자료구조 실습 보고서

[제 10 주] 연결 체인으로 구현한 환형 큐

제출일	2017/05/15
학 번	201000287
소 속	일어일문학과
이 름	유다훈

1 프로그램 설명서

1 주요 알고리즘 및 자료구조

- 알고리즘
  - 입출력
    - ◆ 키보드에서 문자를 반복 입력 받는다.
      - i 한 번에 한 개의 문자를 입력 받는다.
    - ◆ 입력 받은 문자가 '!' 이라면 더이상 입력을 받지 않고 종료한다.
    - ◆ 영문자('A' ~ 'Z' / 'a' ~ 'z')를 입력 받는다면 큐에 삽입한다.
      - i 입력 받은 영문자가 큐에 삽입되면 메세지를 출력한다.
      - ii 만일 큐가 꽉 차 있다면 꽉 차 있다는 메세지를 출력한다.
    - ◆ 숫자문자('0' ~ '9') 를 입력 받으면 해당 수 만큼 큐에서 삭제한다.
      - i 숫자를 입력 받으면 숫자만큼의 원소를 삭제한다는 메세지를 출력한다.
      - ii 삭제될 때 마다 원소가 삭제되었다는 메세지를 출력한다.
      - iii 삭제를 시도하였으나 큐가 비어 있으면 큐에 원소가 없다는 메세지를 출력한다.
    - ◆ '-'(하이픈)을 입력 받으면 큐의 맨 앞 원소를 삭제한다.
      - i 삭제할 때 마다 원소가 삭제되었다는 메세지를 출력한다.
      - ii 삭제를 하려는데 큐가 비어 있으면 큐에 원소가 없다는 메세지를 출력한다.
    - ◆ '/'(슬래시)를 입력 받으면 큐의 내용을 큐의 앞부터 뒤까지 차례로 출력한다.
    - ◆ '^'(곡절부호)를 입력 받으면 큐의 맨 앞의 원소의 값을 출력한다.
      - i 큐의 내용은 변하지 않고 맨 앞 원소만 출력한다.
      - ii 만일 큐가 비어 있다면 큐에 원소가 없다는 메세지를 출력한다.
    - ◆ '#'(샵)을 입력 받으면 큐의 사이즈를 출력한다.
    - ◆ 그 밖의 문자들을 입력 받으면 의미 없는 문자가 입력되었다는 메세지를 출력하고 무시한다.

- 자료구조
  - FIFO 구조의 큐를 구현하는 제네릭 타입의 배열
    - ◆ Queue : First-In-First-Out 형태의 자료구조
  - 각종 입출력 시 알림을 전달하는 Enum
  -

1 함수 설명서

Class	AppController			
Method	메소드	파라미터 설명	리턴값	메소드 설명
	AppController()	없음	없음	_appView 변수와 입력받은 문자와 추가된 문자, 무시된 문자의 수를 초기화하는 메소드를 실행하는 생성자 메소드
	run()	없음	없음	환형 큐 프로그램을 실행하는 실행 메소드
	void showMessage(MessageID aMessageID)	에러 메세지 혹은 알림	없음	에러 메세지 혹은 알림에 따라 메세지를 출력하는 메소드
	void initCharCounts()	없음	없음	입력받은 문자, 추가된 문자, 무시된 문자의

				갯수를 0 으로 초기화하는 메소드
	void countAdded()	없음	없음	추가글자 카운트 상승
	void countIgnored()	없음	없음	무시글자 카운트 상승
	void countInputChar()	없음	없음	입력글자 카운트 상승
	void showFrontElement()	없음	없음	큐 내부의 맨 앞의 원소를 출력지시
	void showQueueSize()	없음	없음	큐 내부의 전체 원소 갯수 출력 지시
	void showAll()	없음	없음	큐 내부의 원소를 앞에서부터 순서대로 출력 지시
	void add(Character anElement)	문자	없음	문자를 큐에 삽입
	void removeOne()	없음	없음	큐 내부 원소 1 개 삭제
	void removeN(int numberOfCharsToBeRemoved)	삭제하고 싶은 원소의 갯수	없음	큐 내부의 원소를 전달받은 수 만큼 삭제
	void conclusion()	없음	없음	큐 내부의 원소를 하나하나 삭제하며 출력지시 및 입력글자, 추가글자, 무시글자 의 갯수를 출력.

