# DATA STRUCTURE

## DAY 4 -29/07/2024

## 1.Infix to postfix in stack

## PROGRAM:

```
#include <stdio.h>
#include <stdlib.h>
#include <ctype.h>
#include <string.h>
#define MAX 100
typedef struct {
    int top;
    char items[MAX];
} Stack;
void initStack(Stack *s) {
    s->top = -1;
}
  int isEmpty(Stack *s) {
    return s->top == -1;
}
void push(Stack *s, char item) {
    if (s->top < (MAX - 1)) {
        s->items[++(s->top)] = item;
    } else {
        fprintf(stderr, "Stack overflow\n");
        exit(EXIT_FAILURE);
    }
}
char pop(Stack *s) {
    if (!isEmpty(s)) {
```

```
        return s->items[(s->top)--];
    } else {
        fprintf(stderr, "Stack underflow\n");
        exit(EXIT_FAILURE);
    }
}
 char peek(Stack *s) {
    if (!isEmpty(s)) {
        return s->items[s->top];
    } else {
        return '\0';
    }
}
 int isOperator(char c) {
    return (c == '+' || c == '-' || c == '*' || c == '/' || c == '^');
}
int precedence(char op) {
    switch (op) {
        case '+':
        case '-':
            return 1;
        case '*':
        case '/':
            return 2;
        case '^':
            return 3;
        default:
            return 0;
    }
```

```
}

void infixToPostfix(const char *infix, char *postfix) {
    Stack s;
    initStack(&s);
    int k = 0;
    for (int i = 0; infix[i]; i++) {
        if (isalnum(infix[i])) {
            postfix[k++] = infix[i];
        } else if (infix[i] == '(') {
            push(&s, infix[i]);
        } else if (infix[i] == ')') {
            while (!isEmpty(&s) && peek(&s) != '(') {
                postfix[k++] = pop(&s);
            }
            if (!isEmpty(&s) && peek(&s) == '(') {
                pop(&s);
            } else {
                fprintf(stderr, "Mismatched parentheses\n");
                exit(EXIT_FAILURE);
            }
        } else if (isOperator(infix[i])) {
            while (!isEmpty(&s) && precedence(peek(&s)) >= precedence(infix[i])) {
                postfix[k++] = pop(&s);
            }
            push(&s, infix[i]);
        } else
            fprintf(stderr, "Invalid character encountered: %c\n", infix[i]);
            exit(EXIT_FAILURE);
```

```
        }
    }
    while (!isEmpty(&s)) {
        char top = peek(&s);
        if (top == '(' || top == ')') {
            fprintf(stderr, "Mismatched parentheses\n");
            exit(EXIT_FAILURE);
        }
        postfix[k++] = pop(&s);
    }
    postfix[k] = '\0';
}
int main() {
    char infix[MAX], postfix[MAX];
    printf("Enter infix expression: ");
    if (fgets(infix, MAX, stdin) == NULL) {
        fprintf(stderr, "Error reading input\n");
        return EXIT_FAILURE;
    }
        size_t len = strlen(infix);
    if (len > 0 && infix[len - 1] == '\n') {
        infix[len - 1] = '\0';
    }
    infixToPostfix(infix, postfix)
    printf("Postfix expression: %s\n", postfix);
    return 0;
}
```

**OUTPUT:**

Enter infix expression:  A+B*C

Postfix expression:  ABC*+

## 2. Queue using array

**PROGRAM:**

```
#include <stdio.h>
#include <stdlib.h>
#define MAX_SIZE 100
int queue[MAX_SIZE];
int front = -1, rear = -1;
void enqueue(int value) {
    if (rear == MAX_SIZE - 1) {
        printf("Queue is full.\n");
    } else {
        if (front == -1) {
            front = 0;
        }
        rear++;
        queue[rear] = value;
    }
}
void dequeue() {
    if (front == -1 || front > rear) {
        printf("Queue is empty.\n");
    } else {
        printf("Dequeued element: %d\n", queue[front]);
        front++;
    }
}
```

```
void display() {
    if (front == -1) {
        printf("Queue is empty.\n");
    } else {
        printf("Queue elements: ");
        for (int i = front; i <= rear; i++) {
            printf("%d ", queue[i]);
        }
        printf("\n");
    }
}
int main() {
    enqueue(10);
    enqueue(20);
    enqueue(30);
    display();
    dequeue();
    dequeue();
    display();
    return 0;
}
```

**OUTPUT:**

Queue elements: 10 20 30

Dequeued element: 10

Dequeued element: 20

Queue elements: 30

## 3.Queue using linkedlist

**PROGRAM:**

```
#include <stdio.h>
#include <stdlib.h>
 struct Node {
    int data;
    struct Node* next;
};
 struct Queue {
    struct Node* front;
    struct Node* rear;
};
struct Node* createNode(int data) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->data = data;
    newNode->next = NULL;
    return newNode;
}
struct Queue* createQueue() {
    struct Queue* queue = (struct Queue*)malloc(sizeof(struct Queue));
    queue->front = queue->rear = NULL;
    return queue;
}
int isEmpty(struct Queue* queue) {
    return (queue->front == NULL);
}
 void enqueue(struct Queue* queue, int data) {
    struct Node* newNode = createNode(data);
    if (isEmpty(queue)) {
        queue->front = queue->rear = newNode;
    } else {
```

```
        queue->rear->next = newNode;
        queue->rear = newNode;
    }
}
int dequeue(struct Queue* queue) {
    if (isEmpty(queue)) {
        printf("Queue is empty.\n");
        return -1;
    } else {
        struct Node* temp = queue->front;
        int data = temp->data;
        queue->front = queue->front->next;
        free(temp);
        return data;
    }
}
 int front(struct Queue* queue) {
    if (isEmpty(queue)) {
        printf("Queue is empty.\n");
        return -1;
    } else {
        return queue->front->data;
    }
}
 int rear(struct Queue* queue) {
    if (isEmpty(queue)) {
        printf("Queue is empty.\n");
        return -1;
    } else {
```

```
        return queue->rear->data;
    }
}
void display(struct Queue* queue) {
    if (isEmpty(queue)) {
        printf("Queue is empty.\n");
    } else {
        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: ");
    display(queue);
    printf("Front element: %d\n", front(queue));
    printf("Rear element: %d\n", rear(queue));
    printf("Dequeued element: %d\n", dequeue(queue));
    printf("Dequeued element: %d\n", dequeue(queue));
    printf("Queue elements after dequeue: ");
    display(queue);
```

```
    return 0;
}
```

## OUTPUT:

```
Queue elements: 10 20 30 40
Front element: 10
Rear element: 40
Dequeued element: 10
Dequeued element: 20
Queue elements after dequeue: 30 40
```