INDIAN INSTITUTE OF TECHNOLOGY ROORKEE
DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

CSC-101 PROGRAMMING WITH C AND C++
Autumn 2023

Total Marks - 30 | Mid-Term Exam (solutions) | Time - 90 mins.

Note: *It is imperative to provide sufficiently detailed arguments alongside your solutions. Answers without supporting justifications will not receive credit.*

1. (a) Consider the following code snippet:

```c
#include<stdio.h>
int main()
{
    char str[] = "PLAYMATH";
    int len = sizeof(str) - 1;

    for (int i = 0; i < len; i+=2) {
        printf("%c", str[i]);
    }
    return 0;
}
```

Now, suppose the printed characters form a new string S. What will be the character at the position $\left\lceil \frac{|S|}{2} \right\rceil$ in this string? Here $|S|$ represents the length of the string S and $\lceil . \rceil$ is the ceiling function.

[2 Marks]

Sol. From the code, we can see that it's printing every second character of the string "PLAYMATH" starting from index 0. The loop increments by 2, so it prints characters at positions 0, 2, 4, 6, ...

The string "PLAYMATH" has characters at these positions:

P (0) - L (1) - A (2) - Y (3) - M (4) - A (5) - T (6) - H (7)

From the above positions, the loop extracts the characters P, A, M, and T.

So, the new string S formed is "PAMT".

From the above step, we know that $S =$ "PAMT". So, $|S| = 4$.

Finding the Character at $\left\lceil \frac{|S|}{2} \right\rceil$:

Given $|S| = 4$, we need to find the character at position:

$$\left\lceil \frac{4}{2} \right\rceil = 2$$

String indexing starts from 0. So, in the string "PAMT", the character at position 2 (0-indexed) is 'M'.

Answer: The character at the position $\left\lceil \frac{|S|}{2} \right\rceil$ in the string S is 'M'.

(b) Given the following code snippet:

```c
#include<stdio.h>

int main() {
    int a = 3;
    int b = 2;
    int c = a+++b;
    printf("%d %d %d\n", a, b, c);
    return 0;
}
```

Let's consider the printed values of a, b, and c as x, y, and z respectively. Now, using these values, construct a polynomial $P(n)$ of degree 2 as $P(n) = n(x^2 - y) + yz - (x^2 + z)n^2$. If the sum of roots of this polynomial can be represented as α/β, where α and β are co-prime, then compute the value of $\alpha + \beta$.

[3 Marks]

Sol. Consider the line 'int c = a+++b;'. This line is using two separate operators:

- Post-increment on 'a' (i.e., 'a++').
- Addition with 'b'.

So, 'c' is assigned the value of 'a' before it is incremented, plus 'b'. After the operation, 'a' is incremented. Given: a = 3; b = 2;

When we execute 'c = a+++b;', it is interpreted as: c = a++ + b;

Flow of information:

1. 'c' gets the current value of 'a' (3) + 'b'.
2. 'a' is then incremented.

Let's now compute the values:

- Initial values: 'a = 3', 'b = 2'.
- After the operation: 'a = 4', 'c = 3 + 2 = 5'.

So, the output of the code will be: '4 2 5'.

This means: $x = 4$, $y = 2$, $z = 5$

Now, let's plug these values into the polynomial:

$$\begin{aligned} P(n) &= n(x^2 - y) + yz - (x^2 + z)n^2 \\ &= n(4^2 - 2) + 2 \times 5 - (4^2 + 5)n^2 \\ &= 14n + 10 - 21n^2 \\ &= -21n^2 + 14n + 10 \end{aligned}$$

This is a quadratic equation of the form:

$$-21n^2 + 14n + 10 = 0$$

Using the formula for the sum of the roots of a quadratic equation $ax^2 + bx + c = 0$ is given by:

$$\text{Sum} = -\frac{b}{a}$$

Using this, we find the sum of the roots to be:

$$\text{Sum} = -\frac{14}{-21}$$

$$= \frac{2}{3}$$

Where $\alpha = 2$ and $\beta = 3$.
So, $\alpha + \beta = 2 + 3 = 5$.
Hence, the value of $\alpha + \beta$ is 5.

