

分析报告

首先明确补码真值的求法：已知一个数的补码，要求其真值，首先看该补码的最高位是 0 还是 1，0 表示整数，1 表示负数。然后将符号位后面的数值位全部按位取反再加 1 得到一个二进制数，然后求出这个二进制数的十进制数，最后加上符号即可。

例如：100 的真值

最高位符号位：是 1，代表负数

数值位：00 ----> 11

$11 + 01 = 100$ ，再将 100 转换成十进制是 4.

所以最终结果补码 100 的真值是-4

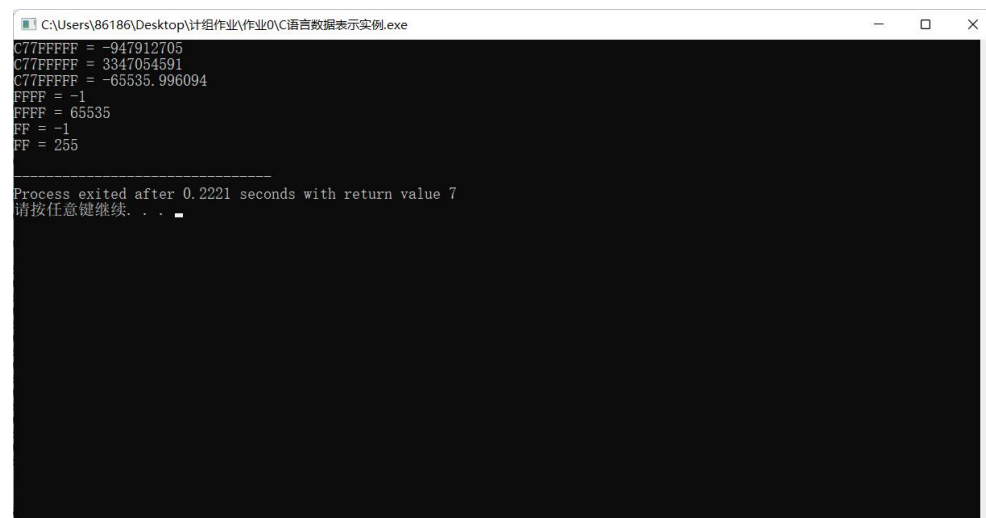
注意：在补码的运算过程中，符号位会参与计算，但在最终结果中，符号位不会参与结果真值的计算，仅仅表示真值的符号正负

一、C 语言数据表示实例

代码如下：

```
#include "stdio.h"
union // 该联合体中所有变量共享 4 字节存储空间，但不同变量字节数不同
{
    int i; unsigned int ui; float f; // i、ui、f 共享 4 字节存储空间，机器码相同，数据类型不同
    short s; unsigned short us; // s、us 共享双字节存储空间，机器码相同，数据类型不同
    char c; unsigned char uc; // c、uc 共享单字节存储空间、机器码相同，数据类型不同
} t;
void hex_out(char a) // 输出 8 位数据的十六进制值
{
    const char HEX[]="0123456789ABCDEF";
    printf("%c%c", HEX[(a&0xF0)>>4],HEX[a&0x0F]);
}
void out_1byte(char *addr) // 用十六进制输出地址中的 8 位数据机器码
{
    hex_out (*(addr+0));
}
void out_2byte(char *addr) // 用十六进制输出地址中的 16 位数据机器码
{ // 小端模式先输出高字节
    hex_out (*(addr+1)); hex_out (*(addr+0));
}
void out_4byte(char *addr) // 用十六进制输出地址中的 32 位数据机器码
{ // 小端模式先输出高字节
    hex_out (*(addr+3)); hex_out (*(addr+2)); hex_out (*(addr+1)); hex_out (*(addr+0));
}
void main()
{
    t.i=0xC77FFFFFFF; // 直接通过机器码赋值，联合体中所有变量共享该机器码
    out_4byte(&t.i); // 输出 i 的机器码和真值，& 表示引用变量的内存地址
    printf(" = %d \n",t.i); // C77FFFFFFF = -947912705
    out_4byte(&t.ui); // 输出 ui 的机器码和真值
    printf(" = %u \n",t.ui); // C77FFFFFFF = 3347054591
    out_4byte(&t.f); // 输出 f 的机器码和真值
    printf(" = %f\n",t.f); // C77FFFFFFF = -65535.996094
    out_2byte(&t.s); // 输出 s 的机器码和真值
    printf(" = %d \n",t.s); // FFFF = -1 整数采用补码表示
    out_2byte(&t.us); // 输出 us 的机器码和真值
    printf(" = %u \n",t.us); // FFFF = 65535
    out_1byte(&t.c); // 输出 c 的机器码和真值
    printf(" = %d\n",t.c); // FF = -1
    out_1byte(&t.uc); // 输出 uc 的机器码和真值
    printf(" = %d\n",t.uc); // FF = 255
}
```

程序执行结果如下：



```
C:\Users\86186\Desktop\计组作业\作业0\C语言数据表示实例.exe
C77FFFFF = -947912705
C77FFFFF = 3347054591
C77FFFFF = -65535.996094
FFFF = -1
FFFF = 65535
FF = -1
FF = 255

Process exited after 0.2221 seconds with return value 7
请按任意键继续...
```

分析：

1. 第一个输出内容：C77FFFFF = -947912705，由于主程序中是直接通过机器码给共用体中所有变量赋值的，因此在等号左边输出的是共用体中 i 的机器码，等号右边输出的是 i 的真值。对于机器码 C77FFFFF，由于是通过有符号数 i 输出的，而有符号数采用补码表示和存储，因此需要看该机器码的二进制形式：1100 0111 0111 1111 1111 1111 1111 1111。在有符号整数中，最高位表示符号位，因为最高位是 1，所以 C77FFFFF 代表的二进制数是一个负数。接下来计算这个二进制数的补码，先按位取反，再加上 1 即可。最终补码为：1011 1000 1000 0000 0000 0000 0000 0001。最高位符号位是 1，表示这个数是负数，将符号位之后的二进制数转换为十进制，结果是 947912705，加上负号后就是输出结果：

C77FFFFF = -947912705

2. 第二个输出内容：C77FFFFF = 3347054591，由于主程序中通过无符号数 ui 对该机器码进行输出，而无符号数在计算机中采用的存储方式是直接以二进制形式存储，因此直接将 C77FFFFF 转换成十进制即可，因此输出内容为 C77FFFFF = 3347054591。

3. 第三个输出内容：C77FFFFF = -65535.996094，由于主程序中通过浮点数 f 输出该机器码，因此需要先将机器码转换成 32 位二进制码：1100 0111 0111 1111 1111 1111 1111 1111（红色部分为符号位 S，蓝色部分为阶码 E，黑色部分为尾数 M），再从该二进制码中分离出 S、E、M，首先最高位符号位 S 是 1，表示该数是负数，在符号位 S 后八位就是该数的阶码 E：10001110，剩余部分均为尾数 M。对于阶码 E，由于 IEEE754 浮点数中的阶码需要采用偏置值，因此在将阶码转换为十进制后还需要减去 127。阶码 E 的十进制：142，减去 127 得到 15。所以 $e = E - 127 = 15$ 。而 $M = 1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111$ ，实际尾数为 $1.M = 1.1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111$

