Lecture 10 : 큐(Queue)

큐(queue)는 스택과 대별되는 가장 대표적인 adaptor이다. 스택은 insertion과 deletion이 하나의 입구에서만 일어나는 것에 비해서 queue는 이 둘이 서로 다른 linear structure의 양끝에서 일어난다. 스택을 Last In First Out(LIFO)구조라고 부르듯이 큐는 First In First Out(FIFO) 구조라고 부르기도 한다.[1)]

사람들이 하나의 줄로 서있는 것은 보통 "큐"라고 부른다.[2)] 줄을 설 때의 기본은 도착하는 순서대로 줄의 제일 뒤(rear)로 가야하고, 나가는 순서는 줄에 앞쪽(front)부터 나간다. 즉 어떤 서비스를 받는 순서는 front의 원소부터이다. 스택에 비해서 큐는 가장 흔히 사용되는 자료구조이므로 코드로 잘 활용할 수 있어야 한다. 지원하는 동작은 스택과 그대로 대응된다. 그 동작, member function의 이름은 stack과는 조금 다른데 stack.push( ), stack.pop( )과 다르게 queue.insert( ), queue.pop( )이 있다.

큐와 유사한 양방향 큐(Double ended Queue, DeQue)가 있는데 이것은 adaptor가 아니라 container이다. adaptor와 container와의 차이는 초기화(initialization)를 할 수 없다는 것이다. 즉 stack과 queue에는 배열과 같이 int x[3]= {1,2,3}, 이런 식으로 한꺼번에 그 안을 채울 수 없다는 것이다. 여러분은 아래 코드에서 container와 adaptor 의 차이를 확인해야 한다.

```
int main() {
   // queue <int> myQ { 31, 28, 31, 30, 31, 31, 30};
   // 이렇게 하며 안된다.
   deque<int>  myD {11, 12, 13, 14, 15, 16, 17} ;
   deque<string> herD {"개", "소", "닭", "뱀", "용", "뽕"} ;
   queue<int, deque<int>> myQ(myD) ;

   dump_Q( "myQ[]= ", myQ ) ;

   queue<int> yourQ ;
   yourQ = myQ ;
   dump_Q("\n yourQ[]= ", yourQ ) ;

   queue<string, deque<string>> herQ(herD) ;
   dump_Q("\n herQ[]= ", herQ ) ;
}
```

Deque는 배열에 추가의 편리한 기능이 추가된 container이다. 즉 제일 앞, 제일 뒤로 원소를 추가할 수도 있고, 같은 방식으로 앞뒤에서 원소를 삭제할 수도 있다. 그리고 Deque는 일반적인 배열에서 가능한 random access, update 등의 작업을 할 수

1) 피포(LIFO), 리포(FIFO)라고 부르는 것은 오래된 개념으로 요즘은 거의 사용하지 않는다. 그냥 스택, 큐로 부르는 것이 일반적이다. 또는 어떤 사람들은 "선입선출" "선입후출" 이렇게도 부르는데 일본식 표현이 아닌가 한다. 이런 정체불명의 용어는 사용하지 않는 것이 좋다.

2) "새치기 하지 마시오!"의 영어식 표현 중에는 "Hey, Do not break the queue"가 있다.

있으므로 매우 편리한 자료구조이다. 뭔가 시간적 순서를 다루는 자료구조라면 꼭 dequeue를 활용해야 하고 이것을 사용하는 연습을 해야 한다. 다음 참고 코드를 보자.

```
int main() {
    deque <int> dq = { 5, 6, 7, -5, -4, -3, 100 };
    int i = 1;
     for(auto w : dq) { // python-like loop이 가능
        cout << "\n "<< ++i << ") " << w ;
    }
    allout("\n 1. dq[ ] 내용 = ", dq ) ;
    dq[3] += 10000 ; // random access가 가능.
    allout("\n 2. dq[ ] 내용 = ", dq ) ;
    dq.resize(3) ;
    allout("\n 3. dq[ ] 내용 = ", dq ) ;
}
```

