

7장. 큐

□ 스택, 큐

- 리스트 작업은 시간과 무관하게 정의
- 스택과 큐의 작업은 시간을 기준으로 정의
- 큐는 가장 먼저 삽입된 데이터가 먼저 삭제되는 특성의 자료형을 추상화

□ 학습목표

- 추상 자료형 큐의 기본 개념을 스택과 대조하여 이해한다.
- 추상 자료형 큐를 구현하기 위한 세 가지 방법을 이해한다.
- 원형 배열이 필요한 이유와 동작원리를 이해한다.
- 큐의 응용 예를 구체적으로 명확하게 이해한다.

큐

□ 큐 = 대기열



□ 대기열을 모델링

- 선입선출, FIFO, FCFS

□ 용어

- 줄의 맨 앞을 큐 프론트(Queue Front)
- 맨 뒤를 큐 리어(Queue Rear)
- 큐 리어에 데이터를 삽입하는 작업 = 큐 애드(Add)
- 큐 프론트의 데이터를 삭제하는 작업 = 큐 리무브(Remove)

	삽입	삭제	검색
ADT 리스트	Insert	Delete	Retrieve
ADT 스택	Push	Pop	GetTop(PeekTop)
ADT 큐	Add(Enqueue)	Remove(Dequeue)	GetFront(PeekFront)

추상 자료형 큐

□ 작업

- Create: 새로운 큐를 만들기
- Destroy: 사용되던 큐를 파기하기
- Add: 현재의 리어 바로 뒤에 새로운 데이터를 삽입하기
- Remove: 프런트에 있는 데이터를 가져오기
- GetFront: 현재 큐 프런트에 있는 데이터를 검색하기(읽기)
- IsEmpty: 현재 큐가 비어있는지 확인하기
- IsFull: 현재 큐가 꽉 차 있는지 확인하기
- GetSize: 현재 큐에 들어가 있는 데이터 개수를 알려주기

□ 액시엄

- $\text{GetFront}(\text{Add}(\text{Create}(Q), V)) = V$
- $\text{GetFront}(\text{Add}(\text{Add}(Q, W), V)) = \text{GetFront}(\text{Add}(Q, W))$
- $\text{IsEmpty}(\text{Create}(Q)) = \text{TRUE}$
- $\text{Remove}(\text{Create}(Q)) = \text{ERROR}$
- $\text{GetFront}(\text{Create}(Q)) = \text{ERROR}$

C++ 연결 리스트에 의한 큐

❑ 코드 7-1: QueueP.h (C++ Interface by Linked List)

```
typedef struct
{ int Data;
  node* Next;
} node;
typedef node* Nptr;
const int MAX = 100;
```

큐 데이터를 정수 형으로 가정
다음 노드를 가리키는 포인터 변수
노드는 구조체 타입
Nptr 타입이 가리키는 것은 노드 타입

```
class queueClass
{ public:
    queueClass( );
    queueClass(const queueClass& Q); 복사 생성자 함수
    ~queueClass( ); 소멸자 함수
    void Add(int Item); Item 값을 큐에 삽입
    void Remove( ); 큐 프론트를 삭제, 리턴 값 없음
    boolean IsEmpty( ); 비어 있는지 확인
    boolean IsFull( ); 꽉 차 있는지 확인
```

생성자 함수
복사 생성자 함수
소멸자 함수
Item 값을 큐에 삽입
큐 프론트를 삭제, 리턴 값 없음
비어 있는지 확인
꽉 차 있는지 확인

```
private:
```

```
    Nptr Rear;
```

