

第一章

C 语言概述

【基本要求】

通过本单元的学习,使学生概要了解 c 语言,初步了解 c 语言的结构,了解数据结构和算法的概念,了解 c 程序的上机操作步骤,为学好以后各章的学习打好理论基础。

【教学重点】

C 语言的产生、发展与支持环境, C 语言的主要特点, C 源程序的编辑、编译、连接、执行(重点), C 源程序和 C 函数的组成(重点), Turbo C 的上机操作步骤(重点)

【本章结构】

- 1、C 语言出现的历史背景
- 2、C 语言的特点
- 3、简单的 C 语言程序介绍

- 4、运行 C 程序的步骤与方法

{ 运行 C 程序的步骤和方法
上机运行 C 程序的方法

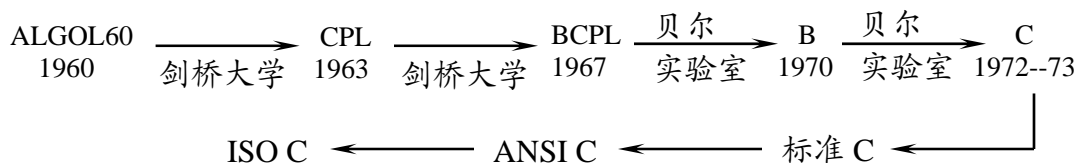
一、 C 语言出现的历史背景

1、C 语言概述

C 语言是国际上流行的、很有发展前途的计算机高级语言。C 语言适合于作为“系统描述语言”。它既可以用来编写系统软件,也可以用来编写应用程序。

以前操作系统等系统软件主要采用汇编语言编写。汇编语言依赖于计算机硬件,程序的可读性、可移植性都比较差。为了提高可读性和可移植性,人们希望采用高级语言编写这些软件,但是一般的高级语言难以实现汇编语言的某些操作,特别是针对硬件的一些操作(如:内存地址的读写-直接硬件、二进制位的操作)。人们设法寻找一种既具有一般高级语言特性,又具有低级语言特性的语言,C 语言就在这种情况下应运而生。

2、C 语言的发展



- **ALGOL60**: 一种面向问题的高级语言。ALGOL60 离硬件较远, 不适合编写系统程序。
- **CPL** (Combined Programming language, 组合编程语言): CPL 是一种在 ALGOL60 基础上更接近硬件的一种语言。CPL 规模大, 实现困难。
- **BCPL** (Basic Combined Programming language, 基本的组合编程语言): BCPL 是对 CPL 进行简化后的一种语言。
- **B 语言**: 是对 BCPL 进一步简化所得到的的一种很简单接近硬件的语言。B 语言取 BCPL 语言的第一个字母。B 语言精练、接近硬件, 但过于简单, 数据无类型。B 语言诞生后, Unix 开始用 B 语言改写。
- **C 语言**: 是在 B 语言基础上增加数据类型而设计出的一种语言。C 语言取 BCPL 的第二个字母。C 语言诞生后, Unix 很快用 C 语言改写, 并被移植到其它计算机系统。
- **标准 C、ANSI C、ISO C**: C 语言标准化。

注: 最初 Unix 操作系统是采用汇编语言编写的, B 语言版本的 Unix 是第一个用高级语言编写的 Unix。在 C 语言诞生后, Unix 很快用 C 语言改写, C 语言良好的可移植性很快使 Unix 从 PDP 计算机移植到其它计算机平台, 随着 Unix 的广泛应用, C 语言也得到推广。从此 C 语言和 Unix 像一对孪生兄弟, 在发展中相辅相成, Unix 和 C 很快风靡全球。

从 C 语言的发展历史可以看出, C 语言是一种既具有一般高级语言特性 (ALGOL60 带来的高级语言特性), 又具有低级语言特性 (BCPL 带来的接近硬件的低级语言特性) 的程序设计语言。C 语言从一开始就是用于编写大型、复杂系统软件的, 当然 C 语言也可以用来编写一般的应用程序。也就是说: C 语言是程序员的语言!

3、C 语言的版本

IBM PC 微机 DOS、Windows 平台上常见的 C 语言版本有:

- **Borland 公司**: Turbo C, Turbo C++, Borland C++, C++ Builder (Windows 版本)
- **Microsoft 公司**: Microsoft C, Visual C++ (Windows 版本)

二、C 语言的特点

C 语言是从“组合编程语言”CPL 发展而来，C 语言既具有一般高级语言特性 (ALGOL60 带来的高级语言特性)，又具有低级语言特性 (BCPL 带来的接近硬件的低级语言特性)。C 语言具有下面特点（其中 1-6 属于高级语言特性，7，8 属于低级语言特性）

(1) C 语言的语言成分简洁，紧凑，书写形式自由

例：将 C 语言程序段与实现同样功能的 PASCAL 语言程序段进行比较。

表 1.1 C 语言与 PASCAL 语言的比较

	C 语言	PASCAL 语言	含义	说明
1	{...}	BEGIN...END	复合语句 (或:语句块)	PASCAL 显得罗嗦
2	if(e)S;	IF(e)THEN S;	条件语句	PASCAL 至少多了一个 THEN 关键词
3	int i;	VAR i:INTEGER	定义 i 为整型变量	PASCAL 至少多了一个 VAR 关键词
4	int a[10];	VAR a:ARRAY[1..10] OF INTEGER	定义 a 为整型一维数组,10 个元素	PASCAL 多了 VAR、ARRAY、OF 等关键词
5	int f();	FUNCTION f(): INTEGER	定义 f 为返回值为整型的函数	PASCAL 至少多了一个 FUNCTION 关键词
6	int *p;	VAR p:^INTEGER	定义 p 为指向整型变量的指针变量	PASCAL 至少多了一个 VAR 关键词
7	i+=2;	i:=i+2	赋值语句	C 中如果将一个变量与另外一个操作数运算后赋值给原来的变量，使用复合的运算符可以不要重复书写此变量。C 形式上更加简洁。
8	I++	I=I+1	I 自增 1	C 定义了常用的自增 1、自减 1 运算符。形式上显得相当简洁

(2) C 语言拥有丰富的数据类型

C 语言具有整型、实型、字符型、数组类型、指针类型、结构体类型、共同体类型等数据类型。能方便地构造更加复杂的数据结构(如：使用指针构造链表、树、栈)。

(3) C 语言的运算符丰富、功能更强大

例如:

① C 语言具有复合的赋值运算符 “+[-*/%]=” (加等、减等、乘等、除等), “>>=” “<<=” (右移等、左移等), “&[^]=” (与等、或等、非等)。

$x+=5$ 等价于 $x=x+5$

② C 语言有条件运算符 “?:” 可代替简单的 if/else 语句。

如果需要表示: “如果 x 小于或等于 0, y 为 0; 否则 y 为 1” 可以采用:

$y=x \leq 0 ? 0 : 1;$

如果用一般的程序设计语言表示就应该像下面这样表示:

```
if(x<=0)y=0;
```

```
else y=1;
```

C 语言避免了一切可能的罗嗦

③ C 语言中连赋值这种操作都定义为运算符, 也就是说赋值操作本身可以作为表达式的一部分, 参与运算。如:

```
if((p=malloc(sizeof(int)))==NULL){printf("Error!");exit(1);}
```

如果改写为一般形式:

```
p=malloc(sizeof(int));
```

```
if(p==NULL){printf("Error!");exit(1);}
```

又如下面算式是正确的:

```
x=y=z=6;
```

如果改写为一般形式:

```
z=6;y=6;x=6;
```

(4) C 语言是结构化的程序设计语言

● C 语言具有结构化的控制语句(if/else,switch/case,for,while,do...while)

● 函数是 C 语言程序的模块单位。

(5) C 语言对语法限制不严格, 程序设计灵活

【具体表现】 字符与整数、逻辑值可以混合使用。 如: $10+'a'+9>6$, C 语言不检查数组下标越界, C 语言不限制对各种数据转化(编译系统可能对不合适的转化进行警告, 但不限制), 不限制指针的使用, 程序正确性由程序员保证。

实践中, C 语言程序编译时会提示: “警告错” “严重错误”。警告错误表示你使用的语法可能有问题, 但是你有时可以忽略, 你的程序仍然可以完成编译工作, 然后运行。(但是一般情况下警告错, 往往意味着程序真的有问题, 你应该认

真地检查)“严重错”是不能忽略的,编译系统发现严重错误,就不会产生目标代码。

【思考题】C 语言对语法限制不严格与程序设计灵活有没有什么联系及可能存在的问题是什么?

灵活和安全是一对矛盾,对语法限制的不严格可能也是 C 语言的一个缺点,比如:黑客可能使用越界的数组攻击你的计算机系统。JAVA 语言是优秀的网络应用程序开发语言,它必须保证安全性,它绝对不允许数组越界。此外 JAVA 不使用指针,不能直接操作客户计算机上的文件,语法检查相当严格,程序正确性容易保证,但是 JAVA 在编程时却缺乏灵活。

(6)C 语言编写的程序具有良好的可移植性

编制的程序基本上不需要修改或只需要少量修改就可以移植到其它的计算机系统或其它的操作系统。

(7) C 语言可以实现汇编语言的大部分功能

- C 语言可以直接操作计算机硬件如寄存器,各种外设 I/O 端口等。
- C 语言的指针可以直接访问内存物理地址。
- C 语言类似汇编语言的位操作可以方便地检查系统硬件的状态。

C 语言适合编写系统软件。

(8) C 语言编译后生成的目标代码小,质量高,程序的执行效率高

有资料显示只比汇编代码效率低 10%-20%。

三、C 语言基本语法成分

1、C 语言字符集

字符是 C 语言的最基本的元素,C 语言字符集由字母、数字、空白、标点和特殊字符组成(在字符串常量和注释中还可以使用汉字等其它图形符号)。由字符集中的字符可以构成 C 语言进一步的语法成分(如,标识符,关键词,运算符等)。

(1)字母: A-Z, a-z

(2)数字: 0-9

(3)空白符：空格，制表符（跳格），换行符（空行）的总称。空白符除了在字符，字符串中有意义外，编译系统忽略其它位置的空白。空白符在程序中只是起到间隔作用。在程序的恰当位置使用空白将使程序更加清晰，增强程序的可读性。

(4)标点符号、特殊字符：

! # % ^ & + - * / = ~ < > \ | . , ; : ? ' " () [] { }

2、标识符（名字）

用来标识变量名、符号常量名、函数名、数组名、类型名等实体(程序对象)的有效字符序列。标识符由用户自定义（取名字）。

C 语言标识符定义规则：

(1)标识符只能由字母、数字和下划线三种字符组成，且第一个字符必须为字母或下划线。

例如：

合法的标识符：sum,average,_total,Class,day,stu_name,p4050

不合法的标识符：M.D.John,\$123,#33,3D64,a>b

(2)大小写敏感。C 程序员习惯：变量名小写，常量名大写，但不绝对，如 windows 编程，应当使用匈牙利表示法(大小写混用，每个单词词首第一个大写，其余小写，如 WinMain)。

例如：sum 不同 Sum。BOOK 不同 book。

(3)ANSI C 没有限制标识符长度，但各个编译系统都有自己的规定和限制(TC 32 个字符，MSC 8 个字符)。

例如：student_name,student_number 如果取 8 个，这两个标识符是相同的。

(4)标识符不能与“关键词”同名，也不与系统预先定义的“标准标识符”同名。

(5)建议：标识符命名应当有一定的意义，做到见名知义。

3、运算符

运算符将常量、变量、函数连接起来组成表达式，表示各种运算。运算符可以由一个或多个字符组成。

运算符根据参与运算的操作数的个数分为：单目、双目、三目运算符。

【学习运算符的关键】

①运算符及其作用

②连接的运算对象（包括：个数、数据类型和表示格式）

③运算的优先级（包括：同类运算和不同类的运算）。如： $10+2*5/3+9>6$
&& $5<10$; $10+2*5/3+9$;

④运算的结合方向。如： $-i++$; $a=i++$;

⑤运算的结果（值）及其类型

4、分隔符

逗号，空格。起分隔、间隔作用。

5、注释符

“/*” 和 “*/” 构成一组注释符。编译系统将/* ... */之间的所有内容看作为注释，编译时编译系统忽略注释。

注释在程序中作用：提示、解释作用。

注释与软件的文档同等重要，要养成使用注释的良好习惯，这对软件的维护相当重要。记住：程序是要给别人看的，自己也许还会看自己几年前编制的程序（相当别人看你的程序），清晰的注释有助于他人理解您的程序、算法的思路。

在软件开发过程中，还可以将注释用于程序的调试-暂时屏蔽一些语句。例如，在调试程序时暂时不需要运行某段语句，而你又不希望立即从程序中删除它们，可以使用注释符将这段程序框起来，暂时屏蔽这段程序，以后可以方便地恢复。

【思考题】 关键字的概念？ 关键字的分类？（4种）？ 表示数据类型；表示存储类别；用于流程控制； sizeof(计算表达式或类型的字节数) 使用关键字的注意事项？(由系统定义，用户只能引用；用户不能将其作为它用。如：作为变量名或函数名等；注意关键字与标识符的区别)

四、C 程序结构

例 1.1:

```
main()
{
    printf("This is a C program.\n");
}
```

程序 1.1 C 语言程序示例

说明：本程序的功能是输出一行信息：This is a C program.

其中：

(1)main 表示“主函数”。每个 C 语言程序都必须有一个 main 函数，它是每一个 C 语言程序的执行起始点（入口点）。main()表示“主函数”main 的函数头。

(2)用{}括起来的是“主函数”main 的函数体。main 函数中的所有操作（或语句）都在这一对{}之间。也就是说 main 函数的所有操作都在 main 函数体中。

(3)“主函数”main 中只有一条语句，它是 C 语言的库函数，功能是用于程序的输出（显示在屏幕上），本例用于将一个字符串“This is a C program.\n”的内容输出。即在屏幕上显示：

This is a C program.

_ (回车/换行)

(4)注意：每条语句用“；”号结束语句。

【C 源程序的书写规则】一行可写多各语句；一个语句可写在多行上(怎样续行？)；区分字母的大小写；注释的使用和格式；

例：printf(“This is a c Program\n”)；等价于：printf(“This is a \ “”为续行符号，即行连接符号 c Program\n”)；

例 1.2:

```
main() /* 计算两数之和 */
{
    int a,b,sum; /* 这是定义变量 */
    a=123;b=456; /* 以下 3 行为 C 语句 */
    sum=a+b;
    printf(“sum=%d\n”,sum);
}
```

程序 1.2 计算两数之和

说明：本程序计算两数之和，并输出结果。

(1) 同样此程序也必须包含一个 main 函数作为程序执行的起点。{}之间为 main 函数的函数体，main 函数所有操作均在 main 函数体中。

(2) /* */括起来的部分是一段注释，注释只是为了改善程序的可读性，在编译、运行时不起作用（事实上编译时会跳过注释，目标代码中不会包含注释）。注

释可以放在程序任何位置，并允许占用多行，只是需要注意“/*”、“*/”匹配，一般不要嵌套注释。

(3) `int a,b,sum;`是变量声明。声明了三个具有整数类型的变量 `a,b,sum`。C 语言的变量必须先声明再使用。

(4) `a=123;b=456;`是两条赋值语句。将整数 123 赋给整型变量 `a`，将整数 456 赋给整型变量 `b`。`a,b` 两个变量分别为 123，456。注意这是两条赋值语句，每条语句均用“;”结束。

也可以将两条语句写成两行，即：

```
a=123;
```

```
b=456;
```

由此可见 C 语言程序的书写可以相当随意，但是为了保证容易阅读要遵循一定的规范。

(5) `sum=a+b;`是将 `a,b` 两变量内容相加，然后将结果赋值给整型变量 `sum`。此时 `sum` 的内容为 579。

(6) `printf("sum=%d\n",sum);`是调用库函数输出 `sum` 的结果。`%d` 为格式控制表示 `sum` 的值以十进制整数形式输出。程序运行后，输出（显示）：

```
sum=579
```

```
_ (回车/换行)
```

例 1.3

```
main()/* 主函数 */
{
    /* main 函数体开始 */
    int a,b,c; /*声明部分定义变量*/
    scanf("%d,%d",&a,&b);

    /* 调用 max，将调用结果赋给 c */
    c=max(a,b);

    printf("max=%d",c);
}/* main 函数体结束 */
```

```
int max(int x,int y)/* 计算两数中较大的数 */
{
    /* max 函数体开始 */
    int z; /* 声明部分，定义变量 */
    if(x>y)z=x;
    else z=y;
    return z; /* 将 z 值返回,通过 max 带回调用处 */
}/* max 函数体结束 */
```

程序 1.3 函数使用示例程序

【说明】

输入两个整数，计算两者较大的数，并输出。

(1)本程序包括两个函数。其中主函数 main 仍然是整个程序执行的起点。函数 max 计算两数中较大的数。

(2)主函数 main 调用 scanf 函数获得两个整数，存入 a,b 两个变量，然后调用函数 max 获得两个数字中较大的值，并赋给变量 c。最后输出变量 c 的值（结果）。

(3)int max(int x,int y)是函数 max 的函数头，函数 max 的函数头表明此函数获得两个整数，返回一个整数。

(4)函数 max 同样也用 {} 将函数体括起来。max 的函数体是函数 max 的具体实现。从参数表获得数据，处理后得到结果 z，然后将 z 返回调用函数 main。

(5)本例还表明 函数除了调用库函数外，还可以调用用户自己定义，编制的函数。

【主函数与其它函数间的调用关系】

主函数可以调用其它任何函数，包括自己定义函数和库函数。

综合上述三个例子，我们对 C 语言程序的基本组成和形式（程序结构）有了一个初步了解：

1、C 程序由函数构成（C 是函数式的语言，函数是 C 程序的基本单位）（以例 1.3 说明）

(1) 一个 C 源程序至少包含一个 main 函数,也可以包含一个 main 函数和若干个其它函数。函数是 C 程序的基本单位。

(2) 被调用的函数可以是系统提供的库函数，也可以是用户根据需要自己编写设计的函数。C 是函数式的语言，程序的全部工作都是由各个函数完成。编写 C 程序就是编写一个个函数。

(3) C 函数库非常丰富，ANSI C 提供 100 多个库函数，Turbo C 提供 300 多个库函数。

2、main 函数（主函数）是每个程序执行的起始点（以例 1.3 说明）

一个 C 程序总是从 main 函数开始执行，而不论 main 函数在程序中的位置。可以将 main 函数放在整个程序的最前面，也可以放在整个程序的最后，或者放在其它函数之间。

3、一个函数由函数首部和函数体两部分组成(以例 1.3 的 max 函数说明)

(1)函数首部：一个函数的第一行。

返回值类型 函数名([函数参数类型 1 函数参数名 1][,...,函数参数类型 2, 函数参数名 2])

注意：函数可以没有参数，但是后面的一对（）不能省略，这是格式的规定。

(2)函数体：函数首部下用一对 {} 括起来的部分。如果函数体内有多个 {}，最外层是函数体的范围。函数体一般包括声明部分、执行部分两部分。

{

[声明部分]: 在这部分定义本函数所使用的变量。

[执行部分]: 由若干条语句组成命令序列（可以在其中调用其它函数）。

}

4、C程序书写格式自由

(1) 一行可以写几个语句，一个语句也可以写在多行上。

(2) C程序没有行号，也没有 FORTRAN, COBOL 那样严格规定书写格式（语句必须从某一行开始）。

(3) 每条语句的最后必须有一个分号“;”表示语句的结束。

5、可以使用/* */对C程序中的任何部分作注释

注释可以提高程序可读性，使用注释是编程人员的良好习惯。

(1) 实践中，编写好的程序往往需要修改、完善，事实上没有一个应用系统是不需要修改、完善的。很多人会发现自己编写的程序在经历了一段时间以后，由于缺乏必要的文档、必要的注释，最后连自己都很难再读懂。需要花费大量时间重新思考、理解原来的程序。这浪费了大量的时间。如果一开始编程就对程序进行注释，刚开始麻烦一些，但日后可以节省大量的时间。

(2) 一个实际的系统往往是多人合作开发，程序文档、注释是其中重要的交流工具。

6、C语言本身不提供输入/输出语句，输入/输出的操作是通过调用库函数 (scanf, printf) 完成。

输入/输出操作涉及具体计算机硬件，把输入/输出操作放在函数中处理，可以简化C语言和C的编译系统，便于C语言在各种计算机上实现。不同的计算机系统需要对函数库中的函数做不同的处理，以便实现同样或类似的功能。

不同的计算机系统除了提供函数库中的标准函数外，还按照硬件的情况提供一些专门的函数。因此不同计算机系统提供的函数数量、功能会有一定差异。

五、C语言程序的编辑、编译、运行(C程序的上机步骤)

1、源程序、目标程序、可执行程序的概念（补充）

(1) 程序：为了使计算机能按照人们的意志工作，就要根据问题的要求，编写相应的程序。程序是一组计算机可以识别和执行的指令，每一条指令使计算机执行特定的操作。

(2) 源程序：程序可以用高级语言或汇编语言编写，用高级语言或汇编语言编写的程序称为源程序。C程序源程序的扩展名为“.c”

事实上我们编写的程序，不管采用什么计算机语言，都是源程序，有几个人还会用机器语言去编程！

源程序不能直接在计算机上执行，需要用“编译程序”将源程序翻译为二进制形式的代码。

(3) 目标程序：源程序经过“编译程序”翻译所得到的二进制代码称为目标程序。目标程序的扩展名为“.obj”

目标代码尽管已经是机器指令，但是还不能运行，因为目标程序还没有解决函数调用问题，需要将各个目标程序与库函数连接，才能形成完整的可执行的程序。

(4) 可执行程序：目标程序与库函数连接，形成的完整的可在操作系统下独立执行的程序称为可执行程序。可执行程序的扩展名为“.exe” (在 dos/windows 环境下)

2、C 语言程序的上机步骤

输入与编辑源程序->编译源程序，产生目标代码——>

连接各个目标代码、库函数，产生可执行程序——>运行程序。

3、Turbo C 的使用(DOS 环境)

Turbo C2.0 是 Borland 公司开发的微机上一个 C 语言集成开发环境。可以在 Turbo C 中完成 C 语言程序的编辑、编译、连接、运行、调试。

具体操作上机时演示（重点：启动，退出，重要的菜单项）

重点菜单项：

File->new,load,save,write to

Compile->compile to obj,make exe file,build all

Run->run,go to cursor,trace into,step over,user screen,program reset

Debug->Evaluate

4. Visual C 的使用（Windows 环境）

Visual C 是 windows 环境下的 C 应用程序,C++应用程序,Windows 应用程序集成开发环境。

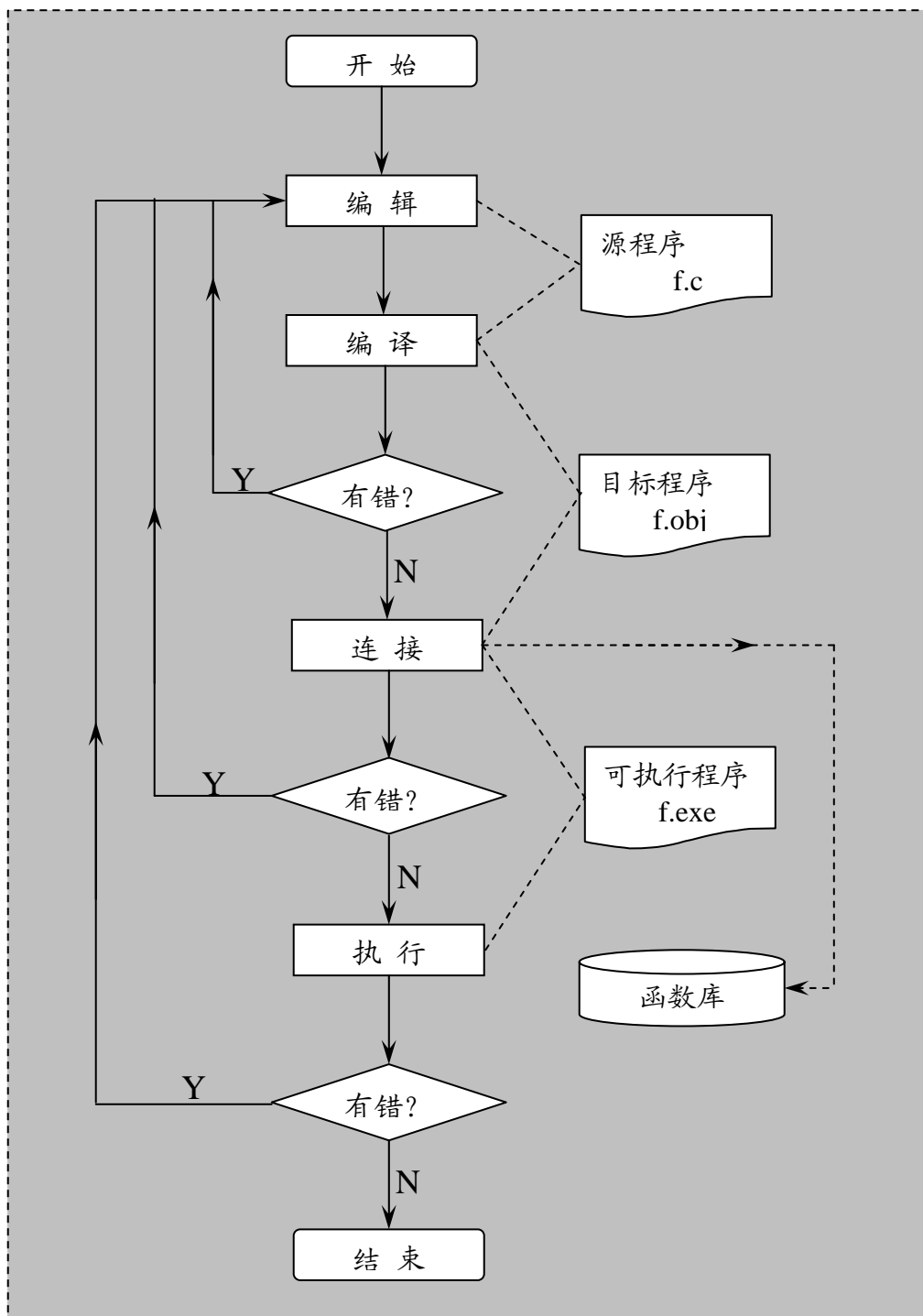


图 1.1 C 程序执行过程

第二章

基本数据类型

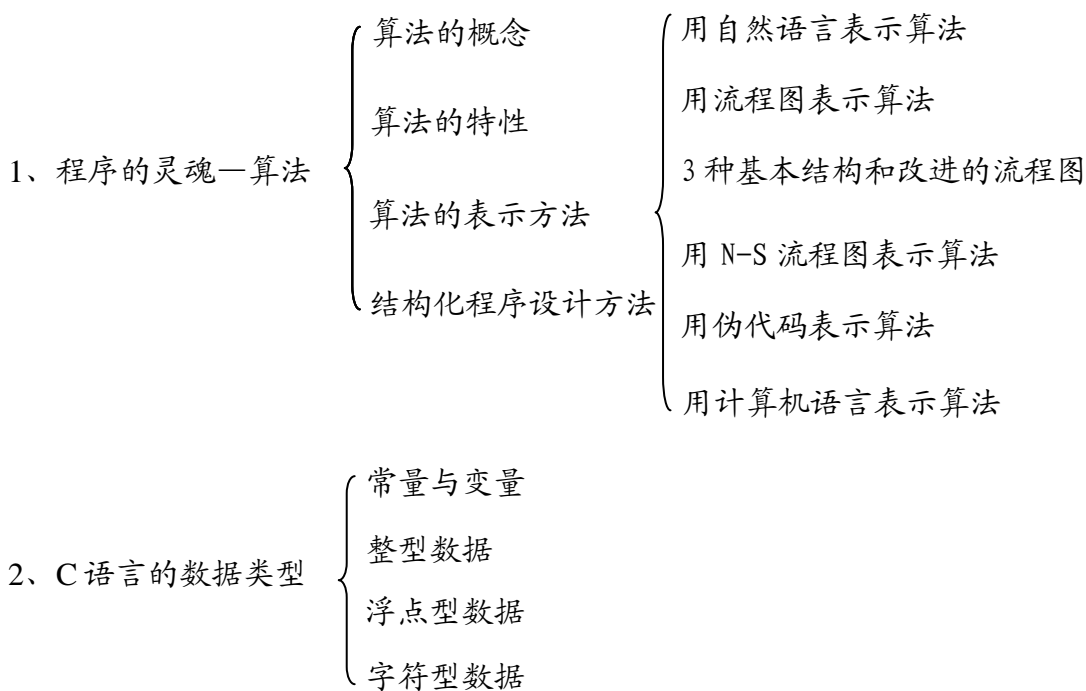
【基本要求】

通过本单元的学习，使学生了解算法的概念和特点，熟悉算法的构成要素和三种基本结构，并掌握算法的表示方法和结构化程序设计方法，掌握 c 语言中数据的表现形式，掌握各种数据类型的基本情况，掌握常量的表示方法，掌握变量的定义方法，能够根据要解决的问题的实际情况选用适当的数据类型，熟练掌握数值表达式的构造方法，能够用表达式完成数值计算。

【教学重点】

算法的概念和特点，算法的构成要素和三种基本结构(重点)，算法的表示方法(重点)，结构化程序设计方法(重点)，c 语言中数据的表现形式及各种数据类型的基本情况，常量的表示方法，变量的定义方法(重点)，根据要解决的问题的实际情况选用适当的数据类型(重点)，数值表达式的构造方法(重点)，能够用表达式完成数值计算(重点)。

【本章结构】



3、变量赋初值

4、各类数值型数据间的混合运算

5、运算符和表达式 { 算术运算符和算术表达式
赋值运算符和赋值表达式
逗号运算符和逗号表达式

第一部分 算法及其要素、算法表示

一、算法的概念和特点

1、概念

2、计算机算法与分类

数值计算算法，非数值计算算法

【思考题】数值计算算法与非数值计算算法可能存在的差别及它们可能的共同点？

3、算法的特点

有穷性；确定性；有效性；有零个或多个输入；有一个或多个输出

二、算法的构成要素、三种基本结构与表示方法

1、算法的构成要素

操作(主要包括各种运算)，控制结构(用于控制各种操作的执行顺序)

结构化程序设计方法规定：一个程序只能由三种基本控制结构（或其派生结构）组成。

2、算法的三种基本结构

顺序结构；选择结构；循环结构（包括：当型循环；直到型循环。**注意两者的区别**）

三、算法的表示方法

1、自然语言表示法(缺点是什么？)

2、流程图示法

(1)传统流程图示法

各种流程图符的作用是什么？

传统流程图示法对三种基本结构的表示形式

(2)N-S 图示法（无流程线）

N-S 图示法对三种基本结构的表示形式

【思考题】N-S 图示法与传统流程图示法的区别？

(3) PAD (Problem Analysis Diagram: 问题分析图) 图表示

【补充】

PAD 图对三种基本结构的表示形式(无流程线；只有两重关系)，如图 2.1 所示

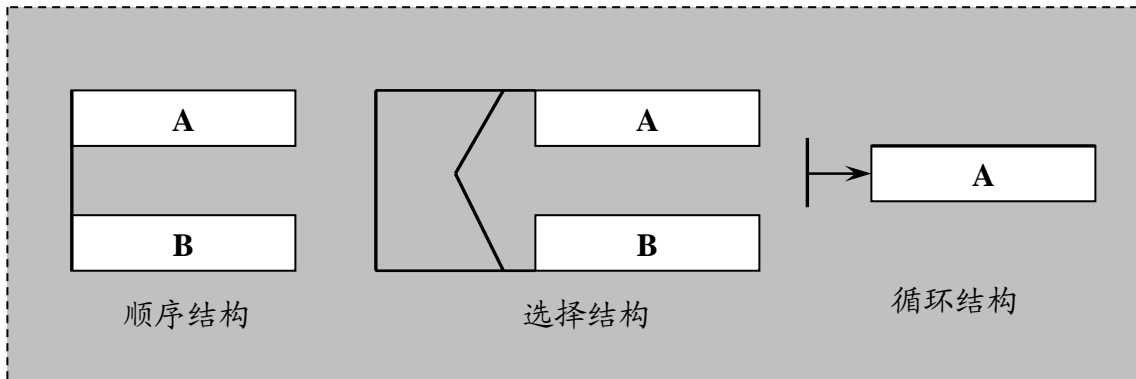


图 2.1 PAD 图三种基本结构

(4) 伪代码 (Pseudo Code) 表示法

(5) 计算机语言表示法即编写程序

包括两个部分： 算法的设计； 算法的实现（真正的实现要上机运行）

例 2.1 交换两个变量的值，不用中间变量。(比较几种算法的表示方法)

(1)用传统流程图表示算法，如图 2.2 所示

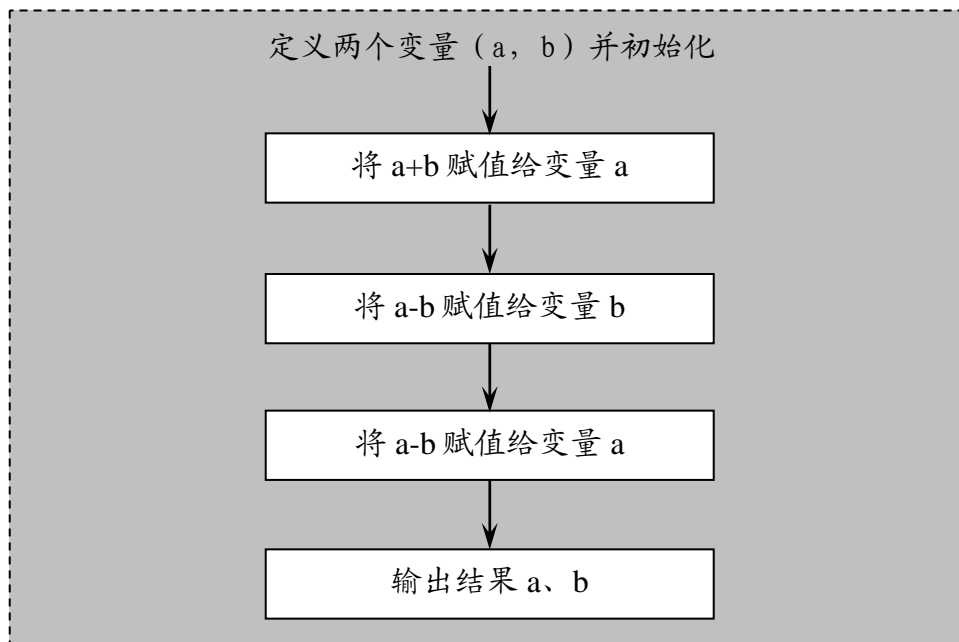


图 2.2 传统流程图法

(2) N-S 流程图表示算法(没有流程线)，如图 2.3 所示

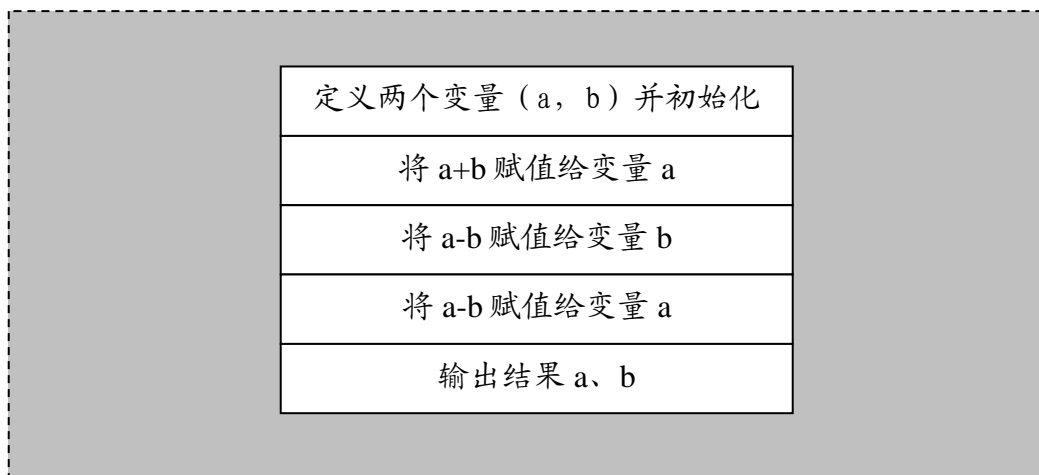


图 2.3 N-S 流程图

(3) 用伪代码表示算法

int a,b 给 a、b 赋值 $a+b \Rightarrow a$ $a-b \Rightarrow b$ $a-b \Rightarrow a$ 输出变量 a、b 的值

(4) C 语言表示其算法

```
main(){
int a=10,b=20; a=a+b; b=a-b; a=a-b; printf("a=%d b=%d",a,b);
}
```

四、结构化程序设计方法

至顶向下;

逐步细化;

模块化设计;

结构化编码。

第二部分 数据类型及整型、实型数据

一、C 的数据类型

程序、算法处理的对象是数据。数据以某种特定的形式存在(如整数、实数、字符),而且不同的数据还存在某些联系(如由若干整数构成的数组)。数据结构就是指数据的组织形式(逻辑结构、物理结构)。处理同样的问题如果数据结构不同,算法也不同,应当综合考虑算法和数据结构、选择最佳的数据结构和算法。

C 语言的数据结构是以数据类型的形式体现。也就是说 C 语言中数据是有类型的,数据的类型简称数据类型。例如,整型数据、实型数据、整型数组类型、字符数组类型(字符串)分别代表我们常说的整数、实数、数列、字符串。

C 语言的数据类型结构如图 2.4 所示

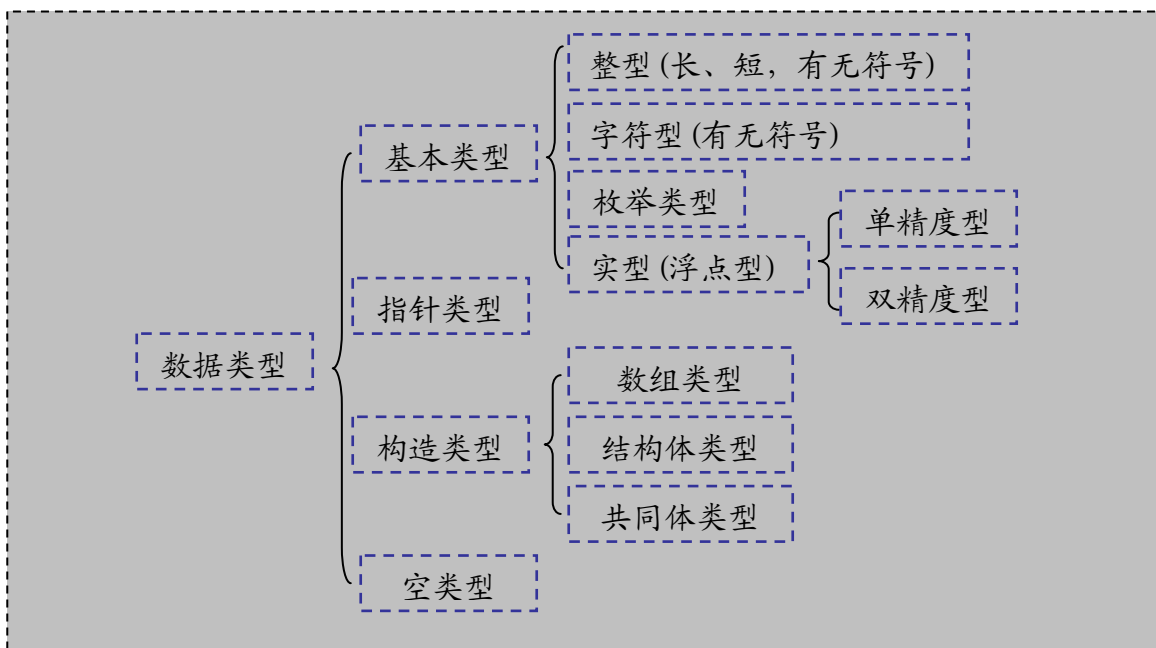


图 2.4 C 语言数据类型结构图

【注意】

(1)不同的数据类型有不同的取值范围。如有符号整数取值范围-32768 ~ 32767, 浮点数-3.4e-38 ~ 3.4e38。

(2)不同的数据类型有不同的操作。如整型数可以取余操作, 实型数据却不行; 整型、实型数据可以有加法, 字符数组不行。

(3)不同的数据类型即使有相同的操作有时含义也不同, 如指针数据自增 1 与整数自增 1 含义是不同的。

(4)不同的数据类型对计算机可能出现的错误不同。如整型数的溢出错误, 浮点数的精度的丢失(有效数字位数不够)。

(5)C 语言的数据类型可以构造复杂的数据结构。如使用结构体数组可以构造线性表。使用指针类型、结构体类型可以构造线性链表(栈、队列)、树、图。

(在《数据结构》课程介绍)

(6)C 语言中的数据有变量与常量, 它们分别属于上述这些类型。

二、常量与变量

1、常量: 在程序的运行过程中, 其值不能改变的量称为常量。

【注意】

(1)常量有不同的类型, 如 12、0、-3 为整型常量, 4.6、-1.23 为实型常量, 'a'、'd' 字符常量。常量可以从字面形式即可判断字面常量或直接常量。

(2)符号常量

```
#define PI 3.1416
```

使用符号常量的好处:

①含义清楚、见名知意。

②修改方便、一改全改。

例 2.2: 符号常量应用(如程序 2.1 所示)

```
#define PI 3.14
main()
{
    float area;
    area=10*10*PI;
    printf("area=%f\n",area);
}
结果: area=314.000000
```

程序 2.1 符号常量应用

2、变量: 在程序的运行过程中, 其值可以改变的量称为变量。

【注意】

(1)变量名(用标识符表示)、变量在内存中占据的存储单元、变量值三者关系。

变量名在程序运行过程中不会改变, 变量的值可以改变。变量名遵守标识符准则。如图 2.5 所示

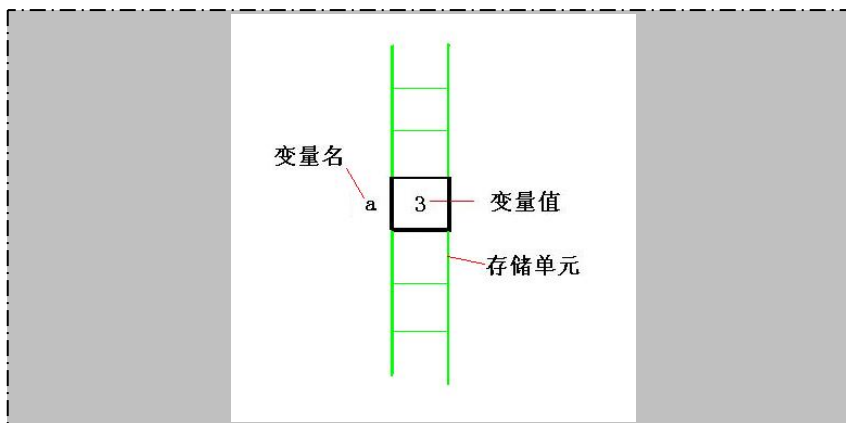


图 2.5 变量示意图

(2)C 语言中变量: “先定义, 后使用”。即就是说, C 要求对所有用到的变量做强制定义。

①只有申明过的变量才可以在程序中使用, 这使得变量名的拼写错误容易出现。BASIC 却不是这样。有时你会用了两个相近似变量而你根本没有发现, 却当作同一个变量在使用。

②申明的变量属于确定的类型, 编译系统可方便地检查变量所进行运算的合法性。

③在编译时根据变量类型可以为变量确定存储空间，“先定义后使用”使程序效率高。

三、整型数据

1、整型常数的三种表示方法

(1)十进制。

例如 123, -456, 0。

(2)八进制。以 0 开头，后面跟几位的数字（0-7）。

例如：0123= (123) 8= (83) 10; -011= (-11) 8= (-9) 10。

(3)十六进制。以 0x 开头，后面跟几位的数字（0-9, A-F）。

例如：0x123=291, -0x12=-18。

整型常量的类型(整型常数的后缀)在整型变量部分介绍，这里只要知道怎么表示。

整型常量后可以用：

u 或 U 明确说明为无符号整型数

l 或 L 明确说明为长整型数。

2、整型变量

(1)整型数据在内存中的存放形式

数据在内存中以二进制形式存放，事实上以补码形式存放。

例 2.3: 定义一个整型变量 i=10，如图 2.6 所示

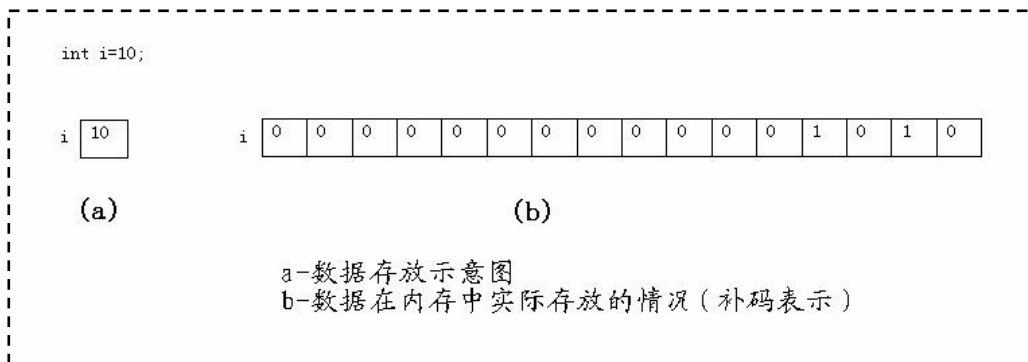


图 2.6 整型变量定义及存储

【注意】

(1)带符号数的表示，原码、反码、补码。

(2)原码-补码相互转化。正数的补码与其原码相同，负数的补码是其对应的原码数值位按位取反+1。

例 2.4: 10, -10 的计算机机内表示。

思路：先将数值表示为二进制形式（十进制=>二进制，除 2 取余），即获得数值的原码。将原码转化为补码，就是机内表示。

$10 = (1010)_2 = (0000, 0000, 0000, 1010)$ 原 $= (0000, 0000, 0000, 1010)$ 补。

$-10 = (-1010)_2 = (1000, 0000, 0000, 1010)$ 原 $= (1111, 1111, 1111, 0110)$ 补。

从 10、-10 的计算机机内表示可以看出正数、负数机内表示（补码表示）看上去明显不同。

3、整型变量的分类

整型变量的基本类型为 int。通过加上修饰符，可定义更多的整数数据类型。

(1) 根据表达范围可以分为：基本整型 (int)、短整型 (short int)、长整型 (long int)。用 long 型可以获得大范围的整数，但同时会降低运算速度。

(2) 根据是否有符号可以分为：有符号 (signed, 默认)，无符号 (unsigned) - 目的：扩大表示范围，有些情况只需要用正整数。

有符号整型数的存储单元的最高位是符号位 (0: 正、1: 负)，其余为数值位。无符号整型数的存储单元的全部二进制位用于存放数值本身而不包含符号。

归纳起来可以用 6 种整型变量：

- 有符号基本整型：[signed]int
- 有符号短整型：[signed]short[int]
- 有符号长整型：[signed]long[int]
- 无符号基本整型：unsigned [int]
- 无符号短整型：unsigned short [int]
- 无符号长整型：unsigned long [int]

例 2.5：保存整数 13 的各种整型数据类型。如图 2.6 所示

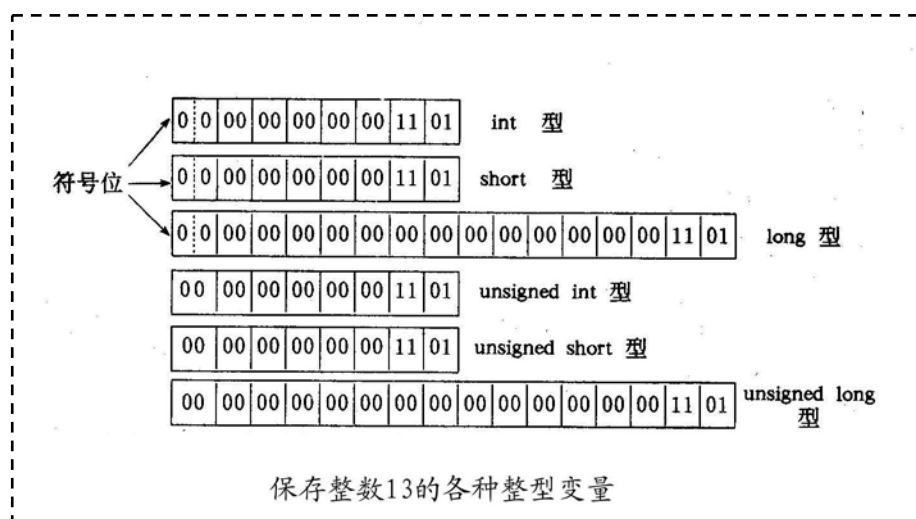


图 2.6 整型变量的存储

C 标准没有具体规定上面数据类型所占用的字节数，只要求 long 型数据长度不短于 int 型，short 型不长于 int 型。具体如何实现，由各计算机系统自行决定。如微机上 short,int 都是 16 位，而 long 是 32 位；VAX750 机，int, long 都是 32 位，而 short 是 16 位。

表格 2.1 数据类型占用字节数比较

数据类型 \ 机型	IBM PC	PDP-11	VAX-11	Honeywell	IBM370
int	16	16	32	36	32
short	16	16	16	36	16
long	32	32	32	36	32
unsigned int	16	16	32	36	32
unsigned short	16	16	16	36	16
unsigned long	32	32	32	36	32

表格 2.2 数的表示范围

	所占位数	数的范围
int	16	-32768——32767
short[int]	16	-32768——32767
long[int]	32	-2147483648——2147483647
unsigned[int]	16	0——65535
unsigned short	16	0——65535
unsigned long	32	0——4294967295

4、整型变量的定义

格式：数据类型名 变量名表；

例 2.6：如程序 2.2 所示

```
main()
{
    int a,b,c,d;
    unsigned u;
    a=12; b=-24; u=10;
    c=a+u; d=b+u;
    printf(“%d,%d\n”,c,d);
}
```

程序 2.2 整型变量的定义

【注意】

(1) 变量定义时，可以说明多个相同类型的变量。各个变量用“，”分隔。类型说明与变量名之间至少有一个空格间隔。

(2) 最后一个变量名之后必须用“;”结尾。

(3) 变量说明必须在变量使用之前。

(4) 可以在定义变量的同时，对变量进行初始化。

例 2.7 变量的初始化，如程序 2.3 所示

```
main()
{
    int a=3, b=5;
    printf("a+b=%d\n", a+b);
}
```

程序 2.3 变量的初始化

5、整型数据的溢出

【思考】整型数最大允许值+1，最小允许值-1，会出现什么情况？

$32767+1=-32768$ ； $-32768-1=32767$ 。

例 2.8 整型数据的溢出

```
main()
{
    int a,b;
    a=32767;
    b=a+1;
    printf("\na=%d,a+1=%d\n", a,b);

    a=-32768;
    b=a-1;
    printf("\na=%d,a-1=%d\n", a,b);

    getch();
}

a=32767,a+1=-32768
a=-32768,a-1=32767
```

程序 2.4 整型数据的溢出

超出范围就发生“溢出”，运行时不报错。

四、实型数据

1、实型常量的表示方法

实数（浮点数）有两种表示形式：

(1)十进制小数形式。由数字，小数点组成（必须有小数点）。

例如：.123、123.、123.0、0.0

(2)指数形式。格式：aEn。

例如：123e3、123E3 都是实数的合法表示。

【注意】

(1)字母 e 或 E 之前必须有数字，e 后面的指数必须为整数。

例如：e3、2.1e3.5、.e3、e 都不是合法的指数形式。

(2)规范化的指数形式。在字母 e 或 E 之前的小数部分，小数点左边应当有且只能有一位非 0 数字。用指数形式输出时，是按规范化的指数形式输出的。

例如：2.3478e2、3.0999E5、6.46832e12 都属于规范化的指数形式。

(3)实型常量都是双精度，如果要指定它为单精度，可以加后缀 f（实型数据类型参看实型变量部分说明）。

2、实型变量

(1)实型数据在内存中的存放形式

一个实型数据一般在内存中占 4 个字节（32 位）。与整数存储方式不同，实型数据是按照指数形式存储的。系统将实型数据分为小数部分和指数部分，分别存放。实型数据存放的示意图，如图 2.7 所示

