

The **IEEE-754 Binary-32 Floating-Point Converter** is a web application that allows users to convert between decimal and binary representations, including handling special cases like NaN (Not a Number). The application provides detailed steps of the conversion process and displays the final result in both binary and hexadecimal formats.

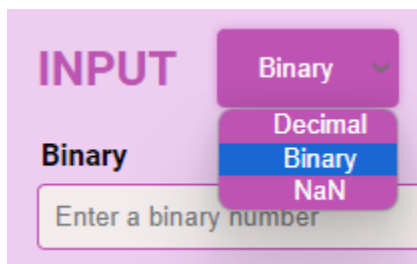
**Developers:**

- Atienza, Marielle Angelene
- Ong, Barron Brandon Conroy
- Rivera, Jose Carlos
- Tolentino, Maxene Allison

## How to Use the IEEE-754 Binary-32 Floating-Point Converter

[Google Drive link](#) to Video Demonstration

### 1. Select Input Type

The image shows a web interface for selecting the input type. On the left, the word "INPUT" is displayed in large purple letters. Below it, the word "Binary" is shown in a smaller font. To the right of "INPUT" is a dropdown menu with four options: "Binary" (selected), "Decimal", "Binary", and "NaN". Below the dropdown menu is a text input field with the placeholder text "Enter a binary number".

Choose the type of input you want to convert.

- ❖ You can pick between **Decimal** (base-10 exponent), **Binary** (base-2 exponent), or **NaN** from the dropdown menu.
- ❖ Depending on your selection, the input fields and labels will adjust accordingly.

### 2. Enter Mantissa and Exponent

#### ❖ Mantissa Input

- **Decimal:** Enter a decimal number (e.g., 15.75)
- **Binary:** Enter a binary number (e.g., 101.01).
- **NaN:** Enter qnan or snan.

#### ❖ Exponent Input

- For **Decimal/Binary:** Enter the exponent (e.g., **5** for  $15.75 \times 10^5$  or **2** for  $101.01 \times 2^2$ ).
- ❖ Both positive (+) and negative (-) symbols are allowed for the Mantissa and Exponent input.

### 3. Click Compute

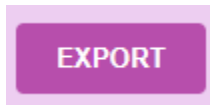
COMPUTE

- ❖ After entering the values, press the 'Compute' button to perform the conversion.
- ❖ The application will validate the inputs and perform the conversion.

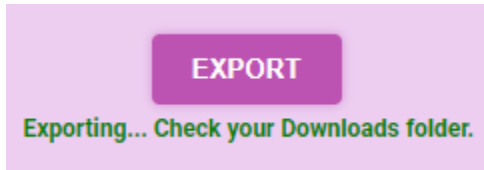
PROCESS			
Binary Equivalent	Normalized Binary	Final Exponent	
1.00111	1.00111	5	
Sign Bit	E'	Significand Fractional Part	Special Case
0	10000100	0011100000000000000000	

- ❖ The results will be displayed in the process section, showing:
  - Binary Equivalent
  - Normalized Binary
  - Final Exponent
  - Sign Bit
  - E'
  - Significand Fractional Part
  - Special Case (if any)

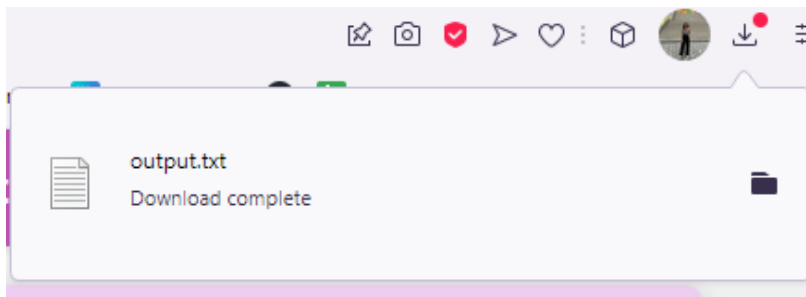
#### 4. Export Results



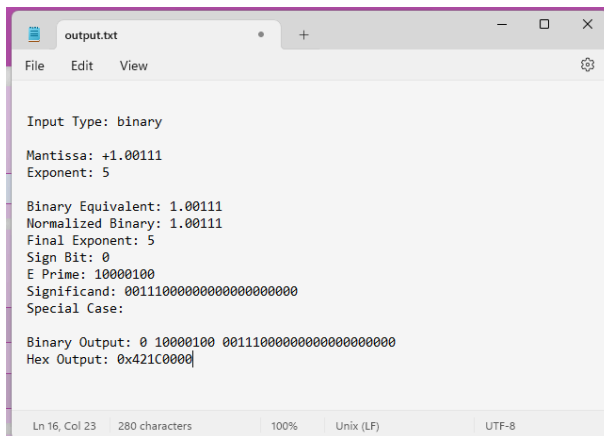
- ❖ If you wish to save the results, click the 'Export' button.



