LAB 5 DSA

Q1.[A] Write a menu driven program to perform the following operations in a single linked
list by using suitable user defined functions for each case.
1. Write a program to search an element in a simple linked list, if found delete that node and
insert that node at beginning. Otherwise display an appropriate message.
2. Write a program to find the middle node in a single linked list (without counting the
number of nodes.)
3. Write a program to reverse the first m elements of a linked list of n nodes.
4. Write a program to check whether a given linked list is sorted or not.
5. Given a linked list which is sorted, write a program to insert an element into the linked
list in sorted way.
6. Write a program to find the intersections elements of two linked list and store them in a
third linked list.
7. Write a program to modify the linked list such that all even numbers appear before all the
odd numbers in the modified linked list.
8. Write a program to check whether a singly linked list is a palindrome or not.
9. A linked list is said to contain a cycle if any node is visited more than once while traversing the list. Write a program to detect a cycle in a linked list.
10. Write a program to reverse only even position nodes in a singly linked list.
11. Write a program to swap kth node from beginning with kth node from end in a Linked
List
12. Given a linked list, write a function to reverse every k nodes. (where k is an input to the
function). If a linked list is given as 12->23->45->89->15->67->28->98->NULL and k =
3 then output will be 45->23->12->67->15->89->98->28->NULL.
13. Given a singly 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 *createNode(int value)
{
   struct node *p = (struct node *)malloc(sizeof(struct node));
   p->data = value;
   p->next = NULL;
   return p;
}
void push(struct node **head, int value)
{
   struct node *p = createNode(value);
   struct node *temp;
   if (*head == NULL)
   {
      *head = p;
   }
   else
   {
      temp = *head;
      while (temp->next != NULL)
      {
         temp = temp->next;
      }
      temp->next = p;
   }
}
void traverse(struct node *head)
{
   while (head != NULL)
   {
      printf("%d ", head->data);
      head = head->next;
   }
   printf("\n");
}
int length(struct node **head)
{
   struct node *t = *head;
   int c = 0;
   while (t != NULL)
```

```
    {
        c++;
        t = t->next;
    }
    return c;
}
void condition(struct node **head, int value)
{
    struct node *t = *head;
    struct node *prev;
    struct node *temp;
    while (t->data != value && t != NULL)
    {
        prev = t;
        t = t->next;
    }
    temp = t;
    prev->next = t->next;
    temp->next = *head;
    *head = temp;
}
void middle(struct node **head)
{
    struct node *fast = *head;
    struct node *slow = *head;
    while (fast->next != NULL && fast->next->next != NULL)
    {
        fast = fast->next->next;
        slow = slow->next;
    }

    printf("middle node element is %d\n", slow->data);
}
int checkSort(struct node **head)
{
    struct node *t = *head;
    struct node *q = t->next;
    while (t->next != NULL)
    {
        if (t->data > q->data)
        {
            return 0;
        }
        else
        {
            q = q->next;
        }
```

```
        if (q == NULL)
        {
            t = t->next;
            q = t->next;
        }
    }
    return 1;
}
void insertElement(struct node **head)
{
    struct node *t = *head;
    printf("enter the value ");
    int val;
    scanf("%d", &val);
    struct node *new = createNode(val);
    struct node *prev;
    while (t->next != NULL)
    {
        prev = t;
        t = t->next;
        if (val < t->data && prev->data < val)
        {
            printf("1\n");
            new->next = t;
            prev->next = new;
            break;
        }
    }
    if (t->data <= new->data)
    {
        t->next = new;
    }

    traverse(*head);
}
void insertionTwoll(struct node **first, struct node **second)
{
    struct node *t = *first;
    struct node *t1 = *second;
    struct node *third = NULL;
    struct node *t3 = third;
    while (t->next != NULL || t1->next != NULL)
    {
        if (t->data == t1->data && third == NULL)
        {

