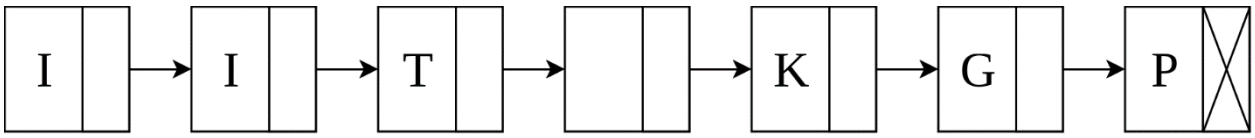
Section 6 Autumn-2024

PDS Lab **Lab-11** **05.11.2024**

Instructions:

- This lab is based on the topics: Structures.
- You should save each program with the file name as specified against each problem as <Lab#>-<Assignment#>-<Roll#>.c. For example, **11-01-24NA10006.c** to save Program to 1st assignment in Lab 11 with Roll Number 24NA10006
- You should upload each program to the Moodle system. Also, copy + paste your programs to the text window on the test page.
- A few test cases against each problem are given for your reference, including but not limited to.
- There are three problems and the maximum time allowed is 150 minutes.
- **Remember to comment your program**

1. Write a C program to do the following.
    a. Read any string from the keyboard and store it.
    b. Store the characters in the input string in a single linked list. For example, the string "IIT KGP" will be stored as a linked list as shown,
    c. Traverse the entire list and print the character stored in each node.



NOTE: Your input string may contain any character including space.
HINT: Use command line argument to give input to the program.

[10 + 10 + 5 = 25]

```c
// Code creator: Atonu (atonughosh@kgpian.iitkgp.ac.in)
#include <stdio.h>
#include <stdlib.h>

// Define the node structure
struct Node {
    char data;
    struct Node* next;
};

// Function to create a new node with given character data
struct Node* createNode(char data) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->data = data;
    newNode->next = NULL;
    return newNode;
}

// Function to append a character to the linked list
void appendNode(struct Node** head_ref, char data) {
    struct Node* newNode = createNode(data);
    if (*head_ref == NULL) {
        *head_ref = newNode;
    } else {
        struct Node* temp = *head_ref;
        while (temp->next != NULL) {
            temp = temp->next;
        }
        temp->next = newNode;
    }
}
```

```c
// Function to traverse and print the linked list
void printList(struct Node* head) {
    struct Node* temp = head;
    while (temp != NULL) {
        printf("%c", temp->data);
        temp = temp->next;
    }
    printf("\n");
}

// Main function
int main(int argc, char *argv[]) {
    if (argc != 2) {
        printf("Usage: %s <string>\n", argv[0]);
        return 1;
    }

    struct Node* head = NULL;
    char *input = argv[1];

    // Store each character in the linked list
    for (int i = 0; input[i] != '\0'; i++) {
        appendNode(&head, input[i]);
    }

    // Print the linked list
    printf("Stored characters in linked list: ");
    printList(head);

    return 0;
}
```

2. Write a C program to do the following.

    a. Read any two strings, say **str1** and **str2** from the keyboard.
    b. Store the strings in two linked lists, say **lstr1** and **lstr2**.
    c. Merge the two linked lists **lst1** and **lstr2** to **lstr**.
    d. Display the contents of lstr on the screen.

NOTE: You should not take any extra memory while the linked lists are being merged.

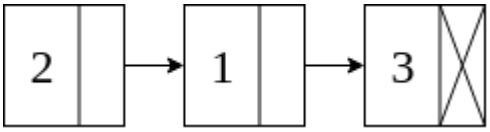
[5 + (5 + 5) + 15 = 30]

```c
// Code creator: ArunB (arunbsmn@kgpian.iitkgp.ac.in)
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

// Define the linked list node structure
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));
    newNode->data = data;
    newNode->next = NULL;
    return newNode;
}

// Function to append characters of a string to a linked list
Node* stringToLinkedList(const char *str) {
    Node *head = NULL, *tail = NULL;
    for (int i = 0; str[i] != '\0'; i++) {
        Node *newNode = createNode(str[i]);
        if (head == NULL) {
            head = tail = newNode;
        } else {
            tail->next = newNode;
            tail = newNode;
        }
    }
    return head;
}

// Function to merge two linked lists without using extra memory
Node* mergeLinkedLists(Node *lstr1, Node *lstr2) {
    if (lstr1 == NULL) return lstr2;
```

```c
    if (lstr2 == NULL) return lstr1;

    Node *tail = lstr1;
    while (tail->next != NULL) {
        tail = tail->next;
    }
    tail->next = lstr2;
    return lstr1;
}

// Function to display the contents of a linked list
void displayLinkedList(Node *head) {
    while (head != NULL) {
        printf("%c", head->data);
        head = head->next;
    }
    printf("\n");
}

// Function to free the memory of a linked list
void freeLinkedList(Node *head) {
    Node *temp;
    while (head != NULL) {
        temp = head;
        head = head->next;
        free(temp);
    }
}

int main() {
    char str1[100], str2[100];

    // Read two strings from the keyboard
    printf("Enter first string: ");
    fgets(str1, sizeof(str1), stdin);
    str1[strcspn(str1, "\n")] = 0; // Remove trailing newline if any

    printf("Enter second string: ");
    fgets(str2, sizeof(str2), stdin);
    str2[strcspn(str2, "\n")] = 0; // Remove trailing newline if any

    // Convert strings to linked lists
    Node *lstr1 = stringToLinkedList(str1);
    Node *lstr2 = stringToLinkedList(str2);

    // Merge the two linked lists
    Node *lstr = mergeLinkedLists(lstr1, lstr2);
```

```
    // Display the merged linked list
    printf("Merged linked list: ");
    displayLinkedList(lstr);

    // Free the memory allocated for the linked list
    freeLinkedList(lstr);

    return 0;
}
```