마지막 노드를 가리키는 포인터

```
}
```

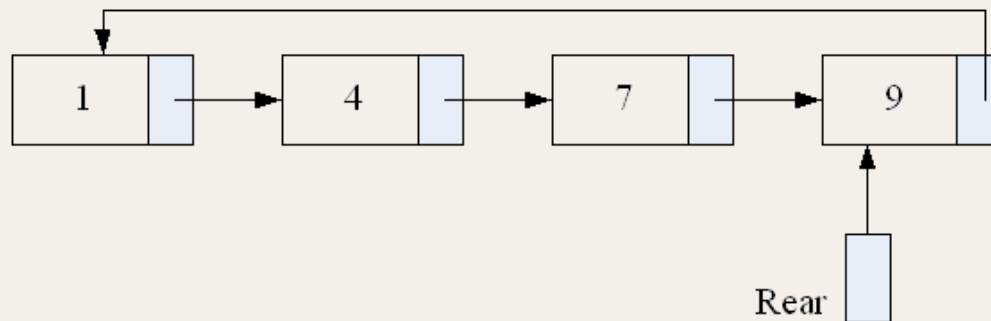

C++ 연결 리스트에 의한 큐

❑ 큐의 삽입 삭제

- 양쪽 끝에서 일어남(삽입은 리어에, 삭제는 프런트에서)
- 연결 리스트의 첫 노드를 프런트로, 마지막을 리어로 간주
- 스택은 삽입 삭제 모두 한쪽 끝에서 일어남

❑ 리어 포인터 하나로 구현한 큐

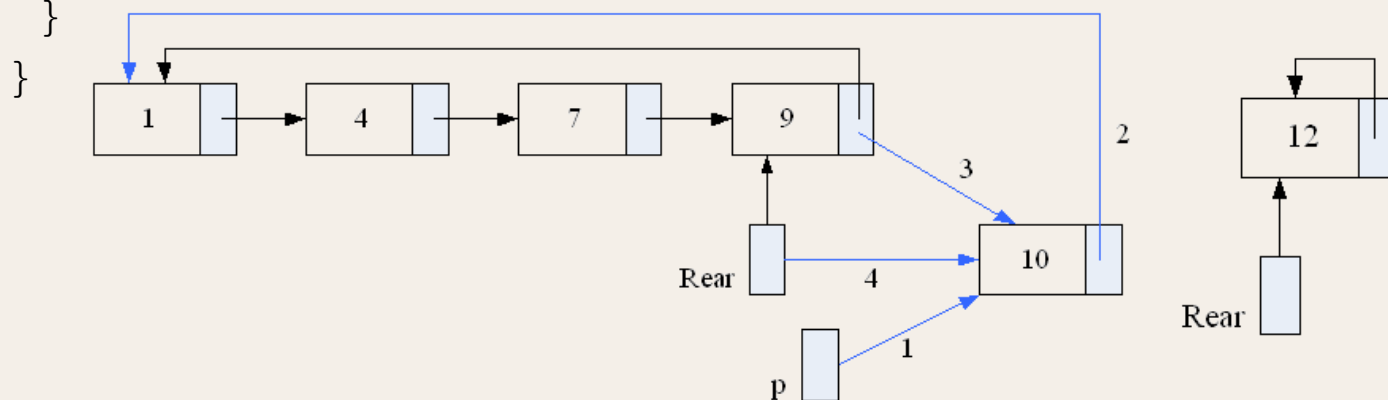
- 원형 연결 리스트
- 첫요소는 $\text{Rear} \rightarrow \text{Next}$ 에 의해 접근 가능
- 프런트 포인터 하나로만 구현하면 삽입을 위해 끝까지 순회해야 함.



원형 연결 리스트 큐의 삽입

```
void queueClass::Add(int Item)
```

```
{ if (! IsEmpty( ))                현재 하나 이상의 노드가 있다면
{  Nptr p = new node;              포인터 p가 공간의 새 노드의 시작주소를 가리키게
  p->Data = Item;                   새 노드의 데이터 필드를 채우고
  p->Next = Rear->Next;             이 노드가 결국 마지막 노드가 될 것이니
                                     이 노드가 현재의 프런트 노드를 가리키게
  Rear->Next = p;                   현재의 마지막 노드가 새 노드를 가리키게
  Rear = p;                         새 노드를 마지막 노드로
}
else                                현재 비어있는 큐 이라면
{  p = new node;                   포인터 p가 공간의 새 노드의 시작주소를 가리키게
  p->Data = Item;                   새 노드의 데이터 필드를 채우고
  p->Next = p;                     자기 자신을 가리키게
  Rear = p;                         새 노드를 마지막 노드로
}
```



원형 연결 리스트 큐의 삭제

```
void queueClass::Remove( )
```

```
{ Nptr Front = Rear->Next;
```

```
  if (GetSize( ) >= 2)
```

```
  { Rear->Next = Front->Next;
```

```
    delete Front;
```

```
  }
```

```
  else if (GetSize( ) == 1)
```

```
  { Rear = NULL;
```

```
    delete Front;
```

```
  }
```

```
  else if (GetSize( ) == 0)
```

```
  { cout << "Deletion on Empty Queue";
```

```
  }
```

```
}
```

