数据类型

2020年7月13日 19:52

C语言的类型

- 整数
  - char、short、int、long、long long
- 浮点数
  - float、double、long double
- 逻辑
  - bool
- 指针
- 自定义类型

蓝色的是C99的类型

类型有何不同

- 类型名称：int、long、double
- 输入输出时的格式化：%d、%ld、%lf
- 所表达的数的范围：char < short < int < float < double
- 内存中所占据的大小：1个字节到16个字节
- 内存中的表达形式：二进制数（补码）、编码

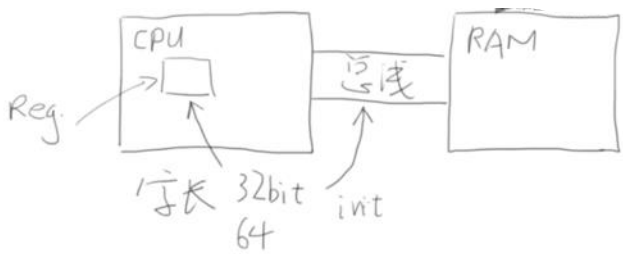
sizeof

- 是一个运算符，给出某个类型或变量在内存中所占据的字节数
  - sizeof(int)
  - sizeof(i)
- 是静态运算符，它的结果在编译时刻就决定了
- 不要在sizeof的括号里做运算，这些运算不会做的

```
MOOC:cc\ $gcc test.c -o test
MOOC:cc\ $./test
sizeof(char)=1
sizeof(short)=2
sizeof(int)=4
sizeof(long)=8
sizeof(long long)=8
MOOC:cc\ $gcc test.c -o test -m32
MOOC:cc\ $./test
sizeof(char)=1
sizeof(short)=2
sizeof(int)=4
sizeof(long)=4
sizeof(long long)=8
```

整数

- char：1字节（8比特）
- short：2字节
- int：取决于编译器（CPU），通常的意义是“1个字”
- long：取决于编译器（CPU），通常的意义是“1个字”
- long long：8字节



int就是用来表达寄存器的

整数越界

- 整数是以纯二进制方式进行计算的，所以：
  - 11111111 + 1 —> 100000000 —> 0
  - 01111111 + 1 —> 10000000 —> -128
  - 10000000 - 1 —> 01111111 —> 127

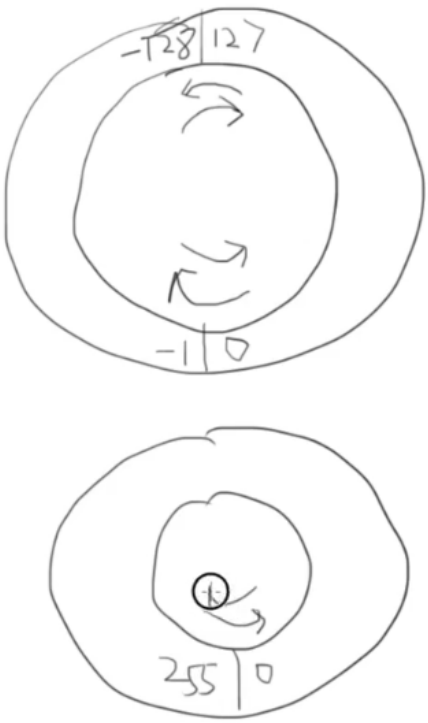
1台计算机的字长指的是寄存器是多少宽，也就是说这个寄存器是几个bit。比如说，当我们说寄存器是32个bit的，每一个寄存器可以表达32个bit的数据，同时也是在说，CPU和RAM之间在总线上传递数据的时候，每一次的传递是32个bit，也就是说，当要从内存RAM取数据到CPU里面去，每一次就会要取32个bit。

除了32，现在更常见的是64个bit。

字长在C语言中，反映为int。int表达的是1个寄存器的大小。所以，在不同的平台、CPU上面，int会不一样大。

```c
{
    char c = 255;
    int i = 255;
    printf("c=%d,i=%d\n", c, i);
    //  11111111
    //  00000000 00000000 00000000 11111111
    return 0;
}
```

```
c=-1,i=255
```




```c
#include <stdio.h>

int main()
{
    char c = -1;
    int i = -1;
    printf("c=%u,i=%u\n", c, i);
    return 0;
}
```

```
c=4294967295,i=4294967295
[Finished in 0.1s]
```