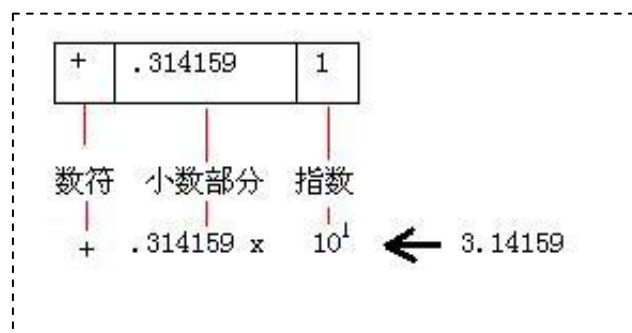


图 2.7 实型数据的内存存放形式

标准 C 没有规定用多少位表示小数，多少位表示指数部分，由 C 编译系统自定。例如，很多编译系统以 24 位表示小数部分，8 位表示指数部分。小数部分占的位数多，实型数据的有效数字多，精度高；指数部分占的位数多，则表示的数值范围大。

3、实型变量的分类

实型变量分为：单精度(float)、双精度(double)、长双精度(long double)。

ANSI C 没有规定每种数据类型的长度、精度和数值范围。表 2.3 列出微机常用的 C 编译系统的情况，不同的系统会有差异。

表格 2.3 数据类型的长度、精度和数值范围

类型	比特数	有效数字	数值范围
float	32	6-7	$-3.4 \times 10^{+38} \sim 3.4 \times 10^{+38}$
double	64	15-16	$-1.7 \times 10^{+308} \sim 1.7 \times 10^{+308}$
long double	128	18-19	$-1.2 \times 10^{+4932} \sim 1.2 \times 10^{+4932}$

对于每一个实型变量也都应该先定义后使用。如：

float x,y;

double z;

long double t;

4、实型数据的舍入误差（对比：整型数据的溢出）

实型变量是用有限的存储单元存储的，因此提供的有效数字是有限的，在有效位以外的数字将被舍去，由此可能会产生一些误差。

例 2.9 实型数据的舍入误差（实型变量只能保证 7 位有效数字，后面的数字无意义）

```
main()
{
    float a,b;
    a=123456.789e5;
    b=a+20;
    printf("a=%f,b=%f\n",a,b);
    printf("a=%e,b=%e\n",a,b);
}

a=12345678848.000000,b=12345678848.000000
a=1.23457e+10,b=1.23457e+10
```

程序 2.5 实型数据的舍入误差

【结论】

由于实数存在舍入误差，使用时要注意：

- (1) 不要试图用一个实数精确表示一个大整数，记住：浮点数是不精确的。
- (2) 实数一般不判断“相等”，而是判断接近或近似。
- (3) 避免直接将一个很大的实数与一个很小的实数相加、相减，否则会“丢失”小的数。

(4)根据要求选择单精度、双精度。

例 2.10 根据精度要求，选择实数类型

```
main()
{
    float a;
    float b;
    a=33333.33333;
    b=33333.3333333333;
    printf("a=%f,b=%f\n",a,b);
}
```

程序 2.6 根据精度选择类型

5、实型常量的类型

(1)许多 C 编译系统将实型常量作为双精度实数来处理，这样可以保证较高的精度，缺点是运算速度降低。在实数的后面加字符 f 或 F，如 1.65f、654.87F，使编译系统按单精度处理实数。

(2)实型常量可以赋值给一个 float、double、long double 型变量。根据变量的类型截取实型常量中相应的有效数字。

第三部分 字符型数据、变量赋值

一、字符型数据

1、字符常量

字符常量是用单引号（' '）括起来的一个字符。字符常量主要用下面几种形式表示：

(1)可显示的字符常量直接用单引号括起来，如，'a'、'x'、'D'、'?','\$'等都是字符常量。

所有字符常量（包括可以显示的、不可显示的）均可以使用字符的转义表示法表示（ASCII 码表示）。

(2)转义表示格式：'\ddd' 或 '\xhh'（其中 ddd,hh 是字符的 ASCII 码，ddd 八进制、hh 十六进制）。注意：不可写成 '\0xhh' 或 '\0ddd'（整数）。

(3)预先定义的一部分常用的转义字符。如 '\n' -换行，'\t' -水平制表。

字符形式	功 能
<code>\n</code>	换行 (将当前位置移到下一行开头)
<code>\t</code>	横向跳格(即跳到下一个输出区) (水平制表)
<code>\v</code>	竖向跳格
<code>\b</code>	退格 (将当前位置移到前一位)
<code>\r</code>	回车 (将当前位置移到本行开头)
<code>\f</code>	走纸换页 (将当前位置移到下页开头)
<code>\\</code>	反斜杠字符“\”
<code>\'</code>	单引号(撇号)字符
<code>\ddd</code>	1 到 3 位 8 进制数所代表的字符
<code>\xhh</code>	1 到 2 位 16 进制数所代表的字符

图 2.8 各种字符格式

【注意】

(1) `\b`,`\t`对输出的控制作用。

(2) 在打印机打印和在显示器上输出的不同效果。打印机打印过的地方可以永久保留，显示器不能在同一个位置只能显示最后输出的字符。

2、字符变量

字符型变量是用来存放字符数据，同时只能存放一个字符。所有编译系统都规定以一个字节来存放一个字符，或者说，一个字符变量在内存中占一个字节。

3、字符数据在内存中的存储形式及其使用

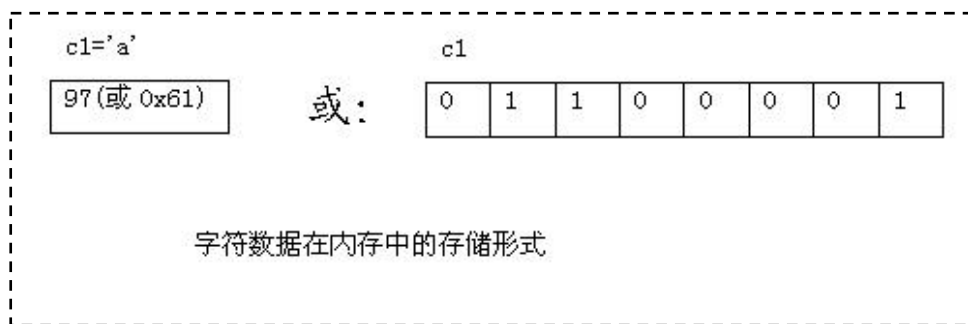


图 2.9 字符数据在内存中的存储

字符数据在内存中的存储形式：以字符的 ASCII 码，以二进制形式存放，占用 1 个字节。

可以看出字符数据以 ASCII 码存储的形式与整数的存储形式类似，这使得字符型数据和整型数据之间可以通用（当作整型量）。具体表现为：

- (1)可以将整型量赋值给字符变量，也可以将字符型量赋值给整型变量。
- (2)可以对字符数据进行算术运算，相当于对它们的 ASCII 码进行算术运算。
- (3)一个字符数据既可以以字符形式输出（ASCII 码对应的字符），也可以以整数形式输出（直接输出 ASCII 码）。

【注意】

尽管字符型数据和整型数据之间可以通用，但是字符型只占 1 个字节，即如果作为整数使用范围 0-255（无符号）-128-127（有符号）。

例 2.11 给字符变量赋以整数（字符型、整型数据通用）

```
main() /* 字符 'a' 的各种表达方法 */
{
    char c1='a';
    char c2='\x61';/* note: '\x..','\...' */
    char c3='\141';

    char c4=97;
    char c5=0x61; /* note: 0x...,0... */
    char c6=0141;

    printf("\nc1=%c,c2=%c,c3=%c,c4=%c,c5=%c,c6=%c\n",c1,c2,c3,c4,c5,c6);
    printf("c1=%d,c2=%d,c3=%d,c4=%d,c5=%d,c6=%d\n",c1,c2,c3,c4,c5,c6);
    getch();
}

c1=a,c2=a,c3=a,c4=a,c5=a,c6=a
c1=97,c2=97,c3=97,c4=97,c5=97,c6=97
过程：整型数=>机内表示(两个字节)>取低 8 位赋值给字符变量
```

程序 2.7 字符变量赋值

例 2.12 大小写字母的转换（ASCII 码表：小写字母比对应的大写字母的 ASCII 码大 32，本例还可以看出允许字符数据与整数直接进行算术运算，运算时字符数据用 ASCII 码值参与运算）

```
main()
{
    char c1,c2,c3;
    c1='a';
    c2='b';
    c1=c1-32;
    c2=c2-32;
    c3=130;

    printf("\nc1=%c %c %c\n",c1,c2,c3);
    printf("%d %d %d\n",c1,c2,c3);
    getch();
}
A B ?65 66 -126
```

程序 2.8 字母转换

4、字符串常量

字符串变量：是一对双引号（“”）括起来的字符序列。

例如：“How dow you do?”，“CHINA”，“a”，“\$123.45”。

【注意】

(1)区分字符常量与字符串常量。如“a”和‘a’。

C 语言规定：在每个字符串的结尾加一个“字符串结束标志”，以便系统据此判断字符串是否结束。C 规定以‘\0’（ASCII 码为 0 的字符）作为字符串结束标志。

如：“CHINA”在内存中的存储应当是：（长度=6）

C	H	I	N	A	‘\0’
---	---	---	---	---	------

(2)不能将字符串赋给字符变量。

(3)C 语言没有专门的字符串变量，如果想将一个字符串存放在变量中，可以使用字符数组。即用一个字符数组来存放一个字符串，数组中每一个元素存放一个字符。

二、变量赋初值

程序中常常需要对一些变量预先设置初值，C 语言允许在定义变量的同时使变量初始化。

例如：

```
int a=3;      /* 指定 a 为整型变量，初值为 3 */  
float f=3.56; /* 指定 f 为实型变量，初值为 3.56 */  
char c='a';   /* 指定 c 为字符型变量，初值为'a' */
```

可以只对定义的一部分变量赋初值。

```
int a,b=2,c=5;
```

/* 指定 a,b,c 为整型变量，只对 b、c 初始化，b 的初值为 2，c 的初值为 5*/

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予初值的，相当于有一个赋值语句。

```
int a=3;
```

相当于：

```
int a;
```

```
a=3;
```

三、各类数值型数据（整型、实型、字符型）的混合运算

整型（包括 int, short, long）和实型（包括 float, double）数据可以混合运算，另外字符型数据和整型数据可以通用，因此，整型、实型、字符型数据之间可以混合运算。

例如：表达式 $10 + 'a' + 1.5 - 8765.1234 * 'b'$ 是合法的。

在进行运算时，不同类型的数据先转换成同一类型，然后进行计算，转换的方法有两种：自动转换（隐式转换）；强制转换。

1、自动转换（隐式转换）

自动转换发生在不同类型数据进行混合运算时，由编译系统自动完成。转换规则：如图

(1) 类型不同，先转换为同一类型，然后进行运算。

(2) 图中纵向的箭头表示当运算对象为不同类型时转换的方向。可以看到箭头由低级别数据类型指向高级别数据类型，即数据总是由低级别向高级别转换。即按数据长度增加的方向进行，保证精度不降低。

(3) 图中横向向左的箭头表示必定的转换（不必考虑其它运算对象）。如字符数据参与运算必定转化为整数，float 型数据在运算时一律先转换为双精度型，以提高运算精度（即使是两个 float 型数据相加，也先都转换为 double 型，然后再相加）。

(4) 赋值运算，如果赋值号“=”两边的数据类型不同，赋值号右边的类型转换为左边的类型。这种转换是截断型的转换，不会四舍五入。

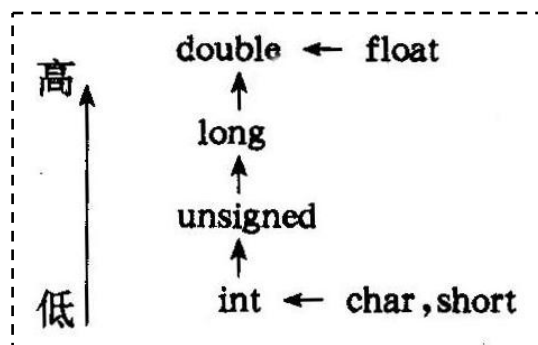


图 2.10 类型转换

2、强制转换

强制转换是通过类型转换运算来实现。

一般形式：（类型说明符）表达式

功能：把表达式的结果强制转换为类型说明符所表示的类型。

例如:

(int)a 将 a 的结果强制转换为整型量。

(int)(x+y) 将 x+y 的结果强制转换为整型量。

(float)a+b 将 a 的内容强制转换为浮点数, 再与 b 相加。

【说明】

(1) 类型说明和表达式都需要加括号 (单个变量可以不加括号)

(2) 无论隐式转换, 强制转换都是临时转换, 不改变数据本身的类型和值。

例 2.13: 强制类型转换

```
main()
{
    float f=5.75;
    printf("(int)f=%d\n", (int)f); /* 将 f 的结果强制转换为整型, 输出 */
    printf("f=%f\n", f); /* 输出 f 的值 */
}
结果:
(int)f=5
f=5.750000
```

程序 2.9 强制类型转换示例

第四部分 C 运算符

一、C 运算符简介

运算符: 狭义的运算符是表示各种运算的符号。

表达式: 使用运算符将常量、变量、函数连接起来, 构成表达式。

C 语言运算符丰富, 范围很宽, 把除了控制语句和输入/输出以外的几乎所有的基本操作都作为运算符处理, 所以 C 语言运算符可以看作是操作符。C 语言丰富的运算符构成 C 语言丰富的表达式 (是运算符就可以构成表达式)。运算符丰富、表达式丰富、灵活。

在 C 语言中除了提供一般高级语言的算术、关系、逻辑运算符外, 还提供赋值运算符, 位操作运算符、自增自减运算符等等。甚至数组下标, 函数调用都作为运算符。

C 的运算符有以下几类：如图 2.11 所示

1. 算术运算符	(+ - * / %) ++, --
2. 关系运算符	(> < == >= <= !=)
3. 逻辑运算符	(! &&)
4. 位运算符	(<< >> ~ ! ^ &)
5. 赋值运算符	(= 及其扩展赋值运算符)
6. 条件运算符	(? :)
7. 逗号运算符	(,)
8. 指针运算符	(* 和 &)
9. 求字节数运算符	(sizeof)
10. 强制类型转换运算符	(类型)
11. 分量运算符	(. →)
12. 下标运算符	([])
13. 其它	(如函数调用运算符())

图 2.11 C 语言的运算符

二、算术运算符和算术表达式

1、算术运算符

+ (加法运算符。如 3+5)

- (减法运算符或负值运算符。如 5-2,-3)

* (乘法运算符。如 3*5)

/ (除法运算符。如 5/3, 5.0/3)

% (模运算符或求余运算符，%要求两侧均为整型数据。如 7%4 的值为 3)。

除了负值运算符-单目运算符外，其它都是双目运算符。

【说明】

(1) 两个整数相除的结果为整数，如 5/3 的结果为 1，舍去小数部分。但是如果除数或被除数中有一个为负值，则舍入的方向是不固定的，多数机器采用“向 0 取整”的方法（实际上就是舍去小数部分，注意：不是四舍五入）。

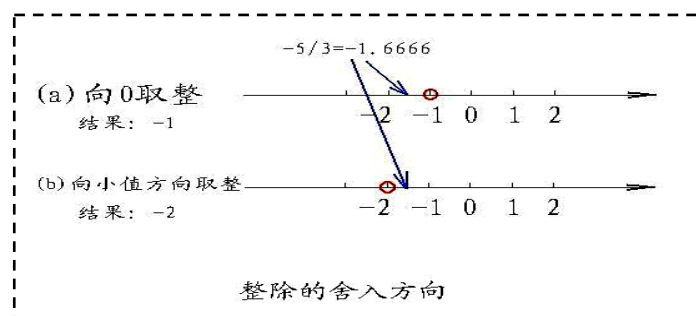


图 2.12 整数相除

(2)如果参加+,-,*,/运算的两个数有一个为实数,则结果为 double 型,因为所有实数都按 double 型进行计算。

(3)求余运算符%,要求两个操作数均为整型,结果为两数相除所得的余数。求余也称为求模。一般情况,余数的符号与被除数符号相同。

例如: $-8\%5=-3$; $8\%-5=3$

2、算术表达式

算术表达式:用算术运算符和括号将运算对象(也称操作数)连接起来的、符合 C 语法规则的式子,称为算术表达式。运算对象可以是常量、变量、函数等。

例如,下面是一个合法的 C 算术表达式。 $a*b/c-1.5+'a'$ 。

【注意】

C 语言算术表达式的书写形式与数学表达式的书写形式有一定的区别:

(1) C 语言算术表达式的乘号(*)不能省略。例如:数学式 b^2-4ac ,相应的 C 表达式应该写成: $b*b-4*a*c$ 。

(2) C 语言表达式中只能出现字符集允许的字符。例如,数学 πr^2 相应的 C 表达式应该写成: $PI*r*r$ 。(其中 PI 是已经定义的符号常量)

(3) C 语言算术表达式不允许有分子分母的形式。例如, $(a+b)/(c+d)$ 。

(4) C 语言算术表达式只使用圆括号改变运算的优先顺序(不要指望用{}[])。可以使用多层圆括号,此时左右括号必须配对,运算时从内层括号开始,由内向外依次计算表达式的值。

3、(算术)运算符的优先级与结合性

C 语言规定了进行表达式求值过程中,各运算符的“优先级”和“结合性”。

(1)C 语言规定了运算符的“优先级”和“结合性”。在表达式求值时,先按运算符的“优先级别”高低次序执行。

如表达式: $a-b*c$ 等价于 $a-(b*c)$,”*”运算符优先级高于“-”运算符。

(2)如果在一个运算对象两侧的运算符的优先级别相同,则按规定的“结合方向”处理。

【思考题】例如: $a-b+c$,到底是 $(a-b)+c$ 还是 $a-(b+c)$? (b 先与 a 参与运算还是先于 c 参与运算?)

可知: +/ - 运算优先级别相同,结合性为“自左向右”,即就是说 b 先与左边的 a 结合。所以 $a-b+c$ 等价于 $(a-b)+c$ 。

左结合性(自左向右结合方向):运算对象先与左面的运算符结合。

右结合性(自右向左结合方向):运算对象先与右面的运算符结合。

(3)在书写多个运算符的表达式时，应当注意各个运算符的优先级，确保表达式中的运算符能以正确的顺序参与运算。对于复杂表达式为了清晰起见可以加圆括号“()”强制规定计算顺序。

三、赋值运算符和赋值表达式

1、赋值运算符、赋值表达式

赋值运算符：赋值符号“=”就是赋值运算符。

赋值表达式：由赋值运算符组成的表达式称为赋值表达式。

一般形式：

〈变量〉〈赋值符〉〈表达式〉

赋值表达式的求解过程：将赋值运算符右侧的表达式值赋给左侧的变量，同时整个赋值表达式的值就是刚才所赋的值。赋值的含义：将赋值运算符右边的表达式的值存放到左边变量名标识的存储单元中。

例如：x=10+y；执行赋值运算（操作），将 10+y 的值赋给变量 x,同时整个表达式的值就是刚才所赋的值。

【说明】

(1)赋值运算符左边必须是变量，右边可以是常量、变量、函数调用或常量、变量、函数调用组成的表达式。

例如：x=10 y=x+10 y=func()都是合法的赋值表达式。

(2)赋值符号“=”不同于数学的等号，它没有相等的含义。（“==”相等）

例如：C 语言中 x=x+1 是合法的（数学上不合法），它的含义是取出变量 x 的值加 1，再存放到变量 x 中。

(3)赋值运算时，当赋值运算符两边数据类型不同时，将由系统自动进行类型转换。

转换原则是：先将赋值号右边表达式类型转换为左边变量的类型，然后赋值。

- 将实型数据（单、双精度）赋给整型变量，舍弃实数的小数部分。
- 将整型数据赋给单、双精度实型变量，数值不变，但以浮点数形式存储到变量中。
- 将 double 型数据赋给 float 型变量时，截取其前面 7 位有效数字，存放到 float 变量的存储单元中(32bits)。但应注意数值范围不能溢出。将 float 型数据赋给 double 型变量时，数值不变，有效位数扩展到 16 位(64bits)。
- 字符型数据赋给整型变量时，由于字符只占一个字节，而整型变量为 2 个字节，因此将字符数据（8bits）放到整型变量低 8 位中。有两种情况：

如果所使用的系统将字符处理为无符号的量或对 unsigned char 型变量赋值, 则将字符的 8 位放到整型变量的低 8 位, 高 8 位补 0。

如果所使用的系统将字符处理为带符号的量 (signed char) (如 Turbo C), 若字符最高位为 0, 则整型变量高 8 位补 0; 若字符最高位为 1, 则整型变量高 8 位全补 1。这称为符号扩展, 这样做的目的是使数值保持不变。

- 将一个 int, short, long 型数据赋给一个 char 型变量时, 只是将其低 8 位原封不动地送到 char 型变量 (即截断)。
- 将带符号的整型数据(int 型)赋给 long 型变量时, 要进行符号扩展。即, 将整型数的 16 位送到 long 型低 16 位中, 如果 int 型数值为正, 则 long 型变量的高 16 位补 0, 如果 int 型数值为负, 则 long 型变量的高 16 位补 1, 以保证数值不变。反之, 若将一个 long 型数据赋给一个 int 型变量, 只将 long 型数据中低 16 位原封不动地送到整型变量 (即截断)。
- 将 unsigned int 型数据赋给 long int 型变量时, 不存在符号扩展问题, 只要将高位补 0 即可。将一个 unsigned 类型数据赋给一个占字节相同的整型变量, 将 unsigned 型变量的内容原样送非 unsigned 型变量中, 但如果数据范围超过相应整数的范围, 则会出现数据错误。
- 将非 unsigned 型数据赋给长度相同的 unsigned 型变量, 也是原样照赋。

总之: 不同类型的整型数据间的赋值归根到底就是: 按照存储单元的存储形式直接传送。(由长型整数赋值给短型整数, 截断直接传送; 由短型整数赋值给长型整数, 低位直接传送, 高位根据低位整数的符号进行符号扩展)。

(4)C 语言的赋值符号“=”除了表示一个赋值操作外, 还是一个运算符, 也就是说赋值运算符完成赋值操作后, 整个赋值表达式还会产生一个所赋的值, 这个值还可以利用。

赋值表达式的求解过程是:

- (1)先计算赋值运算符右侧的“表达式”的值
- (2)将赋值运算符右侧“表达式”的值赋值给左侧的变量。
- (3)整个赋值表达式的值就是被赋值变量的值。

例 2.14: 分析 $x=y=z=3+5$ 这个表达式。

根据优先级: 原式 $\Leftrightarrow x=y=z=(3+5)$; 根据结合性 (从右向左):
 $\Leftrightarrow x=(y=(z=(3+5)))\Leftrightarrow x=(y=(z=3+5))$

$z=3+5$: 先计算 $3+5$, 得值 8 赋值给变量 z , z 的值为 8, $(z=3+5)$ 整个赋值表达式值为 8;

$y=(z=3+5)$: 将上面 $(z=3+5)$ 整个赋值表达式值 8 赋值给变量 y , y 的值为 8, $(y=(z=3+5))$ 整个赋值表达式值为 8;

$x=(y=(z=3+5))$: 将上面 $(y=(z=3+5))$ 整个赋值表达式值 8 赋值给变量 x , x 的值为 8, 整个表达式 $x=(y=(z=3+5))$ 的值为 8。

最后, x, y, z 都等于 8

运算步骤如下:

表 2.4 表达式运算步骤示例

序号	表达式	变量及值	表达式的值
1	$z=3+5$	$z (8)$	8
2	$y=(z=3+5)$	$y (8)$	8
3	$x=(y=(z=3+5))$	$x (8)$	8

将赋值表达式作为表达式的一种, 使赋值操作不仅可以出现在赋值语句中, 而且可以以表达式的形式出现在其它语句中。

2、复合赋值运算符

在赋值符 “=” 之前加上某些运算符, 可以构成复合赋值运算符, 复合赋值运算符可以构成赋值表达式。C 语言中许多双目运算符可以与赋值运算符一起构成复合运算符, 即:

$+=, -=, *=, /=, \%, <<=, >>=, \&=, |=, ^=$

复合赋值表达式一般形式: $\langle \text{变量} \rangle \langle \text{双目运算符} \rangle = \langle \text{表达式} \rangle$

等价于: $\langle \text{变量} \rangle = \langle \text{变量} \rangle \langle \text{双目运算符} \rangle \langle \text{表达式} \rangle$

例如:

$n+=1$ 等价于 $n=n+1$

$x*=y+1$ 等价于 $x=x*(y+1)$ 注意: 赋值运算符、复合赋值运算符的优先级比算术运算符低。

3、赋值运算符、赋值表达式举例

(1) $a=5$

(2) $a=b=5$

(3) $a=(b=4)+(c=3)$

(4) 假如 $a=12$, 分析: $a+=a-=a*a$

$a+=a-a*a \Leftrightarrow a+=a-(a*a) \Leftrightarrow a+=(a-(a*a)) \Leftrightarrow a+=(a-a-(a*a)) \Leftrightarrow a+=(a-a-a*a) \Leftrightarrow a+=a+(a-a-a*a)$

四、自增、自减运算符

单目运算符，使变量的值增 1 或减 1。如：

$++i, i++ \quad --i, i--$

【注意】

(1) $++i, --i$ (前置运算)：先自增、减，再参与运算； $i++, i--$ (后置运算)：先参与运算，再自增、减。

例如： $i=3$, 分析 $j=++i; j=i++$;

(2) 自增、减运算符只用于变量，而不能用于常量或表达式。

例如： $6++, (a+b)++, (-i)++$ 都不合法。

(3) $++, --$ 的结合方向是“自右向左” (与一般算术运算符不同)。

例如： $-i++ \Leftrightarrow -(i++)$ 合法。

(4) 自增、自减运算符常用于循环语句中，使循环变量自动加 1，也用于指针变量，使指针指向下一个地址。

【说明】

(1) C 运算符和表达式使用灵活，利用这一点可以巧妙处理许多在其它语言中难以处理的问题。但是 ANSI C 并没有具体规定表达式中的子表达式的求值顺序，允许各编译系统自己安排。这可能导致有些表达式对不同编译系统有不同的解释，并导致最终结果的不一致。

例 1： $a=f1()+f2()$ 中 $f1, f2$ 哪个先调用。

例 2： $i=3$, 表达式 $(i++)+(i++)+(i++)$ 的值。有些系统等价 $3+4+5$ ，Turbo C 等价 $3+3+3$

(2) C 语言有的运算符为一个字符，有的由两个字符组成，C 编译系统在处理时尽可能多地将若干字符组成一个运算符 (在处理标识符、关键字时也按同一原则处理)。如 $i+++j$ 将解释为 $(i++)+j$ 而不是 $i+(++j)$ 。为避免误解，最好采用大家都能理解的写法，比如通过增加括号明确组合关系，改善可读性。

(3) C 语言中类似的问题还有函数调用时，实参的求值顺序，C 标准也无统一规定。

如: `i=3,printf("%d,%d",i,i++);`有些系统执行的结果为 3, 3; 有些系统为 4, 3。

总之, 不要写别人看不懂(难看懂)、也不知道系统会怎样执行的程序。

五、逗号运算符和逗号表达式

C 语言提供一种特殊的运算符-逗号运算符(顺序求值运算符)。用它将两个或多个表达式连接起来, 表示顺序求值(顺序处理)。用逗号连接起来的表达式称为逗号表达式。

例如: `3+5,6+8`

逗号表达式的一般形式: 表达式 1, 表达式 2,...表达式 n

逗号表达式的求解过程是: 自左向右, 求解表达式 1, 求解表达式 2,...,求解表达式 n。整个逗号表达式的值是表达式 n 的值。

例如: 逗号表达式 `3+5,6+8` 的值为 14。

例 2.15 `a=3*5,a*4`

查运算符优先级表可知, “=” 运算符优先级高于 “,” 运算符(事实上,逗号运算符级别最低)。

所以上面的表达式等价于: `(a=3*5),(a*4)`

故整个表达式计算后值为: 60(其中 `a=15`)

例 2.16 程序题

```
main()
{
    int x,a;

    x=(a=3,6*3); /* a=3 x=18 */
    printf("%d,%d\n",a,x);
    x=a=3,6*a; /* a=3 x=3 */
    printf("%d,%d\n",a,x);
}

3,18
3,3
```

程序 2.10

逗号表达式主要用于将若干表达式“串联”起来，表示一个顺序的操作（计算），在许多情况下，使用逗号表达式的目的只是想分别得到各个表达式的值，而并非一定需要得到和使用整个逗号表达式的值。

第三章

顺序程序设计

【基本要求】

通过本单元的学习，使学生了解 C 语句的基本知识，熟悉顺序结构程序设计的概念，掌握数据输出的方法，掌握数据输入的方法，会设计简单程序。

【教学重点】

C 语句的构成与分类，C 中数据的输入/输出，C 顺序结构程序设计实例

【本章结构】

1、C 语句概述

2、赋值语句

3、字符数据的输入输出 { putChar 函数
getChar 函数

4、格式输入与输出 { putChar 函数
getChar 函数

5、顺序结构程序举例

程序设计时，通常采用三种不同的程序结构，即顺序结构、选择结构和循环结构。其中顺序结构是最基本、最简单的程序结构。通过本章顺序程序设计的學習，使大家可以开始最简单的 C 程序设计。

一、C 语句概述

C 语言的语句用来向计算机系统发出操作指令。一个语句经过编译后产生若干条机器指令。实际程序包含若干条语句。语句都是用来完成一定操作任务的。声明部分的内容不应当称为语句。函数包含声明部分和执行部分，执行部分由语句组成。

C 程序结构：一个 C 程序可以由若干个源程序文件组成，一个源文件可以由若干个函数和预处理命令以及全局变量声明部分组成，一个函数由数据定义部分和执行语句组成。程序包括数据描述（由声明部分来实现）和数据操作（由语句来实现）。数据描述主要定义数据结构（用数据类型表示）和数据初值。数据操作的任务是对已提供的数据进行加工。C 程序结构如图 3.1 所示。

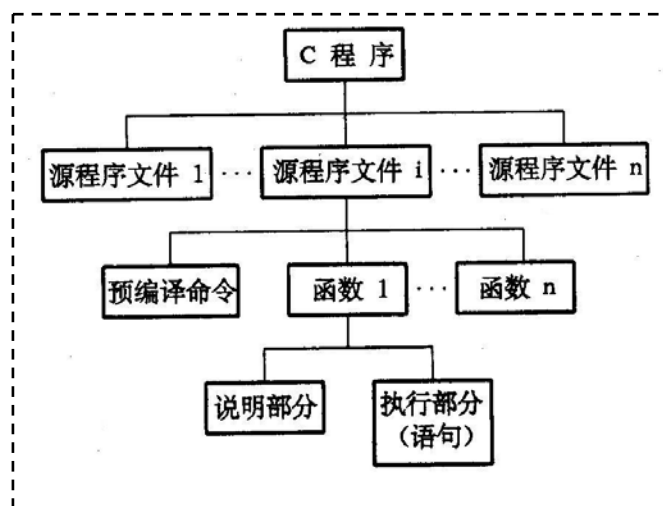


图 3.1 C 语言程序结构

C 语句可以分为以下 3 大类：

1、控制语句-完成一定控制功能的语句（主要用于控制程序流程）。

C 有 9 种控制语句，它们是：

① if () ~ else ~	(条件语句)
② for () ~	(循环语句)
③ while () ~	(循环语句)
④ do ~ while ()	(循环语句)
⑤ continue	(结束本次循环语句)
⑥ break	(中止执行 switch 或循环语句)
⑦ switch	(多分支选择语句)
⑧ goto	(转向语句)
⑨ return	(从函数返回语句)

2、表达式语句-用表达式构成语句，表示一个运算或操作。

表达式语句是在表达式最后加上一个“；”组成。一个表达式语句必须在最后出现分号，分号是表达式语句不可缺少的一部分。C 程序中大多数语句是表达式语句（包括函数调用语句）。

表达式语句常见的形式可以有：赋值语句、函数调用语句、空语句。

(1)赋值语句：由赋值表达式加上一个分号构成赋值语句。

C 语言的赋值语句先计算赋值运算符右边的子表达式的值，然后将此值赋值给赋值运算符左边的变量。C 语言的赋值语句具有其它高级语言的赋值语句的一切特点和功能。

(2)函数调用语句：由函数调用表达式加一个分号构成函数调用语句。

例如: `printf("This is a C statement.");`

(3)空语句: 只有一个分号的语句, 它什么也不做(表示这里可以有一个语句, 但是目前不需要做任何工作)。

例如:

空循环 100 次, 可能表示一个延时, 也可能表示目前还不必在循环体中做什么事情。

```
for(i=0;i<100;i++);
```

`/* 循环结构要求循环体, 但目前什么工作都不要做。; 表示循环体 */`

如果条件满足什么都不做, 否则完成某些工作。(; 表示 if 块, 什么都不做)

```
if()
```

```
;
```

```
else
```

```
{
```

```
.....
```

```
}
```

3、复合语句

用 {} 把一些语句 (语句序列, 表示一系列工作) 括起来成为复合语句, 又称语句块、分程序。

一般情况凡是允许出现语句的地方都允许使用复合语句。在程序结构上复合语句被看作一个整体的语句, 但是内部可能完成了一系列工作。

【注意】

C 语言允许一行写几个语句, 也允许一个语句拆开写在几行上, 书写格式无固定要求。一般将彼此关联的、或表示一个整体的一组较短的语句写在一行上。

二、输入/输出及其 C 语言的实现 (补充)

1、计算机由主机 (CPU、内存), 外围设备 (输入/输出设备), 接口组成。

2、输入/输出: 从计算机向外部设备 (如显示器、打印机、磁盘等) 输出数据称为 “输出”, 从外部设备 (如键盘、鼠标、扫描仪、光盘、磁盘) 向计

计算机输入数据称为“输入”。输入/输出是以计算机主机为主体而言的。如图 3.2 所示

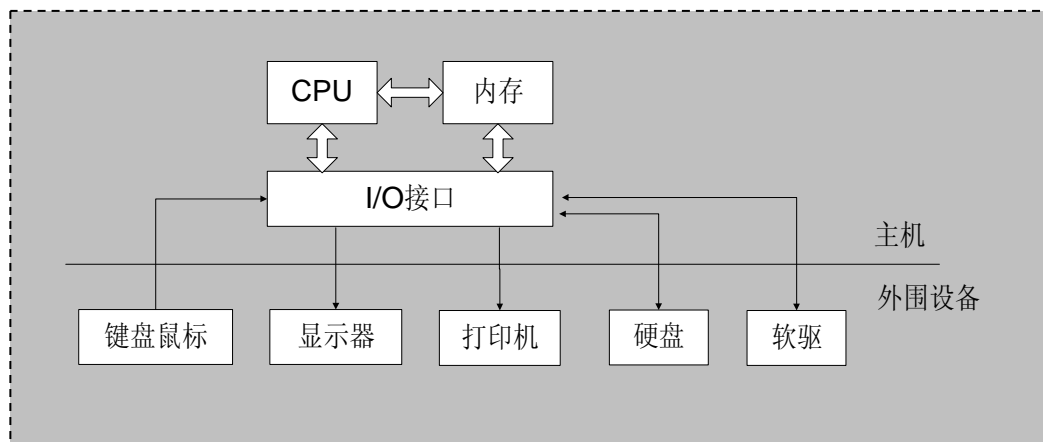


图 3.2 计算机系统结构

3、C 语言本身不提供输入/输出语句，输入/输出操作由函数实现。在 C 标准函数库中提供了一些输入/输出函数，如 printf 函数，scanf 函数。不要将两者看作是输入/输出语句。实际上完全可以不用这两个函数，而另外编制输入/输出函数。

C 编译系统与 C 函数库是分别设计的，因此不同的计算机系统所提供函数的数量、名字、功能不完全相同。但是，有些通用的函数各种计算机系统都提供，成为各种计算机系统的标准函数。

C 函数库中有一批“标准输入/输出函数”，它是以标准的输入/输出设备（一般为终端）为输入/输出对象的。其中有：putchar（输出字符）,getchar（输入字符）,printf（格式化输出）,scanf（格式化输入）,puts（输出字符串）,gets（输入字符串）。

4、在使用 C 库函数时，要用预编译命令“#include”将有关的“头文件”包含到用户源文件中。头文件包含库中函数说明，定义的常量等。每个库一般都有相应的头文件。

比如 printf 等函数属于标准输入/输出库，对应的头文件是 stdio.h。也就是说如果要使用 printf 等函数，应当在程序的开头#include <stdio.h>。又如 fabs 函数属于数学库，对应的头文件是 math.h，如果要使用 fabs 函数计算绝对值，那么应当在程序的开头#include <math.h>。

【注意】

(1)函数说明检查函数调用，进行数据类型转换，并产生正确的调用格式。许多编译系统强制要求函数说明（函数原型声明），否则编译不成功。

(2)Turbo C 中可以用 F1 查看一个函数的说明（包含属于哪个头文件）。

三、格式输入/输出

1、printf 函数（格式输出函数）

功能：按照用户指定的格式，向系统隐含的输出设备（终端）输出若干个任意类型的数据。

(1)printf 函数的一般格式：printf(格式控制字符串，输出表列)；如图 3.3 所示。

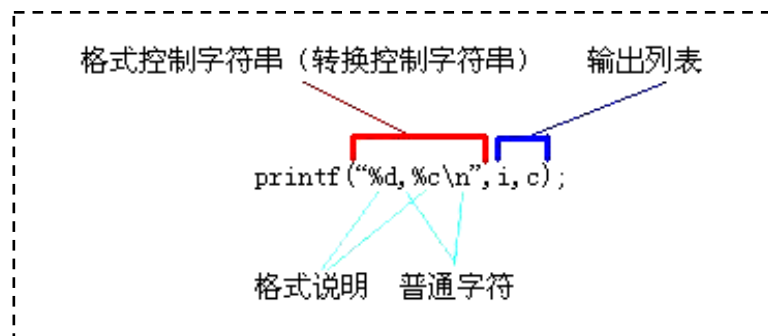


图 3.3 printf 函数格式

函数参数包括两部分：

① “格式控制”字符串是用双引号括起来的字符串，也称“转换控制字符串”，它指定输出数据项的类型和格式。

它包括两种信息：

- 格式说明项：由“%”和格式字符组成，如%d,%f等。格式说明总是由“%”字符开始，到格式字符终止。它的作用是将输出的数据项转换为指定的格式输出。输出表列中的每个数据项对应一个格式说明项。
- 普通字符：即需要原样输出的字符。例子中的逗号和换行符。

② “输出列表”是需要输出的一些数据项，可以是表达式。

例如：假如 $a=3, b=4$ ，那么 `printf("a=%d b=%d", a, b);` 输出 `a=3 b=4`。其中两个“%d”是格式说明，表示输出两个整数，分别对应变量 `a, b`，“a=”，“b=”是普通字符，原样输出。

由于 printf 是函数，因此“格式控制”字符串和“输出表列”实际上都是函数的参数。printf 函数的一般形式可以表示为：

`printf(参数 1、参数 2、参数 3、...参数 n)`

printf 函数的功能是将参数 2-参数 n 按照参数 1 给定的格式输出。

2、格式字符（构成格式说明项）

对于不同类型的数据项应当使用不同的格式字符构成的格式说明项。常用的有以下几种格式字符：（按不同类型数据，列出各种格式字符的常用用法）

(1)d 格式符。用来输出十进制整数。有以下几种用法：

- %d,按照数据的实际长度输出
- %md, m 指定输出字段的宽度（整数）。如果数据的位数小于 m，则左端补以空格（右对齐），若大于 m，则按照实际位数输出。
- %-md, m 指定输出字段的宽度（整数）。如果数据的位数小于 m，则右端补以空格（左对齐），若大于 m，则按照实际位数输出。
- %ld, 输出长整型数据，也可以指定宽度%mld。

(2)o 格式符。以八进制形式输出整数。注意是将内存单元中的各位的值按八进制形式输出，输出的数据不带符号，即将符号位也一起作为八进制的一部分输出。如：

```
int a=-1;
```

```
printf(“%d,%o,%x”,a,a,a);
```

-1 的原码：1000,0000,0000,0001。

-1 在内存中的补码表示为：

1111,1111,1111,1111=1,111,111,111,111,111=1,7,7,7,7=ffff

输出：-1,177777,ffff

-1 是十进制，177777 是八进制，ffff 是十六进制。

(3)x 格式符。以十六进制形式输出整数。与 o 格式一样，不出现负号。

(4)u 格式符。用来输出 unsigned 无符号型数据，即无符号数，以十进制形式输出。

一个有符号整数可以用%u 形式输出，反之，一个 unsigned 型数据也可以用%d 格式输出。

(5)c 格式符。用来输出一个字符。一个整数只要它的值在 0-255 范围内，也可以用字符形式输出。反之，一个字符数据也可以用整数形式输出。如

```
main()
```

```
{
```

```
    char c='a';
```

```
    int i=97;
```

```
    printf(“%c,%d\n”,c,c);
```

```
    printf(“%c,%d\n”,i,i);
```

```
}
```

运行结果：

a,97

a,97

也可以指定字段宽度。%mc,m-整数

(6)s 格式符。用来输出一个字符串。有几种用法:

- %s, 输出字符串
- %ms, 输出的字符串占 m 列, 如果字符串长度大于 m, 则字符串全部输出; 若字符串长度小于 m, 则左补空格 (右对齐)。
- %-ms, 输出的字符串占 m 列, 如果字符串长度大于 m, 则字符串全部输出; 若字符串长度小于 m, 则右补空格 (左对齐)。
- %m.ns, 输出占 m 列, 但只取字符串左端 n 个字符, 左补空白 (右对齐)。
- %-m.ns, 输出占 m 列, 但只取字符串左端 n 个字符, 右补空白 (左对齐)。

(7)f 格式符。用来输出实数 (包括单、双精度, 单双精度格式符相同), 以小数形式输出。有以下几种用法:

- %f, 不指定宽度, 使整数部分全部输出, 并输出 6 位小数。注意, 并非全部数字都是有效数字, 单精度实数的有效位数一般为 7 位 (双精度 16 位)。如图 3.4 所示。

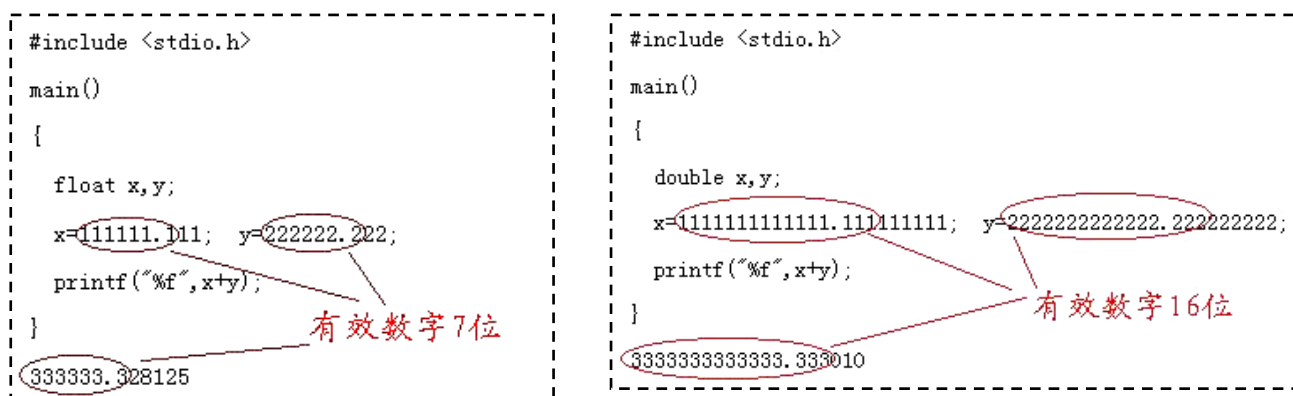


图 3.4 输出精度示意图

- %m.nf, 指定数据占 m 列, 其中有 n 位小数。如果数值长度小于 m, 左端补空格 (右对齐)。
- %-m.nf, 指定数据占 m 列, 其中有 n 位小数。如果数值长度小于 m, 右端补空格 (左对齐)。

(8)e 格式符, 以指数形式输出实数。可用以下形式:

- %e, 不指定输出数据所占的宽度和小数位数, 由系统自动指定, 如 6 位小数, 指数占 5 位-e 占 1 位, 指数符号占 1 位, 指数占 3 位。数值按照规格化指数形式输出 (小数点前必须有而且只有 1 位非 0 数字)。

例如: 1.234567e+002。 (双精度)

- %m.ne 和 %-m.ne, m 总的宽度, n 小数位数。

(9)g 格式符，用来输出实数，它根据数值的大小，自动选 f 格式或 e 格式（选择输出时占宽度较小的一种），且不输出无意义的 0（小数末尾 0）。如：

```
#include <stdio.h>
main()
{
    float f=123.0;
    printf("%f,%e,%g\n",f,f,f);
}
123.000000,1.23000e+02,123
```

以上介绍的 9 种格式符，归纳如图 3.5 所示：

格式字符	说 明
d	以带符号的十进制形式输出整数(正数不输出符号)。
o	以 8 进制无符号形式输出整数(不输出前导符 0)。
x	以 16 进制无符号形式输出整数(不输出前导符 0x)。
u	以无符号 10 进制形式输出整数。
c	以字符形式输出，只输出一个字符。
s	输出字符串。
f	以小数形式输出单、双精度数，隐含输出 6 位小数。
e	以标准指数形式输出单、双精度数，数字部分小数位数为 6 位。
g	选用%f 或 %e 格式中输出宽度较短的一种格式，不输出无意义的 0。

在格式说明中，在 % 和上述格式字符间可以插入以下几种附加符号。

字符	说 明
字母 l	用于长整型整型，可加在格式符 d、o、x、u 前面。
m(代表一个正整数)	数据最小宽度。
.n(代表一个正整数)	对实数，表示输出 n 位小数；对字符串，表示截取的字符个数。
-	输出的数字或字符在域内向左靠。

图 3.5 格式符说明

Turbo C 中 printf 函数格式字符串的一般形式：

%	±	m	.	n	h/l	格式字符
[开始符]	[标志字符]	[宽度指示符]		[精度指示符]	[长度修正符]	[格式转换符]

图 3.6 printf 函数格式字符串形式

3、使用 printf 函数的几点说明：

- 除了 X,E,G 外，其它格式字符必须用小写字母。如 %d 不能写成 %D。
- 可以在“格式控制”字符串中包含转义字符。如 “...\n...”
- 格式符以 % 开头，以上述 9 个格式字符结束。中间可以插入附加格式字符。

- 如果想输出字符%，则应当在“格式控制”字符串中用两个%表示。

4、scanf 函数（格式输入函数）

(1)scanf 函数的一般格式：scanf(格式控制字符串，地址列表)

其中：

- 格式控制字符串的含义与 printf 类似，它指定输入数据项的类型和格式。
- 地址列表是由若干个地址组成的列表，可以是变量的地址（&变量名）或字符串的首地址。

例如，用 scanf 函数输入数据。

```
main()
{
    int a,b,c;
    scanf("%d%d%d",&a,&b,&c);
    printf("%d,%d,%d\n",a,b,c);
}
```

- &是地址运算符，&a 指变量 a 的地址。scanf 的作用是将键盘输入的数据保存到&a,&b,&c 为地址的存储单元中，即变量 a,b,c 中。
- %d%d%d 表示要求输入 3 个十进制整数。输入数据时，在两个数据之间以一个或多个空格分隔，也可以用回车键，跳格键(tab)分隔。这种格式不能用逗号分隔数据。

例如，合法的输入：

- 3 4 5
- 3
4 5
- 3（按 tab 键）4
5

非法的输入：3,4,5

(2)格式说明

与 printf 函数中的格式说明相似，以%开始，以一个格式字符结束，中间可以插入附加字符。

【说明】

- 对 unsigned 型变量所需的数据，可以用%u,%d 或%o,%x 格式输入。
- 可以指定输入数据所占列数，系统自动按它截取所需数据。

如：

```
int i1,i2;
```

char c;

scanf("%3d%3c%3d",&i1,&c,&i2);

输入: 123---456 后, i1=123,i2=456,c='-'

- 如果%后有“*”附加格式说明符,表示跳过它指定的列数,这些列不赋值给任何变量。

如:

scanf("%3d%*3c%2d",&i1,&i2);

输入: 123456789 后, i1=123,i2=78,(456被跳过)

在利用现有的一批数据,有时不需要其中某些数据,可以用此方法“跳过”它们。

- 输入数据时可以指定数据字段的宽度,不能规定数据的精度。
- 例如, scanf("%7.2f",&a);是不合法的。不能指望使用这种形式通过输入 1234567 获得 a=12345.67。

(3)使用 scanf 函数应当注意的问题

①scanf 函数中“格式控制”后面应当是变量地址,而不应是变量名。

例如: scanf("%d,%d",a,b);不合法。(原因: C 是传值调用,不能由形参返回值)

如果在“格式控制”字符串中除了格式说明以外还有其它字符,则在输入数据时在对应位置应当输入与这些字符相同的字符。建议不要使用其它的字符。

如:

scanf("%d,%d,%d",&a,&b,&c); 应当输入 3,4,5; 不能输入 3 4 5。

scanf("%d:%d:%d",&h,&m,&s); 应当输入 12:23:36

scanf("a=%d,b=%d,c=%d",&a,&b,&c); 应当输入 a=12,b=24,c=36(太罗嗦)

②在用“%c”格式输入字符时,空格字符和转义字符都作为有效字符输入。%c 只要求读入一个字符,后面不需要用空格作为两个字符的间隔。

对于 scanf("%c%c%c",&c1,&c2,&c3);

输入: a b c<CR>后, c1='a',c2=' ',c3='b'

③在输入数据时,遇到下面情况认为该数据结束:

- 遇到空格,或按“回车”或“跳格”(tab)键。

如:

int a,b,c;

scanf("%d%d%d",&a,&b,&c);

输入: 12 34 (tab) 567<CR>后, a=12,b=34,c=567

- 按指定的宽度结束
- 遇到非法的输入

如:

float a,c; char b;

```
scanf("%d%c%f",&a,&b,&c);
```

输入： 1234a123o.26<CR> 后， a=1234.0,b='a',c=123.0(而不是希望的1230.26)

C 语言的格式输入输出的规定比较繁琐，重点掌握最常用的一些规则和规律即可，其它部分可在需要时随时查阅。

四、其它输入/输出函数 (#include <stdio.h>)

1、 putchar 函数 (字符输出函数)

一般形式： putchar(字符表达式);

功能：向终端（显示器）输出一个字符（可以是可显示的字符，也可以是控制字符或其它转义字符）。

例如：

```
putchar('y'); putchar('\n'); putchar("\101"); putchar('\\');
```

2、 getchar 函数 (字符输入函数)

一般形式： c=getchar();

功能：从终端（键盘）输入一个字符，以回车键确认。函数的返回值就是输入的字符。

3、 puts 函数 (字符串、字符数组中字符串输出函数)

一般形式： puts(char *str);

功能：将字符串或字符数组中存放的字符串输出到显示器上。

例如： putstr("China\nBeijing\n");

4、 gets 函数 (字符串输入函数)

一般形式： gets(char *str);

功能：接收从键盘输入的一个字符串，存放在字符数组中。

例如：

```
char s[81];
```

```
gets(s);
```

第四章

选择结构程序设计

【基本要求】

通过本单元的学习，使学生了解关系运算符及表达式及逻辑运算符及表达式的基础知识，掌握条件运算符及表达式的应用，熟练掌握单条件选择 if 语句的形式及具体应用，掌握开关分支 switch 语句。

【教学重点】

关系运算和逻辑运算及其规则，if 语句及其几种形式的应用，switch 语句及其应用，选择结构程序设计实例。

【本章结构】

- 1、关系运算符和关系表达式 { 关系运算符及其优先次序
关系表达式
- 2、逻辑运算符和逻辑表达式 { 逻辑运算符及其优先次序
逻辑表达式
- 3、if 语句 { if 语句的 3 种形式
if 语句的嵌套
条件运算符
- 4、switch 语句
- 5、程序举例

