Final Day Core Programming Solution

Q1)

```c
// Check Leap Year
#include <stdio.h>

int main()
{
    system("cls");
    int year;
    printf("Enter a year: ");
    scanf("%d", &year);

    // Leap year condition
    if ((year % 4 == 0 && year % 100 != 0) || (year % 400 == 0))
        printf("True");
    else
        printf("False");

    return 0;
}
```

Test Case : Input : 2024 Output : True

Input : 100 Output : False

Input : 2100 Output : False

Q2)

```c
// Q1.Write a program to check if a string containing parentheses is balanced.

#include <stdio.h>
#include <string.h>

int main()
{
    system("cls");

    char parenthesis[100];
    printf("Enter a string of parentheses: ");
    scanf("%s", parenthesis);

    int length = strlen(parenthesis);
    int count = 0;

    printf("Parenthesis = %s, Length = %d\n", parenthesis, length);

    for (int i = 0; i < length; i++)
    {
        if (parenthesis[i] == '(')
```

```
        {
            count++;
        }
        else if (parenthesis[i] == ')')
        {
            count--;
            if (count < 0)
            {
                printf("\nNot balanced\n");
                return 0;
            }
        }
    }

    if (count == 0)
        printf("\nBalanced\n");

    else
        printf("\nNot balanced\n");

    return 0;
}
/*
Eg: checks :
Balanced :
()
(())
(()())
((()))
()()()
((()()))

Unbalanced :
(
)
(()
())
)()( (incorrect order)
((()) (missing closing parenthesis)
())(() (mismatched pairs)
*/
```

Q3)

```
Count Set Bits in an Integer
Write a program to count the number of 1s in the binary representation of a given integer.

#include <stdio.h>

int countSetBits(int num)
{
    int count = 0;

    while (num > 0)
```

```c
    {
        if (num % 2 == 1)
            count++;

        num = num / 2;
    }

    return count;
}

int main()
{
    int num;

    printf("Enter an integer: ");
    scanf("%d", &num);

    int setBits = countSetBits(num);

    printf("Number of 1s in binary representation of %d: %d\n", num, setBits);

    return 0;
}

// Replace the code inside while loop with these to test

/*      Better => works faster because works on hardware level
        count += num & 1;  // Check if the least significant bit is 1
        num >>= 1;         // Right shift to process the next bit
*/


/*
     Best => Brian Kernighan's Algorithm
    num &= (num - 1);  // Clear the least significant 1 bit
    count++;
*/

Example:
Input: 5 → Binary: 101 → Output: 2
Input: 10 → Binary: 1010 → Output: 2
Input: 500 → Binary: 111110100 → Output: 6
```

Q4)

```c
/*
Spiral Matrix
Given a 2D matrix, return its elements in spiral order.
*/
#include <stdio.h>

void spiralOrder(int matrix[][100], int rows, int cols)
{
    int top = 0, bottom = rows - 1, left = 0, right = cols - 1;
    int i;
```

```
    printf("Spiral Order: ");
    while (top <= bottom && left <= right)
    {
        // Traverse from left to right
        for (i = left; i <= right; i++)
        {
            printf("%d ", matrix[top][i]);
        }
        top++; // Move top boundary down

        // Traverse from top to bottom
        for (i = top; i <= bottom; i++)
        {
            printf("%d ", matrix[i][right]);
        }
        right--; // Move right boundary left

        // Traverse from right to left
        if (top <= bottom)
        {
            for (i = right; i >= left; i--)
            {
                printf("%d ", matrix[bottom][i]);
            }
            bottom--; // Move bottom boundary up
        }

        // Traverse from bottom to top
        if (left <= right)
        {
            for (i = bottom; i >= top; i--)
            {
                printf("%d ", matrix[i][left]);
            }
            left++; // Move left boundary right
        }
    }
    printf("\n");
}

int main()
{
    int matrix[100][100], rows, cols;

    printf("Enter the number of rows and columns: ");
    scanf("%d %d", &rows, &cols);

    printf("Enter the elements of the matrix:\n");
    for (int i = 0; i < rows; i++)
    {
        for (int j = 0; j < cols; j++)
        {
            scanf("%d", &matrix[i][j]);
        }
```

```
    }

    spiralOrder(matrix, rows, cols);

    return 0;
}
```

Test Cases : For black return black

For Row 0 or Column 0 , return the same input.

