DATA STRUCTURE

DAY-4

1. Covert infix to postfix

Program:

```
#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <ctype.h>

 #define MAX 100

 char stack[MAX];

int top = -1;

 void push(char item) {

   if (top >= MAX - 1) {

       printf("Stack Overflow\n");

   } else {

       stack[++top] = item;

   }

}

 char pop() {

   if (top < 0) {

       printf("Stack Underflow\n");

       exit(1);

   } else {

       return stack[top--];

   }

}

int precedence(char symbol) {
    if (symbol == '^') return 3;

    if (symbol == '*' || symbol == '/') return 2;

    if (symbol == '+' || symbol == '-') return 1;

    return 0;

}

 void infixToPostfix(char infix[], char postfix[]) {

    int i = 0, j = 0;

    char symbol, temp;

    push('(');

    strcat(infix, ")");

 while ((symbol = infix[i]) != '\0') {

      if (symbol == '(') {

         push(symbol);

      } else if (isalnum(symbol)) {

         postfix[j++] = symbol;

         if (isalnum(infix[i + 1])) {

            while (isalnum(infix[i + 1])) {

               postfix[j++] = infix[++i];

            }

         }

         postfix[j++] = ' ';

         while (stack[top] != '(') {

            postfix[j++] = pop();

            postfix[j++] = ' ';

         }

         pop();

      } else {

         while (top != -1 && precedence(stack[top]) >= precedence(symbol)) {
            postfix[j++] = pop();
            postfix[j++] = ' ';
          }
          push(symbol);
      }
      i++;
   }
   postfix[j] = '\0';
}
 int main() {
   char infix[MAX], postfix[MAX];
 printf("Enter an infix expression: ");
   if (fgets(infix, sizeof(infix), stdin) == NULL) {
      printf("Error reading input.\n");
      return 1;
   }
      infix[strcspn(infix, "\n")] = '\0';
   infixToPostfix(infix, postfix);
   printf("Postfix expression: %s\n", postfix);
   return 0;
}
```

Output:

```
Enter an infix expression: A+(B*C+D)/E
Postfix expression: A B C * D + E / +
```

2.Queue using array

Program:

```
#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;
    }
}
int main() {
    struct Queue* queue = createQueue();
   enqueue(queue, 10);
    enqueue(queue, 20);
    enqueue(queue, 30);
   printf("Dequeued item: %d\n", dequeue(queue));
```

```
    printf("Dequeued item: %d\n", dequeue(queue));
    return 0;
}
```

Output:

```
Dequeued item: 10
Dequeued item: 20
```

3.Queue using Linked list

Program:

```
#include <stdio.h>
#include <stdlib.h>
struct Node {
    int data;
    struct Node* next;
};
struct Queue {
    struct Node *front, *rear;
};
void enqueue(struct Queue* q, int data) {
    struct Node* temp = (struct Node*)malloc(sizeof(struct Node));
    temp->data = data;
    temp->next = NULL;
    if (q->rear == NULL) {
        q->front = q->rear = temp;
        return;
    }
    q->rear->next = temp;
    q->rear = temp;
}
void dequeue(struct Queue* q) {
   if (q->front == NULL)
      return;
   struct Node* temp = q->front;
   q->front = q->front->next;
   if (q->front == NULL)
      q->rear = NULL;
   free(temp);
}
int main() {
   struct Queue* q = (struct Queue*)malloc(sizeof(struct Queue));
   q->front = q->rear = NULL;
   enqueue(q, 10);
   enqueue(q, 20);
   dequeue(q);
   dequeue(q);
   enqueue(q, 30);
   enqueue(q, 40);
   enqueue(q, 50);
   dequeue(q);
   printf("Queue Front : %d \n", q->front->data);
   printf("Queue Rear : %d", q->rear->data);
   return 0;
}
```

Output:

```
Queue Front : 40
Queue Rear : 50
```