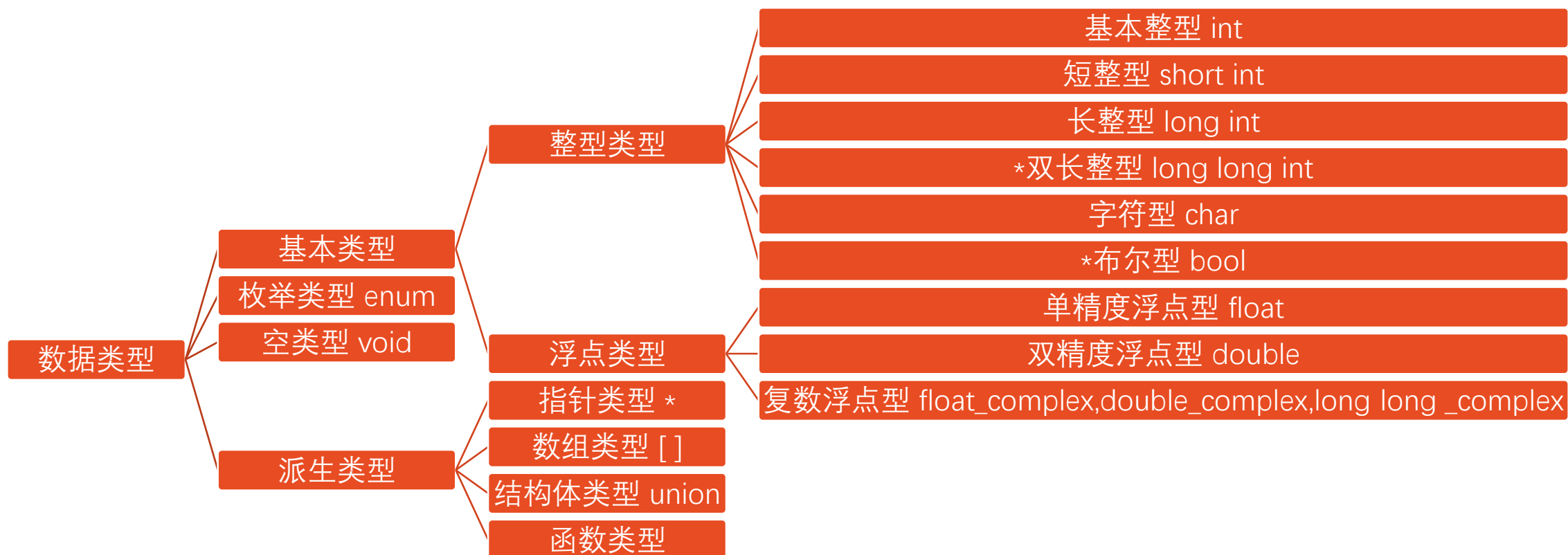


数据类型

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。



整型数据

| 整型数据类型 | 缺省形式的整型数据类型 | 字节数 | 取值范围 |
|--------------------------|--------------------|-----|--|
| [signed]int | int | 4 | -2147483648~2147483647 ($-2^{31} \sim 2^{31}-1$) |
| unsigned [int] | Unsigned | 4 | 0~4294967295 ($0 \sim 2^{32}-1$) |
| [signed] short [int] | short | 2 | -32768~32767 ($-2^{15} \sim 2^{15}-1$) |
| unsigned short [int] | unsigned short | 2 | 0~65535 ($0 \sim 2^{16}-1$) |
| [signed]long [int] | long | 4 | -2147483648~2147483647 ($-2^{31} \sim 2^{31}-1$) |
| unsigned long [int] | unsigned long | 4 | 0~4294967295 ($0 \sim 2^{32}-1$) |
| [signed]long long [int] | long long | 8 | -9223372036854775808~9223372036854775807 ($-2^{63} \sim 2^{63}-1$) |
| unsigned long long [int] | unsigned long long | 8 | 0~18446744073709551615 ($0 \sim 2^{64}-1$) |

说明: 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, 只要求
 $\text{sizeof}(\text{short}) \leq \text{sizeof}(\text{int}) \leq \text{sizeof}(\text{long}) \leq \text{sizeof}(\text{long long})$, 具体由各编译系统自行决定的。
sizeof是测量类型或变量长度的运算符。

整型数据

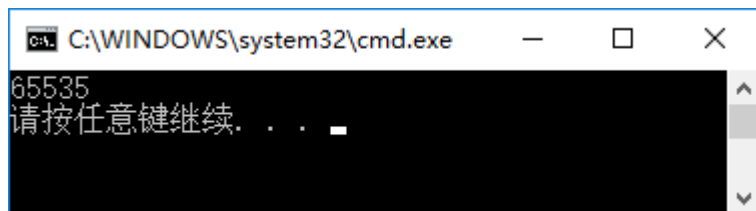
- (1) 只有整型(包括字符型)数据可以加signed或unsigned修饰符，实型数据不能加。
- (2) 对无符号整型数据用“%u”格式输出。%u表示用无符号十进制数的格式输出。如:

```
unsigned short price=50;    //定义price为无符号短整型变量  
printf("%u\n",price);      //指定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("%u\n",price);
```



例 整型变量的定义与使用

运行结果： $a + u = 22$,
 $b + u = -14$

```
#include <stdio.h>
void main()
{
```

说明： 可以看到不同种类的整型数据可以进行算术运算

```
    int a,b,c,d;    /* 指定 a、b、c、d 为整型变量 */
    unsigned u;    /* 指定 u 为无符号整型变量 */
    a = 12; b = -24; u = 10;
    c = a + u; d = b + u;
    printf (" a + u = %d , b + u = %d \n", c , d) ;
}
```

例 整型数据的溢出

```
#include <stdio.h>
void main()
{
    short a,b;
    a=32767;
    b=a+1;
    printf("%d,%d\n",a,b);
}
```

运行结果： 32767,-32768

说明： 数值是以补码表示的。一个整型变量只能容纳-32768 ~ 32767范围内的数，无法表示大于32767或小于-32768的数。遇此情况就发生“溢出”。

字符型数据

ASCII字符集包括:

- 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,包括
- ! " # & ' () * + , - . / : ; < = > ? [\] ^ _ ` { | } ~
- 空格符: 空格、水平制表符(tab)、垂直制表符、换行、换页(form feed)
- 不能显示的字符: 空(null)字符(以'\0'表示)、警告(以'\a'表示)、退格(以'\b'表示)、回车(以'\r'表示)等

ASCII码表

注意

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。

00110001

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。

00000000000000000001

整数运算1+1等于整数2，而字符'1'+ '1'
并不等于整数2或字符'2'。

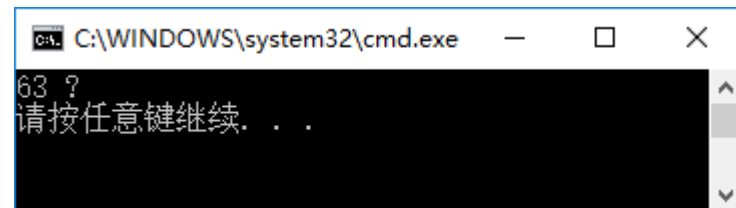
| ASCII 值 | 控制字符 | ASCII 值 | 控制字符 | ASCII 值 | 控制字符 | ASCII 值 | 控制字符 |
|---------|------|---------|---------|---------|------|---------|------|
| 0 | NUL | 32 | (space) | 64 | @ | 96 | , |
| 1 | SOH | 33 | ! | 65 | A | 97 | a |
| 2 | STX | 34 | " | 66 | B | 98 | b |
| 3 | ETX | 35 | # | 67 | C | 99 | c |
| 4 | EOT | 36 | \$ | 68 | D | 100 | d |
| 5 | ENQ | 37 | % | 69 | E | 101 | e |
| 6 | ACK | 38 | & | 70 | F | 102 | f |
| 7 | BEL | 39 | . | 71 | G | 103 | g |
| 8 | BS | 40 | (| 72 | H | 104 | h |
| 9 | HT | 41 |) | 73 | I | 105 | i |
| 10 | LF | 42 | * | 74 | J | 106 | j |
| 11 | VT | 43 | + | 75 | K | 107 | k |
| 12 | FF | 44 | , | 76 | L | 108 | l |
| 13 | CR | 45 | - | 77 | M | 109 | m |
| 14 | SO | 46 | . | 78 | N | 110 | n |
| 15 | SI | 47 | / | 79 | O | 111 | o |
| 16 | DLE | 48 | 0 | 80 | P | 112 | p |
| 17 | DC1 | 49 | 1 | 81 | Q | 113 | q |
| 18 | DC2 | 50 | 2 | 82 | R | 114 | r |
| 19 | DC3 | 51 | 3 | 83 | X | 115 | s |
| 20 | DC4 | 52 | 4 | 84 | T | 116 | t |
| 21 | NAK | 53 | 5 | 85 | U | 117 | u |
| 22 | SYN | 54 | 6 | 86 | V | 118 | v |
| 23 | TB | 55 | 7 | 87 | W | 119 | w |
| 24 | CAN | 56 | 8 | 88 | X | 120 | x |
| 25 | EM | 57 | 9 | 89 | Y | 121 | y |
| 26 | SUB | 58 | : | 90 | Z | 122 | z |
| 27 | ESC | 59 | ; | 91 | [| 123 | { |
| 28 | FS | 60 | < | 92 | \ | 124 | |
| 29 | GS | 61 | = | 93 |] | 125 | } |
| 30 | RS | 62 | > | 94 | ^ | 126 | ~ |
| 31 | US | 63 | ? | 95 | _ | 127 | DEL |

