第三章 最简单的C程序设计—顺序程序设计

作者：石璞东

参考资料：《C程序设计（第四版）》谭浩强

前言

为了能编写出`C`语言程序，必须具备以下的知识和能力：

- 要有正确的解题思路，即学会设计算法，否则无从下手；
- 掌握`C`语言的语法，知道怎样使用`C`语言所提供的功能编写出一个完整的、正确的程序；
- 在写算法和编写程序时，要采用结构化程序设计方法，编写出结构化的程序；

3.1 顺序程序设计举例

1. 华氏温度 ➡️ 摄氏温度

题解：转换公式为`c = 5/9*(f-32)`，其中`f`表示华氏温度，`c`表示摄氏温度。

```c
#include<stdio.h>
int main(){
    float c,f;
    printf("请任意输入一个华氏温度：");
    scanf("%f",&f);
    c = (5.0/9)*(f-32);
    printf("%f对应的摄氏温度为：%f\n",f,c);
    return 0;
}
```

2. 计算存款利息，用p_0表示本金，p_1表示第一种方法的本息和，p_2表示第二种方法的本息和，p_3表示第三种方法的本息和。假设任意输入p_0，想存3年，求出其对应三种可选方法的本息和：
    - 活期，年利率为r_1； ➡️ $p_1 = p_0(1+r_1)$
    - 定期，年利率为r_2； ➡️ $p_2 = p_0(1+r_2)$
    - 两次半年定期，年利率为r_3； ➡️ $p_3 = p_0(1+r_3/2)*(1+r_3/2)$

```c
#include<stdio.h>
int main(){
    float p0,p1,p2,p3;
    float r1 = 0.0036,r2 = 0.0225,r3 = 0.0198;
    printf("请输入本金：");
    scanf("%f",&p0);
    p1 = p0 * (1+r1);
    p2 = p0 * (1+r2);
    p3 = p0 * (1+r3/2) * (1+r3/2);
    printf("p1=%f\np2=%f\np3=%f\n",p1,p2,p3);
    return 0;
}
```

3.2 数据的表现形式及其计算

3.2.1 常量和变量

在计算机高级语言中，数据有两种表现形式：**常量**和**变量**。

1. 常量
    - 概念：在程序运行过程中，其值不能被改变的量称为常量；
    - 常用常量的分类：
        - 整型常量：如100、600、-123等；
        - 实型常量（包含两种表示形式）：
            - 十进制小数形式：如123.456、0.97等；
            - 指数形式：如12.34e3（表示12.34*10^3），由于在计算机输入或输出时，无法表示上角或下角，故规定以字母`e`或`E`代表以10为底的指数；
        - 字符常量（包含两种形式）：
            - 普通字符：字符常量只能是一个字符，如`a`、`A`、`z`等，当在计算机存储单元中进行存储时，并非存储字符本身，而一般通过`ASCII`码进行存储；
            - 转义字符：即以字符`\`开头的字符序列，如`\n`表示换行符，`\t`表示将输出的位置跳到下一个`tab`位置（制表位置），一个`tab`位置为8列，常见的转义字符如表3.1所示；
        - 字符串常量：字符串常量是双撇号中的全部字符，但不包含双撇号本身，如"CHINA"、"Jeffery"等，**单撇号只能包含一个字符，双撇号内可以包含一个字符串**；
        - 符号常量：用`#define`指令，指定用一个符号名称代表一个常量，如`#define PI 3.1416`，在对程序进行编译前，预处理器先对`PI`进行处理，把所有`PI`全部置换为3.1416，在预编译后，符号常量已全部变成字面常量，使用符号常量有以下好处：
            - 含义清楚：在定义符号常量名时应考虑**见名知意**，如看到`#define PI 3.1416`时

从`PI`就可大致知道其表示圆周率；

- 在需要改变程序中多处用到的同一个常量时，能做到**一改全改**；
- 变量与符号常量的区别在于符号常量不占用内存，预编译成功后，该符号就不存在了，即不能执行赋值等操作；

表 3.1　转义字符及其作用

转义字符	字符值	输出结果
\'	一个单撇号(')	具有此八进制码的字符
\"	一个双撇号(")	输出此字符
\?	一个问号(?)	输出此字符
\\	一个反斜线(\)	输出此字符
\a	警告(alert)	产生声音或视觉信号
\b	退格(backspace)	将当前位置后退一个字符
\f	换页(form feed)	将当前位置移到下一页的开头
\n	换行	将当前位置移到下一行的开头
\r	回车(carriage return)	将当前位置移到本行的开头
\t	水平制表符	将当前位置移到下一个 tab 位置
\v	垂直制表符	将当前位置移到下一个垂直制表对齐点
\o、\oo 或\ooo 其中 o 代表一个八进制数字	与该八进制码对应的 ASCII 字符	与该八进制码对应的字符
\xh[h…] 其中 h 代表一个十六进制数字	与该十六进制码对应的 ASCII 字符	与该十六进制码对应的字符

「注」：

- 表3.1中倒数第二行是一个以八进制数表示的字符，如`\101`代表八进制数`101`的`ASCII`字符，即`A`；
- 表3.1中倒数第一行是一个以十六进制数表示的`ASCII`字符，如`\x41`代表十六进制数`41`的`ASCII`字符，即`A`；

