华清远见 FARSIGHT 始于2004 | 嵌入式培训专家

10年口碑积累，成功培养50000多名研发工程师，铸就专业品牌形象

华清远见的企业理念是不仅要做良心教育、做专业教育，更要做受人尊敬的职业教育。

《单片机 C 语言入门》（修订版）

作者：华清远见

专业始于专注 卓识源于远见

第 2 章 C51 语言基本语法

本章目标

C51 语言是针对 8051 系列及其扩展系列单片机的语言，支持符合 ANSI 标准的 C 语言程序设计，同时针对 8051 系列单片机的一些特点进行了扩展。本章介绍 C51 语言的一些基础知识，主要包括如下内容。

- C51 的标识符
- C51 的数据类型
- C51 的运算量
- C51 的运算符
- C51 的表达式

专业始于专注 卓识源于远见

2.1 C51 的标识符和关键字

C51 的标识符用来标识源程序中某个对象的名字，这些对象包括常量、变量、数据类型、语句标号以及用户自定义函数的名称等。合法的标识符由字母、数字和下划线组成，并且第一个字符必须是字母或下划线。标识符区分大小写，因此“a”和“A”代表不同的标识符。例如以下都是合法的标识符：

```
Summary，status，price3，_time，f_value，F_vAlue
```

而以下都是不合法的标识符：

```
5times，447#，day*month
```

关键字是编程语言保留的特殊标识符，具有固定名称和特定含义。在编写程序时，不允许标识符和关键字相同。表 2-1 列出了 ANSI C 的 32 个关键字。

表 2-1　　ANSI C 的 32 个关键字

关　键　字	用　　途	说　　明
auto	存储种类声明	用以声明局部变量，默认值为此
break	程序语句	退出最内层循环体
case	程序语句	switch 语句中的选择项
char	数据类型声明	单字节整型数或字符型数据
const	存储类型声明	在程序执行过程中不可修改的变量值
continue	程序语句	转向下一次循环
default	程序语句	switch 语句中的缺省选择项
do	程序语句	构成 do…while 循环结构
double	数据类型声明	双精度浮点数
else	程序语句	构成 if…else 选择结构
enum	数据类型声明	枚举
extern	数据类型声明	在其他程序模块中声明了的全局变量
float	数据类型声明	单精度浮点数
for	程序语句	构成 for 循环结构
goto	程序语句	构成 goto 转移结构
if	程序语句	构成 if…else 选择结构
int	数据类型声明	基本整型数
long	数据类型声明	长整型数
register	存储类型声明	使用 CPU 内部寄存器的变量
return	程序语句	函数返回
short	数据类型声明	短整型数
signed	数据类型声明	有符号数，二进制数据的最高位为符号位
sizeof	运算符	计算表达式或整型类型的字节数
static	存储类型声明	静态变量
struct	数据类型声明	结构类型数据
switch	程序语句	构成 switch 选择结构
typedef	数据类型声明	重新进行数据类型定义

华清远见

union	数据类型声明	联合类型数据
unsigned	数据类型声明	无符号数据
void	数据类型声明	无类型数据
volatile	数据类型声明	声明该变量在程序执行中可被隐含地改变
while	程序语句	构成 while 和 do…while 循环结构

C51 编译器除了支持 ANSI C 标准的关键字以外，还根据 51 单片机的特点扩展了特有的关键字，如表 2-2 所示。

表 2-2　　C51 扩展的关键字

关　键　字	用　　途	说　　明
_at_	地址定位	为变量进行存储器绝对空间地址定位
alien	函数特性声明	用以声明与 PL/M51 兼容的函数
bdata	存储器类型声明	可位寻址的 8051 内部数据存储器
bit	位变量声明	声明一个位变量或位类型的函数
code	存储器类型声明	8051 程序存储器空间
compact	存储器模式	指定使用 8051 外部分页寻址数据存储器空间
data	存储器类型声明	直接寻址的 8051 内部数据存储空间
idata	存储器类型声明	间接寻址的 8051 内部数据存储空间
intertupt	中断函数声明	定义一个中断服务函数
large	存储模式	指定使用 8051 外部数据存储空间
pdata	存储器类型声明	分页寻址的 8051 外部数据存储器
_priority	多任务优先声明	规定 RTX51 或 RTX51 Tiny 的任务优先级
reentrant	再入函数声明	定义一个再入函数
sbit	位变量声明	声明一个可位寻址变量
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
small	存储器模式	指定使用 8051 内部数据存储空间
task	任务声明	定义实时多任务函数
using	寄存器组定义	定义 8051 的工作寄存器组
xdata	存储器类型声明	8051 外部数据存储器

2.2　C51 的运算量

2.2.1　常量

在程序运行过程中其值保持不变的量称为常量。常量可以区分为不同的类型，如−9、0、5 为整型常量，2.3、−3.6 为实型变量，“1”、“*”、“s”为字符常量，“ABCD”为字符串常量。程序中一般用大写的标识符代表一个常量，例如下面的例子用 PRICE 代表一个常量 10。

【例 2-1】使用标识符代表一个常量。在本例中用 PRICE 代表一个常量 10。

```
#include <reg52.h>                    //特殊寄存器的头文件
#include <stdio.h>                    //I/O 库函数原型说明
#define PRICE 10                   //用标识符 PRICE 代表一个常量 10
void main()
{
    unsigned char quanlity;           //数量
    unsigned int cost;                //总价格

    SCON = 0x50;                      //SCON: 模式 1，8-bit UART，开始接收
    TMOD = (TMOD & 0x0F) | 0x20;  //TMOD: 定时器 1，模式 2，自动装载方式
    TH1 = 221;                        //TH1: 1200bit/s，16MHz
    IE |= 0x90 ;                   //IE: 允许中断
    TR1 = 1;                          //TR1: 定时器 1 运行
    TI=1;                             //TI: 设置 TI=1，以发送 UART 第一个字符

    quanlity=20;
    cost=PRICE*quanlity;
    printf("\nThe cost is %d.",cost);
//由于没有操作系统接受函数返回值，因此必须有一个循环保证程序不会中止
    while (1)

    {}
}
```

在程序开头用#define 命令行定义了一个符号常量 PRICE 代表常量 10，此后凡在此源文件中出现的 PRICE 都代表常量 10。程序运行结果为：

```
The cost is 200.
```

注意 符号常量的值在其作用域（本例中为 main 函数）内不能改变，也不能再次赋值。

2.2.2 变量

在程序运行过程中其值可以改变的量称为变量。一个变量主要由变量名和变量值两部分组成。每一变量都有一个变量名，在存储器中占据称为地址的一定的存储单元，在该存储单元中存放变量值。例 2-1 中的 cost 就是一个变量。

2.3 C51 的数据类型

数据是单片机程序处理的主要对象。所谓数据就是格式化了的数字，而数据类型就是数据的不同格式。C51 的数据类型如图 2.1 所示：

C51 支持如下几种基本数据类型：无符号字符型（unsigned char）、有符号字符型（signed char 或 char）、无符号短整型（unsigned short）、有符号短整型（signed short 或 short）、无符号整型（unsigned int）、有符号整型（signed int 或 int）、无符号长整型（unsigned long）、有符号整型（signed long 或 long）、浮点型（float）。此外，还包括专门用于 8051 硬件和 C51 编译器的 bit、sbit、sfr、sfr16 等数据类型。各种数据类型的长度和值域如表 2-3 所示。

