Kartheeka.R

192372289

CSE-AI

29/07/2024

1. Infix to postfix:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

// Function to check if character is operator
int isOperator(char c) {
   if (c == '+' || c == '-' || c == '*' || c == '/' || c == '^')
      return 1;
   return 0;
}

// Function to check operator precedence
int precedence(char c) {
   if (c == '+' || c == '-')
      return 1;
   else if (c == '*' || c == '/')
      return 2;
   else if (c == '^')
      return 3;
   return 0;
}

// Function to convert infix to postfix
void infixToPostfix(char* infix, char* postfix) {
   int i, j = 0;
   char stack[100];
   int top = -1;

   for (i = 0; infix[i]; i++) {
      if (infix[i] == ' ')
         continue;
```

```
        if (infix[i] == '(')
            stack[++top] = infix[i];
        else if (infix[i] == ')') {
            while (top != -1 && stack[top] != '(')
                postfix[j++] = stack[top--];
            top--;
        } else if (!isOperator(infix[i]))
            postfix[j++] = infix[i];
        else {
            while (top != -1 && isOperator(stack[top]) && precedence(stack[top]) >=
precedence(infix[i]))
                postfix[j++] = stack[top--];
            stack[++top] = infix[i];
        }
    }

    while (top != -1)
        postfix[j++] = stack[top--];

    postfix[j] = '\0';
}

int main() {
    char infix[100], postfix[100];

    printf("Enter infix expression: ");
    scanf("%s", infix);

    infixToPostfix(infix, postfix);

    printf("Infix expression: %s\n", infix);
    printf("Postfix expression: %s\n", postfix);

    return 0;
}
```

OUTPUT:

```
Enter infix expression: a*(b+c+d)
Infix expression: a*(b+c+d)
Postfix expression: abc+d+
```

QUEUE:

2. Queue Implementation using Array:

```
#include <stdio.h>
#include <stdlib.h>

#define MAX_SIZE 10

int queue[MAX_SIZE];
int front = -1;
int rear = -1;

void enqueue(int value) {
    if (rear == MAX_SIZE - 1) {
        printf("Queue is full\n");
        return;
    }
    if (front == -1) {
        front = 0;
    }
    rear++;
    queue[rear] = value;
}

void dequeue() {
    if (front == -1) {
        printf("Queue is empty\n");
        return;
    }
    printf("Dequeued: %d\n", queue[front]);
    front++;
    if (front > rear) {
        front = -1;
        rear = -1;
    }
}

void display() {
    if (front == -1) {
        printf("Queue is empty\n");
        return;
    }
    printf("Queue: ");
    for (int i = front; i <= rear; i++) {
        printf("%d ", queue[i]);
    }
    printf("\n");
}

int main() {
    enqueue(1);
    enqueue(2);
    enqueue(3);
    display();
    dequeue();
    display();
    return 0;
}
```

OUTPUT:

```
Queue: 1 2 3
Dequeued: 1
Queue: 2 3
```

3. Queue implementation using linked list:

```
#include <stdio.h>
#include <stdlib.h>

// Node structure
typedef struct Node {
    int data;
    struct Node* next;
} Node;

// Queue structure
typedef struct Queue {
    Node* front;
    Node* rear;
} Queue;

// Function to create a new node
Node* createNode(int data) {
    Node* newNode = (Node*)malloc(sizeof(Node));
    newNode->data = data;
    newNode->next = NULL;
    return newNode;
}

// Function to initialize the queue
void initQueue(Queue* q) {
    q->front = NULL;
    q->rear = NULL;
}

// Function to check if the queue is empty
int isEmpty(Queue* q) {
    return q->front == NULL;
}

// Function to add an element to the queue
void enqueue(Queue* q, int data) {
    Node* newNode = createNode(data);
    if (isEmpty(q)) {
        q->front = newNode;
        q->rear = newNode;
    } else {
        q->rear->next = newNode;
        q->rear = newNode;
    }
}

// Function to remove an element from the queue
int dequeue(Queue* q) {
    if (isEmpty(q)) {
        printf("Queue is empty\n");
        return -1;
    }
    int data = q->front->data;
    Node* temp = q->front;
    q->front = q->front->next;
    if (q->front == NULL) {
        q->rear = NULL;
    }
    free(temp);
    return data;
}

// Function to display the queue
void display(Queue* q) {
    Node* temp = q->front;
    while (temp != NULL) {
        printf("%d ", temp->data);
        temp = temp->next;
    }
    printf("\n");
}

int main() {
    Queue q;
    initQueue(&q);
    enqueue(&q, 1);
    enqueue(&q, 2);
    enqueue(&q, 3);
    display(&q);
    printf("Dequeued: %d\n", dequeue(&q));
    display(&q);
    return 0;
}
```

OUTPUT:

```
Queue: 1 2 3
Dequeued: 1
Queue: 2 3
```