CONCORDIA UNIVERSITY

SOEN 6011 - SOFTWARE ENGINEERING PROCESS
SUMMER 2022

ETERNITY
FUNCTION (F5) : ab^x

Harshkumar Nileshkumar Patel
Student ID : 40165709

Github Link :- https://github.com/harshp189/SOEN_6011_F5

Contents

1 Problem 1 : Introduction

1.1 Description

F5 : ab^x is an exponential function, where a is a constant value, $a \neq 0$ and it represents starting (initial) value , b is called base and is a positive real number, $b > 1$, x is called the exponent (power). In this function, b is a constant and x is independent variable [1].

1.2 Domain

The domain for exponential function is the set of real numbers.
$x \in R$, $-\infty < x < \infty$, Domain : $\{x \mid x \in R\}$

1.3 Co-Domain

Co-Domain is the set of all possible function output values.
For the given function , suppose $y = f(x) = ab^x$, then $-\infty < y < \infty$, so the range will be $(-\infty, \infty)$. So the range of the exponential function is R, $y \in R$, where $x \in R$, $b > 0$.

1.4 Characteristics

- In exponential function, if $b > 0$, then it is known as exponential growth function (increasing function). Its graphical representation shown in the left part of the figure 1.

- In exponential function, if $0 < b < 1$, then it is known as exponential decay function (decreasing function). Its graphical representation shown in the right part of the figure 1.

- Exponential function have horizontal asymptote (i.e function approaches to a imaginary horizontal line but never crosses) at $Y = 0$ (i.e $X - $ axis).

- They are continuous function.

- There is no symmetry in exponential function, so they are neither odd nor even function.

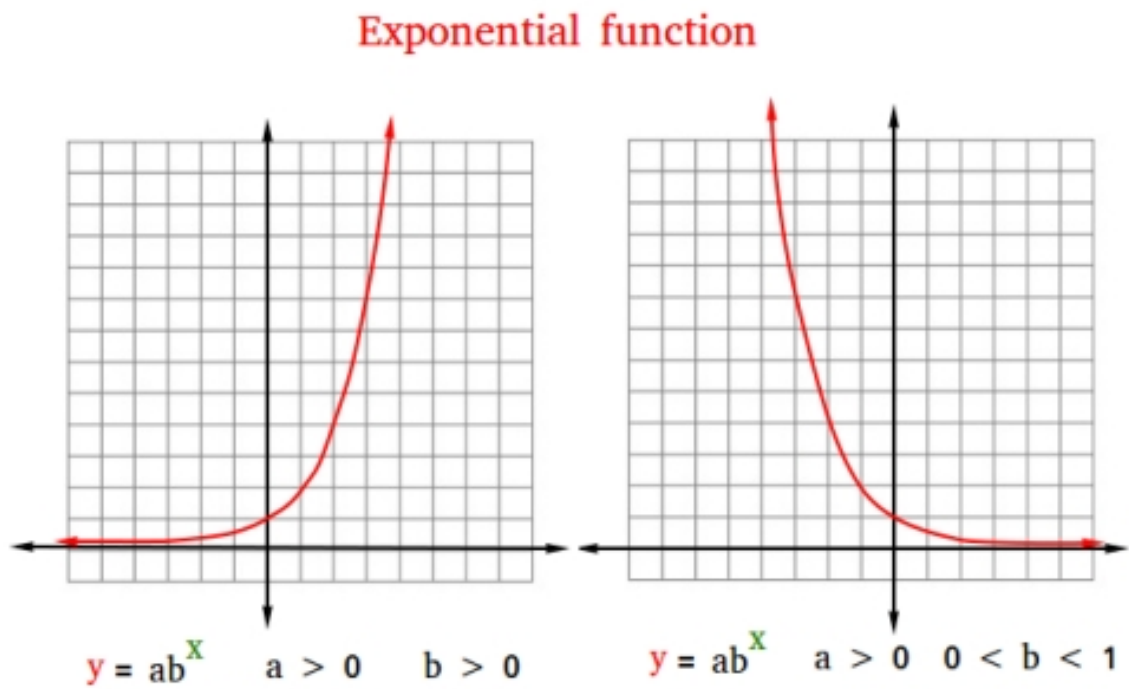- Exponential function is not injective but is surjective [2].

