第3章 顺序结构程序设计

案例导入——计算圆面积问题

【问题描述】 输入一个圆的半径，计算并输出该圆的面积。

【解题分析】 假设圆半径用变量 r 表示，圆面积用变量 s 表示，圆周率近似取为 π=3.14，我们就可以利用圆面积的公式 $s=\pi r^2$ 来计算给定半径的圆的面积。

上面的过程如果用 C 语言来编程，应该如何实现呢？可以先定义计算过程中用到的变量，然后调用输入语句由用户输入半径的值，再通过适当的语句计算对应的圆面积，最后输出计算结果。

像这种按照顺序的方式依次执行特定语句的程序设计方式称为顺序结构程序设计。具体来说，本题的 C 语言程序可以编写如下：

```
#include <stdio.h>
int main( )
{
    float r;
    float s;
    scanf("%f ",&r);
    s=3.14 * r * r;
    printf("r = %.2f, s = %.2f\n",r,s);
    return 0;
}
```

程序运行时，通过键盘输入 2 后回车，则输出结果如下：

```
2
r = 2.00, s = 12.56
```

顺序结构程序设计的编程思想比较简单，相对容易理解。不过在编写 C 语言程序的过程中往往涉及变量输入输出的格式、表达式计算等细节问题，需要多加注意。


顺序结构导学

导学与自测

扫二维码观看本章的预习视频——顺序结构导学，并完成以下课前自测题目。

【自测题 1】 数学表达式$\frac{ab}{2(a+b)}$在 C 语言程序中的正确表达为(　　)。

A. ab/2(a+b)　　B. a * b/2 * a+b

C. a * b/2 * (a+b)　　D. a * b/(2 * (a+b))

【自测题 2】 把数学表达式 $s=\frac{\sqrt{3}}{4}a^2$ 写成符合 C 语言规则的语句,正确的是(　　)。

A. s=3^(1/2)/4a^2　　B. s=sqrt(3)/4 * a^2

C. s=sqrt(3)/4 * a * a　　D. s=sqrt(3)/(4 * a**2)

程序里要对数据进行各种操作,其中进行各种运算操作是最基本的操作之一。在 C 语言程序中,使用表达式,即通常所说的计算式子,描述各种运算。表达式是由参与运算的数据和表示运算的符号,按照一定的规则组成的式子。描述运算的符号称为运算符,由一个或两个特定符号表示一种运算。

C 语言具有丰富的运算符,可分为多种类型。

(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(=,+=,-=,*=,/=,%=,等)。

(6) 条件运算符(? 和:)。

(7) 逗号运算符(,)。

(8) 指针运算符(* 和 &)。

(9) 取类型长度运算符(sizeof)。

(10) 强制类型转换运算符((数据类型))。

(11) 分量运算符(.和->)。

(12) 下标运算符([])。

(13) 其他(如函数调用运算符())。

C 语言的运算符按照参与运算的数据个数,分为单目运算符(一个运算数)、双目运算符(两个运算数据)和三目运算符(三个运算数据);按照运算的先后次序,分为 15 个优先等级(见附录 B);另外还规定了运算符的结合性,即在相同优先级的运算符相邻时,是先计算左边的还是先计算右边的,先计算左边的是左结合,先计算右边的是右结合。双目运算符都是左结合的,单目和三目运算符都是右结合的。

本章主要讨论算术运算、赋值运算、自增自减运算以及位运算的运算符,以及由它们构成的表达式,同时介绍各种运算符的优先级和结合性,最后讲解顺序结构程序设计方法。

3.1 算术运算和算术表达式

算术运算是最常用的运算，在大多数程序里都要进行。表 3.1 列出了 C 语言提供的算术运算符。

表 3.1 算术运算符

C 语言中的操作	算术运算符	C 语言中的操作	算术运算符
加法运算	+	除法运算	/
减法运算	-	取模运算(求余数)	%
乘法运算	*		

需要注意的是，C 语言中使用的特殊符号，星号(*)表示乘号，斜杠(/)表示除号，百分号(%)表示求整数相除的余数。另外，加号(+)除了可以表示两个数相加外，还表示正号，例如，+5。类似地，减号(-)除了可以表示两个数相减外，还表示负号，例如，-12。

3.1.1 整数算术运算

如果参与运算的操作数都是整数，例如：

```
3+5,5-7,4*3,6/4,7%4
```

C 语言规定，运算的结果一定是整数。也就是说，3+5 的结果是 8，5-7 的结果是-2，4*3 的结果是 12，6/4 的结果是 1，7%4 的结果是 3。特别需要注意的是，整数除法运算和求余数运算与数学中的规定有所不同。

