Data Structures

Sungyoon Lee

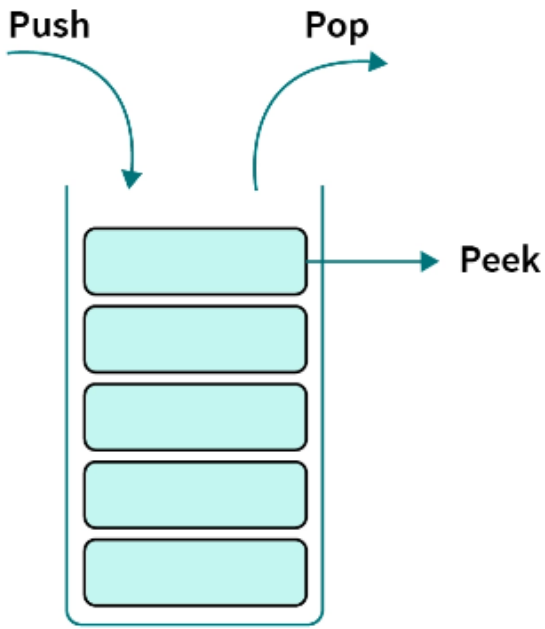
Department of Computer Science
Hanyang University

sungyoonlee@hanyang.ac.kr

Stack

Stack

- insertions and deletions can be performed **at one end** of the list
- insert $\sim$ push / delete $\sim$ pop
- Last In, First Out



e.g. ctrl+z, call stack, postfix evaluation (will be covered in TA class)

(function) call stack

```
1  void f2() {
2      return;
3  }
4
5  void f1() {
6      f2();  //calling f2()
7      return;
8  }
9
10 //This is main function
11 int main() {
12     f1();  // calling f1()
13 }
```

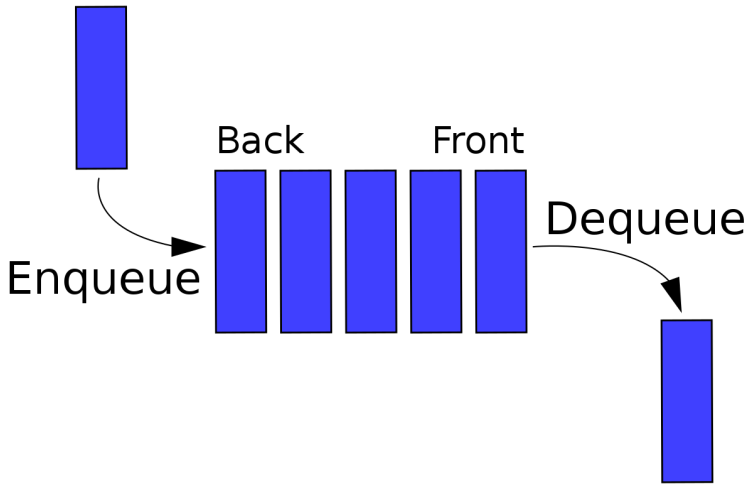
Q. What happens when we use a recursive function?

```
1 int factorial(int n) {
2     if (n == 1)
3         return n;
4     return n*factorial(n-1);
5 }
6
7 int main(){
8     return factorial(3);
9 }
```

