# 数据类型（了解底层原理，难点）

## 整型

*C语言标准并没有明确的给出每种整型的具体大小，它仅仅给出了一个大小关系建议*

*Sizeof(short) <= sizeof(int) <= sizeof(long) <= sizeof(long long)*

### 无符号整型 unsigned

**范围**

表示的就是大于等于0的正整数。

- unsigned short 无符号短整型 （2字节） 0 ~ 2^16 - 1

- unsigned int 无符号整型 （4字节） 0 ~ 2^32 - 1

- unsigned long 无符号长整型 （4字节） 0 ~ 2^32 - 1

- unsigned long long 无符号长长整形 (8字节） 0 ~ 2^64 - 1

范围的计算

例：无符号短整型

2字节- 16位 最大值 1111 1111 1111 1111 因为最大值加1等于1 \* 2^16所以最大值为1 \* 2^16 - 1

存储方式

无符号整型的存储

直接以数值的二进制形式存储。

例如：

unsigned short age = 18;

1、18(十进制) --> 10010(二进制)

2、从低位开始按位放入内存中，高位空余补零

0000 0000 0001 0010

---- ---- ---- ----

unsigned int age = 18;

0000 0000 0000 0000 0000 0000 0001 0010

---- ---- ---- ---- ---- ---- ---- ----

### 有符号整型 signed

范围

- short 短整型 (2字节） -2^15 ~ 2^15 -1

- int 整型 (4字节） -2^31 ~ 2^31 -1

- long 长整型 (4字节） -2^31 ~ 2^31 -1

- long long 长长整形 (8字节） -2^63 ~ 2^63 -1

范围的计算：由于用补码存储所以有特殊值1000 0000（一字节时）表示最小2^7(用的是排列组合 2\*2\*2……)(符号位占一位)（2的几次方是从0开始的）

存储方式

- 有符号整型内存中的存储

以补码的形式进行存储！

- 例如：

short age = 18;

0 000 0000 0001 0010 （正整数原反补相同）（相当于正数用的就是原码）

- --- ---- ---- ----

short b = -18;

- --- ---- ---- ----

1 000 0000 0001 0010 (原码)

1 111 1111 1110 1101 (反码)

1 111 1111 1110 1110 (补码)

**可以通过补码形式用加法代替减法**。

- 例如

short a = 1;

short b = -1;

short c = a + b; // 1 - 1 = 1 + -1 = 0

- --- ---- ---- ----

a : 0 000 0000 0000 0001

b : 1 111 1111 1111 1111

c : 1 0 000 0000 0000 0000

b(原) : 1 000 0000 0000 0001

b(反) : 1 111 1111 1111 1110

b(补) : 1 111 1111 1111 1111

*- 例子*

*假定有一台机器，这台机器能够存储十进制整数，但是只能存储两位十进制整数*

*65 - 39 = 26*

*= 65 - 39 + 100 - 100*

*= 65 + 100 - 39 - 100*

*= 65 + 99 + 1 - 39 - 100*

*= 65 + 99 - 39 + 1 -100*

*(机器无法表示，不要了)*

*= 65 + (99 - 39 + 1)*

*= 65 + 61 = 26*

*1111 1111*

*- 0101 1100*

*= 1010 0011*

- C语言中整型，某个整型数据类型的最大值 +1 变为最小值，最小值 -1 变为最大值，这种现象称为整型数值溢出。

## **实型**

*实型（浮点数类型）：C语言中保存小数的类型*

*C语言标准给出的建议：sizeof(float) <= sizeof(double) <= sizeof(long double)*

- float （4字节）单精度浮点数类型

- double （8字节）双精度浮点数类型

- long double （8字节）长双精度浮点数类型

- **\*浮点数类型无法做到精确存储！**

因为十进制小数转二进制时，极大概率转换不尽，

尾码在存储时要进行截断，舍弃一部分精度

浮点数类型只能在限定范围内做到近似存储

float仅能在6-7位数中做到近似存储（尽量保证精度）

- 例子

#include <stdio.h>

#include <math.h> // C语言数学库

int main(void)