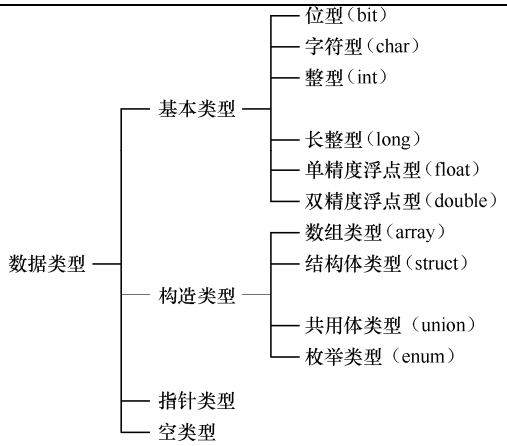

图 2.1　C51 的数据类型

表 2-3　　各种数据类型的长度和值域

数 据 类 型	长度（bit）	长度（Byte）	值　　域
bit	1	…	0、1
unsigned char	8	1	0～255
signed char	8	1	−128～127
unsigned int	16	2	0～65 535
signed int	16	2	−32 768～32 767
unsigned long	32	4	0～4 294 967 295
signed long	32	4	−2 147 483 648～2 147 483 647
float	32	4	±1.76E−38～±3.10E+38（6 位有效数字）
double	64	8	±1.76E−38～±3.10E+38（10 位有效数字）
一般指针	24	3	存储空间，0～65 535

2.3.1　字符型（char）

1．字符型常量

字符常量是用单引号括起来的一个字符，如“a”，“$”，“1”，“A”等都是字符常量。注意，字符型常量区分大小写，因此“a”和“A”是不同的字符常量。不可以显示的控制字符，可以在该字符前面加一个“\”组成转义字符，也就是把“\”后面的字符转变成另外的意义。常用的转义字符如表 2-4 所示。

表 2-4　　转义字符

规　定　符	等　价　于	含　　义
\f	\X0C	换页
\r	\X0D	回车
\t	\X09	制表键
\n	\X0A	换行
\\	\X5C	\符
\’	\X27	‘符
\”	\X22	“符

例 2-2 是一个在程序中使用字符型常量的例子。

华清远见

【例 2-2】使用字符型常量和转义字符。

```
#include <reg52.h>                          //特殊寄存器的头文件
#include <stdio.h>                          //I/O 库函数原型说明
#define  CHR_CONSTANT 'a'                   //定义一个字符型常量 CHR_CONSTANT

void main()
{
    SCON = 0x50;                            //SCON: 模式 1，8-bit UART，开始接收
    TMOD = (TMOD & 0x0F) | 0x20;        //TMOD: 定时器 1，模式 2，自动装载方式
    TH1 = 221;                              //TH1: 1200bit/s，16MHz
    IE |= 0x90 ;                        //IE: 允许中断
    TR1 = 1;                                //TR1: 定时器 1 运行
    TI=1;                                   //TI: 设置 TI=1，以发送 UART 第一个字符

    printf("\n");                       //输出转义字符\n
    printf("CHR_CONSTANT is '%c'.",CHR_CONSTANT);//输出字符常量
//由于没有操作系统接收函数返回值，因此必须有一个循环保证程序不会中止
    while (1)
    {}
}
```

程序输出结果为：

```
CHR_CONSTANT is ‘a’.
```

2．字符型变量

字符型变量的长度为一个字节（即 8 位），而 8051 单片机每次可以处理 8 位数据，因此字符型变量非常适合于 8051 单片机。

字符型变量分为无符号和有符号两种。如果没有显式地指明是无符号还是有符号，则默认为有符号字符型变量。例如：

```
char chr;               //定义 chr 为有符号字符型变量
signed char chr;    //定义 chr 为有符号字符型变量
unsigned char chr;      //定义 chr 为无符号字符型变量
```

对字符型变量赋值有两种方法：既可以将用单引号括起来的一个字符赋给字符型变量，也可以将一个在其取值范围内的正整数赋给字符型变量，例如：

```
unsigned char chr1,chr2;//定义两个无符号字符型变量 chr1、chr2
chr1='#';               //把用单引号括起来的一个字符赋给字符型变量
chr2=0x39;              //把一个在其取值范围内的正整数赋给字符型变量
```

对于有符号的变量，最具有重要意义的是其最高位（8 位中最左一位）。在此位上，1 代表“负”，0 代表“正”，剩余的低 7 位代表变量的绝对值，因此有符号字符型变量所能表示的数值范围是从−128～+127。而无符号的变量的最高位不作为符号位，这与人们的习惯比较一致，不易出错。

当代表 0～127 的数值时，有符号字符型变量和无符号字符型变量的表示方法一样，都是 0～0x7f，但对于大于等于 0x80 的数值，二者的表示方法不同。这是因为当 ASCII 码值大于等于 0x80 时，该字节的最高位为 1，对于有符号字符型变量，单片机会认为这是一个负数，用补码表示（正数的补码与原码相同，而负数的补码等于其绝对值按位取反后加 1）。例如，对于 ASCII 值为 0x91 的字符，定义为 signed char 时，最高位的 1 被认为是负数的标志，其余的 7 位按位取反后再加 1，结果是 0x6f，因此实际表示的是十六进制的−0x6f。下面的例子表明了有符号型和无符号型的区别。

【例 2-3】数值在小于 0x80 和大于等于 0x80 的情况下 signed char 与 unsigned char 的表现。程序中用到了

强制类型转换如(int)(a1)，在后面的章节中还会详细解释，现在读者只需了解这是为了便于以整型变量的格式输出字符型变量。

```
#include <reg52.h>                    //特殊寄存器的头文件
#include <stdio.h>                    //I/O 库函数原型说明

void main()
{
    unsigned char a1,a2;          //定义两个无符号字符型变量
    signed char b1,b2;                //定义两个有符号字符型变量

    SCON = 0x50;                      //SCON: 模式 1，8-bit UART，开始接收
    TMOD = (TMOD & 0x0F) | 0x20;  //TMOD: 定时器 1，模式 2，自动装载方式
    TH1 = 221;                        //TH1: 1200bit/s，16MHz
    IE |= 0x90 ;                  //IE: 允许中断
    TR1 = 1;                          //TR1: 定时器 1 运行
    TI=1;                             //TI: 设置 TI=1，以发送 UART 第一个字符

    a1=0x7f;                          //将无符号字符型变量赋值为 0x7f
    b1=0x7f;                          //将有符号字符型变量赋值为 0x7f
    printf("a1=%d ",(int)(a1));       //无符号字符型变量的表示为 0x7f（即 127）
    printf("b1=%d ",(int)(b1));       //有符号字符型变量的表示为 0x7f（即 127）
    printf("\n");

    a2=0x91;                          //将无符号字符型变量赋值为 0x91
    b2=0x91;                          //将有符号字符型变量赋值为 0x91

    printf("a2=%d ",(int)(a2));       //无符号字符型变量的表示为 0x91（即 145）
    printf("b2=%d ",(int)(b2));       //有符号字符型变量的表示为-0x6f（即-111）
//由于没有操作系统接收函数返回值，因此必须有一个循环保证程序不会中止
    while (1)
    {}
}
```

程序运行结果为：

```
a1=127 b1=127
a2=145 b2=-111
```

3．字符串常量

字符串常量是用一对双引号括起来的字符序列，例如下面都是字符串常量：

```
"China", "!*&", "128u", "a"
```

可以用如下的方法输出一个字符串：

```
printf("China");
```

不要把字符常量与字符串常量混淆，'A'是字符常量而"A"是字符串常量。不能把一个字符串赋给一个字符变量，例如下面的代码是正确的：

```
char chr;
chr='A';
```

而下面的代码则是错误的：

```
char chr;
chr="A";
```

这是因为在 C51 中，系统在每一个字符串的结尾加一个被称为“字符串结束标志”字符'\0'以判断字符串是否结束。由于如果有一个字符串"A"，则其在内存中的实际存储方式为：

A	\0

该字符串包含了'A'和'\0'两个字节，因此，无法存储到只有一个字节的存储空间的字符型变量中。同样地，"China"占用了 6 个字节而不是 5 个字节。

注意 '\0'是系统自动加上的，在写字符串时不必加'\0'。

2.3.2 整型（char）

整型变量可以分为整型常量和整型变量。

1．整型常量

整型常量按进制有以下 3 种表示形式。

- 十进制整数：例如 145、3、−9、0。
- 八进制整数：以 0 开头的数，例如 034 表示八进制数 $(34)_8$，等于十进制数 28；−057 表示八进制数 $(-57)_8$，等于十进制数−47。
- 十六进制数：以 0x 或 0X 开头的数是十六进制数，例如 0x3a 代表十六进制数 $(3a)_{16}$，等于十进制数 58；−0x56 代表十六进制数 $(56)_{16}$，等于十进制数−88。