Class	AppView			
Method	메소드	파라미터 설명	리턴값	메소드 설명
	AppView()	없음	없음	생성자 메소드 값을 입력받는 스캐너 생성
	String inputString()	없음	문자열값	입력 받은 문자열을 리턴하는 메소드
	char inputCharacter()	없음	문자값	입력받은 문자값의 맨 앞 글자(알파벳 1 개)만 리턴하는 메소드
	void outputMessage(String aMessage)	문자열	없음	전달 받은 문자열을 출력하는 메소드
	void outputAdd(char anElement)	문자값	없음	큐에 삽입된 원소가 무엇인지 메세지를 출력하는 메소드
	void outputElement(char anElement)	문자값	없음	큐의 원소 한 개를 출력하는 메소드
	void outputFrontElement(char anElement)	문자값	없음	큐의 맨 앞 원소를 출력하는 메소드
	void outputQueueSize(int aQueueSize)	정수	없음	큐의 사이즈를 출력하는 메소드
	void outputRemove(char anElement)	문자값	없음	큐에서 삭제된 문자값을 전달받아 무엇이 삭제되었는지 출력하는 메소드
	void outputRemoveN(int numberOfCharsToBeRemoved)	정수	없음	큐에서 전달받은 정수값 만큼의 원소를 삭제하겠다는 알림 메세지를 출력하는 메소드
	void outputResult(int aNumberOfInputChars, int	입력받은 문자갯수, 무시된 문자	없음	입력받은 문자 갯수, 무시된 문자 갯수, 저장된 문자 갯수를 출력하는

	aNumberOfIgnoredChars, int aNumberOfAddedChars)	갯수, 저장된 문자 갯수		메소드

Class	CircularlyLinkedQueue<T>			
Method	메소드	파라미터 설명	리턴값	메소드 설명
	CircularlyLinkedQueue()	없음	없음	디폴트값으로 큐의 최대 용량을 초기화하는 생성자 메소드
	boolean isEmpty()	없음	boolean	큐가 비어 있으면 true, 비어있지 않으면 false 를 리턴하는 메소드
	boolean isFull()	없음	boolean	큐가 꽉 차 있다면 true, 그렇지 않다면 false
	int size()	없음	큐 내 원소의 갯수	큐 내부의 원소의 총 갯수 출력
	int capacity()	없음	정수	큐의 최대용량값을 리턴하는 메소드
	T elementAT(int aPosition)	원소가 존재하는 위치	원소	입력 받은 위치에 존재하는 스택의 원소를 반환
	T frontElement()	없음	원소	큐 내부의 맨 앞 원소를 리턴
	boolean enQueue(T anElement)	원소	boolean	큐에 전달받은 원소를 입력하면 true, 꽉 차서 입력하지 못하면 false 를 반환하는 메소드
	T deQueue()	없음	원소	삭제할 원소를 반환하는 메소드
	void clear()	없음	없음	큐 내부를 null 값으로 초기화시키는 메소드. 큐를 비우게 하는 것.

Class	Node<T>			
Method	메소드	파라미터 설명	리턴값	메소드 설명
	Node()	없음	없음	노드를 초기값으로 초기화하는 생성자 메소드
	Node(T givenElement)	원소값	없음	노드의 원소값을 전달받은 원소값으로 초기화하는 생성자 메소드
	Node(T givenElement, Node<T> givenNode)	원소값 노드	없음	노드의 원소값과 노드의 다음 노드값을 전달받은 값으로 초기화하는 생성자 메소드
	T element()	없음	원소값	해당 노드의 원소값을 리턴
	void setElement(T newElement)	원소값	없음	전달받은 원소로 새롭게 원소값을 지정하는 메소드
	Node<T> next()	없음	노드	노드의 다음 노드값을 리턴하는 메소드
	void setNext(Node<T> newNext)	노드	없음	해당 노드의 다음 노드값을 새롭게 지정하는 메소드

