DATA STRUCTURE

DAY 04 , 29/07/24 , CSA0390

1 . write a c program of convert infix to postfix .

```
#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#define MAX 100

char stack[MAX];

int top = -1;

void push(char item) {

        if (top >= MAX - 1) {

                printf("Stack Overflow\n");

        } else {

                top = top + 1;

                stack[top] = item;

        }

}

char pop() {

        char item;

        if (top < 0) {

                printf("Stack Underflow\n");

                exit(1);

        } else {
```

```
        item = stack[top];
        top = top - 1;
        return item;
    }
}
int precedence(char symbol) {
    if (symbol == '^') {
        return 3;
    } else if (symbol == '*' || symbol == '/') {
        return 2;
    } else if (symbol == '+' || symbol == '-') {
        return 1;
    } else {
        return 0;
    }
}
void infixToPostfix(char infix[], char postfix[]) {
    int i = 0, j = 0;
    char symbol, temp;
    push('(');
    strcat(infix, ")");
    while (infix[i] != '\0') {
        symbol = infix[i];
        if (symbol == '(') {
            push(symbol);
```

```
} else if (isalnum(symbol)) {
        postfix[j] = symbol;
        j++;
} else if (symbol == ')') {
        temp = pop();
        while (temp != '(') {
                postfix[j] = temp;
                j++;
                temp = pop();
        }
} else {
        while (precedence(stack[top]) >= precedence(symbol)) {
                temp = pop();
                postfix[j] = temp;
                j++;
        }
        push(symbol);
}
                i++;
        }
        postfix[j] = '\0';
}
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 :

infix expression : a+b*c

postfix expression : ? @

2 . write a c program implementation of queue using array list .

```
#include <stdio.h>
#include <stdlib.h>
#define MAX_SIZE 100
struct Queue {
    int items[MAX_SIZE];
    int front;
    int rear;
};
struct Queue* createQueue() {
    struct Queue* queue = (struct Queue*)malloc(sizeof(struct Queue));
    queue->front = -1;
    queue->rear = -1;
    return queue;
}
int isEmpty(struct Queue* queue) {
```

```
    if (queue->rear == -1)
        return 1;
    else
        return 0;
}
int isFull(struct Queue* queue) {
    if (queue->rear == MAX_SIZE - 1)
        return 1;
    else
        return 0;
}
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;
    }
}
int dequeue(struct Queue* queue) {
    int item;
    if (isEmpty(queue)) {
        printf("Queue is empty\n");
```

```
        return -1;
    } else {
        item = queue->items[queue->front];
        queue->front++;
        if (queue->front > queue->rear) {
            queue->front = queue->rear = -1;
        }
        return item;
    }
}
void display(struct Queue* queue) {
    int i;
    if (isEmpty(queue))
        printf("Queue is empty\n");
    else {
        printf("Front index: %d\n", queue->front);
        printf("Items: ");
        for (i = queue->front; i <= queue->rear; i++)
            printf("%d ", queue->items[i]);
        printf("\nRear index: %d\n", queue->rear);
    }
}
int main() {
    struct Queue* queue = createQueue();
    enqueue(queue, 10);
```

```
    enqueue(queue, 20);
    enqueue(queue, 30);
    display(queue);
    printf("Dequeued: %d\n", dequeue(queue));
    printf("Dequeued: %d\n", dequeue(queue));
    display(queue);
    return 0;
}
```

OUTPUT :

```
Front index: 0
Items: 10 20 30
Rear index: 2
Dequeued: 10
Dequeued: 20
Front index: 2
Items: 30
Rear index: 2
```

3 . write a c program implementation of 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;
                return;
        }
        queue->rear->next = temp;
        queue->rear = temp;
}
void deQueue(struct Queue* queue) {
        if (queue->front == NULL)
```

```
        return;
    struct Node* temp = queue->front;
    queue->front = queue->front->next;
    if (queue->front == NULL)
        queue->rear = NULL;
    free(temp);
}
int main() {
  struct Queue* queue = createQueue();
  enQueue(queue, 10);
  enQueue(queue, 20);
  deQueue(queue);
  enQueue(queue, 30);
  enQueue(queue, 40);
  deQueue(queue);
  printf("Queue Front: %d\n", queue->front->data);
  printf("Queue Rear: %d\n", queue->rear->data);
return 0;
}
```

OUTPUT :

Queue Front: 30

Queue Rear: 40