Figure 1: Exponential Function Graph (Growth And Decay) [3].
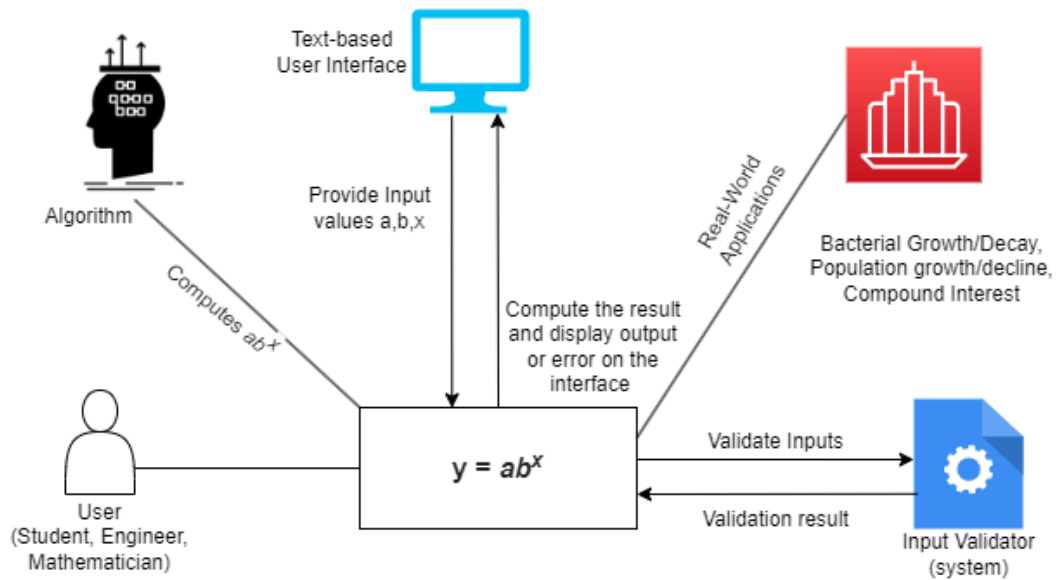
1.5 Context of Use Model

Figure 2: Context of Use model diagram in 'Boxes-and-Lines'

2   Problem 2 : Requirements

2.1   Assumptions

1. Assumption 1

   - **ID** : A1
   - **Description** : The user should provide input values for all "a", "b" *and* "x". As no default values are assumed for the calculation of the function.

2. Assumption 2

   - **ID** : A2
   - **Description** : Inputs for the function should be real number and the output of the function is also a real number.

3. Assumption 3

   - **ID** : A3
   - **Description** : For large input values which are out of the range, the results are undefined.

2.2   Requirements

The Functional and Non-Functional Requirements are expressed using guidelines from the ISO/IEC/IEEE 29148:2018 standard [4].

1. Requirement 1

   - **ID** : REQ1
   - **Version** : 1.0
   - **Type** : Functional Requirement
   - **Priority** : High
   - **Difficulty** : Easy
   - **Description** : In the exponential function ab^x, the input value a should not be 0 i.e., $a \neq 0$ (or else it will result in output of the function to be 0 for every input), also input value of base b must be greater than 1 i.e., $b > 1$ (or else if $b = 1$, it will result in output of the function to be 'a' for every input of x).

2. Requirement 2

   - **ID** : REQ2
   - **Version** : 1.0
   - **Type** : Functional Requirement
   - **Priority** : High

- **Difficulty** : Easy
- **Description** : In the exponential function ab^x, the input value of base b must not be negative as it will result in complex numbers so $b > 0$. Also, the input value of x must be any real number. i.e $x \in R$.

3. Requirement 3

- **ID** : REQ3
- **Version** : 1.0
- **Type** : Functional Requirement
- **Priority** : High
- **Difficulty** : Medium
- **Description** : The system must take input values of a, b and x from the users and return the output of ab^x function. For example, if $a = 2, b = 3, x = 2$, the output should be 18.

4. Requirement 4

- **ID** : REQ4
- **Version** : 1.0
- **Type** : Functional Requirement
- **Priority** : High
- **Difficulty** : Medium
- **Description** : The system must be able to compute the value of the function ab^x when the input value of exponent x is fractional or negative and must handle all the cases.

5. Requirement 5

- **ID** : REQ5
- **Version** : 1.0
- **Type** : Functional Requirement
- **Priority** : High
- **Difficulty** : Easy
- **Description** : If the input values are not of numeric type (real number), the system must not accept it and display the error and ask for numeric values as input.

6. Requirement 6

- **ID** : REQ6
- **Version** : 1.0
- **Type** : Functional Requirement

- **Priority** : High
- **Difficulty** : Easy
- **Description** : If any of the input values a, b or x are not provided by the user, the system should not accept that input and ask user to provide the missing values.

7. Requirement 7

- **ID** : REQ7
- **Version** : 1.0
- **Type** : Non-Functional Requirement
- **Priority** : High
- **Difficulty** : Medium
- **Description** : The computation of the function must be performed within some desired time to provide better performance.

8. Requirement 8

- **ID** : REQ8
- **Version** : 1.0
- **Type** : Non-Functional Requirement
- **Priority** : High
- **Difficulty** : Medium
- **Description** : The text-based user interface must be user-friendly and appropriate error messages should be displayed for the corresponding error by the system.

9. Requirement 9

- **ID** : REQ9
- **Version** : 1.0
- **Type** : Non-Functional Requirement
- **Priority** : High
- **Difficulty** : Medium
- **Description** : The system must be maintainable, i.e, it should be easy to modify the existing system or add any new features/functionality over time.

