9주차 3차시 큐의 구현2

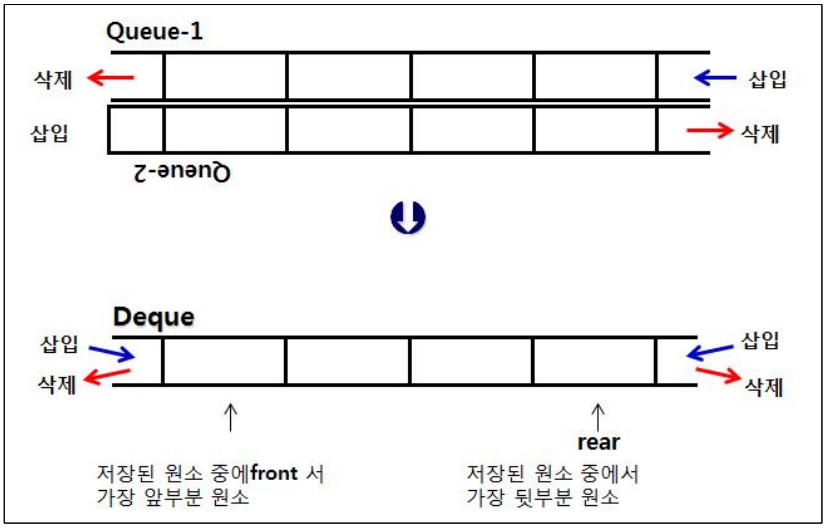
【학습목표】

1. 양방향 링크 필드를 가진 이중 연결 리스트를 이용하여 연결 자료구조를 구현할 수 있다.
2. 선입선출 방식으로 처리해야 하는 문제에 큐를 이용할 수 있다.

학습내용1 : 덱의 이해

1. 덱

- 큐의 양쪽 끝에서 삽입 연산과 삭제 연산을 수행할 수 있는 확장된 큐, 즉 큐 2개를 반대로 붙여서 만든 자료구조



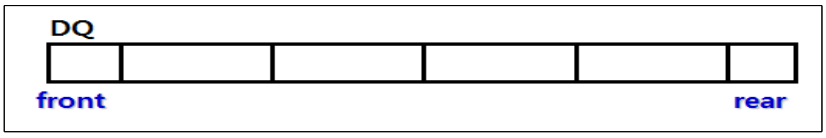
2. 덱에 대한 추상 자료형

```
ADT deque
   데이터 : 0개 이상의 원소를 가진 유한 순서 리스트
   연산 :
        DQ∈deque; item∈Element;
        createDeque() ::= create an empty DQ;
            // 공백 덱을 생성하는 연산

        isEmpty(DQ) ::= if (DQ is empty) then return true
                        else return false;
            // 덱이 공백인지 아닌지를 확인하는 연산

        insertFront(DQ, item) ::= insert item at the front of DQ;
            // 덱의 front 앞에 item(원소)을 삽입하는 연산

        insertRear(DQ, item) ::= insert item at the rear of DQ;
            // 덱의 rear 뒤에 item(원소)을 삽입하는 연산

        deleteFront(DQ) ::= if (isEmpty(DQ)) then return null
                            else { delete and return the front item of DQ };
            // 덱의 front에 있는 item(원소)을 덱에서 삭제하고 반환하는 연산

        deleteRear(DQ) ::= if (isEmpty(DQ)) then return null
                           else { delete and return the rear item of DQ };
            // 덱의 rear에 있는 item(원소)을 덱에서 삭제하고 반환하는 연산
```

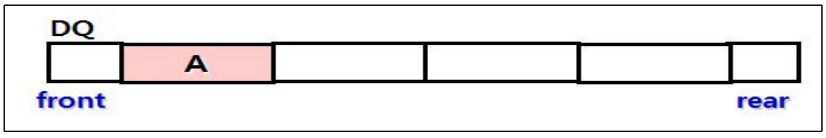
```
        removeFront(DQ) ::= if (isEmpty(DQ)) then return null
                            else { remove the front item of DQ };
            // 덱의 front에 있는 item(원소)을 삭제하는 연산

        removeRear(DQ) ::= if (isEmpty(DQ)) then return null
                           else { remove the rear item of DQ };
            // 덱의 rear에 있는 item(원소)을 삭제하는 연산

        getFront(DQ) ::= if (isEmpty(DQ)) then return null
                         else { return the front item of the DQ };
            // 덱의 front에 있는 item(원소)을 반환하는 연산

        getRear(DQ) ::= if (isEmpty(DQ)) then return null
                        else { return the rear item of the DQ };
            // 덱의 rear에 있는 item(원소)을 반환하는 연산

    End deque
```

* 덱에서의 연산 과정

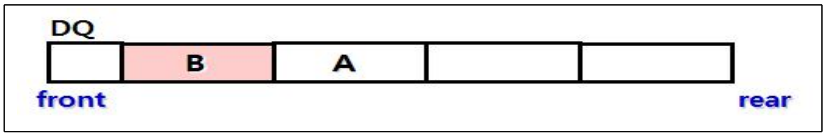
① createDeque();



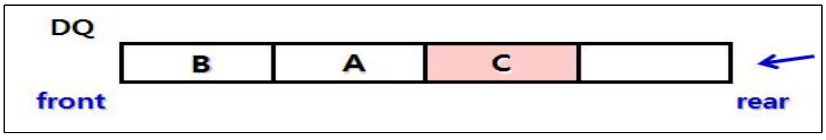
② insertFront(DQ, 'A');