2. 变量

- 定义：变量代表一个有名字的、具有特定属性的一个存储单元，它用来存放数据，也就是存放变量的值，在程序运行期间，变量的值是可以改变的；
- 使用方法：先定义后使用；
- 解释：变量名实际上是以一个名字代表的一个存储地址，在对程序编译连接时由编译系统给每一个变量名分配对应的内存地址，从变量中取值，实际上是通过变量名找到相应的内存地址，从该存储单元中读取数据；
- 案例：如图3.3中`a`是变量名，3是变量`a`的值，即存放在变量`a`的内存单元中的数据；

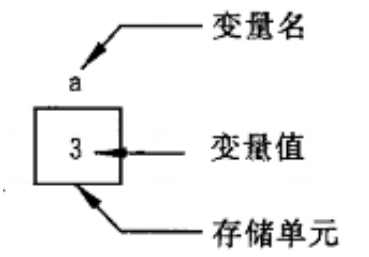

图 3.3

3. 常变量

如`const int a =3`表示`a`被定义为一个整型变量，指定其值为3，而且在变量存在期间其值不能改变，常变量的与常量的异同是：

- 常变量具有变量的基本属性，即有类型、占内存单元，仅仅不允许被改变值；
- 常变量是有名字的不变量，而常量是没有名字的不变量，有了名字就便于在程序中被引用；

请看示例：

- 定义符号常量：`#define Pi 3.1415926`
- 定义常变量：`const float pi=3.1415926`

符号常量`Pi`和常变量`pi`都代表3.1415926，在程序中都能使用，但二者性质不同：

- 定义符号常量用`#define`指令，它是预编译指令，它只是用符号常量代表一个字符串，在预编译时仅是进行字符串替换，在预编译后，符号常量就不存在了（全部置换成3.1415926），对符号常量的名字是不分配存储单元的；
- 常变量要占用存储单元，有变量值，只是该值不改变而已；

综上所述，常变量具有符号常量的优点，而且使用方便，有了常变量以后，可以不必多用符号常量。

4. 标识符

在计算机高级语言中，用来对变量、符号常量名、函数、数组、类型等命名的有效字符序列统称为标识符，简单来说，标识符就是一个对象的名字，前面用到的变量名`p1`、`p2`、`c`、`f`、符号常量名`PI`、`PRICE`、函数名、`printf`等都是标识符，`C`语言规定标识符只能由**字母**、**数字**和**下划线**3种字符组成，且第一个字符必须为字母或下划线：

- 合法的变量名：sum、shipudong1、find_max_num；
- 不合法的变量名：K.D.Jeffery、¥123、3hahaha；

「注」：编译系统将大写字母和小写字母认为是两个不同的字符，因此`sum`和`SUM`是两个不同的变量名。

3.2.2 数据类型

C语言要求在定义所有的变量时都要指定变量的类型，在计算机中，数据是存放在存储单元中的，它是具体存在的，而且，存储单元是由有限的字节构成的，每一个存储单元中存放数据的范围是有限的，不可能存放**无穷大**的数，也不能存放循环小数。

