ECE 4960, Spring 2018 **Assignemnt 1**
Name: Vishisht M. Tiwari, Deepak Agrawal
NetID: vmt28, da475

1 Structure of Program

The program is created in form of 9 files with 1 header file, 1 main file and 7 different classes implementing different sections of the program. All these files have been described below:

1. Assignment1_Header.h - Common header for all files
2. Assignment1_Main.cpp - Creates an object for all classes and runs all the functions
3. Integer_Overflow.cpp - Different tests for causing integer overflow
4. Floating_Overflow.cpp - Test for floating overflow
5. Floating_Operations.cpp - Generation, behavior, detection and interaction of exceptions such as INF, NINF, NAN with each other and signed 0
6. Signed_Zero.cpp - Behavior of Signed zero
7. Gradual_Underflow.cpp - Testing for gradual underflow
8. Calculating_Pi.cpp - Testing precision by calculating the value of pi

All the 7 different classes have been declared in the header file and the functions are initialized in their respective files. The main file creates an object of all the classes and implements the different functions of each class in an order.

The header file also includes the global variables such as infinity(INF), negative infinity(NINF), Not a Number(NAN), Positive Zero(POSZERO) and Negative Zero(NEGZERO). All these variables are declared in the header file but are initialized in the 'Floating_Operations.cpp' file.

The header file also contains a global function called 'Check_For_Exceptions_Zeros(x)'. If it finds that input is either an exception or zero, then it throws the corresponding exception. On the other hand, if it finds that the input was neither an exception, nor signed zero, then it returns the input in form of a string along with a message saying that the number in question is neither an exception nor signed zero.

The program was tested on MAC OS X and Linuc Ubuntu.

2 Observation of exception rules

The following sections describe the observation of behavious of exception rules.

2.1 Integer Overflow

The section is implemented in the 'Integer_Overflow.cpp' file. In this section, different methods were tried to find the value at which integer starts to overflow. This methods have been described below:

2.1.1 Integer Overflow using multiplication

Design
Integer overflow was first evaluated using a multiplication method. In this method, 2 hard-coded numbers were multiplied to see if integer overflows or not. To cause an overflow, $1,000,000,000$ was multiplied with 3. This would cause an integer overflow because the largest integer value possible in C++ is $2,147,483,647$. Another multiplication was performed between $1,000,000,000$ was multiplied with 2 to show the case where integer overflow does not occur.

Exception Check
To check if an exception has occurred or not, the multiplied number is divided with one of the numbers used for multiplication. If the answer of this division is equal to the other number used for multiplication, then no overflow has occurred. If this number is not same then overflow has occurred. This has been shown in the following example:
Lets say:

$$1000000000 * 3 = x$$

Now when x is divided by $1,000,000,000$, the answer should be 3. If the answer is 3 then overflow has not occurred while if the answer is not 3, then overflow has occurred.

Result
In case of MAC OS X, integer overflow occurred when $1,000,000,000$ was multiplied with 3 and no overflow occurred when $1,000,000,000$ was multiplied with 2. This was expected as the maximum value an integer can hold is $2,147,483,647$.

In Linux the same result was observed.

2.1.2 Integer Overflow using factorials

Design
Next method used for integer overflow evaluation was factorials. Different values of factorials were tried to create integer overflow situation. Using a while loop, all factorials from 1 and onwards were tried until an integer overflow occurred. This overflow was then reported to the user.

Exception Check
To check if an overflow has occurred, every factorial value was divided by the number for

which factorial is being calculated and then is divided by the factorial of a number less than the one provided. This has been shown below:

$$\frac{\frac{n!}{n}}{(n-1)!}$$

Because the factorial value is started from 1, (n-1)! is initialized as 1 because 0! is 1. The value of factorial at the end of every loop is saved to check overflow in the next loop.

If the result of the above division is 1, then no integer overflow has occurred but if the answer is not 1, then integer overflow has occurred.

Result
In case of MAC OS X, the exception occurred at 13!. This was expected because 13! is $6,227,020,800$ and the largest value, an integer can hold is $2,147,483,647$. The value of 12! is $479,001,600$ hence it did not overflow at 12!.

