NAME : P. VINEETHA .

DATA STRUCTURE : DAY-4

1. convert infix to postfix   using c language.

```
#include <limits.h>
#include <stdio.h>
#include <stdlib.h>
#define MAX 20
char stk[20];
int top = -1;
int isEmpty(){
        return top == -1;
}
int isFull(){
        return top == MAX - 1;
}
char peek(){
        return stk[top];
}
char pop(){
        if(isEmpty())
                return -1;
        char ch = stk[top];
        top--;
        return(ch);
}
void push(char oper){
        if(isFull())
                printf("Stack Full!!!!");
      else{
                top++;
                stk[top] = oper;
        }
}
int checkIfOperand(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;
}
int covertInfixToPostfix(char* expression)
{
    int i, j;
    for (i = 0, j = -1; expression[i]; ++i)
    {
       if (checkIfOperand(expression[i]))
            expression[++j] = expression[i];
        else if (expression[i] == '(')
            push(expression[i]);
        else if (expression[i] == ')')
        {
            while (!isEmpty() && peek() != '(')
                expression[++j] = pop();
            if (!isEmpty() && peek() != '(')
                return -1; // invalid expression
            else
                pop();
        }
        else // if an opertor
        {
            while (!isEmpty() && precedence(expression[i]) <= precedence(peek()))
                expression[++j] = pop();
            push(expression[i]);
        }
    }
    while (!isEmpty())
        expression[++j] = pop();
    expression[++j] = '¥0';
    printf( "%s", expression);
}
int main()
{
char expression[] = "((p+(q*r))-s)";
    covertInfixToPostfix(expression);
    return 0;
}
```

OUTPUT :

```
pqr*+s-
```

2 . Write a c programming for Queue using array .

```
#include <stdio.h>
#define MAX_SIZE 100
int queue[MAX_SIZE];
int front = -1;
int rear = -1;
void enqueue(int value) {
    if (rear == MAX_SIZE - 1) {
        printf("Queue is full. Cannot enqueue.¥n");
        return;
    }
    if (front == -1) {
        front = 0;
    }
    rear++;
    queue[rear] = value;
    printf("%d enqueued successfully.¥n", value);
}
void dequeue() {
    if (front == -1 || front > rear) {
        printf("Queue is empty. Cannot dequeue.¥n");
        return;
    }
    printf("%d dequeued successfully.¥n", queue[front]);
    front++;
}
```

```
int peek() {
    if (front == -1 || front > rear) {
        printf("Queue is empty. Cannot peek.¥n");
        return -1;
    }
    return queue[front];
}

int isEmpty() {
    if (front == -1 || front > rear) {
        return 1;
    }
    return 0;
}
int isFull() {
    if (rear == MAX_SIZE - 1) {
        return 1;
    }
    return 0;
}
int size() {
    if (front == -1 || front > rear) {
        return 0;
    }
    return rear - front + 1;
```

```
}

void display() {

        if (front == -1 || front > rear) {

                printf("Queue is empty. Nothing to display.¥n");

                return;

        }

        printf("Queue elements: ");

        for (int i = front; i <= rear; i++) {

                printf("%d ", queue[i]);

        }

        printf("¥n");

}

int main() {

        enqueue(10);

        enqueue(20);

        enqueue(30);

        enqueue(40);

        enqueue(50);

display();

      printf("Front element: %d¥n", peek());

        printf("Queue size: %d¥n", size());

      dequeue();

        dequeue();

      display();

    printf("Is queue empty? %s¥n", isEmpty() ? "Yes" : "No");
```

```
        printf("Is queue full? %s¥n", isFull() ? "Yes" : "No");

        return 0;

}
```

OUTPUT :

```
10 enqueued successfully.

20 enqueued successfully.

30 enqueued successfully.

40 enqueued successfully.

50 enqueued successfully.

Queue elements: 10 20 30 40 50

Front element: 10

Queue size: 5

10 dequeued successfully.

20 dequeued successfully

Queue elements: 30 40 50

Is queue empty? No

Is queue full? No
```

3. Write a C programming 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;
        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
```