

**CHAPTER 13**

**큐 구현 및 응용 실습**

# 실습 1

## 문제 1)

정수를 저장하는 큐를 구현한 다음, 입력으로 주어지는 명령을 처리하는 프로그램을 만들어보자.

push : 정수를 큐에 넣는 연산

pop : 큐에서 정수를 빼고 출력한다. 큐에 정수가 없는 경우에는 -1 출력

size : 큐에 들어있는 정수의 개수 출력.

empty : 큐가 비어있으면 1, 아니면 0 출력

front : 큐의 가장 앞에 있는 정수 출력. 큐에 정수가 없는 경우에는 -1 출력

back : 큐의 가장 뒤에 있는 정수 출력. 큐에 정수가 없는 경우에는 -1 출력

```
15
push 1
push 2
front
1
back
2
size
2
empty
0
pop
1
pop
2
pop
-1
size
0
empty
1
pop
-1
push 3
empty
0
front
3
```

# 실습 1 정답

```
#define _CRT_SECURE_NO_WARNINGS
#include<stdio.h>
#include<string.h>
#define SIZE 10001
```

```
int queue[SIZE];
int front = -1, rear = -1;
```

```
void push(int data) {
    queue[++rear] = data;
}
```

```
int pop() {
    if (front == rear) {
        return -1;
    }
    else {
        return queue[++front];
    }
}
```

```
int size() {
    return rear - front;
}
```

```
int empty() {
    if (front == rear) {
        return 1;
    }
    else {
        return 0;
    }
}
```

```
int frontCheck() {
    if (front == rear) {
        return -1;
    }
    else {
        return queue[front + 1];
    }
}
```

```
int rearCheck() {
    if (front == rear) {
        return -1;
    }
    else {
        return queue[rear];
    }
}
```

```
int main() {
    int T;
    scanf("%d", &T);

    for (int i = 0; i < T; i++) {
        char sel[6];
        scanf("%s", sel);

        if (!strcmp(sel, "push")) {
            int data;
            scanf("%d", &data);
            push(data);
        }
        else if (!strcmp(sel, "pop")) {
            printf("%d\n", pop());
        }
        else if (!strcmp(sel, "size")) {
            printf("%d\n", size());
        }
        else if (!strcmp(sel, "empty")) {
            printf("%d\n", empty());
        }
        else if (!strcmp(sel, "front")) {
            printf("%d\n", frontCheck());
        }
        else {
            printf("%d\n", rearCheck());
        }
    }

    return 0;
}
```

## 실습 2

### 문제 2)

1번부터 N번까지 N명의 사람이 원을 이루면서 앉아있고, 양의 정수  $K(\leq N)$ 가 주어진다. 이제 순서대로 K번째 사람을 제거한다. 한 사람이 제거되면 남은 사람들로 이루어진 원을 따라 이 과정을 계속해 나간다. 이 과정은 N명의 사람이 모두 제거될 때까지 계속된다. 원에서 사람들이 제거되는 순서를 (N, K)-요세푸스 순열이라고 한다.

N과 K가 주어지면 (N, K)-요세푸스 순열을 구하는 프로그램을 작성하시오.

 Microsoft Visual Studio 디버그 콘솔

```
8 3
<3, 6, 1, 5, 2, 8, 4, 7>
```

# 실습 2 정답

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

#define SIZE 1001

typedef struct {
    int queue[SIZE];
    int front, rear;
}QueueType;