再如：

```
3/5,结果为 0。
3%5,结果为 3(商 0 余 3)。
```

特别注意：

(1) 运算符%只能用于整数运算。

(2) **a%b 的值为 0，则 a 能被 b 整除。**

(3) **a%b 的值为 0～b-1。**

(4) **a%b 结果的符号与 a 相同。**

这些性质在以后的程序中会经常用到。

思考：在 C 语言中，1+1/2+1/3+1/4+1/5 的运算结果是多少？

3.1.2 实数算术运算

如果参与运算的操作数都是实数，例如：

```
3.4+5.7,5.1-7.3,4.7*3.2,6.5/4.6
```

C 语言规定，运算的结果一定是实数。也就是说，3.4＋5.7 的结果是 9.1，5.1－7.3 的结果是－2.2，4.7 * 3.2 的结果是 15.04，6.5/4.6 的结果是 1.41。特别需要注意的是，**实数不能使用运算符%**。

思考：在 C 语言中，1.0＋1.0/2.0 的运算结果是多少？

3.1.3 混合算术运算

如果参与运算的操作数一个是整数，另一个是实数，例如：

```
3+5.7,5.1-7,4.7*3,6/4.6
```

C 语言规定，运算的结果一定是实数。也就是说，3＋5.7 的结果是 8.7，5.1－7 的结果是－1.9，4.7 * 3 的结果是 14.1，6/4.6 的结果是 1.3。这种情况也不能使用运算符%。

思考：在 C 语言中，1＋1.0/2 的运算结果是多少？

3.1.4 算术表达式

算术表达式是由参与算术运算的操作数(可以是常量、变量、函数等)、算术运算符和圆括号组成的符合 C 语言语法规则的式子，形式和数学中的算术表达式类似，但 C 语言中的算术表达式必须写成一行的形式，例如，数学中的$\frac{3}{5}$，在 C 语言中必须写成 3/5 的形式。

再如，数学表达式$\frac{x_1+x_2+x_3+x_4}{5}$的 C 语言表达式是(x1＋x2＋x3＋x4)/5。

数学表达式 b^2-4ac 的 C 语言表达式是 b * b－4 * a * c。

数学表达式$\frac{a+b}{c-d}$的 C 语言表达式是(a＋b)/(c－d)。

数学表达式 πr^2 的 C 语言表达式是 3.1415926 * r * r(π 是常数，不可以写成符号 π)。

数学表达式$\frac{a}{x}+by$ 的 C 语言表达式是 a/x＋b * y。

3.1.5 算术表达式的计算规则

算术表达式按照运算符的优先规则是从左到右计算的，C 语言算术运算符的优先规则和数学中的规定是一样的，即

```
高：*    /    %
低：+    -
```

例如，算术表达式 8－13/5＋4 * 8－7＋6%3，先从左到右计算 13/5、4 * 8、6%3，得到

8－2＋32－7＋0，再从左到右计算，得到 31。

和数学中一样，算术表达式中可以有括号，但无论多少层，都只使用圆括号。圆括号中的表达式优先级别是最高的，要先计算括号中的表达式。

例如，((8－13)/5＋4) * 8－(7＋6%3)，先计算(8－13)和(7＋6%3)，其中(7＋6%3)先计算 6%3，再计算 7＋0，结果是 7；再计算(－5/5＋4)，再计算 3 * 8，再计算 24－7，结果是 17(见图 3.1)。

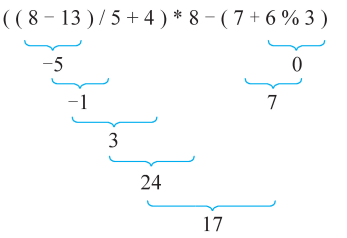

图 3.1　表达式计算过程示意图

3.2　赋值运算和赋值表达式

3.2.1　赋值运算符

1. 基本赋值运算

C 语言的赋值运算符是＝，一般表达形式如下：

```
变量=表达式
```

其中，表达式可以是常量、变量、函数等。赋值运算是优先级很低的运算，赋值运算过程：先计算赋值运算符右边的表达式的值，然后将计算结果赋给赋值运算符左边的变量。

这里一定要注意，＝左边一定要是一个变量，否则有语法错误。赋值运算是改变变量取值的一个重要手段。例如：

```
a=3                  //将 3 赋值给 a
b=sum/30             //将 sum 的值除以 30 再赋值给 b
```

2. 赋值运算的类型转换问题

经常会遇到赋值运算符两侧的数据类型不一致的情况，在执行赋值运算时就要进行类型转换。转换时，以赋值运算符左侧的变量的类型为准进行。例如，有以下定义：

```
int   a;
float   x;
```

