## NATIONAL UNIVERSITY OF COMPUTER & EMERGING SCIENCES ISLAMABAD CAMPUS

## Object Oriented Programming – Summer 2024

**ASSIGNMENT- 2**

**WELCOME TO OOP :) LET’S HAVE SOME FUN**

**Due Date: 16 July 2024**

Please follow the following submission instructions. Failure to submit according to the format would result in a deduction of marks. Submissions other than Google classroom (e.g., email etc.) will not be accepted. No late submission will be accepted. Correct and timely submission of the assignment is the responsibility of every student; hence no relaxation will be given to anyone.

# **Instructions: Total Marks: 130**

1. Make sure that you read and understand each and every instruction. If you have any questions or comments you are encouraged to discuss your problems with your colleagues (and instructors) on google classroom.
2. The student is solely responsible for checking the final .cpp files for issues like corrupt files, viruses in the file, or mistakenly exe sent. If we cannot download the file from Google Classroom, it will lead to zero marks in the assignment.
3. If there is a syntax error in the code, zero marks will be awarded in that part of the assignment.
4. Displayed output should be well mannered and well presented. Use appropriate comments and indentation in your source code.

**Honor Policy**

Plagiarism is a grave academic offense that can severely undermine your academic integrity and reputation. Any instance of a student found to have plagiarized their assignment, whether from a peer or external source, will be subject to strict consequences. This may result in a zero score for the current or all assignments, or in severe cases, even a failure in the course. Furthermore, all instances of plagiarism will be promptly reported to the **Disciplinary Committee** for further action.

# **Submission Guidelines**

1. For each question in your assignment, make a separate .cpp file e.g., for question 1, make q1.cpp and so on. Each file that you submit must contain your name, student-id, and assignment on top of the file in the comments.
2. Keep a backup of your work always that will be helpful in preventing any mishap and avoid last hour submissions
3. Combine all your work in one folder. The folder must contain only .cpp files (no binaries, no exe files etc.).
4. Rename the folder as ROLL-NUM\_PROGRAM\_SECTION (e.g., i230001\_CY-SE/CS) and compress the folder as a zip file. (e.g., i230001\_CY-SE/CS.zip). Strictly follow this naming convention, otherwise marks will be deducted.
5. Submit the .zip file on Google Classroom within the deadline.

**Note: Start early so that you can finish it on time.**

**Question 1: [15 marks]**

**Problem Statement:**

You are given an array `prices` where `prices[i]` represents the price of a given stock on the `i-th` day. You aim to maximize your profit by choosing multiple pairs of days to buy and sell stocks. However, there are several additional constraints:

1. After selling a stock, you must wait for at least two days before buying another stock.

2. You can only engage in at most 3 transactions.

3. Each transaction (a buy followed by a sell) must be completed before initiating the next.

You need to return the maximum profit you can achieve from these transactions. If you cannot achieve any profit, return 0.

**For example:**

1. If the input is `prices = [3,3,5,0,0,3,2,1,4]`, the output should be 6.

- Buy on day 4 (price = 0) and sell on day 5 (price = 3), profit = 3-0 = 3.

- Skip day 6.

- Buy on day 7 (price = 1) and sell on day 8 (price = 4), profit = 4-1 = 3.

- Total profit = 3 + 3 = 6.

1. If the input is `prices = [1,2,3,4,5]`, the output should be 4.

- Buy on day 1 (price = 1) and sell on day 5 (price = 5), profit = 5-1 = 4.

- Total profit = 4.

1. If the input is `prices = [7,6,4,3,1]`, the output should be 0.

- In this case, no transactions are done and the maximum profit is 0.

**Function :**

int maxProfit(int\* prices, int size);

**Question 02 : 20 Marks**

**Problem Statement:**

You are given two strings, `s` and `t`, of lengths `m` and `n` respectively, and an integer `k`. Your task is to find the smallest substring in `s` such that every character in string `t` appears at least `k` times in this substring. If there is no such substring, return an empty string.

The solution should ensure that:

1. All characters in `t` (including duplicates) must appear at least `k` times in the resulting substring.
2. The substring selected should be the smallest possible that satisfies the above condition.

**Examples:**

1. If the input is `s = "ADOBECODEBANC"`, `t = "ABC"`, and `k = 1`, the output should be "BANC". The smallest window substring "BANC" includes 'A', 'B', and 'C' from string t, each appearing at least once.
2. If the input is `s = "aaadbccba"`, `t = "abc"`, and `k = 2`, the output should be "adbccba". The smallest window substring "adbccba" contains at least 2 'a's, 2 'b's, and 2 'c's.
3. If the input is `s = "aaaaabbbbccccc"`, `t = "abc"`, and `k = 3`, the output should be "abbbccc". The smallest window substring "abbbccc" contains at least 3 'a's, 3 'b's, and 3 'c's.
4. If the input is `s = "thisisateststring"`, `t = "tist"`, and `k = 1`, the output should be "tstri". The smallest window substring "tstri" contains 't', 'i', 's', and another 't' from string t, each appearing at least once.