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

int is_full(QueueType* q)
{
    return ((q->rear + 1) % SIZE == q->front);
}

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

```
void push(QueueType* q, int e)
{
    if (is_full(q))
        return;

    q->rear = (q->rear + 1) % SIZE;
    q->queue[q->rear] = e;
}

int pop(QueueType* q)
{
    if (is_empty(q))
        return -1;

    q->front = (q->front + 1) % SIZE;
    return q->queue[q->front];
}

int size(QueueType* q)
{
    if (q->front < q->rear)
        return q->rear - q->front;
    else
        return SIZE - q->front + q->rear;
}
```

```
int main()
{
    QueueType Q; init_queue(&Q);

    int i, j, N, K, tmp;

    scanf("%d %d", &N, &K);

    for (i = 0; i < N; i++)
    {
        push(&Q, i + 1);
    }

    printf("<");
    for (i = 0; i < N; i++)
    {
        for (j = 0; j < K - 1; j++)
        {
            tmp = pop(&Q);
            push(&Q, tmp);
        }
        if (size(&Q) == 1)
            break;
        tmp = pop(&Q);
        printf("%d, ", tmp);
    }
    printf("%d>", pop(&Q));

    return 0;
}
```

## 실습 3

문제 3)

N장의 카드가 있다. 각각의 카드는 차례로 1부터 N까지의 번호가 붙어 있으며, 1번 카드가 제일 위에, N번 카드가 제일 아래인 상태로 순서대로 카드가 놓여 있다. 이제 다음과 같은 동작을 카드가 한 장 남을 때까지 반복하게 된다. 우선, 제일 위에 있는 카드를 바닥에 버린다. 그 다음, 제일 위에 있는 카드를 제일 아래에 있는 카드 밑으로 옮긴다.

N이 주어졌을 때, 제일 마지막에 남게 되는 카드를 구하는 프로그램을 작성하시오.



# 실습 3 정답

```
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>
#define size 500000
int que[size];
int main()
{
    int n, i, front = 0, rear;
    scanf("%d", &n);

    for (i = 0; i < n; i++) que[i] = i + 1;
    rear = n - 1;

    while (1) {
        front = (front + 1) % n;
        if (rear == front) break; // 확인
        rear = (rear + 1) % n;
        que[rear] = que[front];
        front = (front + 1) % n;
        if (rear == front) break; // 확인
    }
    printf("%d", que[rear]); // 출력
    return 0;
}
```

## 실습 4

### 문제 4)

참가인원을 입력 받고 인원 수만큼의 약실을 만든 다음, 랜덤한 위치에 총알 한 개를 넣는다. 총알이 발사되기 전까지 쏘거나 멈출 수 있다. 출력 예시는 다음과 같고, 이외의 상황은 없다고 가정한다.

```
참가 인원: 10
Shot(1) or Stop(0)
1
Shot(1) or Stop(0)
1
Shot(1) or Stop(0)
1
Shot(1) or Stop(0)
1
Shot(1) or Stop(0)
1
WASTED
Game Over!!!
```

```
참가 인원: 5
Shot(1) or Stop(0)
1
Shot(1) or Stop(0)
0
YOU ALIVE!!
```



# 실습 4 정답

```
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>
#include <malloc.h>
#include <stdlib.h>
#include <time.h>
typedef struct _Node {
    int bullet;//총알(1:장전, 0:없음)???
    struct _Node* next;
}Node;
int Game();
//게임
void MakeRoulette(Node** head, int n); //권총 약실이 n개인 러시아 룰렛 생성
void ClearRoulette(Node** head); //러시아 룰렛 지우기
int main() {
    int key;
    srand((unsigned)time(0)); //랜덤에 사용할 seed값 설정 ???

    Game();//게임??

}
```

```
,
int Game() {
    int n;
    int choice;
    Node* head;
    printf("참가 인원: ");
    scanf("%d", &n); //게임 참가 인원 입력
    MakeRoulette(&head, n); //참가 인원만큼의 러시아 룰렛 생성
    while (1) {
        printf("Shot(1) or Stop(0)\n");
        scanf_s("%d", &choice); //발사 여부 입력 ???????
        if (choice == 1) //발사를 선택할 때 ???????
        {
            if (head->bullet) //총알이 있을 때 ???????????
            {
                printf("WASTED\n");
                printf("Game Over!!!\n");
                ClearRoulette(&head); //러시아 룰렛 지우기
                return 0; //반복문 탈출(게임 종료) ???????????
            }
            else //없다면 ???????????
            {
                head = head->next; //다음으로 이동 ???????????
            }
        }
        else //발사를 선택하지 않았다면 ???????
        {
            printf("YOU ALIVE!!");
            return 1;
        }
    }
    return 0;
}
```