편의상 프론트 포인터를 별도로 생성

노드 두 개 이상일 때

마지막 노드가 두 번째 노드를 가리키게

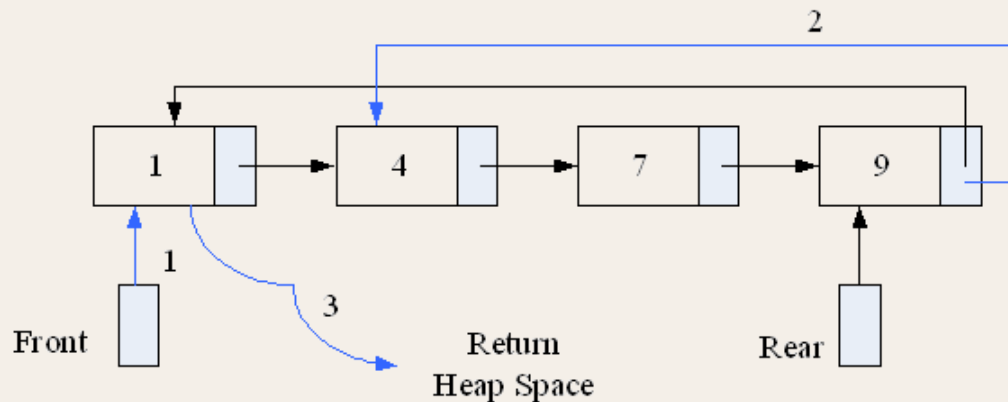
첫 노드가 사용하던 공간을 반납

노드 하나일 때

삭제하면 빈 큐가 되므로 빈 큐를 표시

첫 노드이자 마지막 노드의 공간 반납

빈 큐에 삭제명령은 오류로 처리



C++ 배열에 의한 큐

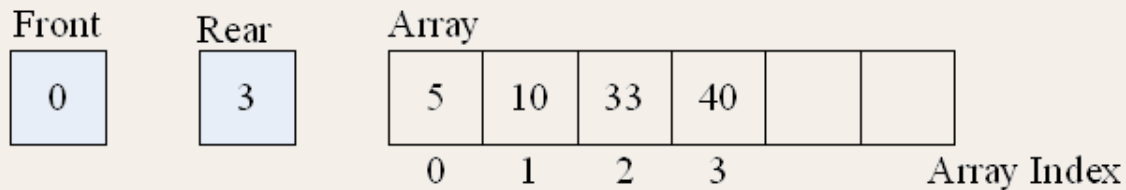
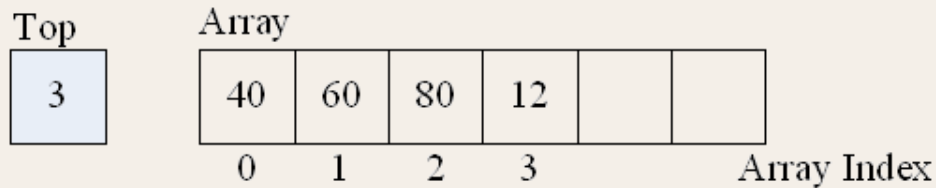
❑ 코드 7-4: QueueA.h (C++ Interface by Array)

```
const int MAX = 100;
class queueClass
{ public:
    queueClass( );           생성자 함수
    queueClass(const queueClass& Q); 복사 생성자 함수
    ~queueClass( );         소멸자 함수
    void Add(int Item);      Item 값을 큐에 삽입
    void Remove( );         큐 프론트를 삭제, 리턴 값 없음
    boolean IsEmpty( );     비어 있는지 확인
    boolean IsFull( );      꽉 차 있는지 확인
private:
    int Front, Rear;        프론트, 리어 인덱스를 추적
    int Count;              원형연결 배열에 사용
    int Queue[MAX];         큐 데이터는 정수형, 최대 100
                             개
}
```