3 Problem 3 : Algorithms

Two algorithms (Algorithm 1 and Algorithm 2) have been selected for implementing the function $y = ab^x$. The pseudocode for each of the algorithm is provided in this section followed by their detailed description and the advantages as well as the disadvantages of each approach are discussed later.

3.1 Pseudocode format
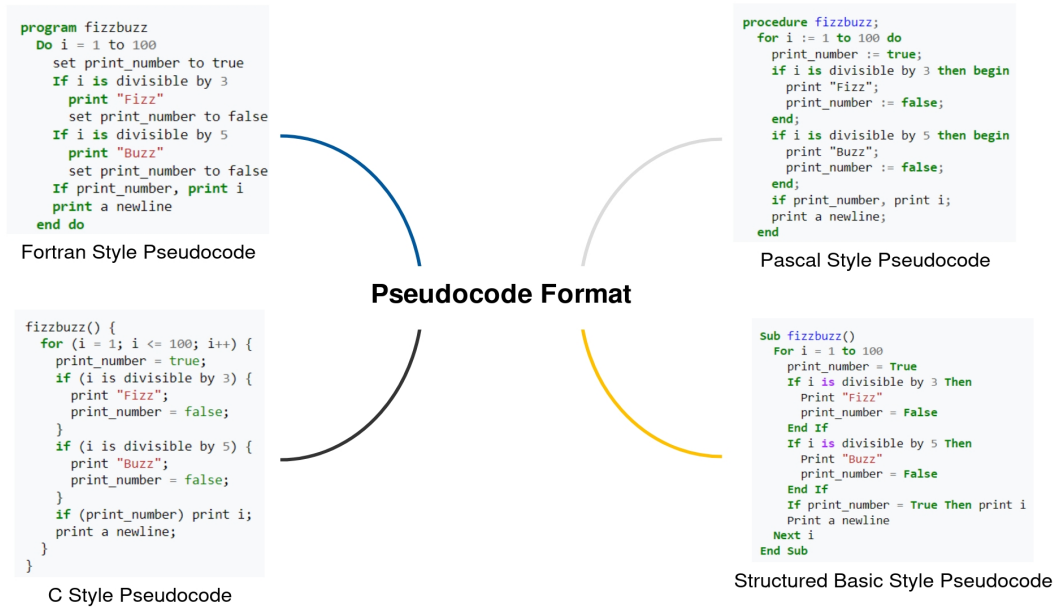
Figure 3: Mindmap of different pseudocode formats

As shown in the figure, there are four types of pseudocode formats, Fortran Style, Pascal Style, C Style, Structured Basic Style for describing the pseudocode for the algorithms [5], I have selected the Structured Basic Style Pseudocode format as LaTeX provides support for this kind of format for typesetting algorithms by the help of several packages, I have used the algpseudocode package and algorithm package.

3.2 Algorithms and Pseudocode

Algorithm 1 Iterative-Exponent(a,b,x)

    Input : double number a, b, x
    Output : value of the function ab^x represented by double number result
  1: $temp \leftarrow 1$
  2: **while** $x > 0$ **do**
  3:    $temp \leftarrow temp \times b$
  4:    $x \leftarrow x - 1$
  5: **end while**
  6: $result \leftarrow a \times temp$
  7: return $result$

In this algorithm, the iterative approach is used for calculation of the given function, ab^x, Initially a temporary variable called $temp$ is initialized to 1 and to calculate b^x, While loop is used for all the values of x (exponent) and in each iteration, $temp$ is multiplied by b (base) and the value of x is decremented by 1. The loop continue until the exponent value x reaches to 0. At last, a is multiplied with the $temp$ value $(a * b^x)$ and the output is obtained in the result and it is returned.

Algorithm 2 Recursive-Exponent(a,b,x)

    Input : double number a, b, x
    Output : value of the function ab^x represented by double number result
  1: $temp \leftarrow recursive\_power(b, x)$
  2: $result \leftarrow a \times temp$
  3: return $result$

Algorithm 2.1 recursive_power(b,x)

    Input : double number b, x.

    Output : value of b^x will be returned.

1: **if** $x \neq (Integer)x$ **then**
2:     return $fraction\_power(b, x)$
3: **end if**
4: $helper \leftarrow recursive\_power(b, x/2)$
5: **if** $x < 0$ **then**
6:     $b \leftarrow 1.0/b$
7:     $x \leftarrow -x$
8:     **if** $x \bmod 2 = 0$ **then**
9:       return $helper \times helper$
10:     **else**
11:       $x \leftarrow x - 1$
12:       return $helper \times helper \times b$
13:     **end if**
14: **else if** $x = 0$ **then**
15:     return 1.0
16: **else if** $x = 1$ **then**
17:     return b
18: **else if** $x \bmod 2 = 0$ **then**
19:     return $helper \times helper$
20: **else**
21:     $x \leftarrow x - 1$
22:     return $helper \times helper \times b$
23: **end if**