- ❖ If the inputs are valid, a notification will appear indicating that the file is being exported.



- ❖ The result will be saved as a text file on your local machine.



- ❖ The **text file** will include the input values, conversion process, and the binary and hexadecimal outputs.

## Test Cases and Results

### Case 1: Normal with Positive Binary Input (+ symbol included)

Input: **+1.00111**

Exponent: **5**

#### INPUT

Binary 

Binary

+1.00111

Exponent (Base-2)

5

COMPUTE

#### PROCESS

Binary Equivalent

1.00111

Normalized Binary

1.00111

Final Exponent

5

Sign Bit

0

E'

10000100

Significand Fractional Part

001110000000000000000000

Special Case

#### OUTPUT

Final Answer (Binary)

0 10000100 001110000000000000000000

Final Answer (Hex)

0x421C0000

EXPORT

## Case 2: Normal with Positive Binary Input (+ symbol excluded)

Input: 1.101

Exponent: 3

INPUT				
Binary	Exponent (Base-2)			COMPUTE
1.101	3			
PROCESS				
Binary Equivalent	Normalized Binary	Final Exponent		
1.101	1.101	3		
Sign Bit	E'	Significand Fractional Part	Special Case	
0	10000010	1010000000000000000000		
OUTPUT				
Final Answer (Binary)	Final Answer (Hex)			EXPORT
0 10000010 1010000000000000000000	0x41500000			

Test case #1 and #2 shows that the program can accept a positive binary with and without a positive symbol included as a valid input. It also shows that the program can process the binary and exponent (base-2) input to provide the appropriate Binary Equivalent, Normalized Binary, Final Exponent, Sign Bit, E prime, and Significand Fractional Part. The blank special case indicates that the sample test case is normal and does not belong to any special cases in the conversion of binary 32 floating-point. The program also outputs the final answer in binary and in hex successfully.

The following test cases will also show a formatted final answer in binary, following the format [sign bit] [E'] [Significand]; and a formatted final answer in hex following the format [0x][final answer in hex].

### Case 3: Normal with Negative Binary Input

Input: **-100.111**

Exponent: **-7**

#### INPUT

Binary 

Binary

-100.111

Exponent (Base-2)

-7

COMPUTE

#### PROCESS

Binary Equivalent

100.111

Normalized Binary

-1.00111

Final Exponent

-5

Sign Bit

1

E'

01111010

Significand Fractional Part

001110000000000000000000

Special Case

#### OUTPUT

Final Answer (Binary)

1 01111010 001110000000000000000000

Final Answer (Hex)

0xBD1C0000

EXPORT

This test case shows that the program can accept a negative binary with a negative symbol, and a negative exponent included as a valid input. The blank special case indicates that the sample test case is normal and does not belong to any special cases in the conversion of binary 32 floating-point.

#### Case 4: Normal with Negative Binary and Positive Exponent Inputs

Input: **-0.000100111**

Exponent: **15**

##### INPUT

Binary

Binary

-0.000100111

Exponent (Base-2)

15

COMPUTE

##### PROCESS

Binary Equivalent

0.000100111

Normalized Binary

-1.00111

Final Exponent

11

Sign Bit

1

E'

10001010

Significand Fractional Part

001110000000000000000000

Special Case

##### OUTPUT

Final Answer (Binary)

1 10001010 001110000000000000000000

Final Answer (Hex)

0xC51C0000

EXPORT

This test case shows that the program can accept a negative binary with a negative symbol, and a positive exponent included as a valid input. The blank special case indicates that the sample test case is normal and does not belong to any special cases in the conversion of binary 32 floating-point.