배열에 의한 스택/큐 배열 비교

□ 스택, 큐

- 스택은 탑 변수
- 큐는 프런트, 리어 변수



배열에 의한 큐

□ 삽입, 삭제

- Front 0, Rear -1로 초기화
- 삽입이면 Rear++ 한 후에 삽입
- 삭제이면 Front++

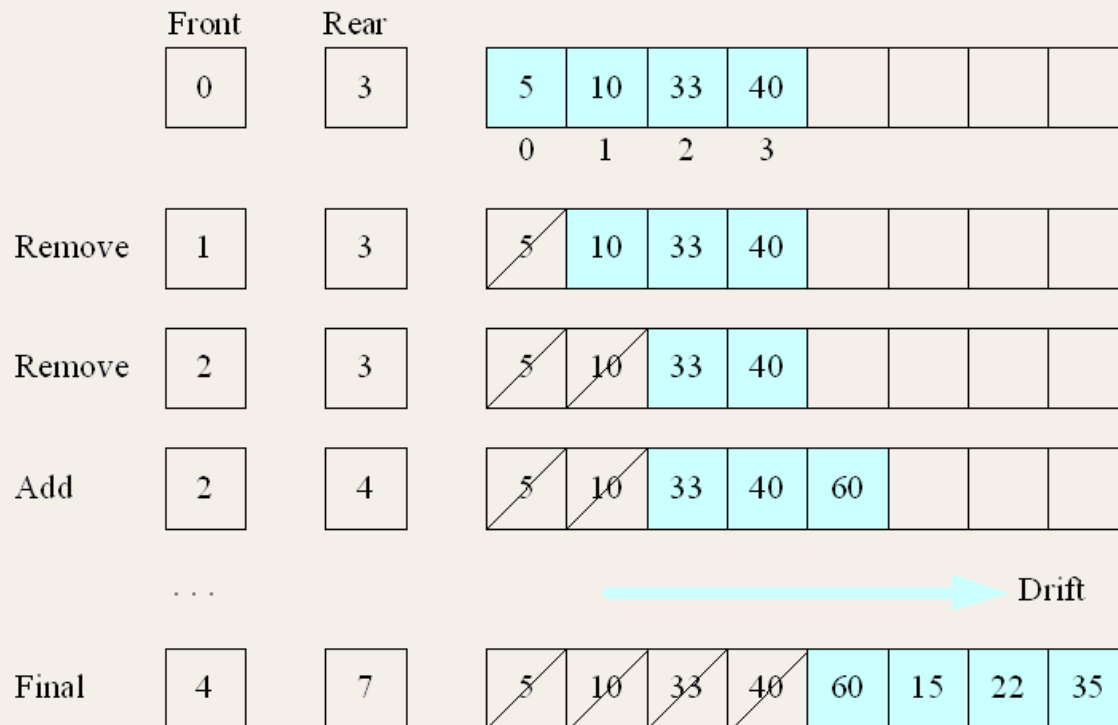
□ 큐의 상태 판단

- Front > Rear이면 빈 큐
- Front == Rear이면 큐 아이템 하나

배열에 의한 큐의 표류

□ 표류

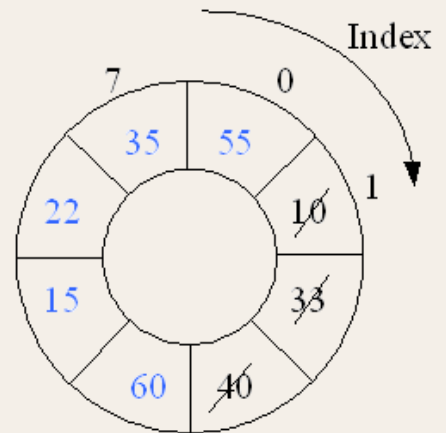
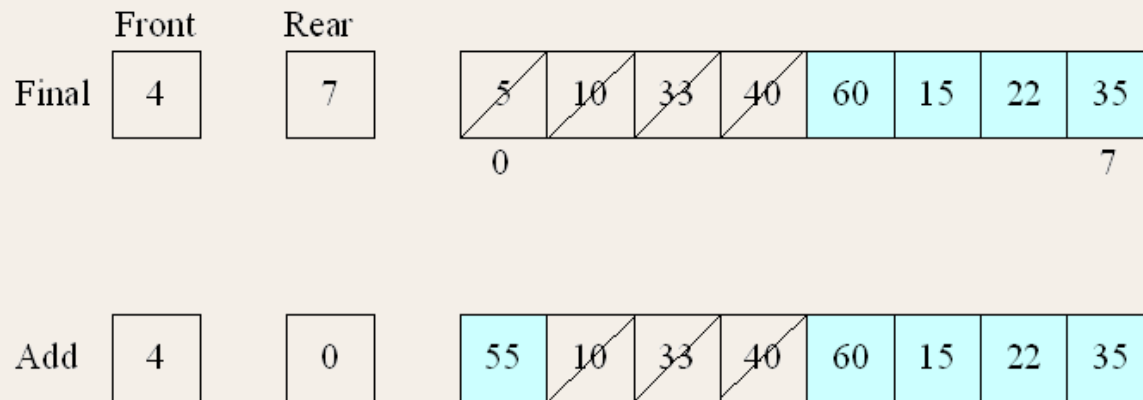
- 연속된 삽입, 삭제에 의한 오른쪽 이동
- 왼쪽 빈 공간이 있음에도 오른쪽에서 막힘
- 대책: 왼쪽 쉬프트
 - 삭제될 때마다
 - 오른쪽 끝에 막힐 때 몰아서 한번에



