# Rajalakshmi Engineering College

Name: Karthikeyan M
Email: 240801150@rajalakshmi.edu.in
Roll no: 2116240801150
Phone: 8056008890
Branch: REC
Department: I ECE FB
Batch: 2028
Degree: B.E - ECE

Scan to verify results

## NeoColab_REC_CS23231_DATA STRUCTURES

## REC_DS using C_Week 1_COD_Question 3

Attempt : 1
Total Mark : 10
Marks Obtained : 10

## Section 1 : Coding

1. Problem Statement

Imagine you are working on a text processing tool and need to implement a feature that allows users to insert characters at a specific position.

Implement a program that takes user inputs to create a singly linked list of characters and inserts a new character after a given index in the list.

*Input Format*

The first line of input consists of an integer N, representing the number of characters in the linked list.

The second line consists of a sequence of N characters, representing the linked list.

The third line consists of an integer index, representing the index(0-based) after which the new character node needs to be inserted.

The fourth line consists of a character value representing the character to be inserted after the given index.

*Output Format*

If the provided index is out of bounds (larger than the list size):

1. The first line of output prints "Invalid index".
2. The second line prints "Updated list: " followed by the unchanged linked list values.

Otherwise, the output prints "Updated list: " followed by the updated linked list after inserting the new character after the given index.

Refer to the sample output for formatting specifications.

*Sample Test Case*

Input: 5
a b c d e
2
X
Output: Updated list: a b c X d e

*Answer*

```
#include <stdio.h>
#include <stdlib.h>

typedef struct Node {
    char data;
    struct Node* next;
} Node;

// Function to create a new node
Node* createNode(char data) {
    Node* newNode = (Node*) malloc(sizeof(Node));
    if (!newNode) {
        printf("Memory allocation failed\n");
        exit(1);
    }
    newNode->data = data;
    newNode->next = NULL;
    return newNode;
}

// Append node at the end of the list
void appendNode(Node** head_ref, char data) {
    Node* newNode = createNode(data);
    if (*head_ref == NULL) {
        *head_ref = newNode;
        return;
    }
    Node* temp = *head_ref;
    while (temp->next != NULL)
        temp = temp->next;
    temp->next = newNode;
}

// Print the list
void printList(Node* head) {
    Node* temp = head;
    while (temp != NULL) {
        printf("%c ", temp->data);
        temp = temp->next;
    }
    printf("\n");
}

// Insert new node after given index
int insertAfterIndex(Node* head, int index, char data) {
    Node* temp = head;
    int count = 0;

    // Traverse to the node at position index
    while (temp != NULL && count < index) {
        temp = temp->next;
        count++;
    }
    // If index is out of bounds
    if (temp == NULL)
        return 0;

    Node* newNode = createNode(data);
    newNode->next = temp->next;
    temp->next = newNode;

    return 1;
}

int main() {
    int N;
    scanf("%d", &N);

    Node* head = NULL;

    // Read N characters
    for (int i = 0; i < N; i++) {
        char ch;
        scanf(" %c", &ch);
        appendNode(&head, ch);
    }

    int index;
    scanf("%d", &index);

    char newChar;
    scanf(" %c", &newChar);

    int success = insertAfterIndex(head, index, newChar);

    if (!success) {
        printf("Invalid index\n");
    }

    printf("Updated list: ");
    printList(head);

    // Free memory (optional)
    Node* current = head;
    while (current != NULL) {
        Node* temp = current;
        current = current->next;
        free(temp);
    }

    return 0;
}
```

***Status :*** Correct ***Marks : 10/10***