            t3->data = t->data;
```

```
            t = t->next;
            t1 = t1->next;
        }

        else if (t->data == t1->data && third != NULL)
        {
            struct node *p = createNode(t->data);
            t3->next = p;
            t3 = p;
            t = t->next;
            t1 = t1->next;
        }
        else
        {
            t = t->next;
            t1 = t1->next;
        }
    }
    traverse(third);
}
void modify1(struct node **head)
{
    struct node *eve = NULL;
    struct node *odd = NULL;
    struct node *t = *head;
    struct node *todd;
    while (t != NULL)
    {
        if (t->data % 2 == 0)
        {
            if (eve == NULL)
            {
                eve = t;
                *head = eve;
            }
            else
            {
                struct node *p = createNode(t->data);
                eve->next = p;
                eve = eve->next;
            }
        }
        else
        {
            if (odd == NULL)
            {
                odd = t;
```

```
            todd = odd;
        }
        else
        {
            struct node *p = createNode(t->data);
            todd->next = p;
            todd = todd->next;
        }
      }
      t = t->next;
   }
   eve->next = odd;
   traverse(*head);
}
void reverseNode(struct node *current, struct node *previous, struct node **head)
{
   if (!current->next)
   {
      *head = current;
      current->next = previous;
      return;
   }
   struct node *next = current->next;
   current->next = previous;
   reverseNode(next, current, head);
}
void palindrom(struct node **head) // wrong
{
   struct node *t = *head;
   struct node *p = *head;
   reverseNode(t, NULL, &t);
   traverse(t);
   printf("\n");
   traverse(p);
   //    while (p!=NULL)
   //    {
   //      if (p->data ==t->data)
   //      {
   //        t=t->next;
   //        p=p->next;
   //      }
   //      else{
   //        return 0;
   //      }

   //    }
   //    return 1;
```

```
}
int cycle(struct node **head)
{
   int n = 3;
   int c = 0;
   while (c == 0)
   {
      struct node *t = *head;
      struct node *prev = t;
      for (int i = 0; i < n - 1; i++)
      {
         t = t->next;
      }
      while (prev->data != t->next->data && t != NULL)
      {
         t = t->next;
         prev = prev->next;
      }
      if (prev->data == t->next->data)
      {
         c++;
      }
      else
         n++;
   }
   if (c > 0)
   {
      return 1;
   }
   else
   {
      return 0;
   }
}
void reverseEven(struct node **head)
{
   struct node *t = *head;
   struct node *eve = NULL;
   struct node *teve;

   while (t != NULL)
   {
      if (t->data % 2 == 0)
      {
         if (eve == NULL)
         {
            eve = t;
```

```
            teve = eve;
        }
        else
        {
            struct node *p = createNode(t->data);
            teve->next = p;
            teve = teve->next;
        }
    }
    t = t->next;
  }
  reverseNode(eve, NULL, &eve);
  traverse(eve);
}
void swapNode(struct node **head, int k1, int k2)
{
  int len = length(head) - k2 + 1;
  struct node *t = *head;
  struct node *j = t;
  for (int i = 0; i < k1; i++)
  {
    t = t->next;
  }
  for (int i = 0; i < len; i++)
  {
    j = j->next;
  }
  int temp = t->data;
  t->data = j->data;
  j->data = temp;
  traverse(*head);
}
struct node *reverseKNode(struct node **head, int k)
{
  struct node *t = *head;
  struct node *rev = t;
  struct node *prev;
  int n = k;
  if (t == NULL)
    return rev;

  while (k > 0 && t != NULL)
  {
    prev = t;
    t = t->next;
    k--;
  }
```

```
    prev->next = NULL;
    int temp = rev->data;
    rev->data = prev->data;
    prev->data = temp;
    struct node *p = reverseKNode(&t, n);
    prev->next = p;
    return *head;
}
struct node *  counterClockwise(struct node ** head,int k)
{
    struct node *t =*head;
    struct node *rev = t;
    struct node *prev;
    while (k > 0 && t != NULL)
    {
        prev = t;
        t = t->next;
        k--;
    }
    prev->next = NULL;
    if (t!=NULL)
    *head =t;
    while (t->next!=NULL)
    {
        t=t->next;
    }
    if (t!=NULL)
    {

    t->next=rev;
    }

    return *head;

