# 变量与常量、数据类型、取值范围、算数运算符与逻辑运算符

## 补充：printf函数和scanf函数的使用

一般而言c程序运行都包含输入输出，因为要进行计算，就必须给出数据，而运算的结果当然需要输出，以便人们应用。输入输出是程序中最基础的操作之一。

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。程序调用标准输入输出函数，就必须在本程序的开头用预处理指令#include把有关头文件放在本程序中，如下：

```
1 #include <stdio.h> //这是编译预处理指令
```

在这里简单介绍两个基础的c语言输入输出函数。

printf() 是 C 语言标准库函数，用于将格式化后的字符串输出到标准输出。标准输出，即标准输出文件，对应终端的屏幕。printf() 申明于头文件 stdio.h。

调用格式：**printf(格式控制, 输出表列)**

例：printf("%d,%c\n",a,b)

scanf() 是C语言标准库中的输入函数，功能是从标准输入 stdin 读取格式化输入，其一般形式为：**scanf(格式控制，地址列表)**

例：scanf("%f %f",&a,&b)

格式控制部分是一个字符串，其中格式声明以 '%' 开始，以一个格式字符结束，中间可以插入附加字符，表示属性。需要注意的是，如果格式控制字符串中除了格式声明，还含有其他的普通字符，那么在输入时应该在对应位置输入相同字符，不能改写也不能漏写，因为系统是逐个对照检查的。

| 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，不输出的无意义的0。用G时，若以指数形式输出，则指数以大写表示 |

printf()中用到的格式附加字符：

| 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 |

输出列表是程序需要输出的一些数据，可以是常量、变量或者表达式。

地址列表是由若干个地址组成的列表，如变量的地址。

例：

```c
#include <stdio.h>
int main()                          //定义主函数
{                                   //函数开始标志
    int a, b;
    scanf("%d %d", &a, &b);
    printf("%d %d\n", a, b);
    return 0;                       //函数执行完毕返回函数值0
}                                   //函数结束的标志
```

> **运行结果：**
>
> 输入：3 5
>
> 输出：3 5

程序运行后，在终端处输入“3[空格]5”（即运行结果输入后的内容），然后回车Enter就会继续运行输出第二行结果（如运行结果第二行输出后所示内容）。

在初学阶段可以多使用printf（）输出运行结果来简单地判断代码正误或者运行情况。

# 变量与常量

## 什么是变量

变量是程序可操作的存储区的名称。C 中每个变量都有特定的类型，类型决定了变量存储的大小和布局，该范围内的值都可以存储在内存中，运算符可应用于变量上。

例如下面一段代码：

```
1 #include <stdio.h>
2 int main()
3 {
4  int a = 1;
5  int b = 2;
6  int c = a + b;
7  printf("%d\n", c);
8  return 0;
9 }
```

> 运行结果：
>
> 3

程序中的“a”、“b”和“c”表示变量。 变量与常量相对，表示在程序运算过程中可以变化的量。“int”是变量的数据类型，“=”是运算符，这个符号在这里不表示“等于”，而表示“赋值”，意思是把等号右边的数据存储在名为等号左边的变量里。

如上面第4行代码的意思可以等同如下：

```
1 int a;//构造一个数据类型为整型int的变量a
2 a = 1;//给变量a赋值为1
```

“//”表示注释符号，在程序中不是一定要存在的，只是为了方便解释代码含义和理解而存在。至于上文提到的数据类型、运算符等等后文会尽量详细介绍。

## 什么是常量

- 常量是固定值，在程序执行期间不会改变。这些固定的值，又叫做字面量。
- 常量可以是任何的基本数据类型，比如整数常量、浮点常量、字符常量，或字符串字面值，也有枚举常量。
- 常量就像是常规的变量，只不过常量的值在定义后不能进行修改。

