FUJITSU

C 语言手册

基础篇

富士通复旦应用研究中心

目录

第一章　C 语言概述

C 语言是一种通用的计算机程序设计语言，在国际上十分流行，它既可用来编写计算机的系统程序，也可用来编写一般的应用程序。以前计算机的系统软件主要使用汇编语言编写的，对于单片机应用系统来说更是如此。由于汇编语言程序的可读性和可移植性都较差，采用汇编语言编写单片机应用系统程序的周期长，而且调试和排错也比较困难。而一般效率高的高级语言难以实现汇编语言对于计算机硬件直接进行操作（如对内存地址的操作、移位操作等）的功能。而 C 语言既具有一般高级语言的特点，又能直接对计算机的硬件进行操作，并且采用 C 语言编写的程序能够很容易地在不同类型的计算机之间进行移植，因此许多以前只能采用汇编语言来解决的问题现在可以改用 C 语言来解决。

第一节　C 语言的特点

C 语言可以用来编写科学计算或其他应用程序，但它更适合于编写计算机的操作系统程序以及其他一些需要对机器硬件进行操作的场合，有的大型应用软件也采用 C 语言进行编写。这主要是因为 C 语言具有很好的可移植性和硬件控制能力，表达和运算能力也较强。

概括来说，C 语言具有以下一些特点：

1. 语言简洁紧凑，使用方便灵活

   C 语言一共只有 32 个关键字，9 中控制语句，主要用小写字母表示，压缩了一切不必要的成分；C 语言程序书写形式自由，可以用简单的方法构造除复杂的数据类型和程序结构。

2. 运算符丰富

   C 语言把括号、赋值、强制类型转换等都作为运算符处理，从而使 C 的运算类型极其丰富，共有 34 种运算符。C 表达式类型多样化，灵活使用各种运算符可以实现其他高级语言难以实现的运算。

3. 数据结构类型丰富

   C 的数据结构类型丰富，根据需要可以采用：整型、实型、字符型、数组类型、指针类型、结构体类型、共用体类型等多种数据类型来实现复杂数据结构的运算。尤其是指针类型数据，使用起来非常灵活多样。

4. 可进行结构化程序设计

   C 语言是以函数作为程序设计的基本单位的，用函数作为程序模块以实现程序的模块化，是结构化的理想语言。

5. 语法限制不严格，程序设计自由度大

   C 语言的语法规则不太严格，程序设计的自由度比较大。“限制”和“灵活”是一对矛盾。C 语言放宽了语法检查，所以程序员应当仔细检查程序，而不要过分依赖 C 编译程序去查错。

6. C 语言允许直接访问物理地址

   C 语言允许直接访问物理地址，能进行位（bit）操作，能实现汇编语言的大部分功能，可以直接对硬件进行操作。这样它可以对单片机的内部寄存器和 I/O 口进行操作，可以直接访问片内或片外存储器。

7. 生成目标代码质量高

众所周知，汇编语言程序目标代码的效率是最高的。但统计表明，C 语言编写的程序生成代码的效率仅比汇编语言低 10~20%。

8. 程序可移植性好

汇编语言完全依赖于机器硬件，因而不具有可移植性。C 语言是通过编译来得到可执行代码的，C 语言的编译程序便于移植，基本上不作修改就能用于各种机器和操作系统。

尽管 C 语言具有许多的优点，但和其他任何一种程序设计语言一样也有其自身的缺点，但总的来说，C 语言的优点远远超过了它的缺点，经验表明，程序设计人员一旦学会使用 C 语言后，就会对它爱不释手，单片机应用系统的程序设计人员更是如此。

第二节　C 语言的程序结构

C 语言程序是由若干个函数单元组成的，每个函数都是完成某个特殊任务的子程序段。组成一个程序的若干个函数可以保存在一个源程序文件中，也可以保存在几个源程序文件中，最后再将它们连接在一起。C 语言源程序文件的扩展名为“.c”。

一个 C 语言程序必须有而且只能有一个名为 main( )的函数，它是一个特殊的函数，也称为该程序的主函数，程序的执行都是从 main（）函数开始的。下面我们先来看一个简单的程序例子。

[例 1.1]　已知 x=10，y=20，计算 z=x+y 的结果。

```
Main（）                    /* 主函数名 */
{                           /* 主函数体开始 */
    int  x，y，z；          /* 主函数的内部变量类型说明 */
    x =10；y =20；          /* 变量赋值 */
    z = x + y；             /* 计算 z = x + y 的值 */
}                           /* 程序结束 */
```

本例的作用是求两个整数 x 和 y 之和 z。

其中 main 是主函数名，要执行的主函数内容称为主函数体，主函数体用花括弧号“{ }”围起来。函数体中包含若干条将被执行的程序语句，每条语句都必须以分号“;”为结束符。

/* …… */表示注释部分。注释是为了使程序便于阅读和理解，给人看的，对编译和运行不起作用。注释可以加在程序中任何位置，由“/*”开始，“*/”结束。

例 1 的程序很简单，它只有一个主函数 main（）。下面我们再来看一个例子。

[例1.2]　求最大值。

```
Int max（int x，int y）；
Main（）                    /* 主函数名 */
{                           /* 主函数体开始 */
int a，b，c；               /* 主函数的内部变量类型说明 */
a=10；b=20；                /* 变量赋值 */
c=max（a，b）；             /* 调用 max 函数，将得到的值赋给 c */
}

int max（x，y）；           /* 定义函数 max，函数值为整型，x，y 为形式参数 */
int x，y；                  /* 对形参 x，y 作类型定义 */
```

```
{
int z;                    /* max 函数中用到的变量 z，也要加以定义 */
if（x>y）      z=x;       /* 计算最大值 */
else       z=y;
return（z）;              /*将计算的最大值 z 返回，通过 max 带回调用处 */
}
```

本例中除了 main（）函数之外，还用到了功能函数调用。

函数 max 是一个被调用的功能函数，其作用是将变量 x 和 y 中较大者的值赋给变量 z。变量 x 和 y 在函数 max 中是一种形式变量，它的实际值是通过 main（）函数中的调用语句传送过来的。变量 z 是函数 max 要返回的值，return 语句将 z 的值返回给 main（）函数的调用处。

Main 函数中第三行调用 max 函数，在调用时将实际参数 a 和 b 的值分别传送给 max 函数中的形式参数 x 和 y。经过执行 max 函数得到一个返回值（即 max 函数中变量 z 的值），把这个值赋给变量 c。

通过以上两个例子，可以看到一般 C 语言程序具有如下结构：

```
预处理命令          #include< >
函数说明            int   fun1（）;
                    char fun2（）;
功能函数1           fun1（）
                    {
函数体              …
                    }
主函数              main（）
                    {
主函数体            …
                    }
功能函数 2          fun2（）
                    {
函数体              …
                    }
                    ….
```

C 语言的开头部分通常是预处理命令，#include 命令。它是通知编译器在对程序进行编译时，将所需要的头文件读入后再一起进行编译。

C 语言程序是由函数所组成的。一个程序至少应包含一个主函数 main（）。函数之间可以相互调用，但 main（）函数只能调用其他的功能函数，而不能被其他函数所调用。不管 main（）函数处于程序中的什么位置，程序总是从 main（）函数开始执行。

第三节　C 语言的关键字和标识符

C 语言的标识符是用来标识源程序中某个对象名字的。这些对象可以是函数、变量、常量、数组、数据类型、存储方式、语句等。一个标识符由字符串、数字和下划线等组成，第一个字符必须是字母或下划线，通常以下划线开头的标识符是编译系统专用的，因此在编写 C 语言源程序是一般不要使用下划线开头的标识符，而将下划线用作分段等。

以下标识符是合法的：

a, x, 3x,BOOK 1,sum5

以下标识符是非法的：

3s　　以数字开头

s*T　　出现非法字符*

-3x　　以减号开头

bowy-1　出现非法字符-(减号)

使用标识符时应注意，标识符的大小写是有区别的。例如 BOOK 和 book 是两个不同的标识符。另外，标识符虽然可由程序员随意定义，但标识符是用于标识某个量的符号。因此，命名应尽量有相应的意义，以便于阅读理解，作到“顾名思义”。

关键字是由C语言规定的具有特定意义的字符串，其实是一类具有固定名称和特定含义的特殊标识符，通常也称为保留字。用户定义的标识符不应与关键字相同。C语言的关键字分为以下几类：

1．类型说明符

用于定义、说明变量、函数或其它数据结构的类型。如前面例题中用到的 int,double。

2．语句定义符

用于表示一个语句的功能。如例 1.3 中用到的 if else 就是条件语句的语句定义符。

3．预处理命令字

用于表示一个预处理命令。如前面各例中用到的 include。

C 语言中的关键字见附录 1。

第二章　C 语言的基本语法

第一节　数据类型

在 C 语言中, 每个变量在使用之前必须定义其数据类型。C 语言有以下几种数据类型：整型(int)、浮点型(float)、字符型(char)、指针型(*)、无值型(void)、数组类型(array)以及结构(struct)和联合(union)。其中前五种是 C 的基本数据类型、后三种数据类型(数组、结构、联合)将在后面介绍。

2.1.1 整型（int）

2.1.1.1 整型数说明

加上不同的修饰符, 整型数有以下几种类型;

signed short int	有符号短整型数，简写为 short 或 int。 字长为 2 字节共 16 位二进制数, 数的范围是-32768~32767。
signed long int	有符号长整型数，简写为 long, 字长为 4 字节共 32 位二进制数, 数的范围是-2147483648~2147483647。
unsigned short int	无符号短整型数，简写为 unsigned int。 字长为 2 字节共 16 位二进制数, 数的范围是 0~65535。
unsigned long int	无符号长整型数说明。简写为 unsigned long, 字长为 4 字节共 32 位二进制数, 数的范围是 0~4294967295。

2.1.1.2 整型变量定义

可以用下列语句定义整型变量

```
int a, b;                /*a、b 被定义为有符号短整型变量*/
unsigned long c;         /*c 被定义为无符号长整型变量*/
```

2.1.1.3 整型常量表示

按不同的进制区分, 整型常量有三种表示方法:

十进制数:　以非 0 开始的数，如:220, -560, 45900

八进制数:　以 0 开始的数，如:06; 0106, 05788

十六进制数:以 0X 或 0x 开始的数，如:0X0D, 0XFF, 0x4e

另外, 可在整型常量后添加一个"L"或"l"字母表示该数为长整型数, 如 22L, 0773L, 0Xae4l。

2.1.2 浮点型（float）

2.1.2.1 浮点数说明

C 中有以下两种类型的浮点数:

float　　单浮点数。字长为 4 个字节，数的范围是 3.4x10-38E~3.4x10+38E。

double　　双浮点数。字长为 8 个字节，数的范围是 1.7x10-308E~1.7x10+308E。

说明:

浮点数均为有符号浮点数，没有无符号浮点数。

2.1.2.2 浮点型变量定义

可以用下列语句定义浮点型变量:

```
float a, f;        /*a, f 被定义为单浮点型变量*/
double b;          /*b 被定义为双浮点型变量*/
```

2.1.2.3 浮点常量表示

例:　+29.56, -56.33, -6.8e-18, 6.365

说明:

1. 浮点常数只有一种进制(十进制)。
2. 所有浮点常数都被默认为 double。
3. 绝对值小于 1 的浮点数，其小数点前面的零可以省略。如:0.22 可写为.22, -0.0015E-3 可写为-.0015E-3。
4. 在 C 语言中，默认格式输出浮点数时，最多只保留小数点后六位。