字符变量

字符变量是用类型符char定义字符变量。

```
char c='?';           //定义c为字符型变量并使初值为字符'?'。'?'的ASCII代码是63，系统把整数63赋给变量c。
```

```
printf("%d %c\n",c,c); //用"%d"格式输出十进制整数63，用"%c"格式输出字符'?'
```



A screenshot of a Windows command prompt window. The title bar shows the path C:\WINDOWS\system32\cmd.exe. The window contains the output of a C program: the number 63 followed by a space and the character '?', and a prompt '请按任意键继续...' (Press any key to continue...) below it.

```
C:\WINDOWS\system32\cmd.exe
63 ?
请按任意键继续...
```


例 向字符变量赋以整数

```
include <stdio.h>
void main()
{
    char c1,c2;
    c1=97;
    c2=98;
    printf("%c %c\n",c1,c2);
    printf("%d %d\n",c1,c2);
}
```

- 运行结果：

| | |
|----|----|
| a | b |
| 97 | 98 |

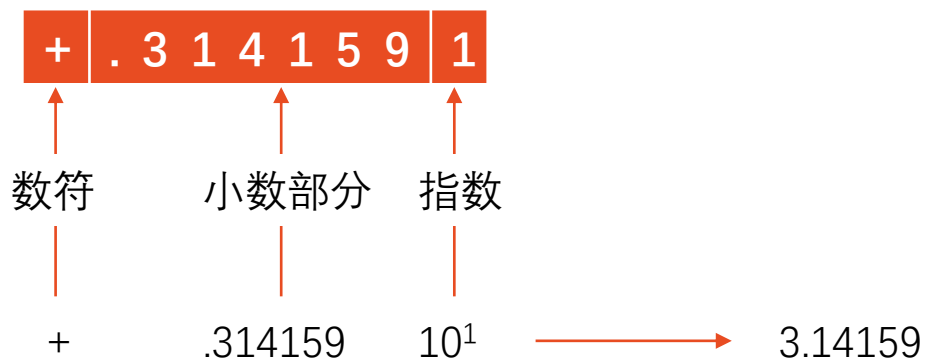
- 说明： 在第 3 和第 4 行中，将整数 97 和 98 分别赋给 c1 和 c2，它的作用相当于以下两个赋值语句：
c1 = ' a ' ; c2 = ' b ' ;
因为 'a' 和 'b' 的 ASCII 码为 97 和 98

浮点型格式

$$3.14159 = 3.14159 * 10^0 = 0.314159 * 10^1 = 314.159 * 10^{-2}$$

由于小数点位置可以浮动，所以实数的指数形式称为**浮点数**。

浮点数类型包括float(单精度浮点型)、double(双精度浮点型)、long double(长双精度浮点型)。



注意

由于用二进制形式表示一个实数以及存储单元的长度是有限的，因此不可能得到完全精确的值，只能存储成有限的精确度。小数部分占的位（bit）数愈多，数的有效数字愈多，精度也就愈高。指数部分占的位数愈多，则能表示的数值范围愈大。

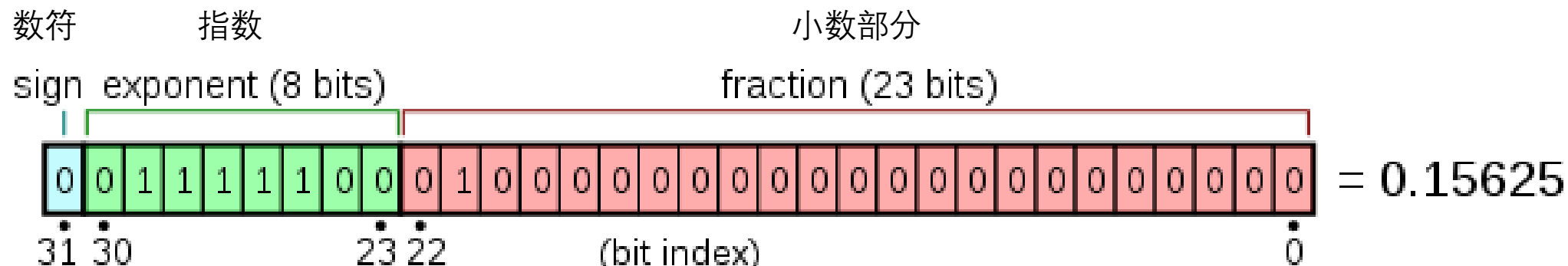
IEEE754单精度浮点型数据：32位

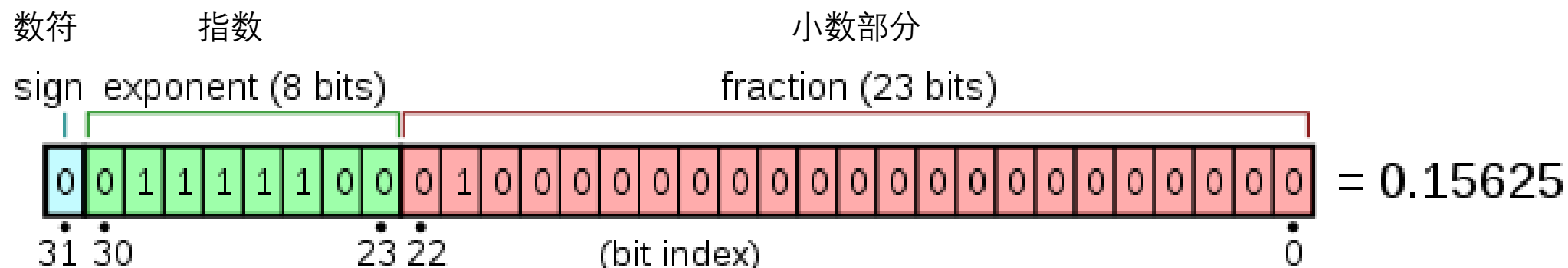
数符：1bit 指数：8bit 小数部分：24位（23位显式存储，第一位为隐含的1，除非指数为全0）

$$(-1)^{b_{31}} \times 2^{(b_{30}b_{29}\dots b_{23})_2 - 127} \times (1.b_{22}b_{21}\dots b_0)_2.$$

which yields

$$\text{value} = (-1)^{\text{sign}} \times 2^{(e-127)} \times \left(1 + \sum_{i=1}^{23} b_{23-i} 2^{-i} \right).$$





- $\text{sign} = b_{31} = 0$,
- $(-1)^{\text{sign}} = (-1)^0 = +1 \in \{-1, +1\}$,
- $e = b_{30}b_{29} \dots b_{23} = \sum_{i=0}^7 b_{23+i}2^{+i} = 124 \in \{1, \dots, (2^8 - 1) - 1\} = \{1, \dots, 254\}$,
- $2^{(e-127)} = 2^{124-127} = 2^{-3} \in \{2^{-126}, \dots, 2^{127}\}$,
- $1.b_{22}b_{21} \dots b_0 = 1 + \sum_{i=1}^{23} b_{23-i}2^{-i} = 1 + 1 \cdot 2^{-2} = 1.25 \in \{1, 1 + 2^{-23}, \dots, 2 - 2^{-23}\} \subset [1; 2 - 2^{-23}] \subset [1; 2)$.
- $\text{value} = (+1) \times 1.25 \times 2^{-3} = +0.15625$.

Note:

- $1 + 2^{-23} \approx 1.000\,000\,119$,
- $2 - 2^{-23} \approx 1.999\,999\,881$,
- $2^{-126} \approx 1.175\,494\,35 \times 10^{-38}$,
- $2^{+127} \approx 1.701\,411\,83 \times 10^{+38}$.

