Delivery 2 (Problem 4 and 6)

Software Engineering Process - SOEN 6011

Submitted By :
Aravind Ashoka Reddy
https://github.com/ReddyAravindAru/SOEN6011$_{C}alculator_{P}roject$
40103248

Contents

1 Changes from D1 to D2

Problem 2

R-3 : when the input for the program is a non-positive integer, the program prints the result as Infinity.

R-4 : When the input (x) is a positive real number, the function validates and returns a gamma value which is positive.

R-5 : when the input for the program is a non-positive real number except negative integers, the gamma value will be be a negative or positive value.

Problem 3

Algorithm 1(Algorithm, which uses mathematical function implementations and formulas)

Description: This algorithm is pretty easy to implement because it involves basic mathematical function implementations and calculations. Disadvantage is as it involves lot of constant numerical values, approximation is not precise.

Algorithm 1 Gamma Function

```
procedure LOGGAMMA(x)
    1. $PI \leftarrow 3.1415926535$
    2. $Tmp \leftarrow (x–0.5) * logarithm(x + 4.5)–(x + 4.5)$

    3. $Str \leftarrow 1.0 + 76.18009173/(x + 0) - 86.50532033/(x + 1) + 24.01409822/(x + 2) - 1.231739516/(x + 3) + 0.00120858003/(x + 4) - 0.00000536382/(x + 5)$

    4. return $Tmp + logarithm(Str * squarerootCalc(2 * PI))$

procedure LOGARITHM(lgx)

    while $iteration > 0$ do
        if $iteration\ mod\ 2\ not\ 0$ then
            $result \leftarrow result + powerfunc(mix, iteration)/iteration.$
        iteration - -
        return $result * 2$


        procedure EXPONENTIAL(n, x)
            $expVal \leftarrow$1.0
  for s$i = n - 1; i > 0; - - i$
            $expVal \leftarrow$1+x*expVal/i
                return $expVal$

            procedure POWER(x, y)
                if $y == 0$ then
                    return 1
else if $y\ mod\ 2 == 0$ then
                    return $power(x, y/2) * power(x, y/2)$
  else
                    return $x * power(x, y/2) * power(x, y/2)$
```

Algorithm-1 continuation..

```
procedure NEGATIVEGAMMAFUNC(x)
    $n$ ←-1.0 * x
negGamma←-PI/(n * gamma(n) * sine(PI * n))
        return $x * power(x, y/2) * power(x, y/2)$

    procedure SQUAREROOTCALC(x)
        if  Val == 0 then return 0

        last ← 0.0
        res ← 1.0
        loop:
        if res == last then
            last ← res.
            res ← (res+2 ÷ res) ÷ 2.
            goto loop.
            close;

        return res
```

Algorithm 2(Gamma value with sterling's approximation)

Description: Sterling's approximation has lot of constants to give the precise value for the gamma function. It uses bernouli's number ans integration concept to obtain the precision value.

Advantage: It is more broadly defined for all complex numbers other than non-positive integers.

disadvantage: This approximation is mainly for approximation for factorials.

Algorithm 2 Gamma algorithm with stirling's approximation

```
    procedure SQRT(x)
        if  Val == 0 then return 0
        last ← 0.0
        res ← 1.0
        loop:
        if res == last then
            last ← res.
            res ← (res+2 ÷ res) ÷ 2.
            goto loop.
            close;
        return res

    procedure POWER(x, y)
        if y == 0 then
        return 1
else if y mod 2 == 0 then
        return $power(x, y/2) * power(x, y/2)$
     else
        return $x * power(x, y/2) * power(x, y/2)$


procedure GAMMA(x)
    return $sqrt(2 * PI/x) * power((x/E), x)$
=0
```

2 Eclipse Debugger

An in-built Java debugger gives all standard debugging functionality, including the capacity to perform step execution, to set breakpoints and watchpoints, to inspect variable values, and to suspend and resume threads. Furthermore, you can debug applications that are running on a remote machine.

The Eclipse debugger itself exists as a standard module included inside the Eclipse binaries. Eclipse additionally has an uncommon Debug view that enables you to deal with the debugging or running of a program in the Workbench. It shows the runtime stacks for the suspended threads for each objective you are debugging. Each thread in your program shows up as a hub in the tree, and the Debug view shows the procedure for each target you are running. In the event that the thread is suspended, its stack casings are appeared as child components.

Advantages

1. Debug view – Visualizes call stack and provides operations on that.

2. Breakpoints view – Shows all the breakpoints.

3. Variables/Expression view – Shows the declared variables and their values.

4. Display view – Allows to Inspect the value of a variable, expression or selected text during debugging.

5. Remote Debugging

6. It has Watchpoints, Exception Breakpoints, Conditional Breakpoints

Limitations

1. exception breakpoints cannot be placed

2. values cannot be changed during debugging.

