Assignment 5

DSA LAB

2029196

Adarsh Kumar

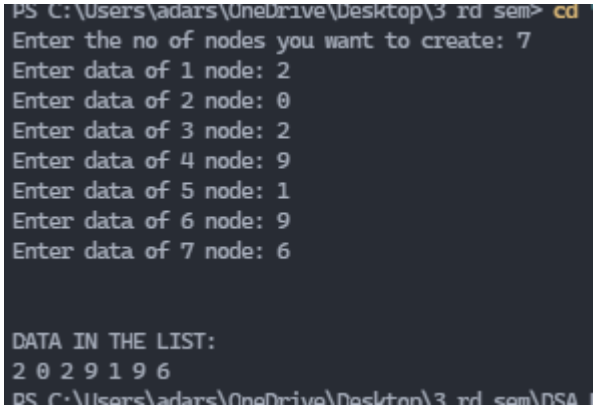
Q1. WAP to create a double linked list of n nodes and display the linked list by using suitable user defined functions for create and display operations.

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

struct nodeDLL
{
    int data;
    struct nodeDLL *prev;
    struct nodeDLL *next;
} *head = NULL, *tail = NULL;

void createDLL(int n)
{
    int i, data;
    struct nodeDLL *newNode;

    if (n >= 1)
    {
        head = malloc(sizeof(struct nodeDLL));

        if (head != NULL)
        {
            printf("Enter data of 1 node: ");
            scanf("%d", &data);

            head->data = data;
            head->prev = NULL;
            head->next = NULL;

            tail = head;
            for (i = 2; i <= n; i++)
            {
                newNode = malloc(sizeof(struct nodeDLL));
                if (newNode != NULL)
```

```
                {
                    printf("Enter data of %d node: ", i);
                    scanf("%d", &data);

                    newNode->data = data;
                    newNode->prev = tail;
                    newNode->next = NULL;

                    tail->next = newNode;
                    tail = newNode;
                }
                else
                    break;
            }
        }
    }
}

void displayDLL()
{
    struct nodeDLL *temp;

    if (head == NULL)
    {
        printf("List is empty.");
    }
    else
    {
        temp = head;
        printf("\n\nDATA IN THE LIST:\n");

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

int main()
{
    int n;
    printf("Enter the no of nodes you want to create: ");
    scanf("%d", &n);
```

```
    createDLL(n);
    displayDLL();

    return 0;
}
```

OUTPUT:-



Q2. WAP to reverse the sequence elements in a double linked list.

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

struct NodeDLL
{
    int data;
    struct NodeDLL *next;
    struct NodeDLL *prev;
} *tail = NULL, *head = NULL;

void createDLL()
{
    int n, data;
    struct NodeDLL *newNode;

    printf("Enter the total number of nodes in list: ");
    scanf("%d", &n);

    head = malloc(sizeof(struct NodeDLL));
    scanf("%d", &data);
    head->data = data;
    head->prev = NULL;
    head->next = NULL;
    tail = head;
    for (int i = 2; i <= n; i++)
    {
```

```
        newNode = malloc(sizeof(struct NodeDLL));
        scanf("%d", &data);
        newNode->data = data;
        newNode->prev = tail;
        newNode->next = NULL;
        tail->next = newNode;
        tail = newNode;
    }
}

void printListDLL()
{
    struct NodeDLL *current = head;
    while (current != NULL)
    {
        printf("%d ", current->data);
        current = current->next;
    }
    printf("\n");
}

void Reverse()
{
    struct NodeDLL *current, *temp;

    current = head;
    while (current != NULL)
    {
        temp = current->next;
        current->next = current->prev;
        current->prev = temp;
        current = temp;
    }
    temp = head;
    head = tail;
    tail = temp;
}

int main()
{

    createDLL();

    printf("Traversal in forward direction \n");
```

```
    printListDLL();
    printf("Traversal in reverse direction \n");
    Reverse();
    printListDLL();
}
```

OUTPUT:-

```
Enter the total number of nodes in list: 7
2 0 2 9 1 9 6
Traversal in forward direction
2 0 2 9 1 9 6
Traversal in reverse direction
6 9 1 9 2 0 2
PS C:\Users\adars\OneDrive\Desktop\3 rd sem\D
```

Q3. Write a menu driven program to perform the following operations in a double linked list by using suitable user defined functions for each case. a) Traverse the list forward, b) Traverse the list backward, c) Check if the list is empty d) Insert a node at the certain position (at beginning/end/any position) e) Delete a node at the certain position (at beginning/end/any position) f) Delete a node for the given key, g) Count the total number of nodes, h) Search for an element in the linked list Verify & validate each function from main method.