2．整型变量

整型变量可以分为基本型、短整型、长整型和无符号型 4 种。

- 基本型：以 int 表示，长度为两个字节。
- 短整型：以 short int 表示，长度为两个字节。
- 长整型：以 long int 表示，长度为四个字节。
- 无符号型：以 unsigned 表示，这种类型与前 3 种类型匹配而构成无符号整型（unsigned int）、无符号短整型（unsigned short）和无符号长整型（unsigned long）。各种无符号整型变量的长度和相应的有符号整型变量相同，但由于全部的位都用来存放数本身而不包括符号位，因此只能存放不带符号的正数。

注意 int、short int、long int 的缺省类型为有符号型。

说明整型变量的一般形式为：

```
类型说明符 变量名 [变量名…]
```

例如：

```
int a,b;                          //定义 a、b 为基本整型变量
long a;                               //定义 a 为有符号长整型变量
unsigned long c;                  //定义 c 为无符号长整型变量
```

同类型的整型变量之间和不同类型的整型变量之间都可以进行算术运算。

【例 2-4】整型变量之间进行算术运算。

```
#include <reg52.h>                        //特殊寄存器的头文件
```

```
#include <stdio.h>                    //I/O 库函数原型说明

void main()
{
    unsigned int a,b,c;               //定义 a、b、c 为 unsigned int 类型
    long int d,e;                  //定义 d、e 为 long int 类型

    SCON = 0x50;                      //SCON: 模式 1，8-bit UART，开始接收
    TMOD = (TMOD & 0x0F) | 0x20;  //TMOD: 定时器 1，模式 2，自动装载方式
    TH1 = 221;                        //TH1: 1200bit/s，16MHz
    IE |= 0x90 ;                   //IE: 允许中断
    TR1 = 1;                          //TR1: 定时器 1 运行
    TI=1;                             //TI: 设置 TI=1，以发送 UART 第一个字符

    a=23;                             //23→a
    b=34;                             //34→b
    d=2556;                           //2556→d
    c=a+b;                            //两个同类型的整型变量相加
    e=a+d;                            //两个不同类型的整型变量相加

    printf("\n");                  //输出转义字符\n
    printf("%d\n",c);
    printf("%ld\n",e);
//由于没有操作系统接收函数返回值，因此必须有一个循环保证程序不会中止
    while (1)
    {}
}
```

程序运行结果如下：

```
a+b=57
a+c=2579
```

同样，整型变量和字符型变量之间也可以进行算术运算，如例 2-5 所示。

【例 2-5】整型变量和字符型变量之间进行算术运算。

```
#include <reg52.h>                    //特殊寄存器的头文件
#include <stdio.h>                    //I/O 库函数原型说明

void main()
{
    unsigned char a;                  //定义 a 为 unsigned char 类型
    long int b,c;                  //定义 b、c 为 long int 类型

    SCON = 0x50;                      //SCON: 模式 1，8-bit UART，开始接收
    TMOD = (TMOD & 0x0F) | 0x20;  //TMOD: 定时器 1，模式 2，自动装载方式
    TH1 = 221;                        //TH1: 1200bit/s，16MHz
    IE |= 0x90 ;                   //IE: 允许中断
    TR1 = 1;                          //TR1: 定时器 1 运行
    TI=1;                             //TI: 设置 TI=1，以发送 UART 第一个字符

    a='A';                            //'A'→a
    b=2556;                           //2556→b
    c=a+b;                            //整型变量和字符型相加

    printf("\n");                  //输出转义字符\n
    printf("a+b=%ld",c);
```

```
    //由于没有操作系统接收函数返回值，因此必须有一个循环保证程序不会中止
        while (1)
        {}
    }
```

本例中，将'A'赋给字符型变量 a。由于字符'A'的 ASCII 码值为 65，因此 a 中存储的数据就是 65，可以与一个整型变量进行算术运算。

2.3.3 浮点型（float）

由于浮点型数据可以直接表示小数，因此许多复杂的数学表达式都采用浮点型数据。浮点型数据也分为浮点型常量和浮点型变量。

1．浮点型常量

例如+96.3、65.36、−2.3、.654、−3.3E9、0.3e−7 等都是浮点型常量。浮点常量只有十进制这一种进制，并且都被默认为下面将要介绍的 double 型。对于绝对值小于 1 的浮点数可以省略小数点前面的零，如.654 就是 0.654 的缩略形式。形如−3.3E9 的浮点数则是由尾数和阶码两部分构成的，−3.3E9 就等于−3.3 乘以 10 的 9 次方。

2．浮点型变量

浮点型变量分为单精度型（float）和双精度型（double），长度都是 4 个字节。可以用下列语句说明浮点型变量：

```
    float a;     //定义 a 为单精度浮点数
    double b;        //定义 b 为双精度浮点数
```

2.3.4 指针型

指针型是一种特殊的数据类型，其本身就是一个变量，但在其中存放的是另一个数据的地址。在 C51 中，指针的长度一般是 3 个字节。根据所指向的变量类型的不同指针变量也有不同的类型，而指针变量的类型也就表示了该指针指向的地址中的数据的类型。指针类型的表示方法是在指针符号“*”前面冠以数据类型符号，例如：

```
    char *a;          //定义 a 为字符型指针
    unsigned int *b; //定义 b 为无符号整型指针
    float *c;             //定义 c 为浮点型指针
    struct *d;            //定义 d 为结构型指针
    union *e          //定义 e 为联合型指针
```

2.3.5 位变量（bit）

位变量的长度是 1 位（bit），位变量和前面介绍的字符型变量是可以直接被 8051 单片机所处理的。位变量的值可以取 0（false）或 1（true）。与 8051 单片机硬件特性操作有关的位变量必须定位在 8051 单片机片内 RAM 的可位寻址空间中，也就是字节地址为 20H～2FH 的 16 字节单元。对位变量进行定义的语法如下：

```
bit flag1;          //定义 flag1 为位变量
bit send_en=1;      //定义 send_en 为位变量并初始化为 1
bit error_flg;      //定义 error_flg 为位变量
```

注意 不能定义一个位变量指针，如不能定义 bit *flag1；也不能定义一个位变量数组，如不能定义 bit flags[3]。

2.3.6 特殊功能寄存器（sfr）

8051 系列单片机具有多种内部寄存器，其中有 21 个特殊功能寄存器（SFR），分散在片内 RAM 区的 80H～0FFH 字节空间中，例如串口控制寄存器 SCON、中断允许寄存器 IE 等。对这些 SFR 的操作只能用直接寻址方式，为此 C51 编译器扩充了关键字 sfr 和 sfr16，利用这两种数据类型可以在源程序中直接对 8051 单片机的特殊功能寄存器进行定义。sfr 类型的长度为一个字节，其定义方法如下：

```
sfr 特殊功能寄存器名=地址常量;
```

例如：

```
sfr P1  =0x90;   //定义一个名为 P1 的 SFR，其地址为 0x90
sfr SCON=0x98;   //定义一个名为 SCON 的 SFR，其地址为 0x98
```

8051 单片机中，地址为 0x90 的 SFR 是 P1 端口的寄存器，因此，P1 就表示 P1 端口，在随后的程序中对 P1 进行处理就是对 P1 端口进行处理。同理，地址为 0x98 的 SFR 是串口控制寄存器的地址，在随后的程序中对 SCON 进行处理就是对串口控制寄存器进行处理。

这里需要注意的是，在关键字 sfr 后面必须是一个名字，名字可以任意选取，但应符合一般的习惯。等号后面必须是常数，不允许有带运算符的表达式，而且该常数必须在特殊功能寄存器的地址范围之内（80H～0FFH）。