{

float a = 2.45;

float b = 0.35;

float c = 2.1;

printf("a-b = %.20f\n", a-b);

printf("c = %.20f\n", c);

// |(a-b) - c| <= 0.000001

if (abs(a-b - c) <= 0.000001)

printf("相等\n");

else

printf("不相等\n");

return 0;

}

- 存储方式

- IEEE-754标准（1985年提出）

- IEEE ：美国电气和电子工程师协会

- 大家会在《计算机组成原理》详细去学

- 例子： float 存储 4.25（十进制）

0 10000001 00010000000000000000000

- -------- -----------------------

| | |

符号（1) 阶码 (8) 尾码(23)

1、将十进制小数转换成二进制

4.25（十进制） -> 100.01（二进制）

2、将二进制小数准换成科学计数法形式

100.01 -> 1.0001 \* 2 ^ 2

3、求阶码，阶码 = 2 + 127 = 129

4、求尾码，尾码 = 0001

- 符号位：正数存0 负数存1

- 阶码：阶码 = 2的次幂 + 127（移码）

- 尾码：小数点后面的就是尾码

# 常量 与 字面值（使用标准形式）

*- 常量： C语言里面的常量是指不会变化的常数或编码，储存在常量区（非RAM）*

- const 变量：只读变量，常变量（常量）

- 在定义的变量前面 加 const，表示当前变量是只读

变量，其值不允许发生修改。

- 被const修饰的变量，不允许进行赋值。

- **被const修饰的变量，必须进行初始化。**

*-附： 与常量的区别：只读变量在定义时会被分配一块内存，但是这块内存中的值不能修改；常量只是一个静态的值，不用分配内存*

- 字面值：

*字面值是指在程序中无需变量保存，可直接表示为一个具体的数字或字符串的值。*

- 整型字面值

- **\*整型字面默认数据类型均为 int**

- 可以通过在字面值后面加后缀改变字面值类型

- U ：unsigned

- L ：long

- **整型字面值无法表示short短整型**

- 字面值无法表示二进制

- **十六进制字面值 不区分大小写** 0X1F 0x1f

- 例如： int最大值 0x7fffffff

- 12L : (long)

- 12LL : (long long)

- 12U : (unsigned int)

- 12ULL : (unsigned long long)

- 12 : （int）十进制正整数

- -12 : （int）十进制负整数

- 0x12 : （int）十六进制整数

- 012 : （int）八进制整数

- 浮点数类型字面值

- **\*小数字面值默认的类型都是 double类型**

- 可以通过加后缀更改小数字面值的类型

- F : float

- L : long

- 3.14L : (long double)

- 3.14F : (float)

- 3.14 : (double)

- 3. : (double) 3.0

- .34 : (double) 0.34

- 1.23E-2 : (double)1.23 \* 10 ^ -2 = 0.0123

- 1e-6 : (double) 1\*10^-6 = 0.000001

**fabs(a-b) <= 1e-6**

- 字符型字面

- 字符型字面值的类型是 char 类型

- **字符型字面值需要用单引号引起来，字符型字面值仅**

**能表示一个ASCII码表字符**

- 字符型字面值中会有一些特殊字符，无法直接表示的

字符，通常以 \ 开头，进行转义，下面是常见的

**转义字符**

- '\n' : 换行符

- '\t' : 制表符

- '\'' : 单引号 '

- '\"' : 双引号 "

- '\r' : 回车符

- '\a' : 响铃符

- '\ddd' : 通过3位八进制数，表示某个字符

就是3位八进制数的十进制对应的

ASCII码代表的字符

- '\101' : 'A'

101(八进制)->65(十进制)->'A'

- '\061' : '1'

