Q1. WAP to create a linked list that represents a polynomial expression with single variable (i.e., $5x^7-3x^5+x^2+9$) and display the polynomial by using user defined functions for creation and display.

```
#include <stdio.h>
#include <stdlib.h>
typedef struct link
{
  int coeff;
  int pow;
  struct link *next;
} my_poly;

void create(my_poly **);
void display(my_poly *);

int main(void)
{
  int ch;
  do
  {
    my_poly *poly1, ;

    printf("\nCreate 1st expression\n");
    create(&poly1);
    printf("\nStored the 1st expression");
    display(poly1);

    printf("\nAdd two more expressions? (Y = 1/N = 0): ");
    scanf("%d", &ch);
  } while (ch);
  return 0;
}

void create(my_poly **node)
{
  int flag;
  int coeff, pow;
  my_poly *tmp;
  tmp = (my_poly *)malloc(sizeof(my_poly));
  *node = tmp;
  do
  {

    printf("\nEnter Coeff:");
    scanf("%d", &coeff);
    tmp->coeff = coeff;
    printf("\nEnter Pow:");
    scanf("%d", &pow);
    tmp->pow = pow;
```

```
    tmp->next = NULL;
    printf("\nContinue adding more terms to the polynomial list?(Y = 1/N = 0): ");
    scanf("%d", &flag);
    if (flag)
    {
      tmp->next = (my_poly *)malloc(sizeof(my_poly));
      tmp = tmp->next;
      tmp->next = NULL;
    }
  } while (flag);
}

void display(my_poly *node)
{
  printf("\nThe polynomial expression is:\n");
  while (node != NULL)
  {
    printf("%dx^%d", node->coeff, node->pow);
    node = node->next;
    if (node != NULL)
      printf(" + ");
  }
}
```

OUTPUT:-

```
Create 1st expression

Enter Coeff:5

Enter Pow:2

Continue adding more terms to the polynomial list?(Y = 1/N = 0): 1

Enter Coeff:3

Enter Pow:1

Continue adding more terms to the polynomial list?(Y = 1/N = 0): 1

Enter Coeff:1

Enter Pow:0

Continue adding more terms to the polynomial list?(Y = 1/N = 0): 0

Stored the 1st expression
The polynomial expression is:
5x^2 + 3x^1 + 1x^0
Add two more expressions? (Y = 1/N = 0): 0
```

Q2. WAP by modifying the LA1 program to add two polynomials with single variable. Use the same function in LA1 written for creation & display operations and write a new function for addition operations.

```
#include <stdio.h>
#include <stdlib.h>
typedef struct link
{
  int coeff;
  int pow;
  struct link *next;
} my_poly;

void create(my_poly **);
void display(my_poly *);
void add(my_poly **, my_poly *, my_poly *);

int main(void)
{
  int ch;
  do
  {
    my_poly *poly1, *poly2, *poly3;

    printf("\nCreate 1st expression\n");
    create(&poly1);
    printf("\nStored the 1st expression");
    display(poly1);

    printf("\nCreate 2nd expression\n");
    create(&poly2);
    printf("\nStored the 2nd expression");
    display(poly2);

    add(&poly3, poly1, poly2);
    display(poly3);

    printf("\nAdd two more expressions? (Y = 1/N = 0): ");
    scanf("%d", &ch);
  } while (ch);
  return 0;
}

void create(my_poly **node)
{
  int flag;
  int coeff, pow;
  my_poly *tmp;
  tmp = (my_poly *)malloc(sizeof(my_poly));
```

```
  *node = tmp;
  do
  {

    printf("\nEnter Coeff:");
    scanf("%d", &coeff);
    tmp->coeff = coeff;
    printf("\nEnter Pow:");
    scanf("%d", &pow);
    tmp->pow = pow;

    tmp->next = NULL;
    printf("\nContinue adding more terms to the polynomial list?(Y = 1/N = 0): ");
    scanf("%d", &flag);
    if (flag)
    {
      tmp->next = (my_poly *)malloc(sizeof(my_poly));
      tmp = tmp->next;
      tmp->next = NULL;
    }
  } while (flag);
}

void display(my_poly *node)
{
  printf("\nThe polynomial expression is:\n");
  while (node != NULL)
  {
    printf("%dx^%d", node->coeff, node->pow);
    node = node->next;
    if (node != NULL)
      printf(" + ");
  }
}

void add(my_poly **result, my_poly *poly1, my_poly *poly2)
{
  my_poly *tmp;
  tmp = (my_poly *)malloc(sizeof(my_poly));
  tmp->next = NULL;
  *result = tmp;

  while (poly1 && poly2)
  {
    if (poly1->pow > poly2->pow)
    {
      tmp->pow = poly1->pow;
      tmp->coeff = poly1->coeff;
      poly1 = poly1->next;
```

```
    }
    else if (poly1->pow < poly2->pow)
    {
      tmp->pow = poly2->pow;
      tmp->coeff = poly2->coeff;
      poly2 = poly2->next;
    }
    else
    {
      tmp->pow = poly1->pow;
      tmp->coeff = poly1->coeff + poly2->coeff;
      poly1 = poly1->next;
      poly2 = poly2->next;
    }
    if (poly1 && poly2)
    {
      tmp->next = (my_poly *)malloc(sizeof(my_poly));
      tmp = tmp->next;
      tmp->next = NULL;
    }
  }
  while (poly1 || poly2)
  {
    tmp->next = (my_poly *)malloc(sizeof(my_poly));
    tmp = tmp->next;
    tmp->next = NULL;

    if (poly1)
    {
      tmp->pow = poly1->pow;
      tmp->coeff = poly1->coeff;
      poly1 = poly1->next;
    }
    if (poly2)
    {
      tmp->pow = poly2->pow;
      tmp->coeff = poly2->coeff;
      poly2 = poly2->next;
    }
  }
  printf("\nAddition Complete");
}
```