第一部分 关系与逻辑

选择结构是三种基本结构（顺序、选择、循环）之一。作用是根据所指定的条件是否满足，决定从给定的两组操作选择其中的一种。

C 语言中的选择结构是用 if 语句实现的。if 语句的常用的形式是：

if(关系/逻辑表达式)语句 1 else 语句 2

例 4、1：用 C 语言实现图示的分段函数。

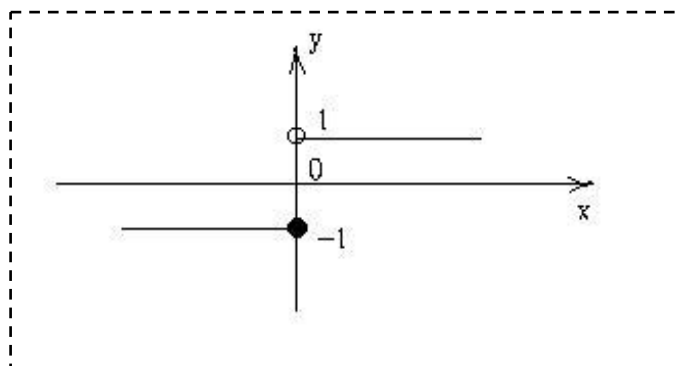


图 4.1 分段函数表示

数学上描述此分段函数：

$$y = \begin{cases} 1, & x > 0 \\ -1, & \text{其它} \end{cases}$$

用 C 语言描述：

```
if(x>0)y=1;else y=-1;
```

其中： $x > 0$ 是一个关系表达式，“ $>$ ”是一个关系运算符。 $x > 0$ 关系表达式成立，也就是说由关系表达式 $x > 0$ 构成的逻辑运算为真时， $y = 1$ ，否则 $y = -1$ 。

一、关系运算符和关系表达式

关系运算是逻辑运算中比较简单的一种，“关系运算”就是“比较运算”。即，将两个值进行比较，判断是否符合或满足给定的条件。如果符合或满足给定的条件，则称关系运算的结果为“真”；如果不符合或不满足给定的条件，则称关系运算的结果为“假”。

例 4、1 中， $x > 0$ 是比较运算，也就是关系运算，“ $>$ ”是一种关系运算符。

假如 $x = 4$ ，那么 $x > 0$ 条件满足，就是说关系运算 $x > 0$ 的结果为“真”。

1、关系运算符及其优先次序

C 语言提供 6 种关系运算符：如图 4.2 所示

关于优先次序：

- 前 4 种关系运算符的优先级别相同，后两种也相同。前 4 种高于后两种。

- 关系运算符的优先级低于算术运算符。
- 关系运算符的优先级高于赋值运算符。

①	<	(小于)	} 优先级相同 (高)
②	<=	(小于或等于)	
③	>	(大于)	
④	>=	(大于或等于)	
⑤	=	(等于)	} 优先级相同 (低)
⑥	!=	(不等于)	

图 4.2 C 语言的 6 种关系运算符

例 4、2:

$c > a + b$ 等价于 $c > (a + b)$; 关系运算符的优先级低于算术运算符
 $b == c$ 等价于 $(a > b) == c$; “>” 优先级高于 “==”
 $a == b < c$ 等价于 $a == (b < c)$; “<” 优先级高于 “==”
 $a = b > c$ 等价于 $a = (b > c)$; 关系运算符的优先级高于赋值运算符。

2、关系表达式

用关系运算符将两个表达式 (算术、关系、逻辑、赋值表达式等) 连接起来所构成的表达式，称为关系表达式。

关系表达式的值是一个逻辑值，即“真”或“假”。C 语言没有逻辑型数据，以 1 代表“真”，以 0 代表“假”。

例 4、3: 假如 $a=3, b=2, c=1$ ，则:

关系表达式 “ $a > b$ ” 的值为“真”，即表达式的值为: 1。

关系表达式 “ $b + c < a$ ” 的值为“假”，即表达式的值为: 0。

二、逻辑运算符和逻辑表达式

逻辑表达式: 用逻辑运算符 (逻辑与、逻辑或、逻辑非) 将关系表达式或逻辑量连接起来构成逻辑表达式。

1、逻辑运算符及其优先顺序

C 语言提供三种逻辑运算符:

(1) && 逻辑与 (相当日常生活中: 而且, 并且, 只在两条件同时成立时为“真”)

(2) || 逻辑或 (相当日常生活中: 或, 两个条件只要有一个成立时即为“真”)

(3) ! 逻辑非（条件为真，运算后为假，条件为假，运算后为真）

“&&”，“||”是双目运算符，“!”是单目运算符。

例 4、4: 逻辑运算举例

a&&b 若 a、b 为真，则 a&&b 为真。

a||b 若 a、b 之一为真，则 a||b 为真。

! a 若 a 为真，则 ! a 为假，反之若 a 为假，则 ! a 为真。

表 4.1 为逻辑运算的真值表

a	b	a&&b	a b
真	真	真	真
真	假	假	真
假	真	假	真
假	假	假	假

a	! a
真	假
假	真

在一个逻辑表达式中如果包含多个逻辑运算符，则按照以下的优先顺序：

(1) ! （非）-&&（与）-||（或），“!”为三者中最高。

(2) 逻辑运算符中的&&和||低于关系运算符，! 高于算术运算符。

例 4、5:

b&& x>y 等价于 (a>b) && (x>y)

a==b||x==y 等价于 (a==b)|| (x==y)

! a||a>b 等价于 (! a)|| (a>b)

2、逻辑表达式

逻辑表达式：用逻辑运算符（逻辑与、逻辑或、逻辑非）将关系表达式或逻辑量连接起来构成逻辑表达式。

逻辑表达式的值是一个逻辑量“真”或“假”。C 语言编译系统在给出逻辑运算结果时，以 1 代表“真”，以 0 代表“假”，但在判断一个量是否为“真”时，以 0 代表“假”，以非 0 代表“真”（即认为一个非 0 的数值是“真”）。

例 4、6: 非 0 值作为逻辑值参与运算=“真”（此时与 1 的作用一样）

若 $a=4$, 则 $!a=0$ (假)。

若 $a=4, b=5$, 则 $a\&b=1$ (真), $a||b=1$ (真), $!a||b=1$ (真)

$4\&\&0||2=1$ (真)

$'c'(\text{真})\&\&'d'(\text{真})=1$

从例子还可以看出：系统给出的逻辑运算结果不是 0 就是 1，不可能是其它数值。而在逻辑表达式中作为参与逻辑运算的运算对象可以是 0 (作为“假”)也可以是非 0 的数值 (按“真”对待)。事实上，逻辑运算符两侧的对象不但可以是 0 和 1 或者是 0 和非 0 的整数，也可以是任何类型的数据 (如字符型、实型、指针型)。

如果在一个表达式中不同位置上出现数值，应区分哪些是作为数值运算或关系运算的对象 (原值)，哪些是作为逻辑运算的对象 (逻辑值)。

例 4、7: 计算: $5>3\&\&2||8<4-!0$ (注意运算符优先级、数值所起作用-是逻辑值, 原值), 如图 4.3 所示

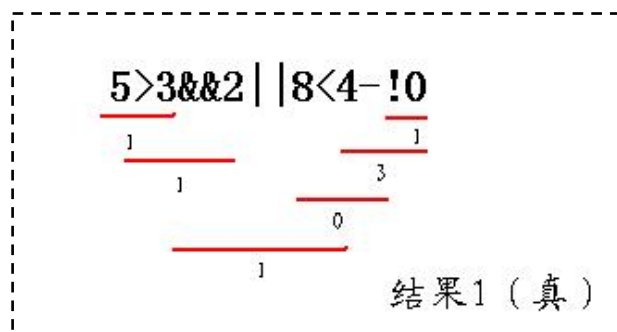


图 4.3 逻辑运算示例

在逻辑表达式的求解中，并不是所有的逻辑运算符都被执行，只是在必须执行下一个逻辑运算符才能求出表达式的解时，才执行该运算符。

例如:

$a\&\&b\&\&c$, 只有 a 为真，才需要判别 b 的值; 只有 a 、 b 都为真，才需要判别 c 的值。只要 a 为假，此时整个表达式已经确定为假，就不必判别 b, c ; 如果 a 为真， b 为假，不判断 c 。

$a||b||c$, 只要 a 为真，整个表达式已经确定为真，就不必判断 b 和 c ; 只有 a 为假，才判断 b ; a 、 b 都为假才判断 c 。

如果 a, b, c, d, m, n 分别为: 1, 2, 3, 4, 1, 1, 分析整个表达式 $(m=a>b)\&\&(n=c>d)$ 结果和 m, n 的结果。由于“ $a>b$ ”为假 (0)，所以赋值后 $m=0$ ，赋值表达式 $m=a>b$ 也为 0。此时整个表达式的结果已经知道 (0)，所以不进行表达式 $n=c>d$ 的计算，所以表达式计算机结束后， $n=1$ (未改变)。

掌握 C 语言的关系运算符和逻辑运算符后，可以用一个逻辑表达式来表示一个复杂的条件。

例如：判断某一年是否闰年。（闰年的条件是符合下面两个条件之一：1、能被 4 整除，但不能被 100 整除；2、能被 4 整除，又能被 400 整除）。因为能够被 400 整除一定能被 4 整除所以第二个条件可以简化为能够被 400 整除。判断闰年的条件可以用一个逻辑表达式表示：

```
(year%4==0&&year%100!=0)||year%400==0
```

表达式为“真”，闰年条件成立，是闰年，否则非闰年。

第二部分 If 语句及条件运算符

if 语句用来判定所给定的条件是否满足，根据判定的结果（真或假）决定执行给出的两种操作之一。

1、if 语句三种形式(如图 4.4 所示)

(1) if(表达式)语句

例如：if(x>y)printf(“%d”,x);

(2) if(表达式)语句 1 else 语句 2

例如：if(x>y)printf(“%d”,x);else printf(“%d”,y);

(3) if(表达式 1)语句 1

else if(表达式 2)语句 2

else if(表达式 3)语句 3

...

else if(表达式 m)语句 m

else 语句 m+1

(实际是 else 子句中嵌套 if 语句)

例如：

```
if(number>500) cost=0.15;
else if(number>300)cost=0.10;
else if(number>100)cost=0.075;
else if(number>50) cost=0.05;
else cost=0;
```

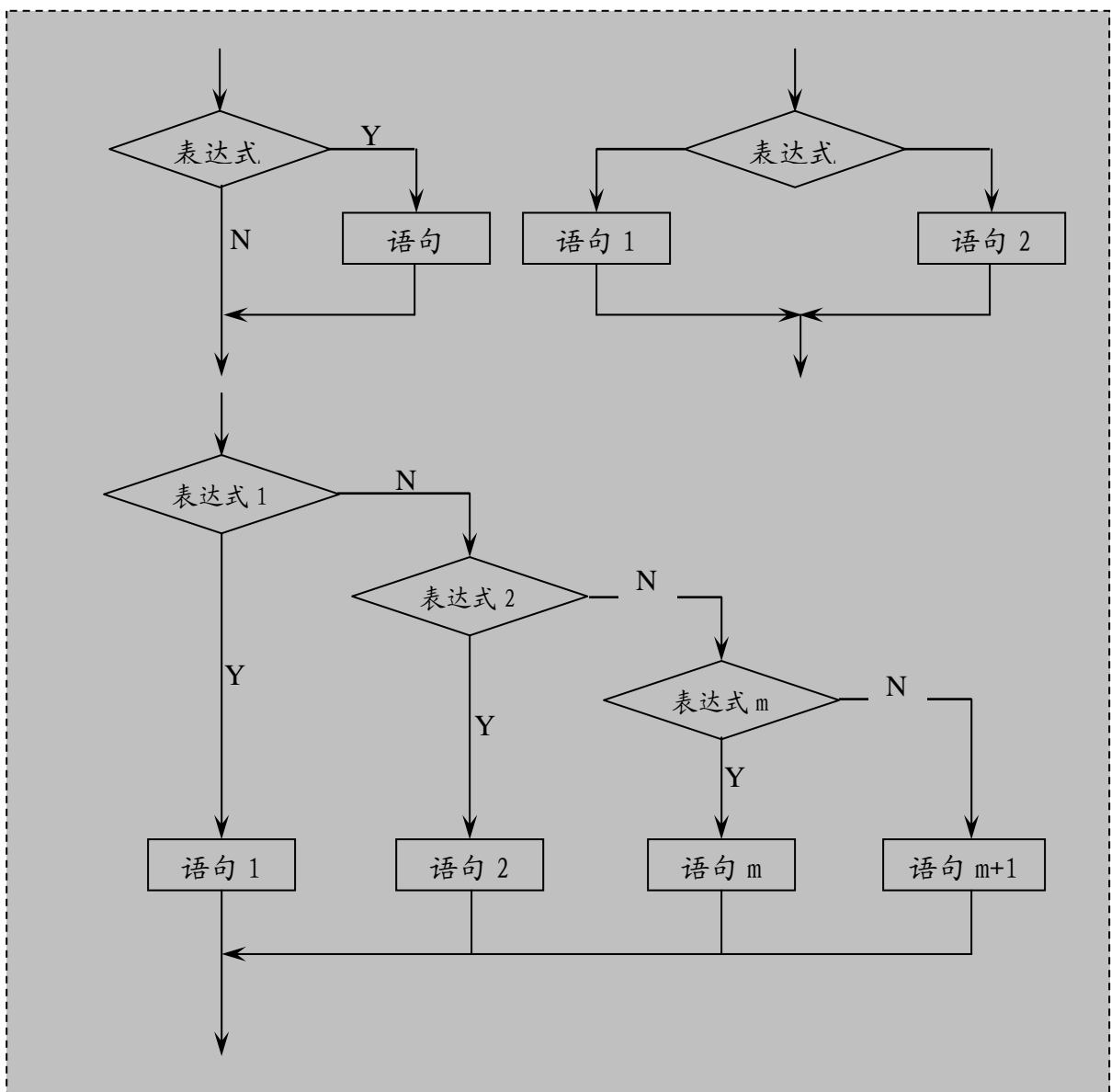


图 4.4 if 语句三种结构图

【说明】

(1) 三种形式的 if 语句中的“表达式”一般为关系表达式或逻辑表达式，但不限于这两种表达式。记住：C 语言中需要逻辑值的地方，只有 0 代表“假”，非 0（其它）均代表“真”。

例如：if('a')printf("%d",'a');- printf(...) 这条语句一定运行，因为'a'=97，即“真”。

例如：P198 例 11-13 倒数 12 行，bit=(mask&n)?1:0。mask,n 都是整数，&（位与）后也是整数。

(2) else 子句不能单独使用，必须是 if 语句的一部分，与 if 配对使用。

(3) 在 if 和 else 后面可以只含一个内嵌的操作语句，也可以有多个操作语句构成的语句块（复合语句）。语句块用{}括起来,语句块后面不要“;”号。

例如:

```
if(a+b>c&&b+c>a&&c+a>b){ s=0.5*(a+b+c); area=sqrt(s*(s-a)*(s-b)*(s-c));  
    printf(area);}  
else printf("it is not a triangle")
```

例 4.8: 输入两个实数，按数值由小到大的次序输出这两个数。(难点：交换数据算法)

```
main()  
{  
    float a,b,t; /* t-临时变量 */  
    scanf(a,b);  
    if(b<a){t=a;a=b;b=t;} /* 交换 a,b */  
    printf(a,b);  
}
```

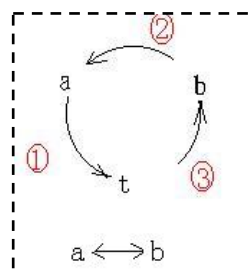


图 4.5 实数交换

例 4、9: 输入三个数 a,b,c,按照由小到大的顺序输出。

算法: (伪代码-三个数的选择法排序)

if b<a 将 a,b 交换 (a 是 a,b 中的小者)

if c<a 将 a,c 交换 (a 是 a,c 中的小者, 因此 a 是三者中的最小者)

if c<b 将 b,c 交换 (在剩下的两个数 b,c 中选次小数, 存放在 b 中)

输出 a,b,c。如程序 4.1 所示

```
main()  
{  
    float a,b,c,t; /* t-临时变量 */  
    scanf(a,b,c);  
    if(b<a){t=a;a=b;b=t;} /* 交换 a,b */  
    if(c<a){t=a;a=c;c=t;} /* 交换 a,b */  
    if(c<b){t=b;b=c;c=t;} /* 交换 a,b */  
    printf(a,b,c);  
}
```

程序 4.1 例 4.9 程序示例

2、if 语句的嵌套

if 语句的嵌套：if 语句的 if 块或 else 块中，由包含一个 if 语句。一般形式：

```
if(...)
    if(...)语句 1;
    else 语句 2;
else
    if(...)语句 3;
    else 语句 4;
```

【注意】应当注意 if 与 else 的配对关系。else 总是与它上面的最近的未配对的 if 配对。特别是 if/else 子句数目不一样时（if 子句数量只会大于或等于 else 子句数量）。如果一个 ELSE 的上面又有一个未配对的 ELSE，则先处理上面（内层）的 ELSE 的配对；直到把所有 ELSE 用完为止。

可以用下面两种方法解决匹配问题：

- （1）利用“空语句”，使 if 子句数量与 else 子句数量相同。
- （2）利用 {} 确定配对关系。将没有 else 子句的 if 语句用 {} 括起来。

例 4.10：函数：

$$y = \begin{cases} -1 & (x < 0) \\ 0 & (x = 0) \\ 1 & (x > 0) \end{cases}$$

编一个程序，输入 x，输出 y。

算法 1：

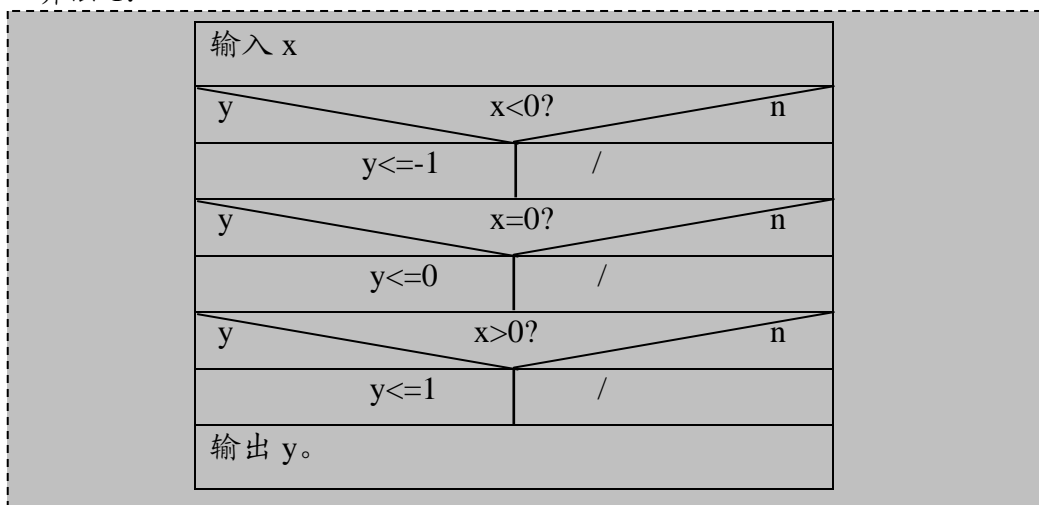


图 4.6 例 4.10 算法 1 N-S 图

```
main()
{
    int x,y;
    scanf(&x);
    if(x<0)y=-1;
    if(x=0)y=0;
    if(x>0)y=1;
    printf(x,y);
}
```

程序 4.2 例 4.10 算法 1 程序

算法 2:

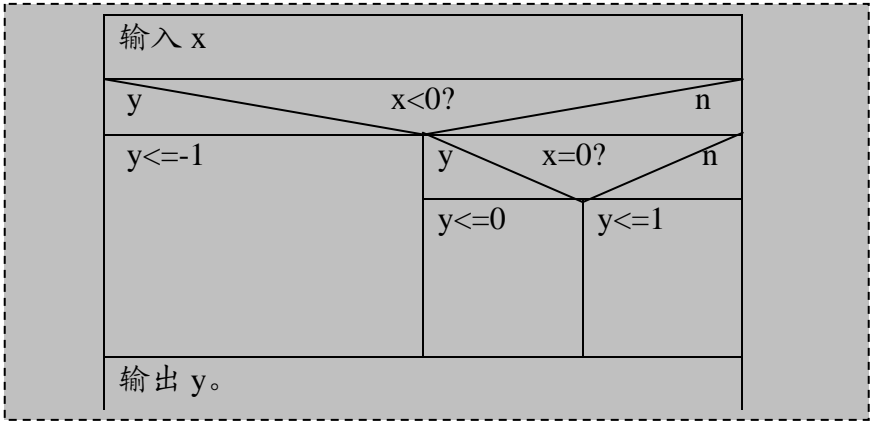


图 4.7 例 4.10 算法 2N-S 图

```
main()
{
    int x,y;
    scanf(&x);
    if(x<0)y=-1;
    else if(x=0)y=0;
    else y=1;
    printf(x,y);
}
```

程序 4.3 例 4.10 算法 2 程序

3、条件运算符 (? :)

在 if 语句中，在表达式为“真”和“假”时，都只执行一个赋值语句给同一个变量赋值时，可以使用简单的条件运算符来处理。

例如:

```
if(a>b)max=a;
else max=b;
```


可以使用条件运算符来处理：max=a>b?a:b；（注：a>b 关系运算有没有（）都一样）。它这样执行，a>b 为“真”吗？如果为“真”，条件表达式为 a，否则为 b。

条件表达式的一般形式：

表达式 1？表达式 2：表达式 3

执行过程如图 4.8 所示：

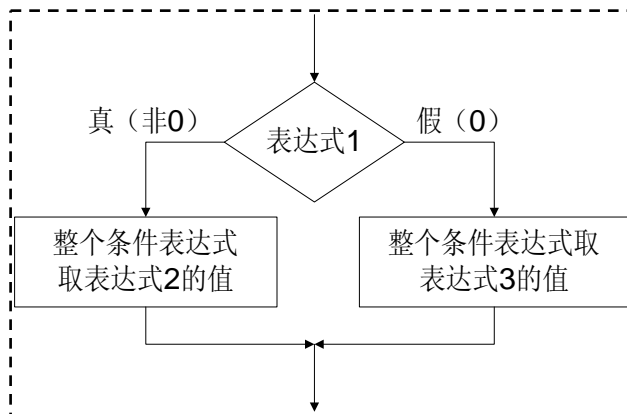


图 4.8 条件运算符执行过程

【说明】

(1)条件运算符的执行顺序：先求解表达式 1，若为非 0（真）则求解表达式 2，表达式 2 的值就是整个条件表达式的值。若表达式 1 的值为 0（假），则求解表达式 3，此时表达式 3 的值就是整个条件表达式的值。

(2)条件运算符的优先级高于赋值运算符，低于关系运算符和算术运算符。

例如：max=a>b?a:b 等价于：max=((a>b)?a:b)

(3)条件运算符的结合性“自右向左”。

例如：a>b?a:c>d?c:d。

先考虑优先级、再考虑结合性：上面表达式等价与：(a>b)?a:((c>d)?c:d)。

(4)表达式 2 和表达式 3 不仅可以是数值表达式，还可以是赋值表达式，函数表达式。

例如：

a>b?(a=100):(b=100) ；

a>b?printf(“%d”,a): printf(“%d”,a) ；

(5)表达式 1, 表达式 2, 表达式 3 的类型都可以不同。条件表达式值的类型是表达式 2, 表达式 3 中类型较高的类型。

例如: $x > y ? 1 : 1.5$ 整个表达式类型为实型。

例 4.11:输入一个字符, 如果是大写字母, 转换为小写, 如果不是不转换。最后输出。如程序 4.4 所示

```
main()
{
    char ch;
    scanf("%c", &ch);
    ch = (ch >= 'A' && ch <= 'Z') ? (ch + 32) : ch;
    printf("%c", ch);
}
```

表达式 $ch = (ch >= 'A' \&\& ch <= 'Z') ? (ch + 32) : ch$ 中的 () 括号都可以不要。
有 () 括号看上去更加清楚些。

程序 4.4 例 4.11 程序

第三部分 Switch 语句及其应用

一、switch 语句

多分支可以使用嵌套的 if 语句处理, 但如果分支较多, 嵌套的 if 语句层数多, 程序冗长, 降低可读性。

C 语言中 switch 语句是多分支选择语句。其一般形式:

```
switch(表达式)
{
    case 常量表达式 1: 语句 1
    case 常量表达式 2: 语句 2
    ... ..
    case 常量表达式 n: 语句 n
    [default: 语句 n+1]
}
```

switch 语句 case 中语句的终止可以使用 break.

【说明】

(1)switch 括号后面的表达式, 允许为任何类型。

(2)当“表达式”的值与某个 case 后面的常量表达式的值相等时，就执行此 case 后面的语句。如果表达式的值与所有常量表达式都不匹配，就执行 default 后面的语句(如果没有 default 就执行跳出 switch,执行 switch 语句后面的语句)。

(3)各个常量表达式的值必须互不相同，否则出现矛盾。

(4)各个 case，default 出现的顺序不影响执行结果。

(5)执行完一个 case 后面的语句后，流程控制转移到下一个 case 中的语句继续执行。此时，“case 常量表达式”只是起到语句标号的作用，并不在此处进行条件判断。在执行一个分支后，可以使用 break 语句使流程跳出 switch 结构，即终止 switch 语句的执行（最后一个分支可以不用 break 语句）。

(6)case 后面如果有多条语句，不必用{}括起来。

(7)多个 case 可以共用一组执行语句。（注意 break 使用的位置）

例子 4.12 求一元二次方程 $ax^2+bx+c=0$ 的根。

例 4.10 介绍过基本的算法，事实上应该有以下几种可能：

(1) $a=0$,不是二次方程。

(2) $b^2-4ac=0$,有两个相等的实根

(3) $b^2-4ac>0$,有两个不等的实根

(4) $b^2-4ac<0$,有两个共轭复数根

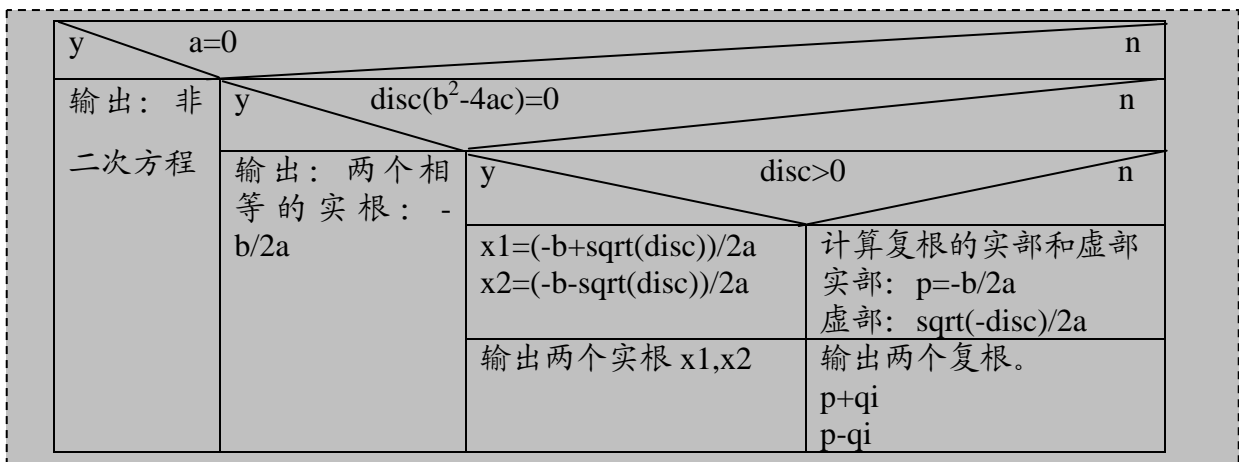


图 4.9 例 4.12 N-S 图

```

#include <math.h>
#define FLOATZERO 1e-6
main()
{
    float a,b,c,d,disc,x1,x2,realpart,imagpart;
    scanf(&a,&b,&c);
    printf("The equation ");

    if(fabs(a)<=FLOATZERO)                /* a=0 */
        printf("is not a quadratic");
    else
    {
        disc=b*b-4*a*c;                  /* 计算 disc */
        if(fabs(disc)<= FLOATZERO)        /* disc=0 */
            printf("has two equal root:%f\n",-b/(2*a));
        else if(disc> FLOATZERO)         /* disc>0 */
        {
            x1=(-b+sqrt(disc))/(2*a);
            x2=(-b-sqrt(disc))/(2*a);
            printf("has distinct real roots:%f,%f\n",x1,x2);
        }
        else                             /* disc<0 */
        {
            realpart=-b/(2*a);
            imagpart=sqrt(-disc)/(2*a);
            printf("has complex roots:\n");
            printf("%f+%fi\n",realpart,imagpart);
            printf("%f-%fi\n",realpart,imagpart);
        }
    }
}

```

程序 4.5 例 4.12 程序

【注意】 判断 δ (disc) 为 0，不能直接用 `disc==0`。

disc 是实数，实数在计算机中的存储和计算有微小的误差。“==”（等于运算符）是精确按位比较。如果使用 `disc==0` 比较，这可能导致原来为 0 的量由于上述误差而被判别为不等于 0，导致程序错误。

可以通过判断 disc 的绝对值是否小于一个很小的实数，小于此数可以认为等于 0。同理，如果需要判断两个浮点数相等也不能使用“==”运算符，而是采用判断两数之差的绝对值小于一个很小的数。

例 4.13 运输公司对用户计算运费。

路程 s 越远，每公里运费越低。标准如表 4.2 所示：

表 4.2 运输公司运费标准

s	c=(int)(s/250)	d
$s < 250\text{km}$	0	没有折扣
$250 \leq s < 500$	1	2%
$500 \leq s < 1000$	2, 3	5%
$1000 \leq s < 2000$	4, 5, 6, 7	8%
$2000 \leq s < 3000$	8, 9, 10, 11	10%
$s \geq 3000$	12, 13...	15%

设每公里，每吨货物的基本运费为 p (p (price 缩写))，货物重量为 w (w (weight)，距离为 s ，折扣为 d (discount)，则总运费 f (f (freight 的缩写))的计算公式为：

$$f = p * w * s * (1 - d)$$

```
main()
{
    int c,s;
    float p,w,d,f;
    scanf(&p,&w,&s);

    c=s/250;
    switch( c )
    {
        case 0: d=0;break;
        case 1: d=2;break;
        case 2:
        case 3: d=5;break;
        case 4:
        case 5:
        case 6:
        case 7: d=8;break;
        case 8:
        case 9:
        case 10:
        case 11:d=10;break;
        default: d=15;
    }
    f=p*w*s*(1-d/100.0);

    printf(f);
}
```

p-每公里、每吨基本运费(可以定义为常量)

这里可以没有 break 语句

图 4.10 例 4.13 程序

二、程序举例(实例分析)

【实例分析一】从键盘上输入三个整数，找出其中的我最大值并输出。

/*简单的if语句的使用。*/

```
int max1(int x,int y,int z)
{
    int max=x; if(max<y) max=y;
    if(max<z) max=z; return(max);
}
```

/*使用简单的if...else... 语句*/

```
int max2(int x,int y,int z)
{
    int max;
    if(x>y) if(x>z) max=x; /* x>y  && x>z */
    else max=z; /* z>=x  && x>z */
    else /* y>=x */
    if(y>z) max=y; /* y>=x  && y>z */
    else max=z; /* z>=y  && y>=x */
    return(max);
}
```

/*使用简单的if...elseif... 语句*/

```
int max3(int x,int y,int z)
{
    int max=x; if(x>y) ;
    else if(y>z) max=y; /* y>=x  && y>z */
    else max=z; /* z>=y  && y>=x */
    return(max);
}
```

/*使用嵌套的条件运算实现*/

```
int max4(int x,int y,int z)
{
    return(x>y?(x>z?x:z):(y>z?y:z));
}
```

main()

```
{
    int a,b,c;
    printf("%%d%%d%%d=>a b c:");
    scanf("%d %d %d",&a,&b,&c);
    printf("Max(%d,%d,%d)=%d",a,b,c,max3(a,b,c));
}
```

【实例分析二】简单的if 语句的使用。写一程序，从键盘上输入1 年份year (4

位十进制数)，判断其是否闰年。闰年的条件是：能被4整除、但不能被100整除，或者能被400整除。 算法设计要点：

如果X能被Y整除，则余数为0，即如果 $X \% Y$ 的值等于0，则表示X能被Y整除！

预置为0（非闰年），这样仅当year为闰年时将leap置为1即可。这种处理两种状态值的方法，对优化算法和提高程序可读性非常有效。

```
main()
{
    int year,leap=0; /* leap=0: 预置为非闰年*/
    printf("Please input the year:");
    scanf("%d",&year);
    if (year % 4==0)
    {
        if (year % 100 != 0) leap=1;
    }
    else
    {
        if (year%400==0) leap=1;
    }
    if (leap)
        printf("%d is a leap year.\n",year);
    else
        printf("%d is not a leap year.\n",year);
}
```

第五章

循环结构程序设计

【基本要求】

通过本单元的学习，使学生循环结构程序设计的概念，掌握 while 语句，do ... while 语句以及 for 语句，熟悉循环结构嵌套，熟练掌握 break 语句和 continue 语句，理解语句标号和 goto 语句的使用。

【教学重点】

实现循环的两种基本方法，循环结构的三种形式及其应用，转向语句及其作用，循环结构程序设计实例。

【本章结构】

- 1、循环的实现 $\left\{ \begin{array}{l} \text{用 while 语句实现循环} \\ \text{用 do.....while 语句实现循环} \\ \text{用 for 语句实现循环} \end{array} \right.$
- 2、循环的嵌套
- 3、几种循环的比较
- 4、break 语句和 continue 语句
- 5、goto 语句及用 goto 语句构成循环
- 6、程序举例

第一部分 While 与 Do...While

许多问题的求解归结为重复执行的操作，比如数值计算中的方程迭代求根，非数值计算中的对象遍历。重复执行就是循环。重复工作是计算机特别擅长工作之一。

重复执行不是简单地重复，每次重复，操作的数据（状态、条件）都可能发生变化。

重复的动作是受控制的，比如满足一定条件继续做，一直做直到某个条件满足，做多少次结束。也就是说重复工作需要进行控制-循环控制。C 语言提供了三种循环控制语句（不考虑 goto/if 构成的循环），构成了三种基本的循环结构：

- (1) while 语句构成的循环结构(“当型循环”)
- (2) do-while 语句构成的循环结构(“直到型循环”)

(3) for 语句构成的循环结构（“当型循环”）

一、while 语句（当型循环）

while 语句的一般形式是：如图 5.1 所示

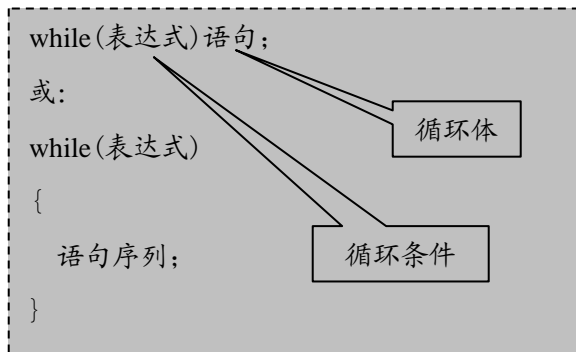


图 5.1 while 语句结构

其中：表达式称为“循环条件”，语句称为“循环体”。为便于初学者理解，可以读做“当条件（循环条件）成立（为真），循环执行语句（循环体）”

执行过程是：

(1)先计算 while 后面的表达式的值，如果其值为“真”则执行循环体，

(2)在执行完循环体后，再次计算 while 后面的表达式的值，如果其值为“真”则继续执行循环体，如果表达式的值为假，退出此循环结构。

【注意】使用 while 语句需要注意以下几点：

(1)while 语句的特点是先计算表达式的值，然后根据表达式的值决定是否执行循环体中的语句。因此，如果表达式的值一开始就为“假”，那么循环体一次也不执行。

(2)当循环体为多个语句组成，必须用{}括起来，形成复合语句。

(3)在循环体中应有使循环趋于结束的语句，以避免“死循环”的发生。

例 5.1: 利用 while 语句，编写程序计算 $1+2+3+\dots+100$ 。

【分析】

算法 1: 直接写出算式

S1: $\text{result} \leftarrow 1+2+3+4+5+\dots+100$

很简单。但是写都写得累死了。

算法 2:

考虑到 $1+2+3+\dots+100$ 可以改写为: $((1+2)+3)+\dots+100$,

S1: $p1 \leftarrow 1+2$

S2: $p2 \leftarrow p1+3$

S3: $p3 \leq p2 + 4$

...

S99: $p99 \leq p98 + 100$ 结果在 $p100$ 里。

此算法也一样麻烦，要写 99 步，同时要使用 99 个变量。本算法同样不适合编程。

但是可以从本算法看出一个规律。即：每一步都是两个数相加，加数总是比上一步加数增加 1 后参与本次加法运算，被加数总是上一步加法运算的和。可以考虑用一个变量 i 存放加数，一个变量 p 存放上一步的和。那么每一步都可以写成： $p+i$ ，然后让 $p+i$ 的和存入 p ，即：每一步都是 $p \leq p+i$ 。也就是说 p 既代表被加数又代表和。这样可以得到算法 3。执行完步骤 S99 后，结果在 p 中。

算法 3:

S0: $p \leq 0, i \leq 1$

S1: $p \leq p+i, i \leq i+1$

S2: $p \leq p+i, i \leq i+1$

S3: $p \leq p+i, i \leq i+1$

...

S99: $p \leq p+i, i \leq i+1$

从算法 3 表面上看与算法 2 差不多，同样要写 99 步。但是从算法 3 可以看出 S1-S99 步骤实际上是一样的，也就是说 S1-S99 同样的操作重复做了 99 次。计算机对同样的操作可以用循环完成，循环是计算机工作的强项（计算机高速度运算）。算法 4 就是在算法 3 的基础上采用循环功能的算法实现。

算法 4:

S0: $p \leq 0, i \leq 1$ (循环初值)

S1: $p \leq p+i, i \leq i+1$ (循环体)

S2: 如果 i 小于或等于 100，返回重新执行步骤 S1 及 S2；否则，算法结束（循环控制）。

p 中的值就是 $1+2+\dots+100$ 的值。

【结论】 编制循环程序要注意下面几个方面：

(1) 遇到数列求和，求积的一类问题，一般可以考虑使用循环解决。

(2) 注意循环初值的设置。一般对于累加器常常设置为 0，累乘器常常设置为 1。

(3) 循环体中做要重复的工作，同时要保证使循环倾向于结束。循环的结束由 while 中的表达式（条件）控制。

例 5.2: 利用 while 语句，计算 $1+1/2+1/4+\dots+1/50$ 。

观察数列 1, 1/2, ..., 1/50。=1/1,1/2,,1/50。分子全部为 1，分母除第一项外，全部是偶数。同样考虑用循环实现。其中累加器用 sum 表示（初值设置为第一项 1，以后不累加第一项），循环控制用变量 i（i: 2-50）控制，数列通项：1/i。

<pre>main() { float sum=1; int i=2; while(i<=50) { sum=sum+1.0/i; i+=2; } printf(sum); }</pre>	<pre>main() { float sum=1; int i=1; while(2*i<=50) { sum=sum+1.0/(2*i); i+=1; } printf(sum); }</pre>
---------------------------------------------------------------------------------------------------------------------------------------	---------------------------------------------------------------------------------------------------------------------------------------------

程序 5.1 例 5.2 程序

二、do-while 语句（直到型循环）

do-while 语句的一般形式是：

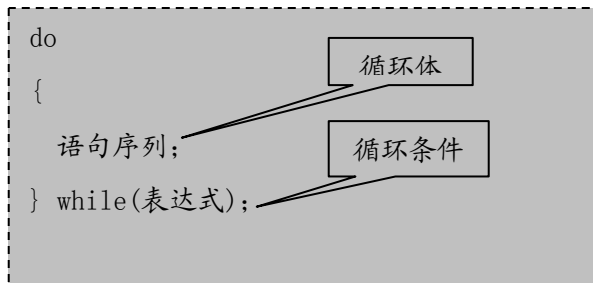


图 5.2 dowhile 循环的一般形式

其中：表达式称为“循环条件”，语句称为“循环体”。为便于初学者理解，可以读做：“执行语句（循环体），当条件（循环条件）成立（为真）时，继续循环”或“执行语句（循环体），当条件（循环条件）不成立（为假）时，循环结束”-直到型循环。

执行过程是：

(1) 执行 do 后面循环体语句。

(2) 计算 while 后面的表达式的值，如果其值为“真”则继续执行循环体，如果表达式的值为假，退出此循环结构。

【说明】

(1) do-while 循环，总是先执行一次循环体，然后再求表达式的值，因此，无论表达式是否为“真”，循环体至少执行一次。

(2) do-while 循环与 while 循环十分相似，它们的主要区别是：while 循环先判断循环条件再执行循环体，循环体可能一次也不执行。do-while 循环先执行循环体，再判断循环条件，循环体至少执行一次。

(3) 其它：复合语句{}，避免死循环要求同 while 循环。

(4) C 语言没有 do-until（做...直到条件满足后循环停止），但是用 do-while 可以实现它，只要控制表达式为：！就可以了。

(5) 表达式必须用（）括起来；可以是关系、逻辑表达式、任意类型的常量（或变量）；

(6) 表达式的具体形式可以是：常量、变量、表达式、函数调用；

③先执行一次循环体，然后进行表达式判断，若其值为真（用非 0 表示），继续执行语句体；


(7) 语句体为多个语句时须用{}括起来；也可以是空语句（；）

(8) while（表达式）要加分号（；）（与 while 的区别）

【实例分析一】

例 5、3：利用 do-while 语句计算机 $1+1/2+1/4+\dots+1/50$ 。

```
main()
{
    float sum=1; int i=2;
    do
    {
        sum=sum+1.0/i;
        i+=2;
    } while(i<=50)
    printf(sum);
}
```



程序 5.2 例 5.3 程序

【实例分析之二】

例 5.4 用 do-while 语句求解 1~100 的累计和。

```
int dowhile(void)
{
    int i=1, sum=0; /*定义并初始化循环控制变量，以及累计器*/
    do {
        sum += i; i++; /*累加*/
    } while(i<=100); /*循环继续条件： i<=100*/
    return(sum); }

main()
```

```

{
    printf("%d",dowhile())
}

```

等价的循环体程序：

```
do{ sum+=i++ }while(i<=100);
```

【思考题】上述循环体语句可否改为：

```
{ ++i; /*实现累加*/ sum=sum+i; } /*循环控制变量 i 增 1*/
```

不行，分析原因

do-while 语句比较适用于处理：不论条件是否成立，先执行 1 次循环体语句组的情况。除此之外，do-while 语句能实现的，for 语句也能实现，而且更简洁。

第二部分 For 语句

C 语言中的 for 语句使用更为灵活，不仅可以用于循环此书已经确定的情况，而且可以用于循环次数不确定而只给出循环结束条件的情况，它完全可以代替 while 语句。

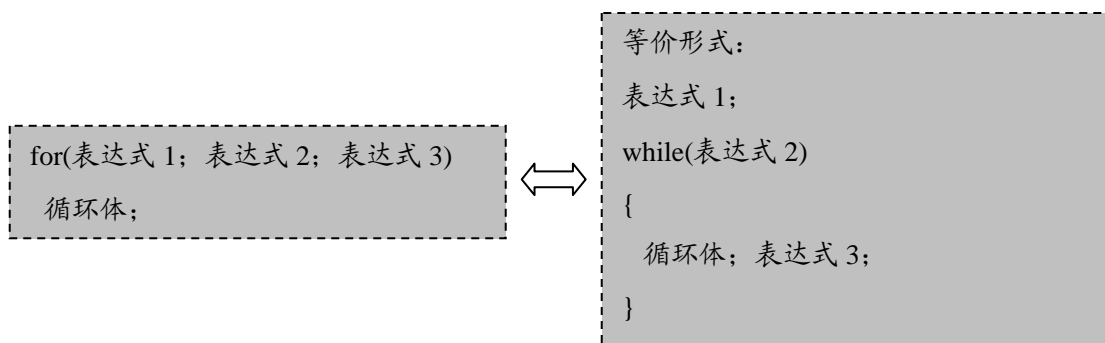


图 5.3 for 语句的一般格式

一、for 语句的一般形式是：

for 是关键词，其后有 3 个表达式，各个表达式用“；”分隔。3 个表达式可以是任意的表达式，通常主要用于 for 循环控制。

for 循环执行过程如下：

(1) 计算表达式 1。

(2) 计算表达式 2，若其值为非 0（循环条件成立），则转 3）执行循环体；若其值为 0（循环条件不成立），则转 5）结束循环。

(3) 执行循环体。

(4)计算表达式 3，然后转 2) 判断循环条件是否成立。

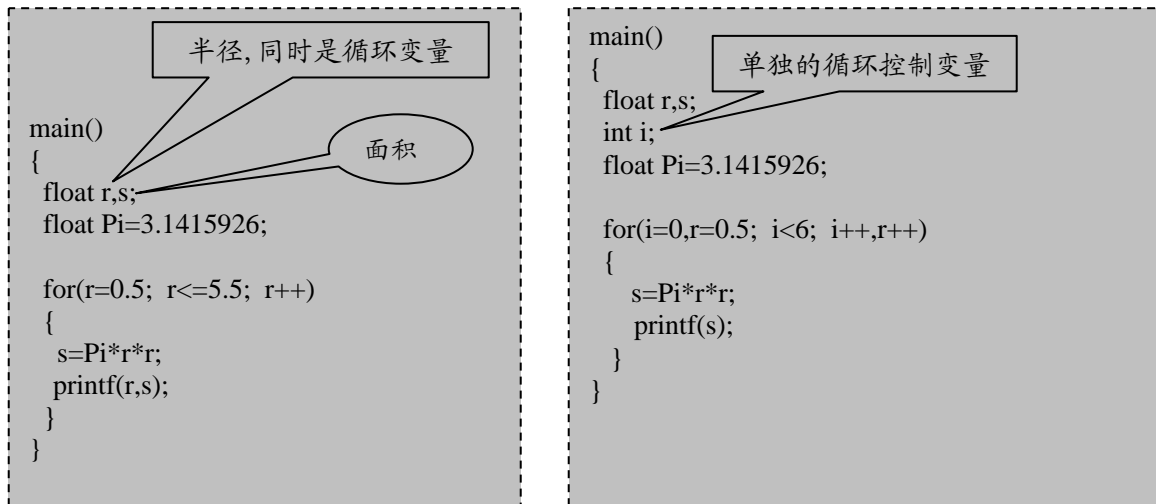
(5)结束循环，执行 for 循环之后的语句。

例 5.5: 写一个程序，计算半径为 0.5,1.5,2.5,3.5,4.5,5.5mm 时圆的半径。

解:

本例要求计算 6 个不同半径的圆的面积，可以每次计算一个圆的面积，共计算 6 次，采用循环处理此问题。

注意到半径变化是有规律的：从 0.5mm 开始按 1mm 的规律递增。假设半径 r , r 从 0.5 开始，每循环一次 r 增加 1， $r \leq 5.5$ 。



程序 5.4 例 5.5 程序

例 5.6: 求正整数 n 的阶乘 $n!$, 其中 n 由用户输入。

解: $n! = 1 \times 2 \times \dots \times n$; 设置变量 $fact$ 为累乘器(被乘数), i 为乘数, 兼做循环控制变量。

```
main()
{
    float fact;
    int i,n;

    scanf(&n);

    for(i=1,fact=1.0; i<=n; i++)
        fact=fact*i;

    printf(fact);
}
```

程序 5.5 例 5.6 程序

for 语句最容易理解、最常用的形式是:

for(循环变量赋初值; 循环条件; 循环变量修正)

循环体;

【说明】

(1)for 语句中表达式 1, 表达式 2, 表达式 3 都可以省略, 甚至三个表达式都同时省略, 但是起分隔作用的“; ”不能省略。

(2)如果省略表达式 1, 即不在 for 语句中给循环变量赋初值, 则应该在 for 语句前给循环变量赋初值。如程序 5.6 所示

<pre>main() { ... for(i=1,fact=1.0; i<=n; i++) fact=fact*i; ... }</pre>	<pre>main() { i=1,fact=1; for(; i<=n; i++) fact=fact*i; ... }</pre>
------------------------------------------------------------------------------------------------	--------------------------------------------------------------------------------------------

程序 5.6 for 语句循环变量赋初值

(3)如果省略表达式 2, 即不在表达式 2 的位置判断循环终止条件, 循环无终止地进行, 也就是认为表达式 2 始终为“真”。则应该在其它位置(如: 循环体)安排检测及退出循环的机制。如程序 5.7 所示。

```
main()
{
    ...
    for(i=1,fact=1.0; ; i++)
    {
        fact=fact*i;
        if(i==n)break;
    }
    ...
}
```

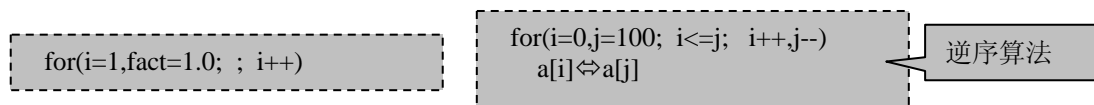
程序 5.7 for 语句循环终止条件

(4)如果省略表达式 3, 即不在此位置进行循环变量的修改, 则应该其它位置(如: 循环体)安排使循环趋向于结束的工作。

```
main()
{
    ...
    for(i=1,fact=1.0; i<=n; )
    {
        fact=fact*i;
        i++;
    }
    ...
}
```

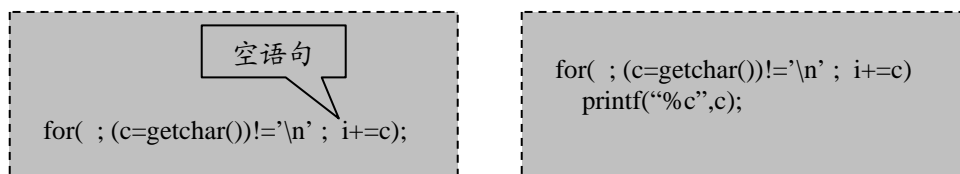
程序 5.8 for 语句循环变量的修改

(5)表达式 1 可以是设置循环变量初值的表达式,也可以是与循环变量无关的其它表达式; 表达式 1, 表达式 3 可以是简单表达式,也可以是逗号表达式。



程序 5.9 for 循环中的表达式 1

(6)表达式 2 一般为关系表达式或逻辑表达式,也可以是数值表达式或字符表达式,事实上只要是表达式就可以。



程序 5.10 for 循环中的表达式 2

【结论】

从上面的说明可以看出, C 语言的 for 语句功能强大,使用灵活,可以把循环体和一些与循环控制无关的操作也都可以作为表达式出现,程序短小简洁。但是,如果过分使用这个特点会使 for 语句显得杂乱,降低程序可读性。建议不要把与循环控制无关的内容放在 for 语句的三个表达式中,这是程序设计的良好风格。

二、 语句标号和 goto 语句 (不做重点要求,但要了解)

1、 语句标号

语句标号: 一个标识符+ “:”, 它表示程序指令的地址。语句标号的标识符遵守“标识符”命名规定。

例如:

```
loop: ERR:
```

2、 goto 语句 (无条件转移语句)

goto 语句-无条件转移 (转向) 语句。格式为: goto 语句标号;

goto 语句的功能: 程序无条件转移到“语句标号”处执行。

结构化程序设计方法主张“限制”(注意不是“禁止”)使用 goto 语句。因为 goto 语句不符合结构化程序设计的准则-模块化,使用三种基本程序结构。无条

件转移使程序结构无规律，可读性变差。但是，任何事情都是一分为二，如果能大大提高程序的执行效率，也可以使用。

goto 语句常见的两种用途（现在没有人会这么用）：

(1)if/goto 构成循环-被 while，do-while，for 代替。

例 5.7: 仅仅作为 goto，语句标号概念的理解，实际建议不要这样使用。

```
main()
{
    int i=1,sum=0;

    LOOP:
        sum+=i;
        i++;
        if(i>100)goto PRT;
        goto LOOP;
    PRT:
        printf("sum=%d",sum);
}
```

程序 5.11 goto 语句的使用

(2)从循环体跳到循环体外-被 break，continue 代替。（break 跳出本层循环 continue 结束本次循环，多层循环可以设置一个标志变量，逐层跳出）。

第三部分 循环方式的比较及循环中断

一、几种循环的比较

1、循环结构的基本组成部分（小结）

从前面的循环结构的语法和例子的介绍，我们可以看出循环结构由 4 部分组成：

- (1)循环变量、条件（状态）的初始化
- (2)循环变量、条件（状态）检查，以确认是否进行循环
- (3)循环变量、条件（状态）的修改，使循环趋于结束
- (4)循环体处理的其它工作。