Queue

Queue

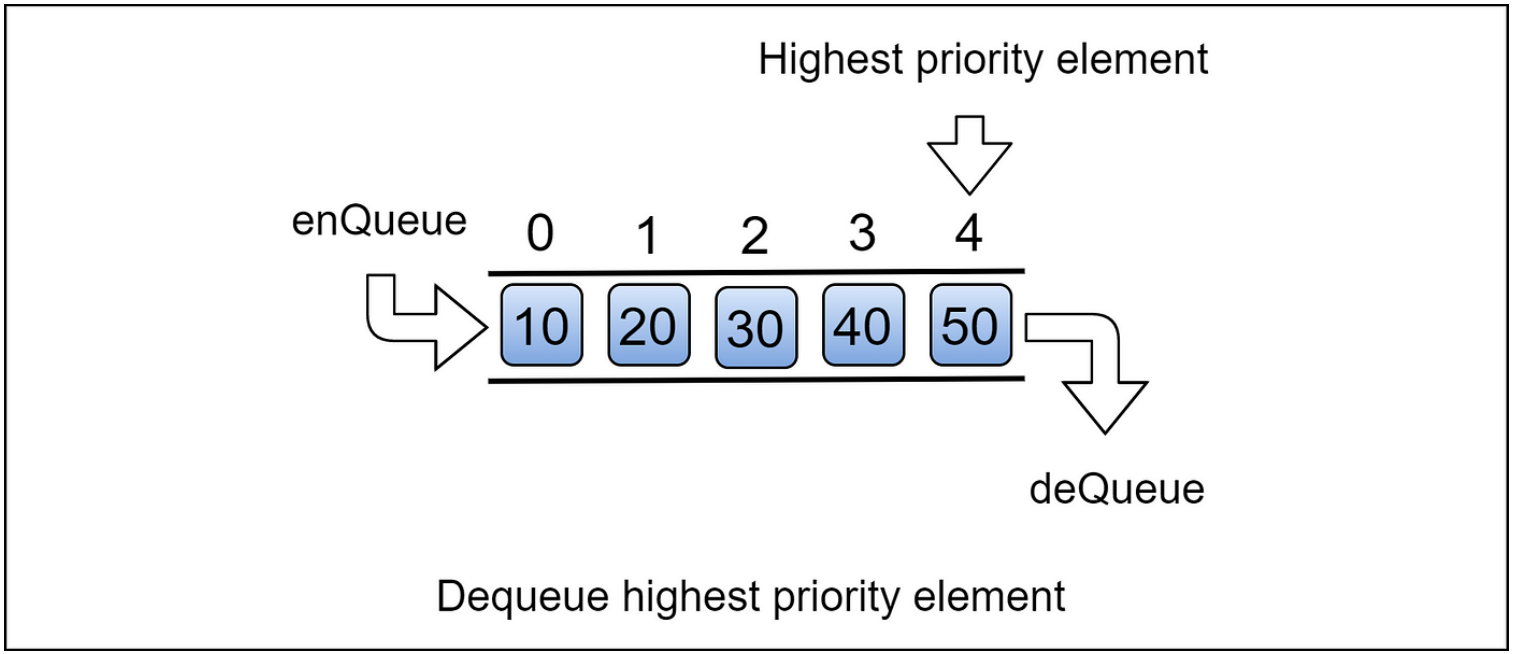
- queue (UK) = line (US)
- insertion is done at one end (rear/back), whereas deletion is performed at the other end (front)
- insert $\sim$ enqueue
- delete $\sim$ dequeue
- First In, First Out



