2020-2
Week7

데이터구조및실습

7주차 실습

연습1-1

//배열이용한 선형큐

```c
#include <stdio.h>
#include <stdlib.h>
#define MAX_QUEUE_SIZE 5

typedef int element;
typedef struct {
    int front;
    int rear;
    element data[MAX_QUEUE_SIZE];
}QueueType;

void error(const char *message) {
    fprintf(stderr, "%s₩n", message);
    exit(1);
}

void init_queue(QueueType*q) { // queue 초기화
    q->front = -1;
    q->rear = -1;
}

bool is_full(QueueType*q) {
    return (q->rear == MAX_QUEUE_SIZE - 1);
}

bool is_empty(QueueType*q) {
    return (q->front == q->rear);
}

void enqueue(QueueType*q, element item) {
    if (is_full(q)) {
        error("queue is full");
        return;
    }
    else q->data[++q->rear] = item;     // rear 를 증가시키고 값 대입
}
```

연습1-2

//배열이용한 선형큐

```
element dequeue(QueueType*q) {
    if (is_empty(q)) {
        error("queue is empty");
        return -1;
    }
    else return q->data[++(q->front)]; // front를 증가시키고 증가된 front가 가리키는 값 반환
}

void print_queue(QueueType* q) {
    for (int i = 0; i < MAX_QUEUE_SIZE; i++) {
        if (i <= q->front || i > q->rear) printf("    |"); // queue가 비었으면 빈 상태로 출력
        else printf("%3d |", q->data[i]);
    }
    printf("₩n");
}

int main(void)
{
    int item = 0;
    QueueType q;
    init_queue(&q);
    enqueue(&q, 10); print_queue(&q);
    enqueue(&q, 20); print_queue(&q);
    enqueue(&q, 30); print_queue(&q);

    item = dequeue(&q); print_queue(&q);
    item = dequeue(&q); print_queue(&q);
    item = dequeue(&q); print_queue(&q);
    item = dequeue(&q); print_queue(&q);
    return 0;
}
```

```
 10 |    |    |    |    |
 10 | 20 |    |    |    |
 10 | 20 | 30 |    |    |
    | 20 | 30 |    |    |
    |    | 30 |    |    |
    |    |    |    |    |
queue is empty
```

연습2-1

//배열이용한 원형큐

```c
#include <stdio.h>
#include <stdlib.h>
#define MAX_QUEUE_SIZE 5

typedef int element;
typedef struct {
    int front;
    int rear;
    element data[MAX_QUEUE_SIZE];
}QueueType;

void error(const char* message) {
    fprintf(stderr, "%s₩n", message);
    exit(1);
}

void init_queue(QueueType* q) { // queue 초기화
    q->front = 0;
    q->rear = 0;
}

bool is_full(QueueType* q) {
    return ((q->rear + 1) % MAX_QUEUE_SIZE == q->front); // rear를 증가시킨 값이 front와 같으면 full
}

bool is_empty(QueueType* q) {
    return (q->front == q->rear); // front와 rear가 같으면 empty
}

void enqueue(QueueType* q, element item) {
    if (is_full(q)) error("queue is full");
    q->rear = (q->rear+1) % MAX_QUEUE_SIZE;     //rear를 증가시킴
    q->data[q->rear] = item;                    // 증가된 rear위치에 값 대입
}

element dequeue(QueueType* q) {
    if (is_empty(q)) error("queue is empty");
    q->front = (q->front + 1) % MAX_QUEUE_SIZE; // front를 증가시킴
    return q->data[q->front];                   // 증가된 front위치에 있는 값 반환
}
```

