作业1

- 运行以下的C语言程序：

```
#include <iostream>
#include <stdio.h>
int main()
{
    int x = -2147483648;
    printf("real number: %d;\nopposite number: %d;", x, -x);
}
```

分析以上程序的运行结果，为什么

- 运行结果

```
PS C:\Users\Dell\Desktop\计算机系统原理\hw\01> cd "c:\Users\Dell\Desktop\计算机系统原理\hw\01\" ; if ($?) { g++ 01.cpp -o 01 } ; if ($?) { .\01 }
real number: -2147483648;
opposite number: -2147483648;
```

即输出 x = -x = -2147483648。

- 分析原因：
  - 冯诺依曼计算机中数据按二进制数的方式进行储存。其中正数按其原码的二进制形式储存，而负数按补码形式储存，即二进制数每一位按位取反，最后加 `1`。
  - 对于C语言中 `int` 类型的有符号数，计算机分配32个 `bit` 进行储存，其中1~31位为数值位，第32位为符号位。在输出的时候，计算机按有符号数的形式输出
  - `int` 类型本身只能表示-2147483648-2147483647，故2147483648（-x）会发生溢出，其表示为

$$2147483648_{(10)} = 1000\cdots000_{(2)}\,(\text{共}\,31\text{个}\,0)$$

而-2147483648（x）的二进制也为

$$-2147483648_{(10)} = 1000\cdots000_{(2)}\,(\text{共}\,31\text{个}\,0)$$

故输出的时候，按有符号数输出，结果相同为-2147483648

  - 使用C语言程序进行验证：

```
#include <iostream>
#include <stdio.h>
#define BYTE 8
#define TRUE 1
void decToBin(int X);
int main()
{
    int x = 2147483648;
    printf("the binary code of X is: ");
```

```
    decToBin(x);
    printf("the binary code of -X is: ");
    decToBin(-x);
    return 0;
}

void decToBin(int X)
{
    int i, rep, tmp;
    rep = sizeof(int) * BYTE;                       // 得到int类型数据所有的bit数量
    for(i = 0; i < rep; i++)
        printf("%d", (X >> (rep - i - 1)) & TRUE); // 与1作AND操作，输出当前的bit
    printf("\n");                                   // 输出32次
}
```

输出结果，验证了分析：

```
the binary code of X is: 10000000000000000000000000000000
the binary code of -X is: 10000000000000000000000000000000
```

- 系统里以白特（byte）为寻址单位，写个C语言程序，说明一个4白特的int：0x12345678在机器里是怎么存放的?
  - 即分析int类型类型数据是按大端形式存放，还是小端形式存放
  - 根据C语言联合体union数据结构的特性，有如下代码：

```
#include <stdio.h>
union Byte8;
typedef union Byte8 INT;
union Byte8{
    char firstByte;
    short firstTwoByte;
    int fullFourByte;
};

int main()
{
    INT number;
    number.fullFourByte = 0x12345678;
    printf("The number is stored as:\n");
    if( number.firstByte == 0x12 )
        printf("12|34|56|78\n");
    else if( number.firstByte == 0x78 )
        printf("78|56|34|12\n");
    printf("sequentially in the memory\n");
    return 0;
}
```

因为数据在内存中是按顺序(sequentially)储存的，所以如果输出方式如果为第一种，则为大端储存方式；如果为第二种，则为小端储存方式。输出结果为：

```
The number is stored as:
78|56|34|12
sequentially in the memory
```

说明我的机器是用的小端储存方式，即低位字节序内容存放在低地址处，高位字节序的内容存放在高地址处。