Int Assignment 2 Design Document

Little Slice of π

Dylan Do

CSE13S - Fall 2021

Purpose: The purpose of this assignment is to have the student implement two mathematical functions(e^x and $\sqrt{x}$) that would mimic the use of importing <math.h>. Once done, the student should create a scientific write up on the findings they find comparing their coded functions with the ones provided in the math library's.

Calculating e: Euler's number is an irrational mathematical constant that equates to around 2.71828. Here is the function to calculate the number:

$$e = \sum_{k=0}^{\infty} \frac{1}{k!} = 1 + \frac{1}{1} + \frac{1}{2} + \frac{1}{6} + \frac{1}{24} + \frac{1}{120} + \frac{1}{720} + \frac{1}{5040} + \frac{1}{40320} + \frac{1}{362880} + \frac{1}{3628800} + \cdots$$

Function provided from the assignment 2 pdf.

The amount of terms that is required to compute will be determined in experimenting as part of the assignment.

$$\frac{x^k}{k!} = \frac{x^{k-1}}{(k-1)!} \times \frac{x}{k}.$$

Function provided from the assignment 2 pdf.

Here is a refined version of the formula. According to the assignment pdf, it explains that this is simpler because the only computation required is x/k starting at k = !0 (which equates to 1). Then using this calculation and multiplying the next term. Done easel with simple for or while loop.

Calculating π: The assignment pdf highlights multiple functions that can be used to approximate π.

The Leibniz Formula:

$$p(n)=4\sum_{k=0}^{n}\frac{(-1)^k}{2k+1}=4\left(1-\frac{1}{3}+\frac{1}{5}-\frac{1}{7}+\frac{1}{9}-\cdots\right)=\left((-1)^n H_{\frac{n}{2}+\frac{1}{4}}-H_{\frac{n}{2}-\frac{1}{4}}\right)+\pi$$

Function provided from the assignment 2 pdf.

Unfortunately the assignment pdf states that this function is not reasonable for approximating π as it converges extremely slow.

The Madhava Series:

$$\sum_{k=0}^{\infty}\frac{(-3)^{-k}}{2k+1}=\sqrt{3}\tan^{-1}\frac{1}{\sqrt{3}}=\frac{\pi}{\sqrt{12}}$$

Function provided from the assignment 2 pdf.

This formula is stated to be related to $\tan^{-1} x$ in the assignment pdf. A more rapid converging version of this formula is:

$$p(n)=\sqrt{12}\sum_{k=0}^{n}\frac{(-3)^{-k}}{2k+1}=\sqrt{12}\left[\frac{1}{2}3^{-n-1}\left((-1)^n\Phi\left(-\frac{1}{3},1,n+\frac{3}{2}\right)+\pi 3^{n+\frac{1}{2}}\right)\right]$$

Function provided from the assignment 2 pdf.

The only issue now is to calculate √12, and using the library is prohibited. The Lerch transcendent (Φ) goes to zero at the limit, which gives the remaining term:

$$\frac{\pi}{2}3^{-n-1}3^{n+\frac{1}{2}}=\frac{\pi}{2\sqrt{3}}=\frac{\pi}{\sqrt{12}}$$

Function provided from the assignment 2 pdf.

The Wallis Series:

$$p(n) = 2\prod_{k=1}^{n}\frac{4k^2}{4k^2-1} = \frac{\pi\Gamma(n+1)^2}{\Gamma\left(n+\frac{1}{2}\right)\Gamma\left(n+\frac{3}{2}\right)}$$

Function provided from the assignment 2 pdf.

This formula is easy to calculate since the series is purely multiplicative.

Euler's Solution:

$$\sum_{k=1}^{\infty}\frac{1}{k^2} = \frac{1}{1^2}+\frac{1}{2^2}+\frac{1}{3^2}+\cdots = H_{\infty}^{(2)},$$

Function provided from the assignment 2 pdf.

This equation involves harmonics but gave birth to this series for approximating π.

$$p(n) = \sqrt{6\sum_{k=1}^{n}\frac{1}{k^2}}$$

Function provided from the assignment 2 pdf.

However, now there is a square root that must be solved.

The Bailey-Borwein-Plouffe Formula:

$$p(n) = \sum_{k=0}^{n}16^{-k}\left(\frac{4}{8k+1}-\frac{2}{8k+4}-\frac{1}{8k+5}-\frac{1}{8k+6}\right).$$

Function provided from the assignment 2 pdf.

According to the assignment pdf, it states that this formula is remarkably simple. Reducing it to the least number of multiplication, it can be rewritten in *Horner normal form*:

$$p(n) = \sum_{k=0}^{n}16^{-k}\times\frac{(k(120k+151)+47)}{k(k(k(512k+1024)+712)+194)+15}$$

Function provided from the assignment 2 pdf.

Viète's Formula: This formula is an infinite product of nested radicals that can be used for the approximation of π. However, the assignment pdf states that the previous formulas are known to produce an approximation with greater accuracy.

$$\frac{2}{\pi} = \prod_{k=1}^{\infty} \frac{a_i}{2}$$

Where $a^1 = \sqrt{2}$ and $a_k = \sqrt{(2+a_{k-1})}$for all $k > 1$.

Function provided from the assignment 2 pdf.

Fastest Series: According to the assignment pdf. This one is the most interesting.

$$\frac{1}{\pi} = \frac{2\sqrt{2}}{9801} \sum_{k=0}^{\infty} \frac{(4k)!(1101+26390k)}{(k!)^4 396^{4k}}$$

Function provided from the assignment 2 pdf.

Code: For this assignment, they ask to implement the mathematical functions e^x and $\sqrt{x}$, mimicking <math.h>, and using them to compute the fundamental constants *e* and π. The use of functions from <math.h> or a factorial function is strictly forbidden. There should be a total of 7 files. For the design pdf, I will head each pseudo code section with the file name and proceed to put the designated pseudo code for the file.