In Linux same result was observed.

2.1.3 Integer Overflow using Fibonacci sequence

Design
The last method that was used for integer overflow was the Fibonacci sequence. In this, the value of the n^{th} value was saved in an integer variable. In Fibonacci sequence, the n^{th} value is calculated using the sum of $n-1^{th}$ and $n-2^{th}$ value. This test will observe for which value of Fibonacci sequence does the integer overflow.

Checking Exception
Whether the integer has overflown or not is checked using the subtraction where the $n-1^{th}$ and $n-2^{th}$ values are subtracted from n^{th} value. This has been shown below.

$$((n^{th} - (n-1)^{th}) - (n-2)^{th})$$

If the above value is zero, then integer has not overflown but if its not zero, then integer has overflown. The $(n-1)^{th}$ is initialized as 0 and $(n-2)^{th}$ is initialized as 1.

Result
In case of MAC OS X, the exception occurred at n = 46 in the Fibonacci sequence. This was expected because at n = 46, the value in Fibonacci sequence is $2,971,215,073$ and the largest value, an integer can hold, is $2,147,483,647$. The value of n = 45 is $1,836,311,903$ hence it did not overflow at n = 45 in the Fibonacci sequence.

In Linux same result was observed.

2.2 Division by 0

Design
In this test, a hard-coded value was divided by 0 to see what the result is. For this test, 4 was divided by 0 to observe what happens in division by 0 case. This, in most cases, would crash out the program, hence a user interface was created where, if the user did not want to run this test, then they can choose to do so by not writing 'cont' as the user input.

Result
In case of MAC OS X, the program crashed out due to a floating point exception.

In Linux same result was observed.

2.3 Floating Point Overflows

Design
C++ reports whenever a floating point has overflown and we will use this reporting to check for which iteration, floating point has overflown. For causing the overflow, a while loop is used which keeps multiplying the 10 to a double until it overflows. In a way, this is checking for what value n does the value 10^n overflows in a double.

Exception Check
C++ reports about an overflow using a flag called $'STD :: FE\_OVERFLOW'$. This value stays at 0 when no overflow has occurred and is not 0 when an overflow has occurred. This flag is checked at the end of every loop to see if an overflow has occurred after multiplying the number with 10. At the very start of this method, $'STD :: FE\_OVERFLOW'$ flag is set to 0 to make sure no false results are observed due to the previous value of flag.

Result
In case of MAC OS X, this flag was raised at the 309th iteration. This means that at value 10^{309}, the double value overflowed. This was expected as the largest value, a double can hold, is $1.8 * 10^{308}$.

In case of linux, same result was observed.

2.4 Floating Point Operations of INF, NINF, NAN

Design
In this section, INF, NINF(Negative Infinity), NAN, Positive ZERO and Negative ZERO was created in the following way:

$$INF = \frac{1.0}{0.0}$$

$$NINF = -INF$$

$$NANVAL = INF * 0.0$$

$$PositiveZERO = 1/INF$$

$$NegativeZERO = 1/NINF$$

It is important to note here that these values are made global and are then not initialized anywhere else in the program. All other classes use the value of the above variables initialized in this class.

These values are then checked in the constructor when this class is called. This checking is done using the using the global function 'Check_For_Exceptions_Zeros(x)' which checks for all the exceptions and signed zero. If any of this exceptions occurred then it returns those exceptions, otherwise it just returns the number as it is.

The program then checks the behavior of INF, NINF and NAN in the following functions:

1. $\frac{1}{x}$
2. $sin(x)$
3. $exp(x)$

The program also checks the interaction of the above programs with each other and with signed zeros. The following checks were done for understanding the interaction.

1. Addition of each exception with itself
2. Addition of each exception with one another
3. Addition of each exception with signed zero
4. Subtraction of each exception with itself
5. Subtraction of each exception with one another
6. Subtraction of each exception with signed zero
7. Multiplication of each exception with itself
8. Multiplication of each exception with one another
9. Multiplication of each exception with signed zero
10. Division of each exception with itself
11. Division of each exception with one another
12. Division of each exception with signed zero

Exception Check

In all the above checks for understanding behavior and interaction, every output value was checked using the global function called 'Check_For_Exceptions_Zeros' which checks for exception or signed zeros. This is for the detection of the exceptions and signed zeros.