}
int main()
{
    struct node *head = NULL;
    push(&head, 1);
    push(&head, 6);
    push(&head, 4);
    push(&head, 2);
    push(&head, 1);
    traverse(head);
    struct node *p = NULL;
    push(&p, 2);
    push(&p, 3);
```

```
    push(&p, 40);
    push(&p, 55);
    push(&p, 61);
    push(&p, 67);
    printf("This is menu drive program \n");

    int a;
    printf("1.search an element ,delete it and add at first\n2. find middle node
element \n3.reverse m element\n4.check sorted or not\n5. insert an element into
the linked list in sorted way\n6.the intersections elements of two linked list and
store them in a third linked list.\n7. program to modify the linked list such that\n8.
palindrom \n9. cycle in ll \n10.reverse only even position nodes\n11. swap kth node
from beginning with kth node from end\n12.reverse every k nodes.\n13.rotate the
linked list counter-clockwise by k nodes.\n14.EXIT\n");
    while (a != 14)
    {
        printf("enter your choice\n");
        scanf("%d", &a);
        switch (a)
        {
        case 1:
            condition(&head, 21);
            traverse(head);
            break;
        case 2:
            middle(&head);
            break;
        case 3:

            break;
        case 4:
            if (checkSort(&head))
            {
                printf("sorted");
            }
            else
                printf("not sorted");

            break;
        case 5:
            insertElement(&p);
            break;
        case 6:
            insertionTwoll(&head, &p);
            break;
        case 7:
            modify1(&p);
```

```
    break;
case 8:
    // if (palindrom(&head))
    // {
    //     printf("the ll is palindromic ");
    // }else{
    //     printf("the ll is not a plaindromic");
    // }
    palindrom(&head);
    break;
case 9:
    if (cycle(&head))
    {
        printf("cycle is present ");
    }
    else
    {
        printf("cycle not present");
    }

    break;
case 10:
    reverseEven(&head);
    break;
case 11:
    printf("enter kth node from beginning");
    int k1;
    scanf("%d", &k1);
    printf("enter kth node from end");
    int k2;
    scanf("%d", &k2);
    swapNode(&head, k1, k2);
    break;
case 12:
    printf("enter no of node ");
    int k;
    scanf("%d", &k);
    traverse(reverseKNode(&head, k));
    break;
case 13:
   printf("enter no of node ");
    int v;
    scanf("%d", &v);
   traverse( counterClockwise(&head,v));
    break;
case 14:
    exit(0);
```

```
        break;
      default:
        printf("you enter a invalid input\n");
        break;
      }
   }
}
```

OUTPUT

1 6 4 2 1
This is menu drive program
1.search an element ,delete it and add at first
2. find middle node element
3.reverse m element
4.check sorted or not
5. insert an element into the linked list in sorted way
6.the intersections elements of two linked list and store them in a third linked list.
7. program to modify the linked list such that
8. palindrom
9. cycle in ll
10.reverse only even position nodes
11. swap kth node from beginning with kth node from end
12.reverse every k nodes.
13.rotate the linked list counter-clockwise by k nodes.
14.EXIT
enter your choice
2
middle node element is 4
enter your choice
3
1 2 4 6 1
enter your choice

[B] Write a program to merge two sorted linked list.