연습2-2

//배열이용한 원형큐

```c
void print_queue(QueueType* q) {
    printf("Queue(front:%d, rear: %d) = ", q->front, q->rear); // front와 rear 위치 출력
    if (!is_empty(q)) {                    // 큐에 데이터가 있으면
        int i = q->front;                  // i는 현재의 front부터 시작
        do {
            i = (i + 1) % MAX_QUEUE_SIZE;  // front+1 부터 출력(front는 비어있음)
            printf("%3d |", q->data[i]);
            if (i == q->rear) break;       // 증가시킨 위치가 rear와 동일하면 다 출력한 것이므로 종료
        } while (i != q->front);           // 최대 한바퀴 돌아서 front위치까지 돌아오면 종료
    }
    printf("₩n");
}

int main(void)
{
    QueueType q;
    init_queue(&q);
    int element;

    printf("<데이터 추가>₩n");
    while (!is_full(&q)) {
        printf("정수입력: ");
        scanf("%d", &element);
        enqueue(&q, element);
        print_queue(&q);
    }
    printf("Queue is full.₩n");
    printf("<데이터 가져오기>₩n");
    while (!is_empty(&q)) {
        printf("%3d₩n", dequeue(&q));
        print_queue(&q);
    }
    printf("Queue is empty.₩n");
    return 0;
}
```

```
<데이터 추가>
정수입력: 1
Queue(front:0, rear: 1) =   1 |
정수입력: 2
Queue(front:0, rear: 2) =   1 |  2 |
정수입력: 3
Queue(front:0, rear: 3) =   1 |  2 |  3 |
정수입력: 4
Queue(front:0, rear: 4) =   1 |  2 |  3 |  4 |
Queue is full.
<데이터 가져오기>
  1
Queue(front:1, rear: 4) =   2 |  3 |  4 |
  2
Queue(front:2, rear: 4) =   3 |  4 |
  3
Queue(front:3, rear: 4) =   4 |
  4
Queue(front:4, rear: 4) =
Queue is empty.
```

연습3-1

//연결리스트 큐

```c
#include <stdio.h>
#include <stdlib.h>
#define MAX_QUEUE_SIZE 5

typedef int element;
typedef struct QueueNode {
    element data;
    struct QueueNode* link;
}QueueNode;

typedef struct {
    QueueNode* front, * rear;          // front 노드와 rear 노드 저장
}LinkedQueueType;

void error(const char* message) {
    fprintf(stderr, "%s₩n", message);
    exit(1);
}

void init_queue(LinkedQueueType* q) { // queue 초기화
    q->front = NULL;
    q->rear = NULL;
}

bool is_full(LinkedQueueType* q) {
    return 0;
}

bool is_empty(LinkedQueueType* q) {
    return (q->front == NULL); // front가 NULL이면 empty
}
```

연습3-2

```
void enqueue(LinkedQueueType* q, element item) {
    QueueNode* temp = (QueueNode*)malloc(sizeof(QueueNode)); // 새 노드 동적할당 //연결리스트 큐
    temp->data = item;                  // 데이터 저장
    temp->link = NULL;                  // 링크는 null
    if (is_empty(q)) {                  // queue가 empty면 새로 추가된 node가 front이자 rear임
        q->front = temp;
        q->rear = temp;
    }
    else {                              // 공백이 아니면
        q->rear->link= temp;            // 기존의 rear가 새로 추가된 노드를 가리키게 하고
        q->rear = temp;                 // 새로 추가된 node가 rear가 됨
    }
}

element dequeue(LinkedQueueType* q) {
    if (is_empty(q)) error("queue is empty");
    element data;
    QueueNode* temp = q->front;         // 기존의 front가 가리키던 노드 위치를 저장해놓고
    data = temp->data;                  // 기존의 front 가리키던 값을 반환을 위해 저장
    q->front = q->front->link;          // 기존의 front가 가리키던 노드가 새로운 front가 됨
    if (q->front == NULL) q->rear = NULL; // 노드가 한 개 있는 경우라면 queue를 공백으로 만들어줌
    free(temp);                         // 기존 front 노드 메모리 반환
    return data;                        // dequeue 된 값 반환
}
```