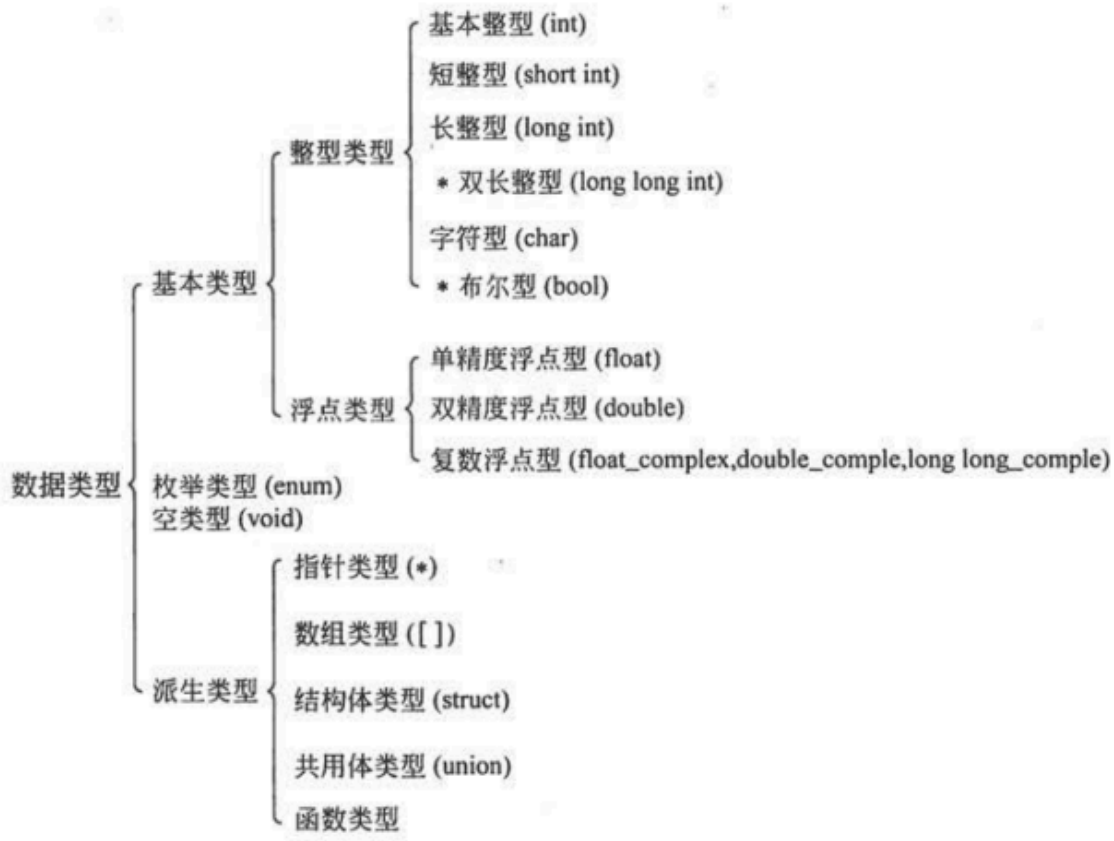

图 3.4

所谓类型，就是对数据分配存储单元的安排，包括存储单元的长度（占多少字节）以及数据的存储形式，不同的类型分配不同的长度和存储形式。

其中基本类型（包括整型和浮点型）和枚举类型变量的值都是数值，统称为算术类型，算术类型和指针类型统称为纯量类型，因为其变量的值以数字来表示的，枚举类型是程序中用户定义的整数类型，数组类型和结构体类型统称为组合类型，共用体类型不属于组合类型，因为在同一时间内只有一个成员具有值，函数类型用来定义函数，描述一个函数的借口，包括函数返回值的数据类型和参数的类型。

不同类型的数据在内存中占用的存储单元长度是不同的，例如，Visual C++ 6.0为`char`型（字符型）数据分配1个字节，为`int`型（基本整型）数据分配4个字节，存储不同类型数据的方法也是不同的。

3.2.3 整型数据

1. 整型数据的分类
    - 基本整型 `int`

编译系统分配给`int`型数据2个字节（16个二进制位）或4个字节（由具体的`C`编译系统自行决定），在存储单元中的存储方式是：用整数的补码形式存放，请看案例：

一个整数的补码是此数的二进制形式，如5的二进制形式是101，如果用两个字节存放一个整数，则在存储单元中数据形式如图3.5所示，如果是一个负数，则应先求出负数的补码。

求负数的补码的方法是：先将此数的绝对值写成二进制形式，然后对其后面所有各二进位按位取反，再加一，如图3.6所示。

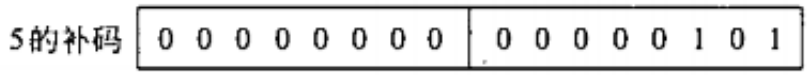

图 3.5

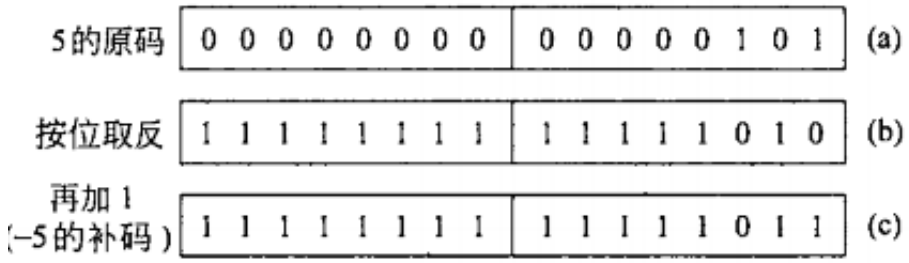

图 3.6

在存放整数的存储单元中，最左面一位是用来表示符号的，如果该位为0，表示数值为正；如果该位为1，表示数值为负。

「注」：如果给整型变量分配2个字节，则存储单元中能存放的最大值为0111111111111111，第1位为0代表正数，后面15位全位1，则其取值范围位-2^{15}～（2^{15}-1）；同理若给整型变量分配4个字节，其取值范围为-2^{31}～（2^{31}-1）。

- 短整型 `short int`

  类型名为`short int`或`short`，`Visual C++ 6.0`分配给`short int`2个字节，存储方式与`int`型相同，其取值范围为-2^{15}～（2^{15}-1）。

- 长整型 `long int`

  类型名为`long int`或`long`，`Visual C++ 6.0`分配给`long`4个字节，其取值范围为-2^{31}～（2^{31}-1）。

- 双长整型`long long int`

  类型名为`long long int`或`long long`，一般分配8个字节。

  「注」：`C`标准没有具体规定各种类型数据所占用存储单元的长度，这是由编译系统自行决定的，`C`标准只要求`long`型数据长度不短于`int`型，`short`型不短于`int`型，即`sizeof(short) <= sizeof(int) <= sizeof(long) <= sizeof(long long)`。由于不同的编译系统对不同的数据类型所分配到的字节数不同，因此容易出现**数据溢出**。

2. 整型变量的符号属性

变量值在存储单元中都是以补码形式存储的，存储单元中的第一个二进制位代表符号，整型变量的值的范围包括负数到正数。

表 3.2　整型数据常见的存储空间和值的范围

类　型	字节数	取值范围
int(基本整型)	2	$-32768\sim32767$，即 $-2^{15}\sim(2^{15}-1)$
	4	$-2147483648\sim2147483647$，即 $-2^{31}\sim(2^{31}-1)$
unsigned int(无符号基本整型)	2	$0\sim65535$，即 $0\sim(2^{16}-1)$
	4	$0\sim4294967295$，即 $0\sim(2^{32}-1)$
short(短整型)	2	$-32768\sim32767$，即 $-2^{15}\sim(2^{15}-1)$
unsigned short(无符号短整型)	2	$0\sim65535$，即 $0\sim(2^{16}-1)$
long(长整型)	4	$-2147483648\sim2147483647$，即 $-2^{31}\sim(2^{31}-1)$
unsigned long(无符号长整型)	4	$0\sim4294967295$，即 $0\sim(2^{32}-1)$
long long(双长型)	8	$-9223372036854775808\sim9223372036854775807$ 即 $-2^{63}\sim(2^{63}-1)$
unsigned long long (无符号双长整型)	8	$0\sim18446744073709551615$，即 $0\sim(2^{64}-1)$

