SET A

United International University

Department of Computer Science and Engineering

Final Examination Fall 2023

Course Code: **CSE 1112** Course Title: **Structured Programming Language Laboratory**

Date: January 1, 2024 Time: 09:00 AM – 10:00 AM (1 hour) Full marks: 25

Name: Student ID:

Write down C programs for the following problems in Code Blocks (or any C compiler you prefer), and present the code to your instructor after the time is up. You can make rough calculations in this paper.

Any examinee found adopting unfair means will be expelled from the trimester/program as per UIU disciplinary rules.

Problem 1 (Marks: 13)

Write a program that spells every digit in that number with the following functions and capabilities. The spelled-out number can be viewed in uppercase or lowercase format according to user choice.

1. **void toSpelledOut(int number, char* str)**: Takes the number and a blank character array. It generates the string and moves the array sent through the parameter.
2. **void upperCase(char* num)**: If called, it prints the generated string in uppercase.
3. In the main function, the user is asked to enter a number and a choice of uppercase or lowercase printing.

*Note: consider the numbers do not start or end with '0'. You can write other functions as you like.

Sample Input	Sample Output
170042083 1	For uppercase press 1 For lowercase press 2 ONE SEVEN ZERO ZERO FOUR TWO ZERO EIGHT THREE
1216 2	For uppercase press 1 For lowercase press 2 one two one six

Problem 2 (Marks: 12)

Write a C program to check whether a number is **Even powered prime number** or not. A number is an even powered prime number if it satisfies the following conditions.

- The sum of EVEN digits is greater than the sum of ODD digits in that number.
- It is a Prime number

Write the following functions to solve this problem:

a. **int even_powered_number (int x):** if the number is an even powered number, the function returns 1 else returns 0.

b. **int is_even_sum_greater (int num, int evenSum, int oddSum):** This is a recursive function that calculates the sum of even and odd digits in a given number. It returns 1 (or a true value) if the sum of even digits is greater than the sum of odd digits, and 0 otherwise. **You must write this function using recursion.**

c. **int prime_checker (int x):** This function serves as your prime number detector, taking an integer x as input and returning 1 (or a true value) if x is prime, and 0 otherwise.

d. **void find_even_powered_prime_number (int start, int end):** This function prints all even powered prime numbers hidden within the range **[start, end]**.

Explanation: 163 is an even powered prime number because the sum of odd digits, oddSum = 1 + 3 = 4, and the sum of even digits, evenSum = 6. And evenSum > oddSum. So, the condition-1 is true. 163 is also a prime number. So, the condition-2 is also true. So, 163 satisfies both conditions and it is an even powered prime number.

23 is not an even powered prime number. Though it is a prime number, the sum of odd digits, oddSum = 3, and the sum of even digits, evenSum = 2. And evenSum < oddSum. So, this number satisfies the second condition but not the first condition. That is why it is not an even powered prime number.

Sample Input	Sample Output
Enter lower limit: 1 Enter upper limit: 200	Even Powered Prime Numbers in the range [1, 200]: 2 41 43 61 83 163 181

SET B

United International University

Department of Computer Science and Engineering

Final Examination Fall 2023

Course Code: **CSE 1112** Course Title: **Structured Programming Language Laboratory**

Date: January 1, 2024 Time: 09:00 AM – 10:00 AM (1 hour) Full marks: 25

Name: Student ID:

Write down C programs for the following problems in Code Blocks (or any C compiler you prefer), and present the code to your instructor after the time is up. You can make rough calculations in this paper.

Any examinee found adopting unfair means will be expelled from the trimester/program as per UIU disciplinary rules.

Problem 1 (Marks:13)

The Caesar cipher is a simple encryption technique that was used by Julius Caesar to send secret messages to his allies. It works by shifting the letters in the plaintext message by a certain number of positions, known as the "shift" or "key". Here is an example of how to use the Caesar cipher to encrypt the message "Hello" with a shift of 3:

Original message: **Hello**

Shift Value: **3**

Replace each letter in the original message with the letter that is three positions to the right in the alphabet. H becomes K (shift 3 from H), e becomes h (shift 3 from e), l becomes o (shift 3 from l), l becomes o (shift 3 from l), o becomes r (shift 3 from l).

The encrypted message is now **Khoor.**

The process wraps around the alphabet, so for letters near the end of the alphabet, the shift continues from the beginning. For instance, with a shift of 3: X/x becomes A/a, Y/y becomes B/b, Z/z becomes C/c

Another example: **Hello** becomes **Lipps** using shift value 4 (calculate this on your own).

Your task is to implement Caesar Cipher by defining the following functions:

1. **encoder(char *p, int shift):** Takes the text to encrypt and shift value. This function adds the shift value with every character in the text and creates an encoded secret message. It also considers special cases(such as X/x, Y/y, Z/z etc).
2. **main():** This function takes the message input as a String and the shift value as an integer. The string can have both small and capital letters. It calls the encoder function to encode the message and finally prints the encoded message.

Sample Input	Sample Output
BangLaDesh 3	EdqjOdGhvk
BangLaDesh 4	FerkPeHiwl

SET B

Problem 2 (Marks: 12)

A group of enthusiastic students at UIU have been working on Number Theory for a few months, and their primary aim is to find special numbers with some interesting properties. For this purpose, they initially gathered background knowledge about prime numbers, recurrence relations, and factorials. Based on their learnings, they invented a number, which is called "***PseudoFuntorialPrime***".

To check whether a number is *PseudoFuntorialPrime* or not:

- **At first**, calculate the *funtorial* of the number by multiplying and adding, alternatively, consecutive smaller integers.
- **Secondly**, find the sum of all the digits of the result getting from the *funtorial* operation.
- **Finally**, check whether the sum of the digits is prime or not. In the final stage, if the result is true, then the given number is a *PseudoFuntorialPrime*; otherwise, it is not.

For example, 5 is a *PseudoFuntorialPrime* number. Because, the *funtorial*(5) = 5×(4+(3×(2+1)) = 5×(4+(3×3)) = 5×(4+9) = 5×13 = 65, the sum of the digits of 65 is: 6+5 = 11, and 11 is a prime.

For that purpose, you have to write the following functions:

a. **int funtorial(int n, int i):** This function takes two integers ***n*** and ***i*** (optional) as input and returns the functorial of the integer ***n***. You must write this function **using recursion.**
b. **int digitSum(int n):** This function takes an integer ***n*** and returns the sum of all the digits of the integer.
c. **int isPrime(int n):** This function takes an integer ***n*** and returns 1 if ***n*** is a prime number and 0 otherwise.
d. **int isPseudoFuntorialPrime(int n):** This function takes an integer ***n*** and returns 1 if ***n*** is a *PseudoFuntorialPrime* number (as described before) and 0 otherwise. **In this function, you should make function calls to Functions (a), (b),** and **(c)**.

Input: A positive integer, ***n* (0 ≤ *n* ≤ 18).**
Output: Print "**YES**" if the number is *PseudoFuntorialPrime*, otherwise print "**NO**".

Sample Input	Sample Output
5	YES
7	NO
10	NO
14	YES
18	NO