Unit Testing Summary Report

1. Introduction

1.1: Group Information

Group # 2

Section # 2

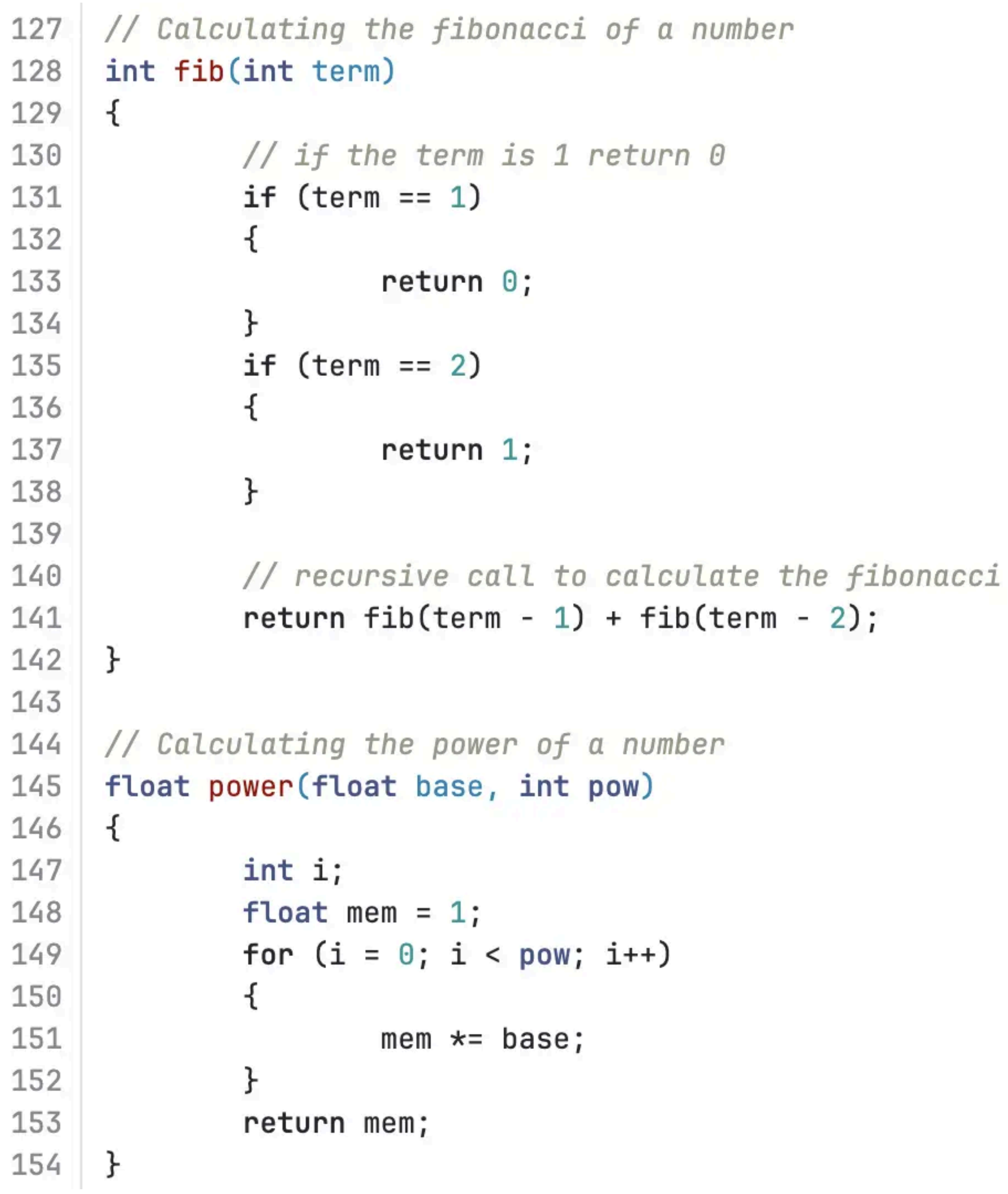
1.2: Code Base

Calculator

1.3: Contributors

Name	Student ID
Huraimah Fatima	1305776
Jacob Good	1300566
Duaa Imran	1279038
Cavan Chung	1271222
Yash Tandon	1258096
Daman Kumar	1306900