可以得到最终的数字为： $N = (-1) * 1.M * \text{pow}(2, e) = -(1.1111\ 1111\ 1111\ 1111\ 1111\ 1111\ 1111) * \text{pow}(2, 15) = -1.9999998807907104 * 32768 = 65535.996093749$ 所以最后输出的结果为 C77FFFFF = -65535.996094。

4. 第四个输出内容: **FFFF = -1**, 由于主程序通过 short 类型变量输出该机器码, short 类型是 16 位, 而原机器码是 32 位, 所以会出现**截断现象**, 只会输出低四位 FFFF。而 short 类型又属于有符号数, 在计算机中采用补码形式保存, 因此先求出十六进制数 FFFF 的补码。先求 FFFF 的二进制形式: 1111 1111 1111 1111, 再求其反码: 1000 0000 0000 0000, 再加上 1 即可得到补码: 1000 0000 0000 0001。由于最高位符号位是 1, 表示负数, 因此最终输出结果就是 **FFFF = -1**。

5. 第五个输出内容: **FFFF = 65535**, 类似第三个输出内容, 程序通过无符号 short 类型变量输出 C77FFFFFFF, 首先会出现**截断现象**, 输出 FFFF, 再根据**无符号数在计算机中是以二进制形式进行存储**, 可以得知 $FFFF = 1111\ 1111\ 1111\ 1111$, 将其转换成十进制就是 65535。因此输出内容是 **FFFF = 65535**。

6. 第六个输出内容: **FF = -1**, 由于程序通过有符号 char 类型变量输出机器码 C77FFFFFFF, 而 char 类型是 8 位, 因此会出现截断现象, 只会输出 FF。根据有符号数在计算机中是以补码形式存储, 先将 FF 转换成二进制: 1111 1111, 再求其反码: 1000 0000 0000 0000, 最后加上 1 就得到补码: 1000 0000 0000 0001, 最高位符号位是 1, 代表负数, 所以最后输出内容是: **FF = -1**。

7. 第七个输出内容: **FF = 255**, 根据第六个和第五个输出内容分析, 同样可以得知无符号 char 类型变量输出机器码时出现了截断现象, 而无符号数在计算机中存储方式是二进制形式, 将 FF 转换成二进制是 1111 1111, 其值就为 255。因此最后输出内容就是 **FF = 255**。

二、C 语言运算溢出实例

代码如下:

```
C语言运算溢出实例.c
1  #include "stdio.h"
2  void main()
3  {
4      short s1=32767,s2=-32768,s; //-32768 为 16 位最小负数, 32767 为 16 位最大正数
5      unsigned char uc1=128,uc2=255,uc; //255 为 8 位无符号最大值
6      s=s1+1;
7      printf(" %d + 1 = %d\n",s1,s); // 输出 32767 + 1 = -32768 正正得负, 正上溢
8      s=s2-3;
9      printf(" %d - 3 = %d\n",s2,s); // 输出 -32768 - 3 = 32765 负负得正, 负上溢
10     uc=uc1+uc2;
11     printf(" %d + %d = %d\n",uc1,uc2,uc); // 输出 128 + 255 = 127 越加越小, 无符号溢出
12     uc=uc1-uc2;
13     printf(" %d - %d = %d\n",uc1,uc2,uc); // 输出 128 - 255 = 129 越减越大, 无符号溢出
14 }
```

程序执行结果如下:

```
C:\Users\86186\Desktop\计组作业\作业0\C语言运算溢出实例.exe
32767 + 1 = -32768
-32768 - 3 = 32765
128 + 255 = 127
128 - 255 = 129

-----
Process exited after 0.1851 seconds with return value 17
请按任意键继续. . .
```

分析：

1. 第一个输出： **$32767 + 1 = -32768$** ，程序中通过 short 类型变量输出 32767，其为 16 位有符号数，在计算机中采用补码形式保存。首先将 32767 转换成二进制补码的形式：0111 1111 1111 1111，加 1 得到 1000 0000 0000 0000，对于该结果，最高位为 1，表示结果是负数，然后对其数值位求真值：首先全部按位取反得到 111 1111 1111 1111，再加 1 得到 1000 0000 0000 0000，可知其值为 32768，加上符号得到-32768。所以最终输出的结果就是 **$32767 + 1 = -32768$** 。

2. 第二个输出： **$-32768 - 3 = 32765$** ，程序中通过 short 类型变量输出，先将该变量存的数值-32768 转换成二进制补码形式：1 1000 0000 0000 0000，减 3 得到 0 0111 1111 1111 1101，最高位符号位为 0，表示正数，之后的数值位 0111 1111 1111 1101 转换成十进制就是 32765。所以加上符号就是 32765。最终结果输出就为： **$-32768 - 3 = 32765$** 。

3. 第三个输出： **$128 + 255 = 127$** ，程序中变量 uc1 和 uc2 都是无符号 char 类型变量，他们进行运算就是直接进行二进制形式的运算，但其最终结果同样通过无符号 char 类型变量 uc 输出，所以其比特位不会变化。先将 128 和 255 转换成二进制形式：128 对应 1000 0000，255 对应 1111 1111。两者相加得到 1 0111 1111 只取后 8 位：0111 1111，其十进制是 127 产生无符号溢出。所以最终输出结果就是 **$128 + 255 = 127$** 。

4. 第四个输出： **$128 - 255 = 129$** ，同第三个输出，将两者的二进制形式进行相减得到：1000 0001，对应十进制为 129，产生无符号溢出。所以最终输出结果为： **$128 - 255 = 129$** 。