### Case 5: Normal with Positive Binary and Negative Exponent Inputs

Input: 0.000100111

Exponent: -15

#### INPUT

Binary ▾

Binary

0.000100111

Exponent (Base-2)

-15

COMPUTE

#### PROCESS

Binary Equivalent

0.000100111

Normalized Binary

1.00111

Final Exponent

-19

Sign Bit

0

E'

01101100

Significand Fractional Part

001110000000000000000000

Special Case

#### OUTPUT

Final Answer (Binary)

0 01101100  
001110000000000000000000

Final Answer (Hex)

0x361C0000

EXPORT

This test case shows that the program can accept a positive binary without a positive symbol, and a negative exponent included as a valid input. The blank special case indicates that the sample test case is normal and does not belong to any special cases in the conversion of binary 32 floating-point.



### Case 6: Normal with Positive Decimal Input (+ symbol included)

Input: **+4.0**

Exponent: **0**

#### INPUT

Decimal

Decimal

+4.0

Exponent (Base- 10)

0

COMPUTE

#### PROCESS

Binary Equivalent

00000100.0

Normalized Binary

1.000

Final Exponent

2

Sign Bit

0

E'

10000001

Significand Fractional Part

000000000000000000000000

Special Case

#### OUTPUT

Final Answer (Binary)

0 10000001 000000000000000000000000

Final Answer (Hex)

0x40800000

EXPORT

### Case 7: Normal with Positive Decimal Input (+ symbol excluded)

Input: **5.75**

Exponent: **2**

#### INPUT

Decimal

Decimal

5.75

Exponent (Base-10)

2

COMPUTE

#### PROCESS

Binary Equivalent

00000101.11

Normalized Binary

1.0111

Final Exponent

4

Sign Bit

0

E'

10000011

Significand Fractional Part

011100000000000000000000

Special Case

#### OUTPUT

Final Answer (Binary)

0 10000011 011100000000000000000000

Final Answer (Hex)

0x41B80000

EXPORT

Test case #6 and #7 shows that the program can accept a positive decimal with and without a positive symbol included as a valid input. The blank special case indicates that the sample test case is normal and does not belong to any special cases in the conversion of binary 32 floating-point.

### Case 8: Normal with Negative Decimal Input

Input: 5.75

Exponent: -3

#### INPUT

Decimal ▾

Decimal

5.75

Exponent (Base- 10)

-3

COMPUTE

#### PROCESS

Binary Equivalent

00000101.11

Normalized Binary

1.0111

Final Exponent

-1

Sign Bit

0

E'

01111110

Significand Fractional Part

011100000000000000000000

Special Case

#### OUTPUT

Final Answer (Binary)

0 01111110  
011100000000000000000000

Final Answer (Hex)

0x3F380000

EXPORT

This test case shows that the program can accept a negative decimal and a negative exponent (base-10) included as a valid input. The blank special case indicates that the sample test case is normal and does not belong to any special cases in the conversion of binary 32 floating-point.

Case 9: Normal with Positive Zero Decimal Input			
Input: 0.0		Exponent: 0	
<div style="display: flex; justify-content: space-between; align-items: center;"> <div> <b>INPUT</b> <span style="background-color: #e91e63; color: white; padding: 2px 5px; border-radius: 3px;">Decimal</span> </div> </div> <div style="display: flex; justify-content: space-between; margin-top: 10px;"> <div style="width: 45%;"> <b>Decimal</b>  <input style="width: 90%;" type="text" value="0.0"/> </div> <div style="width: 45%;"> <b>Exponent (Base-10)</b>  <input style="width: 90%;" type="text" value="0"/> </div> <div style="width: 10%; text-align: right;"> <b>COMPUTE</b> </div> </div>			
<div style="display: flex; justify-content: space-between; align-items: center;"> <div><b>PROCESS</b></div> </div> <div style="display: flex; justify-content: space-between; margin-top: 10px;"> <div style="width: 25%;"> <b>Binary Equivalent</b>  <input style="width: 95%;" type="text" value="00000000.0"/> </div> <div style="width: 25%;"> <b>Normalized Binary</b>  <input style="width: 95%;" type="text" value="0.0"/> </div> <div style="width: 25%;"> <b>Final Exponent</b>  <input style="width: 95%;" type="text" value="0"/> </div> </div> <div style="display: flex; justify-content: space-between; margin-top: 10px;"> <div style="width: 25%;"> <b>Sign Bit</b>  <input style="width: 95%;" type="text" value="0"/> </div> <div style="width: 25%;"> <b>E'</b>  <input style="width: 95%;" type="text" value="00000000"/> </div> <div style="width: 25%;"> <b>Significand Fractional Part</b>  <input style="width: 95%;" type="text" value="000000000000000000000000"/> </div> <div style="width: 25%;"> <b>Special Case</b>  <input style="width: 95%;" type="text" value="Positive Zero"/> </div> </div>			
<div style="display: flex; justify-content: space-between; align-items: center;"> <div><b>OUTPUT</b></div> </div> <div style="display: flex; justify-content: space-between; margin-top: 10px;"> <div style="width: 45%;"> <b>Final Answer (Binary)</b>  <input style="width: 95%;" type="text" value="0 00000000 000000000000000000000000"/> </div> <div style="width: 45%;"> <b>Final Answer (Hex)</b>  <input style="width: 95%;" type="text" value="0x00000000"/> </div> <div style="width: 10%; text-align: right;"> <b>EXPORT</b> </div> </div>			

