/\*

240401 큐

\*/

// 함수 정의

객체: 0개 이상의 요소들로 구성된 선형 리스트

∙연산:

▪ create(max\_size) ::= 최대 크기가 max\_size인 공백큐를 생성한다.

▪ init(q) ::= 큐를 초기화한다.

▪ is\_empty(q) ::= if(size == 0) return TRUE;

else return FALSE;

▪ is\_full(q) ::= if(size == max\_size) return TRUE;

else return FALSE;

▪ enqueue(q, e) ::= if( is\_full(q) ) queue\_full 오류;

else q의 끝에 e를 추가한다.

▪ dequeue(q) ::= if( is\_empty(q) ) queue\_empty 오류;

else q의 맨 앞에 있는 e를 제거하여 반환한다.

▪ peek(q) ::= if( is\_empty(q) ) queue\_empty 오류;

else q의 맨 앞에 있는 e를 읽어서 반환한다.

// 1번 예제 : 선형큐 실습코드

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <stdio.h>

#include <stdlib.h>

#define MAX\_QUEUE\_SIZE 5

typedef int element;

typedef struct { // 큐 타입

int front;

int rear;

element data[MAX\_QUEUE\_SIZE];

} QueueType;

void error(const char\* message) // 오류 함수

{

fprintf(stderr, "%s\n", message);

exit(1);

}

void init\_queue(QueueType\* q)

{

q->rear = -1;

q->front = -1;

}

void queue\_print(QueueType\* q)

{

for (int i = 0; i < MAX\_QUEUE\_SIZE; i++) {

if (i <= q->front || i > q->rear) printf(" | ");

else printf("%d | ", q->data[i]);

}

printf("\n");

}

int is\_full(QueueType\* q)

{

if (q->rear == MAX\_QUEUE\_SIZE - 1) return 1;

else return 0;

}

int is\_empty(QueueType\* q)

{

if (q->front == q->rear) return 1;

else return 0;

}

void enqueue(QueueType\* q, int item)

{

if (is\_full(q)) {

error("큐가 포화상태입니다.");

return;

}

q->data[++(q->rear)] = item;

}

int dequeue(QueueType\* q)

{

if (is\_empty(q)) {

error("큐가 공백상태입니다.");

return -1;

}

int item = q->data[++(q->front)];

return item;

}

int main(void)

{

int item = 0;

QueueType q;

init\_queue(&q);

enqueue(&q, 10); queue\_print(&q);

enqueue(&q, 20); queue\_print(&q);

enqueue(&q, 30); queue\_print(&q);

item = dequeue(&q); queue\_print(&q);

item = dequeue(&q); queue\_print(&q);

item = dequeue(&q); queue\_print(&q);

return 0;

}

원형큐 코드

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <stdio.h>

#include <stdlib.h>

// ===== 원형큐 코드 시작 ======

#define MAX\_QUEUE\_SIZE 5

typedef int element;

typedef struct { // 큐 타입

element data[MAX\_QUEUE\_SIZE];

int front, rear;

} QueueType;

// 오류 함수

void error(const char\* message)

{

fprintf(stderr, "%s\n", message);

exit(1);

}

// 초기화 함수

void init\_queue(QueueType\* q)

{

q->front = q->rear = 0;

}

// 공백 상태 검출 함수

int is\_empty(QueueType\* q)

{

return (q->front == q->rear);

}

// 포화 상태 검출 함수

int is\_full(QueueType\* q)

{

return ((q->rear + 1) % MAX\_QUEUE\_SIZE == q->front);

}

// 원형큐 출력 함수

void queue\_print(QueueType\* q)

{

printf("QUEUE(front=%d rear=%d) = ", q->front, q->rear);

if (!is\_empty(q)) {

int i = q->front;

do {

i = (i + 1) % (MAX\_QUEUE\_SIZE);

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

if (i == q->rear)

break;

} while (i != q->front);

}

printf("\n");

}

// 삽입 함수

void enqueue(QueueType\* q, element item)

{

if (is\_full(q))

error("큐가 포화상태입니다");

q->rear = (q->rear + 1) % MAX\_QUEUE\_SIZE;

q->data[q->rear] = item;

}

// 삭제 함수

element dequeue(QueueType\* q)

{

if (is\_empty(q))

error("큐가 공백상태입니다");

q->front = (q->front + 1) % MAX\_QUEUE\_SIZE;

return q->data[q->front];

}

// peek 함수

element peek(QueueType\* q)

{

if (is\_empty(q))

error("큐가 공백상태입니다");

return q->data[(q->front + 1) % MAX\_QUEUE\_SIZE];

