=29（00011101）。

提示	在有些高级程序设计语言中，将运算符“^”作为求幂运算，要注意区分。

5. 移位运算符

移位运算符是将源操作数按二进制展开，然后将对应位按要求进行移动。C51中移位运算符有两种，分别将操作数的各位按要求向左或向右移动，其使用格式如下。

- 左移语句形式是：变量名（或操作数）<<左移位数
- 右移语句形式是：变量名（或操作数）>>右移位数

在C51程序中使用移位运算符需要注意如下三点。

- C51语言中的移位不是循环移位，当某位从一端移出时，另一端移入0。从一端移出的位并不送回到另一端去，移去的位永远丢失了，同时在另一端补0。
- 右移运算，对无符号位数左端补0，称为“逻辑右移”；如果为负数，即符号位为1，则左端补1，这种补1保持负号的方法称为“算术右移”。
- 左移运算有时可用来做乘法，右移运算有时可用来做除法。例如，x=7，将其进行移位操作，结果如表3.8所示。

从表中可以看出，每左移一位，结果乘2，每右移一位相当于被2除。注意表中x<<2后，原x的信息已经丢失了，因为一位“1”已经从一端移出。因此，需要注意这种用法的范围。

6. 程序示例

前面具体介绍了各个位运算符，这里举例来演示位运算符在程序中的具体应用。位运算符的程序示例如下。

表3.8 移位操作进行乘和除

字符x	每个语句执行后的x	x的值
x=7	00000111	7
x<<1	00001110	14
x<<3	01110000	112
x<<2	11000000	192
x>>1	01100000	96
x>>2	00011000	24

```
#include <stdio.h>                                          //头文件

void main()                                                 //主函数
{
     int a,b,c;                                             //定义变量
     a=23;                                                  //变量赋值
     b=217;
     c=a&b;                                                 //按位与运算
     printf("a&b=%d\n",c);                                  //输出结果
     c=a|b;                                                 //按位或运算
     printf("a|b=%d\n",c);                                  //输出结果
     c=a^b;                                                 //按位异或运算
     printf("a^b=%d\n",c);                                  //输出结果
     c=~a;                                                  //按位取反运算
     printf("~a=%d\n",c);                                   //输出结果
     c=a<<3;                                                //左移两位
     printf("a<<3=%d\n",c);                                 //输出结果
     c=b>>2;                                                //右移三位
     printf("b>>2=%d\n",c);                                 //输出结果
}
```

该程序可在Keil μ Vision3集成开发环境中运行，执行结果如下。

```
a&b=17
a|b=223
a^b=206
~a=-24
a<<3=184
b>>2=54
```

读者可以参考前面介绍的内容，分析一下程序运行的结果和预期的是否一致。读者同样可以采用不同数进行运算，从而熟练掌握位算符的操作。

3.8.5 “,”运算符

“,”运算符是把几个表达式串在一起，并用括号括起来，按照顺序从左向右计算的运算符。“,”运算符左侧表达式的值不作为返回值，只有最右侧表达式的值作为整个表达式的返回值。程序示例如下。

```
#include <stdio.h>                                    //头文件

void main()                                           //主函数
{
     int a,b,c;                                       //定义变量
     a=37;                                            //赋值
     b=179;
     c=(a++,++b,b+a);                                 //执行","运算符,为c赋值
     printf("c=%d\n",c);                              //输出结果
}
```

该程序可在Keil μ Vision3集成开发环境中运行，执行结果如下。

```
c=218
```

本例中首先执行a++，a自增1，然后执行++b，b自增1，最后执行b+a，并将结果赋给变量c。

技巧	逗号运算符可以将多个表达式串联起来，从而构成语法上的“一个”表达式。这在语法上要求出现“一个”表达式时很有用（如for语句中的初始化、判断、改变循环变量这三部分都只能出现一个表达式，这时，就可使用逗号表达式来进行多个变量的赋值和运算操作）。但需要注意，一般不是特别需要，不建议读者滥用逗号表达式，因为其最终结果初学者不好确认，应用程序容易出现不易查找的错误。

3.8.6 “？”运算符

“？”运算符首先计算表达式1的值，然后根据表达式1的真假接着计算其余表达式的值，并输出结果。“？”运算符是三目操作符，其一般形式如下。

```
EXP1?EXE2:EXP3;
```