在实际应用中，有的数据的范围常常只有正值（如学号、年龄、库存量、存款额等）。为了充分利用变量的值的范围，可以将变量定义为**无符号**类型，即在类型符号前面加上修饰符`unsigned`，表示指定该变量是**无符号整数**类型；如果加上修饰符`signed`，则是**有符号类型**，因此，在以上四种整型数据的基础上可以扩展为以下8种整型数据：

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

「注」：如果既未指定为`signed`也未指定为`unsigned`，则默认为**有符号类型**，如`signed int a`与`int a`等价。

「总结」：有符号整型数据存储单元中最高位代表符号（0为正，1为负），如果指定为`unsigned`（无符号）型，存储单元中全部二进制位都用作存放数值本身，而没有符号。无符号型变量只能存放不带符号的整数，如12345678等，不能存放负数，如-12345678。由于左面最高位不再用来表示符号，而用来表示数值，因此无符号整型变量中可以存放的正数的范围比一般整型变量中正数的范围扩大一倍，如果在程序中定义`a`和`b`两个短整型变量（占2个字节），其中`b`为无符号短整型：

```
short a;
```

```
unsigned short b;
```

则变量`a`的数值范围为-32768~32767，而变量`b`的数值范围为0～65535。

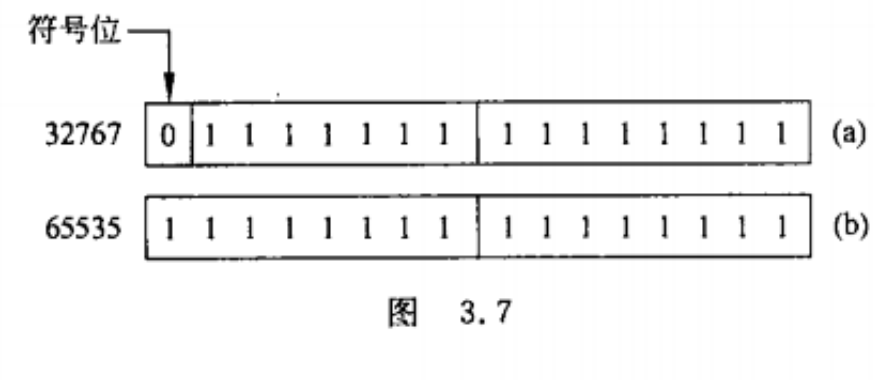

图 3.7

「注」：

- 只有整型（包括字符型）数据可以加`signed`或`unsigned`修饰符，实型数据不能加；
- 对无符号整型数据用`%u`格式输出，`%u`表示用无符号十进制数的格式输出。

在将一个变量定义为无符号整型后，不应向它赋予一个负值，否则会得到错误的结果，如：

```
unsigned short price = -1 //不能把一个负整数存储在无符号变量中
printf("%d\n",price)
```

请看演示结果：

```
#include<stdio.h>
int main(){
    unsigned short price = -1; //不能把一个负整数存储在无符号变量中
    printf("%d\n",price);
    return 0;
}

```

```
65535
Program ended with exit code: 0
```

最终结果与我们期待值不符合，原因是：系统对-1先转换成补码形式，就是全部二进制位都是1，然后把它存入变量`price`中，由于`price`是无符号短整型变量，其左面第一位不代表符号，按`%d`格式输出，就是65535。

3.2.4 字符型数据

1. 字符与字符代码

   字符与字符代码并不是任意写一个字符，程序都能识别的，只能使用系统的字符集中的字符，目前大多数系统采用`ASCII`字符集，各种字符集（包括`ASCII`字符集）的基本集都包括了127个字符：

   - 字母：大写英文字母A~Z，小写英文字母a～z；

- 数字：0～9；
- 专门符号：29个（!、“、#、‘、 （）、*、+、，、<等）；
- 空格符：空格、水平制表符（tab）、垂直制表符、换行、换页；
- 不能显示的字符：空字符、警告、退格、回车等；

字符是以整数形式的（字符的`ASCII`代码）存放在内存单元中的，如：

- 大写字母`A`的`ASCII`代码是十进制数65，二进制形式是1000001；
- 小写字母`a`的`ASCII`代码是十进制数97，二进制形式是1100001；

所有127个字符都可以用7个二进制位表示（`ASCII`代码为127时，二进制形式为1111111），所以在`C`中，指定用1个字节（8位）存储一个字符（所有系统都不例外），如小写字母`a`在内存中的存储情况如图3.9所示：

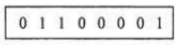

图 3.9

「注」：字符`1`和整数`1`是不同的概念，字符`1`只是代表一个形状为`1`的符号，在需要时按原样输出，在内存中以`ASCII`码形式存储，占1个字节，而整数1是以整数存储方式（二进制补码方式）存储的，占2个或4个字节，如图3.10所示。