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

struct nodeDLL
{
    int data;
    struct nodeDLL *next;
    struct nodeDLL *prev;
} *head = NULL;

void printListForwardDLL();

void insertAtBegining(int newData)
{
    struct nodeDLL *newNode = malloc(sizeof(struct nodeDLL));

    newNode->data = newData;
    newNode->next = head;
    newNode->prev = NULL;
    if (head != NULL)
        head->prev = newNode;
    head = newNode;
}

void insertAtEnd(int newData)
{
```

```
    struct nodeDLL *newNode = malloc(sizeof(struct nodeDLL));
    struct nodeDLL *last = head;

    newNode->data = newData;
    newNode->next = NULL;

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

    while (last->next != NULL)
        last = last->next;

    last->next = newNode;
    newNode->prev = last;
    return;
}

void insertAtIndex(int newData)
{

    int position;
    printf("Enter the index: ");
    scanf("%d", &position);

    struct nodeDLL *newNode, *ptr;

    if (head == NULL)
        printf("Error, List is Empty\n");
    else
    {
        ptr = head;
        int i = 1;
        while (i < position - 1 && ptr != NULL)
        {
            ptr = ptr->next;
            i++;
        }
        if (position == 1)
            insertAtBegining(newData);
        else if (ptr != NULL)
```

```
        {
            newNode = malloc(sizeof(struct nodeDLL));
            newNode->data = newData;
            newNode->next = ptr->next;
            newNode->prev = ptr;

            if (ptr->next != NULL)
                ptr->next->prev = newNode;
            ptr->next = newNode;
        }
        else
            insertAtEnd(newData);
    }
}

void insertInDLL()
{
    int ch, Data;
    printf("\n");
    printf("1. Insert a node at beginning\n");
    printf("2. Insert a node at any position\n");
    printf("3. Insert a node at end\n");
    printf("Enter your choice: ");
    scanf("%d", &ch);
    printf("Enter the Data: ");
    scanf("%d", &Data);
    if (ch == 1)
        insertAtBegining(Data);
    else if (ch == 2)
        insertAtIndex(Data);
    else if (ch == 3)
        insertAtEnd(Data);
    else
        printf("Invalid argument");

    printListForwardDLL();

    printf("\n");
}

void checkDLLForEmpty()
{
    if (head == NULL)
        printf("The List is empty\n\n");
```

```
    else
        printf("The List is Not empty\n\n");
}

void deleteAtBegining()
{
    struct nodeDLL *current;
    if (head == NULL)
        printf("List is empty ERROR Deleting");
    else
    {
        current = head;
        head = head->next;
        head->prev == NULL;
        free(current);
    }
}

void deleteAtIndex()
{
    int position;
    printf("Enter the index to delete: ");
    scanf("%d", &position);

    struct nodeDLL *temp = head;

    for (int i = 0; i < position && temp != NULL; i++)
    {
        temp = temp->next;
    }
    if (temp != NULL)
    {
        temp->prev->next = temp->next;
        temp->next->prev = temp->prev;
        free(temp);
    }
    else
        printf("\nInvalid position!\n");
}

void deleteAtEnd()
{
    struct nodeDLL *current;
    if (head == NULL)
```

```
        printf("List is empty ERROR Deleting");
    else
    {
        current = head;
        while (current->next != 0)
        {
            current->prev = current;
            current = current->next;
        }
        free(current);
        current->prev->next = 0;
    }
}

void deleteInDLL()
{
    int ch;
    printf("\n");
    printf("1. Delete a node at beginning\n");
    printf("2. Delete a node at any position\n");
    printf("3. Delete a node at end\n");
    printf("Enter your choice: ");
    scanf("%d", &ch);
    if (ch == 1)
        deleteAtBegining();
    else if (ch == 2)
        deleteAtIndex();
    else if (ch == 3)
        deleteAtEnd();
    else
        printf("Invalid argument");

    printListForwardDLL();

    printf("\n");
}

void printListForwardDLL()
{
    struct nodeDLL *node = head, *last;
    printf("\nTraversal in forward direction \n");
    while (node != NULL)
    {
        printf(" %d ", node->data);
```