执行 a＝45.78 时，a 的取值是 45。

执行 x＝623 时，x 的取值是 623.000000。

3. 复合赋值运算符

C 语言允许将形式为

```
变量=变量　算术运算符　表达式
```

的表达式简洁地写成

变量　算术运算符=　表达式

注意：在算术运算符与赋值运算符之间不允许有任何空格。共有 5 种算术复合赋值运算符，如表 3.2 所示。

表 3.2　算术复合赋值运算符

C 语言中的操作	算术复合赋值运算符	C 语言中的操作	算术复合赋值运算符
加赋值运算	+=	除赋值运算	/=
减赋值运算	-=	取余赋值运算	%=
乘赋值运算	*=		

例如：

a+=1 等同于 a=a+1(将 a 的当前值加上 1 后，再赋值给 a)。

x-=y+1 等同于 x=x-(y+1)。

a*=b 等同于 a=a*b。

x/=n+1 等同于 x=x/(n+1)。

x%=10 等同于 x=x%10。

C 语言还提供了 5 种位逻辑复合赋值运算符：<<=，>>=，&=，^=，|=。

3.2.2　赋值表达式

由赋值运算符将一个变量和一个表达式连接起来的式子称为赋值表达式。

例如，x=5+6*a 是一个赋值表达式。如果 a 的值是 10，则 C 语言规定，表达式 5+6*a 的值为 65，变量 x 赋值后的值为 65，表达式 x=5+6*a 的值也为 65。

赋值运算符右侧的表达式可以是任意合法的表达式，当然也可以是一个赋值表达式。例如，x=(y=15)，赋值运算符右侧圆括号内的 y=15 就是一个赋值表达式，它的值等于 15。执行表达式 x=(y=15)，相当于执行 y=15 得到 15 后，再执行为 x 赋值的操作，结果 x 也取值 15。

赋值运算符是右结合性的运算符，即按照自右而左的结合顺序，因此，x=(y=15)与 x=y=15 等价。

赋值表达式也可以包含复合的赋值运算符。例如，a+=a*=a，假设 a 的初值为 10，按照自右而左的结合顺序，先进行 a*=a 的运算，它相当于 a=a*a，a*a 的值为 10*10，等于 100，赋值给赋值运算符左侧的 a，a 的新值成为 100，也就是表达式 a*=a 的值是 100；再进行 a+=100 的运算，相当于 a=a+(100)，a 的最终值为 100+100=200。

作为表达式的一种，赋值表达式还可以出现在其他语句(如输出语句、循环语句等)中。例如，语句“printf("%d",x=y);”，如果 y 的值为 23，先完成表达式 x=y 的运算，x 的值为 23，再输出表达式 x=y 的值，其值为 23。这样在一个语句中可以完成赋值和输出两种操作功能。

3.3 自增自减运算

在程序设计中经常会用到给变量加 1 或减 1 的运算，C 语言为此专门提供了两个运算符：自增运算符(++)和自减运算符(--)，如表 3.3 所示。

表 3.3 自增自减运算符用法表

C 语言中的操作	运 算 符	用 法	含 义
自增运算	++	++n	先将 n 的值加 1，后使用 n 的值
	++	n++	先使用 n 的值，后将 n 的值加 1
自减运算	--	--n	先将 n 的值减 1，后使用 n 的值
	--	n--	先使用 n 的值，后将 n 的值减 1

自增自减运算符只对单个变量进行操作，称为单目运算符。常用于循环语句中使循环控制变量自动加 1 或减 1，也用于指针变量，使指针指向上一个或下一个地址。

对于++n 和 n++，单独使用时，意义相同，执行运算后，都是使变量 n 的值加 1。如果用在赋值语句中，意义有所不同。例如，假设有变量定义“int n,x,y;”，并对变量 n 赋初值“n=4;”，执行语句“x=++n;”与执行语句“y=n++;”，变量 x 和 y 的值是不同的。

执行“x=++n;”是先将 n 的值加 1，再将 n 的值赋值给 x，因此 x 的值为 5。

执行“y=n++;”是先将 n 的值赋值给 y，再将 n 的值加 1，因此 y 的值为 4。

例 3.1 自增运算符的前置后置对比。

程序如下：

```
#include <stdio.h>
int main( )
{
    int  n,x,y;
    n=4;
    x=++n;
    printf("n=%d\tx=%d\n",n,x);
    n=4;
    y=n++;
    printf("n=%d\ty=%d\n",n,y);
    return  0;
}
```

程序运行结果：

```
n=5     x=5
n=5     y=4
```

类似地，对于--n 和 n--，单独使用时，意义也相同，都是使变量 n 在原值的基础上减 1。但如果用在赋值语句中，意义就有所不同。

例 3.2 自减运算符的前置与后置对比。

程序如下：

```
#include <stdio.h>
int main( )
{
    int  n,x,y;
    n=4;
    x=--n;
    printf("n=%d\tx=%d\n",n,x);
    n=4;
    y=n--;
    printf("n=%d\ty=%d\n",n,y);
    return  0;
}
```

