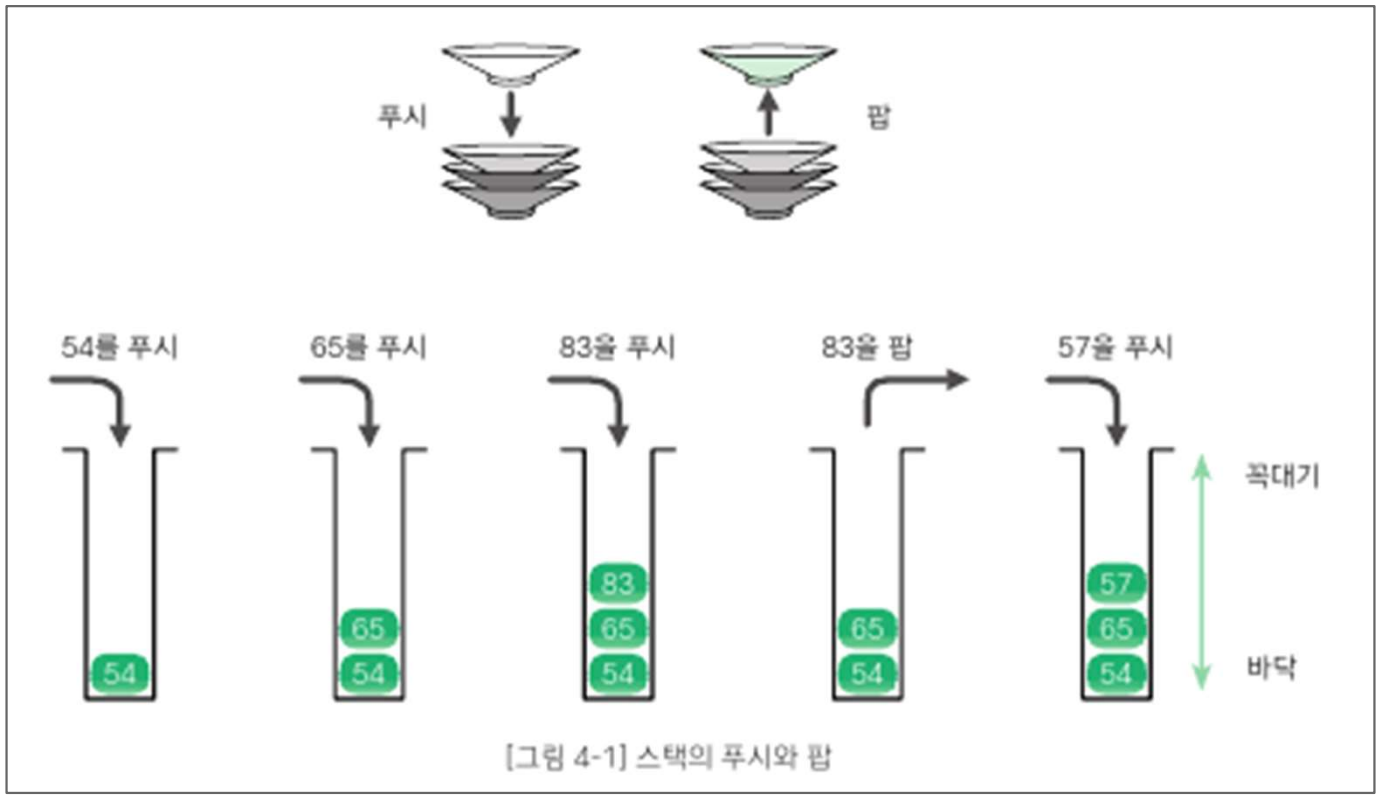
Do it! 자료구조와 함께 배우는 알고리즘 입문 C 언어편 **4단원**

2021210088 허지혜

목차

1. 스택(Stack)

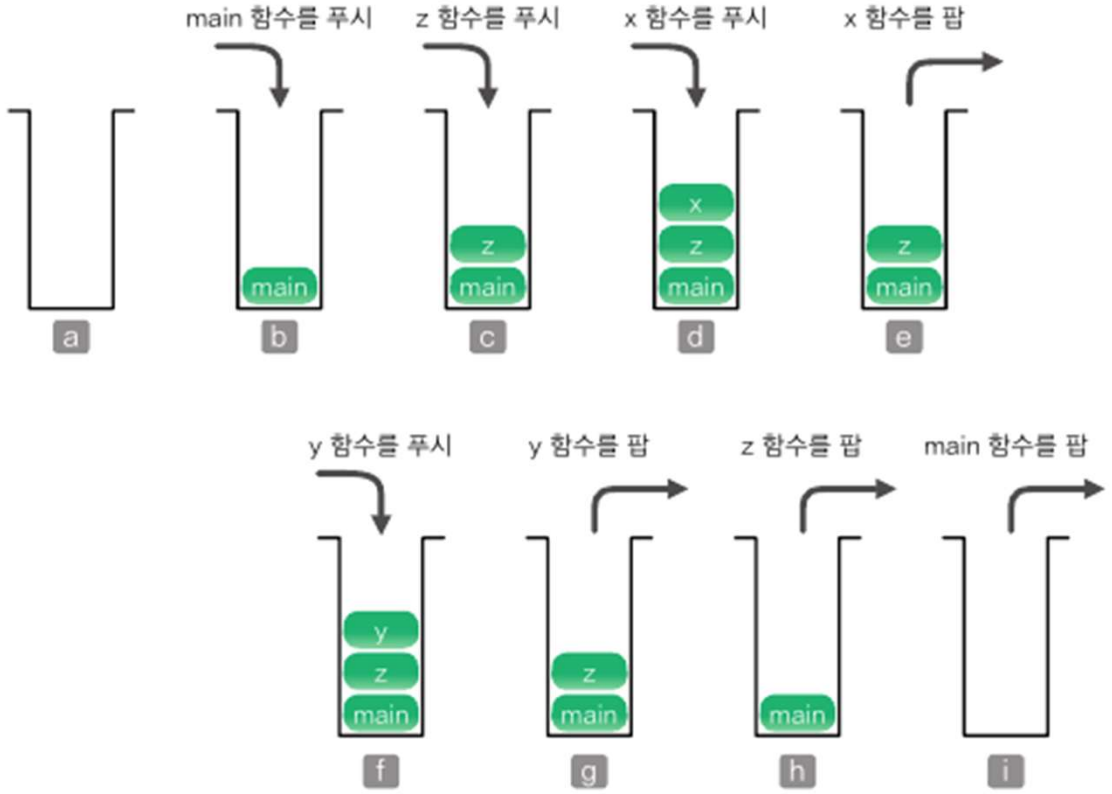


[그림 4-1] 스택의 푸시와 팝

- **스택(Stack)**
- 데이터를 일시적으로 저장하기 위해 사용하는 자료구조이다.
- 후입선출

- 푸시(Push) : 스택에 데이터 넣기
- 팝(Pop) : 스택에 데이터 꺼내기
- 꼭대기(Top) : 스택의 윗 부분
- 바닥(Bottom) : 스택의 아랫 부분