```
#include <stdio.h>
#include <stdlib.h>
struct Node
{
   int data;
   struct Node *next;
};
struct Node *SortedMerge(struct Node *a, struct Node *b)
{
   struct Node *c;

   if (a->data >= b->data)
   {
      c = b;
      b = b->next;
   }
   else
   {
      c = a;
      a = a->next;
   }
   struct Node *head = c;

   while (a != NULL && b != NULL)
   {

      if (a->data >= b->data)
      {
         c->next = b;
         c = c->next;
         b = b->next;
      }
      else
      {
         c->next = a;
         c = c->next;
         a = a->next;
      }
   }
```

```
    if (a == NULL)
    {
        while (b != NULL)
        {
            c->next = b;
            c = c->next;
            b = b->next;
        }
    }
    if (b == NULL)
    {

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

    return head;
}
void push(struct Node **head_ref, int new_data)
{

    struct Node *new_node =
        (struct Node *)malloc(sizeof(struct Node));

    new_node->data = new_data;

    new_node->next = (*head_ref);

    (*head_ref) = new_node;
}


void printList(struct Node *node)
{
    while (node != NULL)
    {
```

```
        printf("%d ", node->data);
        node = node->next;
    }
}

int main()
{

    int n, m;
    printf("enter no of node to be created of first ll");
    scanf("%d", &n);
    printf("enter no of node to be created of second ll");
    scanf("%d", &m);

    /* Start with the empty list */
    struct Node *res = NULL;
    struct Node *a = NULL;
    struct Node *b = NULL;
    int arr[n];
    printf("enter the value of first ll node in sorted form");
    for (int i = 0; i < n; ++i)
    {
        scanf("%d", &arr[i]);
    }

    for (int i = n - 1; i >= 0; i--)
    {
        push(&a, arr[i]);
    }
    printf("enter the value of second ll node in sorted form");
    int arr1[m];
    for (int i = 0; i < m; i++)
    {
        scanf("%d", &arr1[i]);
    }
    for (int i = m - 1; i >= 0; i--)
    {
        push(&b, arr1[i]);
    }
    res = SortedMerge(a, b);
```

```
    printList(res);
    printf("\n");
    return 0;
}
```

OUTPUT

enter no of node to be created of first ll4
enter no of node to be created of second ll4
enter the value of first ll node in sorted form4
5
6
7
enter the value of second ll node in sorted form8
9
10
11
4 5 6 7 8 9 10 11

[C] Write a program to represent a polynomial using linked list. Write a function to add two
polynomials.

```
#include <stdio.h>
#include <stdlib.h>
struct node
{
    int cof;
    int exp;
    struct node *next;
};
struct node *createNode(int cof, int exp)
{
    struct node *p = (struct node *)malloc(sizeof(struct node));
    p->cof = cof;
    p->exp = exp;
    p->next = NULL;
}
void push(struct node **head, int cof, int exp)
{
    struct node *t = *head;
    struct node *p = createNode(cof, exp);
    if (*head == NULL)
```

```
    {
        *head = p;
    }
    else
    {

        struct node *prev = t;
        while (t->next != NULL && t->exp > p->exp)
        {
            prev = t;
            t = t->next;
        }
        if (prev->exp > p->exp)
        {
            t->next = p;
        }
    }
}
void traverse(struct node *head)
{
    while (head != NULL)
    {
        printf("%dx^%d ", head->cof, head->exp);
        head = head->next;
    }
    printf("\n");
}
void add(struct node *head, struct node *second, struct node **third)
{

    struct node *t = *third;
    while (head != NULL && second != NULL)
    {
        if (head->exp == second->exp)
        {
            int sum = head->cof + second->cof;
            push(&t, sum, head->exp);
            head = head->next;
            second = second->next;
        }
```

```
        else
        {
            if (head->exp > second->exp)
            {
                push(&t, head->cof, head->exp);
                head = head->next;
            }
            else
            {
                push(&t, second->cof, second->exp);
                second = second->next;
            }
        }
    }
    if (head == NULL)
    {

        while (second != NULL)
        {
            push(&t, second->cof, second->exp);
            second = second->next;
        }
    }
    if (second == NULL)
    {

        while (head != NULL)
        {
            push(&t, head->cof, head->exp);
            head = head->next;
        }
    }
    traverse(t);
}
int main()
{
    struct node *head = NULL;
    push(&head, 3, 3);
    push(&head, 3, 2);
    push(&head, 2, 1);
```