원형 배열

삽입

삽입 인덱스: $(\text{Rear} + 1) \% \text{MAX}$



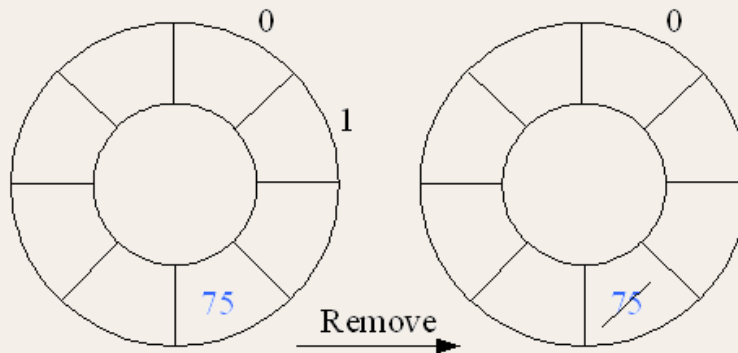
원형 배열

□ 빈 큐와 꽉찬 큐 판정불가

- 인덱스로 보서는 둘 다 동일 조건
- $\text{Front} = \text{Rear} + 1$

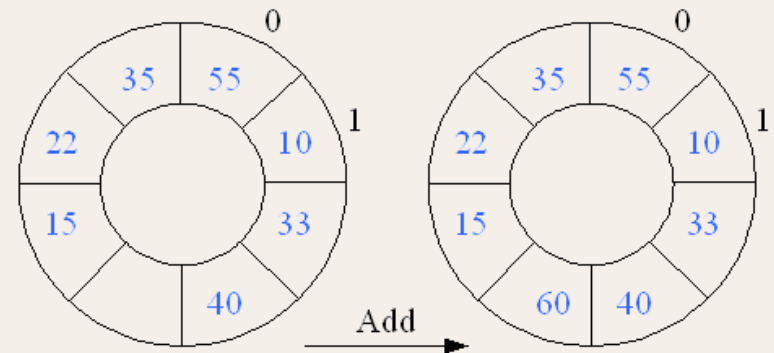
□ 별도의 Count 변수 유지

- 삽입시 Count ++
- 삭제시 Count --



Front = 3
Rear = 3

Front = 4
Rear = 3



Front = 5
Rear = 3

Front = 5
Rear = 4

원형 배열 큐 함수

- ❑ 코드 7-5: 원형 배열 큐의 초기화, 삽입, 삭제

`queueClass::queueClass()` 생성자 함수

```
{ Front = 0;
  Rear = -1;
  Count = 0;
}
```

데이터 개수 0

`void queueClass::Add(int Item)`

```
{ if (Count <= Max)
  { Rear = (Rear + 1) % MAX;
    Queue[Rear] = Item;
    Count++;
  }
}
```

아직 데이터 수가 최대가 아니라면
리어 인덱스 증가(원형으로)
큐 배열에 데이터 복사
데이터 수 증가

`void queueClass::Remove()`

```
{ if (Count > 0)
  { Front = (Front + 1) % MAX;
    Count--;
  }
}
```

데이터가 하나라도 있다면
프런트 인덱스 증가(원형으로)
데이터 수 감소

추상 자료형 리스트에 의한 큐

코드 7-6: QueueL.h (C++ Interface by ADT LIST)

```
#include <ListP.h>
```

```
class queueClass
```

```
{ public:
```

```
    코드 6-7(또는 코드 6-5)과 동일
```

```
    private:
```

```
        listClass L;
```

```
};
```

```
void queueClass::Add(int Item)
```