Case 10: Normal with Positive Zero Binary Input			
Input: 0.0		Exponent: 0	
<div style="display: flex; justify-content: space-between; align-items: center;"> <div> <b>INPUT</b> <span style="background-color: #e91e63; color: white; padding: 2px 5px; border-radius: 3px;">Binary</span> </div> </div> <div style="display: flex; justify-content: space-between; margin-top: 10px;"> <div style="width: 45%;"> <b>Binary</b>  <input style="width: 90%;" type="text" value="0.0"/> </div> <div style="width: 45%;"> <b>Exponent (Base-2)</b>  <input style="width: 90%;" type="text" value="0"/> </div> <div style="width: 10%; text-align: right;"> <b>COMPUTE</b> </div> </div>			
<div style="display: flex; justify-content: space-between; align-items: center;"> <div><b>PROCESS</b></div> </div> <div style="display: flex; justify-content: space-between; margin-top: 10px;"> <div style="width: 25%;"> <b>Binary Equivalent</b>  <input style="width: 95%;" type="text" value="0.0"/> </div> <div style="width: 25%;"> <b>Normalized Binary</b>  <input style="width: 95%;" type="text" value="0.0"/> </div> <div style="width: 25%;"> <b>Final Exponent</b>  <input style="width: 95%;" type="text" value="0"/> </div> </div> <div style="display: flex; justify-content: space-between; margin-top: 10px;"> <div style="width: 25%;"> <b>Sign Bit</b>  <input style="width: 95%;" type="text" value="0"/> </div> <div style="width: 25%;"> <b>E'</b>  <input style="width: 95%;" type="text" value="00000000"/> </div> <div style="width: 25%;"> <b>Significand Fractional Part</b>  <input style="width: 95%;" type="text" value="000000000000000000000000"/> </div> <div style="width: 25%;"> <b>Special Case</b>  <input style="width: 95%;" type="text" value="Positive Zero"/> </div> </div>			
<div style="display: flex; justify-content: space-between; align-items: center;"> <div><b>OUTPUT</b></div> </div> <div style="display: flex; justify-content: space-between; margin-top: 10px;"> <div style="width: 45%;"> <b>Final Answer (Binary)</b>  <input style="width: 95%;" type="text" value="0 00000000 000000000000000000000000"/> </div> <div style="width: 45%;"> <b>Final Answer (Hex)</b>  <input style="width: 95%;" type="text" value="0x00000000"/> </div> <div style="width: 10%; text-align: right;"> <b>EXPORT</b> </div> </div>			

Test case #9, #10, #11, and #12 shows that the program can accept positive zero binary and decimal, and negative zero binary and decimal as a valid input. The program then identifies the test case as an infinity special case and provides the equivalent final answer in binary and in hex.