IEEE754

| Name | Common name | Base | Significant bits or digits | Decimal digits | Exponent bits | Decimal E max | Exponent bias ^[11] | E min | E max | Notes |
|----------------------------|---------------------|------|----------------------------|----------------|---------------|---------------|-------------------------------|---------|---------|-----------|
| binary16 | Half precision | 2 | 11 | 3.31 | 5 | 4.51 | $2^4 - 1 = 15$ | -14 | +15 | not basic |
| binary32 | Single precision | 2 | 24 | 7.22 | 8 | 38.23 | $2^7 - 1 = 127$ | -126 | +127 | |
| binary64 | Double precision | 2 | 53 | 15.95 | 11 | 307.95 | $2^{10} - 1 = 1023$ | -1022 | +1023 | |
| binary128 | Quadruple precision | 2 | 113 | 34.02 | 15 | 4931.77 | $2^{14} - 1 = 16383$ | -16382 | +16383 | |
| binary256 | Octuple precision | 2 | 237 | 71.34 | 19 | 78913.2 | $2^{18} - 1 = 262143$ | -262142 | +262143 | not basic |
| decimal32 | | 10 | 7 | 7 | 7.58 | 96 | 101 | -95 | +96 | not basic |
| decimal64 | | 10 | 16 | 16 | 9.58 | 384 | 398 | -383 | +384 | |
| decimal128 | | 10 | 34 | 34 | 13.58 | 6144 | 6176 | -6143 | +6144 | |

实型数据

| 类型 | 字节数 | 有效数字 | 数值范围（绝对值） |
|-------------|-----|------|---|
| float | 4 | 6 | 0以及 $1.2 \times 10^{-38} \sim 3.4 \times 10^{38}$ |
| double | 8 | 15 | 0以及 $2.3 \times 10^{-308} \sim 1.7 \times 10^{308}$ |
| long double | 8 | 15 | 0以及 $2.3 \times 10^{-308} \sim 1.7 \times 10^{308}$ |
| | 16 | 19 | 0以及 $3.4 \times 10^{-4932} \sim 1.1 \times 10^{4932}$ |

例 浮点型数据的舍入误差

```
#include <stdio.h>
void main()
{
    float a,b;
    a = 123456.789e5;
    b = a + 20 ;
    printf("%f\n",b);
}
```

运行结果： 123456.789e5

说明： 一个浮点型变量只能保证的有效数字是6位有效数字，后面的数字是无意义的，并不准确地表示该数。应当避免将一个很大的数和一个很小的数直接相加或相减，否则就会“丢失”小的数

常量的类型

'n'——字符常量

23——整型常量

3.14159——浮点型常量

- 从常量的表示形式即可以判定其类型。
- 不带小数点的数值是整型常量，但应注意其有效范围。
- 在一个整数的末尾加大写字母L或小写字母l，表示它是长整型(long int)，例如：123l.432L.0L
- 一个整常量后面加一个字母u或U，认为是unsigned int型。如果写成-12345u，则先将-12345转换成其补码53191，然后按无符号数存储。
- 凡以小数形式或指数形式出现的实数均是浮点型常量，在内存中都以指数形式存储。
- C编译系统把浮点型常量都按双精度处理，分配8个字节，除非后面加字母f或F，分配4个字节。

整型常量的表示方法

整型常量即整常数。在C语言中，整常数可用以下三种形式表示：

(1) 十进制整数。

如：123，-456。

(2) 八进制整数。以0头的数是八进制数。

如：0123表示八进制数123，等于十进制数83，-011表示八进制数-11，即十进制数-9。

(3) 十六进制整数。以0x开头的数是16进制数。

如：0x123，代表16进制数123，等于十进制数 291。 -0x12等于十进制数-10。

浮点型常量的表示方法

两种表示形式 { 小数 0.123 ✓ 1e3、1.8e-3、-123e-6、-.1e-3
 { 指数 3e-3 ✗ e3、2.1e3.5、.e3、e

注意:字母e(或E)之前必须有数字, 且e后面的指数必须为整数

字符常量

- (1)用单引号包含的一个字符是字符型常量
- (2)只能包含一个字符

'a', 'A', '1'
'abc', "a" ✗

字符串常量

字符串常量是一对双撇号括起来的字符序列。

合法的字符串常量:

“How do you do.”, “CHINA”, “a”, “\$123.45”

可以输出一个字符串, 如

```
printf("How do you do.");
```

不能把个字符串常量
赋给一个字符变量。



✓ c = 'a';

✗ c = "a";

c = "China";

C 规定以字符 '\0' 作为字符串结束标志。如果有一个字符串常量 "CHINA", 实际上在内存中是:

| | | | | | |
|---|---|---|---|---|----|
| C | H | I | N | A | \0 |
|---|---|---|---|---|----|

它占内存单元不是 5 个字符, 而是 6 个字符, 最后一个字符为 '\0'。但在输出时不输出 '\0'。

常量、变量与类型

```
float a=3.14159; //3.14159为双精度浮点常量，分配8个字节；a为float变量，分配4个字节
```

编译时系统会发出警告(warning: truncation from 'const double' to 'float'), 提醒用户注意这种转换可能损失精度

一般不影响结果的正确性，但会影响结果的精度。

可以在常量的末尾加专用字符，强制指定常量的类型：

```
float a=3.14159f;           //把此3.14159按单精度浮点常量处理，编译时不出现“警告”  
long double a = 1.23L;      //把此1.23作为long double型处理
```

类型是变量的一个重要的属性。变量是具体存在的实体，占用存储单元,可以存放数据。而类型是变量的共性，是抽象的，不占用存储单元，不能用来存放数据。

```
int a; a=3;                //正确。对整型变量a赋值  
int=3;                     //错误。不能对类型赋值
```

JAVA的数据类型

- 内置数据类型Java语言提供了八种基本类型。六种数字类型（四个整数型，两个浮点型），一种字符类型，还有一种布尔型。
 - byte (8bit整数), short, int, long, float, double, boolean, char (16bit unicode)
- 如果基本的整数和浮点数精度不够满足需求，那么可以使用java.math包中的两个很有用的类：BigInteger和BigDecimal。这两个类可以处理包含任意长度数字序列的数值，BigInteger类实现了任意精度的整数运算，BigDecimal实现了任意精度的浮点数运算。

Python的数据类型

- Number (数字)
 - int、float、bool、complex (复数)
- String (字符串)
- List (列表)
- Tuple (元组)
- Set (集合)
- Dictionary (字典)

运算符和表达式

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 | * & |
| 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() |

常用的算数运算符

| 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除以b的商 |
| % | 求余运算符 | a%b | a除以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 |



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



%运算符要求参加运算的运算对象
(即操作数)为整数，结果也是整数

自增 (++) 自减 (--) 运算符

++i, --i

在使用 i 之前, 先使 i 的值加/减1

i++, i--

在使用 i 之后, 使 i 的值加/减1

++i是先执行i=i+1, 再使用i的值; 而i++是先使用i的值, 再执行i=i+1。

```
int i=3,j;  
j=++i; //i的值先变成4, 再赋给j, i的值为 4
```

```
int i=3;  
printf("%d",++i); //输出 4
```

```
int i=3,j;  
j=i++; //先将 i的值3赋给 j, j 的值为 3, 然后 i 变为 4
```

```
int i=3;  
printf("%d",i++); //输出3
```

建议谨慎使用++和--运算符, 只用最简单的形式, 即i++, i--, 且把它们作为单独的表达式。

i+++j, (i++)+j, i+(++j)



算术表达式和运算符的优先级与结合性

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**C算术表达式**。

运算对象包括常量、变量、函数等。

C语言规定了运算符的**优先级**(例如先乘除后加减)，还规定了运算符的**结合性**。


在表达式求值时，先按运算符的优先级别顺序执行，当在一个运算对象两侧的运算符的优先级别相同时，则按规定的“结合方向”处理。C语言规定了各种运算符的结合方向（结合性），**“自左至右的结合方向”**又称**“左结合性”**，即运算对象先与左面的运算符结合。相反**“自右至左的结合方向”**称为**“右结合性”**。




不同类型数据间的混合运算

如果一个运算符两侧的数据类型不同，则先自动进行类型转换，使二者成为同一种类型，然后进行运算。整型、实型、字符型数据间可以进行混合运算。**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。如果字符型数据与实型数据进行运算，则将字符的ASCII代码转换为double型数据，然后进行运算。

不同类型数据间的混合运算

```
int i=3,j;  
float f=2.5;  
double d=7.5;  
printf("%lf",10+'a'+i*f-d/3);
```



程序分析

10+'a'+i*f-d/3

- ① 进行10+'a'的运算，'a'的值是整数97，运算结果为107。
- ② 由于“*”比“+”优先级高，先进行i*f的运算。先将i与f都转成double型，运算结果为7.5，double型。
- ③ 整数107与i*f的积相加。先将整数107转换成双精度数，相加结果为114.5，double型。
- ④ 进行d/3的运算，先将3转换成double型，d/3结果为2.5，double型。
- ⑤ 将10+'a'+i*f的结果114.5与d/3的商2.5相减，结果为112.0，double型。