2.3.7 16 位特殊功能寄存器（sfr16）

在新一代的 8051 单片机中，特殊功能寄存器在功能上经常组合成 16 位来使用。为了有效地访问这种 16 位的特殊功能寄存器，可采用关键字 sfr16。sfr16 类型的长度为两个字节，其定义语法与 8 位 SFR 相同，但 16 位 SFR 的低端地址必须作为 sfr16 的定义地址。例如对 8052 单片机的定时器 T2，可采用如下的方法来定义：

```
sfr16 T2=0xCC;   //定义 TIME2，其地址为 T2L=0CCH，T2H=0CDH
```

这里 T2 为特殊功能寄存器，等号后面是其低字节地址，其高字节地址必须在物理上直接位于低字节之后。

注意 这种定义方法适用于所有新一代的 8051 单片机中新增加的特殊功能寄存器，但不能用于定时器/计算器 TIMER0 和 TIMER1 的定义。

2.3.8 可寻址位（sbit）

在 8051 单片机的实际应用中经常需要访问特殊功能寄存器中的某些位，C51 编译器为此提供了一种扩充关键字 sbit，利用 sbit 可以访问可位寻址对象，使用方法有如下 3 种。

1．sbit 位变量名=位地址

这种方法将位的绝对地址赋给位变量，位地址必须位于 80H～0FFH 之间。例如：

```
sbit P0_0=0x80;      //定义 P0_0 位为绝对地址 0x80
sbit IT0=0x88;       //定义 IT0 位为绝对地址 0x88
```

2．sbit 位变量名=特殊功能寄存器名^位位置

当可寻址位位于特殊功能寄存器种时可采用这种方法，“位位置”是一个 0～7 之间的常数。例如：

```
sfr P0=0x80;     //定义一个特殊寄存器 P0
sbit P0_1=P0^1;      //定义 P0_0 为 P0 的第 1 位
sbit P0_5=P0^5;  //定义 P0_5 为 P0 的第 5 位
```

3．sbit 位变量名=字节地址^位位置

这种方法以一个常数（字节地址）作为基础，该常数必须在 80H～0FFH 之间。“位位置”是一个 0～7 之间的常数。例如：

```
sbit P0_1=0x80^1;    //定义 P0_0 为字节地址为 0x80 的 SFR 的第 1 位
sbit P0_5=0x80^5;    //定义 P0_5 为字节地址为 0x80 的 SFR 的第 5 位
```

注意 sbit 是一个独立的关键字，不要与关键字 bit 相混淆。

2.4 C51 的存储种类和存储器类型

在前面的章节中，定义变量都是采用如下格式：

```
数据类型  变量名表
```

但实际上，完整的变量定义格式如下：

```
[存储种类] 数据类型 [存储器类型] 变量名表
```

在定义格式中除了数据类型和变量名表是必要的，存储种类和存储器类型都是可选项。本节将对存储种类和存储器类型分别进行介绍。

2.4.1 存储种类

存储种类有 4 种：自动（auto）、外部（extern）、静态（static）和寄存器（register），默认类型为自动（auto）。

1．自动变量

在函数体内部或者复合语句中定义的变量，如果省略存储种类说明或者在变量名前面加上存储种类说明符“auto”，即可将该变量定义为自动变量。通常采用缺省形式，即省略存储种类说明。例如：

```
char chr='a';
```

等价于：

```
auto char chr='a';
```

自动变量的作用域在定义其的函数体或复合语句内部。在进入函数体或复合语句时，编译程序为自动型变

量分配堆栈空间，退出函数或复合语句时，堆栈空间就立即消失，从而这些自动型变量也就不复存在，不能被其他函数引用。因此，自动变量始终是相对于函数或复合语句的局部变量。下面的例 2-6 演示了自动变量的作用域。

【例 2-6】自动变量的作用域。

在 main 函数和复合语句中分别定义了一个同名的字符型自动变量并输出，运行结果证明复合语句中自动变量的作用域仅限于复合语句中。

```
#include <reg52.h>
#include <stdio.h>

void main (void)
{
    char chr='a';                  //在 main 函数内定义一个自动型变量 chr

    SCON = 0x50;                       //SCON: 模式 1，8-bit UART，开始接收
    TMOD = (TMOD & 0x0F) | 0x20;  //TMOD: 定时器 1，模式 2，自动装载方式
    TH1 = 221;                         //TH1: 1200bit/s，16MHz
    IE |= 0x90 ;                   //IE: 允许中断
    TR1 = 1;                           //TR1: 定时器 1 运行
    TI=1;                              //TI: 设置 TI=1，以发送 UART 第一个字符

    {
        char chr='b';              //在复合语句内定义一个自动型变量 chr
        printf("%c\n",chr);            //输出复合语句中的 chr
    }
    printf("%c",chr);              //输出 main 函数中的 chr
//由于没有操作系统接收函数返回值，因此必须有一个循环保证程序不会中止
    while(1)
    {}
}
```

程序运行结果为：

```
b
a
```

2．外部变量

在所有函数外部定义的变量或者使用存储种类说明符“extern”定义的变量称为外部变量。一个外部变量被定义之后，就被分配了固定的内存空间，并且可以被一个程序中的所有函数使用，因此外部变量实质上属于全局变量，其作用域是整个程序，在程序的任何地方均可以通过不同的名字对这种变量进行访问，但如果有同名变量，则只有内部变量起作用。

C51 语言允许将大型程序分解为若干个独立的程序模块文件，各个模块可以分别进行编译然后再连接在一起。在这种情况下，如果某个变量需要在其他程序模块文件中使用，只要在一个程序模块文件中将该变量定义为全局变量，而在其他程序模块文件中使用“extern”说明该变量是已被定义过的外部变量就可以了。在整个程序（可能有多个文件）中都具有相同名字的外部变量只能在一处进行定义和初始化。

下面的例 2-7 说明了外部变量在同一个文件中的使用方法。

【例 2-7】外部变量在同一个文件中的定义和使用。字符型变量 chr_1 在函数之外定义，根据缺省规则，chr_1 是已经是一个全局变量，不需要使用 extern 进行声明；而 chr_2 在定义前被使用，因此在使用前必须用 extern 进行声明。

```
#include <reg52.h>
```

```
#include <stdio.h>

char chr_1='a';
void main (void)
{
    extern char chr_2;                  //在 main 函数内定义一个自动型变量 chr_2

    SCON = 0x50;                        //SCON: 模式 1，8-bit UART，开始接收
    TMOD = (TMOD & 0x0F) | 0x20;  //TMOD: 定时器 1，模式 2，自动装载方式
    TH1 = 221;                          //TH1: 1200bit/s，16MHz
    IE |= 0x90 ;                    //IE: 允许中断
    TR1 = 1;                            //TR1: 定时器 1 运行
    TI=1;                               //TI: 设置 TI=1，以发送 UART 第一个字符

    printf("\n%c",chr_1);          //输出外部变量 chr_1
    printf("\n%c",chr_2);          //输出 main 函数中的 chr_2
//由于没有操作系统接收函数返回值，因此必须有一个循环保证程序不会中止
    while(1)
    {}
}
char chr_2='b';
```

程序运行结果为：

```
a
b
```

3．静态变量

静态变量的定义方法是在类型定义语句之前加关键字 static。静态变量分为内部静态变量（又称局部静态变量）和外部静态变量（又称全局静态变量）。

内部静态变量是在函数内部定义的，与自动变量相比，其作用域同样限于定义内部静态变量的函数内部，但内部静态变量始终都是存在的，其初值只是在进入时赋值一次，退出函数之后变量的值仍然保存但不能访问。

外部静态变量是在函数外部被定义的，与外部变量相比，其作用域同样是从定义点开始，一直到程序结束，但外部静态变量只能在被定义的模块文件中访问，其数据值可以为该文件内所有的函数所共享，退出该文件后，虽然变量的值仍然保存着，但不能被其他模块文件访问。