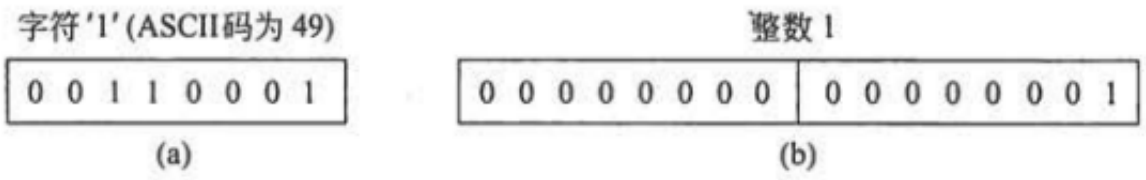

图 3.10

2. 字符变量

字符变量是用类型符`char`定义字符变量，如：`char c='?'`，定义`c`为字符型变量并使初值为字符`?`，其`ASCII`代码为`63`，系统把整数`63`赋给变量`c`，`c`是字符变量，实质上是一个字节的整型变量，由于它常用来存放字符，所以称为字符变量，可以把0～127之间的整数赋给一个字符变量，在输出字符变量的值时，可以选择以十进制整数形式输出，或以字符形式输出，如`printf("%d %c\n",c,c)`，其输出结果为：

```c
#include<stdio.h>
int main(){
    char c = '?';
    printf("%d %c\n",c,c);
    return 0;
}
```

```
63 ?
Program ended with exit code: 0
```

表 3.3　字符型数据的存储空间和值的范围

类　　型	字节数	取 值 范 围
signed char(有符号字符型)	1	−128～127,即-2^7～(2^7-1)
unsigned char(无符号字符型)	1	0～255,即 0～(2^8-1)

在使用有符号字符型变量时，允许存储的值为-128～127，但字符的代码不可能为负值，所以在存储字符时实际上只用到0～127这部分，如果将一个负整数赋给有符号字符型变量是合法的，但它不代表一个字符，而作为一字节整型变量存储负整数。

3.2.5 浮点型数据

在指数形式的多种表示方式中把小数部分中小数点前的数字为0、小数点后第一位数字不为0的表示形式称为**规范化的指数形式**，如0.314159*10^1就是3.14159的规范化的指数形式。

浮点型数据是用来表示具有小数点的实数的，浮点数类型包括三种：

- `float`型（单精度浮点型）：编译系统为每一个`float`型变量分配4个字节，数值以规范化的二进制数指数形式存放在存储单元中，在存储时，系统将实型数据分成小数部分和指数部分两个部分，分别存放，小数部分的小数点前面的数为0，如3.14159在内存中的存放形式如图所示：

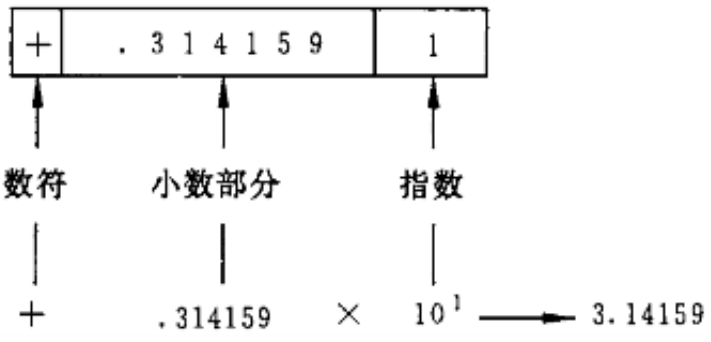

图　3.11

小数部分占的位数越多，数的有效数字越多，精度也就越高；指数部分占的位数越多，则能表示的数值范围越大。

- `double`型（双精度浮点型）

  为了扩大能表示的数值范围，用8个字节存储一个`double`型数据，在`C`语言中进行浮点数的算数运算时，将`float`型数据都自动转换为`double`型，然后进行计算；

- `long double`型（长双精度）

  不同的编译系统对`long double`型的处理方法不同，分配到的字节数也不同；

表 3.4　实型数据的有关情况

类　型	字节数	有效数字	数值范围(绝对值)
float	4	6	0 以及 $1.2\times10^{-38}\sim3.4\times10^{38}$
double	8	15	0 以及 $2.3\times10^{-308}\sim1.7\times10^{308}$
long double	8	15	0 以及 $2.3\times10^{-308}\sim1.7\times10^{308}$
	16	19	0 以及 $3.4\times10^{-4932}\sim1.1\times10^{4932}$

【`float`和`double`的取值范围】参考链接：https://blog.csdn.net/wsccdsn/article/details/8180301

3.2.6 怎样确定常量的类型

在`C`语言中，不仅变量有类型，常量也有类型。为什么要把常量分为不同的类型呢？在程序中出现的常量是要存放在计算机中的存储单元中的，这就必须确定分配给它多少字节，按什么方式存储。

怎样确定常量的类型呢？从常量的表示形式即可以判定其类型。对于字符常量很简单，只要看到由单撇号括起来的单个字符或转义字符就是字符常量；对于数值常量，不带小数点的数值是整型常量，但应注意其有效范围，凡以小数形式或指数形式出现的实数，是浮点型常量，在内存中都以指数形式存储。

