1a. For $t \in [-1, 1)$ Taylor's theorem implies

$$\log(1-t) = -\Big(\sum_{j=1}^{n} \frac{t^j}{j}\Big) - \frac{1}{1-c}\Big(\frac{t^{n+1}}{n+1}\Big)$$

where c is an unknown constant between 0 and t. Let $t = 0.5$. Solve for n to ensure that the Taylor polynomial of degree n approximates $\log(0.5)$ with a relative error less than 5×10^{-16}.

Let $y = \log(0.5)$ and $y_A = T_n(0.5)$ be the approximation of y given by the Taylor polynomial of degree n. Then for some c between 0 and 0.5 we obtain

$$|\mathrm{Rel}(y_A)| = \Big|\frac{\log(0.5) - T_n(0.5)}{\log(0.5)}\Big| = \frac{0.5^{n+1}}{|\log(0.5)|(1-c)(n+1)} \le \frac{0.5^n}{0.6931(n+1)}$$

since $|\log(0.5)| \ge 0.6931$. We now choose n to be the least integer such that

$$\frac{0.5^n}{0.6931(n+1)} \le 5 \times 10^{-16}$$

to obtain n that ensures the approximation meets the error tolerance. The Maple script

```
restart;
for n from 1 to 50
do
   if 0.5^n/0.6931/(n+1)<= 5*10^(-16)
   then
      printf("The least n is n=%g\n",n);
      break;
   end
end;
```

with output

```
The least n is n=46
```

indicates the minimum value of n.

1b. [Extra Credit and Math/CS 666] Show that

$$2^{n+1}(n+1) \geq 10^{16}$$

implies the relative error of the Taylor polynomial satisfies

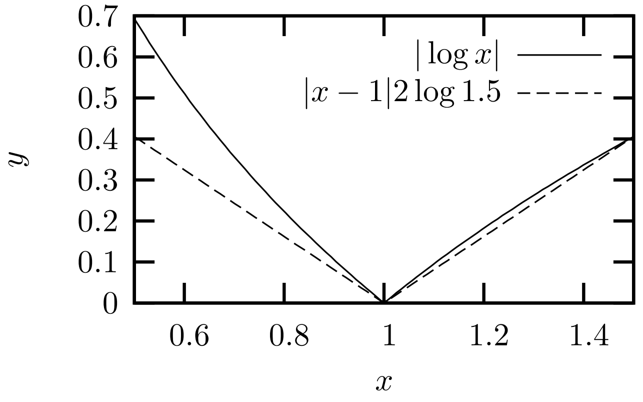
$$|\mathrm{Rel}(T_n(1-x))| = \left|\frac{\log x - T_n(1-x)}{\log x}\right| \leq 5 \times 10^{-16}$$

for all $x \in [0.5, 1.5]$.

Since

$$\frac{d^2 \log x}{dx^2} = \frac{-1}{x^2} < 0 \qquad \text{for all} \qquad x > 0,$$

then $\log x$ is a concave function. Therefore $|\log x| \geq |x-1| 2 \log 1.5$. This is further illustrated in the graph



Since $\log 1.5 \geq 0.4$ we obtain for some c between -0.5 and 0.5 that

$$|\mathrm{Rel}(T_n(x-1))| = \frac{|1-x|^{n+1}}{|\log(x)|(1-c)(n+1)} \leq \frac{|1-x|^n}{|\log 1.5|(n+1)} \leq \frac{2^{-n}}{0.4(n+1)}.$$

Therefore, to ensure the error tolerance it is sufficient that

$$\frac{2^{-n}}{0.4(n+1)} \leq 5 \times 10^{-16}$$

or equivalently

$$2^{n+1}(n+1) \geq 10^{16}.$$

Solving for the least n using a program similar to part 1a gives $n = 47$.

1c. Write a program that uses a suitable Taylor polynomial $T_n(x)$ to approximate $\log x$ for $x \in [0.5, 1.5]$. Make your computation as accurate as possible. Compare your results to the builtin log function and compute the relative error for $x = 0.5$, $x = 1.03$ and $x = 1.4$.

We choose $n = 47$ so the Taylor polynomial will approximate the log function good to within 16 significant digits. The C code

```c
/*  m1c.c -- Compute natural logarithm by Taylor series
    By Eric Olson November 28, 2008 for Math/CS 466/666 */

#include <stdio.h>
#include <math.h>

const int n=47;
double tlog(double x){
    double t=1.0-x;
    double y=1.0/n;
    int j;
    for(j=n-1;j>=1;j--){
        y=y*t+1.0/j;
    }
    return -t*y;
}

double X[]={0.5, 1.03, 1.4};
int main(){
    int i;
    printf("Math/CS 466/666 Programming Assignment 01 Part 1c.\n\n");
    printf("Computing with n=%d...\n",n);
    printf(" %5s  %22s  %22s  %20s\n",
        "x","log(x)","Tn(1-x)","Rel");
    for(i=0;i<sizeof(X)/sizeof(double);i++){
        double x=X[i];
        double ya=tlog(x),y=log(x);
        double rel=(y-ya)/y;
        printf(" %5g  %22.15e  %22.15e  %20.13e\n",
            x,y,ya,rel);
    }
    return 0;
}
```

produces the output

```
Math/CS 466/666 Programming Assignment 01 Part 1c.

Computing with n=47...
     x                  log(x)                 Tn(1-x)                   Rel
   0.5  -6.931471805599453e-01  -6.931471805599452e-01   1.6017132519075e-16
  1.03   2.955880224154443e-02   2.955880224154443e-02   1.1737440927418e-16
   1.4   3.364722366212129e-01   3.364722366212129e-01   0.0000000000000e+00
```

The relative error is less than 5×10^{-16} for all tested values as expected.