-1的二进制全是1，所有小于int的变量传给printf时会转化为int,y

- 在整形类型前加上unsigned使得它们成为无符号的整数
- 内部的二进制表达没变，变的是如何看待它们
  - 如何输出
- 11111111
  - 对于char，是-1
  - 对于unsigned char，是255
- 如果一个字面量常数想要表达自己是unsigned，可以在后面加u或U
  - 255U
- 用l或L表示long(long)
- *unsigned的初衷并非扩展数能表达的范围，而是为了做纯二进制运算，主要是为了移位

```c
#include <stdio.h>

int main()
{
  int a=0,b=0;
  while (++a>0)
    ;
  printf("int数据类型最大数是:%d\n",a-1);
  b++;
  while ((a=a/10)!=0)
  {
  b++;
  }
  printf("int数据类型最大的数的数位是:%d",b);
  return 0;
}
```

```
nt数据类型最大数是:2147483647
nt数据类型最大的数的数位是:10[Finished in 8.9s]
```

unsigned int 最大范围

```c
#include <stdio.h>

int main(void)
{
    unsigned int a = 1;
    while ( a!=0 )
        a++;
    printf("unsigned int max = %u\n", a-1);
    return 0;
}
```

整数的输入输出

- 只有两种形式：int或long long
  - %d：int
  - %u：unsigned
  - %ld：long long

-1的二进制全是1，所有小于int的变量传给printf时会转化为int,y有符号数扩展到所有位都是1，然后表示为unsigned输出。
如果不是unsigned，输出-1，-1
计算机中的数，你以不同的方式看待它，就会得出不一样的结果。这和数字在计算机内部没有关系，取决于你怎么看待它。

- %lu：unsigned long long

选择整数类型

- 为什么整数要有那么多种?
  - 为了准确表达内存，做底层程序的需要
- 没有特殊需要，就选择int
  - 现在的CPU的字长普遍是32位或64位，一次内存读写就是一个int，一次计算也是一个int，选择更短的类型不会更快，甚至可能更慢
  - * 现代的编译器一般会设计内存对齐，所以更短的类型实际在内存中有可能也占据一个int的大小（虽然sizeof告诉你更小）
- unsigned与否只是输出的不同，内部计算是一样的

8进制和16进制

- 一个以0开始的数字字面量是8进制
- 一个以0x开始的数字字面量是16进制
- %o用于8进制，%x用于16进制
- 8进制和16进制只是如何把数字表达为字符串，与内部如何表达数字无关

```c
#include <stdio.h>

int main()
{
    char c = 012;
    int i = 0x12;
    printf("c=%d,i=%d\n", c, i);
    return 0;
}
```

```
c=10,i=18
```

```c
#include <stdio.h>

int main()
{
    char c = 012;
    int i = 0x12;
    printf("c=%o,i=%x\n", c, i);
    return 0;
}
```

```
c=12,i=12
```

```c
#include <stdio.h>

int main()
{
    char c = 012;
    int i = 0x1A;
    printf("c=%o,i=%X\n", c, i);
    return 0;
}
```

小写x输出小写字母，大写X输出大写字母

浮点类型

类型	字长	范围	有效数字
float	32	±(1.20x10 0,±inf,nan	7
double	64	±(2.2x10 0,±inf,nan	15

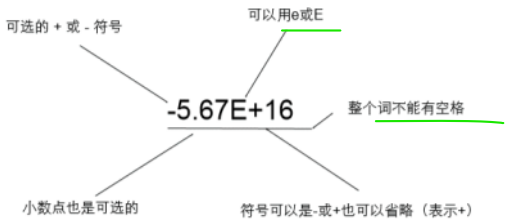
inf：正负的无穷大
nan：表示不是一个有效的数字
%e：输出科学计数法

浮点的输入输出

类型	scanf	printf
float	%f	%f, %e
double	%lf	%f,%e

```c
#include <stdio.h>

int main()
{
    double ff = 1234.56789;
    printf("%E,%f\n", ff,ff);

    return 0;
}
```

```c
#include <stdio.h>

int main()
{
    double ff = 1E-10;
    printf("%E,%f\n", ff,ff);

    return 0;
}
```



