데이터 구조 5장 실습과제

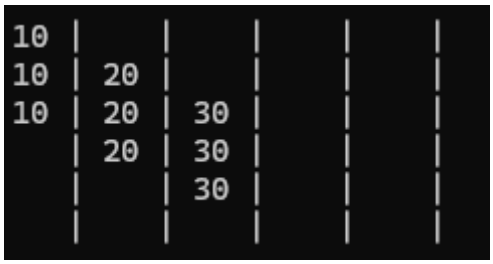
20223100 박신조

5.1 선형큐 프로그램

```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(char* message)
{
    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;
}
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 포화 상태입니다.");
        return;
    }
    q->data[++(q->rear)] = item;
}

int dequeue(QueueType* q)
{
    if (is_empty(q)) {
        error("큐가 공백 상태입니다.");
        return -1;
    }
    int item = q->data[++(q->front)];
    return item;
}
```

```c
int main()
{
    int item;
    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;
}
```

코드 결과 출력



배열을 이용한 선형 큐를 구현한 코드입니다.
선형큐는 선입선출(FIFO) 방식입니다.
선입선출이란 먼저 입력된 데이터가 먼저 출력되는 자료구조입니다.
ex) 프린터, 매표소에서 계산할 때 등등

typedef int element

element -> int 로 매크로 설정을 합니다.

typedef struct { . . . } QueueType

선형 큐를 구현할 구조체입니다.
front, rear는 삽입, 삭제를 element 타입의 data배열은 큐의 요소들을 저장하는 역할을 합니다.

void error(char *message)

오류 메시지를 출력하고 프로그램을 종료합니다.
덱이 포화 상태일 경우나 공백일 경우에 사용됩니다.

void init_queue(QueueType*q)

큐를 초기화하는 함수입니다.
front와 rear 인덱스를 -1으로 설정하여 초기 상태로 만듭니다.

int is_empty()

큐가 비어 있는지를 확인하는 함수입니다.
front 와rear 가 같다는 것은 큐가 비어있음을 의미합니다.

int is_full()

큐가 가득 찼는지를 확인하는 함수입니다.
rear 가 MAX_QUEUE_SIZE - 1에 도달하면 큐는 포화 상태입니다.

void enqueue(int item)

큐에 새로운 데이터를 추가하는 함수입니다.
is_full 함수를 통해 큐가 가득 찼는지 확인 후, rear를 1만큼 증가시키고 해당 위치에 데이터를 저장합니다.

삽입 후 큐의 요소(배열)를 확인할 수 있도록 print_queue 함수를 호출합니다.

int dequeue()

큐에서 요소를 제거하는 함수입니다.
is_empty 함수를 통해 큐에 요소가 있는지 확인 후, front를 1만큼 증가시키고 해당 위치의 인덱스를 반환합니다.
배열에서 실제로 데이터를 삭제하지는 않지만, front를 통해 범위를 조정합니다.

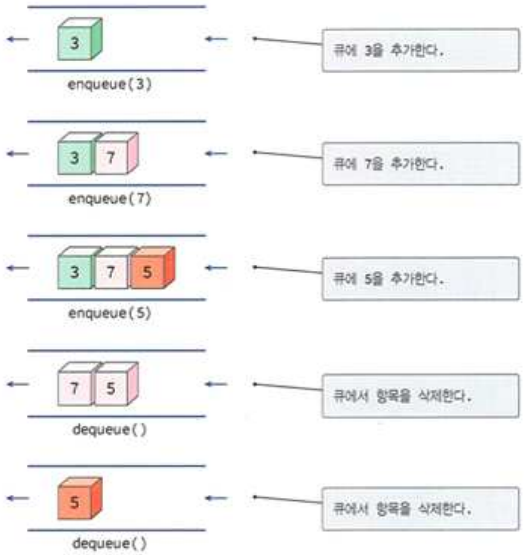
void print_queue()