三、C 语言整形数据类型转换

(1) 相同字长的整形数据转换

代码如下：

```
C语言整形数据类型转换 (相同字长的整形数据转换) .c
1  #include "stdio.h"
2  union // 该联合体中所有变量共享 4 字节存储空间, 但不同变量字节数不同
3  {
4      int i; unsigned int ui; float f; // i、ui、f 共享 4 字节存储空间, 机器码相同, 数据类型不同
5      short s; unsigned short us; // s、us 共享双字节存储空间, 机器码相同, 数据类型不同
6      char c; unsigned char uc; // c、uc 共享单字节存储空间, 机器码相同, 数据类型不同
7  } t;
8  void hex_out(char a) // 输出 8 位数据的十六进制值
9  {
10     const char HEX[]="0123456789ABCDEF";
11     printf("%c%c", HEX[(a&0xF0)>>4],HEX[a&0x0F]);
12 }
13 void out_1byte(char *addr) // 用十六进制输出地址中的 8 位数据机器码
14 {
15     hex_out (*(addr +0));
16 }
17 void out_2byte(char *addr) // 用十六进制输出地址中的 16 位数据机器码
18 { // 小端模式先输出高字节
19     hex_out (*(addr +1)); hex_out (*(addr +0));
20 }
21 void out_4byte(char *addr) // 用十六进制输出地址中的 32 位数据机器码
22 { // 小端模式先输出高字节
23     hex_out (*(addr +3)); hex_out (*(addr +2)); hex_out (*(addr +1)); hex_out (*(addr +0));
24 }
25 void main()
26 {
27     unsigned char uc1=255,uc;
28     char c1=-127,c;
29     c=(char) uc1; // 相同宽度数据转换, 无符号转有符号, 强制类型转换可以省略, 机器码不变
30     out_1byte(&uc1); printf(" = uc1 = %u \n",uc1); // 输出原数据的机器码和真值 FF = uc1 = 255
31     out_1byte(&c); printf(" = c = %d \n",c); // 输出转换后的机器码和真值 FF = c = -1
32     uc=c1; // 相同宽度数据转换, 有符号转无符号, 机器码不变
33     out_1byte(&c1); printf(" = c1 = %d \n",c1); // 输出原数据的机器码和真值 81 = c1 = -127
34     out_1byte(&uc); printf(" = uc = %u \n",uc); // 输出转换后的机器码和真值 81 = uc = 129
35 }
```

程序执行结果如下：

```
C:\Users\86186\Desktop\计组作业\作业0\C语言整形数据类型转换 (相同字长的整形数据转换) .exe
FF = uc1 = 255
FF = c = -1
81 = c1 = -127
81 = uc = 129

-----
Process exited after 0.2147 seconds with return value 13
请按任意键继续. . .
```

分析：

1. 第一个输出：FF = uc1 = 255，FF = c = -1。由于 uc1 和 c 的字长相同，仅仅是进行无符号数到有符号数的转换，所以机器码不会发生改变。而 uc1 是无符号数，在计算机中存储方式是二进制，c 是有符号数，在计算机中存储方式是补码形式，所以 uc1 会输出 FF 的二进制形式：1111 1111，它的十进制值为 255；c 会输出 FF 的补码形式：1000 0001，它的十进制值为-1。所以最终输出结果为 FF = uc1 = 255，FF = c = -1。

2. 第二个输出：81 = c1 = -127，81 = uc = 129。同样的，由于两者字长相同，仅仅是有符号数和无符号数的区别，所以机器码不变。对于有符号数 c1，它在计算机中以补码形式存储，而 81 对应的补码是 1111 1111，对应的十进制是-127；对于无符号数 uc，它在计算机中以二进制形式存储，81 对应的二进制是 1000 0001，对应的十进制为 129。所以最终输出的结果是 81 = c1 = -127，81 = uc = 129。

(2) 小字长转大字长

代码如下：

```
C语言整形数据类型转换（小字长转大字长）.c
1 #include "stdio.h"
2 union // 该联合体中所有变量共享 4 字节存储空间，但不同变量字节数不同
3 {
4     int i; unsigned int ui; float f; // i、ui、f 共享 4 字节存储空间，机器码相同，数据类型不同
5     short s; unsigned short us; // s、us 共享双字节存储空间，机器码相同，数据类型不同
6     char c; unsigned char uc; // c、uc 共享单字节存储空间、机器码相同，数据类型不同
7 } t;
8 void hex_out(char a) // 输出 8 位数据的十六进制值
9 {
10     const char HEX[]="0123456789ABCDEF";
11     printf("%c%c", HEX[(a&0xF0)>>4],HEX[a&0x0F]);
12 }
13 void out_1byte(char *addr) // 用十六进制输出地址中的 8 位数据机器码
14 {
15     hex_out (*(addr +0));
16 }
17 void out_2byte(char *addr) // 用十六进制输出地址中的 16 位数据机器码
18 { // 小端模式先输出高字节
19     hex_out (*(addr +1)); hex_out (*(addr +0));
20 }
21 void out_4byte(char *addr) // 用十六进制输出地址中的 32 位数据机器码
22 { // 小端模式先输出高字节
23     hex_out (*(addr +3)); hex_out (*(addr +2)); hex_out (*(addr +1)); hex_out (*(addr +0));
24 }
25 void main ()
26 {
27     unsigned char uc=254;
28     char c=uc;
29     int i; unsigned ui;
30     i=uc; // 无符号小字长转有符号大字长，机器码零扩展
31     ui=uc; // 无符号小字长转无符号大字长，机器码零扩展
32     out_1byte(&uc); printf(" = uc = %d \n",uc); // 输出原数据的机器码和真值 FE = uc = 254
33     out_4byte(&i); printf(" = i = %d \n",i); // 输出转换后的机器码和真值 000000FE = i = 254
34     out_4byte(&ui); printf(" = ui = %u \n",ui); // 输出转换后的机器码和真值 000000FE = ui = 254
35     i=c; // 有符号小字长转大字长，机器码符号扩展
36     ui=c; // 有符号小字长转大字长，机器码符号扩展
37     out_1byte(&c); printf(" = c = %d \n",c); // 输出原数据的机器码和真值 FE = c = -2
38     out_4byte(&i); printf(" = i = %d \n",i); // 输出转换后的机器码和真值 FFFFFFFE = i = -2
39     out_4byte(&ui); printf(" = ui = %u \n",ui); // 输出转换后的机器码和真值 FFFFFFFE = ui = 4294967294
40 }
```

程序执行结果如下：



分析：

1. 第一个输出：FE = uc = 254，000000FE = i = 254，000000FE = ui = 254。uc 是 char16 位，i 和 ui 是 32 位，这里发生的事小字长转大字长，在这里我们需要根据原数据是否有符号数对机器码进行不同的位扩展。若原数据是无符号，则进行零扩展；否则进行符号扩展。

零扩展（无符号数）：将小字长的数据转换为大字长时，无论小字长的最高位符号位是 0 还是 1，转换时都会填充 0。

符号扩展（有符号数）：将原始数的符号位复制到所有新增的高位上。

因此，由于 uc 是无符号 16 位，所以在进行小字长转大字长时会发生零扩展，所以 i 和 ui 的机器码全部变成 0000 00FE。也因此，最终的输出结果为：FE = uc = 254，000000FE = i = 254，000000FE = ui = 254。

2. 第二个输出：FE = c = -2，FFFFFFFE = i = -2，FFFFFFFE = ui = 4294967294。由于在这里 c 是有符号数，所以 i 和 ui 会进行符号扩展。首先 FE 的补码形式为 1111 1110，对于变量 i，保留 FE 补码形式中的符号位 1，然后进行扩展，可以得到变量 i 的补码 1111 1111 1111 1110，可以看出这时候 i 的机器码就变成了 FFFFFFFE。对于变量 ui 也是同理，只是 i 和 ui 的区别在于有符号与无符号，机器码均为 FFFFFFFE。所以最后输出的结果就是 FE = c = -2，FFFFFFFE = i = -2，FFFFFFFE = ui = 4294967294。