# 실습 4 정답

```
void MakeNode(Node** head, int bullet) {
    Node* now = (Node*)malloc(sizeof(Node));
    now->bullet = bullet;
    if ((*head) == NULL)//빈 상태???
    {
        (*head) = now;//맨앞이 now ???????
        now->next = now;//현재 now 하나여서 now->next도 now ???
    }
    else//비어있지 않음???
    {
        //(*head)와 (*head) 다음 노드 사이에 now를 연결???????
        now->next = (*head)->next;
        (*head)->next = now;
    }
}
```

```
void MakeRoulette(Node** head, int n) {
    int value = rand() % n + 1; //value에 1~n 사이의 랜덤한 수 발생 ???
    (*head) = NULL;//리스트의 맨 앞은 NULL로 초기화 ???
    while (n > 0)//만들 개수가 0보다 클 때???
    {
        if (value == n)//check와 n이 같으면 ???????
        {
            MakeNode(head, 1);//총알을 장전한 노드 생성????????
        }
        else//아니면 ???????
        {
            MakeNode(head, 0);//총알을 장전하지 않은 노드 추가
            가 ???????
        }
        n--;
    }
}
```

```
void ClearRoulette(Node** head) {
    Node* seek = (*head)->next;//(*head)의 다음 노드로 초기화
    Node* prev = (*head);//seek의 앞 노드를 (*head)로 설정???
    while (seek != (*head))//seek가 (*head)가 아니면 반복???
    {
        prev = seek;
        //seek를 이동하기 전에 prev에 설정 ???????
        seek = seek->next;
        //seek를 다음 노드로 이동
        free(prev); //설정했던 seek의 앞 노드를 제거 ???
    }
    free(*head); //마지막으로 (*head)노드를 제거
}
```

## 실습 5

문제 5)

다음 그림과 같은 결과를 출력할 수 있는 덱을 구현해라

```
Add front(1)/Add Rear(2)/Delete Front(3)/Delete Rear(4)/Quit(0) : 1
Input number : 2
2
Add front(1)/Add Rear(2)/Delete Front(3)/Delete Rear(4)/Quit(0) : 1
Input number : 3
3 | 2
Add front(1)/Add Rear(2)/Delete Front(3)/Delete Rear(4)/Quit(0) : 2
Input number : 5
3 | 2 | 5
Add front(1)/Add Rear(2)/Delete Front(3)/Delete Rear(4)/Quit(0) : 3
2 | 5
Add front(1)/Add Rear(2)/Delete Front(3)/Delete Rear(4)/Quit(0) : 4
2
Add front(1)/Add Rear(2)/Delete Front(3)/Delete Rear(4)/Quit(0) : 5
Error!

Add front(1)/Add Rear(2)/Delete Front(3)/Delete Rear(4)/Quit(0) : 0
```

# 실습 5 정답

```
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>

#include <stdlib.h>

#define TRUE 1
#define FALSE 0
#define ERROR -1
#define MAX_DEQUE_SIZE 5

typedef int boolean;
typedef int element;

typedef struct __duqueueType {
    element* data;

    int rear, front;
}Deque;

/*    덱 초기화    */
void init_deque(Deque *q) {
    q->data = (element*)malloc(sizeof(element)*MAX_DEQUE_SIZE);

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

```
/*    덱이 공백인지 검사    */
boolean is_empty(Deque* q) {
    return (q->front == q->rear);
}

/*    덱의 rear를 반환    */
element get_rear(Deque* q) {
    if (is_empty()) {
        printf("Empty\n");

        return ERROR;
    }

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

/*    덱의 앞의 요소를 반환    */
element get_front(Deque* q) {
    if (is_empty(q)) {
        printf("Empty\n");

        return ERROR;
    }

    return q->data[(q->front + 1) % MAX_DEQUE_SIZE];
}
```

```
/*    덱이 포화인지 검사    */
boolean is_full(Deque* q) {
    if (((q->rear + 1) % MAX_DEQUE_SIZE) == q->front) return TRUE;
    else return FALSE;
}