**Notes:**

* If multiple minimum length substrings satisfy the condition, returning any one of them is acceptable.
* Characters in `t` might repeat, and hence they need to appear at least `k` times within the window.
* Ensure your solution is optimized for performance, especially considering the potentially large values of `m` and `n`.

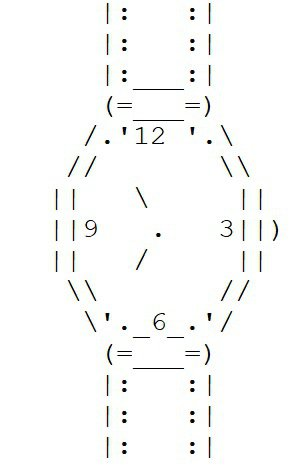
**Function:**

char\* smallest\_substring\_with\_k\_occurrences(const char \*s, const char \*t, int k);

**Question 03 : 25 Marks**

**Problem Statement**

Create a C++ program that prints the pattern of the fixed time for your last demo slot of Assignment 1. The output should clearly and attractively display the fixed time. Submit only the source code file (`.cpp`). Ensure that your program is recursive, with no more than three functions allowed. Any helper functions must also be recursive, and the use of global or static variables is prohibited. Below is an example of the desired output.



**Question no 04 : 30 Marks**

# 

## Objective

The goal is to develop a simplified Banking Management System that simulates the basic operations of a bank and an ATM without employing advanced data structures like LinkedLists or Queues. Instead, the focus will be on fundamental programming concepts such as classes, loops, and conditionals.

## Key Features

The system should offer an interactive user interface with the following functionalities:

* Bank Entrance:
  + Pay Bills
  + Withdraw/Deposit Cash
  + Open New Account
  + View Account Details
  + Exit to Main Menu
* ATM Entrance:
  + Pin and Debit Card Validation
  + Withdraw Cash
  + Change Pin
  + Balance Inquiry
  + Exit
* Administrator Access (Restricted Entry):
  + User Authentication
  + Bill Payment Processing
  + Account Opening Processing
  + Exit
* System Exit:
  + Safe termination of the program with a farewell message.

## Program Flow

* Welcome Screen: A greeting message followed by a prompt to advance to the main menu.
* Main Menu: Provides 4 primary options:
  + Enter Bank
  + Enter ATM
  + Administrator Access
  + Exit System
* Bank Section:
  + Presents bank-related services.
  + Requires customer interaction for transactions.
* ATM Section:
  + Requests user PIN and card number.
  + Provides ATM services upon authentication.
* Administrator Section:
  + Requires admin credentials.
  + Allows managing queued operations.
* Exit Sequence: Displays a parting message before program closure.

## Restrictions

* Do not use any pre-built data structures provided by programming libraries.
* Manage users and transactions using simple arrays or standalone variables.
* Implement error checking and handle incorrect user inputs gracefully.
* Use operator overloading where applicable to demonstrate an understanding of the concept.
* Focus on encapsulating functionality within appropriate classes and methods.

## Conclusion

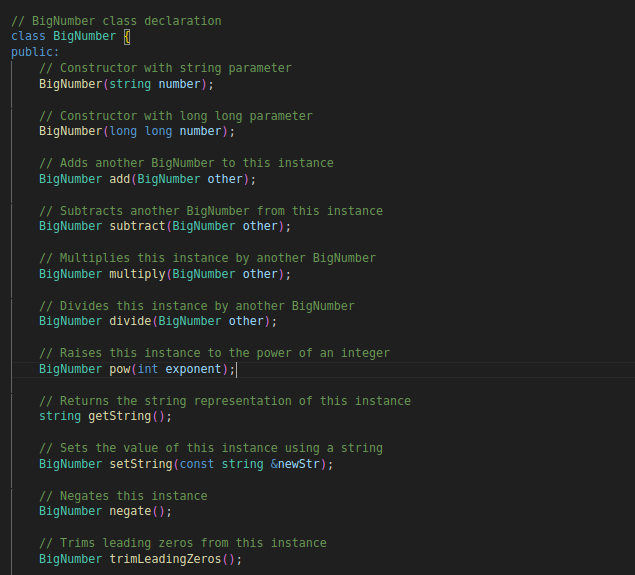
This Banking Management System aims to showcase practical application of object-oriented programming principles while providing a user-friendly banking simulation. Emphasizing simplicity and core concepts will ensure a strong foundation for further improvements and the addition of complex features in subsequent versions of the system.

