数据存储

1 数据类型存储

类型的意义：1）使用类型决定了开辟内存的大小（内存大小决定了使用范围）2）如何看待内存的视角。

整型家族：char(以ASCII码存储的); int; short; long; long long.对于有符号和无符号类型（unsigned、signed）定义更精确，比如正数。

浮点型家族：float、double

构造类型：自定义类型，如结构体类型、数组类型、enum枚举类型、union联合体类型等。

指针类型：int\* p

空类型：void函数返回类型void test(void)，指针中会用到void\* p。

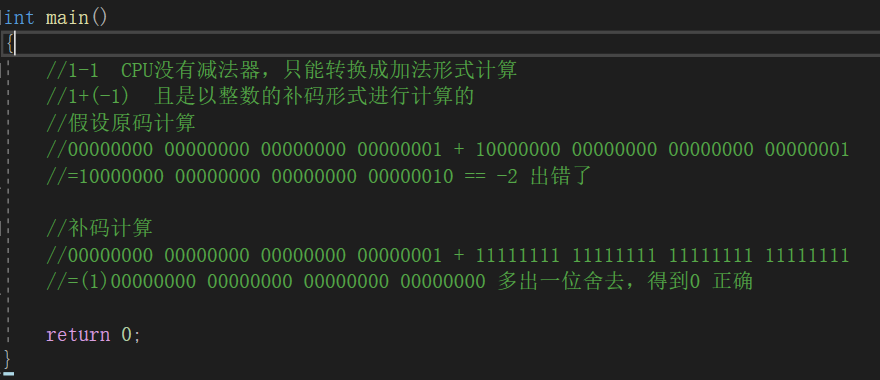
1.1 整数类型存储

整数在内存中是以二进制补码形式存储的。

正整数：符号位为0，原码、反码、补码形式相同。

负整数：符号位为1，反码=原码符号位不变其余位取反，补码=反码+1，而要知道负整数的原码，则原码=补码-1再取反（符号位不变）。

在计算机系统中，整数一律用补码来表示和存储。原因在于，使用补码可以将符号位和数值域统一处理（符号位带入计算不影响）。同时加减法也可以统一处理（CPU只有加法器）。此外，补码原码相互转换，其运算过程是相同的（都可以用反码加一转换），不需要额外的硬件电路。



（1）大端小端存储

大端字节序：把数据的高位放到低地址处、数据的低位放到高地址处的存放顺序。

小端字节序：把数据的低位放到低地址处、数据的高位存储到高地址处的存储顺序。VS2019是小端字节序存储的。大于一个字节数据的存储需要配置一个顺序。

例：写一个小程序判断编译器是小端还是大端存储。

#include<stdio.h>

int main()

{

int a = 1;

char\* p = (char\*)&a;//获得整型数据的首个地址，赋值为字符型指针，则得到一个字节的地址

if (\*p == 1)//判断一个字节的数据为01 还是00

{

printf("小端\n");

}

else

{

printf("大端\n");

}

return 0;

}

**练习：**1）读程序

#include<stdio.h>

int main()

{

char a = -1;//32个1 截断为8个1 字符要加单引号‘1’，这里没加上，说明只是数字。

signed char b = -1;//32个1 截断为8个1

unsigned char c = -1;//11111111 255 无符号数则不认为最高位是符号位

printf("a=%d, b=%d, c=%d\n", a, b, c);//整型提升补符号位 -1 -1 255

return 0;

}

注：char类型没有内置规定其为signed（一般）还是unsigned型，但是只写int则表示signed int类型，表示最高位为其符号位。

**unsigned的含义：无符号数则不认为最高位是符号位，不变二进制位。**

2）读程序

#include<stdio.h>

int main()

