Department of Computer Science

COS132 2015
Practical 5

Mr Madoda Nxumalo

0 Introduction

0.1 Uploads and Deadline

This practical is an in-lab practical, and as such students must write it at the Informatorium during the normal practical session hours. The uploads will become available at practical session hours as of **17 April - 23 April**.

Make sure that you have submitted your code to Fitchfork all your tasks at the end of the practical session. **No late submissions will be accepted**.

0.2 Plagiarism Policy

All submissions for this practical should (as usual) be documented as specified in our plagiarism policy. Refer to the notes about our plagiarism policy which can be found on the COS132 website on ClickUP.

1 Expressions

The natural exponential function is expressed as e^x, where e is Euler's number, a number (approximately 2.718281828) such that the function e^x is its own derivative. The notations $ln\ x$ and $log_e x$ both refer to the natural logarithm of x. e^x can be calculated using the following infinite series:

$$e^x = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \frac{x^4}{4!} + \ldots, x \in \mathbb{R}$$

$x \in \mathbb{R}$ means that x is any real number. Note that the first two terms in the series can respectively be expressed as $\frac{x^0}{0!}$ and $\frac{x^1}{1!}$.

The following Taylor series can be used to calculate $ln(1 + x)$ at around 0:

$$ln(1 + x) = x - \frac{x^2}{2} + \frac{x^3}{3} - \frac{x^4}{4} + \ldots, x \in (-1, 1]$$

$x \in (-1, 1]$ means, that x is a real value in the range (-1, 1], this range excludes -1 and includes 1.

Implement the functions defined in the following section.

2 Functions

`int` `factorial(`**`int`**`)` - the factorial of n, denoted by $n!$ is the product $1 \times 2 \times 3 \times \ldots \times n$. The function should return the factorial of its parameter. $0!$ is defined to be the product of the natural numbers in the series with zero terms, i.e. the product of no values. Since the unit of multiplication is 1, this value is 1. If the parameter is negative, the function should return -1.

`double` `sumOfSeries(`**`const double`** `v[],` **`size_t`** `t)`
return the sum of first t entries of the array passed as a parameter to the function.

`void` `eSeries(`**`double`** `x,` **`double`** `v[],` **`size_t`** `t)`
fills the array passed as parameter with the first t terms of the series to calculate e^x, where x is the value passed in the first parameter.

`void` `eSeries(`**`double`** `x,` **`double`** `v[][2],` **`size_t`** `t)`
fills the array passed as parameter with the numerators and denominators of the first t terms of the series to calculate e^x, where x is the value passed in the first parameter. `v[x][0]` should hold the numerator of the x^{th} term while `v[x][1]` should hold the denominator of this term.

`void` `lnSeries(`**`double`** `x,` **`double`** `v[],` **`size_t`** `t)`
fills the array passed as parameter with the first t terms of the series to calculate $ln(1 + x)$, where x is the value passed in the first parameter. It is the responsibility of the caller of the function to ensure that the function is not called with x values outside the specified range $(-1, 1]$.

void lnSeries(**double** x, **double** v[][2], **size_t** t)

fill the array passed as parameter with the numarators and denominators of the first t terms of the series to calculate $ln(1+x)$, where x is the value passed in the first parameter. `v[x][0]` should hold the numerator of the x^{th} term while `v[x][1]` should hold the denominator of this term. You may assume that the function is called only with valid x values.

void printSeries(**const double** v[], **size_t** t)

print the series of which the terms are the entries in the array passed as a parameter in the following format

$$v[0] + v[1] + v[2] + v[3] + \ldots + v[n].$$

void printSeries(**const double** v[][2], **size_t** t)

prints the first t terms of the series passed as a parameter to the function in the following general format:

$$\frac{v[0][0]}{v[0][1]} + \frac{v[1][0]}{v[1][1]} + \frac{v[2][0]}{v[2][1]} + \ldots + \frac{v[n][0]}{v[n][1]}$$

3 The program

You are given an incomplete program that can be used to compare the values calculated using these series with values calculated using the functions and constants in the **cmath** header.