OUTPUT :-

```
Create 1st expression

Enter Coeff:5

Enter Pow:2
```

```
Continue adding more terms to the polynomial list?(Y = 1/N = 0): 1

Enter Coeff:3

Enter Pow:1

Continue adding more terms to the polynomial list?(Y = 1/N = 0): 1

Enter Coeff:1

Enter Pow:0

Continue adding more terms to the polynomial list?(Y = 1/N = 0): 0

Stored the 1st expression
The polynomial expression is:
5x^2 + 3x^1 + 1x^0
Create 2nd expression

Enter Coeff:6

Enter Pow:
2

Continue adding more terms to the polynomial list?(Y = 1/N = 0): 1

Enter Coeff:
4

Enter Pow:1

Continue adding more terms to the polynomial list?(Y = 1/N = 0): 1

Enter Coeff:2

Enter Pow:0

Continue adding more terms to the polynomial list?(Y = 1/N = 0): 0

Stored the 2nd expression
The polynomial expression is:
6x^2 + 4x^1 + 2x^0
Addition Complete
The polynomial expression is:
11x^2 + 7x^1 + 3x^0
Add two more expressions? (Y = 1/N = 0): 0
```

Q3. A matrix m × n that has relatively few non-zero entries is called sparse matrix. It may be represented in much less than m × n space. An m × n matrix with k non-zero entries is sparse if k << m × n. It may be faster to represent the matrix compactly as a list of the non-zero indexes and associated entries. WAP to represent a sparse matrix using linked list.

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

struct node
{
    int row, col, val;
};

struct colHead
{
    int col;
    struct colHead *next;
};

struct rowHead
{
    int row;
    struct rowHead *next;
};

struct sparsehead
{
    int rowCount, colCount;
    struct rowHead *frow;
    struct colHead *fcol;
};

struct sparse
{
    int row, *data;
    struct sparsehead *smatrix;
    struct rowHead **rowPtr;
    struct colHead **colPtr;
};

int count = 0;

void initialize(struct sparse *sPtr, int row, int col)
{
    int i;
    sPtr->rowPtr = (struct rowHead **)calloc(1, (sizeof(struct rowHead) * row));
    sPtr->colPtr = (struct colHead **)calloc(1, (sizeof(struct colHead) * col));
```

```
    for (i = 0; i < row; i++)
        sPtr->rowPtr[i] = (struct rowHead *)calloc(1, sizeof(struct rowHead));

    for (i = 0; i < row - 1; i++)
    {
        sPtr->rowPtr[i]->row = i;
        sPtr->rowPtr[i]->next = sPtr->rowPtr[i + 1];
    }

    for (i = 0; i < col; i++)
        sPtr->colPtr[i] = (struct colHead *)calloc(1, sizeof(struct colHead));

    for (i = 0; i < col - 1; i++)
    {
        sPtr->colPtr[i]->col = i;
        sPtr->colPtr[i]->next = sPtr->colPtr[i + 1];
    }
    sPtr->smatrix = (struct sparsehead *)calloc(1, sizeof(struct sparsehead));
    sPtr->smatrix->rowCount = row;
    sPtr->smatrix->colCount = col;
    sPtr->smatrix->frow = sPtr->rowPtr[0];
    sPtr->smatrix->fcol = sPtr->colPtr[0];
    return;
}
void inputMatrix(struct sparse *sPtr, int row, int col)
{
    int i, n, x = 0, y = 0;
    n = row * col;
    sPtr->data = (int *)malloc(sizeof(int) * n);
    for (i = 0; i < n; i++)
    {
        if (y != 0 && y % col == 0)
        {
            x++;
            y = 0;
        }
        printf("data[%d][%d] : ", x, y);
        scanf("%d", &(sPtr->data[i]));
        if (sPtr->data[i])
            count++;
        y++;
    }
    return;
}
void displayInputMatrix(struct sparse s, int row, int col)
{
    int i;
    for (i = 0; i < row * col; i++)
    {
```

