

6

QUEUE ADT

목적

이번 실습에서 여러분은

- 큐의 배열 표현을 기본으로 하고 단일 연결 리스트 표현에 기본을 두는 두 개의 Queue ADT를 구현한다.
- 줄을 서 있는 고객들의 흐름을 모의실험하는 프로그램을 작성한다.
- Dequeue의 배열 구현을 작성한다.
- 배열과 연결 리스트 큐 표현에 메모리 요구사항을 분석하여라.

개요

이 실습은 또 다른 제한된 선형 자료구조인 큐에 중점을 둔다. 큐의 원소들은 이전에 추가된 것(front)부터 최근에 추가(rear)된 순서로 정렬된다. 삽입은 rear에서 수행되고 삭제는 front에서 수행된다. 여러분은 큐에 원소를 삽입하는데 enqueue를 사용하고 원소를 제거하는데 dequeue를 사용한다. enqueue와 dequeue의 순서는 아래와 같다.

| Enqueue | Enqueue | Enqueue | Dequeue | Dequeue |
|---------|---------|---------|---------|---------|
| a | a b | a b c | b c | c |
| ←front | ←front | ←front | ←front | ←front |

queue에서 원소의 이동은 줄을 서 있는 고객 흐름처럼 데이터 채널을 통한 정보 전송과 비슷한 “선입, 선출(first in, first out)”방식이다. 큐는 시스템에서 물리적 객체, 정보 자원 요청의 흐름을 제어하는데 사용된다. 예를 들면 운영체제는 프린터, 파일 통신회선들의 시스템 자원의 접근 제어에 큐를 사용한다. 큐는 시스템에서 객체, 정보 흐름의 모델 구현에 사용된다.

QUEUE ADT

원소

큐 원소들은 일반적인 QE 형태이다.

구조

큐 원소들은 이전에 추가된 원소(front)에서 최근에 추가된 원소(rear)가 선형으로 정렬된다. 원소는 큐의 rear에서 추가(enqueue)되고 front에서 제거(dequeue)된다.

기능

Queue (int maxNumber = defMaxQueueSize)

요구사항:

없음.

결과:

객체 생성자. 빈 큐를 생성한다. 큐가 maxNumber 원소를 수용할 수 있는 메모리를 할당한다.

~Queue ()

요구사항:

없음.

결과:

객체 파괴자. 큐의 할당에 사용된 메모리를 할당 해제한다.

void enqueue (const QE &newElement)

요구사항:

큐는 가득 차 있지 않다.

결과:

newElement를 큐의 rear에 삽입.

QE dequeue ()

요구사항:

큐는 비어 있지 않다.

결과:

큐에서 가장 먼저 추가된 원소를 제거하고 제거한 원소를 반환.

void clear ()

요구사항:

없음.

결과:

큐의 모든 원소를 삭제.

int empty () const

요구사항:

없음.

결과:

큐가 비어 있으면 1을 아니면 0을 반환.

int full () const

요구사항:

없음.

결과:

큐가 가득 차 있으면 1을 아니면 0을 반환.

void showStructure () const

요구사항:

없음.

결과:

큐의 원소들을 출력한다. 큐가 비어 있으면 "Empty queue"를 출력한다. 이 연산자는 시험/디버깅 목적에 사용된다. C++에서 미리 지정된 데이터 형태(int, char등)인 큐의 원소들만 지원한다.

실습 6 : 표지

반과 _____ 구분 _____

성명 _____

교수가 여러분에게 할당한 다음에 연습문제들에 할당된 열에 체크표시를 하고 이 겹장을
앞으로의 연구에서 여러분이 제출할 과제물 앞에 붙이시요



| | 할 일 | 완 료 |
|-----------|-----|-----|
| 실습 전 연습 | V | |
| 연 결 연습 | V | |
| 실습 중 연습 1 | | |
| 실습 중 연습 2 | | |
| 실습 중 연습 3 | | |
| 실습 후 연습 1 | | |
| 실습 후 연습 2 | | |
| | 총 점 | |

실습 6 : 실습 전 연습

날짜 _____ 구분 _____
 성명 _____

이번 실습에서 여러분은 두 개의 Queue ADT를 만든다. 하나는 배열을 기본으로 하고 다른 하나는 단일 연결 리스트를 기본으로 한다.

1단계 : 큐의 원소를 적재하는데 배열을 사용하는 Queue ADT내에서 기능들을 구현하라. 큐의 크기는 변하기 때문에 큐가 가질 수 있는 원소의 최대 수와 front와 rear에 있는 원소들의 배열 인덱스를 저장할 필요가 있다.