2.1.3 字符型（char）

2.1.3.1 字符型数据说明

加上不同的修饰符，可以定义有符号和无符号两种类型的字符型变量，例如:

```
char a:                  /*a 被定义为有符号字符变量*/
unsigned char l;         /*l 被定义为无符号字符变量*/
```

在计算机中以其 ASCII 码方式表示，其长度为 1 个字节，有符号字符型数取值范围为-128~127，无符号字符型数到值范围是 0~255。因此在 C 语言中，字符型数据在操作时将按整型数处理，如果某个变量定义成 char，则表明该变量是有符号的，即它将转换成有符号的整型数。

C 中规定对 ASCII 码值大于 0x80 的字符将被认为是负数。例如 ASCII 值为 0x8c 的字符，定义成 char 时，被转换成十六进制的整数 0xff8c 。这是因当 ASCII 码值大于 0x80 时，该字节的最高位为 1，计算机会认为该数为负数，对于 0x8c 表示的数实际上是-74(8c

的各位取反再加 1)，而-74 转换成两字节整型数并在计算机中表示时就是 0xff8c(　对 0074 各位取反再加 1)。因此只有定义为 unsigned char 0x8c 转换成整型数时才是 8c。这一点在处理大于 0x80 的 ASCII 码字符时(例如汉字码)要特别注意。一般汉字均定义为 unsigned char(在以后的程序中会经常碰到)。

另外，也可以定义一个字符型数组(关于数组后面再作详细介绍)，此时该数组表示一个字符串。例如:　　char str[10];

计算机在编译时，将留出连续 10 个字符的空间，即 str[0]到 str[9]共 10 个变量，但只有前 9 个供用户使用。第 10 个 str[9]用来存放字符串终止符 NULL 即"\0"，但终止符是编编译程序自动加上的，这一点应特别注意。

2.1.3.2 字符常量表示

能用符号表示的字符可直接用单引号括起来表示，如'a'，'9'，'Z'，　也可用该字符的 ASCII 码值表示，例如十进制数 85 表示大写字母'U'，十六进制数 0x5d 表示']'，八进制数 0102 表示大写字母'B'。

一些不能用符号表示的控制符，只能用 ASCII 码值来表示，如十进制数 10 表示换行，下六进制数 0x0d 表示回车，八进制数 033 表示 Esc。C 语言中也有另外一种表示表示方法，如'\033'表示 Esc，　这里'\ 0' 符号后面的数字表示十六进制的 ASCII 值当然这种表示方法也适用于可睦接用符号表示的字符。

另外，有些常用的字符用以下特殊规定来表示:

字符形式	功能
\n	换行
\t	横向跳格
\v	竖向跳格
\b	退格
\r	回车
\f	走纸换页
\\	反斜杠字符
\'	单引号
\ddd	8 进制数表示的对应 ASCII 码字符
\xhh	16 进制数表示的对应 ASCII 码字符

表 2.1 常用的特殊字符

C 语言除了允许使用字符常量外，还允许使用字符串常量。字符串常量一般用双引号括起来表示，如"Hello World"。

2.1.4 指针型（*）

指针是一种特殊的数据类型，在其它语言中一般没有。指针是指向变量的地址，实质上指针就是存贮单元的地址。根据所指的变量类型不同，可以是整型指针(int *)、浮点型指针(float *)、字符型指针(char *)、结构指针(struct *)和联合指针(union *)。

2.1.5 无值型（void）

无值型字节长度为 0, 主要有两个用途:　一是明确地表示一个函数不返回任何值; 一是产生一个同一类型指针(可根据需要动态分配给其内存)。

例如:

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

2.1.7 各类数值型数据间的混合运算

整型、单精度型、双精度型数据可以混合运算。字符型数据可以与整型通用，所以，各类数值型数据间可以混合运算。在进行运算时，不同类型的数据要先转换成同一类型，然后进行运算。转换的规则如图 2.1 所示。

图中横向向左的箭头表示必定的转换，如字符数据必定先转换为整数，short 型转为 int 型等。纵向的箭头表示当运算对象为不同类型时转换的方向。例如 int 型与 double 型数据进行运算，先将 int 型数据转换成 double 型，然后在两个同类型（double 型）数据间进行运算，结果为 double 型。上述的类型转换是由系统自动进行的。

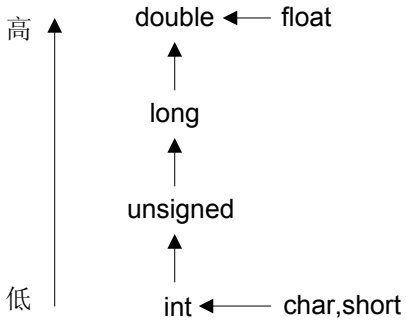


图 2.1 各类数据类型混合运算的转换

<注意〉:

箭头方向只表示数据类型级别的高低，由低向高转换。并非是 int 型先转成 unsigned 型，再转成 long 型，再转成 double 型。

第二节　常量与变量

2.2.1 常量和变量

C 语言的数据有常量和变量之分。

常量：在程序运行过程中，其值不能改变的量称为常量。常量可以有不同的数据类型。

变量：在程序运行中，其值可以改变的量称为变量。变量是由变量名和变量值组成的。每个变量都有一个变量名，在内存中占据一定的存储单元（地址），并在该内存单元存放该变量的值。

对于单片机来说，变量类型或数据类型的选择是值得一提的。对于 C 这样的高级语言，不管使用何种数据类型，从字面上看其操作都非常简单。然而，实际上编译器需要一系列机器指令对其进行复杂的变量类型、数据类型处理。特别是浮点运算，明显增加运算时间和程序的长度。所以应该慎重进行变量和数据类型的选择，不要使用大量不必要的变量类型。

<注意>：

1．有符号数和无符号数选择

在编写程序时，如果使用 signed 和 unsigned 两种数据类型，那么就得使用两种格式类型的库函数，将是占用的存储空间增长。因此，如果只强调运算速度而又不进行负数运算时，最好采用无符号（unsigned）格式。

2．无符号字符类型的使用

无论何时，应尽可能地使用无符号字符变量，字符变量的长度为 1 字节，这很适合单片机。而有符号字符变量（signed char）虽然也只占有一个字节，但需要进行额外的操作来测试代码的符号位，这无疑会降低代码效率。

2.2.2 变量赋初值

程序中常需要对一些变量预先设置初值。这个初始化不是在编译阶段完成的，而是在程序运行时执行本函数时赋以初值的。

C 规定，可以在定义变量时同时使变量初始化，如：

```
int  a=3;          /* 指定 a 为整型变量，初值为 3 */
```

也可以使被定义的变量的一部分赋初值，如：

```
int  a,b,c=3;      /* 表示 a、b、c 为整型变量，只对 c 初始化，值为 3 */
```

如果对几个变量赋以同一初始值，但不表示在整个程序中三个变量一直相等，如：

```
int  a=b=c=3;      /* 表示 a、b、c 的初值都是 3 */
```

第三节　运算符和表达式

C 语言中运算符和表达式数量之多， 在高级语言中是少见的。运算符按其在表达式中所起的作用，可分为赋值运算符、算术运算符、关系运算符、逻辑运算符、位运算符、逗号运算符、条件运算符、指针和地址运算符、强制类型转换运算符和 sizeof 运算符等。根据运算符在表达式中与运算对象的关系，又可分为单目运算符、双目运算符和三目运算符。单目运算符既是只需要一个运算对象，双目运算符则要求由两个运算对象。

2.3.1 算术运算符和算术表达式

算术运算符用于各类数值运算。包括加(+)、减(-)、乘(*)、除(/)、求余(或称模运算，%)、自增(++)、自减(--)共七种。除了求余运算符只用于整型、长整型、字符型外，其他用于所有数据类型。

后两种运算符是 C 语言特有的运算符，常用于循环语句中使循环变量自动加（减）1，也用于指针变量，使指针指向下一个地址。自增运算符和自减运算符的作用是使变量加 1 或减 1，它们只能用于变量，不能用于常量或表达式。关于自增、自减运算符，需要注意的是下列两种使用方法的不同之处：

++i，--i　　在使用 I 之前，先使 I 的值加（减）1
i++，i--　　在使用 I 之后，使 I 的值加（减）1

用算术运算符和括号将运算对象（也称操作数）连接起来、符合 C 语法规则的式子，称为算术表达式。C 语言规定了运算符的优先级和结合性。表达式求值时，先按运算符的优先级别高低次序执行，优先级别相同则按“结合方向”执行。算术运算符的优先级为：先自增、自减，再乘、除、求余，后加、减。 前五种运算符的结合方向是“自左向右”，后两种运算符++ 和--的结合方向是“自右至左”。

2.3.2 赋值运算符和赋值表达式

2.3.2.1 基本赋值运算符

在 C 语言里，符号“=”是个特殊的运算符，称为赋值运算符。它的作用时将一个数据的值赋给一个变量，利用赋值运算符将一个变量与表达式连接起来的式子称为赋值表达式，在赋值表达式的后面加一个分号“;”就构成了赋值语句。一个赋值语句的格式如下：

变量=表达式;

该语句的意义是先计算出右边表达式的值，然后将该值赋给左边的变量。上式中的“表达式”还可以是一个赋值表达式，即 C 语言允许多重赋值。例如：

```
x = 9;          /* 将常数 9 赋给变量 x */
x = y = 8;      /* 将常数 8 同时赋给变量 x 和 y */
```

都是合法的赋值语句。

2.3.2.2 复合赋值运算符

在赋值赋“=”之前加上其他运算符，可以构成复合的运算符。复合赋值运算首先对变量进行某种运算，然后将运算的结果在赋给该变量。复合赋值运算表达式的一般形式为：

变量　　复合赋值运算符　表达式;

例如：a + = 3 等价于 a = a+3；x * =y + 8 等价于 x=x *（y+8）。凡是二目运算符，都可以与赋值符一起组合成复合赋致富。C 语言规定可以使用 10 种复合赋值运算符。即：

+=，-=，*=，/=，%=，《=，　》=，&=，^=，|=

后 5 中是有关位运算的，将在后面章节介绍。

2.3.3 关系运算符和关系表达式

2.3.3.1 关系运算符

所谓“关系运算”实际上是“比较运算”。将两个值进行比较，判断比较的结果是否符合给定的条件。C 语言提供了 6 种关系运算符：

大于	>	大于等于	>=
小于	<	小于等于	<=

等于　　==　　不等于　　!=

2.3.3.2 关系运算符的优先级：

1) >、>=、<、<=优先级相等，==、!=优先级相等，前者高于后者。
2) 关系运算符的优先级低于算术运算符。
3) 关系运算符的优先级高于赋值运算符。

2.3.3.3 关系表达式：

用关系运算符将若干个表达式连接起来即成为关系表达式。关系表达式的一般形式为：

表达式 1　关系运算符　表达式 2

关系表达式通常用来判别某个条件是否满足，关系表达式的值是逻辑值，可以是“真”或“假”。C 语言中用“0”表示“假”，用“1”表示“真”。

例：1<5 的值为真；5<1 的值为假；

a=1，b=5 时 a==b 的值为 0；a<b 的值为 1。

2.3.4 逻辑运算符和逻辑表达式

C 语言中有 3 种逻辑运算符：

&&　逻辑与；　（相当于 AND）
||　逻辑乘；　（相当于 OR）
!　逻辑非；　（相当于 NOT）

逻辑运算符的优先级为：！高于 && 高于 || 。其中&&、|| 低于关系运算符，！高于算术运算符。

逻辑运算符用来求某个条件时的逻辑值，用逻辑运算符将关系表达式或逻辑量连接起来就是逻辑表达式。逻辑表达式的一般形式为：

逻辑与：条件式 1 && 条件式 2
逻辑或：条件式 1 || 条件 2
逻辑非：！条件式

逻辑运算的真值表见表 2.2:

表达式 a	表达式 b	!a	!b	a&&b	a || b
1（真）	1（真）	0（假）	0（假）	1（真）	1（真）
1（真）	0（假）	0（假）	1（真）	0（假）	1（真）
0（假）	1（真）	1（真）	0（假）	0（假）	1（真）
0（假）	0（假）	1（真）	1（真）	0（假）	0（假）

表 2.2 逻辑运算真值表

2.3.5 位运算符

能对运算对象进行按位操作是 C 语言的一大特点，正是由于这一特点使 C 语言具有汇编语言的一些功能，从而能对计算机的硬件直接进行操作。C 语言中共有 6 种位运算符：

&	按位与	|	按位或	~	按位取反
^	按位异或	>>	右移	<<	左移

位运算符的作用是按位对变量进行运算，并不改变参与运算的变量的值。若希望按位改变运算变量的值，则应利用相应的复合赋值运算符。另外按位运算是对字节或字中的实际位进行检测、设置或移位，它不能用来对浮点型数据进行操作。

位运算符的优先级从高到低依次是：按位取反（~）——左移（<<）和右移（>>）——按位与（&）——按位异或（^）——按位或（|）。

要注意区别按位运算符和逻辑运算符的不同，例如，若 x=7，则 x&&8 的值为真(两个非零值相与仍为非零)，而 x & 8 的值为 0。

位运算的一般形式如下：

变量 1　位运算符　变量 2

位运算符中的移位操作比较复杂。移位运算符">>"和"<<"是指将变量中的每一位向右或向左移动，其通常形式为：

右移:　变量名>>移位的位数

左移:　变量名<<移位的位数

经过移位后，一端的位被"挤掉"，而另一端空出的位以 0 填补，所以，C 中的移位不是循环移动的。如果是对有符号类型数据进行右移操作，则在其左端补入原来数据的符号位（即保持原来的符号不变），其右段的移出位被丢弃。

2.3.6 逗号运算符

在 C 语言中，“,”是一个特殊的运算符，可以用它将两个（或多个）表达式连接起来，称为逗号表达式。逗号表达式的一般形式为：

表达式 1，表达式 2，……表达式 n

程序运行时对于逗号表达式的处理，是从左至右依次计算出各个表达式的值，而整个逗号表达式的值是最右边表达式（即表达式 n）的值。

许多情况下，使用逗号表达式的目的只是为了分别得到各个表达式的值，而并不是一定要得到和使用整个逗号表达式的值。另外，并不是在程序的任何地方出现的逗号，都可以认为是逗号运算符。例如函数中的参数也是用逗号来间隔的，但并不是逗号表达式。

2.3.6 条件运算符和条件表达式

条件运算符“？　：”是 C 语言中唯一的一个三目运算符，用它可以将三个表达式连接构成一个条件表达式。条件表达式的一般形式如下：

逻辑表达式？表达式 1：表达式 2

条件表达式作用如下：首先计算逻辑表达式，当其值为真时，将表达式 1 的值作为整个条件表达式的值；当逻辑表达式的值为假时，将表达式 2 的值作为整个条件表达式的值。例如：max = （a>b）？a：b 的执行结果是将 a 和 b 中较大值赋值给变量 max。

2.3.7 强制类型转换运算符

C 语言中的圆括号“( )”也可作为一种运算符使用，这就是强制类型转换运算符，它的作用是将表达式或变量的类型强制转换成为所指定的类型。

C 语言程序中进行算术运算时，需要注意数据类型的转换。前面第一节介绍的数据类型转换是隐式转换，是在对程序进行编译时编译器自动处理的。利用强制类型转换运算符可以进行显示转换。强制类型转换运算符的一般形式为：

（类型）= 表达式

来看下面的例子：

int a=10000，b=10000，c=10000；　　c = a*b/c；

这可能会出错，因为 a*b 超过 int 的取值范围了。在这种情况下就可以利用强制类型转换运算符来进行显式转换：将 c=a*b/c 改为 c=(long)a*b/c 就不会错了，(long)是显式数据类型的转换。

2.3.8 sizeof 运算符

C 语言中提供了一种用于求取数据类型、变量以及表达式的字节数的运算符：sizeof，该运算符的一般使用形式如下：

sizeof（表达式）或 sizeof（数据类型）

应该注意的是，sizeof 是一种特殊的运算符，不要错误地认为它是一个函数。实际上，字节数的计算在程序编译时就完成了，而不是在程序执行的过程中才计算出来的。

2.2.9 运算符的优先级和结合方向

C 语言中的运算符很多，前面介绍了常用的一些运算符，在复合运算时必须要考虑到运算符的优先级和结合方向。

有关运算符的优先级和结合性参看附录 2。这里做几点说明：

1．同一优先级的运算符优先级别相同，运算次序有结合方向决定。

例如，* 和 / 具有相同的优先级别，其结合方向为自左至右，因此，3*5/4 的运算次序是先乘后除。- 和++为同一优先级，结合方向为自右至左，因此-i++ 相当于-(i++)。

2．不同的运算符要求有不同的运算对象个数。

例如：

算术运算符+（加）和-（减）为双目运算符，要求在运算符两侧各有一个运算对象（如 3+5、8-3 等等）。

运算符++和-（负号）运算符是单目运算符，只能在运算符的一测出现一个运算对象（如 -a、i++、--i、(float) i、sizeof(int)、*p 等）。

条件运算符是 C 语言中唯一的一个三目运算符，如 x？a：b。

3．附表中的优先级别是由上到下递减。

从表中大致可以归纳出各类运算符的优先级如图 2.2 所示。初等运算符优先级最高，逗号运算符优先级最低。

位运算符的优先级比较分散（有的在算术运算符之前（如~），有的在关系运算符之前（如<<和>>），有的在关系运算符之后（如&、^、|））。为了容易记忆，使用位运算符时可以加圆括号（）。

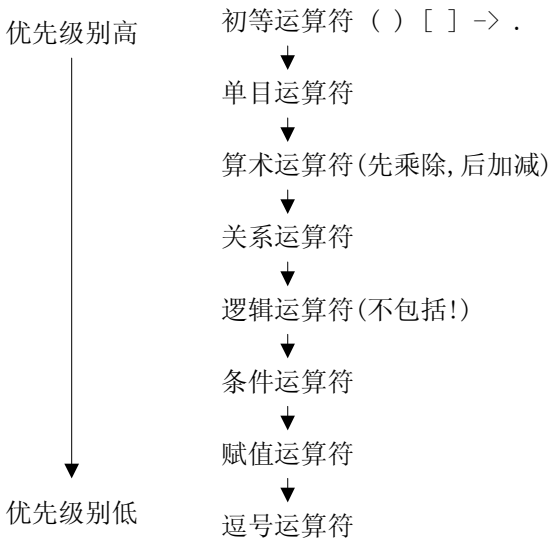


图 2.2 各类运算符优先级

第四节　表达式语句

2.4.1 基本表达式语句

表达式语句是最基本的一种语句。在表达式的后边加一个分号“;”就构成了表达式语句。下面的语句都是合法的表达式语句：

a=8；b=7；

z=（x+y）/a；

表达式语句也可以仅有一个分号“;”组成，这种语句称为空语句。空语句是表达式语句的一个特例。它在程序设计中有时是很有用的。例如，在用 while 语句构成的循环语句后面加一个分号，形成一个不执行其他操作的空循环体。通常用在等待某个事件发生时，比如用一个空语句 while（IO_SSR0.bit.RDRF==0）；来等待单片机串行口接收数据。

2.4.2 复合语句

复合语句是由若干条语句组合而成的一种语句，它是用一个大括号“{}”将若干条语句组合在一起而形成的一种功能块。复合语句不需要以分号“;”结束，但它内部的各条单语句仍需以分号“;”结束。复合语句的一般形式为：

```
{
局部变量定义;
语句 1;
语句 2;
……
语句 n;
}
```

复合语句在执行时，其中的各条单语句依次顺序执行。整个复合语句在语法上等价于一条单语句，因此在 C 语言程序中可以将复合语句视为一条单语句。复合语句允许嵌套。

通常复合语句都出现在函数中，实际上，函数的执行部分（即函数体）就是一个符合语句。复合语句中的单语句可以是变量的定义语句（说明变量的数据类型）。用复合语句内部变量定义语句所定义的变量称为该复合语句中的局部变量，它仅在当前这个复合语句中有效。这是 C 语言的一个重要特征。

第五节　判断选取控制语句

2.5.1 条件语句——if 语句

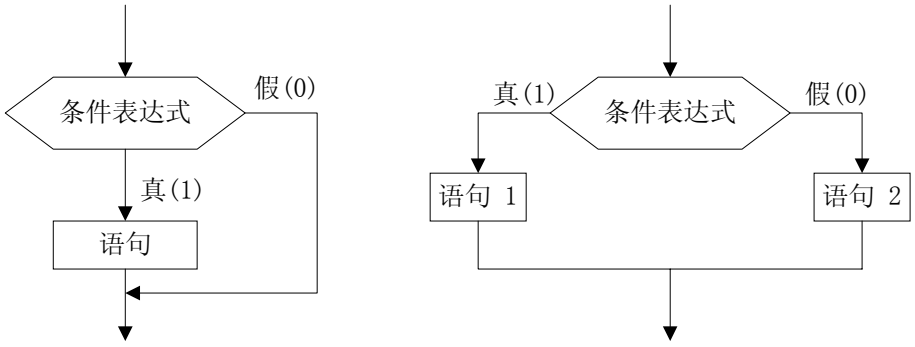
条件语句又称为分支语句，它是用关键字 if 构成的。C 语言提供了三种形式的条件语句：

2.5.1.1 if 语句的简化形式

if 语句的简化形式为：

```
if（条件表达式）　　语句
```

其含义为：若条件表达式的结果为真（1），就执行后面的语句；反之，若条件表达式的结果为假（0），就不执行后面的语句。这里的语句也可以是复合语句，这种条件语句的执行过程如图 2.3（a）所示。



(a) 简化形式　　(b) 基本形式

图 2.3 if 语句执行过程

2.5.1.2 if 语句的基本形式

if 语句的基本形式为：

```
if（条件表达式）　　语句 1
else　　语句 2
```

上述结构表示：如果条件表达式的值为 1（即真），则执行语句 1；反之，如果条件表达式的值为 0（即假），则执行语句 2。这里语句 1 和语句 2 都可以是复合语句。这种条件语句的执行过程如图 2.3（b）所式。