1. 스택(Stack)



[그림 4-2] 함수 호출과 스택

Code

```
void x()

void y()

void z() {

    x();

    y();

}

int main() {

z();

}
```

1. 스택(Stack)

```
# 스택을 구현하는 구조체

typedef struct{

    int max;

    int ptr;

    int stk;

} IntStack;
```

IntStack은 3개의 멤버로 구성되어 있다.

1) **스택으로 사용할 배열을 가리키는 포인터 stk**
- 스택으로 푸시된 데이터를 저장할 용도로 배열을 가리키는 포인터이다.
- index가 0인 요소가 바닥이고 메모리 저장 공간 할당을 Initialize 함수로 생성한다.

1) **스택의 최대 용량 max**
- 배열 stk의 요소 개수와 동일하다.

1) **스택에 쌓여 있는 데이터 개수 ptr**
- 스택이 비어있으면 0이고 가득 차 있으면 max 값이다.
- 가장 먼저 푸시된 바닥의 데이터를 stk[0], 가장 나중에 푸시된 꼭대기 데이터를 stk[ptr-1]라고 한다.

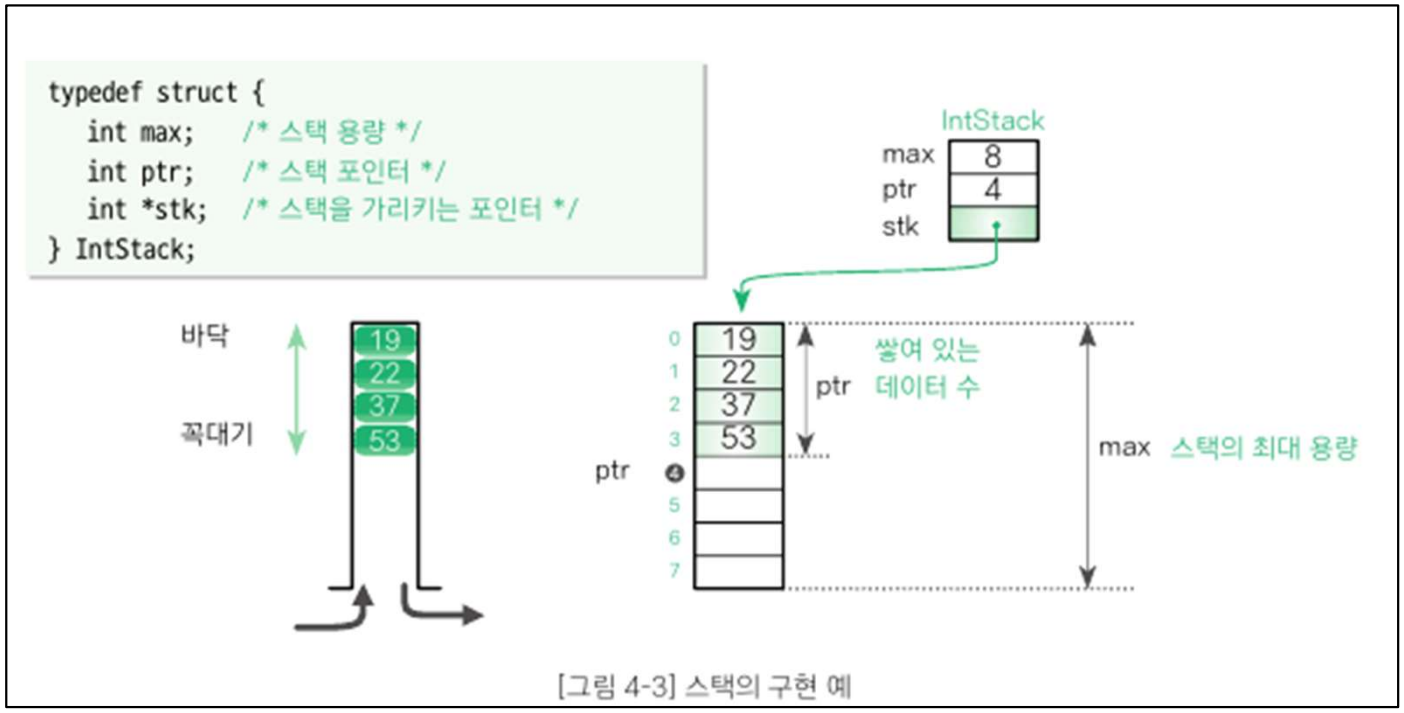
1. 스택(Stack)

```c
/* int형 스택 IntStack (소스) */
#include <stdio.h>
#include <stdlib.h>
#include "IntStack.h"

/*--- 스택 초기화 ---*/
int Initialize(IntStack *s, int max)
{
    s->ptr = 0;
    if ((s->stk = calloc(max, sizeof(int))) == NULL) {
        s->max = 0;                          /* 배열의 생성에 실패 */
        return -1;
    }
    s->max = max;
    return 0;
}
```

초기화 함수 Initialize

- 스택의 메모리 공간인 배열을 확보하는 등 준비작업을 위한 함수이다.
- 메모리 공간 안에 데이터가 하나도 쌓여 있지 않아야 한다.
- ptr = 0, 요소의 개수는 max인 배열 stk 생성
- 매개변수 max로 받은 값을 스택 최대 용량을 나타내는 구조체의 멤버 max에 저장한다.