其中，EXP1、EXP2和EXP3是表达式。“？”运算符作用是在计算表达式EXP1的值后，如果其值为真，则计算表达式EXP2的值，并将其结果作为整个表达式的结果；如果表达式EXP1的值为假，则计算表达式EXP3的值，并将结果作为整个表达式的结果。“？”运算符的程序示例如下。

```
#include <stdio.h>                                    //头文件
void main()                                           //主函数
{
int x,y;                                              //定义变量
x=17;                                                 //赋值
y=x>5?10:20;                                          //使用"?"运算符,为y赋值
printf("y=%d\n",y);                                   //输出结果
}
```

该程序可在Keil μ Vision3集成开发环境中运行，执行结果如下。

```
y=10
```

本例中，首先判断x>5，其值为真，所以执行第一个表达式将10赋给y。若用if-else语句改写，有下面的等价程序。

```
#include <stdio.h>                                    //头文件

void main()                                           //主函数
{
     int x,y;                                         //定义变量
     x=17;                                            //赋值
     if (x>5)                                         //使用if语句,为y赋值
     {
          y=10;
     }
     else
     {
          y=20;
     }
     printf("y=%d\n",y);                              //输出结果
}
```

这段程序同样可以在Keil μ Vision3集成开发环境中运行，执行结果和前面相同。比较两程序，从中可以看出使用“？”运算符，可以大大简化程序。

3.8.7 “sizeof”运算符

“sizeof”运算符返回变量所占的字节或类型长度字节。“sizeof”运算符是单目操作符。在C51语言中，“sizeof”运算符类似于C语言中length函数。使用“sizeof”运算符的程序示例如下。

```
#include <stdio.h>                                    //头文件

void main()                                           //主函数
{
 char ch[]="hello everyone!";                         //定义字符串数组
 int i,j;                                             //定义整型变量
 i=sizeof(ch);                                        //获取字符串数组的长度
 j=sizeof(float);                                     //获取float类型数据的长度
 printf("i=%d\nj=%d\n",i,j);                          //输出结果
}
```

该程序可在Keil μ Vision3集成开发环境中运行，执行结果如下。

```
i=16
j=4
```

这段程序首先定义并初始化字符串数组，然后获取该字符串所占的长度，包括最后的空字符。程序中还使用sizeof运算符来获取float数据类型的长度。

> 提示 看起来sizeof不像是一个运算符，更像是一个函数。但sizeof确实是C51语言中的一个运算符。

3.8.8 地址操作运算符

地址操作运算符用来对变量的地址进行操作。在C51语言中，地址操作运算符主要有两种：

“ * ”和“ & ”。其中，“ * ”运算符是单目操作符，其返回位于某个地址内存储的变量值；“ & ”运算符也是一个单目操作符，其返回操作数的地址。“ * ”运算符和“ & ”运算符是相对应的。程序示例如下。

```
#include <stdio.h>                                    //头文件

main()                                                //主函数
{
char ch1,ch2;                                         //定义字符型变量
char *p;                                              //定义指针型变量
ch1='A';                                              //为字符型变量ch1赋值
p=&ch1;                                               //将变量ch1的地址赋给p
ch2=*p;                                               //地址p所指的单元值赋给ch2
printf("ch2=%c\n",ch2);                               //输出ch2
}
```

该程序可在Keil μ Vision3集成开发环境中运行，执行结果如下。

```
ch2=A
```

本例中，字符型变量ch1赋值为字符'A'，然后将其的地址赋给指针型变量p，最后将地址p中的数值赋给变量ch2，这样变量ch2便有了和变量ch1同样的内容，因此ch2输出为字符'A'。

3.8.9 联合操作运算符

联合操作运算符主要用来简化一些特殊的赋值语句，这类赋值语句的一般形式如下。

```
<变量1>=<变量1><操作符><表达式>
```

利用联合操作运算符可以简化为如下形式。

```
<变量1><操作符>=<表达式>
```

联合操作运算符适合于所有的双目操作符。C51语言中常用的联合操作运算符示例如下。

- a+=b，相当于a=a+b。
- a*=b，相当于a=a*b。
- a&=b，相当于a=a&b。
- a|=b，相当于a=a|b。
- a/=x+y - z，相当于a=a/(x+y - z)。