2.5.1.3 if 语句的多分支形式

if 语句的多分支形式为：

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

这种条件语句常用来实现多方向条件分支，其执行过程如图 2.4 所示。

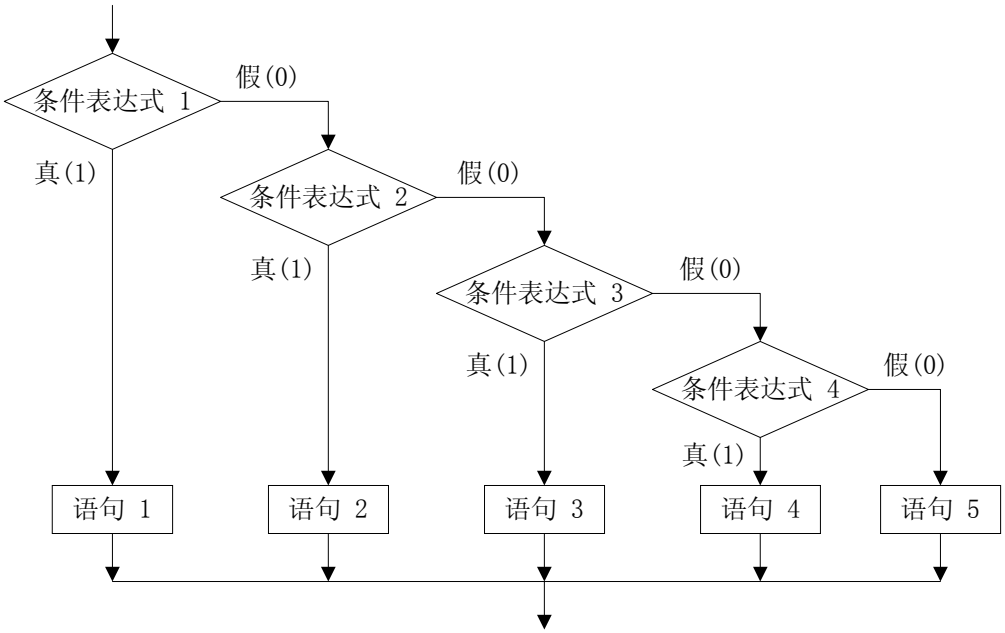


图 2.4 多分支 if 语句执行过程

If 语句允许嵌套，在 if 语句中又含有一个或多个 if 语句。但是需要注意 if 与 else 的对应关系，C 规定 else 总是与它上面最近的一个 if 相对应。所以，最好有良好的书写习惯，使嵌套层次一目了然，便于阅读检查程序。

2.5.2 开关语句——switch 语句

switch 语句是多分支选择语句。相比较 if 语句的多分支形式，switch 语句可以直接处理多分支选择，使程序结构清晰，使用方便。

switch 语句的一般形式如下：

```
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 d
}
```

开关语句的执行过程是：将 switch 后面表达式的值与 case 后面各个常量表达式的值逐个进行比较，若遇到相等时，就执行相应的case后面的语句，再执行break语句跳出switch语句。若无相等的情况则只执行语句 d。开关语句 switch 的执行过程如图 2.5 所示。

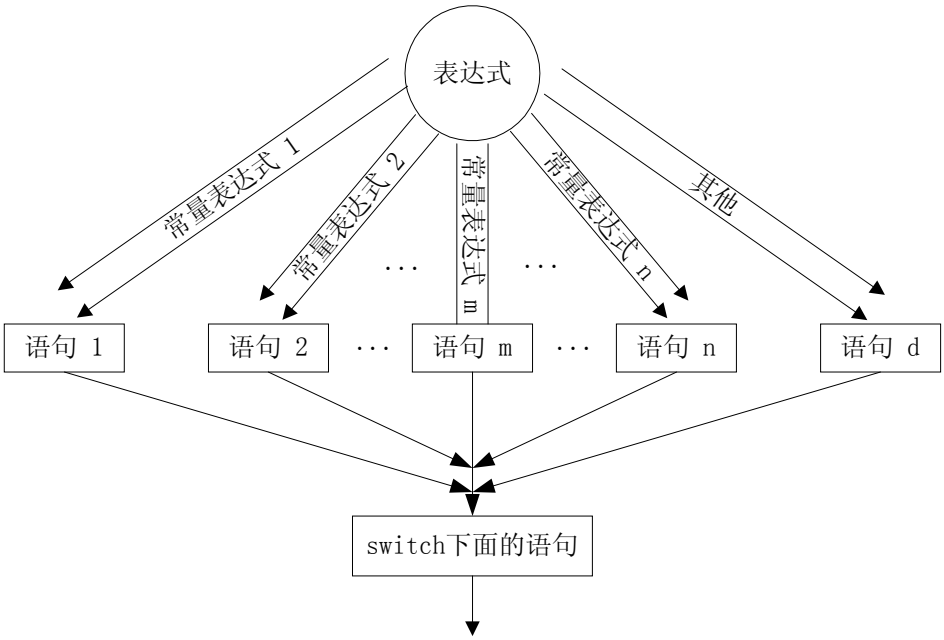


图 2.5 switch 语句的执行过程

break 语句又称间断语句，它的功能是中止当前语句的执行，使程序跳出 switch 语句。如果在 case 语句中遗忘了 break，则程序在执行了本行 case 选择之后，将执行后续的 case 语句而不会按规定的跳出 switch 语句。

第六节　循环语句

在许多实际问题中，需要进行有规律的重复操作，如累加求和，数据块的搬移等等。循环结构是结构化程序的三种基本结构之一，C 语言中用来构成循环控制的语句，分述如下。

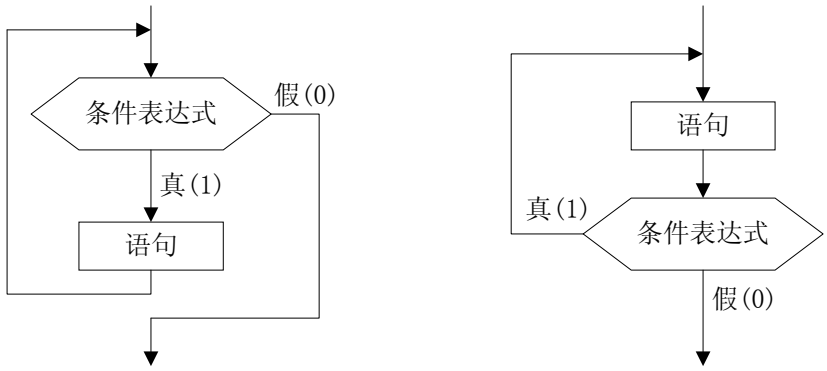
2.6.1 while 语句

while 循环的一般形式为:

```
while(条件表达式)　　语句;
```

while 循环表示当条件表示式的结果为真（1）时，程序便执行后面的语句，直到条件表达式的结果变化为假（0）时才结束循环，并继续执行循环程序外的后续语句。

这种循环结构是先检查后执行，如果条件表达式的结果一开始就为假，则后面的语句一次也不会被执行。While 语句的执行过程如图 2.6（a）所示。



(a) while 语句　　(b) do-while 语句

图 2.6 while 语句和 do-while 语句的执行过程

2.6.2 do——while 语句

do——while 语句构成循环结构的一般形式如下：

```
do　　语句
while　（条件表达式）;
```

do——while 语句先执行循环体中的语句，然后再判断条件表达式的值是否为真。如果为真则重复执行循环体语句，直到条件表达式的值变为假时为止。

这种循环结构的特点是先执行再检查。因此，do——while 语句即使条件表达式的值一开始就为假，循环体语句也会被执行一次。do——while 的执行过程如图 2.6（b）所示。注意到 while 语句和 do——while 语句在执行过程上的不同。

2.6.3 for 语句

for 语句的一般形式如下：

```
for（<初始化表达式>；<循环条件表达式>；<更新表达式>）　　语句
```

初始化表达式一般是一个赋值语句，它用来给循环控制变量赋初值；循环条件表达式决定什么时候退出循环；更新表达式定义循环控制变量每循环一次后按什么方式变化。这三个部分之间用“;”分开。

for 语句的执行过程是：先计算出初始化表达式的值作为循环控制变量的初值，再检查循环条件表达式的结果，当满足循环条件时就执行循环体语句并计算更新表达式，然后再根据更新表达式的计算结果来判断循环条件是否满足……一直进行到循环条件表达式的结果为假时，退出循环体。

C 语言的循环结构中，for 语句的使用最为灵活。for 语句中的三个表达式是相互独立的，并不一定要求三个表达式之间又依赖关系。并且 for 语句中的三个表达式都可能缺省，但无论缺省哪一个表达式，其中的两个分号都不能缺省。最好不要缺省循环条件表达式，以免形成死循环。

for 语句的执行过程如图 2.7 所示。

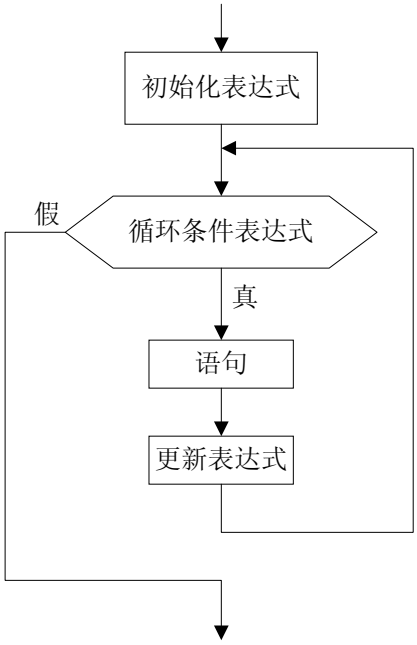


图 2.7 for 语句执行过程

2.6.4 goto 语句

goto 语句是一个无条件转向语句，它的一般形式为：

```
goto　　语句标号；
```

goto 语句中的语句标号是一个带冒号“：”的标识符。将 goto 语句和 if 语句一起使用，可以构成一个循环结构。下面的程序，可以用来累加 1~100 求和：

```
main（）
{
int　a，sum=0；
a=1；
loop：　if（a<=100）
　　{　sum=sum+a；
　　a++；
　　goto　　loop；
　　}
}
```

除此之外，goto 语句还可以在多重循环中从内层循环跳到外层循环。但是结构化程序设计方法主张限制使用 goto 语句，因为滥用 goto 语句会使程序流程无规律，可读性差。因此，在程序设计中，应尽量少用 goto 语句。

2.6.5 continue 语句

continue 语句是一种中断语句，它一般用在循环结构中，其作用是结束本次循环，即跳过循环体中下面尚未执行的语句，把程序流程转移到当前循环语句的下一个循环周期，接着进行下一次循环控制条件的判断。continue 语句的一般形式为：

```
continue；
```

continue 语句只用在 for、while、do-while 等循环体中，它也是一种具有特殊功能的无条件转移语句，但与 break 语句不同，continue 语句并不跳出循环体，而只是结束本次循环。

第七节　返回语句

返回语句用于终止函数的执行，并控制程序返回到调用该函数时所处的位置。返回语句有两种形式：

```
return（表达式）；
return；
```

如果是第一种 return 语句，后边带有表达式，则要计算表达式的值，并将表达式的值作为该函数的返回值。若使用不带表达式的第 2 种形式，则被调用函数返回主调用函数时，函数值不确定。

一个函数的内部可以含有多个 return 语句，但程序仅执行其中的一个 return 语句而返回主调用函数。一个函数的内部也可以没有 return 语句，在这种情况下，当程序执行到最后一个界限符“}”处时，就自动返回主调用函数。

第三章　函数

函数是 C 语言中的一种基本模块。进行程序设计种，如果所设计的程序较大，一般将其分成若干个子程序模块，每个子程序模块完成一种特定的功能。C 语言中，子程序是用函数来实现的，所以实际上 C 语言程序就是由若干个模块化的函数所构成的。

