**实验项目名称 浮点数表示实验**

## 一、 实验目的

理解IEEE754编码基本规则和特点，能分析程序中浮点运算的执行结果，并利用程序实际执行验证真实计算机中浮点数据的表示规则，了解IEEE754编码表示容易引起的一些问题。

## 二、 主要仪器设备、实验环境

Dev C++集成开发环境

## 三、 实验任务

(1)小练习:假设你已经身家千万,请问:你最少有多少钱(以元为单位的整数)才能导致32位的单精度浮点数无法精确表示?你最少有多少钱才能导致64位的双精度浮点数无法精确表示,写一个小程序将这两个数字用十进制打印出来,同时请精确输出32位单精度浮点数的最小非规格化数、最大非规格化数的十进制值。

|  |
| --- |
| #include <iostream>  #include <float.h>  #include <limits.h>  using namespace std;  int main() {  printf("float 类型能存储的最大值:%.1f\n", FLT\_MAX);  printf("float 类型能存储的最小值%.308f\n", FLT\_MIN);  cout << "double 类型能存储的最大值和最小值" << endl;  cout << "DBL\_MAX = " << DBL\_MAX << endl;  cout << "DBL\_MIN = " << DBL\_MIN << endl;  cout << "你的32位钱：";  printf("%.1f\n", FLT\_MAX + 1);  cout << "你的64位钱：";  printf("%.1f\n", DBL\_MAX + 1);  return 0;  } |

(2)无法表示的整数:运行下面的C语言程序,观察运行输出结果,并解释运行现象。

|  |
| --- |
| #include <stdio.h>  #include <float.h>  #include <limits.h>  int main(int argc,char\*\*argv)  {  float f1=16777216.0;  float f2=16777217.0;  float f3=16777218.0;  printf("16,777,216:%f\n",f1);  printf("16,777,217:%f\n",f2);  printf("16,777,218:%f\n",f3);  printf("f1==f2?%s",f1==f2?"true":"false");  return 0;  } |

(3)整数除零问题:运行下面的C语言程序,观察运行输出结果,并解释运行现象。

|  |
| --- |
| //file: int\_div\_by\_zero.c  #include "stdio.h"  main()  {  int a=1; a=a/0;  printf("a=%d",a);  return;  } |

(4)浮点数除零问题:运行下面的C语言程序,观察运行输出结果,并解释运行现象。

|  |
| --- |
| //file: float\_div\_by\_zero.c  #include "stdio.h"  main()  {  float a=0.0, b;  a=a/0; b=-sqrt(-1);  printf ("a=%f b=%f",a,b); //输出值是什么意思？  return;  } |

(5)浮点数双零问题:运行下面的C语言程序,观察运行输出结果,并解释运行现象。

|  |
| --- |
| //file: float\_two\_zero.c  #include "stdio.h"  union  {  char c[4]; //联合体，多变量共享存储空间  float f;  int i;  } t1,t2;  int main()  {  t1.i=0X80000000; //直接机器码赋值，如解释为浮点数为负零  t2.i=0X00000000; //直接机器码赋值，如解释为浮点数为正零  if (t1.f==t2.f)  printf("float data is equal\n");  if (t1.i!=t2.i)  printf("int data is not equal\n");  } |

(6)浮点数的精度问题:运行下面的C语言程序,观察运行输出结果,并解释运行现象。.

|  |
| --- |
| // file : double\_comparisons.c  #include "stdio.h"  int main ( int argc , char \*\* argv )  {  double a = 0.1;  double b = 0.2;  double c = 0.3;  double d = a + b;  printf ("a (0.1) : %.30g\n",a);  printf ("b (0.2) : %.30g\n",b);  printf ("c (0.3) : %.30g\n",c);  printf ("d (0.3) : %.30g\n",d);  printf ("(c == d) ? %s\n", c == d ? "true" : "false");  printf ("(c < d) ? %s\n", c < d ? "true" : "false");  return 0;  } |