# 数据类型

## 数据存储原理

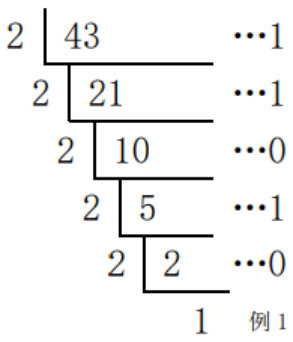
### 概述

我们人类可以很容易的分清数字与字符的区别，但是计算机并不能。因此，在每个编程语言里都会有一个叫数据类型的东西，其实就是对常用的各种数据类型进行了明确的划分。

你想让计算机进行整数运算，你就传整数类型给它，你想让他处理文字，就传字符串类型给它。

### 正数存储

在计算机中，正数采用二进制的形式存储，如5用二进制表示为：101b，其含义是：5=1*2^2+0*2^1+1*2^0；10用二进制表示为：1010b，其含义为：10=1*2^3+0*2^2+1*2^1+0*2^0。其中代表二进制的位权，其含义是这位所代表的数字大小，将二进制转化为十进制只需要将二进制每一位乘以对应位权即可，b代表这个数是个二进制数。十进制转二进制采用除二取余法：将二进制数不断除以2(保留余数)，其最后余数从下往上读即为该数所代表的二进制数（见例1）




二进制加减法遵循逢二进一 逢一退二的计数法则。

对于二进制加法来说（见例2）：

$$1b + 1b = 10b$$
$$1b + 0b = 1b$$
$$0b + 0b = 0b$$
$$0b + 1b = 1b$$

对于二进制减法来说（见例3）：

$$1b-1b=0b$$
$$1b-0b=1b$$
$$0b-0b=0b$$
$$0b-1b=-1b$$

$$\begin{array}{rr} & 1\ 1\ 0 \\ + & {\scriptstyle 1}\ 1\ 0 \\ \hline \text{例 2} & 1\ 0\ 0\ 0 \end{array}$$

$$\begin{array}{rr} & 1\ 0\ 0\ 0 \\ - & {\scriptstyle 1\ 1}\ 1\ 0 \\ \hline \text{例 3} & 1\ 1\ 0 \end{array}$$

什么决定了类型的取值范围？数据存储在内存中，1字节为8位，每位的状态都可以是0或1，以1字节的unsigned char型为例，可以表示的状态为2^8=256种，所以unsigned char型的取值范围为0~255。

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |

图表 1

不同的编译系统对整数类型的处理方法不同，其分配字节数和取值范围也不同。

## 负数存储

> 为避免歧义，我们约定二进制数前面为最高位，二进制后面为最低位。

在计算机中，我们可以直接将0表示为正数，1表示为负数直接添加在一串数字前面，这种表示方法称为原码，浮点数就采用这种方式表示。如-10(-1010b)采用原码表示为：

$$1\ 0001010b$$

其中：前面用空格单独隔开的数称为符号位，这也导致这个二进制正数最大表示范围被缩小了：其正数表示范围从减少到，负数表示范围从0扩大到。特别的，原码表示是存在正负0的。

除此之外，还有另一种表达方式：我们对负数每一位1变成0，0变成1，然后再加1。于是我们就得到一种新的二进制表示方法：补码，有符号整型采用了这种表示方式。如-10用补码表示为：

$$1\ 1110110b$$

补码的正数表示范围为，负数的表示范围为。补码被计算机广泛运用的原因是它可以将减法运算转换为加法运算，并且没有正负0。具体在这不做展开，感兴趣可以自行了解。

## 浮点数存储

浮点数是基于科学计数法的一种数据类型，在C语言中，它遵循IEEE754标准，感兴趣可以自行了解，这里不做展开。

## 字符存储

字符使用ASCII编码存储大小写字母，数字，(半角)符号。汉字，希腊字母，emoji，全角符号等采用其他的编码类型。涵盖比较广泛的是万国码(unicode)，常用类型为UTF-8，汉字除了使用UTF-8外常用的还有GB2312。

# 数据类型

## 整数类型

整型是表述整数的一种数据表示方式，它根据包不包含负数分为有符号类型(signed)和无符号类型（unsigned）。

| 名称 | 类型 | 表达范围 | 存储大小 |
| --- | --- | --- | --- |
| 整型 | int | $[-2^{31}, 2^{31}-1]$ | 4字节 |
| 短整型 | short int | $[-2^{15}, 2^{15}-1]$ | 2字节 |
| 长整型 | long int | $[-2^{31}, 2^{31}-1]$ | 4字节 |
| 双长整型 | long long int | $[-2^{63}, 2^{63}-1]$ | 8字节 |
| 字符型 | char | $[-2^{7}, 2^{7}-1]$ | 1字节 |