1d. Use Newton's method to find $\sqrt{2}$ by solving $x^2 - 2 = 0$. Then use the identity $\log 2 = 2 \log \sqrt{2}$ to compute $\log 2$ as accurately as possible.

Newton's iteration is

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

and choose the starting value $x_0 = 1$. The C code

```
/*  m1d.c -- Compute log(2) using identitities
    By Eric Olson November 28, 2008 for Math/CS 466/666 */

#include <stdio.h>
#include <math.h>

extern double tlog(double x);

double nsqrt2(){
    double xn=1.0;
    int n;
    for(n=0;n<100;n++){
        double xnp1=(xn*xn/2.0+1.0)/xn;
        if(fabs(xnp1-xn)<5e-16) return xnp1;
        xn=xnp1;
    }
    printf("nsqrt2: failed to converge!\n");
    return xn;
}

int main(){
    double s2=nsqrt2();
    double rs2=(sqrt(2.0)-s2)/sqrt(2.0);
    double l2=2.0*tlog(s2);
    double rl2=(log(2.0)-l2)/log(2.0);
    printf("Math/CS 466/666 Programming Assignment 01 Part 1d.\n\n");
    printf("   sqrt(2) =%22.15e  Rel =%22.15e\n",s2,rs2);
    printf("    log(2) =%22.15e  Rel =%22.15e\n",l2,rl2);
    return 0;
}
```

produces the output

```
Math/CS 466/666 Programming Assignment 01 Part 1d.

   sqrt(2) = 1.414213562373095e+00  Rel = 0.000000000000000e+00
    log(2) = 6.931471805599455e-01  Rel =-3.203426503814918e-16
```

The square root $\sqrt{2}$ computed by Newtons method matches exactly the value returned by the builtin function. The computation of $\log 2$ has relative error is less that 5×10^{-16}.

1e. For $x > 0$ let k be the unique integer such that $2^{k-1} < x \leq 2^k$. Let $w = x/2^k$ so that $0.5 < w \leq 1$. Use the identity $\log x = k \log 2 + \log w$ and parts 1c and 1d to create a program that computes $\log x$ for all values of $x > 0$. Compare your results to the builtin log function and compute the relative error for $x = 17$, $x = 1083$ and $x = 0.19$.

In order to find k we can use the exponent already present in the IEEE 754 floating point representation. Since bits 2–12 of a double precision variable contain the value of $k + 1023$ we can find k easily. The following C code assumes a little endian ordering of the bit fields as found on Intel microprocessors. Note that this code must be modified to work on big endian machines such as IBM PowerPC and Silicon Graphics/MIPS architectures.

The C code

```c
/*  m1e.c -- Compute log(x) using identitities
    By Eric Olson November 28, 2008 for Math/CS 466/666 */

#include <stdio.h>
#include <math.h>

extern double tlog(double x);
extern double nsqrt2();

typedef union {
    double a;
    struct {
        char m2[6];
        unsigned int m1:4;
        unsigned int e:11;
        unsigned int s:1;
    } b;
} doublele;

/*  Find k such that 2^(k-1)<x<=2^k and set w=x/2^k using specific
    information about the little endian IEEE 754 representation */

static double s2;
double t2log(double x){
    doublele w=(doublele)x;
    int k=w.b.e-1023;
    w.b.e=1023;
    return 2*k*tlog(s2)+tlog(w.a);
}

double X[]={17, 1083, 0.19};
int main(){
    double x=1.0/2;
    int i;
    s2=nsqrt2();
    printf("Math/CS 466/666 Programming Assignment 01 Part 1e.\n\n");
    printf(" %5s  %22s  %22s  %20s\n",
        "x","Builtin Log","My Log","Rel Error");
    for(i=0;i<sizeof(X)/sizeof(double);i++){
        double x=X[i];
        double ya=t2log(x),y=log(x);
        double rel=(y-ya)/y;
        printf(" %5g  %22.15e  %22.15e  %20.13e\n",
            x,y,ya,rel);
    }
    return 0;
}
```

produces the output

```
Math/CS 466/666 Programming Assignment 01 Part 1e.

      x             Builtin Log                 My Log            Rel Error
     17   2.833213344056216e+00   2.833213344056217e+00  -3.1348801231769e-16
   1083   6.987490247000991e+00   6.987490247000992e+00  -2.5421958050856e-16
   0.19  -1.660731206821651e+00  -1.660731206821651e+00  -2.6740583185642e-16
```

The values computed with the Taylor series agree with the builtin log function to 16 significant digits.