(7)浮点数运算误差问题:运行下面的C语言程序,观察运行输出结果,并解释运行现象。

|  |
| --- |
| //file: float\_precision.c  #include "stdio.h"  test\_float\_cal()  {  float a,b,c; int d;  b=3.3; c=1.1;  a=b/c;  d=b/c;  printf("a=%f,d=%d",a,d);  if (3.0==a)  printf("\nFloat a=3.3/1.1==3.0");  }  test\_double\_cal()  {  double a,b,c; int d;  b=3.3; c=1.1;  a=b/c;  d=b/c;  printf("\n\na=%f,d=%d",a,d);  if (3.0!=a)  printf("\nDouble a=3.3/1.1 != 3.0");  }  main()  {  test\_float\_cal();  test\_double\_cal();  } |

(8)浮点数结合律问题:运行下面的C语言程序,观察运行输出结果,并解释运行现象。

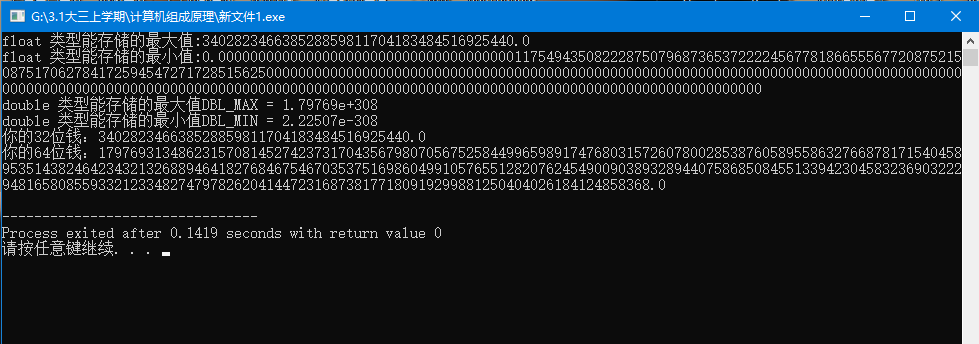
|  |
| --- |
| // file : float\_associativity.c example from standford cs107  # include "stdio.h"  int main ( int argc , char \*\* argv )  {  float a = 3.14;  float b = 1e20;  printf (" (3.14 + 1e20 ) - 1e20 = %f\n", (a + b) - b);  printf (" 3.14 + (1 e20 - 1e20 ) = %f\n", a + (b - b));  return 0;  } |

(9)浮点数溢出问题:运行下面的C语言程序,观察运行输出结果,并解释运行现象。

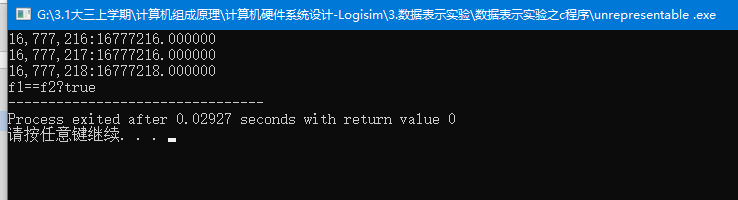
|  |
| --- |
| // file : float\_add\_overflow.c  # include "stdio.h"  union  {  char c[4];  float f;  int i;  } t1,t2,t3,t4;  //输出字符的十六进制编码  void char\_hex\_out(char a)  {  const char HEX[]="0123456789ABCDEF";  int index=a&0x0F;  printf("%c%c", HEX[(a&0xF0)>>4],HEX[a&0x0F]);  }  //输出4个字节数据的十六进制编码，可用于输出4字节变量的机器码  void four\_byte\_out (char \*addr)  {  char\_hex\_out (\*(addr +3)); //输出指针变量的值，指针本质上是内存地址，无符号数  char\_hex\_out (\*(addr +2)); //假设采用小端存储  char\_hex\_out (\*(addr +1));  char\_hex\_out (\*(addr +0));  printf("\n");  }  void float\_add\_overflow()  {  t1.i=0x7F000000; t2.i=0x7F000000;  t3.f=t1.f+t2.f;  four\_byte\_out (&t1.f); //该函数见1.1.3节程序中的定义  four\_byte\_out (&t2.f);  four\_byte\_out (&t3.f);  printf("t1 = %.60f \n",t1.f);  printf("t2 = %.60f \n",t2.f);  printf("t1+t2 = %.60f \n",t3.f);  }  void float\_sub\_overflow()  {  t1.i=0x00C00000; t2.i=0x00800000;  t3.f=t1.f-t2.f;  four\_byte\_out (&t1.f); //该函数见1.1.3节程序中的定义  four\_byte\_out (&t2.f);  four\_byte\_out (&t3.f);  printf("t1 = %.61f \n",t1.f);  printf("t2 = %.61f \n",t2.f);  printf("t1-t2 = %.61f \n",t3.f);  }  main()  {  float\_add\_overflow();  float\_sub\_overflow();  } |