程序运行结果：

```
n=3     x=3
n=3     y=4
```

如果自增/自减运算符分别采用前置或后置方式用于 printf 函数的输出项，输出结果也是不一样的。例如，“printf("n＝%d\n",n＋＋);”和“printf("n＝%d\n",＋＋n);”。前者输出 n 原来的值，然后 n 增 1；后者先执行 n 增 1，然后输出增 1 之后的新 n 值。

例 3.3 自增自减运算用在输出语句中。

程序如下：

```
#include <stdio.h>
int main( )
{
    int  n;
    n=4;
    printf("%d\t",n);
    printf("%d\t",++n);
    printf("%d\n\n",n);
    n=4;
    printf("%d\t",n);
    printf("%d\t",n++);
    printf("%d\n\n",n);
    n=4;
    printf("%d\t",n);
    printf("%d\t",--n);
    printf("%d\n\n",n);
    n=4;
    printf("%d\t",n);
    printf("%d\t",n--);
    printf("%d\n\n",n);
    return 0;
}
```

程序运行结果：

```
4    5    5
4    4    5
4    3    3
4    4    3
```


运算符的优先级与结合性

3.4 优先级和类型转换

如果一个表达式中有多个运算符，那么运算的先后顺序就显得很重要。在 C 语言中，优先级顺序由符合标准数学用法的一系列排列规则决定，称为优先级法则。如果一个表达式中参与运算的数据类型不同，在运算前就需要进行类型转换，在 C 语言中有自动类型转换和强制类型转换两种方法。

3.4.1 优先级

3.1 节已经介绍了，在由多个算术运算符构成的算术表达式中，运算符的优先级别由高到低是：圆括号(())；双目算术运算符乘、除、取余(*，/，%)；双目算术运算符加、减(+，-)。这些运算符的结合性是由左向右。

如果加入单目算术运算符++，--，+(正号)，-(负号)，则算术运算符的优先级别由高到低变为：圆括号(())；单目算术运算符(++，--，+，-)；双目算术运算符乘、除、取余(*，/，%)；双目算术运算符加、减(+，-)。其中，单目算术运算符的结合性是由右向左。

赋值运算符的优先级别低于所有算术运算符，结合性是由右向左。

上述规则用表格总结如表 3.4 所示。

表 3.4 前面已学的运算符的优先级和结合性

运 算 符	优 先 级 别	结 合 性	类 型
()	高 ↑ 低	由左向右	圆括号
++，--，+，-		由右向左	单目算术运算符
*，/，%		由左向右	双目算术运算符乘、除、取余
+，-		由左向右	双目算术运算符加、减
=，+=，-=，*=，/=，%=		由右向左	赋值运算符

3.4.2 类型转换

1. 自动类型转换

在 3.1 节和 3.2 节中都提到，不同类型数据参与运算时，C 语言采用自动类型转换的方

式处理，转换处理隐含在计算过程中，将一种类型的数据转换为另一种兼容的类型。

例如，表达式 2+4.5 的计算，将整型数据 2 转换为 double 型数 2.0，再与 4.5 进行加法运算，这个转换是 C 语言编译系统自动完成的，无须程序员关心。

在有赋值运算符参与的表达式中，也可以进行自动转换。例如，如果有变量定义：double x，则赋值运算 x＝45 的结果是变量 x 取值为 45.0，整型数据 45 自动转换为 double 型数据。

注意：这时如果将一个 double 型数据赋值给一个整型变量，会出现截尾情况，例如，如果有变量定义：int a，执行赋值运算 a＝123.756 的结果是变量 a 取值 123，小数部分被截去。

C 语言自动类型转换规则如图 3.2 所示，当不同数据进行混合运算时，基本原则就是表达数据范围小的类型会自动转换为表达数据范围大的类型进行运算。

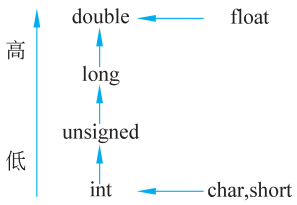

图 3.2　类型自动转换规则

2. 强制类型转换

C 语言也允许根据需要按与自动类型转换不同的方式进行强制类型转换。

例 3.4　自动转换效果示例。

程序如下：

```
#include <stdio.h>
int main( )
{
    int  x,y;
    float  ave;
    x=12 ;y=25 ;
    ave=(x+y)/2;
    printf("ave=%f\n",ave);
    return  0;
}
```

程序运行结果：

```
ave=18.000000
```

其中，表达式(x+y)/2，由于变量 x、y 都为整数，因此要依据整数除法的规则进行，按照 C 语言的运算规则，表达式的值为整数 18，赋值给 float 型变量 ave，结果为 18.0，所以在输出时得到了 ave＝18.000000 的结果(Visual C++ 默认 6 位小数)。