2、几种循环的比较

C 语言中，三种循环结构（不考虑用 if/goto 构成的循环）都可以用来处理同一个问题，但在具体使用时存在一些细微的差别。如果不考虑可读性，一般情况下它们可以相互代替。

(1)循环变量初始化：while 和 do-while 循环，循环变量初始化应该在 while 和 do-while 语句之前完成；而 for 循环，循环变量的初始化可以在表达式 1 中完成。

(2)循环条件：while 和 do-while 循环只在 while 后面指定循环条件；而 for 循环可以在表达式 2 中指定。

(3)循环变量修改使循环趋向结束：while 和 do-while 循环要在循环体内包含使循环趋于结束的操作；for 循环可以在表达式 3 中完成。

(4)for 循环可以省略循环体，将部分操作放到表达式 2，表达式 3 中，for 语句功能强大。

(5)while 和 for 循环先测试表达式，后执行循环体，而 do-while 是先执行循环体，再判断表达式。（所以 while,for 循环是典型的当型循环，而 do-while 循环可以看作是直到型循环）。

(6)三种基本循环结构一般可以相互替代，不能说哪种更加优越。具体使用哪一种结构依赖于程序的可读性和程序设计者个人程序设计的风格（偏好）。我们应当尽量选择恰当的循环结构，使程序更加容易理解。（尽管 for 循环功能强大，但是并不是在任何场合都可以不分条件使用）。

例 5.8: 将 50-100 之间的不能被 3 整除的数输出（用三种循环结构实现）

<pre>/* 用 while 语句实现 */ main() { int i=50; while(i<=100) { if(i%3!=0)printf("%4d",n); i++; } }</pre>	<pre>/* 用 do-while 语句实现 */ main() { int i=50; do { if(i%3!=0)printf("%4d",n); i++; } while(i<=100) }</pre>	<pre>/* 用 for 语句实现 */ main() { int i; for(i=50; i<=100; i++) if(i%3!=0)printf("%4d",n); }</pre>
-----------------------------------------------------------------------------------------------------------------------------------------	-----------------------------------------------------------------------------------------------------------------------------------------------	--------------------------------------------------------------------------------------------------------------------

程序 5.12 三种循环结构的比较

对计数型的循环或确切知道循环次数的循环，用 for 比较合适，对其它不确定循环次数的循环许多程序设计者喜好用 while/do-while 循环（如链表操作）。

二、break 和 continue 语句

1、break 语句

前面介绍的三种循环结构都是在执行循环体之前或之后通过对一个表达式的测试来决定是否终止对循环体的执行。在循环体中可以通过 `break` 语句立即终止循环的执行，而转到循环结构的下一语句处执行。

`break` 语句的一般形式为：`break;`

`break` 语句的执行过程是：终止对 `switch` 语句或循环语句的执行（跳出这两种语句），而转移到其后的语句处执行。

【说明】

(1) `break` 语句只用于循环语句或 `switch` 语句中。在循环语句中，`break` 常常和 `if` 语句一起使用，表示当条件满足时，立即终止循环。注意 `break` 不是跳出 `if` 语句，而是循环结构。

(2) 循环语句可以嵌套使用，`break` 语句只能跳出（终止）其所在的循环，而不能一下子跳出多层循环。要实现跳出多层循环可以设置一个标志变量，控制逐层跳出。

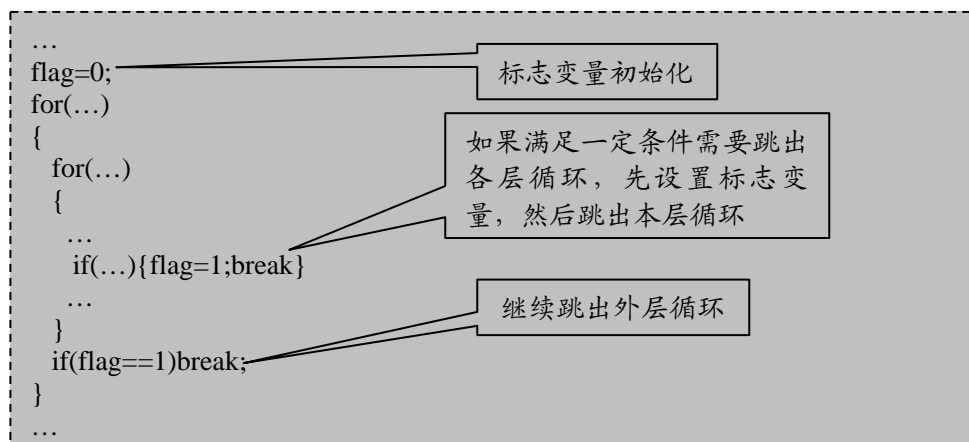


图 5.4 `break` 语句的作用

例 5.9：从键盘上连续输入字符，并统计其中大写字母的个数，直到输入“换行”字符时结束。

```
main()
{
    int i,s=0;
    int sum=0;

    for(i=1; i<=10; i++)
    {
        s=s+2;
        if(s>5) break;
        printf(sum);
    }
}
```

```
main()
{
    char ch;
    int sum=0;

    while(1)
    {
        ch=getchar();
        if(ch=='\n') break;
        if(ch>='A' && ch<='Z') sum++;
    }
    printf(sum);
}
```

程序 5.13 例 5.9 程序

2、continue 语句（翻译为“继续”（循环））

continue 语句的一般形式是：continue;

continue 语句的功能是结束本次循环。即跳过本层循环体中余下尚未执行的语句，接着再一次进行循环条件的判定。注意：执行 continue 语句并没有使整个循环终止。注意与 break 语句进行比较。

在 while 和 do-while 循环中，continue 语句使流程直接跳到循环控制条件的测试部分，然后决定循环是否继续执行。在 for 循环中，遇到 continue 后，跳过循环体中余下的语句，而去对 for 语句中的表达式 3 求值，然后进行表达式 2 的条件测试，最后决定 for 循环是否执行。

例 5.10：从键盘输入 30 个字符，并统计其中数字字符的个数

```
main()
{
    int sum=0,i;
    char ch;
    for(i=0; i<30; i++)
    {
        ch=getchar();
        if(ch<'0' || ch>'9') continue;
        sum++;
    }
    printf(sum);
}
```

程序 5.14 例 5.10 程序

例 5.11：编程验证 continue 对流程的控制作用。

```
#include "stdio.h"
main()
{
    int i=0,s=0;
    for(;i<=100;i++) {
        printf("\nContinue : %d",i);
        if(!(i%13))
            s+=i,printf("%4d",i);
        else
            continue;
        printf("\ns=%d",s);
    }
}
```

```

getch();/* 暂停程序运行，等待输入 */
clrscr();/* 清屏幕 */
i=1;
while(i<=100)
{
    if(!(i%13)) s+=i;
    else {i++; continue;} i++;
    printf("%4d",i);
} }

```

【结论】

break,continue 主要区别:

continue 语句只终止本次循环，而不是终止整个循环结构的执行;

break 语句是终止循环，不再进行条件判断。

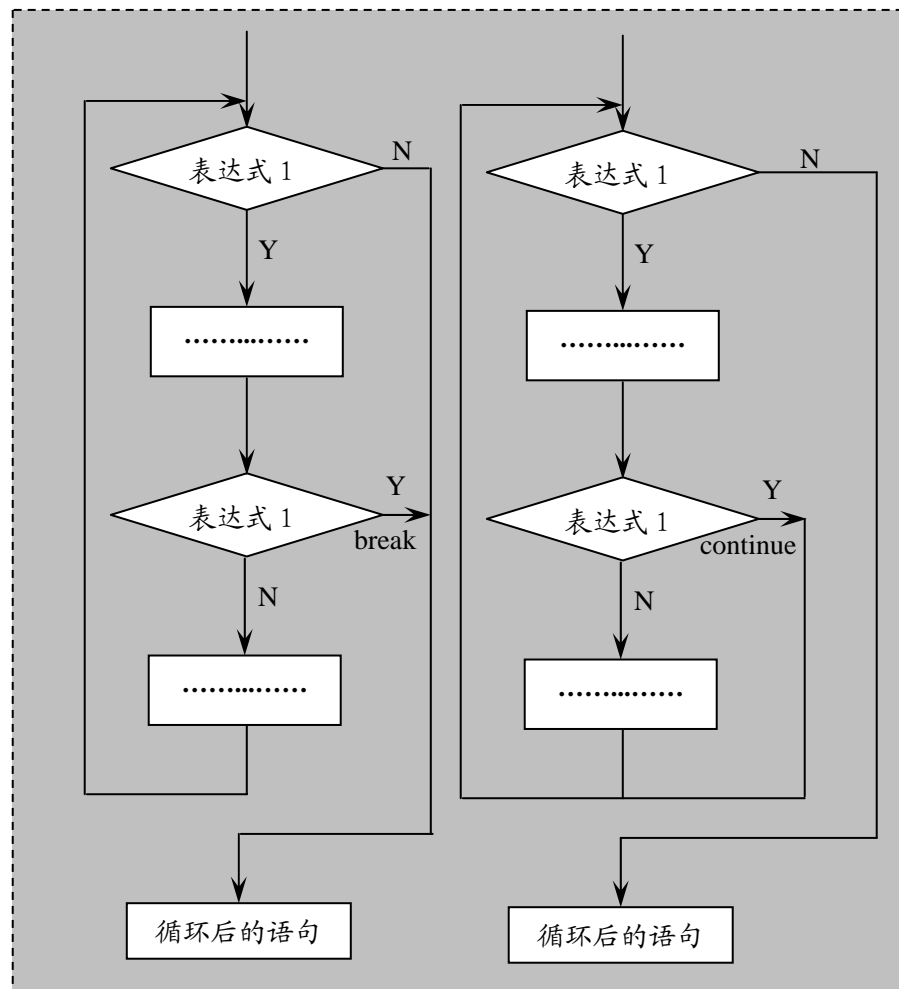


图 5.5 break 与 continue 语句执行过程比较

第四部分 循环语句实例分析

应用举例(实例分析)

【实例分析一】从键盘输入一个大于 2 的整数 n ，判断是不是素数。

素数定义：只能被 1 和它本身整除的数是素数。为了判断一个数 n 是否为素数，可以让 n 除以 2 到 $n-1$ （实际上只要到 \sqrt{n} ）之间的每一个整数，如果 n 能够被某个整数整除，则说明 n 不是素数，否则 n 是素数。

说明：

(1)math.h

(2)do-while 读键盘输入，保证 $n>2$

(3)flag 标志变量（开关变量 0-素数，1-非素数）

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

main()
{
    int n,i,m,r,flag;;

    do{
        scanf(&n);
    }while(n<=2);

    m=sqrt(n);
    flag=0;

    for(i=2; i<=m; i++)
    {
        r=n%i;
        if(r==0)
        {
            flag=1;
            break;
        }
    }

    if(flag==1)
        printf("%d is not a prime number\n",n);
    else
        printf("%d is a prime number\n",n);
}
```

i 从 2-m 进行测试

若 n 能被 i 整除 ($r=0$)， n 就不是素数，素数标志 flag 置 1

程序 5.15 实例分析一程序

【实例分析二】 用 do-while、while、for 三种循环语句实现求 1 ~ 100 的累计和。

```
int sumwhile(int n)
{
    int sum=0,i=1;
    while(i<=n)
        sum+=i++;
    return(sum);
}
int dowhile(int n)
{
    int sum=0,i=1;
    do
        sum+=i++;
    while(i<=n); return(sum);
}
int for123(int n)
{
    int sum,i;
    for(sum=0,i=1;i<=n;i++)
        sum+=i;
    return(sum);
}
int sumfor23(int n)
{
    int sum=0,i=1;
    for(;i<=n;i++)
        sum+=i;
    return(sum);
}
int sumfor2(int n)
{
    int sum=0,i=1;
    for(;i<=n;)
        sum+=i++;
    return(sum);
}
int sumfor(int n)
{
    int sum=0,i=1;
    for(;;) {
        sum+=i;
        i++;
        if(i>n)
```

```

        break;
    }
    return(sum);
}
main(){ printf("sum(100)=%d\n",sumfor(100));}

```

请认真分析 for 循环的几种具体情况；在主函数中可以调用任何一个自定义函数验证各循环求和的结果。

【实例分析三】 求两个正整数（m和n）的最大公约数和最少公倍数（m放大数，n放小数）

【原理分析】 $r=m\%n$, $m=n$, $n=r$;当其余数（实际上是n）为0时的n即为最大公约数；

```

main()
{
    int gbs,gys,m,n,r,p;
    printf("Input two integers(m,n):");
    scanf("%d,%d",&m,&n);
    if(m<n){ //保证 m 中存放大的数
        p=m*n; //先保存 m*n 以便求最少公倍数
        m=m+n; n=m-n; m=m-n;
    }
    while(n!=0){ r=m%n; m=n; n=r; }
    printf("The common divisor: %d\n",m);
    printf("The common multiple: %d\n",p/m);
}

```

【实例分析四】 求 $S_n=a+aa+aaa+\dots+aa(n \text{ 个 } a)$ 的值，其中 a 是一个 1-9 之间的数字。如：2+22+222+2222

【原理分析】 后面一个数总是前面一个数的值乘以 10+a;

```

main()
{
    int a,n,ts,i;
    unsigned long sum;
    printf("Input two integers(a,n):");
    scanf("%d,%d",&a,&n);
    if(n<1) {
        printf("the value of n can't less than 1");
        exit(1);
    }
    for(sum=0,ts=a,i=1;i<=n;i++) {sum+=ts; ts=ts*10+a; }
    printf("n:%d sum= %lu\n",n,sum);
}

```

【实例分析五】 求 $S_n=1!+2!+3!+\dots+n!$ 的值；

【原理分析】 后面一个数总是前面一个数的值乘以当前项值；


```

#include "math.h"
main()
{
    float s,t;int n,i;
    printf("Input one integer(n):");
    scanf("%d",&n);n=fabs(n);
    if(n<2) s=1;
    else
        for(s=0,t=1,i=1;i<=n;i++)
            t*=i,s+=t;
    printf("n:%d sum= %f\n",n,s);
}

```

【实例分析六】 换零钱：将一元人民币全部兑换为零钱（1、2、5 分），又多少种兑换方法？

```

main()
{
    int i,j,k,n;
    for(n=100,k=i=0;i<=n/5;i++)    //i 计数 5 分币
        for(j=0;j<=(n-i*5)/2;k++,j++)    //k 用于多少兑换方法；j 计数 2 分币
            printf("5 Cent:%d\t 2 Cent:%d\t 1 Cent:%d\n",i,j,n-i*5-j*2);
    printf("Total times:%d\n",k);
}

```

【实例分析七】 九九加法、乘法表的设计制作演示。源程序

```

void fplus()
{ /*九九加法*/
    int i,j;
    for(i=1;i<=9;i++)
    {
        for(j=1;j<=i;j++)
            printf("%d+%d=%-2d%%c",i,j,i+j,(j!=i?'':'\n'));
    } /*end for first for*/
}

void fmulti()
{ /*九九乘法*/
    int i,j;
    for(i=1;i<=9;i++)
    {
        for(j=1;j<=i;j++)

```

```

        printf("%d*%d=%-3d",i,j,i*j);
        printf("\n");
    } /*end for first for*/
}
main(){
    int sele;
    system("cls"); /*调用 DOS 命令“cls”清屏幕 */
    while(1){
        printf("select(1:plus,2:multi,0:exit):");
        scanf("%d",&sele);
        switch(sele){
            case 1: clrscr(); /*库函数，清屏幕 */ fplus(); printf("\n\n"); break;
            case 2: system("cls"); fmulti(); printf("\n\n"); break;
            case 0: exit(0); /*可以退出无限循环 while ( 1 ) */
        } /*end for switch*/
    } /*end for while*/
} /*end for main*/

```

【说明】

(1)库函数 exit()用于终止本程序的运行，返回到系统调用（如操作系统状态）。其参数为 int（0：表示正常结束；非 0 表示异常结束）。

(2)库函数 system()用于引用 dos 命令。

第六章

函数

【基本要求】

通过本单元的学习，使学生了解文件概念，学会文件指针，掌握打开和关闭文件的操作，并能够熟练应用常用文件操作函数。

【教学重点】

函数（分类、定义、声明、调用、参数与返回值），函数的属性（内部函数与外部函数），变量的属性（类别属性、存储属性），工程文件（多文件的编译、连接、运行）。

【本章结构】

- 1、函数定义的一般形式
 - 无参函数定义的一般格式
 - 有参函数定义的一般格式
 - 空函数

- 2、函数参数和函数的值
 - 形式参数和实际参数
 - 函数的返回值

- 3、函数的调用
 - 函数调用的一般形式
 - 函数调用的方式
 - 对被调用函数的声明和函数原型
 - 函数的嵌套调用
 - 函数的递归调用

- 4、数组作为函数参数
 - 数组元素作为函数实参
 - 数组名作为函数参数
 - 多维数组名作为函数参数

- 5、变量的存储类别
 - 动态存储方式与静态存储方式
 - auto 变量
 - 用 static 声明局部变量
 - register 变量
 - 用 extern 声明外部变量
 - 用 static 声明外部变量
 - 关于变量的声明和定义

第一部分 函数基础知识

一、函数概述

1、C 语言的函数:

C 语言的函数是子程序的总称，包括函数和过程。（有返回值、无返回值，教材中称为：有返回值函数，无返回值函数）。

C 语言函数可以分为库函数、用户自定义函数。库函数由系统提供，程序员只需要使用（调用），用户自定义函数需要程序员自己编制。

2、C 语言的程序由函数组成，函数是 C 语言程序的基本单位。

前面章节介绍的所有程序都是由一个主函数 main 组成的。程序的所有操作都在主函数中完成。事实上，C 语言程序可以包含一个 main 函数，也可以包含一个 main 函数和若干个其它函数。

C 语言程序的结构如图 6.1 所示。在每个程序中，主函数 main 是必须的，它是所有程序的执行起点，main 函数只调用其它函数，不能为其它函数调用。如果不考虑函数的功能和逻辑，其它函数没有主从关系，可以相互调用。所有函数都可以调用库函数。程序的总体功能通过函数的调用来实现。

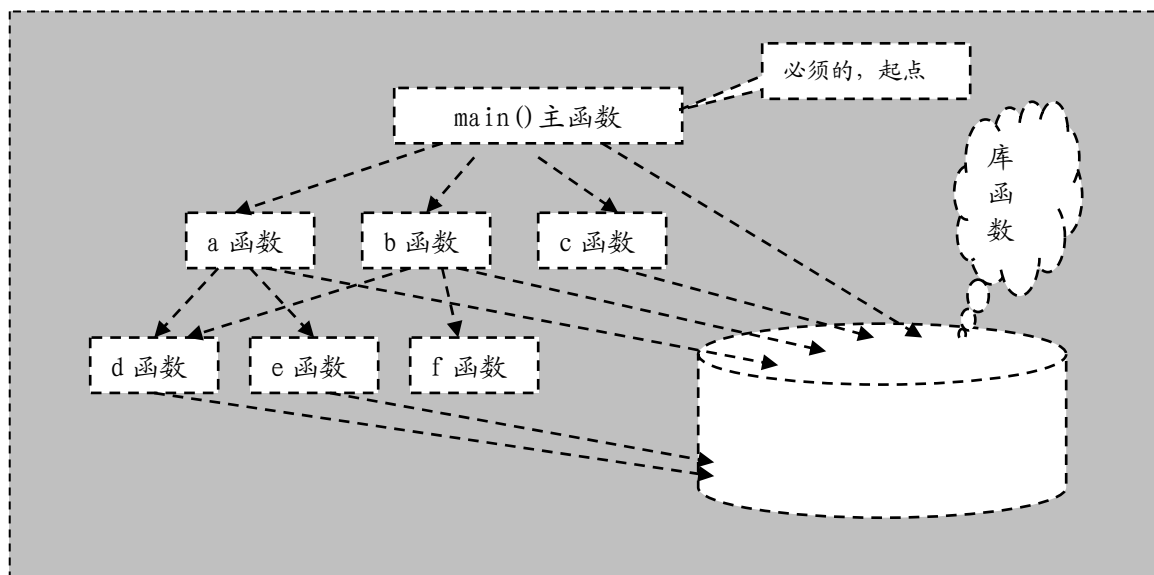


图 6.1 函数调用图示

3、使用函数的意义（补充）

【思考】 只用一个 `main` 函数就可以编程，为什么这么复杂，还要将程序分解到函数，还要掌握这么多概念，太麻烦了？对于小程序可以这样做，但是对于一个有一定规模的程序这样做就不合适了。使用函数的几个原因：

(1)使用函数可以控制任务的规模

一般应用程序都具有较大的规模。例如：一个齿轮误差分析软件系统的源程序行数要数千行。一个传动链计算机辅助设计系统的源程序行数 5 万多行。

使用函数可以将程序划分为若干功能相对独立的模块，这些模块还可以再划分为更小的模块，直到各个模块达到程序员所能够控制的规模。然后程序员再进行各个模块的编制。因为各个模块功能相对独立，步骤有限，所以流程容易控制，程序容易编制，修改。

一般一个模块的规模控制在源程序 60 行以内（但是也不必教条化）。

(2)使用函数可以控制变量的作用范围

变量在整个模块范围内全局有效，如果将一个程序全部写在 `main()` 函数内，大家可以想象，变量可以在 `main` 函数内任何位置不加控制地被修改。如果发现变量的值（状态）有问题，你可能要在整个程序中查找哪里对此变量进行了修改，什么操作会对此变量有影响，改动了一个逻辑，一不留神又造成了新的问题，最后程序越改越乱，有时连程序员自己都不愿意再看自己编写的程序。都是“大”惹的祸。

使用函数后，变量局限于自己的函数内，60 行代码内，太好了，能控制吗？

函数-函数通过接口（参数表，返回值）通讯，交换数据。

(3)使用函数，程序的开发可以由多人分工协作。

一个 `main()` 模块，怎么合作？

将程序划分为若干模块（函数），各个相对独立的模块（函数）可以由多人完成，每个人按照模块（函数）的功能要求，接口要求编制代码，调试，确保每个模块（函数）的正确性。最后将所有模块（函数）合并，统一调试、运行。

(4)使用函数，可以重新利用已有的、调试好的、成熟的程序模块

想象一下，如果要用到求平方根，如果系统不提供 `sqrt` 这样的函数，怎么办？（找数学书，考虑算法，编制求平方根代码）。C 语言的库函数（标准函数）就是系统提供的，调试好的、常用的模块，我们可以直接利用。事实上我们的代码也可以重新利用，可以将已经调试好的，功能相对独立的代码改成函数，供以后调用。

4、函数的一些概念

(1)主函数、其它函数

(2)主调函数（调用其它函数的函数）、被调函数（被其它函数调用的函数）

(3)标准函数（库函数）和用户自定义函数

(4)无参函数、有参函数

(5)无返回值函数、有返回值函数

5、C语言使用函数的一些说明

(1)C程序是由函数构成

- 一个 C 源程序至少包含一个 main 函数,也可以包含一个 main 函数和若干个其它函数。函数是 C 程序的基本单位。
- 被调用的函数可以是系统提供的库函数,也可以是用户根据需要自己编写设计的函数。
- C 函数库非常丰富,ANSI C 提供 100 多个库函数,Turbo C 提供 300 多个库函数。

(2) main 函数（主函数）是每个程序执行的起始点

一个 C 程序总是从 main 函数开始执行,而不论 main 函数在程序中的位置。可以将 main 函数放在整个程序的最前面,也可以放在整个程序的最后,或者放在其它函数之间。

二、函数的一般形式

函数应当先定义,后调用。

函数定义的一般形式: 如图 6.2 所示

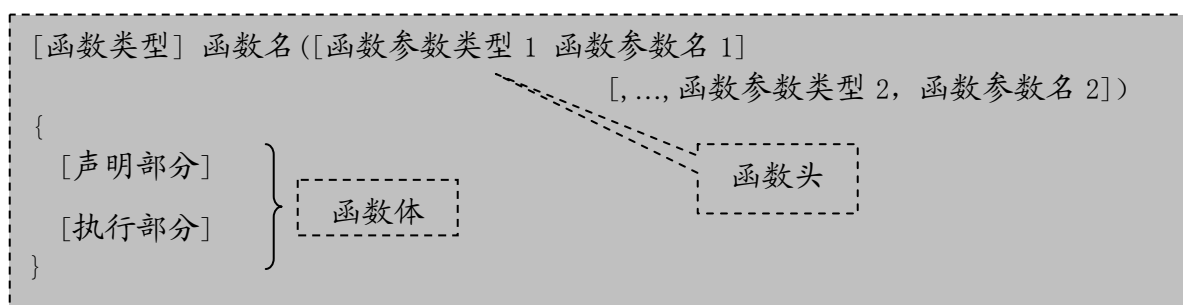


图 6.2 函数定义一般形式

【说明】 一个函数（定义）由函数头（函数首部）和函数体两部分组成

(1)函数头（首部）：说明了函数类型、函数名称及参数。

- 函数类型：函数返回值的数据类型,可以是基本数据类型也可以是构造类型。如果省略默认为 int, 如果不返回值, 定义为 void 类型。

- 函数名：给函数取的名字，以后用这个名字调用。函数名由用户命名，命名规则同标识符。
- 函数名后面是参数表，无参函数没有参数传递，但“（）”号不能省略，这是格式的规定。参数表说明参数的类型和形式参数的名称，各个形式参数用“，”分隔。

(2)函数体：函数首部下用一对{}括起来的部分。如果函数体内有多个{}，最外层是函数体的范围。函数体一般包括声明部分、执行部分两部分。

- 声明部分：在这部分定义本函数所使用的变量和进行有关声明（如函数声明）。
- 执行部分：程序段，由若干条语句组成命令序列（可以在其中调用其它函数）。

【注意】函数不能单独运行，函数可以被主函数或其它函数调用，也可以调用其它函数，但是不能调用主函数。

例 6.1：输入三个整数，求三个整数中的最大值，打印。

<p>不使用函数(除 main 外)</p> <pre>main() { int n1,n2,n3,nmax; scanf("%d%d%d",&n1,&n2,&n3); if(n1>n2) nmax=n1; else nmax=n2; if(n3>nmax) nmax=n3; printf("max=%d\n",nmax); }</pre>	<p>使用函数</p> <pre>extern int max(int,int,int); main() { int n1,n2,n3,nmax; scanf("%d%d%d",&n1,&n2,&n3); nmax=max(n1,n2,n3); printf("max=%d\n",nmax); } int max(int x,int y,int z) { int m; if(x>y) m=x; else m=y; if(z>m)m=z; return m; }</pre> <div style="position: absolute; left: 670px; top: 550px; border: 1px solid black; padding: 2px;">像调用库函数一样</div> <div style="position: absolute; left: 690px; top: 700px; border: 1px solid black; padding: 2px;">函数体</div>
-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

程序 6.1 例 6.1 程序

【说明】

(1)比较两个程序，使用函数好象程序更长了。

【思考】，如果程序中要调用 100 次求三个数最大值又会是什么情况呢？

(2)解释 max 函数的定义的几个部分。简要解释 max 函数的调用。

三、函数的参数和返回值

1、形式参数与实际参数

(1)形式参数（形参）：函数定义时设定的参数。

例 6.1 中，函数头 `int max(int x,int y,int z)` 中 `x,y,z` 就是形参，它们的类型都是整型。

(2)实际参数（实参）：调用函数时所使用的实际的参数。

例 6.1 中，主函数中调用 max 函数的语句是：`nmax=max(n1,n2,n3)`；其中 `n1,n2,n3` 就是实参，它们的类型都是整型。

(3)参数的传递

在调用函数时，主调函数和被调函数之间有数据的传递-实参传递给形参。具体的传递方式有两种：

- 值传递方式（传值）：将实参单向传递给形参的一种方式。
- 地址传递方式（传址）：将实参地址单向传递给形参的一种方式。

【注意】

(1)单向传递：不管“传值”、还是“传址”，C 语言都是单向传递数据的，一定是实参传递给形参，反过来不行。也就是说 C 语言中函数参数传递的两种方式本质相同-“单向传递”。

(2)“传值”、“传址”只是传递的数据类型不同（传值-一般的数值，传址-地址）。传址实际是传值方式的一个特例，本质还是传值，只是此时传递的是一个地址数据值。

(3)系统分配给实参、形参的内存单元是不同的，也就是说即使在函数中修改了形参的值，也不会影响实参的值。如图 6.3 所示。

- 对于传值，即使函数中修改了形参的值，也不会影响实参的值。
- 对于传址，即使函数中修改了形参的值，也不会影响实参的值。但是，**注意**：不会影响实参的值，不等于不影响实参指向的数据。
- 传址与传值一样不能通过参数返回数据，但因为传递的是地址，那么就可能通过实参参数所指向的空间间接返回数值。

(4)两种参数传递方式中，实参可以是变量、常量、表达式；形参一般是变量，要求两者类型相同或赋值兼容。

2、函数的返回值

C 语言可以从函数（被调用函数）返回值给调用函数（这与数学函数相当类似）。在函数内是通过 `return` 语句返回值的。使用 `return` 语句能够返回一个值或不返回值（此时函数类型是 `void`）。

Return 语句的格式:

Return [表达式]; 或 return (表达式);

【说明】

(1)函数的类型就是返回值的类型，return 语句中表达式的类型应该与函数类型一致。如果不一致，以函数类型为准（赋值转化）。

(2)函数类型省略，默认为 int。

(3)如果函数没有返回值，函数类型应当说明为 void（无类型）。

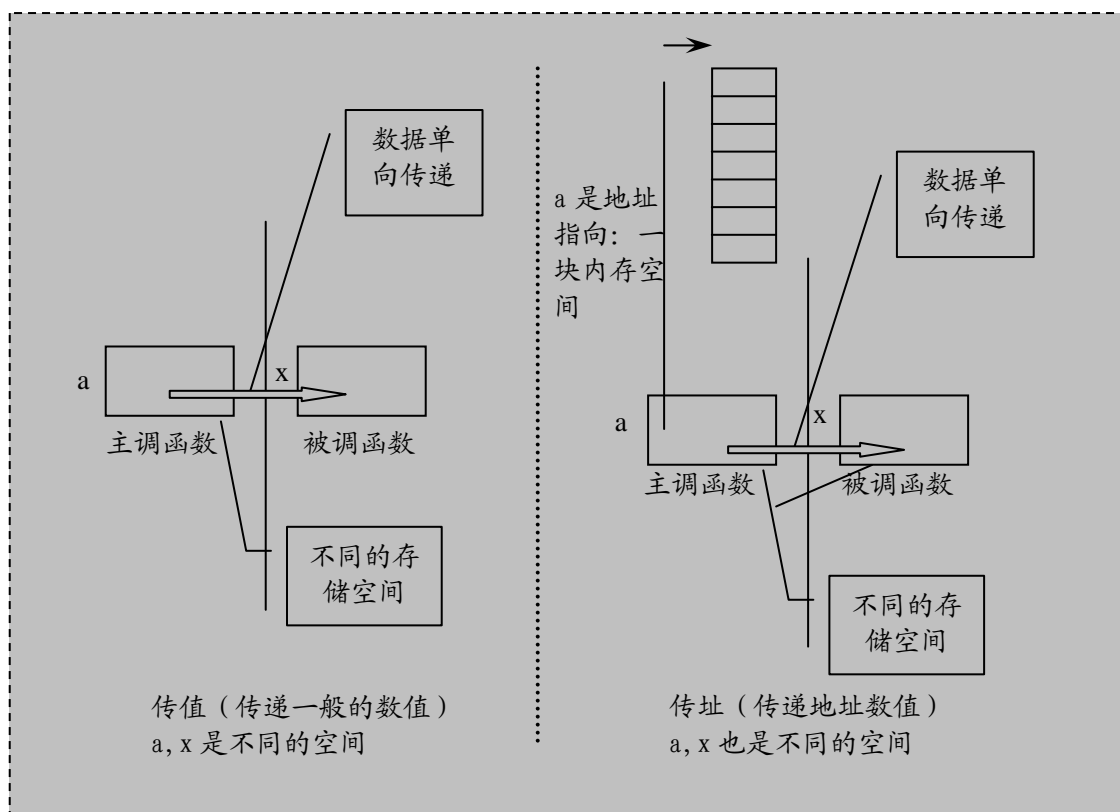


图 6.3 传值与传址方式比较

第二部分 函数调用及实例分析

一、函数的调用

1、函数调用的一般方法

函数名 ([实参表列]) [;]

【说明】

(1)无参函数调用没有参数，但是“()”不能省略，有参函数若包含多个参数，各参数用“，”分隔，实参参数个数与形参参数个数相同，类型一致或赋值兼容。

(2)函数调用可以出现的位置

- 以单独语句形式调用(注意后面要加一个分号，构成语句)。以语句形式调用的函数可以有返回值，也可以没有返回值。

例如：

```
printf("max=%d",nmax);
```

```
swap(x,y);
```

```
puts(s);
```

- 在表达式中调用(后面没有分号)。在表达式中的函数调用必须有返回值。

例如：

```
if(strcmp(s1,s2)>0).....//函数调用 strcmp ( ) 在关系表达式中。
```

```
nmax=max(n1,n2,n3) ; //函数调用 max ( ) 在赋值表达式中，
```

“；”是赋值表达式作为语句时加的，不是max函数调用的。

```
fun1(fun2()); //函数调用 fun2 ( )在函数调用表达式 fun1()中。
```

函数调用 fun2()的返回值作为 fun1 的参数。

2、函数调用时数据的传递（函数之间的通讯）

函数是相对独立的，但是不是孤立的，它们通过调用时

(1)参数传递

(2)函数的返回值

(3)全局变量（后面介绍）来相互联系。

已经介绍了函数参数的传递和返回值。现在以数组应用为例介绍参数传递中的传址方式的使用。

例 6.2：求学生平均成绩。(注意参数传递的是数组的地址)

【思考】如果在函数 ave 中加入程序段：

```
for(i=0; i<n1; i++)
```

```
    b[i]=sqrt(b[i])*10;
```

那么实参数组 a 中的数据是否改变？

3、函数的声明

函数定义的位置可以在调用它的函数之前，也可以在调用它的函数之后，甚至位于其它的源程序模块中。

(1)函数定义位置在前，函数调用在后，不必声明，编译程序产生正确的调用格式。

(2)函数定义在调用它的函数之后或者函数在其它源程序模块中，且函数类型不是整型，这时，为了使编译程序产生正确的调用格式，可以在函数使用前对函数进行声明。这样不管函数在什么位置，编译程序都能产生正确的调用格式。

函数声明的格式：

函数类型 函数名 ([参数类型][, ..., [参数类型]]);

C 语言的库函数就是位于其它模块的函数，为了正确调用，C 编译系统提供了相应的.h 文件。.h 文件内许多都是函数声明，当源程序要使用库函数时，就应当包含相应的头文件。

【实例分析一】

```
void swap(int x,int y)
{
    x=x+y,y=x-y,x=x-y;
    printf("swap: x=%d y=%d\n",x,y);
}
main()
{
    int a=3,b=5;
    swap(a,b);
    printf("main: a=%d b=%d\n",a,b);
}
结果为： swap: x=5 y=3 main: a=3 b=5
```

【实例分析二】

用数组元素作为参数(实际参数,形式参数为变量)

```
main()
{
    void swap(int a,int b);
    int a[]={4,9};
    printf("a[0]=%d\n",a[0]);
    printf("a[1]=%d\n",a[1]);
    swap(a[0],a[1]);
    printf("a[0]=%d a[1]=%d\n",a[0],a[1]);
}
void swap(int a,int b){
    int t;
    t=a,b=a,a=t;
}
结果为： a[0]=4 a[1]=9 a[0]=4 a[1]=9
```

二、函数的嵌套调用和递归调用

函数允许嵌套调用和递归调用。递归调用是嵌套调用的特例。

1、函数的嵌套调用

函数嵌套调用：函数调用中又存在调用。如函数 1 调用函数 2，函数又调用函数 3。函数之间没有从属关系，一个函数可以被其它函数调用，同时该函数也可以调用其它函数。

例 6.3：计算两整数的最小公倍数。P113.

思路：

(1)数学上：两数的最小公倍数=两数乘积/两数的最大公约数；两数的最大公约数-使用连除，取余法。

(2)N-S 流程

(3)程序

2、函数的递归调用

(1)函数的递归调用：是指函数直接调用或间接调用自己，或调用一个函数的过程中出现直接或间接调用该函数自身。前者称为直接递归调用，后者称为间接递归调用。

例如：->f1()->f1()直接递归调用；->f1()->f2()->f1()间接递归调用。

(2)使用递归调用解决问题的方法：（有限递归）

- 原有的问题能够分解为一个新问题，而新问题又用到了原有的解法，这就出现了递归。
- 按照这个原则分解下去，每次出现的新问题是原有问题的简化的子问题
- 最终分解出来的新问题是一个已知解的问题。

(3)递归调用过程（两个阶段）

- 递推阶段：将原问题不断地分解为新的子问题，逐渐从未知的向已知的方向推测，最终达到已知的条件，即递归结束条件，这时递推阶段结束。
- 回归阶段：从已知条件出发，按照“递推”的逆过程，逐一求值回归，最终到达“递推”的开始处，结束回归阶段，完成递归调用。

【实例分析一】求 n!

```
float fac(int n)
{
    float f=1;
    if(n<0)
    {
        printf("n<0,DataError");
        exit(1);
    }
}
```

```

    }
    if(n==0||n==1) f=1;
    else f=fac(n-1)*n;return(f);
}
main()
{
    int n;printf("n=?");
    scanf("%d",&n);
    printf("%d!=%-15.0f",n,fac(n));
}

```

【实例分析二】 要求不用循环语句（用直接递归），求 $1+2+3+\cdots+100$ （ n 属于正自然数）

```

long sum(int n)
{
    if(n>0)
        return(n+sum(n-1));
    else return(0);
}
main()
{
    printf("sum=%ld",sum(100));
}

```

【实例分析三】 要求不用循环语句（用直接递归），求 $1+2+3+\cdots+100$ （ n 属于正自然数）输入一字符串后，再反向输出（用直接递归）。

程序 1:

```

void conver(char *p)
{
    char q=*p;
    if(*p)
        conver(++p);
    printf("%c",q);
}
main()
{
    char *p="";
    printf("Input a string:");
    gets(p);
    conver(p);
}

```

程序 2:

```

#include "stdio.h" main()
{
    void gt(char);
    gt(getchar());
}

```

```

}
void gt(char s)
{
    if(s!='\n') gt(getchar());
    printf("%c",s);
}

```

【实例分析四】要求不用循环语句（用直接递归），求 $1+2+3+\cdots+100$ （ n 属于正自然数）输入一字符串后，再反向输出（用直接递归）。求 $1-100$ 的合计（用直接递归， n 大于 100，且由参数输入）。

程序 1:

```

main()
{
    float sum(int);
    printf("sum=%f",sum(1000));
}
float sum(int i)
{
    if(i>0)
        return(i+sum(i-1));
    else
        return(0);
}

```

程序 2: 实现系统函数 strcmp () 的功能

```

#include "string.h"
int sum=0;
void um(int i,int n)
{
    if(i<=n)
        um(sm(i),n);
}
int sm(int i)
{
    sum=sum+i;
    return(i+1);
}
int scmp(char a,char b)
{ /*实现系统函数 strcmp ( ) 的功能*/
    for(;*b&&*a;a++,b++)
        if(*a!=*b)
            break;
    return(*a-*b);
}
main()
{
    char a[]="China",b[]="American";
}

```

```

printf("strcmp(%s,%s)=%d\n",a,b,strcmp(a,b));
um(1,100);
printf("1+2+...+100=%d\n",sum);
}

```

【实例分析五】 求 $1! + 2! + \dots + n!$ （用直接递归）。

程序 1:

```

float a=1,m=0;
void sum(int i,int n)
{
    if (i<=n)
        sum(ss(i),n);
}
int ss(int i)
{
    a=a*i;m=m+a;
    return(i+1);
}
main()
{
    int i=1,n=10;
    sum(i,n);
    printf("1!+2!+....+%d!=%f",n,m);
}

```

程序 2:

```

float fac(int i)
{
    float p=1;
    int k=1;
    for(;k<=i;k++)
        p=p*k;
    printf("%d!=%f\n",i,p);
    return(p);
}
main()
{
    float sum=0;
    int i=1;
    for(;i<=10;i++)
        sum=sum+fac(i);
    printf("1!+2!+....+10!=%f",sum);
}

```

程序 3:

```

float pt(int n)
{
    if(n<=1)

```

```

        return(1);
    else
        return(n*pt(n-1));
}
float fat(int i)
{
    if(i<=1)
        return(1);
    else
        return(fat(i-1)+pt(i));
}
main()
{
    printf("1!+2!+....+10!=%f",fat(10));
}

```

程序 4:

```

float fac(int n)
{
    if(n>1)
        return(n*fac(n-1));
    else
        return(1);
}
float sum(int n)
{
    if(n>0)
        return(fac(n)+sum(n-1));
    else
        return(0);
}
main()
{
    int n=1;
    for(;n<20;n++)
        printf("%d!+",n);
    printf("%d!=%f\n",n,sum(20));
}

```

程序 5:

```

#define N 10
float fas(int n)
{
    static float p=1;
    p*=n;
    return(p);
}
float sm(int n)

```



```

{
    if(n<=N)
        return(fas(n)+sm(n+1));
    else
        return(0);
}
main(){
    int n=1;
    for(;n<N;n++)
        printf("%d!+",n);
    for(n=1;n<N;n++)
        printf("%d!+",n);
    printf("%d!=%f\n",n,sm(1));
}

```

三、变量的作用域（有效范围、可见性）

变量从数据类型的角度，可以分为整型、实型、字符型等。

变量的作用域：变量的有效范围或者变量的可见性。变量定义的位置决定了变量的作用域。

变量从作用域（变量的有效范围，可见性）的角度可以分为：局部变量，全局变量。

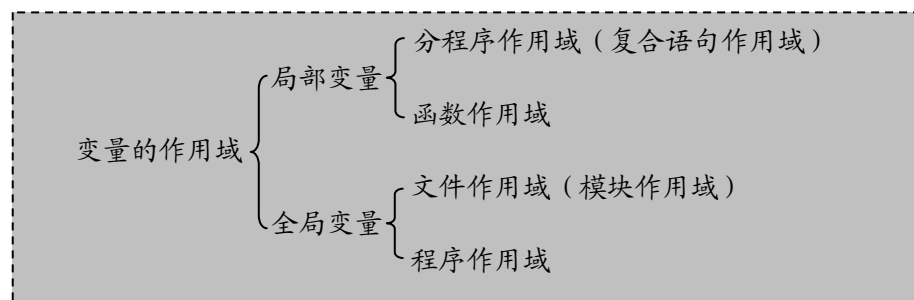


图 6.4 变量按作用域分类

1、局部变量

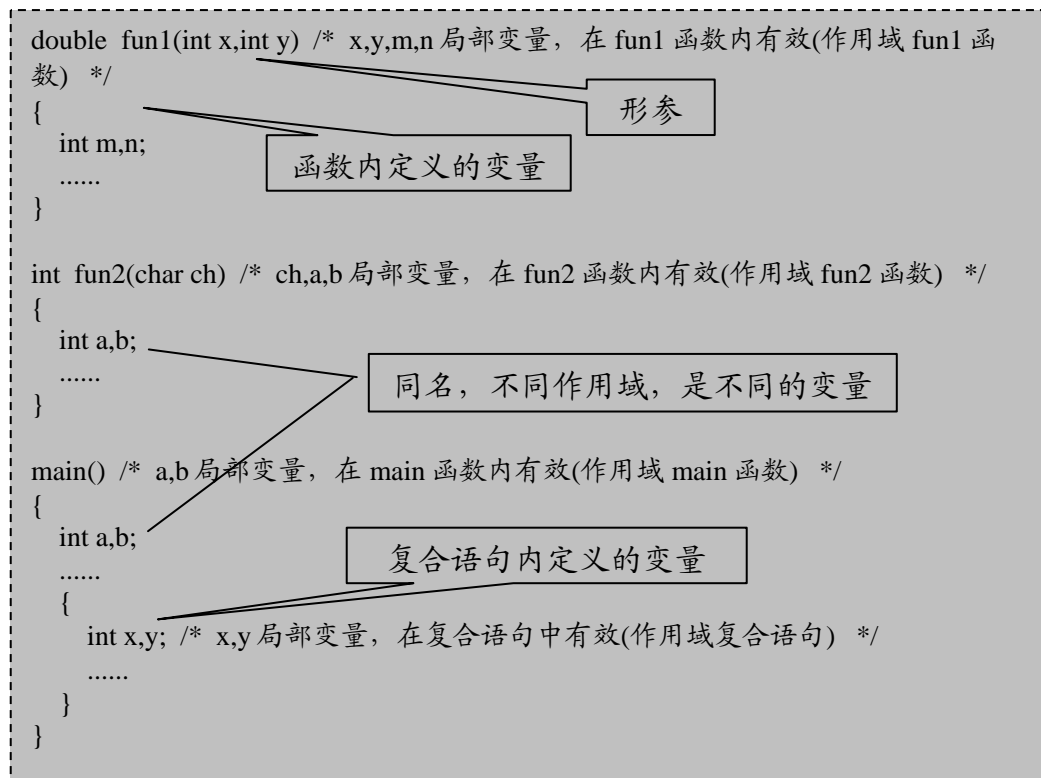
局部变量：是指在一定范围内有效的变量。C 语言中，在以下各位置定义的变量均属于局部变量。

(1)在函数体内定义的变量，在本函数范围内有效，作用域局限于函数体内。

(2)在复合语句内定义的变量，在本复合语句范围内有效，作用域局限于复合语句内。

(3)有参函数的形式参数也是局部变量，只在其所在的函数范围内有效。

例如:



程序 6.2 局部变量作用域示例

【说明】

(1)不同函数中和不同的复合语句中可以定义（使用）同名变量。因为它们作用域不同，程序运行时在内存中占据不同的存储单元，各自代表不同的对象，所以它们互不干预。即：同名，不同作用域的变量是不同的变量。

(2)局部变量所在的函数被调用或执行时，系统临时给相应的局部变量分配存储单元，一旦函数执行结束，则系统立即释放这些存储单元。所以在各个函数中的局部变量起作用的时刻是不同的。

2、全局变量

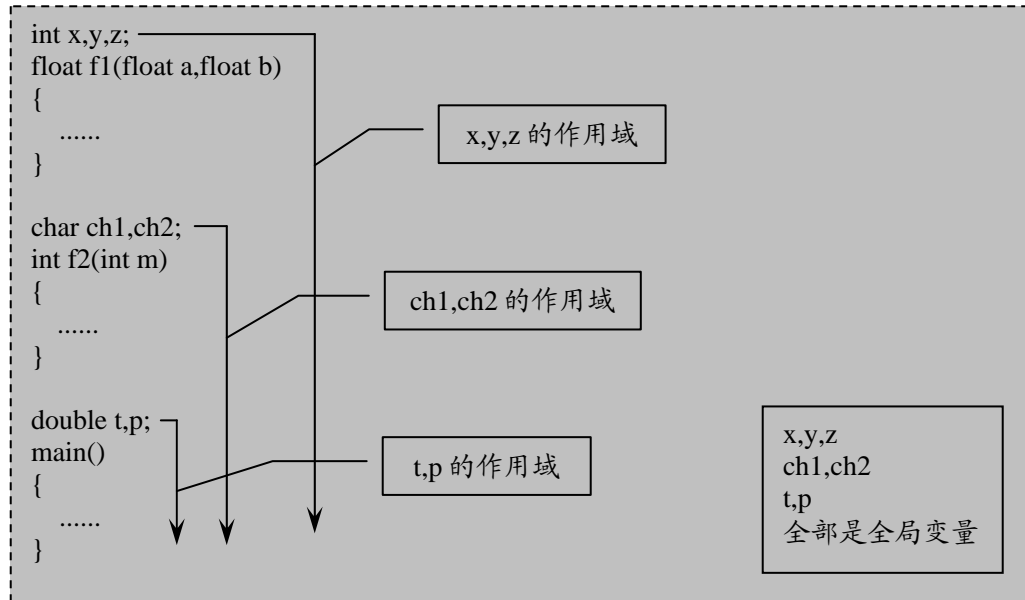
全局变量：在函数之外定义的变量。（所有函数前，各个函数之间，所有函数后）

全局变量作用域：从定义全局变量的位置起到本源程序结束为止。

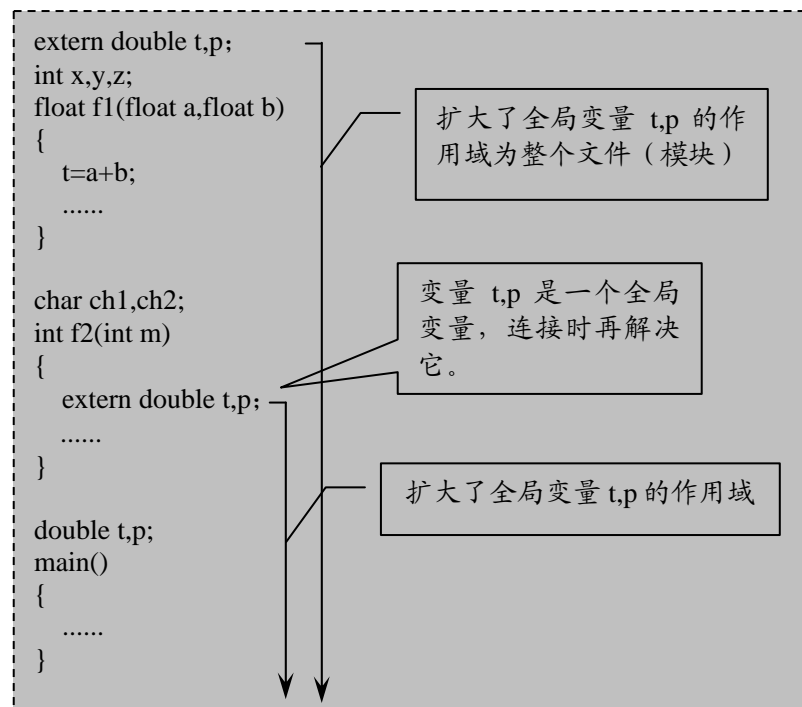
(1)在引用全局变量时如果使用“extern”声明全局变量，可以扩大全局变量的作用域。例如，扩大到整个源文件（模块），对于多源文件（模块）可以扩大到其它源文件（模块）。

(2)在定义全局变量时如果使用修饰关键词 `static`，表示此全局变量作用域仅限于本源文件（模块）。

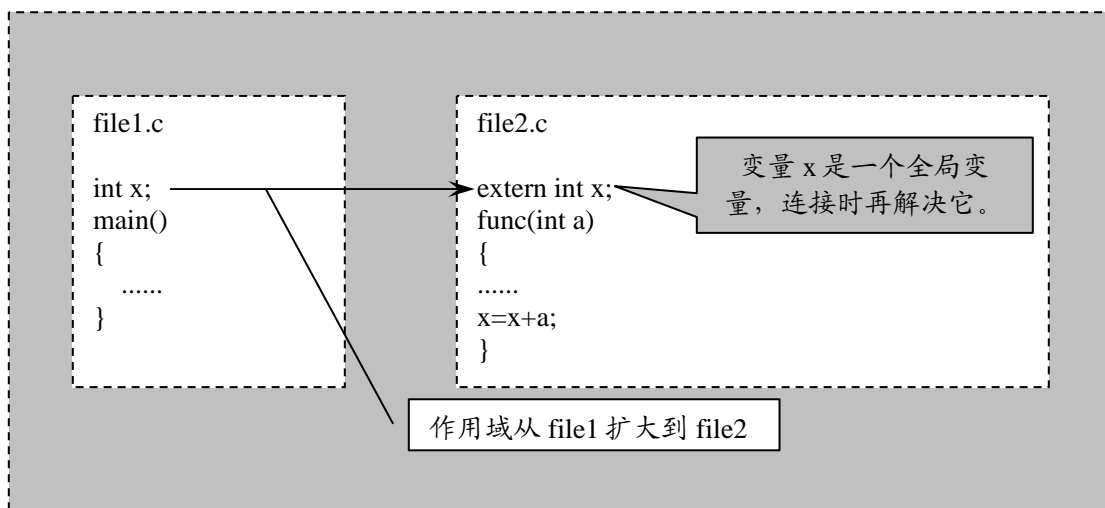
例如：



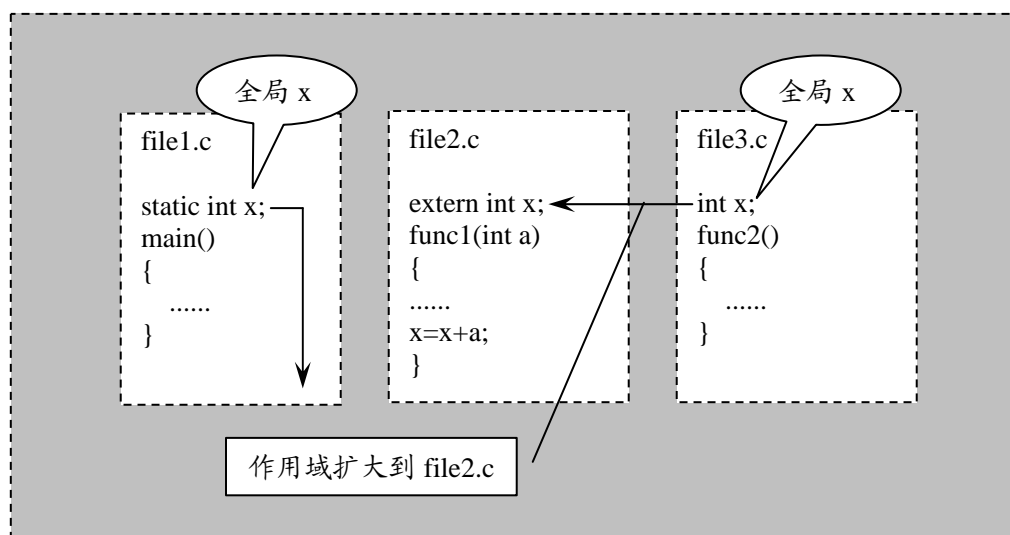
程序 6.3 全局变量作用域示例 1



程序 6.4 全局变量作用域示例 2



程序 6.5 全局变量作用域示例 3



程序 6.6 全局变量作用域示例 4

【说明】

(1)全局变量可以和局部变量同名，当局部变量有效时，同名全局变量不起作用。

(2)使用全局变量可以增加各个函数之间的数据传输渠道，在一个函数中改变一个全局变量的值，在另外的函数中就可以利用。但是，使用全局变量使函数的通用性降低，使程序的模块化、结构化变差，所以要慎用、少用全局变量。

第三部分 变量的存储及内部、外部函数

一、变量的存储类别（生存期、生命期）

变量从空间上分为局部变量、全局变量。

从变量存在的时间的长短（即变量生存期）来划分，变量还可以分为：动态存储变量、静态存储变量。变量的存储方式决定了变量的生存期。

C 语言变量的存储方式可以分为：动态存储方式、静态存储方式。

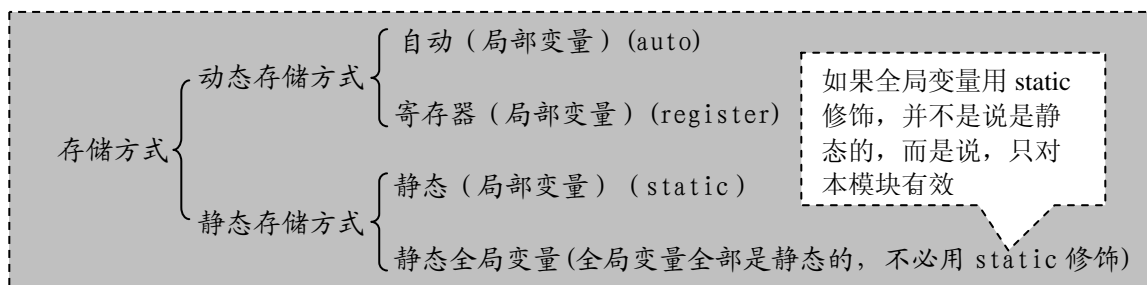


图 6.5 变量存储方式

1、动态存储方式

动态存储方式：在程序运行期间根据需要为相关的变量动态分配存储空间的方式。C 语言中，变量的动态存储方式主要有自动型存储方式和寄存器型存储方式。

(1)自动型存储方式（auto）

`auto` 型存储方式是 C 语言默认的局部变量的存储方式，也是局部变量最常使用的存储方式。

- 自动变量属于局部变量的范畴，作用域限于定义它的函数或复合语句内。
- 自动变量所在的函数或复合语句执行时，系统动态为相应的自动变量分配存储单元，当自动变量所在的函数或复合语句执行结束后，自动变量失效，它所在的存储单元被系统释放，所以原来的自动变量的值不能保留下来。若对同一函数再次调用时，系统会对相应的自动变量重新分配存储单元。

自动（局部）变量的定义格式：`[auto]` 类型说明 变量名；

其中：`auto` 为自动存储类别关键词，可以省略，缺省时系统默认 `auto`。

例如：

前面各章中的函数中的局部变量，尽管没有明确定义为 `auto` 型，但它们都属于 `auto` 型变量。

在函数中定义变量，下面两种写法是等效的。

`int x,y,z;`或 `auto int x,y,z;`它们都定义了 3 个整型 `auto` 型变量 `x,y,z`。

(2)寄存器型存储方式（register）

register 型存储方式是 C 语言使用较少的一种局部变量的存储方式。该方式将局部变量存储在 CPU 的寄存器中，寄存器比内存操作要快很多，所以可以将一些需要反复操作的局部变量存放在寄存器中。

寄存器（局部变量）的定义格式：[register] 类型说明 变量名；

其中：register 为寄存器存储类别关键词，不能省略。

【注意】：CPU 的寄存器数量有限，如果定义了过多的 register 变量，系统会自动将其中的部分改为 auto 型变量。

2、静态存储方式

静态存储方式：在程序编译时就给相关的变量分配固定的存储空间（在程序运行的整个期间内都不变）的存储方式。C 语言中，使用静态存储方式的主要有静态存储的局部变量和全局变量。

(1)静态存储的局部变量

静态局部变量的定义格式：[static] 类型说明 变量名[=初始化值]；

其中：static 是静态存储方式关键词，不能省略。

例如：在函数内定义：static int a=10,b;

【说明】

- 静态局部变量的存储空间是在程序编译时由系统分配的，且在程序运行的整个期间都固定不变。该类变量在其函数调用结束后仍然可以保留变量值。下次调用该函数，静态局部变量中仍保留上次调用结束时的值。
- 静态局部变量的初值是在程序编译时一次性赋予的，在程序运行期间不再赋初值，以后若改变了值，保留最后一次改变后的值，直到程序运行结束。

(2)全局变量全部是静态存储的。

C 语言中，全局变量的存储都是采用静态存储方式，即在编译时就为相应全局变量分配了固定的存储单元，且在程序执行的全过程始终保持不变。全局变量赋初值也是在编译时完成的。

因为全局变量全部是静态存储，所以没有必要为说明全局变量是静态存储而使用关键词 static。

如果使用了 static 说明全局变量，不是说“此全局变量要用静态方式存储”（我们知道全局变量天生是静态存储的），那是有另外的含义-令人困惑的全局变量的 static 说明。

(3)全局变量的 extern 声明及令人困惑的全局变量的 static 定义。

全局变量的 `static` 定义，不是说明“此全局变量要用静态方式存储”（全局变量天生全部是静态存储），而是说，这个全局变量只在本源程序模块有效（文件作用域）。

如果没有 `static` 说明的全局变量就是整个源程序范围有效（真正意义上的全局）。也就是说，变量的作用域有：分程序（复合语句）作用域，函数作用域，文件（模块）作用域，整个程序作用域。

在引用全局变量时如果使用“`extern`”声明全局变量，可以扩大全局变量的作用域。例如，扩大到整个源文件（模块），对于多源文件（模块）可以扩大到其它源文件（模块）。

二、内部函数和外部函数

类似于全局变量的作用域，函数定义时也可以用 `static` 修饰。

使用 `static` 修饰的函数是外部函数；

不使用 `static`（或用 `extern`）修饰的函数是内部函数。

1、内部函数

内部函数：只能被本源文件（模块）中的各个函数所调用，不能为其它模块中函数所调用的函数。内部函数的定义：