/*    덱의 뒤에 요소를 추가    */
void add_rear(Deque* q, element data) {
    if (is_full(q)) {
        printf("Full\n");

        return;
    }

    q->rear = (q->rear + 1) % MAX_DEQUE_SIZE;
    q->data[q->rear] = data;
}
```

# 실습 5 정답

```
/*   덱의 앞의 요소를 반환후 삭제   */
```

```
element delete_front(Deque* q) {  
    if (is_empty(q)) {  
        printf("Empty\n");  
        return ERROR;  
    }  
    element tmp = get_front(q);  
    q->front = (q->front + 1) % MAX_DEQUE_SIZE;  
    return tmp;  
}
```

```
/*   덱의 뒤에 요소를 반환후 삭제   */
```

```
element delete_rear(Deque* q) {  
    if (is_empty(q)) {  
        printf("Empty\n");  
        return ERROR;  
    }  
    element tmp = q->data[q->rear];  
    q->rear = (q->rear - 1 + MAX_DEQUE_SIZE) % MAX_DEQUE_SIZE;  
    return tmp;  
}
```

```
/*   덱의 앞에 요소를 추가   */
```

```
void add_front(Deque* q, element data) {  
    if (is_full(q)) {  
        printf("Full\n");  
        return;  
    }  
    q->data[q->front] = data;  
    q->front = (q->front - 1 + MAX_DEQUE_SIZE)  
        % MAX_DEQUE_SIZE;  
    return;  
}
```

```
/*   덱의 모든 요소 print   */
```

```
void deque_print(Deque* q) {  
    int i = (q->front + 1) % MAX_DEQUE_SIZE;  
    if (is_empty(q)) {  
        printf("Empty\n");  
        return;  
    }  
    while (i != q->rear) {  
        printf("%d | ", q->data[i]);  
        i = (i + 1) % MAX_DEQUE_SIZE;  
    }  
    printf("%d\n", q->data[i]);  
}
```

# 실습 5 정답


```
int main() {  
  
    Deque q;  
  
  
    init_deque(&q);  
  
    int i;  
    int cmd=99;  
    while (cmd) {  
        printf("Add front(1)/Add Rear(2)/Delete Front(3)/Delete Rear(4)/Quit(0) :  
");  
        scanf("%d", &cmd);  
  
        if (cmd == 1) {  
            printf("Input number : ");  
            scanf("%d", &i);  
            add_front(&q, i);  
            deque_print(&q);  
        }  
        else if (cmd == 2) {  
            printf("Input number : ");  
            scanf("%d", &i);  
            add_rear(&q, i);  
            deque_print(&q);  
        }  
        else if (cmd == 3) {  
            delete_front(&q);  
            deque_print(&q);  
        }  
        else if (cmd == 4) {  
            delete_rear(&q);  
  
            deque_print(&q);  
        }  
        else if (cmd == 0) {  
            break;  
        }  
        else {  
            printf("Error!WnWn");  
        }  
    }  
}
```

```
        deque_print(&q);  
    }  
    else if (cmd == 0) {  
        break;  
    }  
    else {  
        printf("Error!WnWn");  
    }  
}  
  
    free(q.data);  
  
    return 0;  
}
```


## 실습 6

문제 6)

덱을 이용하여 숫자로 이루어진 문자열이 회문인지 확인해라  
이때 문자열의 최대 크기는 100이다.

 Microsoft Visual Studio 디버

```
Enter the word : 12321  
회문입니다.
```

 Microsoft Visual Studio 디버그 콘

```
Enter the word : 12121212  
회문이 아닙니다.
```

# 실습 6 정답

```
#define _CRT_SECURE_NO_WARNINGS    /*    덱 초기화    */
#include <stdio.h>

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

#define TRUE 1
#define FALSE 0
#define ERROR -1
#define MAX_DEQUE_SIZE 100

typedef int boolean;
typedef int element;
typedef struct __duqueueType {
    element* data;

    int rear, front;
}Deque;

void init_deque(Deque* q) {
    q->data = (element*)malloc(sizeof(element) * MAX_DEQUE_SIZE);
    q->front = q->rear = 0;
}