2. Code-Under-Test (old code) Report



```c
// Calculating the fibonacci of a number
int fib(int term)
{
        // if the term is 1 return 0
        if (term == 1)
        {
                return 0;
        }
        if (term == 2)
        {
                return 1;
        }

        // recursive call to calculate the fibonacci
        return fib(term - 1) + fib(term - 2);
}

// Calculating the power of a number
float power(float base, int pow)
{
        int i;
        float mem = 1;
        for (i = 0; i < pow; i++)
        {
                mem *= base;
        }
        return mem;
}
```

2.1 Boundary Analysis Testing

2.1.1

Field	**Test Case ID**	**Test Case Strategy**	**Test Case Description**	**Preconditions**	**Input Data**	**Expected Result**	**Actual Result**	**Status**	**Additional Notes**
Description	***Unique identifier for test case.***	***List strategy used for this test case (Structured Basis Testing, Data Flow Testing, Boundary Analysis)***	***Brief description of the test case, indicating the function being tested and the expected behavior.***	***Any setup required before running the test.***	***Parameters or input values passed to the function.***	***Expected outcome after running the function with provided input.***	***Observed outcome after executing the test.***	***Indicate if the test passed or failed based on expected vs. actual results.***	***Any additional comments, observations, or issues noted during testing.***
Calculating the fibonacci of a number	TC003 (Ex. 1)	Boundary Analysis	"Testing the edge case of seeing if one(lowest positive) can be entered"	"The fib function should be defined and accessible."	term = 1	"The function should return 1"	1	pass	
Calculating the fibonacci of a number	TC004 (Ex. 1)	Boundary Analysis	"Testing the edge case of seeing if 47 can be enter"	"The fib function should be defined and accessible."	term = 47	"The function should return 1836311903"	1836311903	pass	
Calculating the power of a number	TC006 (Ex. 2)	Boundary Analysis	"Testing edge case of lower bound for power"	"Function is defined and accessible."	pow = 0 Base = 2	Should return 0.	0	pass	
Calculating the power of a number	TC007 (Ex. 2)	Boundary Analysis	"Testing edge case of lower bound for power"	"Function is defined and accessible."	pow = 1 Base = 2	Should return 2.	2	pass	
Calculating the power of a number	TC008 (Ex. 2)	Boundary Analysis	"Testing edge case of lower bound for power"	"Function is defined and accessible."	pow = -1 Base = 2	Should return 0.5.	1	fail	
Calculating the power of a number	TC009 (Ex. 2)	Boundary Analysis	"Testing edge case of upper bound for power"	"Function is defined and accessible."	pow = 127 Base = 2	Should return 170141183460469231731687303715884105728	-2147483648	fail	2^127≈1.7× 10^38, which is outside the range of a 32-bit float.
Calculating the power of a number	TC010 (Ex. 2)	Boundary Analysis	"Testing edge case of upper bound for power"	"Function is defined and accessible."	pow = 128 Base = 2	Should return an error	-2147483648	fail	2^127≈1.7× 10^38, which is outside the range of a 32-bit float.

2.1.2 Errors in Code:

Defect	Major/Minor	Description
Testing edge case of lower bound for power	Minor	The function return the wrong number as the if statement is off by 1 so it exited
Range of 32-bit floats not accounted for in power function	Major	The test cases test with powers that are out of bounds of a 32 but float that the program can hold.
Negative powers	Major	Negative powers fail as they do not return the expected outcome. In the code of the function, there is a for loop that sets int i to 0. The power is checked to be greater than 0 which fails any cases when power is less than 1.

2.1.3 Potential Solutions:

Defect One:

Changing the for loop statement to be i <= pow rather than just < as this does not run the code for 1 at the moment.

Defect Two:

Using a double or long unsigned double instead of float.

Defect Three:

Removing the condition of power being greater than 0 to give more diversity in power Calculation.