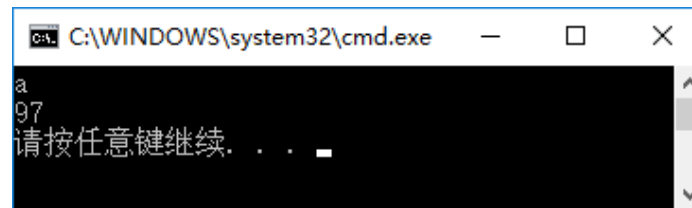
不同类型数据间的混合运算

【例3.3】 给定一个大写字母，要求用小写字母输出。

解题思路： 字符数据以ASCII码存储在内存中，形式与整数的存储形式相同。所以字符型数据和其他算术型数据之间可以互相赋值和运算。

大小写字母之间的关系是：同一个字母，用小写表示的字符的ASCII代码比用大写表示的字符的ASCII代码大32。

```
#include <stdio.h>
int main()
{
    char c1,c2;
    c1='A';           //将字符'A'的ASCII代码放到c1变量中
    c2=c1+32;         //得到字符'a'的ASCII代码，放在c2变量中
    printf("%c\n",c2); //输出c2的值，是一个字符
    printf("%d\n",c2); //输出c2的值，是字符'a'的ASCII代码
    return 0;
}
```



类型 转换

在运算时不必用户干预，系统自动进行的类型转换。

自动类型转换

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。

强制类型转换运算符

(类型名)(表达式)

(double)a 将 a 转换成double型
(int)(x+y) 将x+y的值转换成int型
(float)(5%3) 将5%3的值转换成float型
(int)x+y 只将x转换成整型，然后与y相加

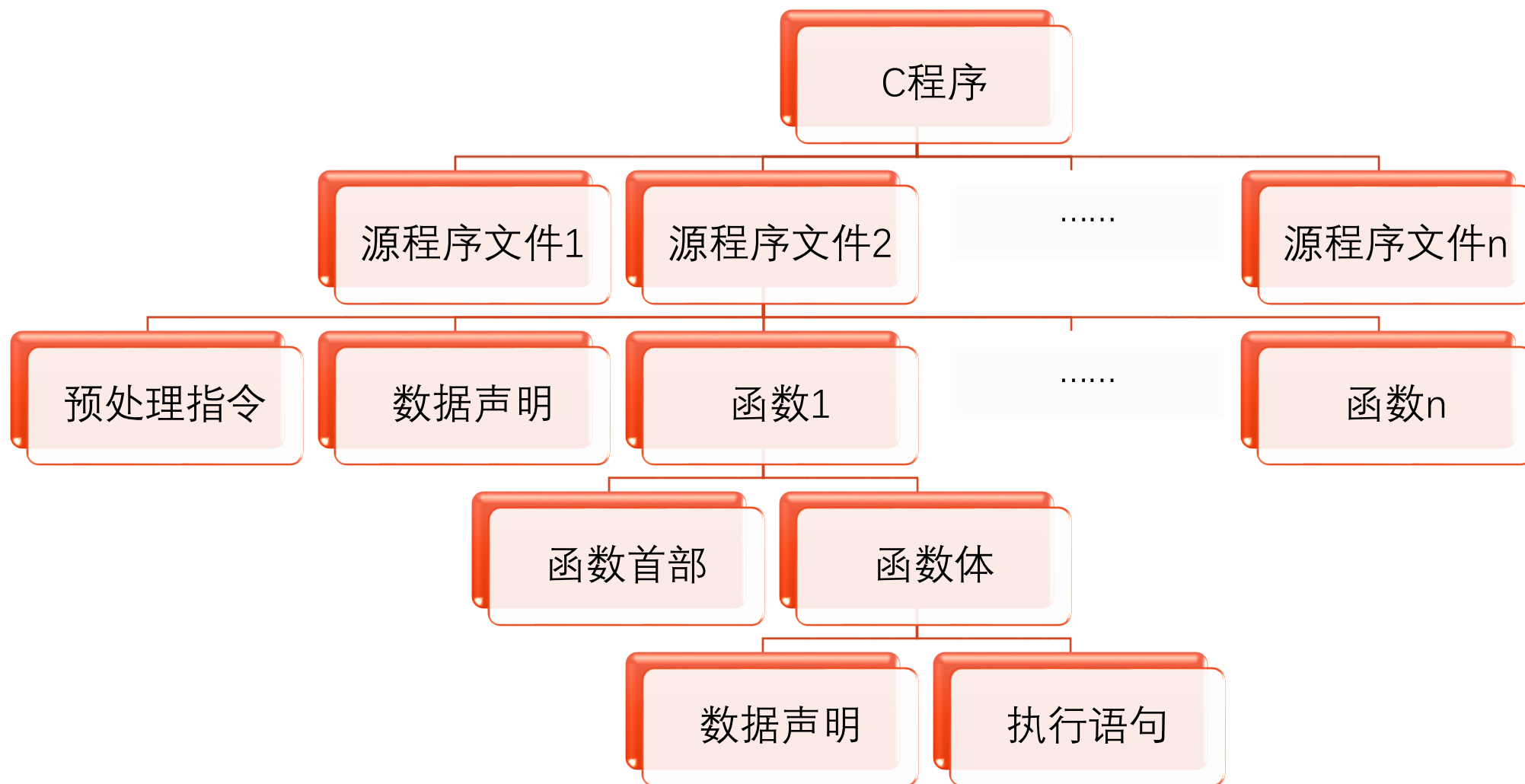
```
int a; float x,y;double b;
```

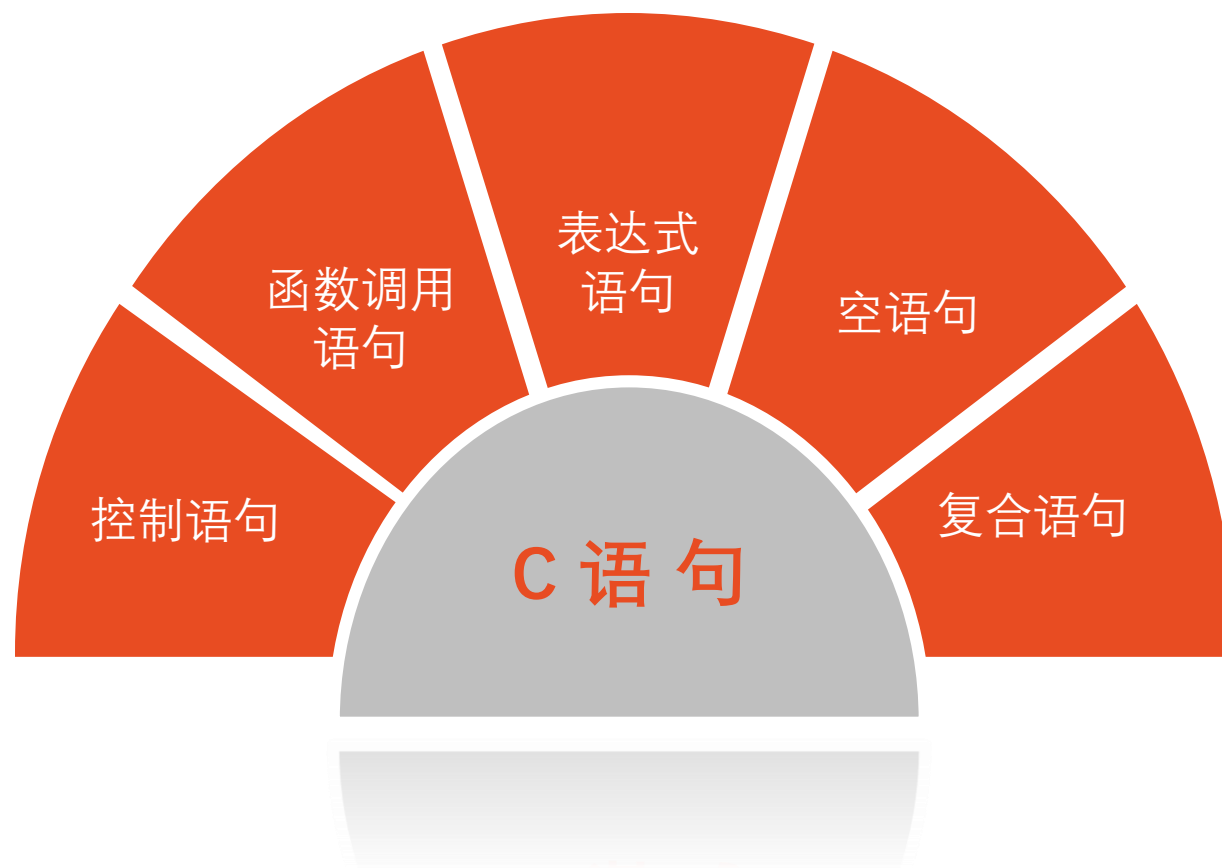
```
a=(int)x;
```

进行强制类型运算(int)x后得到一个int类型的临时值，它的值等于 x 的整数部分，把它赋给a，注意x的值和类型都未变化，仍为float型。该临时值在赋值后就不再存在了。

