Lab Assignment-6

ROLL: 2005535 | NAME: SAHIL SINGH | DATE: 31/08/21

QUES 1: [A] Write a menu driven program to perform the following operations in a double linked list by using suitable user defined functions for each case.

1. Create the list
2. Traverse the list in forward direction
3. Traverse the list in backward direction
4. Add a node after a given data item
5. Add a node before a given data item
6. Delete a node at a given position
7. Add a node a first node
8. Delete the first node
9. Reverse the content of the linked list by traversing each node only once.

SOLUTION:

```c
#include <stdio.h>
#include <stdlib.h>
typedef struct Node
{
    int data;
    struct Node *prev;
    struct Node *next;
} Node;
typedef struct DList
{
    Node *start;
    Node *end;
} DList;
//Utility Functions
Node *create_node(int);
void push_util(Node **, Node **, int);
void display_fwd_util(Node *);
void display_bkwd_util(Node *);
void push_after_util(Node **, Node **, int, int);
void push_before_util(Node **, Node **, int, int);
void pop_at_util(Node **, Node **, int);
void push_first_util(Node **, Node **, int);
void pop_first_util(Node **, Node **);
void reverse_content_util(Node *, Node *);
//Driver Functions
void push(DList *, int);
void display_fwd(DList);
void display_bkwd(DList);
```

```c
void push_after(DList *, int, int);
void push_before(DList *, int, int);
void pop_at(DList *, int);
void push_first(DList *, int);
void pop_first(DList *);
void reverse_content(DList);
int main()
{
    DList dlist = {NULL, NULL};
    int choice, value, pos;
    do
    {
        printf("1) Insert element\n2) Display forward\n3) Display backward\n");
        printf("4) Add node after element\n5) Add node before element\n");
        printf("6) Delete node at position\n7) Insert node as first node\n");
        printf("8) Delete the first node\n9) Reverse list contents\n->: ");
        scanf("%d", &choice);
        switch (choice)
        {
        case 1:
            printf("\nEnter a value: ");
            scanf("%d", &value);
            push(&dlist, value);
            printf("The list: ");
            display_fwd(dlist);
            break;
        case 2:
            printf("\nThe list: ");
            display_fwd(dlist);
            break;
        case 3:
            printf("\nThe list(bkwd): ");
            display_bkwd(dlist);
            break;
        case 4:
            printf("\nEnter a data member after which to insert: ");
            scanf("%d", &pos);
            printf("Enter value to insert: ");
            scanf("%d", &value);
            push_after(&dlist, value, pos);
            printf("The list: ");
            display_fwd(dlist);
            break;
        case 5:
            printf("\nEnter a data member before which to insert: ");
            scanf("%d", &pos);
            printf("Enter value to insert: ");
            scanf("%d", &value);
            push_before(&dlist, value, pos);
            printf("The list: ");
```

```
            display_fwd(dlist);
            break;
        case 6:
            printf("\nEnter the position of node to delete: ");
            scanf("%d", &pos);
            pop_at(&dlist, pos);
            printf("The list: ");
            display_fwd(dlist);
            break;
        case 7:
            printf("\nEnter a value: ");
            scanf("%d", &value);
            push_first(&dlist, value);
            printf("The list: ");
            display_fwd(dlist);
            break;
        case 8:
            printf("\nDeleted...\n");
            pop_first(&dlist);
            printf("The list: ");
            display_fwd(dlist);
            break;
        case 9:
            printf("\nReversed...");
            reverse_content(dlist);
            printf("\nThe list: ");
            display_fwd(dlist);
            break;
        default:
            printf("\nExiting...\n");
        }
        printf("----------------------------\n");
    } while (choice >= 1 && choice <= 9);
    return 0;
}
Node *create_node(int data)
{
    Node *temp = (Node *)malloc(sizeof(Node));
    temp->data = data;
    temp->next = NULL;
    temp->prev = NULL;
    return temp;
}
void push_util(Node **start, Node **end, int data)
{
    Node *temp = create_node(data);