e.g. job scheduling

Priority Queue

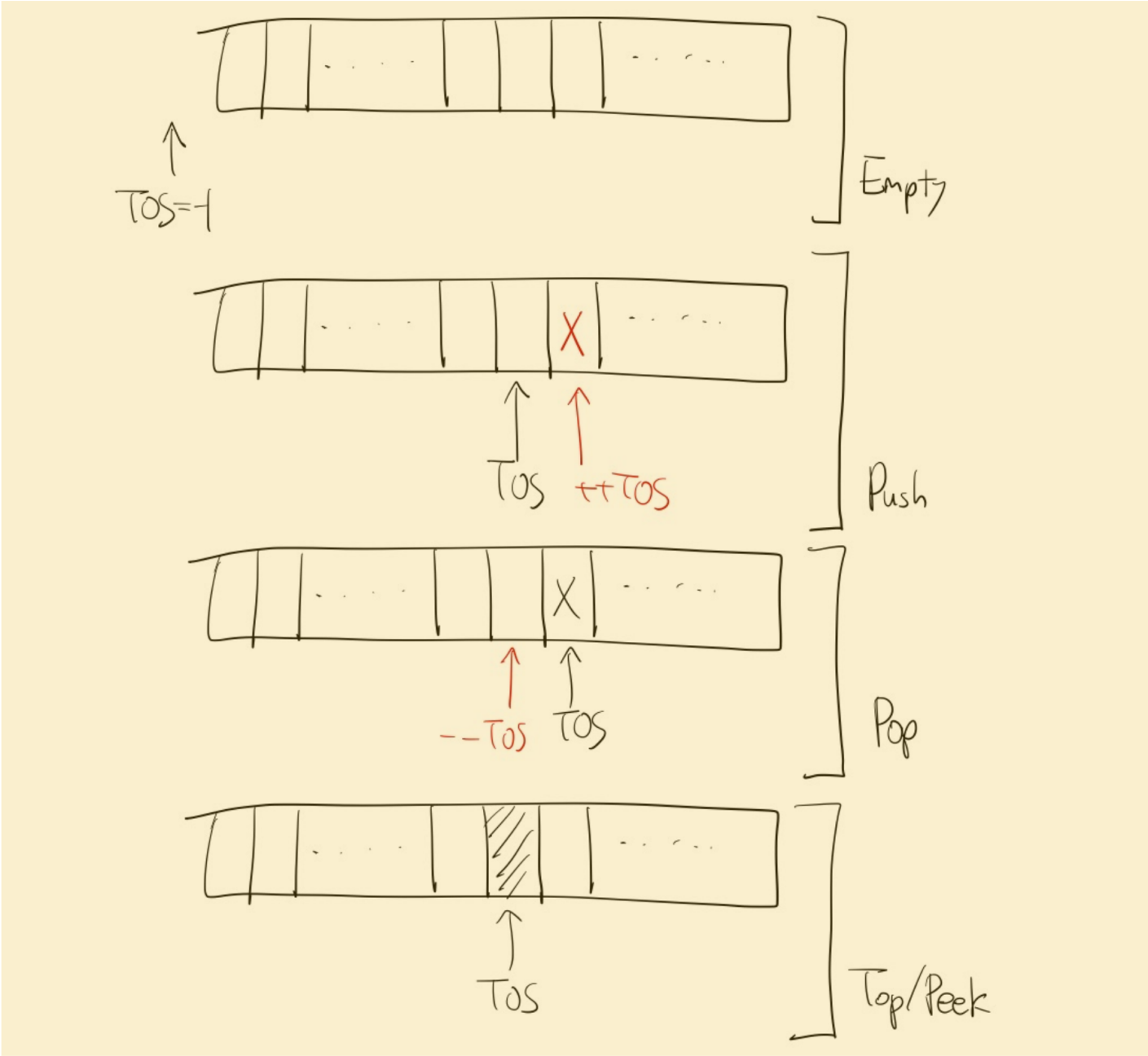
- ~Queue
- elements with high priority are served before elements with low priority
- Highest Priority In, First Out



e.g. to manage bandwidth on a transmission line from a network router

How can we implement Stack and Queue using Array and Linked List?

Stack and Queue - Implementation


TOS=-1
Empty
X
Tos
++TOS
Push
X
--Tos
TOS
Pop
Tos
Top/Peek

stack - array implementation

```
1 typedef struct StackRecord *Stack;
2
3 struct StackRecord
4 {
5     int Capacity;
6     int TopOfStack; // index of array
7     ElementType *Array;
8 };
```

stack - array implementation; CreateStack

```
#define EmptyTOS (-1)

Stack CreateStack( int MaxElements )
{
    Stack S;

    S = malloc(sizeof(struct StackRecord));
    if (S==NULL)
        FatalError("Out of Space!!!");

    S->Array = malloc(sizeof(ElementType)*MaxElements);
    if (S->Array == NULL)
        FatalError("Out of Space!!!");

    S->Capacity = MaxElements;
    S->TopOfStack = EmptyTOS;

    return S;
}
```

stack - array implementation; Push

```
1 void Push( ElementType X, Stack S )
2 {
3     if (IsFull(S))
4         Error("Full stack");
5     else
6         S->Array[++S->TopOfStack] = X;
7 }
```