In this algorithm, the recursive approach using optimized divide and conquer strategy is used for the calculation of the given function ab^x [6]. $recursive\_power(b, x)$ algorithm is used for calculating the value of b^x, the recursive call to the function with value base b and $x = x/2$ i.e $recursive\_power(b, x/2)$ is stored in helper, so that it needs to be computed just once and can be used later on. If the value of exponent x is fractional, it is checked and $fraction\_power(b, x)$ will compute all the power values of such exponents. For negative value of exponent, value of base b is reciprocated and that of exponent x is negated and on the basis of the parity, the algorithm performs the computation. In the base case, if the value of x is 0, value 1 is returned, otherwise, if $x = 1$, value b is returned, if the exponent is divisible by 2, the algorithm recurses on ($helper * helper$) and for odd value of exponent, the value of exponent is decremented by 1 and then it recurses on ($helper * helper * b$), At last, a is multiplied with the returned value from $recursive\_power(b, x)$ and the output is obtained in the $result$, and it is returned.

Algorithm 2.2 fractional_power(b,x)
___
 Input : double number b, x.
 Output : value of b^x will be returned.

1: $e \leftarrow 2.7182818284590$
2: $exponentValue \leftarrow 0$
3: $iterations \leftarrow 20$
4: $logValue \leftarrow logarithm\_calculator(b)$
5: $eValueX \leftarrow (x * logValue)$
6: $eValueR \leftarrow eValueX - (int)\, eValueX$
7: **while** $iterations >= 0$ **do**
8:   $powerValue \leftarrow recursive\_power(eValueR, iterations)$
9:   $factorialValue \leftarrow factorial\_calculator(iterations)$
10:   $exponentValue \leftarrow exponentValue + (powerValue/factorialValue)$
11:   $iterations \leftarrow iterations - 1$
12: **end while**
13: $eValueN \leftarrow recursive\_power(e, (int)\, eValueX)$
14: $eFinalValue \leftarrow exponentValue * eValueN$
15: **return** $eFinalValue$
___

Algorithm 2.3 logarithm_calculator(n)
___
 Input : double number n.
 Output : natural logarithmic value of n $ln(n)$ will be returned.

1: $iterations \leftarrow 100$
2: $logValue \leftarrow 0$
3: **if** $n < 0$ **then**
4:   $n \leftarrow -n$
5: **end if**
6: $baseValue \leftarrow (n-1)/(n+1)$
7: **while** $iterations > 0$ **do**
8:   **if** $iterations$ mod $2 \neq 0$ **then**
9:    $logValue \leftarrow logValue + recursive\_power(baseValue, iterations)/iterations$
10:   **end if**
11:   $iterations \leftarrow iterations - 1$
12: **end while**
13: **return** $logValue * 2$
___

Algorithm 2.4 factorial_calculator(n)

   Input : double number n.

   Output : factorial value of n will be returned.

1: **if** $n = 0$ **then**

2:    return 1

3: **end if**

4: return $(n * factorial\_calculator(n-1))$

Algorithm $fractional\_power(b, x)$ helps in finding the power of any fractional exponent, the given exponent and base are converted as, $b^x = (e^{ln(b)})^x = e^{xln(b)}$ [7], where the natural logarithm value for b is computed using the $logarithm\_calculator(n)$ algorithm. Finally, for the computation of $e^{xln(b)}$, ($eValueX$) it is splitted into e^r ($eValueR$) and e^n ($eValueN$),

   $e^{xln(b)} = e^r * e^n$

where e^r is computed in the while loop by calculating numerator(powerValue) by using $recursive\_power$, passing $eValueR$ as base and *iteration number* as exponent and the denominator is the factorial value of the iteration number obtained using the $factorial\_calulcator(n)$ algorithm, numerator and denominator are divided and their result is stored in the $exponentValue$ which is returned after completion of all the given number of iterations. Similarly e^n ($eValueN$) is computed by using $recursive\_power$, passing the value of e as base and $(int)eValueX$ as the exponent, at last both $exponentValue$ and $eValueN$ are multiplied and their product is returned as the final value.

Algorithms 2.1, 2.2, 2.3 and 2.4 are defined to help in the computation of the main Algorithm 2.

3.3 Advantages and Disadvantages

3.3.1 Algorithm 1

Iterative Approach

   Advantages:

1. Iterative algorithms are simple and easy to develop; they can be easily understood by the reader.

2. In Iterative approach there is no overhead of function calls and they do not use stack memory and hence don't suffer from stack overflow.

   Disadvantages:

1. Some iterative algorithms are slower when compared to other approaches, Algorithm 1 which was described previously has time complexity of $O(n)$, so for larger inputs it is inefficient.