The following test cases for zero special case will be having an output (for the process) of the following:

- Sign bit: 0
- E': 0000 0000
- Significand: 000 0000 0000 0000 0000 0000

## Case 11: Normal with Negative Zero Decimal Input

Input: 0.0

Exponent: 0

### INPUT

Decimal ▾

Decimal

-0.0

Exponent (Base-10)

0

COMPUTE

### PROCESS

Binary Equivalent

00000000.0

Normalized Binary

0.0

Final Exponent

0

Sign Bit

1

E'

00000000

Significand Fractional Part

000000000000000000000000

Special Case

Negative Zero

### OUTPUT

Final Answer (Binary)

1 00000000 000000000000000000000000

Final Answer (Hex)

0x80000000

EXPORT

## Case 12: Normal with Negative Zero Binary Input

Input: 0.0

Exponent: 0

### INPUT

Binary ▾

Binary

-0.0

Exponent (Base-2)

0

COMPUTE

### PROCESS

Binary Equivalent

0.0

Normalized Binary

0.0

Final Exponent

0

Sign Bit

1

E'

00000000

Significand Fractional Part

000000000000000000000000

Special Case

Negative Zero

### OUTPUT

Final Answer (Binary)

1 00000000 000000000000000000000000

Final Answer (Hex)

0x80000000

EXPORT

### Case 13: Denormalized with Positive Binary Input

Input: 1.1110

Exponent: -142

INPUT

Binary

Exponent (Base-2)

1.1110

-142

COMPUTE

PROCESS

Binary Equivalent

Normalized Binary

Final Exponent

1.1110

0.00000000000000011110

-126

Sign Bit

E

Significand Fractional Part

Special Case

0

00000000

00000000000000011110000

Denormalized

OUTPUT

Final Answer (Binary)

Final Answer (Hex)

0 00000000 00000000000000011110000

0x000000F0

EXPORT

### Case 14: Denormalized with Negative Binary Input

Input: -1.1110

Exponent: -130

### INPUT

Binary

Binary

Exponent (Base-2)

-1.1110

-130

COMPUTE

### PROCESS

Binary Equivalent

Normalized Binary

Final Exponent

1.1110

-0.0001111

-126

Sign Bit

E'

Significand Fractional Part

Special Case

1

00000000

000111100000000000000000

Denormalized

### OUTPUT

Final Answer (Binary)

Final Answer (Hex)

1 00000000 000111100000000000000000

0x800F0000

EXPORT

Test cases #13 and #14 shows that the program can accept both positive and negative binary as a valid input. The program then identifies the test case as a denormalized special case and provides the equivalent normalized binary value.

### Case 15: Denormalized with Positive Decimal Input

Input: **4.0**

Exponent: **142**

#### INPUT

Decimal

Decimal

4.0

Exponent (Base- 10)

-142

COMPUTE

#### PROCESS

Binary Equivalent

00000100.0

Normalized Binary

0.00000000000001000

Final Exponent

-126

Sign Bit

0

E'

00000000

Significand Fractional Part

0000000000001000000000

Special Case

Denormalized

#### OUTPUT

Final Answer (Binary)

0 00000000 00000000000001000000000

Final Answer (Hex)

0x00000200

EXPORT

### Case 16: Denormalized with Negative Decimal Input

Input: **-4.0**

Exponent: **-142**

#### INPUT

Decimal

Decimal

-4.0

Exponent (Base-10)

-142

COMPUTE

#### PROCESS

Binary Equivalent

00000100.0

Normalized Binary

-0.00000000000001000

Final Exponent

-126

Sign Bit

1

E'

00000000

Significand Fractional Part

0000000000001000000000

Special Case

Denormalized

#### OUTPUT

Final Answer (Binary)

1 00000000 00000000000001000000000

Final Answer (Hex)

0x80000200

EXPORT

Test cases #15 and #16 shows that the program can accept both positive and negative decimal as a valid input. The program then identifies the test case as a denormalized special case and provides the equivalent normalized binary value.

### Case 17: Infinity with Positive Binary Input (+ symbol excluded)

Input: 1.111

Exponent: 999

#### INPUT

Binary

Binary

1.111

Exponent (Base-2)

999

COMPUTE

#### PROCESS

Binary Equivalent

1.111

Normalized Binary

1.111

Final Exponent

999

Sign Bit

0

E'

11111111

Significand Fractional Part

000000000000000000000000

Special Case

Positive Infinity

#### OUTPUT

Final Answer (Binary)

0 11111111 000000000000000000000000

Final Answer (Hex)

0x7F800000

EXPORT

### Case 18: Infinity with Positive Binary Input (+ symbol included)

Input: +1.111

Exponent: 999

#### INPUT

Binary

+1.111

Exponent (Base-2)

999

COMPUTE

#### PROCESS

Binary Equivalent

1.111

Normalized Binary

1.111

Final Exponent

999

Sign Bit

0

E'

11111111

Significand Fractional Part

000000000000000000000000

Special Case

Positive Infinity

#### OUTPUT

Final Answer (Binary)

0 11111111 000000000000000000000000

Final Answer (Hex)

0x7F800000

EXPORT

Test case #17 and #18 shows that the program can accept a positive binary with and without a positive symbol as a valid input. The program then identifies the test case as an infinity special case and provides the equivalent final answer in binary and in hex.