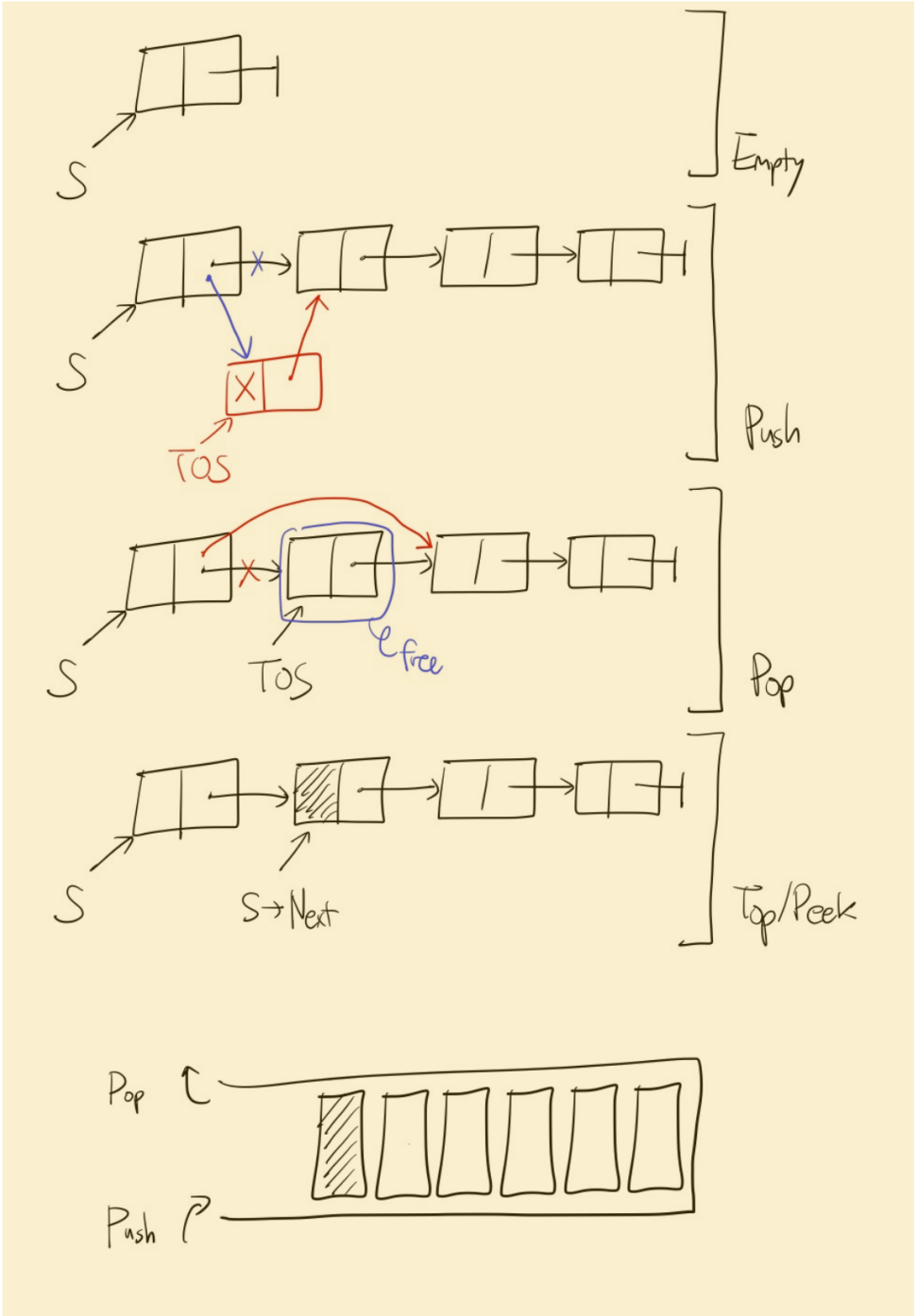
- 연산 우선 순서
- insertion/deletion과 다름 (time complexity는?).