输出精度

- 在%和f之间加上.n可以指定输出小数点后几位，这样的输出是做4舍5入的
  - printf("%.3f\n", -0.0049);
  - printf("%.30f\n", -0.0049);
  - printf("%.3f\n", -0.00049);

```
1.234568E+03,1234.567890
```

```
1.000000E-10,0.000000
```

```c
#include <stdio.h>

int main()
{
    printf("%.3f\n", -0.0049);
    printf("%.30f\n", -0.0049);
    printf("%.3f\n", -0.00049);

    return 0;
}
```

浮点数的表示是离散的，有些不能表达的数字靠近什么，就是什么

浮点数有误差！

```
-0.005
-0.004899999999999999841793218991
-0.000
```

```c
#include <stdio.h>

int main()
{
    printf("%d\n", 12.0/0.0);
    printf("%f\n", -12.0/0.0);
    printf("%f\n", 0.0/0.0);

    return 0;
}
```

```
inf
-inf
nan
```

```c
#include <stdio.h>

int main()
{
    printf("%d\n", 12/0);

    return 0;
}
```

```
/Users/wengkai/cc/test.c:5:19: warning: d
        printf("%d\n", 12/0);
                       ~~
1 warning generated.
```

- printf输出inf表示超过范围的浮点数：±∞
- printf输出nan表示不存在的浮点数

无穷大可以用浮点数表达，但是不能用整数表达

浮点数表达中有三种特殊的值表示inf,nan

浮点运算的精度

```c
float a, b, c;

a = 1.345f;
b = 1.123f;
c = a + b;
if (c == 2.468)
   printf("相等\n");
else
   printf("不相等! c=%.10f, 或%f\n",c,c);
```

- 带小数点的字面量是double而非float
- float需要用f或F后缀来表明身份

Android计算器低级错误？都是二进制惹的祸！

http://www.guokr.com/article/27173/

- f1 == f2可能失败
- fabs(f1-f2) < 1e-12

选择浮点类型

- 如果没有特殊需要，只使用double
- 现代CPU能直接对double做硬件运算，性能不会比float差，在64位的机器上，数据存储的速度也不比float慢

字符类型

- char是一种整数，也是一种特殊的类型：字符。这是因为：
  - 用单引号表示的字符字面量：'a', '1'
  - ''也是一个字符
  - printf和scanf里用%c来输入输出字符

字符的输入输出

- 如何输入'1'这个字符给char c?
  - scanf("%c", &c);—>1
  - scanf("%d", &i); c=i; —>49
- '1'的ASCII编码是49，所以当c==49时，它代表'1'
  - printf("%i %c\n", c,c);
- 一个49各自表述！

```c
#include <stdio.h>

int main()
{

    char c;
    char d;
    c = 1;
    d = '1';
    if ( c == d ) {
        printf("相等\n");
    } else {
        printf("不相等\n");
    }
    printf("c=%d\n", c);
    printf("d=%d\n", d);

    return 0;
}
```

```
不相等
c=1
d=49
```

```c
#include <stdio.h>

int main()
{
    if ( 49 == '1' ) {
        printf("OK");
    }
    // printf("c='%c'\n", c);

    return 0;
}
```

```
OK[Finished in 0.2s]
```

```c
#include <stdio.h>

int main()
{

    char c;
    scanf("%c", &c);
    printf("c=%d\n", c);
    printf("c='%c'\n", c);

    return 0;
}
```

```
MOOC:cc\ $gcc test.c
MOOC:cc\ $./a.out
1
c=49
c='1'
MOOC:cc\ $more test.c
#include <stdio.h>

int main()
{

    char c;
    scanf("%c", &c);
    printf("c=%d\n", c);
    printf("c='%c'\n", c);

    return 0;
}
```

```c
int main()
{
        int i;
        char c;
        scanf("%d", &i);
        c = i;
        printf("c=%d\n", c);
        printf("c='%c'\n", c);

        return 0;
}
```

```
MOOC:cc\ $./a.out
1
c=1
c=''
MOOC:cc\ $./a.out
49
c=49
c='1'
```