- 所有类型都有对应的无符号类型，在类型前面加上unsigned前缀，
- 无符号类型不可表示负数，而占用字节数不变，所以可以存放的正数范围比整形变量的正数范围扩大约一倍。其它类型以此类推。
- 需要注意的是，int类型大小由编译器自行定义，通常等于处理器字长。

## 浮点类型

浮点类型是采用科学计数法的一种数据存储类型，它可表达范围比整型大，但精度有限。浮点型转整型的舍入方式是向0舍入，即直接截取小数点后的数。

| 名称 | 类型 | 精度 | 表达范围 | 存储大小 |
| --- | --- | --- | --- | --- |
| 单精度浮点型 | float | 6有效数字 | $\pm 10^{38}$ | 4字节 |
| 双精度浮点型 | double | 15有效数字 | $\pm 10^{308}$ | 8字节 |
| 长双精度浮点型 | long double | 15有效数字 | $\pm 10^{4932}$ | 8字节 |

- 这里要提到，不同的编译系统对long double型的处理方法不同：Turbo C下对long double分配16字节，而VC++下对long double分配8字节。
- 结合实际需要，若需要计算更精确的数据，则可以使用double甚至是long double来计算以获得更精确的数据。

## 其它类型

| 名称 | 类型 | 简述 |
| --- | --- | --- |
| 枚举类型 | enum | 一种自己定义的集合，用来表示C语言中没有定义的元素，如星期、职位等。能让程序变得更加易读。 |
| 空类型 | void | 表示没有指派类型的一种类型，通常用在函数类型、参数的声明和无类型指针的声明。 |

## 派生类型

| 名称 | 定义方式 | 简述 |
| --- | --- | --- |
| 指针类型 | 类型名 *指针名 | 表示另一个元素地址的一种数据类型，它一般不单独使用，而是像int*这样使用，表示所指向的对象是一个int类型的元素。 |
| 数组类型 | 类型名 数组名[下标] | 表示一串连续变量的数据类型。 |
| 结构体类型 | struct 结构体名<br>{<br>成员表列<br>}变量表列 | 一种自定义的数据类型，使用结构体可以将多个元素具有相关信息的元素进行封装，如：名称 学号 手机号等，使用结构体封装可以让数据管理更加方便。 |
| 共用体类型 | union 共用体名<br>{<br>成员表列<br>}变量表列 | 对同一个数据的不同解读方式，如：将浮点型当作整型读取。平时不常用。 |
| 函数类型 | 类型名 函数名( ) | 表示函数返回值的类型。 |

# 运算符

## 算数运算符

下表显示了 C 语言支持的一些常用算术运算符。假设变量 A 的值为 10，变量 B 的值为 20，则：

| 运算符 | 含义 | 实例 |
| --- | --- | --- |
| + | 正号运算符（单目运算符） | +A相当于+10 |
| - | 负号运算符（单目运算符） | -A相当于-10 |
| * | 乘法运算符 | A*B相当于A和B的积 |
| / | 除法运算符 | A/B相当于A除以B的商 |
| % | 求余运算符 | A%B相当于A除以B的余数 |
| + | 加法运算符 | A+B相当于A和B的和 |
| - | 减法运算符 | A-B相当于A和B的差 |

## 关系运算符

| 运算符 | 描述 | 实例 |
| --- | --- | --- |
| == | 检查两个操作数的值是否相等，如果相等则结果为真。 | (A == B) 为假。 |
| != | 检查两个操作数的值是否相等，如果不相等则结果为真。 | (A != B) 为真。 |
| > | 检查左操作数的值是否大于右操作数的值，如果是则结果为真。 | (A > B) 为假。 |
| < | 检查左操作数的值是否小于右操作数的值，如果是则结果为真。 | (A < B) 为真。 |
| >= | 检查左操作数的值是否大于或等于右操作数的值，如果是则结果为真。 | (A >= B) 为假。 |
| <= | 检查左操作数的值是否小于或等于右操作数的值，如果是则结果为真。 | (A <= B) 为真。 |