例 3.5　使用强制类型转换示例。

程序如下：

```
#include <stdio.h>
int main( )
{
    int  x,y;
    float  ave;
    x=12 ;y=25 ;
```

```
    ave=(float)(x+y)/2;
    printf("ave=%f\n",ave);
    return  0;
}
```

程序运行结果：

```
ave=18.500000
```

其中，表达式(float)(x+y)/2 中，将整型表达式(x+y)强制转换为 float 类型，再按照自动转换规则进行实数除法的运算，表达式的值为 18.5。为了得到 18.5 的实型结果，程序中也可不对 x+y 进行强制转换，而将整数 2 写成 2.0，读者可自行尝试。

例如，在以下程序中：

```
#include <stdio.h>
int main( )
{
    double  sum;
    sum=1+1/2+1/3+1/4+1/5;
    printf("sum=%f\n",sum);
    return  0;
}
```

由于所有的分数计算 1/2、1/3 中分子小于分母，分数结果为 0(整数除法结果)，若 sum 计算改为如下的形式：

```
sum=1+1/(double)2+1/(double)3+1/(double)4+1/(double)5;
```

虽然可以得到结果：sum＝2.283333，但书写稍显烦琐，为此可将计算 sum 的语句写为

```
sum=1+1.0/2+1.0/3+1.0/4+1.0/5;
```

此外，对于 x/y 这样的分数表达式，如果 x 和 y 都是整型变量，要想得到实型的商，可以使用下列表达之一：

```
1.0*x/y            x/1.0/y
```

强制类型转换的一般形式为

```
(类型名)表达式
```

其中，类型名表示 C 语言的标准数据类型，表达式可以是常量、变量或表达式。

例如：

```
(double)ave          ave 按照 double 类型参与运算
(int)(x+y)           将 x+y 的值转换成整型参与运算
(float)(7%3)         将 7%3 的值转换成 float 型参与运算
```

需要注意的是，知道了 C 语言的自动运算规则，可以灵活采用简洁的形式达到所需的要求，并不一定要使用强制类型转换。强制类型转换并不是真的把被转换的对象变成了强制的类型，而是让被强制的对象在这次运算时按照强制的类型参与运算。例如，(double)ave，如果 ave 原来是整型，经过(double)ave 后，ave 还是整型，不会因为曾被强

制按 double 型参与运算而改变 ave 本身的类型。


位运算符

3.5 位运算符

位运算是指按二进制位进行的运算。在系统软件中，常常需要处理二进制位的问题。C 语言提供了 6 个位运算符。位运算符的操作对象只能是整型或字符型数据，不能是浮点型数据，即只能用于带符号或无符号的 char、int、short、long 与 long long 类型。

C 语言提供的位运算符如表 3.5 所示。

表 3.5 位运算符

运算符	含义	描述
&	按位与	如果两个相应的二进制位都为 1，则该位的结果值为 1，否则为 0
|	按位或	两个相应的二进制位中只要有一个为 1，该位的结果值为 1
^	按位异或	若参加运算的两个二进制位值相同则为 0，否则为 1
~	按位取反	用来对一个二进制数按位取反，即将 0 变 1，将 1 变 0
<<	按位左移	用来将一个数的各二进制位左移 N 位，右补 0
>>	按位右移	将一个数的各二进制位右移 N 位，移到右端的低位被舍弃，对于无符号数，高位补 0

3.5.1 按位与运算符

按位与运算是指参加运算的两个数据按二进制位进行与运算。如果两个相应的二进制位都为 1，则该位的结果值为 1，否则为 0。这里的 1 可以理解为逻辑中的 true，0 可以理解为逻辑中的 false。按位与其实与后面要讲的逻辑运算符 && 的运算规则一致。逻辑运算符 &&，只有参与运算的运算数都不为 0（这里的 0 是指十进制数据，而不是二进制位，也就是说参与 && 运算的数据整体的值，而不是某一个二进制位的值），结果才为 1，只要有一个为 0，结果为 0。

若 A＝true，B＝true，则 A&&B＝true。

例如，3&5，3 的二进制编码是$(11)_2$（为了区分十进制和其他进制，本教材中，凡是非十进制的数据均给数据加上圆括号，圆括号后下标注明其进制，二进制则下标为 2）。内存储器存储数据的基本单位是字节（B），一字节由 8 个二进制位（b）组成。位是用以描述数据量的最小单位。在二进制系统中，每个 0 或 1 就是一个二进制位。将$(11)_2$补足成一字节，则是$(00000011)_2$。5 的二进制编码是$(101)_2$，将其补足成一字节，结果则是$(00000101)_2$。