삽입 함수

```
{ L.Insert(L.Length( ) + 1, Item);
```

```
}
```

```
void queueClass::Remove( )
```

삭제 함수

```
{L.Delete(1);
```

```
}
```

```
int queueClass::GetFront(int& FrontItem) 검색 함수
```

```
{ L.Retrieve(1, FrontItem) ;
```

```
}
```

큐 응용 예

□ 회문

- 앞에서 읽으나 뒤에서 읽으나 동일한 단어, 문자열
- 토마토, 기러기

□ 코드 7-7: 회문 판정

```
Q.Create( ); S.Create( );           큐와 스택을 생성
for (i = 0; i < stringLength( ); i++) 문자열 끝까지
{  Read Character into C;           하나씩 읽어서 변수 C에 저장
   Q.Add(C); S.Push(C);             큐에 삽입, 스택에 삽입
}
Matched = TRUE:                     매칭된 것으로 초기화
while (!IsEmpty( ) && Matched) 비어 있지 않고, 미스매치 되지 않는 동안
{  if (Q.GetFront( ) == S.GetTop( )) 큐 프론트와 스택 탑이 일치하면
   { Q.Remove( ); S.Pop( );         각각 삭제
   }
   else Matched = FALSE;           일치하지 않으면 빠져나감
}
return MatchedSoFar;               비어서 빠져나오면 트루를 리턴
}
```

큐 응용 예

□ 시뮬레이션

- 모의실험, 큐잉 이론
- 이벤트 발생시기: 시간 구동 시뮬레이션, 사건구동 시뮬레이션

□ 대기시간

- (A, 20, 5) (B, 22, 4) (C, 23, 2) (D, 30, 3)의 순서로 일이 발생

Event	CustomerQueue				Arrival	Start	End
20	A				A: 20	20	25
22	B						
23	B	C					
25	C				B: 22	25	29
29					C: 23	29	31
30	D						
31					D: 30	31	34
34							

□ 메저와 트리거

□ 그래픽 입력모드

● 리퀘스트 모드

- 프로그램이 요구하는 입력장비로부터 입력
- 프로그램 실행 중에 메저값을 요구

● 샘플 모드

- 프로그램이 요구하는 입력장비로부터 입력
- 현재의 메저값을 가져다 실행

● 이벤트 모드

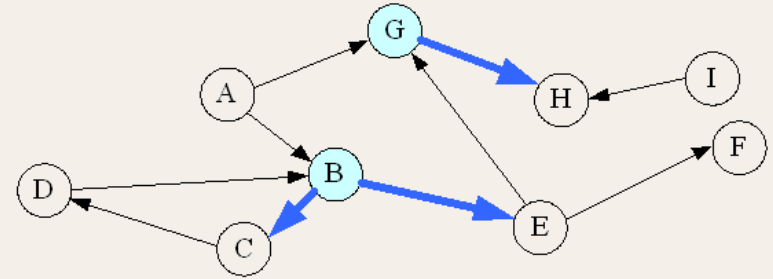
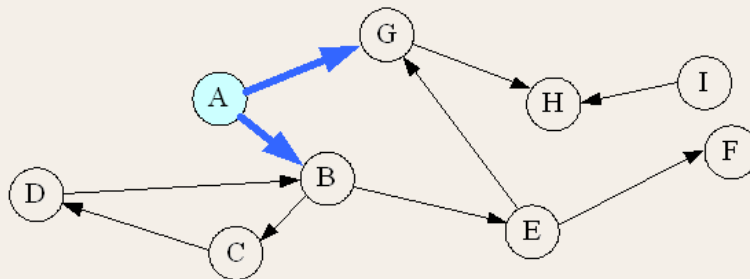
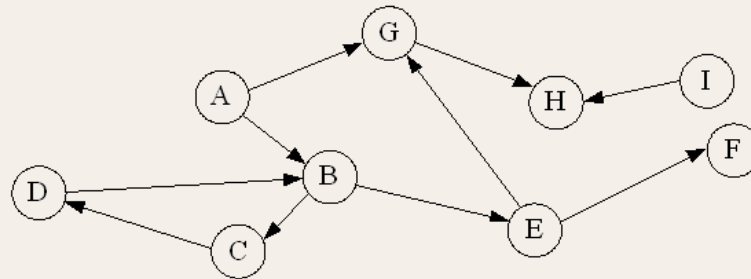
- 사용자가 선택한 입력장비가 우선권을 가짐
- 이벤트 큐와 콜백 함수