```
        last = node;
        node = node->next;
    }
}

void printListBackwardDLL()
{
    struct nodeDLL *node = head, *last;
    printf("\nTraversal in reverse direction \n");
    while (last != NULL)
    {
        printf(" %d ", last->data);
        last = last->prev;
    }
}

void deleteByKeyInDLL()
{
    int key;
    printf("Enter the key: ");
    scanf("%d", &key);

    struct nodeDLL *temp = head;
    if (head->data == key)
    {
        head = head->next;
        head->prev = NULL;
        temp->next = temp->prev = NULL;
        free(temp);
    }
    else
    {
        while (temp->next->data != key)
            temp = temp->next;
        struct nodeDLL *temp2 = temp->next;
        temp->next = temp2->next;
        temp2->next->prev = temp;
    }

    printListForwardDLL();
    printf("\n");
}

void countNodeInDLL()
```

```
{
    int count = 1;
    while (head->next != NULL)
    {
        head = head->next;
        count++;
    }
    printf("Total number of Nodes are %d\n", count);
}

bool search(int n)
{
    struct nodeDLL *current = head;
    while (current != NULL)
    {
        if (current->data == n)
        {
            return true;
        }
        current = current->next;
    }
    return false;
}

void searchListDLL()
{
    int num;
    printf("Enter the value you want to search: ");
    scanf("%d", &num);

    search(num) ? printf("\nThe value is present\n") : printf("\nThe
 value is not present\n");
}

int main()
{
    int ch;
    while (1)
    {
        printf("\n---LINKED LIST PROGRAMS---\n");
        printf("1. Traverse the list forward\n");
        printf("2. Traverse the list backward\n");
        printf("3. Check if the list is empty\n");
```

```
        printf("4. Insert a node at the certain position (at beginni
ng/end/any position)\n");
        printf("5. Delete a node at the certain position (at beginni
ng/end/any position)\n");
        printf("6. Delete a node for the given key\n");
        printf("7. Count the total number of nodes\n");
        printf("8. Search for an element in the linked list\n");
        printf("9. EXIT\n\n");
        printf("Enter your choice: ");
        scanf("%d", &ch);
        switch (ch)
        {
        case 1:
            printListForwardDLL();
            break;
        case 2:
            printListBackwardDLL();
            break;
        case 3:
            checkDLLForEmpty();
            break;
        case 4:
            insertInDLL();
            break;
        case 5:
            deleteInDLL();
            break;
        case 6:
            deleteByKeyInDLL();
            break;
        case 7:
            countNodeInDLL();
            break;
        case 8:
            searchListDLL();
            break;
        case 9:
            exit(1);
            break;
        default:
            printf("INVALID CHOICE!!");
            break;
        }
    }
```

```
    return 0;
}
```

OUTPUT:-

```
Traversal in forward direction
 2  0  2  9  1  9

---LINKED LIST PROGRAMS---
1. Traverse the list forward
2. Traverse the list backward
3. Check if the list is empty
4. Insert a node at the certain position (at beginning/end/any position)
5. Delete a node at the certain position (at beginning/end/any position)
6. Delete a node for the given key
7. Count the total number of nodes
8. Search for an element in the linked list
9. EXIT

Enter your choice: 4

1. Insert a node at beginning
2. Insert a node at any position
3. Insert a node at end
Enter your choice: 3
Enter the Data: 6

Traversal in forward direction
 2  0  2  9  1  9  6

---LINKED LIST PROGRAMS---
1. Traverse the list forward
2. Traverse the list backward
3. Check if the list is empty
4. Insert a node at the certain position (at beginning/end/any position)
5. Delete a node at the certain position (at beginning/end/any position)
6. Delete a node for the given key
7. Count the total number of nodes
8. Search for an element in the linked list
9. EXIT

Enter your choice: 7
Total number of Nodes are 7
```

Q4. WAP to create a single circular double linked list of n nodes and display the linked list by using suitable user defined functions for create and display operations.

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

struct nodeSCLL
{
    int data;
    struct nodeSCLL *next;
    struct nodeSCLL *prev;
} *head = NULL;

void insert(int newData)
{
    struct nodeSCLL *newNode = malloc(sizeof(struct nodeSCLL));
    struct nodeSCLL *last = head;
```

```
    newNode->data = newData;
    newNode->next = NULL;

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

    while (last->next != NULL)
        last = last->next;

    last->next = newNode;
    newNode->prev = last;
    return;
}

void printList()
{
    struct nodeSCLL *ptr = head;

    printf("The Data in the List is: ");
    if (head != NULL)
    {
        do
        {
            printf("%d ", ptr->data);
            ptr = ptr->next;
        } while (ptr != head);
    }
}

int main()
{
    insert(2);
    insert(0);
    insert(2);
    insert(9);
    insert(1);
    insert(9);
    insert(6);
```

```
    printList();

    return 0;
}
```