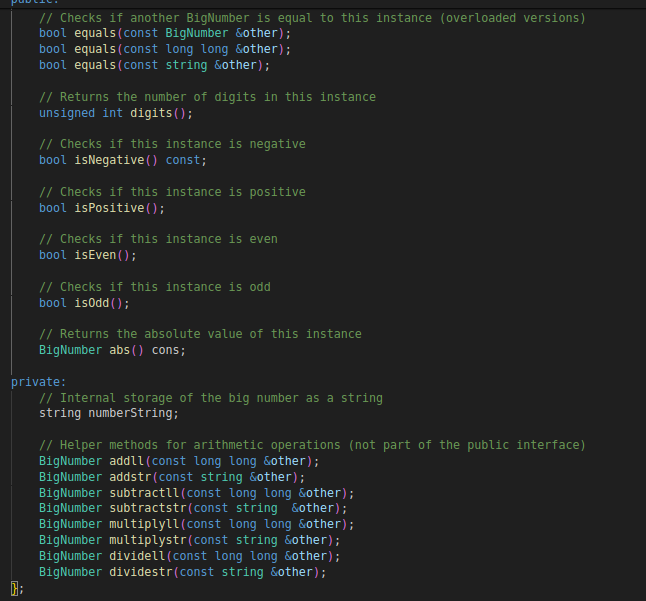
## Additional Considerations

Marks will be awarded based not only on the functionality of the program but also on the quality and intuitiveness of the user interface. Therefore, attention should be paid to both the backend logic as well as the frontend presentation. It is crucial to proceed with a clear plan for both aspects of your Banking Management System to ensure a comprehensive evaluation of your skills.

**Question No 05: 20 Marks**

The goal is to design and implement a BigNumber class in C++ capable of handling arithmetic for numbers larger than what can be stored in standard data types such as int, long, or long long. Native data types have limitations on the maximum value they can store, and our task is to overcome this limitation by representing and performing calculations on large numbers that may exceed these bounds.





Note :

This problem statement outlines the essential requirements for the BigNumber class. Implementers may also need to consider memory management, error handling, and potential optimizations

**Question no 06: 20 Marks**

## Advanced Problem Statement for Car Rental Platform

You are developing an advanced car rental platform that enables users to create, update, and manage their rental reservations with a high degree of complexity. The platform should incorporate intricate validation mechanisms, dynamic pricing algorithms, and efficient resource management to ensure a seamless user experience. Implement a RentalReservation class with the following specifications:

### Class: RentalReservation

#### Private Instance Variables

1. reservation\_id: A unique identifier for each reservation, requiring stringent format validation.
2. customer\_name: Name of the customer making the reservation.
3. car\_make\_model: The make and model of the car being rented.
4. pickup\_date: Date and time when the customer intends to pick up the car.
5. return\_date: Date and time when the customer plans to return the car.
6. rental\_duration: Duration of the rental period in hours, minutes, and days, calculated dynamically.
7. rental\_rate: The rate per hour or day depending on the car type and rental period.
8. rental\_total: The total cost of the rental, calculated based on the rental duration and rate.
9. additional\_services: Optional services such as GPS, child seat, insurance, etc., with additional costs.
10. customer\_age: Age of the customer, which can affect the rental rate due to insurance reasons.

#### Constructor

* Initializes the reservation with parameters including reservation\_id, customer\_name, car\_make\_model, pickup\_date, return\_date, additional\_services, and customer\_age.
* Validates and sets the reservation\_id format, ensuring it meets specific criteria.
* Calculates and sets rental\_duration, rental\_rate, and rental\_total based on the pickup and return dates, car's rental rates, additional services, and customer age.

#### Destructor

* Prints a message indicating when the object is destroyed, ensuring proper resource management and logging.

#### Methods

1. validateReservationID: A method to validate if the reservation\_id adheres to a specific format:
   * The ID should consist of 8 alphanumeric characters, followed by 2 special characters, and ending with 4 numbers. The sum of these four numbers must be less than 18.
2. validateCarMakeAndModel: Checks if the car\_make\_model provided is a valid option from a dynamically loaded list of available cars, considering new car models can be added or removed.
3. calculateRentalRate: Computes and returns the rental rate for a specific car based on its make and model, rental duration, customer age, and peak or off-peak periods.
4. calculateRentalTotal: Computes and returns the total cost of the rental based on rental\_duration, rental\_rate, and any additional services selected by the customer.
5. getReservationDetails: Retrieves and returns detailed information about the reservation, including reservation\_id, customer\_name, car\_make\_model, pickup\_date, return\_date, rental\_duration, rental\_rate, rental\_total, and additional\_services.
6. updateReservation: Allows users to update their rental reservation information, including pickup\_date, return\_date, car\_make\_model, rental\_rate, and additional\_services. Ensures all updates are validated and recalculates rental\_duration, rental\_rate, and rental\_total accordingly.

### Additional Requirements

* Implement robust error handling and validation mechanisms for user inputs, ensuring no invalid data is processed.
* Ensure the system manages resource allocation and deallocation efficiently, especially when handling multiple reservations concurrently.
* Incorporate a dynamic pricing algorithm that adjusts rental rates based on factors such as demand, seasonality, customer age, and rental duration.
* Implement an efficient data retrieval mechanism, utilizing appropriate data structures to optimize performance for reservation management.
* Include logging and monitoring features to track reservation changes and system performance, aiding in debugging and maintenance.
* Implement a user authentication and authorization system to ensure only authorized users can create or modify reservations.
* Provide an interface for administrators to manage car inventory, including adding, updating, or removing car models and setting their rental rates.

This advanced problem statement introduces complexities in validation, dynamic pricing, and resource management, making the implementation more challenging and robust.

***\*\* Good Luck \*\*\****