1. 스택(Stack)



[그림 4-3] 스택의 구현 예

1. 스택(Stack)

```c
/*--- 스택에 데이터를 푸시---*/
int Push(IntStack *s, int x)
{
    if (s->ptr >= s->max)                    /* 스택이 가득 참 */
        return -1;
    s->stk[s->ptr++] = x;
    return 0;
}

/*--- 스택에서 데이터를 팝 ---*/
int Pop(IntStack *s, int *x)
{
    if (s->ptr <= 0)                         /* 스택이 비어 있음 */
        return -1;
    *x = s->stk[--s->ptr];
    return 0;
}
```

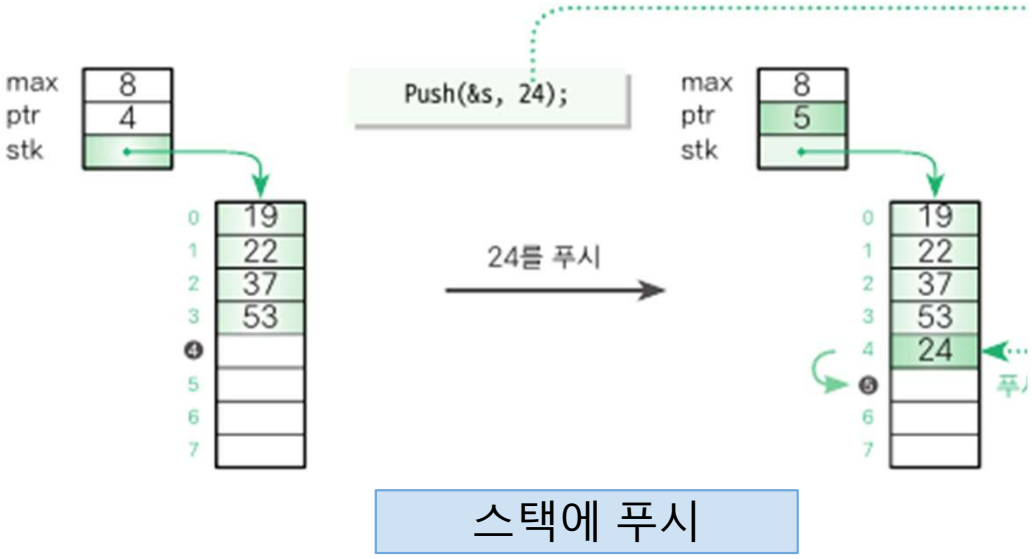
푸시 함수 Push

- 스택이 가득 차 푸시할 수 없는 경우 -1 반환
- 새로 추가할 데이터를 stk[ptr]에 저장하고 ptr을 증가시킨다.
- 푸시에 성공하면 0 반환

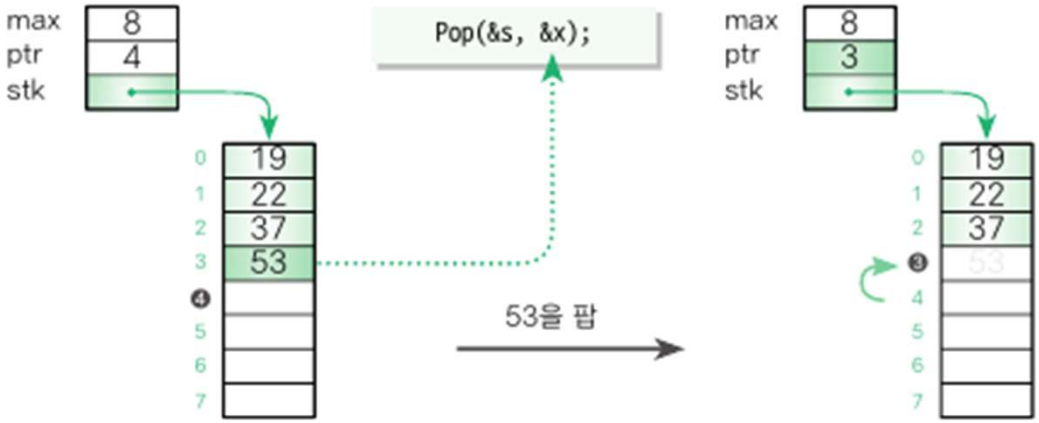
팝 함수 Pop

- 팝 성공시 0 반환
- 스택이 비어있어 팝을 못 하면 -1 반환

1. 스택(Stack)



스택에 푸시



스택에서 팝

1. 스택(Stack)

```c
/*--- 스택에서 데이터를 피크 ---*/
int Peek(const IntStack *s, int *x)
{
    if (s->ptr <= 0)                    /* 스택이 비어 있음 */
        return -1;
    *x = s->stk[s->ptr - 1];
    return 0;
}

/*--- 스택 비우기 ---*/
void Clear(IntStack *s)
{
    s->ptr = 0;
}
```

피크 함수 Peak

- 꼭대기 데이터를 몰래 엿보는 함수
- 피크 성공시 1 실패시 -1
- 스택이 비어있지 않으면 꼭대기 요소 값을 포인터 x가 가리키는 변수에 저장
- 데이터의 입출력이 없어 스택 포인터 변화 x

삭제 함수 Clear

- 스택에 쌓여 있는 모든 데이터 삭제

1. 스택(Stack)