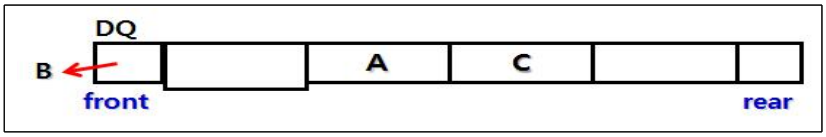
③ insertFront(DQ, 'B');



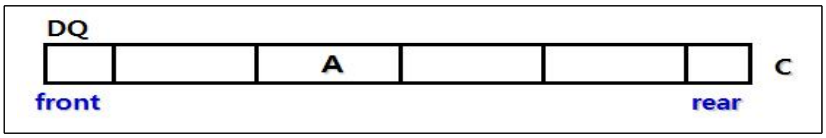
④ insertRear(DQ, 'C');



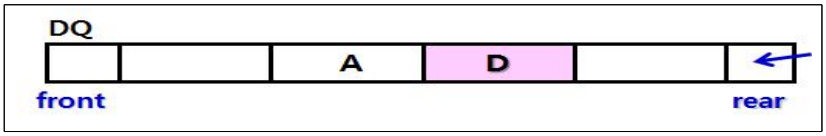
⑤ deleteFront(DQ);



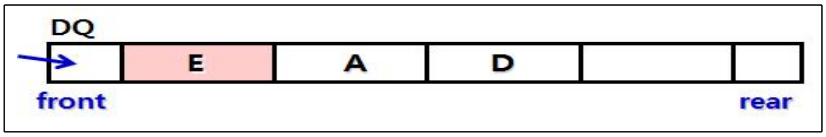
⑥ deleteRear(DQ);



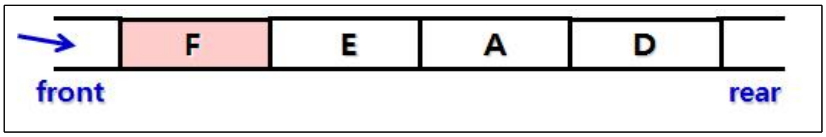
⑦ insertRear(DQ, 'D');



⑧ insertFront(DQ, 'E');



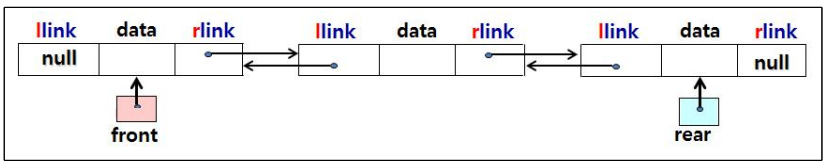
⑨ insertFront(DQ, 'F');



학습내용2 : 덱의 구현

*** 양쪽 끝에서 삽입, 삭제 연산을 수행하면서 크기 변화와 저장된 원소 순서 변화가 많으므로 순차 자료규조는 비효율적임**

*** 양방향으로 연산이 가능한 이중 연결 리스트를 사용한다**



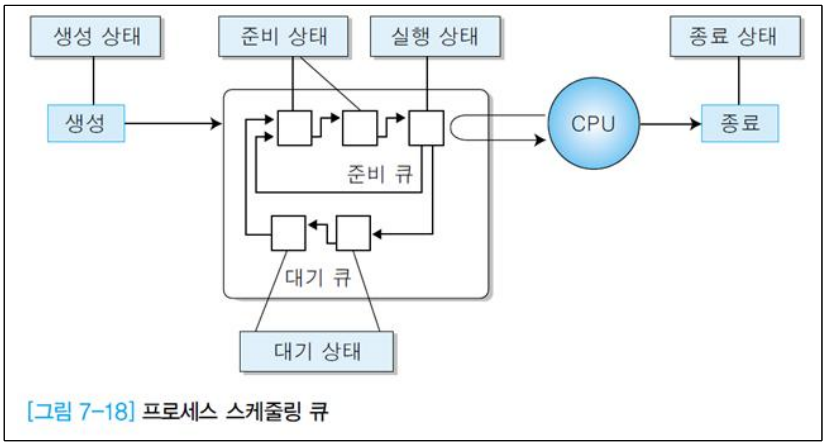
*** [예제 7-4] 이중 연결 리스트를 이용한 덱의 C 프로그램 참조**

학습내용3 : 큐의 응용

1. 운영체제의 작업 큐

① 프린터 버퍼 큐
- CPU에서 프린터로 보낸 데이터 순서대로(선입선출) 프린터에서 출력하기 위해서 선입선출 구조의 큐 사용

② 스케줄링 큐
- CPU 사용을 요청한 프로세서들의 순서를 스케줄링하기 위해서 큐를 사용



[그림 7-18] 프로세스 스케줄링 큐

③ 시뮬레이션 큐잉 시스템
- 시뮬레이션을 위한 수학적 모델리에서 대기행렬과 대기시간 등을 모델링하기 위해서 큐잉 이론 사용

【학습정리】

1. 덱은 큐의 양쪽 끝에서 삽입과 삭제가 모두 가능한 큐로서 스택의 성질과 큐의 성질을 모두 가진다.
- 덱에서 수행하는 양방향 삽입과 삭제를 구현하기 위해서 양방향 링크 필드를 가진 이중 연결 리스트를 이용하여 연결 자료구조를 구현한다.

2. 큐는 선입선출 방식으로 처리해야 하는 문제에 이용할 수 있는데 작업 버퍼 큐, 프로세스 스케줄링, 대기 행렬을 모델링하는 시뮬레이션 등에서 사용할 수 있다.