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

Report: hw4

Author: F74086381 蘇恩質 <grgy078033@gmail.com>

Class: 112資訊系 (乙班)

Description:

因助教的工作站貌似無法compile <math.h>的函式，所以先自訂一個power()函式來代替<math.h>中的pow()，方便後面進行次方運算。

由於浮點數沒辦法直接跟integer做AND計算，因此用cast和pointer的技巧來取得浮點數的address再從address中取值出來做計算。

設一個mask = 0x80000000(做float時)或是

mask = 0x8000000000000000(做double時)，也就是最高位元是1剩下都是0，與ip做for迴圈AND運算，將x從最後一個bit印到第一個bit，得到floating point number的bit pattern。

把bit pattern分成Signed,exp,fraction三個部分，然後用IEEE754的算法回推原本的floating point number。

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

#include <stdio.h>

#include <stdlib.h>

double power(int x, int y)

{

int i;

double result = 1.0;

if(y > 0)

{

for(i = 0; i < y; i++)

result \*= x;

}

else if(y < 0)

{

int z = 0 - y;

for(i = 0; i < z; i++)

result /= x;

}

return result;

}

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

{

int z = atoi(argv[1]);

if(z == 1)

{

float x = atof(argv[2]);

int ip, i;

ip = \*(int\*)&x;

unsigned mask = 0x80000000;

for(i = 0; i < 32; i++)

{

if(i == 0)

{

if(x >= 0)

printf("0 ");

else

printf("1 ");

}

if(i >= 1 && i < 9)

{

printf("%d", ((ip&mask)>>(31 - i)));

if(i == 8)

printf(" ");

}

if(i >= 9 && i < 32)

{

printf("%d", ((ip&mask)>>(31 - i)));

if(i == 31)

printf("\n");

}

mask >>= 1;

}

}

else if(z == 2)

{

double x = atof(argv[2]);

long long ip;

int i;

ip = \*(long long\*)&x;

unsigned long long mask = 0x8000000000000000;

for(i = 0; i < 64; i++)

{

if(i == 0)

{

if(x >= 0)

printf("0 ");

else

printf("1 ");

}

if(i >= 1 && i < 12)

{

printf("%lld", ((ip&mask)>>(63 - i)));

if(i == 11)

printf(" ");

}

if(i >= 12 && i < 64)

{

printf("%lld", ((ip&mask)>>(63 - i)));

if(i == 63)

printf("\n");

}

mask >>= 1;

}

}

else if(z == 3)

{

char Signed, exp[8], fraction[23];

int i, binary\_exp = 0;

float binary\_fra = 1.0, answer = 0.0;

sprintf(&Signed, "%c", argv[2]);

sprintf(exp, "%8s", argv[3]);

sprintf(fraction, "%23s", argv[4]);

int x = atoi(argv[2]);

for(i = 7;i >= 0; i--)

binary\_exp += ( power(2, 7-i) \* (exp[i] - '0') );

for(i = 0; i < 23; i++)

binary\_fra += ( power(2, -1 \* (i+1) ) \* (fraction[i] - '0') );

answer = power(2, binary\_exp - 127) \* binary\_fra;

if(x == 0)

{

printf("");

printf("%f\n", answer);

}

else if(x == 1)

{

printf("-");

printf("%f\n", answer);

}

}

else if(z == 4)

{

char Signed, exp[11], fraction[52];

int i, binary\_exp = 0;

float binary\_fra = 1.0, answer = 0.0;

sprintf(&Signed, "%c", argv[2]);

sprintf(exp, "%11s", argv[3]);

sprintf(fraction, "%52s", argv[4]);

int x = atoi(argv[2]);

for(i = 10; i >= 0; i--)

binary\_exp += ( power(2, 10-i) \* (exp[i] - '0') );

for(i = 0; i < 52; i++)

binary\_fra += ( power(2, -1 \* (i+1) ) \* (fraction[i] - '0') );

answer = power(2, binary\_exp - 1023) \* binary\_fra;

if(x == 0)

{

printf("");

printf("%f\n", answer);

}

else if(x == 1)

{

printf("-");

printf("%f\n", answer);

}

}

return 0;

}

Compilation:

gcc -o hw4 hw4.c

^

Execution:

***Example 1:***

./hw4 1 11.25

***Example 2:***

./hw4 2 11.25

***Example 3:***

./hw4 3 0 10000010 01101000000000000000000

***Example 4:***

./hw4 4 0 10000000010 0110100000000000000000000000000000000000000000000000

Output:

***Example 1:***

0 10000010 01101000000000000000000

***Example 2:***

0 10000000010 0110100000000000000000000000000000000000000000000000

***Example 3:***

11.250000

***Example 4:***

11.250000

2

2-1 是，首先，exponent的部分會是127+real exponent，而若exponent的bit pattern是00000000且mantissa也都是0的話，此數即為0，因此最小的floating point number應是建立於exponent的bit pattern為00000001的情況下，也就是real exponent = -126且matissa也都為0時，因此得出的值就是:

1 \* 2-126 =1.1754943508222875079687365372222456778186655567720 87521508751706278417259454727172851560500000000000 000000000000000000000e-38f

2-2 0 00000000 00000000000000000000000

2-3 std::numeric\_limits<float>::max\_digits10的值為9，即float的最大表示區分度，也就是說從第9位開始的數皆不影響float的所有值，由此可知，在1.1754943(8位)後的所有數字都無法用來區分float的所有值，因此f1 = f2

2-4 Infinity: 0 11111111 00000000000000000000000

NaN: 0 11111111 10000000000000000000000

* 1. i. 3.141593(取到小數點後第六位)

ii. 0.333333((取到小數點後第六位)