```
static 函数类型 函数名（形参表）  
{  
    .....  
}
```

【说明】

(1)内部函数又称为静态函数，其使用范围仅限于定义它的模块（源文件）内。对于其它模块它是不可见的。

- 有一些涉及机器硬件、操作系统的底层函数，如果使用不当或错误使用可能造成问题。为避免其它程序员直接调用，可以将此类函数定义为静态函数，而开放本模块的其它高层函数，供其它程序员使用。
- 还有一种情况就是，程序员自己认为某些函数仅仅是程序员自己模块中其它函数的底层函数，这些函数不必要由其它程序员直接调用。此时也常常将这些函数定义为静态函数。

(2)不同模块中的内部函数可以同名，它们的作用域不同-事实上根本就是不同的函数。

(3)内部函数定义，`static` 关键词不能省略。

2、外部函数

外部函数：能被任何源文件（模块）中的任何函数所调用的函数。

外部函数的定义：

```
[extern] 函数类型 函数名（形参表）  
{  
    .....  
}
```

【说明】

(1)外部函数定义，extern 关键词可以省略。如果省略，默认是外部函数。

(2)外部函数可以在其它模块中被调用。如果需要在某个模块中调用它，可以在模块中某个位置声明 extern 函数类型 函数名（形参表）；就可以了。

三、函数应用举例

【实例分析】 利用插入法将 10 个字符从小到大进行排序

排序方法：冒泡法（大数沉底），选择法，插入排序。

1、“插入排序”的过程，如图 6.6 所示。

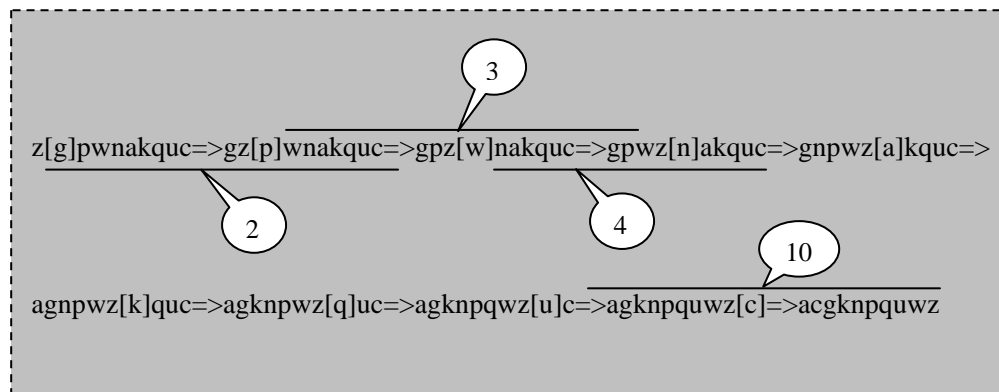


图 6.6 冒泡排序过程

[]号中的字符是要插入前面已排序序列的字符。

(1)第 1 个字符本身可以认为已经排好序。

(2)取第 2 个字符，在已经排序的第 1 个字符适当位置将第 2 个字符插入，保证这 2 个字符由小到按序排列。

(3)取第 3 个字符，在已经排序的第 1，第 2 个字符适当位置将第 3 个字符插入，保证这 3 个字符由小到按序排列。

.....

(4)取第 i 个字符,在已经排序的第 1, 2, ... $i-1$ 个字符适当位置将第 i 个字符插入,保证这 i 个字符由小到大按序排列。

.....

(5)取第 N 个字符,在已经排序的第 1, 2, ... $N-1$ 个字符适当位置将第 N 个字符插入,保证这 N 个字符由小到大按序排列。至此排序结束。

2、“插入排序”分析:

(1)从第 2 字符 $a[1]$ 到最后一个字符 $a[N-1]$,共需要进行 $N-1$ 次字符插入。可以用循环(外层循环)完成这 $N-1$ 次插入。

(2)每次插入字符 $a[i]$,要完成两项工作:

- 在 $a[i]$ 前面已经排序的字符序列($a[0] \sim a[i-1]$)中找到合适的插入位置(插入点)。
- 将 $a[i]$ 插入。

(3)在已经排序的字符序列($a[0] \sim a[i-1]$)中查找合适的插入位置(插入点):

因为 $a[0] \sim a[i-1]$ 已经由小到大按序排列,所以可以将 $a[i]$ 与前面的字符 $a[j](j=i-1 \sim 0)$ “从后向前”进行比较查找:

如果 $a[i] \geq a[j]$,那么插入位置找到,停止查找(此时, $a[i]$ 应当插入到 $a[j]$ 后);否则继续向前进行比较查找,直到 $j < 0$ 时停止查找(所有 $a[j]$ 均大于 $a[i]$,此时应该在字符序列最前面插入)。可以看出查找插入点的过程也是一个重复的过程,所以也使用循环完成(内层循环)。

(4)用程序完成“插入排序”时,在查找插入点的同时,同时可能进行数据移动,为插入的数据准备空间。边查找,边准备空间,一旦确定位置,直接将数据写入该位置。

3、N-S 流程图,如图 6.7 所示:

```
#define N 10
```

i : 欲插入的字符的下标(位置)。

j : 插入字符的位置(先搜索插入位置,搜索到以后,在该位置后即 $j+1$ 位置插入字符)

4、程序

```
#define N 10
void insert(char s[])
{
    int i,j,t;
```

```

for(i=1; i<=N-1; i++)
{
    t=s[i];
    j=i-1;
    while((j>=0)&& t<s[j])
    {
        s[j+1]=s[j];
        j--;
    }
    s[j+1]=t;
}
}
main()
{
char a[N+1]; int i;

printf("Input 10 char:");
for(i=0; i<N; i++)
    a[i]=getchar();
a[i]='\0';

insert(a);

printf("This is 10 char:%s\n",a);
}

```

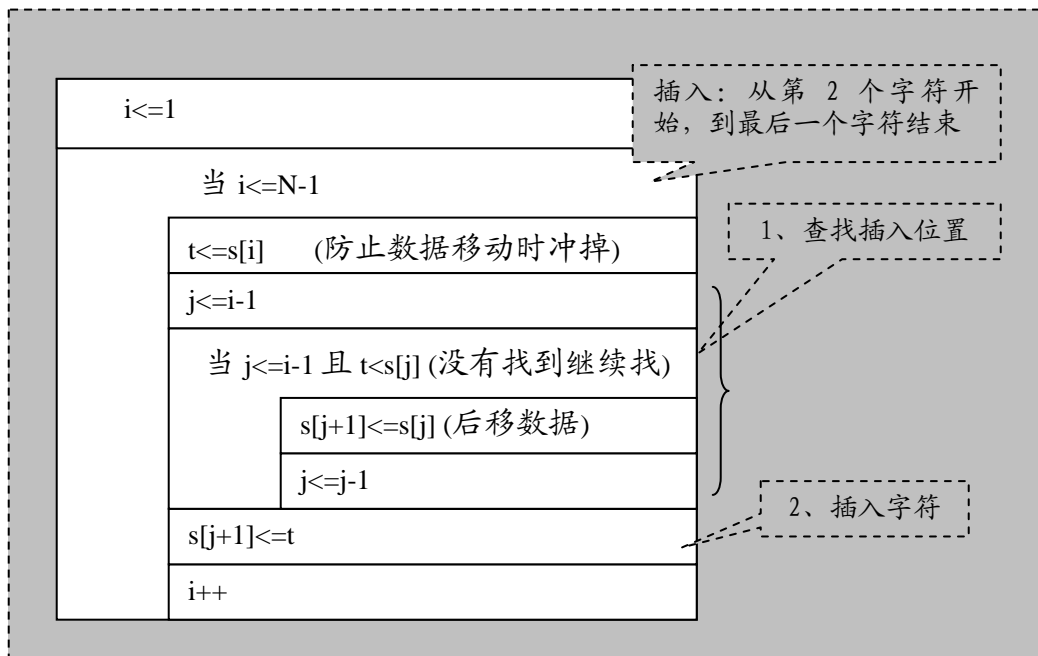


图 6.7 冒泡排序 N-S 图

第七章

指针

【基本要求】

通过本单元的学习，使学生了解指针的基本概念，掌握指针变量的定义，学会指针变量的赋值，熟练掌握指针变量的引用，及指针变量作为函数参数的情况。

【教学重点】

指针、变量的指针、指针变量、指针的指针（定义、初始化、引用、应用、），一维数组的指针、指向一维数组的指针变量（定义、引用、应用），多维数组的指针、指向多维数组的指针变量、指针数组（定义、引用、应用），函数的指针、函数指针变量、指针类函数（定义、调用、应用），指针作为函数参数、主函数的参数与返回，指针与数组的区别。

【本章结构】

- 1、变量的指针和指向变量的指针变量
 - 定义一个指针变量
 - 指针变量的引用
 - 指针变量作为函数参数
- 2、数组与指针
 - 指向数组元素的指针
 - 通过指针引用数组元素
 - 用数组名作为函数参数
 - 多位数组与指针
- 3、字符串与指针
 - 字符串的表示形式
 - 字符指针作为函数参数
 - 字符指针变量和字符数组
- 4、指向函数的指针
 - 用函数指针变量调用函数
 - 指向函数指针作函数参数
 - 返回指针值的函数
- 5、指针数组和指向指针的指针
 - 用函数指针变量调用函数
 - 指向函数指针作函数参数
 - 返回指针值的函数

6、有关指针的数据类型和指针运算的小结

指针数据类型小结
指针运算小结
void 指针类型

第一部分 指针变量及其操作

指针使用灵活、方便，并可以使程序简洁、高效、紧凑。可以说，指针是 C 语言的精髓。

指针涉及数据的物理存储，概念复杂，使用灵活且容易出错，所以较难掌握。学习指针要抓住基本的概念、多用图示分析问题解决问题。

简单地说，地址（内存空间或变量的）-指针；地址变量-指针变量

一、变量的地址和指针变量

1、地址（指针）、地址变量（指针变量）概念及变量的存取方式

内存，内存地址（物理存储器的概念-补充）

内存（内部存储器）：是由大规模集成电路芯片组成存储器，包括 RAM、ROM。运行中的程序和数据都是存放在内存中的。与内存相对的是外存，外存是辅助存储器（包括软盘、硬盘、光盘），一般用于保存永久的数据。一定要记住：程序、数据是在内存中由 CPU 来执行和处理的。外存上尽管可以保存程序和数据，但是当这些数据在没有调入内存之前，是不能由 CPU 来执行和处理的。

内存地址：内存是由内存单元（一般称为字节）构成的一片连续的存储空间，每个内存单元都有一个编号。内存单元的编号就是内存地址，简称地址。

CPU 是通过内存地址来访问内存，进行数据存取（读/写）。

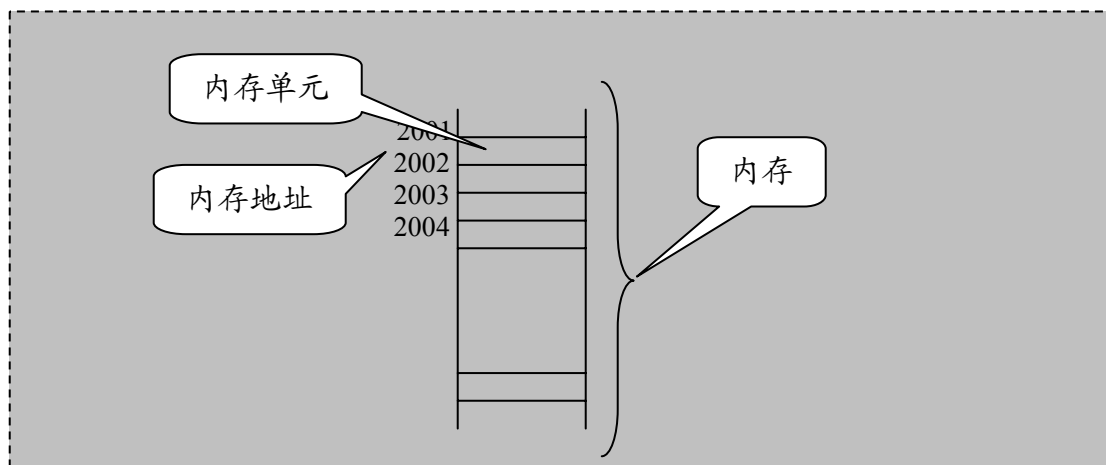


图 7.1 内存示意图

2、变量、变量名、变量的地址、变量值（高级语言的存储器概念-复习）

变量：命名的内存空间。变量在内存中占有一定空间，用于存放各种类型的数据。

变量有下面几个概念：

变量名：变量名是给内存空间取的一个容易记忆的名字。

变量的地址：变量所使用的内存空间的地址。

变量值：在变量的地址所对应的内存空间中存放的数值即为变量的值或变量的内容。

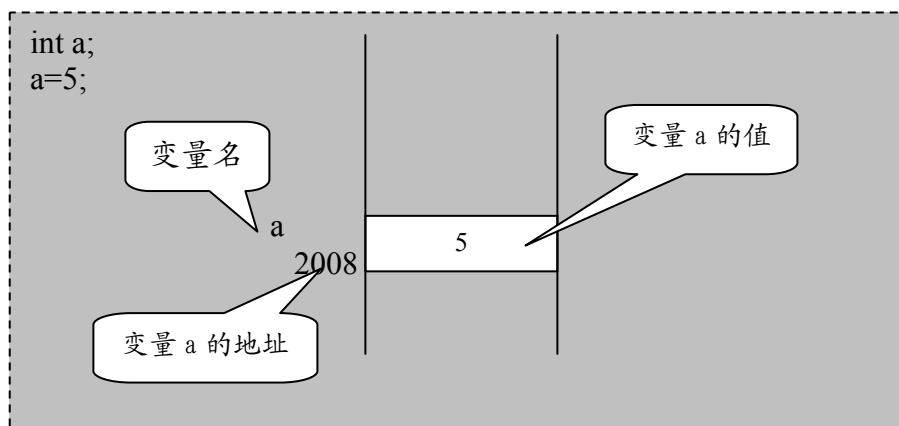


图 7.2 变量名、变量值及变量地址

3、指针、变量的指针和指针变量（重点）

指针：就是“内存单元的地址”。指针指向一个内存单元。

变量的指针：就是“变量的地址”。变量的指针指向一个变量对应的内存单元。

指针变量：就是地址变量。地址（指针）也是数据，可以保存在一个变量中。
保存地址（指针）数据的变量称为指针变量。

指针变量 p 中的值是一个地址值，可以说指针变量 p 指向这个地址。如果这个地址是一个变量 i 的地址，则称指针变量 p 指向变量 i 。指针变量 p 指向的地址也可能仅仅是一个内存地址。

【注意】

(1)牢记：指针-地址。指针变量-地址变量。

(2)指针变量是变量，它也有地址，指针变量的地址-指针变量的指针（指针的指针）。

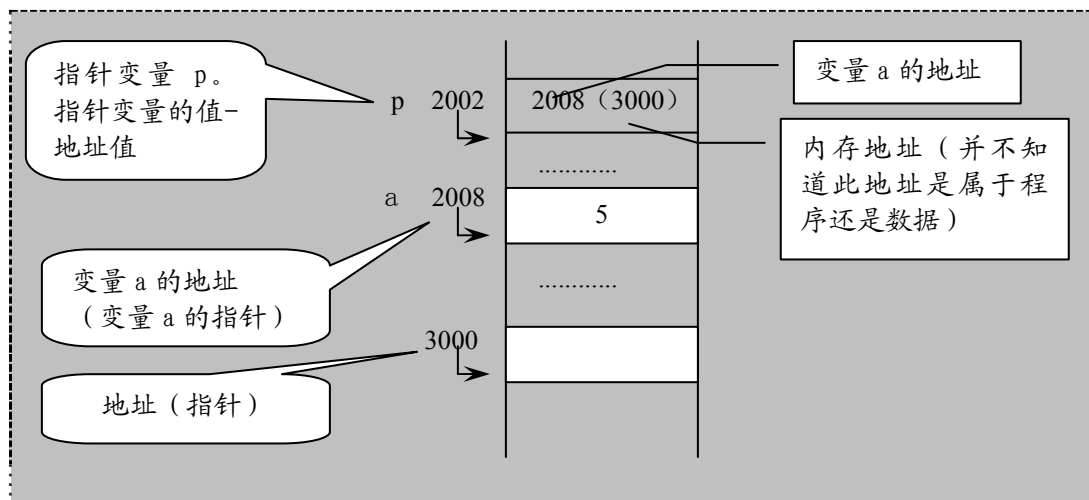


图 7.3 变量示意图

4、系统访问变量的两种方式（变量的存取方式）（重点）

(1) 直接访问：按地址存取内存的方式称为“直接访问”

● 按变量名直接访问,按变量地址直接访问。 如: `a=3; *(&a)=3;`

从系统的角度看不管是按变量名访问变量，还是按变量地址访问变量,本质上都是对地址的直接访问。用变量名对变量的访问属于直接访问，因为编译后，变量名-变量地址之间有对应关系，对变量名的访问系统自动转化为利用变量地址对变量的访问。

● 按地址直接访问（访问内存存储空间）-一般不要这么使用，这里仅为了说明概念。

如: `*((int*)(2008))=5;` 在地址 2008 保存一个整数。

`farptr=(char far *)0xA0000000; *(farptr+offs+i1*MAXB)=*farbit;` 操作显示内存。

(2) 间接访问（使用指针变量访问变量）

将变量 a 的地址（指针）存放在指针变量 p 中，p 中的内容就是变量 a 的地址，也就是 p 指向 a，然后利用指针变量 p 进行变量 a 的访问。如图 7.4 所示。

`p=&a, *p=3。`

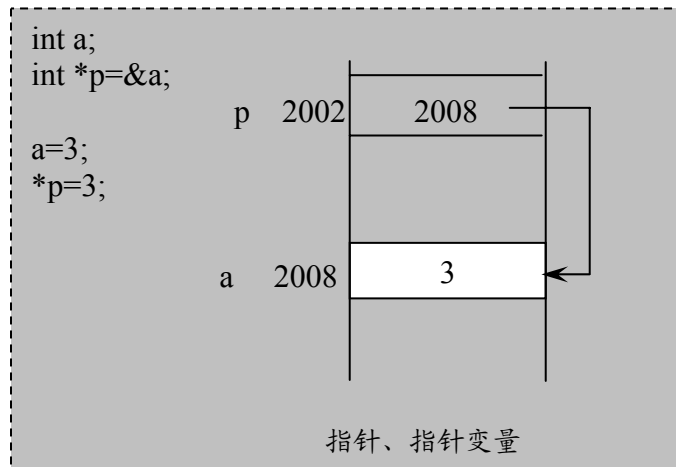


图 7.4 指针、指针变量

从变量名获得变量地址用“&”运算符，从地址获得地址指向的数据用“*”运算符。

二、指针变量的操作

1、指针变量的定义

指针变量的定义格式：基类型 *变量名；

例如：

int *pt1, *pt2; 定义两个指针变量 pt1, pt2。基类型为整型，即指向的数据类型为整型。

float *f; 定义指针变量 f。基类型为浮点型，即指向的数据类型为浮点型。

char *pc; 定义 pc。基类型为字符型，即指向的数据类型为字符型。

【说明】

(1) C 语言变量先定义后使用，指针变量也不例外，为了表示指针变量是存放地址的特殊变量，定义变量时在变量名前加“*”号。

(2) 指针变量的基类型（简称：指针变量类型）：指针变量所指向数据的类型。我们知道，整型数据占用 2 个字节，浮点数据占用 4 个字节，字符数据占用 1 个字节。指针变量类型使得指针变量的某些操作具有特殊的含义。比如，pt1++；不是将地址值增 1，而是表示将地址值+2（指向后面一个整数）。

(3) 指针变量存放地址值，PC 机用 2 个字节表示一个地址，所以指针变量无论什么类型，其本身在内存中占用的空间是 2 个字节。

sizeof(pt1)=sizeof(f)=sizeof(pc)=2。

2、指针变量的赋值

指针变量一定要有确定的值以后，才可以使用。禁止使用未初始化或未赋值的指针（此时，指针变量指向的内存空间是无法确定的，使用它可能导致系统的崩溃）。

指针变量的赋值可以有两种方法：

(1)将地址直接赋值给指针变量（指针变量指向该地址代表的内存空间）

例如：

`char far *farptr=(char far *)0xA0000000;` 将显存首地址赋值给指针。

`float *f=(float *)malloc(4);` `malloc` 动态分配了 4 个字节的连续空间，返回空间首地址，然后将首地址赋值给浮点型指针 `f`。这样浮点型指针 `f` 指向这个连续空间的第一个字节。

(2)将变量的地址赋值给指针变量（指针变量指向该变量）

例如：

`int i,*p; p=&i;`

3、指针变量的引用(如程序 7.1 所示)

`&`、`*`运算符。注意是用在指针变量上的，不是“位与”，“乘”运算符。

(1) `&`运算符（取地址运算符）：表示取变量的地址。

(2) `*`运算符（指针运算符、间接访问运算符）：访问指针变量指向的变量的值。

例 7.1 指针变量的引用

```
main()
{
    int i=100,j=10; /* 定义两个整型变量 */
    int *pi,*pj; /* 定义两个整型指针变量 */

    pi=&i; /* pi->i */
    pj=&j; /* pj->j */

    printf("%d,%d",i,j); /* 直接访问变量 i,j */
    printf("%d,%d",*pi,*pj); /* 间接访问变量 i,j */
}
结果：
10, 100
10, 100
```

程序 7.1 指针变量的引用

【说明】

(1) “`int *pi,*pj;`”语句定义了两个变量 `pi,pj` 是指向整型数据的指针变量，但是没指定它们具体指向哪个变量。

(2) “pi=&i; pj=&j;” 将 i,j 的地址分别赋值给指针变量 pi,pj。也就是说 pi,pj 分别指向变量 i,j。

(3) “printf(“%d,%d”,i,j);” 使用变量名 i,j 直接访问变量 i,j，这是我们以前常用的方法。

(4) “printf(“%d,%d”,*pi,*pj);” 使用 *pi,*pj 访问 pi,pj 所指向的空间的内容，pi,pj 分别指向变量 i,j，所以 *pi,*pj 访问的是变量 i,j 的值。即使用指针变量 pi,pj 间接访问变量 i,j。“*运算符”是间接访问运算符（与定义时不同，定义指针变量所使用的“*”只表示是指针变量，不是运算符）。

【其他的指针使用方面的说明】

(1)指针（指针变量）加 1，不是指针的地址值加 1，而是加 1 个指针基类型的字节数。P137.

(2)指针运算符“*”不仅可以用在指针变量上，也可以使用在任何能够获取地址（指针）的表达式上。

如果 a<=100,那么&a 就是 a 的地址，也就是 *&a=100。

如果 pa<=&a,那么取 pa 所指向变量的内容就是变量 a 的内容，*pa=a,也就是 &*pa=&a。

4、指针变量作为函数的参数

变量可以作为函数参数，指针变量同样可以作为函数参数。

指针变量作为函数参数时，同样是从实参单向传递指针变量的内容给形参，只是传递的内容是一个地址值。可以通过这个地址值间接改变实参、形参所共同指向的变量。所以尽管不能改变实际参数地址本身，但是可以间接改变地址所指向的变量。

例 7.2: 输入 a,b; 交换 a,b 数据后输出。如程序 7.2 所示。

【说明】

(1)本例算法很简单，交换两个数据，本例子的主要目的是演示指针作为函数参数的应用。

(2)先分析函数参数使用一般变量的情况。被调用函数 swap 中交换了形参 i,j 的值，但是因为参数传递是单向的，形参、实参占用的是不同的内存空间，所以在尽管在 swap 中交换了形参 i,j 值，实参 a,b 不会改变。程序可以编译通过，但结果并不是我们所希望的。

```

void swap(int i,int j)
{
    int temp;
    temp=i; i=j; j=temp; /* exchange i,j */
}

main()
{
    int a=5,b=10;

    swap(a,b);

    printf("swaped:\n");
    printf("a=%d,b=%d\n",a,b);
}

```

程序 7.2 例 7.2 程序

(3)再分析函数参数使用指针变量，在函数中交换指针变量值的情况。被调用函数 swap 中交换了形参指针变量 pi,pj 的值(地址)，但是因为同样是参数的单向传递，形参、实参占用的是不同的内存空间，所以在尽管在 swap 中交换了形参指针变量 pi,pj 值，实参指针变量 pa,pb 不会改变,还是分别指向 a,b。程序可以编译通过，但结果也不是我们所希望的。

```

void swap(int *p1,int *p2)
{
    int *ptemp;
    ptemp=p1; p1=p2; p2=ptemp; /* exchange pi,pj */
}

main()
{
    int a=5,b=10;
    int *pa,*pb;
    pa=&a; pb=&b;

    swap(pa,pb);

    printf("swaped:\n");
    printf("a=%d,b=%d\n",*pa,*pb);
}

```

程序 7.3 使用指针作为函数参数

(4)最后分析函数参数使用指针变量，在函数中交换指针变量所指向的变量的值情况。

被调用函数 swap 中通过参数传递获得了实参指针变量指向的变量地址，此时形参指针变量 p1,p2 也已经分别指向实参指针变量所指向的变量 a,b。也就是说实参、形参指针变量指向共同的变量 a,b。在函数 swap 中可通过形参指针变量间接访问、修改形参、实参指针所共同指向的变量 a,b，本例是交换了形参指针变量 p1,p2 所指向的 a,b。返回 main()函数后 pa,pb 仍然指向 a,b。但是 a,b 的值已经交换。

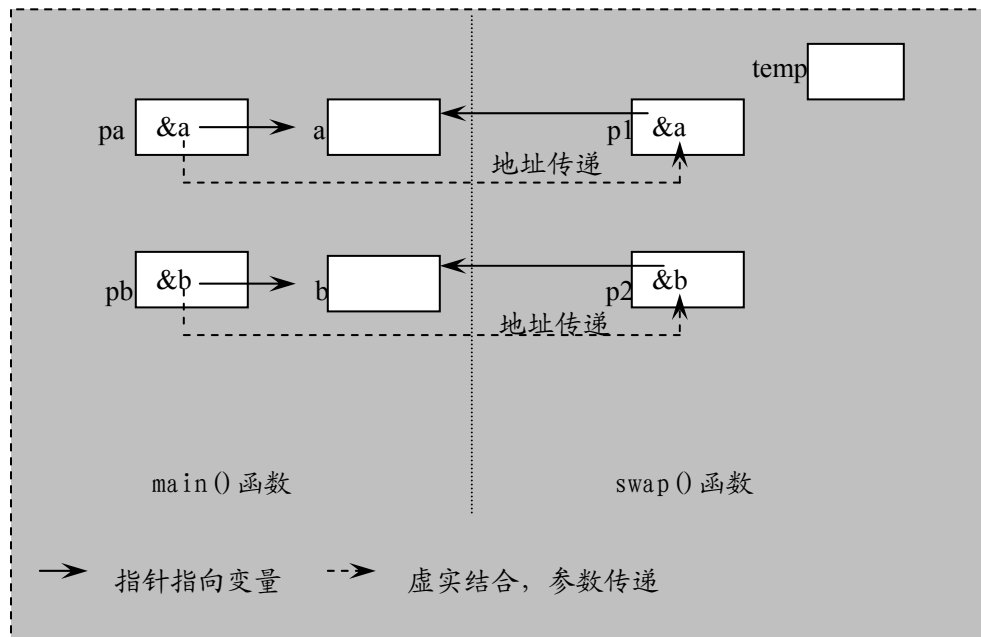


图 7.5 地址传递示意图

```
void swap(int *p1,int *p2)
{
    int temp;
    temp=*p1; *p1=*p2; *p2=temp;  /* exchange *p1,*p2 */
}

main()
{
    int a=5,b=10;
    int *pa,*pb;
    pa=&a; pb=&b;

    swap(pa,pb);

    printf("swaped:\n");
    printf("a=%d,b=%d\n",*pa,*pb);
}
```

程序 7.4 地址传递程序

【结论】 结论：要在被调用函数中，修改主调函数的变量值应当：

(1)将主调函数变量的地址传递给被调用函数，就是说函数应当传递的是变量的地址。这样被调用函数的形参应当使用指针变量接受主调函数的地址值。

(2)在被调用函数中通过形参指针变量间接访问，修改实参、形参地址所共同指向的变量。本例的操作是交换两个指针变量所指向的变量。

```
//输入 a,b; 交换 a,b 数据后输出

void swap(int *p1,int *p2)交换 p1,p2 指向的两个整数
{
    int temp,*ptemp;
    temp=*p1; *p1=*p2; *p2=temp; /* exchange *p1,*p2 */
    ptemp=p1; p1=p2; p2=ptemp; /* exchange p1,p2 */
}

main()
{
    int a,b;
    int *pa,*pb;
    pa=&a,pb=&b;

    scanf("%d%d",pa,pb); /* &a,&b */
    printf("a=%d,b=%d\n",a,b); /* *pa,*pb */
    printf("pa=%d,pb=%d\n",pa,pb);

    swap(pa,pb); /* or: swap(&a,&b); */

    printf("swaped:\n");
    printf("a=%d,b=%d\n",a,b);
    printf("pa=%d,pb=%d\n",pa,pb);
}
```

程序 7.5 例 7.2 程序

第二部分 数组指针和字符串指针

一、数组的指针和指向数组的指针变量

数组：相同类型元素构成的有序序列。

数组元素的指针：数组元素在内存中占据了一组连续的存储单元，每个数组元素都有一个地址，数组元素的地址就是数组元素的指针。

数组的指针：就是数组的地址。数组的地址指的是数组的起始地址（首地址），也就是第一个数组元素的地址。C 语言还规定数组名代表数组的首地址。

例如：对于整型数组 `int a[10]`；数组的指针就是数组的起始地址`&a[0]`，也可以用数组名 `a` 表示。如图 7.6 所示。

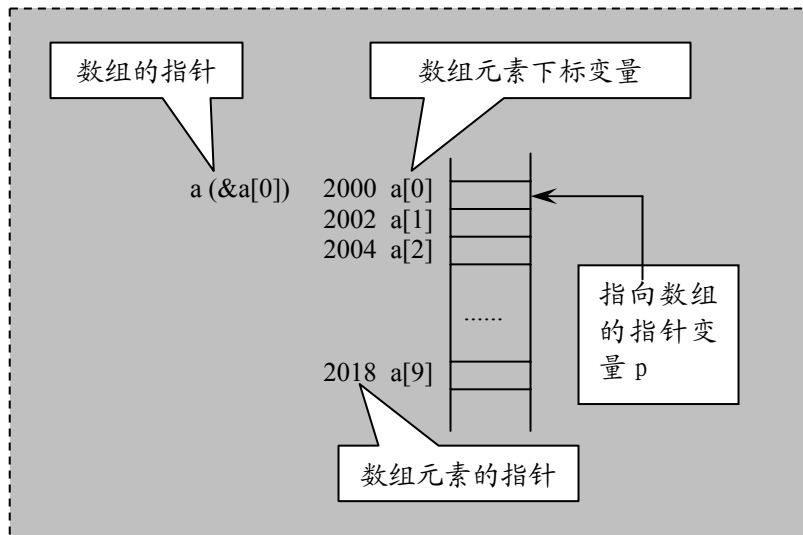


图 7.6 数组的指针

1、指向数组的指针变量(指向数组元素的指针变量)

(1)指向数组的指针变量：存放数组元素地址（初始时一般为数组首地址）的变量，称为指向数组的指针变量（简称：数组的指针变量）。

(2)数组的指针变量的定义和初始化：

数组基类型 `*p`;

`p=数组名; /* p=&数组名[0] */`

或：

数组基类型 `*p=数组名`;

【说明】

(1)数组的指针变量的定义与数组元素的指针变量的定义相同。实质就是基类型指针变量的定义。

例如：`int a[10],*p;` 定义了一个整型数组 `a`，如果需要定义指向该数组的指针变量就要定义一个整型指针变量 `p`。

(2)数组的指针变量的初始化可以用两种方法：

- 定义时初始化，可以使用已经定义的数组的数组名来初始化数组的指针变量。
- 通过赋值初始化，将数组的首地址赋值给数组的指针变量（数组的指针变量的赋值也与一般的指针变量的赋值相同）。

例如：

int a[10],*p; 定义了一个整型数组 a, 一个整型指针变量 p。

p=a; 或者 p=&a[0]; 将数组 a 的首地址赋值给整型变量 p,此时 p 就是指向数组的指针变量。

也可以:

int a[10],*p=a; 在定义数组的指针变量 p 的同时初始化指向已经定义的数组 a。

2、通过指针（数组的指针、数组的指针变量）引用数组元素

(1)指针 p+i 的含义（复习）：不是地址值 p 增加 i 个字节后的地址值，而是指 p 向后移动 i 个基类型元素后的地址值。p-i,p++, p--都有类似的含义。

(2)指针与数组的关系

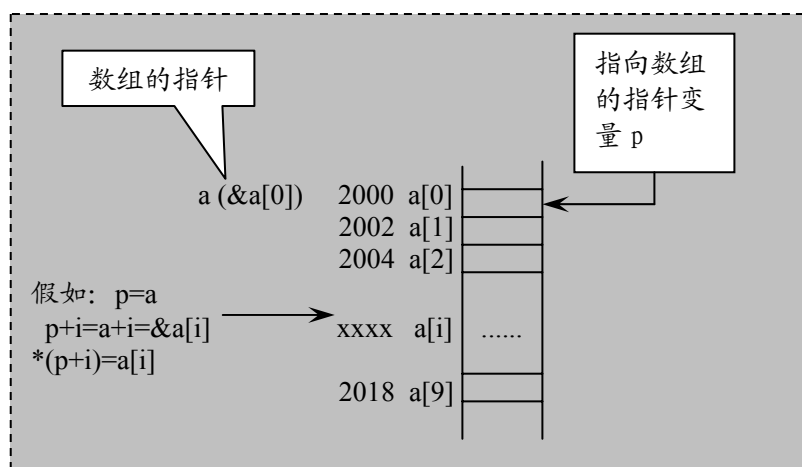


图 7.7 指针与数组的关系

数组元素在内存中连续存放，如果指针 p 指向数组 a，那么，p+i 指向数组 a 的第 i 个元素 a[i]。也就是 p+i=&a[i]，此时对 a[i] 的访问完全可以转化为对 *(p+i) 的访问。数组与指针的关系：数组元素可以用下标访问也可以使用指针访问。

3、通过指针引用数组元素

(1)前面的章节都是通过下标（索引）来访问数组元素的，数组元素的访问还可以通过指针完成。

(2)数组元素的地址表示。

假如：p 定义为指向数组 a 的指针。数组元素 a[i] 的地址可以表示为：
&a[i],p+i,a+i。

(3)数组元素的访问

例如：数组元素 a[i] 的访问可以是：a[i],*(p+i),*(a+i)。

数组指针变量，数组名在许多场合甚至可以交换使用。

假如：p=a,那么 a[i]甚至可以表示为 p[i]（指针变量带下标）

【说明】

数组名，数组指针变量使用时的区别：数组名是常量指针，它指向数组首地址，数组指针变量是变量，它的值可以改变。在不至于混淆的场合，数组名，数组指针变量可以统称数组指针。

例如：假设 a、b 是数组名，p 是同类型的数组指针变量。

a++; *(a++); a=a+i; a=b; 错误

而 p++; *(p++); p=p+i; p=a; 都是可以的。

【结论】 引用数组元素可以采用两种方法：(假设 p=a)

(1)下标法：通过数组元素的序号（索引）来访问数组元素。如 a[i]或 p[i]。

(2)指针法：通过数组元素的地址访问数组元素。如*(a+i),*(p+i)

4、数组名作为函数参数

例子：（复习）

<pre>main() { int a[10]; sort(a,10); }</pre>	<pre>sort(int x[],int n) { }</pre>
--------------------------------------------------------------------------	----------------------------------------------

程序 7.6 参数传递示例

sort（）函数对数组元素进行排序。main（）函数调用 sort（）函数时将实参数组 a 的首地址 a，单向传递给形参数组 x，形参数组的数组名 x 接受了实参数组 a 的首地址，也就是形参数组也指向了 main（）函数中数组 a。sort（）函数中对数组 x 的操作也就是对数组 a 的操作。sort()函数排序完成返回 main()函数后，a 的数组内容已经被排序。

我们知道数组名是常量指针，所以说虚实结合时，形参数组的数组名 x 接受了实参数组 a 的首地址是不严格的，能够接受地址的变量应当是指针变量。但是在当时我们也只能这么理解。通过本章指针的学习，我们将对数组作为参数会有更加深刻的认识。

C 编译系统将形参数组名作为数组的指针变量来处理。所以函数 sort(int x[], int n); 可以认为就是：函数 sort(int *x,int n); 在函数调用时,形参数组指针接受来自实参数组的首地址，也就是指向了实参数组 a。由数组的指针访问方式可知，指针变量 x 指向数组后可以带下标，即 x[i],*(x+i)等价，它们都代表数组中下标为 i 的元素。

数组指针作为函数参数可以分为 4 种情况：

(1)形参、实参都是数组名（前面都是这么用的，不再介绍）

(2)实参是数组名，形参是指针变量

例 7.3：用选择法对 10 个整数进行排序。

排序算法，即选择排序的过程：就是选择相对最大数并交换到位置的过程。选择相对最大数并交换共进行 $n-1$ 次，由外层循环实现。每次选择相对最大数时，内层循环只记录相对最大数的下标 k ，在内层循环结束后，相对最大数的下标确定，根据相对最大数所在的位置，数据本身最多只要交换一次（ $k \neq i$ 的情况），这对大块数据的排序是一种优化。不同之处是：

- 使用了函数。

- 形参采用指针变量，所有数组元素均由指针表示法表示。如：
 $*(b+j), *(b+k)$ 。当然尽管形参采用指针变量，所有数组元素还是可以采用下标表示的。

【思考】

形参采用指针变量，将 sort 函数修改为用下标表示。

(1)形参、实参都是指针变量；

(2)实参是指针变量，形参是数组名。

总之，数组作为函数参数时，不管参数是数组还是指针，只是接口形式不同，数组元素可以使用下标表示，也可以使用指针表示。

5、指向多维数组的指针和指针变量

指针可以指向一维数组，也可以指向多维数组。

多维数组的指针：多维数组的地址（首地址）。

多维数组的指针变量：存放多维数组地址的变量。

多维数组的数组元素与一维数组的数组元素一样既可以用下标表示（访问），又可以用指针表示（访问），还可以用下标与指针组合表示（访问）。为了清楚理解多维数组的各种表示方法，并在程序设计中灵活应用，需要对多维数组的地址、指针进行详细分析。

二维数组是常用的多维数组，后面以二维数组为例进行分析，分析的结果也可以推广到一般的多维数组。

(1)二维数组的地址（分析）

我们知道指针和地址密切相关，要清楚地理解二维数组指针，必须对二维数组地址首先应该有清晰的认识。

假设一个二维数组 `int S[3][4]`。

S 数组是一个 3x4 (3 行 4 列) 的二维数组。可以将它想象为一个矩阵 (图 1)。各个数组元素按行存储,即先存储 s[0]行各个元素 (s[0][0],...s[0][3]), 再存储 s[1]行各个元素(s[1][0],...s[1][3]), 最后存储 s[2]行各个元素(s[2][0],...s[2][3])。

二维数组 S 可以看成由一维数组作为数组元素的数组:

C 语言中, 数组的元素允许是系统定义的任何类型, 也可以是自己定义的任何类型, 也就是说如果将每一行数组元素作为一个整体, 那么 S 数组可以看作作为一个一维的数组 (图 2)。在这个一维数组中每个数组元素表示为: s[0],s[1],s[2]。

一维数组 s 的每个数组元素 s[0],s[1],s[2]本身不是数值, 它们又分别是三个一维数组 (图 3), 这三个一维数组的数组名分别是 s[0],s[1],s[2]。

根据一维数组地址、指针的概念可以知道: s 是元素为行数组的一维数组的数组名, 就是说 s 是元素为行数组的一维数组的首地址。s+i 即就是元素为行数组的一维数组的第 i 个元素的地址, 即: $*(s+i) = s[i]$ 。.....(1) (可以认为是行转列)

同理: s[i](i=0-2)是第 i 个行数组的数组名, s[i]+j 就是第 i 个行数组中第 j 个元素的地址。也就是说, 二维数组任何一个元素 s[i][j]的地址可以表示为: s[i]+j, 就是说二维数组任何一个元素 $s[i][j] = *(s[i]+j)$(2)

综合 (1) (2) 两个式子:

二维数组任何一个元素 s[i][j]的地址可以表示为:

$\&s[i][j] = s[i]+j = *(s+i)+j$(3)

二维数组任何一个元素可以表示为:

$s[i][j] = *(s[i]+j) = (*(s+i)+j)$(4)

事实上:

s[i][j] 还可以表示为: $*(s+i)[j], *(&s[0][0]+m*i+j)$ (m 列数)

推广到三维数组, 三维数组任何一个元素可以表示为: $s[i][j][k] = (*(s+i)+j)+k$ 。

(2)指向多维数组的指针变量

指向数组元素的指针变量 (数组基类型指针变量)

例 7.4: 用指向元素的指针变量输出数组元素的值。

```
main()
{
    int a[3][4]={{0,2,4,6},{1,3,5,7},{9,10,11,12}};
    int *p;
    for(p=a[0]; p<a[0]+12; p++)
    {
        if((p-a[0])%4==0)printf("\n");
        printf(*p);
    }
}
```

要打印的是第几个元素

程序 7.7 例 7.4 程序

例 7.5: 用指向元素的指针变量输出数组元素 $a[1][2]$ 的值。 (补充)

```
main()
{
    int a[3][4]={{0,2,4,6},{1,3,5,7},{9,10,11,12}};
    int *p;
    int i=1,j=2,m=4; /* m-col */

    p=a[0]+i*m+j;
    printf(*p);
}
```

或者:
p=a[0];
n=i*m+j;
printf(*(p+n))

程序 7.8 例 7.5 程序

指向由 m 个元素组成的一维数组的指针变量 (指向一维数组的指针变量, 行指针变量)

定义格式: 基类型 $(*p)[m]$;

说明: 指定 p 是一个指针变量, 它指向包含 m 个元素的一维数组。()不能省略, 否则表示指针数组。

例如:

```
int s[3][4];
```

```
int (*p)[4]; p=s;
```

例 7.6: 使用行指针输出数组元素 $a[i][j]$ 的值。

```
main()
{
    int a[3][4]={{0,2,4,6},{1,3,5,7},{9,10,11,12}};
    int (*p)[4],i,j;

    scanf("%d%d",&i,&j);

    p=s;
    printf("%d\n",*(*(p+i)+j));
}
运行:
1 2
5
```

程序 7.9 例 7.6 程序

6、指向多维数组的指针变量作为函数的参数

多维数组的地址可以作为函数参数。多维数组的地址作为函数参数时, 形参应该是指针变量。用指针变量作为形参接受实参数组传递的地址时有两种方法:

(1) 用指向数组元素 (变量) 的指针变量

(2)用指向一维数组的指针变量

例 7.7: 有一个班级, 3 个学生, 各学 4 门课程, 计算总平均分数以及第 n 个学生的成绩。

```
void average(float *p,int n);
void search(float (*p)[4],int n);
main()
{
    float score[3][4]={ {65,75,54,80},{78,90,89,76},{66,76,87,90}};
    int n;

    average(score[0],12); /* or: *score,&score[0][0] */

    printf("enter num:");scanf("%d",&n);
    search(score,n);
}

void average(float *p,int n) /* 计算总平均分数并打印 */
{
    float aver=0,*pend=p+n;
    for(;p<pend;p++)
        aver=aver+(*p);
    aver=aver/n;
    printf("average:%5.1f\n",aver);
}

void search(float (*p)[4],int n) /* 打印第 n 个学生的成绩 */
{
    int i;
    for(i=0; i<4; i++)
        printf("%5.1f",*(p+n+i));
    printf("\n");
}
```

结果:

```
average: 77.2
enter num:1
78.0 90.0 89.0 76.0
```

程序 7.10 例 7.7 程序

当二维数组的数组名作为实参时, 对应的形参必须是一个行指针变量, 当使用一维数组的数组名作为实参时, 对应的形参必须是一个指向元素 (变量) 的指针变量。

二、字符串的指针和指向字符串的指针变量

1、字符串的表示形式。

C 语言允许使用两种方法实现一个字符串的引用。（字符数组、字符指针）。

(1)字符数组

将字符串的各个字符（包括结尾标志 ‘\0’ ）依次存放到字符数组中，利用数组名或下标变量对数组进行操作。

例 7.8: 字符数组应用