## 四、 实验过程及结果分析

（1）



（2）



16777217.0无法正常显示

原因：IEEE 754的float (binary32)只有24位(二进制)有效数字。每次赋值时都需要做保留24位有效数字的操作。

默认的「中值取偶」规则可以认为是:

如果第25位是0,不进位; (因为X.OY显然小于中值X.1,离X更近)

如果第25位是1,

如果后续存在非0位,进位; (因为X.1Y1Z显然大于中值X.1,离X+1更近)

如果后续位都为0,如果第24位是0,不进位; (中值X.1的x是偶数,在×和x+1中取偶数x)

如果第24位是1,进位。(中值X.1的 是奇数,在x和X+1中取偶数X+ 1)问题中16777217的二进制数一共有25位:

第24位为0,第25位为1,后续位都为0 (没有后续位) 。

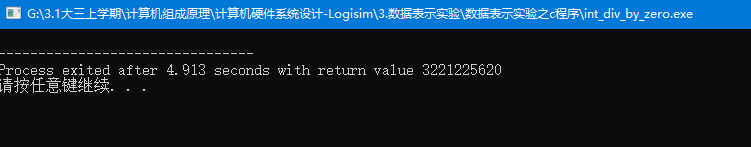
根据上面的规则,此处不需要进位,将第25位置零,即得到最终的16777216

(16777217)10 =(1 0000 0000 0000 0000 0000 0001)2

≈(1 0000 0000 0000 0000 0000 0001)2

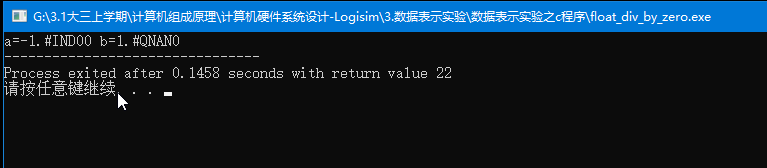
=(16777216)10

（3）



除数不能为零，除数为0，会导致程序出现错误，导致程序异常退出。

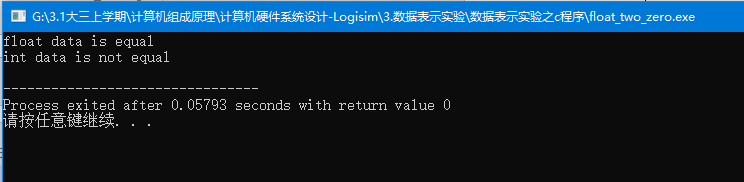
（4）

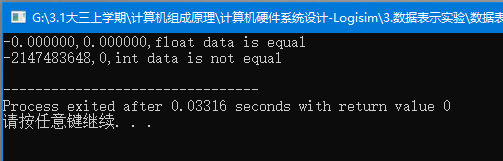


-1.#IND000值,代表无穷小

1.#QNAN是一个打印呈现，QNAN是指Quiet Not a Number，类似的浮点错误还有SNaN（Signaling Not a Number），通常如0.0/0.0、给负数开平方根等溢出或无效运算就会产生一个NAN结果的表示值。

