补码

学习目标:

在编译器中一个 int 类型变量所能存储的数字的范围是多少？

int 类型变量所能存储的最大正整数用 16 进制表示是：7FFFFFFF；

int 类型变量所能存储的绝对值最大的负整数用 16 进制表示：80000000；

具体可以参见“8 位二进制所代表的十进制”

最小负数的二进制代码是多少？

最大正数的二进制代码是多少？

已知一个整数的二进制代码求出最原始的数字。

数字超过最大正数会怎么样？

就会溢出，会把最后面的保存下来。

不同类型数据的相互转化

什么是编码?

一个东西在计算机中到底用哪个二进制代码存储，必须要有一个规定，比如 A 怎么存储的、10 怎么存储的，这个就需要用编码来规定。人们造出了很对编码(原码、反码、补码、移码……)

原码:

也叫:符号-绝对值码

最高位 0 表示正，1 表示负，其余二进制位是该数字的绝对值二进制位

eg:-5 表示为:1 0101

原码简单易懂;

加减运算复杂;

存在加减乘除四种运算，增加了 CPU 的复杂度;

零的表示不唯一(0 可以表示正，也可以表示数字 0);

原码在计算机中从来没有被使用。

反码:

反码运算不便，也没有在计算机中应用。

移码:

移码表示数值平移 n 位，n 称为移码量;

移码主要用于浮点数的阶码的存储。

补码:主要是用来解决整数的存储，计算机中统一使用的是补码来表示

一、已知十进制求二进制

Ⅰ、求正整数的二进制

除 2 取余，直至商为零，余数倒叙排序。

Ⅱ、求负整数的二进制

先求与该负整数相对应的正整数的二进制代码(除 2 取余),然后将所有位取反(就是将原来的 0 表示为 1，1 表示为 0)，末尾加 1，不够位数时(关键看存放变量的变量名是什么数据类型，例如 32 位系统，一个 int 类型变量占 4 个字节，一个

字节 8 位，总共 4*8=32 位，位数不够，补 1），左边补 1。

eg： -3 对应的二进制表示为：3 表示为 0011，取反 1100，末尾加 1 是 1101，-3 表示为（28 个 1）后跟 1101，4 个 1 是一个 F（1111 用 16 进制表示为 F），所以表示为 0X（FFFFFFFD）--（如果 i 内保存的是-3，则使用%#X 输出是就是 0XFFFFFFFD）。

```c
#include<stdio.h>
int main(void){
    int i = -1;
    printf("%#X\n", i);

    return 0;
}
/*
    在dev c++中的输出结果是:
    ------------------------
    0XFFFFFFFF
    ------------------------
*/
```

```c
#include<stdio.h>
int main(void){
    int i = -3;
    printf("%#X\n", i);

    return 0;
}
/*
    在dev c++中的输出结果是:
    ------------------------
    0XFFFFFFFD
    ------------------------
*/
```

eg：求-100 的二进制

[补码_1.cpp]

```c
#include<stdio.h>
int main(void){
    int i = -100;
    printf("%#X\n", i);
    return 0;
}
/*
先将(-100)10的正整数表示出来(100)10,
先用16进制表示为(64)16,再将6和4分别用2进制表示6表示为(0110)2,4表示为(0100),
    --得知用16进制表示的数字,每个数字可以用4个二进制代码表示
再取反表示为10011011,加1表示为(1001)(1100),用16进制表示为9C,位数不够,左边加1
24个1后跟9C，4个1为一个F，所以最终用16进制输出为6个F后跟9C
    在dev c++中的输出结果是:
    ------------------------
    0XFFFFFF9C
    ------------------------
*/
```

III、求零的二进制

全是零

二、已知二进制求十进制（如果是负数，必须写够 32 位，如果写不够，计算机会自动补 0，整数就无所谓）

I、如果首位是 0，则表明是正整数，按普通方法来求

II、如果首位是 1，则表明是负整数（必须写够 32 位）：

将所有位取反，末尾加 1，所得数字就是该负数的绝对值

[补码_2. cpp]

```c
#include<stdio.h>
int main(void)
{
    int i = 0XFFFFFFEF;//定义的数字是16进制的,在数字前加0X
    printf("%d\n", i);
    int j = 0xFFFFFFCA;
    printf("%d\n", j);
    return 0;
}
/*
    在dev c++中的输出结果是:
    ------------------------
    -17
    -54
    ------------------------
*/
```

Ⅲ、如果全是零，则对应的十进制数字就是零

//从 0 到 127, 127 之后是-128 最后到-1

链表：

算法：

通俗定义：

解题的方法和步骤。

狭义定义：

对存储数据的操作；

对不同的存储结构，要完成某一个功能所执行的操作是不一样的。

比如：

要输出数组中所有元素的操作和要输出链表中所有元素的操作是不一样的；

这说明：

算法是依附于存储结构的；

不同的存储结构，所执行的算法是不一样的。

广义定义：

广义的算法也叫泛型；

无论数据是如何存储的，对数据的操作都是一样的。

我们至少可以通过两种结构来存储数据

数组

优点：

存取速度快；

缺点：

需要一个连续的很大的内存；

插入和删除元素的效率很低。

链表

专业术语：

头结点

头结点的数据类型和首节点是一模一样的

头结点是首节点前面的那个节点

头节点并不存放有效数据

设置头结点的目的是为了方便链表的操作

头指针

首节点

存放第一个有效数据的节点

尾节点

存放最后一个有效数据的节点，指针域为 0

头指针

存放头结点地址的指针变量

确定一个链表需要一个参数——头指针(头结点的地址)

优点：

插入删除元素效率高；

不需要一个连续的很大的内存。

缺点：

查找某个位置的元素效率低。