```
main()
{
    char string[]="I am a student.";
    printf("%s\n",string); /* 整体输出 */
    printf("%c,%c\n",string[3],*(string+3)); /* 输出其中一个字符 */
}
运行:
I am a student.
m,m
```

字符数组

程序 7.11 字符数组应用

(2)字符指针

可以不定义字符数组，直接定义指向字符串的指针变量，利用指针变量对字符串进行操作。

例 7.9: 字符指针的应用。

```
main()
{
    char *string="I am a student.";
    printf("%s\n",string); /* 整体输出 */
    printf("%c,%c\n",string[3],*(string+3)); /* 输出其中一个字符 */
}
运行:
I am a student.
m,m
```

字符指针

等价:
char temp[]="I am a student";
string=temp;

程序 7.12 字符指针应用

可以看出：C 语言允许除了使用字符数组进行字符串的处理外，还可以使用字符指针。

例 7.10: 输入两个字符串, 比较是否相等。相等输出 yes, 不等输出 no。(要求使用字符指针处理字符串)。

```
#include <stdio.h>
main()
{
    int t=0; /* t-标志变量, t=0 两串相等, t=1 两串不等 */
    char *s1,*s2;

    s1=(char *)malloc(80); s2=(char *)malloc(80);
    gets(s1); gets(s2);

    while(*s1!='\0'&&*s2!='\0') /* s1,s2 只要有一个到达串尾, 结束比较 */
    {
        if(*s1!=*s2){t=1;break;}
        s1++;s2++;
    }

    if(t==0&&*s1==*s2)printf("YES\n"); /* 考虑为什么要*s1==*s2 */
    else printf("NO\n");

    free(s1); free(s2);
}

运行:
this
these
NO
```

程序 7.13 字符串比较

2、字符串指针作为函数参数

使用字符串(字符数组)指针变量(实际就是字符指针变量)可以作为函数形参接受来自实参字符串(字符数组)的地址。在函数中改变字符指向的字符串的内容, 在主调函数中得到改变了的字符串。

例 7.11: 将输入字符串中的大写字母改为小写字母, 然后输出字符串。

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

void inv(char *s) /* 或 char s[] */
{
    int i;
    for(i=0; i<strlen(s); i++) /* A-Z */
        if(*(s+i)>=65&&*(s+i)<91)*(s+i)+=32; /* 或 if(s[i]>=65&&s[i]<91)s[i]+=32; */
}
```

```

main()
{
    char *string; string=(char *)malloc(80);
    gets(string);

    inv(string);
    puts(string);

    free(string);
}

```

运行结果:

ACeBd

acebd

用指向字符串（字符数组）的字符指针对字符串进行操作，比使用字符数组进行操作更加灵活、方便。

例如：inv 函数还可以改写为下面两种形式：

```

void inv(char *s)
{
    while(*s!='\0')
    {
        if(*s>=65&&*s<91)*s+=32; s++;
    }
}

void inv(char *s)
{
    for(;*s!='\0';s++)
        if(*s>=65&&*s<91)*s+=32;
}

```

程序 7.14 inv 函数改写

3、字符数组和字符指针的区别。（实际上就是数组与指针的区别）

字符数组和字符（串）指针都能够实现对字符串的操作，但它们是有区别的，主要区别在下面几个方面：

(1)存储方式的区别

字符数组由若干元素组成，每个元素存放一个字符。字符指针存放的是地址（字符数组的首地址），不是将整个字符串放到字符指针变量中。

(2)赋值方式的区别

对字符数组只能对各个元素赋值，不能将一个常量字符串赋值给字符数组（字符数组定义例外）。可以将一个常量字符串赋值给字符指针，但含义仅仅是将常量串首地址赋值给字符指针。

例如：

不允许：char str[100]; str="I am a student.";

允许：char *pstr; pstr="I am a student.";

(3) 定义方式的区别

定义数组后，编译系统分配具体的内存单元（一片连续内存空间），各个单元有确切的地址。定义一个指针变量，编译系统只分配一个 2 字节存储单元，以存放地址值。也就是说字符指针变量可以指向一个字符型数据（字符变量或字符数组），但是在对它赋以具体地址前，它的值是随机的（不知道它指向的是什么）。所以字符指针必须初始化才能使用。

例如：

允许：char *s[10]; gets(s);

不允许 char *ps; get(ps); /* 尽管可能也可以使用，但是这是很危险的 */

(4) 运算方面的区别

指针变量的值允许改变（++，--，赋值等），而字符数组的数组名是常量地址，不允许改变。

第三部分 指针变量

一、指向函数的指针变量

1、函数的指针，使用函数指针调用函数。

函数的指针：函数的入口地址（函数的首地址）。C 语言规定函数的首地址就是函数名，所以函数名就是函数的指针。

指向函数的指针变量：存放函数入口地址（函数指针）的变量，称为指向函数的指针变量。简称函数的指针变量。

函数可以通过函数名调用，也可以通过函数指针调用。

通过函数指针实现函数调用的步骤：

(1) 指向函数的指针变量的定义：类型（* 函数指针变量名）();

例如 int (*p)(); 注意：两组括号（）都不能少。int 表示被指向的函数的类型，即被指向的函数的返回值的类型。

(2) 指向函数的指针变量的赋值，指向某个函数：函数指针变量名=函数名；

(3) 利用指向函数的指针变量调用函数：（* 函数指针变量名）（实参表）。

例 7.12: 输入 10 个数, 求其中的最大值。

```
/* 使用函数名调用函数 */
main()
{
    int i,m,a[10];
    for(i=0; i<10; i++)
        scanf("%d",&a[i]);
    m=max(a); /* 函数调用格式: 函数名(实参表) */
    printf("max=%d\n",m);
}
int max(int *p) /* max 在 10 个整数中选择最大值 */
{
    int i,t=*p;
    for(i=1; i<10; i++)
        if(*(p+i)>t)t=*(p+i);
    return t;
}
结果:
-52 87 29 79 -32 94 23 -112 46 67
max=94
```

程序 7.15 最大值函数

/* 使用函数指针变量调用函数 */

main()

```
{
    int i,m,a[10],max(int *); /* declare func */
    int (*pf)(); /* define func pointer */
```

声明函数

```
    for(i=0; i<10; i++)
        scanf("%d",&a[i]);
    pf=max; /* pf->max() */
    m=(*pf)(a); /* call max */
```

指针的定义:
定义函数指针变量 pf
(返回整型数)

```
    printf("max=%d\n",m);
}
```

指针的初始化:
函数指针 pf 指向 max

int max(int *p)

```
{
    int i,t=*p;
    for(i=1; i<10; i++)
        if(*(p+i)>t)t=*(p+i);
    return t;
}
```

指针的引用:
调用函数指针 pf 指向的函数 max

程序 7.16 函数指针变量调用函数

【说明】

(1)定义函数指针变量时，两组括号（）都不能少。如果少了前面的一组括号=>返回值类型 * 函数名(); -返回值为地址值（指针）的函数。

(2)函数指针变量的类型是被指向的函数的类型，即返回值类型。

(3)函数指针的赋值，只要给出函数名，不必给出参数。（不要给出实参或形参）。

(4)用指针变量调用函数时，(* 函数指针)代替函数名。参数表与使用函数名调用函数一样。

(5)可以看出，定义的函数指针变量可以用于一类函数，只要这些函数返回值类型（函数类型）相同。

函数可以通过函数名调用，也可以通过函数指针调用。函数指针常常用在函数需要作为函数参数的情况。

2、用指向函数的指针作为函数的参数（常用于编制“通用”的函数）

函数的参数除了可以是变量、指向变量的指针，数组（实际是指向数组的指针）、指向数组的指针以外，还可以是函数的指针。

函数的指针可以作为函数参数，在函数调用时可以将某个函数的首地址传递给被调用的函数，使这个被传递的函数在被调用的函数中调用（看上去好象是将函数传递给一个函数）。函数指针的使用在有些情况下可以增加函数的通用性，特别是在可能调用的函数可变的情况下。

例 7.13：编制一个对两个整数 a,b 的通用处理函数 process，要求根据调用 process 时指出的处理方法计算 a, b 两数中的大数、小数、和。

```
int max(int ,int );
int min(int ,int );
int add(int ,int );
int add1(int);

main()
{
    int a,b;
    printf("Enter two num to a,b:");scanf("%d%d",&a,&b);

    printf("max=%d\n",process(a,b,max)); /* 调用通用处理函数 */
    printf("min=%d\n",process(a,b,min));
    printf("add=%d\n",process(a,b,add));
    printf("add1=%d\n",process(a,b,add1));
}

int max(int x,int y){ return x>y?x:y; } /* 返回两数之中较大的数 */
```

```
int min(int x,int y){ return x<y?x:y; } /* 返回两数之中较小的数 */
int add(int x,int y){ return x+y; } /* 返回两数的和 */
int add1(int x){ return x+1; }
```

```
int process(int x,int y, int (*f)()) /* 通用两数的处理函数 */
{
    return (*f)(x,y);
} □
```

结果:

Enter two num to a,b:3 8

max=8

min=3

add=11

add1=4

【说明】

(1)函数 process 处理两个整数数，并返回一个整型值。同时又要求 process 具有通用处理能力（处理求大数、小数、和），所以可以考虑在调用 process 时将相应的处理方法（“处理函数”）传递给 process。

(2)process 函数要接受函数作为参数，即 process 应该有一个函数指针作为形式参数，以接受函数的地址。这样 process 函数的函数原型应该是：

```
int process(int x,int y,int (*f)());
```

(3)“函数指针作为函数参数”的使用，即函数指针变量的定义-在通用函数 process 的形参定义部分实现；函数指针变量的赋值-在通用函数的调用的虚实结合时实现；用函数指针调用函数-在通用函数内部实现。

(4)main 函数调用通用函数 process 处理计算两数中大数的过程是这样的：

- 将函数名 max（实际是函数 max 的地址）连同要处理的两个整数 a,b 一起作为 process 函数的实参，调用 process 函数。
- process 函数接受来自主调函数 main 传递过来的参数，包括两个整数和函数 max 的地址，函数指针变量 f 获得了函数 max 的地址。
- 在 process 函数的适当位置调用函数指针变量 f 指向的函数，即调用 max 函数。本例直接调用 max 并将值返回。这样调用点就获得了两数大数的结果，由 main 函数 printf 函数输出结果。

同样，main 函数调用通用函数 process 处理计算两数小数、和的过程基本一样。

(5)process 函数头部：函数指针定义中不需要指定形参个数。但是一般情况函数指针指向的函数参数个数一般是数量类型相同的，以使用统一的格式如(*f)(x,y)去调用。

process 函数是一个“通用”整数处理函数，它使用函数指针作为其中的一个参数，以实现同一个函数中调用不同的处理函数。

二、返回指针值的函数

函数可以返回整型、实型、字符型等类型的数据，还可以返回地址值-即返回指针值。

返回指针值的函数定义：类型名 * 函数名（参数表）

例如：

int *fun(int x,int y)表示 func 是返回整型指针的函数，返回的指针值指向一个整型数据。该函数还包含两个整型参数 x,y。

系统内存分配函数 void *malloc(size_t size);也是一个返回指针值的函数。返回的指针指向一片分配好的内存空间。

例 7.14：返回两个数中大数地址的函数。

```
int *fun(int,int);
main()
{
    int i,j,*p;
    printf("enter two num to i,j:"); scanf("%d%d",&i,&j);

    p=fun(i,j); /* 调用 fun, 返回大数地址, 赋值给指针变量 p */
    printf("max=%d\n",*p); /* 打印 p 指向的数据 */
}

int *fun(int x,int y) /* fun 函数返回形参 x,y 中较大数的地址（指针） */
{
    int *z;
    if(x>y)z=&x; else z=&y;
    return z;
}
结果：
enter two num to i,j:12 38
max=38
```

程序 7.17 例 7.14 程序

【说明】

(1)main 函数从键盘获得两个整数 i,j（本例 12，38）。将 i,j 作为实参调用 fun。

(2)通过虚实结合，fun 函数的形参 x,y 获得了这两个整数（本例 12，38），将大数的地址返回（本例是&y）。

(3)返回的地址值赋值给 main 函数的指针变量 p,main 函数打印 p 指向的整型数，即 y 的值。

【思考】

阅读下面的程序，分析结果，为什么会是这样？

```
int *fun(int,int);
int add(int,int);
main()
{
    int *p,i,j,sum;
    printf("enter two num to i,j:"); scanf("%d%d",&i,&j);
    p=fun(i,j);
    sum=add(444,444);
    printf("max=%d\n",*p); printf("sum=%d\n",sum);
}

int *fun(int x,int y)
{
    int *z;
    if(x>y)z=&x;
    else z=&y;
    return z;
}

int add(int a,int b)
{
    return a+b;
}

结果:
enter two num to i,j:12 38
max=444
sum=888□
```

程序 7.18 程序阅读

三、指针数组与指向指针的指针

1、指针数组

数组的指针：指向数组元素的指针。数组元素可以是一般数据类型，也可以是数组、结构体等数据类型。数组指针的定义与数组元素指针的定义相同。

指针数组：一个数组，如果其数组元素均为指针，那么此数组为指针数组。

一维指针数组的定义：类型名 *数组名[数组长度];

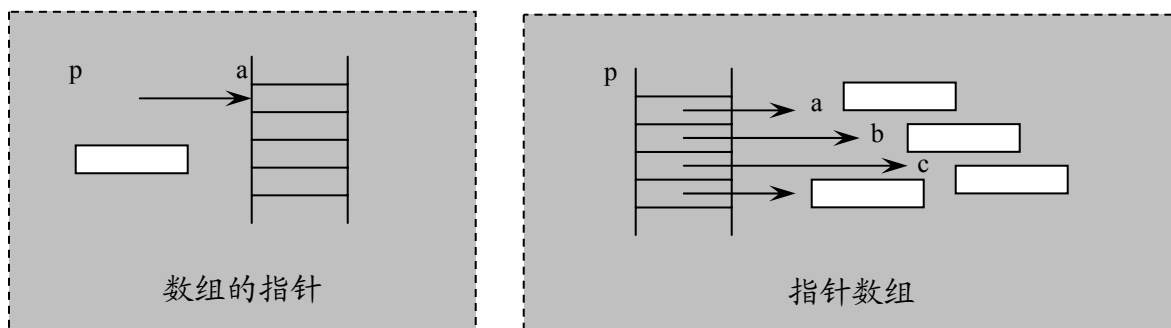


图 7.8 数组指针和指针数组比较图

例如：int *p[4];

定义一个 4 个元素的数组 p，其中每个元素是一个整型指针，也即数组 p 是一个 4 元素整型指针数组。

又如：char *p[4];

定义一个 4 个元素的字符指针数组 p，其中每个数组元素是一个字符指针，可以指向一个字符串。也就是说利用此字符指针数组可以指向 4 个字符串。

指针数组用得最多的是“字符型指针数组”，利用字符指针数组可以指向多个长度不等的字符串，使字符串处理更加方便、灵活，节省内存空间。

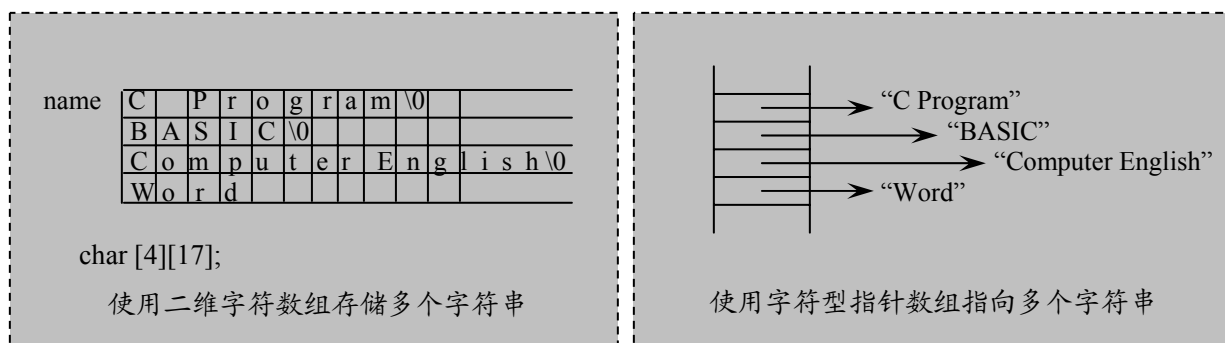


图 7.9 字符型指针数组

【比较】 使用字符型指针数组指向多个字符串与使用二维字符数组存储多个字符串的比较：

(1) 节省存储空间(二维数组要按最长的串开辟存储空间)

(2) 便于对多个字符串进行处理，节省处理时间。（使用指针数组排序各个串不必移动字符数据，只要改变指针指向的地址）

例 7.15: 将若干字符串按字母顺序由小到大输出。

```
void sort(char *name[],int n) /* 选择法排序 */
{
    char *temp;
    int i,j,k;

    /* n 个串，只要进行 n-1 次选择相对最小串（对应的指针），就可完成排序*/
    for(i=0; i<n-1; i++)
    {
        k=i; /* 假设当前位置 i 是相对最小串的指针所在的位置，保存 k 中 */

        for(j=i+1; j<n; j++) /* 与后面各个位置指针指向串进行比较 */
            if(strcmp(name[j],name[k])<0)k=j; /* 如果有更小的串，记录其指针的位置于 k */

        if(k!=i)
        {
            temp=name[i];name[i]=name[k];name[k]=temp; /* 交换指针 name[i]<->name[k]*/
        }
    }
}

main()
{
    char *name[]={"C Program","BASIC","Computer English","Word"};
    int i,n=4;

    sort(name,n);

    for(i=0; i<n; i++)
        printf("%s\n",name[i]);
} □

结果:
BASIC
C Program
Computer English
Word
```

程序 7.19 字符串按字母顺序输出程序

【说明】

(1)main()中定义了指针数组 name，它有 4 个元素，其初值分别是"C Program","BASIC","Computer English","Word"四个字符串常量的首地址。

(2)函数 sort 使用选择排序法对指针数组指向的字符串进行排序（按字母顺序），在排序过程中不交换字符串本身，只交换指向字符串的指针（name[k]<->name[i]）。

(3)利用字符指针数组进行字符串排序。

2、指针的指针

(1)指针的指针：指向指针变量的指针变量。指针的指针存放的是指针变量地址。

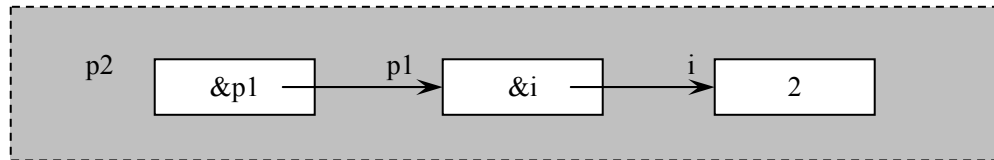


图 7.10 指针的指针

指针变量的指针变量（指针的指针）的定义：类型 **指针变量名；

例如：

```
int i=2; /* 定义整型变量 i */
```

```
int *p1,**p2; /* 定义 p1 为整型指针，定义 p2 为整型指针的指针 */
```

```
p1=&i; /* i 的地址=>p1，即，指针 p1 指向变量 i */
```

```
p2=&p1; /* 指针 p1 的地址=>p2，即，指针 p2 指向指针 p1 */
```

对变量 i 的访问可以是 i,*p1,又因为*p2=p1，即，**p2=*p1，所以对变量 i 的访问可以是 i,*p1,**p2。

(2)指针的指针与指针数组的关系

数组的指针是指向数组元素的指针（整型、实型、字符型一维数组的指针分别是指向整型、实型、字符型指针，二维数组的指针是指向一维数组的指针）；同理：

指针数组的指针，也是指向其数组元素的指针。指针数组的数组元素是指针，所以指向指针数组的指针就是指针的指针。也就是说，可以使用“指针的指针”指向指针数组。

例 7.16：指向指针的指针变量的应用。

```
main()
{
    char *name[]={"C Program","BASIC","Computer English","Word"};
    char **p;
    for(p=name; p<name+4; p++)
        printf("%s\n",*p); /* *p=name[i],name[i] is a address */
}
```

程序 7.20 指向指针的指针变量的应用

3、指针数组的应用-指针数组作为 main()函数的参数。

(1)main()函数可以带参数。

main()函数是整个可执行程序入口（执行起点）。main 函数也与其它函数一样可以带参数，指针数组的一个重要应用是作为 main 函数的形参。人们习惯将 argc,argv 作为 main()函数的形参名。

带参数 main 函数的完整的原型是：类型 main(int argc,char *argv[]);

其中：

(1) argc 是传递给 main()函数的参数的个数。（包括可执行程序名）

(2) argv 是传递给 main()函数的字符指针数组，该数组各个元素是字符指针，分别指向调用 main()函数时在操作系统命令行输入的各个字符串。（包括可执行程序名）

2、main 函数如何获得参数？-从操作系统命令行获得参数。

- C 语言源程序经过编译、连接获得一个在操作系统下可以直接执行的程序（可执行程序）。操作系统调用可执行程序的方法是在操作系统命令提示符下输入：

C: > 可执行程序名(命令名) 参数表<CR> （操作系统命令行）

- 操作系统调用可执行程序时，将操作系统命令行中的“（程序）命令名”以及各个参数（字符串）作为 main()函数的参数，传递给 main()函数的 argc,argv。然后程序由 main()函数开始运行程序。

例如：DOS 的 xcopy 命令（外部命令）实际是一个程序。

其使用格式是：xcopy c:\mycprg*.c d:\mybak /s<CR>

例 7.17：编写一个命令文件，把键入的字符倒序打印出来。设文件名为 invert.c。

```
main(int argc,char *argv[])
{
    int i;
    for(i=argc-1; i>=0; i--)
        printf("%s\n",argv[i]);
}
E:\10a>invert I love China
China
love
I
E:\10A\INVERT.EXE
```

程序 7.21 倒序打印键入的字符

四、指针运算举例

例 7.18: 编写函数 `length(char *s)`, 函数返回指针 `s` 所指字符串的长度。

```
int length(char *s)
{
    int n=0;
    while(*(s+n)!='\0')n++;/* 当 s+n 没有指向串尾, n 增 1,直到 s+n 指向串尾 */
    return n;
}

main()
{
    char str[]="this is a book";
    printf("%d\n",length(str));
} □
结果: 17
```

程序 7.22 字符串长度计算程序

例 7.19: 已知存放在数组 `a` 中的数不相重复, 在数组 `a` 中查找与值 `x` 相等的位置。若找到, 输出该值和该值在数组 `a` 中的位置; 若没有找到, 输出相应的信息。

```
#define NUM 20
```

```
main()
```

```
{
```

```
    int a[NUM],x,n,p;
```

```
        /* a-数表, x-要查找的数字, n-数表长度, p-x 在数表中的位置 */
```

```
    n=input(a);          /* 输入各个整数 */
```

```
    printf("enter the number to search:x=");
```

```
    scanf("%d",&x);      /* 输入要查找的整数 */
```

```
    p=search(a,n,x);     /* 查找 x 在数组中的位置 */
```

```
    if(p!=-1)printf("%d index is:%d\n",x,p);
```

```
    else printf("%d cannot be found!\n",x);
```

```
}
```

```
int input(int *a) /* 输入整数的个数 (≤19), 依次输入各个整数 */
```

```
{
```

```
    int i,n;
```

```
    printf("Enter number to elements,0<n<=%d:",NUM);
```

```
    scanf("%d",&n);      /* 输入整数个数 */
```

```
    for(i=0; i<n; i++) /* 依次输入各个整数 */
```

```
        scanf("%d",a+i);
```

```
    return n;
```

```
}
```

```

int search(int *a,int n,int x) /* 在数组中查找 x 的位置，返回-1 未找到 */
{
    /* 返回其它值，找到，返回值就是位置号 */
    int i,p;

    i=0;
    a[n]=x; /* 最后一个数后面，再添加一个整数 x（要查找的整数） */
    while(x!=a[i])i++;
        /* 若还没有找到，继续和下一个比较，直到找到或到达最后 */
        /* 一个元素后的一个元素 */
    if(i==n)p=-1; /* 未找到，p<=-1 */
    else p=i; /* 找到，p<=位置号 */

    return p;
}
Enter number to elements,0<n<20: 10
5 10 15 20 25 30 40 50 60 70
enter the number to search:x=25
25 index is:4

```

```

Enter number to elements,0<n<20: 10
5 10 15 20 25 30 40 50 60 70
enter the number to search:x=35
35 cannot be found!

```

【查找技巧】：边界单元存放要查找的数，循环一定能结束，结束时的下标 i 就暗示着查找是否成功。（边界单元：最后元素之后的一个单元）。本例采用顺序查找法进行数据查找。

例 7.20：输入 10 个整数，将其中最小数与第一个数交换，把最大数与最后一个交换。写三个函数分别完成：（1）输入 10 个整数；（2）进行处理；（3）输出 10 个数，用指针法处理。

```

void input(int *a,int n)
{
    int *p;
    for(p=a; p<a+n; p++)
        scanf("%d",p);
}

void output(int *a,int n)
{
    int *p;
    for(p=a; p<a+n; p++)
        printf("%5d",*p);
    printf("\n");
}

```

```

void invert(int *a,int n)
{
    int *p1,*p2,*p;
    int temp;

    p1=p2=a; /* p1 指向最大元素所在存储单元,p2 指向最小元素所在存储单元 */
    for(p=a+1; p<a+n; p++) /* 循环查找最大, 最小元素所在存储单元 */
        if(*p>*p1)p1=p;
        else if(*p<*p2)p2=p;

    if(p2!=a) /* *a<->*p2 交换第一个元素和最小元素 */
        {temp=*p2; *p2=*a; *a=temp;}

    if(p1!=a+n-1) /* *(a+n-1)<->*p1 交换最后一个元素和最大元素 */
        {temp=*p1; *p1=*(a+n-1); *(a+n-1)=temp;}
}

main()
{
    int x[10];
    int n=10;

    input(x,n);
    output(x,n);
    invert(x,n);
    output(x,n);
}

2 3 0 9 -1 23 -98 5 12 -23
2  3  0  9 -1 23 -98  5 12 -23
-98  3  0  9 -1 -23  2  5 12 23

```

第八章

数组

【基本要求】

通过本单元的学习，使学生了解数组的基本知识，熟悉为一维数组的定义、引用、初始化及使用，掌握二维数组的定义初始化、引用、初始化及其使用，熟练掌握字符数组的输入和输出及字符串处理函数和应用。

【教学重点】

一维数组的定义、元素引用、初始化、应用，二维数组的定义、元素引用、初始化、应用，字符数组。

【本章结构】

- | | | |
|--------------|---|-------------------------------------------------------------------------------------|
| 1、一维数组的定义和引用 | { | 一维数组的定义
一维数组元素的引用
一维数组的初始化
一维数组程序举例 |
| 2、二维数组的定义和引用 | { | 二维数组的定义
二维数组元素的引用
二维数组的初始化
二维数组程序举例 |
| 3、字符数组 | { | 字符数组的定义
字符数组的初始化
字符数组元素的引用
字符串和字符串结束标志
字符数组的输入输出
字符串处理函数
字符数组程序举例 |

第一部分 一维数组的定义和初始化

前面各章所使用的数据都属于基本数据类型（整型、实型、字符型），C 语言除了提供基本数据类型外，还提供了构造类型的数据，它们是数组类型、结构体类型、共同体类型。构造数据类型是由基本数据类型的数据按照一定的规则组成，所以也称为“导出类型”。

数组：具有相同数据类型的数据的有序的集合。

数组元素：数组中的元素。数组中的每一个数组元素具有相同的名称，不同的下标，可以作为单个变量使用，所以也称为下标变量。在定义一个数组后，在内存中使用一片连续的空间依次存放数组的各个元素。

数组的下标：是数组元素的位置的一个索引或指示。

数组的维数：数组元素下标的个数。根据数组的维数可以将数组分为一维、二维、三维、多维数组。

例如：int a[10];

定义了一个一维数组 a，该数组由 10 个数组元素构成的，其中每一个数组元素都属于整型数据类型。数组 a 的各个数据元素依次是 a[0],a[1],a[2]...a[9]（注意：下标从 0-9）。每个数据元素都可以作为单个变量使用（如赋值，参与运算，作为函数调用的参数等）。

一维数组可以看作一个数列，向量。

例如：float b[3][3];

定义了一个二维数组 b，该数组由 9 个元素构成，其中每一个数组元素都属于浮点（实数）数据类型。数组 b 的各个数据元素依次是:b[0][0],b[0][1],b[0][2],b[1][0],b[1][1],b[1][2],b[2][0],b[2][1],b[2][2]（注意：下标从 0-2）。每个数据元素也都可以作为单个变量使用。

二维数组可以看作一个矩阵。

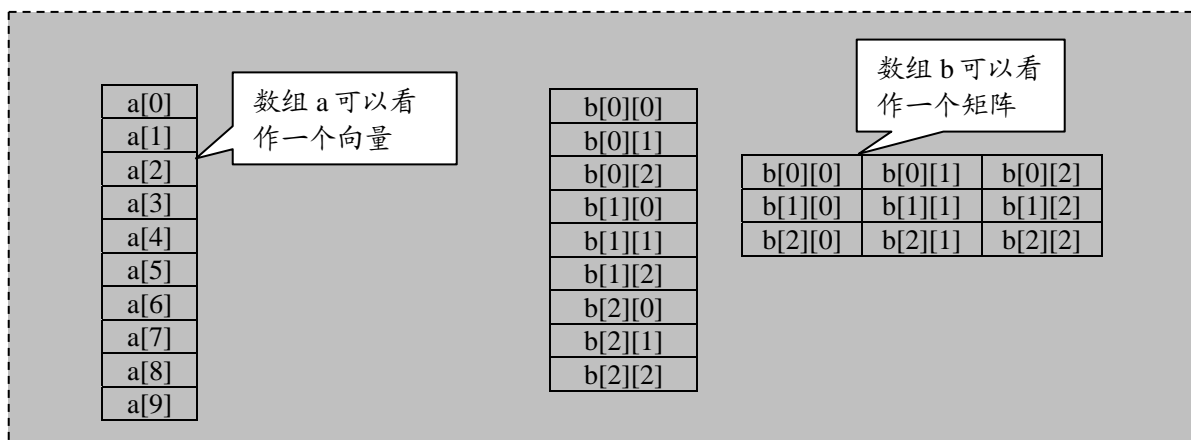


图 8.1 一维数组和二维数组示意图

一维数组中的各个数组元素是排成一行的一组下标变量，用一个统一的数组名来标识，用一个下标来指示其在数组中的位置。下标从 0 开始。一维数组通常和一重循环相配合，对数组元素进行处理。

一、一维数组的定义（先定义后使用）

定义一维数组的格式：类型说明 数组名[整型常量表达式]

例如：int a[100];定义了一个数组 a，元素个数为 100，数组元素类型为整型。

【说明】

(1)数组名：按标识符规则。本例 a 就是数组名。

(2)整型常量表达式：表示数组元素个数（数组的长度）。可以是整型常量或符号常量，不允许用变量。

整型常量表达式在说明数组元素个数的同时也确定了数组元素下标的范围，下标从 0 开始~整型常量表达式-1（注意不是 1~整型常量表达式）。C 语言不检查数组下标越界，但是使用时，一般不能越界使用，否则结果难以预料（覆盖程序区-程序飞出，覆盖数据区-数据覆盖破坏，操作系统被破坏，系统崩溃）。

本例数组元素个数是 100 个，下标从 0-99。

(3)类型说明：指的是数据元素的类型，可以是基本数据类型，也可以是构造数据类型。类型说明确定了每个数据占用的内存字节数。比如整型 2 字节，实型 4 字节，双精度 8 字节，字符 1 字节。

本例数组元素是整型，每个元素占 2 个字节，因为有 100 个数组元素，所以占用 200 字节。

(4)C 编译程序为数组分配了一片连续的空间。

(5)C 语言还规定，数组名是数组的首地址。即 a=&a[0]。

二、一维数组的初始化

数组可以在定义时初始化，给数组元素赋初值。

数组初始化常见的几种形式：

(1)对数组所有元素赋初值，此时数组定义中数组长度可以省略。

例如：int a[5]={1,2,3,4,5};或 int a[]={1,2,3,4,5};

(2)对数组部分元素赋初值，此时数组长度不能省略。

例如：int a[5]={1,2};

a[0]=1,a[1]=2,其余元素为编译系统指定的默认值 0。

(3)对数组的所有元素赋初值 0。

例如: `int a[5]={0};`

注意: 如果不进行初始化, 如定义 `int a[5];` 那么数组元素的值是随机的, 不要指望编译系统为你设置为默认值 0。

【说明】

(1)定义时直接全部初始化数组元素时, 给定数据的个数不能超过定义的数组元素的个数; 且数据值只能用{}括起来, 且只能用逗号分隔开; 即{}中提供的数据的个数不能超过数组元素的个数(长度), 否则出现语法错误。

(2)给定的数据的值的类型必须与数组定义的类型兼容;

例:

```
int a[6]={2,3,4,5,6,x};
char s[8]={d,f,r,67,98}
main()
{
    int a[7]={1,2,3,4,5,'b'},i=0;
    char s[7]='X','y',65,98,102,'\t';
    float f[7]={12,34,56,7,'s',56};
    for(i=0;i<=6;i++)
        printf("a[%d]=%d ",i,a[i]);
    printf("\n");
    for(i=0;i<=6;i++)
        printf("s[%d]=%c",i,s[i]);
    printf("\n");
    for(i=0;i<=6;i++)
        printf("f[%d]=%f ",f[i]);
    printf("\n");
}
```

(3)定义时若给所有元素都赋值, 则可以不指定数组长度。数组长度由指定的值的个数决定; 例:

`int a[]={2,3,4,5,6,x};` 有 6 个元素

(4)若被定义时数组长度与提供的初值的个数不等, 则数组长度不能省略;

例:

`int a[7]={1,2,3,4}`

(5)定义时若只初始化部分元素, 剩余的元素的值为相应类型的初值。

`int:0;float:0.000;char:\0`

(6)数组元素相当于简单的内存变量, 其赋值方法与内存变量相同。

(7)静态类数组系统自动初始化为相应类型的初值。

int:0;float:0.000;char:\0

例: static int a[10]; static char s[20];static float f[10];

第二部分 一维数组的引用和应用

一、一维数组元素的引用

数组元素的引用形式: 数组名[下标]

【注意】数组元素引用时,下标为整型的表达式,可以使用变量。

例 8.1: 数组元素的引用例子。

```
main()
{
    int a[10],b[5]={55,44,33,22,11},i;

    for(i=0; i<10; i++)
    {
        a[i]=i+1;
        printf("%4d",a[i]);
    }
    printf("\n");

    for(i=0; i<5; i++)
    {
        printf("%4d",b[i]);
    }
    printf("\n");
}
```

图 8.2 数组元素引用示例

【说明】

(1)引用数组元素时,下标可以是整型常数、已经赋值的整型变量或整型表达式。

(2)数组元素本身可以看作是同一个类型的单个变量,因此对变量可以进行的操作同样也适用于数组元素。也就是数组元素可以在任何相同类型变量可以使用的位置引用。

(3)引用数组元素时,下标不能越界。否则结果难以预料(覆盖程序区-程序飞出,覆盖数据区-数据覆盖破坏,操作系统被破坏,系统崩溃)。

二、一维数组的应用

例 8.3: 从键盘输入 10 个整型数据, 找出其中的最小值并显示出来。

【思考】找最大值算法。

例 8.4: 采用“冒泡法”对任意输入的 10 个整数按由小到大的顺序排序。

冒泡法排序思路:

S_0 (第 0 步): 将 n 个数, 从前向后, 将相邻两个数进行比较 (共比较 $n-1$ 次), 将小数交换到前面 (将大数交换到后面), 逐次比较, 直到将最大的数移到最后; (此时最大的数在最后, 固定下来, 目前固定 1 个大数)。

S_1 (第 1 步): 将前面 $n-1$ 个数, 从前向后, 将相邻两个数进行比较 (共比较 $(n-1)-1=n-2$ 次), 将小数交换到前面 (将大数交换到后面), 逐次比较, 直到将次大的数移到倒数第二个位置; (此时次大的数在倒数第二个位置, 同样也固定下来, 目前固定 2 个大数)。

S_2 (第 2 步): 将前面 $n-2$ 个数, 从前向后, 将相邻两个数进行比较 (共比较 $(n-2)-1=n-3$ 次), 将小数交换到前面 (将大数交换到后面), 逐次比较, 直到将第三大数移到倒数第三个位置; (此时第三大数在倒数第三个位置, 同样也固定下来, 目前固定了 3 个大数)。

.....

依照上面的规律:

S_i (第 i 步): 将前面 $p=n-i$ 个数, 从前向后, 将相邻两个数进行比较 (共比较 $p-1=(n-i)-1=n-i-1$ 次), 将小数交换到前面 (将大数交换到后面), 逐次比较, 直到将第 $i+1$ 大数移到倒数第 $i+1$ 个位置; (大数沉底)

.....

S_{n-2} (第 $n-2$ 步): 将最后 2 个数, 进行比较 (比较 1 次), 交换。此时, 所有的整数已经按照从小到大的顺序排列。如程序 8.1 所示。

【分析】

(1) 从完整的过程 (步骤 S_0-S_{n-2}) 可以看出, 排序的过程就是大数沉底的过程 (或小数上浮的过程), 总共进行了 $n-2-0+1=n-1$ 次, 整个过程中的每个步骤都基本相同, 可以考虑用循环实现-外层循环。

(2) 从每一个步骤看, 相邻两个数的比较, 交换过程是从前向后进行的, 也是基本相同的, 共进行了 $n-i-1$ 次, 所以也考虑用循环完成-内层的循环。

(3)为了便于算法的实现，考虑使用一个一维数组存放这 10 个整型数据，排序的过程中数据始终在这个数组中（原地操作，不占用额外的空间），算法结束后，结果在也在此数组中。

```
#define N 10
main()
{
    int a[10],i,j,t;

    for(i=0; i<N; i++)
        scanf(&a[i]);

    for(i=0; i<N-1; i++)
        for(j=0; j<N- (i+1) ; j++)
            if(a[j]>a[j+1])
            {
                t=a[j];a[j]=a[j+1];a[j+1]=t;
            }

    for(i=0; i<N; i++)
        printf(a[i]);
}
```

程序 8.1 冒泡排序法程序

【思考】

(1)如果分析的时候步骤 S_0-S_{n-2} 用步骤 S_1-S_{n-1} 进行分析，结果有什么不同，为什么？应该怎么做才能保证不会错。（提示：边界情况检查法）

(2)如何对任意输入的 10 个整数按由大到小的顺序排序。

(3)对典型算法要作到理解，记忆。

例 8.5：采用“选择法”对任意输入的 10 个整数按由大到小的顺序排序。

选择法排序思路：

S_0 ：将 n 个数依次比较，保留最大数的下标（位置），然后将最大数和第 0 个数数组元素交换位置。（此后可以固定第 0 个数数组元素）

S_1 ：将后面 $n-1$ 个数依次比较，保留次大数的下标（位置），然后将次大数与第 1 个数数组元素交换位置。（此后可以固定第 1 个数数组元素）

S_2 ：将后面 $n-2$ 个数依次比较，保留第 3 大数的下标（位置），然后将第 3 大数与第 2 个数数组元素交换位置。（此后可以固定第 1 个数数组元素）

.....

按照此规律：

S_i : 将后面 $n-i$ 个数依次比较, 保留第 $i+1$ 大数的下标 (位置), 然后将第 $i+1$ 大数与第 i 个数组元素交换位置。

.....

S_{n-2} : 将最后面 2 个数 (因为 $n-i=2$, 所以 $i=n-2$) 比较, 保留第 $n-2+1=n-1$ 大数的下标, 然后将第 $n-1$ 大数与第 $n-2$ 个数组元素交换位置。

【分析】

从完整的过程 (步骤 S_0-S_{n-2}) 可以看出, 选择排序的过程就是选择较大数并交换到前面的过程, 总共进行了 $n-2-0+1=n-1$ 次, 整个过程中的每个步骤都基本相同, 可以考虑用循环实现-外层循环。

从每一个步骤看, 也都是在若个数中比较 (比较进行若干次), 搜索大数, 记录其下标, 并将大数交换到它应该占有的前面的某个位置的过程, 共进行了 $n-i-1$ 次比较 (只进行 1 次数据交换)。所以也考虑用循环完成-内层的循环。

为了便于算法的实现, 考虑使用一个一维数组存放这 10 个整型数据, 排序的过程中数据始终在这个数组中 (原地操作, 不占用额外的空间), 算法结束后, 结果在也在此数组中。

```
#define N 10
main()
{
    int b[10], i, j, t, max, max_i;

    for(i=0; i<N; i++)
        scanf("%d", &b[i]);

    for(i=0; i<N-1; i++)
    {
        max=b[i]; max_i=i; /* 第 i 次找大数, 假设 b[i] 就是大数 */
        for(j=i+1; j<N; j++) /* 从 b[i+1] 开始, 到 b[N-1] 结束 */
        {
            if(b[j]>max) /* 如果某个元素>当前最大值 */
            {
                max=b[j]; max_i=j; /* 记录其下标, 并设置大数值 */
            }
        }
        if(i!=max_i){t=b[i]; b[i]=b[max_i]; b[max_i]=t;} /* 交换大数到 b[i] 位置 */
    }

    for(i=0; i<N; i++)
        printf("%4d", b[i]);
    printf("\n");
}
```

大数的值, 大数的位置

选择法排序。共进行 $N-1$ 次选大数, 并交换到前面。

输入 10 个整数

输出排序后的 10 个整数

程序 8.2 选择法程序

第三部分 二维数组及多维数组

二维数组：数组元素是双下标变量的数组。二维数组的数组元素可以看作是排列为行列的形式（矩阵）。二维数组也用统一的数组名来标识，第一个下标表示行，第二个下标表示列。下标从0开始。

一、二维数组的定义

类型说明符 数组名[整型常量表达式 1][整型常量表达式 2];

例如：int a[3][4];

【说明】

(1)二维数组中的每个数组元素都有两个下标，且必须分别放在单独的“[]”内。

(2)二维数组定义中的第 1 个下标表示该数组具有的行数，第 2 个下标表示该数组具有的列数，两个下标之积是该数组具有的数组元素的个数。

(3)二维数组中的每个数组元素的数据类型均相同。二维数组的存放规律是“按行排列”。

(4)二维数组可以看作是数组元素为一维数组的数组。

二、二维数组的初始化

二维数组的初始化的几种常见形式：

(1)分行给二维数组所有元素赋初值

例如：int a[2][4]={{1,2,3,4},{5,6,7,8}};

(2)不分行给二维数组所有元素赋初值

例如：int a[2][4]={1,2,3,4,5,6,7,8};

(3)给二维数组所有元素赋初值，二维数组第一维的长度可以省略

（编译程序可计算出长度）

例如：int a[][4]={1,2,3,4,5,6,7,8};

或：int a[][4]={{1,2,3,4},{5,6,7,8}};

(4)对部分元素赋初值

例如：int a[2][4]={{1,2},{5}};

【说明】

(1)定义时直接全部初始化数组元素时，行方式初始化时,指定的行数不能超过指定的行数；每行提供的初值个数不能超过定义时指定的每行的元素个数；列方式初始化时,初值个数不能数组元素的总个数；

(2)数据值只能用{}括起来，元素值之间只能用逗号分隔开；即{}中提供的数据的个数不能超过数组元素的个数(长度)，否则出现语法错误。

(3)给定的数据的值的类型必须与数组定义的类型兼容；

(4)定义时若给所有元素或所有行都赋值，则可以不指定数组行维数，但第二维不能省略。数组的行数由提供初值的行数（行初始化方式）或提供的值的初值个数和指定的列数共同决定（列初始化方式）；

例： `int a[][4]={ {2,3},{4,5,6},{ x } };`

指定了 3 行，则由 3 行；（行初始化方式）

`int a[][4]={2,3,4,5,6, x ,7,8,9,12,23,21};`指定列为 4，共 12 个元素，则有 3 行（列方式）

(5)定义时若只初始化部分元素，剩余的元素的值为相应类型的初值

(`int:0;float:0.000;char: \0`)；

(6)数组元素相当于简单的内存变量，其赋值方法与内存变量相同

(7)静态类数组系统自动初始化为相应类型的初值(`int:0;float:0.000;char:\0`)；

例： `static int a[4][10]; static char s[5][20]; static float f[5][10];`

三、二维数组元素的引用

定义了二维数组后，就可以引用该数组的所有元素。

引用形式：数组名[下标 1][下标 2]

【说明】

数组元素可以赋值，可以输出，也就是说任何可以出现变量的地方都可以使用同类型的数组元素。

【思考】下标是从 1 开始使用的，哪些元素在此例子中未使用？它们的值是多少？（随机值）

四、二维数组应用举例

二维数组的遍历访问（扫描），一般都采用双重循环处理（行循环，列循环）。

例 8.6 二维数组中元素的表示：

```
main()
{
    int
    a[4][5]={ {11,12,13,14,15},{21,22,23,24,25},{31,32,33,34,35},{41,42,43,44,45} };
    int i=0,j=0,(*p)[5]=a;
    printf("Array a[]:\n");
    for(i=0;i<4;i++)
    {
        for(j=0;j<5;j++)
            printf("%6d",a[i][j]);
        printf("\n");
    }
    printf("Array p[]:\n");
    for(i=0;i<4;i++)
    {
        for(j=0;j<5;j++)
            printf("%6d",p[i][j]);
        printf("\n");
    }
    printf("Array a[i]+j:\n");
    for(i=0;i<4;i++)
    {
        for(j=0;j<5;j++)
            printf("%6d",*(a[i]+j));
        printf("\n");
    }
    printf("Array p[i]+j:\n");
    for(i=0;i<4;i++)
    {
        for(j=0;j<5;j++)
            printf("%6d",*(p[i]+j));
        printf("\n");
    }
    printf("Array *(p+i)+j:\n");
    for(i=0;i<4;i++)
    {
        for(j=0;j<5;j++)
            printf("%6d",*(*(p+i)+j));
        printf("\n");
    }
    printf("Array *(a+i)+j:\n");
    for(i=0;i<4;i++)
```

```

    {
        for(j=0;j<5;j++)
            printf("%6d",*(a+i+j));
        printf("\n");
    }
} /*End for main*/

```

五、多维数组

当数组元素的下标在 2 个或 2 个以上时，该数组称为多维数组。其中以 2 维数组最常用。

定义多维数组：类型说明 数组名[整型常数 1][整型常数 2]...[整型常数 k];

例如：int a[2][3][3];

定义了一个三维数组 a[2][3][3]，其中每个数组元素为整型。总共有 $2 \times 3 \times 3 = 18$ 个元素。

【说明】

(1)对于三维数组，整型常数 1，整型常数 2，整型常数 3 可以分别看作“深”维（或：“页”维）、“行”维、“列”维。可以将三维数组看作一个元素为二维数组的一维数组。三维数组在内存中先按页、再按行、最后按列存放。

(2)多维数组在三维空间中不能用形象的图形表示。多维数组在内存中排列顺序的规律是：第一维的下标变化最慢，最右边的下标变化最快。

(3)多维数组的数组元素的引用：数组名[下标 1][下标 2]...[下标 k]。多维数组的数组元素可以在任何相同类型变量可以使用的位置引用。只是同样要注意不要越界。

第四部分 字符数组

字符数组：存放字符型数据的数组。其中每个数组元素存放的值都是单个字符。

字符数组分为一维字符数组和多维字符数组。一维字符数组常常存放一个字符串，二维字符数组常用于存放多个字符串，可以看作是一维字符串数组。

一、字符数组的定义、初始化及引用

字符数组也是数组，只是数组元素的类型为字符型。所以字符数组的定义、初始化，字符数组数组元素的引用与一般的数组类似。（定义时类型说明符为 char，

初始化使用字符常量或相应的 ASCII 码值，赋值使用字符型的表达式，凡是可以
用字符数据的地方也可以引用字符数组的元素）。

例如：

```
char c1[10],str[5][10];  
char c2[3]={'r','e','d'};或 char c2[]={ 'r','e','d'};  
printf(“%c%c%c\n”,c2[0],c2[1],c2[2]);
```

二、字符串与字符数组

1、字符串与字符数组

字符串（字符串常量）：字符串是用双引号括起来的若干有效的字符序列。C
语言中，字符串可以包含字母、数字、符号、转义符。

字符数组：存放字符型数据的数组。它不仅用于存放字符串，也可以存放一般
的、对一般读者看来毫无意义的字符序列。

C 语言没有提供字符串变量（存放字符串的变量），对字符串的处理常常采用
字符数组实现。因此也有人将字符数组看作为字符串变量。C 语言许多字符串处理
库函数既可以使用字符串，也可以使用字符数组。

为了处理字符串方便，C 语言规定以 '\0'（ASCII 码为 0 的字符）作为“字
符串结束标志”。“字符串结束标志”占用一个字节。对于字符串常量，C 编译系统
自动在其最后字符后，增加一个结束标志；对于字符数组，如果用于处理字符串，
在有些情况下，C 系统会自动在其数据后自动增加一个结束标志，在更多情况下结
束标志要由程序员自己负责（因为字符数组不仅仅用于处理字符串）。如果不是处
理字符串，字符数组中可以没有字符串结束标志。

例如：

```
char str1[]={ 'C','H','I','N','A'};  
str1:字符数组，占用空间 5 个字节
```

C	H	I	N	A
---	---	---	---	---

```
char str2[]="CHINA"; 占用空间 6 个字节
```

C	H	I	N	A	'\0'
---	---	---	---	---	------

2、字符数组的初始化（除了一般数组的初始化方法外，增加了一些方法）

(1)以字符常量的形式对字符数组初始化。

一般数组的初始化方法，给各个元素赋初值。注意：这种方法，系统不会自动在最后一个字符后加'\0'。

例如：

```
char str1[]={ 'C','H','I','N','A' };或 char str1[5]={ 'C','H','I','N','A' };
```

没有结束标志。如果要加结束标志，必须明确指定。

```
char str1[]={ 'C','H','I','N','A','\0' };
```

```
char str2[100]={ 'C','H','I','N','A' };
```

还有 $100-5=95$ 个字节暂时未使用，初始化为 0，相当于有字符串结束标志。

(2)以字符串（常量）的形式对字符数组初始化。（系统会自动在最后一个字符后加'\0'）

例如：

```
char str1[]={ "CHINA" };或 char s1[6]="CHINA";
```

```
char str2[80]={ "CHINA" };或 char s2[80]="CHINA";
```

还有 $100-6=94$ 个字节暂时未使用。

【说明】

以字符串常量形式对字符数组初始化，系统会自动在该字符串的最后加入字符串结束标志；以字符常量形式对字符数组初始化，系统不会自动在最后加入字符串结束标志。

3、字符数组的输入输出（两种形式：逐个字符输入/输出，整串输入/输出）

(1)逐个字符输入/输出：采用“%c”格式说明和循环，像处理数组元素一样输入输出。

【说明】

- 格式化输入是缓冲读。必须在接受到“回车”时，scanf 才开始读取数据。
- 读字符数据时，空格、回车都保存进字符数组。
- 如果按“回车”键时，输入的字符少于 scanf 循环读取的字符时，scanf 继续等待用户将剩下的字符输入；如果“回车”键时，输入的字符多于 scanf 循环读取的字符时，scanf 循环只将前面的字符读入。
- 逐个读入字符结束后，不会自动在末尾加'\0'。所以输出时，最好也使用逐个字符输出。

