Student Information

- **Name: [ATUL KUMAR]**
- **Student ID: [590011190]**
- **Branch: [MCA]**
- **Batch: [B1]**
- **Instructor: [Dr. Sourbh Kumar]**
- **Date: [07/11/2024]**

Lab Experiment: 03 **Batch: 1 & 2**
Subject: Data Structures Lab **MCA**
Semester: 1st

Assignment Tasks

1. **Singly Linked List Implementation:**

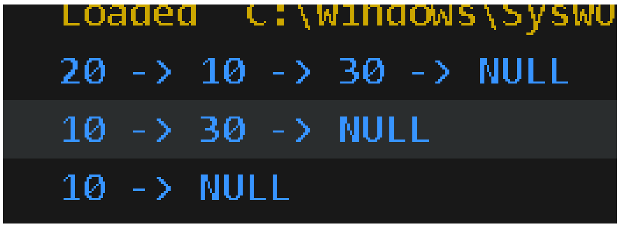
- Create a structure for a singly linked list node with data and a next pointer.
- Implement functions for:
- Insertion at the beginning, end, and a specified position.
- Deletion from the beginning, end, and a specified position.
- Displaying the list.

Solution:--

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

// Node structure
struct Node {
int data;
struct Node* next;
};

// Insert a node at the beginning
void insert_at_beginning(struct Node** head, int data) {
struct Node* new_node = (struct Node*)malloc(sizeof(struct
      Node));
new_node->data = data;
new_node->next = *head;
*head = new_node;
}

// Insert a node at the end
void insert_at_end(struct Node** head, int data) {
struct Node* new_node = (struct Node*)malloc(sizeof(struct
      Node));
new_node->data = data;
new_node->next = NULL;

if (*head == NULL) {
*head = new_node;
} else {
struct Node* temp = *head;
while (temp->next != NULL) {
temp = temp->next;
}
temp->next = new_node;
}
}

// Delete the node at the beginning
void delete_from_beginning(struct Node** head) {
if (*head == NULL) {
printf("List is empty.\n");
return;
}
struct Node* temp = *head;
*head = (*head)->next;
free(temp);
}

// Delete the node at the end
void delete_from_end(struct Node** head) {
if (*head == NULL) {
printf("List is empty.\n");
return;
}
if ((*head)->next == NULL) {
free(*head);
*head = NULL;
} else {
struct Node* temp = *head;
while (temp->next->next != NULL) {
temp = temp->next;
}
free(temp->next);
temp->next = NULL;
}
}

// Display the list
void display(struct Node* head) {
struct Node* temp = head;
while (temp != NULL) {
printf("%d -> ", temp->data);
temp = temp->next;
}
printf("NULL\n");
}

// Example usage
int main() {
struct Node* head = NULL;
```

```
            insert_at_beginning(&head, 10);
            insert_at_beginning(&head, 20);
            insert_at_end(&head, 30);
            display(head);  // Expected Output: 20 -> 10 -> 30 -> NULL

            delete_from_beginning(&head);
            display(head);  // Expected Output: 10 -> 30 -> NULL

            delete_from_end(&head);
            display(head);  // Expected Output: 10 -> NULL

            return 0;
        }
```

Output:--



```
Loaded  C:\Windows\SysWO
20 -> 10 -> 30 -> NULL
10 -> 30 -> NULL
10 -> NULL
```

2. **Doubly Linked List Implementation:**

- Modify the singly linked list to a doubly linked list by adding a prev pointer.
- Implement the same insertion, deletion, and display functions.

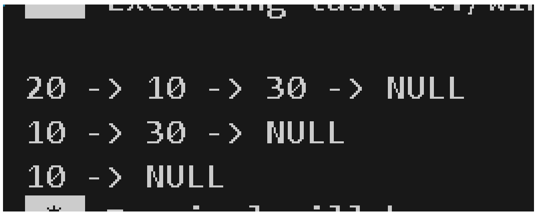
Solution:--

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

// Node structure for doubly linked list
struct Node {
int data;
struct Node* next;
struct Node* prev;
};

// Insert a node at the beginning
void insert_at_beginning(struct Node** head, int data) {
struct Node* new_node = (struct Node*)malloc(sizeof(struct
      Node));
new_node->data = data;
new_node->next = *head;
new_node->prev = NULL;

if (*head != NULL) {
(*head)->prev = new_node;
}
*head = new_node;
}
```

```
// Insert a node at the end
void insert_at_end(struct Node** head, int data) {
struct Node* new_node = (struct Node*)malloc(sizeof(struct
      Node));
new_node->data = data;
new_node->next = NULL;

if (*head == NULL) {
new_node->prev = NULL;
*head = new_node;
return;
}

struct Node* temp = *head;
while (temp->next != NULL) {
temp = temp->next;
}

temp->next = new_node;
new_node->prev = temp;
}

// Delete the node at the beginning
void delete_from_beginning(struct Node** head) {
if (*head == NULL) {
printf("List is empty.\n");
return;
}

struct Node* temp = *head;
*head = (*head)->next;

if (*head != NULL) {
(*head)->prev = NULL;
}

free(temp);
}

// Delete the node at the end
void delete_from_end(struct Node** head) {
if (*head == NULL) {
printf("List is empty.\n");
return;
}

struct Node* temp = *head;

if (temp->next == NULL) {
*head = NULL;
free(temp);
return;
}

while (temp->next != NULL) {
temp = temp->next;
}

temp->prev->next = NULL;
free(temp);
}

// Display the list in forward direction
void display_forward(struct Node* head) {
struct Node* temp = head;
while (temp != NULL) {
printf("%d -> ", temp->data);
temp = temp->next;
}
printf("NULL\n");
}

// Display the list in reverse direction
void display_reverse(struct Node* head) {
if (head == NULL) {
printf("List is empty.\n");
return;
}

struct Node* temp = head;
while (temp->next != NULL) {
temp = temp->next;
}

while (temp != NULL) {
printf("%d -> ", temp->data);
temp = temp->prev;
}
printf("NULL\n");
}

// Example usage
int main() {
struct Node* head = NULL;

insert_at_beginning(&head, 10);
insert_at_beginning(&head, 20);
insert_at_end(&head, 30);

printf("Forward display: ");
```

```
        display_forward(head);  // Expected output: 20 -> 10 -> 30 ->
            NULL

        printf("Reverse display: ");
        display_reverse(head);  // Expected output: 30 -> 10 -> 20 ->
            NULL

        delete_from_beginning(&head);
        printf("After deleting from beginning: ");
        display_forward(head);  // Expected output: 10 -> 30 -> NULL

        delete_from_end(&head);
        printf("After deleting from end: ");
        display_forward(head);  // Expected output: 10 -> NULL

        return 0;
}
```