연습3-3

//연결리스트 큐

```
void print_queue(LinkedQueueType* q) {
    for (QueueNode*p=q->front; p != NULL ; p= p->link) {
        printf("%d -> ", p->data);
    }
    printf("NULL₩n");
}

int main(void)
{
    int item = 0;
    LinkedQueueType q;
    init_queue(&q);

    printf("<데이터 추가>₩n");
    enqueue(&q, 1); print_queue(&q);
    enqueue(&q, 2); print_queue(&q);
    enqueue(&q, 3); print_queue(&q);

    printf("<데이터 가져오기>₩n");
    dequeue(&q);    print_queue(&q);
    dequeue(&q);    print_queue(&q);
    dequeue(&q);    print_queue(&q);
    dequeue(&q);    print_queue(&q);
    return 0;
}
```

```
<데이터 추가>
1 -> NULL
1 -> 2 -> NULL
1 -> 2 -> 3 -> NULL
<데이터 가져오기>
2 -> 3 -> NULL
3 -> NULL
NULL
queue is empty
```

연습4-1

//배열이용한 덱

```c
#include <stdio.h>
#include <stdlib.h>
#define MAX_DEQUE_SIZE 5

typedef int element;
typedef struct {
    int front;
    int rear;
    element data[MAX_DEQUE_SIZE];
}DequeType;

void error(const char* message) {
    fprintf(stderr, "%s₩n", message);
    exit(1);
}

void init_deque(DequeType* dq) { // deque 초기화
    dq->front = 0;
    dq->rear = 0;
}

bool is_full(DequeType* dq) {
    return ((dq->rear + 1) % MAX_DEQUE_SIZE == dq->front); // rear를 증가시킨 값이 front와 같으면 full
}

bool is_empty(DequeType* dq) {
    return (dq->front == dq->rear);                // front와 rear가 같으면 empty
}
```

연습4-2

//배열이용한 덱

```
void add_rear(DequeType* dq, element item) {          // enqueue와 동일
    if (is_full(dq)) error("deque is full");
    dq->rear = (dq->rear+1) % MAX_DEQUE_SIZE;         //rear를 증가시킴
    dq->data[dq->rear] = item;                        // 증가된 rear위치에 값 대입
}

void add_front(DequeType* dq, element item) {
    if (is_full(dq)) error("deque is full");
    dq->data[dq->front] = item;                       // 기존의 front위치에 값 대입
    dq->front = (dq->front - 1 + MAX_DEQUE_SIZE) % MAX_DEQUE_SIZE; // front를 감소시켜 새로운 front로 변경
}

element get_rear(DequeType*dq){
    if (is_empty(dq)) error("deque is empty");
    return dq->data[dq->rear];                        // rear 위치에 있는 값 반환(rear는 그대로)
}

element del_rear(DequeType* dq) {
    if (is_empty(dq)) error("deque is empty");
    int prev = dq->rear;                              // 기존의 rear위치 임시저장
    dq->rear = (dq->rear - 1 + MAX_DEQUE_SIZE) % MAX_DEQUE_SIZE; // rear를 감소시켜 새로운 rear로
    return  dq->data[prev];                               // 기존의 rear위치에 있는 값 반환
}

element del_front(DequeType* dq) {                    // dedeque와 동일
    if (is_empty(dq)) error("deque is empty");
    dq->front = (dq->front + 1) % MAX_DEQUE_SIZE;     // front를 증가시킴
    return dq->data[dq->front];                       // 증가된 front위치에 있는 값 반환
}

element get_front(DequeType* dq) {
    if (is_empty(dq)) error("deque is empty");
    return dq->data[(dq->front + 1) % MAX_DEQUE_SIZE]; // front 다음 위치에 있는 값 반환(front는 그대로)
}
```