2.1.4: Members:

Daman Kumar
Jacob Good

2.2 Structured Basis Testing

2.2.1:

Field	Test Case ID	Test Case Strategy	Test Case Description	Preconditions	Input Data	Expected Result	Actual Result	Status	Additional Notes
Calculating a number in the fibonacci sequence of a given term	HC001	Structured Basis Testing	This test case is going to be testing the fibonacci function 'fib'. The test case is testing the first conditional statement for case term = 1. The expected result is 0.	The set up required is the option of the scientific calculator being selected first from the main program and then selecting option 5 for fibonacci in the scientific calculator menu.	term = 1	The function should return 0	0	pass	
Calculating a number in the fibonacci sequence of a given term	HC002	Structured Basis Testing	This test case is going to be testing the fibonacci function 'fib'. The test case is testing the first conditional statement for case term = 2. The expected result is 1.	The set up required is the option of the scientific calculator being selected first from the main program and then selecting option 5 for fibonacci in the scientific calculator menu.	term = 2	The function should return 1	1	pass	
Calculating a number in the fibonacci sequence of a given term	HC003	Structured Basis Testing	This test case is going to be testing the fibonacci function 'fib'. The test case is testing the first conditional statement for case term = 47. The expected result is 2971215073.	setup required is the option of the scientific calculator being selected first from the main program and then select option 5 for fibonacci in the scientific calculator menu.	term = 47	The function should return 2971215073	2971215073	pass	
Calculating the power of a number	HC004	Structured Basis Testing	The test case is going to be testing the power function by each statement in the block. The test case is testing for accurate results when the base is 0 and the power is 0 as well.	setup required is the option of the scientific calculator being selected first from the main program and then select option 1 for power function in the scientific calculator menu.	base = 0, pow = 0	The function should return 1	1	pass	
Calculating the power of a number	HC005	Structured Basis Testing	The test case is going to be testing the power function	setup required is the option of the scientific	base = 2, pow = 0	The function should return 1	1	pass	

			by each statement in the block. The test case is testing for accurate results when the base is 2 and the power is 0.	calculator being selected first from the main program and then select option 1 for power function in the scientific calculator menu.					
Calculating the power of a number	HC006	Structured Basis Testing	The test case is going to be testing the power function by each statement in the block. The test case is testing for accurate results when the base is 0 and the power is 1.	setup required is the option of the scientific calculator being selected first from the main program and then select option 1 for power function in the scientific calculator menu.	base = 0, pow = 1	The function should return 0	0	pass	
Calculating the power of a number	HC007		The test case is going to be testing the power function by each statement in the block. The test case is testing for accurate results when the base is a float 25.35 and the power is 4.	setup required is the option of the scientific calculator being selected first from the main program and then select option 1 for power function in the scientific calculator menu.	base = 2.5 12pow = 4	Function should return 412963.677506	412963.677506	pass	

2.2.2: Errors in code:

Defect	Major/Minor	Description
None	0	All tests passed

2.2.3: Potential Solutions:

2.2.4: Members:

Huraimah Fatima

Cavan Chung

2.3 Data Flow Testing

2.3.1:

Field	Test Case ID	Test Case Strategy	Test Case Description	Preconditions	Input Data	Expected Result	Actual Result	Status	Additional Notes
Description	DF001	Data Flow Testing	Check fib function returns 0 if term is entered as 1.	Input 1 as the term	Term = 1	Should return 0	**0**	pass	
Description	DF002	Data Flow Testing	The fib function returns 1 if the term is entered as 2.	Input 2 as the term	Term = 2	Should return 1	**1**	pass	
Description	DF003	Data Flow Testing	Check fib returns 3 if term is entered as 5	Input 5 as the term	Term = 5	Should return 3	**3**	pass	
Description	DF004	Data Flow Testing	Check that the power function returns correct output if float number is inputted as base	Input 2.5 as the base and 4 as the power	Base = 2.5 Power = 4	Should return 39.0625	39.065	pass	
Description	DF005	Data Flow Testing	Check that the power function returns correct output if pow = 0	Input 5 as the base and 0 as the power	Base = 5 Power = 0	Should return 1 as a float	1.00	pass	
Description	DF006	Data Flow Testing	Check that the power function returns correct output if float number is inputted as power	Input 5 as the base and 9.7 as the power	Base = 5 Power = 9.7	Should return 6025721.316	1953125	fail	
			Check that the				**0**	pass	