OUTPUT:-

```
The Data in the List is: 2 0 2 9 1 9 6
PS C:\Users\adars\OneDrive\Desktop\3 rd s
```

Q5. WAP to remove the duplicates in a sorted double linked list.

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

struct node
{
    int data;
    struct node *previous;
    struct node *next;
};

struct node *head, *tail = NULL;

void addNode(int data)
{
    struct node *newNode = malloc(sizeof(struct node));
    newNode->data = data;

    if (head == NULL)
    {
        head = tail = newNode;
        head->previous = NULL;
        tail->next = NULL;
    }
    else
    {
        tail->next = newNode;
        newNode->previous = tail;
        tail = newNode;
        tail->next = NULL;
    }
}

void sortList()
```

```
{
    struct node *current = NULL, *index = NULL;
    int temp;
    if (head == NULL)
        return;
    else
    {
        for (current = head; current-
>next != NULL; current = current->next)
        {
            for (index = current-
>next; index != NULL; index = index->next)
            {
                if (current->data > index->data)
                {
                    temp = current->data;
                    current->data = index->data;
                    index->data = temp;
                }
            }
        }
    }
}

void removeDuplicateNode()
{
    struct node *current, *index, *temp;

    if (head == NULL)
        return;
    else
    {
        for (current = head; current != NULL; current = current-
>next)
        {
            for (index = current-
>next; index != NULL; index = index->next)
            {
                if (current->data == index->data)
                {
                    temp = index;
                    index->previous->next = index->next;
                    if (index->next != NULL)
                        index->next->previous = index->previous;
```

```
                    temp = NULL;
                }
            }
        }
    }
}

void display()
{
    struct node *current = head;
    if (head == NULL)
    {
        printf("List is empty\n");
        return;
    }
    while (current != NULL)
    {
        printf("%d ", current->data);
        current = current->next;
    }
    printf("\n");
}

int main()
{
    addNode(2);
    addNode(0);
    addNode(2);
    addNode(9);
    addNode(1);
    addNode(9);
    addNode(6);

    printf("Originals list: ");
    display();
    printf("List aftersorting: ");
    sortList();
    display();
    printf("List after removing duplicates: ");
    removeDuplicateNode();
    display();

    return 0;
}
```

OUTPUT:-

```
Originals list: 2 0 2 9 1 9 6
List aftersorting: 0 1 2 2 6 9 9
List after removing duplicates: 0 1 2 6 9
```

Q6. WAP to convert a given singly linked list to a circular list.

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

struct node
{
    int data;
    struct node *next;
} *head = NULL;

void create(int val)
{
    struct node *temp;
    temp = malloc(sizeof(struct node));
    temp->data = val;
    temp->next = head;
    head = temp;
}

void display()
{
    struct node *temp = head;
    printf("Displaying the list elements\n");
    printf("%d ", temp->data);
    temp = temp->next;
    while (head != temp)
    {
        printf("%d ", temp->data);
        temp = temp->next;
    }
    printf("and back to %d %d %d.....\n", temp->data, temp->next->data, temp->next->next->data);
}

void tocircular()
{
    struct node *rear;
```

```
    rear = head;
    while (rear->next != NULL)
    {
        rear = rear->next;
    }
    rear->next = head;
}

int main()
{
    create(6);
    create(9);
    create(1);
    create(9);
    create(2);
    create(0);
    create(2);
    tocircular();
    printf("Circular list generated\n");
    display();

    return 0;
}
```

OUTPUT:-

```
c q6.c -o q6 } ; if ($?) { .\q6 }
Circular list generated
Displaying the list elements
2 0 2 9 1 9 6 and back to 2 0 2..
PS C:\Users\adars\OneDrive\Desktop\3
```

Q7. WAP to implement a doubly linked list by using singly linked

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

struct node
{
    int data;
    struct node *next;
} *head = NULL;

void create(int val)
{
    struct node *temp, *rear;
```

```
    temp = (struct node *)malloc(sizeof(struct node));
    temp->data = val;
    temp->next = NULL;
    if (head == NULL)
    {
        head = temp;
    }
    else
    {
        rear->next = temp;
    }
    rear = temp;
}

