

Batch: E2 Roll No.: 16010123325

Experiment No. 6
Grade: AA / AB / BB / BC / CC / CD /DD

Title: Implementation of various types of LL- doubly LL, circular LL, circular doubly LL

Objective: To understand the use of linked list as data structures for various application.

Expected Outcome of Experiment:

CO	Outcome
CO 2	Apply linear and non-linear data structure in application development.

Books/ Journals/ Websites referred:

Types of linked list:

1. Singly Linked List
2. Doubly Linked List
3. Circular Linked List
4. Doubly Circular Linked List

Algorithm for creation, insertion, deletion, traversal and searching an element in assigned linked list type:

- **Create Node:**

- Allocate memory for a new node.
- Set the node's value and point prev and next to itself.

- **Display List:**

- If the list is empty, print a message.
- Start from the head and traverse using next, printing each node's value until returning to the head.

- **Insert at End:**

- Create a new node.
- If the list is empty, set the new node as the head.
- Otherwise, link the new node to the tail and update the head's prev pointer.

- **Delete Node:**

- Traverse the list to find the node with the specified value.
- If found, update the links of neighboring nodes.
- Free the memory of the deleted node.
- If deleting the head, update the head pointer.

- **Search Node:**

- Traverse the list to find a node with the specified value.
- Return the node if found; otherwise, return NULL.

- **Main Menu:**

- Display a menu for user interactions (create list, insert, delete, display, search, exit).
- Process user choices using a switch-case structure to call corresponding functions.

Implementation of the linked list type allotted with snapshot of output (Do not paste the snapshot of code. Copy the complete code here)

Code-

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

typedef struct Node
{
    int val;
    struct Node* prev;
    struct Node* next;
} Node;

Node* createNode(int data) {
    Node* newNode = (Node*)malloc(sizeof(Node));
    newNode->val = data;
    newNode->prev = newNode;
    newNode->next = newNode;
    return newNode;
}

void display(Node* head)
{
    if (head == NULL)
    {
        printf("List is empty.\n");
        return;
    }
    Node* current = head;
    do
    {
        printf("%d <--> ", current->val);
        current = current->next;
    }
    while (current != head);
    printf("(head)\n");
}

Node* insertEnd(Node* head, int n)
{
    Node* newNode = createNode(n);
    if (head == NULL) {
        return newNode;
    }
```

```
    }  
    Node* tail = head->prev;  
    tail->next = newNode;  
    newNode->prev = tail;  
    newNode->next = head;  
    head->prev = newNode;  
    return head;  
}  
  
Node* deleteNode(Node* head, int n)  
{  
    if (head == NULL)  
    {  
        printf("List is empty.\n");  
        return head;  
    }  
    Node* current = head;  
    Node* prev = NULL;  
    do  
    {  
        if (current->val == n)  
        {  
            if (current->next == current)  
            {  
                free(current);  
                return NULL;  
            }  
            if (current == head)  
            {  
                head = head->next;  
            }  
            prev = current->prev;  
            prev->next = current->next;  
            current->next->prev = prev;  
            free(current);  
            return head;  
        }  
        current = current->next;  
    }  
    while (current != head);  
  
    printf("Node with data %d not found.\n", n);  
    return head;  
}  
  
Node* search(Node* head, int n)  
{  
    if (head == NULL)
```

```
{
    printf("List is empty.\n");
    return NULL;
}

Node* current = head;
do
{
    if (current->val == n)
    {
        return current;
    }
    current = current->next;
}
while (current != head);
return NULL;
}

int main()
{
    Node* head = NULL;
    int choice, data;

    while (1) {
        printf("\nMenu:\n");
        printf("1. Create list\n");
        printf("2. Insert at end\n");
        printf("3. Delete node\n");
        printf("4. Display list\n");
        printf("5. Search node\n");
        printf("6. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice)
        {
            case 1:
                printf("Enter data for first node: ");
                scanf("%d", &data);
                head = insertEnd(head, data);
                char ch = 'y';

                while (ch == 'y' || ch == 'Y')
                {
                    printf("Enter data for next node: ");
                    scanf("%d", &data);
                    head = insertEnd(head, data);
                    printf("Do you want to continue? (y/n): ");
                    getchar();
                }
            }
        }
    }
```