/*    덱이 공백인지 검사    */
boolean is_empty(Deque* q) {
    return (q->front == q->rear);
}

/*    덱의 rear를 반환    */
element get_rear(Deque* q) {
    if (is_empty) {
        printf("Empty\n");
        return ERROR;
    }
    return q->data[q->rear];
}

/*    덱의 앞의 요소를 반환    */
element get_front(Deque* q) {
    if (is_empty(q)) {
        printf("Empty\n");
        return ERROR;
    }
    return q->data[(q->front + 1) % MAX_DEQUE_SIZE];
}

/*    덱이 포화인지 검사    */
boolean is_full(Deque* q) {
    if (((q->rear + 1) % MAX_DEQUE_SIZE) == q->front) return TRUE;
    else return FALSE;
}
```



# 실습 6 정답

```
/*      덱의 뒤에 요소를 추가      */  
  
void add_rear(Deque* q, element data) {  
    if (is_full(q)) {  
        printf("Full\n");  
        return;  
    }  
  
    q->rear = (q->rear + 1) % MAX_DEQUE_SIZE;  
    q->data[q->rear] = data;  
}
```

```
/*      덱의 앞의 요소를 반환후 삭제      */  
  
element delete_front(Deque* q) {  
    if (is_empty(q)) {  
        printf("Empty\n");  
        return ERROR;  
    }  
  
    element tmp = get_front(q);  
    q->front = (q->front + 1) % MAX_DEQUE_SIZE;  
  
    return tmp;  
}  
  
/*      덱의 뒤에 요소를 반환후 삭제      */  
  
element delete_rear(Deque* q) {  
    if (is_empty(q)) {  
        printf("Empty\n");  
        return ERROR;  
    }  
  
    element tmp = q->data[q->rear];  
    q->rear = (q->rear - 1 + MAX_DEQUE_SIZE) % MAX_DEQUE_SIZE;  
  
    return tmp;  
}
```

# 실습 6 정답

```
/*      덱의 앞에 요소를 추가      */  
  
void add_front(Deque* q, element data) {  
  
    if (is_full(q)) {  
  
        printf("Full\n");  
  
        return;  
  
    }  
  
    q->data[q->front] = data;  
  
    q->front = (q->front - 1 + MAX_DEQUE_SIZE)  
  
        % MAX_DEQUE_SIZE;  
  
    return;  
  
}
```

```
/*      덱의 모든 요소 print      */  
  
void deque_print(Deque* q) {  
  
    int i = (q->front + 1) % MAX_DEQUE_SIZE;  
  
    if (is_empty(q)) {  
  
        printf("Empty\n");  
  
        return;  
  
    }  
  
    while (i != q->rear) {  
  
        printf("%d | ", q->data[i]);  
  
        i = (i + 1) % MAX_DEQUE_SIZE;  
  
    }  
  
    printf("%d\n", q->data[i]);  
  
}
```

```
int main() {  
  
    Deque q;  
  
  
    init_deque(&q);  
  
    char input[101];  
    printf("Enter the word : ");  
    scanf("%s", input);  
  
    for (int i = 0; i < strlen(input); i++) {  
        add_front(&q, input[i] - '0');  
    }  
    int status = 1;  
    for (int i = strlen(input); i > 1; i-=2) {  
        int f = delete_front(&q);  
        int r = delete_rear(&q);  
        if (f != r) {  
            status = 0;  
            break;  
        }  
    }  
  
    status ? printf("회문입니다.") : printf("회문이 아닙니다.");  
    free(q.data);  
  
    return 0;  
  