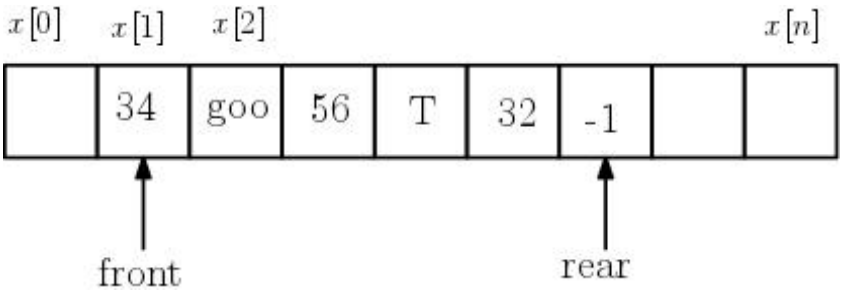
가장 중요한 것은 앞서 stack의 경우와 같이 어떤 상황에서 queue를 사용해야 하는가를 이해하는 것이다.

10.1 Linear queue의 기본동작 (구현은 어떻게 해도 상관이 없다.)

(a) 임의의 두 원소 x, y에 대하여 앞뒤 순서가 항상 정해지며

(b) 만일 x < y 이며 동시에 y < z 이면 x < z인 transitive 규칙이 항상 적용된다.

(b) 삽입(삭제)은 rear(front)에서만 일어난다. 출입구가 2군데. (stack은 1군데)



10.2 그런데 실제 추상적 의미의 일반 큐(Queue)는 실제 STL에서는 양쪽에서 입출력이 가능한 deque[3]로 구현되어 있다. 허용되는 동작과 그 실제의 구현은 다르다. 만일 이 deque를 사용한다면 STL에서 #include <deque>를 반드시 선언해야 한다. 중요한 것은 deque는 container이며 queue은 그 위에 뭔가를 한 겹(layer) 씌운 또 다른 layer인 adaptor라는 것이다. stack이나 Queue 그 자체는 저장 공간이 아니라, 어떻게 자료를 넣고 빼고할 것인지를 정의하는 어댑터임을 항상 명심해야 한다.

- 앞뒤 모두에서 삽입+삭제가 가능하다.
- 접근자도 임의접근 반복자를 사용할 수 있다.
- 자료구조의 끝이 아니라 그 중간에 삽입+삭제는 불가능하다. 왜냐고 ?

3) 이 자료구조는 어떤 사람은 "디큐"로, 또는 "데크", "덱"으로 발음하기도 한다. dequeue인 경우는 "디큐", deque는 "데크"로 부르는 것이 일반적이다.

만일 중간 삽입, 삭제, 그 동적인 작업을 원한다면 다른 자료구조인 list를 써야 한다.
소 잡는 칼, 연필깍는 칼, 통나무를 패는 일에 따라서 가장 적절한 도구를 사용해야 한다.

10.3 Deque에서 지원하는 동작은 다음과 같다.

생성 : **deque( )**

위치파악 : **at( )** **back( )** **front( )**

넣고+빼기: **push_back( ), push_front( )**
pop_back( ), pop_front( )

```
int main() {

  deque <string>  beerbar ;

  beerbar.push_front("Larger") ;   //앞쪽으로 넣습니다.
  beerbar.push_back ("Ale") ;
  beerbar.push_front("IPA") ;
  beerbar.push_back ("Stout") ;
  beerbar.push_front("Mead") ;
  beerbar.push_front("막걸리") ;
  beerbar.push_back ("Dunkel") ;

  allout("Beerbar Before: ", beerbar ) ;
  cout << "Size() = " << beerbar.size() << endl ;

  beerbar.pop_back() ;
  beerbar.pop_back() ;
  beerbar.pop_front() ;
  allout("Beerbar After: ", beerbar ) ;
  return 0;
} // end of main(
```

10.4 Queue, Deque에서 발생하는 오류를 나열해보시오. 그리고 실제 STL queue에서 하나 이상의 System 오류를 발생해보고 그것을 구체적으로 기록해 보시오.

10.4 만일 이것을 교재에서 말한 대로 C/C++ array로 구현하는 것과 STL queue를 사용하는 것과 어떤 점에 차이가 있는지 설명해 보시오.