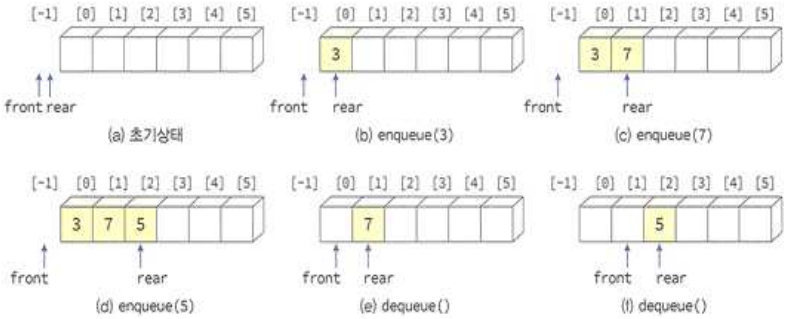
큐의 현재 상태를 출력하는 함수입니다.
배열 전체를 반복하면서front보다 크고rear보다 작거나 같은 위치에 있는 값만 출력합니다.

int main()

먼저 큐를 초기화 해줍니다.
enqueue 함수를 사용하여 요소(10,20,30)를 큐에 삽입하고, dequeue를 호출하여 데이터를 반환(제거)합니다.
데이터를 삽입, 삭제 할때마다 print_queue 함수를 호출하여 큐의 상태를 출력합니다.

선형 큐의 작동 순서





5.2 원형큐 프로그램

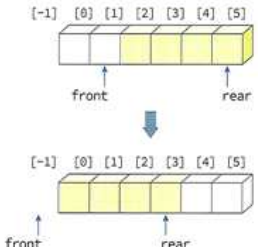
```c
#include <stdio.h>
#include <stdlib.h>
#define MAX_QUEUE_SIZE 5

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

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

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

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

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

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");
}

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;
}

element dequeue(QueueType* q)
{
    if (is_empty(q))
        error("큐가 공백상태입니다.");
    q->front = (q->front + 1) % MAX_QUEUE_SIZE;
    return q->data[q->front];
}

element peek(QueueType* q)
{
    if (is_empty(q))
        error("큐가 공백상태입니다.");
    return q->data[(q->front + 1) % MAX_QUEUE_SIZE];
}

int main()
{
    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("큐가 포화상태입니다.₩n₩n");
```

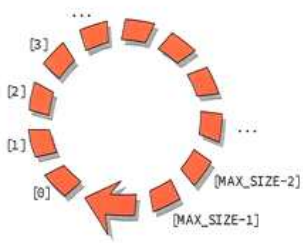
```
    printf("--데이터 삭제 단계--₩n");
    while (!is_empty(&queue)) {
        element = dequeue(&queue);
        printf("꺼내진 정수 : %d ₩n", element);
        queue_print(&queue);
    }
    printf("큐는 공백 상태입니다.₩n");
    return 0;
}
```

코드 결과 출력

```
--데이터 추가 단계--
정수를 입력하시오 : 1
QUEUE(front=0 rear=1) = 1 |
정수를 입력하시오 : 2
QUEUE(front=0 rear=2) = 1 | 2 |
정수를 입력하시오 : 3
QUEUE(front=0 rear=3) = 1 | 2 | 3 |
정수를 입력하시오 : 4
QUEUE(front=0 rear=4) = 1 | 2 | 3 | 4 |
큐가 포화상태입니다.

--데이터 삭제 단계--
꺼내진 정수 : 1
QUEUE(front=1 rear=4) = 2 | 3 | 4 |
꺼내진 정수 : 2
QUEUE(front=2 rear=4) = 3 | 4 |
꺼내진 정수 : 3
QUEUE(front=3 rear=4) = 4 |
꺼내진 정수 : 4
QUEUE(front=4 rear=4) =
큐는 공백 상태입니다.
```

선형 큐의 단점





원형 큐의 구조

배열을 이용하여 원형큐를 구현한 코드입니다.
위에 나온 선형 큐는 삽입과 삭제를 반복하면 앞쪽 공간이 비어도 사용할 수 없는 비효율적인 구조였습니다.
(front를 사용하여 데이터를 출력했기 때문)
원형 큐는 배열의 양 끝이 연결된 형태로 구성되어 있습니다.
원형 큐는 운영체제에서 사용됩니다.

typedef int element

element -> int 로 매크로 설정을 합니다.

typedef struct { . . . } QueueType

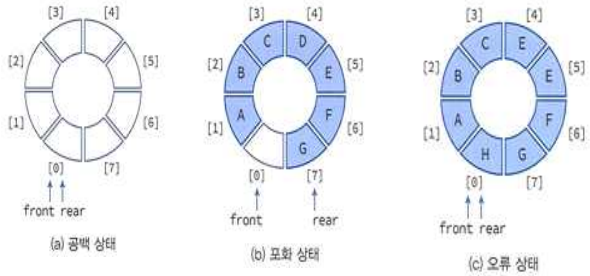
선형 큐를 구현할 구조체입니다.
front, rear는 삽입, 삭제를 element 타입의 data배열은 큐의 요소들을 저장하는 역할을 합니다.