```
    traverse(head);
    struct node *second = NULL;
    push(&second, 4, 3);
    push(&second, 5, 2);
    push(&second, 8, 1);
    push(&second, 7, 0);
    struct node *third = NULL;
    traverse(second);

    add(head, second, &third);
    return 0;
}
```

OUTPUT

enter no of node to be created of first ll 4
enter no of node to be created of second ll 4
enter the value of first ll node in sorted form 4
5
6
7
enter the value of second ll node in sorted form 8
9
10
11
4 5 6 7 8 9 10 11

[D]Write a program to represent a sparse matrix in three tuple format using an array and perform addition.

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

struct Node
{
    int data;
    int row_pos;
    int col_pos;
    struct Node *next;
};

struct Node *createNode(int d, int r, int c)
```

```
{
   struct Node *n = (struct Node *)malloc(sizeof(struct Node));
   n->data = d;
   n->row_pos = r;
   n->col_pos = c;
   n->next = NULL;
   return n;
}

void createSparse(struct Node **start, int d, int r, int c)
{
   struct Node *n = createNode(d, r, c);
   if (*start == NULL)
      *start = n;
   else
   {
      struct Node *curr_Node = *start;
      while (curr_Node->next != NULL)
         curr_Node = curr_Node->next;
      curr_Node->next = n;
   }
}

void printSparse(struct Node *start)
{
   while (start != NULL)
   {
      printf("%d\t%d\t%d\n", start->row_pos, start->col_pos, start->data);
      start = start->next;
   }
   printf("\n");
}

struct Node *addSparse(struct Node *m1, struct Node *m2)
{
   struct Node *curr_Node1, *curr_Node2, *m3, *curr_Node3;
   curr_Node1 = m1;
   curr_Node2 = m2;
   if (curr_Node1->row_pos < curr_Node2->row_pos)
   {
```

```
      curr_Node3 = createNode(curr_Node1->data, curr_Node1-
>row_pos, curr_Node1->col_pos);
      curr_Node1 = curr_Node1->next;
   }
   else if (curr_Node1->row_pos > curr_Node2->row_pos)
   {
      curr_Node3 = createNode(curr_Node2->data, curr_Node2-
>row_pos, curr_Node2->col_pos);
      curr_Node2 = curr_Node2->next;
   }
   else
   {
      if (curr_Node1->col_pos < curr_Node2->col_pos)
      {
         curr_Node3 = createNode(curr_Node1->data, curr_Node1-
>row_pos, curr_Node1->col_pos);
         curr_Node1 = curr_Node1->next;
      }
      else if (curr_Node1->col_pos > curr_Node2->col_pos)
      {
         curr_Node3 = createNode(curr_Node2->data, curr_Node2-
>row_pos, curr_Node2->col_pos);
         curr_Node2 = curr_Node2->next;
      }
      else
      {
         curr_Node3 = createNode(curr_Node1->data + curr_Node2->data,
curr_Node1->row_pos, curr_Node1->col_pos);
         curr_Node1 = curr_Node1->next;
         curr_Node2 = curr_Node2->next;
      }
   }

   m3 = curr_Node3;

   while (curr_Node1 && curr_Node2)
   {
      if (curr_Node1->row_pos < curr_Node2->row_pos)
      {
```

```
            curr_Node3->next = createNode(curr_Node1->data, curr_Node1-
>row_pos, curr_Node1->col_pos);
            curr_Node1 = curr_Node1->next;
            curr_Node3 = curr_Node3->next;
        }
        else if (curr_Node1->row_pos > curr_Node2->row_pos)
        {
            curr_Node3->next = createNode(curr_Node2->data, curr_Node2-
>row_pos, curr_Node2->col_pos);
            curr_Node2 = curr_Node2->next;
            curr_Node3 = curr_Node3->next;
        }
        else
        {
            if (curr_Node1->col_pos < curr_Node2->col_pos)
            {
                curr_Node3->next = createNode(curr_Node1->data,
