**Data Structure Day 3**

**Conversion of Infix to postfix**

#include <stdio.h>

#include <stdlib.h>

#include <ctype.h>

#include <string.h>

#define MAX 100

typedef struct Stack {

int top;

char items[MAX];

} Stack;

void initStack(Stack\* s) {

s->top = -1;

}

int isEmpty(Stack\* s) {

return s->top == -1;

}

int isFull(Stack\* s) {

return s->top == MAX - 1;

}

void push(Stack\* s, char item) {

if (isFull(s)) {

printf("Stack overflow\n");

return;

}

s->items[++s->top] = item;

}

char pop(Stack\* s) {

if (isEmpty(s)) {

printf("Stack underflow\n");

return '\0';

}

return s->items[s->top--];

}

char peek(Stack\* s) {

if (isEmpty(s)) {

return '\0';

}

return s->items[s->top];

}

int precedence(char op) {

switch (op) {

case '+':

case '-':

return 1;

case '\*':

case '/':

return 2;

case '^':

return 3;

default:

return 0;

}

}

int isOperator(char ch) {

return ch == '+' || ch == '-' || ch == '\*' || ch == '/' || ch == '^';

}

void infixToPostfix(char\* infix, char\* postfix) {

Stack stack;

initStack(&stack);

int i = 0, j = 0;

char ch;

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

if (isspace(ch)) continue;

if (isalnum(ch)) {

postfix[j++] = ch;

} else if (ch == '(') {

push(&stack, ch);

} else if (ch == ')') {

while (!isEmpty(&stack) && peek(&stack) != '(') {

postfix[j++] = pop(&stack);

}

if (!isEmpty(&stack) && peek(&stack) == '(') {

pop(&stack); // Pop the '('

}

} else if (isOperator(ch)) {

while (!isEmpty(&stack) && precedence(peek(&stack)) >= precedence(ch)) {

postfix[j++] = pop(&stack);

}

push(&stack, ch);

}

}

while (!isEmpty(&stack)) {

postfix[j++] = pop(&stack);

}

postfix[j] = '\0';

}

int main() {

char infix[MAX], postfix[MAX];

printf("Enter an infix expression: ");

fgets(infix, MAX, stdin);

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))+f

Postfix expression: ab+cde/\*-f+

**Queue Using Array**

#include <stdio.h>

#include <stdlib.h>

#define MAX 5

typedef struct {

int items[MAX];

int front;

int rear;

} Queue;

void initQueue(Queue\* q) {

q->front = -1;

q->rear = -1;

}

int isFull(Queue\* q) {

return q->rear == MAX - 1;

}

int isEmpty(Queue\* q) {

return q->front == -1 || q->front > q->rear;

}

void enqueue(Queue\* q, int item) {

if (isFull(q)) {

printf("Queue is full\n");

return;

}

if (q->front == -1) {

q->front = 0;

}

q->items[++q->rear] = item;

printf("%d enqueued to queue\n", item);

}

int dequeue(Queue\* q) {

if (isEmpty(q)) {

printf("Queue is empty\n");

return -1;

}

return q->items[q->front++];

}

void display(Queue\* q) {

if (isEmpty(q)) {

printf("Queue is empty\n");

return;

}

printf("Queue elements: ");

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

printf("%d ", q->items[i]);

}

printf("\n");

}

int main() {

Queue q;

initQueue(&q);

enqueue(&q, 10);

enqueue(&q, 20);

enqueue(&q, 30);

display(&q);

dequeue(&q);

display(&q);

return 0;

}

**Output:**

10 enqueued to queue

20 enqueued to queue

30 enqueued to queue

Queue elements: 10 20 30

10 dequeued from queue

Queue elements: 20 30

**Queue using Linked List**

#include <stdio.h>

#include <stdlib.h>

typedef struct Node {

int data;

struct Node\* next;

} Node;

typedef struct {

Node\* front;

Node\* rear;

} Queue;

Node\* newNode(int data) {

Node\* temp = (Node\*)malloc(sizeof(Node));

temp->data = data;

temp->next = NULL;

return temp;

}

void initQueue(Queue\* q) {

q->front = q->rear = NULL;

}

int isEmpty(Queue\* q) {

return q->front == NULL;

}

void enqueue(Queue\* q, int data) {

Node\* temp = newNode(data);

if (q->rear == NULL) {

q->front = q->rear = temp;

printf("%d enqueued to queue\n", data);

return;

}

q->rear->next = temp;

q->rear = temp;

printf("%d enqueued to queue\n", data);

}

int dequeue(Queue\* q) {

if (isEmpty(q)) {

printf("Queue is empty\n");

return -1;

}

Node\* temp = q->front;

int data = temp->data;

q->front = q->front->next;

if (q->front == NULL) {

q->rear = NULL;

}

free(temp);

printf("%d dequeued from queue\n", data);

return data;

}

void display(Queue\* q) {

if (isEmpty(q)) {

printf("Queue is empty\n");

return;

}

Node\* temp = q->front;

printf("Queue elements: ");

while (temp != NULL) {

printf("%d ", temp->data);

temp = temp->next;

}

printf("\n");

}

int main() {

Queue q;

initQueue(&q);

enqueue(&q, 10);

enqueue(&q, 20);

enqueue(&q, 30);

display(&q);

dequeue(&q);

display(&q);

return 0;

}

**Output:**

10 enqueued to queue

20 enqueued to queue

30 enqueued to queue

Queue elements: 10 20 30

10 dequeued from queue

Queue elements: 20 30