第一节　函数的定义

3.1.1 函数的分类

从用户使用的角度划分，函数有两种：一种是标准库函数，一种是用户自定义函数。标准库函数是由 C 编译系统的函数库提供的，不需要用户定义，可以直接调用。用户自定义函数是用户根据自己的需要编写的函数，它必须先定义之后才能调用。

从函数定义的形式上划分，函数可以分为以下三种形式：

无参数函数：

此种函数在被调用时，既无参数输入，也不返回结果给调用函数。

有参数函数：

在调用此种函数时，必须提供实际的输入参数，此种函数在被调用时，必须说明与实际参数一一对应的形式参数，并在函数结束时返回结果供调用它的函数使用。

空函数：

此种函数体内无语句，是空白的。定义空函数往往是为了以后扩充程序功能。

3.1.2 无参数函数的定义方法

无参数函数的定义形式为：

```
函数类型　　函数名（）
{
局部变量定义
函数体语句
}
```

其中，“函数类型”说明了自定义函数返回值的类型。“函数名”是自定义函数的名字。“局部变量定义”是对在函数内部使用的局部变量进行定义。无参数函数一般不带回返回值，可以省略函数类型。

3.1.3 有参数函数的定义方法

有参数函数的定义形式为：

```
函数类型　　函数名（形式参数列表）
形式参数说明
{
局部变量定义
函数体语句
}
```

“形式参数列表”中列出的是在主调用函数与被调用函数之间传递数据的形式参数，参数之间由逗号隔开。“形式参数说明”是对形式参数类型加以说明。也可以把形式参数说明放在形式参数列表中。例如，

```
int powe（x,n）
int x,n；
```

可以写成 int power（int x，int n）的形式。注意不可写成 int power（int x，n）。

3.1.4 空函数的定义方法

空函数的定义形式为：

```
函数类型　　函数名（）
{　　}
```

3.1.5 函数的返回值

函数的返回值是通过函数中的 return 语句获得的。被调用函数一次只能返回一个变量值，即使有多于一个的 return 语句也必须在选择结构中使用。

函数的返回值的类型是由函数类型指定的。C 语言规定，凡是不加函数类型标识符的函数，都按整型（int）处理。如果函数返回值类型说明和 return 语句中的变量类型不一致，则以函数类型为标准进行强制转换。

为了明确表示被调用函数不带返回值，可以将函数定义为“无类型”——“void”。 C 语言规范书写中，为了保证函数的正确调用，减少出错，凡是没有返回值的函数，函数类型都应该定义为“void”。

第二节　函数的调用

3.2.1 函数的调用形式

所谓函数调用就是在一个函数体中引用另外一个已经定义了的函数，前者称为主调用函数，后者称为被调用函数。主调用函数调用被调用函数的一般形式为：

```
函数名（实际参数列表）
```

实际参数列表必须在个数、类型及顺序上和对应的形式参数列表严格保持一致，以便将实际参数的值正确地传递给形式参数。

在 C 语言中可以采用三种方式完成函数的调用：

1．函数语句。在主调用函数中将函数调用作为一条语句，例如：

fun（）；

这是无参调用，它不要求被调函数返回确定值。

2．函数表达式。在主调函数中函数调用作为一个运算对象直接出现在表达式中，如：

c = max（x，n）+max（y，m）；

这其实是将两个函数调用的返回值相加赋值给 c，所以要求被调函数有返回值。

3．函数参数。在主调函数中将被调用函数作为另一个函数调用的实际参数。例如：

y = max（power（x，n），b）；

其中 power（x，n）是一次函数调用，其返回值作为另一个函数调用 max（）的实际参数之一，再进行另一次函数调用。这种调用方式实际上属于嵌套函数调用。

3.2.2 对被调用函数的说明

与使用变量一样，在调用一个函数之前，必须对该函数的类型进行说明，即“先说明，后调用”。如果调用的是库函数，一般应在程序的开始处用预处理命令#include 将有关函数说明的头文件包含进来，详情见后面章节“预处理”。

如果调用的是用户自定义函数，而且该函数与调用它的主调函数在同一个文件中，一般应该先对被调函数的类型进行说明。函数说明的一般形式为：

```
函数类型标识符　被调用的函数名（形式参数说明列表）；
```

为了便于阅读且风格统一，程序设计中最好将主函数 main（）作为文件的第一个函数，在 main（）函数前对其他被调用的函数进行函数说明，然后在 main（）函数后在进行被调用函数的定义。例如：

```
void power（int x，int n）；　　/* 进行函数类型声明 */
int main()
{
……
}
void power（x，n）
int x，n；
{
……
}
```

3.2.3 函数的参数和传递方式

C 语言采用函数之间的参数传递方式，大大提高了函数的通用性和灵活性。函数之间的参数传递，由函数调用时，主调用函数的实际参数与被调用函数的形式参数之间进行数据传递来实现。

3.2.3.1 形式参数和实际参数

形式参数是定义函数时函数名后面括号中的变量名，简称“形参”。实际参数是函数调用时主调用函数名后面括号中的表达式，简称“实参”。形参和实参的类型必须一致，否则会发生错误。

被调用函数的形式参数在函数没有被调用之前，并不占用实际内存单元，当函数调用发生时，形参才被分配给内存单元。调用结束后，形参所占有的内存则被释放。

3.2.3.2 实际参数的传递方式

对于不同类型的实际参数，有三种不同的参数传递方式：

1．基本类型的实际参数传递

函数的参数是基本类型的变量时，主调函数将实际参数的值传递给被调函数中的形式参数，这种方式称为值传递。值传递方式是将实际参数的值传递到位被调函数的形式参数分配的临时存储单元中。值传递是一种单向传递。

2．数组类型的实际参数传递

函数的参数是数组类型的变量时，主调函数将实际参数数组的起始地址传递到被调函数中形式参数的临时存储单元，这种方式称为地址传递。地址传递是一种双向传递。

3．指针类型的实际参数传递

函数的参数是指针类型的变量时，主调函数将实际参数的地址传递给被调函数中形式参数的临时存储单元，也属于地址传递。数组和指针类型实际参数的传递在下一章中将详细介绍。

3.2.4 函数的嵌套和递归调用

C 语言中函数不能嵌套定义，但可以嵌套调用函数。就是说，在调用一个函数的过程中，允许调用另一个函数。根据单片机片内 RAM 堆栈空间大小，对函数嵌套的深度有限制，一般限制在 4、5 层以内。

如果在调用一个函数的过程又间接或直接地调用该函数本身，这成为函数的递归调用。例如计算阶乘函数 f（n）=n!，可以先计算 f（n-1）=（n-1)!，而计算 f（n-1）时又可以先计算 f（n-2）=（n-2)!，这就是递归算法。C 语言允许函数的递归调用和嵌套。

第三节　中断服务函数

在 C 语言源程序中直接编写单片机的中断服务函数程序，可以减轻采用汇编语言编写中断服务程序的繁琐程度。定义中断服务函数的一般形式为：

```
__interrupt  void     Interrupt      函数名（void）;
#pragma    intvect                   函数名   n
main（）
{
……
```

```
}
__interrupt
void　　函数名(void)
{
    ……
}
```

同其他函数一样，在 main 函数前对中断服务函数做说明，中断服务函数不带参数，无返回值，类型为“void”。#pragma　intvect 是伪指令，是为中断函数生成相应的中断向量表，n 是中断源的相应中断号，有关中断号和中断向量参照硬件手册。在中断函数定义时要先使用关键字__interrupt。注意不能直接调用中断函数，否则会产生编译错误。有关的编译知识参见后面章节。

第四节　　变量的种类和存储

3.4.1 局部变量与全局变量

按照变量的有效作用范围可划分为局部变量和全局变量：

局部变量是在一个函数内部定义的变量，它只在该函数内有效，也就是说只有在本函数内才能引用它们，在此函数以外是不能使用这些变量的。主函数内定义的变量也是局部变量，其它函数不能引用。不同的函数可以使用相同的局部变量名，不会互相干扰。

全局变量是在函数之外定义的变量，又称为外部变量。在全局变量定义点之后的所有函数都可以使用该变量，如果全局变量在一个程序文件的开始处定义，则整个文件中其它函数都可以使用它。

如果想在全局变量的定义点之前引用该变量，或是不在本文件内引用，则要在引用该变量的函数中用关键字“extern”将其说明为“外部变量”。外部变量说明只是声明该变量是一个已在外部定义过的变量，可以在任何需要使用它的函数内作说明，但作说明的同时不能给该变量赋初值。

<注意>

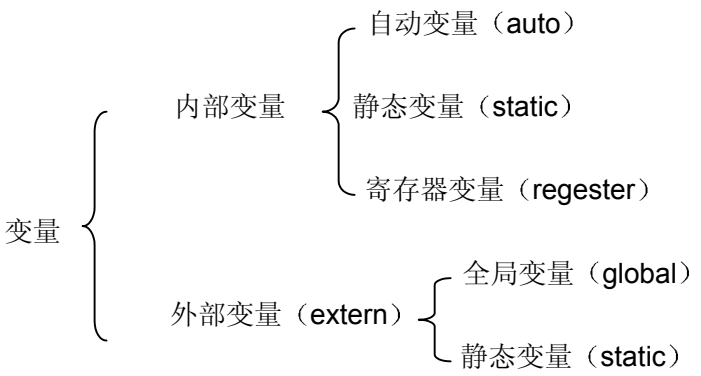
1. 局部变量与外部变量同名时，在局部变量定义的函数内，外部变量不起作用，也不能在该函数内作此外部变量说明，否则会产生编译错误。
2. 全局变量增加了函数间联系的渠道。但不提倡用这种方式，因为它降低了函数的独立性。使得模块间的耦合增加，提高了程序的复杂性。

3.4.2 变量的存储种类

按变量的存储方式划分存储种类，C 语言中有三种变量存储类型如下：

自动（auto）　　静态（static）　　外部（extern）　　寄存器（register）

它们和全局变量，局部变量的关系如下：



3.4.2.1 自动变量(auto)

定义变量时，在变量名前面加上存储种类说明符“auto”，该变量即定义为自动变量。按照缺省规则，如果定义内部变量时省略存储种类说明，该变量即为自动变量。习惯上采用缺省形式，例如：char x；等价于 auto char x；

3.4.2.2 静态变量（static）

使用存储种类说明符“static”定义的变量称为静态变量。根据变量的类型可以分为静态局部变量和静态全局变量。

1. 静态局部变量

静态局部变量与局部变量中的自动变量区别在于：在函数退出时，静态局部变量的值仍然保持，但不能被其它函数使用，当再次进入该函数时，将保存上次的结果。

2. 静态全局变量

静态全局变量是指在函数外部定义，作用范围从定义点开始一直到程序结束的变量。它与全局变量的区别是：全局变量可以再说明为外部变量(extern)，被其它源文件使用，而静态全程变量却不能再被说明为外部的，只能被所在的源文件使用。

3.4.2.3 外部变量（extern）

使用存储种类说明符“extern”定义的变量称为外部变量。按照缺省规则，凡是在所有函数之前，在函数外部定义的变量都是外部变量，定义时可不写 extern 说明符。但是，在一个函数体内说明一个已在该函数体外或别的程序模块文件中定义过的外部变量时，则必须要使用 extern 说明符。