例 2-8 显示了内部静态变量的用法以及与自动变量的区别。

【例 2-8】内部静态变量的用法以及与自动变量的区别。

在一个复合语句中分别定义一个内部静态变量和自动变量，先后进入此复合语句 3 次，结果显示由于退出复合语句时内部静态变量仍然存在并保存其值，自动变量则不复存在，因此内部静态变量实现了累加，而自动变量不能。

```
#include <reg52.h>
#include <stdio.h>

void main (void)
{
    char i;

    SCON = 0x50;                        //SCON: 模式 1，8-bit UART，开始接收
    TMOD = (TMOD & 0x0F) | 0x20;  //TMOD: 定时器 1，模式 2，自动装载方式
```

```
    TH1 = 221;                          //TH1: 1200bit/s，16MHz
    IE |= 0x90 ;                      //IE: 允许中断
    TR1 = 1;                            //TR1: 定时器 1 运行
    TI=1;                               //TI: 设置 TI=1，以发送 UART 第一个字符

    for(i=1;i<=3;i++)
    {
        static int s_int=1;             //定义内部静态变量
        int a_int=1;                    //定义自动变量
        printf("\n");
        printf("s_int=%d ",s_int);//输出内部静态变量
        printf("a_int=%d",a_int); //输出自动变量
        s_int=s_int+1;                  //内部静态变量加 1
        a_int=a_int+1;                  //自动变量加 1
    }
//由于没有操作系统接收函数返回值，因此必须有一个循环保证程序不会中止
    while(1)
    {}
}
```

程序运行结果为：

```
s_int=1 a_int=1
s_int=2 a_int=1
s_int=3 a_int=1
```

4．寄存器变量

定义一个变量时在变量名前加上存储种类符号“register”即可将该变量定义为寄存器变量，例如：

```
register int i;
register unsigned char chr;
```

使用寄存器变量的目的在于将一些使用频率最高的变量定义为能够直接使用硬件寄存器。寄存器变量可以认为是自动变量的一种，其作用域与自动变量相同。将变量定义为寄存器变量只是给编译器一个建议，该变量能否真正成为寄存器变量还有由编译器根据实际情况决定；另一方面，编译器可以自行识别使用频率最高的变量，在可能的情况下，即使程序中并未将变量定义为寄存器变量，编译器也会自动将其作为寄存器变量处理。

注意 目前已不推荐寄存器变量这种方式。

2.4.2 存储器类型

8051 系列单片机在物理上有 4 个存储空间：片内程序存储空间、片外程序存储空间、片内数据存储空间、片外数据存储空间，每个存储空间包括从 0 到最大存储范围的连续的字节地址空间，程序中定义的任何数据类型必须以一定的存储类型的方式定位在 8051 的某一存储区内。

编译器通过将数据（包括常量和变量）定义成 DATA、BDATA、IDATA、PDATA、XDATA、CODE 等不同的存储类型可以将每个变量明确地定位到不同的存储区中。对内部数据存储器的访问比对外部数据存储器的访问快许多，因此应将频繁使用的变量放在内部数据存储器中，将较少使用的变量放在外部数据存储器中。

存储器类型与 8051 实际存储空间的对应关系如表 2-5 所示。

表 2-5　　存储器类型与实际存储空间的对应关系

存储器类型	与物理存储空间的对应关系
DATA	直接寻址片内数据存储器的低 128 字节，访问速度快
BDATA	DATA 区中可位寻址区域（16 字节），允许位与字节混合访问
IDATA	间接寻址片内数据存储区（256 字节），可访问片内全部 RAM 空间
PDATA	外部数据存储区的开头 256 字节，通过 P0 端口的地址对其访问
XDATA	片外数据存储区（64KB），通过 DPTR 访问
CODE	代码存储区（64KB），通过 DPTR 访问

1．DATA 区

DATA 区声明中的存储类型标识符为 data，通常指低 128 字节的内部存储区存储的变量，可直接寻址。DATA 区是存放临时性传递变量或使用频率较高的变量的理想场所。声明举例如下：

```
unsigned int data sum;
extern char data chr;
static unsigned char data c;
```

2．BDATA 区

BDATA 区声明中的存储类型标识符为 bdata，指内部可位寻址的 16 字节存储区。BDATA 区其实就是 DATA 区中的位寻址区，在这个区声明变量就可以进行位寻址。在 BDATA 区声明和使用位变量的例子如下：

```
unsigned int bdata status;    //在 BDATA 区定义一个变量 status
sbit status_1=status^1;       //将 status_1 定义为 status 的第一位
if(status_1==1)               //使用 status_1
   {...}
```

注意　不允许在 BDATA 区声明 float 和 double 型的变量。

3．IDATA 区

8051 系列的一些单片机如 8052 有附加的 128 字节的内部 RAM，位于从 80H 开始的 128 字节地址空间中，被称为 IDATA。因为 IDATA 区的地址和 SFR 的地址重叠，所以通过寻址方式来区分二者，IDATA 区只能通过间接寻址来访问。IDATA 区也可存放使用比较频繁的变量，使用寄存器作为指针进行寻址。IDATA 区声明中的存储类型标识符为 idata，声明举例如下：

```
unsigned char idata sum;
int idata i;
float idata f_value;
```

4．PDATA 区和 XDATA 区

PDATA 区和 XDATA 区属于片外数据存储空间。外部存储空间是可以读写的存储区，最多可以有 64KB。PDATA 区和 XDATA 区声明中的存储类型标识符分别为 pdata 和 xdata，xdata 存储类型标识符可以指定片外数据区 64KB 空间内的任何地址，而 pdata 存储类型标识符仅能指定 256 字节的片外数据区。声明举例如下：

```
unsigned char xdata sum;
int pdata i;
float pdata f_value;
```

5．CODE 区

CODE 区也称代码段，是只读的，用来存放可执行代码，16 位寻址空间可达 64KB。除了可执行代码，还可在 CODE 区中存放其他非易失信息，例如查询表。CODE 区中对象要在编译的时候进行初始化，否则就会产生错误。CODE 区声明中的存储类型标识符分别为 code。下面的代码把一个数组存放在 CODE 区中。

```
unsigned char code chr[5]={1,2,3,4,5}
```

2.4.3 存储模式

存储模式决定了默认的存储器类型，此存储器类型将应用于函数参数、局部变量和定义时没有显式地包含存储类型的变量。在命令行中使用 SMALL、COMPACT、LARGE 控制命令指定存储器类型。定义变量时，使用存储器类型显式定义将屏蔽由存储模式决定的默认存储器类型。

1．小（SMALL）模式

在该模式下所有变量都默认位于片内数据存储器，这和使用 data 指定存储器类型的作用一样。此模式对变量访问的效率很高，但所有的数据对象和堆栈的总大小不能超过内部 RAM 的大小。当连接器/定位器将变量都配置在片内数据存储器时，SMALL 模式是最佳选择。

2．紧凑（COMPACT）模式

在该模式下所有变量都默认位于片外数据存储器的一页（256 字节）内，但堆栈位于片内数据存储区中，这和使用 pdata 指定存储器类型的作用一样，该存储模式适用于变量不超过 256 字节的情况。地址的高字节往往通过端口 2 输出，其值必须在启动代码中设置。这种模式不如 SMALL 模式高效，对变量访问的速度要慢一些。

3．大（LARGE）模式

在该模式下所有变量都默认位于片外数据存储器内，这和使用 xdata 指定存储器类型的作用一样。使用数据指针 DPTR 进行寻址，通过 DPTR 访问片外数据存储器的效率较低，特别是当变量为两个字节或更多字节时，该模式的数据访问要比前两种模式产生更多代码。

2.5 C51 运算符和表达式

C 语言把除了控制语句和输入输出以外的几乎所有基本操作都作为运算符处理。按其在表达式中所起的作用，可分为算术运算符、赋值运算符、增量与减量运算符、关系运算符、逻辑运算符、位运算符、条件运算符、逗号运算符、指针和地址运算符、强制类型转换运算符、求字节运算符、分量运算符和下标运算符等。