Description	DF007	Data Flow Testing	power function returns correct value when base is 0	Input 0 as the base and 7 as the power	Base = 0 Power = 7	Should return 0			
Description	DF008	Data Flow Testing	Check that the power function returns correct value when power is a negative integer	Input 17 as the base and -3 as the power	Base = 17 Power = -3	Should return 0.0002035416243	1	fail	
Description	DF009	Data Flow Testing	Check that the power function returns correct value when base is a negative integer	Input is -6 as the base and 5 as power	Base = -6 Power = 5	Should return -7776	1	fail	
Description	DF010	Data Flow Testing	Check that the power function returns correct value when base is a negative float	Input is -6.7 and 4	Base = -6.7 Power = 4	Should return 2015.1121	2015.1121	pass	
Description	DF011	Data Flow Testing	Check that the power function returns correct value when power is a negative float	Input is 8 as base and power is -4.5	Base = 8 Power = -4.5	Should return 0.0000863167…	1	fail	
Description	DF012	Data Flow Testing	Check that the power function returns correct value when power is a negative integer and the base is also a negative integer	Input is -8 as base and power is -4	Base = -8 Power = -4	Should return 0.000244140625	1	fail	
			Check that the power function returns correct				1	fail	

Description	DF013	Data Flow Testing	value when power is a negative float and the base is also a negative float	Input is -9.8 as base and power is -2.5	Base = -9.8 Power = -2.5	Should return a complex number			

2.3.2: Errors in code:

Defect	Major/Minor	Description
Unable to take negative integer as the exponent in the power function	Major	Negative powers are unaccounted for in the power( ) function. As such, a negative power always returns a value of 1.
Unable to take a float number as an exponent in the power function.	Major	The power function is defined as power(float base, int exponent), which renders the function unable to compute float powers.
Unable to take negative integer as base	Major	Negative bases are unaccounted for in the power function. As such, a negative base returns a value of 1.
Unable to take a negative float number as an exponent in the power function.	Major	The power function is defined as power(float base, int exponent), which renders the function unable to compute negative float powers.
Unable to take a negative float number as an exponent in the power	Major	The power function is defined as power(float base, int exponent), which renders the function unable to compute negative float powers or negative float bases.

function, and is unable to take a negative float base.		
Unable to take a negative integer as an exponent in the power function, and is unable to take a negative integer base.	Major	A negative power and negative base implementation is not included in the original power function. As such, this operation will always return 1.

2.3.3: Potential Solutions:

Defect One:

Code an implementation of the power function for when the value of the exponent is less than 0.

Defect Two:

Create multiple implementations of the power function, each taking different parameters to account for the different types of inputs the user may have. In this case the function should be created in order to compute a float power.

Defect Three:

Code an implementation of the power function for when the value of the base is less than 0.

Defect Four:

Create multiple implementations of the power function, each taking different parameters to account for the different types of inputs the user may have. In this case it should be able to compute a negative float power.

Defect Five:

Create multiple implementations of the power function, each taking different parameters to account for the different types of inputs the user may have. In this case it should be able to compute a negative float power and a negative float base.

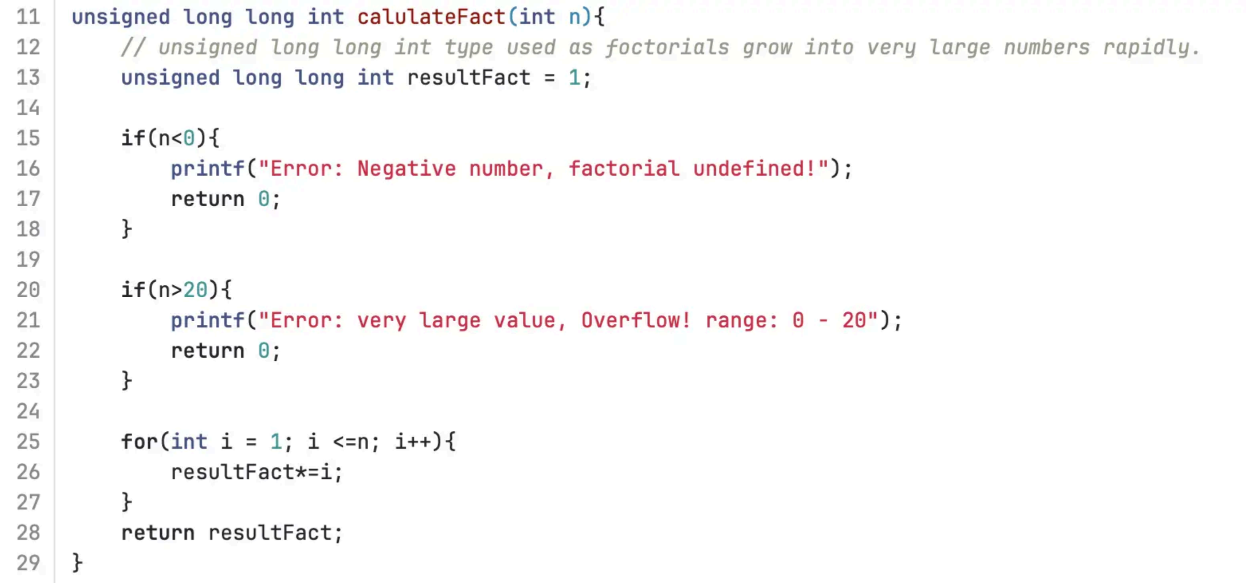
Defect Six:

Create multiple implementations of the power function, each taking different parameters to account for the different types of inputs the user may have. In this case it should be able to compute a negative integer power and a negative integer base.

2.3.4: Members:

Yash Tandon
Duaa Imran

3. New Code Report



```c
unsigned long long int calulateFact(int n){
    // unsigned long long int type used as foctorials grow into very large numbers rapidly.
    unsigned long long int resultFact = 1;

    if(n<0){
        printf("Error: Negative number, factorial undefined!");
        return 0;
    }

    if(n>20){
        printf("Error: very large value, Overflow! range: 0 - 20");
        return 0;
    }

    for(int i = 1; i <=n; i++){
        resultFact*=i;
    }
    return resultFact;
}
```

3.1 Boundary Analysis Testing

3.1.1

Field	Test Case ID	Test Case Strategy	Test Case Description	Preconditions	Input Data	Expected Result	Actual Result	Status	Additional Notes
Description	HJ001 (Ex. 3)	Boundary Analysis	"Testing edge case of upper bound of the range by putting an input of greater than 20"	Select the scientific calculator from the main menu then select the option for calculating factorial.	n = 20	Should return 0	0	Pass	
Description	HJ002 (Ex. 3)	Boundary Analysis	"Test case for when we go below lower bound by entering a negative number"	Select the scientific calculator from the main menu then select the option for calculating factorial.	n = -1	Should return 0	0	Pass	

Description	HJ003 (Ex. 3)	Boundary Analysis	"Testing edge case of lower bound by entering 0, no statements should be entered"	Select the scientific calculator from the main menu then select the option for calculating factorial.	n = 0	Should return 1	1	Pass	

3.1.2 Errors in Code:

Defect	Major/Minor	Description
None	0	All cases passed

3.1.3 Potential Solutions:

3.1.4: Members:

Huraimah Fatima
Jacob Good

3.2 Structured Basis Testing

3.2.1:

Field	Test Case ID	Test Case Strategy	Test Case Description	Preconditions	Input Data	Expected Result	Actual Result	Status	Additional Notes
	HJ004 (Ex. 3)	Structured Basis Testing	“Testing case of a valid input within the range of 0 to 20”	Select the scientific calculator from the main menu then select the option for calculating factorial.	n = 18	Should return 6402373705728000	6402373705728000	Pass	

Description	HJ005 (Ex. 3)	Structured Basis Testing	“Test case for when invalid input is entered, by entering a float because the function can only an integer”	Select the scientific calculator from the main menu then select the option for calculating factorial.	n = 2.2	Should return an error of mismatched data type by returning false	Fail = 2	fail	The test case is supposed to fail so ultimately it passes.
Description	HJ006 (Ex. 3)	Structured Basis Testing	“Test case for statement in for loop by inputting 1 for n”	Select the scientific calculator from the main menu then select the option for calculating factorial.	n = 1	Should return 1	1	Pass	

3.2.2: Errors in code:

Defect	Major/Minor	Description
HJ005	Minor	Test case for when invalid input is entered, by entering a float because the function can only be an integer. The function end up converting the float to an integer and removed the .2

3.2.3: Potential Solutions:

Defect One:

The code works fine, we thought it would fail by entering a float however the code rounds to an integer.