技巧	初学者在学习使用联合操作运算符时，可能有些不习惯，但是这种表达方式比较简练，易于维护。C51编译器对这类代码进行优化，可生成高质量的目标代码。

3.8.10 类型转换运算符

类型转换运算符用于强制使某一表达式的结果变为特定数据类型。类型转换运算符的一般形式如下所示。

```
（类型）表达式
```

其中，“（类型）”中的类型必须是C51中的一种数据类型。类型转换运算符的使用示例如下。

```
(float)x/2                                          //将x/2的结果转换为浮点型
```

在C51语言中，“/”运算的结果将取其整数，为确保表达式x/2具有准确的结果，所以使用类型转换运算符强制运算结果转换为浮点型数据。类型转换运算符的程序示例如下。

```
#include <stdio.h>                                  //头文件

void main()                                         //主函数
{
 int i;                                             //定义整型循环变量
 for(i=0;i<7;i++)                                   //for循环语句
 {
          printf("%d/3=%f\n",i,(float)i/3);         //循环输出i/3的数值
   }
}
```

该程序可在Keil μ Vision3集成开发环境中运行，执行结果如下。

```
0/3=0.000000
1/3=0.333333
2/3=0.666667
3/3=1.000000
4/3=1.333333
5/3=1.666667
6/3=2.000000
```

本例中，将从0到6的整数进行除法运算，由于“/”运算符只取商的整数部分，为了结果的准确性，使用类型转换运算符，强制将运算结果转换为浮点型，这样将得到准确的计算结果。

3.8.11 运算符优先级和结合性

在C51语言中，当一个表达式中有多个运算符参与运算时，要按照运算符的优先级别进行运算。在复杂的表达式中，除了要判断运算优先级外，还要考虑结合性或者关联性。

1. 算术运算符的优先级

算术运算符的优先级由高到低依次为自增自减（++、--）和取负（-）、乘法除法（*、/）和取模（%）、加和减（+、-）。

这里需要强调的是，在C51程序编译时对同级运算符一般按从左到右的顺序进行计算，由于括号的优先级最高，所以括号会改变计算顺序。

2. 关系运算符和逻辑运算符的优先级

关系运算符和逻辑运算符的相对优先级最高的是！，其次是>、<、>=和<=，然后是==和!=，后面是&&，最后是||。关系运算符和逻辑运算符的优先级符合以下两点（表3.9）。

- 关系和逻辑运算符的优先级比算术运算符低，像表达式10>1+12和表达式10>(1+12)计算的结果是一样的。
- 在关系或逻辑表达式中可以使用括号来修改原计算的优先级顺序。

3. 运算符的结合性

在一个复杂的表达式中，常常有许多运算符和变量，除了要判断优先级外，还要考虑结合

性。C51语言中具有“左结合性”和“右结合性”的概念。左结合性即变量（或常量）与左边的运算符结合。右结合性即变量（或常量）与右边的运算符结合。示例如下。

```
-6+23;
```

表3.9 运算符的优先级和结合性

优先级	运算符（高→低）	结合性	运算符类型
（高	( ) [ ] -> .	从左至右	双目运算符
	! ~ ++ - - sizeof + - * & （类型）	从右至左	单目运算符
	* / %	从左至右	双目运算符
	+ -	从左至右	双目运算符
	<< >>	从左至右	双目运算符
	< <= > >=	从左至右	双目运算符
	== !=	从左至右	双目运算符
	&	从左至右	双目运算符
	^	从左至右	双目运算符
	|	从左至右	双目运算符
	&&	从左至右	双目运算符
	||	从左至右	双目运算符
	?:	从右至左	三目运算符
低）	= += -= *= /= %= &= ^= |= <<= >>=	从左至右	双目运算符