Input : 1 2 Output : 1 2 4 3

3 4

Input : 1 2 3 Output : 1 , 2, 3, 6, 9, 8, 7, 4, 5

4 5 6

7 8 9

Q5)

```
/*
String Compression
Implement a function that compresses a string by replacing consecutive repeated characters with
the character followed by the count. If the compressed string is not smaller than the original,
return the original.

Example:
Input: "aabcccccaaa" → Output: "a2b1c5a3"
*/
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

// Function to compress a string
char *compressString(const char *str)
{
    int len = strlen(str);
    char *compressed = (char *)malloc(2 * len + 1);
    int compressedIndex = 0;

    for (int i = 0; i < len;)
    {
        char currentChar = str[i];
        int count = 0;

        // Count consecutive occurrences of currentChar
        while (i < len && str[i] == currentChar)
        {
            count++;
            i++;
        }
```

```
        // Append the character and its count to the compressed string
        compressed[compressedIndex++] = currentChar;
        compressedIndex += sprintf(&compressed[compressedIndex], "%d", count);
    }

    compressed[compressedIndex] = '\0';
    return compressed;
}

int main()
{
    char str[100];
    printf("Enter a string: ");
    scanf("%s", str);

    char *compressed = compressString(str);
    printf("Compressed string: %s\n", compressed);

    free(compressed); // Free the allocated memory
    return 0;
}
```

Test Case : Input aaaaaabccccddddddd Output : a6b1c4d6

Q6)

```
/*
 Rotate Matrix by 90 Degrees
Problem Statement:
Given an n x n matrix, rotate it 90 degrees clockwise in place.
In place means modifying the matrix in-place (without using extra space for another matrix) and
rotating 90 degrees can be done by first transposing then reversing each row
*/
#include <stdio.h>

void rotateMatrix(int matrix[][3], int n)
{
    for (int i = 0; i < n; i++)
    {
        for (int j = i + 1; j < n; j++)
        {
            int temp = matrix[i][j];
            matrix[i][j] = matrix[j][i];
            matrix[j][i] = temp;
        }
    }

    for (int i = 0; i < n; i++)
    {
        for (int j = 0; j < n / 2; j++)
        {
            int temp = matrix[i][j];
            matrix[i][j] = matrix[i][n - j - 1];
            matrix[i][n - j - 1] = temp;
        }
```

```
    }
}

void printMatrix(int matrix[][3], int n)
{
    for (int i = 0; i < n; i++)
    {
        for (int j = 0; j < n; j++)
        {
            printf("%d ", matrix[i][j]);
        }
        printf("\n");
    }
}

int main()
{
    int matrix[3][3] = {
        {1, 2, 3},
        {4, 5, 6},
        {7, 8, 9}};

    printf("Original Matrix:\n");
    printMatrix(matrix, 3);

    rotateMatrix(matrix, 3);

    printf("\nRotated Matrix:\n");
    printMatrix(matrix, 3);

    return 0;
}
```

Test Cases : Input : 1 2 3 4

Output : 1

2

3

4

Input : 1 2 Output : 3 1

3 4 4 2

Q7)

```
/*
Remove Duplicates from Sorted Array
Write a function to remove duplicates from a sorted array in-place.

Example:
Input: [0, 0, 1, 1, 2, 3, 3, 4] → Output: [0, 1, 2, 3, 4]
```

```c
*/

#include <stdio.h>

int removeDuplicates(int arr[], int n)
{
    if (n == 0)
        return 0;

    int uniqueIndex = 1;

    for (int i = 1; i < n; i++)
    {
        if (arr[i] != arr[i - 1])
        {
            arr[uniqueIndex] = arr[i];
            uniqueIndex++;
        }
    }

    return uniqueIndex;
}

int main()
{
    int arr[] = {1, 1, 2, 2, 3, 3, 4};
    int n = sizeof(arr) / sizeof(arr[0]);

    int newLength = removeDuplicates(arr, n);

    printf("Array after removing duplicates: ");
    for (int i = 0; i < newLength; i++)
    {
        printf("%d ", arr[i]);
    }

    printf("\nNew length: %d\n", newLength);

    return 0;
}
```

Test Case : Input : 1 2 2 2 2 3 Output : 1 2 3

Input : 1 1 1 1 1 1 1 1 Output : 1

Input : 9 9 10 10 11 Ouput : 9 10 11

Q8)

```c
/*
Given a positive integer n, find the minimum number of operations required to reduce n to 1. You
can perform three operations :
```

```c
    Subtract 1 from n.
    If n is divisible by 2,divide n by 2.
    If n is divisible by 3, divide n by 3.
*/
#include <stdio.h>
#include <limits.h>

int minOperations(int n)
{
    int dp[n + 1]; // Array to store the minimum operations for each number
    dp[1] = 0;     // Base case: It takes 0 operations to reduce 1 to 1

    for (int i = 2; i <= n; i++)
    {
        // Start with the operation of subtracting 1
        int min_ops = dp[i - 1];

        // If divisible by 2, consider the divide by 2 operation
        if (i % 2 == 0)
        {
            min_ops = (min_ops < dp[i / 2]) ? min_ops : dp[i / 2];
        }

        // If divisible by 3, consider the divide by 3 operation
        if (i % 3 == 0)
        {
            min_ops = (min_ops < dp[i / 3]) ? min_ops : dp[i / 3];
        }

        // Store the result for dp[i]
        dp[i] = 1 + min_ops;
    }

    return dp[n];
}

int main()
{
    int n;

    printf("Enter a positive integer n: ");
    scanf("%d", &n);

    printf("Minimum number of operations to reduce %d to 1: %d\n", n, minOperations(n));

    return 0;
}
```