void error()

오류 메시지를 출력하고 프로그램을 종료합니다.
덱이 포화 상태일 경우나 공백일 경우에 사용됩니다.

void init_queue()

큐를 초기화하는 함수입니다.
front와 rear 인덱스를 0으로 설정하여 초기 상태로 만듭니다.



int is_empty()

큐가 비어 있는지를 확인하는 함수입니다.
front 와rear 가 같다는 것은 큐가 비어있음을 의미합니다.

int is_full()

큐가 가득 찼는지를 확인하는 함수입니다.
rear 가 MAX_QUEUE_SIZE - 1에 도달하면 큐는 포화 상태입니다.

void print_queue()

큐의 현재 상태를 출력하는 함수입니다.
front 다음 위치부터rear 위치까지 출력합니다.
원형 큐의 특성상 인덱스가 순환되므로,
while 반복문을 사용하여 조건을 `(rear + 1) % MAX_QUEUE_SIZE`로 설정해 줍니다.
이때 종료 조건을 설정해주지 않으면 원형 큐를 순환하며 계속해서 인덱스를 반환합니다.

void enqueue()

큐에 새로운 데이터를 추가하는 함수입니다.
is_full 함수를 통해 큐가 가득 찼는지 확인 후, rear를 1만큼 증가시키고 해당 위치에 데이터를 저장합니다.
삽입 후 큐의 요소(배열)를 확인할 수 있도록 print_queue 함수를 호출합니다.

int dequeue()

큐에서 요소를 제거하는 함수입니다.
is_empty 함수를 통해 큐에 요소가 있는지 확인 후, front를 1만큼 증가시키고 해당 위치의 인덱스를 반환합니다.
배열에서 실제로 데이터를 삭제하지는 않지만, front를 통해 범위를 조정합니다.

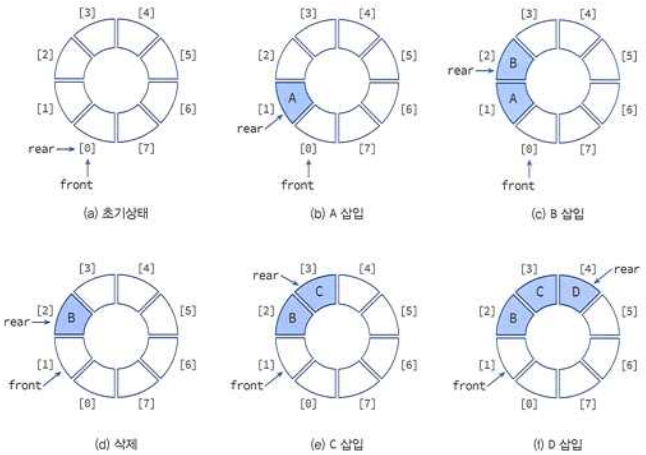
int peek()

큐에서 front+1의 인덱스 값만 리턴해주는 함수입니다.
is_empty 함수를 통해 큐에 요소가 있는지 확인 후, front+1의 인덱스를 반환합니다.
front값을 실제로 변경하지 않고 +1을 시켜 배열의 요소만 확인합니다.

int main()

먼저 큐를 초기화해줍니다. 이후 enqueue 함수와 dequeue 함수를 사용하여 삽입, 삭제 과정을 출력합니다.
이때 while문을 사용하여 원형 큐의 크기만큼 데이터를 저장하고 다시 while 문을 통해 원형 큐의 요소가 0이 될 때까지 데이터를 출력합니다.
이를 통해 원형 큐의 삽입, 삭제를 구현해볼 수 있습니다.