其中，运算符“-”和“+”相对于运算的操作数来说是“左”结合的，所以实际参与计算的是“-6”和“+23”，即相当于（-6）+（23）运算的结果为17。

运算符优先级和结合性的顺序如表3.9所示，表中优先级从上往下逐渐降低，同一行优先级从左往右逐渐降低。

从表中可以看出，C51语言中具有右结合性的运算符包括所有单目运算符以及三目运算符，其他都是左结合性，知道这个规律可以很方便地记住运算符的结合性。

为了程序移植、防止歧义和阅读的方便，在实际程序设计中，如果代码行中的运算符比较多，应当多用括号确定表达式的操作顺序，避免使用默认的优先级。示例如下。

```
(num<<4)|a                                  //用括号把需要先运算的括起来
(a|c)&&(b&c)                                //用括号把需要先运算的括起来
```

4. 程序示例

关于运算符优先级和结合性的程序示例如下。

```
#include<stdio.h>                           //头文件

void main()                                 //主函数
{
int a=5;                                    //定义整型变量
a%=9-2;                                     //联合操作
printf("a=%d\n",a);                         //输出结果
a+=a/=a*=2;                                 //复杂的运算
```

```
printf("a=%d\n",a);                                    //输出结果
}
```

该程序可在Keil μ Vision3集成开发环境中运行，执行结果如下。

```
a=5
a=0
```

本例中，由于“ % ”和“ = ”运算符的优先级别低于“ - - ”运算符，a%=9 - 2即a%=7，等价于a=a%7即a=5%7=5。表达式a+=a/=a*=2，开始时a=5，表达式赋值是从左至右进行的，表达a*=2使得a=10，此表达式的值也为10。接着a/=a*=2。相当于a/=10，也就是a=a/10，因此，a=0。最后，表达式a+=a/=a*=2相当于a+=0，也就是a=a+0=0，所以最终a的值为0。

3.9 C51的表达式

表达式是需要进行运算的各个量由运算符连接起来而构成的一个整体。表达式是由操作数和运算符组成的，其中操作数一般包括常量和变量，甚至也可以包括函数和表达式等。表达式也是C51语言中的基本组成部分。C51中主要有算术表达式、赋值表达式、逗号表达式、关系表达式和逻辑表达式5种表达式，下面分别对其进行介绍。

3.9.1 算术表达式

算术表达式是指用算术运算符将操作数连接起来的式子，其中也可以使用括号，例如(a - (b+c)*3)/2 - 12。算术表达式虽然比较简单，但是在使用时要注意算术运算符的计算优先级和结合性，否则很容易使程序出现错误。算术表达式的程序示例如下。

```
#include <stdio.h>                                     //头文件

void main()                                            //主函数
{
     int i,j,x,y;                                      //声明变量
     i=105;                                            //赋值
     j=2;
     x=i+j*2;                                          //算术运算
     y=(i-j*2)%3;                                      //算术运算
     printf("x=%d\ny=%d\n",x,y);                       //输出结果
}
```

该程序可在Keil μ Vision3集成开发环境中运行，执行结果如下。

```
x=105
i-j=2
```

在该程序中，首先为变量i和j赋值，然后分别通过算术表达式来对x和y赋值，最后输出算术表达式的计算结果。

3.9.2 赋值表达式

赋值表达式是指，由赋值运算符“ = ”将一个变量和一个常量或者表达式连接起来的式子。赋值表达式的一般形式介绍如下。

```
<变量><赋值运算符><表达式>
```

例如，“a=27”就是一个简单的赋值表达式，表示将常量27赋值给变量a。赋值表达式在执行时，首先求解赋值运算符右边的表达式的值，然后将该值赋给左边的变量。赋值表达式的程序示例如下。

```
#include <stdio.h>                                          //头文件

void main()                                                 //主函数
{
     int i,j;                                               //声明变量
     char a,b;
     float f;
     i=12;                                                  //赋值
     j=i-5;
     a='A';                                                 //变量赋值
     b=a+1;
     f=17.5/2;
     printf("i=%d\nj=%d\n",i,j);                            //输出结果
     printf("a=%c\nb=%c\n",a,b);                            //输出结果
     printf("f=%f\n",f);                                    //输出结果
}
```