File e.c:

initialize an int variable called *num_terms* and set it to 0.

```
e(){
   initialize a double variable called return_value and set it to 1.
   initialize a double variable called power_num and set it to 1.
   for int i = 1; 1/factorial > epsilon; i++{
      power_num = power_num * i
      return_value += 1 / (factorial)
   }
   return return_value;
}
```

```
e_terms(){
    return num_terms;
}
```

File madhava.c:

initialize an int variable called *num_terms* and set it to 0.

```
pi_madhava(){
    initialize a float variable called return_value and set it to 0;
    initialize an int variable called top_half and set it to 1;
    for int i = 0;((1/top_half)/(2(i) +1)) > Epsilon ; i++{
        num_terms += 1
        if i != 0{
            top_half = top_half * -3
        }
        return_value = return_value + (1/top_half)/(2(i) +1)
    }
    return_value = sqrt_newton(12) * return_value;
    return return_value
}

madhava_terms(calculated_value){
        return num_terms;
}
```

File euler.c:

initialize an int variable called *num_terms* and set it to 0.

```
pi_euler(){
    initialize a double variable called return_value and set it to 0
```

```
    initialize a double variable called fraction and set it to 1
    for i = 1; i <= fraction > Epsilon; i++{
        num_terms += 1
        return_value = return_value + 1/(i*i);
    }
    return_value = return_value * 6;
    return_value = sqrt_newton(return_value);
    return return_value
}

euler_terms(calculated_value){
        return num_terms
}
```

File bbp.c:

```
initialize an int variable called num_terms and set it to 0.

pi_bbp(){
    initialize a double variable called return_value and set it to 0
    initialize a double variable called last_iteration and set it to -1
    initialize an int variable called power_term and set it to 1
    initialize an int variable called term_one and set it to 0
    initialize an int variable called term_two and set it to 0
    initialize an int variable called term_three and set it to 0
    initialize an int variable called term_four and set it to 0
    for i = 0; return_value - last_iteration > Epsilon; i++{
        num_terms += 1;
        if i != 0{
            power_term = power_term * 16;
        }
        term_one = 4 / ((8 * i) + 1);
```

```
        term_two = 2 / ((8 * i) + 4);
        term_three = 1 / ((8 * i) + 5);
        term_four = 1 / ((8 * i) + 6);
        return_value = return_value + (1 / power_term) * (term_one - term_two - term_three -
term_four)
    }
    return return_value;
}

bbp_terms(calculated_value){
    return num_terms;
}
```

File viete.c:

initialize an int variable called *num_fact* and set it to 0.

pi_viete(num){

    initialize double variable called *return_value* and set it to 1

    initialize double variable called *last_iteration* and set it to 0

    initialize double variable called *a* and set it to sqrt_newton(2)

```
    while return_value - last_iteration > Epsilon{
        num_fact += 1
        return_value  = return_value * a/2;
        a = sqrt_newton(2 + a);
        num_fact += 1
    }
    return return_value;
}


viete_factors(calculated_value){
```

```
    return num_terms;
}
```

File newton.c:

initialize an int called count and set it to 0;

```
sqrt_newton(num){
    This code is given in the assignment pdf document.
    count = 0;
    initialize a double variable called z and have it equal to 0.0
    initialize a double variable called return_value and have it equal to 1.0
    while absolute value of y-z > epsilon{
        count += 1;
        z = return_value
        return_value = 0.5 * (z + num/z);
    }
return return_value
}

sqrt_newton_iters(){
    return count;
}
```

File mathlib-test.c:

initialize a boolean variable called *e_test* and have it equal to false

initialize a boolean variable called *b_test* and have it equal to false

initialize a boolean variable called *m_test* and have it equal to false

initialize a boolean variable called *r_test* and have it equal to false

initialize a boolean variable called *v_test* and have it equal to false

initialize a boolean variable called *n_test* and have it equal to false initialize a boolean variable called *s_verbose* and have it equal to false

initialize a boolean variable called *h_help* and have it equal to false

initialize a boolean variable called *default_statement* and have it equal to true

```
main(takes in first character of each approximation method){
  while there are command lines {
      if user imputed a{
        set all test booleans to true
        set default_statement to false
        break
      }
       if user imputed e{
         set e_test boolean to true
         set default_statement to false
         break
      }
       if user imputed b{
         set b_test boolean to true
         set default_statement to false
         break
      }
       if user imputed m{
         set m_test boolean to true
         set default_statement to false
         break
      }
       if user imputed r{
         set r_test boolean to true
         set default_statement to false
         break
      }
```

```
        if user imputed v{
          set v_test boolean to true
          set default_statement to false
          break
        }
        if user imputed n{
          set n_test boolean to true
          set default_statement to false
          break
        }
        if user imputed s{
          set s_test boolean to true
          set default_statement to false
          break
        }
        if user imputed h{
          set h_test boolean to true
          break
        }
    }
    if h_help
        set all test statements to false
        set default_statement to true
}

if e_test{
    print test statement
    if s_verbose{
        print number of terms for corresponding test.
    }
}
```

```
if b_test{
    print test statement
    if s_verbose{
        print number of terms for corresponding test.
    }
}
if m_test{
    print test statement
    if s_verbose{
        print number of terms for corresponding test.
    }
}
if r_test{
    print test statement
    if s_verbose{
        print number of terms for corresponding test.
    }
}
if v_test{
    print test statement
    if s_verbose{
        print number of terms for corresponding test.
    }
}
if n_test{
    print test statement
    if s_verbose{
        print number of terms for corresponding test.
    }
}
if default_statement{
```

```
    print the default statement
}
```