COMP 350: Assignment #2.
Monorina Mukhopadhyay (ID: 260364335)

Question 1.

The code is attached at the end of this file, as A2part1.c
To solve this problem, the sequence of floating point numbers are multiplied together, and then if there is overflow (product is inf) then the numbers are divided by 10 (and K is set to the number of divisions) and the product is recalculated until there is no overflow.
In case of underflow (product is zero when none of the numbers are zero), the numbers are multiplied by 10 until there is no underflow (and K is set to the number of multiplications).
I tested the results with three sets of numbers, one causing overflow, one causing underflow, and one resulting in a number allowed in IEEE single format.
When scaling the numbers up or down, I scale all numbers in the sequence by the same factor of 10. This could also be done by scaling the numbers by 10 one at a time, but that would take more iterations and increase computation time.
It can be seen that single format floating point multiplication in C causes significant rounding errors that could lead to inaccuracies in the final results.

Result:
Overflow:

```
The numbers are:
Number 1 1.000000e+25
Number 2 1.000000e+25
Number 3 1.000000e+12
The product is: 9.999998e+37 times 10 to the power of 24
```

Underflow

```
The numbers are:
Number 1 1.000000e-30
Number 2 1.000000e-10
Number 3 1.000000e-12
The product is: 9.949219e-44 times 10 to the power of -9
```

And a result within range

```
The numbers are:
Number 1 1.000000e+15
Number 2 1.000000e+05
Number 3 1.000000e+01
The product is: 1.000000e+21 times 10 to the power of 0
```

Q2.
a)
This is implemented in function part2a in A2part2.c and is printed below. The result is given below:

```
x1=8.28427125e-01 x1-log(x0+1)= 1.35279944e-01
x2=7.56828460e-01 x2-log(x0+1)= 6.36812795e-02
x3=7.24061861e-01 x3-log(x0+1)= 3.09146808e-02
```