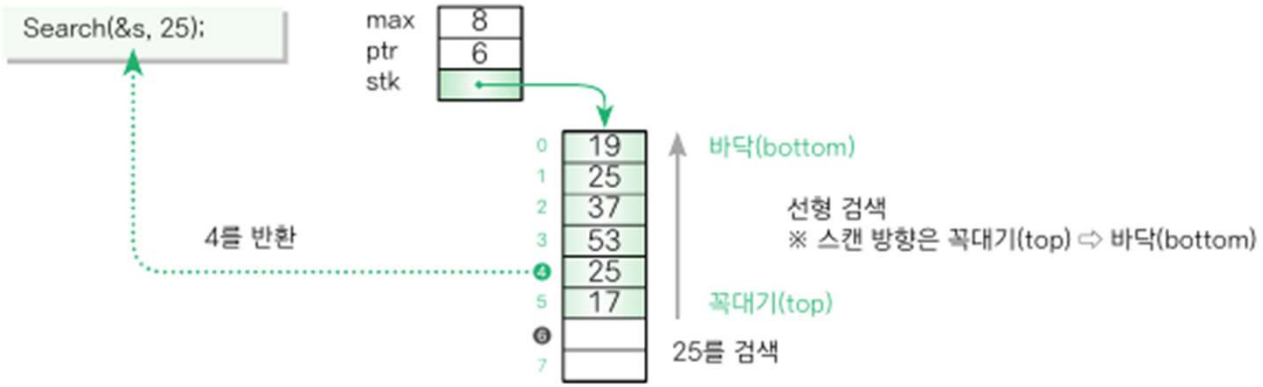
용량을 확인하는 함수 Capacity
데이터의 개수를 확인하는 함수 Size
스택이 비어있는지 검사하는 함수 IsEmpty 1 or 0
스택이 가득찼는지 검사하는 함수 IsFull 1 or 0
모든 데이터를 표시해주는 함수 Print
종료 함수 Terminate

```c
/*--- 스택 용량 ---*/
int Capacity(const IntStack *s)
{
    return s->max;
}

/*--- 스택에 쌓여 있는 데이터 수 ---*/
int Size(const IntStack *s)
{
    return s->ptr;
}

/*--- 스택이 비어 있는가? ---*/
int IsEmpty(const IntStack *s)
{
    return s->ptr <= 0;
}

/*--- 스택은 가득 찼는가? ---*/
int IsFull(const IntStack *s)
{
    return s->ptr >= s->max;
}
```

```c
/*--- 모든 데이터 표시 ---*/
void Print(const IntStack *s)
{
    int i;

    for (i = 0; i < s->ptr; i++)        /* 바닥(bottom) → 꼭대기(top)로 스캔 */
        printf("%d ", s->stk[i]);
    putchar('\n');
}

/*--- 스택 종료 ---*/
void Terminate(IntStack *s)
{
    if (s->stk != NULL)
        free(s->stk);       /* 배열을 삭제 */
    s->max = s->ptr = 0;
}
```

1. 스택(Stack)

```c
/*--- 스택에서 검색 ---*/
int Search(const IntStack *s, int x)
{
    int i;

    for (i = s->ptr - 1; i >= 0; i--)   /* 꼭대기(top) → 바닥(bottom)으로 선형 검색 */
        if (s->stk[i] == x)
            return i;       /* 검색 성공 */
    return -1;              /* 검색 실패 */
}
```



[그림 4-6] 스택에서 검색

스택에서 임의의 값을 검색하는 함수 Search

- 임의의 값의 데이터가 스택의 어디에 있는지 검사하는 함수이다.
- 꼭대기에서 바닥으로 선형 검색을 수행한다.
- 검색에 성공하면 찾은 요소의 인덱스를 반환하고 실패시 -1 반환

1. 스택(Stack)

테스트

```c
/* int형 스택 IntStack의 사용 */
#include <stdio.h>
#include "IntStack.h"

int main(void)
{
    IntStack s;
    if (Initialize(&s, 64) == -1) {
        puts("스택 생성에 실패하였습니다.");
        return 1;
    }

    while (1) {
        int menu, x;

        printf("현재 데이터 수 : %d / %d₩n", Size(&s), Capacity(&s));
        printf("(1) 푸시 (2) 팝 (3) 피크 (4) 출력 (0) 종료 : ");
        scanf("%d", &menu);
        if (menu == 0) break;

        switch (menu) {
        case 1: /*--- 푸시---*/
            printf("데이터 : ");
            scanf("%d", &x);
            if (Push(&s, x) == -1)
                puts("₩a오류 : 푸시에 실패하였습니다.");
            break;

        case 2: /*--- 팝 ---*/
            if (Pop(&s, &x) == -1)
                puts("₩a오류 : 팝에 실패하였습니다.");
            else
                printf("팝 데이터는 %d입니다.₩n", x);
            break;

        case 3: /*--- 피크 ---*/
            if (Peek(&s, &x) == -1)
                puts("₩a오류 : 피크에 실패하였습니다.");
            else
                printf("피크 데이터는 %d입니다.₩n", x);
            break;

        case 4: /*--- 출력 ---*/
            Print(&s);
            break;
        }
    }

    Terminate(&s);

    return 0;
}
```

1. 스택(Stack)

테스트 결과

실행 결과

현재 데이터 수 : 0/64
(1) 푸시 (2) 팝 (3) 피크 (4) 출력 (0) 종료 : 1
데이터 : 1 ---- 1을 푸시

현재 데이터 수 : 1/64
(1) 푸시 (2) 팝 (3) 피크 (4) 출력 (0) 종료 : 1
데이터 : 2 ---- 2를 푸시

현재 데이터 수 : 2/64
(1) 푸시 (2) 팝 (3) 피크 (4) 출력 (0) 종료 : 1
데이터 : 3 ---- 3을 푸시

현재 데이터 수 : 3/64
(1) 푸시 (2) 팝 (3) 피크 (4) 출력 (0) 종료 : 1
데이터 : 4 ---- 4를 푸시

현재 데이터 수 : 4/64
(1) 푸시 (2) 팝 (3) 피크 (4) 출력 (0) 종료 : 3
피크 데이터는 4입니다. ---- 4를 피크

현재 데이터 수 : 4/64
(1) 푸시 (2) 팝 (3) 피크 (4) 출력 (0) 종료 : 4
1 2 3 4 ---- 스택의 내용을 출력

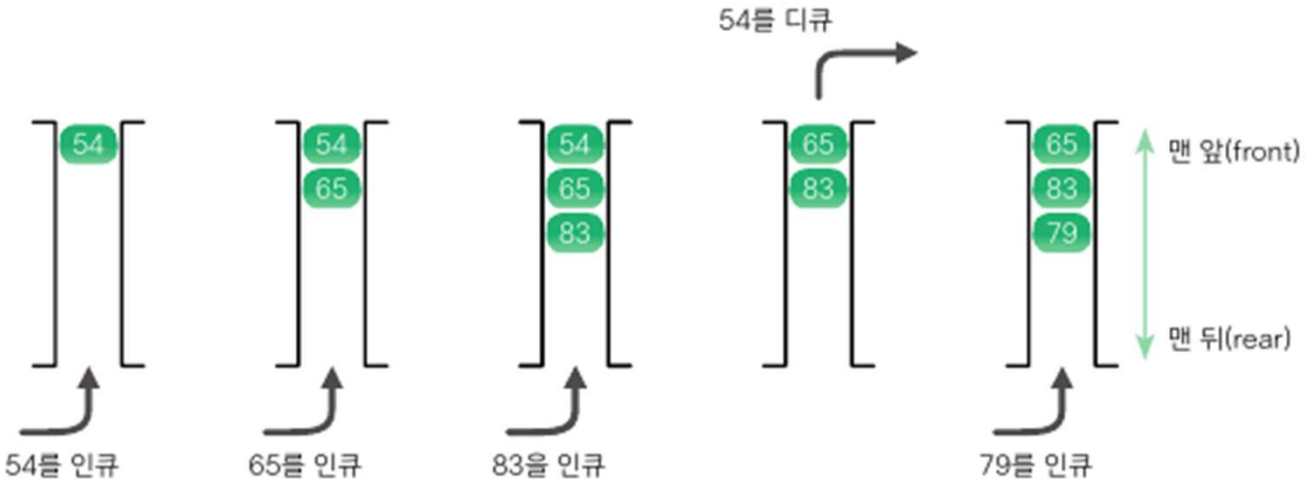
현재 데이터 수 : 4/64
(1) 푸시 (2) 팝 (3) 피크 (4) 출력 (0) 종료 : 2
팝 데이터는 4입니다. ---- 4를 팝