    if (!*start)
    {
        *end = *start = temp;
        return;
```

```
    }
    temp->prev = *end;
    (*end)->next = temp;
    *end = temp;
}
void display_fwd_util(Node *start)
{
    if (start)
        printf("null");
    while (start)
    {
        printf("<-%d->", start->data);
        start = start->next;
    }
    printf("null\n");
}
void display_bkwd_util(Node *end)
{
    if (end)
        printf("null");
    while (end)
    {
        printf("<-%d->", end->data);
        end = end->prev;
    }
    printf("null\n");
}
void push_after_util(Node **start, Node **end, int data, int pos)
{
    Node *temp = create_node(data);
    if (!*start)
    {
        *start = *end = temp;
        return;
    }
    Node *tempStart = *start;
    while (tempStart && tempStart->data != pos)
    {
        tempStart = tempStart->next;
    }
    if (!tempStart || tempStart == *end)
    {
        temp->prev = *end;
        (*end)->next = temp;
        *end = temp;
        return;
    }
    temp->next = tempStart->next;
    tempStart->next->prev = temp;
    temp->prev = tempStart;
```

```
    tempStart->next = temp;
}
void push_before_util(Node **start, Node **end, int data, int pos)
{
    Node *temp = create_node(data);
    if (!*start)
    {
        *start = *end = temp;
        return;
    }
    Node *tempStart = *start;
    while (tempStart && tempStart->data != pos)
    {
        tempStart = tempStart->next;
    }
    if (!tempStart)
    {
        temp->prev = *end;
        (*end)->next = temp;
        *end = temp;
        return;
    }
    else if (tempStart == *start)
    {
        temp->next = *start;
        (*start)->prev = temp;
        *start = temp;
        return;
    }
    temp->prev = tempStart->prev;
    temp->prev->next = temp;
    temp->next = tempStart;
    tempStart->prev = temp;
}
void pop_at_util(Node **start, Node **end, int pos)
{
    Node *tempStart = *start;
    while (tempStart && pos - 1)
    {
        tempStart = tempStart->next;
        pos--;
    }
    if (!tempStart || pos <= 0)
        return;
    else if (tempStart == *start)
    {
        *start = (*start)->next;
        if (!*start)
            *end = NULL;
        else
```

```
            (*start)->prev = NULL;
        free(tempStart);
        return;
    }
    else if (tempStart == *end)
    {
        *end = (*end)->prev;
        if (!*end)
            *start = NULL;
        else
            (*end)->next = NULL;
        free(tempStart);
        return;
    }
    tempStart->next->prev = tempStart->prev;
    tempStart->prev->next = tempStart->next;
    free(tempStart);
}
void push_first_util(Node **start, Node **end, int data)
{
    Node *temp = create_node(data);
    if (!*start)
    {
        *start = *end = temp;
        return;
    }
    (*start)->prev = temp;
    temp->next = *start;
    *start = temp;
}
void pop_first_util(Node **start, Node **end)
{
    if (!*start)
        return;
    Node *ptr = *start;
    *start = (*start)->next;
    if (*start)
        (*start)->prev = NULL;
    else
        *end = NULL;
    free(ptr);
}
void reverse_content_util(Node *start, Node *end)
{
    while (start != end && start->prev != end)
    {
        int temp = start->data;
        start->data = end->data;
        end->data = temp;
        start = start->next;
```

```
        end = end->prev;
    }
}
//////////////////////////////////////////////////////////////////
void push(DList *dlist, int data)
{
    push_util(&dlist->start, &dlist->end, data);
}
void display_fwd(DList dlist)
{
    display_fwd_util(dlist.start);
}
void display_bkwd(DList dlist)
{
    display_bkwd_util(dlist.end);
}
void push_after(DList *dlist, int data, int pos)
{
    push_after_util(&dlist->start, &dlist->end, data, pos);
}
void push_before(DList *dlist, int data, int pos)
{
    push_before_util(&dlist->start, &dlist->end, data, pos);
}
void pop_at(DList *dlist, int pos)
{
    pop_at_util(&dlist->start, &dlist->end, pos);
}
void push_first(DList *dlist, int data)
{
    push_first_util(&dlist->start, &dlist->end, data);
}
void pop_first(DList *dlist)
{
    pop_first_util(&dlist->start, &dlist->end);
}
void reverse_content(DList dlist)
{
    reverse_content_util(dlist.start, dlist.end);
}
```