```
        if (i % col == 0)
            printf("\n");
        printf("%d ", s.data[i]);
    }
    printf("\n");
    return;
}
```

OUTPUT:-

```
Enter the rows and columns:3
3
data[0][0] : 1
data[0][1] : 0
data[0][2] : 0
data[1][0] : 1
data[1][1] : 0
data[1][2] : 0
data[2][0] : 1
data[2][1] : 0
data[2][2] : 0
Given Sparse Matrix has 3 non-zero elements
Input Sparse Matrix:

1 0 0
1 0 0
1 0 0
```

Q4. WAP to find out the transpose of a sparse matrix.

```
#include <stdio.h>

#define MAX 20

void printsparse(int[][3]);
void readsparse(int[][3]);
void transpose(int[][3], int[][3]);

int main()
{
    int b1[MAX][3], b2[MAX][3], m, n;
    printf("Enter the size of matrix (rows,columns):");
    scanf("%d%d", &m, &n);
    b1[0][0] = m;
    b1[0][1] = n;
    readsparse(b1);
    transpose(b1, b2);
    printsparse(b2);
}

void readsparse(int b[MAX][3])
```

```
{
    int i, t;
    printf("\nEnter no. of non-zero elements:");
    scanf("%d", &t);
    b[0][2] = t;
    for (i = 1; i <= t; i++)
    {
        printf("\nEnter the next triple(row,column,value):");
        scanf("%d%d%d", &b[i][0], &b[i][1], &b[i][2]);
    }
}

void printsparse(int b[MAX][3])
{
    int i, n;
    n = b[0][2];\
    printf("\nAfter Transpose:\n");
    printf("\nrow\t\tcolumn\t\tvalue\n");
    for (i = 0; i <= n; i++)
        printf("%d\t\t%d\t\t%d\n", b[i][0], b[i][1], b[i][2]);
}

void transpose(int b1[][3], int b2[][3])
{
    int i, j, k, n;
    b2[0][0] = b1[0][1];
    b2[0][1] = b1[0][0];
    b2[0][2] = b1[0][2];
    k = 1;
    n = b1[0][2];
    for (i = 0; i < b1[0][1]; i++)
        for (j = 1; j <= n; j++)
            if (i == b1[j][1])
            {
                b2[k][0] = i;
                b2[k][1] = b1[j][0];
                b2[k][2] = b1[j][2];
                k++;
            }
}
```

OUTPUT:-

```
Enter the size of matrix (rows,columns):3 3

Enter no. of non-zero elements:3

Enter the next triple(row,column,value):1 1 2

Enter the next triple(row,column,value):2 3 4
```

```
Enter the next triple(row,column,value):2 1 3

After Transpose:
row             column          value
3               3               3
1               1               2
1               2               3
```

Q5. WAP to determine whether the given matrix is a sparse matrix or not.

```
int main()
{
    struct sparse input, output;
    int row, col, spcheck;
    printf("Enter the rows and columns:");
    scanf("%d%d", &row, &col);
    initialize(&input, row, col);
    initialize(&output, row, col);
    inputMatrix(&input, row, col);
    printf("Given Sparse Matrix has %d non-zero elements\n", count);
    spcheck = 0.5*row*col;
    if(count < spcheck)
    {
    printf("Input Sparse Matrix:\n");
    displayInputMatrix(input, row, col);
    }
    else{
        printf("Entered Matrix is not sparse matrix");
        displayInputMatrix(input, row, col);
    }
    printf("\n");
    return 0;
}



#include <stdio.h>

void main ()
{
    int matrix[10][10];
    int i, j, m, n;
    int sparse_counter = 0;

    printf("Enter the order of the matix \n");
    scanf("%d %d", &m, &n);
    printf("Enter the elements of the matix \n");
    for (i = 0; i < m; ++i)
    {
        for (j = 0; j < n; ++j)
```

```
        {
            scanf("%d", &matrix[i][j]);
            if (matrix[i][j] == 0)
            {
                ++sparse_counter;
            }
        }
    }
    if (sparse_counter > ((m * n) / 2))
    {
        printf("The given matrix is Sparse Matrix !!! \n");
    }
    else
        printf("The given matrix is not a Sparse Matrix \n");
    printf("There are %d number of Zeros.", sparse_counter);
}
```

OUTPUT:-

```
Enter the order of the matix
3
3
Enter the elements of the matix
1
0
0
0
1
0
1
0
0
The given matrix is Sparse Matrix !!!
There are 6 number of Zeros.
```