C 语言中常将大型程序分解为若干个独立的程序模块文件，如果莫个变量需要在所有模块文件中使用，只要在一个程序模块文件中将其定义为全局变量，在其他程序模块文件中用 extern 说明该变量为外部变量就可以了，例如：

文件 1 为 file1.c

```
int i, j;              /* 定义全局变量 */
char c;
main（）
{  ……     }
```

文件 2 为 file2.c

```
extern int i, j;          /* 说明变量 i, j 在文件 1 中定义 */
extern char c;            /* 说明变量 c 在文件 1 中定义 */
main（）
{  ……     }
```

外部变量也不能使用过多，否则会增加程序调试排错时的困难，使程序不便维护。

3.4.2.4 寄存器变量（register）

存储类型说明符 register 定义的变量称为寄存器变量。它能够直接使用硬件寄存器，从而提高运算速度。它只能用于整型和字符型变量。

由于硬件寄存器是有限的，寄存器变量的定义受到数目限制。通常在程序中定义寄存器变量时，只是给编译器一个建议，是否能真正成为寄存器变量，要由编译器实际情况来确定。所以不建议使用这种存储类型。

对于以上所介绍的变量类型和变量存储类型将会在以后的学习中，通过例行程序中的定义、使用来逐渐加深理解。

第四章　数组与指针

第一节　数组的定义和引用

数组类型是 C 语言中一种使用广泛的数据类型。数组是一组有序数据的集合，数组中的每一个数据都属于同一个数据类型。C 语言中数组必须先定义，然后才能使用。一维数组的定义形式如下：

数据类型　数组名[常量表达式];

其中，“数据类型”说明了数组中各个元素的类型。“数组名”是整个数组的标识符，它的定名方法与变量的定名方法一样。“常量表达式”说明了该数组的长度，及该数组中的元素个数。常量表达式必须用“[ ]”括起来，而且其中不能含有变量。例如：

char x[5];　是正确的

int y[n];　是错误的

当程序中设定了一个数组时，C 编译器就会在系统的存储空间中开辟一个区域用于存放该数组的内容。数组名表示着数组存储空间的首地址。

定义多维数组时，只要在数组名后面增加相应于维数的常量表达式即可。比如，二维数组的定义形式为：

数组类型　数组名[常量表达式 1][常量表达式 2];

例如要定义一个 10x10 的整数矩阵 A，定义方法如下：

int A[10][10];

需要指出的是，C 语言中数组的下标是从 0 开始的，因此对于数组 char x[5]来说，其中的 5 个元素是 x[0]~x[4]，不存在元素 x[5]，这一点在引用数组时应该注意。另外，C 语言规定，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（字符数组除外）。

第二节　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。字符数组中的每个元素都是一个字符，因此可用字符数组来存放字符串。字符数组的定义方法和一般数组相同。如：char welcome[30];

C 语言中字符串是作为字符数组来处理的。为了测定字符串的实际长度，C 语言规定以‘\0’作为字符串结束标志，对字符串常量也自动加一个‘\0’作为结束符。因此字符数组 char welcome[30]可存储一个长度≤29 的字符串。

在访问字符数组时，遇到‘\0’就表示字符串结束。因此，在定义字符数组时，应使数组长度大于它允许存放的最大字符串的长度。另外，符号‘\0’不是一个可显示字符，而是一个“空操作符”，在这里仅仅起一个结束标志的作用。

对于字符数组的访问可以通过数组中的元素逐个进行访问，也可以对整个数组进行访问。这不同于其他一般数组。

第三节　数组的初始化

数组中各个元素的赋值是在程序运行过程中进行的，可以分别赋值，但对大型数组而言比较繁琐。如果希望在定义数组的同时给数组中各个元素赋以初值，可用如下方法定义：

数据类型　数组名[常量表达式]={常量表达式表};

其中，“数据类型”是指数组元素的数据类型；“常量表达式表”中给出各个数组元素的初值。例如：

char x[5]={0x11，0x22，0x33，0x44，0x55};

需要注意的是，在定义数组的同时对数组元素赋初值时，初值的个数必须小于或等于数组中元素的个数（即数组的长度），否则在程序编译时出错。如果常量表达式表中的数据个数比数组的元素个数少，则不足的数组元素自动赋 0。数组名后面的“常量表达式”可省略，此时数组的长度由实际初值的个数决定。例如：

char y[]={4,56,43,3,7,88,46,52,32,14};　数组长度为 10。

对于多维数组可以采用同样的方法来赋值，例如可以这样定义并赋初值：

int A[3][4]= {{8, 2, 5,0} ,{27, 5, 19, 3}, {1, 8, 14, 2}};

第四节　数组作为函数的参数

前面说过数组类型的变量可以作为函数的参数，实际上，是用数组名作为函数的参数。一个数组的数组名表示该数组的首地址。进行函数调用时，实际参数数组的首地址传递给被调函数中的形式参数数组，这样一来两个数组就占用同一段内存单元。

用数组名作为函数的参数，应该在主调函数和被调函数中分别进行数组定义，当然两个函数中定义的数组类型必须一致。实参数组和形参数组的长度可以一致也可以不一致，编译器对形参数组的长度不做检查。定义形参数组时可以不指定长度，只在数组名后面跟一个空的方括号[ ]。

例如，编写一个程序，找出一个含有 10 个整数元素的数组的最小值：

```
int min(int values[10]) ;
main()
{
int   minimum_score ;
int   scores[10]={6,5,7,9,1,4,2,8,0,3};
minimum_score = min(scores);
}
int min(values)
int values[10];
{
int minimum, j ;
for(j=1;j<10;j++)
    if(values[j]<minimum)
    minimum = values[j];
    return(minimum);
}
```

用数组名作为函数的参数，参数的传递过程采用的是地址传递。地址传递方式具有双向传递的性质，即形式参数的变化将导致实际参数也发生变化。

多维数组作为函数的参数与一维数组的情形类似，不过形式参数数组的长度说明时，要注意：可以省略第一维的长度说明，不能省略其他高维的长度说明。因为在内存中数组时按行存放的，并不区分行和列。不说明列数，编译器无法确定几行几列。

第五节　指针的概念

指针，一直被认为是 C 语言中的精华。指针、地址、数组及其相互关系是 C 语言中最有特色的部分。规范地使用指针，可以编写有效、正确和灵活的程序。

4.5.1 指针变量的基本概念

我们知道，计算机执行任何一个程序时都要涉及到许多的寻址操作，所谓寻址，就是按照内存单元的地址来访问该存储单元中的内容。一个地址指向一个存储单元。C 语言中为了能够实现直接对内存单元进行操作，引入了指针类型的数据。

指针类型数据是专门用来确定其他类型数据地址的，因此一个变量的地址就称为该变量的指针。如果有一个变量专门用来存放另一个变量的地址，则称之为“指针变量”。C 语言中用符号“*”来表示“指向”，称为指针运算符。它和指针变量联用，其作用时表示指针变量指向的变量。符号“&”是取地址运算符，它可以与一个变量联用，其作用是求取该变量的地址。如果变量 a 的存放地址是 40H，而：

```
p = &a;
```

是取得变量 a 的地址并赋给指针变量 p，即指针变量 p 指向了变量 a。再看：

```
a = 50H;
*p = 50H;
```

这两句就都是给同一个变量赋值 50H。图 4.1 就直观的反映了 p 和 a 之间的关系。

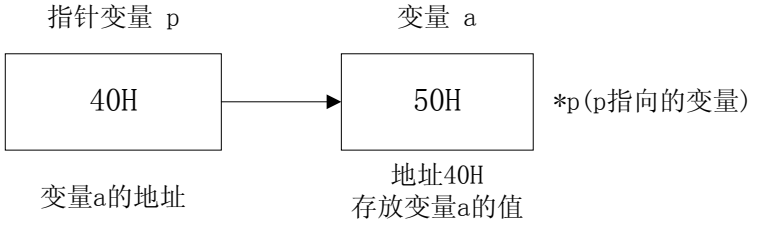


图 4.1 指针变量和它所指向的变量

从图可见，对于同一个变量 a，可以通过变量名 a 来访问它，也可以通过指向它的指针变量 p，用*p 来访问它。前者称为直接访问，后者称为间接访问。

4.5.2 指针变量的定义

指针变量的定义与一般变量的定义类似，其一般形式如下：

```
数据类型　　* 标识符;
```

其中，“标识符”是所定义的指针变量名。“数据类型”说明了该指针变量所指向的变量的类型。例如：int　*p；

标识符前加了“*”号，表示该变量是指针变量。一个指针变量只能指向同一种类型的变量。上面的定义，p 只能指向 int 类型的变量。

指针变量在定义中允许带初始化项。如：

```
int x, *ip = &x ；
```

<注意>

这里是用&x 对 ip 初始化,而不是对*ip 初始化。和一般变量一样，对于外部或静态指针变量在定义中若不带初始化项，指针变量被初始化为 NULL，它的值为 0。当指针值为零时，指针不指向任何有效数据，有时也称指针为空指针。

4.5.3 指针变量的引用

指针变量是含有一个数据对象地址的特殊变量，指针变量中只能存放地址。因此，在使用中不要将一个整数赋给一指针变量。下面的赋值是不合法的：

```
int　*ip；
ip=100；
```

指针变量经过定义之后可以像其他基本类型变量一样引用。例如：

变量定义

```
int x，y，*px，*py；
```

指针赋值

```
px = &x；     /* px  指向 x */
py = &y；     /* py  指向  y */
```

指针变量引用

```
*px = 0；     /*  等价于 x=0；  */
*py = 1；     /*  等价于 y=0；  */
*px += 1；    /*  等价于 x +=1；  */
(*px)++；     /*  等价于 x ++；  */
```

指向相同类型数据的指针之间可以相互赋值。例如：px = py；原来指针 px 指向 x，py 指向 y，经上述赋值之后，px 和 py 都指向 y。

4.5.4 指针的地址运算

前面已经讲过，指针的赋初值可以是 NULL（零），也可以是变量、数组以及函数等的地址。除此之外，指针还能做下面这些运算：

1．指针与整数的加减

指针可以与一个整数或整数表达式进行加减运算，从而获得该指针当前所指位置前面或后面某个数据的地址。假设 p 为一个指针变量，n 为一个整数，则 p+n 表示离开指针 p 当前位置的前面第 n 个数据的地址。

2．指针与指针相减

指针与指针相减的结果为一整数值，但它并不是地址，而是表示两个指针之间的距离或元素的个数。注意，这两个指针必须指向同一类型的数据。

3．指针与指针的比较

指向同一类型数据的两个指针可进行比较运算，从而获得两指针所指地址的大小关系。

需要指出的是，指针的运算是很有限的，它只能进行如上所述的运算操作，其他的指针运算都是非法的。特别是，不允许对两个指针进行加、乘、除、移位或屏蔽运算，也不允许用 float 类型数据与指针作加减运算。此外，在使用指针间接取值运算时，应该注意运算符的优先级和结合规则。

4.5.5 指针变量作为函数的参数

前面函数章节里说过，函数的参数可以是指针类型的数据。指针变量作为函数的参数时，它的作用是将一个变量的地址传送到另一个函数中去，地址传递是双向的，即主调用函数不仅可以向被调用函数传递参数，而且还可以从被调用函数返回其结果。可以看到，这和数组名做函数参数时类似。下面来看个例子：

```
void swap(int *a , int *b);
main()
{
    int x, y, *px, *py ;
    x=3;
    y=4;
    px = &x;
    py = &y;
    swap(px , py);
}
void swap(int *a , int *b)
{
int c;
c = *a;
*a = *b;
*b = c;
}
```