stack - array implementation; Pop

```
1 void Pop( Stack S )
2 {
3     if (IsEmpty(S))
4         Error("Empty stack");
5     else
6         S->TopOfStack--;
7 }
```

stack - array implementation; Top/Peek

```
1 ElementType Top( Stack S )
2 {
3     if (!IsEmpty(S))
4         return S->Array[S->TopOfStack];
5
6     Error("Empty stack");
7     return 0;
8 }
```


S
Empty
S
X
X
TOS
Push
S
TOS
free
Pop
S
S→Next
Top/Peek
Pop
Push

stack - linked list implementation

```
1 struct Node;
2 typedef struct Node *PtrToNode;
3 typedef PtrToNode Stack;
4
5 struct Node{
6     ElementType Element;
7     PtrToNode Next;
8 };
```

stack - linked list implementation; CreateStack

```
1  Stack CreateStack(){
2      Stack S;
3      S = malloc(sizeof(struct Node));
4
5      if (S==NULL)
6          FatalError("Out of space!!!");
7
8      S->Next = NULL;
9      return S;
10 }
```

stack - linked list implementation; MakeEmpty

```
1 void MakeEmpty( Stack S ){
2     if (S==NULL)
3         Error("No stack exists");
4     else
5         while(!IsEmpty(S))
6             Pop(S);
7 }
```

stack - linked list implementation; Push

```
1  void Push( ElementType X, Stack S )
2  {
3      PtrToNode TmpCell;
4      TmpCell = malloc(sizeof(struct Node));
5
6      if (TmpCell == NULL){
7          FatalError("Out of space!!!");
8      } else{
9          TmpCell->Element = X;
10         TmpCell->Next = S->Next;
11         S->Next = TmpCell;
12     }
13 }
```