Output:--



```
  Executing task: C:\Wi

20 -> 10 -> 30 -> NULL
10 -> 30 -> NULL
10 -> NULL
```

3. **Application Example:**
    - Demonstrate an application of linked lists, such as managing a to-do list or implementing a simple stack/queue.

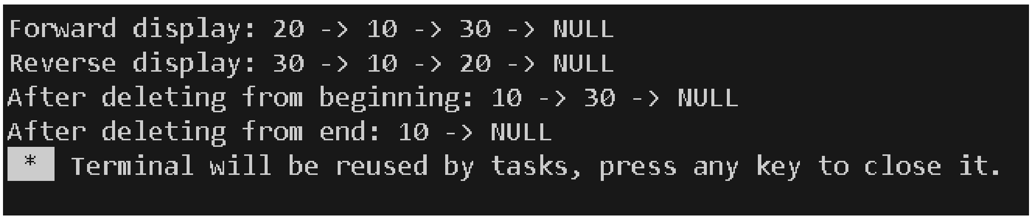
Solution:--

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

// Node structure for each to-do task
struct Task {
  char description[100];
  struct Task* next;
  struct Task* prev;
};

// Add a new task at the end of the to-do list
void add_task(struct Task** head, const char* description) {
  struct Task* new_task = (struct Task*)malloc(sizeof(struct Task));
  strncpy(new_task->description, description, sizeof(new_task->description) -
      1);
  new_task->description[sizeof(new_task->description) - 1] = '\0';
  new_task->next = NULL;

  if (*head == NULL) {
    new_task->prev = NULL;
    *head = new_task;
  } else {
    struct Task* temp = *head;
```

```
        while (temp->next != NULL) {
            temp = temp->next;
        }
        temp->next = new_task;
        new_task->prev = temp;
    }
}

// Mark a task as done by removing it from the list
void remove_task(struct Task** head, const char* description) {
    if (*head == NULL) {
        printf("No tasks in the list.\n");
        return;
    }

    struct Task* temp = *head;
    while (temp != NULL && strcmp(temp->description, description) != 0) {
        temp = temp->next;
    }

    if (temp == NULL) {
        printf("Task not found: %s\n", description);
        return;
    }

    if (temp->prev != NULL) {
        temp->prev->next = temp->next;
    } else {
        *head = temp->next;
    }

    if (temp->next != NULL) {
        temp->next->prev = temp->prev;
    }

    free(temp);
    printf("Task completed and removed: %s\n", description);
}

// Display all tasks in the to-do list
void display_tasks(struct Task* head) {
    if (head == NULL) {
        printf("No tasks in the to-do list.\n");
        return;
    }

    struct Task* temp = head;
    printf("To-Do List:\n");
    while (temp != NULL) {
        printf("- %s\n", temp->description);
        temp = temp->next;
    }
}
```

```
        int main() {
            struct Task* to_do_list = NULL;

            // Adding tasks to the to-do list
            add_task(&to_do_list, "Buy groceries");
            add_task(&to_do_list, "Finish homework");
            add_task(&to_do_list, "Call mom");

            // Display tasks
            printf("Current To-Do List:\n");
            display_tasks(to_do_list);

            // Completing a task
            remove_task(&to_do_list, "Finish homework");