원형 큐의 작동 순서

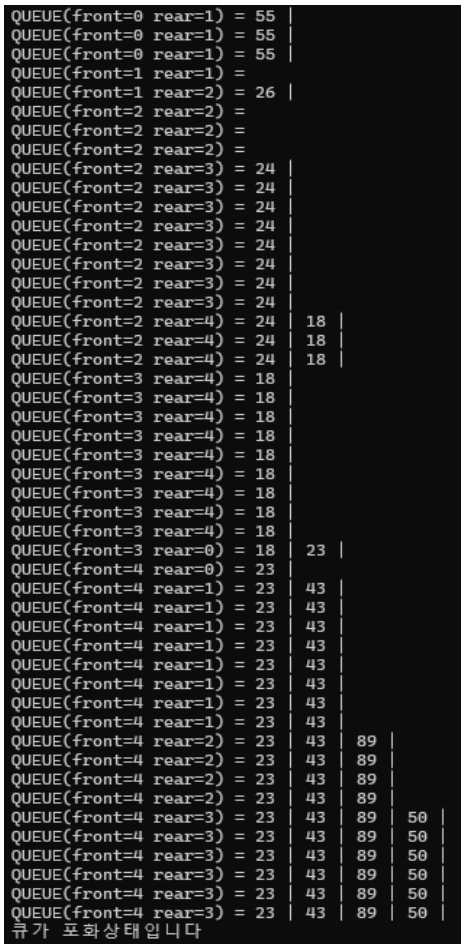


5.3 큐 응용 프로그램

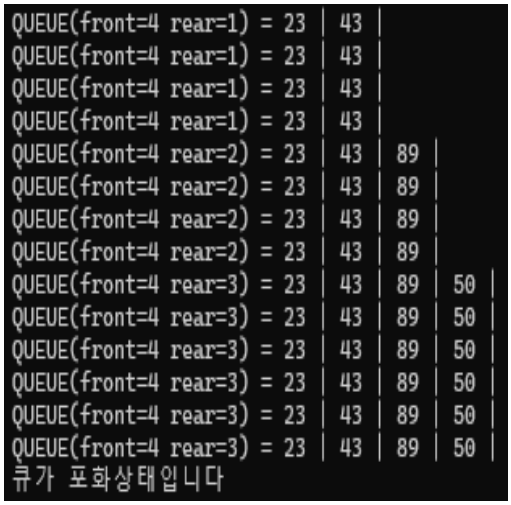
```c
#include <stdio.h>
#include <stdlib.h>
#define MAX_QUEUE_SIZE 5

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

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

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

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

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

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");
}

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;
}

element dequeue(QueueType *q){
    if (is_empty(q))
        error("큐가 공백상태입니다");
    q->front = (q->front + 1) % MAX_QUEUE_SIZE;
    return q->data[q->front];
}

element peek(QueueType *q){
    if (is_empty(q))
        error("큐가 공백상태입니다");
    return q->data[(q->front + 1) % MAX_QUEUE_SIZE];
}

int main(void){
    QueueType queue;
    int element;

    init_queue(&queue);
    srand(time(NULL));

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

코드 결과 출력

실행 1 결과



실행 2 결과



큐는 서로 다른 속도로 실행되는 두 프로세스 간의 상호 작용을 조화시키는 버퍼 역할을 합니다.
위 코드는 원형 큐를 사용하여 버퍼를 구현한 코드입니다.

typedef int element

element -> int 로 매크로 설정을 합니다.

typedef struct { . . . } QueueType

선형 큐를 구현할 구조체입니다.
front, rear는 삽입, 삭제를 element 타입의 data배열은 큐의 요소들을 저장하는 역할을 합니다.

void error()

오류 메시지를 출력하고 프로그램을 종료합니다.
덱이 포화 상태일 경우나 공백일 경우에 사용됩니다.

void init_queue()

큐를 초기화하는 함수입니다.
front와 rear 인덱스를 0으로 설정하여 초기 상태로 만듭니다.

int is_empty()

큐가 비어있는지를 확인합니다.
front와 rear가 같은 경우 큐가 비어있음을 의미합니다.

int is_full()

큐가 가득 찼는지 확인하는 함수입니다.
원형 큐에서는 rear의 다음 위치가 front와 같을 때 포화 상태입니다.

void queue_print()

큐의 현재 상태를 출력해주는 함수입니다.
front에서 rear까지 순환하며 데이터를 출력합니다.
출력 시 큐의 front와 rear 값을 함께 표시해 상태를 명확하게 보여줍니다.

void enqueue()

큐에 새로운 데이터를 추가하는 함수입니다.
is_full 함수를 통해 큐가 가득 찼는지 확인 후, rear를 1만큼 증가시키고 해당 위치에 데이터를 저장합니다.
삽입 후 큐의 요소(배열)를 확인할 수 있도록 print_queue 함수를 호출합니다.

int dequeue()

큐에서 요소를 제거하는 함수입니다.
is_empty 함수를 통해 큐에 요소가 있는지 확인 후, front를 1만큼 증가시키고 해당 위치의 인덱스를 반환합니다.
배열에서 실제로 데이터를 삭제하지는 않지만, front를 통해 범위를 조정합니다.

int peek()

큐에서 front+1의 인덱스 값만 리턴해주는 함수입니다.
is_empty 함수를 통해 큐에 요소가 있는지 확인 후, front+1의 인덱스를 반환합니다.
front값을 실제로 변경하지 않고 +1을 시켜 배열의 요소만 확인합니다.

int main()

먼저 큐를 선언한 후 초기화합니다.
srand(time(NULL))를 사용하여 랜덤한 숫자를 생성할 수 있도록 해줍니다.
이때 난수를 생성하는 이유는 매번 다른 결과 값을 얻기 위해서 난수 생성을 해줍니다.
이후 for문을 통해 큐에 삽입, 삭제를 100번 반복합니다. 반복하는 동안 큐가 포화상태이거나 공백상태이면 종료합니다.

반복하는 과정
20% 확률로 난수를 큐에 삽입합니다. -> 난수 % 5를 하여 1/5 확률로 큐에 삽입합니다.
다음엔 큐의 현재 상태를 출력합니다..
그리고 10% 확률로 큐에서 요소 1개을 삭제(출력)합니다. -> 난수 % 10를 하여 1/10 확률로 큐의 요소를 삭제합니다.
다시 큐의 현재 상태를 출력합니다.
이 과정을 100번 반복합니다. 오류 발생 시 종료합니다.

5.4 원형 덱 프로그램

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

typedef int element;

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

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

void init_deque(DequeType* q) {
    q->front = q->rear = 0;
}

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

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

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]);
        } while (i != q->rear);
    }
    printf("₩n");
}

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;
}
element get_front(DequeType* q) {
    if (is_empty(q)) {
        error("큐가 공백상태입니다");
    }
    return q->data[(q->front + 1) % MAX_QUEUE_SIZE];
}

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];
}

element delete_rear(DequeType* q) {
    if (is_empty(q)) {
        error("큐가 공백상태입니다");
    }
    int prev = q->rear;
    q->rear = (q->rear - 1 + MAX_QUEUE_SIZE) % MAX_QUEUE_SIZE;
    return q->data[prev];
}

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 get_rear(DequeType* q) {
    if (is_empty(q)) {
        error("큐가 공백상태입니다");
    }
    return q->data[q->rear];
}
```

```c
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;
}
```