- 有何不同?
  - scanf("%d %c", &i, &c);
  - scanf("%d%c", &i, &c);

```c
#include <stdio.h>

int main()
{
    int i;
    char c;
    scanf("%d %c", &i, &c);
    printf("i=%d, c=%d, c='%c'\n",i,c,c);

    return 0;
}
```

```
MOOC:cc\ $gcc test.c
MOOC:cc\ $./a.out
12 1
i=12, c=49, c='1'
MOOC:cc\ $./a.out
12a
i=12, c=97, c='a'
MOOC:cc\ $./a.out
12  1
i=12, c=49, c='1'
MOOC:cc\ $gcc test.c
MOOC:cc\ $./a.out
12 1
i=12, c=32, c=' '
MOOC:cc\ $./a.out
12a
i=12, c=97, c='a'
MOOC:cc\ $./a.out
12  1
i=12, c=32, c=' '
MOOC:cc\ $
```

有空格的话，整数后面如果有空格，要把空格读到，
如果直接加上了字符，不读字符，读整数结束
没有空格，整数后面的一个是什么，字符读什么

字符计算

```
char c = 'A';
c++;
printf("%c\n", c);
```

```
int i = 'Z' - 'A';
printf("%d\n", i);
```

- 一个字符加一个数字得到ASCII码表中那个数之后的字符
- 两个字符的减，得到它们在表中的距离

大小写转换

- 字母在ASCII表中是顺序排列的
- 大写字母和小写字母是分开排列的，并不在一起
- 'a'-'A'可以得到两段之间的距离，于是
  - a+'a'-'A"可以把一个大写字母变成小写字母，而
  - a+'A'-'a'可以把一个小写字母变成大写字母

逃逸字符

- 用来表达无法印出来的控制字符或特殊字

```
printf("请分别输入身高的英尺和英寸，"
    "如输入\"5 7\"表示5英尺7英寸：");
```

个字符

字符	意义	字符	意义
\b	回退一格	\"	双引号
\t	到下一个表格位	\'	单引号
\n	换行	\\	反斜杠本身
\r	回车		

```
#include <stdio.h>

int main()
{
    printf("123\b\n456\n");

    return 0;
}
```

```
123
456
```

```
int main()
{
        printf("123\b\n456\n");

        return 0;
}
MOOC:cc\ $gcc test.c
MOOC:cc\ $./a.out
123
456
```

程序运行时有一个黑色的窗口，它是别人写的程序，叫做终端（shell），会再背后执行我们的程序。我们的键盘，眼睛是和shell联系起来的。
程序和shell打交道，shell负责翻译，不是那么诚实的，键盘回退了，shell要处理，转义字符也要处理。不同的shell会对转义字符做出不一样的反应。

```
#include <stdio.h>

int main()
{
    printf("123\bA\n456\n");

    return 0;
}
```

```
MOOC:cc\ $./a.out
12A
456
```

回去一格，有输出，有东西会替换掉，不输出东西，什么也没有

自动类型转换

- 当运算符的两边出现不一致的类型时，会自动转换成较大的类型
  - 大的意思是能表达的数的范围更大
  - char —> short —> int —> long —> long long
  - int —> float —> double
- 对于printf，任何小于int的类型会被转换成int；float会被转换成double
- 但是scanf不会，要输入short，需要%hd

强制类型转换

- 要把一个量强制转换成另一个类型（通常是较小的类型），需要：
  - (类型)值
- 比如：
  - (int)10.2
  - (short)32
- 注意这时候的安全性，小的变量不总能表达大的量
  - (short)32768

- 只是从那个变量计算出了一个新的类型的值，它并不改变那个变量，无论是值还是类型都不改变

强制类型转换

```
double a = 1.0;
double b = 2.0;
int i = (int)a / b;
```
int i = (int)(a / b);

- 强制类型转换的优先级高于四则运算

```
int a = 5;
int b = 6;
double d = (double)(a / b);
```

强制类型转换不会改变值，是在计算一个新的值！

```
1  #include <stdio.h>
2
3  int main()
4  {
5      int i=32768;
6      short s = (short)i;
7      printf("%d\n", i);
8
9      return 0;
10 }
11
```

```
1  #include <stdio.h>
2
3  int main()
4  {
5      printf("%d\n", (short)32768);
6
7      return 0;
8  }
9
```

