실습과제 6.1

[과제 설명]

큐(Queue)는 데이터의 삽입과 삭제가 서로 다른 위치에서 일어나는 자료구조이다. 먼저 입력된 것이 먼저 출력되기 때문에 선입선출, FIFO(First-In-First-Out)이라고도 한다. 스택과 다르게 데이터의 삽입과 인출이 다른 곳에서 일어나기 때문에 위치를 표시하는 index가 rear(삽입), front(삭제)로 2개 필요하다.

삽입과 삭제를 할 때는 index를 하나씩 증가하면서 수행하게 되는데, 단순 배열로 구현한 큐는 이렇게 하면 데이터 삭제(dequeue)로 인해 발생되는 빈 공간 처리가 필요하다. 삭제를 할 때마다 데이터를 이동하면 리소스가 낭비되기 때문에 이것을 해결하기 위한 것이 원형 큐이다.

원형 큐를 쓰는 경우 빈 공간은 해결 되지만 full과 empty가 구문되지 않는 경우가 발생한다. 이것을 방지하기 위해 배열 한 칸을 비워서 사용한다. 그렇게 하면 front가 rear의 바로 앞에 위치하면 큐가 full인 상태이고, front와 rear의 위치가 같으면 큐가 empty인 상태인 것이다.

선형 배열을 이용한 큐를 원형처럼 사용하는 방법은 데이터를 넣고 뺄 때 전체 크기로 %연산을 하면 된다. 또, full과 emtpy를 구분하기 위해 한 칸을 비워서 사용하는 방법일 때 full을 검사하기 위해서는 rear+1을 사이즈로 나눈 나머지를 비교한다.

[코드]

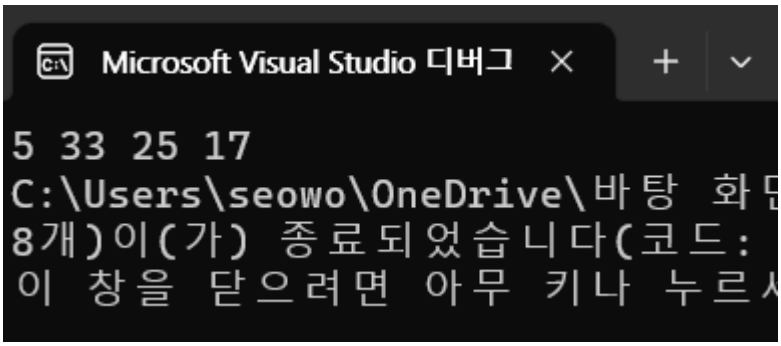
메인 함수는 교안에서 미리 구현된 것을 사용했으므로 보고서에는 생략하고 구현된 함수 세부 내용이 포함된 헤더 파일(MyCircularQueue.hpp)의 코만 첨부한다.

```
#ifndef __MY_CIRCULAR_QUEUE_HPP__
#define __MY_CIRCULAR_QUEUE_HPP__
#define QUEUE_SIZE 100
template <typename T>
class CircularQueue {
private:
  int front; //데이터 시작 위치(-1)
  int rear; //데이터 마지막 위치
  T* arr; //큐의 실질적 데이터 저장 공간
```

```
  int qSize; //큐의 길이를 저장
public:
  CircularQueue() {
    front = -1;
    rear = -1;
    arr = new T[QUEUE_SIZE];
    qSize = QUEUE_SIZE;
  }
  CircularQueue(int size) {
    front = -1;
    rear = -1;
    arr = new T[size];
    qSize = size;
  }

  //큐가 비어 있는지 확인
  bool IsEmpty() {
    if (front == rear)
      //front가 rear와 같은 위치에 있으면 비어있는 것
      return true;
    else
      return false;
  }

  //큐가 가득 차 있는지 확인
  bool IsFull() {
    if (front == (rear + 1) % QUEUE_SIZE)
      //front가 rear보다 한 칸 앞에 있으면 가득 찬 것
      return true;
    else
      return false;
  }

  //큐에 데이터를 삽입
  bool Enqueue(T item) {
    if (IsFull()) {
      puts("큐가 가득 찼습니다.");
      exit(1);
    }

    rear = (rear + 1) % QUEUE_SIZE;
    arr[rear] = item;
  } //rear(삽입 위치)를 하나 증가시키고(원형이기 때문에 전체 크기로 %연산을 한다)
  //그 위치에 해당되는 곳에 데이터를 삽입

  //큐에서 데이터를 꺼냄
  T Dequeue() {
    if (IsEmpty()) {
      puts("큐가 비었습니다.");
      exit(1);
    }
    front = (front + 1) % QUEUE_SIZE;
    return arr[front];
  } //front(인출 위치)를 하나 증가시키고(원형이기 때문에 rear처럼 전체 크기로 %연산을 한다)
  //그 위치에 해당되는 곳에 데이터를 인출한다.

  //최상단 데이터를 확인
  T Peek() {
```

```
    return arr[rear];
  }//데이터 확인만 하기 때문에 rear, front를 움직이지 않는다.

  ~CircularQueue() {
    delete[] arr;
  }
};
#endif
```

[실행 결과]



[과제에 대한 고찰]

스택은 선입후출이고, 큐는 선입선출이라서 배열로 구현해도 조금만 추가되고 크게 달라지는 것이 없을 것 같았다. 하지만 리소스를 최대한 적게 사용하려고 배우는 것이 자료구조인 만큼, 자료구조의 특징이 조금만 달라져도 많은 부분이 달라졌던 것 같다.

스택은 일렬로 사용하기 때문에 데이터 인출을 하지 않아도 전체 데이터를 볼 때도 쉬웠던 것 같은데, 큐는 사용하는 방법 때문인지 삽입된 데이터를 보려고 해도 간접적으로 출력하는 것 같았고, 제대로 데이터가 들어갔는지 보려면 데이터 인출을 해야해서 이 부분을 개선하는 방법을 찾는 것도 좋을 것 같다.