https://github.com/Parkjisu2021049002/homework5/upload
2021049002 원예과학과 박지수 hw5

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

#define MAX_QUEUE_SIZE 4

typedef char element;
typedef struct {
        element queue[MAX_QUEUE_SIZE];
        int front, rear;
}QueueType;


QueueType *createQueue();
int freeQueue(QueueType *cQ);
int isEmpty(QueueType *cQ);
int isFull(QueueType *cQ);
void enQueue(QueueType *cQ, element item);
void deQueue(QueueType *cQ, element* item);
void printQ(QueueType *cQ);
void debugQ(QueueType *cQ);
element getElement();
int main(void)
{ printf("%d \n", 2021049002);
 printf("박지수");
        QueueType *cQ = createQueue();
        element data;
        char command;

        do{

printf("\n-----------------------------------------------------\n");
                printf("                Circular Q                \n");

printf("-----------------------------------------------------\n");
                printf(" Insert=i,  Delete=d,  PrintQ=p,  Debug=b,  Quit=q \n");

printf("-----------------------------------------------------\n");
```

```
		printf("Command = ");
		scanf(" %c", &command);

		switch(command) {
		case 'i': case 'I':
			data = getElement();
			enQueue(cQ, data);
			break;
		case 'd': case 'D':
			deQueue(cQ, &data);
			break;
		case 'p': case 'P':
			printQ(cQ);
			break;
		case 'b': case 'B':
			debugQ(cQ);
			break;
		case 'q': case 'Q':
		freeQueue(cQ);
			break;
		default:
			printf("\n	>>>>>	Concentration!!	<<<<<	\n");
			break;
		}

	}while(command != 'q' && command != 'Q');


	return 1;
}

QueueType *createQueue()
{
	QueueType *cQ;
	cQ = (QueueType *)malloc(sizeof(QueueType));
	cQ->front = 0;
	cQ->rear = 0;
	return cQ;
}

int freeQueue(QueueType *cQ)
```

```
{
    if(cQ == NULL) return 1;
    free(cQ);
    return 1;
}

element getElement()
{
        element item;
        printf("Input element = ");
        scanf(" %c", &item);
        return item;
}


/* complete the function */
int isEmpty(QueueType *cQ)
{

    return 0;
}

/* complete the function */
int isFull(QueueType *cQ)
{
   return 0;
}


/* complete the function */
void enQueue(QueueType *cQ, element item)
{
        return 0;
}

/* complete the function */
void deQueue(QueueType *cQ, element *item)
{
    return 0;
}
```

```
void printQ(QueueType *cQ)
{
	int i, first, last;

	first = (cQ->front + 1)%MAX_QUEUE_SIZE;
	last = (cQ->rear + 1)%MAX_QUEUE_SIZE;

	printf("Circular Queue : [");

	i = first;
	while(i != last){
		printf("%3c", cQ->queue[i]);
		i = (i+1)%MAX_QUEUE_SIZE;

	}
	printf(" ]\n");
}


void debugQ(QueueType *cQ)
{

	printf("\n---DEBUG\n");
	for(int i = 0; i < MAX_QUEUE_SIZE; i++)
	{
		if(i == cQ->front) {
			printf("  [%d] = front\n", i);
			continue;
		}
		printf("  [%d] = %c\n", i, cQ->queue[i]);

	}
	printf("front = %d, rear = %d\n", cQ->front, cQ->rear);
}
```

```
/**
 * postfix.c
 *
 * School of Computer Science,
 * Chungbuk National University
 */
#include<stdio.h>
#include<stdlib.h>
#include<ctype.h>
#include<string.h>
#define MAX_STACK_SIZE 10
#define MAX_EXPRESSION_SIZE 20
/* stack 내에서 우선순위는 내림차순, lparen = 0 가장 낮음 */
typedef enum{
lparen = 0, /* ( 왼쪽 괄호 */
rparen = 9, /* ) 오른쪽 괄호*/
times = 7, /* * 곱셈 */
divide = 6, /* / 나눗셈 */
plus = 5, /* + 덧셈 */
minus = 4, /* - 뺄셈 */
operand = 1 /* 피연산자 */
} precedence;
char infixExp[MAX_EXPRESSION_SIZE];
char postfixExp[MAX_EXPRESSION_SIZE];
char postfixStack[MAX_STACK_SIZE];
int evalStack[MAX_STACK_SIZE];
int postfixStackTop = -1;
int evalStackTop = -1;
int evalResult = 0;
void postfixPush(char x)
char postfixPop()
void evalPush(int x)
int evalPop()
void getInfix()
precedence getToken(char symbol)
precedence getPriority(char x)
void charCat(char* c)
void toPostfix()
```