（3）大字长转小字长

代码如下：

C语言整形数据类型转换（大字长转小字长）.c

```

1  #include "stdio.h"
2  union // 该联合体中所有变量共享 4 字节存储空间，但不同变量字节数不同
3  {
4      int i; unsigned int ui; float f; // i、ui、f 共享 4 字节存储空间，机器码相同，数据类型不同
5      short s; unsigned short us; // s、us 共享双字节存储空间，机器码相同，数据类型不同
6      char c; unsigned char uc; // c、uc 共享单字节存储空间、机器码相同，数据类型不同
7  } t;
8  void hex_out(char a) // 输出 8 位数据的十六进制值
9  {
10     const char HEX[]="0123456789ABCDEF";
11     printf("%c%c", HEX[(a&0xF0)>>4],HEX[a&0x0F]);
12 }
13 void out_1byte(char *addr) // 用十六进制输出地址中的 8 位数据机器码
14 {
15     hex_out (*(addr +0));
16 }
17 void out_2byte(char *addr) // 用十六进制输出地址中的 16 位数据机器码
18 { // 小端模式先输出高字节
19     hex_out (*(addr +1)); hex_out (*(addr +0));
20 }
21 void out_4byte(char *addr) // 用十六进制输出地址中的 32 位数据机器码
22 { // 小端模式先输出高字节
23     hex_out (*(addr +3)); hex_out (*(addr +2)); hex_out (*(addr +1)); hex_out (*(addr +0));
24 }
25 void main()
26 {
27     int i=0xFFFF1001;
28     short s; unsigned short us;
29     s=i; // 大字长转小字长，机器码被截短
30     us=i; // 大字长转小字长，机器码被截短
31     out_4byte(&i); printf(" = i = %d \n",i); // 输出原数据的机器码和真值 FFFF1001 = i = -61439
32     out_2byte(&s); printf(" = s = %d \n",s); // 输出转换后的机器码和真值 1001 = s = 4097
33     out_2byte(&us); printf(" = us = %u \n",us); // 输出转换后的机器码和真值 1001 = us = 4097
34 }

```

代码执行结果如下：

```

C:\Users\86186\Desktop\计组作业\作业0\C语言整形数据类型转换（大字长转小字长）.exe
00000000
FFFF1001 = i = -61439
1001 = s = 4097
1001 = us = 4097

-----
Process exited after 0.1957 seconds with return value 14
请按任意键继续. . .

```

分析：

输出为：FFFF1001 = i = -61439，1001 = s = 4097，1001 = us = 4097。在进行大字长转小字长时，通常编译器会直接将机器码截短处理。比如 32 位变量 i 存放的机器码为 FFFF1001，将其转换成 16 位有符号变量 s 和无符号变量 us 时，就发生了截短现象，机器码均为 1001，对应的十进制为 4097。所以最终的输出结果为 FFFF1001 = i = -61439，1001 = s = 4097，1001 = us = 4097。

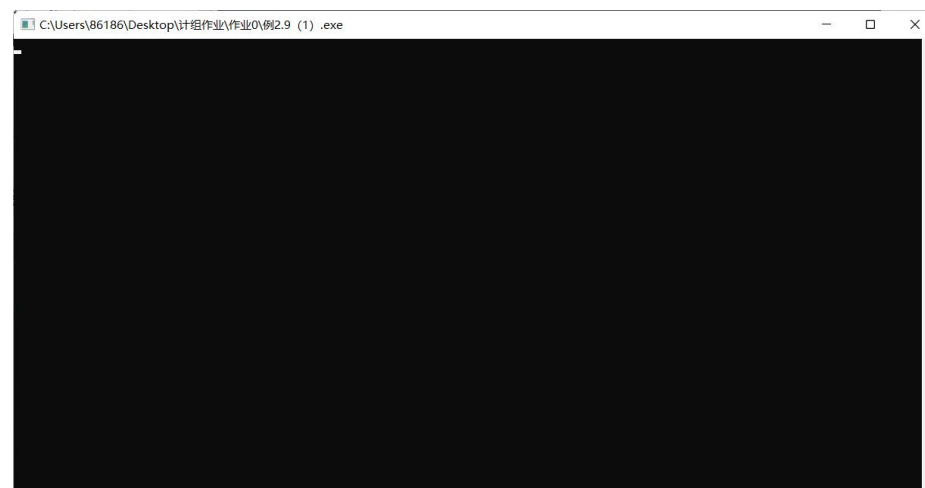
四、P38 的例 2.9

第（1）问

代码如下：

```
例2.9 (1) .c
1  #include "stdio.h"
2
3  int f1(unsigned n) {
4      int sum = 1, power = 1;
5      for (unsigned i = 0; i <= n - 1; i++) {
6          power *= 2;
7          sum += power;
8      }
9      return sum;
10 }
11
12 int main() {
13     int ans;
14     ans = f1(0);
15     printf ("%d\n", ans);
16     return 0;
17 }
```

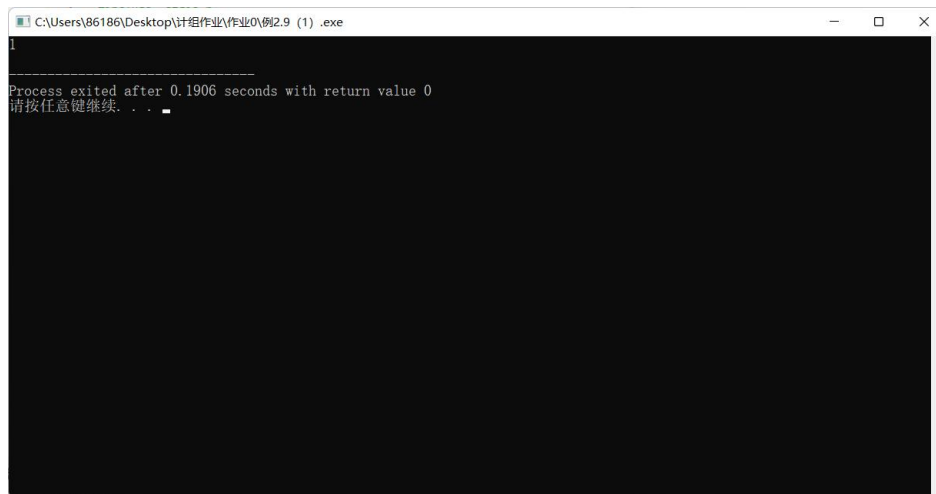
输出结果如下：



更改后代码如下：

```
例2.9 (1) .c
1  #include "stdio.h"
2
3  int f1(int n) {
4      int sum = 1, power = 1;
5      for (int i = 0; i <= n - 1; i++) {
6          power *= 2;
7          sum += power;
8      }
9      return sum;
10 }
11
12 int main() {
13     int ans;
14     ans = f1(0);
15     printf ("%d\n", ans);
16     return 0;
17 }
```

更改后输出结果如下：



分析：可以看到，在变量 i 和 n 都定义成 `unsigned` 类型时，取 $n = 0$ ，程序不会有任何输出，进入死循环；而将 i 和 n 都定义成 `int` 类型时，就会有输出结果，不会进入死循环。