C 语 句

C程序结构

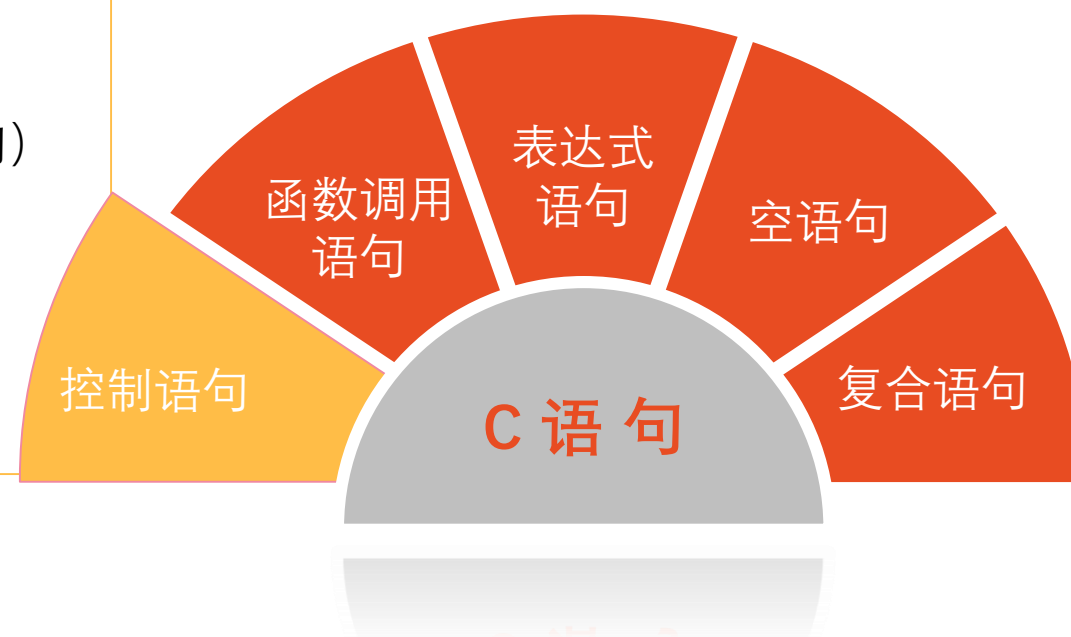




- ① if()…else… (条件语句)
- ② for()… (循环语句)
- ③ while()… (循环语句)
- ④ do…while () (循环语句)
- ⑤ continue (结束本次循环语句)
- ⑥ break (中止执行switch或循环语句)
- ⑦ switch (多分支选择语句)
- ⑧ return (从函数返回语句)
- ⑨ goto (转向语句, 在结构化程序中基本不用goto语句)

()表示括号中是一个判别条件

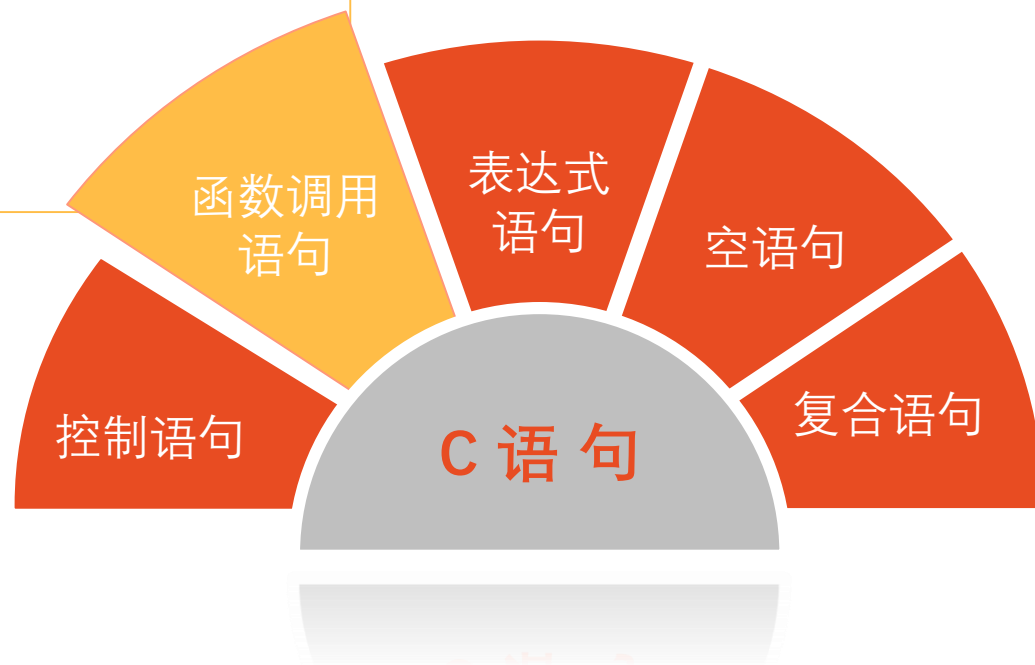
…表示内嵌的语句



函数调用语句由一个函数调用加一个分号构成。

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

其中printf("This is a C statement. ")是一个函数调用，加一个分号成为一个语句。



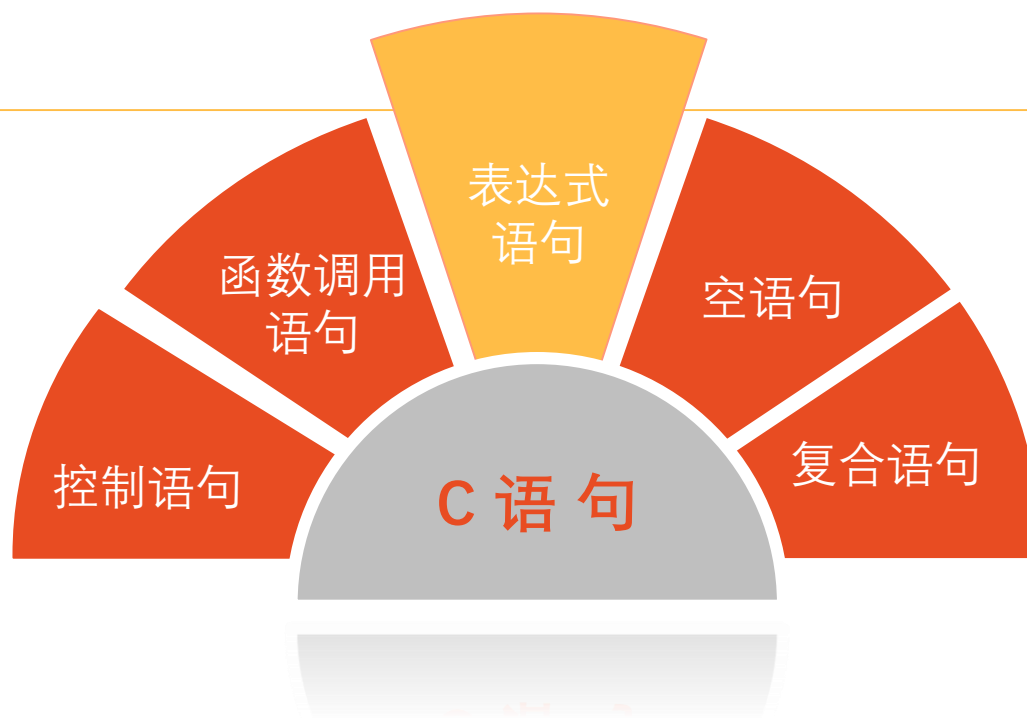
表达式语句由一个表达式加一个分号构成，最典型的是由赋值表达式构成一个赋值语句。例如：