2. If the termination condition of control variable is not defined properly, it may lead to infinite loop.

3.3.2 Algorithm 2

Recursive Approach
    Advantages:

1. Recursive Algorithms have less time complexity for certain problems; Algorithm 2 which was described previously has time complexity of $O(logn)$ which is better than that of Algorithm 1.

2. Recursive Algorithms are very useful in situations where particular solution is to be applied repeatedly [8].

 Disadvantages:

1. Recursion is more difficult to understand in some algorithms and tracing them is difficult

2. Recursive algorithms utilize too much memory and when the base case is not defined properly it may lead to infinite loop or crashing of the CPU.

Decision

Algorithm 2 follows optimized divide and conquer approach and provides $O(logn)$ time complexity as well as space complexity, so the recursive algorithm is faster in comparison to Algorithm 1 which provides time complexity of $O(n)$. Recursive approach is better when the problem can be divided into smaller problems which are repeated and the function ab^x, can be partitioned into sub problems so it is better to choose the recursive algorithm. Using recursive strategy in the optimized divide and conquer approach leads to reduction of problem size at each step and uses less time for the computation than the iterative approach. Also, all the cases like negative exponent, fractional exponent are efficiently handled by the recursive algorithm. As a result, I have opted to use Algorithm 2(recursive algorithm) for the implementation of the function ab^x as it is a better option as compared to Algorithm 1(iterative algorithm).

4 Problem 4

4.1 Debugger

4.1.1 Description

Debugging provides the facility for identifying and eliminating bugs, errors present in the program. This process can help in tracing the reasons behind the improper functional behaviour of any program. For debugging, I have used the IntelliJ debugger tool for java language and it provides support for debugging code in different languages as well by installation of various plugins. IntelliJ debugger allows the facility to run the program step by step and keep a track of the values of the associated variables [9].

4.1.2 Advantages :

1. Breakpoints are special kind of markers which when encountered, the debugger temporarily halts the program execution (suspended program state) and it makes easier to analyse and debug those special lines.

2. It allows to trace the values of the variable throughout the program, such trace would be helpful to identify the points where the incorrect values are assigned to the variables, also it provides information about the status of the threads.

3. It enables execution of any arbitrary piece of code in the middle of the program's execution as well as throw exceptions to test the functional behaviour of the program under different circumstances. Also allows running multiple debugging sessions at the same time.

4. IntelliJ debugger provides a feature called stepping tool which provides control over the step-by-step execution of the program and helps in deducing the source of the error.

5. IntelliJ debugger supports the feature to modify and adjust the parts of code without requiring to termination of the ongoing debugging process, it eliminates the need to modify the values of the variable and run the debugger again from start.

4.1.3 Disadvantages :

1. With the help of debugger, the location of the problem can be identified but the logical error associated with it cannot be known.

2. Debugger does not provide insights on design issues as well as structure of the code and there is a learning curve associated with it.