Test Case :

Input : 10 Output : 3

Input ; 95 Output : 8

Q9)

```c
/*
 Next Greater Element
Problem Statement:
Given an array, find the next greater element for each element. If no greater element exists,
output -1.
Eg: for a given array 3,5,2,6,9 output will be 5,6,6,9,-1

*/
#include <stdio.h>
#include <stdlib.h>

void nextGreaterElement(int arr[], int n)
{
    int *result = (int *)malloc(n * sizeof(int)); // Array to store results
    int stack[n], top = -1;                       // Stack to track next greater elements

    for (int i = n - 1; i >= 0; i--)
    {
        // Pop elements from stack while they're <= current element
        while (top >= 0 && stack[top] <= arr[i])
        {
            top--;
        }

        // If stack is empty, no next greater element
        if (top == -1)
        {
            result[i] = -1;
        }
        else
        {
            result[i] = stack[top]; // Top of stack is the next greater element
        }

        // Push current element to stack
        stack[++top] = arr[i];
    }

    // Print the result
    for (int i = 0; i < n; i++)
    {
        printf("%d -> %d\n", arr[i], result[i]);
    }

    free(result);
}

int main()
{
    int arr[] = {3, 5, 2, 6, 9};
    int n = sizeof(arr) / sizeof(arr[0]);
```

```
    nextGreaterElement(arr, n);

    return 0;
}
```

Test Cases : Input *3,5,2,6,9 output will be 5,6,6,9,-1*

```
Input : {3, 5, 2, 6, 9}   Output : 5 6 6 9 -1
```

Q10)

```
/*
Majority Element
Problem Statement:
Find the element that appears more than n/2 times in an array. Assume such an element always
exists.
Eg: In a given array 1,1,2,1,2 return 1
*/
#include <stdio.h>

int findMajorityElement(int arr[], int n)
{
    int candidate = arr[0], count = 1;

    for (int i = 1; i < n; i++)
    {
        if (arr[i] == candidate)
        {
            count++;
        }
        else
        {
            count--;
            if (count == 0)
            {
                candidate = arr[i];
                count = 1;
            }
        }
    }

    // Phase 2: Return the candidate (guaranteed to be valid as per problem)
    return candidate;
}

int main()
{
    int arr[] = {2, 2, 1, 1, 1, 2, 2, 3, 4, 5, 1, 1, 1};
    int n = sizeof(arr) / sizeof(arr[0]);

    int majorityElement = findMajorityElement(arr, n);
    printf("Majority Element: %d\n", majorityElement);

    return 0;
}
```

```
Test Cases : Input : 5, 5, 5, 6, 5, 7, 1 , 1, 5    Output : 5
             Inpur : 3, 6 ,9 ,3 ,3 ,3 , 5           Output : 3
```

Q11)

```c
/*
Find Duplicates in an Array
Problem Statement:
Given an array of n integers where each integer is in the range 1 to n, some elements appear more
than once. Find all duplicates in the array.
*/

#include <stdio.h>
#include <stdlib.h>

void findDuplicates(int arr[], int n)
{
    printf("Duplicate elements: ");
    for (int i = 0; i < n; i++)
    {
        int index = abs(arr[i]) - 1;

        // Check if the value at that index is already negative
        if (arr[index] < 0)
            printf("%d ", abs(arr[i])); // It's a duplicate

        else
            arr[index] = -arr[index]; // Mark as visited
    }
}

int main()
{
    system("cls");
    int arr[] = {1, 4, 5, 5, 3, 2, 7, 8, 2, 3, 1};
    int n = sizeof(arr) / sizeof(arr[0]);

    findDuplicates(arr, n);
    return 0;
}
```

```
Test Cases : Input : {1, 4, 5, 5, 3, 2, 7, 8, 2, 3, 1}    Output : 5 2 3 1
```