（5）

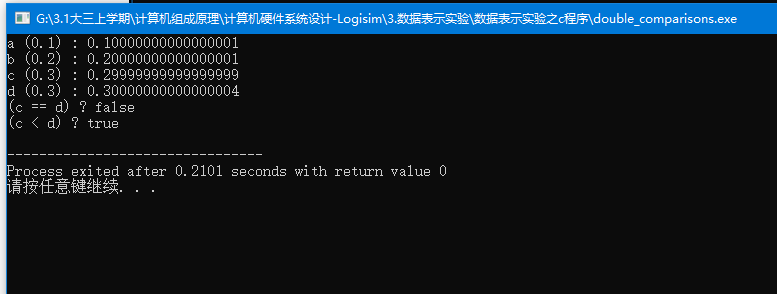




float 正零和负零在有一个专门的寄存器将其标记为0，都是0，所以相等

int 一个是-2147483648另一个是0，所以不相等。

（6）



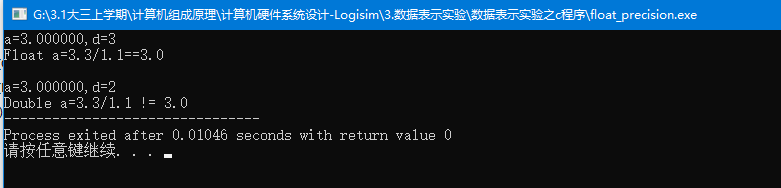
double存储有精度问题，0.3存储为0.29999999999999999

0.1存储为0.10000000000000001；0.2存储为0.20000000000000001

相加后再次出现精度问题，运算结果为：0.30000000000000004

所以二者不相等。

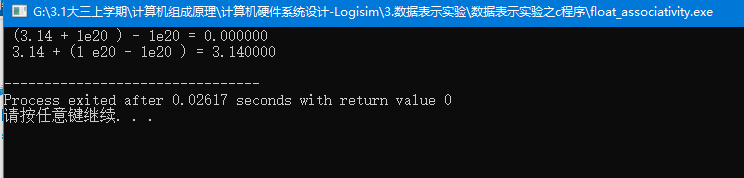
（7）



float的计算结果为3.0000几，在转换为int型是变为3.

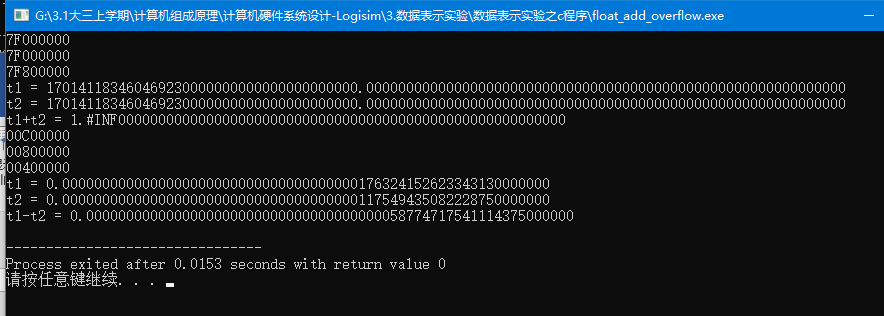
而double的计算结果为2.9999几，虽然结果也是近似接近3，但是int的转换机制是只保留整数位，所以double转int，结果就变成了2。

（8）



3.14+1e20加完之后由于精度表示问题，3.14丢失了，所以在减1e20之后等于0。第二个是3.14+0等于3.14。

（9）



float运算，t1+t2后，阶码全1，发生上溢。

## 五、实验思考

(1)银行如何表示一角钱、一分钱?如果用浮点数表示可能会存在什么问题?

银行在表示一角钱、一分钱是都用整数再加上相应的单位来表示，例如0.1元=1角。如果用浮点数表示的话，由于其误差会出现一些不可控的错误问题。

(2)在编写C语言程序时,浮点数是否适合进行比较?为什么?

不合适，浮点数由于其精度原因会存在误差，这些误差在数值很接近的时候会导致程序结果出现错误，就像第6题中的0.1+0.2！=0.3一样，这会使程序出现不可预知的错误。