10.5 Queue에서 지원하는 대표적인 동작은 다음과 같다. 각각의 return되는 value가 무엇인지 실제 프로그램으로 파악해보자.

- queue( ) back( ) front( )
- empty( ) size( )
- push( ) pop( )

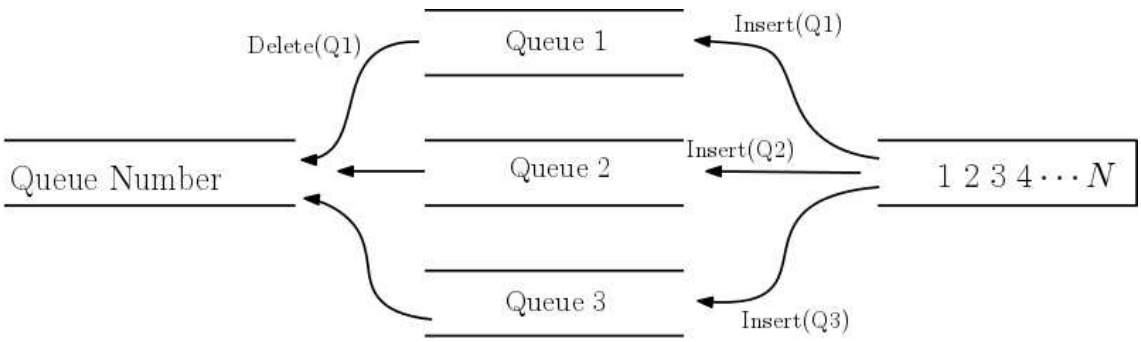
10.6 deque에서도 insert( ), erase( )가 지원된다. 어떤 차이점이 있을까 ?
다음 2개 동작을 비교하시오,

a) my_deque.insert (my_deque.begin( ) + 5 , x)
b) my_vector.insert(my_vector.begin( ) + 5 , x)

10.7 vector를 지원하는 동작 reserve( )를 사용해보고, 이것을 deque( )에 사용하는 의미에 대하여 논의해보시오.

10.8 $k-$Queue Number, (k개의 Queue로 출력되는 어떤 순서 $[\pi_i]$가 가능한지를 결정하는 문제이다.

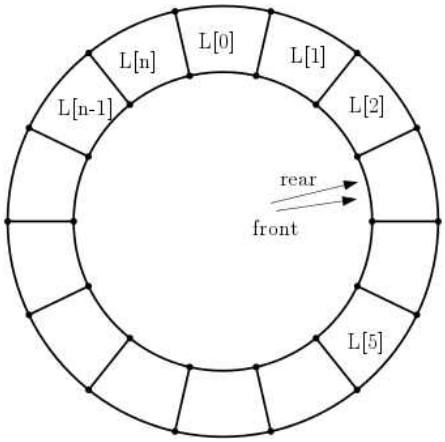
예) 3 5 2 1 7 6 4 는 2-Queue number 인가 ?

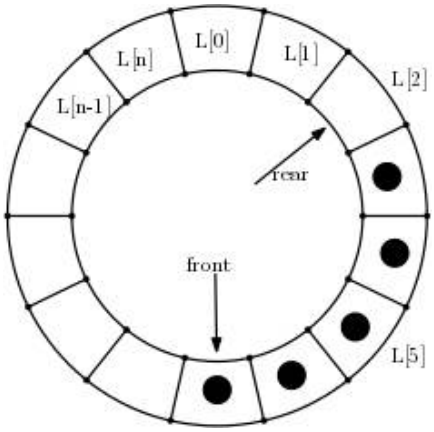


10.9 환형 큐(Circular Queue)[4]
- 배열로 Queue를 구현 할 경우 반드시 해결해야 하는 문제

원소는 없는데 준비된 Queue의 공간은 부족하다.