2 종합 설명서

- 평범한 영문자를 입력했을 때
  - CircularyLinkedQueue 의 enQueue()메소드를 이용하여 큐에 넣는다.
  - 만약 꽉 차 있다면 isFull()메소드에 의하여 false 가 반환되고, 큐에 원소가 꽉 찼다는 메세지가 출력된다.
- 숫자문자를 입력했을 때
  - AppView 의 outputRemoveN() 메소드를 통하여 입력받은 숫자만큼의 원소가 삭제된다는 알림 메세지를 출력한다.
  - AppController 의 removeN()메소드를 통해 입력 받은 숫자만큼 반복하며 removeOne() 메소드를 실행한다.
  - removeOne()은 CircularyLinkedQueue 의 deQueue()메소드를 실행시켜 원소를 삭제하고, 삭제된 원소를 반환하여 AppView 의 outputRemove 를 통해 출력한다.
  - 만일 CircularyLinkedQueue 의 isEmpty()를 통해 큐가 비어 있다면 에러 메세지를 출력한다.
- '-'(하이픈) 을 입력했을 때
  - removeOne()을 통해 큐의 원소를 하나 삭제한다.
  - removeOne()은 CircularyLinkedQueue 의 deQueue()메소드를 실행시켜 원소를 삭제하고, 삭제된 원소를 반환하여 AppView 의 outputRemove 를 통해 출력한다.
  - 만일 CircularyLinkedQueue 의 isEmpty()를 통해 큐가 비어 있다면 에러 메세지를 출력한다.
- '#'(샵) 을 입력했을 때
  - showQueueSize()를 실행한다.
  - showQueueSize()는 CircularyLinkedQueue 의 size()를 실행하여 원소의 갯수를 리턴 받은 뒤 이것을 AppView 의 outputQueueSize()를 통하여 출력한다.
- '/'(슬래시) 를 입력했을 때
  - showAll()를 실행한다.
  - showAll()는 큐의 사이즈만큼 횟수를 반복하며 CircularyLinkedQueue 의 elementAt()를 실행하고 이것을 AppView 의 outputElement()를 통하여 출력한다.
- '^'(곡절 부호) 를 입력했을 때
  - showFrontElement()를 실행한다.
  - showFrontElement()는 CircularyLinkedQueue 의 frontElement()를 실행하여 큐 내부의 맨 앞 원소를 리턴받은 뒤 AppView 의 outputFrontElement()를 통하여 출력한다.
- '!'(느낌표) 를 입력했을 때
  - conclusion()을 실행한다.
  - conclusion()은 큐의 사이즈만큼 반복하며 removeOne()을 이용하여 큐 내부의 원소를 삭제하며 삭제된 원소가 무엇인지 출력한다.

- AppView 의 outputResult()를 이용하여 입력 받은 문자 갯수, 무시된 문자 갯수, 큐에 추가된 문자 갯수를 출력한다.
- 프로그램을 종료한다.

2 프로그램 장단점 분석

- 장점
  - 환형(끝이 이어져있는 둥근 원형태)의 Queue 를 구현하여 얼마나 입력될지 모르는 상태에서 처음부터 많은 메모리공간을 배열 형태로 선언할 필요가 없다.
  - FIFO(선입선출) 형태의 Queue 를 구현해볼 수 있다.
  - 연결 체인의 형태를 이용하여 메모리의 제한 없이 환형 큐 구조를 구현해볼 수 있다.

- 단점
  - 환 형태의 연결체인을 구현하는 데에 약간의 이해도가 필요하다.
  - 본 프로그램에서는 중복검사를 할 수 없다.
  - Queue 의 특성상 무조건 먼저 들어온 것은 먼저 나가야 한다.(First-In-First-Out)
    - 삭제를 할 때 맨 앞의 노드의 원소를 삭제해야 한다.
    - 만일 환 형태의 배열이 아니라면, 앞의 것을 삭제하면 뒤에 것을 하나하나씩 당겨주어야 한다.
    - 그렇지 않으면 앞에는 남아있지만 쓸 수 없는 배열 공간이 발생하므로 메모리 낭비를 하게 된다.

3 실행 결과 분석

1 입력과 출력

프로그램 실행

```
> 프로그램을 시작합니다.
[큐 입력을 시작합니다.]
```

입출력 1-1 - 영문자 입력 및 큐 삽입

```
- 문자를 입력하시오 : A
[EnQueue] 삽입된 원소는 A 입니다
```

입출력 1-2 - 영문자 입력 및 큐가 꽉 찼을 때의 출력

```
- 문자를 입력하시오 : z
ERROR : 큐가 꽉 차서 삽입이 불가능합니다.
```

입출력 2 - '#'(샵) 입력 및 큐의 내용물 총 갯수 출력

```
- 문자를 입력하시오 : #
[Size] 큐에는 현재 2개의 원소가 있습니다.
```