연습4-3

//배열이용한 덱

```
void print_deque(DequeType* dq) {
    printf("Deque(front:%d, rear: %d) = ", dq->front, dq->rear); // front와 rear 위치 출력
    if (!is_empty(dq)) {                        // 덱에 데이터가 있으면
        int i = dq->front;                      // i는 현재의 front부터 시작
        do {
            i = (i + 1) % MAX_DEQUE_SIZE;       // front+1 부터 출력(front는 비어있으므로)
            printf("%3d |", dq->data[i]);
            if (i == dq->rear) break;           // 증가시킨 위치가 rear와 동일하면 다 출력한 것이므로 종료
        } while (i != dq->front);               // 최대 한바퀴 돌아서 front위치까지 돌아오면 종료
    }
    printf("₩n");
}

int main(void)
{
    DequeType dq;
    init_deque(&dq);
    int element;

    printf("<front에 데이터 추가>₩n");
    for(int i = 0; i<3; i++) {
        printf("정수입력: ");
        scanf("%d", &element);
        add_front(&dq, element);
        print_deque(&dq);
    }
    printf("<rear에서 데이터 가져오기>₩n");
    for(int i = 0; i<4; i++) {
        printf("%3d₩n", del_rear(&dq));
        print_deque(&dq);
    }
    return 0;
}
```

```
<front에 데이터 추가>
정수입력: 0
Deque(front:4, rear: 0) =   0 |
정수입력: 1
Deque(front:3, rear: 0) =   1 |  0 |
정수입력: 2
Deque(front:2, rear: 0) =   2 |  1 |  0 |
<rear에서 데이터 가져오기>
  0
Deque(front:2, rear: 4) =   2 |  1 |
  1
Deque(front:2, rear: 3) =   2 |
  2
Deque(front:2, rear: 2) =
deque is empty
계속하려면 아무 키나 누르십시오 . . .
```

연습5

//원형큐활용 버퍼(main)

```c
int main(void)
{
    QueueType q;
    init_queue(&q);
    int element;
    srand(time(NULL));

    for (int i = 0; i < 50; i++) {
        if (rand() % 5 == 0) enqueue(&q, rand() % 100);
        print_queue(&q);
        if (rand() % 10 == 0) dequeue(&q);
        print_queue(&q);
    }
    return 0;
}
```

연습6-1

//이중연결리스트 덱

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

typedef int element;
typedef struct DequeNode { // 이중연결 노드구조
    element data;
    struct DequeNode *llink; // left link
    struct DequeNode *rlink; // right link
} DequeNode;

typedef struct {
    DequeNode* front;
    DequeNode* rear;
}DequeType;

void deque_init(DequeType*dq) { // 이중연결리스트 초기화
    dq->front = NULL;
    dq->rear = NULL;
}

void error(const char* X) {
    fprintf(stderr, "%s", X);
    exit(1);
}

bool is_empty(DequeType* dq) {
    return  (dq->front == NULL); // front가 NULL이면 빈 deque
}

bool is_full(DequeType* dq) {
    return  0;
}
```

연습6-2

//이중연결리스트 덱

```
void add_front(DequeType*dq, element val) { // 새로운 데이터를 front insert
    DequeNode*newnode = (DequeNode*)malloc(sizeof(DequeNode)); // 새로운 노드생성
    newnode->data = val;              // 값 대입
    if (dq->front == NULL) {          // 비어있는 덱이면
        newnode->rlink = NULL;        // 새로 insert되는 노드의 양쪽 link모두 NULL
        newnode->llink = NULL;
        dq->front = newnode;          // front와 rear 모두 새로 insert된 노드를 가리킴
        dq->rear = newnode;
    }
    else {                            // 기존에 노드가 있으면
        newnode->rlink = dq->front;   // 새로운 노드의 rlink는 기존의 front를 가리킴
        newnode->llink = NULL;        // 새로운 노드의 llink는 NULL(선형이중연결리스트)
        dq->front->llink = newnode;   // 기존 front의 llink가 새로운 노드를 가리킴
        dq->front = newnode;          // 새로운 노드가 new front가 됨
    }
}

void add_rear(DequeType* dq, element val) { // 새로운 데이터를 기존노드 pnode의 오른쪽에 insert
    DequeNode* newnode = (DequeNode*)malloc(sizeof(DequeNode)); // 새로운 노드생성
    newnode->data = val;              // 값 대입
    if (dq->front == NULL) {          // 비어있는 덱이면
        newnode->rlink = NULL;        // 새로 insert되는 노드의 양쪽 link모두 NULL
        newnode->llink = NULL;
        dq->front = newnode;          // front와 rear 모두 새로 insert된 노드를 가리킴
        dq->rear = newnode;
    }
    else {                            // 기존에 노드가 있으면
        newnode->llink = dq->rear;    // 새로운 노드의 llink가 기존의 rear를 가리킴
        newnode->rlink = NULL;        // 새로운 노드의 rlink는 NULL(선형이중연결리스트)
        dq->rear->rlink = newnode;    // 기존rear의 rlink가 새로운 노드를 가리킴
        dq->rear = newnode;           // 새로운 노드가 new rear가 됨
    }
}
```

연습6-3

//이중연결리스트 덱