3.2.4: Members:

Huraimah Fatima
Jacob Good

3.3 Data Flow Testing

3.3.1:

Field	Test Case ID	Test Case Strategy	Test Case Description	Preconditions	Input Data	Expected Result	Actual Result	Status	Additional Notes
Description	DC001	Data Flow Testing	Tests the program’s ability to compute the factorial of a small valid integer.	The input n=5 is within the valid range 0-20	n=5	Should return 120	120	Pass	
Description	DC002	Data flow testing	Verifies that the program handles the edge case of 0! correctly, which is defined as 1	Input is 0	n=0	Should return 1	1	Pass	
Description	DC003	Data flow testing	Ensures that inputs larger than 20 are flagged as invalid to avoid overflow	Input is 21	n=21	Should return 0	0	Pass	
Description	DC004	Data flow testing	Tests the computation for the maximum value input without triggering overflow	Input is 20	n=20	Should return 2432902008176640000	2432902008176640000	Pass	
Description	DC005	Data flow testing	Verifies that negative inputs are flagged as invalid	Input is -1	n=-1	Should return zero and print an error message	0	Pass	
Description	DC006	Data flow testing	Ensures the program handles non-integer input correctly by rounding it before computation	Input is 10.5	n=10.5	Should return 362880	362880	Pass	

3.3.2: Errors in code:

Defect	Major/Minor	Description
None	0	All Test cases passed

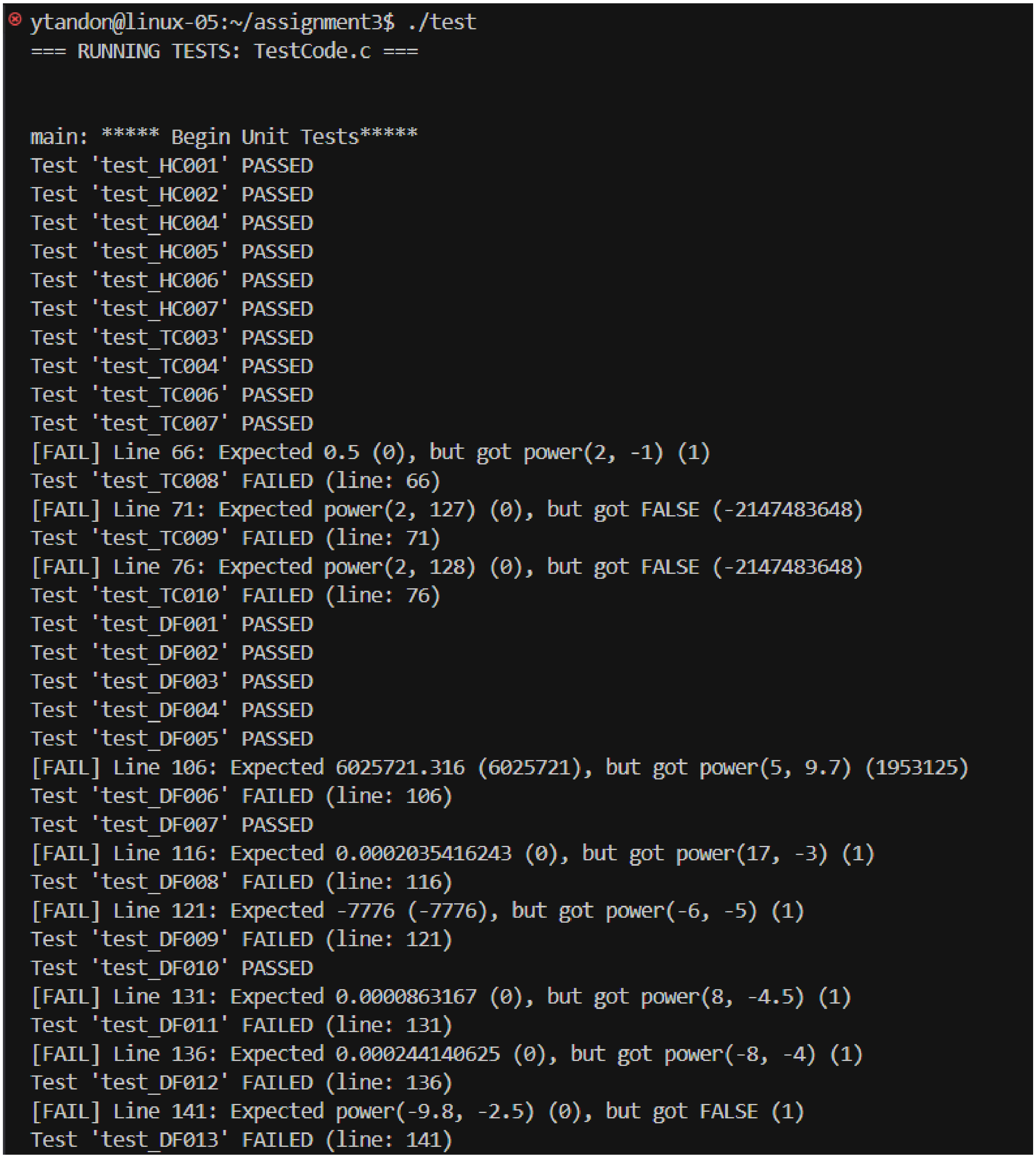
3.3.3: Potential Solutions:

3.3.4: Members:

Cavan Chung

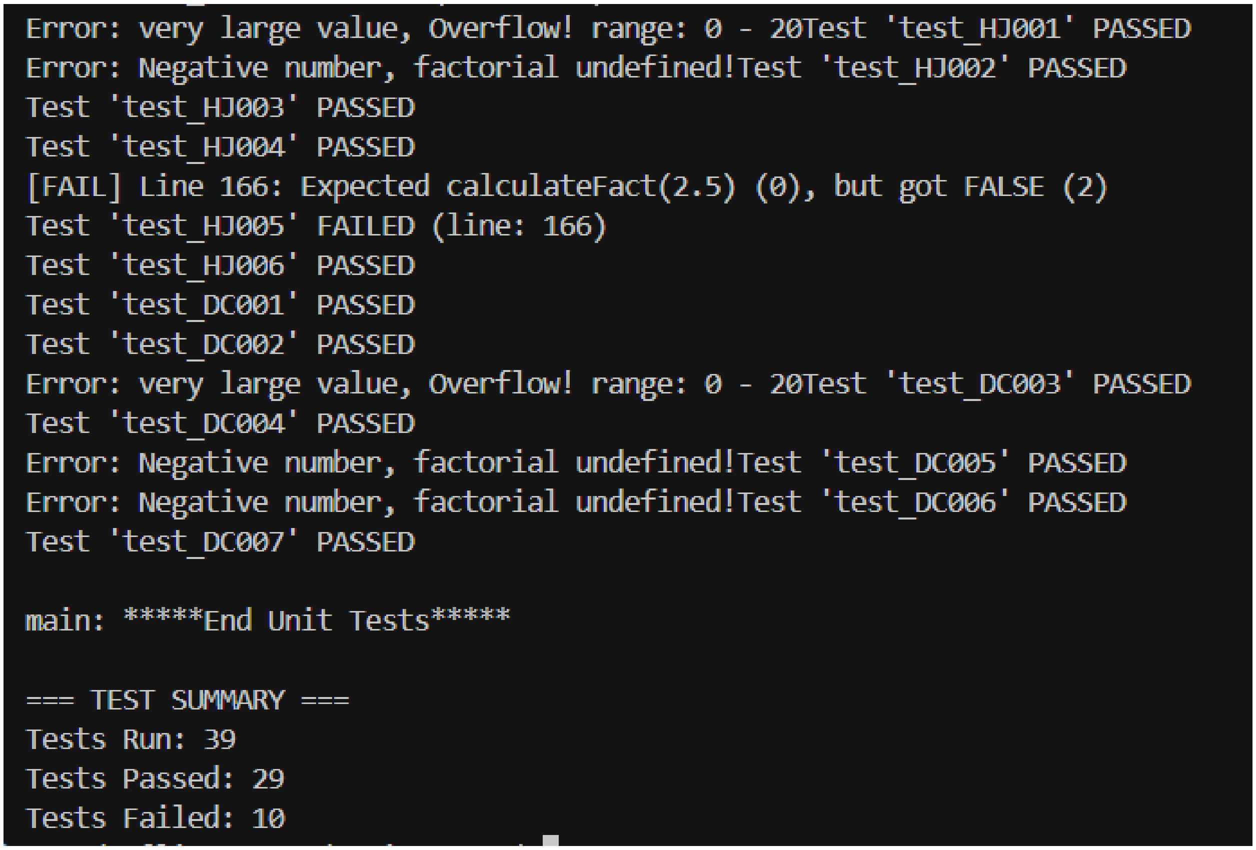
Daman Kumar

4.0. Screenshots of Testing:



```
ytandon@linux-05:~/assignment3$ ./test
=== RUNNING TESTS: TestCode.c ===


main: ***** Begin Unit Tests*****
Test 'test_HC001' PASSED
Test 'test_HC002' PASSED
Test 'test_HC004' PASSED
Test 'test_HC005' PASSED
Test 'test_HC006' PASSED
Test 'test_HC007' PASSED
Test 'test_TC003' PASSED
Test 'test_TC004' PASSED
Test 'test_TC006' PASSED
Test 'test_TC007' PASSED
[FAIL] Line 66: Expected 0.5 (0), but got power(2, -1) (1)
Test 'test_TC008' FAILED (line: 66)
[FAIL] Line 71: Expected power(2, 127) (0), but got FALSE (-2147483648)
Test 'test_TC009' FAILED (line: 71)
[FAIL] Line 76: Expected power(2, 128) (0), but got FALSE (-2147483648)
Test 'test_TC010' FAILED (line: 76)
Test 'test_DF001' PASSED
Test 'test_DF002' PASSED
Test 'test_DF003' PASSED
Test 'test_DF004' PASSED
Test 'test_DF005' PASSED
[FAIL] Line 106: Expected 6025721.316 (6025721), but got power(5, 9.7) (1953125)
Test 'test_DF006' FAILED (line: 106)
Test 'test_DF007' PASSED
[FAIL] Line 116: Expected 0.0002035416243 (0), but got power(17, -3) (1)
Test 'test_DF008' FAILED (line: 116)
[FAIL] Line 121: Expected -7776 (-7776), but got power(-6, -5) (1)
Test 'test_DF009' FAILED (line: 121)
Test 'test_DF010' PASSED
[FAIL] Line 131: Expected 0.0000863167 (0), but got power(8, -4.5) (1)