x4=7.08380519e-01 x4-log(x0+1)= 1.52333383e-02
x5=7.00708757e-01 x5-log(x0+1)= 7.56157637e-03
x6=6.96914307e-01 x6-log(x0+1)= 3.76712675e-03
x7=6.95027342e-01 x7-log(x0+1)= 1.88016188e-03
x8=6.94086413e-01 x8-log(x0+1)= 9.39232292e-04
x9=6.93616585e-01 x9-log(x0+1)= 4.69404199e-04
x10=6.93381830e-01 x10-log(x0+1)= 2.34649140e-04
x11=6.93264492e-01 x11-log(x0+1)= 1.17311333e-04
x12=6.93205833e-01 x12-log(x0+1)= 5.86523576e-05
x13=6.93176506e-01 x13-log(x0+1)= 2.93253508e-05
x14=6.93161843e-01 x14-log(x0+1)= 1.46624681e-05
x15=6.93154512e-01 x15-log(x0+1)= 7.33118049e-06
x16=6.93150846e-01 x16-log(x0+1)= 3.66558214e-06
x17=6.93149013e-01 x17-log(x0+1)= 1.83279752e-06
x18=6.93148097e-01 x18-log(x0+1)= 9.16405215e-07
x19=6.93147639e-01 x19-log(x0+1)= 4.58252716e-07
x20=6.93147410e-01 x20-log(x0+1)= 2.29147363e-07
x21=6.93147295e-01 x21-log(x0+1)= 1.14594686e-07
x22=6.93147238e-01 x22-log(x0+1)= 5.77840088e-08
x23=6.93147210e-01 x23-log(x0+1)= 2.98443316e-08
x24=6.93147197e-01 x24-log(x0+1)= 1.68058155e-08
x25=6.93147190e-01 x25-log(x0+1)= 9.35523492e-09
x26=6.93147182e-01 x26-log(x0+1)= 1.90465432e-09
x27=6.93147182e-01 x27-log(x0+1)= 1.90465432e-09
x28=6.93147182e-01 x28-log(x0+1)= 1.90465432e-09
x29=6.93147182e-01 x29-log(x0+1)= 1.90465432e-09
x30=6.93147182e-01 x30-log(x0+1)= 1.90465432e-09
x31=6.93147182e-01 x31-log(x0+1)= 1.90465432e-09
x32=6.93146706e-01 x32-log(x0+1)= -4.74932504e-07
x33=6.93145752e-01 x33-log(x0+1)= -1.42860682e-06
x34=6.93145752e-01 x34-log(x0+1)= -1.42860682e-06
x35=6.93145752e-01 x35-log(x0+1)= -1.42860682e-06
x36=6.93145752e-01 x36-log(x0+1)= -1.42860682e-06
x37=6.93145752e-01 x37-log(x0+1)= -1.42860682e-06
x38=6.93115234e-01 x38-log(x0+1)= -3.19461849e-05
x39=6.93115234e-01 x39-log(x0+1)= -3.19461849e-05
x40=6.93115234e-01 x40-log(x0+1)= -3.19461849e-05
x41=6.92871094e-01 x41-log(x0+1)= -2.76086810e-04
x42=6.92382812e-01 x42-log(x0+1)= -7.64368060e-04
x43=6.91406250e-01 x43-log(x0+1)= -1.74093056e-03
x44=6.91406250e-01 x44-log(x0+1)= -1.74093056e-03
x45=6.87500000e-01 x45-log(x0+1)= -5.64718056e-03
x46=6.87500000e-01 x46-log(x0+1)= -5.64718056e-03
x47=6.87500000e-01 x47-log(x0+1)= -5.64718056e-03
x48=6.87500000e-01 x48-log(x0+1)= -5.64718056e-03
x49=6.25000000e-01 x49-log(x0+1)= -6.81471806e-02
x50=5.00000000e-01 x50-log(x0+1)= -1.93147181e-01
x51=5.00000000e-01 x51-log(x0+1)= -1.93147181e-01
x52=0.00000000e+00 x52-log(x0+1)= -6.93147181e-01
x53=0.00000000e+00 x53-log(x0+1)= -6.93147181e-01
x54=0.00000000e+00 x54-log(x0+1)= -6.93147181e-01
x55=0.00000000e+00 x55-log(x0+1)= -6.93147181e-01
x56=0.00000000e+00 x56-log(x0+1)= -6.93147181e-01
x57=0.00000000e+00 x57-log(x0+1)= -6.93147181e-01

```
x58=0.00000000e+00 x58-log(x0+1)= -6.93147181e-01
x59=0.00000000e+00 x59-log(x0+1)= -6.93147181e-01
x60=0.00000000e+00 x60-log(x0+1)= -6.93147181e-01
```

As n becomes larger, the 2^{-n} term becomes smaller and the $\sqrt{2^{-n}x_n + 1} - 1$ term gets closer to zero. Therefore, this small difference is amplified in the x_{n+1} term due to the presence of the 2^{n+1} and the difference of $x_n - \log(x_0 + 1)$ becomes larger as n increases. Finally as n increases, (for n>51) x_n becomes zero and the difference is constant.

b) In order to prevent the inaccuracies in Part a), the subtraction term needs to be eliminated. This is done by multiplying numerator and denominator by $\sqrt{2^{-n} + 1} + 1$
The equation is now:

$$x_n = 2^{n+1}\sqrt{2^{-n}x_n + 1} - 1\,\frac{\sqrt{2^{-n}x_n + 1} + 1}{\sqrt{2^{-n}x_n + 1} + 1} = \frac{2x_n}{\sqrt{2^{-n}x_n + 1} + 1}$$