현재 데이터 수 : 3/64
(1) 푸시 (2) 팝 (3) 피크 (4) 출력 (0) 종료 : 2
팝 데이터는 3입니다. ---- 3을 팝

현재 데이터 수 : 2/64
(1) 푸시 (2) 팝 (3) 피크 (4) 출력 (0) 종료 : 4
1 2 ---- 스택의 내용을 출력

현재 데이터 수 : 2/64
(1) 푸시 (2) 팝 (3) 피크 (4) 출력 (0) 종료 : 0

2. 큐(Queue)



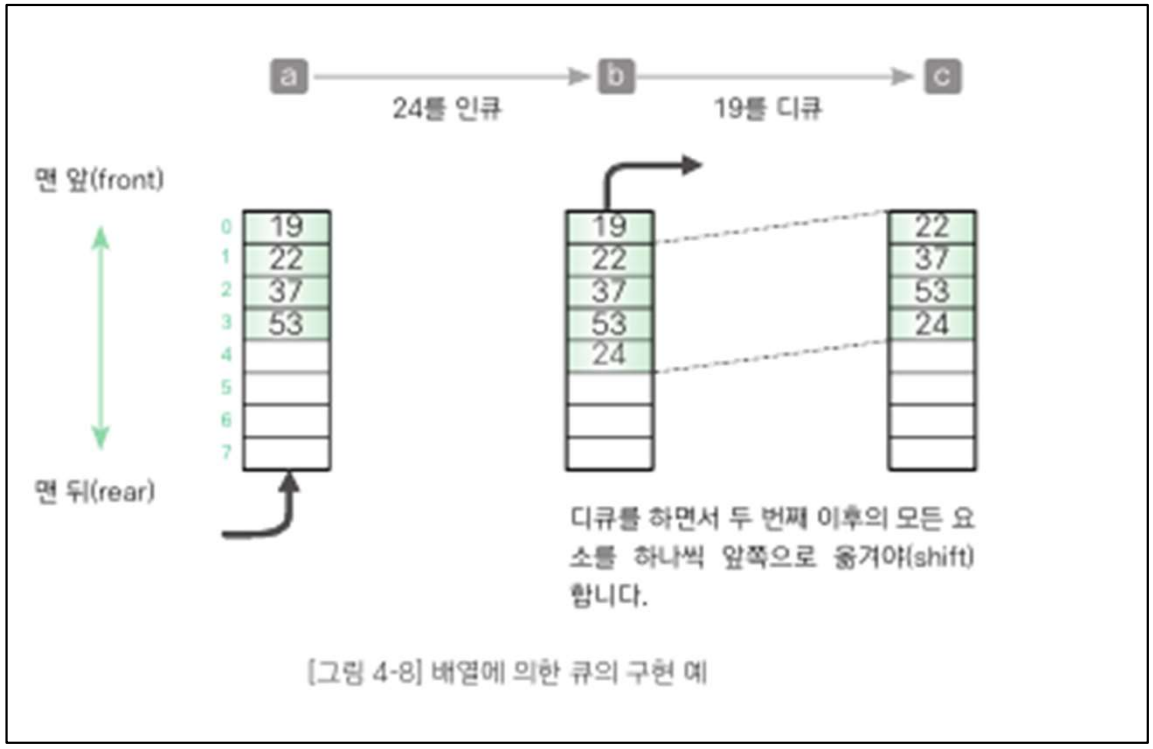
[그림 4-7] 인큐와 디큐

- **큐(Queue)**
- 데이터를 일시적으로 저장하기 위해 사용하는 자료구조이다.
- 선입선출

- 인큐(Enque) : 큐에 데이터 넣기
- 디큐(Deque) : 큐에 데이터 꺼내기
- 프런트(Front) : 데이터 꺼내는 쪽
- 리어(Reer) : 데이터 넣는 쪽

2. 큐(Queue)

배열로 큐 만들기



[그림 4-8] 배열에 의한 큐의 구현 예

1) 24 인큐

- que[4]에 24를 저장
- 처리 복잡도는 O(1)으로 적은 비용으로 구현할 수 있다.