(2)整串输入/输出：采用“%s”格式符来实现

【说明】

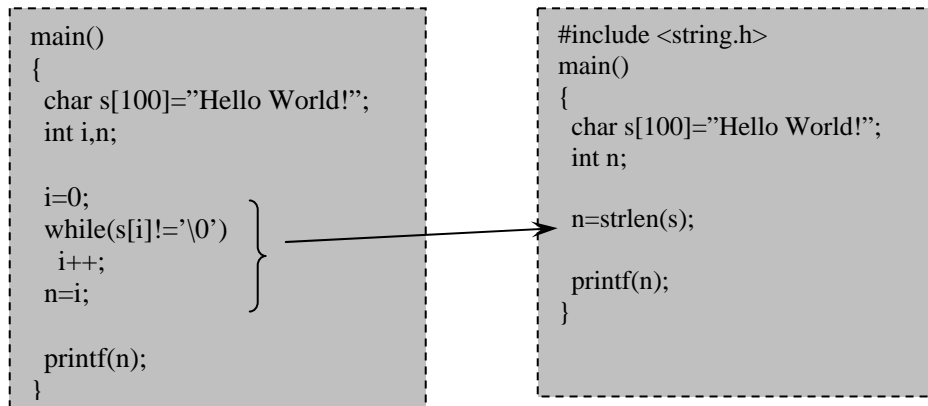
- 格式化输入输出字符串，参数要求字符数组的首地址，即字符数组名。

- 按照%s 格式格式化输入字符串时，输入的字符串中不能有空格（空格，Tab），否则空格后面的字符不能读入，scanf 函数认为输入的是两个字符串。如果要输入含有空格的字符串可以使用 gets()函数。
- 按照%s 格式格式化输入字符串时，并不检查字符数组的空间是否够用。如果输入长字符串，可能导致数组越界，应当保证字符数组分配了足够的空间。
- 按照%s 格式格式化输入字符串时，自动在最后加“字符串结束标志”。
- 按照%s 格式格式化输入字符串时，可以用“%c”或“%s”格式逐个输出。
- 不是按照%s 格式格式化输入的字符串在输出时，应该确保末尾有“字符串结束标志”。

第五部分 字符串处理函数及字符数组应用

字符串（字符数组）的处理可以采用数组的一般的处理方法进行处理-既对数组元素进行处理，这在对字符串中字符做特殊的处理时相当有效。C 语言库函数为我们提供了大量的字符串处理函数，对于一般的任务我们应当考虑是否可以采用库函数来解决问题。

例如：计算字符串的长度



程序 8.3 字符串长度计算程序

一、字符串处理函数

1、字符串输入，输出函数(<stdio.h>)

(1)字符串输入 gets(str);

功能：从键盘输入一个字符串（可包含空格），直到遇到回车符，并将字符串存放由 str 指定的字符数组（或内存区域）中。

参数: str 是存放字符串的字符数组 (或内存区域) 的首地址。函数调用完成后, 输入的字符串存放在 str 开始的内存空间中。

(2) 字符串输出 puts(str);

功能: 从 str 指定的地址开始, 依次将存储单元中的字符输出到显示器, 直到遇到 “字符串” 结束标志。

【注意】 puts 将字符串最后的 '\0' 转化为 '\n' 并输出。

2、字符串处理函数(<string.h>)

(1) 求字符串的长度 strlen(str)

功能: 统计 str 为起始地址的字符串的长度 (不包括 “字符串结束标志”), 并将其作为函数值返回。

(2) 字符串连接函数 strcat(str1, str2)

功能: 将 str2 为首地址的字符串连接到 str1 字符串的后面。从 str1 原来的 '\0' (字符串结束标志) 处开始连接。

【注意】

str1-一般为字符数组, 要有足够的空间, 以确保连接字符串后不越界;

str2-可以是字符数组名, 字符串常量或指向字符串的字符指针 (地址)。

(3) 字符串复制函数 strcpy(str1, str2)

功能: 将 str2 为首地址的字符串复制到 str1 为首地址的字符数组中。

【注意】

str1-一般为字符数组, 要有足够的空间, 以确保复制字符串后不越界;

str2-可以是字符数组名, 字符串常量或指向字符串的字符指针 (地址)。

字符串 (字符数组) 之间不能赋值, 但是通过此函数, 可以间接达到赋值的效果。

(4) 字符串比较函数 strcmp(str1, str2)

功能: 将 str1, str2 为首地址的两个字符串进行比较, 比较的结果由返回值表示。

当 str1=str2, 函数的返回值为: 0;

当 str1<str2, 函数的返回值为: 负整数; (绝对值是 ASCII 码的差值)

当 str1>str2, 函数的返回值为: 正整数; (绝对值是 ASCII 码的差值)

字符串之间的比较规则: 从第一个字符开始, 对两个字符串对应位置的字符按 ASCII 码的大小进行比较, 直到出现出现第一个不同的字符, 即由这两个字符的大小决定其所在串的大小。

字符串（字符数组）之间不能直接比较，但是通过此函数，可以间接达到比较的效果。

二、字符数组应用举例

例 8.7: 由键盘任意输入一个字符串和一个字符，要求从该字符串中删除所指定的字符。

【分析】

(1)很自然地考虑使用两个字符数组 s,temp。其中 s-存放任意输入的一个字符串；temp-存放删除指定字符后的字符串。

(2)设置两个整型变量 i,j 分别作为 s,temp 两个数组的下标（位置指针，索引），以指示正在处理的位置。

(3)开始处理前 $i=j=0$ ，即都是指向第一个数组元素。

(4)检查 s 中的当前的字符，

● 如果不是要删除的字符，那么将此字符复制（赋值）到 temp 数组,j 增 1（temp 下次字符复制的位置）；

● 如果是要删除的字符，不复制字符，j 也不必增 1（因为这次没有字符复制）。

(5)i 增 1（准备检查 s 的下面一个元素）

(6)重复(4)(5)直到 s 中的所有字符扫描了一遍。最后 temp 中的内容就是删除了指定字符的字符串。

【算法】

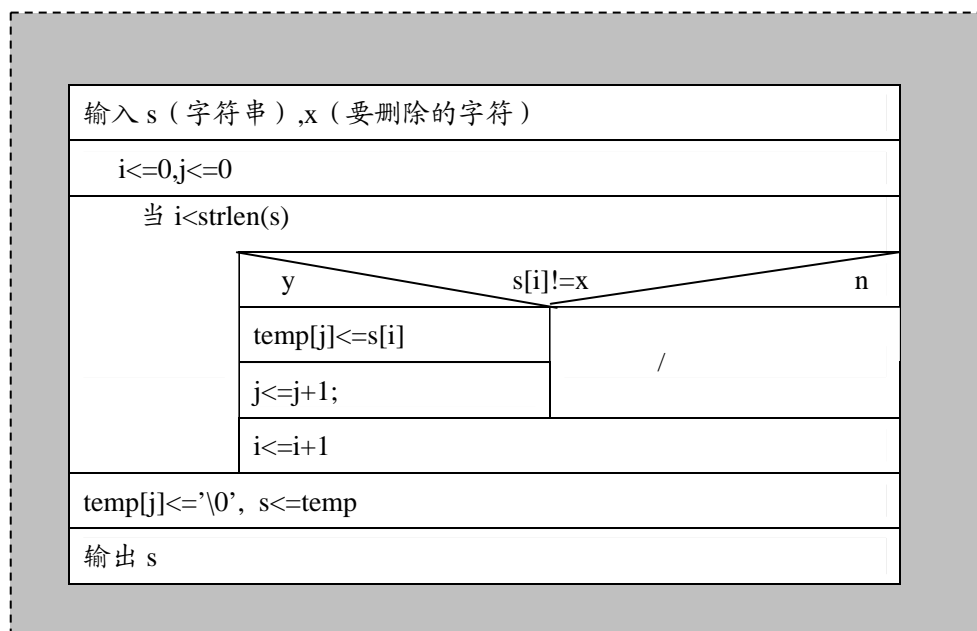


图 8.3 删除字符 N-S 图

【程序】

<pre>#include <stdio.h> main() { char s[20],temp[20],x; int i,j; gets(s); printf("delete?");scanf("%c",&x); for(i=0,j=0; i<strlen(s); i++) { if(s[i]!=x) { temp[j]=s[i]; j++; } } temp[j]='\0'; strcpy(s,temp); puts(s); }</pre> <p>运行程序 how do you do? delete?o hw d yu d?</p> <p>多设置了一个数组</p> <p>可以改为 s[j]=s[i]</p> <p>可以改为 s[j]='\0'</p> <p>删除</p>	<pre>#include <stdio.h> main() { char s[20],x; int i,j; gets(s); printf("delete?");scanf("%c",&x); for(i=0,j=0; i<strlen(s); i++) { if(s[i]!=x) { s[j]=s[i]; j++; } } s[j]='\0'; puts(s); }</pre> <p>运行程序 how do you do? delete?o hw d yu d?</p>
---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

程序 8.4 删除字符串程序

【思考】

不用 temp 数组，直接将 temp 数组改为 s 数组可以吗？为什么？

事实上将我们的算法中使用的 temp 直接改为 s，就是教材上的算法。为什么可以这样改呢？

第六部分 数组应用实例分析

【实例分析一】

```
main()
{
    char sa[][20]={"China","Japan","American"};
    char sb[][20]={{'C','h','i','n','a'},{'J','a','p','a','n'},{'A','m','e','r','i','c','a','n'}};
    char sc[][4]={'A','B','C','\0','D','E','F','\0','G','H','I','\0','J','K','L'};
};
int i,j;
for(i=0;i<3;i++)
```

```

        printf("%s ",sa[i]);
    printf("\n");
    for(i=0;i<3;i++)
    {
        for(j=0;sb[i][j];j++)
            printf("%c",sa[i][j]);
        printf(" ");
    }
    printf("\n");
    for(i=0;i<4;i++)
        printf("%s ",sc[i]);
}

```

【实例分析二】

对 10 个数由小到大排序（用选择法：每轮比较求出最小数的下标，然后和前面的数交换）

```

main()
{
    int i,j,k,a[10]={ 12,65,78,14,34,98,102,32,64,87};
    for(i=0;i<9;i++)
    {
        k=i;
        for(j=i+1;j<10;j++)
            if(a[k]>a[j])
                k=j; /* 从小到大 */
                /* if(a[k]<a[j]) k=j; 从大到小 big to small */
        if(k!=i)
            a[i]=a[i]+a[k],a[k]=a[i]-a[k],a[i]=a[i]-a[k];
    } /*End for*/
    for(i=0;i<10;i++)
        printf("%d ",a[i]);
}

```

【实例分析三】

自定义函数计算任意字符串的长度（实现函数 strlen()的功能）。

```

main()
{
    char *p="This is a c program!";
    printf("%d",mystrlen(p));
}
int mystrlen(char *s)
{
    int i; /* for(i=0;*s;i++,s++) ; */
    for(i=0;*s++;i++)
        return(i);
}

```

【实例分析四】

自定义函数实现任意两个字符串的连接(实现函数 strcat ()的功能)。

```
main()
{
    char *mystrcat(char *s1,char *s2);
    char *p1="This is a c program!";
    char *p2="My c program Test";
    printf("%s",mystrcat(p1,p2));
}
char * mystrcat(char *s1,char *s2)
{
    char *p=s1;for(;;*p;p++) ;
    for(;;(*p++=*s2++)!='\0')
        return(s1);
}
```

【实例分析五】

自定义函数实现任意两个字符串的复制(实现函数 strcpy()的功能)。

```
main()
{
    char *mystrcpy(char *s1,char *s2);
    char *p1="", *p2="My c program Test";
    printf("%s",mystrcpy(p1,p2));
}
char * mystrcpy(char *s1,char *s2)
{
    char *p=s1;
    for(;;(*p++=*s2++)!='\0')
        return(s1);
}
```

【实例分析六】

自定义函数实现将任意字符串中的大写字母转换为小写字母(实现函数 strlwr()的功能)。

```
void mystrlwr(char s[])
{
    int i=0;
    for(;;s[i];i++)
        if(s[i]>='A'&&s[i]<='Z')
            s[i]+=32;
}
char *slwr(char s[])
{
    int i=0;
    char *p=s;
```

```

    for(;s[i];i++)
        if(s[i]>='A'&& s[i]<='Z')
            s[i]+=32;
    return(p);
}

```

```

main()
{
    char s[]="CHina";
    mystrlwr(s);
    printf("%s  %s",s,slwr(s));
}

```

【实例分析七】

```

main()
{
    char a[4][6]={"China","Japan","Langua","chines"};
    char b[]="This is a c program";
    char s[5]={'A','B','C','D','E'};
    int i=0;
    printf("b[]:%s == %d\n",b,strlen(b));
    printf("s[]:%s ==%d\n",s,strlen(s));
    for(i=0;i<4;i++)
        printf("a[%d]:%s == %d\n",i,a[i],strlen(a[i]));
}

```

【字符数组与字符串的区别】

字符串存放在字符数组中；但字符串可以与字符数组不等长；字符串以‘\0’作为结束标志；

【实例分析八】

交换有 n 个元素的一维数组的元素值。

```

#define PN printf("\n");
void inv(int x[],int n)
{
    int t,i=0,j,m=(n-1)/2;
    for(i=0;i<=m;i++)
    {
        j=n-1-i;
        t=x[i];
        x[i]=x[j];
        x[j]=t;
    }
}
main()
{
    int a[10]={10,20,30,40,50,60,70,80,90,100},*p,i;
    for(i=0;i<10;i++)

```



```

        printf("%-6d",*(a+i));
    PN inv(a,10);
    for(p=a,i=0;i<10;i++)
        printf("%-6d",*p++);
    PN
}

```

【本例分析】

(1)本题的关键是元素的交换顺序：将 $a[0]$ 与 $a[n-1]$ 交换；再将 $a[1]$ 与 $a[n-2]$ 交换；最后将 $a[(n-1)/2]$ 与 $a[n-(n-1)/2]$ 交换；前面的元素个数为： $(n-1)/2$ ；

(2)当 n 为奇数时，最后一次将 $a[(n-1)/2]$ 与 $a[n-1-(n-1)/2]$ 交换；例： $n=11$ 最后一次将 $a[(11-1)/2]$ 与 $a[11-1-(11-1)/2]$ 交换；即 $a[5]$ 和 $a[5]$ 交换；

(3)当 n 为偶数时，最后一次将 $a[(n-1)/2]$ 与 $a[n-1-(n-1)/2]$ 交换；例： $n=10$ 最后一次将 $a[(10-1)/2]$ 与 $a[10-1-(10-1)/2]$ 交换；即 $a[4]$ 和 $a[5]$ 交换；

▲可能出现的“警告”提示：Possible incorrect assignment(可能的不正确赋值)；当编译程序遇到赋值操作符作为条件表达式的主操作符时，会发出警告，通常是由于把赋值号当作符号使用了。可以将它用 () 括起来，且与 0 作显示比较即可消除。

例：for(*a++=*b++); 改为：for((*a++=*b++)!='\0');

【实例分析九】

当形式参数为数组名时，该数组名是变量，相当与一个指针变量,可进行++、--运算，指针在变化。

```

char *scat(char a[],char *s)
{
    char *t=a;
    printf("%p\n",&a[0]);
    for(;*a;a++)
        printf("%c[%p]\n",*a,a);
    for((*a++=*s++)!='\0');
    printf("%p\n",&a[0]);
    return(t);
}
char *sre(char b[][9])
{
    b++;
    printf("%s\n",*b++);
    return(*b);
}
main()
{
    char *p="china",*q="Japan";
    char s[][9]={"China","Japan","American"};
}

```

```
printf("%s\n",scat(p,q));  
printf("%s",sre(s));  
}
```

【本例分析】

二维形式参数数组名自加，相当于行指针变量自加，即移动一行,指针在变。
一维形式参数数组名自加，相当于列指针变量自加，即移动一列,指针在变。

第九章

编译预处理

【基本要求】

通过本单元的学习，使学生了解宏的基本知识，熟悉无参数和有参数的宏的定义和使用，掌握文件包含的应用，掌握条件编译及其应用。

【教学重点】

宏定义（无参数的宏；有参数的宏），文件包含，条件编译

【本章结构】

- 1、宏定义
 - 不带参数的宏定义
 - 带参数的宏定义
- 2、“文件包含”处理
- 3、条件编译

编译指令（编译预处理指令）：C 源程序除了包含程序命令（语句）外，还可以使用各种编译指令（编译预处理指令）。编译指令（编译预处理指令）是给编译器的工作指令。这些编译指令通知编译器在编译工作开始之前对源程序进行某些处理。编译指令都是用“#”引导。

编译预处理：编译前根据编译预处理指令对源程序的一些处理工作。C 语言编译预处理主要包括宏定义、文件包含、条件编译。

编译工作实际分为两个阶段：编译预处理、编译。广义的编译工作还包括连接。过程如图 9.1 所示

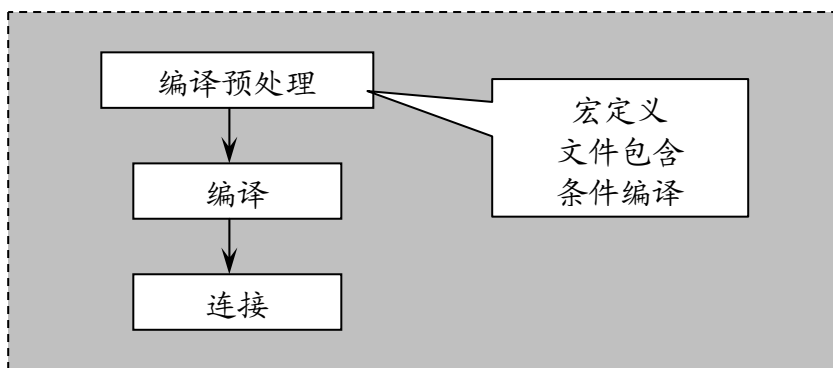


图 9.1 编译过程示意图

一、宏定义

宏定义：用标识符来代表一个字符串（给字符串取个名字）。C 语言用“#define”进行宏定义。C 编译系统在编译前将这些标识符替换成所定义的字符串。

宏定义分为不带参数的宏定义和带参数宏定义。

1、不带参数宏定义（简单替换）

(1)不带参数宏定义格式

#define 标识符 字符串

其中：标识符-宏名。

(2)宏调用：在程序中用宏名替代字符串。

(3)宏展开：编译预处理时将字符串替换宏名的过程，称为宏展开。

例 9.1:

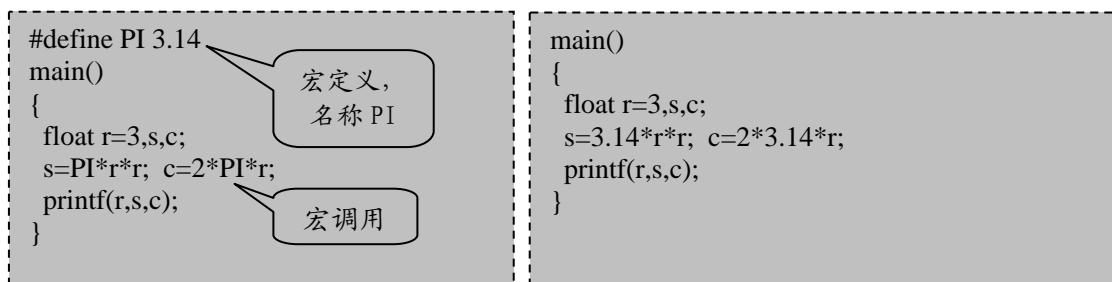


图 9.2 宏的定义及调用

【说明】

(1)宏名遵循标识符规定，习惯用大写字母表示，以便区别普通的变量。

(2)#define 之间不留空格，宏名两侧空格（至少一个）分隔。

(3)宏定义字符串不要以分号结束，否则分号也作为字符串的一部分参加展开。

从这点上看宏展开实际上是简单的替换。

例如：#define PI 3.14; 展开为 s=3.14; *r*r; (导致编译错误)

(4)宏定义用宏名代替一个字符串，并不管它的数据类型是什么，也不管宏展开后的词法和语法的正确性，只是简单的替换。是否正确，编译时由编译器判断。

例如：#define PI 3.14 照样进行宏展开（替换），是否正确，由编译器来判断。

(5)#define 宏定义宏名的作用范围从定义命令开始直到本源程序文件结束。可以通过#undef 终止宏名的作用域。(如图 9.3 所示)

(6)宏定义中，可以出现已经定义的宏名，还可以层层置换。(如图 9.4 所示)

(7)宏名出现在双引号“”括起来的字符串中时，将不会产生宏替换。（因为出现在字符串中的任何字符都作为字符串的组成部分）

(8)宏定义是预处理指令，与定义变量不同，它只是进行简单的字符串替换，不分配内存。

(9)使用宏的优点:

- 程序中的常量可以用有意义的符号代替, 程序更加清晰, 容易理解 (易读)。
- 常量值改变时, 不要在整个程序中查找, 修改, 只要改变宏定义就可以。比如, 提高 PI 精度值。
- 带参数宏定义比函数调用具有更高的时间效率, 因为相当于代码的直接嵌入。(空间效率: 多次调用占用空间较多, 一次调用没有什么影响)。

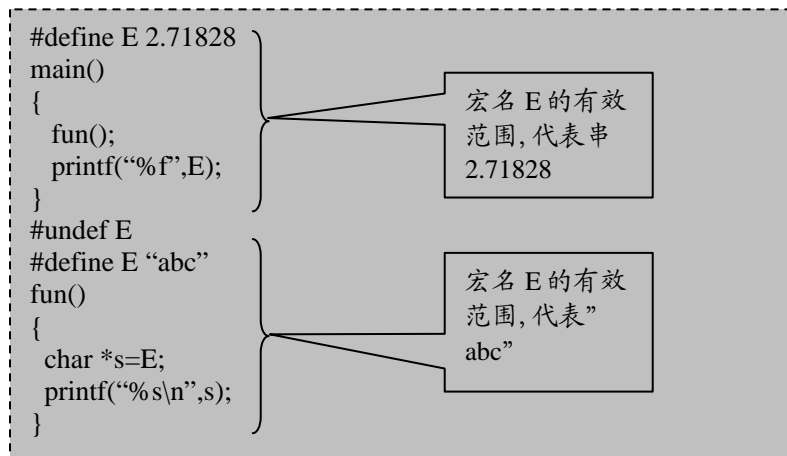


图 9.3 宏的作用范围

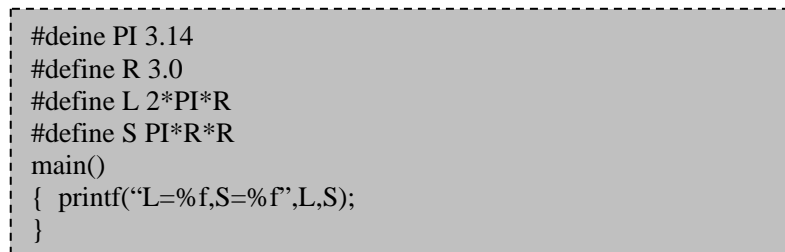


图 9.4 宏的层层置换

2、带参数宏定义

带参数宏定义不只是进行简单的字符串替换, 还要进行参数替换。

(1)带参数宏定义的格式:

`#define 宏名(参数表) 字符串`

类似函数头, 但是没有类型说明, 参数也不要类型说明。

例如:

```
#define S(a, b) a*b
```

其中 S-宏名, a, b 是形式参数。

程序调用 S(3,2)时, 把实参 3, 2 分别代替形参 a, b。

`area=S(3,2); => area=3*2;`

(2)带参数宏定义展开规则:

在程序中如果有带实参的宏定义，则按照#define 命令行中指定的“字符串”从左到右进行置换（扫描置换）。如果串中包含宏定义中的形参，则将程序中相应的实参代替形参，其它字符原样保留，形成了替换后的字符串。

【注意】这是一个字符串的替换过程，只是将形参部分的字符串用相应的实参字符串替换。

例 9.1 用带参数宏定义表示两数中的较大数

```
#define MAX(a,b) (a>b)?a:b
main()
{
    int i=15,j=20;
    printf("MAX=%d\n",MAX(i,j));=> printf("MAX=%d\n",(i>j)?i:j);
}
```

宏展开:
a, b 用 i, j

图 9.5 带参数宏的使用

【注意】

(1)正因为带参宏定义本质还是简单字符替换（除了参数替换），所以容易发生错误。

例如：

```
#define S(a,b) a*b
```

程序中 `area=S(a+b,c+d);=>area=a+b*c+d`;明显和我们的意图不同。

假如：宏定义的字符串中的形参用（）括号括起来，即：

```
#define S(a,b) (a)*(b)
```

此时程序中：

```
area=S(a+b,c+d);=>area=(a+b)*(c+d);
```

符合我们的意图。

为了避免出错，建议将宏定义“字符串”中的所有形参用括号（）括起来。以后，替换时括号作为一般字符原样照抄，这样用实参替换时，实参就被括号括起来作为整体。不至于发生类似错误。

(2)定义带参数宏时还应该注意宏名与参数表之间不能有空格。有空格就变成了不带参数的宏定义。

如：`#define S(r) PI*r*r`

```
area=S(3.0);=>area=(r)PI*r*r(3.0);
```

(3)带参数的宏定义在程序中使用时，它的形式及特性与函数相似，但本质完全不同。区别在下面几个方面：

- 函数调用，在程序运行时，先求表达式的值，然后将值传递给形参；带参宏展开只在编译时进行的简单字符置换。

- 函数调用是在程序运行时处理的，在堆栈中给形参分配临时的内存单元；宏展开是在编译时进行，展开时不可能给形参分配内存，也不进行“值传递”，也没有“返回值”。
- 函数的形参要定义类型，且要求形参、实参类型一致。宏不存在参数类型问题。

如：程序中可以用 MAX(3,5)也可以 MAX(3.4,9.2)

(4)许多问题可以用函数也可以用带参数的宏定义。

(5)宏占用的是编译时间，函数调用占用的是运行时间。在多次调用时，宏使得程序变长，而函数调用不明显。

例 9.2: 计算三角形面积。（使用带参数宏定义）

作业：

(1)将 S(a,b,c)改为 S(a1,b1,c1);AREA(a,b,c)改为 AREA(a2,b2,c2).将 main () 中 a,b,c 改为 i,j,k 重新写 9-3 程序。

(2)对最后 printf 进行宏展开（分两步走）。

二、文件包含

文件包含：一个 C 源文件可以使用文件包含命令将另外一个 C 源文件的全部内容包含进来。

格式：#include “文件名”或#include <文件名>

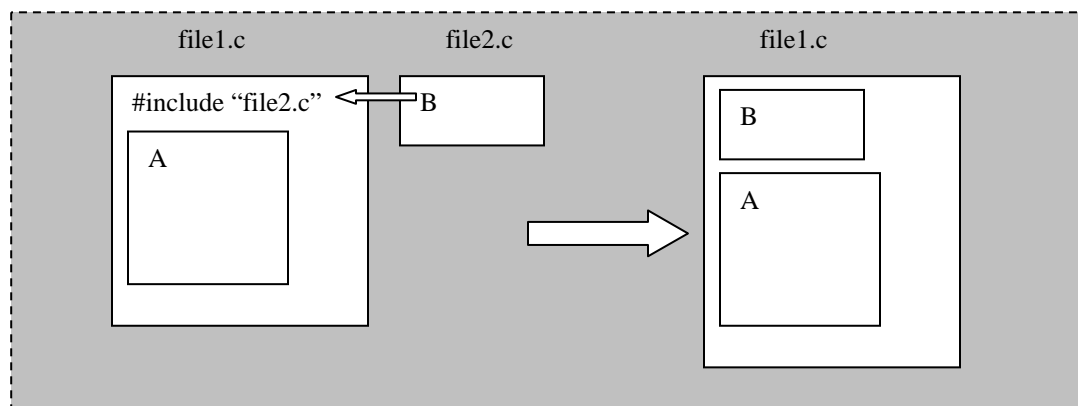


图 9.6 文件包含示意

【说明】

(1)被包含的文件常常被称为“头文件”（#include 一般写在模块的开头）。头文件常常以“.h”为扩展名（也可以用其它的扩展名，.h 只是习惯或风格）。

(2)一条#include 只能包含一个头文件，如果要包含多个头文件，使用多条#include 命令。

(3)被包含的头文件可以用“”括起来，也可以用<>括起来。区别在于：<>先在C系统目录中查找头文件，“”先在用户当前目录查找头文件。

习惯上，用户头文件一般在用户目录下，所以常常用“”；系统库函数的头文件一般在系统指定目录下，所以常常用<>。

(4)头文件的应用（补充）。

在多模块应用程序的开发上，经常使用头文件组织程序模块。

- 头文件成为共享源代码的手段之一。程序员可以将模块中某些公共内容移入头文件，供本模块或其它模块包含使用。比如，常量，数据类型定义。
- 头文件可以作为模块对外的接口。比如，可以供其它模块使用的函数、全局变量声明。
- 头文件常常包含如下内容：
 - 用户定义的常量
 - 用户定义的数据类型
 - 用户模块中定义的函数和全局变量的声明。

三、条件编译

1、条件编译的作用

使对源程序的部分内容指定编译条件,使得满足条件的行才被编译;

2、命令格式与执行编译方式

【条件语句与条件编译的区别】

```
/*所有语句均被译*/
int max(int x,int y){ if(x>y) return(x); else retrun(y); }
/*部分语句被编译*/
#define OK 1
main()
{
    char s[]="C Language",c;
    int i=0;while((c=s[i++])!='\0')
    {
        #if OK if(c>='a' && c<='z') c-=32;
        #elseif(c>='A' && c<='Z') c+=32;
        #endifprintf("%c",c);
    }
}
```

四、宏的实例分析

有参数的宏与有参数的自定义函数的区别

1、函数调用时是将函数的实参表达式的结果值传递给对应的形式参数，即先对实参表达式进行计算。传递的是表达式的结果值，而不是表达式整体。而带参数的宏执行时，是将实参表达式本身简单地替换了宏体中对应的变量名而已，替换之前没有计算过程。

例：

```
#define SUM(x,y) x*y
int add(int x,int y){ return (x*y); }

main()
{
    int a,b;a=SUM(2+3,5+6); /*宏替换*/
    b=add(2+3,5+6); /*函数调用*/
    printf("a=%d b=%d",a,b);
}
```

以上程序执行时，`a=SUM(2+3,5+6)`；将被替换为：`a=2+3*5+6`；只是简单地用宏体 `x*y` 代替 `SUM(2+3,5+6)`，再将 `x` 替换为 `2+3`，`y` 替换为 `5+6`。所以结果 `a=23`。以上程序执行时，`b=add(2+3, 5+6)`；是将 `5` 传递给对应的形式参数变量 `x`，将 `11` 传递给对应的形式参数变量 `y`，函数的返回值为 `55`。

2、函数的调用是在程序运行时处理的，分配的临时存储单元。而宏展开则是在编译时进行的，在展开时并不分配存储单元，宏替换没有返回值。

3、函数定义时对形式参数和函数本身的返回值要指定数据类型（包括空类）；定义宏时不存在类型问题，宏名无类型，宏的参数也无类型，宏的参数仅仅只是一个符号而已。但要注意有参数的宏在替换时可能引起的与宏定义时的本意不相符合的“副作用”。如上述例中宏定义的本意是求 `x` 乘以 `y` 的结果值，但结果不正确。解决办法：将宏体和宏体中的参数均用小括号 `()` 括起来。

例： `#define SUM(x,y) ((x)*(y))`

实例分析之一

定义宏实现求 `n!`

```
#include "stdio.h"#include "factn.h" /*包含自定义头文件，其中定义了函数 fatn
*/

#define fact(n) float t=1; int i; for(i=1;i<=n;i++) t=t*i; /*定义宏实现求 n! */
float fat(int n)
{
    /*定义函数 fact 实现求 n! */
    float f=1; int k=1; for(;k<=n;k++) f=f*k;return(f);
}

main()
```

```
{
fact(5) printf("5!=%.0f\n",t); printf("5!=%.0f\n",fat(5)); printf("5!=%.0f\n",fatn(5));
}
```

实例分析之二

当无参数的宏定义，宏体为表达式时宏的替换方式。宏可能产生的副作用。

```
#define N 2+5 /*可能产生副效应*/
#define M 10*N /*可能产生副效应*/
#define S 5*(N) /*嵌套的宏，可能产生副效应*/
#define T (5*(N)) /*加上括号之后不产生副效应*/
main(){ int a=10;
printf("a=10 N=2+5 M=10*N S=5*(N) T=(5*(N))\n");
printf("a*N=%d N*a=%d\n",N*a,a*N);
printf("a/N=%d N/a=%d\n",a/N,N/a);
printf("a*N=%d N*a=%d\n",a*N,N*a);
printf("a/N=%d N/a=%d\n",a/N,N/a);
printf("a*M=%d M*a=\n",a*M,M*a );
printf("a*S=%d a/S=%d\n",a*S,a/S);
printf("a*T=%d T*a=%d\n",a*T,T*a);
printf("a/T=%d T/a=%d\n",a/T,T/a);
}
```

请认真分析以上宏定义在具体执行过程中的替换方式，以及程序的运行结果极其原因。

实例分析之三

有参数的宏定义，宏体的替换方式。有参数的宏可能产生的副作用。

```
#define S(a,b) a*b /*可能产生的副作用*/
#define M(a,b) (a*b) /*可能产生的副作用*/
#define G(a,b) ((a)*(b)) /*不产生副作用 */
main(){
int a=10; printf("S(a,b)=a*b\n");
printf("S(2+3,4+5)=%d\n",2+3*4+5);
printf("S(4+5,2+3)=%d\n",4+5*2+3);
printf("S(2+3,4+5)/a=%d\n",2+3*4+5/a);
printf("a/S(2+3,4+5)=%d\n",a/2+3*4+5);
printf("M(a,b)=(a*b)\n");
printf("M(2+3,4+5)=%d\n",(2+3*4+5));
printf("M(4+5,2+3)=%d\n",(4+5*2+3));
```

```

    printf("M(2+3,4+5)/a=%d\n",(2+3*4+5)/a);
printf("a/M(2+3,4+5)=%d\n\n",a/(2+3*4+5));
    printf("G(a,b)=((a)*(b))\n");
    printf("G(2+3,4+5)=%d\n",((2+3)*(4+5)));
printf("G(4+5,2+3)=%d\n",((4+5)*(2+3)));
printf("G(2+3,4+5)/a=%d\n",((2+3)*(4+5))/a);
    printf("a/G(2+3,4+5)=%d\n",a/((2+3)*(4+5)));
}

```

请认真分析以上宏定义在具体执行过程中的替换方式，以及程序的运行结果极其原因。

有参数的宏的定义时，宏体及其各形式参数，即其中的变量应用括号括起来，否则可能有副作用。

第十章

结构体、共用体和用户定义类型

【基本要求】

通过本单元的学习，使学生掌握结构体的概念，掌握结构体类型的定义，掌握结构体变量、结构体数组的定义和引用，掌握指向结构体的指针的定义和使用，掌握结构体数据作函数参数的用法，了解枚举类型的定义和使用，了解使用 typedef 定义类型的方法。

【教学重点】

结构体数据类型及其变量、结构体变量指针与其指针变量，共用体数据类型及其变量、共用体变量指针与其指针变量，枚举数据类型及其变量，数据类型再定义 [typedef]，结构体/结构体指针变量与单向链表

【本章结构】

- 1、结构体
 - 结构体类型变量的定义
 - 结构体变量的引用
 - 结构体变量的初始化
 - 结构体数组
 - 指向结构体类型数据的指针
- 2、共用体
 - 共用体的概念
 - 共用体变量的引用方式
 - 共用体类型数据的特点
- 3、枚举类型

第一部分 结构体变量定义及引用

例子：新生入学登记表，要记录每个学生的学号，姓名，性别，年龄，身份证号，家庭住址，家庭联系电话等信息。

学号	姓名	性别	年龄	身份证号	家庭住址	家庭联系电话
11301	pin. zhang	F	19	320406841001264	changzhou	(0519)8754267
11302	min. li	M	20	612301830314261	xi'an	(029)3870909

使用数组：

因为要有很多学生的信息要处理，按照我们前面学习过的知识，这个任务要使用数组。但是数组是由相同类型的数据构成。所以我们可以使用 7 个单独的数组（学号数组 no、姓名数组 name、性别数组 sex、年龄数组 age、身份证号数组 pno、家庭住址数组 addr、家庭联系电话数组 tel）分别保存这几类信息。分立的几个数组将给数据的处理造成麻烦，但很多计算机语言只能这样处理（如：早期的 FORTRAN，PASCAL，BASIC）。

使用结构体：

C 语言利用结构体将同一个对象的不同类型属性数据，组成一个有联系的整体。也就是说可以定义一种结构体类型将属于同一个对象的不同类型的属性数据组合在一起。本例可以将属于同一个学生的各种不同类型的属性数据组合在一起，形成整体的结构体类型数据。可以用结构体类型变量存储、处理单个学生的信息。

结构体是一种自定义数据类型。如果要存储、处理多个学生（对象）的信息，可以使用数组元素为结构体类型的数组，其中每个元素是一个学生（对象）的相关的整体的信息。

结构体是一种构造类型（自定义类型），除了结构体变量需要定义后才能使用外，结构体的类型本身也需要定义。结构体由若干“成员”组成。每个成员可以是一个基本的数据类型，也可以是一个已经定义的构造类型。

一、结构体类型定义的一般形式

```
struct 结构体名
{
    类型 1 成员 1;
    类型 2 成员 2;
    .....
    类型 n 成员 n;
};
```

【说明】

(1)结构体名：结构体类型的名称。遵循标识符规定。

(2)结构体有若干数据成员，分别属于各自的数据类型，结构体成员名同样遵循标识符规定，它属于特定的结构体变量（对象），名字可以与程序中其它变量或标识符同名。

(3)使用结构体类型时，struct 结构体名作为一个整体，表示名字为“结构体名”的结构体类型。

(4)结构体类型的成员可以是基本数据类型，也可以是其它的已经定义的结构体类型-结构体嵌套。结构体成员的类型不能是正在定义的结构体类型（递归定义，结构体大小不能确定），但可以是正在定义的结构体类型的指针。

例如：定义关于学生信息的结构体类型。

```
struct student
{
    int no; char name[20]; char sex; int age; char pno[19];
    char addr[40]; char tel[10];
};
```

【说明】

(1)struct student 是结构体类型名，struct 是关键词，在定义和使用时均不能省略。

(2)该结构体类型由 7 个成员组成，分别属于不同的数据类型，分号“;”不能省略。成员含义同前。

(3)在定义了结构体类型后，可以定义结构体变量（int 整型类型，可以定义整型变量）。

二、结构体变量的定义（三种方法）

1、先定义结构体类型，再定义结构体变量（概念、含义相当清晰），即：

结构体类型定义；

结构体变量定义；

其中：结构体变量定义：struct 结构体类型名 结构体变量名；

例如：

```
struct student{.....}; /* 类型定义，定义结构体类型 struct student */
```

```
struct student student1,student2; /* 变量定义，定义 2 个类型为 struct student */  
/* 的结构体变量 student1,student2 */
```

2、定义结构体类型的同时定义结构体变量。

struct 结构体名

{ ...

(成员)

...

}结构体变量名表;

例如: struct student{.....}student1,student2;

【说明】

- 这一种紧凑的格式，既定义类型，也定义变量;
- 如果需要，在程序中还可以使用所定义的结构体类型，定义其它同类型变量。

3、直接定义结构体变量（不给出结构体类型名，匿名的结构体类型）

struct

{ ... (成员) ...

}结构体变量名表;

例如: struct {...}student1,student2;

结构体类型、变量是不同的概念:

- 在定义时一般先定义一个结构体类型，然后定义变量为该类型;
- 赋值、存取或运算只能对变量，不能对类型;
- 编译时只对变量分配空间，对类型不分配空间。

三、结构体变量的引用

1、引用结构体变量中的一个成员

结构体变量名.成员名

其中：“.”运算符是成员运算符。

例如:

```
student1.num=11301;
```

```
scanf("%s",student1.name); if(strstr(student1.addr,"shanxi")!=NULL)...;
```

```
student1.age++;
```

2、成员本身又是结构体类型时的子成员的访问-使用成员运算符逐级访问。

例如:

```
student1.birthday.year
```

```
student1.birthday.month
```

```
student1.birthday.day
```

3、同一种类型的结构体变量之间可以直接赋值(整体赋值, 成员逐个依次赋值)。

例如: student2=student1;

4、不允许将一个结构体变量整体输入/输出

例如: scanf("%...",&student1); printf("%...",student1); 都是错误的。

【说明】

(1)结构体变量的引用:求结构变量的地址;同类结构变量之间相互赋值;结构体变量作为实参时可引用结构体变量作为整体引用;

(2)一般情况下对结构变量的引用是通过结构体变量成员的引用实现的.其格式为: 结构体变量名.成员名

(3)若结构体成员有嵌套的情况, 只能逐级访问和存取最低层的成员变量.

(4)可以对结构体变量名的成员进行各种相关的运算;

(5)结构体成员变量的引用有两种方式[三种形式]:通过结构体变量名引用;通过结构体指针变量引用.

(6)同一函数中结构体变量成员与结构体外的同名变量互不影响; 不同结构体变量的同名成员互不影响;

第二部分

一、结构体变量的初始化

结构体变量也可以在定义时进行初始化，但是变量后面的一组数据应该用“{}”括起来，其顺序也应该与结构体中的成员顺序保持一致。

```
main()
{
    struct student
    {
        int no; char name[20]; char sex; int age; char pno[19];
        char addr[40]; char tel[20];
    } student1={11301,"Zhang Ping",'F',19,
        "320406841001264","changzhou","(0519)8754267"};

    printf("no=%d,name=%s,sex=%c,age=%d,pno=%s\naddr=%s,tel=%s\n",\
        student1.no,student1.name,student1.sex,student1.age,\
        student1.pno,student1.addr,student1.tel);
}
结果:
no=11301,name=Zhang Ping,sex=F,age=19,pno=320406841001264
addr=changzhou,tel=(0519)8754267
```

程序 10.1 结构体变量初始化

本例中，结构体变量 student1 在定义的同时，其各个成员也按顺序被赋予了相应的一组数据。

二、结构体变量的初始化方法

1、定义时直接初始化其成员变量；但应注意初始值的类型\个数\顺序与定义时指定的成员的类型\个数\顺序成一一对应关系,数据类型\个数\顺序应相匹配；

2、先定义结构体变量再初始化其成员变量；但应注意初始值的类型\个数\顺序与定义时指定的成员的类型\个数\顺序成一一对应关系,数据类型\个数\顺序应相匹配；

3、可以将其成员已经初始的结构体变量赋值给同类的结构体变量名（即整体赋值）；同类结构体变量之间可以相互赋值；同类结构体变量的相同成员之间可以相互赋值；

4、不能将一组常量赋值给一个结构体变量,但在定义结构体变量时可以直接用一组常量值初始化该变量.

三、结构体变量的输入/输出方法

不能将一个结构体变量作为一个整体进行输入/输出，只能对结构体变量各成员进行输入输出；

结构体变量的作为函数参数

(1)当结构体变量作为函数的参数时,形式参数的变量成员值的改变不能影响实参结构体的成员的值;

(2)当结构体变量作为函数的参数时,实参与形式参数的类型应完全一致[相同];

返回结构体变量的函数的返回值可以是基本类型\指针类\空类;也可以是结构体类;

四、结构体数组

结构体数组-数组元素的类型为结构体类型的数组。

C 语言允许使用结构体数组存放一类对象的数据。

1、结构体数组的定义

类似结构体变量定义,只是将“变量名”用“数组名[长度]”代替),也有 3 种方式。

(1)先定义结构体类型,然后定义结构体数组:

struct 结构体名 {.....}; struct 结构体名 结构体数组名[];

(2)定义结构体类型同时定义结构体数组:

struct 结构体名 {.....} 结构体数组名[数组的长度];

(3)匿名结构体类型

struct {.....} 结构体数组名[数组的长度];

	学号	姓名	性别	年龄	身份证号	家庭住址	家庭联系电话
stu[0]	11301	pin. zhang	F	19	320406841001264	changzhou	(0519)8754267
stu[1]	11302	min. li	M	20	612301830314261	xi'an	(029)3870909
	...						
stu[29]							

图 10.1 结构体数组示例

例如: 定义 30 个元素的结构体数组 stu,其中每个元素都是 struct student 类型。

```
struct student{  
    int no; char name[20]; char sex; int age; char pno[19];
```

```
char addr[40]; char tel[20];  
}stu[30];
```

定义了结构体数组后，可以采用：数组元素.成员名。引用结构体数组某个元素的成员。

2、结构体数组的初始化:

1)结构体数组元素的初始化: [行方式]将每个元素的成员变量的初值分别用{}括起来; 再将整个数据用{}括起来(即行方式初始化之)。[列方式]将所有元素的成员变量的初值用一个{}括起来; 系统按照先后顺序给每个成员赋值,但应**注意**: 初始值的类型\个数\顺序与定义时指定的成员的类型\个数\顺序成一一对应关系,数据类型\个数\顺序应相匹配;

2)可以在定义数组时直接初始化;也可以先定义后初始化之,但不能将一组常量赋值给一个结构体变量数组元素,但在定义结构体变量时可以直接用一组常量值初始化该元素。

3、结构体数组的输出

不能将一个结构体数组元素作为一个整体进行输入/输出,只能对结构体元素各成员进行输入输出。

4、结构体数组作为函数的参数

当结构体数组作为函数的参数时,形式参数变量的成员值的改变将影响实参的成员的值。

第三部分 结构体指针变量

结构体指针变量: 指向结构体变量的指针变量。结构体指针变量的值是结构体变量(在内存中的)起始地址。

一、结构体指针变量的定义

```
struct 结构体名 *结构体指针变量名;
```

例如:

struct student *p;定义了一个结构体指针变量,它可以指向一个 struct student 结构体类型的数据。

二、通过结构体指针变量访问结构体变量的成员(两种访问形式)

(1) (*结构体指针变量名).成员名。(理解: *结构体指针变量名=所指向的结构体变量名,注意: “.” 运算符优先级比 “*” 运算符高)

(2) 结构体指针变量名->成员名（其中：“->”是指向成员运算符，很简洁，更常用）

例如：可以使用(*p).age 或 p->age 访问 p 指向的结构体的 age 成员。

例 10.1：用指针访问结构体变量及结构体数组

（数组的指针就是指向其元素的指针，访问数组元素和访问变量所需要定义的指针变量完全相同；指向数组元素和指向变量的指针变量在使用上也完全相同）。

```
main()
{
    struct student /* 结构体类型定义 */
    {
        int num;
        char name[20];
        char sex;
        int age;
        float score;
    };

    /* 结构体数组 stu，结构体变量 student1 定义和初始化 */
    struct student stu[3]={11302,"Wang",'F',20,483},
        {11303,"Liu",'M',19,503},
        {11304,"Song",'M',19,471.5}};
    struct student student1={11301,"Zhang",'F',19,496.5},*p,*q;
    int i;

    /* p 指向结构体变量 */
    p=&student1;
    printf("%s,%c,%5.1f\n",student1.name,(*p).sex,p->score); /* 访问结构体变量 */

    /* q 指向结构体数组的元素 */
    q=stu;
    for(i=0; i<3; i++,q++) /* 循环访问结构体数组的元素（下标变量） */
        printf("%s,%c,%5.1f\n",q->name,q->sex,q->score);
}
结果：
Zhang,F,496.5
Wang,F,483.0
Liu,M,503.0
Song,M,471.5
```

程序 10.2 指针访问结构体变量及结构体数组

三、结构体变量、结构体指针变量作函数参数

结构体变量、结构体指针变量都可以像其它数据类型一样作为函数的参数，也可以将函数定义为结构体类型或结构体指针类型（返回值为结构体、结构体指针类型）。

例 10.2: 对年龄在 19 岁以下 (含 19 岁) 同学的成绩增加 10 分。

```
struct student
{
    int num;
    char name[20];
    char sex;
    int age;
    float score;
};

struct student stu[3]={ {11302,"Wang",'F',20,483},
    {11303,"Liu",'M',19,503},
    {11304,"Song",'M',19,471.5}};

print(struct student s) /* 打印学生姓名, 年龄, 成绩。形参: 结构体类型 */
{
    printf("%s,%d,%5.1f\n",s.name,s.age,s.score);
}

add10(struct student *ps) /* 年龄≤19, 成绩加 10 分。形参: 结构体指针类型 */
{
    if(ps->age<=19)ps->score=ps->score+10;
}

main()
{
    struct student *p;
    int i;

    for(i=0; i<3; i++)print(stu[i]);    /* 循环打印学生的记录 */
    for(i=0,p=stu; i<3; i++,p++)add10(p); /* 循环判断, 加分 */
    for(i=0,p=stu; i<3; i++,p++)print(*p); /* 循环打印学生的记录 */
}
```

或:

```
for(i=0; i<3; i++)print(stu[i]);
for(i=0; i<3; i++)add10(&stu[i]);
for(i=0; i<3; i++)print(stu[i]);
```

程序 10.3 例 10.2 程序

【说明】

- 函数 print 的形参 s 属于结构体类型, 所以实参也用结构体类型 stu[i]或*p。
- 函数 add10 的形参 ps 属于结构体指针类型, 所以实参用指针类型&stu[i]或 p。

例 10.3: 将上例中的函数 add10 改写为返回结构体类型值的函数。

```

...
struct student add10(struct student s)
{
    if(s.age<=19)s.score=s.score+10;
    return s;
}
...
main()
{

    struct student *p;
    int i;

    /*
    for(i=0; i<3; i++)print(stu[i]);
    for(i=0; i<3; i++)stu[i]=add10(stu[i]);
    for(i=0; i<3; i++)print(stu[i]);*/

    for(i=0,p=stu; i<3; i++,p++)print(*p);
    for(i=0,p=stu; i<3; i++,p++)*p=add10(*p);
    for(i=0,p=stu; i<3; i++,p++)print(*p);
}

```

程序 10.4 例 10.3 程序

【说明】

- 函数 add10 修改为返回结构体类型的函数，那么形参的传址就不必要了。
- 主函数调用 add10 时，将返回值赋值给结构体数组元素。

第四部分 指向结构体的指针变量及应用

一、指向结构体类数组的指针变量

存放结构体数组的起始地址的指针变量

1、指向结构体数组的指针变量相关运算及其含义

设有:struct{long num; char name[20];}x[3],*p=x; 则:p++->num: 先取 p 指向的结构体变量中成员 num 的值,然后使 p 指针指向下一个结构体元素; (++作用于 p 指针) ++p->num: 取 p 指向的结构体变量中成员 num 的值,并使该成员值加 1; (++作用于 num 成员) (p++)->num: 等价于 p++->num。 (++作用于 p 指针) (++p)->num: 先使 p 指针指向下一个结构体数组元素,取 p 指向的结构体元素的成员 num 的值, (++作用于 p 指针)。 (++p)->num 不同于 ++p->num。因为运算符号 () 和++的优先级别问题。 p++->num++: 得到 p 指向的结构体变量中的成员 num 的值之后,使该值加 1,再使 p 指向下一个元素; (p++)->num++:同 p++->num++

(++p)->num++:先使 p 指向下一个元素,再得到 p 指向的结构体变量中的成员 num 的值,再使该值加 1; 但 ++p->num++ 表示有错,不能编译; ++(p->n) 等价于 ++p->n。

【实例分析一】

```
#include "stdio.h"
struct ss
{
    int n;
    int *m;
} b={10},*q=&b,*p;
void main()
{
    static int d[5]={10,20,30,40,50},i=0;
    struct ss a[5]={100,&d[0],200,&d[1],300,&d[2],400,&d[3],500,&d[4]};
    p=a;
    printf("%d\t",++q->n);
    printf("%d\t",q->n++);
    printf("%d\n",q->n); [11 11 12]
    printf("%d\t",++p->n);
    printf("%d\n",p->n); [101 101]
    printf("%d\t",(++p)->n);
    printf("%d\n",p->n); [200 200]
    printf("%d\t",p++->n);
    printf("%d\n",p->n); [200 300]
    printf("%d\t",(p++)->n);
    printf("%d\n",p->n); [300 400]
    printf("\n");
    p=a;
    p->n=100;
    printf("%d\t",++p->n);
    printf("%d\n",p->n); [101 101]
    printf("%d\t",(++p)->n++);
    printf("%d\n",p->n); [200 201]
    printf("%d\t",p++->n++);
    printf("%d\n",p->n); [201 300]
    printf("%d\t",(p++)->n++);
    printf("%d\n",p->n); [300 400]
    printf("%d\t",++(p->n)); [401]
    printf("%d\t",++(*p->m)); [41]
    printf("\n");
    p=a;
    for(;i<5;i++,p++)
        printf("%d\t",p->n); [101 202 301 400]
}
```

2、指向结构体变量的指针变量与指向结构体类数组的指针变量作为函数的参数

3、结构体数组作为函数的参数

【说明】

(1)结构体指针变量不能存放结构体成员变量的地址;

(2)当结构体数组和指向结构体变量的指针变量与指向结构体数组的指针变量作为函数的参数时,形式参数的变量成员值的改变将影响实参的成员的值得;

(3)当结构体变量作为函数的参数时,形式参数的变量成员值的改变不能影响实参结构体的成员的值得;