```
element del_front(DequeType*dq) {   // 맨 앞에 있는 기존노드(front)를 delete
    DequeNode* removed = dq->front; // delete할 노드 주소를 저장(나중에 free목적)
    if (is_empty(dq)) return 0;     // 공백상태의 리스트
    else {                          // 노드가 있으면
        element temp = removed->data;   // data 저장(나중에 반환 목적)
        if (dq->front->rlink == NULL) { // 삭제할 노드가 유일한 노드이면
            dq->front = NULL;           // 새로운 front, rear는 모두 NULL(빈 덱)
            dq->rear = NULL;
        }
        else {                          // 노드가 두개 이상이면
            dq->front = dq->front->rlink;   // front를 하나 뒤로 이동
            dq->front->llink = NULL;        // 새로운 front의 llink는 NULL
        }
        free(removed);  // removed 메모리 반환
        return temp;    // 값 반환
    }
}

element del_rear(DequeType* dq) {   // 맨 마지막에 있는 기존노드(rear)를 delete
    DequeNode* removed = dq->rear;  // delete할 노드 주소를 저장(나중에 free목적)
    if (is_empty(dq)) return NULL;  // 공백상태의 덱
    else {                          // 노드가 있으면
        element temp = removed->data;   // data 저장(나중에 반환 목적)
        if (dq->rear->llink == NULL) {  // 노드가 하나밖에 없는 경우
            dq->rear = NULL;            // 새로운 front, rear는 모두 NULL(빈 덱)
            dq->front = NULL;
        }
        else {                          // 노드가 두개 이상이면
            dq->rear = dq->rear->llink; // rear를 하나 뒤로 이동
            dq->rear->rlink = NULL;     // 새로운 front의 rlink는 NULL
        }
        free(removed);  // removed 메모리 반환
        return temp;    // 값 반환
    }
}
```

연습6-4

//이중연결리스트 덱

```
element peek_front(DequeType* dq) { // 맨 앞에 있는 기존노드(front)를 peek
    if (is_empty(dq)) return NULL;  // 공백상태의 리스트
    return dq->front->data;         // front 노드의 값 반환
}

element peek_rear(DequeType* dq) {  // 맨 마지막에 있는 기존노드(rear)를 peek
    if (is_empty(dq)) return 0;     // 공백상태의 리스트
    return dq->rear->data;          // rear 노드의 값 반환
}

void print_deque(DequeType*dq) {    // 이중연결리스트 출력
    DequeNode*p = NULL;
    for (p=dq->front; p->rlink != NULL; p = p->rlink) // link를 이동해가면서 data 출력
        printf("%2d -", p->data);
    printf("%2d#n", p->data);
}

int main(void)
{
    DequeType* dq = (DequeType*) malloc(sizeof(DequeType));
    deque_init(dq);  // 덱 초기화
    printf("add front 1:    "); add_front(dq, 1); print_deque(dq);
    printf("add front 2:    "); add_front(dq, 2); print_deque(dq);
    printf("add rear  3:    "); add_rear(dq, 3); print_deque(dq);
    printf("del front  :    "); del_front(dq); print_deque(dq);
    printf("del rear   :    "); del_rear(dq); print_deque(dq);
    printf("add rear  4:    "); add_rear(dq, 4); print_deque(dq);
    printf("add front 5:    "); add_front(dq, 5); print_deque(dq);
    printf("add front 6:    "); add_front(dq, 6); print_deque(dq);
    printf("peek front: %d #n", peek_front(dq));
    printf("peek rear: %d #n", peek_rear(dq));
    return 0;
}
```

```
add front 1:     1
add front 2:     2 - 1
add rear  3:     2 - 1 - 3
del front  :     1 - 3
del rear   :     1
add rear  4:     1 - 4
add front 5:     5 - 1 - 4
add front 6:     6 - 5 - 1 - 4
peek front: 6
peek rear: 4
```