**Practice Module 1**

1. Write a C program for addition of two polynomials represented using circular linked list.
2. Write a C program for subtraction of one Polynomial from another using doubly linked list.
3. Write a C program for the evaluation a polynomial for some real value of x stored in a doubly circular linked list.
4. Write a C program for the addition of two 2-variable Polynomials. The polynomials are stored in the circular linked lists.
5. Write a C program for the removing all the terms with even exponents from a single variable polynomial stored in a circular linked list.
6. Write a C program for the removing all the terms with odd coefficients from a single variable polynomial stored in a doubly linked list.
7. Write a C program for the removing all the terms with even exponents (summation of x exponent and y exponent) from a 2-variable polynomial stored in a doubly circular linked list.
8. Write a C program for the multiplication of two polynomials represented using circular linked list.
9. Write a C program for the multiplication of two 2-variable Polynomials. The polynomials are stored in the singly linked lists.
10. Write a C program for computing the derivative of the polynomial stored in a circular linked list.
11. Write a C Program for computing the integration of the polynomial stored in a doubly linked list.
12. Write a C Program for computing the Partial differentiation of a 2-variable polynomial stored in a doubly linked list. Write a C program for implementing polynomial Division stored a circular linked list.
13. Write a C program for computing the transpose of a sparse matrix stored in a circular linked list.
14. Write a C program for multiplication of two sparse matrix represented using doubly linked list.

#include <stdio.h>

#include <stdlib.h>

// Node structure for doubly linked list

struct Node {

    int row;

    int col;

    int value;

    struct Node \*next;

    struct Node \*prev;

};

// Function to create a new node

struct Node\* createNode(int row, int col, int value) {

    struct Node\* newNode = (struct Node\*)malloc(sizeof(struct Node));

    newNode->row = row;

    newNode->col = col;

    newNode->value = value;

    newNode->next = NULL;

    newNode->prev = NULL;

    return newNode;

}

// Function to insert a node in a sorted order

void insert(struct Node\*\* head, int row, int col, int value) {

    struct Node\* newNode = createNode(row, col, value);

    if (\*head == NULL) {

        \*head = newNode;

        return;

    }

    struct Node\* temp = \*head;

    while (temp->next != NULL) {

        temp = temp->next;

    }

    temp->next = newNode;

    newNode->prev = temp;

}

// Function to multiply two sparse matrices

struct Node\* multiply(struct Node\* mat1, struct Node\* mat2, int rows, int cols, int common) {

    struct Node\* result = NULL;

    // For every element in the first matrix

    for (struct Node\* ptr1 = mat1; ptr1 != NULL; ptr1 = ptr1->next) {

        // For every element in the second matrix

        for (struct Node\* ptr2 = mat2; ptr2 != NULL; ptr2 = ptr2->next) {

            // If the column of the first element matches the row of the second

            if (ptr1->col == ptr2->row) {

                int newRow = ptr1->row;

                int newCol = ptr2->col;

                int newValue = ptr1->value \* ptr2->value;

                // Insert the product into the result matrix, or update if it already exists

                struct Node\* temp = result;

                int found = 0;

                while (temp != NULL) {

                    if (temp->row == newRow && temp->col == newCol) {

                        temp->value += newValue;

                        found = 1;

                        break;

                    }

                    temp = temp->next;

                }

                if (!found) {

                    insert(&result, newRow, newCol, newValue);

                }

            }

        }

    }

    return result;

}

// Function to print a sparse matrix

void printMatrix(struct Node\* head, int rows, int cols) {

    struct Node\* temp = head;

    printf("Sparse Matrix:\n");

    for (int i = 0; i < rows; i++) {

        for (int j = 0; j < cols; j++) {

            if (temp != NULL && temp->row == i && temp->col == j) {

                printf("%d ", temp->value);

                temp = temp->next;

            } else {

                printf("0 ");

            }

        }

        printf("\n");

    }

}

int main() {

    struct Node\* mat1 = NULL;

    struct Node\* mat2 = NULL;

    int rows1, cols1, rows2, cols2;

    // Input for matrix 1

    printf("Enter number of rows and columns for Matrix 1: ");

    scanf("%d %d", &rows1, &cols1);

    int n1;

    printf("Enter the number of non-zero elements in Matrix 1: ");

    scanf("%d", &n1);

    for (int i = 0; i < n1; i++) {

        int row, col, value;

        printf("Enter row, column and value: ");

        scanf("%d %d %d", &row, &col, &value);

        insert(&mat1, row, col, value);

    }

    // Input for matrix 2

    printf("Enter number of rows and columns for Matrix 2: ");

    scanf("%d %d", &rows2, &cols2);

    if (cols1 != rows2) {

        printf("Matrix multiplication not possible due to incompatible dimensions.\n");

        return 0;

    }

    int n2;

    printf("Enter the number of non-zero elements in Matrix 2: ");

    scanf("%d", &n2);

    for (int i = 0; i < n2; i++) {

        int row, col, value;

        printf("Enter row, column and value: ");

        scanf("%d %d %d", &row, &col, &value);

        insert(&mat2, row, col, value);

    }

    // Multiply matrices

    struct Node\* result = multiply(mat1, mat2, rows1, cols2, cols1);

    // Print the result matrix

    printMatrix(result, rows1, cols2);

    return 0;

}

1. Write a C program for implementation of insertion and deletion in a linked priority queue.