**Data Structures Fall 2023**

# Lab Task 03: Prompt Engineering for Templates and Unit Testing

## Introduction to Prompt Engineering:

Prompt engineering refers to the process of designing and formulating effective prompts or instructions to interact with language models, such as Chat GPT. The goal of prompt engineering is to elicit specific and desired responses from the language model by crafting input prompts in a way that guides the model's output in a particular direction.

## Advantages of Prompt Engineering:

**1.Controlled Responses**: By carefully crafting prompts, you can guide the model to generate responses that align with your specific needs. This control helps in getting accurate and relevant outputs.

**2.Customization**: Prompt engineering allows you to customize the language model's output according to your preferences, domain-specific requirements, or the tone and style you desire.

**3.Improved Relevance**: Well-designed prompts increase the likelihood that the generated content will be contextually relevant and on-topic, as they direct the model's attention to the desired subject matter.

**4.Efficiency:** Instead of relying on trial and error to get the desired response, effective prompt engineering can save time by increasing the chances of obtaining the right output on the first try.

**Note: All the lab tasks require you to submit the screenshots of the meaningful prompts given to ChatGPT. ChatGPT will only be allowed for giving prompts in this lab.**

# Lab Work

## Task 1:

There are two vectors and to check how closely they are aligned you are supposed to take their dot product. Write a template function **DotProduct(array1, array 2)** that takes two are arrays as input and returns their dot product.

Example:
**Input:**
Vector1 = {1, 2, 3,}
Vector 2 = {4, 5, 6}
**Output:** 32

**Input:**
Vector1 = {1.1, 4.2, 2.3,}
Vector 2 = {6.4, 1.5, 8.6}
**Output:** 33.12

## Task 2:

Write a function **findNthModifiedFibonacci**() using templates to find the nth Fibonacci number, but with a twist. Here each number is the sum of previous three numbers. The sequence starts with 0,1,1. In this you have to return the nth number in the sequence. N will start from 0.

**Example**
**Input:** n=3
**Explanation:** 0,1,1,2 hence at nth position we have 2
**Output:** 2

**Input:** n= 5
**Explanation:** 0, 1, 1, 2, 4,7 hence at nth position we have 7
**Output:** 7

# Task 3:

You have a 2D matrix and your task is to traverse it in a spiral order, starting from top-left corner and moving clockwise. Write a template function **spiralOrderTraversal**() that takes a 2D array as your matrix and returns an array with values stored after spiral order traversal.

**Example**

**Input:**

```
Int matrix[3][3] = {
        {1, 2, 3},
        {4, 5, 6},
        {7, 8, 9}
}
```

**Explanation**: When traversed in spiral order 1, 2, 3, 6, 9, 8, 7, 4, 5 will be the answer.

**Output:** arr[9]={1, 2, 3, 6, 9, 8,7, 4, 5}

**Input:**

```
Int matrix[4][4] = {
        {1.1, 2.1, 3.1, 4.1},
        {5.2, 6.2, 7.2, 8.2},
        {9.3, 10.3, 11.3, 12.3},
        {13.4, 14.4, 15.4, 16.4},
}
```

**Explanation**: When traversed in spiral order 1.1, 2.1, 3.1, 4.1, 8.2, 12.3, 16.4, 15.4, 14.4, 13.4, 9.3, 5.2, 6.2, 7.2, 11.3, 10.3 will be the answer.

**Output:** arr[16] = {1.1, 2.1, 3.1, 4.1,8.2, 12.3, 16.4, 15.4,14.4, 13.4, 9.3, 5.2,6.2, 7.2, 11.3, 10.3 }

# Task 4:

You have to make a class for an online Banking system. The bank keeps a record of all the transactions made along with the current balance. The bank allows a specific number of transactions to be made.

You have to define a template class **OnlineBank** which can store different types of balance e.g. int, double, float etc. Bank will have same type of elements at a time. The **OnlineBank** class should have the following member variables:

- **T currentBalance** The current balance int the bank.
- **T* withdrawals** A pointers to set the withdrawals made from the account. Before making a withdrawal, it should be checked if its smaller than or equal to the current balance.
- **T* deposits** A pointers to set the deposits made from the account.
- **numOfTransactions** this is the number of withdrawals and deposits that can be made
- **counterWithdrawal** this is the number of current withdrawals made
- **counterDeposits** this is the number of current deposits made

The class should consist of following functions:

- **Constructor** with currentbalance and number of transactions as parameter
- **isWithdrawal** checks if the number of current withdrawals is less than the numOfTransactions
- **isDeposit** checks if the number of current deposits is less than the numOfTransactions
- **makeWithdrawal** which will return true if the value passed in the parameter is less than or equal to the currentbalance and its counter is less than the limit. It will deduct the given amount from currentbalance and update it.
- **makeDeposit** which will get a value to be added in the current balance if its counter is less than the limit
- **getCurrentBalance()** will return the current balance
- **print** this will print the current balance