`a=3`

是一个赋值表达式，而

`a=3;`

是一个赋值语句。

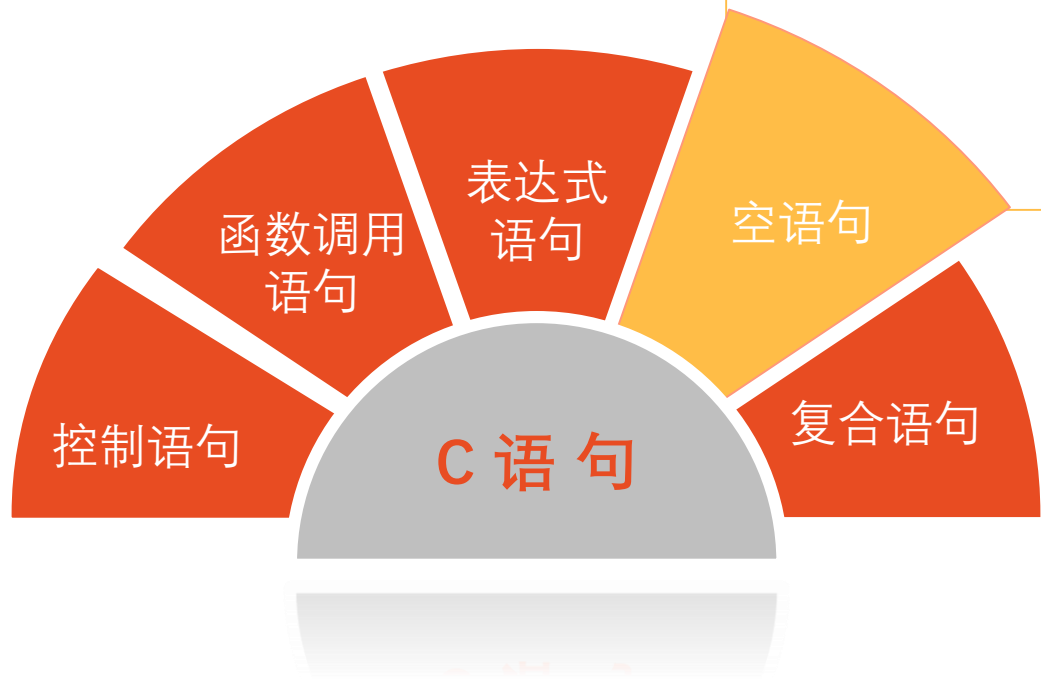


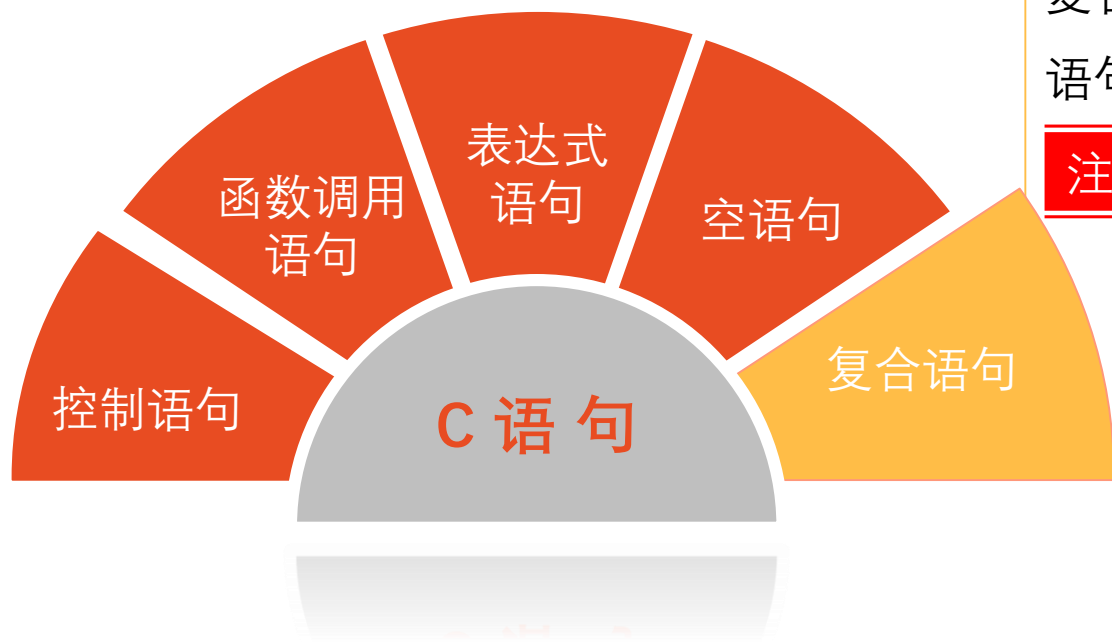
;

只有一个分号的语句即为空语句。

可以用来作为流程的转向点(流程从程序其他地方转到此语句处);

也可用来作为循环语句中的循环体（循环体是空语句，表示循环体什么也不做）。





可以用{}把一些语句和声明括起来成为复合语句(又称语句块)。

```
{  
    float pi=3.14159, r=2.5, area; //定义变量  
    area=pi*r*r;  
    printf("area=%f",area);  
}
```

复合语句常用在if语句或循环中，此时程序需要连续执行一组语句。

注意 复合语句中最后一个语句末尾的分号不能忽略不写。

赋值语句

【例3.4】给出三角形的三边长，求三角形面积。

解题思路：假设给定的三个边符合构成三角形的条件：任意两边之和大于第三边。

从数学知识已知求三角形面积的公式为： $\text{area} = \sqrt{s(s-a)(s-b)(s-c)}$ ，其中 $s = (a+b+c)/2$ 。

```
#include <stdio.h>
#include <math.h>
int main ()
{
    double a,b,c,s,area;
    a=3.67;
    b=5.43;
    c=6.21;
    s=(a+b+c)/2;
    area=sqrt(s*(s-a)*(s-b)*(s-c));
    printf("a=%f\tb=%f\t%f\n",a,b,c);
    printf("area=%f\n",area);
    return 0;
}
```

//定义各变量，均为double型
//对边长a赋值
//对边长b赋值
//对边长c赋值
//计算s
//计算area
//输出三边a,b,c的值
//输出面积area的值

 为提高精度，变量都定义为双精度类型

 sqrt函数是求平方根的函数。由于要调用数学函数库中的函数，必须在程序的开头加一条#include指令，把头文件“math.h”包含到程序中来。

 转移字符'\t'用来调整输出的位置，使输出的数据清晰、整齐、美观

赋值运算符 “=”

“=”的作用是将一个数据赋给一个变量。

例如：a=3的作用是执行一次赋值操作（或称赋值运算）。把常量3赋给变量a。

也可以将一个表达式的值赋给一个变量。

*复合赋值运算符

在赋值符=之前加上其他运算符，可以构成复合的运算符。

$a+=3$ 等价于 $a=a+3$

$x*=y+8$ 等价于 $x=x*(y+8)$

$x\%=3$ 等价于 $x=x\%3$

凡是二元（二目）运算符，都可以与赋值符一起组合成复合赋值符。

有关算术运算的复合赋值运算符有 $+=$ ， $-=$ ， $=$ ， $/=$ ， $\%=$ 。

注意

如果赋值符右边是包含若干项的表达式，则相当于它有**括号**。例如， $x\%=y+3$ 等价于 $x=x\%(y+3)$ ，切勿错写为 $x=x\%y+3$ 。

赋值表达式：变量 赋值运算符 表达式

赋值表达式的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的双重功能。

对赋值表达式求解的过程是：

①求赋值运算符右侧的“表达式”的值，②赋给赋值运算符左侧的变量。既然是一个表达式，就应该有一个值，**表达式的值等于赋值后左侧变量的值**。赋值运算符左侧应该是一个可修改值的“**左值**”(left value，简写为lvalue)。能出现在赋值运算符右侧的表达式称为“**右值**”(right value，简写为rvalue)。

注意

并不是任何形式的数据都可以作为左值的，**左值应当为存储空间并可以被赋值**。变量可以作为左值，而算术表达式 $a+b$ 就不能作为左值，常量也不能作为左值。

赋值表达式： 变量 赋值运算符 表达式

a=(b=5)

括号内的b=5是一个赋值表达式，它的值等于5。执行表达式“a=(b=5)”，就是执行b=5和a=b两个赋值表达式。因此a的值等于5，整个赋值表达式的值也等于5。赋值运算符按照“自右而左”的结合顺序，因此，(b=5)外面的括号可以不要，即a=(b=5)和a=b=5等价，都是先求b=5的值（得5），然后再赋给a。

| | |
|----------------|-------------------------|
| a=b=c=5 | 表达式值为5，a,b,c值均为5 |
| a=5+(c=6) | 表达式值为11，a值为11，c值为6 |
| a=(b=4)+(c=6) | 表达式值为10，a值为10，b等于4，c等于6 |
| a=(b=10)/(c=2) | 表达式值为5，a等于5，b等于10，c等于2 |
| a=(b=3*4) | 表达式值为12，a,b值均为12 |

赋值表达式使得赋值操作不仅可以出现在赋值语句中，而且可以出现在其他语句中(如输出语句、循环语句等)