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

int main()
{
    create(2);
    create(0);
    create(2);
    create(9);
    create(1);
    create(9);
    create(6);
    printf("Displaying list:\n");
    display();

    return 0;
}
```

OUTPUT:-

```
c q7.c -o q7 } ; i
Displaying list:
2 0 2 9 1 9 6
PS C:\Users\adars\
```

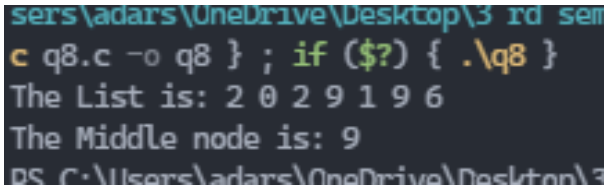
Q8. WAP to print the middle of a double linked list.

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

struct node
{
    int data;
    struct node *next;
    struct node *prev;
} *head = NULL;

void create(int val)
{
    struct node *temp = malloc(sizeof(struct node));
    temp->data = val;
    if (head == NULL)
    {
        head = temp;
        temp->prev = NULL;
        temp->next = NULL;
    }
    else
    {
        struct node *temp1 = head;
        while (temp1->next != NULL)
        {
            temp1 = temp1->next;
        }
        temp->prev = temp1;
        temp1->next = temp;
        temp->next = NULL;
    }
}

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

```
}

void middle()
{
    struct node *temp = head;
    int c = 0;
    while (temp != NULL)
    {
        c++;
        temp = temp->next;
    }
    temp = head;
    int p = 1;
    if (c % 2 == 0)
    {
        int mid = c / 2;
        while (temp != NULL)
        {
            if (p == mid)
                break;
            p++;
            temp = temp->next;
        }
        printf("%d\n", temp->next->data);
    }
    else
    {
        int mid = (c + 1) / 2;
        while (temp != NULL)
        {
            if (p == mid)
                break;
            p++;
            temp = temp->next;
        }
        printf("%d\n", temp->data);
    }
}

int main()
{
    create(2);
    create(0);
    create(2);
```

```
    create(9);
    create(1);
    create(9);
    create(6);
    printf("The List is: \n");
    display();
    printf("The Middle node is: ");
    middle();
    return 0;
}
```

OUTPUT:-



Q9. Given a double linked list, rotate the linked list counter-clockwise by k nodes. Where k is a given positive integer. For example, if the given linked list is 10->20->30->40->50->60 and k is 4, the list should be modified to 50->60->10->20->30->40. Assume that k is smaller than the count of nodes in linked list.

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

struct node
{
    int data;
    struct node *next;
    struct node *prev;
} *head = NULL;

void create(int newData)
{
    struct node *temp = malloc(sizeof(struct node));
    temp->data = newData;
    if (head == NULL)
    {
        head = temp;
        temp->prev = NULL;
        temp->next = NULL;
    }
    else
    {
        struct node *temp1 = head;
        while (temp1->next != NULL)
        {
```

```
            temp1 = temp1->next;
        }
        temp->prev = temp1;
        temp1->next = temp;
        temp->next = NULL;
    }
}
void displayList()
{
    printf("The node is: ");
    struct node *current = head;
    while (current != NULL)
    {
        printf("%d ", current->data);
        current = current->next;
    }
    printf("\n");
}

void modified(int k)
{
    if (k == 0)
        return;
    struct node *temp = head;
    while (temp->next != NULL)
    {
        temp = temp->next;
    }
    temp->next = head;
    head->prev = temp;
    int count = 1;
    while (count <= k)
    {
        head = head->next;
        temp = temp->next;
        count++;
    }
    temp->next = NULL;
    head->prev = NULL;
    displayList();
}
int main()
{
    create(2);
```

```
    create(0);
    create(2);
    create(9);
    create(1);
    create(9);
    create(6);
    displayList();
    int n;
    printf("Enter value of k: ");
    scanf("%d", &n);
    modified(n);
    return 0;
}
```

OUTPUT:-

```
c q9.c -o q9 } ; if ($?) { .\q9
The node is: 2 0 2 9 1 9 6
Enter value of k: 5
The node is: 9 6 2 0 2 9 1
PS C:\Users\adars\OneDrive\Deskt
```