061(八进制)->49(十进制)->'1'

- '\61' : '1'

- '\xhh' : 通过2位十六进制数，表示某个字符

- '\x61' : 'a'

61(十六进制) -> 97(十进制) -> 'a'

- 'A' : char

- '9' : char

- '\n' : char 换行符

- 字符串字面值

- C语言中字符串字面值使用**双引号**引起来

- **字符串字面值末尾会自动添加一个**

**结束符（'\0'，ASCII码是 0）**

- **\*字符个数：**字符串字面值一共包含的字符数量

- **\*字符串长度：**字符串中第一个'\0'之前字符的数量

- "Hello World" : 字符个数:12 字符串长度:11

- "" : 空字符串 字符个数:1 字符串长度:0

- "a\x61b" : 字符个数:4 字符串长度:3

- "a\0b" : 字符个数:4 字符串长度:1

- "a\061b" : 字符个数:4 字符串长度:3

**- "ab""cd" : "abcd" 字符个数:5 字符串长度:4**

**两个字符串字面值相邻时，会自动拼接到一起**

**- 'A' 与 "A" 的区别**

'A' : 字符字面值，只有一个字符

"A" : 字符串字面值，有两个字符

# 输入 与 输出（使用标准形式）

*C语言本身不提供输入输出的相应功能，C语言中的输入输出均由"C语言标准输入输 出库"来提供*

- #include <stdio.h> // 引入输入输出库

## - printf ：格式化输出函数

- printf(格式化字符串, 参数列表)

- 格式化字符串 : 包含格式转换说明符的字符串

- 参数列表 : 往格式化字符串中替换的参数

int a = 13, b = 15;

printf("%d + %d = %d\n", a, b, a+b);

**printf(格式化字符串, 参数列表)**

**- 格式化字符串 : 包含格式转换说明符的字符串**

**- 参数列表 : 往格式化字符串中替换的参数**

**- 格式转换说明符**：

**%[标志字符][最小宽度说明][精度说明][长度修正说明符]<转换操作符>**

**- %d : (\*)int**

- %ld : long

**- %u : (\*)unsigned int**

- %lu : unsigned long

- %x %X : 整型的十六进制形式(0x1f)

- %o : 整型的八进制形式(017)

**- %c : (\*)char**

**- %f : (\*)float / double**

- %lf : long double

- %e : 以科学计数法形式输出

- %s : 字符串

- %p : 指针地址

- 格式转换说明符输出类型 一定要与后面的参数类型相匹配

如果不匹配，**结果是未定义的（难以预料）**

- 格式字符串中格式转换说明符数量 **一定亚于后面**的参数数

量相匹配，如果不匹配，结果是未定义

- 控制浮点数输出时精度的位数

float num = 3.1415926F;

printf("%6.2f\n", num);

- 控制输出的最小宽度

int a = 12;

int b = 321;

int c = 1024;

printf("%4d\n", a);

printf("%4d\n", b);

printf("%4d\n", c);

- 如果实际输出的类型和格式转换说明符的类型不匹配

，那么输出结果是未定义的

## - scanf ：格式化输入函数

- 按照指定格式在输入数据中心解析所需数据

- scanf函数需要知道保存数据的变量的地址，

所以在使用时需要传递变量的地址

- C语言中通过在变量前面加 & 取地址符来获取

变量地址

- scanf在匹配输入格式时，是匹配不到 \n

scanf中**不要去写 \n**

- scanf在匹配格式时自动跳过空格、换行、制表符

- **scanf()执行后会返回一个整数值，表示成功读取**

**到几个值**

- scanf中格式字符串里面的格式转换说明符，绝大多

数都跟printf中的类似，唯一需要注意的是

**double ： printf中用 %f**

**scanf中用 %lf**

- putchar / getchar

putchar() 在命令行输出一个字符

getchar() 在命令行读取一个字符

- **getchar() 在读取时不会忽略任何字符**