DS Lab 5

Deque

```c
#include <stdio.h>
#include <stdlib.h>
#define n 4

int f = -1;
int r = -1;
int deque[n];

void insertFront() {
    if ((r + 1) % n == f) {
        printf("Deque is full. Cannot insert at the front.\n");
        return;
    }
    if (f == -1 && r == -1) {
        f = r = 0;
    } else {
        f = (f - 1 + n) % n;
    }
    printf("Enter the item to be added to the queue: ");
    scanf("%d",&deque[f]);
}

void insertRear() {
    if ((r + 1) % n == f) {
        printf("Deque is full. Cannot insert at the rear.\n");
        return;
    }
    if (f == -1 && r == -1) {
        f = r = 0;
    } else {
        r = (r + 1) % n;
    }
    printf("Enter the item to be added to the queue: ");
    scanf("%d",&deque[r]);
}

void deleteFront() {
    if (f == -1 && r == -1) {
        printf("Deque is empty. Cannot delete from the front.\n");
        return;
    }
    printf("%d has been deleted...!\n",deque[f]);
    if (f == r) {
        f = r = -1;
```

```
    } else {
        f = (f + 1) % n;
    }
}

void deleteRear() {
    if (f == -1 && r == -1) {
        printf("Deque is empty. Cannot delete from the rear.\n");
        return;
    }
    printf("%d has been deleted...!\n",deque[r]);
    if (f == r) {
        f = r = -1;
    } else {
        r = (r - 1 + n) % n;
    }
}

void display() {
    if (f == -1 && r == -1) {
        printf("Deque is empty.\n");
        return;
    }
    int i = f;
    printf("Queue elements are: ");

    do
        {
         printf("\t%d", deque[i]);
         i = (i + 1) % n;
        } while (i != (r + 1) % n);
    printf("\n");
}

void main()
{
    printf("<<- Menu ->>\n\n1. Enqueue at front\n2. Enqueue at rear\n3.
Dequeue at front\n4. Dequeue at rear\n5. Display the Deque\n6. Exit\n");
    int choice;
    while (1)
    {
        printf("\nEnter the operation to be executed:");
        scanf("%d", &choice);
        switch (choice)
        {
        case 1:
            insertFront();
            break;
```

```
        case 2:
            insertRear();
            break;
        case 3:
            deleteFront();
            break;
        case 4:
            deleteRear();
            break;
        case 5:
            display();
            break;
        case 6:
            exit(0);
        default:
            printf("Invalid Operation\n");
        }
    }
}
```

Sparse Matrix Transpose

```
#include <stdio.h>

struct triple {
    int row, col, data;
};

void main() {
    int row, col;
    printf("Enter the number of rows: ");
    scanf("%d", &row);
    printf("Enter the number of columns: ");
    scanf("%d", &col);

    int matrix[row][col];
    int count = 1; // 0th position to store the number of non-zero elements
    printf("Enter the elements of the normal matrix:\n");
    for (int i = 0; i < row; i++) {
        for (int j = 0; j < col; j++) {
            scanf("%d", &matrix[i][j]);
            if (matrix[i][j] != 0) {
                count++;
            }
        }
    }
```

```
    printf("\nInputted Matrix:\n");
    for (int i = 0; i < row; i++) {
        printf("|");
        for (int j = 0; j < col; j++) {
            printf("%2d ", matrix[i][j]);
        }
        printf("|\n");
    }

    struct triple sparse[count];
    struct triple transpose[count];

    transpose[0].row = sparse[0].row = row;
    transpose[0].col = sparse[0].col = col;
    transpose[0].data = sparse[0].data = count;

    int k = 1;
    for (int i = 0; i < row; i++) {
        for (int j = 0; j < col; j++) {
            if (matrix[i][j] != 0) {
                sparse[k].row = i;
                sparse[k].col = j;
                sparse[k].data = matrix[i][j];
                k++;
            }
        }
    }

    printf("\nSparse Matrix (in triple form):\n");
    for (int i = 1; i < count; i++) {
        printf("%d %d %d\n", sparse[i].row, sparse[i].col, sparse[i].data);
    }

    k = 1;
    for (int i = 0; i < col; i++) {
        for (int j = 1; j < count; j++) {
            if (sparse[j].col == i) {
                transpose[k].row = sparse[j].col;
                transpose[k].col = sparse[j].row;
                transpose[k].data = sparse[j].data;
                k++;
            }
        }
    }

    int final[col][row];
    for (int i = 0; i < col; i++) {
        for (int j = 0; j < row; j++) {
```

```
            final[i][j] = 0;
        }
    }

    k = 1;
    for (int j = 1; j < count; j++) {
        int i = transpose[j].col;
        int m = transpose[j].row;
        final[i][m] = transpose[j].data;
    }

    printf("\nMatrix after transpose:\n");
    for (int i = 0; i < col; i++) {
        printf("|");
        for (int j = 0; j < row; j++) {
            printf("%2d ", final[i][j]);
        }
        printf("|\n");
    }
}
```