# DATA STRUCTURE

## DAY - 04  29-07-2024  CSA0390

## 1) WRITE C PROGRAM FOR INFIX TO POSTFIX

```
#include <stdio.h>
#include <ctype.h>
#include <stdlib.h>
#include <string.h>
#define MAX 100
struct Stack {
	int top;
	char items[MAX];
};
struct Stack* createStack() {
	struct Stack* stack = (struct Stack*)malloc(sizeof(struct Stack));
	stack->top = -1;
	return stack;
}
int isEmpty(struct Stack* stack) {
	return stack->top == -1;
}
void push(struct Stack* stack, char item) {
	if (stack->top == MAX - 1) {
		printf("Stack overflow¥n");
		return;
	}
	stack->items[++stack->top] = item;
}
char pop(struct Stack* stack) {
	if (isEmpty(stack)) {
		printf("Stack underflow¥n");
		return '¥0';
	}
	return stack->items[stack->top--];
}
char peek(struct Stack* stack) {
	if (isEmpty(stack)) {
		return '¥0';
	}
	return stack->items[stack->top];
}
int isOperand(char ch) {
```

```
    return isalpha(ch) || isdigit(ch);
}
int precedence(char ch) {
    switch (ch) {
        case '+':
        case '-':
            return 1;
        case '*':
        case '/':
            return 2;
        case '^':
            return 3;
    }
    return -1;
}
void infixToPostfix(char* infix, char* postfix) {
    struct Stack* stack = createStack();
    int i, j = 0;
    for (i = 0; infix[i]; ++i) {
        if (isOperand(infix[i])) {
            postfix[j++] = infix[i];
        } else if (infix[i] == '(') {
            push(stack, infix[i]);
        } else if (infix[i] == ')') {
            while (!isEmpty(stack) && peek(stack) != '(') {
                postfix[j++] = pop(stack);
            }
            pop(stack);    // Pop '('
        } else {    // Operator
            while (!isEmpty(stack) && precedence(infix[i]) <= precedence(peek(stack))) {
                postfix[j++] = pop(stack);
            }
            push(stack, infix[i]);
        }
    }
    while (!isEmpty(stack)) {
        postfix[j++] = pop(stack);
    }

    postfix[j] = '¥0';    // Null terminate the postfix expression
    free(stack);    // Free the stack memory
}
int main() {
    char infix[MAX], postfix[MAX];
    printf("Enter an infix expression: ");
    scanf("%s", infix);
    infixToPostfix(infix, postfix);
    printf("Postfix expression: %s¥n", postfix);
    return 0;
}
```

## OUTPUT :

```
Enter an infix expression: A+B+C+(D-E)
Postfix expression: AB+C+DE-+
```

## 2) WRITE C PROGRAM FOR QUEUE USING ARRAY

```
#include <stdio.h>
#include <stdlib.h>
#define MAX 100
struct Queue {
	int front, rear, size;
	unsigned capacity;
	int* array;
};
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 function
	queue->array = (int*)malloc(queue->capacity * sizeof(int));
	return queue;
}
int isFull(struct Queue* queue) {
	return (queue->size == queue->capacity);
}
int isEmpty(struct Queue* queue) {
	return (queue->size == 0);
}
void enqueue(struct Queue* queue, int item) {
	if (isFull(queue)) {
		printf("Queue is full¥n");
		return;
	}
	queue->rear = (queue->rear + 1) % queue->capacity;
	queue->array[queue->rear] = item;
	queue->size = queue->size + 1;
	printf("%d enqueued to queue¥n", item);
}
int dequeue(struct Queue* queue) {
	if (isEmpty(queue)) {
		printf("Queue is empty¥n");
		return -1;
	}
	int item = queue->array[queue->front];
	queue->front = (queue->front + 1) % queue->capacity;
```

```
	queue->size = queue->size - 1;
	return item;
}
int front(struct Queue* queue) {
	if (isEmpty(queue)) {
		return -1;
	}
	return queue->array[queue->front];
}
int rear(struct Queue* queue) {
	if (isEmpty(queue)) {
		return -1;
	}
	return queue->array[queue->rear];
}
int main() {
	struct Queue* queue = createQueue(MAX);
	enqueue(queue, 10);
	enqueue(queue, 20);
	enqueue(queue, 30);
	enqueue(queue, 40);
	printf("%d dequeued from queue¥n", dequeue(queue));
	printf("Front item is %d¥n", front(queue));
	printf("Rear item is %d¥n", rear(queue));
	free(queue->array);
	free(queue);
	return 0;
}
```

## OUTPUT :

```
10 enqueued to queue
20 enqueued to queue
30 enqueued to queue
40 enqueued to queue
10 dequeued from queue
Front item is 20
Rear item is 40
```

## 3) WRITE C PROGRAM FOR QUEUE USING LINKED LIST

```
#include <stdio.h>
#include <stdlib.h>
struct Node {
	int data;
	struct Node* next;
};
```

```
struct Queue {
	struct Node *front, *rear;
};
struct Node* newNode(int data) {
	struct Node* temp = (struct Node*)malloc(sizeof(struct Node));
	temp->data = data;
	temp->next = NULL;
	return temp;
}
struct Queue* createQueue() {
	struct Queue* queue = (struct Queue*)malloc(sizeof(struct Queue));
	queue->front = queue->rear = NULL;
	return queue;
}
void enqueue(struct Queue* queue, int data) {
	struct Node* temp = newNode(data);
	if (queue->rear == NULL) {
		queue->front = queue->rear = temp;
		printf("%d enqueued to queue¥n", data);
		return;
	}
	queue->rear->next = temp;
	queue->rear = temp;
	printf("%d enqueued to queue¥n", data);
}
int dequeue(struct Queue* queue) {
	if (queue->front == NULL) {
		printf("Queue is empty¥n");
		return -1;
	}
	struct Node* temp = queue->front;
	int data = temp->data;
	queue->front = queue->front->next;
	if (queue->front == NULL) {
		queue->rear = NULL;
	}
	free(temp);
	return data;
}
int front(struct Queue* queue) {
	if (queue->front == NULL) {
		return -1;
	}
	return queue->front->data;
}

// Function to get the rear item of the queue
int rear(struct Queue* queue) {
	if (queue->rear == NULL) {
		return -1;
	}
```

```
    return queue->rear->data;
}
int main() {
    struct Queue* queue = createQueue();
    enqueue(queue, 10);
    enqueue(queue, 20);
    enqueue(queue, 30);
    enqueue(queue, 40);
    printf("%d dequeued from queue¥n", dequeue(queue));
    printf("Front item is %d¥n", front(queue));
    printf("Rear item is %d¥n", rear(queue));
    while (queue->front != NULL) {
        dequeue(queue);
    }
    free(queue);
    return 0;
}
```

# OUTPUT:

```
10 enqueued to queue
20 enqueued to queue
30 enqueued to queue
40 enqueued to queue
10 dequeued from queue
Front item is 20
Rear item is 40
```