코드 결과 출력

```
QUEUE(front=4 rear=0) = 0 |
QUEUE(front=3 rear=0) = 1 | 0 |
QUEUE(front=2 rear=0) = 2 | 1 | 0 |
QUEUE(front=2 rear=4) = 2 | 1 |
QUEUE(front=2 rear=3) = 2 |
QUEUE(front=2 rear=2) =
```

구조체를 사용하여 원형 덱(Deque)을 구현한 코드입니다.
덱이란 앞(front)과 뒤(rear) 양쪽에서 삽입 및 삭제가 가능한 자료구조입니다.
스택과 큐의 특징을 모두 가진 자료구조입니다.

void error()

오류 메시지를 출력하고 프로그램을 종료합니다.
덱이 포화 상태일 경우나 공백일 경우에 사용됩니다.

void init_deque()

덱 초기화 함수입니다.
front 와 rear 를 0으로 설정하여 비어있는 상태로 초기화해줍니다.

int is_empty()

덱이 공백 상태인지 확인하는 함수입니다.
front 와 rear가 같으면 덱이 비어있는 상태입니다.

int is_full()

덱이 포화 상태인지 확인하는 함수입니다.
rear+1을 MAX_QUEUE_SIZE로 나눈 값이 front와 같으면 포화 상태로 1을 반환합니다.

void deque_print()

현재 덱의 상태를 출력해주는 함수입니다.
덱의 front, rear, 인덱스 요소를 함께 출력해 줍니다.

void add_rear()

