DS ASSIGNMENT - 1

Mahika Gupta
PES1UG20CS243

1. Header file:

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

int sparse_matrix[10][10];
int sr,sc,er,ec;

typedef struct Node
{
   int value;
   int row_position;
   int column_postion;
   struct Node *link;
}NODE;

typedef struct Nodemove
{
   int row_position;
   int column_postion;
   struct Nodemove *rlink;
   struct Nodemove *llink;
}NODEM;

typedef struct list
{
   NODE *head;
}LIST;
typedef struct listmove
{
   NODEM *head;
}LISTM;
```

```
void initlist(LIST* p);
void initlistm(LISTM* p);
void create_sparsematrix(LIST* spm, int ele,int row, int column);
void create(LISTM* spm,int row, int column);
void display(LISTM* spm);
void read(LIST* spm);
void movement(LIST* spm,LISTM* move);
```

2. Client File:

```
#include "PES1UG20CS243_H.h"

int main()
{
   LIST spm;
   initlist(&spm);
   LISTM move;
   initlistm(&move);
   read(&spm);
   movement(&spm,&move);
   display(&move);
   return 0;
}
```

3. Server File:

```
#include"PES1UG20CS243_H.h"

void initlist(LIST* p)
{
   p->head=NULL;
}
void initlistm(LISTM* p)
{
   p->head=NULL;
}
NODE* getnode(int ele,int r,int c)
{
   NODE* temp = (NODE*)malloc(sizeof(NODE));
   temp->value = ele;
   temp->link= NULL;
```

```
   temp->row_position = r;
   temp->column_postion = c;
   return temp;
}
NODEM* getnodem(int r,int c)
{
   NODEM* temp = (NODEM*)malloc(sizeof(NODEM));
   temp->rlink= NULL;
   temp->llink= NULL;
   temp->row_position = r;
   temp->column_postion = c;
   return temp;
}
void create_sparsematrix(LIST* spm, int ele,int row, int column)
{
   NODE* temp, *pres;
   pres=spm->head;
   temp = getnode(ele,row,column);
   if (spm->head == NULL)
   {
       spm->head=temp;
   }
   else
   {
       while (pres->link != NULL)
       pres = pres->link;
       pres->link=temp;
   }
}
void create(LISTM* spm,int row, int column)
{
   NODEM* temp =getnodem(row,column);
   NODEM* pres=spm->head;
   if(pres==NULL)
   {
       spm->head=temp;
   }
   else if (pres->rlink == NULL)
   {
       temp->llink=pres;
       pres->rlink=temp;
   }
```

```
    else
    {
        while (pres->rlink != NULL)
            pres = pres->rlink;
        temp->llink=pres;
        pres->rlink=temp;
    }
}
void read(LIST* spm)
{
    FILE *ptr;
    ptr = fopen("C:\\nisarga\\CS\\DSA\\input.txt","r");
    if(ptr == NULL)
    {
        printf("Error!");
        exit(1);
    }
    fscanf(ptr,"%d%d%d%d",&sr,&sc,&er,&ec);
    for (int i = 0; i < er+1; i++)
    {
        for (int j = 0; j < ec+1; j++)
        {
            char c;
            if (fscanf(ptr, " %c", &c) != 1)
            printf("…report read failure and exit…");
            else if (isdigit((unsigned char)c))
            sparse_matrix[i][j] = c - '0';
            else
            sparse_matrix[i][j] = 0;
        }
    }
    for (int i = 0; i < er+1; i++)
    {
        for (int j = 0; j < ec+1; j++)
        {
            if (sparse_matrix[i][j]!=0)
            {
            create_sparsematrix(spm, 1, i, j);
            }
        }
    }
    fclose(ptr);
```

```
}
void delete_key(LISTM* p,int r,int c)
{
   NODEM *pres = p->head;

   if(p->head==NULL)
       printf("Empty list\n");
   else
   {
       while(pres!=NULL)
       {
           if(pres->row_position==r && pres->column_postion==c)
           {
               NODEM *temp = pres->rlink;
               if (pres==p->head)
               {
                   p->head=pres->rlink;
                   pres->rlink=NULL;
                   free(pres);
               }
               else if(pres->rlink==NULL)
               {
                   pres->llink->rlink = NULL;
                   pres->llink=NULL;
                   free(pres);
               }
           else
           {
               pres->llink->rlink = pres->rlink;
               pres->rlink->llink = pres->llink;
               pres->llink=NULL;
               pres->rlink=NULL;
               free(pres);
           }
           pres = temp;
       }
       else
           pres = pres->rlink;
       }

   }
}
```

```
void movement(LIST* spm,LISTM* move)
{
    int posr=0,posc=0;
    create(move,0,0);
    NODEM* m=move->head;
    NODE* t=spm->head;
    while (posr<(er+1) && posc<(ec+1))
    {
        if (posr==t->row_position && (posc+1)==t->column_postion)
        {
            posr=posr+1;
            while (t->row_position!=posr)
            {
                t=t->link;
            }
            while (t->column_postion<posc)
            {
                t=t->link;
            }
        if((t->row_position==posr)&&(t->column_postion==posc))
        {
            m=m->llink;
            delete_key(move,posr-1,posc);
            posc=m->column_postion-1;
        }
        else
        {
            posc=posc-1;
        }
    }
    else if ((posc+1)==ec && posr!=t->row_position)
    {
        posr=posr+1;
        while (t->row_position!=posr)
        {
            t=t->link;
        }
        while (t->column_postion<posc)
        {
            t=t->link;
        }
        posc=t->column_postion-1;
```

```
    }
    posc=posc+1;
    create(move,posr,posc);
    m=move->head;
    while (m->rlink!=NULL)
    {
        m=m->rlink;
    }
    if (posr==er && posc==ec)
    {
        display(move);
        break;
    }
    }
    if (posr>er && posc>ec)
    {
        FILE* ptr = fopen("out.txt","w");
        fprintf(ptr,"%d",-1);
        fclose(ptr);
    }
}
void display(LISTM* spm)
{
    NODEM *pres = spm->head;
    FILE* ptr = fopen("out.txt","w");
    while(pres != NULL)
    {
        fprintf(ptr,"%d %d\n",pres->row_position,pres->column_postion);
        pres = pres->rlink;
    }
    fclose(ptr);
}
```

4. Output File:

out.txt — Edited

```
0 0
0 1
0 2
0 3
0 4
1 4
1 5
2 5
2 6
3 6
3 7
4 7
5 7
6 7
7 7
8 7
8 8
8 9
9 9
```