该程序可在Keil μ Vision3集成开发环境中运行，执行结果如下。

```
i=12
j=7
a=A
b=B
f=8.750000
```

赋值语句比较简单，读者可以自行分析其运行结果。在C51语言中使用赋值表达式时，要注意数据类型的转换。数据类型转换是指不同类型的变量混用时，不同类型之间的转化。赋值表达式中类型转换的规则是等号右边的值转换为等号左边的值所属的类型。程序示例如下。

```
#include <stdio.h>                                          //头文件

void main()                                                 //主函数
{
     int i;                                                 //声明变量
     float f;
     i=5;
     f=5*2;                                                 //赋值
     printf("f=%f\n",f);                                    //输出结果
     f=i/2;                                                 //i/2的值赋值给f
     printf("f=%f\n",f);                                    //输出结果
     f=i/2.0;                                               // i/2.0的值赋值给f
     printf("f=%f\n",f);                                    //输出结果
     f=(float)i/2;                                          //使用强制类型转换
     printf("f=%f\n",f);                                    //输出结果
}
```

该程序可在Keil μ Vision3集成开发环境中运行，执行结果如下。

```
f=10.000000
f=2.000000
f=2.500000
f=2.500000
```

本例中，5*2的值为整型数据，而f为浮点型变量，因此，赋值时自动转换为浮点型数据。i/2即5/2按照除法的运算规则，除不尽时取整数部分，且等式左边的f为浮点型变量，因此f=2.000000。而对于i/2.0即5/2.0，操作数2.0为浮点型，因此计算的结果为2.500000，保留了完整的结果。如果操作数均为整型数据，则可使用强制类型转换(float)i/2，计算的结果为2.500000，同样可以保留结果的完整性。

3.9.3 逗号表达式

逗号表达式是用逗号运算符“,”以及括号将两个或多个表达式连接在一起的式子。其一般形式如下所示。

```
表达式1,表达式2,表达式3,...表达式n
```

逗号表达式的应用示例如下。

```
#include <stdio.h>                                          //头文件

void main()                                                 //主函数
{
     int a,b;                                               //声明变量
     b=(a=4*10,a*5,a/10);                                   //逗号表达式
     printf("a=%d\nb=%d\n",a,b);                            //输出结果
}
```

该程序可在Keil μ Vision3集成开发环境中运行，执行结果如下。

```
a=40
b=20
```

本例中，先求解a=4*10，则a的值为40，再求解a*5，结果为200，最后求解a/10=20作为整个表达式的值，即b的值为20。在使用逗号表达式时，需要注意以下几点。

- 一个逗号表达式还可以与另一个表达式组成新的逗号表达式，计算时同样需要自左向右计算。示例如下。

```
(a=4*10,a*5,a/10), (a+10)
```

- 程序中使用逗号表达式，有时并不一定为了求整个逗号表达式的值，而可能需要分别求出逗号表达式内各表达式的值。示例如下。

```
(a=3*4, a*5), (a+10);
```

本例中，整个逗号表达式没有赋值给任何变量，主要是计算各个中间表达式的值。

- 逗号表达式可以构成嵌套结构，即逗号表达式中的表达式也可以是逗号表达式，计算的顺序仍然是自左向右，整个表达式的值为最后一个表达式的结果。
- 在程序中，需要区分逗号表达式和逗号分隔符。例如，在变量声明、函数参数表中的逗号只是用作变量之间的分隔符。示例如下。

```
int a,b,c;
```

3.9.4 关系表达式

关系表达式是指两个表达式用关系运算符连接起来的式子。关系运算又称为“比较运算”。

示例如下。

```
x<(19+y)
x!=y
(x<5)>=7
```

关系表达式的计算结果是逻辑“真”（True）或者逻辑“假”（False）。当结果为真时，表达式的值为1；当结果为假时，表达式的值为0。关系表达式的程序示例如下。

```
#include <stdio.h>                                    //头文件

void main()                                           //主函数
{
      int a,b,c,d;                                    //声明变量
      a=27;                                           //赋值
      b=237;
      c=a>(b-200);                                    //计算关系表达式
      d=(a!=(b-100));
      printf("c=%d\nd=%d\n",c,d);                     //输出结果
}
```