## 逻辑运算符

逻辑运算一切非0的数值都为真，0为假。下表显示了 C 语言支持的所有关系运算符。假设变量 A 的值为 1，变量 B 的值为 0，则：

| 运算符 | 描述 | 实例 |
| --- | --- | --- |
| && | 称为逻辑与运算符。如果两个操作数都为真，则结果为真。 | (A && B) 为假。 |
| \|\| | 称为逻辑或运算符。如果两个操作数中有任意一个真，则结果为真。 | (A \|\| B) 为真。 |
| ! | 称为逻辑非运算符。用来逆转操作数的逻辑状态。如果条件为真则逻辑非运算符将使其为假。 | !(A && B) 为真。 |

## 赋值运算符

| 运算符 | 描述 | 实例 |
| --- | --- | --- |
| = | 简单的赋值运算符，把右边操作数的值赋给左边操作数 | C = A + B 将把 A + B 的值赋给 C |
| += | 加且赋值运算符，把右边操作数加上左边操作数的结果赋值给左边操作数 | C += A 相当于 C = C + A |
| -= | 减且赋值运算符，把左边操作数减去右边操作数的结果赋值给左边操作数 | C -= A 相当于 C = C - A |
| *= | 乘且赋值运算符，把右边操作数乘以左边操作数的结果赋值给左边操作数 | C *= A 相当于 C = C * A |
| /= | 除且赋值运算符，把左边操作数除以右边操作数的结果赋值给左边操作数 | C /= A 相当于 C = C / A |
| %= | 求模且赋值运算符，求两个操作数的模赋值给左边操作数 | C %= A 相当于 C = C % A |
| <<= | 左移且赋值运算符 | C <<= 2 等同于 C = C << 2 |
| >>= | 右移且赋值运算符 | C >>= 2 等同于 C = C >> 2 |
| &= | 按位与且赋值运算符 | C &= 2 等同于 C = C & 2 |
| ^= | 按位异或且赋值运算符 | C ^= 2 等同于 C = C ^ 2 |
| \|= | 按位或且赋值运算符 | C \|= 2 等同于 C = C \| 2 |

## 位运算符

位运算符作用于位，是直接对整型数据的二进制进行运算。和逻辑运算相比，位运算是每一位都进行一次逻辑运算逐位执行操作。

C语言中六种位运算符：

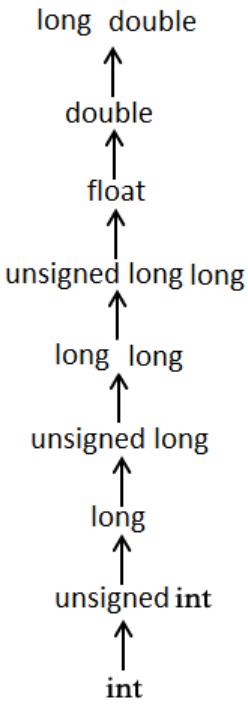
| 运算符 | 描述 | 运算过程 |
| --- | --- | --- |
| & | 按位与运算符 | 逐位与 |
| \| | 按位或运算符 | 逐位或 |
| ^ | 按位异或运算符 | 逐位异或 |
| ~ | 取反运算符 | 逐位非 |
| >> | 右移运算符 | 将整数数值逐位右移指定位数 |
| << | 左移运算符 | 将整数数值逐位左移指定位数 |

&、 | 和 ^ 的真值表如下所示：

| p | q | p & q | p \| q | p ^ q |
| --- | --- | --- | --- | --- |
| 0 | 0 | 0 | 0 | 0 |
| 0 | 1 | 0 | 1 | 1 |
| 1 | 1 | 1 | 1 | 0 |
| 1 | 0 | 0 | 1 | 1 |