3. objects cannot be placed on the heap memory.

4. Setting a breakpoint in a compiled method results in an error.

3 Eclipse-cs (Checkstyle plug-in 8.18.0)

The Checkstyle Plugin (eclipse-cs) integrates the outstanding source code analyzer Checkstyle into the Eclipse IDE. Checkstyle is an improvement apparatus to enable you to guarantee that your Java code adheres to a lot of coding norms. Checkstyle does this by reviewing your Java source code and calling attention to things that veer off from a characterized set of coding rules. With the Checkstyle Plugin your code is continually investigated for issues.

Inside the Eclipse workbench you are informed of issues by means of the Eclipse Problems View and source code explanations similarly as you would see with compiler errors or warnings. An agreeable Checkstyle setup editor causes you make and keep up your review rule configurations. Utilizing filters and file sets you can characterize which files get checked and which don't.

Advantages

1. Eclipse-cs can check numerous parts of your source code. It can discover class design issues, method design issues.

2.It additionally can check code layout and formatting issues.

3. The programming style received by a software development project can guarantee that the code compiles with great programming rehearses which improves the quality, lucidness, re-ease of use of the code and may diminish the expense of development.

Limitations

1. The checks performed by Eclipse-cs (Eclipse checkstyle) are for the most part constrained to the presentation of the code.

2. Java code ought to be composed with ASCII characters only, no UTF-8 support.

3. You can't check if there are redundant type casts or unused public methods.

4 Efforts made

Correctness – This implemented program involves implementation of many mathematical functions so the aim and effort was to get more accurate values from all functions in order to get almost precise value for gamma function, which is achieved in my program.

Efficient – I can say my program is efficient because there are no many loops and I have used many constants so results are calculated with basic mathematics at most places, So I conclude saying my program takes lesser time to execute and all the stack overflow, underflow exceptions are taken care.

Maintainable – In the future individual functions in the program can be improved for more precision values with ease as all independent functions are separately coded which is easy to read and understand the flow of the program. Any new programmer can easily trace the program functionality and maintain it if needed.

Robustness - To achieve this I have meticulously coded to handle especially for all the invalid, unexpected inputs. All the errors are clearly stated to the user. All kinds of exceptions are taken care for all kinds of inputs like integers, real numbers, non-positive numbers, zeros and other invalid numbers.

Usable – As my program has implementations of several mathematical functions, any future program wants to implement any of my independent functions they can inherit my functions or reuse the function so that they can save time and resources. It can used across many platforms as long as inputs are valid.

5 Challenges and practices

1. What's special about my implementation or tests?

1. My program involves implementations of many mathematical functions and all are separately coded as individual functions, which makes **tracing and reading easy**.

2. It is portable as it can be **run on many different platforms** (Operating systems)

3. It is very efficient as it occupies **less memory and processing time** is too low. Even in the functions stack overflow and underflow conditions are taken care.

4. My implementation is very **flexible** because, in order to improve the results of particular subtask or a function, there is no need to rewrite it instead change few lines of code to get the desired results.

5. My program satisfies **generality** feature as it can be used by other programs which needs to implement the similar tasks which are existed in my program So it satisfies inheritance property as well. Due to this lot of time and resources are not wasted.

6. **Documentation:** A Javadoc is generated for my program, which is very useful to understand the prime functionality of each subtask and it also describes the parameters and return values.

7. My program is **structured**, as in whole task is divided into number of subtasks and developed independently So it becomes more readable, and the testing and documentation process gets easier.

8. For testing, I used **Junit** which is very essential when developing many functions which are dependent on each other because, to do unit test on each function separately by which we can know if all functions return expected results then whole program is correct.

2. What practices did I follow to make them special?

I followed **agile methodology**, through which I developed my subtasks one by one periodically. This helped me to concentrate on only one function (subtask) at one particular time so it made development easier and error free.

I documented along with the subtasks implementation which helped me to understand the flow and made integration stress free.

I used to implement unit test cases as soon as the subtask is done with the implementation. Unit test cases are run for all the subsequent subtask implementations to make sure dependency on functions is working fine.

3. How can I prove any of it to the others?

Using any of the source code review tool, debugger, test framework to validate the above mentioned features in my source code. I can give my program to end user to use it so that I can show program does not break for any kind of invalid or wrong inputs and also it prompts with appropriate error messages. After reviewing and testing , the reports can be used to prove my code versatility.

Bibliography

[1] https://github.com/checkstyle/eclipse-cs

[2] https://www.aicas.com/jamaica/3.2/doc/html/debugging.html

[3] https://marketplace.eclipse.org/content/checkstyle-plug

[4] https://en.wikipedia.org/wiki/Checkstyle