□ 시공유 시스템의 일괄처리 작업

- 사용자 ID 별로 우선순위를 분류.
- `queueType MultiQueue[MaxPriority];` 이면 우선순위 별로 `MultiQueue[0]`, `MultiQueue[1]`, `MultiQueue[2]`를 형성
- 배치 잡이 제출되면, 운영체제 중 디스패처 (Dispatcher)는 사용자 ID 를 기준으로 해당 큐에 삽입
- 먼저 실행 중이던 잡이 끝나면, 운영체제 중 스케줄러(Job Scheduler)는 사용자 ID 를 기준으로 해당 큐에서 삭제

❑ 너비우선 탐색(BFS: Breadth First Search)

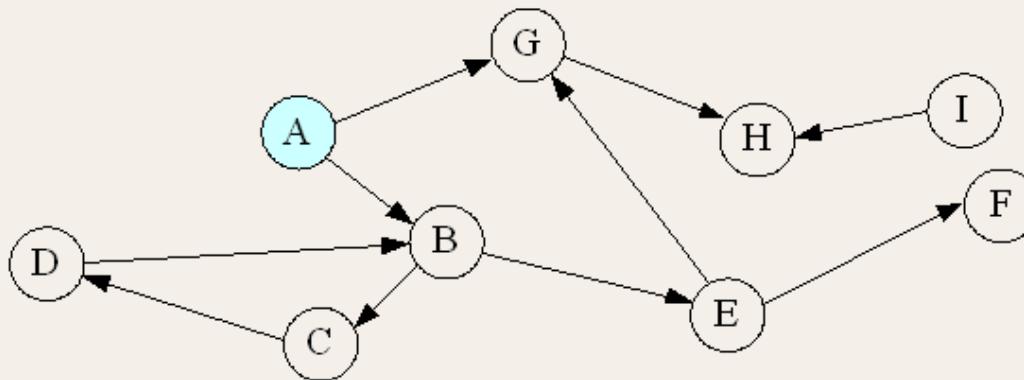
- 깊이보다는 폭을 취함
- A-B-G-C-E-H-D-F
- A에서 거리 1인 노드, 다시 각각으로부터 거리 1인 노드,



큐 응용 예

❑ 너비우선 탐색을 위한 큐

- 시작 노드를 삽입
- 삭제와 동시에 인접 노드를 삽입
- 소모적 탐색
- 한번 간 노드는 다시 안 감



A

B G

G C E

C E H

E H D

H D F

D F

F

너비우선 탐색

❑ 코드 7-8: 너비우선 탐색

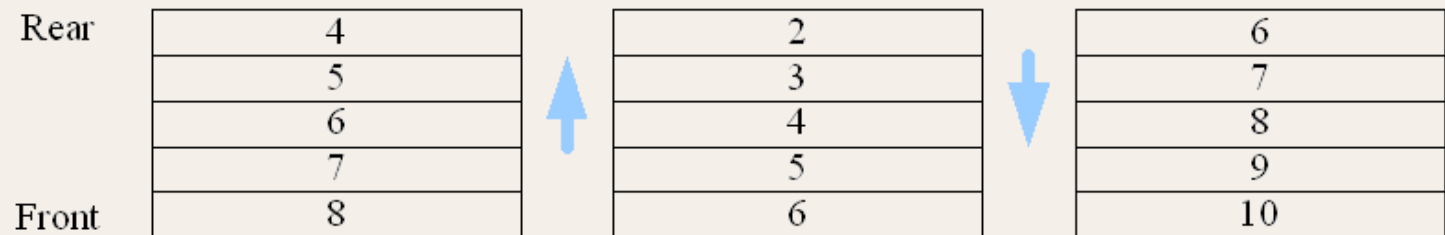
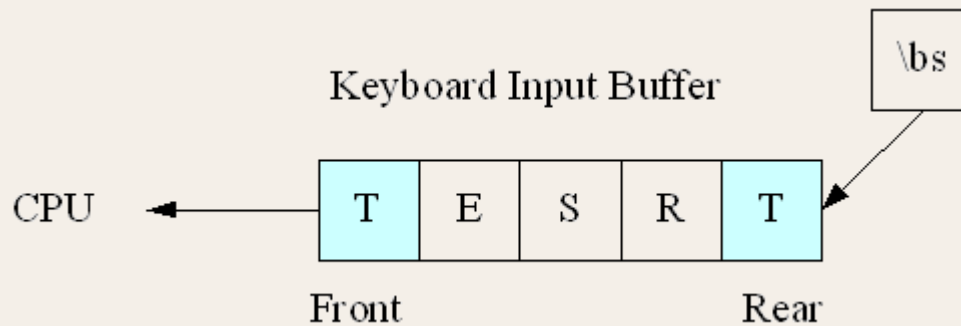
BreadthFirstSearch(Origin)

```
{  Q.Create( );           새로운 큐를 만들기
  Q.Add(Origin);          출발지를 큐에 삽입
  Mark Origin as Visited;  출발지를 가 본 것으로 표시
  while (!Q.IsEmpty( ))   빈 큐가 아닐 동안
  {    Q.GetFront(Front);  큐 프런트에 있는 노드를 Front로
    복사
      Q.Remove( );         큐 프런트를 제거
      for (Each Unvisited Nodes C Adjacent to Front)
      {    Q.Add(C);        큐에 삽입
          Mark C as Visited; 가 본 것으로 표시
        }
      }
  }
```