 파일 `queuearr.h`에 있는 아래의 선언을 기본 구현하라. ShowStructure 연산자의 구현  은 파일 `show6.cpp`에 있다.

```
Const int defMaxQueueSize=10; // Default maximum queue size
template <class QE>
class Queue
{
public:
    // Constructor
    Queue (int ignored = 0);

    // Destructor
    ~Queue ( );

    // Queue manipulation operations
    void enqueue (const QE &newElement); // Enqueue element
    QE dequeue ( );                       // Dequeue element
    void clear ( );                       // Clear queue

    // Queue status operations
    int empty ( ) const;                  // Queue is empty
    int full ( ) const;                   // Queue is full

    //Output the queue structure--used in testing/debugging
    void showStructure ( ) const;
```



```

private:
    // Data members
    int maxSize, // Maximum number of elements in the queue
        front,  // Index of the front element
        rear;   // Index of the rear element
    QE * element; // Array containing the queue elements
};

```

2단계 : 여러분의 Queue ADT의 구현을 파일 *queuearr.cpp*에 저장하라. 여러분의 코드를 문서화하라.

3단계 : 큐 원소를 저장하는데 단일 연결 리스트를 사용하는 Queue ADT의 기능을 구현하라. 단일 링크드 리스트의 각 노드는 큐의 원소와 다음 원소를 가리키는 포인터를 포함해야 한다. 여러분의 구현은 *front*와 *rear*를 갖는 노드들의 포인터를 유지해야 한다. 파일 *queueInk.h*에 있는 아래의 선언을 기본으로 구현하라. *showStructure* 기능은 *show6.cpp*에서 주어진다.

```

template <class QE>
class QueueNode // Facilitator class for the Queue class
{
private:
    // Constructor
    QueueNode (const QE &elem, QueueNode *nextPtr);

    // Data members
    QE element; // Queue element
    QueueNode *next; // Pointer to the next element

    friend class Queue <QE>;
};

template <class QE>
class Queue
{
public:

    // Constructor
    Queue (int ignored = 0);

```

```

// Destructor
~Queue ( );

// Queue manipulation operations
void enqueue (const QE &newElement); // Enqueue element
QE dequeue ( );                       // Dequeue element
void clear ( );                       // Clear queue

// Queue status operations
int empty ( ) const;                  // Queue is empty
int full ( ) const;                   // Queue is full

//Output the queue structure--used in testing/debugging
void showStructure ( ) const;
private:

// Data members
QueueNode<QE> *front, // Pointer to the front element
              *rear;  // Pointer to the rear element
};

```

4단계 : 파일 `queuelnk.cpp`로 Queue ADT의 링크드 리스트 구현을 저장하라. 여러분의 코드를 문서화하라.

실습 6 : 연결 연습

날짜 _____ 구분 _____
 성명 _____

여러분의 강사와 함께 여러분의 실습 전이나 실습 중에 이 연습문제를 완료할 수 있는지 점검하라.

파일 `test6.cpp`의 시험 프로그램으로 여러분은 상호적으로 아래의 명령어를 사용하는 Queue ADT의 구현을 시험할 수 있다.

| 명령어 | 동작 |
|-----|---------------------|
| +x | 원소 x를 enqueue한다. |
| - | 원소를 dequeue하고 출력한다. |
| E | 큐가 비었는지 알린다. |
| F | 큐가 차 있는지 알린다. |
| C | 큐를 clear한다. |
| Q | 테스트 프로그램을 종료한다. |

1단계 : 시험 프로그램을 링크하고 컴파일하라. 이 프로그램을 컴파일하는 것은 문자들의 큐에 대한 배열 구현을 실행하기 위해 Queue ADT의 배열 구현을 컴파일하는 것이다.

2단계 : 다음의 시험 계획에 여러분의 시험 항목을 추가하여 완성하라

- 일련의 dequeue로 비게 된 큐에 원소를 enqueue한다.
- 배열(배열 구현)의 “마지막 부근으로 가도록” enqueue와 dequeue를 조합하라.
- 가득찬 큐에서 원소를 dequeue한다(배열 구현).
- 큐를 비운다.

3단계 : 여러분의 시험 계획을 수행하라. Queue ADT의 배열 구현에서 잘못이 있으면 수정하고 다시 실행하라.

4단계 : 파일 `queuelnk.cpp`의 Queue ADT의 연결 리스트 구현이 배열 구현의 위치에 포함되도록 시험 프로그램을 수정하라.

5단계 : 시험 프로그램을 다시 연결하고 컴파일하라. 이 프로그램을 재컴파일하는 것은 여러분의 문자들의 큐에 대한 연결 리스트 구현을 만들기 위해 Queue ADT의 연결 리스트를 컴파일할 것이다.

6단계 : Queue ADT의 연결 리스트 구현 검사에 시험 계획을 사용하라. 여러분의 구현에서 잘못이 발견되면 수정하고 다시 실행하라.

| Queue ADT내의 연산 시험 계획 | | | |
|----------------------|-------------|---------|----|
| 시험 항목 | 명령어 | 예상 결과 | 검사 |
| enqueuees 연속 | +a +b +c +d | a b c d | |
| dequeuees 연속 | --- | d | |
| enqueuees 추가 | +e +f | d e f | |
| dequeuees 추가 | - | f | |
| Empty ? Full? | E F | 0 0 | |
| 큐 비우기 | - | 빈 큐 | |
| Empty ? Full? | E F | 1 0 | |

※ 참조 : front 원소는 굵은 체로 표시