3.2.7 运算符和表达式

几乎每一个程序都需要进行运算，对数据进行加工处理，否则程序就没有意义了。要进行运算，就需规定可以使用的运算符。`C`语言的运算符范围很宽，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，例如将赋值符 = 作为赋值运算符、方括号作为下标运算符等。

1. 基本的算术运算符

表 3.5　最常用的算术运算符

运算符	含　义	举例	结　果
+	正号运算符(单目运算符)	+a	a 的值
−	负号运算符(单目运算符)	−a	a 的算术负值
*	乘法运算符	a * b	a 和 b 的乘积
/	除法运算符	a/b	a 除 b 的商
%	求余运算符%	a%b	a 除 b 的余数
+	加法运算符	a+b	a 和 b 的和
−	减法运算符	a−b	a 和 b 的差

「注」：

- 两个实数相除的结果是双精度实数，两个整数相除的结果为整数，如5/3==1，舍去小数部分；
- **%** 运算符要求参加运算的运算对象为整数，结果也是整数，如8%3==2；
- 除 **%** 以外的运算符的操作数都可以是任何算术类型；

2. 自增、自减运算符

   - ++i、--i：前置自增（减），先增（减）后用；
   - i++、i--：后置自增（减），先用后增（减）；

   「注」：自增运算符和自减运算符只能用于变量，而不能用于常量或表达式。

3. 算术表达式和运算符的优先级与结合性

   - 用算数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**`C`算数表达式**，其中运算对象包括常量、变量和函数；
   - `C`语言除了规定了运算符的优先级外，还规定了运算符的**结合性**：
     - 左结合性：运算对象先与左面的运算符结合，如a-b+c；
     - 右结合性：运算对象先与右面的运算符结合，如a=b=c；

4. 不同类型数据间的混合运算

   如果一个运算符的两侧的数据类型不同，则先自动进行类型转换，使二者具有同一类型，然后进行运算，其转换规律为：

   - +、-、*、/ 运算的两个数中有一个数为`float`或`double`型，结果是`double`型，因为系统将所有`float`型数据都先转换为`double`型，然后进行运算；
   - 如果`int`型与`float`或`double`型数据进行运算，先把`int`型和`float`型数据转换为`double`型，然后进行运算，结果是`double`型；
   - 字符`char`型数据与整型数据进行运算，就是把字符的`ASCII`代码与整型数据进行运算，如：12+'A'，由于字符`A`的`ASCII`代码是65，相当于65+12，其值为77。

   请看演示案例：

```c
#include<stdio.h>
int main(){
    char c1,c2;
    c1 = 'A';
    c2 = c1+32;
    printf("%c\n",c2);
    printf("%d\n",c2);
    return 0;
}
```

请看演示结果：

```c
#include<stdio.h>
int main(){
    char c1,c2;
    c1 = 'A';
    c2 = c1+32;
    printf("%c\n",c2);
    printf("%d\n",c2);
    return 0;
}
```

```
a
97
Program ended with exit code: 0
```

5. 强制类型转换运算符

   一般形式为：（类型名）（表达式），如`(double) a`、`(int) (x+y)`等，在强制类型转换时，得到一个所需类型的中间数据，而原来变量的类型未发生变化，有两种类型转换：

   - 自动类型转换：用户不用干预，系统自动进行的类型转换，如6+3.5等；
   - 强制类型转换：当自动类型转换不能实现目的时，可以用强制类型转换；

6. `C`运算符

(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) 成员运算符	(. −>)
(12) 下标运算符	([ ])
(13) 其他	(如函数调用运算符())

3.3 C语句

3.3.1 C语句的作用和分类

一个C程序可以由若干个源程序文件（编译时以文件模块为单位）组成，一个源文件可以由若干个函数和预处理指令以及全局变量声明部分组成，一个函数由数据声明部分和执行语句组成，语句的作用是向计算机系统发出操作指令，要求执行相应的操作，一个C语句经过编译后产生若干条机器指令，声明部分不是语句，它不产生机器指令，只是对有关数据的声明。