큐 응용 예

❑ 덱 (DEQUE: Double Ended Queue)

- 키보드 입력버퍼, 화면 스크롤
- 양쪽에서 삽입, 삭제의 필요성이 존재



❑ 덱(DEQUE: Double Ended Queue)

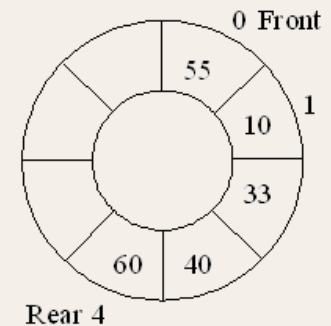
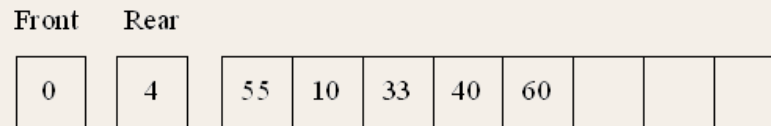
- AddLast(), AddFirst(), RemoveLast(), RemoveFirst()
- 큐: RemoveFirst(), AddLast() 만을 사용
- 스택: RemoveLast(), AddLast()만 사용

- 덱이 일반 클래스(General Class)라면
- 큐나 스택은 특수 클래스(Special Class, Adaptor Class)

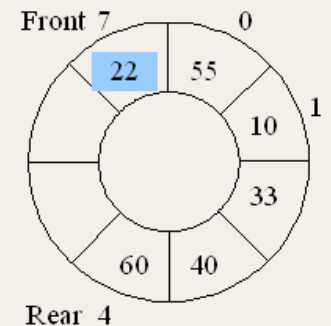
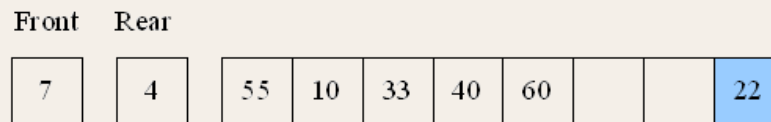
큐 응용 예

□ 원형배열 덱

- 양쪽에서 삽입 삭제: 두 개의 포인터를 유지하는 것이 유리
- 고정된 한 쪽 끝에서 Add나 Remove가 일어나면 쉬프트 필요
- 프론트 삭제 시에 $\text{Front}++$;에 의해 프론트가 전진
- 프론트 삽입 시에 $\text{Front}--$;에 의해 프론트가 후진



AddFront (22)



큐 응용 예

□ Josephus의 문제

- 유대인 41명이 로마군에 쫓겨 동굴에 갇혔다. 잡혀 죽기를 원치 않던 이 사람들은 [그림 7-15]와 같이 동글게 둘러앉아 아무도 남은 사람이 없을 때까지 다음 세 번째 사람(Every 3rd Person)을 죽이기로 했다. Josephus 는 이러한 죽음을 원치 않았다. 생존자 두 사람 중 하나가 되기 위해서는 몇 번째에 서야 하는가.