            // Display tasks after removing one
            printf("\nTo-Do List after completing a task:\n");
            display_tasks(to_do_list);

            // Cleaning up the remaining tasks
            while (to_do_list != NULL) {
                remove_task(&to_do_list, to_do_list->description);
            }

            return 0;
}
```

<u>Output:--</u>



```
Forward display: 20 -> 10 -> 30 -> NULL
Reverse display: 30 -> 10 -> 20 -> NULL
After deleting from beginning: 10 -> 30 -> NULL
After deleting from end: 10 -> NULL
 *  Terminal will be reused by tasks, press any key to close it.
```

4. **Memory Usage and Dynamic Allocation:**

- Use malloc and free to dynamically allocate and deallocate memory.
- Ensure memory is correctly freed after operations to prevent memory leaks.

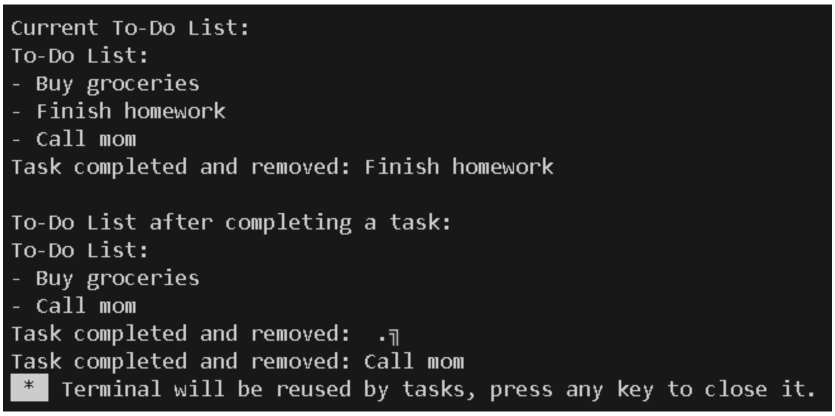
<u>Solution:--</u>

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

// Node structure for each to-do task
struct Task {
char description[100];
struct Task* next;
struct Task* prev;
};

// Add a new task at the end of the to-do list
void add_task(struct Task** head, const char* description) {
struct Task* new_task = (struct Task*)malloc(sizeof(struct Task));
if (new_task == NULL) {
```

```
printf("Memory allocation failed.\n");
return;
}
strncpy(new_task->description, description, sizeof(new_task->description) - 1);
new_task->description[sizeof(new_task->description) - 1] = '\0';
new_task->next = NULL;

if (*head == NULL) {
new_task->prev = NULL;
*head = new_task;
} else {
struct Task* temp = *head;
while (temp->next != NULL) {
temp = temp->next;
}
temp->next = new_task;
new_task->prev = temp;
}
}

// Mark a task as done by removing it from the list
void remove_task(struct Task** head, const char* description) {
if (*head == NULL) {
printf("No tasks in the list.\n");
return;
}

struct Task* temp = *head;
while (temp != NULL && strcmp(temp->description, description) != 0) {
temp = temp->next;
}

if (temp == NULL) {
printf("Task not found: %s\n", description);
return;
}

if (temp->prev != NULL) {
temp->prev->next = temp->next;
} else {
*head = temp->next;
}

if (temp->next != NULL) {
temp->next->prev = temp->prev;
}

free(temp);
printf("Task completed and removed: %s\n", description);
}

// Display all tasks in the to-do list
void display_tasks(struct Task* head) {
if (head == NULL) {
printf("No tasks in the to-do list.\n");
return;
}
```

```
struct Task* temp = head;
printf("To-Do List:\n");
while (temp != NULL) {
printf("- %s\n", temp->description);
temp = temp->next;
}
}

// Free all tasks in the list to prevent memory leaks
void free_all_tasks(struct Task** head) {
struct Task* temp = *head;
while (temp != NULL) {
struct Task* next_task = temp->next;
free(temp);
temp = next_task;
}
*head = NULL;
printf("All tasks have been freed.\n");
}

int main() {
struct Task* to_do_list = NULL;

// Adding tasks to the to-do list
add_task(&to_do_list, "Buy groceries");
add_task(&to_do_list, "Finish homework");
add_task(&to_do_list, "Call mom");

// Display tasks
printf("Current To-Do List:\n");
display_tasks(to_do_list);

// Completing a task
remove_task(&to_do_list, "Finish homework");

// Display tasks after removing one
printf("\nTo-Do List after completing a task:\n");
display_tasks(to_do_list);

// Freeing all tasks at the end to prevent memory leaks
free_all_tasks(&to_do_list);

return 0;
}
```

Output:--



```
Current To-Do List:
To-Do List:
- Buy groceries
- Finish homework
- Call mom
Task completed and removed: Finish homework

To-Do List after completing a task:
To-Do List:
- Buy groceries
- Call mom
Task completed and removed:  .╗
Task completed and removed: Call mom
 *  Terminal will be reused by tasks, press any key to close it.
```

THANK ΨOY…..