1) 19 디큐

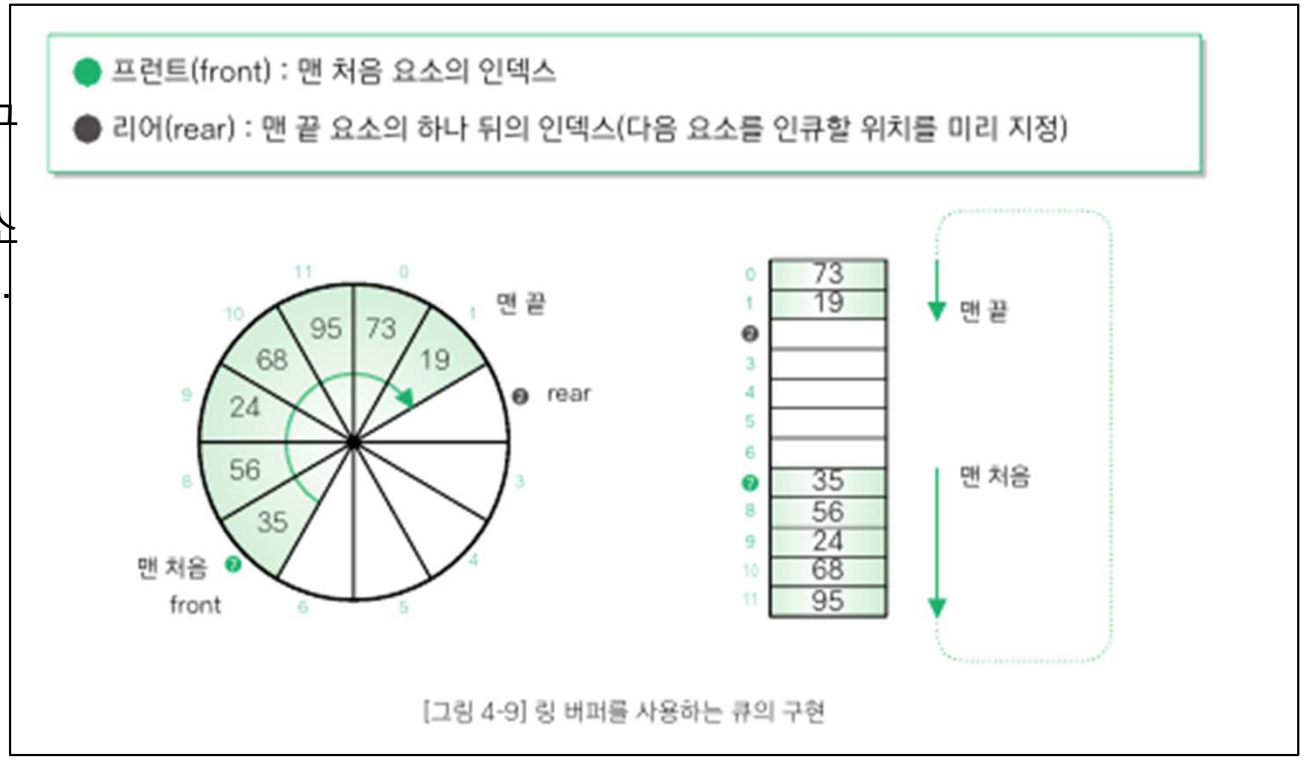
- que[0]에서 19를 꺼내고 다른 요소들을 맨 앞으로 옮김
- O(n)라 처리하면 효율이 떨어진다.

2. 큐(Queue)

배열 요소를 앞쪽으로 옮기지 않는 큐 구현
=> **링 버퍼** 자료구조 이용

링 버퍼

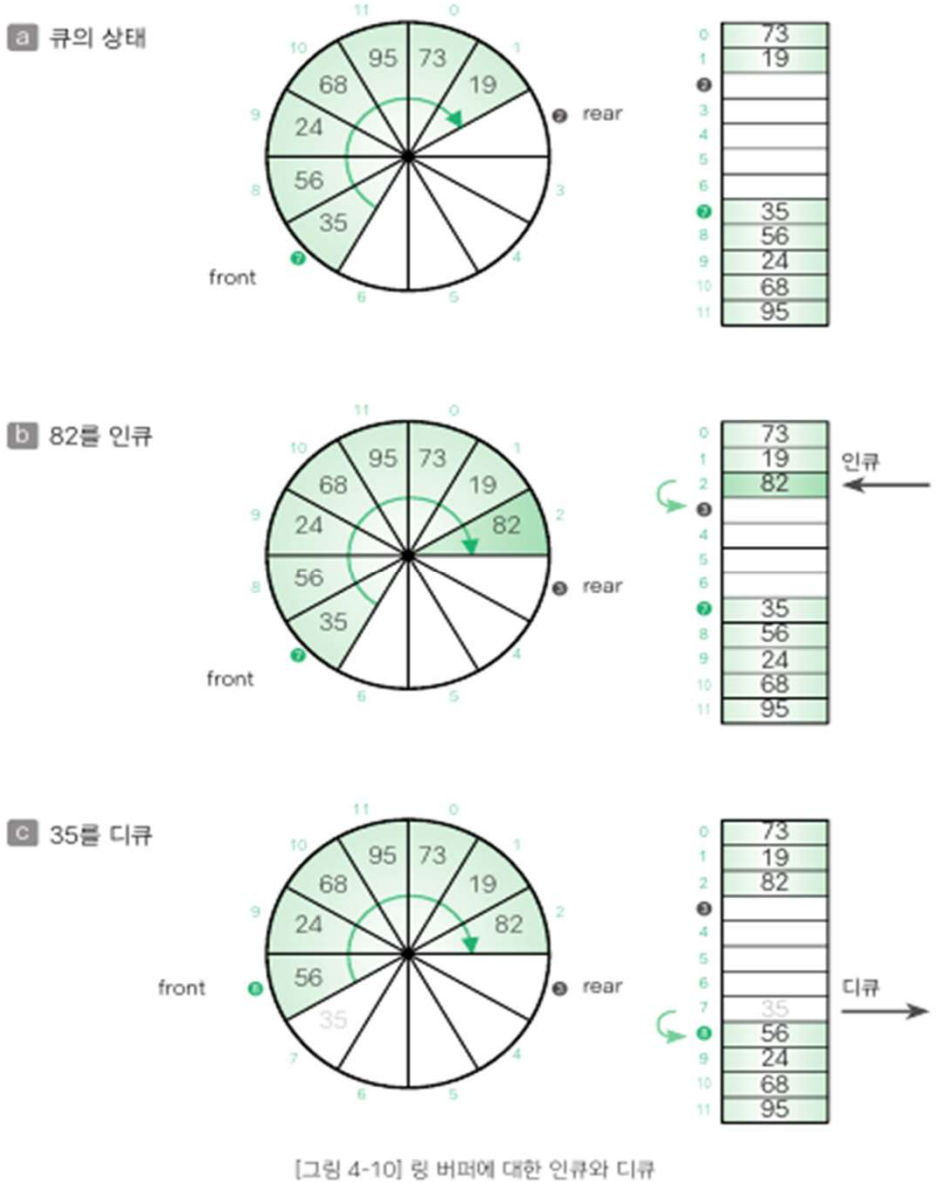
- 처음이 끝과 연결되었다고 보는 자료구조
- 논리적으로 구별하기 위해 첫번째 요소를 프런트, 마지막 요소를 리어가 된다.