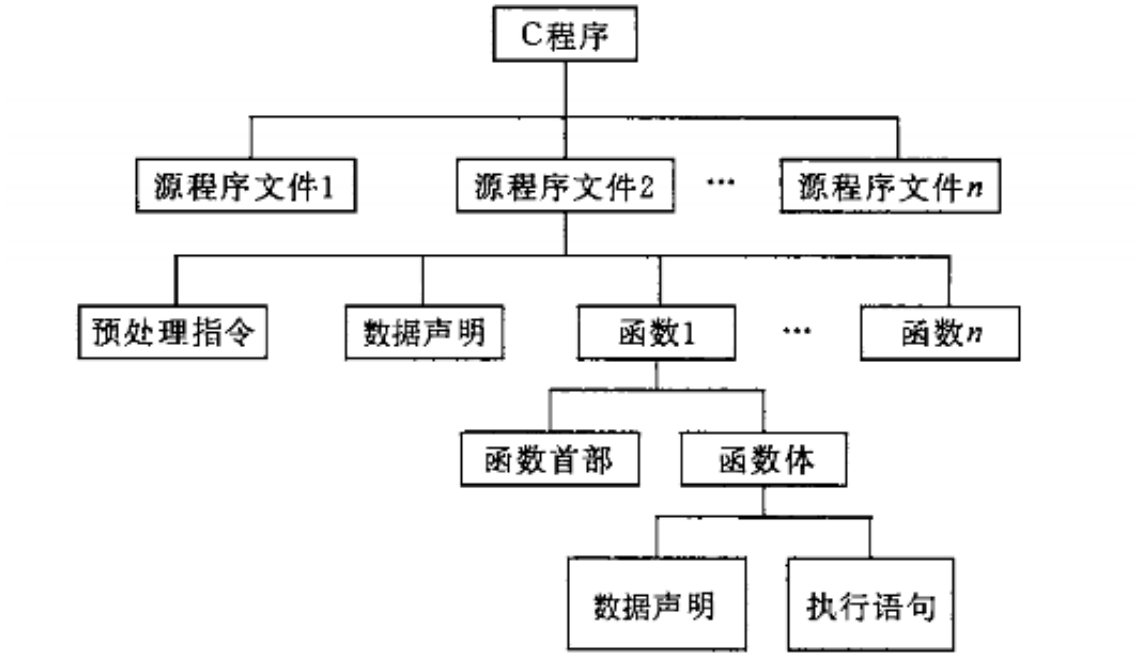

图 3.14

C语句分为以下5类：

- 控制语句（9种）
  - 条件语句：`if()...else...`
  - 循环语句：`for()...`、`while()...`、`do...while()`
  - 结束本次循环语句：`continue`
  - 中止执行`switch`或循环语句：`break`
  - 多分支选择语句：`switch`
  - 从函数返回语句：`return`
  - 转向语句：`goto`
- 函数调用语句

  由一个函数调用加一个分号构成；
- 表达式语句

  由一个表达式加一个分号构成，如`i +=1`是表达式，不是语句；`i +=1;`是语句；
- 空语句

  只有一个分号，可以用来作为流程的转向点，也可以用来作为循环语句中的循环体；
- 复合语句

用`{}`把一些语句和声明括起来成为复合语句，即语句块，复合语句中最后一个语句中最后的分号不能忽略不写。

3.3.2 最基本的语句—赋值语句

`c`程序中最常用的语句是：赋值语句和输入输出语句。

例3.1 给出三角形的三边长，求三角形面积。

```c
#include<stdio.h>
#include<math.h>
int main(){
    float a,b,c,s,area;
    printf("请输入三角形三边长：");
    scanf("%f %f %f",&a,&b,&c);
    if(a+b>c&&b+c>a&& a+c>b){
        s = (a+b+c)/2;
        area = sqrt(s*(s-a)*(s-b)*(s-c));
        printf("三角形三边长分别为a=%1.2f b=%1.2f c=%1.2f,面积为%1.2f\n",a,b,c,area);
    }else{
        printf("当前所输入的三边长不能构成三角形，请重新输入！\n");
    }
    return 0;
}
```

请看演示效果：

```c
#include<stdio.h>
#include<math.h>
int main(){
    float a,b,c,s,area;
    printf("请输入三角形三边长：");
    scanf("%f %f %f",&a,&b,&c);
    if(a+b>c&&b+c>a&& a+c>b){
        s = (a+b+c)/2;
        area = sqrt(s*(s-a)*(s-b)*(s-c));
        printf("三角形三边长分别为a=%1.2f b=%1.2f c=%1.2f,面积为%1.2f\n",a,b,c,area);
    }else{
        printf("当前所输入的三边长不能构成三角形，请重新输入！\n");
    }
    return 0;
}
```

```
请输入三角形三边长：3 4 5
三角形三边长分别为a=3.00 b=4.00 c=5.00,面积为6.00
Program ended with exit code: 0
```

1. 赋值运算符

   赋值运算符的作用就是将一个数据赋给一个变量，如`a=3`的作用就是将常量3赋值给变量`a`；

2. 复合的赋值运算符

   复合的运算符可以理解为在赋值符前加上其他运算符，如`+=`、`-=`、`*=`、`/=`、`%=`；

3. 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**，其形式为**变量 赋值运算符 表达式**，其作用是将一个表达式的值赋给一个变量，因此赋值表达式具有计算和赋值两个功能，如下所示：

   - a=b=c=5
   - a=5+(c=6)
   - a=(b=4)+(c=6)
   - a=(b=10)/(c=2)

4. 赋值过程中的类型转换

   类型转换规则：

   - 整型变量 = 浮点型数据（包括单、双精度）：对浮点数取整，即舍弃小数部分，然后赋予整型变量；
   - 单双精度变量 = 整型数据：数值不变，但以浮点数形式存储到变量中；
   - `float`变量 = `double`变量：先将双精度数转换为单精度，应注意双精度数值的大小不能超出`float`型变量的数值范围；
   - 整型变量 = 字符型变量：将字符的`ASCII`代码赋给整型变量；
   - 占字节少的整型变量或字符变量 = 占字节多的整型数据：只将其低字节原封不动地送到被赋值的变量，即发生**截断**，如：

     ```
     int i =289;
     char c = 'a';
     c = i;
     ```

     请看演示效果：

     ```
     #include<stdio.h>
     int main(){
         int i =289;
         char c = 'a';
         c = i;
         printf("%d\n",c);
         return 0;
     }
     ```