当运算符的运算对象只有一个时，称为单目运算符；当运算对象为两个时，称为双目运算符；当运算对象为三个时，称为三目运算符。

表达式是由运算符和运算对象所组成的具有特定含义的式子。

2.5.1 算术运算符和算术表达式

C51 中最基本的算术运算符有如下 5 个。

+：加法运算符或取正值运算符。

−：减法运算符或取负值运算符。

*：乘法运算符。

/：除法运算符。

%：模（取余）运算符。

上面的运算符中，加法、减法和乘法运算符合一般的算术运算规则，需要注意的是除法和取余运算符。对于除法运算，如果是两个整数相除，其结果仍为整数，舍去小数部分；如果是两个浮点数相除，其结果仍是浮点数。取余运算要求两个运算对象均为整型数据。另外，由于字符型数据会自动转换成整型数据，因此字符型数据也可以参加算术运算。

用算术运算符将运算对象连接起来组成的式子就是算术表达式，其一般形式为：

```
表达式 1 算术运算符 表达式 2
```

例 2-9 演示了算术运算符和算术表达式的用法。

【例 2-9】算术运算符和算术表达式以及字符型变量和整型变量之间的运算。

```
#include <reg52.h>
#include <stdio.h>

void main (void)
{
    unsigned char chr='b';                    //定义一个 char 型变量
    unsigned int u_value1=20,u_value2=56; //定义两个 unsigned int 型变量
    float f_value1=2.3,f_value2=4.7;      //定义两个 float 型变量

    SCON = 0x50;                              //SCON: 模式 1，8-bit UART，开始接收
    TMOD = (TMOD & 0x0F) | 0x20;          //TMOD: 定时器 1，模式 2，自动装载方式
    TH1 = 221;                                //TH1: 1200bit/s，16MHz
    IE |= 0x90 ;                              //IE: 允许中断
    TR1 = 1;                                  //TR1: 定时器 1 运行
    TI=1;                                     //TI: 设置 TI=1，以发送 UART 第一个字符

   printf("\n u_value2/u_value1=%d",u_value2/u_value1);//两个整型变量进行除法运算
   printf("\n u_value2%u_value1=%d",u_value2%u_value1);//两个整型变量进行取余运算
   printf("\n f_value2/f_value1=%f",f_value2/f_value1);//两个浮点变量进行除法运算
    printf("\n chr%u_value1=%d",chr%u_value1);//字符型变量和整型变量进行取余运算
//由于没有操作系统接收受函数返回值，因此必须有一个循环保证程序不会中止
    while(1)
    {}
```

```
}
```

程序运行结果为：

```
u_value2/u_value1=2
u_value2%u_value1=16
f_value2/f_value1=2.043478
chr%u_value1=18
```

2.5.2 赋值运算符和赋值表达式

前面章节中多次用到的赋值符号“=”，即赋值运算符，其功能是将数据赋给变量。用赋值运算符将一个变量与一个表达式连接起来的式子为赋值表达式。其一般形式为：

```
变量=表达式
```

上式中的“表达式”既可以是一个常量、变量、算术表达式，也可以是一个赋值表达式。也就是说，允许进行多重赋值，例如：

```
a=b=8;  //将常量 8 分别赋给变量 a 和 b
```

如果赋值号两侧的类型不一致，系统会自动将右侧表达式求得的数据按赋值号左边的变量类型进行转换，但这种转换仅限于数值型数据之间（如地址值就不能赋给一般变量）。转换规则如下。

- 将实型数赋值给整型数时，舍弃实数的小数部分。例如：

```
int i;
i=3.8;
```

则 i 的值为 3。

- 将整型数赋值给实型数时，数值不变，以浮点数的形式存储到变量中。例如：

```
float f;
f=8;
```

则 f 的值是 8.0，而不是 8（整型）。

- 字符型数据赋给整型变量时：当字符型数据为无符号数据时，将字符数据放到整型数据的低 8 位，高 8 位均补 0；当字符型数据为有符号数据时，将字符数据放到整型数据的低 8 位，高 8 位均补字符数据的符号位。例如：

```
unsigned char u_chr=0x68;    //u_chr=0b01101000
signed char s_chr=0x89;      //s_chr=0b10001001
unsigned int i;
i=u_chr;                 //i=0b0000000001101000
i=s_chr;                 //i=0b1111111110001001
```

- 将整型数据赋给长整型数据时，当整型数据为无符号数据时，将整型数据放到长整型数据的低 16 位，高 16 位均补 0；当整型数据为有符号数据时，将整型数据放到长整型数据的低 16 位，高 16 位均补整型数据的符号位。
- 将有符号数赋给长度相同的无符号数据时，连符号一起作为数值传递。要注意：当有符号数为正数时，赋值后的值不变；当有符号数为负值时，赋值后的值就会发生变化。例如：

```
signed int s_int1=100;       //s_int1=100
signed int s_int2=-100;      //s_int1=-100
unsigned int i;
i=s_int1;                    //i=100
i=s_int2;                    //i=65436
```

- 将无符号数赋给长度相同的有符号数据时，赋值机制同上。如果无符号数的最高位为 1，赋值后按负数处理。例如：

```
unsigned char u_chr=0x87;    //u_chr=135
signed char s_chr;
s_chr=u_chr;              //s_chr=-7
```

2.5.3 增量、减量运算符与增量、减量表达式

增量运算符“++”的作用是使变量的值加 1，减量运算符“--”的作用是使变量的值减 1。增量、减量表达式随着运算符的位置不同有不同的形式和含义，如表 2-6 所示。

表 2-6　　增量、减量表达式的不同形式与其含义

增量、减量表达式	含　义
变量++	在使用变量的值之后使变量的值加 1
变量--	在使用变量的值之后使变量的值减 1
++变量	在使用变量的值之前使变量的值加 1
--变量	在使用变量的值之前使变量的值减 1

注意　增量、减量运算符只能用于变量，而不能用于常量或表达式，如 5++或--（a+b）都是不合法的。

2.5.4 关系运算符与关系表达式

关系运算实际上就是“比较运算”，将两个表达式进行比较以判断是否和给定的条件相符。关系运算符包括：“<”（小于）、“<=”（小于等于）、“>”（大于）、“>=”（大于等于）、“==”（等于）、“!=”（不等于）。这些运算符都是是双目运算符。关系表达式的一般形式为：

```
表达式 1 关系运算符 表达式 2
```

关系表达式的结果只有两种：1（true）或 0（false）。如果表达式 1 与表达式 2 的关系符合关系运算符所给定的关系，则关系表达式的结果为 1，否则为 0。下面的例 2-10 计算出了两个变量的各种关系运算的值：

【例 2-10】对 x=5 和 y=7 进行各种关系运算。

```
#include <reg52.h>
#include <stdio.h>
void main (void)
{
   unsigned int x,y,result;
   x=5;
   y=7;

   SCON = 0x50;                    //SCON: 模式 1，8-bit UART，开始接收
   TMOD = (TMOD & 0x0F) | 0x20;  //TMOD: 定时器 1，模式 2，自动装载方式
   TH1 = 221;                      //TH1: 1200bit/s，16MHz
   IE |= 0x90;                     //IE: 允许中断
   TR1 = 1;                        //TR1: 定时器 1 运行
   TI=1;                           //TI: 设置 TI=1，以发送 UART 的第一个字符

   printf("\nx=%u,y=%u.",x,y);
```

```
    result=(x<y);                //<
    printf("\nThe result of x<y is %u",result);
    result=(x<=y);                  //<=
    printf("\nThe result of x<=y is %u",result);
    result=(x>y);                //>
    printf("\nThe result of x>y is %u",result);
    result=(x>=y);                  //>=
    printf("\nThe result of x>=y is %u",result);
    result=(x==y);                  //==
    printf("\nThe result of x==y is %u",result);
    result=(x!=y);                  //!=
    printf("\nThe result of x!=y is %u",result);
    while(1)
    {}
}
```