```
        scanf("%c", &ch);
    }
    break;

    case 2:
        printf("Enter data to insert: ");
        scanf("%d", &data);
        head = insertEnd(head, data);
        printf("Node inserted.\n");
        break;

    case 3:
        printf("Enter data to delete: ");
        scanf("%d", &data);
        head = deleteNode(head, data);
        break;

    case 4:
        display(head);
        break;

    case 5:
        printf("Enter data to search: ");
        scanf("%d", &data);
        Node* result = search(head, data);
        if (result != NULL)
        {
            printf("Node with data %d found.\n", result->val);
        }
        else
        {
            printf("Node with data %d not found.\n", data);
        }
        break;

    case 6:
        printf("Exiting...\n");
        exit(0);

    default:
        printf("Invalid choice. Please try again.\n");
    }
}

return 0;
}
```

Output:-

```

Menu:
1. Create list
2. Insert at end
3. Delete node
4. Display list
5. Search node
6. Exit
Enter your choice: 1
Enter data for first node: 44
Enter data for next nodes: 55
Do you want to continue? (y/n): n

Menu:
1. Create list
2. Insert at end
3. Delete node
4. Display list
5. Search node
6. Exit
Enter your choice: 2
Enter data to insert: 16
Node inserted.

Menu:
1. Create list
2. Insert at end
3. Delete node
4. Display list
5. Search node
6. Exit
Enter your choice: 4
44 <--> 55 <--> 16 <--> (head)

Menu:
1. Create list
2. Insert at end
3. Delete node
4. Display list
5. Search node
6. Exit
Enter your choice: 3
Enter data to delete: 55

Menu:
1. Create list
2. Insert at end
3. Delete node
4. Display list
5. Search node
6. Exit
Enter your choice: 4
44 <--> 16 <--> (head)

Menu:
1. Create list
2. Insert at end
3. Delete node
  
```

```

5. Search node
6. Exit
Enter your choice: 2
Enter data to insert: 16
Node inserted.

Menu:
1. Create list
2. Insert at end
3. Delete node
4. Display list
5. Search node
6. Exit
Enter your choice: 4
44 <--> 55 <--> 16 <--> (head)

Menu:
1. Create list
2. Insert at end
3. Delete node
4. Display list
5. Search node
6. Exit
Enter your choice: 3
Enter data to delete: 55

Menu:
1. Create list
2. Insert at end
3. Delete node
4. Display list
5. Search node
6. Exit
Enter your choice: 4
44 <--> 16 <--> (head)

Menu:
1. Create list
2. Insert at end
3. Delete node
4. Display list
5. Search node
6. Exit
Enter your choice: 5
Enter data to search: 44
Node with data 44 found.

