Manisa Celal Bayar University – Department of Computer Engineering
CSE 3121 & 3239 Numerical Analysis for Computer Engineers – Midterm Exam

Name and Surname	
Student Id	
Signature	

Question	1	2	3	4	Total
Score					

Questions

Q1 (25 Points) Build a **binary** representation to store colored points in a 2D coordinate system. This representation should include 5 parts to hold the following information. Fill the table given below with the required **number of bits** to build such representation.

x-coordinate (Base-10 Signed Integer) : From -999 to 999
y-coordinate (Base-10 Signed Integer) : From -999 to 999
RGB Color Code (3 of Base-10 Unsigned Integers) : From (0,0,0) to (255,255,255)

Part	x	y	R	G	B	TOTAL
Number of Bits	11	11	8	8	8	46

Bits — Max. Value

$1 \to 0..1 \to 2$

$2 \to 0-3 \to 4$

$3 \to 0..7 \to 8$

$\vdots$

$n \to 0..2^{n-1} \to 2^n$

From -999 to 999

we have $(999 \cdot 2) + 1$ values

$\Rightarrow 1999$

999 negative values
1 zero
999 positive values

$2^n \geq 1999 \Rightarrow \log_2(2^n) \geq \log_2 1999$

$n \geq 10.965$

$n = 11$

For 0... 255

we have 256 values

$2^n \geq 256 \Rightarrow n \geq \log_2 256$

$n = 8$

Q2 (25 Points) We want to write a Python program which can generate a desired number of points in the type of that defined in Q1. Finally, this program calculates the average RGB color of these points. In terms of the CPU-time and Memory **performance**, write a `main` function by using **some** of the following predefined functions.

```
import numpy as np
import random

def function_A(n: int) -> list:
    points = []
    for i in range(n):
        points.append(
            (
                random.randint(0, 999),
                random.randint(0, 999),
                random.randint(0, 255),
                random.randint(0, 255),
                random.randint(0, 255),
            )
        )
    return points

def function_B(n: int) -> np.ndarray:
    points = np.array([], dtype=int).reshape(0, 5)
    for i in range(n):
        point = np.array(
            (
                random.randint(0, 999),
                random.randint(0, 999),
                random.randint(0, 255),
                random.randint(0, 255),
                random.randint(0, 255),
            ),
            dtype=int,
        )
        points = np.append(
            points,
            [point],
            axis=0,
        )
    return points

def function_C(n: int) -> np.ndarray:
    points = np.zeros((n, 5), dtype=int)
    for i in range(n):
        points[i] = (
            random.randint(0, 999),
            random.randint(0, 999),
            random.randint(0, 255),
            random.randint(0, 255),
            random.randint(0, 255),
        )
    return points

def function_D(points: list) -> np.ndarray:
    return np.array(points)

def function_E(points: np.ndarray) -> list:
    return points.tolist()

def function_F(points: list) -> tuple:
    r, g, b = 0, 0, 0
    for point in points:
        r += point[2]
        g += point[3]
        b += point[4]
    return (
        r // len(points),
        g // len(points),
        b // len(points)
    )

def function_G(points: np.ndarray) -> tuple:
    r, g, b = 0, 0, 0
    for point in points:
        r += point[2]
        g += point[3]
        b += point[4]
    return (
        r // len(points),
        g // len(points),
        b // len(points)
    )

def main(number_of_points: int) -> tuple:
    points = function_A(number_of_points)
    points = function_D(points)
    return function_G(points)

if __name__ == "__main__":
    main(100000000)
```

Q3 (25 Points) Find the **Single** Precision IEEE 754 representation of the number **-19.75** and fill the boxes given below with its **hexadecimal** digits.

Single Precision (1-bit Sign + 8-bit Exponent + 23-bit Mantissa) in Hex

C	1	9	E	0	0	0	0

Sign Bit: 1

Integer: 19

$19 \% 2 = 1$
$9 \% 2 = 1$
$4 \% 2 = 0$
$2 \% 2 = 0$
1

Fractional: 0.75

$0.75 \times 2 = 1.5 \rightarrow 1$
$0.5 \times 2 = 1.0 \rightarrow 1$

10011.11

Normalization

$1.001111 \cdot 2^4$

For 8 bit exponent, bias $2^{8-1} - 1 = 127$

$1.001111 \cdot 2^{4+127}$

$131 \% 2 = 1$
$65 \% 2 = 1$
$32 \% 2 = 0$
$16 \% 2 = 0$
$8 \% 2 = 0$
$4 \% 2 = 0$
$2 \% 2 = 0$
1

$1.001111 \cdot 2^{10000011}$

Exponent: 10000011

Mantissa: 001111 -------- padding

1100 | 0001 | 1001 | 1110 | 0000 | 0000 | 0000 | 0000

12 → C, 1 → 1, 9 → 9, 14 → E, 0 → 0, 0 → 0, 0 → 0, 0 → 0

Q4 (25 Points) Build a **custom** IEEE 754 representation precision with 1 sign bit, 3 exponent bits, and 4 mantissa bits. If we want to represent **7.1** with precision, calculate the **error**, which is the absolute difference between the real value and the stored value of this number.

Custom Precision (1-bit Sign + 3-bit Exponent + 4-bit Mantissa) in Binary

0	1	0	1	1	1	0	1

Error

0.15

Integer Part : 7

$7 \% 2 = 1$

$3 \% 2 = 1$

1

$(7)_{10} = (111)_2$

Fractional Part : 0.1

$0.1 \times 2 = 0.2 \rightarrow 0$

$0.2 \times 2 = 0.4 \rightarrow 0$

$0.4 \times 2 = 0.8 \rightarrow 0$

$0.8 \times 2 = 1.6 \rightarrow 1$

$0.6 \times 2 = 1.2 \rightarrow 1$

$0.2 \times 2 = 0.4 \rightarrow 0$

$0.4 \times 2 = 0.8 \rightarrow 0$

$0.8 \times 2 = 1.6 \rightarrow 1$

$0.6 \times 2 = 1.2 \rightarrow 1$

$\vdots \quad \vdots$

Same Patterns

$(0.1)_{10} = (00011001100011\ldots)_2$

The number is $111.000110011\overline{0011}\ldots$

Normalization $1.11000110011\overline{0011} \cdot 2^2$

Bias for 3 bits is $2^{3-1} - 1 = 3$

The final number is $1.1100011001\overline{10011} \cdot 2^{2+3}$

$5 = (101)_2$

Sign : 0, Exponent : 101, Mantissa : 1101 → with round-up

Converted Number is $1.1101 \cdot 2^2 = \underbrace{111}_{7}.\underbrace{01}_{25}$

7.25

Error is $|7.1 - 7.25| = 0.15$

	QUESTIONS VS PÇB MATRIX																											
	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28
Q1	✓																											
Q2	✓						✓																					
Q3	✓	✓																										
Q4	✓	✓					✓																					