按位与运算：

$$\begin{array}{r} (00000011)_2 \\ \underline{\&\ (00000101)_2} \\ (00000001)_2 \end{array}$$

由此可知，3&5＝1。

用 C 语言程序验证：

```
#include <stdio.h>
int main( )
{
    int a=3;
    int b =5;
    printf("%d\n",a&b);
    return  0;
}
```

程序运行结果：

```
1
```

按位与有以下用途。

(1) 清零。若想对一个存储单元清零，即使其全部二进制位为 0，只要找一个二进制数，其中各位符合以下条件：原来的数中为 1 的位，新数中相应位为 0。然后使二者进行按位与运算，即可达到清零的目的。

例如，原数为 43，即$(00101011)_2$；另找一个数，设它为 148，即$(10010100)_2$；将两者按位与运算：

$$\begin{array}{r} (00101011)_2 \\ \underline{\&\ (10010100)_2} \\ (00000000)_2 \end{array}$$

C 语言源代码：

```
#include <stdio.h>
int main()
{
    int a=43;
    int b =148;
    printf("%d\n",a&b);
    return 0;
}
```

程序运行结果：

```
0
```

(2) 取一个数中某些指定位。若有一个整数 a(2 字节)，想要取其中的低字节，只需要将 a 与 8 个 1 按位与即可。

$$\begin{array}{r}(0010110010101100)_2 \leftarrow a\\ \&(0000000011111111)_2 \leftarrow b\\ \hline (0000000010101100)_2 \leftarrow c\end{array}$$

(3) 保留指定位。与一个数进行按位与运算，此数在指定位取 1。

例如，有一个数 84，即$(01010100)_2$，想把其中从左边算起的第 3、4、5、7、8 位保留下来，运算如下：

$$\begin{array}{r}(01010100)_2 \leftarrow a\\ \&(00111011)_2 \leftarrow b\\ \hline (00010000)_2 \leftarrow c\end{array}$$

即 a=84，b=59，c=a&b=16。

C 语言源代码：

```
#include <stdio.h>
int main()
{
    int a=84;
    int b =59;
    printf("%d\n",a&b);
    return 0;
}
```

程序运行结果：

```
16
```

3.5.2 按位或运算符

按位或运算是指两个相应的二进制位中只要有一个为 1，该位的结果值为 1。借用逻辑学中或运算的规则来说就是一真为真。

例如，$(60)_8|(17)_8$，将八进制数 60 与八进制数 17 进行按位或运算。

$$\begin{array}{r}(00110000)_2 \leftarrow a\\ |\ (00001111)_2 \leftarrow b\\ \hline (00111111)_2 \leftarrow c\end{array}$$

C 语言源代码：

```
#include <stdio.h>
int main()
{
    int a=060;
    int b =017;
    printf("%d",a|b);
    return 0;
}
```

程序运行结果：

```
63
```

按位或运算常用来对一个数据的某些位定值为 1。例如，如果想使一个数 a 的低 4 位改为 1，则只需要将 a 与$(17)_8$进行按位或运算即可。

3.5.3 按位异或运算符

按位异或的运算规则是：若参加运算的两个二进制位值相同则为 0，否则为 1。即 0^0=0，0^1=1，1^0=1，1^1=0。例如：

$$\begin{array}{r} (00111001)_2 \leftarrow a \\ \wedge\ (00101010)_2 \leftarrow b \\ \hline (00010011)_2 \leftarrow c \end{array}$$

C 语言源代码：

```
#include <stdio.h>
int main( )
{
    int a=071;
    int b =052;
    printf("%d\n",a^b);
    return 0;
}
```

程序运行结果：

```
19
```

按位异或运算有以下 3 个用途。

(1) 使特定位翻转。设有数$(01111010)_2$，想使其低 4 位翻转，即 1 变 0，0 变 1，可以将其与$(00001111)_2$进行异或运算，即

$$\begin{array}{r} (01111010)_2 \\ \wedge\ (00001111)_2 \\ \hline (01110101)_2 \end{array}$$

运算结果的低 4 位正好是原数低 4 位的翻转。可见，要使哪几位翻转，就将与其进行按位异或运算的这几位置为 1 即可。

(2) 与 0 相异或，保留原值。例如：

$$\begin{array}{r} 12\wedge 0=12 \\ (00001010)_2 \\ \wedge\ (00000000)_2 \\ \hline (00001010)_2 \end{array}$$

因为原数中的 1 与 0 进行异或运算得 1，0^0 得 0，故保留原数。

(3) 交换两个值，不用临时变量。例如，a＝3，即$(11)_2$；b＝4，即$(100)_2$。想将 a 和 b 的值互换，可以用以下赋值语句实现：

```
a=a^b;
b=b^a;
a=a^b;
```