curr_Node1->row_pos, curr_Node1->col_pos);
                curr_Node1 = curr_Node1->next;
                curr_Node3 = curr_Node3->next;
            }
            else if (curr_Node1->col_pos > curr_Node2->col_pos)
            {
                curr_Node3->next = createNode(curr_Node2->data,
curr_Node2->row_pos, curr_Node2->col_pos);
                curr_Node2 = curr_Node2->next;
                curr_Node3 = curr_Node3->next;
            }
            else
            {
                curr_Node3->next = createNode(curr_Node1->data +
curr_Node2->data, curr_Node1->row_pos, curr_Node1->col_pos);
                curr_Node1 = curr_Node1->next;
                curr_Node2 = curr_Node2->next;
                curr_Node3 = curr_Node3->next;
            }
        }
    }

    while (curr_Node1)
```

```
    {
        curr_Node3->next = createNode(curr_Node1->data, curr_Node1->row_pos, curr_Node1->col_pos);
        curr_Node1 = curr_Node1->next;
        curr_Node3 = curr_Node3->next;
    }

    while (curr_Node2)
    {
        curr_Node3->next = createNode(curr_Node2->data, curr_Node2->row_pos, curr_Node2->col_pos);
        curr_Node2 = curr_Node2->next;
        curr_Node3 = curr_Node3->next;
    }

    return m3;
}

int main()
{
    int SparseData[5]; // First row of sparse matrix for containing number of rows, columns, and elements with values for matrices 1, 2, and the sum matrix respectively
    SparseData[2] = SparseData[3] = SparseData[4] = 0;
    int d;
    printf("Enter number of rows and columns: ");
    scanf("%d%d", &SparseData[0], &SparseData[1]);
    struct Node *matrix1 = NULL, *matrix2 = NULL;
    printf("Enter the elements for matrix 1:\n");
    for (int i = 0; i < SparseData[0]; i++)
    {
        for (int j = 0; j < SparseData[1]; j++)
        {
            scanf("%d", &d);
            if (d)
            {
                createSparse(&matrix1, d, i, j);
                SparseData[2]++;
            }
        }
```

```
    }
    printf("\nEnter the elements for matrix 2:\n");
    for (int i = 0; i < SparseData[0]; i++)
    {
        for (int j = 0; j < SparseData[1]; j++)
        {
            scanf("%d", &d);
            if (d)
            {
                createSparse(&matrix2, d, i, j);
                SparseData[3]++;
            }
        }
    }
    printf("\nEntered Sparse Matrices are:\n");
    printf("\nSparse Matrix 1:\n");
    printf("Row\tColumn\tValue\n");
    //printf("Row\tColumn\tValue\n%d\t%d\t%d\n", SparseData[0],
SparseData[1], SparseData[2]);
    printSparse(matrix1);
    printf("\nSparse Matrix 2:\n");
    printf("Row\tColumn\tValue\n");
    //printf("Row\tColumn\tValue\n%d\t%d\t%d\n", SparseData[0],
SparseData[1], SparseData[3]);
    printSparse(matrix2);
    struct Node *matrix3 = addSparse(matrix1, matrix2);
    printf("\nResult of summation of the given Sparse Matrices:\n");
    printf("Row\tColumn\tValue\n");
    printSparse(matrix3);
    return 0;
}
```

OUTPUT

Enter number of rows and columns: 3
3
Enter the elements for matrix 1:
2
0
3
4
0

5
0
56
2

Enter the elements for matrix 2:
1
4
5
9
0
5
3
0
0

Entered Sparse Matrices are:

Sparse Matrix 1:

Row	Column	Value
0	0	2
0	2	3
1	0	4
1	2	5
2	1	56
2	2	2

Sparse Matrix 2:

Row	Column	Value
0	0	1
0	1	4
0	2	5
1	0	9
1	2	5
2	0	3

Result of summation of the given Sparse Matrices:

Row	Column	Value
0	0	3
0	1	4
0	2	8
1	0	13
1	2	10
2	0	3
2	1	56
2	2	2

RISHIKESH
2105734
CSE32