3. Debugger faces several problems while dealing with multithreading programs.
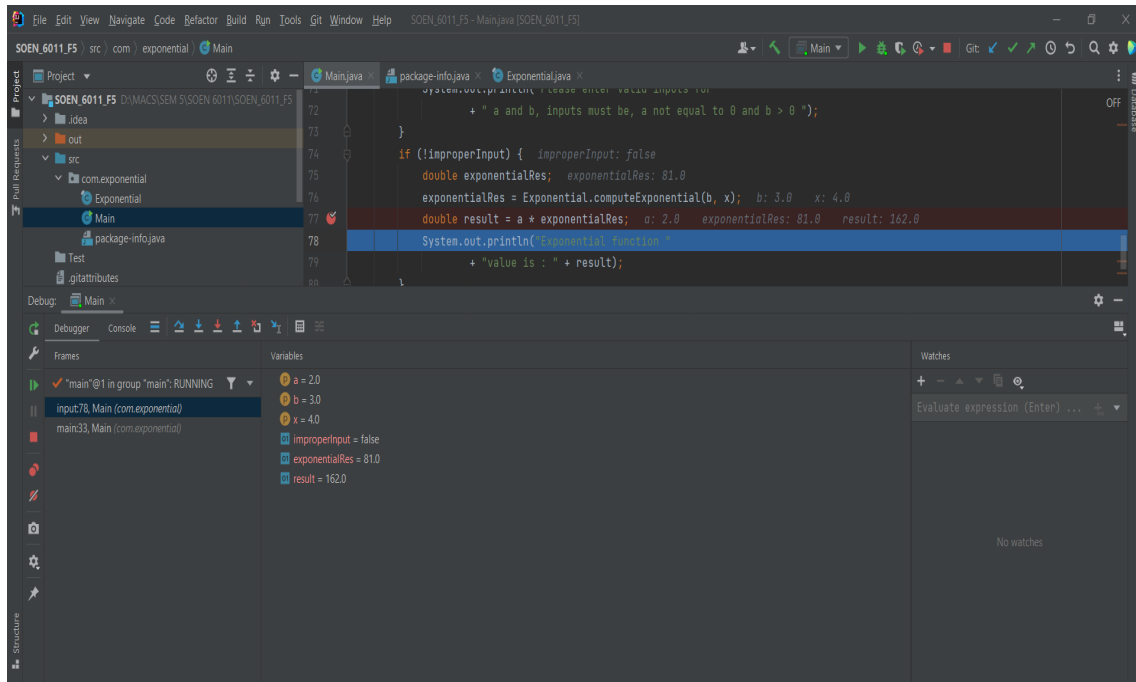
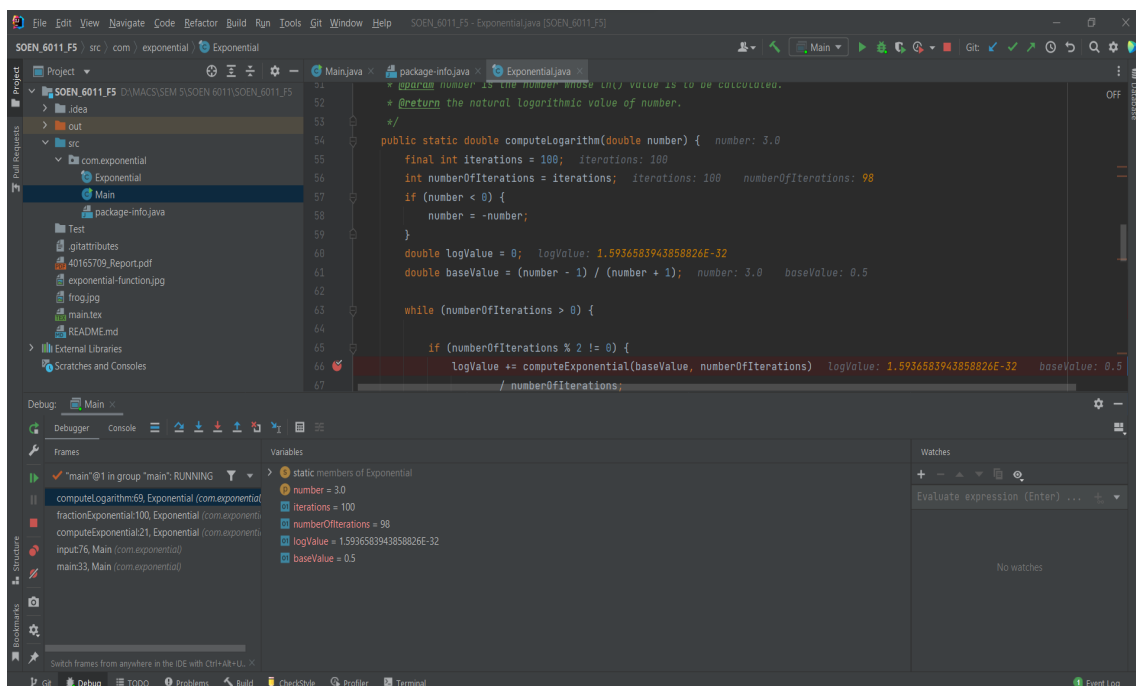Figure 4: Debugging computation of exponential function

Figure 5: Debugging computation of fractional exponent value

4.2 Quality Attributes

This section highlights and describes several efforts made to achieve the desired quality attributes related to the java program [10].

1. **Efficient** – The program uses recursive algorithm to compute the exponential value and it provides time complexity of $O(logn)$ and it is the same for space complexity due to recursive stack space. It is more efficient and better choice in comparison to other approaches. As a result, the program runs quickly, and the function value is computed within required time window.

2. **Maintainable** – The program is partitioned into various methods each responsible for certain kind of computation, I have divided the user interface, error handling part ($Main.java$) from the core logic part ($Exponential.java$) related to the calculation of exponential function value. It allows easy modification of features/functionalities as well as adding new ones. It makes the code maintainable.

3. **Correctness** – I have designed several test cases to check the program against all the possible values of the base and exponent including invalid inputs, fractional exponent, negative exponents, every test case runs successfully and hence the developed program provides correct results and agrees to the desired specifications.

4. **Robust** – Input values are validated as they are entered by the user and all the invalid inputs are rejected. On entering such inputs, the program displays corresponding error message and asks for new valid input values from the user. In this way error handling is performed to avoid any kind of failure.

5. **Usable** – The program provides text-based user interface to the user for entering the values and displays the computed result. The error messages are meaningful and can be easily understood. It provides functionality for user to continue the computation for other values or exit the system. In this way, the program is made user-friendly.

6. **Portable** - The program is developed in java language, so it can be executed on other platforms as well as systems without requiring any modifications.

4.3 Quality review of source code

To check the pragmatic quality of the code, I have used the Checkstyle tool and installed the Checkstyle plug-in in the IntelliJ IDE. It is a development tool used to ensure that proper coding standards are adhered by the java code. It automates the process of checking the java code and removes the need to manually perform that task. By default, it provides support for Sun code conventions and Google Java style standards. I have checked my java code against the Google Java programming style coding standards [11] .