这是因为，当 i 和 n 为 `unsigned` 型无符号数并且 $n = 0$ 时， $n - 1$ 的机器数为全 1，值为 $\text{pow}(2, 32) - 1$ ，为无符号整型的最大值，for 循环判断条件“ $i \leq n - 1$ ”永真，进入死循环。

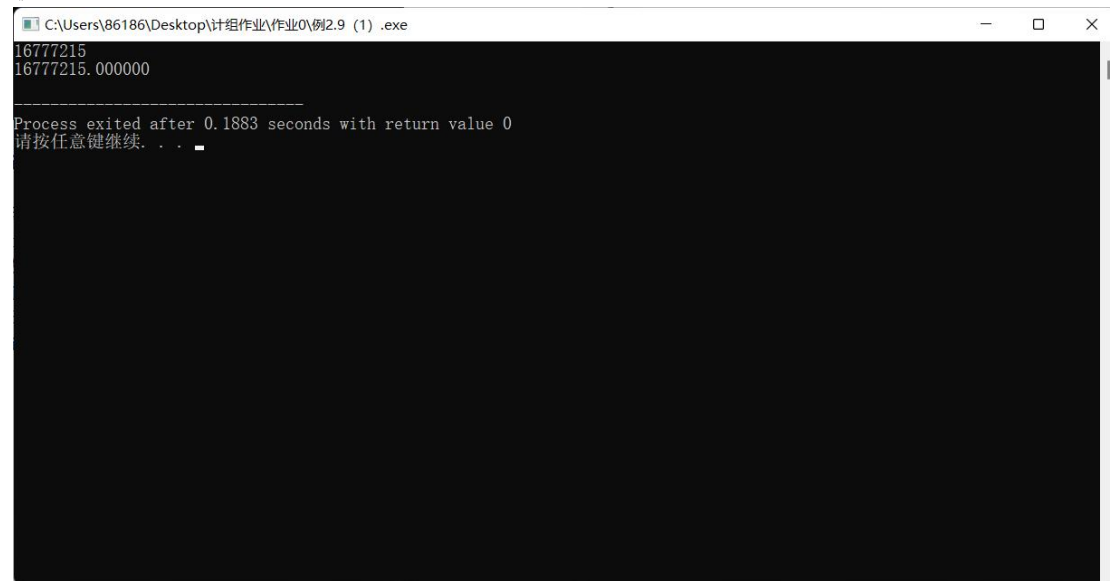
如果将 i 和 n 改为有符号的 `int` 类型，则 $n - 1 = -1$ ，for 循环的判断条件就不会成立，直接退出循环。

第（2）问

代码如下：

```
1  #include "stdio.h"
2
3  int f1(unsigned n) {
4      int sum = 1, power = 1;
5      for (unsigned i = 0; i <= n - 1; i++) {
6          power *= 2;
7          sum += power;
8      }
9      return sum;
10 }
11
12 float f2(unsigned n) {
13     float sum = 1, power = 1;
14     for (unsigned i = 0; i <= n - 1; i++) {
15         power *= 2;
16         sum += power;
17     }
18     return sum;
19 }
20
21 int main() {
22     int ans1;
23     double ans2;
24     ans1 = f1(23);
25     ans2 = f2(23);
26     printf ("%d\n%f\n", ans1, ans2);
27     return 0;
28 }
```

执行结果如下：



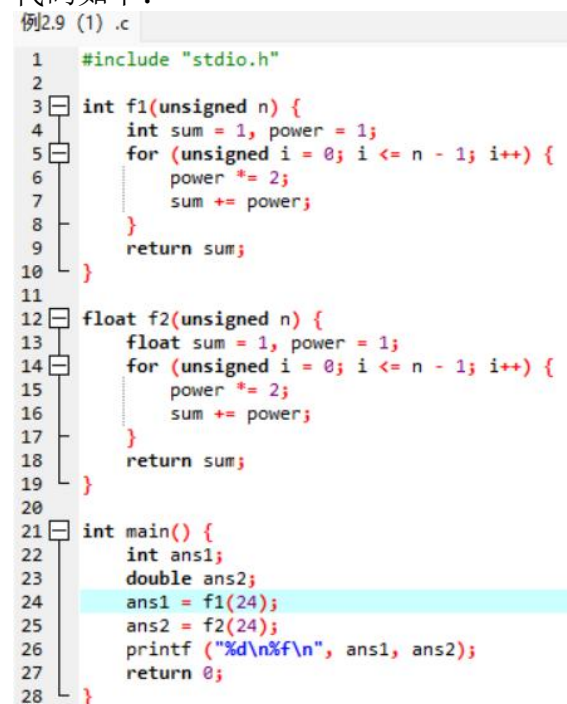
```
C:\Users\86186\Desktop\计组作业\作业0\例2.9 (1) .exe
16777215
16777215.000000

Process exited after 0.1883 seconds with return value 0
请按任意键继续. . .
```

分析：根据代码的输出结果可知， $f1(23)$ 和 $f2(23)$ 的返回值相等。根据计算， $f1(23)$ 的二进制为 1111 1111 1111 1111 1111 1111，并没有超出 `int` 的数据范围，其机器码为 00FF FFFF。当将该数转换为 `float` 类型时，尾数为 1.M，其中， $M = 111\ 1111\ 1111\ 1111\ 1111\ 1111$ ，共 23 个 1，所以阶码 = $23 + 127 = 150$ ，将 150 用二进制表示为 1001 0110，符号位为 0，因此最终该浮点数的机器码的二进制为 0100 1011 0111 1111 1111 1111 1111 1111，转化为十六进制为 4B7F FFFF。

