1. Write a C program to implement infix, prefix and postfix notations for arithmetic expressions using stack

Code :

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

#define MAX_SIZE 100

// Structure for stack
struct Stack {
    int top;
    unsigned capacity;
    char *array;
};

// Functions for stack operations
struct Stack* createStack(unsigned capacity) {
    struct Stack* stack = (struct Stack*)malloc(sizeof(struct Stack));
    stack->capacity = capacity;
    stack->top = -1;
    stack->array = (char*)malloc(stack->capacity * sizeof(char));
    return stack;
}

int isFull(struct Stack* stack) {
    return stack->top == stack->capacity - 1;
}

int isEmpty(struct Stack* stack) {
    return stack->top == -1;
}

void push(struct Stack* stack, char item) {
    if (isFull(stack)) {
        return;
    }
    stack->array[++stack->top] = item;
}
```

```
char pop(struct Stack* stack) {
    if (isEmpty(stack)) {
        return '\0';
    }
    return stack->array[stack->top--];
}

char peek(struct Stack* stack) {
    if (isEmpty(stack)) {
        return '\0';
    }
    return stack->array[stack->top];
}

int isOperand(char ch) {
    return (ch >= 'a' && ch <= 'z') || (ch >= 'A' && ch <= 'Z');
}

int precedence(char ch) {
    switch (ch) {
        case '+':
        case '-':
            return 1;
        case '*':
        case '/':
            return 2;
        case '^':
            return 3;
    }
    return -1;
}

// Function to convert infix to postfix
void infixToPostfix(char* infix, char* postfix) {
    struct Stack* stack = createStack(strlen(infix));
    int i, k;
    for (i = 0, k = -1; infix[i]; ++i) {
        if (isOperand(infix[i])) {
            postfix[++k] = infix[i];
        } else if (infix[i] == '(') {
```

```
            push(stack, infix[i]);
        } else if (infix[i] == ')') {
            while (!isEmpty(stack) && peek(stack) != '(') {
                postfix[++k] = pop(stack);
            }
            if (!isEmpty(stack) && peek(stack) != '(') {
                return; // Invalid expression
            } else {
                pop(stack);
            }
        } else {
            while (!isEmpty(stack) && precedence(infix[i]) <= precedence(peek(stack)))
{
                postfix[++k] = pop(stack);
            }
            push(stack, infix[i]);
        }
    }
    while (!isEmpty(stack)) {
        postfix[++k] = pop(stack);
    }
    postfix[++k] = '\0';
}

int main() {
    char infix[MAX_SIZE], postfix[MAX_SIZE];

    printf("Enter the infix expression: ");
    scanf("%s", infix);

    infixToPostfix(infix, postfix);

    printf("The postfix expression is: %s\n", postfix);

    return 0;
}
```

Output: Enter the infix expression: a+b*c-(d/e+f)

The postfix expression is: abc*+de/f+-

2. Write a program that implements Queue (its operations) using i) Arrays ii) Linked list(Pointers).

Code :

Using Arrays:

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

#define MAX_SIZE 100

// Structure for queue
struct Queue {
    int front, rear, size;
    unsigned capacity;
    int* array;
};

// Function to create a queue
struct Queue* createQueue(unsigned capacity) {
    struct Queue* queue = (struct Queue*)malloc(sizeof(struct Queue));
    queue->capacity = capacity;
    queue->front = queue->size = 0;
    queue->rear = capacity - 1;  // This is important, see the enqueue
    queue->array = (int*)malloc(queue->capacity * sizeof(int));
    return queue;
}

// Function to check if the queue is full
int isFull(struct Queue* queue) {
    return (queue->size == queue->capacity);
}

// Function to check if the queue is empty
int isEmpty(struct Queue* queue) {
    return (queue->size == 0);
}

// Function to add an item to the queue
void enqueue(struct Queue* queue, int item) {
    if (isFull(queue)) {
        return;
```

```
    }
    queue->rear = (queue->rear + 1) % queue->capacity;
    queue->array[queue->rear] = item;
    queue->size = queue->size + 1;
}

// Function to remove an item from the queue
int dequeue(struct Queue* queue) {
    if (isEmpty(queue)) {
        return -1;
    }
    int item = queue->array[queue->front];
    queue->front = (queue->front + 1) % queue->capacity;
    queue->size = queue->size - 1;
    return item;
}

// Function to display the queue elements
void display(struct Queue* queue) {
    int i;
    for (i = 0; i < queue->size; i++) {
        printf("%d ", queue->array[(queue->front + i) % queue->capacity]);
    }
    printf("\n");
}

int main() {
    struct Queue* queue = createQueue(MAX_SIZE);

    enqueue(queue, 10);
    enqueue(queue, 20);
    enqueue(queue, 30);
    enqueue(queue, 40);

    printf("Queue elements after enqueuing: ");
    display(queue);

    dequeue(queue);
    dequeue(queue);
```

```
    printf("Queue elements after dequeuing: ");
    display(queue);

    return 0;
}
```