(4)当结构体变量作为函数的参数时,形式参数的变量成员值的改变不能影响实参结构体的成员的值得。

【实例分析二】

结构体变量、结构体变量指针、结构体指针变量、结构体数组。

```
main()
{
    typedef struct
    {
        long num;
        char name[20];
        int sex;
        float score;
    }PR;
    PR a={202001,"HuangJinRong",1,98},b=a,*p=&a;
    PR
    c[3]={202001,"HuangJin",1,98,202002,"HuangRong",2,95,202003,"JinRong",1,9
0},*q=c;
    int i=0;
    printf("%ld %s %d %f\n",a.num,a.name,a.sex,a.score);
    printf("%ld %s %d %f\n",b.num,b.name,b.sex,b.score);
    printf("%ld %s %d %f\n",(*p).num,(*p).name,(*p).sex,(*p).score);
    printf("%ld %s %d %f\n",p->num,p->name,p->sex,p->score);
    printf("%p->num=ld ",p->num); printf("%p->num++=ld ",p->num++);
    printf("%++p->num \n",++p->num);printf("&a=%p&a.num=%p\n",&a,&a.num);
    for(q=c,i=0;i<3;i++,q++)
    {
        printf("%ld %s %d %f\n",c[i].num,c[i].name,c[i].sex,c[i].score);
        printf("%ld %s %d %f\n",q->num,q->name,q->sex,q->score);
    }
    q=c;
    printf("(++q)->num=%ld ",(++q)->num);
    printf("(++q)->num++=%ld ",(++q)->num++);
}
```



```

printf("(q)->num=%ld\n\n",q->num);
q=c;
printf("++q->num=%ld ",++q->num);
printf("q->num=%ld\n\n",q->num);
q=c;
printf("(q++)->num++=%ld ",(q++)->num++);
printf("(q++)->num++=%ld ",(q++)->num++);
printf("(q)->num=%ld\n\n",q->num);
q=c;
printf("q++->num++=%ld ",q++->num++);
printf("q++->num++=%ld",q++->num++);
printf("q->num=%ld\n\n",q->num);
printf("\nc=%p &c[0]=%p &c[0].num=%p\n",c,&c[0],&c[0].num);
}

```

【结果分析】

(1)结构体变量的地址与结构体变量的第一个成员的地址相同;

(2)结构体数组名表示结构体数组首地址、即第一个元素的地址也是第一个元素的第一个成员的地址;

(3)设 p 为指向结构体变量的指针变量, num 为结构体成员变量, 则: p->num: 表示 p 所指向的结构体变量的成员 num 的值。 p->num++: 表示 p 所指向的结构体变量的成员 num 的值, 引用该成员值之后再使其值加 1; ++p->num: 表示 p 所指向的结构体变量的成员 num 的值, 先使该成员值加 1 后再引用该成员值。

(4)设 p 为指向结构体数组 a 的指针变量, num 为结构体成员变量, 则: (++q)->num 与 ++q->num 等价: (++q)->num++ (q++)->num++ 与 q++->num++ 等价。

【实例分析三】

结构体变量/结构体数组/结构指针变量作为参数

```

#include "stdlib.h"
#include "string.h"
typedef struct
{
    long num; char name[20];
    int sex;
    float score;
} PR;
void prnstu(PR a,PR b[],PR *p,PR *q)
{
    int i=0;
    p->num=303009;
    strcpy(p->name,"LiRong");
    p->sex=2;
    p->score=89;
}

```

```

printf("%ld %s %d %f\n",a.num,a.name,a.sex,a.score);
printf("C->B:\n");
for(;i<3;i++)
    printf("%ld %s %d %f\n",b[i].num,b[i].name,b[i].sex,b[i].score);
b[0].num=404001;
strcpy(b[0].name,"FFFFF");
b[0].sex=1;
b[0].score=60;
printf("%ld %s %d %f\n",p->num,p->name,p->sex,p->score);
for(i=0;i<3;i++,q++)
    printf("%ld %s %d %f\n",q->num,q->name,q->sex,q->score);
}
PR news(){
    PR a={202009,"LiLan",2,92};
    return(a);
}
void list(PR a)
{
    printf("\nD.value in main before: ");
    printf("%ld %s %d %f\n",a.num,a.name,a.sex,a.score);
    a.num=9999;
    strcpy(a.name,"MMMMM");
}
main()
{
    PR a={202001,"HuangJinRong",1,98},*p=&a,d;
    PR c[3]={202001,"Huang",1,98,202002,"Jin",2,95,202003,"Rong",1,90},*q=c;
    printf("A.value in main before: ");
    printf("%ld %s %d %f\n",(*p).num,(*p).name,(*p).sex,(*p).score);
    prnstu(a,c,p,q);
    printf("A.value in main after: ");
    printf("%ld %s %d %f\n",a.num,a.name,a.sex,a.score);
    printf("\nC.value in main after: ");
    printf("%ld %s %d %f\n",c[0].num,c[0].name,c[0].sex,c[0].score);
    d=news();
    list(d);
    printf("\nD.value in main after: ");
    printf("%ld %s %d %f\n",d.num,d.name,d.sex,d.score);
}

```

【结构体变量与一维数组的区别】

一维数组名是常量,不能被再赋值,结构变量名称是一个变量,可以将同类变量赋值给它;

二、结构体变量和结构体指针变量的应用[处理单链表]

1、动态分配函数

(1)内存空间动态分配函数: void *malloc(unsigned int size) 的连续空间, 并返回其起始地址; 分配失败返回值为 (NULL), 即空指针。

(2)内存空间动态分配函数: void *calloc(unsigned int n, unsigned int size)

个长度为 size 的连续空间, 并返回其起始地址; 分配失败返回值为 (NULL), 即空指针。

(3)内存空间释放函数: void free(void *p)指向的存储空间, p 是最近一次调用 malloc () 或 calloc () 函数时的返回值; 该函数无返回值。

(4)动态链表的基本操作[建立/插入/删除/保存/修改/排序等]

2、用于建立单链表的结构体的数据结构 包括两个部分:

数据成员部分,存放数据;

地址部分,用于存放各结点的地址,时之形成链。

例如: typedef struct nd{ing no; char name[20];struct nd *next;} ND;

【实例分析四】

动态分配函数的使用与应用

```
#include "stdio.h"
#include "stdlib.h"
main()
{
    int *p,x;
    typedef struct
    {
        long num;
        char name[12];
    } ST;
    typedef union
    {
        int x;
        char na[2];
    } UN;
    ST *st,a;
    UN *un;
    p=(int *)malloc(sizeof(int));
    *p=10;
    x=*p;
    st=(ST *)malloc(sizeof(ST));
    st->num=200201001;
```

```

strcpy(st->name,"LiJiaRong");
a.num=st->num;
strcpy(a.name,st->name);
un=(UN *)malloc(sizeof(UN));
un->x=24897;
printf("x=%d\n",x);
printf("p->num=%ld a.name=%s\n",st->num,a.name);
printf("un->x=%d un->na[0]=%c un->na[1]=%c\n",un->x,un->na[0],un->na[1]);
free(p);
free(st);
free(un);
}

```

结果为：x=10 p->num=200201001 p->name=LiJiaRong un->x=24897 un->na[0]='A' un->na[1]='a'。

【实例分析五】

动态分配函数\用于建立链表的结构体类型的定义与应用

```

#include "stdio.h"
#include "stdlib.h"
#define NULL 0
typedef struct st
{
    int no;
    char na;
    struct st *next;
} ST;
ST *h=NULL,*t=NULL,*n=NULL;
ST *news(char ch)
{
    ST *r;
    r=(ST *)malloc(sizeof(ST));
    r->na=ch;
    r->no=ch%64;
    r->next=NULL;
    return(r);
}
void list(ST *q)
{
    int i=1;
    while(q->next!=NULL)
    {
        printf("%02d:%c %c",q->no,q->na,(i%13?32:'\n'));
        q=q->next;
        i++;
    }
}

```

```

    }
}
main()
{
    int i=65;
    for(;i<=91;i++)
    {
        t=news(i);
        if (h==NULL)
            h=t,n=t;
        else
        {
            n->next=t;
            n=t;
        }
    } /*End for*/
    list(h);
}

```

【实例分析六】

结构体变量/指向结构体指针变量/结构体数组/指向结构体数组的指针变量作为函数参数

```

#include "stdlib.h"
#include "string.h"
typedef struct
{
    long num;
    char name[20];
} PR;
void prn(PR a,PR b[],PR *p,PR *q)
{
    a.num=9999999;
    strcpy(p->name,"VARtoVAR");
    printf("a.num=%ld\n",a.num);
    b[0].num=2222222;
    strcpy(b[0].name,"ARRAYtoARRAY");
    p->num=5555555;
    strcpy(p->name,"POINTERtoPOINTER");
    q->num=3333333;
    strcpy(q->name,"POINTERtoARRAY");
}
PR news()
{
    PR a={202009,"variableNEW"};
    return(a);
}

```

```

}
void list(PR a)
{
    printf("%ld %s\n",a.num,a.name);
}
main()
{
    PR a={202001,"variableA"},b,*p;
    PR c[3]={2020001,"arrayC0",2020002,"arrayC1"},*q=c;
    PR d[2]={4040001,"arrayD0",4040002,"arrayD1"};
    b=news();
    p=&b;
    list(a);list(b);list(c[0]);list(d[0]);
    prn(a,d,p,q);
    list(a);list(b);list(c[0]);list(d[0]);
}

```

【结果分析】

(1)结构体变量作为参数[实参/形参]时,形参变量成员值的改变不影响对应的实参变量成员的值;

(2)结构体指针变量或结构体数组[名]或指向结构体数组的指针变量作为参数[实参/形参]时,形参变量成员值的改变将影响对应的实参变量成员的值。

第五部分 共用体及枚举类型

一、共用体

共用体: 将不同类型的数据项存放于同一段内存单元的一种构造数据类型。

与结构类似,在共用体内可以定义多种不同数据类型的成员;区别是,在共用体类型变量所有成员共用一块内存单元。(虽然每个成员都可以被赋值,但只有最后一次赋予的成员值能够保存且有意义,前面赋予的成员值被后面赋予的成员值所覆盖)。

1、共用体类型、共用体类型变量的定义

(1)共用体类型定义的一般形式:

```

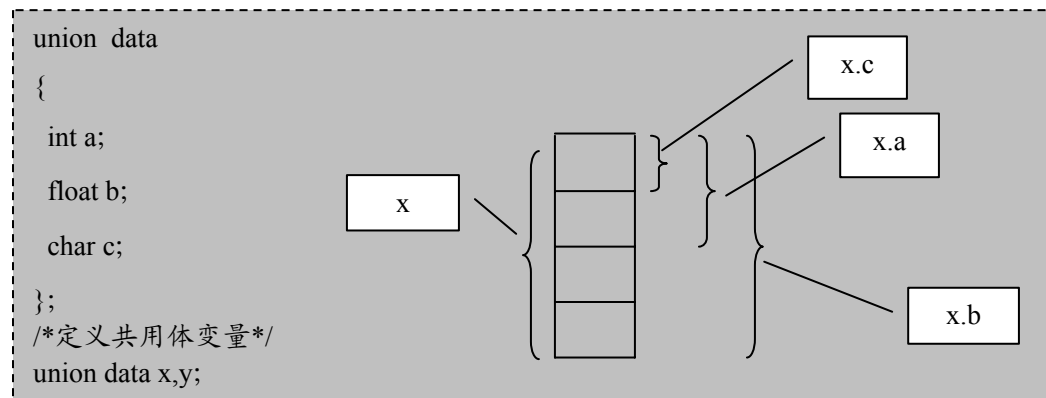
union 共用体名
{
    类型 1 成员 1;
    类型 2 成员 2;
    .....
    类型 n 成员 n;
}

```

```
};
```

2、共用体类型变量的定义，方法同结构体变量的定义（三种形式，同时，前后，匿名）

例如：/*定义共用体类型 data*/



程序 10.5 共用体类型定义示例

3、共用体变量的引用

对共用体变量的赋值，使用都是对变量的成员进行的，共用体变量的成员表示为：

共用体变量名.成员名

使用共用体类型数据时应注意共用体数据的特点：

(1)同一内存段可以用来存放不同类型的成员，但是每一瞬时只能存放其中的一种（也只有一种有意义）。

(2)共用体变量中有意义的成员是最后一次存放的成员。

例如：在 `x.a=1; x.b=3.6; x.c='H'` 语句后；当前只有 `x.c` 有意义（`x.a,x.b` 也可以访问，但没有实际意义）。

(3)共用体变量的地址和它的成员的地址都是同一地址。即，`&x.a=&x.b=&x.c=&x`。

(4)除整体赋值外，不能对共用体变量进行赋值，也不能企图引用共用体变量来得到成员的值。不能在定义共用体变量时对共用体变量进行初始化（系统不清楚是为哪个成员赋初值）；

(5)可以将共用体变量作为函数参数，函数也可以返回共用体，共用体指针。

(6)共用体，结构体可以相互嵌套。

例 11-10：学校的人员数据管理，教师的数据包括：编号、姓名、性别、职务。学生的数据包括：编号、姓名、性别、班号。如果将两种数据放在同一个表格中，那么有一栏，对于教师登记教师的“职务”，对于学生则登记学生的“班号”（对于同一人员不可能同时出现）。写出类型定义，并编写输入信息，输出信息的程序代码。(P195)

解：

```
num:11301
name:zhang
sex:f
job(s-student,t-teacher):s
class:0301
num:99101
name:liu
sex:m
job(s-student,t-teacher):t
position:prof.
num:11201
name:li
sex:m
job(s-student,t-teacher):s
class:0201
num:-1
num  name  sex  job  category:
11301  zhang  f    s    301
99101  liu    m    t    prof.
11201  li     m    s    201
```

程序 10.6 共用体定义

```
struct person /* 结构体类型定义 */
{
    long num; char name[20]; char sex; char job; /* 人员标志: s-学生, t-教师 */
    union /* 匿名共用体类型定义, 并定义共用体变量 category 作为外层结构体的成员 */
    {
        int class; position[20];
    } category;
};
```

程序 10.7 结构体类型定义


```

int input(struct person person[]) /* 输入人员信息，返回人员数 */
{
    int i;

    for(i=0; i<20; i++) /* 循环输入，如果满 20 个或遇到编号-1 结束循环 */
    {
        printf("num:"); scanf("%ld",&person[i].num); fflush();/* 清除缓冲区 */
        if(person[i].num==-1)break;

        printf("name:"); scanf("%s",&person[i].name); fflush();
        printf("sex:"); scanf("%c",&person[i].sex); fflush();
        printf("job(s-student,t-teacher):"); scanf("%c",&person[i].job); fflush();

        if(toupper(person[i].job)=='S') /* 如果是学生的信息 */
        {
            printf("class:"); scanf("%d",&person[i].category.class); fflush();
        }
        else /* 否则是教师的信息 */
        {
            printf("position:");scanf("%s",&person[i].category.position);fflush();
        }
    }
    return i; /* 返回输入人员的数 */
}

void output(struct person person[],int n) /* 输出人员信息 */
{
    int i;

    printf("num\tname\tsex\tjob\tcategory:\n"); /* 打印表头 */
    for(i=0; i<n; i++) /* 循环输出 */
    {
        if(toupper(person[i].job)=='S')
            printf("%ld\t%s\t%c\t%c\t%d\n",person[i].num,person[i].name,
                person[i].sex,person[i].job,person[i].category.class);
        else
            printf("%ld\t%s\t%c\t%c\t%s\n",person[i].num,person[i].name,
                person[i].sex,person[i].job,person[i].category.position);
    }
}

void main()
{
    int n;
    struct person person[20]; /* 人员情况数组 */
    n=input(person); /* 输入数据 */
    output(person,n); /* 输出数据 */
}

```

程序 10.8 例 10.4 程序

二、枚举类型

枚举类型：只能取事先定义值的数据类型是枚举类型。

1、枚举类型定义

enum 枚举类型名 {枚举元素（或：枚举常量）列表};

枚举变量定义（类似结构体变量定义3种形式）

(1)定义枚举类型的同时定义变量：enum 枚举类型名 {枚举常量列表}枚举变量列表;

(2)先定义类型后定义变量：enum 枚举类型名 枚举变量列表;

(3)匿名枚举类型：enum {枚举常量列表}枚举变量列表。

例如：

```
enum weekday {sun,mon,tue,wed,thu,fri,sat};
```

```
/* 定义枚举类型 enum weekday，取值范围：sun，mon...sat。*/
```

```
enum weekday week1,week2;
```

```
/* 定义 enum weekday 枚举类型的变量 week1,week2，其取值范围：  
sun,mon,...,sat。*/
```

```
week1=wed; week2=fri;
```

```
/* 可以用枚举常量给枚举变量赋值 */
```

【说明】

(1)enum 是标识枚举类型的关键词，定义枚举类型时应当用 enum 开头。

(2)枚举元素（枚举常量）由程序设计者自己指定，命名规则同标识符。这些名字是符号，可以提高程序的可读性。

(3)枚举元素在编译时，按定义时的排列顺序取值 0，1，2...。（类似整型常数）。

(4)枚举元素是常量，不是变量（看似变量，实为常量），可以将枚举元素赋值给枚举变量。但是不能给枚举常量赋值。在定义枚举类型时可以给这些枚举常量指定整型常数值（未指定值的枚举常量的值是前一个枚举常量的值+1）。

例如：

```
enum weekday {sun=7,mon=1,tue,wed,thu,fri,sat};
```

(5)枚举常量不是字符串。

(6)枚举变量，常量一般可以参与整数可以参与的运算。如算术运/关系/赋值等运算。

例如：不要希望 `week1=sun; printf("%s",week1);` 能打印出 “sun,...”，可以用下面语句检查输出：`if(week1==sun)printf("sun");`。

三、用 typedef 定义类型

格式：typedef 类型定义 类型名；

【说明】

typedef 是定义了一个新的类型的名字，没有建立新的数据类型，它是已有类型的别名。使用类型定义，可以增加程序可读性，简化书写。

1、使用 typedef 关键词可以定义一种新的类型名代替已有的类型名。

例如：

```
typedef int INTEGER; typedef float REAL;
INTEGER i,j; REAL a,b;
```

2、类型定义的典型应用

(1)定义一种新数据类型，作简单的名字替换。

例如：

```
typedef unsigned int UINT; /* 定义 UINT 是无符号整型类型 */
UINT u1; /* 定义 UINT 类型（无符号整型）变量 u1 */
```

(2)简化数据类型的书写。

```
typedef struct
{
    int month; int day; int year;
}DATE; /* 定义 DATE 是一种结构体类型 */
DATE birthday,*p,d[7];
/* 定义 DATA（结构体类型）类型的变量，指针，数组：birthday,p,d */
```

【注意】

用 typedef 定义的结构体类型不需要 struct 关键词，简洁。

(1)定义数组类型

```
typedef int NUM[100]; /* 定义 NUM 是 100 数整型数组类型（存放 100 个整数） */
```

```
NUM n; /* 定义 NUM 类型（100 数整型数组）的变量 n */
```

(2)定义指针类型

```
typedef char* STRING; /* 定义 STRING 是字符指针类型 */  
STRING p; /* 定义 STRING 类型（字符指针类型）的变量 p */
```

第十一章

位运算

【基本要求】

通过本单元的学习，使学生了解位运算符的基本知识，熟悉为运算符计位运算，掌握位运算的应用，掌握位段及其应用。

【教学重点】

位运算符与位运算及其应用，位段及其应用

【本章结构】

- 1、位运算与位运算
 - “按位与”运算符(&)
 - “按位或”运算符(|)
 - “异或”运算符(^)
 - “取反”运算符(~)
 - 左移运算符(<<)
 - 右移运算符(>>)
 - 位运算赋值运算符
 - 不同长度的数据进行位运算
- 2、位运算举例
- 3、位段

计算机内部，数据的存储、运算都是以二进制形式进行的，1 个字节=8 个二进制位。位运算就是针对二进制位的运算。

位运算的操作对象一般是整型或字符型。

位运算是 C 语言的低级语言特性，广泛应用于对底层硬件，外围设备的状态检测和控制。

1、左移 “<<” 运算符

左移运算符 “<<” 功能：将一个数的各个二进制位全部向左平移若干位（左边移出的部分忽略，右边补 0）。每左移 1 位，相当于乘 2，左移 n 位相当于乘 2^n 。（数字可以展开为 2 二进制，按权展开，数字乘 2，幂升 2，相当于向左移动了 1 位）

例 11.1:

unsigned char a=26; /* (26)₁₀=(0001,1010)₂=(1A)₁₆ */

a=a<<2; /* (0110,1000)₂=(68)₁₆=(104)₁₀ */

2、右移“>>”运算符

右移运算符“>>”功能：将一个数的各个二进制位全部向右平移若干位（右边移出的部分忽略，右边对无符号数补0，有符号数补符号位）。每右移1位，相当于除2，左移n位相当于乘2ⁿ。

例 11.2:

unsigned char a=0x9A; /* (9A)₁₆=(154)₁₀=(1001,1010)₂ */

a=a>>2; /* (0010,0110)₂=(26)₁₆=(38)₁₀ */

3、按位取反“~”运算符

按位取反“~”是单目运算符，对一个二进制数的每一位都取反。0->1,1->0。

例 11.3: a=00011010(1A), ~a=11100101(E5)。

4、按位与“&”运算符

将其两边数据对应的二进制位按位进行“与”运算。二者全为1结果为：1，否则为：0。

例 11.4:

a =10111010(0xBA)

b =01101110(0x6E)

a&b=00101010(0x2A)

【结论】“与1位与”为1，那么该位为1；“与1位与”为0，那么该位为0。“与1位与”可用于检测某个位是1还是0。

例 11.5: 将一个十进制数转化为二进制数。

【分析】C语言标准输出函数只能将一个整数以10，8，16进制输出（使用%x，%o，%d），但是C语言没有2进制输出格式。人工转换的方法是：设置一个屏蔽字，其中只有一个位为1，其余为0，为1的位为测试位置。将此屏蔽字与被转换数进行“位与”运算，根据运算结果判断被测试的位是1还是0。循环测试（一个整数2字节，16位，测试16次，从最高位开始测试，每次测试后屏蔽字右移1位以便测试下一个位）并输出的测试结果就是整数对应的二进制数。

解：如程序 11.1 所示

```
main()
{
    int i,bit;          /* 定义循环变量 i 和位 1/0 标志变量 bit */
    unsigned int n,mask; /* 定义欲转换的整数 n 和屏蔽字变量 mask */

    mask=0x8000; /* 初始屏蔽字 1000, 0000, 0000, 0000, 从左边最高位开始检查 */
    printf("Enter a integer:");scanf("%d",&n); /* 输入要转换的整数 */

    printf("binary of %u is:",n);
    for(i=0; i<16; i++) /* 循环检查 16 位, 并输出结果 */
    {
        if(i%4==0&& i!=0)printf(","); /* 习惯上 2 进制每 4 位用 “,” 分隔以便查看。 */
        bit=(n&mask)?1:0; /* n&mask 非 0, 该位为 1; 否则该位为 0 */
        printf("%ld",bit); /* 输出 1 或 0 */
        mask=mask>>1; /* 右移 1 位得到下一个屏蔽字 */
    }
    printf("\n");
}

结果:
Enter a integer:56
binary of 56 is:0000,0000,0011,1000
```

程序 11.1 按位与运算

5、按位位或 “|” 运算符

将其两边数据对应的二进制位按位进行“或”运算。二者只要有一个为 1 结果为 1；否则为 0。（两者都为 0 时为 0）。

【结论】与 0 “位或”为 1，那么该位为 1；与 0 “位或”为 0，那么该位为 0-就是说任何位“与 0 位或”还是等于这一位（保持不变）。

例 11.6：循环移位的实现。

解：

假设对无符号数循环右移 n 次。（如图 11.1 所示）

(1) 循环右移 n 次后，n 个高位构成：原来的 n 个低位左移 16-n 位。（试着移一下）

(2) 循环右移 n 次后, $16-n$ 个低位构成: 原来的 $16-n$ 个高位右移 n 位。(容易理解)

(3) 因为是无符号整数, 移出的都丢弃, 缺位都补 0, 所以 n 个低位, $16-n$ 个高位的移动等价于无符号数本身的移动。移动后的结果保存在两个变量 hi, low 中。最后合成一个数。

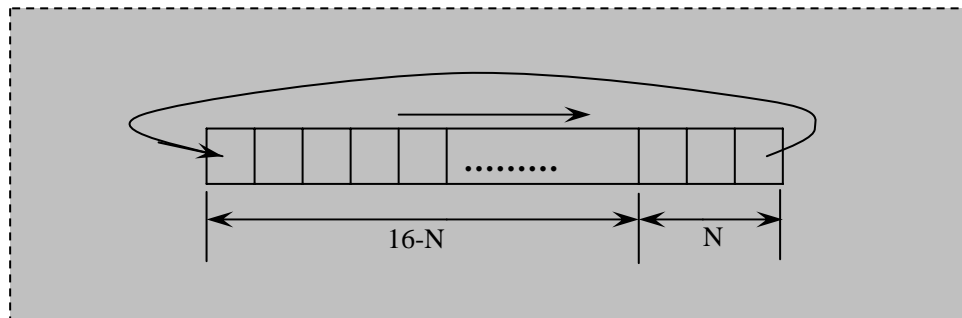


图 11.1 循环移位示意图

```
main()
{
    unsigned x, hi, lo, reslt;
    int n;

    printf("Enter x(0x..):"); scanf("%x", &x);
    printf("Enter n:"); scanf("%d", &n);

    hi=x<<(16-n); /* 左移右补 0, 得到高位 (n 个低位=>n 个高位) */
    lo=x>>n;      /* 右移左补 0, 得到低位 16-n (16-n 个高位=>16-n 个低位) */
    reslt=hi|lo;  /* 合并高/低位 */

    printf("x=0x%x, result=0x%x\n", x, reslt);
}

结果:
Enter x(0x.):56
Enter n:3
x=0x56,result=0xc00a
```

程序 11.2 循环移位运算

6、按位异或 “^” 运算符

将其两边数据对应的二进制位按位进行“异或”运算。若二者相同, 结果为: 0, 若二者不同 (相异), 结果为: 1。

【结论】任何位“与 1 异或”, 等价于对该位取反。

第十二章

文件

【基本要求】

通过本单元的学习，使学生了解文件概念，学会文件指针，掌握打开和关闭文件的操作，并能够熟练应用常用文件操作函数。

【教学重点】

文件数据类型与文件类指针变量，缓冲文件系统中文件的基本操作(建立/打开/关闭/读/写)，非缓冲文件系统中文件的基本操作(建立/打开/关闭/读/写)，文件的定位与出错检查，文件的应用。

【本章结构】

- 1、C 文件概述
- 2、文件类型指针
- 3、文件的各类操作
 - 文件的打开和关闭
 - 文件的读写
 - 文件的定位
 - 出错的检测
- 4、文件输入输出小结

一、文件概述

1、文件

存储在外部介质上一组相关数据的集合。

例如，程序文件就是程序代码的集合；数据文件是数据的集合。

2、文件名：操作系统以文件为单位对数据进行管理，每个文件有一个名称，文件名是文件的标识，操作系统通过文件名访问文件。

例如，通过文件名查找，打开文件，然后读取或写入数据。

3、磁盘文件、设备文件

(1)磁盘文件：文件一般保存在磁介质（如软盘、硬盘）上，所以称为磁盘文件。

(2)设备文件：操作系统还经常将与主机相连接的 I/O 设备（如键盘-输入文件、显示器、打印机-输出文件）也看作为文件，即设备文件。

很多磁盘文件的概念、操作，对设备文件也同样有意义，有效。

4、ASCII 文件、二进制文件

根据文件的组织形式，文件可以分为 ASCII 文件和二进制文件。

(1)ASCII 文件（文本文件）：每个字节存放一个 ASCII 码，代表一个字符。ASCII 文件可以阅读，可以打印，但是它与内存数据交换时需要转换。

(2)二进制文件：将内存中的数据按照其在内存中的存储形式原样输出、并保存在文件中。二进制文件占用空间少，内存数据和磁盘数据交换时无须转换，但是二进制文件不可阅读、打印。

例如：同样的整数 10000，如果保存在文本文件中，就可以用 notepad,edit 等文本编辑器阅读，也可以在 dos 下用 type 显示，它占用 5 个字节；如果保存在二进制文件中，不能阅读，但是我们知道一个整数在内存中用补码表示并占用 2 个字节，所以如果保存在二进制文件中就占用 2 个字节。

文本文件/二进制文件不是用后缀来确定的，而是以内容来确定的，但是文件后缀往往隐含其类别，如*.txt 代表文本文件，*.doc，*.bmp，*.exe 二进制文件。

5、缓冲文件系统、非缓冲文件系统：

(1)缓冲文件系统：系统自动地在内存中为每个正在使用的文件开辟一个缓冲区。在从磁盘读数据时，一次从磁盘文件将一些数据输入到内存缓冲区（充满缓冲区），然后再从缓冲区逐个将数据送给接受变量；向磁盘文件输出数据时，先将数据送到内存缓冲区，装满缓冲区后才一起输出到磁盘。减少对磁盘的实际访问（读/写）次数。ANSI C 只采用缓冲文件系统。

(2)非缓冲文件系统：不由系统自动设置缓冲区，而由用户根据需要设置。

C 语言中，没有输入输出语句，对文件的读写都是用库函数实现的。

二、文件类型指针

1、文件类型（结构体）- FILE 类型

FILE 类型是一种结构体类型，在 stdio.h 中定义，用于存放文件的当前的有关信息。

程序使用一个文件，系统就为此文件开辟一个 FILE 类型变量。程序使用几个文件，系统就开辟几个 FILE 类型变量，存放各个文件的相关信息。

```
typedef struct {
    short      level;          /* fill/empty level of buffer */
    unsigned   flags;          /* 文件状态标志 */
    char       fd;              /* 文件描述符 */
    unsigned char hold;         /* Ungetc char if no buffer */
    short      bsize;           /* 缓冲区大小 */
    unsigned char *buffer;      /* 数据传输缓冲区 */
    unsigned char *curp;        /* 当前激活指针 */
    unsigned   istemp;          /* 临时文件指示器 */
    short      token;           /* 用于合法性校合 */
} FILE;
```

图 12.1 文件指针类型

2、文件指针变量（文件指针）

通常对 FILE 结构体的访问是通过 FILE 类型指针变量（简称：文件指针）完成，文件指针变量指向文件类型变量，简单地说，文件指针指向文件。

事实上只需要使用文件指针完成文件的操作，根本不必关心文件类型变量的内容。在打开一个文件后，系统开辟一个文件变量并返回此文件的文件指针；将此文件指针保存在一个文件指针变量中，以后所有对文件的操作都通过此文件指针变量完成；直到关闭文件，文件指针指向的文件类型变量释放。

例如：

定义一个指向文件的指针变量 fp。 /*

fp=fopen(“mydata.txt”,...);

/* 打开文件时，系统开辟一个文件变量，并返回文件指针，将此指针赋值(保存)给文件指针变量 fp */

.....（文件操作函数,会引用文件指针 fp）

fclose(fp); /* 关闭文件，释放文件指针 fp 指向的文件变量 */

三、文件的打开与关闭

对文件的操作的步骤：先打开，后读写，最后关闭。

1、文件的打开（fopen 函数）

(1)文件打开后才能进行操作，文件打开通过调用 fopen 函数实现。

调用 fopen 的格式是：

open(文件名, 打开方式 open(文件名, 打开方式或使用方式);

【注意】一定将函数返回的文件指针赋值给“文件指针变量”。

例如:

```
FILE *fp;
```

```
fp=fopen( "d:\\a1.txt" , "r" );
```

【说明】

(1)打开 d:盘根目录下文件名为 a1.txt 的文件, 打开方式“r”表示只读。

(2)fopen 函数返回指向 d:\\a1.txt 的文件指针, 然后赋值给 fp, fp->此文件, 即 fp 与此文件关联。

(3)关于文件名要注意: 文件名包含文件名.扩展名; 路径要用“\\”表示。

(4)关于打开方式, 简单地看, 文件打开方式包含下面几类表示打开方式的关键词, 不同类的可以组合。

- “r, w, a” : 读、写、添加
- “b, t” : 二进制, 文本文件。默认为文本方式, 即没有 b 就是以文本方式打开文件。
- “+” : 可读写

2、文件打开方式(使用方式)的说明

(1)文件打开一定要检查 fopen 函数的返回值。因为有可能文件不能正常打开。不能正常打开时 fopen 函数返回 NULL。

可以用下面的形式检查:

```
if((fp=fopen(...))==NULL){ printf("error open file\n"); exit(1); }
```

(2)“r”方式: 只能从文件读入数据而不能向文件写入数据。该方式要求欲打开的文件已经存在。

(3)“w”方式: 只能向文件写入数据而不能从文件读入数据。如果文件不存在, 创建文件, 如果文件存在, 原来文件被删除, 然后重新创建文件(相当覆盖原来文件)。

(4)“a”方式: 在文件末尾添加数据, 而不删除原来文件。该方式要求欲打开的文件已经存在。

(5)“+” (“r+,w+,a+”) : 均为可读、可写。但是“r+”, “a+”要求文件已经存在, “w+”无此要求; “r+”打开文件时文件指针指向文件开头, “a+”打开文件时文件指针指向文件末尾。

(6) “b、t”：以二进制或文本方式打开文件。默认是文本方式，t可以省略。读文本文件时，将“回车”/“换行”转换为一个“换行”；写文本文件时，将“换行”转换为“回车/换行”。

(7)程序开始运行时，系统自动打开三个标准文件：标准输入，标准输出，标准出错输出。一般这三个文件对应于终端（键盘、显示器）。这三个文件不需要手工打开，就可以使用。标准文件：标准输入，标准输出，标准出错输出对应的文件指针是 `stdin, stdout, stderr`。

3、文件的关闭（`fclose` 函数）

文件使用完毕后必须关闭，以避免数据丢失。

格式：`fclose(文件指针)`;

四、文件的读写

常用的文件读写函数：

- 字符读写函数：`fgetc, fputc`
- 字符串读写函数：`fgets, fputs`
- 格式化读写函数：`fscanf, fprintf`
- 数据块读写函数：`fread, fwrite`

【说明】

(1)这些函数都以 `f (file)` 开头。

(2)这些函数都要使用打开文件时获得的文件指针。

(3)前三类函数与标准 I/O 函数使用基本相同。

(4)一般，对于文本文件使用“顺序访问”方式操作，对于二进制文件使用“随机访问”方式操作。

1、字符读写函数

(1) 写一个字符到磁盘文件：

格式：`fputc(ch, fp)`

功能：将字符 `ch`（可以是字符表达式，字符常量、变量等）写入 `fp` 所指向的文件。

返回：输出成功返回值-输出的字符 `ch`；输出失败-返回 `EOF (-1)`。

其它说明：每次写入一个字符，文件位置指针自动指向下一个字节。

例 12.1：从键盘输入一行字符，写入到文本文件 `string.txt` 中。如程序 12.1 所示。

```

#include <stdio.h>
main()
{
    FILE *fp;
    char ch;
    if((fp=fopen("string.txt","w"))==NULL) /* 打开文件 string.txt(写) */
    {
        printf("can't open file\n");exit(1);
    }
    do /* 不断从键盘读字符并写入文件，直到遇到换行符 */
    {
        ch=getchar(); /* 从键盘读取字符 */
        fputc(ch,fp); /* 将字符写入文件 */
    }while(ch!='\n');
    fclose(fp); /* 关闭文件 */
}
结果：
E:\>12-1<CR>
I am a student<CR>

E:\>type string.txt<CR>
I am a student

/* 控制台读字符可以用 getche(), 无须缓冲, 此时循环控制应当是 ch!='\r' */

```

程序 12.1 字符读入函数

(2)从磁盘文件读一个字符

格式: `ch=fgetc(fp)`

功能: 从 `fp` 所指向的文件读一个字符, 字符由函数返回。返回的字符可以赋值给 `ch`, 也可以直接参与表达式运算。

返回: 输入成功返回值-输入的字符; 遇到文件结束-返回 `EOF (-1)`。

其它说明:

- 每次读入一个字符, 文件位置指针自动指向下一个字节。
- 文本文件的内部全部是 ASCII 字符, 其值不可能是 `EOF(-1)`, 所以可以使用 `EOF (-1)` 确定文件结束; 但是对于二进制文件不能这样做, 因为可能在文件中间某个字节的值恰好等于-1, 如果此时使用-1 判断文件结束是不恰当的。为了解决这个问题, ANSI C 提供了 `fgetc(fp)` 函数判断文件是否真正结束。
- `fgetc` 函数既适合文本文件, 也适合二进制文件文件结束的判断。

例 12.2: 将磁盘上一个文本文件的内容复制到另一个文件中。如程序 12.2 所示。

```
#include <stdio.h>
main()
{
    FILE *fp_in,*fp_out;
    char infile[20],outfile[20];
    printf("Enter the infile name:");
    scanf("%s",infile);          /* 输入欲拷贝源文件的文件名 */
    printf("Enter the outfile name:");
    scanf("%s",outfile);         /* 输入拷贝目标文件的文件名 */
    if((fp_in=fopen(infile,"r"))==NULL) /* 打开源文件 */
    {
        printf("can't open file:%s",infile); exit(1);
    }
    if((fp_out=fopen(outfile,"w"))==NULL) /* 打开目标文件 */
    {
        printf("can't open file:%s",outfile); exit(1);
    }
    while(!feof(fp_in))          /* 若源文件未结束 */
    {
        fputc(fgetc(fp_in),fp_out); /* 从源文件读一个字符，写入目标文件 */
    }

    fclose(fp_in); /* 关闭源、目标文件 */
    fclose(fp_out);
}
结果:
Enter the infile name:string.txt
Enter the outfile name:string1.txt
本例使用了 feof()判断文件结束。
```

程序 12.2 文件复制操作

将源文件、目标文件的文件名通过 main 函数的参数传递给程序，这样可以在命令行直接将文件名输入。如程序 12.3 所示

```

#include <stdio.h>
void main(int argc,char *argv[])
{
    FILE *fp_in,*fp_out;
    if(argc<3)
    {
        printf("missing file name\n");  exit(1);
    }
    if((fp_in=fopen(argv[1],"r"))==NULL)
    {
        printf("can't open file:%s",argv[1]);  exit(2);
    }
    if((fp_out=fopen(argv[2],"w"))==NULL)
    {
        printf("can't open file:%s",argv[2]);  exit(3);
    }
    while(!feof(fp_in))
    {
        fputc(fgetc(fp_in),fp_out);
    }
    fclose(fp_in);
    fclose(fp_out);
}

```

程序 12.3 文件复制操作 2

2、字符串读写函数

(1)从磁盘文件读一个字符串

格式: char *fgets(char *str,int n,FILE *fp)

功能: 从 fp 所指向的文件读 n-1 个字符, 并将这些字符放到以 str 为起始地址的单元中。如果在读入 n-1 个字符结束前遇到换行符或 EOF, 读入结束。字符串读入后最后加一个 '\0' 字符。

返回: 输入成功返回值-输入串的首地址; 遇到文件结束或出错-返回 NULL。

例 12.3: 编制一个将文本文件中全部信息显示到屏幕的程序(类似于 dos 的 type 命令)。使用 fgets 的例子。如程序 12.4 所示。


```

#include <stdio.h>
void main(int argc,char *argv[])
{
    FILE *fp;
    char string[81]; /* 最多保存 80 个字符, 外加一个字符串结束标志 */
    if(argc!=2||(fp=fopen(argv[1],"r"))==NULL) /* 打开文件 */
    {
        printf("can't open file"); exit(1);
    }
    while(fgets(string,81,fp)!=NULL)
        /* 如果未读到文件末尾(E0F), 函数不会返回 NULL, 继续循环 (执行循环体) */
        /* 从文件一次读 80 个字符, 遇换行或 E0F, 提前带回字符串 */
        printf("%s",string); /* 打印串 */
    fclose(fp); /* 关闭文件 */
}

```

程序 12.4 字符串读写

```

#include <stdio.h>
void main()
{
    FILE *fp;
    char s[81];

    if((fp=fopen("string.txt","a"))==NULL) /*打开文件*/
    {
        printf("can't open file\n"); exit(1);
    }
    while(strlen(gets(s))>0)/*从键盘读入一个字符串, 遇到空行 (strlen=0) 结束*/
    {
        fputs(s,fp); /*将字符串写进文件*/
        fputs("\n",fp); /*补一个换行符*/
    }
    fclose(fp); /*关闭文件*/
}

```

程序 12.5 字符串写入磁盘操作

(2)写一个字符串到磁盘文件

格式: fputs(char *str,FILE *fp)

功能: 向 fp 所指向的文件写入以 str 为首地址的字符串。

返回：输入成功返回值-0；出错-返回非 0 值。

例 12.4：在文本文件 string.txt 末尾添加若干行字符。使用 fputs 的例子。如程序 12.5 所示。

3、格式化读写函数

格式化文件读写函数 fprintf,fscanf 与函数 printf,scanf 作用基本相同，区别在于 fprintf,fscanf 读写的对象是磁盘文件，printf,scanf 读写的对象是终端。

格式：

fprintf(fp,格式字符串,输出表列);

fscanf(fp,格式字符串,输入表列);

其中：fp 是文件指针。

4、数据块读写函数（一般用于二进制文件读写）。（重点）

从文件（特别是二进制文件）读写一块数据（如一个数组元素，一个结构体变量的数据-记录）使用数据块读写函数非常方便。

数据块读写函数的调用形式为：

int fread(void *buffer,int size,int count,FILE *fp);

int fwrite(void *buffer,int size,int count,FILE *fp);

其中：

(1)buffer 是指针，对 fread 用于存放读入数据的首地址；对 fwrite 是要输出数据的首地址。

(2)size 是一个数据块的字节数(每块大小)，count 是要读写的数据块块数。

(3)fp 文件指针

(4)fread、fwrite 返回读取/写入的数据块块数。（正常情况=count）

(5)以数据块方式读写，文件通常以二进制方式打开。

例如：

float f[2];

FILE *fp=fopen("...", "r");

fread(f,4,2,fp); /* 或 fread(f,sizeof(float),2,fp); */

例 12.5: 从键盘输入一批学生的数据, 然后把它们转存到磁盘文件 stud.dat 中。如程序 12.6 所示。

解:

```
#include <stdio.h>
#include <stdlib.h>
#include <ctype.h>

struct student
{
    int num;
    char name[20];
    char sex;
    int age;
    float score;
}; /* 共 5 个成员, 占用 29 bytes */

main()
{
    struct student stud;
    char numstr[20],ch;
    /* numstr-临时字符串, 保存学号/年龄/成绩, 然后转换为相应类型; ch-Y/N */
    FILE *fp;
    if((fp=fopen("stud.dat","wb"))==NULL) /* 以二进制、写方式打开文件 */
    {
        printf("can't open file stud.dat\n");
        exit(1);
    }
    do
    {
        printf("enter number:"); gets(numstr); stud.num=atoi(numstr);
        printf("enter name:"); gets(stud.name);
        printf("enter sex:"); stud.sex=getchar(); getchar();
        printf("enter age:"); gets(numstr); stud.age=atoi(numstr);
        printf("enter score:"); gets(numstr); stud.score=atof(numstr);
        /* 每次将一个准备好的结构体变量的所有内容写入文件 (写一个记录) */
        fwrite(&stud,sizeof(struct student),1,fp);
        printf("have another student record(y/n)?"); ch=getchar(); getchar();
    }while(toupper(ch)=='Y'); /* 循环读数据/写记录 */
    fclose(fp); /* 关闭文件 */
}
```

程序 12.6 字符串转存磁盘文件操作

【说明】

(1)空读：在输入字符，并按回车后，实际缓冲中有两个字符（如 ‘f/m’ 和 ‘\n’ ），只要前面有意义的字符（ ‘f/m’ ）。可以用“空读”略过 ‘\n’ 。

(2)什么情况要空读？如果后面的读取键盘是读取数字（整数/浮点数），不必空读；如果后面的读取键盘是读字符或字符串，应当“空读”。

(3)思考：如果按照教材 p212 输入两个记录的数据，那么最后产生的文件长度是多少？（58 bytes）

备注：C 语言即使写文本文件，关闭时，也不自动加文件结束符。

五、文件的定位（重点）

对文件的读写可以顺序读写，也可以随机读写。

(1)文件顺序读写：从文件的开头开始，依次读写数据。（从文件开头读写直到文件尾部）

(2)文件随机读写（文件定位读写）：从文件的指定位置读写数据。

(3)文件位置指针：在文件的读写过程中，文件位置指针指出了文件的当前读写位置（实际上是：下一步读写位置），每次读写后，文件位置指针自动更新指向新的读写位置（实际上是：下一步读写位置）。 **注意区分**：文件位置指针，文件指针。

可以通过文件位置指针函数，实现文件的定位读写。文件位置指针函数有：

- rewind 重返文件头函数
- fseek 位置指针移动函数
- ftell 获取当前位置指针函数

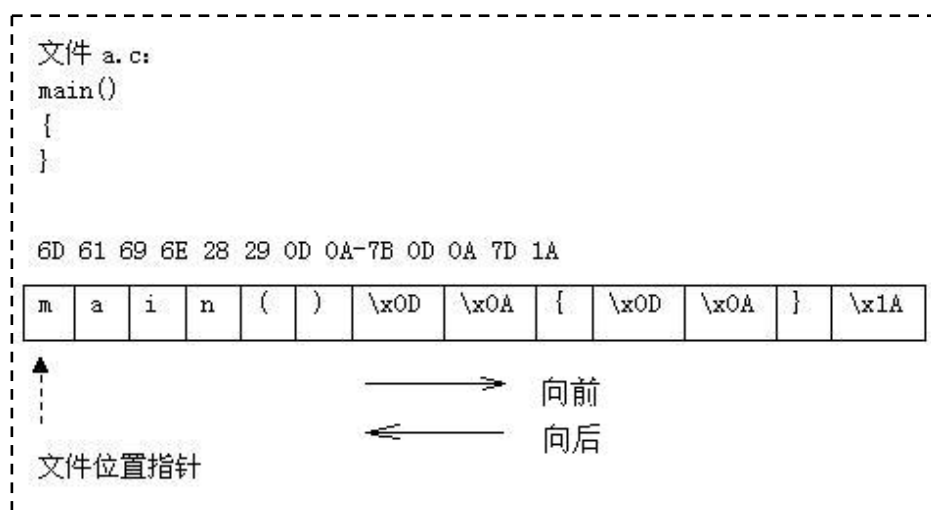


图 12.2 文件定位

1、rewind-重返文件头函数

功能：使文件位置指针重返文件的开头。

例 12.6: 有一个文本文件, 第一次使它显示在屏幕上, 第二次把它复制到另外一个文件中。如程序 12.7 所示。

解:

```
#include <stdio.h>
main()
{
    FILE *fp1,*fp2;
    fp1=fopen("string.txt","r"); /* 打开文件 */
    fp2=fopen("string2.txt","w");
    /* 从文件 string.txt 读出, 写向屏幕 */
    while(!feof(fp1))putchar(getc(fp1));
    /* 重返文件头 */
    rewind(fp1);
    /* 从文件 string.txt 读出, 写向文件 string2.txt */
    while(!feof(fp1))putc(getc(fp1),fp2);
    fcloseall(); /* 关闭文件 */
}
```

程序 12.7 文件位置指针

2、fseek-位置指针移动函数

功能: 移动文件读写位置指针, 以便文件的随机读写。

格式: `fseek(FILE *fp,long offset,int whence);`

参数:

fp-文件指针。

whence-计算起始点 (计算基准)。

计算基准可以是下面符号常量:

表格 12.1 计算标准

符号常量	符号常量的值	含义
SEEK_SET	0	从文件开头计算
SEEK_CUR	1	从文件指针当前位置计算
SEEK_END	2	从文件末尾计算

offset-偏移量 (单位: 字节): 从计算起始点开始再偏移 offset, 得到新的文件指针位置。offset 为正, 向后偏移; offset 为负, 向前偏移。

例如:

```
fseek(fp,100,0);
```

/* 将位置指针移动到: 从文件开头计算, 偏移量为 100 个字节的位置 */

```
fseek(fp,50,1);
```

/* 将位置指针移动到: 从当前位置计算, 偏移量为 50 个字节的位置 */

/* 向前移动 */

```
fseek(fp,-30,1);
```

```
/* 将位置指针移动到：从当前位置计算，偏移量为-30 个字节的位置 */
```

```
/* 向后移动 */
```

```
fseek(fp,-10,2);
```

```
/* 将位置指针移动到：从文件末尾计算，偏移量为-10 个字节的位置 */
```

```
/* 向后移动 */
```

例 12.7：编程读出文件 stu.dat 中第三个学生的数据。如程序 12.7 所示。

```
#include <stdio.h>
struct student
{
    int num;
    char name[20];
    char sex;
    int age;
    float score;
};
main()
{
    struct student stud;
    FILE *fp;
    int i=2;
    if((fp=fopen("stud.dat","rb"))==NULL)
    {
        printf("can't open file stud.dat\n");
        exit(1);
    }
    fseek(fp,i*sizeof(struct student),SEEK_SET); /* 定位第 3 个记录 */
    if(fread(&stud,sizeof(struct student),1,fp)==1) /* 将 1 个记录读出 */
    {
        printf("%d,%s,%c,%d,%f\n",stud.num,stud.name,stud.sex, /* 打印此记录 */
            stud.age,stud.score);
    }
    else
        printf("record 3 does not presented.\n");
    fclose(fp);
}
```

程序 12.8 从文件读出数据操作

例 12.8: 编写一个程序, 对文件 stud.dat 加密。加密方式是对文件中所有第奇数个字符的中间两个二进制位进行取反。如程序 12.9 所示。

```
#include <stdio.h>
main()
{
    FILE *fp;
    unsigned char ch1,ch2;
    if((fp=fopen("stud.dat","rb+"))==NULL) /* 打开已经存在的文件, 可写, 二进制 */
    {
        printf("can't open file stud.dat\n");  exit(1);
    }
    ch2=24; /* 密钥 ch<=0001,1000 */
    ch1=fgetc(fp);/* 从文件读第一个字符, ch1 */
    while(!feof(fp))
    {
        ch1=ch1^ch2; /* 加密字符: 与 1 异或该位取反; 与 0 异或该位不变 */
        fseek(fp,-1,SEEK_CUR); /* 写入原来位置 */
        fputc(ch1,fp);
        fseek(fp,1,SEEK_CUR); /* 跳过一个字符 (偶数字符), ch1 */
        ch1=fgetc(fp); /* 从文件读下一个字符, ch1 */
    }
    fclose(fp); /* 关闭文件 */
}
```

程序 12.9 文件加密

对一个数据取反, 可以用 “~” 运算符; 对一个数据某个位取反可以将该位与 1 异或 “^”。

3、ftell-获取当前位置指针函数

功能: 得到文件当前位置指针的位置, 此位置相对于文件开头的。

格式: long ftell(FILE *fp);

返回值: 就是当前文件指针相对文件开头的位置。

六、出错的检测 (了解)

在调用各种输入/输出函数时, 如果出现错误, 除了函数返回值有所反映外, 还可以用 ferror 函数检查。

ferror(fp);

如果返回 0, 表示没有错误; 非 0, 表示有错误。

注意：每次调用输入/输出函数，均产生一个新的 `ferror` 函数的值，即该值反映最后一次 I/O 操作的状态。

【编程处理文件及其数据的基本步骤】

1、定义变量，至少定义一个文件指针变量。文件指针变量的个数视处理的文件的多少而定。一个文件指针变量对应一个文件。

2、打开文件：`fopen`(被处理的文件名，“操作方式”)，同时给文件指针变量赋值。对文件的建立或打开操作是否成功进行判断。形式如下：

```
if((fp=fopen("被处理的文件名","操作方式"))==NULL)
{ printf("Don't create this file."); exit(1); }
```

3、对文件的操作及其数据的处理。包括：（一般要用循环结构）

1) 数据写入文件。关键是：写入哪些数据（用相关的变量表示，或从键盘输入）？怎样写入（相关的写入操作函数）？将数据写入到哪个文件（用相应的文件指针变量表示）？如何结束写入（对循环的控制）操作？

2) 从文件读取数据。关键是：从哪个文件中读取数据（文件用相应的文件指针变量表示）？读取哪些数据（文件的内容）？怎样读取数据（相关的读取操作函数）？读取的数据置于何处以及对数据的处理（相关的变量）？如何结束读取（对循环的控制）？

4、关闭文件：`fclose`(文件指针变量)