- Python의 List에서는 필요 없다.
- full 조건과 empty 조건을 반드시 구별해야 한다. (까다롭다...)





10.10 Python과 Queue

Python에서 제공하는 List 자료구조는 Deque를 표현하기에 가장 적절한 자료형이다.

```
L= [ ]
L.append( 45 )
....
L.pop( )
L.remove( "Good")
del L[2]
```

10.11 vector에서 제공해주는 부울(Boolean) 함수인 "전체 비교", "상등 관계" 역시 모두 queue에서 지원된다. 여러분에게는 여러 개의 queue가 주어져 있다. 각각의 정수를 모두 담고 있다. 이들을 하나의 덩어리로 파악하여 사전식 순서로 정렬하는 프로그램을 구성하시오. 어떤 문제에서 이 기능을 사용하면 좋은지 그 경우를 생각해보라.

4) 현실적으로 거의 사용되지 않는 old-fashioned data structure이다. 문제는 아직도 여기에 관련된 문제가 각종 자격증 시험에 나온다는 것이다.

Q1	*4*	*67*	*6*	*45*	*-9*	*13*			
Q2	*45*	*34*	*12*	*-9*					
Q3	*4*	*67*	*-7*	*34*					
Q4	*0*	*1*	*2*	*3*	*4*				
Q5	*-9*	*-11*	*6*	*4*	*6*	*7*	*1*	*2*	*3*
Q6	*a*	*1*	*2*	*3*	*4*	*5*			

10.12 Queue가 응용되는 예

- 은행 서비스 번호표 발급기
- Computer에서 process(task) scheduling
- 공항에서 활주로를 사용하는 항공기 관제
- 자동 택배물 배정 시스템
- 그래프 탐색
- 특정 지점에서 거리 k에 있는 지점 모두 출력하기
- facebook에서 나와 가까운 친구 찾기

10.13 [실전문제] k-명의 택배기사가 있다. 이들에게 물류센터에서 배달할 물품을 배정한다. 각 물품은 컨테이너에 1번부터 n번까지 줄지어 들어온다. 각각의 무게는 w_i로 표시된다. 어떤 물품은 매우 무겁기 때문에 택배기사별로 배달 총량의 차이가 크지 않도록 적절히 균형을 맞추어야 한다. 가장 간단한 방법은 현재 준비 중인 k명의 기사의 현재 배정 물품의 총량의 합이 가장 적은 사람에게 먼저 배정한다. 만일 그 값이 같으면 기사의 index i번호가 빠른 사람을 우선하여 배정한다. 아래 예를 보면서 설명해보자. 준비된 기사는 4명으로 가정한다.

i	1	2	3	4	5	6	7	8	9	10	11	12	13	14
w_i	8	3	2	5	11	4	8	6	13	2	4	5	9	3

첫 5개의 물품을 배정하는 과정을 보자. 8(kg)의 1번 물품은 모든 총합이 0이므로 가장 index가 빠른 1번 기사에게 배정한다. 이어 3, 2, 5 물품은 각각 2, 3, 4번 기사에게 배정한다. 이제 5번 $w_i = 11$을 생각해보자. 현재 4명의 기사 중 총합 무게가 가장 적은 사람인 3번(2 kg.)에게 배정하고 2번의 무게는 13이 된다.

1-5번까지 물품의 배정 결과

기사/ t				
1	8			
2	3			
3	2	11		
4	5			

6번에서 10번까지 물품의 배정 결과

기사/ t				
1	8	13		
2	3	4	6	
3	2	11		
4	5	8	2	

여러분은 $w_8 = 6$인 경우를 잘 살펴보아야 한다. w_7 까지 배정했을 때 총합이 가장 작은 기사는 2번 기사로 그 무게가 7이다. 이와 비해서 다른 기사에게 배정된 무게 총합은 다음과 같다. 1번 기사는 8, 2번 기사는 7, 3번 기사는 13, 4번은 13이기 때문에 2번에게 배정한다.