Output: Queue elements after enqueuing: 10 20 30 40
Queue elements after dequeuing: 30 40

Using Linked List:

Code :

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

// Structure for queue node
struct Node {
    int data;
    struct Node* next;
};

// Structure for queue
struct Queue {
    struct Node *front, *rear;
};

// Function to create a new queue node
struct Node* newNode(int data) {
    struct Node* temp = (struct Node*)malloc(sizeof(struct Node));
    temp->data = data;
    temp->next = NULL;
    return temp;
}

// Function to create an empty queue
struct Queue* createQueue() {
    struct Queue* queue = (struct Queue*)malloc(sizeof(struct Queue));
    queue->front = queue->rear = NULL;
    return queue;
}

// Function to add an item to the queue
```

```
void enqueue(struct Queue* queue, int data) {
    struct Node* temp = newNode(data);
    if (queue->rear == NULL) {
        queue->front = queue->rear = temp;
        return;
    }
    queue->rear->next = temp;
    queue->rear = temp;
}

// Function to remove an item from the queue
int dequeue(struct Queue* queue) {
    if (queue->front == NULL) {
        return -1;
    }
    int data = queue->front->data;
    struct Node* temp = queue->front;
    queue->front = queue->front->next;
    if (queue->front == NULL) {
        queue->rear = NULL;
    }
    free(temp);
    return data;
}

// Function to display the queue elements
void display(struct Queue* queue) {
    struct Node* temp = queue->front;
    while (temp != NULL) {
        printf("%d ", temp->data);
        temp = temp->next;
    }
    printf("\n");
}

int main() {
    struct Queue* queue = createQueue();

    enqueue(queue, 10);
    enqueue(queue, 20);
```

```
    enqueue(queue, 30);
    enqueue(queue, 40);

    printf("Queue elements after enqueuing: ");
    display(queue);

    dequeue(queue);
    dequeue(queue);

    printf("Queue elements after dequeuing: ");
    display(queue);

    return 0;
}
```

Output: Queue elements after enqueuing: 10 20 30 40
Queue elements after dequeuing: 30 40

3. Write a C program to implement Queue operations such as ENQUEUE, DEQUEUE and Display

Code :

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

#define MAX_SIZE 10

// Structure for queue
struct Queue {
    int items[MAX_SIZE];
    int front;
    int rear;
};

// Function to create an empty queue
struct Queue* createQueue() {
    struct Queue* queue = (struct Queue*)malloc(sizeof(struct Queue));
    queue->front = -1;
    queue->rear = -1;
    return queue;
```

```
}

// Function to check if the queue is full
int isFull(struct Queue* queue) {
    return (queue->rear == MAX_SIZE - 1);
}

// Function to check if the queue is empty
int isEmpty(struct Queue* queue) {
    return (queue->front == -1 && queue->rear == -1);
}

// Function to add an item to the queue
void enqueue(struct Queue* queue, int value) {
    if (isFull(queue)) {
        printf("Queue is full\n");
    } else {
        if (isEmpty(queue)) {
            queue->front = 0;
        }
        queue->rear ++;
        queue->items[queue->rear] = value;
        printf("%d enqueued to queue\n", value);
    }
}

// Function to remove an item from the queue
int dequeue(struct Queue* queue) {
    int item;
    if (isEmpty(queue)) {
        printf("Queue is empty\n");
        return -1;
    } else {
        item = queue->items[queue->front];
        if (queue->front >= queue->rear) {
            queue->front = -1;
            queue->rear = -1;
        } else {
            queue->front ++;
        }
```

```
        printf("%d dequeued from queue\n", item);
        return item;
    }
}

// Function to display the queue elements
void display(struct Queue* queue) {
    int i;
    if (isEmpty(queue)) {
        printf("Queue is empty\n");
    } else {
        printf("Queue elements: ");
        for (i = queue->front; i <= queue->rear; i++) {
            printf("%d ", queue->items[i]);
        }
        printf("\n");
    }
}

int main() {
    struct Queue* queue = createQueue();

    enqueue(queue, 10);
    enqueue(queue, 20);
    enqueue(queue, 30);
    enqueue(queue, 40);

    display(queue);

    dequeue(queue);
    dequeue(queue);

    display(queue);

    return 0;
}
```

Output:

```
10 enqueued to queue
20 enqueued to queue
30 enqueued to queue
```

40 enqueued to queue
Queue elements: 10 20 30 40
10 dequeued from queue
20 dequeued from queue
Queue elements: 30 40