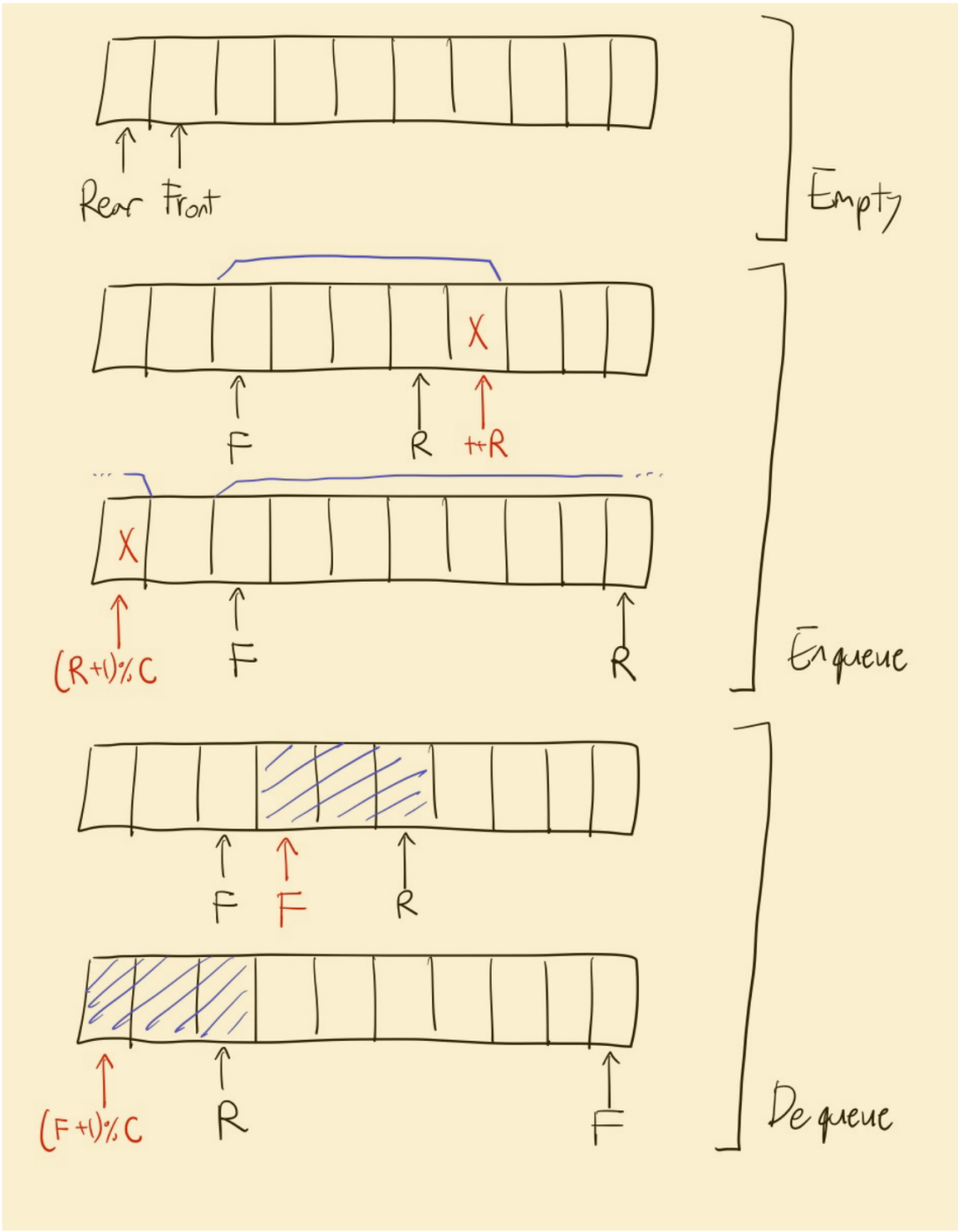
stack - linked list implementation; Pop

```
1  void Pop( Stack S )
2  {
3      PtrToNode FirstCell;
4
5      if (IsEmpty(S))
6          Error("Empty stack");
7      else{
8          FirstCell = S->Next;
9          S->Next = S->Next->Next;
10         free(FirstCell);
11     }
12 }
```

stack - linked list implementation; Top/Peek

```
1 ElementType Top( Stack S )
2 {
3     if (!IsEmpty(S))
4         return S->Next->Element;
5
6     Error("Empty stack");
7     return 0;
8 }
```

Head 쪽에 두어서 빠름


Rear
Front
Empty
X
F
R
++R
X
(R+1)%C
F
R
Enqueue
F
F
R
(F+1)%C
R
F
Dequeue

queue - array implementation

```
1  struct QueueRecord;
2  typedef struct QueueRecord *Queue;
3
4  struct QueueRecord{
5      int Capacity;
6      int Front;
7      int Rear;
8      int Size;
9      ElementType *Array;
10 };
```

queue - array implementation; MakeEmpty

```
1 void MakeEmpty( Queue Q ){
2     Q->Size = 0;
3     Q->Front = 1;
4     Q->Rear = 0;
5 };
```

queue - array implementation; Enqueue

```
1  void Enqueue( ElementType X, Queue Q ){
2      if (IsFull(Q))
3          Error("Full queue");
4      else{
5          Q->Size++;
6          // When front and rear gets to the end of the array,
7          // it is wrapped around to the beginning
8          Q->Rear = (Q->Rear+1)%Q->Capacity;
9          Q->Array[Q->Rear] = X;
10     }
11 };
```

queue - array implementation; Dequeue

```
1 void Dequeue( Queue Q ){
2     ?? // Q. Fill in the codes blanks.
3 };
```