입출력 3-1 – '-'(하이픈) 입력 및 큐에 삭제할 원소가 없을 때

```
- 문자를 입력하시오 : -
[Empty] 큐에 원소가 없습니다.
```

입출력 3-2 - '-'(하이픈) 입력 및 큐에 삭제할 원소가 있을 때 삭제된 원소 출력

```
- 문자를 입력하시오 : -
[DeQueue] 삭제된 원소는 A 입니다
```

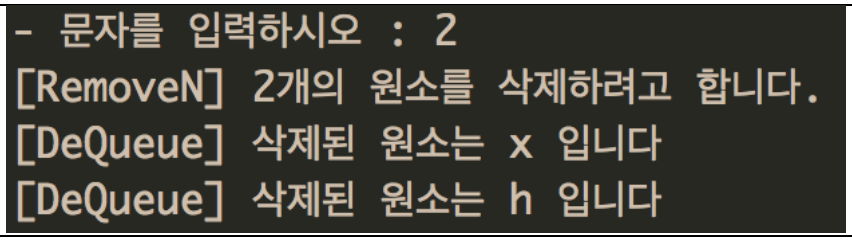
입출력 4 - '/'(슬래시) 입력 및 큐에 들어있는 원소를 차례로 출력

```
- 문자를 입력하시오 : /
[Queue] <Front> A x h <Rear>
```

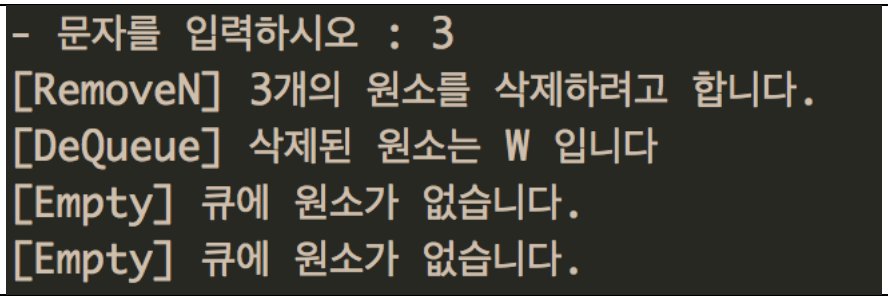
입출력 5 - '^'(곡절 부호) 입력 및 큐의 맨 앞의 원소 출력

```
- 문자를 입력하시오 : ^
[Front] 맨 앞 원소는 x 입니다
```

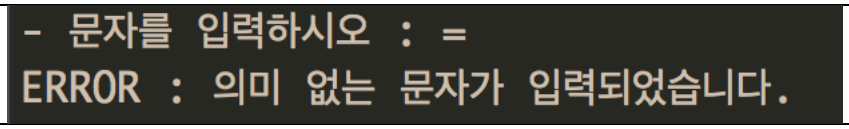
입출력 6-1 - 숫자 입력 및 숫자 개수 만큼의 원소 삭제



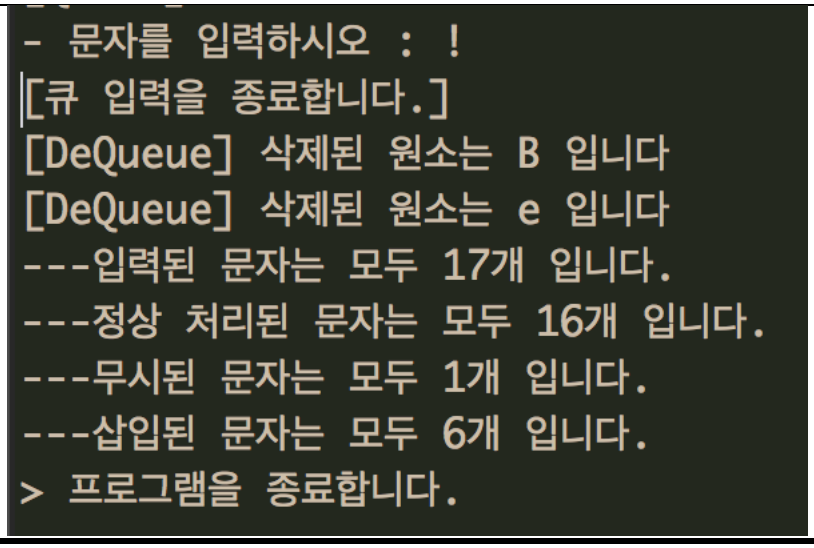
입출력 6-2 - 숫자 입력 및 숫자 개수 만큼의 원소가 존재하지 않을 때.