}
```

# 실습 7

## 문제 7)

N개의 원소를 포함하고 있는 양방향 순환 큐를 가지고 있다. 이 큐에서 몇 개의 원소를 뽑아내려고 한다. 이 큐에서 다음과 같은 3가지 연산을 수행할 수 있다.

1. 첫 번째 원소를 뽑아낸다. 이 연산을 수행하면, 원래 큐의 원소가  $a_1, \dots, a_k$ 이었던 것이  $a_2, \dots, a_k$ 와 같이 된다.
2. 왼쪽으로 한 칸 이동시킨다. 이 연산을 수행하면,  $a_1, \dots, a_k$ 가  $a_2, \dots, a_k, a_1$ 이 된다.
3. 오른쪽으로 한 칸 이동시킨다. 이 연산을 수행하면,  $a_1, \dots, a_k$ 가  $a_k, a_1, \dots, a_{k-1}$ 이 된다.

큐에 처음에 포함되어 있던 수 N이 주어진다. 그리고 지민이가 뽑아내려고 하는 원소의 위치가 주어진다. (이 위치는 가장 처음 큐에서의 위치이다.)

이때, 그 원소를 주어진 순서대로 뽑아내는데 드는 2번, 3번 연산의 최솟값을 출력하는 프로그램을 작성하시오.

```
32 6
27 16 30 11 6 23
59
```

# 실습 7 정답

// 양방향 큐 = 덱이라고 해놓고, 경우 1은 DequeueFront만 해당..

```
#define _CRT_SECURE_NO_WARNINGS
```

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

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

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

```
typedef struct {
```

```
    int max;
```

```
    int num;
```

```
    int front;
```

```
    int rear;
```

```
    int* que;
```

```
} Deck;
```

```
int Initialize(Deck* d, int max) {
```

```
    d->num = d->front = d->rear = 0;
```

```
    if ((d->que = (int*)calloc(max, sizeof(int))) == NULL) {
```

```
        d->max = 0;
```

```
        return -1;
```

```
    }
```

```
    d->max = max;
```

```
    return 0;
```

```
}
```

```
int EnqueueFront(Deck* d, int v) {
```

```
    if (d->num >= d->max)
```

```
        return -1;
```

```
    d->num++;
```

```
    if (--d->front < 0)
```

```
        d->front = d->max - 1;
```

```
    d->que[d->front] = v;
```

```
    return 0;
```

```
}
```

```
int EnqueueRear(Deck* d, int v) {
```

```
    if (d->num >= d->max)
```

```
        return -1;
```

```
    d->num++;
```

```
    d->que[d->rear++] = v;
```

```
    d->rear = d->rear % d->max;
```

```
    return 0;
```

```
}
```

# 실습 7 정답

```
int DequeueFront(Deck* d, int* v) {  
  
    if (d->num <= 0)  
        return -1;  
  
    d->num--;  
    *v = d->que[d->front++];  
  
    d->front = d->front % d->max;  
    return 0;  
}
```

```
int DequeueRear(Deck* d, int* v) {  
  
    if (d->num <= 0)  
        return -1;  
  
    d->num--;  
  
    if (--d->rear < 0)  
        d->rear = d->max - 1;  
  
    *v = d->que[d->rear];  
    return 0;  
}
```

```
void Clear(Deck* d) {  
    d->num = d->front = d->rear = 0;  
}
```

```
int Size(Deck* d) {  
    return d->num;  
}
```

```
int IsEmpty(Deck* d) {  
    return (d->num <= 0);  
}
```

```
int PeekFront(Deck* d, int* v) {  
  
    if (d->num <= 0)  
        return -1;  
  
    *v = d->que[d->front];  
    return 0;  
}
```

```
int PeekRear(Deck* d, int* v) {  
    int temp;
```

```
    if (d->num <= 0)  
        return -1;
```

```
    if ((temp = d->rear - 1) < 0)  
        temp = d->max - 1;
```

```
    *v = d->que[temp];  
    return 0;
```

```
}
```

```
int Find_Sequence(Deck* d, int v) {  
    int index = -1;  
    for (int i = 0; i < d->num; i++) {
```

// 항상 잊지말길. 재조정 과정이 중요하다!!