Q12)

```c
/*
Convert Roman to Integer
Write a function to convert a Roman numeral string into an integer.

Example:
Input: "MCMXCIV" → Output: 1994
*/
#include <stdio.h>
#include <ctype.h>
#include <string.h>
```

```c
// Function to get the integer value of a Roman numeral
int romanToValue(char c)
{
    switch (c)
    {
    case 'I':
        return 1;
    case 'V':
        return 5;
    case 'X':
        return 10;
    case 'L':
        return 50;
    case 'C':
        return 100;
    case 'D':
        return 500;
    case 'M':
        return 1000;
    default:
        return 0;
    }
}

// Function to convert Roman numeral to integer
int romanToInt(char *s)
{
    int total = 0;
    int i = 0;

    while (s[i] != '\0')
    {
        int current = romanToValue(s[i]);
        int next = romanToValue(s[i + 1]);

        // Subtract if current value is smaller than next value
        if (current < next)
        {
            total += (next - current);
            i += 2; // Skip the next character
        }
        else
        {
            total += current;
            i++;
        }
    }

    return total;
}

int main()
{
    char roman[20];
    printf("Enter a Roman numeral: ");
```

```
    scanf("%s", roman);

    for (int i = 0; roman[i] != '\0'; i++)
        roman[i] = toupper(roman[i]);

    int result = romanToInt(roman);
    printf("Integer value: %d\n", result);

    return 0;
}

Test Cases :
Input : xxix Ouput : 29
Input : cx   Output : 110
```

Q13)

```
/*
Count Occurrences
Write a program to count the occurrences of a specific element in an array.
*/
#include <stdio.h>
int main()
{
    system("cls");
    int arr[] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 1, 1, 1, 1, 1, 1, 1, 1, 1};
    int n = sizeof(arr) / sizeof(arr[0]);
    int count = 0;
    int element = 1;
    for (int i = 0; i < n; i++)
    {
        if (arr[i] == element)
        {
            count++;
        }
    }
    printf("The element %d occurs %d times in the array.", element, count);
    return 0;
}
Test Case : Input : {1,2,3,4,5,2,2,3}  n = 2 Output : 3
Input : {2,3,1,4,5,1,2,1,1} n = 1 Output : 4
```

Q14)

```
/*
Check for Perfect Square
Write a pseudo-code to check if a given number is a perfect square.
*/
#include <stdio.h>
#include <math.h>
int main()
{
    system("cls");
    int num;
    printf("Enter a number: ");
    scanf("%d", &num);

    int root = sqrt(num);
```

```
    if (root * root == num)
        printf("%d is a perfect square.\n", num);

    else
        printf("%d is not a perfect square.\n", num);

    return 0;
}
Test Case : Input : 1      Output : True
Input : 100    Output : True
Input : 64     Output : false
```

Q15)

```
//  Kadane's Algorithm to find maximum sum of subarray
// For a given array find out the maximum sum a subarray can have
// Hard problem
#include <stdio.h>

int maxSubArraySum(int arr[], int n)
{
    int max_so_far = arr[0];      // Initialize max sum encountered so far
    int max_ending_here = arr[0]; // Initialize current subarray sum

    for (int i = 1; i < n; i++)
    {
        // Update the maximum subarray ending at index i
        max_ending_here = (arr[i] > max_ending_here + arr[i]) ? arr[i] : max_ending_here + arr[i];

        max_so_far = (max_so_far > max_ending_here) ? max_so_far : max_ending_here;
    }

    return max_so_far;
}

int main()
{
    int arr[] = {-2, 1, -3, -4, -1, 2, 1, -5, -4, 5};
    int n = sizeof(arr) / sizeof(arr[0]);

    int max_sum = maxSubArraySum(arr, n);

    printf("Maximum subarray sum is %d\n", max_sum);

    return 0;
}
Test Case : Input : {-2, 1, -3, -4, -1, 2, 1, -5, -4, 5}    Out[ut : 5
```

Q16)

```
/*
Subarray with Given Sum
Problem Statement:
Given an array of non-negative integers, find a contiguous subarray that sums to a given target S.
*/
```

```c
#include <stdio.h>

void findSubarrayWithSum(int arr[], int n, int S)
{
    int start = 0, current_sum = 0;

    for (int end = 0; end < n; end++)
    {
        current_sum += arr[end];

        // Shrink the window as long as current_sum > target_sum
        while (current_sum > S && start < end)
        {
            current_sum -= arr[start++];
        }

        if (current_sum == S)
        {
            printf("Found subarray from %d to %d\n", start, end);
            return;
        }
    }

    printf("No subarray with sum %d found\n", S);
}

int main()
{
    int arr[] = {11, 1, 2, 2, 3, 7, 5};
    int n = sizeof(arr) / sizeof(arr[0]);
    int S = 12;

    findSubarrayWithSum(arr, n, S);

    return 0;
}
```

Test Case : Input : {11, 1, 2, 2, 3, 7, 5} S = 12 Output : [11, 1]