OUTPUT:

```
1) Insert element
2) Display forward
3) Display backward
4) Add node after element
5) Add node before element
6) Delete node at position
7) Insert node as first node
```

```
8) Delete the first node
9) Reverse list contents
->: 1

Enter a value: 10
The list: null<-10->null
----------------------------
1) Insert element
2) Display forward
3) Display backward
4) Add node after element
5) Add node before element
6) Delete node at position
7) Insert node as first node
8) Delete the first node
9) Reverse list contents
->: 1

Enter a value: 20
The list: null<-10-><-20->null
----------------------------
1) Insert element
2) Display forward
3) Display backward
4) Add node after element
5) Add node before element
6) Delete node at position
7) Insert node as first node
8) Delete the first node
9) Reverse list contents
->: 1

Enter a value: 30
The list: null<-10-><-20-><-30->null
----------------------------
1) Insert element
2) Display forward
3) Display backward
4) Add node after element
5) Add node before element
6) Delete node at position
7) Insert node as first node
8) Delete the first node
9) Reverse list contents
->: 1

Enter a value: 40
The list: null<-10-><-20-><-30-><-40->null
----------------------------
1) Insert element
```

```
2) Display forward
3) Display backward
4) Add node after element
5) Add node before element
6) Delete node at position
7) Insert node as first node
8) Delete the first node
9) Reverse list contents
->: 2

The list: null<-10-><-20-><-30-><-40->null
----------------------------
1) Insert element
2) Display forward
3) Display backward
4) Add node after element
5) Add node before element
6) Delete node at position
7) Insert node as first node
8) Delete the first node
9) Reverse list contents
->: 3

The list(bkwd): null<-40-><-30-><-20-><-10->null
----------------------------
1) Insert element
2) Display forward
3) Display backward
4) Add node after element
5) Add node before element
6) Delete node at position
7) Insert node as first node
8) Delete the first node
9) Reverse list contents
->: 4

Enter a data member after which to insert: 30
Enter value to insert: 35
The list: null<-10-><-20-><-30-><-35-><-40->null
----------------------------
1) Insert element
2) Display forward
3) Display backward
4) Add node after element
5) Add node before element
6) Delete node at position
7) Insert node as first node
8) Delete the first node
9) Reverse list contents
->: 5
```

```
Enter a data member before which to insert: 30
Enter value to insert: 25
The list: null<-10-><-20-><-25-><-30-><-35-><-40->null
----------------------------
1) Insert element
2) Display forward
3) Display backward
4) Add node after element
5) Add node before element
6) Delete node at position
7) Insert node as first node
8) Delete the first node
9) Reverse list contents
->: 6

Enter the position of node to delete: 3
The list: null<-10-><-20-><-30-><-35-><-40->null
----------------------------
1) Insert element
2) Display forward
3) Display backward
4) Add node after element
5) Add node before element
6) Delete node at position
7) Insert node as first node
8) Delete the first node
9) Reverse list contents
->: 7

Enter a value: 5
The list: null<-5-><-10-><-20-><-30-><-35-><-40->null
----------------------------
1) Insert element
2) Display forward
3) Display backward
4) Add node after element
5) Add node before element
6) Delete node at position
9) Reverse list contents
->: 8

Deleted...
The list: null<-10-><-20-><-30-><-35-><-40->null
----------------------------
1) Insert element
2) Display forward
3) Display backward
4) Add node after element
5) Add node before element
```

```
6) Delete node at position
7) Insert node as first node
8) Delete the first node
9) Reverse list contents
->: 9

Reversed...
The list: null<-40-><-35-><-30-><-20-><-10->null
----------------------------
1) Insert element
2) Display forward
3) Display backward
4) Add node after element
5) Add node before element
6) Delete node at position
7) Insert node as first node
8) Delete the first node
9) Reverse list contents
->: 10

Exiting...
----------------------------
```

QUES 2: [B] Write a menu driven program to perform the following operations in a circular linked list by using suitable user defined functions for each case.