程序运行结果为：

```
x=5,y=7.
The result of x<y is 1
The result of x<=y is 1
The result of x>y is 0
The result of x>=y is 0
The result of x==y is 0
The result of x!=y is 1
```

2.5.5 逻辑运算符与逻辑表达式

逻辑运算符是指用形式逻辑原则来建立数值间关系的符号。逻辑运算符如表 2-7 所示。

表 2-7 逻辑运算符

逻辑运算符	含　义
&&	逻辑与
||	逻辑或
!	逻辑非

用逻辑运算符将两个表达式连接起来就构成了逻辑表达式。“&&”与“||”为双目运算符，其逻辑表达式的一般形式为：

```
表达式 1 逻辑运算符 表达式 2
```

“!”为单目运算符，其逻辑表达式的一般形式为：

```
! 表达式
```

与关系表达式相同，逻辑表达式的值也只能是 1（真）和 0（假）两种情况。逻辑“与”和逻辑“或”的真值表如表 2-8 所示。

表 2-8 逻辑“与”和逻辑“或”的真值表

逻辑运算符	表达式 1 的值	表达式 2 的值	逻辑表达式的值
&&	假	假	假
	假	真	假
	真	假	假

	真	真	真
||	假	假	假
	假	真	真
	真	假	真
	真	真	真

逻辑“非”的真值表如表 2-9 所示。

表 2-9　　逻辑“非”的真值表

逻辑运算符	表达式的值	逻辑表达式的值
!	假	真
	真	假

需要注意的是，编译系统在给出逻辑运算结果时，以数值 1 代表“真”，以 0 代表“假”，如上节中例子运行的结果所示；但判断一个量是否为真时，以 0 代表“假”，以非零代表“真”，即凡是不为 0 的量均认为是“真”。例如，对于 a=2，由于 a 是一个非零值，因此，进行逻辑运算时会认为 a 为“真”，则!a 为“假”，用 0 表示，即!a=0。

注意

如果使用“&&”来连接两个表达式，若第一个表达式的值为假，则不再求解第二个表达式，因为使用“&&”连接的两个表达式都为真时，整个逻辑表达式的值才为真，所以若第一个表达式的值为假就没有必要再求解第二个表达式。

同理，如果使用“||”来连接两个表达式，若第一个表达式的值为真，则不再求解第二个表达式，因为使用“||”连接的两个表达式都为假时，整个逻辑表达式的值才为假，所以若第一个表达式的值为真就没有必要再求解第二个表达式。

例 2-11 演示了逻辑运算符和逻辑表达式的用法。

【例 2-11】逻辑运算符和逻辑表达式的用法。

```
#include <reg52.h>
#include <stdio.h>

void main (void)
{
    unsigned int x,y,a,b,result;
    x=5;
    y=7;
    a=0;
    b=2;

    SCON = 0x50;                    //SCON: 模式 1，8-bit UART，开始接收
    TMOD = (TMOD & 0x0F) | 0x20;  //TMOD: 定时器 1，模式 2，自动装载方式
    TH1 = 221;                      //TH1: 1200bit/s，16MHz
    IE |= 0x90;                     //IE: 允许中断
    TR1 = 1;                        //TR1: 定时器 1 运行
    TI=1;                           //TI: 设置 TI=1，以发送 UART 第一个字符

    printf("\nx=%u,y=%u;a=%u,b=%u.",x,y,a,b);
    result=!x;                      // !
    printf("\nThe result of !x is %u",result);
    result=!a;                      // !
    printf("\nThe result of !a is %u",result);
```

```
    result=(x<=y)&&(a>b);          //&&
    printf("\nThe result of (x<=y)&&(a>b) is %u",result);
    result=(x>y)||(!a);              // ||
    printf("\nThe result of (x>y)||(!a) is %u",result)

     while(1)
    {}
}
```

程序运行结果为：

```
x=5,y=7;a=0,b=2.
The result of !x is 0
The result of !a is 1
The result of (x<=y)&&(a>b) is 0
The result of (x>y)||(!a) is 1
```

2.5.6 位运算符与位运算表达式

C51 是面向 8051 系列单片机的语言，在单片机的实际应用中，经常需要控制某一个二进制位，因此，C51 提供了对位运算的完全的支持。位运算符如表 2-10 所示。

表 2-10 位运算符

位 运 算 符	含 义
&	按位与
|	按位或
^	按位异或
～	按位取反
>>	位右移
<<	位左移

位运算符中，除了按位取反运算符“～”是单目运算符外，其他的位运算符都是双目运算符。位运算符的运算量只能是整型或字符型数据。

1．按位与运算符“&”

参加运算的两个运算量，如果两个相应的位都是 1，则结果值中的该位为 1，否则为 0。例如，3&5 的运算过程如下：

```
3 的补码：00000011
5 的补码：00000101
-----------------
3&5：00000001
```

按位与有如下用途。

- 清除一个数中的某些特定位。例如，要将 a=0x33 的最低位清零而其他位不变，只需将 a 与 0xFE 按位与即可。

```
a 的补码：00110011
0xFE 的补码：11111110
--------------------
a&0xFE：00110010
```

- 取出一个数中的某些特定位。例如，要取出 a=0x33 的低 4 位，只需将 a 与 0x0F 按位与即可。

a 的补码：00110011
0x0F 的补码：00001111
a&0x0F：00000011

2．按位或运算符“|”

参加运算的两个运算量，如果两个相应的位至少有一个是 1，则结果值中的该位为 1，否则为 0。例如，315 的运算过程如下：

3 的补码：00000011
5 的补码：00000101
3|5：00000111

按位或运算常用来对一个数据的某些特定位置 1。例如，要将 a=0x33 的最高位置 1 而其他位不变，只需将 a 与 0x80 按位或即可。

a 的补码：00110011
0x80 的补码：10000000
a|0x80：10110011

3．按位异或运算符“^”

参加运算的两个运算量，如果两个相应的位相同，即均为 1 或均为 0，则结果值中的该位为 0，否则为 1。例如，3^5 的运算过程如下：

3 的补码：00000011
5 的补码：00000101
3^5：00000110

按位或运算常用来对一个数据的某些特定位进行翻转。例如，要将 a=0x33 的低 4 位翻转而其他位不变，只需将 a 与 0x0F 按位异或即可：

0x33 的补码：00110011
0x0F 的补码：00001111
a^0x0F：00111100

4．按位取反运算符“~”

“~”是一个单目运算符，用来对一个二进制数按位取反，即将 0 变 1，1 变 0。例如，对 a=0x33 按位取反：

a 的补码：00110011
~a：11001100

5．右移运算符“>>”

右移运算符用来将一个数的各二进制位全部右移若干位，移到右端的低位被舍弃。对无符号数或者有符号数中的正数，左边高位移入 0；对有符号数中的负数，左边高端移入 1。例如 a=0x33，a>>2 表示将 a 中各二进位右移 2 位，结果为 0x0c。

a 的补码：00110011
a>>2：00001100 11
移入 移出

右移 1 位相当于除以 2，右移 n 位相当于除以 2^n，因此 a>>2 相当于 a/4。

6．左移运算符“<<”

右移运算符用来将一个数的各二进制位全部左移若干位，移到左端的高位被舍弃，右边的低位补 0。例如 a=0x33，a<<2 表示将 a 中各二进位左移 2 位，结果为 0x0c。

a 的补码：00110011

a<<2：00110011 00

移出 移入

左移 1 位相当于乘以 2，左移 *n* 位相当于乘以 2*n*，因此 a<<2 相当于 a*4。

2.5.7 复合赋值运算符与复合赋值表达式

凡是二目运算符都可以和赋值运算符结合组成复合赋值运算符。C 语言规定可以使用以下 10 种复合赋值表达式：

```
+=, -=, ×=, /=, %=, <<=, >>=, &=, ^=, |=
```