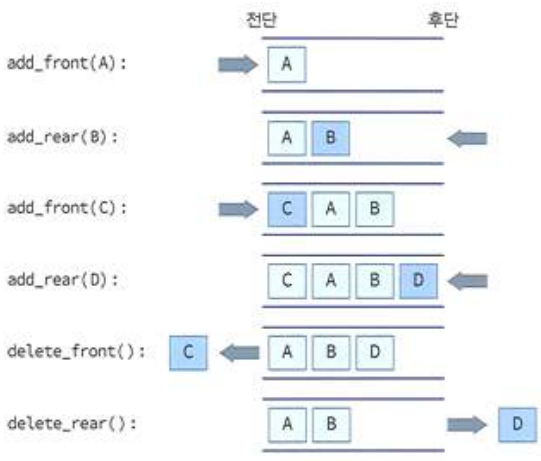
덱의 후단(rear)에 데이터를 추가하는 함수입니다.
포화 상태인지 검사하고, rear를 1 증가시킨 뒤 데이터를 저장합니다.

element delete_front()

덱의 전단(front)에서 데이터를 제거하여 반환합니다.
공백 상태인지 확인 후 front를 1 증가시켜 데이터를 꺼냅니다.

element delete_rear()

덱의 후단(rear)에서 데이터를 제거하여 반환합니다.
공백 상태인지 검사하고, rear를 1 감소시켜 데이터를 꺼냅니다.



선형 덱의 구조

void add_front()

덱의 전단(front)에 데이터를 추가합니다.

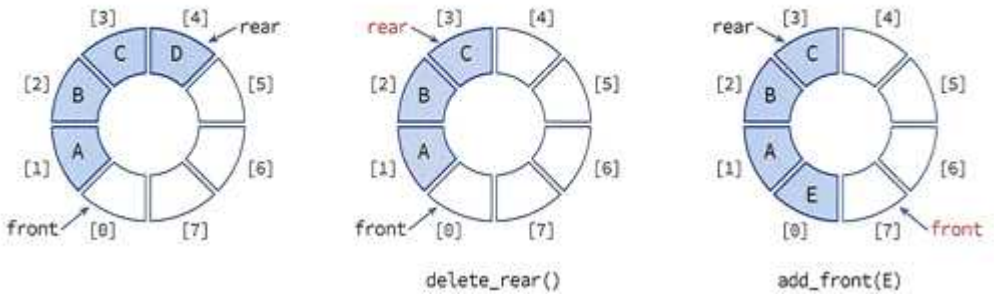
포화 상태인지 검사하고, 새로운 데이터를 저장하고 front를 1 감소시킵니다.

int main()

먼저 덱을 선언하고 초기화시켜 줍니다.

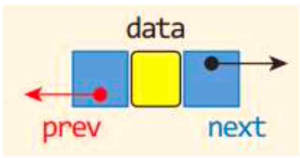
이후 for문을 통해 I의 값을 덱의 전단(add_front)에 삽입시켜 줍니다.

다시 for문을 통해 덱의 후단(delete_rear)부터 요소를 삭제(출력)합니다.

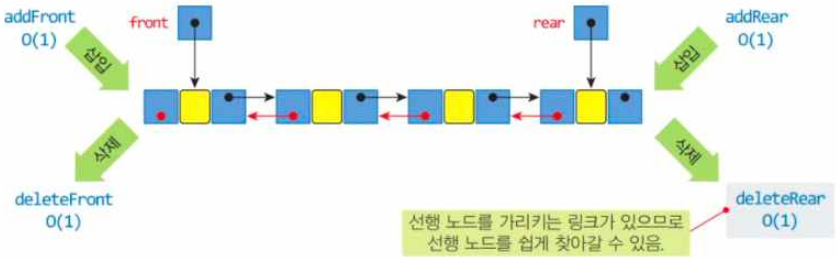


원형 덱의 구조

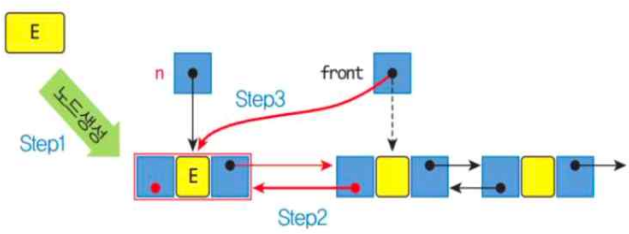
추가적으로 스택, 큐와 같이 덱도 연결리스트로 구현이 가능합니다.
덱은 양쪽에서 삽입, 삭제가 가능하기에 조금 더 복잡한 구조를 가집니다.
삽입, 삭제가 양쪽으로 이루어져야 하기 때문에
노드는 선행노드와 후행노드를 가르키는 포인터 변수를 가집니다.
이것이 7장에서 공부할 이중 연결 리스트입니다.