2. In a modern-day security application, the significance of a robust passcode cannot be overemphasized. You are tasked with designing a part of such a system, focusing on the validation mechanism.

   SPECIFICATIONS:

   - The passcode will always be a sequence of four digits.

   - Your program should receive the passcode as an input from the user.

   CRITERIA FOR A VALID PASSCODE:

   1. Every digit in the passcode should be even (only 0, 2, 4, 6, or 8 are acceptable).

   2. The sum of the digits constituting the passcode must be divisible by 4.

   3. No two consecutive digits in the passcode should be identical. For instance, 1224 would be deemed invalid due to the consecutive 2's.

   OUTPUT:

   - If the passcode adheres to the aforementioned criteria, your program should display: "Success! The passcode is valid."

   - If the passcode fails any of the criteria, the program should display: "Error! The passcode is invalid."

   [5 Marks]

   Sol.

```c
#include <stdio.h>

int main() {
    int passcode, digit, prev_digit = -1, sum = 0, even_flag = 1,
    consecutive_flag = 1;

    // Prompt the user for input
    printf("Enter the four-digit passcode: ");
    scanf("%d", &passcode);

    for (int i = 0; i < 4; i++) {
        digit = passcode % 10; // Extract the last digit

```

```
        // Check for evenness
        if (digit % 2 != 0) {
            even_flag = 0;
        }

        // Check for consecutive digits
        if (digit == prev_digit) {
            consecutive_flag = 0;
        }

        sum += digit; // Sum up the digits
        prev_digit = digit; // Store the current digit for the next iteration
        passcode /= 10; // Remove the last digit
    }

    // Check all conditions
    if (even_flag && sum % 4 == 0 && consecutive_flag) {
        printf("Success! The passcode is valid.\n");
    } else {
        printf("Error! The passcode is invalid.\n");
    }

    return 0;
}
```

3. You're given an array of characters, 'arr[]', containing only lowercase letters. You must determine if it's possible to rearrange the characters in this array such that no two adjacent characters are the same. If it's possible, print the rearranged array. If not, print "Not Possible".

   INPUT:

   Your program should first take a single integer, 'n', representing the number of characters in the array. The next line should contain 'n' lowercase letters (the elements of the array) separated by spaces.

   OUTPUT:

   Your program should print the rearranged characters where no two adjacent characters are the same, or "Not Possible" if no such rearrangement exists.

   Example 1:

   INPUT:

   3

   a a b

   OUTPUT:

   a b a

Example 2:

INPUT:

4

a a a b

OUTPUT:

Not Possible

[5 Marks]

Sol.

```c
#include <stdio.h>

int main() {
    int n;
    printf("Enter the number of characters: ");
    scanf("%d", &n);

    char arr[n], result[n + 1];  // +1 for null termination
    int freq[26] = {0};
    int max_freq = 0;
    char max_char;

    printf("Enter the characters separated by spaces: ");
    for (int i = 0; i < n; i++) {
        scanf(" %c", &arr[i]);  // Notice space before %c to ignore white spaces

        freq[arr[i] - 'a']++;
        if (freq[arr[i] - 'a'] > max_freq) {
            max_freq = freq[arr[i] - 'a'];
            max_char = arr[i];
        }
    }

    // If any character frequency is more than (n+1)/2 then it's impossible
    if (max_freq > (n + 1) / 2) {
        printf("Output: Not Possible\n");
        return 0;
    }

    // Place the most frequent character in even positions first
    int pos = 0;
    while (freq[max_char - 'a'] > 0) {
        result[pos] = max_char;
        pos += 2;
        freq[max_char - 'a']--;
    }
```

```
37
38      // Place the rest of the characters
39      for (int i = 0; i < 26; i++) {
40          while (freq[i] > 0) {
41              if (pos >= n) {
42                  pos = 1;  // Reset to the first position if end is
    reached
43              }
44              result[pos] = i + 'a';
45              pos += 2;
46              freq[i]--;
47          }
48      }
49
50      result[n] = '\0';  // Null terminate the result array for printing
51      printf("Output: ");
52      for (int i = 0; i < n; i++) {
53          printf("%c ", result[i]);
54      }
55      printf("\n");
56
57      return 0;
58  }
```