1. Create the list
2. Traverse the list
3. Add a node a first node
4. Delete the first node

SOLUTION:

```c
#include <stdio.h>
#include <stdlib.h>
typedef struct Node
{
    int data;
    struct Node *link;
} Node;
void push(Node **, int);
void display(Node *);
void push_first(Node **, int);
void pop_first(Node **);
int main()
{
    Node *list = NULL;
    int choice, value;
    do
```

```c
    {
        printf("1) Insert element\n2) Display element\n");
        printf("3) Insert first node\n4) Delete first node\n5) Exit\n->: ");
        scanf("%d", &choice);
        switch (choice)
        {
        case 1:
            printf("\nEnter a element: ");
            scanf("%d", &value);
            push(&list, value);
            printf("\nThe list: ");
            display(list);
            break;
        case 2:
            printf("\nThe list: ");
            display(list);
            break;
        case 3:
            printf("\nEnter a element: ");
            scanf("%d", &value);
            push_first(&list, value);
            printf("The list: ");
            display(list);
            break;
        case 4:
            pop_first(&list);
            printf("\nThe list: ");
            display(list);
            break;
        default:
            printf("\nExiting...\n");
        }
        printf("-----------------------------\n");
    } while (choice >= 1 && choice <= 4);
    return 0;
}
void push(Node **start, int data)
{
    Node *temp = (Node *)malloc(sizeof(Node));
    temp->data = data;
    temp->link = *start;
    if (!*start)
    {
        *start = temp;
        temp->link = temp;
    }
    Node *tempStart = *start;
    while (tempStart->link != *start)
    {
        tempStart = tempStart->link;
```

```
    }
    tempStart->link = temp;
}
void display(Node *start)
{
    if (!start)
    {
        printf("null\n");
        return;
    }
    Node *tempStart = start;
    do
    {
        printf("%d->", start->data);
        start = start->link;
    } while (start != tempStart);
    printf("(cir)\n");
}
void push_first(Node **start, int data)
{
    Node *temp = (Node *)malloc(sizeof(Node));
    temp->data = data;
    temp->link = *start;
    if (!*start)
    {
        temp->link = temp;
        *start = temp;
    }
    Node *tempStart = *start;
    while (tempStart->link != *start)
    {
        tempStart = tempStart->link;
    }
    *start = temp;
    tempStart->link = *start;
}
void pop_first(Node **start)
{
    if (!*start)
        return;
    else if ((*start)->link == *start)
    {
        free(*start);
        *start = NULL;
        return;
    }
    Node *tempStart = *start;
    while (tempStart->link != *start)
    {
        tempStart = tempStart->link;
```

```
    }
    Node *ptr = *start;
    *start = (*start)->link;
    tempStart->link = *start;
    free(ptr);
}
```

OUTPUT:

```
1) Insert element
2) Display element
3) Insert first node
4) Delete first node
5) Exit
->: 1
Enter a element: 10
The list: 10->(cir)
------------------------------
1) Insert element
2) Display element
3) Insert first node
4) Delete first node
5) Exit
->: 1

Enter a element: 20

The list: 10->20->(cir)
------------------------------
1) Insert element
2) Display element
3) Insert first node
4) Delete first node
5) Exit
->: 1

Enter a element: 30

The list: 10->20->30->(cir)
------------------------------
1) Insert element
2) Display element
3) Insert first node
4) Delete first node
5) Exit
->: 2

The list: 10->20->30->(cir)
------------------------------
```

```
1) Insert element
2) Display element
3) Insert first node
4) Delete first node
5) Exit
->: 3

Enter a element: 5
The list: 5->10->20->30->(cir)
-----------------------------
1) Insert element
2) Display element
3) Insert first node
4) Delete first node
5) Exit
->: 4

The list: 10->20->30->(cir)
-----------------------------
1) Insert element
2) Display element
3) Insert first node
4) Delete first node
5) Exit
->: 5

Exiting...
-----------------------------
```