[그림 4-9] 링 버퍼를 사용하는 큐의 구현

2. 큐(Queue)

프런트값 7, 리어 값 2



[그림 4-10] 링 버퍼에 대한 인큐와 디큐

2. 큐(Queue)

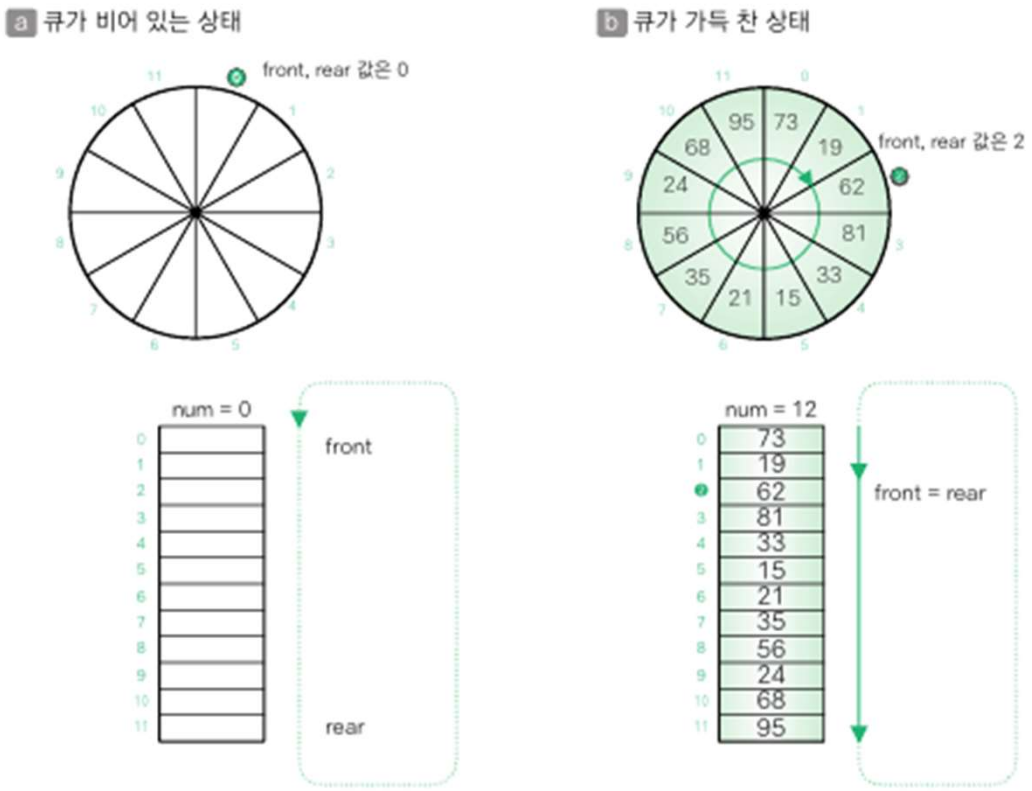
```c
/* int형 큐 IntQueue.h(헤더) */

#ifndef ___IntQueue
#define ___IntQueue

/*--- 큐를 구현하는 구조체 ---*/
typedef struct {
    int max;    /* 큐의 최대 용량 */
    int num;    /* 현재 요소의 개수 */
    int front;  /* 프런트 */
    int rear;   /* 리어 */
    int *que;   /* 큐의 첫 요소에 대한 포인터 */
} IntQueue;
```

큐 구조체 IntQueue

1) 큐로 사용할 배열(que)
2) 큐의 최대 용량(max)
3) 프런트(front)
4) 리어(reer)
5) 현재 데이터 개수(num)

2. 큐(Queue)

num과 max가 있는 이유



[그림 4-11] 비어 있는 큐와 가득 찬 큐

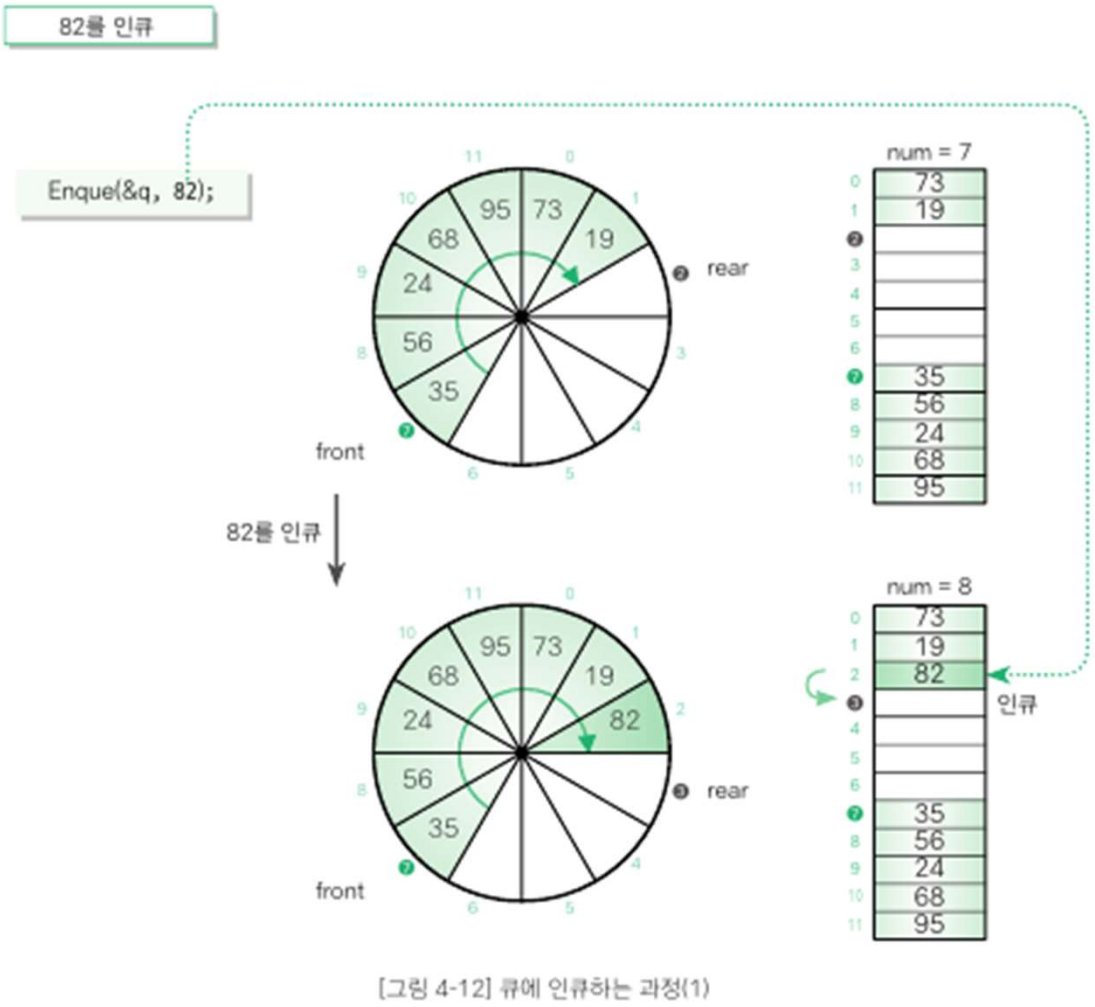
2. 큐(Queue)

```c
/* int형 큐 IntQueue(소스) */
#include <stdio.h>
#include <stdlib.h>
#include "IntQueue.h"

/*--- 큐 초기화 ---*/
int Initialize(IntQueue *q, int max)
{
    q->num = q->front = q->rear = 0;
    if ((q->que = calloc(max, sizeof(int))) == NULL) {
        q->max = 0;                          /* 배열 생성에 실패 */
        return -1;
    }
    q->max = max;
    return 0;
}
```