该程序可在Keil μ Vision3集成开发环境中运行，执行结果如下所示。

```
c=0
d=1
```

本例中，首先为变量a和b赋值，然后通过关系表达式为变量c和d赋值。程序比较简单，读者可以自行分析其运算结果。

3.9.5 逻辑表达式

逻辑表达式是指两个表达式用逻辑运算符连接起来的式子。逻辑表达式中的操作对象可以是任何类型的数据，如整型、浮点型、字符型或指针型等。逻辑表达式的值是逻辑值，即“真”和“假”。当结果为真时，表达式的值为1；反之为0。逻辑表达式的程序示例如下。

```
#include <stdio.h>                                    //头文件

void main()                                           //主函数
{
      int a,b,c,d,e;                                  //声明变量
      float f1,f2;
      a=7;                                            //变量赋值
      b=9;
      c=!a;                                           //计算逻辑表达式
      d=a||b;
      f1=2.3;
      f2=5.7;
      e=(f1!=f2);
      printf("c=%d\nd=%d\ne=%d\n",c,d,e);              //输出结果
}
```

该程序可在Keil μ Vision3集成开发环境中运行，执行结果如下所示。

```
c=0
d=1
e=1
```

关系表达式和逻辑表达式通常是结合在一起使用的，常用于程序控制语句中控制流程运算。用于控制程序的流程时，要配合if、while和for等语句来使用。关系表达式和逻辑表达式用于程序流程控制的程序代码示例如下。

```
#include <stdio.h>                                          //头文件

void main()                                                 //主函数
{
     int a,b,c;                                             //声明变量
     a=5;                                                   //赋值
     b=1;
     if(a>b)                                                //判断a>b=1,为真
          c=(!a)||b;                                        //执行此语句
     else
          c=a||b;                                           //不执行该语句
     printf("c=%d\n",c);                                    //输出结果
}
```

该程序可在Keil μ Vision3集成开发环境中运行，执行结果如下所示。

```
c=1
```

该程序中，首先为变量a和b赋值，然后判断关系表达式a>b的值，如果其为真，则执行if下面的语句，否则执行else下面的语句。这里执行逻辑表达式(!a)||b并赋值给变量c。

3.10 小结

C51语言是单片机设计所广泛采用的程序语言。本章首先介绍了C51语言的基本结构和编程规范，然后分别介绍了C51语言的基本知识，包括标识符、关键字、变量与常量、运算符和表达式等。本章在介绍每个基本知识点时，都给出了完整详细的实例，方便读者进行学习。本章是单片机C51语言的基础，因此读者应该熟练掌握本章所讲内容。

3.11 上机实践

1. C51程序的注释有哪几种方式？

2. 若有以下定义，编写程序计算表达式y+=y-=m*=y的值。

```
int m=5,y=2;
```

3. 已知字母a的ASCII码为十进制数97，且设ch为字符型变量，编写程序，计算表达式ch=‘a’+‘8’-‘3’的值。

4. 若有定义：int x=3,y=2;float a=2.5,b=3.5，编写程序计算表达式 (x+y)%2+(int)a/(int)b的值。

5. 已知int a=0,b=0,c=0; 求c=(a-=a-5),(a=b,b+3);表达式中变量c的结果。

6. 以下程序是否能够通过编译？如果不能通过编译，请说明错误原因，并修改程序使其能正常编译，修改后程序的运行结果要求能输出a、b、c的值。

```
01:  #include <stdio.h>
02:
```

```
03:  void testResult();
04:
05:  void main()
06:  {
07:       int a;
08:
09:       a=12;
10:       testResult() ;
11:  }
12:
13:  void testResult()
14:  {
15:    int b,c;
16:    b = 2 * a;
17:    a = 200;
18:    c = a / 2;
19:
20:    printf("a=%d\n",a);
21:    printf("b=%d\n",b);
22:    printf("c=%d\n",c);
23:  }
```

提示　从变量作用域方面进行思考。

7. 已知“ int x=1,y=0,z=1; ”，编写程序求通过表达式“ !y||++x&&&&++z ”计算后x、y、z的值。

8. 写出以下程序的运行结果，然后将该程序输入到Keil μ Vision3中进行编译调试，验证运行的结果是否与之前分析的结果相同。

```
01:  #include <stdio.h>
02:
03:  void main()
04:  {
05:       int a,b,c;
06:       a=12;
07:       b=3;
08:       c=0;
09:       c = a || b || c;
10:       printf("c=%d\n",c);
11:  }
```