```
void debug()
void reset()
void evaluation()
int main()
{ printf("%d \n", 2021049002);
 printf("박지수");
char command;
do{
printf("----------------------------------------------------------------
\n");
printf(" Infix to Postfix, then Evaluation
\n");
printf("----------------------------------------------------------------
\n");
printf(" Infix=i, Postfix=p, Eval=e, Debug=d, Reset=r,
Quit=q \n");
printf("----------------------------------------------------------------
\n");
printf("Command = ");
scanf(" %c", &command);
switch(command) {
case 'i': case 'I':
getInfix();
break;
case 'p': case 'P':
toPostfix();
break;
case 'e': case 'E':
evaluation();
break;
case 'd': case 'D':
debug();
break;
case 'r': case 'R':
reset();
break;
case 'q': case 'Q':
break;
default:
printf("\n >>>>> Concentration!! <<<<<
\n");
```

```
break;
}
}while(command != 'q' && command != 'Q');
return 1;
}
void postfixPush(char x)
{
 postfixStack[++postfixStackTop] = x;
}
char postfixPop()
{
char x;
 if(postfixStackTop == -1)
 return '\0';
 else {
 x = postfixStack[postfixStackTop--];
 }
 return x;
}
void evalPush(int x)
{
 evalStack[++evalStackTop] = x;
}
int evalPop()
{
 if(evalStackTop == -1)
 return -1;
 else
 return evalStack[evalStackTop--];
}
/**
 * infix expression을 입력받는다.
 * infixExp에는 입력된 값을 저장한다.
 */
void getInfix()
{
 printf("Type the expression >>> ");
 scanf("%s",infixExp);
}
precedence getToken(char symbol)
{
```

```
switch(symbol) {
case '(' : return lparen;
case ')' : return rparen;
case '+' : return plus;
case '-' : return minus;
case '/' : return divide;
case '*' : return times;
default : return operand;
}
}
precedence getPriority(char x)
{
return getToken(x);
}
/**
 * 문자하나를 전달받아, postfixExp에 추가
 */
void charCat(char* c)
{
if (postfixExp == '\0')
strncpy(postfixExp, c, 1);
else
strncat(postfixExp, c, 1);
}
/**
 * infixExp의 문자를 하나씩 읽어가면서 stack을 이용하여 postfix로 변경한다.
 * 변경된 postfix는 postFixExp에 저장된다.
 */
void toPostfix()
{
/* infixExp의 문자 하나씩을 읽기위한 포인터 */
char *exp = infixExp;
char x; /* 문자하나를 임시로 저장하기 위한 변수 */
/* exp를 증가시켜가면서, 문자를 읽고 postfix로 변경 */
while(*exp != '\0')
{
if(getPriority(*exp) == operand)
{
x = *exp;
 charCat(&x);
}
```

```
 else if(getPriority(*exp) == lparen) {
 postfixPush(*exp);
 }
 else if(getPriority(*exp) == rparen)
 {
 while((x = postfixPop()) != '(') {
 charCat(&x);
 }
 }
 else
 {
 while(getPriority(postfixStack[postfixStackTop]) >=
getPriority(*exp))
 {
 x = postfixPop();
 charCat(&x);
 }
 postfixPush(*exp);
 }
 exp++;
}
 while(postfixStackTop != -1)
 {
 x = postfixPop();
 charCat(&x);
 }
}
void debug()
{
printf("\n---DEBUG\n");
printf("infixExp = %s\n", infixExp);
printf("postExp = %s\n", postfixExp);
printf("eval result = %d\n", evalResult);
printf("postfixStack : ");
for(int i = 0; i < MAX_STACK_SIZE; i++)
printf("%c ", postfixStack[i]);
printf("\n");
}
void reset()
{
infixExp[0] = '\0';
```

```
postfixExp[0] = '\0';
for(int i = 0; i < MAX_STACK_SIZE; i++)
postfixStack[i] = '\0';

postfixStackTop = -1;
evalStackTop = -1;
evalResult = 0;
}
void evaluation()
{
int opr1, opr2, i;
int length = strlen(postfixExp);
char symbol;
evalStackTop = -1;
for(i = 0; i < length; i++)
{
symbol = postfixExp[i];
if(getToken(symbol) == operand) {
evalPush(symbol - '0');
}
else {
opr2 = evalPop();
opr1 = evalPop();
switch(getToken(symbol)) {
case plus: evalPush(opr1 + opr2); break;
case minus: evalPush(opr1 - opr2); break;
case times: evalPush(opr1 * opr2); break;
case divide: evalPush(opr1 / opr2); break;
default: break;
}
}
}
evalResult = evalPop();
}
```