3. Write a C program to do the following.
    a. Read any integer number from the keyboard.
    b. Store the digits of the number in a single linked list. For example, if you read a number, say "**213**", then your linked list would look like,



    c. Reverse the linked list. For this, you should not use any additional memory or linked list.
    d. Print the number that the reversed list stores.
    e. Check if the digits in the number form a Palindrome.

HINT: Two linked lists contain the digits in the same order.

[5 + (5 + 10) + 10 + 5 + 10 = 45]

```c
// Code creator: Bhagoti (bhagotimahala@kgpian.iitkgp.ac.in)
#include <stdio.h>
#include <stdlib.h>

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

// Function to create a new node
Node* createNode(int data) {
    Node* newNode = (Node*)malloc(sizeof(Node));
    newNode->data = data;
    newNode->next = NULL;
    return newNode;
}

// Function to add a node at the front of the linked list
void addFront(Node** head, int data) {
    Node* newNode = createNode(data);
    newNode->next = *head;
    *head = newNode;
}

// Function to reverse the linked list
void reverse(Node** head) {
    Node *prev = NULL, *current = *head, *next = NULL;
    while (current != NULL) {
        next = current->next;
        current->next = prev;
```

```
        prev = current;
        current = next;
    }
    *head = prev;
}

// Function to print the linked list
void printList(Node* head) {
    Node* temp = head;
    while (temp != NULL) {
        printf("%d", temp->data);
        if (temp->next != NULL) {
            printf("->");
        }
        temp = temp->next;
    }
    printf("\n");
}

// Function to check if the linked list is a palindrome
int isPalindrome(Node* head) {
    Node *slow = head, *fast = head, *prevSlow = NULL;
    Node *secondHalf, *midNode = NULL;
    int result = 1;

    if (head != NULL && head->next != NULL) {
        // Use fast and slow pointers to reach the middle of the list
        while (fast != NULL && fast->next != NULL) {
            fast = fast->next->next;
            prevSlow = slow;
            slow = slow->next;
        }

        // If there are odd elements, skip the middle element
        if (fast != NULL) {
            midNode = slow;
            slow = slow->next;
        }

        secondHalf = slow;
        prevSlow->next = NULL; // NULL terminate the first half
        reverse(&secondHalf);  // Reverse the second half

        // Compare the two halves
        result = compareLists(head, secondHalf);

        // Reconstruct the original list
        reverse(&secondHalf);
```

```
        if (midNode != NULL) {
            prevSlow->next = midNode;
            midNode->next = secondHalf;
        } else {
            prevSlow->next = secondHalf;
        }
    }
    return result;
}

// Function to compare two linked lists
int compareLists(Node* head1, Node* head2) {
    Node* temp1 = head1;
    Node* temp2 = head2;

    while (temp1 && temp2) {
        if (temp1->data != temp2->data) {
            return 0;
        }
        temp1 = temp1->next;
        temp2 = temp2->next;
    }
    return (temp1 == NULL && temp2 == NULL);
}

int main() {
    Node* head = NULL;
    int number;

    printf("Enter an integer: ");
    scanf("%d", &number);

    // Store each digit in the linked list from left to right
    int temp = number;
    if (temp == 0) {
        addFront(&head, 0);
    } else {
        while (temp > 0) {
            addFront(&head, temp % 10);
            temp /= 10;
        }
    }

    printf("Original number in linked list form: ");
    printList(head);

    // Reverse the linked list
    reverse(&head);
```

```
    printf("Reversed number in linked list form: ");
    printList(head);

    // Check if palindrome
    if (isPalindrome(head)) {
        printf("The number is a palindrome.\n");
    } else {
        printf("The number is not a palindrome.\n");
    }

    return 0;
}
```