具体计算过程如下：

a＝$(011)_2$

b＝$(100)_2$

a＝$(111)_2$(a＝a^b 的结果，a 已变成 7)

b＝$(011)_2$(b＝b^a 的结果，b 已变成 3)

a＝$(100)_2$(a＝a^b 的结果，a 已变成 4)

上述过程等效于以下两步。

① 执行前两个赋值语句："a＝a^b;"和"b＝b^a;"，相当于"b＝b^(a^b);"。

② 再执行第三个赋值语句：a＝a^b。由于 a 的值等于(a^b)，b 的值等于(b^a^b)，因此，相当于 a＝a^b^b^a^b，即 a 的值等于 a^a^b^b^b，等于 b。

C 语言源代码：

```
#include <stdio.h>
int main( )
{
    int a=3;
    int b =4;
    a=a^b;
    b=b^a;
    a=a^b;
    printf("a=%d b=%d\n",a,b);
    return 0;
}
```

程序运行结果：

```
a=4 b=3
```

3.5.4 按位取反运算符

按位取反运算符(～)是单目运算符，用于求整数的二进制反码，即分别将操作数各二进制位上的 1 变为 0，0 变为 1。

例如，求～$(77)_8$的 C 语言源代码：

```
#include <stdio.h>
int main( )
{
    int a=077;
    printf("%d\n",~a);
```

```
    return 0;
}
```

程序运行结果：

```
-64
```

八进制数 077 对应的 8 位二进制数为$(00111111)_2$，按位取反得到$(11000000)_2$，由于整数在计算机中使用补码表示，通过将二进制补码$(11000000)_2$除符号位外各位取反再加 1 得到其对应真值为－64。

3.5.5 按位左移运算符

按位左移运算符(＜＜)是用来将一个数的各二进制位左移若干位，移动的位数由右操作数指定(右操作数必须是非负值)，其右边空出的位用 0 填补，高位左移溢出的则舍弃。

例如，将 a 的二进制数左移 2 位，右边空出的位补 0，左边溢出的位舍弃。若 a＝15，即$(00001111)_2$，左移 2 位得$(00111100)_2$。

C 语言源代码：

```
#include <stdio.h>
int  main( )
{
    int a=15;
    printf("%d\n",a<<2);
    return 0;
}
```

程序运行结果：

```
60
```

左移 1 位相当于该数乘以 2，左移 2 位相当于该数乘以 $2^2=4$。15＜＜2＝60，即 15 乘以 4。但此结论只适用于该数左移时被溢出舍弃的高位中不包含 1 的情况。

假设以 1 字节(8 位)存一个整数，若 a 为无符号整型变量，则 a＝64 时，左移一位时溢出的是 0，而左移 2 位时，溢出的高位中包含 1。

3.5.6 按位右移运算符

按位右移运算符(＞＞)是用来将一个数的各二进制位右移若干位，移动的位数由右操作数指定(右操作数必须是非负值)，移出右端的低位被舍弃，对于无符号数，高位补 0。对于带符号数，某些机器将左边空出的部分用符号位填补(即算术移位)，而另一些机器则将左边空出的部分用 0 填补(即逻辑移位)。注意：对无符号数，右移时左边高位移入 0。对于带符号数，如果符号位原来为 0(该数为正)，则左边也是移入 0；如果符号位原来为

1(即负数),则左边移入 0 还是 1 要取决于所用的计算机系统。有的系统移入 0,有的系统移入 1。移入 0 的称为逻辑移位,即简单移位;移入 1 的称为算术移位。

例如,a 的值是八进制数$(113755)_8$,即$(1001011111101101)_2$。

```
a>>1 (0100101111110110)2  (逻辑右移时)
a>>1 (1100101111110110)2  (算术右移时)
```

在有些系统中,a>>1 得八进制数 045766,而在另一些系统上可能得到的是八进制数 145766。Visual C++ 和其他一些 C 语言编译系统采用的是算术右移,即对有符号数右移时,如果符号位原来为 1,左边移入高位的是 1。

C 语言源代码:

```
#include <stdio.h>
int main( )
{
  short int a=0113755;
  printf("%ho\n",short(a>>1));
  printf("%hd\n",short(a>>1));
  return 0;
}
```

程序运行结果:

```
145766
-13322
```

这里 a 被定义为短整型变量,所占存储空间的长度为 16 位,初值为八进制数 113755,即二进制数 1001011111101101,最高位(即符号位)为 1。由于这里右移运算采取的是算术右移策略,因此左侧空出来的最高位使用原符号位的值填充,即最高位补 1。因此,右移一位后的二进制结果为 1100101111110110,对应的八进制值为 145766,对应的十进制值为−13322。