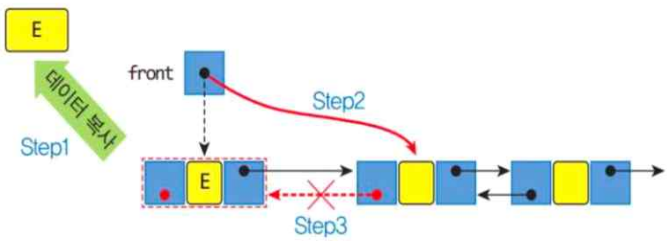
이중 연결리스트의 노드



연결 리스트로 구현한 덱



add_front(), add_rear()



delete_front(), delete_rear()

출처 : https://velog.io/@9e0na/%EC%9E%90%EB%A3%8C%EA%B5%AC%EC%A1%B0-%EB%8D%B1DEQUE

5.5 은행 서비스 시뮬레이션

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

typedef struct {
    int id;
    int arrival_time;
    int service_time;
} element;

#define MAX_QUEUE_SIZE 5
typedef struct {
    element  data[MAX_QUEUE_SIZE];
    int  front, rear;
} QueueType;

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

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

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

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

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");
}

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;


element dequeue(QueueType *q)

    if (is_empty(q))
        error("큐가 공백상태입니다");
    q->front = (q->front + 1) % MAX_QUEUE_SIZE;
    return q->data[q->front];


element peek(QueueType *q)

    if (is_empty(q))
        error("큐가 공백상태입니다");
    return q->data[(q->front + 1) % MAX_QUEUE_SIZE];
```

```c
int main(void)
{
    int minutes = 20;
    int total_wait = 0;
    int total_customers = 0;
    int service_time = 0;
    int service_customer;
    QueueType queue;
    init_queue(&queue);

    srand(time(NULL));
    for (int clock = 0; clock < minutes; clock++) {
        printf("현재시각=%d₩n", clock);
        if ((rand() % 10) < 3) {
            element customer;
            customer.id = total_customers++;
            customer.arrival_time = clock;
            customer.service_time = rand() % 3 + 1;
            enqueue(&queue, customer);
            printf("고객 %d이 %d분에 들어옵니다. 업무처리시간= %d분₩n",
                customer.id, customer.arrival_time, customer.service_time);
        }
        if (service_time > 0) {
            printf("고객 %d 업무처리중입니다. ₩n", service_customer);
            service_time--;
        }
        else {
            if (!is_empty(&queue)) {
                element customer = dequeue(&queue);
                service_customer = customer.id;
                service_time = customer.service_time;
                printf("고객 %d이 %d분에 업무를 시작합니다. 대기시간은 %d분이었습니다.₩n",
                    customer.id, clock, clock - customer.arrival_time);
                total_wait += clock - customer.arrival_time;
            }
        }
    }
    printf("전체 대기 시간=%d분 ₩n", total_wait);
    return 0;
}
```

코드 결과 출력