4. (a) Write a program in C that reads an integer n from the user (where $1 \leq n \leq 100$) and prints a right triangle pattern of numbers, with the height of the triangle being n. The pattern should be such that the first row contains a single '1', the second row contains two numbers: '1 2', the third row contains '1 2 3', and so on, up to n.

   INPUT: - An integer n ($1 \leq n \leq 100$) - representing the height of the triangle.

   OUTPUT: - A right triangle pattern of numbers.

   EXAMPLE:

   INPUT:

   5

   OUTPUT:

   1
   1 2
   1 2 3
   1 2 3 4
   1 2 3 4 5

   [2 Marks]

   Sol.

```
1  #include<stdio.h>
2
3  int main() {
```

```c
    int n;

    // Input the height of the triangle
    printf("Enter the height of the triangle (between 1 and 100): ");
    scanf("%d", &n);

    // Check if the input is within the valid range
    if (n < 1 || n > 100) {
        printf("Please enter a value between 1 and 100.\n");
        return 1;
    }

    // Generate and print the pattern
    for (int i = 1; i <= n; i++) {
        for (int j = 1; j <= i; j++) {
            printf("%d ", j);
        }
        printf("\n"); // Move to the next line after printing each row
    }

    return 0;
}
```

(b) Write a program in C that generates and prints a sequence based on an adapted form of the Fibonacci sequence. The first few terms of this sequence are: 0, 1, 2, 5, 12, 29, 70, 169, 408, 985, ... Your task is to identify the pattern and print the first n terms of the sequence, where n will be provided as input.

INPUT:

A single positive integer n ($1 \leq n \leq 50$).

OUTPUT:

The first n terms of the sequence, separated by spaces.

EXAMPLE:

INPUT:

```
10
```

OUTPUT:

```
0 1 2 5 12 29 70 169 408 985
```

[3 Marks]

Sol. The adapted Fibonacci sequence is the recurrence relation:

$$a_n = 2a_{n-1} + a_{n-2}$$

where $a_0 = 0$ and $a_1 = 1$.

```c

#include<stdio.h>

int main() {
    int n;

    // Input the number of terms
    printf("Enter the number of terms (between 1 and 50): ");
    scanf("%d", &n);

    // Check if the input is within the valid range
    if (n < 1 || n > 50) {
        printf("Please enter a value between 1 and 50.\n");
        return 1;
    }

    // Initialize the first two terms
    long long int a = 0, b = 1;

    // Generate and print the sequence
    for (int i = 1; i <= n; i++) {
        printf("%lld ", a); // Print current term

        long long int next = 2*b + a; // Calculate next term based on the relation
        a = b; // Move to the next term
        b = next;
    }
    printf("\n"); // Move to the next line after printing the sequence

    return 0;
}
```

5. (a) Imagine you are handed a mysterious array filled with ancient numerical inscriptions. Archaeologists believe that the key to unlocking the secrets of this array lies in understanding the relationships between the numbers within. Specifically, they are interested in the sum of the absolute differences between all pairs in the array.

   YOUR TASK:

   Given an unsorted array of finite size with distinct elements, your task is to compute the sum of the absolute differences between all unique pairs of elements in the array.

   INSTRUCTIONS:

   Prompt the user to input the size of the array ($1 \leq \text{size} \leq 100$). Then, take the distinct elements of the array as input. Calculate the sum of the absolute differences of all unique pairs. Print the resulting sum.