The result is now given as:

```
x1=8.28427125e-01 x1-log(x0+1)= 1.35279944e-01
x2=7.56828460e-01 x2-log(x0+1)= 6.36812795e-02
x3=7.24061861e-01 x3-log(x0+1)= 3.09146808e-02
x4=7.08380519e-01 x4-log(x0+1)= 1.52333383e-02
x5=7.00708757e-01 x5-log(x0+1)= 7.56157637e-03
x6=6.96914307e-01 x6-log(x0+1)= 3.76712675e-03
x7=6.95027342e-01 x7-log(x0+1)= 1.88016188e-03
x8=6.94086413e-01 x8-log(x0+1)= 9.39232292e-04
x9=6.93616585e-01 x9-log(x0+1)= 4.69404199e-04
x10=6.93381830e-01 x10-log(x0+1)= 2.34649140e-04
x11=6.93264492e-01 x11-log(x0+1)= 1.17311333e-04
x12=6.93205833e-01 x12-log(x0+1)= 5.86523580e-05
x13=6.93176506e-01 x13-log(x0+1)= 2.93253519e-05
x14=6.93161843e-01 x14-log(x0+1)= 1.46624692e-05
x15=6.93154512e-01 x15-log(x0+1)= 7.33118289e-06
x16=6.93150846e-01 x16-log(x0+1)= 3.66557852e-06
x17=6.93149013e-01 x17-log(x0+1)= 1.83278603e-06
x18=6.93148097e-01 x18-log(x0+1)= 9.16392207e-07
x19=6.93147639e-01 x19-log(x0+1)= 4.58195902e-07
x20=6.93147410e-01 x20-log(x0+1)= 2.29097900e-07
x21=6.93147295e-01 x21-log(x0+1)= 1.14548938e-07
x22=6.93147238e-01 x22-log(x0+1)= 5.72744656e-08
x23=6.93147209e-01 x23-log(x0+1)= 2.86372320e-08
x24=6.93147195e-01 x24-log(x0+1)= 1.43186157e-08
x25=6.93147188e-01 x25-log(x0+1)= 7.15930781e-09
x26=6.93147184e-01 x26-log(x0+1)= 3.57965391e-09
x27=6.93147182e-01 x27-log(x0+1)= 1.78982695e-09
x28=6.93147181e-01 x28-log(x0+1)= 8.94913477e-10
x29=6.93147181e-01 x29-log(x0+1)= 4.47456738e-10
x30=6.93147181e-01 x30-log(x0+1)= 2.23728369e-10
x31=6.93147181e-01 x31-log(x0+1)= 1.11864074e-10
x32=6.93147181e-01 x32-log(x0+1)= 5.59321478e-11
x33=6.93147181e-01 x33-log(x0+1)= 2.79661849e-11
x34=6.93147181e-01 x34-log(x0+1)= 1.39831480e-11
x35=6.93147181e-01 x35-log(x0+1)= 6.99162950e-12
x36=6.93147181e-01 x36-log(x0+1)= 3.49587026e-12
```

```
x37=6.93147181e-01 x37-log(x0+1)= 1.74804615e-12
x38=6.93147181e-01 x38-log(x0+1)= 8.74189610e-13
x39=6.93147181e-01 x39-log(x0+1)= 4.37205827e-13
x40=6.93147181e-01 x40-log(x0+1)= 2.18602914e-13
x41=6.93147181e-01 x41-log(x0+1)= 1.09356968e-13
x42=6.93147181e-01 x42-log(x0+1)= 5.47339951e-14
x43=6.93147181e-01 x43-log(x0+1)= 2.73114864e-14
x44=6.93147181e-01 x44-log(x0+1)= 1.37667655e-14
x45=6.93147181e-01 x45-log(x0+1)= 6.99440506e-15
x46=6.93147181e-01 x46-log(x0+1)= 3.66373598e-15
x47=6.93147181e-01 x47-log(x0+1)= 1.99840144e-15
x48=6.93147181e-01 x48-log(x0+1)= 1.11022302e-15
x49=6.93147181e-01 x49-log(x0+1)= 7.77156117e-16
x50=6.93147181e-01 x50-log(x0+1)= 4.44089210e-16
x51=6.93147181e-01 x51-log(x0+1)= 4.44089210e-16
x52=6.93147181e-01 x52-log(x0+1)= 4.44089210e-16
x53=6.93147181e-01 x53-log(x0+1)= 4.44089210e-16
x54=6.93147181e-01 x54-log(x0+1)= 4.44089210e-16
x55=6.93147181e-01 x55-log(x0+1)= 4.44089210e-16
x56=6.93147181e-01 x56-log(x0+1)= 4.44089210e-16
x57=6.93147181e-01 x57-log(x0+1)= 4.44089210e-16
x58=6.93147181e-01 x58-log(x0+1)= 4.44089210e-16
x59=6.93147181e-01 x59-log(x0+1)= 4.44089210e-16
x60=6.93147181e-01 x60-log(x0+1)= 4.44089210e-16
```

In this case, the error decreases with the iterations and the cancellation error is avoided.