## Case 19: Infinity with Negative Binary Input

Input: **-1.111**

Exponent: **999**

### INPUT

Binary

Binary

-1.111

Exponent (Base-2)

999

COMPUTE

### PROCESS

Binary Equivalent

1.111

Normalized Binary

-1.111

Final Exponent

999

Sign Bit

1

E'

11111111

Significand Fractional Part

000000000000000000000000

Special Case

Negative Infinity

### OUTPUT

Final Answer (Binary)

1 11111111 000000000000000000000000

Final Answer (Hex)

0xFF800000

EXPORT

This test case shows that the program can accept a negative binary as a valid input. The program then identifies the test case as an infinity special case and provides the equivalent final answer in binary and in hex.

The test cases for infinity special case will be having an output (for the process) of the following:

- Sign bit: 0 for positive infinity, 1 for negative infinity
- E': 1111 1111
- Significand: 000 0000 0000 0000 0000 0000



## Case 20: Signaling NaN input:

Input: **snan**

Exponent: **N/A**

INPUT				
NaN		NaN		COMPUTE
snan		Enter an exponent		
PROCESS				
Binary Equivalent	Normalized Binary		Final Exponent	
00000000	1.00000000		0	
Sign Bit	E'	Significand Fractional Part	Special Case	
x	11111111	010000000000000000000000	sNaN	
OUTPUT				
Final Answer (Binary)		Final Answer (Hex)		EXPORT
0 11111111 010000000000000000000000		0x7FA00000		

The following NaN test cases show that the program can accept a NaN as a valid input. The program then identifies the test case as a NaN special case and provides the equivalent final answer in binary and in hex.

The program accepts 2 inputs for NaN special cases being snan (signaling nan) and qnan (quiet nan) respectively.

The following test cases for NaN special case will be having an output (for the process) of the following:

- Sign bit: x (don't care)
- E': 1111 1111
- Significand: 01x xxxx xxxx xxxx xxxx xxxx (sNaN), 1xx xxxx xxxx xxxx xxxx xxxx (qNaN)

### Case 21: Quiet NaN input:

Input: **snan**

Exponent: **N/A**

#### INPUT

NaN

NaN

qnan

NaN

Enter an exponent

COMPUTE

#### PROCESS

Binary Equivalent

00000000

Normalized Binary

1.00000000

Final Exponent

0

Sign Bit

x

E'

11111111

Significand Fractional Part

100000000000000000000000

Special Case

qNaN

#### OUTPUT

Final Answer (Binary)

0 11111111 100000000000000000000000

Final Answer (Hex)

0x7FC00000

EXPORT

### Learnings and Conclusion:

In this simulation project, the group was able to take a look back into the process converting IEEE-754 binary floating point for single precision and implement this learning into a web-based converter through the use of basic html and css for the front end, and javascript for the backend.

In the process of creating the web-based converter, the group was able to disseminate the various normal and special cases among each other, sharpening each member's logic and implementation of their learnings from IEEE-754 binary floating point for single precision. Each normal case was handled independently through dedicated functions and logic such as the creation of utility/helper functions such as integerToBinary, fractionalToBinary, and convertDecimalToBinary to be used by other functions in the program. On the other hand, due to the nature of special cases being set and repetitive, the group was able to deduce that the best course of action in this program was to handle the special cases by brute force, having each input be first checked by each special case function. In the NaN special case in particular, the group made the program to only accept "snan" and "qnan" as inputs to show the program's ability to process the appropriate outputs as advised by the group's professor. In addition, the group was also able to create a debugger into the program to be used in the console through the function "trialQuickPrint" to be able to trace each algorithm and keep check with the output.

This program was able to encapsulate the processes and cases of IEEE-754 binary floating point for single precision conversion finishing with a feature to export each final output into a downloadable text file.