CSARCH2 **S12 Group 04**

Members

- Austria, Rafael Antonio
- Bongon, Janina Alexia
- Dy, Fatima Kriselle
- Estrera, David Joshua
- Jumilla, Sarah Ericka

Analysis Write-up

First problem encountered was handling negatives. We've noticed that if a negative value was inputted, it gets converted into a scientific notation. The way we tried to fix it was trying to parseFloat to hopefully catch the negative value conversion but it resulted in a NaN in coefficient continuation. We then used Number since it doesn't convert negative numbers to scientific notation unnecessarily:

```
convBtn.addEventListener('click', function () {
    let constant1 = Number(document.getElementById('constant').value); //use if positive
    let constant2 = constant1*-1 //use if negative
    let exp = document.getElementById('exp').value;
```

Figure 01. *Negative conversion*

In handling the binary output to hexadecimal conversion, one member of the group encountered problems with the conversion as the actual results did not match up with the expected results. Certain techniques were used to solve the problem such as printing the variables one by one onto the screen and by backtracking and changing the logic or algorithm of a function. The group originally used the Javascript technique of converting a binary number to its hexadecimal counterpart which is by using the following code…

```
parseInt(number, 2).toString(16);
```

The second parameter in the function parseInt indicates that the number to be converted is in binary by inputting the number 2 then the parameter of the function toString indicates that the binary number will be converted to hexadecimal. But when the group used this technique, results were still incorrect when using test cases. So one member of the group changed the contents of the binaryToHex function into a much more manual way of converting from binary to hexadecimal. Results were then almost correct with the exception of some of the first few hexadecimal numbers incorrectly rendering. By doing some backtracking, it was quickly realized that the combination field turned from five (5) to six (6) bits when an input was submitted due to some bug in the program. This problem was raised to the group and removing the extra bit in the combination field finally rendered the correct results.

```
// Function to convert binary to hexadecimal
function binaryToHex(binary) {
    const hexMap = {
        '0000': '0', '0001': '1', '0010': '2', '0011': '3',
        '0100': '4', '0101': '5', '0110': '6', '0111': '7',
        '1000': '8', '1001': '9', '1010': 'A', '1011': 'B',
        '1100': 'C', '1101': 'D', '1110': 'E', '1111': 'F'
    };

    let hex = '';
    for (let i = binary.length - 4; i >= 0; i -= 4) {
        let chunk = binary.slice(i, i + 4);
        hex = hexMap[chunk] + hex;
    }
    return hex;
}

```

Figure 02. *New binaryToHex() function*

Third problem encountered was the implementation of the rounding methods. When more than 34 decimal digits are entered or values like 1234.698626, the number must be rounded to normalized format based on the chosen rounding method of the user. The challenge faced by the members was ensuring the rounded decimal value is correct and its exponent counterpart is updated based on the number or decimal places moved. To address this, the group decided to count the number of digits, truncate the number to 35 digits then the rounding function is applied. For the exponent, once the number of digits are counted, 34 is subtracted to know how many will be added to the exponent.

```
function truncateAndExtract(number) {
  // Convert the number to a string
  const numStr = Math.abs(number).toString();

  // Extract the first 35 digits
  const truncatedStr = numStr.substring(0, 35);

  // Convert back to a number if needed
  // const truncatedNum = parseFloat(truncatedStr); // Uncomment this line if you want the result as a number

  return truncatedStr;
}
```

Figure 03. *Function to truncate and extract decimal more than 34 digits*

```
function roundTo34Digits(number, roundingMode) {
  // Convert the number to a string
  const numStr = Math.abs(number).toString();
  console.log("Rounding numner: "+numStr);
  // Truncate to 34 digits
  if (roundingMode === 'Truncate') {
    return numStr.substring(0, 34);
  }

  // Round up to 34 digits
  if (roundingMode === 'Round up') {
    return numStr.substring(0, 35);
  }

  // Round down to 34 digits
  if (roundingMode === 'Round down') {
    return numStr.substring(0, 34);
  }

  // Round to nearest (ties to even) to 34 digits
  if (roundingMode === 'Round to nearest (ties to even)') {
    // Extract the first 35 digits
    const truncatedStr = numStr.substring(0, 35);
    // Round to nearest (ties to even) by converting to number
    const roundedNum = Number(parseFloat(truncatedStr));
    // Convert back to string
    return roundedNum.toString();
  }

}
```

Figure 04. *Rounding function*

```
if(numdigits > 34){
    numdigits = numdigits - 34;
    exp += numdigits;
    constant1 = Number(truncateAndExtract(constant1)).toFixed(0);
    constant1 = Number(roundTo34Digits(constant1, roundingMethod)).toFixed(0);
}
```

Figure 05. *Algorithm for exponent updating after rounding*