     ```
     33
     Program ended with exit code: 0
     ```

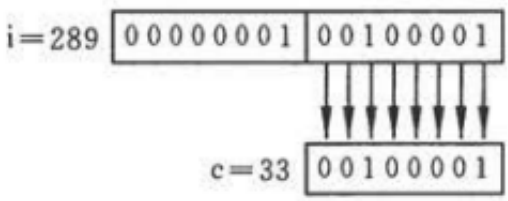

图 3.15

5. 赋值表达式和赋值语句

   赋值表达式的末尾没有分号，而赋值语句的末尾必须有分号，在一个表达式中可以包含一个或多个赋值表达式，但绝不能包含赋值语句。

6. 变量赋初值

   「注」：如果对几个变量赋于同一个初值，应写成`int a=3,b=3,c=3;`而不是`int a=b=c=3;`。

3.4 数据的输入输出

3.4.1 输入输出举例

例3.2 求$ax^2+bx+c=0$方程的根，a,b,c由键盘输入，设$b^2-4ac>0$。

```c
#include<stdio.h>
#include<math.h>
int main(){
    double a,b,c,disc,x1,x2,p,q;
    printf("请输入a,b,c的值：");
    scanf("%lf %lf %lf",&a,&b,&c);
    disc = b*b - 4*a*c;
    p = -b/(2.0*a);
    q = sqrt(disc)/(2.0*a);
    x1 = p+q;
    x2 = p-q;
    printf("方程%1.0fx^2+%1.0fx+%1.0f=0的根为
x1=%1.2f,x2=%1.2f\n",a,b,c,x1,x2);
    return 0;
}
```

请看演示结果：

```c
#include<stdio.h>
#include<math.h>
int main(){
    double a,b,c,disc,x1,x2,p,q;
    printf("请输入a,b,c的值: ");
    scanf("%lf %lf %lf",&a,&b,&c);
    disc = b*b - 4*a*c;
    p = -b/(2.0*a);
    q = sqrt(disc)/(2.0*a);
    x1 = p+q;
    x2 = p-q;
    printf("方程%1.0fx^2+%1.0fx+%1.0f=0的根为x1=%1.2f,x2=%1.2f\n",a,b,c,x1,x2);
    return 0;
}
```

```
请输入a,b,c的值: 1 3 2
方程1x^2+3x+2=0的根为x1=-1.00,x2=-2.00
Program ended with exit code: 0
```

3.4.2 有关数据输入输出的概念

C语言本身不提供输入输出语句，输入输出操作是由C标准函数库中的函数来实现的，输入设备包括：键盘、磁盘、光盘、扫描仪；输出设备包括显示器、打印机等；

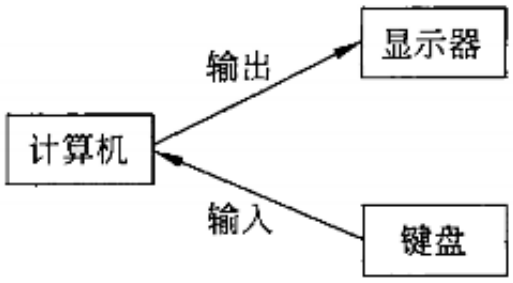

图 3.17

3.4.3 用`printf`函数输出数据

一般格式为：`printf(格式控制，输出表列)`，请看参数介绍：

- 格式控制：用双撇号括起来的一个字符串，包括**格式声明**和**普通字符**两个信息：
  - 格式声明：由 **%** 和**格式字符**组成，其作用是将输出的数据转换成特定的格式然后输出；
  - 普通字符：在输出时原样输出的字符；
- 输出表列：即程序需要输出的一些数据；

常用的几种**格式字符**：

- `d`格式符：用来输出一个有符号的十进制整数，也可指定输出数据的域宽（即所占用的列数），如`%5d`表示输出数据占5列；
- `c`格式符：用来输出一个字符；

- `s`格式符：用来输出一个字符串；
- `f`格式符：用来输出实数，以小数形式输出：
  - 基本型用**%f**：不指定输出数据的长度，由系统根据数据的实际情况决定数据所占的列数；
  - 指定数据宽度和小数位数，用**%m.nf**：如**%7.2f**指定了输出的数据占7列，包括2位小数；
  - 输出的数据向左对齐，用**%-m.nf**：当数据长度不超过`m`时，数据向左靠，右端补空格；
- `e`格式符：指定以指数形式输出实数；

3.4.4 用`scanf`函数输入数据

一般格式为：`scanf(格式控制，地址表列)`。

在输入数值数据时，如输入空格、回车、`Tab`键或遇非法字符（不属于数值的字符），则认为该数据结束。

3.4.5 字符数据的输入输出

- `putchar(c)`函数输出一个字符

```c
#include<stdio.h>
int main(){
    char a='m',b='a',c='n';
    putchar(a);
    putchar(b);
    putchar(c);
    putchar('\n');
    return 0;
}
```

```
#include<stdio.h>
int main(){
    char a='m',b='a',c='n';
    putchar(a);
    putchar(b);
    putchar(c);
    putchar('\n');
    return 0;
}
```

```
man
Program ended with exit code: 0
```

- `getchar(c)`函数输入一个字符

```
#include<stdio.h>
int main(){
    char a,b,c;
    a = getchar();
    b = getchar();
    c = getchar();
    putchar(a);
    putchar(b);
    putchar(c);
    putchar('\n');
    return 0;
}
```

例3.3 从键盘输入大写字母，显示屏上输出小写字母。

```
#include<stdio.h>
int main(){
    char c1,c2;
    c1 = getchar();
    c2 = c1+32;
    putchar(c2);
    putchar('\n');
    return 0;
}
```

课后题：4、5、6、7、8