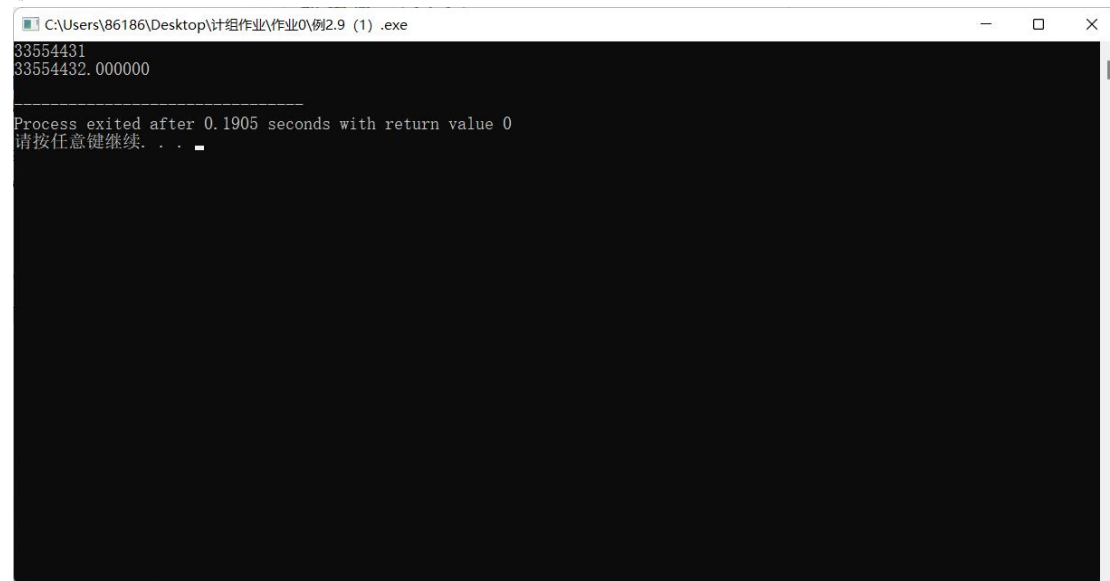
第（3）问

代码如下：



```
例2.9 (1) .c
1  #include "stdio.h"
2
3  int f1(unsigned n) {
4      int sum = 1, power = 1;
5      for (unsigned i = 0; i <= n - 1; i++) {
6          power *= 2;
7          sum += power;
8      }
9      return sum;
10 }
11
12 float f2(unsigned n) {
13     float sum = 1, power = 1;
14     for (unsigned i = 0; i <= n - 1; i++) {
15         power *= 2;
16         sum += power;
17     }
18     return sum;
19 }
20
21 int main() {
22     int ans1;
23     double ans2;
24     ans1 = f1(24);
25     ans2 = f2(24);
26     printf ("%d\n%f\n", ans1, ans2);
27     return 0;
28 }
```

执行结果如下：



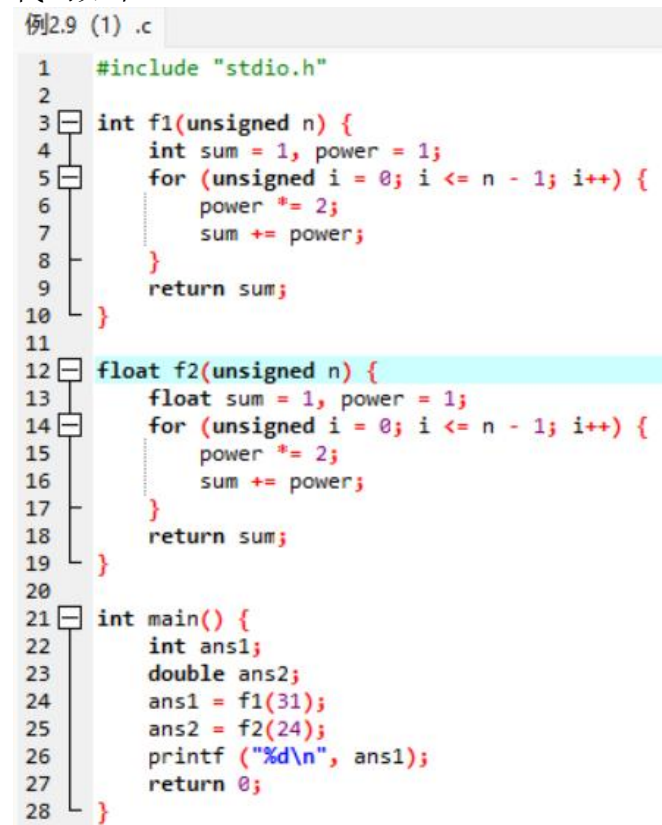
```
C:\Users\86186\Desktop\计组作业\作业0\例2.9 (1) .exe
33554431
33554432.000000

Process exited after 0.1905 seconds with return value 0
请按任意键继续...
```

分析：可以看到， $f1(24)$ 和 $f2(24)$ 的返回值并不相等，这是因为当 $n = 24$ 时，得到数据的二进制表示为 1 1111 1111 1111 1111 1111 1111，总共有 25 个 1，而 float 的有效位算上隐藏位也只有 24 位，舍入之后数值会增大，此时浮点数的最小刻度间距是 2，所以以 float 形式输出这个结果会比 int 形式多 1。

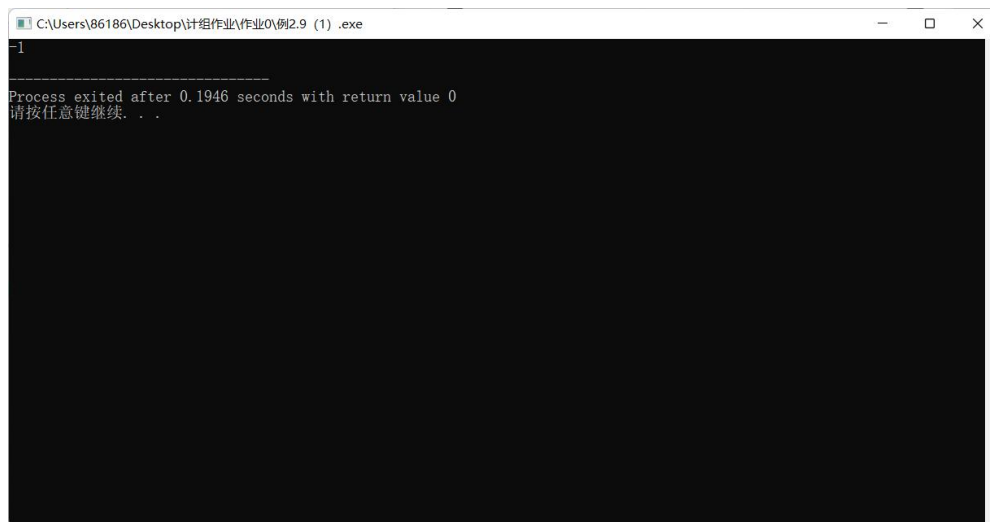
第（4）问

代码如下：



```
例2.9 (1) .c
1  #include "stdio.h"
2
3  int f1(unsigned n) {
4      int sum = 1, power = 1;
5      for (unsigned i = 0; i <= n - 1; i++) {
6          power *= 2;
7          sum += power;
8      }
9      return sum;
10 }
11
12 float f2(unsigned n) {
13     float sum = 1, power = 1;
14     for (unsigned i = 0; i <= n - 1; i++) {
15         power *= 2;
16         sum += power;
17     }
18     return sum;
19 }
20
21 int main() {
22     int ans1;
23     double ans2;
24     ans1 = f1(31);
25     ans2 = f2(24);
26     printf ("%d\n", ans1);
27     return 0;
28 }
```