如: printf("%d", a=b);

如果b的值为3，则输出a的值(也是表达式a=b的值)为3。在一个printf函数中完成了赋值和输出双重功能。

*赋值过程中的类型转换

如果赋值运算符两侧的类型一致，则直接进行赋值。

```
int i;  
i=234;    //直接将整数234存入变量i的存储单元中
```

如果赋值运算符两侧的类型不一致，但都是基本类型时，在赋值时要进行类型转换。类型转换是由系统自动进行的，转换的规则是：

1. 将浮点型数据（包括单、双精度）赋给整型变量时，先对浮点数取整，即舍弃小数部分，然后赋予整型变量。
2. 将整型数据赋给单、双精度变量时，数值不变，但以浮点数形式存储到变量中。
3. 将一个double型数据赋给float变量时，先将双精度数转换为单精度，即只取6~7位有效数字，存储到float型变量的4个字节中。应注意双精度数值的大小不能超出float型变量的数值范围；将一个float型数据赋给double型变量时，数值不变，在内存中以8个字节存储，有效位数扩展到15位。
4. 字符型数据赋给整型变量时，将字符的ASCII代码赋给整型变量。
5. 将一个占字节多的整型数据赋给一个占字节少的整型变量或字符变量时，只将其低字节原封不动地送到被赋值的变量（即发生“截断”）。

赋值表达式和赋值语句

C语言的赋值语句属于表达式语句，由一个赋值表达式加一个分号组成。

在一个表达式中可以包含另一个表达式。

```
if ((a=b)>0)max=a;
```

/*先进行赋值运算（将b的值赋给a），然后判断a是否大于0，如大于0，执行max=a。

请注意，在if语句中的a=b不是赋值语句，而是赋值表达式。*/

注意

区分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。

变量赋初值

可以用赋值语句对变量赋值，也可以在定义变量时对变量赋以初值。

```
int a=3;           //指定a为整型变量，初值为3； 相当于int a; a=3;
float f=3.56;      //指定f为浮点型变量，初值为3.56
char c='a';        //指定c为字符变量，初值为'a'
int a,b,c=5;       //指定a,b,c为整型变量，但只对c初始化，c的初值为5；
                   //相当于int a,b,c; c=5;
```

对几个变量赋予同一个初值：



```
int a=3,b=3,c=3;
```



```
int a=b=c=3;      //可以先定义，再用赋值语句，即int a,b,c; a=b=c=3;
```


- JAVA语言与C语言类似, 由 “; ” 结束。
- Python语句靠缩进区分

```
sum += day
```

```
leap = 0
```

```
if (year % 400 == 0) or ((year % 4 == 0) and (year % 100 != 0)):
```

```
    leap = 1
```

```
if (leap == 1) and (month > 2):
```

```
    sum += 1
```

数据的输入输出

输入输出举例



在printf函数中，在格式符f的前面加了“7.2”，表示在输出x1和x2时，指定数据占7列，其中小数占2列。优点：

- ①可以根据实际需要来输出小数的位数；
- ②如果输出多个数据，可使输出数据整齐美观。

【例3.5】求 $ax^2+bx+c=0$ 方程的根。a,b,c由键盘输入，设 $b^2-4ac > 0$ 。

解题思路：首先要知道求方程式的根的方法。由数学知识已知：如果 $b^2-4ac \geq 0$ ，则一元二次方程有两个实根： $x1 = \frac{-b+\sqrt{b^2-4ac}}{2a}$ ， $x2 = \frac{-b-\sqrt{b^2-4ac}}{2a}$ ，将分式分为两项： $p = \frac{-b}{2a}$ ， $q = \frac{\sqrt{b^2-4ac}}{2a}$ ，则 $x1=p+q$ ， $x2=p-q$ ，有了这些式子，只要知道a,b,c的值，就能顺利地求出方程的两个根。

```
#include <stdio.h>
#include<math.h>
int main()
{
    double a,b,c,disc,x1,x2,p,q;
    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("x1=%7.2f\nx2=%7.2f\n",x1,x2);
    return 0;
}
```

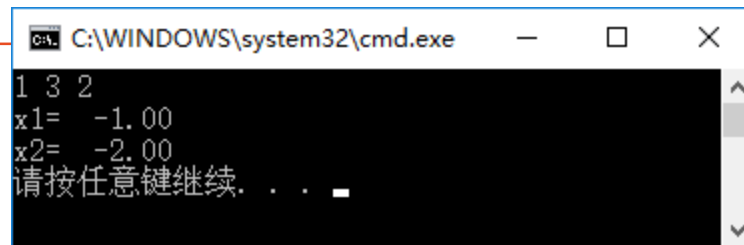
//程序中要调用求平方根函数sqrt

//disc用来存放判别式(b^2-4ac)的值

//输入双精度型变量的值要用格式声明"%lf"

//求出方程的两个根

//输出方程的两个根



```
C:\WINDOWS\system32\cmd.exe
1 3 2
x1= -1.00
x2= -2.00
请按任意键继续...
```



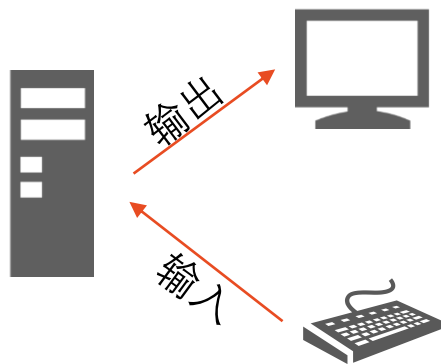
scanf函数用于输入a,b,c的值。

函数中括号内变量a,b,c的前面，要用地址符&。&a表示变量a在内存中的地址。

双引号内用%lf格式声明，表示输入的是双精度型实数。格式声明为"%lf%lf%lf"，要求输入3个双精度实数。程序运行时，输入“1 3 2”，两个数之间用空格分开。输入的虽是整数，但由于指定用%lf格式输入，因此系统会先把这3个整数转换成实数1.0,3.0,2.0，然后赋给变量a,b,c。

有关输入输出的概念

1 输入输出是以计算机主机为主体而言的



2 C语言本身不提供输入输出语句

输入和输出操作是由C标准函数库中的函数来实现的。

优点：

简化编译系统简化

增强通用性和可移植性

3 要在程序文件的开头用预处理指令#include把有关头文件放在本程序中

`#include<stdio.h>`

#include指令说明

三种形式：

`#include "c:\cpp\include\myfile.h"` ❶

`#include "myfile.h"` ❷ ❸

`#include <myfile.h>` ❸

❶ 按指定路径查找文件

❷ 源程序文件所在目录

❸ C编译系统指定的include目录

printf函数

用来向终端（或系统隐含指定的输出设备）输出若干个任意类型的数据。

printf（格式控制，输出表列）

(1) “**格式控制**”是用双引号括起来的一个字符串，称为格式控制字符串，简称格式字符串。包括：

- ① **格式声明**。格式声明由“%”和格式字符组成。作用是将输出的数据转换为指定的格式后输出。
- ② **普通字符**。普通字符即需要在输出时原样输出的字符。

(2) **输出表列**是程序需要输出的一些数据，可以是常量、变量或表达式。

```
printf("i=%d,c=%c\n", i, c)
```

普通字符

格式声明

格式控制

输出列表

printf函数——格式声明

% 附加字符 格式字符

- (1) printf函数输出时，务必注意输出对象的类型应与上述格式说明匹配，否则将会出现错误。
- (2) 除了X,E,G外，其他格式字符必须用小写字母，如%d不能写成%D。
- (3) 可以在printf函数中的格式控制字符串内包含转义字符，如\n,\t,\b,\r,\f和\377等。
- (4) 一个格式声明以“%”开头，以格式字符之一为结束，中间可以插入附加格式字符（也称修饰符）。
- (5) 如果想输出字符“%”，应该在“格式控制字符串”中用连续两个“%”表示，如：
printf(“%f%%\n”,1.0/3);