}

// ===== 원형큐 코드 끝 ======

int main(void)

{

QueueType queue;

int element;

init\_queue(&queue);

printf("--데이터 추가 단계--\n");

while (!is\_full(&queue))

{

printf("정수를 입력하시오: ");

scanf("%d", &element);

enqueue(&queue, element);

queue\_print(&queue);

}

printf("큐는 포화상태입니다.\n\n");

printf("--데이터 삭제 단계--\n");

while (!is\_empty(&queue))

{

element = dequeue(&queue);

printf("꺼내진 정수: %d \n", element);

queue\_print(&queue);

}

printf("큐는 공백상태입니다.\n");

return 0;

}

덱(deque) 프로그램

// 정의

∙객체: n개의 element형으로 구성된 요소들의 순서있는 모임

∙연산:

▪ create() ::= 덱을 생성한다.

▪ init(dq) ::= 덱을 초기화한다.

▪ is\_empty(dq) ::= 덱이 공백상태인지를 검사한다.

▪ is\_full(dq) ::= 덱이 포화상태인지를 검사한다.

▪ add\_front(dq, e) ::= 덱의 앞에 요소를 추가한다.

▪ add\_rear(dq, e) ::= 덱의 뒤에 요소를 추가한다.

▪ delete\_front(dq) ::= 덱의 앞에 있는 요소를 반환한 다음 삭제한다

▪ delete\_rear(dq) ::= 덱의 뒤에 있는 요소를 반환한 다음 삭제한다.

▪ get\_front(q) ::= 덱의 앞에서 삭제하지 않고 앞에 있는 요소를 반환한다.

▪ get\_rear(q) ::= 덱의 뒤에서 삭제하지 않고 뒤에 있는 요소를 반환한다.

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <stdio.h>

#include <stdlib.h>

#define MAX\_QUEUE\_SIZE 5

typedef int element;

typedef struct { // 큐 타입

element data[MAX\_QUEUE\_SIZE];

int front, rear;

} DequeType;

// 오류 함수

void error(const char\* message)

{

fprintf(stderr, "%s\n", message);

exit(1);

}

// 초기화

void init\_deque(DequeType\* q)

{

q->front = q->rear = 0;

}

// 공백 상태 검출 함수

int is\_empty(DequeType\* q)

{

return (q->front == q->rear);

}

// 포화 상태 검출 함수

int is\_full(DequeType\* q)

{

return ((q->rear + 1) % MAX\_QUEUE\_SIZE == q->front);

}

// 원형큐 출력 함수

void deque\_print(DequeType\* q)

{

printf("DEQUE(front=%d rear=%d) = ", q->front, q->rear);

if (!is\_empty(q)) {

int i = q->front;

do {

i = (i + 1) % (MAX\_QUEUE\_SIZE);

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

if (i == q->rear)

break;

} while (i != q->front);

} printf("\n");

}

// 삽입 함수 – 기존 enqueue 함수

void add\_rear(DequeType\* q, element item)

{

if (is\_full(q))

error("큐가 포화상태입니다");

q->rear = (q->rear + 1) % MAX\_QUEUE\_SIZE;

q->data[q->rear] = item;

}

// 삭제 함수 – 기존 dequeue 함수

element delete\_front(DequeType\* q)

{

if (is\_empty(q))

error("큐가 공백상태입니다");

q->front = (q->front + 1) % MAX\_QUEUE\_SIZE;

return q->data[q->front];

}

// peek 함수 – 기존 peek 함수

element get\_front(DequeType\* q)

{

if (is\_empty(q))

error("큐가 공백상태입니다");

return q->data[(q->front + 1) % MAX\_QUEUE\_SIZE];

}

void add\_front(DequeType\* q, element val)

{

if (is\_full(q))

error("큐가 포화상태입니다");

q->data[q->front] = val;

q->front = (q->front - 1 + MAX\_QUEUE\_SIZE) % MAX\_QUEUE\_SIZE;

}

element delete\_rear(DequeType\* q)

{

int prev = q->rear;

if (is\_empty(q))

error("큐가 공백상태입니다");

q->rear = (q->rear - 1 + MAX\_QUEUE\_SIZE) % MAX\_QUEUE\_SIZE;

return q->data[prev];

}

element get\_rear(DequeType\* q)

{

if (is\_empty(q))

error("큐가 공백상태입니다");

return q->data[q->rear];

}

int main(void)

{

DequeType queue;

init\_deque(&queue);

for (int i = 0; i < 3; i++) {

add\_front(&queue, i);

deque\_print(&queue);

}

for (int i = 0; i < 3; i++) {

delete\_rear(&queue);

deque\_print(&queue);

}

return 0;

}