Menu:
1. Create list
2. Insert at end
3. Delete node
4. Display list
5. Search node
6. Exit
Enter your choice: 6
Exiting...
  
```

Conclusion:-

The above program highlights implementation of a doubly circular linked list in C with insert, delete & search functionalities.

Post lab questions:

1. Compare and contrast Singly Linked List and Doubly Linked List

Criteria	Singly Linked List	Doubly Linked List
Node Structure	Each node has data and a pointer to the next node.	Each node has data, a pointer to the next, and a pointer to the previous node.
Direction of Traversal	Can only be traversed in one direction (forward)	Can be traversed in both directions (forward and backward)
Memory Usage	Requires less memory as each node stores only one pointer	Requires more memory since each node stores two pointers
Traversal Efficiency	Requires $O(n)$ time for traversing from start to end	Allows efficient traversal in both directions, improving flexibility
Complexity	Simpler structure and easier to implement.	More complex structure due to two pointers per node.

2. Write a program to add two large numbers using the doubly linked list or circular linked list

Hint: represent each large number using a linked list

Eg: 123456789898989898 + 34343434343412323 = ?

Code-

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

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

Node* createNode(int data)
{
    Node* newNode = (Node*)malloc(sizeof(Node));
    newNode->data = data;
    newNode->prev = newNode->next = NULL;
    return newNode;
}

void append(Node** head, int data)
{
    Node* newNode = createNode(data);
    if (*head == NULL)
    {
        *head = newNode;
    }
    else
    {
        Node* temp = *head;
        while (temp->next != NULL)
        {
            temp = temp->next;
        }
        temp->next = newNode;
        newNode->prev = temp;
    }
}
```

```
void display(Node* head)
{
    while (head != NULL)
    {
        printf("%d", head->data);
        head = head->next;
    }
    printf("\n");
}

Node* reverse(Node* head)
{
    Node* temp = NULL;
    Node* current = head;
    while (current != NULL)
    {
        temp = current->prev;
        current->prev = current->next;
        current->next = temp;
        current = current->prev;
    }
    if (temp != NULL)
    {
        head = temp->prev;
    }
    return head;
}

Node* addLargeNumbers(Node* first, Node* second)
{
    Node* result = NULL;
    int carry = 0;

    first = reverse(first);
    second = reverse(second);

    while (first != NULL || second != NULL || carry != 0)
    {
        int sum = carry;

        if (first != NULL)
        {
            sum += first->data;
            first = first->next;
        }
    }
}
```

```
        if (second != NULL)
        {
            sum += second->data;
            second = second->next;
        }

        carry = sum / 10;
        append(&result, sum % 10);
    }

    result = reverse(result);

    return result;
}

Node* numberToList(char* num)
{
    Node* head = NULL;
    for (int i = 0; num[i] != '\0'; i++)
    {
        append(&head, num[i] - '0');
    }
    return head;
}

int main()
{
    char num1[] = "123456789898989898";
    char num2[] = "34343434343412323";

    Node* first = numberToList(num1);
    Node* second = numberToList(num2);

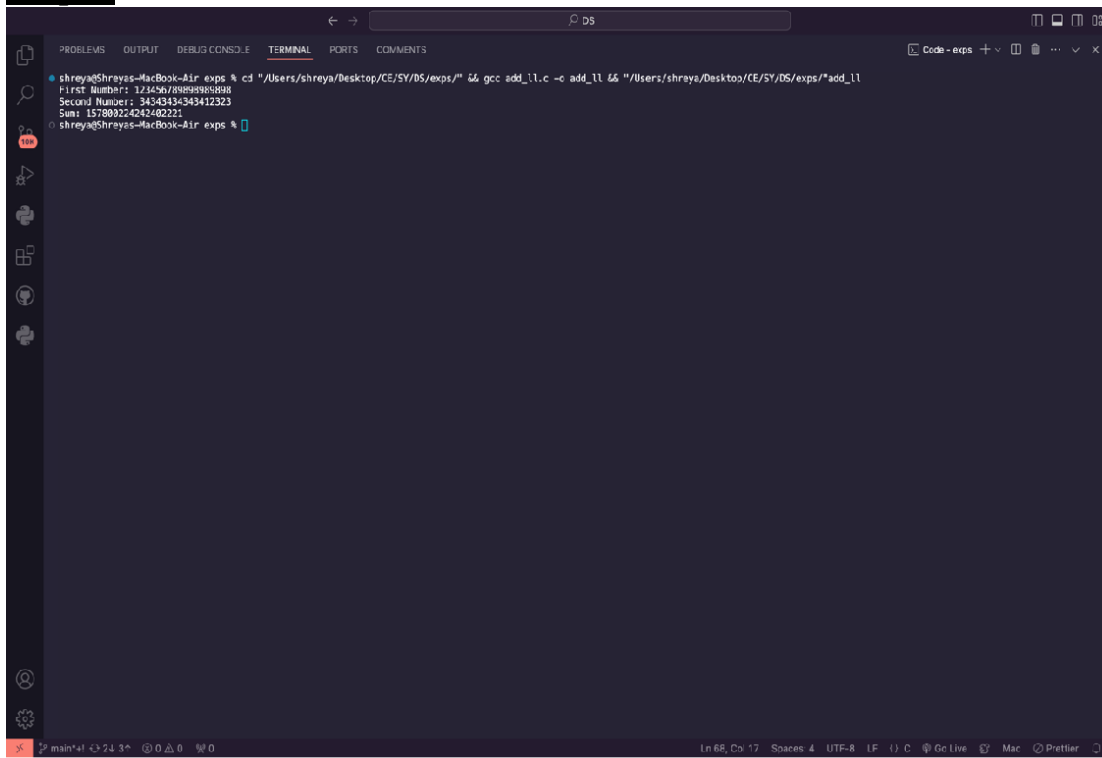
    printf("First Number: ");
    display(first);
    printf("Second Number: ");
    display(second);

    Node* result = addLargeNumbers(first, second);

    printf("Sum: ");
    display(result);

    return 0;
}
```

Output-



```
shreyas@Shreyas-MacBook-Air: ~ % cd "/Users/shreya/Desktop/CE/SY/DS/exps/" && gcc add_11.c -o add_11 && "/Users/shreya/Desktop/CE/SY/DS/exps/"add_11
First Number: 1234567890989098
Second Number: 345434343412323
Sum: 1579992242440221
shreyas@Shreyas-MacBook-Air: ~ %
```