复合赋值表达式的一般形式为：

```
变量 复合赋值运算符 表达式
```

例如：

```
i+=1;        //等价于 i=i+1
x/=y+1;      //等价于 x=x/(y+1)
```

采用这些复合运算符和复合运算表达式，有利于简化程序书写，并且可以提高编译效率，产生质量较高的目标代码。

2.5.8 逗号运算符与逗号表达式

C51 提供了一种特殊运算符——逗号运算符，用逗号运算符可以把两个或多个表达式连接起来，形成逗号表达式。逗号表达式的一般形式为：

```
表达式 1，表达式 2，……，表达式 n
```

逗号表达式的求解过程是从左到右依次计算出每个表达式的值，整个逗号表达式的值等于最右边的表达式（表达式 *n*）的值。例如：

```
unsigned int a,b;
b=2;
a=(b=b×3,b-4);  //逗号表达式
```

执行代码中的逗号表达式时，先计算第一个表达式 b=b×3，然后计算 b−4，将 b−4 的结果赋给 a。需要注意，执行完上述逗号表达式后，b 的值是 6，因为 b−4 这个表达式并没有赋值给 b。上述代码等价于以下代码：

```
unsigned int a,b;
b=2;
b=b×3;
a=b-4;
```

程序中使用逗号表达式，往往并不一定要得到和使用整个逗号表达式的值，而只是为了分别求逗号表达式内各表达式的值。另外，并非程序中任何地方出现的逗号都是逗号运算符，例如在变量定义或函数参数表中，逗号就不是逗号运算符，而是用作各变量之间的间隔符。

2.5.9 条件运算符与条件表达式

条件运算符“?:”是惟一的一个三目运算符，条件表达式的一般形式为：

```
逻辑表达式? 表达式 1: 表达式 2
```

条件表达式的求解过程是首先计算逻辑表达式的值，如果为 1（true），则整个表达式的值为表达式 1 的值，否则为表达式 2 的值。例如：

```
unsigned int x,y;
x=20;
y=x>10?2:1;
```

由于逻辑表达式 x>=10 的值为 1，因此整个表达式的值就是表达式 1 的值，运行的结果为 y=2。

2.5.10 指针与地址运算符

C51 提供了“*”与“&”两个单目运算符，前者的作用是返回一个地址内的变量值，即取内容；后者的作用是返回操作数的地址，即取地址。这两种运算的一般形式分别为：

```
变量=*指针变量
指针变量=&目标变量
```

取内容运算是将指针变量所指向的目标变量的值赋给左边的变量；取地址运算是将目标变量的地址赋给左边的变量。例 2-12 演示了如何取变量的地址和如何取指针变量所指向的变量的值：

【例 2-12】指针与地址运算符的用法。

```
#include <reg52.h>
#include <stdio.h>

void main(void)
{
    unsigned int a,b;
    unsigned int *p_a,*p_b;          //定义两个 unsigned int 型的指针变量
    a=10;
    b=20;
    p_a=&a;                          //地址运算符，把 a 的地址赋给 p_a
    p_b=&b;                          //把 b 的地址赋给 p_b

    SCON = 0x50;                     //SCON: 模式 1，8-bit UART，开始接收
    TMOD = (TMOD & 0x0F) | 0x20;  //TMOD: 定时器 1，模式 2，自动装载方式
    TH1 = 221;                       //TH1: 1200bit/s，16MHz
    IE |= 0x90 ;                   //IE: 允许中断
    TR1 = 1;                         //TR1: 定时器 1 运行
    TI=1;                            //TI: 设置 TI=1，以发送 UART 第一个字符

    printf("\nThe address of a is %p",p_a);
    printf("\nThe address of b is %p",p_b);
    printf("\nThe value of *p_a is %u.",*p_a);//取 p_a 所指向的变量的内容
    printf("\nThe value of *p_b is %u.",*p_b);//取 p_b 所指向的变量的内容
    while(1)
```

```
    {}
}
```

程序运行结果为：

```
The address of a is i:0022
The address of b is i:0024
The value of *p_a is 10.
The value of *p_b is 20.
```

2.5.11 C51 运算符的优先级

C51 语言规定了运算符的优先级。在对有多个运算符参加运算的表达式求值时，按照运算符的优先级别高低次序执行，例如先乘除后加减。如果在设计程序时不注意这一点，往往会导致错误的结果。例如，原意是将 a 右移 2 位后再与 b 相加，结果赋给 c，即：

```
c=b+(a>>2);
```

但由于没有注意运算符的优先级，写成了如下形式：

```
c=b+a>>2;
```

由于“+”的优先级高于“>>”，因此，上面的语句实际相当于：

```
c=(b+a)>>2;
```

其运行过程是先将 a、b 相加，再将相加的结果右移 2 位后赋给 c，这就和程序设计的原意大相径庭。运算符的优先级如表 2-11 所示，表中优先级从上往下逐渐降低，同一行优先级相同。

表 2-11　　运算符的优先级

运　算　符	优　先　级
()（小括号）、[]（数组下标）、.（结构成员）、→（结构成员）	最高
!（逻辑非）、~（按位取反）、-（负号）、++（加 1）、--（减 1）、&（取地址）	↑
*（取内容）、sizeof（长度计算）	↑
*（乘）、/（除）、%（求余）	↑
+（加）、-（减）	↑
<<（位左移）、>>（位右移）	↑
<（小于）、<=（小于等于）、>（大于）、>=（大于等于）	↑
==（等于）、!=（不等于）	↑
&（按位与）	↑
^（按位异或）	↑
|（按位或）	↑
&&（逻辑与）	↑
||（逻辑或）	↑
?:（条件表达式）	↑
=（赋值）、+=、-=、*=、/=、%=、<<=、>>=、&=、^=、|=（复合赋值）	↑
,（逗号运算符）	最低

2.6 小结

C51 中的基本数据类型有字符型（char）、基本整型（int）、短整型（short int）、长整型（long in）、无符号型（unsigned int）、无符号长整型（unsigned long）、单精度实型（float）、双精度实型（double）等，各种类型均有其长度和表示范围。

C51 中的运算量分为常量和变量。常量是在程序运行过程中其值保持不变的量，变量是在程序运行过程中其值可以变化的量。

在不同类型数据的混合运算中，由系统自动实现转换，由少字节类型向多字节类型转换。不同类型的量相互赋值时也由系统自动进行转换，把赋值号右边的类型转换为左边的类型。不同类型的数据之间也可以由强制转换运算符完成转换。

C51 中的运算符有着不同的优先级。一般而言，单目运算符优先级较高，赋值运算符优先级低；算术运算符优先级较高，关系和逻辑运算符优先级较低。

表达式是由运算符连接常量、变量所组成的式子。每个表达式都有一个值和类型，表达式求值按运算符的优先级和结合性所规定的顺序进行。

联系方式

集团官网：www.hqyj.com　　嵌入式学院：www.embedu.org　　移动互联网学院：www.3g-edu.org

企业学院：www.farsight.com.cn　　物联网学院：www.topsight.cn　　研发中心：dev.hqyj.com

集团总部地址：北京市海淀区西三旗悦秀路北京明园大学校内 华清远见教育集团

北京地址：北京市海淀区西三旗悦秀路北京明园大学校区，电话：010-82600386/5

上海地址：上海市徐汇区漕溪路 250 号银海大厦 11 层 B 区，电话：021-54485127

深圳地址：深圳市龙华新区人民北路美丽 AAA 大厦 15 层，电话：0755-25590506

成都地址：成都市武侯区科华北路 99 号科华大厦 6 层，电话：028-85405115

南京地址：南京市白下区汉中路 185 号鸿运大厦 10 层，电话：025-86551900

武汉地址：武汉市工程大学卓刀泉校区科技孵化器大楼 8 层，电话：027-87804688

西安地址：西安市高新区高新一路 12 号创业大厦 D3 楼 5 层，电话：029-68785218

广州地址：广州市天河区中山大道 268 号天河广场 3 层，电话：020-28916067