Results

```
The results are below for mac: Value of Positive Infinity + Positive Infinity is: inf
Value of Positive Infinity + Negative Infinity is: nan
Value of Positive Infinity + NAN is: nan
Value of Positive Infinity + Positive 0 is: inf
Value of Positive Infinity + Negative 0 is: inf
Value of Positive Infinity - Positive Infinity is: nan
Value of Positive Infinity - Negative Infinity is: inf
Value of Positive Infinity - NAN is: nan
Value of Positive Infinity - Positive 0 is: inf
Value of Positive Infinity - Negative 0 is: inf
Value of Positive Infinity * Positive Infinity is: inf
Value of Positive Infinity * Negative Infinity is: -inf
Value of Positive Infinity * NAN is: nan
Value of Positive Infinity * Positive 0 is: nan
Value of Positive Infinity * Negative 0 is: nan
Value of Positive Infinity / Positive Infinity is: nan
Value of Positive Infinity / Negative Infinity is: nan
Value of Positive Infinity / NAN is: nan
Value of Positive Infinity / Positive 0 is: inf
Value of Positive Infinity / Negative 0 is: -inf

Value of Negative Infinity + Positive Infinity is: nan
Value of Negative Infinity + Negative Infinity is: -inf
Value of Negative Infinity + NAN is: nan
Value of Negative Infinity + Positive 0 is: -inf
Value of Negative Infinity + Negative 0 is: -inf
Value of Negative Infinity - Positive Infinity is: -inf
Value of Negative Infinity - Negative Infinity is: nan
Value of Negative Infinity - NAN is: nan
Value of Negative Infinity - Positive 0 is: -inf
Value of Negative Infinity - Negative 0 is: -inf
Value of Negative Infinity * Positive Infinity is: -inf
Value of Negative Infinity * Negative Infinity is: inf
Value of Negative Infinity * NAN is: nan
Value of Negative Infinity * Positive 0 is: nan
Value of Negative Infinity * Negative 0 is: nan
Value of Negative Infinity / Positive Infinity is: nan
Value of Negative Infinity / Negative Infinity is: nan
Value of Negative Infinity / NAN is: nan
Value of Negative Infinity / Positive 0 is: -inf
Value of Negative Infinity / Negative 0 is: inf

Value of NAN + Positive Infinity is: nan
Value of NAN + Negative Infinity is: nan
Value of NAN + NAN is: nan
Value of NAN + Positive 0 is: nan
Value of NAN + Negative 0 is: nan
Value of NAN - Positive Infinity is: nan
Value of NAN - Negative Infinity is: nan
Value of NAN - NAN is: nan
Value of NAN - Positive 0 is: nan
Value of NAN - Negative 0 is: nan
Value of NAN * Positive Infinity is: nan
Value of NAN * Negative Infinity is: nan
Value of NAN * NAN is: nan
Value of NAN * Positive 0 is: nan
Value of NAN * Negative 0 is: nan
Value of NAN / Positive Infinity is: nan
Value of NAN / Negative Infinity is: nan
Value of NAN / NAN is: nan
Value of NAN / Positive 0 is: nan
Value of NAN / Negative 0 is: nan
```