Test 'test_DF011' FAILED (line: 131)
[FAIL] Line 136: Expected 0.000244140625 (0), but got power(-8, -4) (1)
Test 'test_DF012' FAILED (line: 136)
[FAIL] Line 141: Expected power(-9.8, -2.5) (0), but got FALSE (1)
Test 'test_DF013' FAILED (line: 141)
```



```
Error: very large value, Overflow! range: 0 - 20Test 'test_HJ001' PASSED
Error: Negative number, factorial undefined!Test 'test_HJ002' PASSED
Test 'test_HJ003' PASSED
Test 'test_HJ004' PASSED
[FAIL] Line 166: Expected calculateFact(2.5) (0), but got FALSE (2)
Test 'test_HJ005' FAILED (line: 166)
Test 'test_HJ006' PASSED
Test 'test_DC001' PASSED
Test 'test_DC002' PASSED
Error: very large value, Overflow! range: 0 - 20Test 'test_DC003' PASSED
Test 'test_DC004' PASSED
Error: Negative number, factorial undefined!Test 'test_DC005' PASSED
Error: Negative number, factorial undefined!Test 'test_DC006' PASSED
Test 'test_DC007' PASSED

main: *****End Unit Tests*****

=== TEST SUMMARY ===
Tests Run: 39
Tests Passed: 29
Tests Failed: 10
```

Testing summary

Total Major Defects Found: 8
Total Unique Major Defects Found: 8
Total Minor Defects Found: 2
Total Unique Minor Defects Found: 2
Total Unique Major / Total Unique Defects: 8/ 10 = 80.0 %
Total Review Time: 120 minutes
Total Size Reviewed: 37 LOC
Overall Review Rate (LOC / hour) : 18.5 LOC / hr
Overall Defect Detection Rate (maj. / hour) : 4 defects / hour