QUES 3: [C] Write a menu driven program to perform the following operations in a header linked list by using suitable user defined functions for each case. The head node keeps the count of the total number of nodes in the list. The data node keeps the student information: Name, Roll No, CGPA, Address_City, Branch.

1. Create
2. Display student information
3. Display the total number of nodes (in O(1) time)
4. Display the students' details belonging to a particular branch
5. Display the students' details securing > 7.5 CGPA and belonging to a given branch.

SOLUTION:

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
typedef struct Node
{
    char name[20];
    int roll;
```

```c
    float cgpa;
    char city[10];
    char branch[10];
    struct Node *link;
} Node;
typedef struct Header
{
    int count;
    Node *link;
} Header;
void get(Node *);
void put(Node);
Node *create_node(Node);
void push(Header **, Node);
void display(Header *);
int node_count(Header *);
void display_branch(Header *, char *);
void display_cgpa_branch(Header *, char *);
int main()
{
    Header *hlist = NULL; //hlist is the pointer to header node
    Node data;
    int choice;
    do
    {
        printf("1) Insert data\n2) Display data\n3) Count nodes\n");
        printf("4) Search by branch\n5) Above 7.5 CGPA, branch\n6) Exit\n->: ");
        scanf("%d", &choice);
        switch (choice)
        {
        case 1:
            printf("\nEnter following info:\n");
            get(&data);
            push(&hlist, data);
            break;
        case 2:
            printf("\n");
            display(hlist);
            break;
        case 3:
            printf("\nNode count: %d\n", node_count(hlist));
            break;
        case 4:
            printf("\nEnter branch: ");
            scanf(" %[^\n]s", data.branch);
            display_branch(hlist, data.branch);
            break;
        case 5:
            printf("\nEnter branch: ");
            scanf(" %[^\n]s", data.branch);
```

```
            display_cgpa_branch(hlist, data.branch);
            break;
        default:
            printf("\nExiting...\n");
        }
        printf("------------------------------\n");
    } while (choice >= 1 && choice <= 5);
    return 0;
}
void get(Node *data)
{
    printf("Name: ");
    scanf(" %[^\n]s", data->name);
    printf("Roll: ");
    scanf("%d", &data->roll);
    printf("CGPA: ");
    scanf("%f", &data->cgpa);
    printf("City: ");
    scanf(" %[^\n]s", data->city);
    printf("Branch: ");
    scanf(" %[^\n]s", data->branch);
}
void put(Node data)
{
    printf("%s", data.name);
    printf(" %d", data.roll);
    printf(" %0.2f", data.cgpa);
    printf(" %s", data.city);
    printf(" %s\n", data.branch);
}
Node *create_node(Node data)
{
    Node *temp = (Node *)malloc(sizeof(Node));
    strcpy(temp->name, data.name);
    strcpy(temp->city, data.city);
    strcpy(temp->branch, data.branch);
    temp->roll = data.roll;
    temp->cgpa = data.cgpa;
    temp->link = NULL;
    return temp;
}
void push(Header **hlist, Node data)
{
    Node *temp = create_node(data);
    if (!*hlist)
    {
        *hlist = (Header *)malloc(sizeof(Header));
        (*hlist)->count = 1;
        (*hlist)->link = temp;
        return;
```