입출력 7 - 의미 없는 문자가 입력되었을 때



입출력 8 - '!'(느낌표) 입력 및 큐 내부 원소 삭제, 입력받은 문자, 무시된 문자, 삽입된 문자 개수 출력 및 프로그램 종료



2 결과 분석

- First-In-First-Out 개념의 Queue 자료구조를 구현하여 원소를 추가하면 차곡차곡 쌓이고, 원소를 꺼내려면 제일 앞(아래) 원소를 꺼내는 형태의 프로그램을 구현하였다.
- 한 가지의 큐를 구현 해놓고, 제네릭 타입으로 선언하여 똑같은 구조의 여러개의 다른 자료형의 큐를 구현할 수 있다.
- Queue 를 환형(시작점과 끝점이 이어져 있는 둥근 원 형태)로 구현한 이유는?
  - 환형이 아닌 일반 배열을 이용하여 구현할 경우, Queue 의 특성상 먼저 들어온 것은 먼저 나가야 하므로 배열의 앞 부분을 없애 주어야 한다.
- 연결 체인

- 삽입과 삭제
  - i 먼저 들어온 것은 먼저 나가야하는 Queue 의 특성
  - ii 들어온 순서대로 차곡차곡 뒤로 이어져야 하며 삭제는 앞의 노드를 빼야함.
  - iii 그렇기 때문에 맨 앞 노드와 맨 뒤 노드를 다 알고 있어야함.
  - iv 환형 구조로 만들면 맨 뒤의 노드의 다음 노드를 맨 앞 노드로 지정하면 굳이 맨 앞 노드를 알고 있지 않아도 상관 없음.
  - v 삭제 시에는 맨 뒤 노드의 다음 노드를 삭제해주고, 다음 노드를 새로 지정해주는 것으로 끝낼 수 있음.

- 배열과의 장단점
  - 장점
    - i 배열처럼 미리 메모리 공간을 선언하지 않아도 사용할 수 있음
    - ii 배열처럼 삭제할 값과 추가할 값을 위해 위치를 계산하지 않아도 됨.
  - 단점
    - i _rear 값이 무엇인지 모르게 된다면 큐 내부의 모든 데이터를 잃어버리게 됨.
    - ii 큐의 데이터가 증가하면 할 수록 메모리 사용량이 크게 늘어남.
    - iii 원하는 위치의 데이터를 찾기 위해서는 그만큼의 시간이 걸림.
    - iv 미리 사용할 메모리의 용량을 선언한 다음 그것을 환형구조로 이용한다면 용량을 효율적으로 사용할 수 있으나, 연결체인의 경우 메모리 용량을 처음부터 선언하지 않으므로 데이터를 추가하는대로 계속해서 큐의 최대용량이 늘어나게 됨.

- Queue 를 interface 로 선언하는 것은?
  - Queue 에서 쓰이는 개념인 enQueue 와 deQueue 는 어떠한 Queue 라던지 쓰이는 개념이므로 interface 로 선언하고 Queue 를 구현해야할 때라면 언제든지 interface 를 구현하게 하는 것이 좋다고 생각한다.

- 결론
  - Queue 구조를 구현해볼 수 있다.
  - 환 형태의 Queue 구조를 구현해볼 수 있다.
  - 환 형태의 Queue 구조를 구현하기 위해서는 연결 체인의 뒤로 데이터가 입력되고, 삭제는 맨 앞의 데이터가 삭제되어야 한다.
  - 환 형태로 구현하기 위해서는 맨 뒤 노드값의 다음 노드값이 바로 맨 처음 노드값이 된다.
  - 배열과는 달리 위치값을 계산해줄 필요가 없다
  - 배열 형태의 환 구조는 정해진 메모리 용량에서 큐의 특성을 구현할 수 있는 반면, 연결체인 구조는 메모리의 제한 없이 계속해서 추가할 수 있다.
  - 맨 뒤 노드값을 참조할 수 없게 되면 모든 데이터를 잃어버릴 가능성이 높아진다.
  - 환형 Queue 구조를 구현할 때는 배열 형태가 메모리를 효율적으로 관리할 수 있다.