这里 swap 函数是将两个指针变量指向的变量值交换一下。定义形参时要将其定义为指针变量，这样才能接收地址类型的数据值。函数调用时，传递的是 x，y 两变量的地址，通过指针间接存取改变了 px，py 指向的变量的值，即 x，y 变量的值。

4.5.6 用指针引用数组元素

在 C 语言中指针与数组有着十分密切的关系，任何能够用数组实现的运算都可以通过指针来完成。例如定义一个具有十个元素的整型数组可以写成：

```
int   a[10];
```

数组 a 中各个元素分别为 a[0]、a[1]…a[9]。数组名 a 表示数组的首地址，即 a[0]的地址。其实数组名也就是一个指针。而*a 则表示 a 所代表的地址中的内容，即 a[0]。

如果定义一个指向整型变量的指针 pa 并赋以数组 a 中的一个元素 a[0]的地址：

```
int   *pa;      pa = &a[0];
```

则可以通过指针 pa 来操作数组 a。即可用*pa 代表 a[0]，*(pa+i)代表 a[I]等，也可以使用 pa[0]、pa[1]…pa[9]的形式。

任何一个数组及其数组元素都可以用一个指针及其偏移值来表示，但要注意的是：指针是一个变量，而数组名是一个常量，不能像变量那样进行运算，即数组的地址是不能改变的。例如上面的例子中：pa=a；则是将数组 a 的首地址，即指向数组 a 的指针赋给指针变量 pa。如果对数组名 a 进行如下操作：

a=pa；　a++；

就都是错误的。下面再通过一个例子来看看数组和指针的相互关系：

```
int fun(char*);
main()
{
int a;
char str[]="abcdefghijklmn";
a=fun(str);
...
}
int fun(char*s)
{
int i=0;
int num=0;
for(i=0;i<10;)
{ num+= *s; s++; }
return num;
}
```

这个例子中的函数 fun 统计一个字符串中前 10 个字符的 ASCII 码值之和。前面说了，数组名也是一个指针。在函数调用中，当把数组名 str 作为实参传递给形参 s 后，实际是把 str 的值传递给了指针变量 s。指针 s 所指向的地址就和数组名 str 所指向的地址一致，也是指向数组 str 的首地址。但是 str 和 s 各自占用各自的存储空间，在函数体内对指针 s 进行自加 1 运算，并不意味着同时对 str 进行了自加 1 运算。 但如果对*s 进行运算，数组 str[] 的元素就会发生改变。

第六节　函数型指针和指针函数

4.6.1 函数型指针

如果将函数的入口地址赋给一个指针，该指针就是函数型指针。由于函数型指针指向的是函数的入口地址，因此可用指向函数的指针代替函数名来调用该函数。C 语言中函数与变量不同，函数名不能作为参数直接传递给另一个函数。但是利用函数型指针，可以将函数作为参数传递给另一个函数。

定义一个函数型指针的一般形式为：

```
数据类型（*标识符）();
```

其中，“标识符”就是所定义的函数型指针变量名。“数据类型”说明了该指针所指向的函数类型，也就是函数返回值的类型。例如：

```
int（*func1）();
```

定义了一个函数型指针变量 func1，它所指向的函数返回整型数据。函数型指针变量是专门用来存放函数入口地址的，程序中可以对一个函数型指针多次赋值，该指针可以先后指向不同的函数。

给函数型指针赋值的一般形式为：

函数型指针变量名 = 函数名

如果有一个函数 max（x，y），则可用如下的赋值语句将该函数的地址赋给函数型指针 func1，使 func1 指向函数 max；

func1 = max;

引入了函数型指针之后，对于函数的调用可以采用两种方法。例如程序中要求将函数 max（x，y）的值赋给变量 z，可采用如下方法：

z = max（x，y）；或 z =（*func1）（x，y）；

这两种方法实现函数调用的结果是完全一样的。需要注意的是，若采用函数型指针来调用函数，必须预先对该函数指针进行定义并赋值，使之指向所需调用的函数。

4.6.2 指针函数

在函数的调用过程结束时，被调用的函数可以带回一个整型数据、字符型数据等，也可以带回一个指针型数据，即地址。这种返回指针型数据的函数就称为指针函数，它的一般定义形式为：

数据类型　　*函数名（参数表）；

其中，“数据类型”说明了所定义的指针函数在返回时带回的指针所指向的数据类型。例如：

int　*x（a，b）；

定义了一个指针函数 *x，调用它以后可以得到一个指向整型数据的指针，即地址。可以看到，在指针函数 *x 的两侧没有加括号（），这是与函数型指针完全不同的，注意区别。

第七节　指针数组和指针型指针

4.7.1 指针数组

由于指针本身也是一个变量，因此 C 语言允许定义指针数组。指针数组的定义方法与普通数组完全相同，一般格式为：

数据类型　　*数组名[数组长度]

例如：

```
int   *x[2];          /* 指向整型数据的 2 个指针 */
char  *sptr[5];       /* 指向字符型数据的 5 个指针 */
```

指针数组在使用之前需要先赋初值，方法也和一般数组赋初值类似。

指针数组适合于用来指向若干个字符串，使字符串的处理更为方便。现在来看看下面的例子：

```
char  *season[4] = { “spring” , ”summer” , ”fall” , ”winter” };
```

上例中，season 是一个四单元的数组，该数组的每个单元都是一个指针，这些指针各指向一个字符串。

前面说过，C 语言对字符串常量是按字符数组处理的。所以，对于数组中的各个元素而言，它们都是字符指针变量，分别对应相应字符串的首地址。

season[0]是个指针，它指向的类型是 char*，它的值是数组的第 0 号单元——字符串"spring"的首地址；season[1]也是一个指针，它的值是数组的第 1 号单元——字符串“summer”的首地址，它指向的类型是 char *。如此等等。

指针数组和所赋初值的关系如图 4.2。

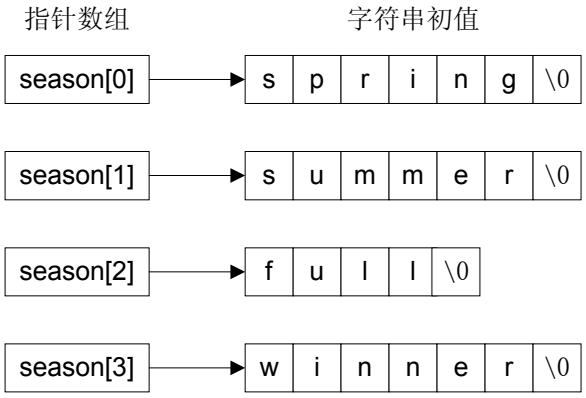


图 4.2 指针数组和所赋初值关系

看起来指针数组和二维数组有些类似，但在内存中实际存储时，指针数组是各行首尾相接的连续区域，各列的长度可以不一致，这样就比二维数组更为有效地利用内存空间，如图 4.3。

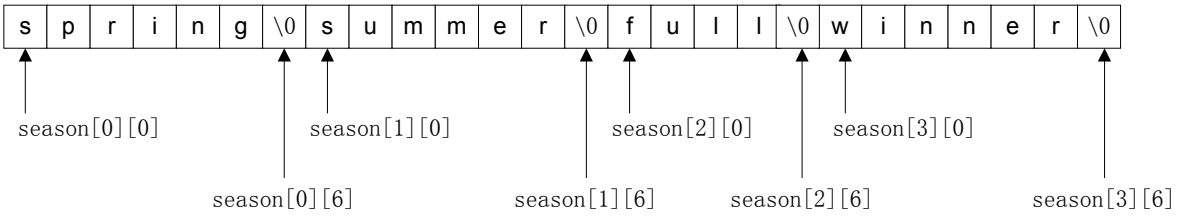


图 4.3 指针数组的初值在内存中的存放格式

season[0] 是一个指针，它的类型是 char，它所指向的类型是 char，它指向数组的第 0 号单元"spring"的第一个字符's'，即 season[0][0]；*season[1]也是一个指针，它的类型是 char*，它所指向的类型是 char，它指向"summer"的第一个字符's'，即 season[1][0]。

4.7.2 指针型指针

指针型指针又称为多级指针，它指向的是另一个指针变量的地址。在前面的例子中，如果定义一个指向指针数组 season 中元素的指针变量，这就是指向指针型数据的指针变量，即指针型指针变量。

定义一个指针型指针变量的一般形式为：

```
数据类型　　** 标识符;
```

其中，“标识符”就是所定义的指针型指针变量名，而“数据类型”则说明一个被指针型指针所指向的指针变量所指向的变量数据类型。如果在前面的例子里定义一个指针型指针 j，并给它赋值：

```
char　　**j;
j = season;
```

可以看到，*j 就是 season[0]的值，即字符串“sping”的首地址。j 是个指针型指针，它指向的类型是 char*，它的值是数组的第 0 号单元——字符串”spring”的首地址。**j 就是 *season[0]，它指向的类型是 char，指向字符串“sping”的第一个字符‘s’。

第五章　结构和联合

在前面介绍数据类型时，曾提到结构和联合数据类型。结构和联合类型以及前面的数组类型都是构造类型的数据。

第一节　结构的概念

结构是一种构造类型的数据，它是将若干个不同类型的数据变量有序地组合在一起而形成的一种数据的集合体。组成该集合体的各个数据变量称为结构成员，整个集合体使用一个单独的结构变量名。

5.1.1 结构变量的定义

结构也是一种数据类型，因此，象其它类型的变量一样，在使用结构变量时要先对其定义。定义结构变量的方法有三种，分别如下：

1．先定义结构类型在定义结构变量名

定义结构类型的一般格式为：

```
struct 结构名
{
类型　变量名;
类型　变量名;
……
};
```

其中，“结构名”是结构的标识符不是变量名。{ }里面的是“结构成员说明表”，是该结构中的各个成员，每个成员都要进行类型说明。类型为五种数据类型之一(整型、浮点型、字符型、指针型和无值型)。

定义好一个结构类型之后，就可以用它来定义结构变量。一般格式为：

```
struct　结构名　结构变量名 1，结构变量名 2，… 结构变量名 n;
```

例如：

```
struct  date
    {
    int       year;
    char      month，day;
    }
struct   data     d1，d2;
```

定义一个日期结构类型 data，它有三个结构成员 year、month、day 组成，然后定义 d1，d2 位具有 struct　data 类型的结构变量。这样 d1，d2 都是由一个整型数据和两个字符型数据所组成的。

2．在定义结构类型的同时定义结构变量名

这种方法是将方法 1 的两个步骤和在一起，一般格式为：

```
struct    结构名
{
结构成员说明表
} 结构变量名 1，结构变量名 2，…结构变量名 n；
```

例如，上述日期结构变量可以这样定义：

```
struct    data
{
int       year；
char      month，day；
} d1，d2；
```

3．直接定义结构变量

这样方法可以省略结构名，又称为无名结构，一般格式为：

```
struct
{
结构成员说明表
} 结构变量名 1，结构变量名 2，…结构变量名 n；
```

例如上述日期结构变量可按如下格式定义：

```
struct
{
int       year；
char      month，day；
} d1，d2；
```