```
현재시각=0
현재시각=1
현재시각=2
현재시각=3
현재시각=4
현재시각=5
현재시각=6
고객 0이 6분에 들어옵니다. 업무처리시간= 2분
고객 0이 6분에 업무를 시작합니다. 대기시간은 0분이었습니다.
현재시각=7
고객 0 업무처리중입니다.
현재시각=8
고객 0 업무처리중입니다.
현재시각=9
고객 1이 9분에 들어옵니다. 업무처리시간= 3분
고객 1이 9분에 업무를 시작합니다. 대기시간은 0분이었습니다.
현재시각=10
고객 1 업무처리중입니다.
현재시각=11
고객 2이 11분에 들어옵니다. 업무처리시간= 2분
고객 1 업무처리중입니다.
현재시각=12
고객 3이 12분에 들어옵니다. 업무처리시간= 1분
고객 1 업무처리중입니다.
현재시각=13
고객 4이 13분에 들어옵니다. 업무처리시간= 1분
고객 2이 13분에 업무를 시작합니다. 대기시간은 2분이었습니다.
현재시각=14
고객 2 업무처리중입니다.
현재시각=15
고객 2 업무처리중입니다.
현재시각=16
고객 3이 16분에 업무를 시작합니다. 대기시간은 4분이었습니다.
현재시각=17
고객 3 업무처리중입니다.
현재시각=18
고객 5이 18분에 들어옵니다. 업무처리시간= 1분
고객 4이 18분에 업무를 시작합니다. 대기시간은 5분이었습니다.
현재시각=19
고객 4 업무처리중입니다.
전체 대기 시간=11분
```

고객의 도착 시간과 서비스 시간을 기반으로 은행업무 서비스를 큐를 이용하여 구현한 코드입니다.
구조체를 이용하여 각 고객의 정보를 저장하고, 큐를 사용해 서비스 순서를 관리합니다.
시뮬레이션을 통해 각 고객의 대기 시간, 시작 시간, 종료 시간을 계산하며, 전체 평균 대기 시간도 함께 출력됩니다.

typedef struct { } element
고객의 정보를 저장할 구조체입니다.
id, arrival_time, service_time 는 각각 ID, 도착시간, 업무처리 시간을 저장하는 역할을 합니다.

typedef struct { . . . } QueueType
선형 큐를 구현할 구조체입니다.
front, rear는 삽입, 삭제를 element 타입의 data배열은 큐의 요소들을 저장하는 역할을 합니다.

void error()
  오류 메시지를 출력하고 프로그램을 종료합니다.
  덱이 포화 상태일 경우나 공백일 경우에 사용됩니다.

void init_queue()
  큐를 초기화하는 함수입니다.
  front와 rear 인덱스를 0으로 설정하여 초기 상태로 만듭니다.

int is_full(QueueType *q)
  큐가 포화 상태인지 확인하는 함수입니다.
  rear+1을 MAX_QUEUE_SIZE로 나눈 값이 front와 같다면 포화입니다.

void queue_print(QueueType *q)
  큐의 현재 상태를 출력하는 함수입니다.
  front부터 rear까지 순환하며 고객 데이터를 보여줍니다.

void enqueue(QueueType*q,elementitem)
  큐에 새로운 고객 데이터를 추가하는 함수입니다.
  큐가 포화 상태인지 검사한 후 rear를 증가시켜 데이터를 저장합니다.

element dequeue(QueueType*q)
  큐에서 고객 데이터를 반환하는 함수입니다.
  front를 증가시킨 후 데이터를 반환합니다.

int peek()
  큐에서 front+1의 인덱스 값만 리턴해주는 함수입니다.
  is_empty 함수를 통해 큐에 요소가 있는지 확인 후, front+1의 인덱스를 반환합니다.
  front값을 실제로 변경하지 않고 +1을 시켜 배열의 요소만 확인합니다.

int main()
  minutes를 20으로 초기화해 줍니다. -> for문이 종료되는 기준을 설정해 줍니다.
  total_wait을 0으로 초기화해주고 이후 대기 시간을 누적 저장하여 총 대기 시간을 나타냅니다.
  total_customers는 ID를 입력해주기 위한 변수입니다.
  service_time은 현재 처리 중인 고객의 남은 업무 시간을 나타내는 변수입니다.
  service_customer는 지금 처리 중인 고객의 ID를 나타내는 변수입니다.
  변수들을 모두 선언한 뒤 큐를 초기화해 줍니다. 이때 큐는 고객을 차례대로 나열해 주는 역할을 합니다.
  for문을 통해 clock이 1번 반복할 때마다 +1됩니다. 이후 minutes가 될 때까지 반복합니다.
   1. 현재 시간(clock)을 출력해 줍니다.
   2. 3/10확률로 고객을 생성합니다. -> ID, 도착 시간, 업무처리 시간(1~3분 사이)를 설정
   3. 큐에 고객을 삽입하고 정보를 출력합니다.

   만약 고객의 업무 시간(service_time) 이 남아있다면 업무처리 중 문구를 출력하고 1만큼 감소 시킵니다.
   그렇지 않다면 큐가 공백상태인지 확인한 후 큐의 요소를 반환하여 customer에 저장합니다.
   이후 업무 시작 문구와 큐에서 대기한 시간(clock - customer.arrival_time)을 출력하고 총 대기 시간(total_wait)에 더해줍니다.

 for문이 종료되고 나면 총 대기 시간을 출력합니다.