4.3.1 Advantages :

1. It is highly configurable and provides support to multiple IDEs (IntelliJ, Eclipse) via the plug-in.

2. Checkstyle can also perform checking for various issues related to formatting and layout of the code.

3. It allows creation and checking against various user-defined rules according to the project requirements.

4. Checkstyle displays a panel with all the error/warnings messages related to the violation of the desired coding standards, it also points out the location and provides suggestion to solve it.

5. Checkstyle is designed as a standalone framework and it is easier to integrate it with other external tools.

4.3.2 Disadvantages :

1. Checkstyle provides functionality to locate the errors but does not support their automatic correction.

2. For any large-scale application with large number of lines of code, it becomes difficult for the programmer to analyse and resolve every errors/warning found by Checkstyle.

3. Checkstyle only checks the code against the coding standards but does not validate the correctness of the java code.

Before using the Checkstyle tool there were around 45+ errors related to coding standards and all of them were solved. The snapshots of error messages in the source code before using Checkstyle tool and after modifying the code according to desired coding standards are attached below.
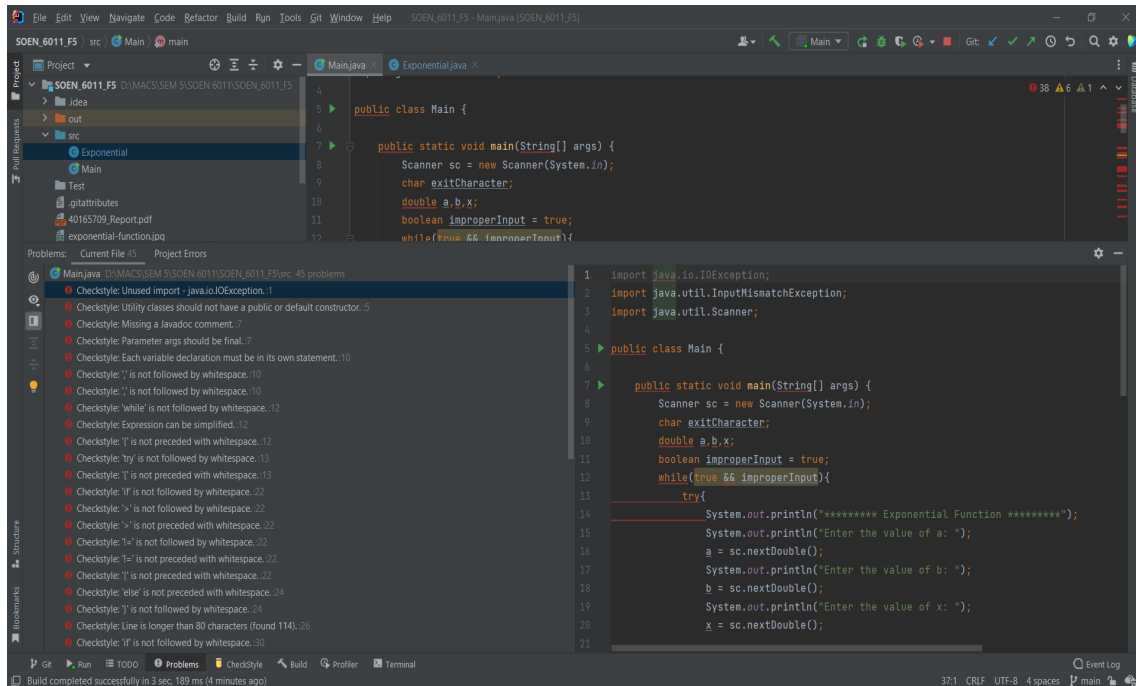
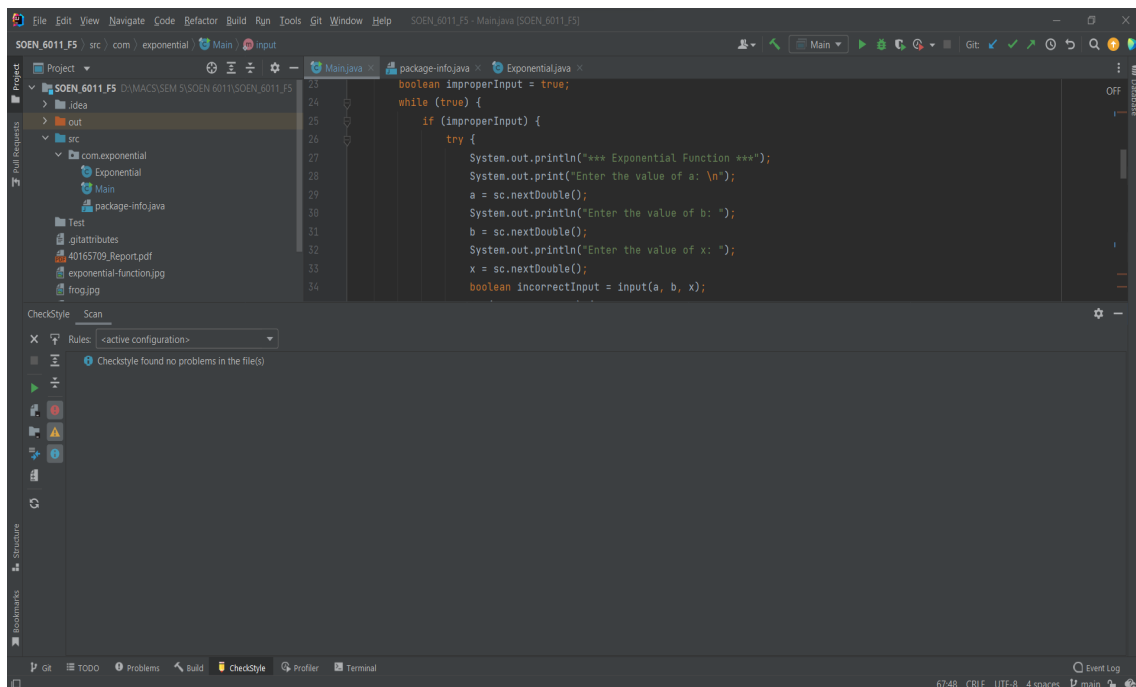Figure 6: Error messages in the source code detected by Checkstyle tool