[**2 Marks**]

Sol.

```c
#include <stdio.h>

int main() {
    int n; // size of the array
    int sum = 0; // variable to store the sum of absolute
   differences

    // Read the size of the array
    printf("Enter the size of the array: ");
    scanf("%d", &n);

    int arr[n]; // declaring the array

    // Read the array elements
    printf("Enter the array elements:\n");
    for (int i = 0; i < n; i++) {
        scanf("%d", &arr[i]);
    }

    // Compute the sum of absolute differences
    for (int i = 0; i < n; i++) {
        for (int j = i + 1; j < n; j++) { // start from i+1 to avoid
     duplicate pairs and self pairs
            sum += abs(arr[i] - arr[j]);
        }
    }

    // Print the result
    printf("The sum of absolute differences is: %d\n", sum);

    return 0;
}
```

(b) You are provided with an array of positive integers $A = [A_1, A_2, ..., A_N]$ with a length of N. Your task is to create a new array B of the same length (N), where each value B_i in the new array is defined as:

$$B_i = \sum_{j=1, j \neq i}^{N} A_j$$

In simpler words, each value in array B should be the sum of all the numbers in the original array A excluding the number at that specific index.

INSTRUCTIONS:

1. Begin by reading the value of N and then read the array A.
2. Output the transformed array B.

Write the complete C program that accomplishes this task.

[3 Marks]

Sol.

```c
#include<stdio.h>

int main() {
    int N;
    printf("Enter the number of elements in the array: ");
    scanf("%d", &N);

    int A[N], B[N];
    int totalSum = 0;

    // Read the array and compute the total sum
    printf("Enter the elements of the array:\n");
    for(int i = 0; i < N; i++) {
        scanf("%d", &A[i]);
        totalSum += A[i];
    }

    // Compute the B array
    for(int i = 0; i < N; i++) {
        B[i] = totalSum - A[i];
    }

    // Print the B array
    printf("Transformed array is:\n");
    for(int i = 0; i < N; i++) {
        printf("%d ", B[i]);
    }

    return 0;
}
```

6. Given a 2D matrix of size $M \times N$ (where M and N are the number of rows and columns, respectively), display the boundary elements in a spiral order starting from the top-left corner and following clockwise direction.

   INPUT: In first line, two integers M and N ($2 \leq M, N \leq 100$) - the number of rows and columns of the matrix. In second line, an $M \times N$ matrix containing integers between -1000 and 1000.

   OUTPUT:

   A sequence of integers representing the boundary elements in the spiral order.

   EXAMPLE:

   INPUT:

4 4

1 2 3 4

5 6 7 8

9 10 11 12

13 14 15 16

OUTPUT:

1 2 3 4 8 12 16 15 14 13 9 5 6 7 11 10

[5 Marks]

Sol.

```c
#include <stdio.h>

int main() {
    int M, N;
    printf("Enter the number of rows and columns separated by space: ");
    scanf("%d %d", &M, &N);

    int matrix[M][N];
    printf("Enter the matrix elements row-wise:\n");
    for(int i=0; i<M; i++) {
        for(int j=0; j<N; j++) {
            scanf("%d", &matrix[i][j]);
        }
    }

    int i, rowStart = 0, colStart = 0;
    int rowEnd = M, colEnd = N;

    // Traverse the boundary in spiral order
    while(rowStart < rowEnd && colStart < colEnd) {
        // Print the first row
        for(i = colStart; i < colEnd; i++) {
            printf("%d ", matrix[rowStart][i]);
        }
        rowStart++;

        // Print the last column
        for(i = rowStart; i < rowEnd; i++) {
            printf("%d ", matrix[i][colEnd-1]);
        }
        colEnd--;

        // Print the last row if it hasn't been printed
        if(rowStart < rowEnd) {
            for(i = colEnd-1; i >= colStart; i--) {
                printf("%d ", matrix[rowEnd-1][i]);
            }
```

```
38              rowEnd--;
39          }
40
41          // Print the first column if it hasn't been printed
42          if(colStart < colEnd) {
43              for(i = rowEnd-1; i >= rowStart; i--) {
44                  printf("%d ", matrix[i][colStart]);
45              }
46              colStart++;
47          }
48      }
49      printf("\n");
50
51      return 0;
52  }
```