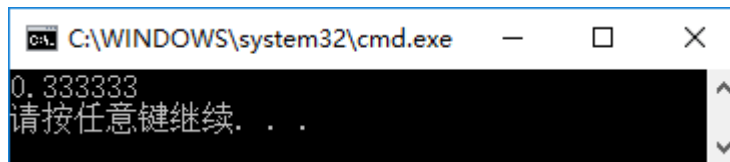
| 格式字符 | 说 明 |
|------|--|
| d,i | 以带符号的十进制形式输出整数（正数不输出符号） |
| o | 以八进制无符号形式输出整数（不输出前导符0） |
| x,X | 以十六进制无符号形式输出整数（不输出前导符0x），用x则输出十六进制数的a~f时以小写形式输出，用X时，则以大写字母输出 |
| u | 以无符号十进制形式输出整数 |
| c | 以字符形式输出，只输出一个字符 |
| s | 输出字符串 |
| f | 以小数形式输出单、双精度数，隐含输出6位小数 |
| e,E | 以指数形式输出实数，用e时指数以“e”表示(如1.2e+02)，用E时指数以“E”表示(如1.2E+02) |
| g,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。用G时，若以指数形式输出，则指数以大写表示 |

| 附加字符 | 说 明 |
|----------------|-------------------------------|
| l | 长整型整数，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 |

printf函数举例

【例3.6】用%f输出实数，只能得到6位小数。

```
#include <stdio.h>
int main()
{
    double a=1.0;
    printf("%f\n",a/3);
    return 0;
}
```

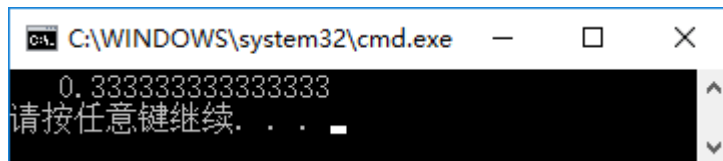


```
C:\WINDOWS\system32\cmd.exe
0.333333
请按任意键继续...
```



虽然a是双精度型，a/3的结果也是双精度型，但是用%f格式声明只能输出6位小数。

```
#include <stdio.h>
int main()
{
    double a=1.0;
    printf("%20.15f\n",a/3);
    return 0;
}
```



```
C:\WINDOWS\system32\cmd.exe
0.333333333333333
请按任意键继续...
```



一个双精度数只能保证15位有效数字的精确度，即使指定小数位数为50(如用%55.50f)，也不能保证输出的50位都是有效的数字。

注意

在用%f输出时要注意数据本身能提供的有效数字，如float型数据的存储单元只能保证6位有效数字。double型数据能保证15位有效数字。

scanf函数

用来输入数据。

scanf（格式控制，地址表列）

(1) “**格式控制**”是用双引号括起来的一个字符串，含义同printf函数。包括：

- ① **格式声明**。以%开始，以一个格式字符结束，中间可以插入附加的字符。
- ② **普通字符**。

(2) **地址表列**是由若干个地址组成的表列，可以是变量的地址，或字符串的首地址。

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

普通字符

格式声明

格式控制

地址列表

scanf函数——格式声明

% 附加字符 格式字符

- (1) scanf函数中的格式控制后面应当是**变量地址**，而不是变量名。

应与上述格式说明匹配，否则将会出现错误。
- (2)如果在格式控制字符串中除了格式声明以外还有其他字符，则在输入数据时在对应的位置上应输入与这些字符相同的字符。
- (3)在用“%c”格式声明输入字符时，空格字符和“转义字符”中的字符都作为有效字符输入。
- (4) 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)，认为该数据结束。

| 格式字符 | 说 明 |
|---------|--|
| d,i | 输入有符号的十进制整数 |
| u | 输入无符号的十进制整数 |
| o | 输入无符号的八进制整数 |
| x,X | 输入无符号的十六进制整数(大小写作用相同) |
| c | 输入单个字符 |
| s | 输入字符串，将字符串送到一个字符数组中，在输入时以非空白字符开始，以第一个空白字符结束。字符串以串结束标志'\0'作为其最后一个字符 |
| f | 输入实数，可以用小数形式或指数形式输入 |
| e,E,g,G | 与f作用相同，e与f、g可以互相替换(大小写作用相同) |

| 附加字符 | 说 明 |
|------|---|
| l | 输入长整型数据（可用%d，%o，%lx，%lu）以及double型数据（用%lf或%le） |
| h | 输入短整型数据（可用%hd，%ho，%hx） |
| 域宽 | 指定输入数据所占宽度（列数），域宽应为正整数 |
| * | 本输入项在读入后不赋给相应的变量 |



putchar函数

从计算机向显示器输出一个字符。

putchar(c)

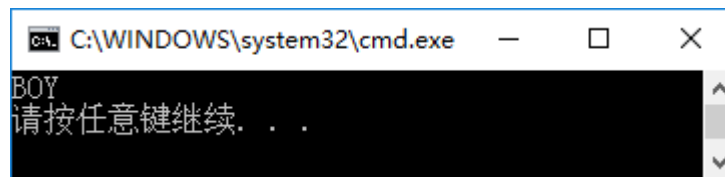
【例3.8】先后输出BOY三个字符。

解题思路：定义3个字符变量，分别赋以初值 'B'，'O'，'Y'，然后用putchar函数输出这3个字符变量的值。

```
#include <stdio.h>
int main()
{
    char a='B',b='O',c='Y'; //定义3个字符变量并初始化
    putchar(a);             //向显示器输出字符B
    putchar(b);             //向显示器输出字符O
    putchar(c);             //向显示器输出字符Y
    putchar ('\n');         //向显示器输出一个换行符
    return 0;
}
```

用putchar函数既可以输出可显示字符，也可以输出控制字符和转义字符。

putchar(c)中的c可以是字符常量、整型常量、字符变量或整型变量(其值在字符的ASCII代码范围内)。



```
#include <stdio.h>
int main()
{
    int a=66,b=79,c=89;
    putchar(a);
    putchar(b);
    putchar(c);
    putchar ('\n');
    return 0;
}
```

getchar函数

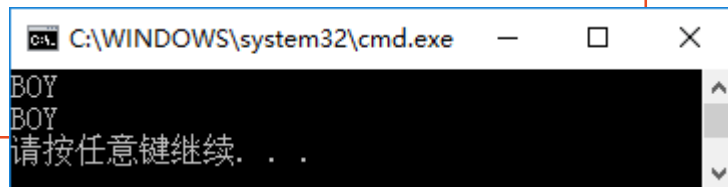
向计算机输入一个字符。

getchar()

【例3.9】从键盘输入BOY 3个字符，然后把它们输出到屏幕。

解题思路：用3个getchar函数先后从键盘向计算机输入BOY 3个字符，然后用putchar函数输出。

```
#include <stdio.h>
int main()
{
    char a,b,c; //定义字符变量a,b,c
    a=getchar();//从键盘输入一个字符，送给字符变量a
    b=getchar();//从键盘输入一个字符，送给字符变量b
    c=getchar();//从键盘输入一个字符，送给字符变量c
    putchar(a); //将变量a的值输出
    putchar(b); //将变量b的值输出
    putchar(c); //将变量c的值输出
    putchar('\n');//换行
    return 0;
}
```



函数没有参数。

函数的值就是从输入设备得到的字符。

只能接收一个字符。

如果想输入多个字符就要用多个函数。

不仅可以从输入设备获得一个可显示的字符，而且可以获得控制字符。

用getchar函数得到的字符可以赋给一个字符变量或整型变量，也可以作为表达式的一部分。如，putchar(getchar());将接收到的字符输出。

getchar函数

【例3.10】 改写例3.3程序，使之可以适用于任何大写字母。

从键盘输入一个大写字母，在显示屏上显示对应的小写字母。

解题思路： 用getchar函数从键盘读入一个大写字母，把它转换为小写字母，然后用putchar函数输出该小写字母。

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

```
int main ()
```

```
{
```

```
    char c1,c2;
```

```
    c1=getchar();    //从键盘读入一个大写字母，赋给字符变量c1
```

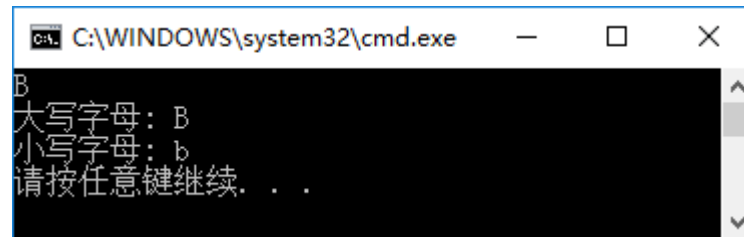
```
    c2=c1+32;    //得到对应的小写字母的ASCII代码，放在字符变量c2中
```

```
    printf("大写字母: %c\n小写字母: %c\n",c1,c2);    //输出c1,c2的值
```

```
    return 0;
```

```
}
```

用printf函数输出



```
C:\WINDOWS\system32\cmd.exe
B
大写字母: B
小写字母: b
请按任意键继续. . .
```

- Java中主要按照流(stream)的模式来实现，对数据流的主要操作都封装在java.io包中。

```
1 System.out.println(1111);//换行打印
2 System.out.print(1111);//不换行打印
3 System.out.write(2222);//字节输出
4 System.out.printf("%+8.3f\n", 3.14);//按格式输出
```

- Python使用print
 - print(a), print(a, "xxxx")