Your program should make use of the above mentioned functions to produce output as shown in the following test runs:

3.1 Test run

```
Enter a value belonging to (-1, 1]:  -0.01
1.  Calculate e^-0.01
2.  Calculate ln(1 + -0.01)
Enter your choice:  1
Enter the number of terms:  4
e^-0.01 =
         1      -0.01     0.0001     -1e-06
    -------- + -------- + -------- + --------
         1          1          2          6

= 1 + -0.01 + 5e-05 + -1.66667e-07 = 0.990049833333
When using the cmath constant and pow function:
e^-0.010000000000 = 0.990049833749
```

3.2 Test run

```
Enter a value belonging to (-1, 1]:  0.61
1.  Calculate e^0.61
2.  Calculate ln(1 + 0.61)
Enter your choice:  2
Enter the number of terms:  5
ln(1 + 0.61) =
      0.61    -0.3721   0.226981  -0.138458  0.0844596
   -------- + -------- + -------- + -------- + --------
         1          2          3          4          5

= 0.61 + -0.18605 + 0.0756603 + -0.0346146 + 0.0168919 = 0.481887656853
When using the cmath log function:
ln(1 + 0.610000000000) = 0.476234178996
```

3.3 Test run

```
Enter a value belonging to (-1, 1]:  -0.01
1.  Calculate e^-0.01
2.  Calculate ln(1 + -0.01)
Enter your choice:  1
Enter the number of terms:  4
e^-0.01 =
          1      -0.01     0.0001     -1e-06
    -------- + -------- + -------- + --------
          1          1          2          6

= 1 + -0.01 + 5e-05 + -1.66667e-07 = 0.990049833333
When using the cmath constant and pow function:
e^-0.010000000000 = 0.990049833749
```

3.4 Test run

```
Enter a value belonging to (-1, 1]:  0.413
1.  Calculate e^0.413
2.  Calculate ln(1 + 0.413)
Enter your choice:  1
Enter the number of terms:  20
e^0.413 = 1 + 0.413 + 0.0852845 + 0.0117408 + 0.00121224 + 0.000100131
+ 6.89236e-06 + 4.06649e-07 + 2.09933e-08 + 9.63357e-10 + 3.97867e-11
+ 1.49381e-12 + 5.14119e-14 + 5.26419e-15 + 3.28434e-15 + 8.65537e-16
+ 3.57489e-16 + -1.02558e-15 + -1.36024e-16 + 4.60336e-16 = 1.511345025934
When using the cmath constant and pow function:
e^0.413000000000 = 1.511345025934
```

4 Comments

- This given test suite does not have full coverage. You are encouraged to test your program with more test cases.
- **size_t** is an alias of one of the fundamental unsigned integer types. It is defined in `<cstddef>`. It is a type that is guaranteed to be able to represent any array index, or the size of any object in bytes. **size_t** is the type returned by the `sizeof` operator and is widely used in the standard library to represent sizes and counts. You should *prefer* using **size_t** instead of **int** or **unsigned** for variables that hold the size of an array, and for loop counters that represent array indexes. For example:

  ```
  for (size_t i = 0; i < len; i++)
      array[i] = ...;
  ```

- The `eSeries, lnSeries` and `printSeries` functions are overloaded to enable the programmer to call these functions with both one-dimensional and two-dimensional arrays in the parameter list.
- The use of `void printSeries(const double v[][2], size_t t)` is suppressed in the given driver program when the number of terms is high.
- The `log` function in the **cmath** returns the natural logarithm, denoted by ln.
- You need to consult C++ documentation to find the name of Euler's constant in the **cmath** header.
- You should not use libraries other than those included in the given h-file.
- The accuracy of the values calculated using the series increases when more terms are used.

5 Submission

You have to upload the following files;

- **main.cpp** - You are given an incomplete version of this program. You have to complete the program by replacing the comments with code.
- **series.h** - This file is given. It contains the prototypes of the functions. You should not change this file.
- **series.cpp** - You have to create this file. It should contain the implementations of the functions.
- **makefile** - you are provided with a makefile containing all the needed commands.

The tarball containing the above mentioned files can be created by the following command.

```
tar -cvz series.h series.cpp main.cpp makefile -f series.tar.gz
```

6 Marks Allocation

Task	**Percentage**	**Maximum mark**
eSeries	?%	16
lnSeries	?%	20
PrintSeries1	?%	10
main	?%	10
Template	?%	8
Total	**100**	**?**

7 Challenge

Can you explain the negative values shown in the 18^{th} and 19^{th} terms in the test run in Section 3.4?