The results are below for linux:

```
Propagation and Interaction of INF, NINF and NAN with each other and +0,-0
Value of Positive Infinity + Positive Infinity is: Positive Infinity
Value of Positive Infinity + Negative Infinity is: NAN
Value of Positive Infinity + NAN is: NAN
Value of Positive Infinity + Positive 0 is: Positive Infinity
Value of Positive Infinity + Negative 0 is: Positive Infinity
Value of Positive Infinity - Positive Infinity is: NAN
Value of Positive Infinity - Negative Infinity is: Positive Infinity
Value of Positive Infinity - NAN is: NAN
Value of Positive Infinity - Positive 0 is: Positive Infinity
Value of Positive Infinity - Negative 0 is: Positive Infinity
Value of Positive Infinity * Positive Infinity is: Positive Infinity
Value of Positive Infinity * Negative Infinity is: Negative Infinity
Value of Positive Infinity * NAN is: NAN
Value of Positive Infinity * Positive 0 is: NAN
Value of Positive Infinity * Negative 0 is: NAN
Value of Positive Infinity / Positive Infinity is: NAN
Value of Positive Infinity / Negative Infinity is: NAN
Value of Positive Infinity / NAN is: NAN
Value of Positive Infinity / Positive 0 is: Positive Infinity
Value of Positive Infinity / Negative 0 is: Negative Infinity

Value of Negative Infinity + Positive Infinity is: NAN
Value of Negative Infinity + Negative Infinity is: Negative Infinity
Value of Negative Infinity + NAN is: NAN
Value of Negative Infinity + Positive 0 is: Negative Infinity
Value of Negative Infinity + Negative 0 is: Negative Infinity
Value of Negative Infinity - Positive Infinity is: Negative Infinity
Value of Negative Infinity - Negative Infinity is: NAN
Value of Negative Infinity - NAN is: NAN
Value of Negative Infinity - Positive 0 is: Negative Infinity
Value of Negative Infinity - Negative 0 is: Negative Infinity
Value of Negative Infinity * Positive Infinity is: Negative Infinity
Value of Negative Infinity * Negative Infinity is: Positive Infinity
Value of Negative Infinity * NAN is: NAN
Value of Negative Infinity * Positive 0 is: NAN
Value of Negative Infinity * Negative 0 is: NAN
Value of Negative Infinity / Positive Infinity is: NAN
Value of Negative Infinity / Negative Infinity is: NAN
Value of Negative Infinity / NAN is: NAN
Value of Negative Infinity / Positive 0 is: Negative Infinity
Value of Negative Infinity / Negative 0 is: Positive Infinity

Value of NAN + Positive Infinity is: NAN
Value of NAN + Negative Infinity is: NAN
Value of NAN + NAN is: NAN
Value of NAN + Positive 0 is: NAN
Value of NAN + Negative 0 is: NAN
Value of NAN - Positive Infinity is: NAN
Value of NAN - Negative Infinity is: NAN
Value of NAN - NAN is: NAN
Value of NAN - Positive 0 is: NAN
Value of NAN - Negative 0 is: NAN
Value of NAN * Positive Infinity is: NAN
Value of NAN * Negative Infinity is: NAN
Value of NAN * NAN is: NAN
Value of NAN * Positive 0 is: NAN
Value of NAN * Negative 0 is: NAN
Value of NAN / Positive Infinity is: NAN
Value of NAN / Negative Infinity is: NAN
Value of NAN / NAN is: NAN
Value of NAN / Positive 0 is: NAN
Value of NAN / Negative 0 is: NAN
```

2.5 Signed Zero

Design
The value of Positive Zero and Negative Zero was initialized in the '*Floating_Operations.cpp*' class and the same values are used in this class as well.

When this class is called, the values of positive zero and negative zero are checked in the constructor. This checking is done using the global function called 'Check_For_Exceptions_Zeros' which checks the validity of the values of positive zero and negative zero.

Signed zero handling is then done using the following functions:

1. $log(x)$
2. $sin(x)$
3. $sinc(x)$

The inputs for the above functions are positive zero and negative zero.

Exception Check
The output of all the above functions are first checked using the $'Check_For_Exceptions_Zeros'$ method which checks the validity of the exceptions or signed zero produced from the above functions.

2.5.1 Underflow

In underflow we are checking if the number underflows or not in case of (x-y) and x/y.

The result of this as follows:

3 floating-point precision

For saving the value of pi, we are using a quad precision variable called long variable. The value of pi is calculated using ramanujam formula found online.

The value of pi observed in mac os is:3.14159265358979323851280895940

The value pi observed in linux is: 3.14159265358979323851280895941