执行结果如下：

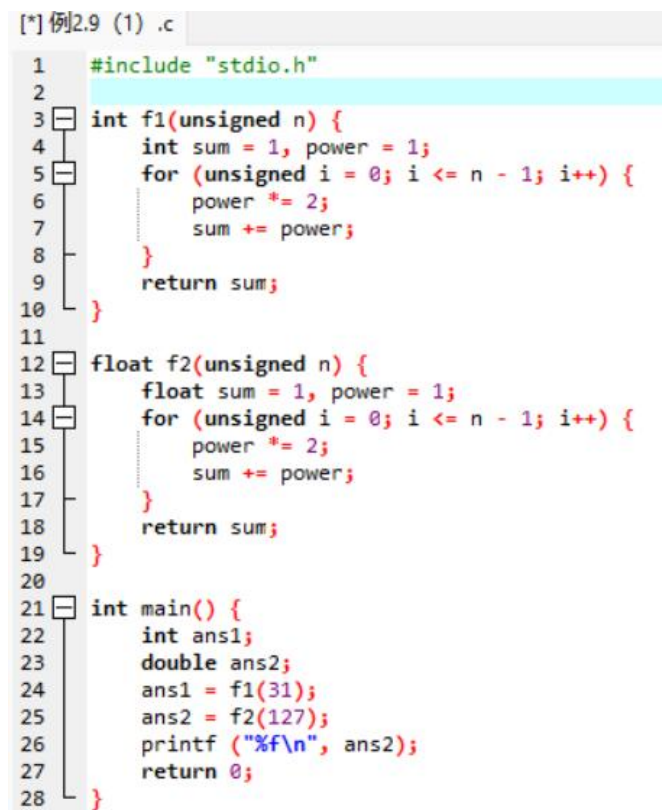


```
C:\Users\86186\Desktop\计组作业\作业0\例2.9 (1) .exe
-1
-----
Process exited after 0.1946 seconds with return value 0
请按任意键继续. . .
```

分析：因为 int 最大可表示的数是 $\text{pow}(2, 31) - 1$ ，也就是 32 位中除了最低位是 0，其余位均为 1。当 $n = 31$ 时，得到的机器数是 32 位均为 1 的数，因此 int 会将其输出为 -1。若要求最大的 n ，只能是小于等于 $\text{pow}(2, 31) - 1$ 的数， $n = 30$ 可以满足。

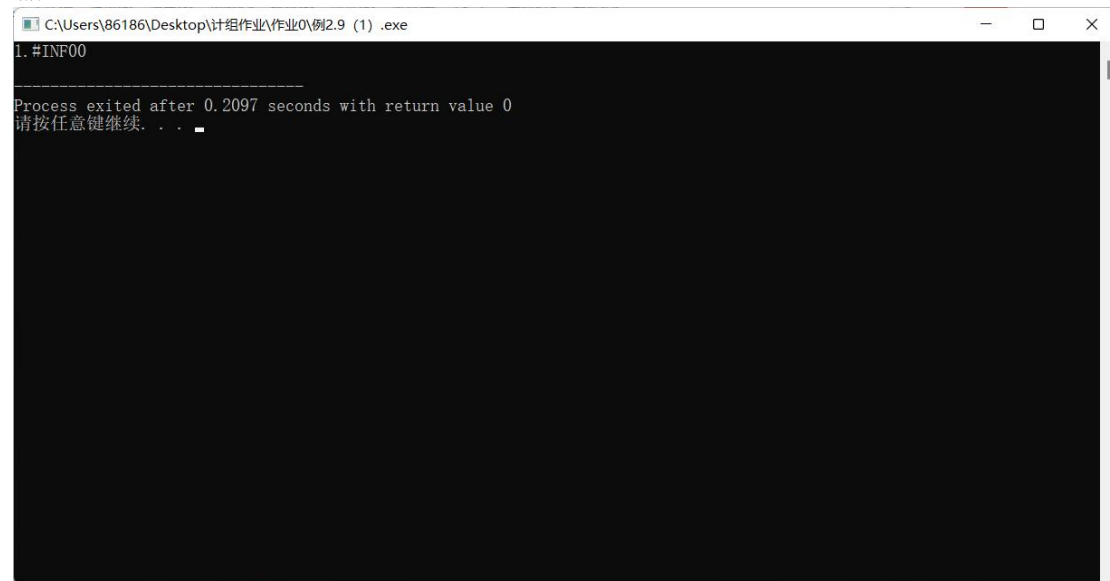
第（5）问

代码如下：



```
[*] 例2.9 (1) .c
1  #include "stdio.h"
2
3  int f1(unsigned n) {
4      int sum = 1, power = 1;
5      for (unsigned i = 0; i <= n - 1; i++) {
6          power *= 2;
7          sum += power;
8      }
9      return sum;
10 }
11
12 float f2(unsigned n) {
13     float sum = 1, power = 1;
14     for (unsigned i = 0; i <= n - 1; i++) {
15         power *= 2;
16         sum += power;
17     }
18     return sum;
19 }
20
21 int main() {
22     int ans1;
23     double ans2;
24     ans1 = f1(31);
25     ans2 = f2(127);
26     printf ("%f\n", ans2);
27     return 0;
28 }
```

输出结果如下：



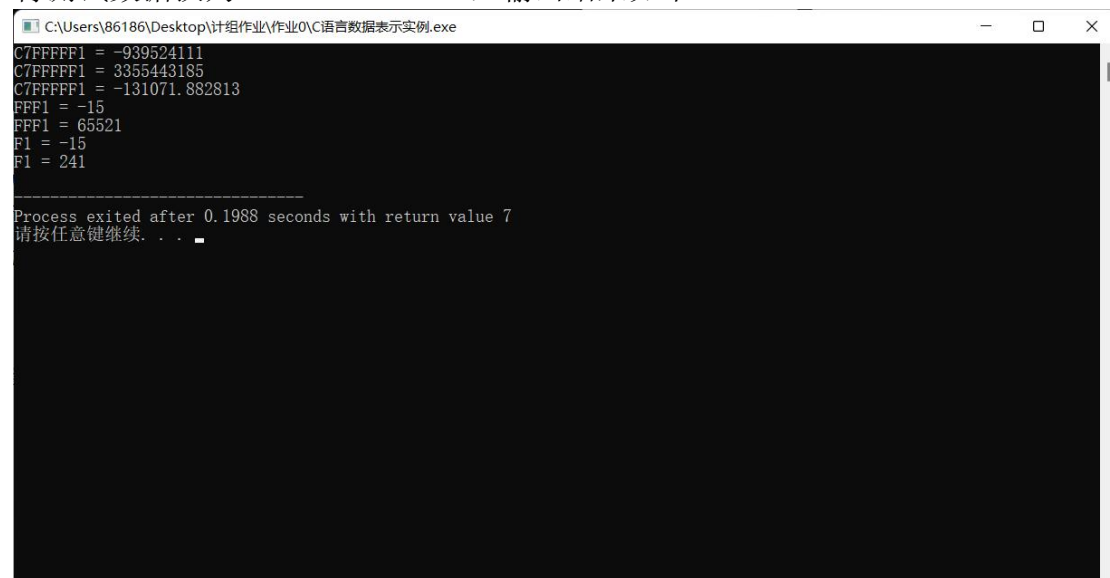
```
C:\Users\86186\Desktop\计组作业\作业0\例2.9 (1) .exe
1. #INF00
-----
Process exited after 0.2097 seconds with return value 0
请按任意键继续. . .
```