```
    }
    Node *tempStart = (*hlist)->link;
    while (tempStart->link)
    {
        tempStart = tempStart->link;
    }
    (*hlist)->count++;
    tempStart->link = temp;
}
void display(Header *hlist)
{
    if (!hlist)
    {
        printf("(null)\n");
        return;
    }
    Node *tempStart = hlist->link;
    while (tempStart)
    {
        put(*tempStart);
        tempStart = tempStart->link;
    }
}
int node_count(Header *hlist)
{
    if (!hlist)
        return 0;
    return hlist->count;
}
void display_branch(Header *hlist, char *branch)
{
    if (!hlist)
    {
        printf("(null)\n");
        return;
    }
    Node *tempStart = hlist->link;
    while (tempStart)
    {
        int match = !strcmp(tempStart->branch, branch);
        if (match)
            put(*tempStart);
        tempStart = tempStart->link;
    }
}
void display_cgpa_branch(Header *hlist, char *branch)
{
    if (!hlist)
    {
        printf("(null)\n");
```

```
        return;
    }
    Node *tempStart = hlist->link;
    while (tempStart)
    {
        int match = !strcmp(tempStart->branch, branch);
        if (match && tempStart->cgpa > 7.5)
            put(*tempStart);
        tempStart = tempStart->link;
    }
}
```

OUTPUT:

```
1) Insert data
2) Display data
3) Count nodes
4) Search by branch
5) Above 7.5 CGPA, branch
6) Exit
->: 1

Enter following info:
Name: Sahil
Roll: 2005535
CGPA: 9.44
City: Lucknow
Branch: CSE
--------------------------------
1) Insert data
2) Display data
3) Count nodes
4) Search by branch
5) Above 7.5 CGPA, branch
6) Exit
->: 1

Enter following info:
Name: KIIT
Roll: 9405001
CGPA: 6.39
City: Bhubaneswar
Branch: CSE
--------------------------------
1) Insert data
2) Display data
3) Count nodes
4) Search by branch
5) Above 7.5 CGPA, branch
6) Exit
```

```
->: 2

Sahil 2005535 9.44 Lucknow CSE
KIIT 9405001 6.39 BhubaneswaCSE CSE
------------------------------
1) Insert data
2) Display data
3) Count nodes
4) Search by branch
5) Above 7.5 CGPA, branch
6) Exit
->: 3

Node count: 2
------------------------------
1) Insert data
2) Display data
3) Count nodes
4) Search by branch
5) Above 7.5 CGPA, branch
6) Exit
->: 4

Enter branch: CSE
Sahil 2005535 9.44 Lucknow CSE
KIIT 9405001 6.39 BhubaneswaCSE CSE
------------------------------
1) Insert data
2) Display data
3) Count nodes
4) Search by branch
5) Above 7.5 CGPA, branch
6) Exit
->: 5

Enter branch: CSE
Sahil 2005535 9.44 Lucknow CSE
------------------------------
1) Insert data
2) Display data
3) Count nodes
4) Search by branch
5) Above 7.5 CGPA, branch
6) Exit
->: 6

Exiting...
------------------------------
```

QUES 4: [D] Write a program to represent a sparse matrix in three tuple format using linked list and write addition function to perform addition.

SOLUTION:

```c
#include <stdio.h>
#include <stdlib.h>
typedef struct Node
{
    int row;
    int col;
    int data;
    struct Node *link;
} Node;
void insert(Node **, int, int, int);
int **create_2d_array(int, int);
Node *matrix_to_tuple(int **, int, int);
void print_tuple(Node *);
Node *add_tuple(Node *, Node *);
int main()
{
    int row, col;
    printf("Enter number of rows and columns for the matrix: ");
    scanf("%d%d", &row, &col);
    printf("Enter elements, sparse matrix 1:\n");
    int **arr1 = create_2d_array(row, col);
    printf("\nEnter elements, sparse matrix 2:\n");
    int **arr2 = create_2d_array(row, col);
    Node *tuple1 = matrix_to_tuple(arr1, row, col);
    Node *tuple2 = matrix_to_tuple(arr2, row, col);
    printf("\nTuple 1:\n");
    print_tuple(tuple1);
    printf("\nTuple 2:\n");
    print_tuple(tuple2);
    Node *tuple3 = add_tuple(tuple1, tuple2);
    printf("\nResultant tuple:\n");
    print_tuple(tuple3);
    return 0;
}
void insert(Node **tuple, int row, int col, int data)
{
    Node *temp = (Node *)malloc(sizeof(Node));
    temp->row = row;
    temp->col = col;
    temp->data = data;
    temp->link = NULL;
    if (!*tuple)
    {
        *tuple = temp;
        return;
    }
```