{

char a = -128;//10000000 00000000 00000000 10000000

//补码:11111111 11111111 11111111 10000000

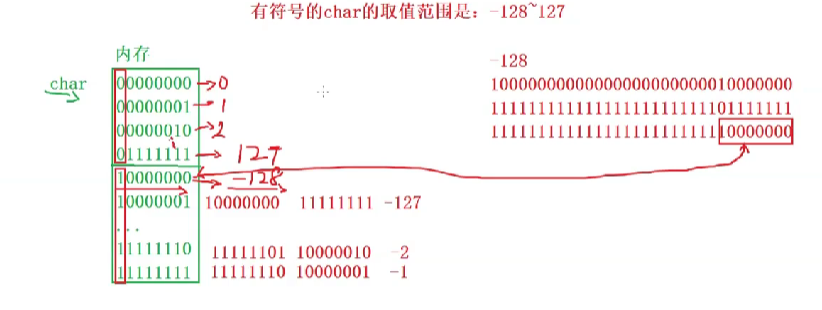
//截断：10000000

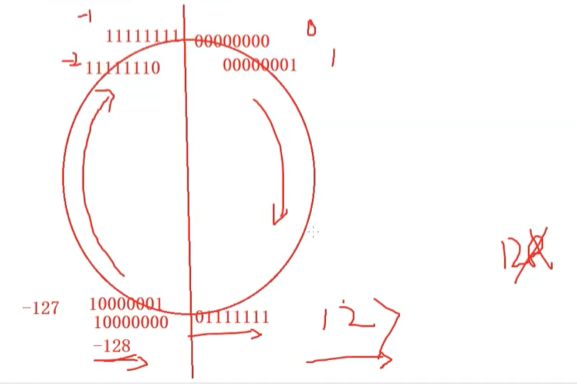
printf("%u\n", a);//%u十进制无符号整数 先提升11111111 11111111 11111111 10000000，再打印时最高位不看做符号位

return 0;

}

**讨论：**char类型的整数大小，或者八位有符号二进制数能存的整数范围。-128是规定的。





3）读程序

#include<stdio.h>

int main()

{

int i = -20;//10000000 00000000 00000000 00010100

//补码：11111111 11111111 11111111 11101100

unsigned int j = 10;//00000000 00000000 00000000 00001010

printf("%d\n", i + j);//补码：11111111 11111111 11111111 11110110

//原码：10000000 00000000 00000000 00001010 -10

return 0;

}

4）读程序

#include<stdio.h>

#include<string.h>

int main()

{

char a[1000];

int i;

for (i = 0; i < 1000; i++)

{

a[i] = -1 - i;//-1 -2 -3 ... -127 -128 127 126 ... 3 2 1 0 -1 -2 ... -127 -128 127 126 ... 1 **循环的 0~255 -128~127**

}

printf("%d\n", strlen(a));//字符串长度找\0，ASCII码对应就是0，所以0处计算结束，128+127=255

return 0;

}

**注：数据的隐形转换，提升和截断很重要！！！**

1.2 浮点型数据的存储

浮点型家族：float, double类型。小数形式或者科学技术法形式，1.2E8，表示1.2\*10^8。Float家族定义的浮点数范围在float.h中。

整型家族的整数范围在limits.h中，INT\_MAX.

1.2.1 浮点数在内存中的存储方式

引例：读程序

#include<stdio.h>

int main()

{

int n = 9;

float\* p = (float\*)&n;

printf("n = %d\n", n);//9 整数的存取对应，打印正确

printf("\*p = %f\n", \*p);//0.000000 整数的存与浮点数的取不对应，打印出错 认为存在内存中的整数的二进制序列是一个浮点数，解析出来为一个很小的数，打印精度认为结果为0.000000

\*p = 9.0;//浮点数的存 1001.0=1.001\*2^3 0 10000010 00100…

printf("n = %d\n", n);//109... 与整数的取不对应，打印出错 将浮点数二进制看成整数二进制打印得

printf("\*p = %f\n", \*p);//9.000000 浮点数的存与取对应，打印正确

return 0;

}

**注：上述例子表明，浮点数与整数的存取方式不同，不能混合使用。**

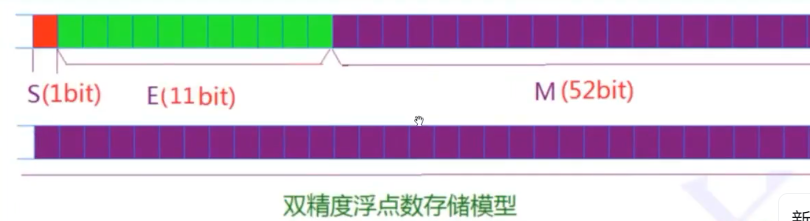
（1）根据IEEE电气和电子工程协会754标准规定，任意一个二进制浮点数可以表示成如下形式：



例：5.5浮点数-10进制 二进制：整数部分为2的非负次幂，**小数部分为2的负次幂**。5.5~101.1~**科学记数法**左移两位：(-1)^0\*1.011\*2^2。

浮点数的存储：





规定存储（先化为标准科学计数法形式）：S存最高位，0表示正数，1表示负数；E加上中间值，127（float）或1023（double）再存入8位或11位中；M只存有效数字的小数位，因为整数位一定为1；最后注意还原原数的方法。

如float 5.5 = (-1)^0\*1.011\*2^2，可得S=0，E=2+127=129=10000001，M=01100000000000000000000

规定取出浮点数分为三种情况：（1）E不全为0或E不全为1，则真实指数为E-127（或E-1023），有效数字M还原加一个整数1；（2）E全为0，则认为真实指数为1-127，现在有效数字不加整数1，表示很小的一个数；（3）E全为1，则认为指数很大255-127=128，表示无穷大。