# Data Structures

## Assignment 2–All sections

### Submission Items:

1. **Complete the code files (that have been provided with the assignment). Comments with all core functions are mandatory and hold 10% of the assignment marks. You will have a separate question file for each question.**

### Assignment Information

- **Coding language: C++**

### Prerequisite for this assignment

**You need knowledge about**

- **Variable**
- **Input and output**
- **Conditions**
- **Loops**
- **Structures**
- **Classes**
- **Recursions**
- **Arrays**
- **Linked List**

### Instructions (before starting the assignment):

1. Assignments are to be done individually.
2. The code you write must be your own and you must understand each part of your code. You are encouraged to get help from the course instructors through google classroom and email.
3. Do not use any string built-in functions (such as strlen, strcmp etc). **Caution**: zero marks will be awarded. (Question 4)
4. Apply all validations for invalid inputs.
5. Do not edit **Function Prototypes**. **Caution**: zero marks will be awarded. (unless specified)
6. **Plagiarism**: Plagiarism of any kind (copying from others, copying from the internet, etc) is not allowed. If found plagiarized, you will be awarded zero marks in the assignment. Repeating such an act can lead to strict disciplinary actions and failure in the course.
7. **Please start early otherwise you will struggle with the assignment.**

### Submission Guidelines

a. Only submit .cpp file for each question Your submission.cpp file must contain your name, student-id, and assignment # on the top of the file in the comments. Example the first line of every question should be

//Bilal_Khalid_Dar_22i2348 .
Missing this will result in 20% marks deduction in each question.

b. Move your all .cpp in one folder. The folder must contain only submission.cpp files (no binaries, no exe files etc.,). If we are unable to download your submission due to any reason you will be awarded zero mark.
c. Run and test your program on a lab machine before submission. If there is a syntax error, zero marks will be awarded in that specific question.
d. Rename the folder as ROLL-NUM_SECTION (e.g. 21i-0001_A) and compress the folder as a zip file. (e.g. 21i-0001_A.zip). Only zip file will be acceptable.
e. Submit the .zip file on Google Classroom within the deadline. **Late submission will be marked zero**. No exceptions
f. Submissions other than Google classroom (e.g. email etc.) will not be accepted.
g. The student is solely responsible to check the final zip files for issues like corrupt files, viruses in the file, mistakenly exe sent. If we cannot download the file from Google classroom due to any reason it will lead to zero marks in the assignment.

## General Rules

- This is an individual assignment.
- All coding standards must be followed (meaningful variable name, code comments etc.)
- Apply all validations for invalid inputs.

## Judging Criteria

- The effectiveness of the algorithm used in solving the problem statement
- Understanding of the basic concepts involved
- The simplicity and ease of use of input interface, and the aesthetics of the display
- Completion of program as per given instructions.

**For any query email: bilal.khalid@nu.edu.pk**

# Question 1 – Managing FAST Society Data

FAST University allows students to make different societies. Currently, there are three main active societies which are FAST Gaming Club (FGC), FAST Adventure society (FAS) and Fast Dramatic Society (FDS). The university allows their students to be part of multiple societies. All of these societies maintain the record of their students in a singly linked list name FGC_List, FAS_List and FDS_List respectively. The university you find the following information with the help of this list.

**Q1: Provide a combined list of all students which are part of different societies.**

For example

```
FGC_List = {Ali, Usman, Haider, Maryam, Masooma, Urooj}

FAS_List = {Ashiq, Manika, Ali, Masooma, Akbar, Urooj}

FDS_List = {Masooma, Bilal, Amna, Madiha, Rohail, Urooj}
```

The result will be : `List_of_all_student (string array) = {Ali, Usmanan, Haider, Maryam, Masooma, Urooj, Ashiq, Manika, Akbar, Bilal, Amna, Madhiha, Rohail}`

Please note that the name occurring in multiple list will appear only once.

**Q2 List of all students which are part of all societies.**

For example

```
FGC_List = {Ali, Usman, Haider, Maryam, Masooma, Urooj}

FAS_List = {Ashiq, Manika, Ali, Masooma, Akbar, Urooj}

FDS_List = {Masooma, Bilal, Amna, Madiha, Rohail, Urooj}
```

The result will be: `List_of_common_student (string array): {Masooma, Urooj}`

Note: You cannot declare any new array or list

# Question 2 – Tricky List Search

Binary search is an efficient search algorithm that can be used on sorted lists to find a specific element. It works by repeatedly dividing the search interval in half until the target element is found or the search interval is empty.

To use binary search with a doubly linked list, we need to follow a few steps:

1. Find the middle node of the list.
2. Compare the target value with the value of the middle node.
3. If the target value is smaller than the value of the middle node, search the left half of the list.

4. If the target value is larger than the value of the middle node, search the right half of the list.
5. Repeat steps 1-4 until the target value is found or the search interval is empty.

