\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Report: HW4

Author: E94074029 江羿賢 <e94074029@gs.ncku.edu.tw>

Class: 資訊系111乙班

Description:

這次作業學到了很多東西，有利用整數指標來取得浮點數的值，以及使用浮點數指標來取得整數的值，並且能夠更了解IEEE 754的表示方式。在程式的內部中也使用了左移<<跟右移>>來使那些散亂的數字移動到正確的位置並輸出或是儲存，也有利用了switch去使得選擇更為容易，就不用繁雜的if-else來做判斷，也使得code會更為清楚

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Code:

#include<stdio.h>

#include<stdlib.h>

void float\_to\_bit(float);

void double\_to\_bit(double);

void bit\_to\_float(char \*,char \*,char \*);

void bit\_to\_double(char \*,char \*,char \*);

unsigned long long power(int,int);

int main(int argc,char \*argv[])

{

int choose=atoi(argv[1]);//choose what to use it

float a=atof(argv[2]);//the input float

double b=atof(argv[2]);//the input double

switch(choose)//choose it

{

case 1://do float turn to bit pattern

float\_to\_bit(a);

break;

case 2://do double turn to bit pattern

double\_to\_bit(b);

break;

case 3://do bit pattern turn to float

bit\_to\_float(argv[2],argv[3],argv[4]);

break;

case 4://do bit pattern turn to double

bit\_to\_double(argv[2],argv[3],argv[4]);

break;

}

}

void float\_to\_bit(float input)

{

int bit=32-1;//have 32 bit so 0to31

unsigned int n;

n=\*(unsigned int \*)&input;//integer pointer to float

int count=0;//count print how many times

for(int i=bit;i>=0;i--,count++)

{

if(count==1||count==9)//sign 1 bit exp 8bit

printf(" ");//when it happen on 1or9 print" "

if((n>>i)&1)//print first bit to last bit

printf("1");//by using >> go to front first

else

printf("0");

}

printf("\n");

}

void double\_to\_bit(double input)

{

int bit=64-1;//have 64 bit so 0to63

unsigned long long n;

n=\*(unsigned long long \*)&input;//integer pointer to double

int count=0;

for(int i=bit;i>=0;i--,count++)

{

if(count==1||count==12)//sign 1 bit exp 11 bit

printf(" ");

if((n>>i)&1)//print first bit to last bit

printf("1");//by using >>go to front first

else

printf("0");

}

printf("\n");

}

void bit\_to\_float(char \*s, char \*e, char \*m)

{

unsigned int sign=0;

unsigned int exp=0;

unsigned int mat=0;

int ee=7;// exp 8 bit

int mm=22;// mat 23 bit

if(s[0]==48)//to see sign

sign=0;//set to 0

else

sign=1;//set to 1

for (int i=0;i<8;i++,ee--)//form 0 to 7 check exp

{

if (e[i]==49)//count only it's 1

{

exp+=power(2,ee);//to see it's exp and plus it

}

}

for (int j=0;j<23;j++,mm--)//form 0 to 23 check mat

{

if (m[j]==49)//count only it's 1

{

mat+=power(2,mm);//to see it's exp and plus it

}

}

unsigned long long n;//decalre the number to store it

n=(sign<<31)+(exp<<23)+(mat);//using shift to put correct site

printf("%f",\*(float \*)&n);//float pointer to integert

printf("\n");

}

void bit\_to\_double(char \*s, char \*e, char \*m)

{

unsigned long long sign=0;

unsigned long long exp=0;

unsigned long long mat=0;

int ee=10;//exp 11 bit

int mm=51;//mat 52 bit

if(s[0]==48)//to see the sign is 0 or 1

sign=0;

else

sign=1;

for (int i=0;i<11;i++,ee--)//form 0 to 10 check exp

{

if (e[i]==49)//count only it's 1

{

exp+=power(2,ee);//to see it's exp and plus it

}

}

for (int j=0;j<52;j++,mm--)//form 0 to 51 to check mat

{

if (m[j]==49)//count only it's 1

{

mat+=power(2,mm);//to see it's exp and plus it

}

}

unsigned long long n;

n=(sign<<63)+(exp<<52)+(mat);//using shift put correct site

printf("%lf",\*(double \*)&n);//double pointer to integer

printf("\n");

}

unsigned long long power(int x,int y)//power function

{

unsigned long long total=1;

while(y--)//do y times

{

total\*=x;

}

return total;//return total

}

Compilation:

gcc hw4.c -o hw4

Execution:

./hw4 1 85.125

./hw4 2 85.125

./hw4 3 0 10000101 01010100100000000000000

./hw4 4 0 10000000101 0101010010000000000000000000000000000000000000000000

Output:

0 10000101 01010100100000000000000

0 10000000101 0101010010000000000000000000000000000000000000000000

85.125000

85.125000

2.1

1.175494350822287507968736537222245677818665556772087521508751706278417259454727172851560500000000000000000000000000000000e-38f

並不是最小的浮點數，而在IEEE 754 的表示法中，

最小的浮點數為0 00000001 00000000000000000000000

，而讓這個下去跑程式並把顯示位數弄大一點，得到最小的浮點數會是1.1754943508222875079687365372222456778186655567720875215087517062784172594547271728515625000000000000e-38

而在非零的最後，題目寫的是…605，但最小的是…625

所以它不是最小的浮點數。

2.2

f=0.0的bit pattern為

0 00000000 00000000000000000000000

2.3

Run 的結果為

1.1754943508222875079687365372222456778186655567720875215087517062784172594547271728515625000000000000e-38 = 1.1754943508222875079687365372222456778186655567720875215087517062784172594547271728515625000000000000e-38

因為f1跟f2都已經比最小的float還要小，所以float只能把他們存成最小，所以f1才會跟f2相等

2.4

0 11111111 00000000000000000000000 為inf

0 11111111 00000000000000000000001 為nan

2.5

i 0 10000000 10010010000111111011011

為 3.14159… 猜測應該是pi

ii 0 01111101 01010101010101010101011

為 0.33333… 猜測應該是1/3