The code:
A2part1.c

```c
#include <stdio.h>
#include <unistd.h>
#include <string.h>
#include <math.h>
#include <stdlib.h>

#define MAX_N 60
#define SUCCESS 1

void part1(float num[], int length)
{

    float product;
    int K = 0;
    int flag = 0;
    int zero_flag = 0;
    printf("The numbers are:\n");
    for (int i = 0; i < length ; i++)
    {
        printf("Number %d %e \n",i + 1,num[i]);
        if (num[i] == 0)
            zero_flag = 1;
    }
    if(zero_flag == 1)
        printf("The result is zero times K to the power of 0\n");

    while(!flag)
    {
        product = num[0];
        for(int i = 1; i < length; i++)
        {
            product *= num[i];
        }
        // If the numbers overflow, divide by 10 to scale
        // and retry the multiplication
        if(product == INFINITY){
            for(int i = 0; i < length; i++)
            {
                num[i] /= 10;
            }
            K += length;
        }
        else if(product == 0)
        {
            for(int i = 0; i < length; i++)
            {
                num[i] *= 10;
            }
            K -= length;
        }
        else{
            flag = 1;
            printf("The product is: %.6e times 10 to the power of %d \n", product,K);
        }
        if(abs(K) == INFINITY)
        {
            flag = 1;
        }
    }

}

int main(void)
{
```

```
65      float num[3] = {powf(10,30), powf(10,30), powf(10,1)};
66      part1(num,3);
67      return SUCCESS;
68  }
```

Question 2.
Part a) is implemented in part2a(void) and Part b) is implemented in part2b(void)


```
 1  #include <stdio.h>
 2  #include <unistd.h>
 3  #include <string.h>
 4  #include <math.h>
 5
 6  #define MAX_N 60
 7  #define SUCCESS 1
 8
 9  void part2a (void)
10  {
11      double x_prev;
12      double x_next;
13      double x_0 = 1.0f;
14      x_prev = x_0; //x_0 is 1
15      double n_2poweri = 1.0f; // this is to keep track of the 2^i term
16                               // to do 2^i on doubles since it is not
17                               // allowed in C otherwise and 2^0 = 1
18
19      for(int i = 1; i <= MAX_N; i++)
20      {
21          x_next = 2*n_2poweri*(sqrt(1+(1/n_2poweri)*x_prev) - 1);
22          x_prev = x_next; //update the x_n for the next iteration
23          double diff = x_next - log(2);
24          n_2poweri = 2*n_2poweri;
25          printf("%s%d%s%.8e %s%d%s %.8e \n", "x",i,"=",x_next,"x",i,"-log(x0+1)=",diff);
26      }
27  }
28  void part2b (void)
29  {
30      double x_prev;
31      double x_next;
32      double x_0 = 1.0f;
33      x_prev = x_0; //x_0 is 1
34      double n_2poweri = 1.0f;
35      for(int i = 1; i <= MAX_N; i++)
36      {
37          x_next = 2*x_prev / (sqrt(1+(1/n_2poweri)*x_prev) + 1 );
38          x_prev = x_next;
39          double diff = x_next - log(2);
40          n_2poweri = 2*n_2poweri;
41          printf("%s%d%s%.8e %s%d%s %.8e \n", "x",i,"=",x_next,"x",i,"-log(x0+1)=",diff);
42      }
43  }
44
45  int main(void)
46  {
47      part2a();
48      part2b();
49      return SUCCESS;
50  }
```