第三种方法和第二种方法比较，省略了结构名，这种方法适合于只需要定义几个确定的结构变量而不打算再定义任何别的结构变量的场合。

结构类型与结构变量使两个不同的概念。定义一个结构类型时只是给出了该结构的组织形式，并没有给出具体的组织成员。因此结构名不占用任何存储空间，也不能对一个结构名进行赋值、存取和运算。而结构变量则是一个结构中的具体组织成员，对结构变量名可以进行赋值、存取和运算。

结构可以嵌套，结构成员可以是另一个结构类型的结构变量，但是这种嵌套不能包含其自身，即不能自己定义自己。

5.1.2 结构变量的引用

定义了一个结构变量之后，就可以对它进行引用。即结构变量也可以象其它类型的变量一样赋值、运算，不同的是结构变量以成员作为基本变量，结构变量的引用是通过对其结构成员的引用来实现的。引用结构元素的一般格式为：

```
结构变量名.结构元素
```

其中“.”是存取结构元素的成员运算符。例如：d1.year 表示结构变量 d1 中的成员 year，d2.day 表示结构变量 d2 中的成员 day 等。如果出现结构的嵌套，就要采用若干个成员运

算符，一级一级地对最低级结构成员进行访问。

5.1.3 结构变量的初值

结构变量有三种存储种类，它们是 extern、static 和 auto。这在前面已经分别介绍过了。当结构变量为外部全局变量或静态变量时，可以在定义结构类型时给它赋初值。例如：

```
main( )
{
    struct    data
    {
    int       year;
    char      month，day;
    } d1={ 2001，6，1};
    ……
}
```

对于自动存储种类的动态局部结构变量，不能在定义时赋初值，只能在程序执行中用赋值语句给各个结构元素分别赋值。

5.1.4 结构数组

在实际使用中，结构变量往往不止一个，通常是将多个相同的结构组成一个数组，这就是结构数组。结构数组的定义方法与结构变量一致，赋初值的方法和数组类似，例如：

```
struct    data
{
int       year;
char      month，day;
};
struct   data   d[3] = { { 1990，1，1}，{1999，2，1}，（2001，6，1）};
```

第二节　结构型指针

5.2.1 结构型指针的概念

结构型指针是指向结构类型变量的指针变量，指针变量的值是它所指向的结构变量的起始地址。结构型指针也可用来指向结构数组，或指向结构数组中的成员。

定义结构型指针的一般格式为：

```
struct    结构类型标识符   *结构指针标识符
```

其中“结构指针标识符”就是所定义的结构型指针变量的名字，“结构类型标识符”就是该指针所指向的结构变量的具体类型名字。例如：

```
struct    data      *dp;
```

这里的 dp 即可用来指向 data 类型的结构变量或结构数组。与一般指针一样，结构型指针

也必须先赋值后才能引用。

5.2.2 使用结构型指针访问结构成员

使用结构型指针对结构成员的访问，与结构变量对结构成员的访问在表达方式上有所不同。通过结构型指针来引用结构成员的一般格式为：

结构指针->结构成员

其中，符号“->”是两个符号“-”和“>”的组合。例如：

dp->year 完全等效于 (*dp).year。

结构指针也可以进行运算，运算规则与普通指针相同。

第三节　结构与函数

结构即可作为函数的参数，也可作为函数的返回值。当结构被用做函数的参数时，其用法和普通变量作为参数是一样的，其参数传递属于“值传递”方式。

当结构较大时，可以用结构型指针来作为函数的参数，此时参数的传递是按地址传递方式进行的。这样做的优点是可以节省存储空间，加快程序的执行速度，但是缺点是调用函数时对结构指针所做的变动会影响到原来的结构变量。

第四节　联合

联合也是 C 语言中一种构造类型的数据结构。联合可以将多个不同类型的数据元素包含在一起，放在同一个地址开始的内存单元中。联合是不同的变量分时是用同一个内存空间，提高了内存的利用效率。

5.4.1 联合的定义

联合类型变量的定义方法一般为：

```
union    联合类型名
{ 成员表列 }变量表列;
```

例如：

```
union    data
{
    float    a;
    int      b;
    char     c;
}x,y,x;
```

也可以将类型定义与变量定义分开，即先定义一个 union data 类型，在将 x，y，z 定义为 union data 类型的变量；当然还可以省略联合类型直接定义联合变量。

由此可见，联合类型与结构类型的定义方法很相似，也是三种。但是在内存的分配上，二者却有本质的区别。结构变量所占用的内存长度是其中各个成员所占用内存长度的总和；而联合变量所占用的内存长度是其中最长的成员的长度。

上面的例子里，float 型数据需要 4 个内存单元，因此系统按 4 个字节长度为联合变量 x、y、z 分配内存空间，在不同时刻只能保存一个变量，对一个变量进行操作。

5.4.2 联合变量的引用

与结构变量类似，对联合变量的引用也是通过对其联合成员的引用来实现的。引用联合成员的一般格式为：

联合变量名.联合成员　　或　　联合变量名->联合成员

例如，对于前面定义的联合变量 x、y、z，下面的引用方法都是正确的：

```
x.a     /* 引用联合变量 x 中的 float 型成员 a */
y.b     /* 引用联合变量 y 中的 int 型成员 b */
```

在引用联合成员时，要注意联合变量用法的一致性。在表达式中对它进行处理时，必须注意其类型与表达式所要求的类型保持一致，否则将导致程序运行出错。不能只引用联合变量，否则系统难以确定应该输出哪一个联合成员的值。

因为联合类型的数据可以采用同一个内存段来存放几种不同类型成员的值，但是在每一瞬间只能存放其中一种类型的元素，也就是说，每一瞬间只有一个成员在起作用。所以所读出变量的值是最近放入的某一个元素的值，因此在引用联合变量时一定要注意当前联合变量中存放的究竟是哪一个成员。

不能直接对联合变量进行赋值，也不能在定义时对它进行初始化。另外，联合变量不能作为函数的参数。

第五节　位段

C 语言是为描述系统而设计的，因此它具有汇编语言所能完成的一些功能，比如位操作。前面介绍过 C 语言中的一些位运算符。现在要介绍一个 C 语言中很有用的概念：位段。

5.5.1 位段的概念

所谓位段，就是指以位为单位定义长度的结构体类型中的成员。例如：

```
struct  packed-data
    {
    unsigned char   a       :2;
    unsigned char   b       :6;
    unsigned char   c       :4;
    unsigned char   d       :4;
    int             x;
    } data ;
```

定义了一个结构变量 data，其中的成员位段 a、b、c、d 分别占 2 位、6 位、4 位、4 位，成员 x 为整型，共占 4 个字节。

通过位段的定义，就可以单独对一个字节的某一位或某一段进行操作。这对单片机来说，是非常有用的，特别是当我们设置寄存器时。来看看下面的定义。

```
union   io_pdr0
```

```
{
    unsigned char    byte;
    struct {
        unsigned char    P00:1;
        unsigned char    P01:1;
        unsigned char    P02:1;
        unsigned char    P03:1;
        unsigned char    P04:1;
        unsigned char    P05:1;
        unsigned char    P06:1;
        unsigned char    P07:1;
        } bit;
}P0;
```

这是一个关于 P0 口端口寄存器联合类型的定义，包括一个无符号型字符变量 byte 和一个结构变量 bit，结构变量中使用了位段的定义。从中可以看到，既可以对 P0 口写入一个字节，也可以对某一位单独赋值。例如：

```
P0.byte =0x80;
P0.bit.P00= 1;
```

都是正确的操作。

5.5.2 有关位段的说明

1．位段在存储单元中的空间分配方向为从右到左，如图所示：

	P07	P06	P05	P04	P03	P02	P01	P00	
高位	1	1	1	1	1	1	1	1	低位

图 5.1 位段空间分配图

2．可以定义无名位段。如：

```
struct
{
unsigned char    a:1;
unsigned char     :2;  (这两位空间不用)
unsigned char    b:3;
unsigned char    c:2;
}x;
```

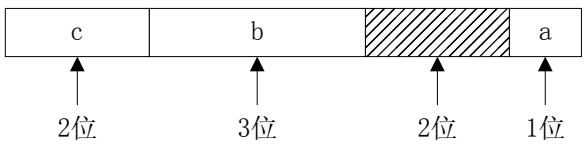


图 5.2 无名位段

如图所示，a 和 b 之间的是无名位段，占 2 位，这两位的空间不用。

3．位段的长度不能存储单元的长度，也不能定义位段数组。

4．要注意位段允许的最大值范围。

如果定义位段占 2 位，那么最大值为 3，超过这个范围的赋值，自动取赋值数的低位。例如：

```
struct   packed-data
    {
    unsigned char     a         :2;
    unsigned char     b         :6;
    } data ;
```

引用位段中的数据时，例如：

```
data.a = 2;
```

是正确的；

```
data.a = 8;
```

是错误的。因为 a 只占 2 位，最大值为 3。在此情况下，自动取 8 的低位。8 的二进制数形式为 1000，a 只有 2 位，所以取 1000 的低 2 位，即得 00，那么 data.a 的值为 0。

附录

附录 1 C 中的关键字

关键字	用途	说明
auto	存储种类说明	用以说明局部变址，缺省值为此
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	数据类型说明	单字节整型数或字符型数据
const	存储类型说明	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	数据类型说明	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
floaat	数据类型说明	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	数据类型说明	基本整型数
long	数据类型说明	长整型数
register	存储种类说明	使用 CPU 内部寄存器的变量
return	程序语句	函数返回
short	数据类型说明	短整型数
signed	数据类型说明	有符号数，二进制数据的最高位为符号位
sizeof	运算符	计算表达式或数据类型的字节数
static	存储种类说明	静态变量
struct	数据类型说明	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	重新进行数据类型定义
union	数据类型说明	联合类型数据
unsigned	数据类型说明	无符号数据
void	数据类型说明	无类型数据
volatile	数据类型说明	说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

附录 2　运算符的优先级和结合性

优先级	运算符	含义	要求运算对象的个数	结合方向
1	**()** **[]** **->** **.**	圆括号 下标运算符 指向结构体成员运算符 结构体成员运算符		自左至右
2	**!** **~** **++** **--** **-** **(类型)** ***** **&** **sizeof**	逻辑非运算符 按位取反运算符 自增运算符 自减运算符 负号运算符 类型转换运算符 指针运算符 地址与运算符 长度运算符	1 （单目运算符）	自右至左
3	***** **/** **%**	乘法运算符 除法运算符 求余运算符	2 （双目运算符）	自左至右
4	**+** **-**	加法运算符 减法运算符	2 （双目运算符）	自左至右
5	**<<** **>>**	左移运算符 右移运算符	2 （双目运算符）	自左至右
6	**<　<=　>　>=**	关系运算符	2 （双目运算符）	自左至右
7	**==** **!=**	等于运算符 不等于运算符	2 （双目运算符）	自左至右
8	**&**	按位与运算符	2 （双目运算符）	自左至右
9	**^**	按位异或运算符	2 （双目运算符）	自左至右
10	**|**	按位或运算符	2 （双目运算符）	自左至右
11	**&&**	逻辑与运算符	2 （双目运算符）	自左至右
12	**||**	逻辑或运算符	2 （双目运算符）	自左至右
13	**?　:**	条件运算符	3 （三目运算符)	自右至左
14	**=　+=　-=　*=** **/=　>>=　<<=** **%= &=　^=　|=**	赋值运算符	2	自右至左
15	**,**	逗号运算符 （顺序求值运算符）	-	自左至右

上表的优先级别由上到下递减。