3.5.7 位逻辑复合赋值运算符

位运算符与赋值运算符可以组成位逻辑复合赋值运算符,如 &=,|=,>>=,<<=,^=等。

例如:

```
a &=b 相当于 a =a & b
a <<=2 相当于 a =a <<2
```

3.6 使用数学库函数

C 语言程序是由一个一个的函数组成的,有一些函数是用户编写的自定义函数(第 6 章详细介绍),还有一些是预先定义的标准 C 函数。程序中经常用到的一些操作(例如,

输入输出等)都被事先编写成相应的函数,保存在 C 的函数库中,可以供用户使用时直接调用。

预定义的含义就是该函数已编写好并已编译。在连接时,与用户写的程序连接在一起形成可执行的程序。

在 ANSI C 中,所有的函数在使用之前都必须被声明。根据标准 C 函数不同的类别,将其声明信息放在不同的头文件中,例如,printf()和 scanf()等输入输出函数的声明信息放在头文件 stdio.h 中,需要时使用预编译处理命令 #include <stdio.h>进行声明。

标准 C 函数中还有一类是数学函数,用来完成一些常用的数学计算。这些数学函数的声明信息放在头文件 math.h 中。需要时使用预编译处理命令 #include <math.h>进行声明,告诉编译器程序将会调用数学函数库,这个命令要放在程序的开始处。

函数通常是按如下顺序书写的:函数名、左圆括号、参数(或用逗号分隔的参数列表)、右圆括号。例如,计算 x 的平方根的函数 sqrt 的书写格式是 sqrt(x),计算 e^x 的函数 exp 的书写格式是 exp(x),计算 x^y 的函数 pow 的书写格式是 pow(x,y),计算 $|x|$ 的函数 fabs 的书写格式是 fabs(x),计算弧度值 x 的三角函数的书写格式是 sin(x)、cos(x)、tan(x),等等。

当需要处理的表达式中出现数学函数时,表达式要按照 C 语言的格式书写。例如:

$(x+2)e^{2x}$ 的 C 语言表达式是(x+2) * exp(2 * x)。

$\frac{-b+\sqrt{b^2-4ac}}{2a}$ 的 C 语言表达式是(-b+sqrt(b * b-4 * a * c))/(2 * a)。

$\text{money}(1+\text{rate})^{\text{year}}$ 的 C 语言表达式是 money * pow((1+rate),year)。

数学函数中没有提供余切函数,可以利用正弦函数和余弦函数进行计算,假设要计算的角度 x 是以角度为单位的,还需要将它转换为弧度。x 的余切计算公式是

```
cos (x * 3.14/180) / sin (x * 3.14/180)
```

或者利用正切函数:

```
1/tan(x * 3.14/180)
```

3.7 C 语句及顺序结构程序设计


顺序结构程序设计

结构化程序的基本结构之一是顺序结构。C 语言程序由函数组成,而函数的功能靠语句实现。因此,需要先学习 C 语句的语法。

3.7.1 C 语句概述

C 语言的语句用来向计算机系统发出指令,完成特定操作。一个语句经编译后产生若干机器指令。

一个实际的 C 语言程序通常包含若干语句。

C语言的语句分成两类：简单语句和控制语句(包括复合语句)。简单语句执行一些基本的操作，控制语句控制程序中语句的执行流程。在默认情况下，C语言程序的执行流程是顺序的，即逐个执行按书写顺序排列的语句，除非遇到控制语句改变流程的执行顺序。

3.7.2 简单语句

简单语句由一个表达式以及紧跟其后的分号构成。虽然任何表达式都可以加分号构成语句，但只有赋值类表达式语句才有实际意义。

表达式语句格式如下：

```
表达式;
```

例如：

```
n++;
x1 =(-b+sqrt(b * b-4 * a * c))/(2 * a);
x2 =(-b-sqrt(b * b-4 * a * c))/(2 * a);
z =x+a%3 * (int)(x+y)%2/4;
w =(float)(a+b)/2+(int)x%(int)y;
p =money * pow((1 +rate), year);
```

函数调用语句格式如下：

```
函数调用;
```

例如：

```
printf("Hello!");
printf("sum=%f\n",sum);
scanf("%d%d",&a,&b);
putchar(ch1);
```

二者结合构成的赋值语句示例如下：

```
ch1=getchar( );
```

3.7.3 顺序结构程序设计举例

顺序结构是最简单、最常用的程序结构，这种结构的C语言程序完全按照语句书写的先后顺序执行，主要由表达式语句和标准库函数调用语句构成。例如，由赋值操作、输入输出操作构成的程序很多都是顺序结构。

通常编写程序解决实际问题的步骤如下。

(1) 分析实际问题。

(2) 写出算法。

(3) 根据算法写出相应的C语言程序。

下面举例说明顺序结构的程序设计过程。