```
    Node *dummyHead = *tuple;
    while (dummyHead->link)
    {
        dummyHead = dummyHead->link;
    }
    (*tuple)->data++;
    dummyHead->link = temp;
}
int **create_2d_array(int row, int col)
{
    int **arr = (int **)malloc(row * sizeof(int *));
    for (int i = 0; i < row; i++)
        arr[i] = (int *)malloc(col * sizeof(int));
    for (int i = 0; i < row; i++)
        for (int j = 0; j < col; j++)
            scanf("%d", &arr[i][j]);
    return arr;
}
Node *matrix_to_tuple(int **arr, int row, int col)
{
    Node *tuple = (Node *)malloc(sizeof(Node));
    tuple->row = row;
    tuple->col = col;
    tuple->data = 0;
    tuple->link = NULL;
    Node *head = tuple;
    for (int i = 0; i < row; i++)
    {
        for (int j = 0; j < col; j++)
            if (arr[i][j] < 0 || arr[i][j] > 0)
            {
                Node *temp = (Node *)malloc(sizeof(Node));
                temp->row = i;
                temp->col = j;
                temp->data = arr[i][j];
                temp->link = NULL;
                tuple->link = temp;
                tuple = temp;
                head->data++;
            }
    }
    return head;
}
void print_tuple(Node *tuple)
{
    while (tuple)
    {
        printf("%d %d %d\n", tuple->row, tuple->col, tuple->data);
        tuple = tuple->link;
    }
```

```
}
Node *add_tuple(Node *tuple1, Node *tuple2)
{
    Node *tuple3 = NULL;
    insert(&tuple3, tuple1->row, tuple1->col, 0);
    tuple1 = tuple1->link;
    tuple2 = tuple2->link;
    while (tuple1 && tuple2)
    {
        if (tuple1->row > tuple2->row)
        {
            insert(&tuple3, tuple2->row, tuple2->col,
                        tuple2->data);
            tuple2 = tuple2->link;
            continue;
        } //that or this
        else if (tuple1->row == tuple2->row)
        {
            if (tuple1->col > tuple2->col)
            {
                insert(&tuple3, tuple2->row, tuple2->col,
                            tuple2->data);
                tuple2 = tuple2->link;
                continue;
            }
        } //Repetation of code to reduce text wrapping in pdf
        else if (tuple1->row < tuple2->row)
        {
            insert(&tuple3, tuple1->row, tuple1->col,
                        tuple1->data);
            tuple1 = tuple1->link;
            continue;
        } //that or this
        else if (tuple1->row == tuple2->row)
        {
            if (tuple1->col < tuple2->col)
            {
                insert(&tuple3, tuple1->row, tuple1->col,
                            tuple1->data);
                tuple1 = tuple1->link;
                continue;
            }
        }
        insert(&tuple3, tuple2->row, tuple2->col,
                    tuple2->data + tuple1->data);
        tuple1 = tuple1->link;
        tuple2 = tuple2->link;
    }
    while (tuple1)
    {
```

```
        insert(&tuple3, tuple1->row, tuple1->col,
                    tuple1->data);
        tuple1 = tuple1->link;
    }
    while (tuple2)
    {
        insert(&tuple3, tuple2->row, tuple2->col,
                    tuple2->data);
        tuple2 = tuple2->link;
    }
    return tuple3;
}
```

OUTPUT:

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

Enter elements, sparse matrix 2:
0 1 2 0
0 1 0 4
0 0 0 0

Tuple 1:
3 4 2
0 2 1
1 1 2

Tuple 2:
3 4 4
0 1 1
0 2 2
1 1 1
1 3 4

Resultant tuple:
3 4 4
0 1 1
0 2 3
1 1 3
1 3 4
```

QUES 5: [E] Write a program to store a polynomial in linked list and write multiplication function to perform multiplication of two polynomials.