32768

-32768

bool

- #include <stdbool.h>
- 之后就可以使用bool和true、false

```
1  #include <stdio.h>
2  #include <stdbool.h>
3
4  int main()
5  {
6      bool b = 6>5;
7      bool t = true;
8      t = 2;
9      printf("")
10     return 0;
11 }
12
```

bool的运算

- bool实际上还是以int的手段实现的，所以可以当作int来计算
- 也只能当作int来输入输出

```
1  #include <stdio.h>
2  #include <stdbool.h>
3
4  int main()
5  {
6      bool b = 6>5;
7      bool t = true;
8      t = 2;
9      printf("%d\n", b);
10
11     return 0;
12 }
13
```

无法printf出true和false

逻辑运算

- 逻辑运算是对逻辑量进行的运算，结果只有0或1
- 逻辑量是关系运算或逻辑运算的结果

运算符	描述	示例	结果

```
[Finished in 0.1s]
```

```
1
```

运算符	描述	示例	结果
!	逻辑非	!a	如果a是true结果就是false，如果a是false结果就是true
&&	逻辑与	a && b	如果a和b都是true，结果就是true；否则就是false
||	逻辑或	a || b	如果a和b有一个是true，结果为true；两个都是false，结果为false

`x > 4 && x < 6`

- 如果要表达数学中的区间，如：x ∈(4,6)或x ∈[4,6]，应该如何写C的表达式？

像 4 < x < 6这样的式子，不是C能正确计算的式子，因为4 < x的结果是一个逻辑值（0或1）

- 如何判断一个字符c是否是大写字母？
  - c >= 'A' && c<= 'Z'

优先级	运算符	结合性
1	()	从左到右
2	! + - ++ --	从右到左（单目的+和-）
3	* / %	从左到右
4	+ -	从左到右
5	< <= > >=	从左到右
6	== !=	从左到右
7	&&	从左到右
8	||	从左到右
9	= += -= *= /= %=	从右到左

条件运算符

- count = (count > 20) ? count -10 : count +10;
- 条件、条件满足时的值和条件不满足时的值

```
if ( count >20 )
  count = count-10;
else
  count = count+10;
```

嵌套条件表达式

- count = (count > 20) ? (cout < 50) ? count - 10 : count - 5 : (count < 10) ? count + 10 : count + 5;
- 条件运算符
  - w < x ? x + w : x < y ? x : y

我们不希望你使用嵌套的条件表达式

- ! age<20

!优先级高，先做！age

优先级

- ! > && > ||
  - !done && (count > MAX)

短路

- 逻辑运算是自左向右进行的，如果左边的结果已经能够决定结果了，就不会做右边的计算
  - a==6 && b==1
  - a==6 && b+=1

不要把赋值，包括复合赋值组合进表达式！

- 对于&&，左边是false时就不做右边了
- 对于||，左边是true时就不做右边了

优先级

- 条件运算符的优先级高于赋值运算符，但是低于其他运算符

```
m<n ? x : a+5
a++ >= 1 && b-- > 2 ? a : b
x=3 * a > 5 ? 5 : 20
```

逗号运算符

- 逗号用来连接两个表达式，并以其右边的表达式的值作为它的结果。逗号的优先级是所有的运算符中最低的，所以它两边的表达式会先计算；逗号的组合关系是自左向右，所以左边的表达式会先计算，而右边的表达式的值就留下来作为逗号运算的结果。

```c
#include <stdio.h>

int main()
{
    int i;
    i = 3+4,5+6;

    printf("%d\n", i);

    return 0;
}
```

先算i=3+4=7,5+6之后算，但是没有东西放5+6

i=7。

```c
#include <stdio.h>

int main()
{
    int i;
    i = (3+4,5+6);

    printf("%d\n", i);

    return 0;
}
```

3+4没有用到。放了括号之后，一个括号里面是一个表达式，逗号表达式的结果是逗号后面的东西。

在for中使用，

```
/Users/wengkai/cc/test.c:6:11:
        i = 3+4,5+6;
              ~~
1 warning generated.
[Finished in 0.1s]
```

- for (i=0, j=10; i<j; i++, j- -) ……

```
/Users/wengkai/cc/test.c:6:
        i = (3+4,5+6);
             ~~
1 warning generated.
11
```