For this program, ask the user to input data in a doubly linked list. Ensure to maintain a counter for the number of elements in the list. Make sure that the list is sorted for the binary search. If not, use insertion sort algorithm to sort the doubly linked list in ascending order. Once the list is sorted, ask user to enter the data that they want to find in the doubly link list. Apply binary search using the following rule. Find the middle of the list by dividing the number of elements /2. If there are 10 elements in the list, the first mid will be Node 5. Build your logic around it and find if the element exists or not.

## Question 3 - Lift System Automation

FAST university has hired you to make an operating system for their lifts. Below are specifications of the lifts OS.

The lift will be placed in a building with 6 floors including a basement. From basement to top floors are labeled as -1,0,1,2,3,4 respectively. In the morning, the lift goes into operation from the basement. The lift can go up and down. If maintenance is required, the lift can be halted. If the lift is halted, the lift not usable during that period. Once unhalted, the lift can be used again.

All this lift data is stored in a doubly linked list. In case university decides to make a new floor, they can enter a new floor as floor 5. Moreover, due to some operational constraints, they might decide to choose that the lift should not stop on some floor for better load management. For example, the university can decide the lift will only operate on basement, 1st floor and 3rd floor then the lift will only operate on these floors. University can decide to change it and make it operational for all floors as well.

With the help of above information, make a lift operating system which will have following functions.

```
int lift_operating_system(int requested_floor, lift_floors *current, lift_floors *head, lift_floors *tail, char lift_status)
```

// this function valises the lift request and then performs the lift up or lift down function. Once the function is performed, the lift operating system returns the new current floor after lift goes up or down.

```
int liftup(lift_floors *current, lift_floors *head, lift_floors *tail,int requested_floor )
```

// a recursive function used to move the lift in upwards direction

```
int liftdown(lift_floors *current, lift_floors *head, lift_floors *tail,int requested_floor )
```

// a recursive function used to move the lift in downwards direction

**char halt_lift(char status)**

//halts the lift, no up and down operation can be performed. Stored H for halting

**char un_halt_lift(char status)**

//Unhalts the lift. Store W which represents that the lift is working.

void add_floor(lift_floors *new_floor, lift_floors *head, lift_floors *tail)

//add a lift floor;

**void skip_floor(lift_floors *head, lift_floors *tail, int floorNo)**

//add logic to make lift skip a certain floor

**void make_floor_operational(lift_floors *head, lift_floors *tail, int floorNo)**

//add logic to make lift operational on a certain floor

**Main function:**

The main maintains the following is in an endless loop and asks the user to select on of the following operations.

(1) Go to a floor.
2) Halt the lift.
3) Unhalt the lift.
4) Add floor.
5) Skip Floor
6) Make floor operational.
7) Exit

# Question 4 - String Manipulation

For question 4, a text file is provided. Use the text file provided to make your solution.

Design a String Manipulation class that can be used to perform following functions related to strings:

1. Read the text file provided with the assignment. Make a singly link list and store each character of the provided file in one node. For example:

File content: I am a boy.

It will result in adding a linked list with 11 nodes where each character represents one node. (See following example)

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| I | | a | m | | a | | b | o | y | . |

Once the list is created, the following functions can be performed on the given list.

## String Manipulation Class:

The class should have followed functions:

**1. int Calculate_length (StringList *head)**

The function should get a stringList as input and return the length of the string. The length should be measured with the help of a recursive function.

**2. bool substring (StringList *head, String str)**

Write code to identify the if a given string existing in the StringList. Ask a user to input the substring. The StringList represents the actual string and String str represents the string that they should find e.g.

StringList = “I am taking the DS Class”

Str = “DS”

The function should return true as the DS exist in the above string.

Another example

StringList = “I am taking the DS Class”

Str = “taking”

The function should return true as it exists in the above string.

Another example

StringList = “I am taking the DS Class”

Str = “that DS”

The function should return false as it does not exist in the above string.

Another example

StringList = “I am taking the DS Class”

Str = “Taking the DS class and doing the assignment”

The function should return false as it does not exist in the above string.

**3. int substring_position (StringList *head, String str)**

The above function returns the index of the main string where the sub string starts.

For example

StringList = “I am taking DS course”

Str = “taking”

The function will return 5 as substring exist and starts at 5th index of **StringList**. In case the substring does not exist, you will return -1.

**4. void replaceString (StringList *head, String find, String replace)**

This function will allow the user to replace a given string with existing string. For example

StringList: I am taking DS class.

FindText: a

ReplaceText: Y

The resultant string is: I Ym tYking DS clYss

**5. void appendText (StringList *head, String appends, int index)**

This function will allow the user to append a string in stringllist, For example.

StringList: I am taking DS class.

Index = 5

Append text: not

Resulting = I am not taking DS class

**6. void deleteText (StringList *head, String delText)**

This function will allow the user to delete a string in stringllist, For example.

StringList: I am taking DS class.

Delete text: taking D

Resulting = I am not S class

If the deleting text does not exist, show error that the text cannot be deleted

## Main function:

- Create the instance of String manipulation structure.
- Create string list with the text file.
- Find its length.
- As the user for sub string, he/she wants to find.
- Find the length of substring.
- Check is substring string exists.
- Replace Text
- Append Text
- Delete Text

**Please ensure all validations are applied!**