```
        if (d->que[(index = (i + d->front) % d->max)] == v)  
            return i;
```

```
    }
```

```
    return -1; // 실패
```

```
}
```

# 실습 7 정답

```
int N, K;
char com[12];
int temp;
int COUNT;

int main() {
    Deck deck;

    scanf("%d %d", &N, &K);
    Initialize(&deck, N);

    for (int i = 1; i <= N; i++) {
        EnqueueRear(&deck, i);
    }

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

        scanf("%d", &temp);

        int seq = Find_Sequence(&deck, temp);

        if (seq < deck.num - seq) {

            while (seq--) {
                DequeueFront(&deck, &temp);
                EnqueueRear(&deck, temp);
                COUNT++;
            }

        }
        else {
            seq = deck.num - seq;
            while (seq--) {
                DequeueRear(&deck, &temp);
                EnqueueFront(&deck, temp);
                COUNT++;
            }
        }
        DequeueFront(&deck, &temp);
    }

    printf("%d\n", COUNT);
}
```

## 실습 8

문제 8)

오름차순으로 정렬되는 우선 순위 큐를 만들어보자.

숫자 외 입력은 주어지지 않는다. 0을 입력하면 종료된다.

```
5
5
3
3 5
1
1 3 5
2
1 2 3 5
4
1 2 3 4 5
0
```

# 실습 8 정답

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

#define size 500000
int que[size];

int compare(const void* a, const void* b) // 오름차순 비교 함수 구현
{
    int num1 = *(int*)a; // void 포인터를 int 포인터로 변환한 뒤 역참조하여
    값을 가져옴
    int num2 = *(int*)b; // void 포인터를 int 포인터로 변환한 뒤 역참조하여
    값을 가져옴

    if (num1 < num2) // a가 b보다 작을 때는
        return -1; // -1 반환

    if (num1 > num2) // a가 b보다 클 때는
        return 1; // 1 반환

    return 0; // a와 b가 같을 때는 0 반환
}
```



# 실습 8 정답

```
int main()
{
    char str[101];
    int n, i, front = 0, rear;

    rear = 0;

    while (1) {
        scanf("%d",&n);
        if (!n)break;
        que[rear++] = n;
        qsort(que,rear, sizeof(int), compare);
        do {
            printf("%d ", que[front]);
            front++;
        } while (front != rear);
        front = 0;
        printf("\n\n");
    }

    return 0;
}
```

## 실습 9

문제 9)

1부터 n까지 순서대로 담긴 큐가 있을 때 하나씩 삭제해주며 걸린 시간을 구하려고 한다. 이때, 세번마다 한번씩 삭제하지 않고 다시 큐에 넣어준다.

```
Size of Queue : 6
Contents of Queue : 0 1 2 3 4 5

1 2 3 4 5
2 3 4 5
3 4 5 2
4 5 2
5 2
2 5
5
Total time : 8
```

# 실습 9 정답

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

#define size 500000
int que[size];

int main()
{
    char str[101];
    int n, front = 0, rear;

    printf("Size of Queue : ");
    scanf("%d", &n);
    for (int i = 0; i < n; i++) que[i] = i;

    rear = n;

    int idx = 0;

    printf("Contents of Queue : ");
    for (int i = front; i < rear; i++) {
        printf("%d ", que[i]);
    }

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

```
while (1) {
    if (idx % 3 == 2) que[rear++] = que[front];
    front++;
    idx++;
    if (front == rear) break;

    for (int i = front; i < rear; i++) {
        printf("%d ", que[i]);
    }
    printf("\n\n");
}

printf("Total time : %d", idx);

return 0;
}
```

## 실습 10

문제 10)

한 프린터는 다음과 같은 조건에 따라 인쇄한다.

1. 현재 Queue의 가장 앞에 있는 문서의 '중요도'를 확인한다.
2. 나머지 문서들 중 현재 문서보다 중요도가 높은 문서가 하나라도 있다면, 이 문서를 인쇄하지 않고 Queue의 가장 뒤에 재배치 한다. 그렇지 않다면 바로 인쇄를 한다.