分析：可以看到，输出结果是正无穷。对于机器数 7F80 0000，其阶码为 255，尾数为 0，在 IEEE754 标准中表示无穷大。当 n 减小 1 时，经过计算可以得到输出结果为：1701411834604692300000000000000000000000.000000。对应的阶码为 $126 + 127 = 253$ ，尾数部分舍入后阶码加 1，因此阶码为 254，是 IEEE754 单精度格式表示的最大阶码，因此满足条件的 n 的最大值为 126。

五、多种数据集下的验证工作

1. C 语言数据表示实例

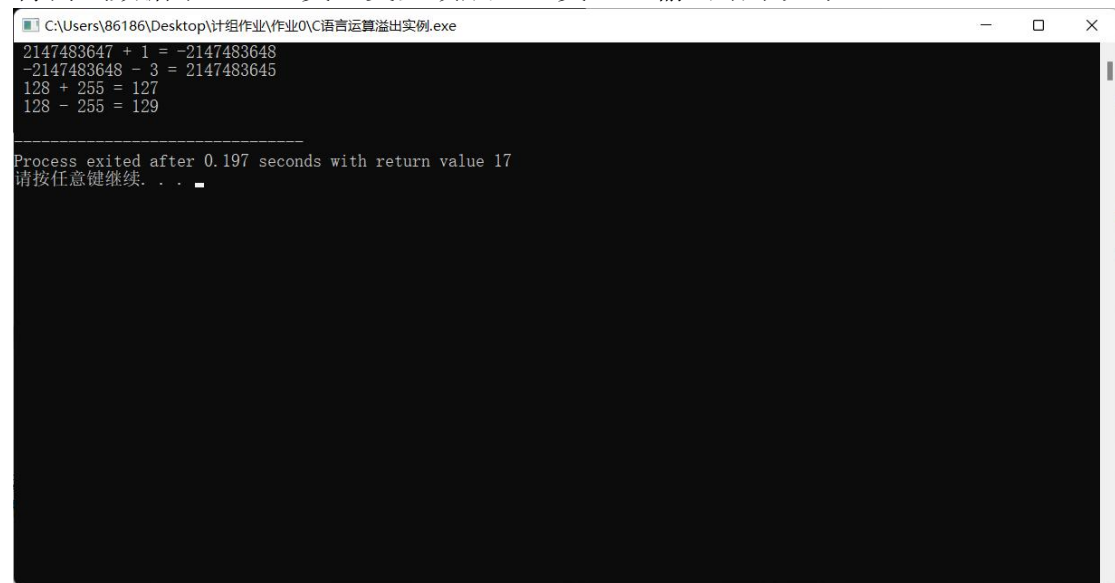
将测试数据换为 $t.i = C7FFFFFF1$ ，输出结果如下



```
C:\Users\86186\Desktop\计组作业\作业0\C语言数据表示实例.exe
C7FFFFFF1 = -939524111
C7FFFFFF1 = 3355443185
C7FFFFFF1 = -131071.882813
FFF1 = -15
FFF1 = 65521
F1 = -15
F1 = 241
-----
Process exited after 0.1988 seconds with return value 7
请按任意键继续. . .
```

2. C 语言运算溢出实例

将测试数据中 short 类型变量改成 int 类型，输出结果如下



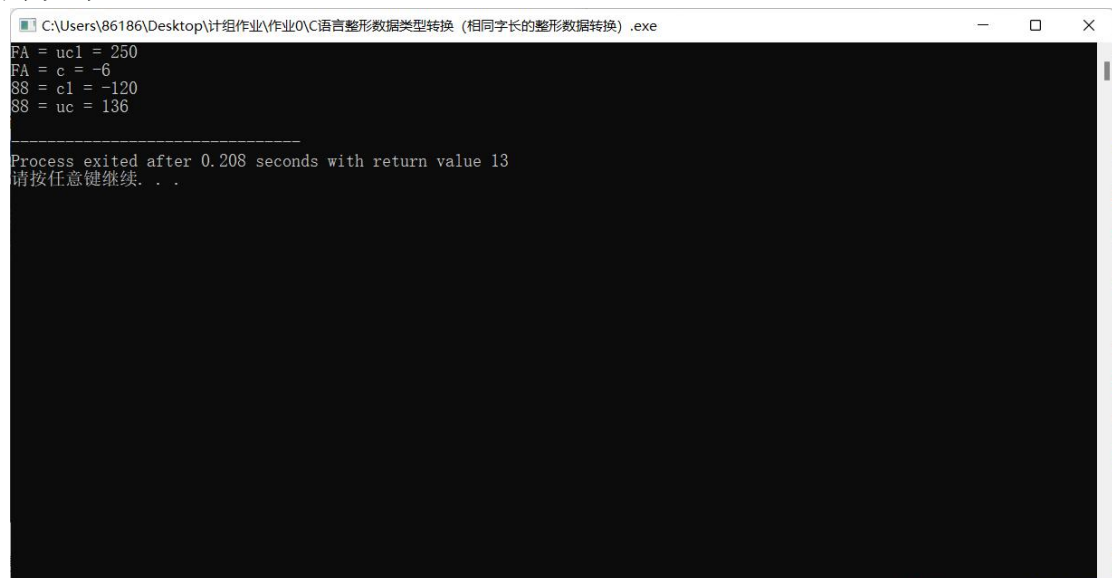
```
C:\Users\86186\Desktop\计组作业\作业0\C语言运算溢出实例.exe
2147483647 + 1 = -2147483648
-2147483648 - 3 = 2147483645
128 + 255 = 127
128 - 255 = 129

-----
Process exited after 0.197 seconds with return value 17
请按任意键继续. . .
```

3. C 语言整型数据类型转换

1). 相同字长

将测试数据改为 unsigned char 类型 uc1 = 250, char 类型 c1 = -120, 输出结果如下:



```
C:\Users\86186\Desktop\计组作业\作业0\C语言整型数据类型转换 (相同字长的整型数据转换) .exe
FA = uc1 = 250
FA = c = -6
88 = c1 = -120
88 = uc = 136

-----
Process exited after 0.208 seconds with return value 13
请按任意键继续. . .
```

2). 小字长转大字长

将测试数据改为 unsigned char uc = 250, 输出结果如下:

```
C:\Users\86186\Desktop\计组作业\作业0\C语言整形数据类型转换 (小字长转大字长) .exe
FA = uc = 250
000000FA = i = 250
000000FA = ui = 250
FA = c = -6
FFFFFFFA = i = -6
FFFFFFFA = ui = 4294967290

-----
Process exited after 0.2102 seconds with return value 20
请按任意键继续. . .
```

3). 大字长转小字长

将测试数据改为 `int i = FFFF0001`, 输出结果如下:

```
C:\Users\86186\Desktop\计组作业\作业0\C语言整形数据类型转换 (大字长转小字长) .exe
FFFF0001 = i = -65535
0001 = s = 1
0001 = us = 1

-----
Process exited after 0.2043 seconds with return value 11
请按任意键继续. . .
```