## 类型转换

编译器会**自动的**将不同类型的提升到相同类型再进行运算，类型提升是计算时才会进行，类型提升按照下图：




但如果要将高级别类型转换到低级别类型则需要强制类型转换。比如要将一个浮点数类型转换为整型，则只需将(int)加在一个需要转换成整形的表达式之前。

使用一个实例来做进一步介绍。

例如：

```c
#include <stdio.h>

int main() {
int a = 3, b = 4, c = 2;
float ave;
//此处为正常运算结果
ave = (a + b + c) / 3;
printf("%f\n", ave);

//此处只有赋值的时候才会进行类型提升
//1/3使用int来进行计算的，由于整数除法向0舍入,此处结果为0
ave = (1 / 3) * (a + b + c);
printf("%f\n", ave);
//这里使用强制类型转换将1转换为浮点类型
//经过类型提升，此处计算结果为0.5f
ave = ((float)1 / 3) * (a + b + c);
printf("%f\n", ave);
c = 3;
//此处c=3 正确结果应该为3.33333
//但由于此处类型为int，计算结果向零舍入
ave = (a + b + c) / 3;
printf("%f\n", ave);
```

```
23 //使用强制类型转换，让表达式在除法运算时变成浮点型在参与运算
24 ave = (float)(a + b + c) / 3;
25 printf("%f\n", ave);
26 return 0;
27 }
```

> 运行结果：
>
> 3.000000
>
> 0.000000
>
> 3.000000
>
> 3.000000
>
> 3.333333

所以，在涉及除法等会引起舍入误差的计算，需要特别注意类型转换，比如此处1/3可以换成0.33333333f或者使用强制类型转换(float)1/3，才能避免结果出错。

## 运算符优先级

运算符的优先级确定表达式中项的组合。这会影响到一个表达式如何计算。某些运算符比其他运算符有更高的优先级，例如，乘除运算符具有比加减运算符更高的优先级。

例如 x = 7 + 3 * 2，在这里，x 被赋值为 13，而不是 20，因为运算符 * 具有比 + 更高的优先级，所以首先计算乘法 3*2，然后再加上 7。

下表将按运算符优先级从高到低列出各个运算符，具有较高优先级的运算符出现在表格的上面，具有较低优先级的运算符出现在表格的下面。在表达式中，较高优先级的运算符会优先被计算。

| 类别 | 运算符 | 结合性 |
| --- | --- | --- |
| 后缀 | ( ) [ ] -> . ++ - - | 从左到右 |
| 一元 | + - ! ~ ++ - - (type)* & sizeof | 从右到左 |
| 乘除 | * / % | 从左到右 |
| 加减 | + - | 从左到右 |
| 移位 | << >> | 从左到右 |
| 关系 | < <= > >= | 从左到右 |
| 相等 | == != | 从左到右 |
| 位与 | & | 从左到右 |

| | | |
|---|---|---|
| 位异或 | ^ | 从左到右 |
| 位或 | \| | 从左到右 |
| 逻辑与 | && | 从左到右 |
| 逻辑或 | \|\| | 从左到右 |
| 条件 | ?: | 从右到左 |
| 赋值 | = += -= *= /= %=>>= <<= &= ^= \|= | 从左到右 |
| 逗号 | , | 从左到右 |

# 思考题

- 用 `*` 构造一个对角线长 5个字符，倾斜放置的菱形（10分）

*(见洛谷习题：https://www.luogu.com.cn/problem/B2025)*

- int 类型的长度和最大值是多少？(10分)
- 若有一个数为4294967290，最低需要使用什么类型来存储？（10分）
- 定义两个 float 类型的变量 f1 和 f2，如下所示： float f1 = 3.3; float f2 = 3.8; 它们转换为 int 类型后的值分别是多少？（10分）
- 一年大概有$3.156*10^7$秒，要求输入你的年龄，然后显示该年龄一共多少秒？（10分）
- 输入两个整数 *a,b*，输出它们的和（$|a|,|b|\leqslant 10^9$）。（10分）

*(见洛谷习题：https://www.luogu.com.cn/problem/P1001)*

- 给定一个字符，用它构造一个底边长 5个字符，高 3个字符的等腰字符三角形。（20分）

*(见洛谷习题：https://www.luogu.com.cn/problem/B2005)*

- 现在需要采购一些苹果，每名同学都可以分到固定数量的苹果，并且已经知道了同学的数量，请问需要采购多少个苹果？（20分）

*(见洛谷习题：https://www.luogu.com.cn/problem/P5703)*