초기화 함수 Initialize

- 큐의 메모리 공간인 배열을 확보하는 등 준비작업을 위한 함수이다.
- 큐를 처음 만들면 비어있으므로 num,front,reer = 0 저장
- 매개변수 max로 받은 값을 스택 최대 용량을 나타내는 구조체의 멤버 max에 저장한다.

2. 큐(Queue)



[그림 4-12] 큐에 인큐하는 과정(1)

2. 큐(Queue)

```c
/*--- 큐에 데이터를 인큐 ---*/
int Enque(IntQueue *q, int x)
{
    if (q->num >= q->max)
        return -1;                      /* 큐가 가득 참 */
    else {
        q->num++;
        q->que[q->rear++] = x;
        if (q->rear == q->max)
            q->rear = 0;
        return 0;
    }
}

/*--- 큐에서 데이터를 디큐 ---*/
int Deque(IntQueue *q, int *x)
{
    if (q->num <= 0)                    /* 큐는 비어 있음 */
        return -1;
    else {
        q->num--;
        *x = q->que[q->front++];
        if (q->front == q->max)
            q->front = 0;
        return 0;
    }
}
```

인큐 함수 Enque

- 인큐 성공시 0, 아님 -1

디큐 함수 Deque

- 디큐 성공시 0, 아님 -1

특징

- 두 함수다 그냥 +1,-1 하면 index 초과 에러가 뜨기 때문에 max값에 따라 0으로 초기화
- 나머지 함수는 스택과 똑같다.

2. 큐(Queue)

테스트

```c
/* int형 큐 IntQueue를 사용하는 프로그램 */
#include <stdio.h>
#include "IntQueue.h"

int main(void)
{
    IntQueue que;

    if (Initialize(&que, 64) == -1) {
        puts("큐의 생성에 실패하였습니다.");
        return 1;
    }

    while (1) {
        int m, x;

        printf("현재 데이터 수 : %d / %d ₩n", Size(&que), Capacity(&que));
        printf("(1) 인큐 (2) 디큐 (3) 피크 (4) 출력 (0) 종료 : ");
        scanf("%d", &m);

        if (m == 0) break;
        switch (m) {
        case 1: /*--- 인큐 ---*/
            printf("데이터 : "); scanf("%d", &x);
            if (Enque(&que, x) == -1)
                puts("₩a오류 : 인큐에 실패하였습니다.");
            break;

        case 2: /*--- 디큐 ---*/
            if (Deque(&que, &x) == -1)
                puts("₩a오류 : 디큐에 실패하였습니다.");
            else
                printf("디큐한 데이터는 %d입니다.₩n", x);
            break;

        case 3: /*--- 피크 ---*/
            if (Peek(&que, &x) == -1)
                puts("₩a오류 : 피크에 실패하였습니다.");
            else
                printf("피크한 데이터는 %d입니다.₩n", x);
            break;

        case 4: /*--- 출력 ---*/
            Print(&que);
            break;
        }
    }

    Terminate(&que);

    return 0;
}
```

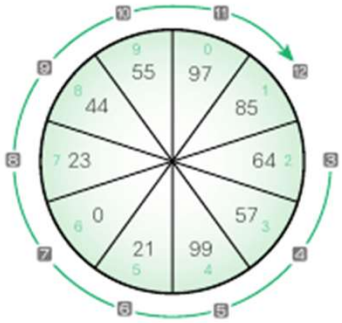
2. 큐(Queue)

테스트 결과

3. 링 버퍼의 활용

```c
/*
    원하는 개수만큼 데이터를 입력하고,
    요소의 개수가 n인 배열에 최근에 입력한 n개만 저장
*/

#include <stdio.h>
#define N 10            /* 저장하는 데이터의 개수 */

int main()
{
    int i;
    int a[N];       /* 입력한 데이터를 저장 */
    int cnt = 0;    /* 입력한 데이터의 개수 */
    int retry;      /* 다시 한 번? */

    puts("정수를 입력하세요.");

    do {
        printf("%d번째 정수 : ", cnt + 1);
        scanf("%d", &a[cnt++ % N]);
        printf("계속할까요? (Yes … 1/No … 0) : ");
        scanf("%d", &retry);
    } while (retry == 1);
        i = cnt - N;
        if (i < 0) i = 0;

    for (; i < cnt; i++)
        printf("%2d번째 정수 = %d₩n", i + 1, a[i % N]);

    return 0;
}
```

15 17 64 57 99 21 0 23 44 55 97 85 입력



※원 안쪽의 숫자 … 요소의 인덱스
■ 안의 숫자 … n번째로 입력한 값

실행 결과

```
정수를 입력하세요.
1번째 정수 : 15
계속할까요?(Yes … 1/No … 0) : 1
2번째의 정수 : 17
계속할까요?(Yes … 1/No … 0) : 1
… 중략 …
12번째 정수 : 85
계속할까요?(Yes … 1/No … 0) : 0
3번째의 정수 = 64
4번째의 정수 = 57
5번째의 정수 = 99
… 중략 …
10번째의 정수 = 55
11번째의 정수 = 97
12번째의 정수 = 85
```

[그림 4C-1] 링 버퍼에 값 입력하기

끝