자신의 인쇄물이 언제 뽑히는지 출력해보자.

```
number of test cases : 3
Queue size : 1 0
My paper : Priority : 5
1
Queue size : 4 2
My paper : Priority : 1 2 3 4
2
Queue size : 6 0
My paper : Priority : 1 1 9 1 1 1
5
```

# 실습 10 정답

```
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>
int main() {
    int cnt, n, m;
    int i, j, k;
    int arr[100] = { 0, };

    printf("number of test cases : ");
    scanf("%d", &cnt);

    for (i = 0; i < cnt; i++)
    {
        printf("Queue size : ");
        scanf("%d", &n);
        printf("My paper : ");
        scanf("%d",&m);
        int ans = 1;
        int front = 0;
        int max = 0;
        printf("Priority : ");
        for (j = 0; j < n; j++)
            scanf("%d", &arr[j]);
```

```
        scanf("%d", &arr[j]);

        while (1)
        {
            for (k = 0; k < n; k++) {
                if (arr[k] > max) max = arr[k];
            }

            while (arr[front] != max)
                front = (front + 1) % n;

            if (front == m) break;

            arr[front] = 0;
            front = (front + 1) % n;
            max = 0;
            ans++;
        }
        printf("%d\\n", ans);
    }
    return 0;
}
```

## 실습 11

문제 11)

1부터  $n$ 까지 있는 큐가 있을 때, front를 삭제한 후 현재 front에 있는 정수를 삭제 후 다시 큐에 넣어준다. 이 과정을 카드 한 개가 남을 때까지 반복하고, 버린 순서대로 출력한다.

```
7
1 3 5 7 4 2 6
```

# 실습 11 정답

```
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>
int main()
{
    int N, i, temp;
    int card[1001];
    scanf_s("%d", &N);
    for (i = 1; i <= N; i++)
    {
        card[i] = i;
    }

    while (N > 0)
    {
        printf("%d ", card[1]);
        for (i = 1; i <= N; i++) // 카드 앞으로 이동
        {
            card[i] = card[i + 1];
        }

        N--;
        temp = card[1];
        for (i = 1; i <= N; i++)
        {
            card[i] = card[i + 1];
        }
        card[N] = temp; // 맨 위 카드 맨 아래로

    }
    return 0;
}
```

## 실습 12

문제 12)

피자를 먹기 위해선 줄을 서야한다. N명의 사람은 배가 부르기 위한 피자 수를 각자 다르다. 피자는 한 번에 한 개씩만 받을 수 있기 때문에 피자 여러 조각을 먹기 위해서는 다시 줄을 서야한다. 각 인원에게 대해서 피자를 먹기 위해 걸린 시간을 구하시오.

```
4
1 3 1 4
1 7 3 9
```



# 실습 12 정답

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

int main(int argc, char* argv[]) {

    int n = 0, * a, * b, sum = 0, num = 0;
    scanf("%d", &n);
    a = (int*)malloc(sizeof(int) * n); //원하는 피자 수를 저장할 배열
    b = (int*)malloc(sizeof(int) * n); //몇의 시간이 걸리는지 저장할 배열

    for (int i = 0; i < n; i++) {
        scanf("%d", &a[i]);
        sum += a[i]; //총 필요한 피자 수
        b[i] = 0; //배열 초기화해 주기
    }

    do {
        for (int i = 0; i < n; i++) {
            if (a[i] != 0) { //필요한 피자가 0이 된 사람은 세지 않는다
                a[i]--;
                num += 1; //현재 몇 번째 피자를 나누어 줬는지 세는 변수
            }
            if (a[i] == 0 && b[i] == 0) b[i] = num; //필요한 피자가 없으며, 시간이
            저장되지 않았다면 저장
        }
    } while (sum != num);

    for (int i = 0; i < n; i++) printf("%d ", b[i]); //출력
    return 0;
}
```