SOLUTION:

```c
#include <stdio.h>
#include <stdlib.h>
typedef struct Node
{
    int coff;
    int expo;
    struct Node *link;
} Node;
void push(Node **, int, int);
void display(Node *);
Node *multiply_polynomial(Node *, Node *);
int main()
{
    Node *l1 = NULL;
    Node *l2 = NULL;
    char chos;
    printf("Enter first polynomial:\n");
    do
    {
        int expo, coff;
        printf("Enter coffecient and exponent: ");
        scanf("%d%d", &coff, &expo);
        push(&l1, coff, expo);
        printf("Insert another element?: ");
        scanf(" %c", &chos);
    } while (chos == 'y' || chos == 'Y');
    printf("\nEnter second polynomial:\n");
    do
    {
        int expo, coff;
        printf("Enter coffecient and exponent: ");
        scanf("%d%d", &coff, &expo);
        push(&l2, coff, expo);
        printf("Insert another element?: ");
        scanf(" %c", &chos);
    } while (chos == 'y' || chos == 'Y');
    Node *l3 = multiply_polynomial(l1, l2);
    printf("\nThe resultant polynomial is:\n");
    display(l3);
    return 0;
}
void push(Node **start, int coff, int expo)
{
    /*Making the new node*/
    Node *temp = (Node *)malloc(sizeof(Node));
    temp->coff = coff;
    temp->expo = expo;
    temp->link = NULL;
    /*Connecting the new node*/
    if (!*start)
```

```
    {
        *start = temp;
        return;
    }
    else if ((*start)->expo <= expo)
    {
        temp->link = *start;
        *start = temp;
        return;
    }
    Node *tempStart = *start;
    while (tempStart->link)
    {
        if (tempStart->link->expo <= expo)
            break;
        tempStart = tempStart->link;
    }
    temp->link = tempStart->link;
    tempStart->link = temp;
}
void display(Node *start)
{
    if (!start)
        printf("0");
    while (start)
    {
        if (start->coff > 0)
            printf("+");
        printf("%dx^%d", start->coff, start->expo);
        start = start->link;
    }
    printf("\n");
}
Node *multiply_polynomial(Node *list1, Node *list2)
{
    Node *list3 = NULL;
    Node *ptr = NULL;
    while (list1)
    {
        ptr = list2;
        while (ptr)
        {
            int coff = list1->coff * ptr->coff;
            int expo = list1->expo + ptr->expo;
            push(&list3, coff, expo);
            ptr = ptr->link;
        }
        list1 = list1->link;
    }
    /*Removing duplicate elements*/
```

```
    ptr = list3;
    Node *ptr2 = NULL;
    while (ptr && ptr->link)
    {
        ptr2 = ptr;
        while (ptr2->link)
        {
            if (ptr->expo == ptr2->link->expo)
            {
                ptr->coff = ptr->coff + ptr2->link->coff;
                //Next element is a duplicate (sorted insertion)
                Node *temp = ptr2->link;
                ptr2->link = ptr2->link->link;
                free(temp);
                continue;
            }
            ptr2 = ptr2->link;
        }
        ptr = ptr->link;
    }
    return list3;
}
```

OUTPUT:

```
Enter first polynomial:
Enter coffecient and exponent: 1 1
Insert another element?: y
Enter coffecient and exponent: 2 2
Insert another element?: y
Enter coffecient and exponent: 5 3
Insert another element?: n

Enter second polynomial:
Enter coffecient and exponent: 2 1
Insert another element?: y
Enter coffecient and exponent: 3 2
Insert another element?: y
Enter coffecient and exponent: 4 3
Insert another element?: n

The resultant polynomial is:
+20x^6+23x^5+20x^4+7x^3+2x^2
```