Figure 7: Modified source code and resolved errors generated by Checkstyle tool

17

5   Problem 5 : Unit Test Cases

In order to write unit tests for the function ab^x , I have used **Junit 4** [12] as the unit testing framework. I have designed various unit tests to test the successful functioning of the system as well as all the methods used in the program for certain computations, I have provided the tracing of the requirements with the designed unit tests along with the description as below.

Requirement :   REQ1
Unit Test :   testInvalidInputValueA()
Description : Tests' if the value of a and b are as required or else displays error message.

Requirement :   REQ2
Unit Test :   testInvalidInputValueB()
Description : Tests' if the value of b is non-negative or else displays error message.

Requirement :   REQ3
Unit Test :   testFunctionForValidInputs(), testFunctionForOddExponent(),
testFunctionForEvenExponent(), testFunctionForZeroValue()
Description : All the unit tests, verifies the actual output and the result computed by the program for all valid inputs, with even and odd exponent as well as for zero value of exponent.

Requirement :   REQ4
Unit Test :   testFunctionForNegativeEvenExponent(),
testFunctionForNegativeOddExponent(), testFunctionForFractionalExponent(),
testFunctionForNegativeFractionalExponent()
Description : All the unit tests, verifies the actual output and the result computed by the program for negative exponent as well as fractional exponent values.

Requirement :   REQ5
Unit Test :   testInvalidInputValue()
Description :   Test if the input values are of the double type or else displays error message.

Requirement :   REQ7
Unit Test :   testFunctionCalculationTime()
Description :   Test if the computation of the function is performed within the desired amount of time to meet the requirement.

References

[1] Exponential Growth and Decay,
https://virtuallearningacademy.net/VLA/LessonDisplay/Lesson6156/MATHAL
GIIBU17Exponential_Decay.html. Accessed 31 July 2022.

[2] Injectivity and Surjectivity of Exponential Function,
https://math.stackexchange.com/q/349902. Accessed 31 July 2022

[3] Exponential Function Graph,
https://www.basic-mathematics.com/exponential-function.html. Accessed 31
July 2022.

[4] "ISO/IEC/IEEE International Standard - Systems and software en-
gineering – Life cycle processes – Requirements engineering," in
ISO/IEC/IEEE 29148:2018(E) , vol., no., pp.1-30, 30 Nov. 2018, doi:
10.1109/IEEESTD.2018.8559686.

[5] Pseudocode,
https://en.wikipedia.org/w/index.php?title=Pseudocode&oldid=1101060705.
Accessed 31 July 2022

[6] Power Function,
https://afteracademy.com/blog/calculate-power-function. Accessed 31 July
2022.

[7] Algorithms,
https://mathonweb.com/help_ebook/html/algorithms.htm. Accessed 31 July
2022.

[8] Advantages and Disadvantages of Recursion
https://medium.com/@williambdale/recursion-the-pros-and-cons-
76d32d75973a. Accessed 31 July 2022.

[9] IntelliJ Debugger,
https://www.jetbrains.com/help/idea/debugging-code.html. Accessed 31 July
2022.

[10] Quality Assurance, Software Quality Attributes,
http://www.qasigma.com/2008/12/software-quality-attributes.html. Accessed
31 July 2022.

[11] Checkstyle – Checkstyle 10.3.1,
https://checkstyle.sourceforge.io/. Accessed 31 July 2022.

[12] Junit 4,
https://www.vogella.com/tutorials/JUnit4/article.html. Accessed 31 July
2022.