Ch3: Lists, Stacks, and Queues

School of Electrical Engineering
Korea University

ADTs: Abstract Data Types

- 정의-형태(specification) 및 연산 (operations)은 정해져 있으나 구현방법은 지정하지 않는 자료구조

  예) list, stack, queue, array

- 허용 연산-생성,추가,제거,상태확인 등

- dynamic/ static ADTs

  규모를 지정(static), 변동(dynamic)에 따라

Operations

* Set – *union, intersection, size, complement*
* List – *insert, delete*
* Stack – *push, pop, empty/full*
* Queue – *enqueue, dequeue*
* Tree – *find, insert, delete, traverse*

Lists (Linear singly linked lists)

- Linear: one-dimensional structure
  - head (front)
  - tail (back)
- Unidirectional (pointer)
  - doubly linked lists
- Add, delete
- Find, count

** Multiple lists may exist at a time.

Stacks (pushdown stacks)

- Last-in First-out (LIFO)
  - TOS: top (pointer to the current object)
  - bottom (when empty)

- *Push* (insert), *Top* (read), *Pop*(delete)

- *IsEmpty, IsFull*

Queues

- First-in First-out (FIFO)
  예) 줄서기 (대기/순서 관리)
- enqueue, dequeue
  -추가(삽입): front, 삭제: back
  -job scheduling/ work assignment
- How to modify the Singly Linked List to implement the queues?

Lists (*linear* singly linked lists)

- Linear: one-dimensional structure
- Unidirectional vs Bidirectional
- Add, delete
- Find, count

** Multiple lists may exist at a time.

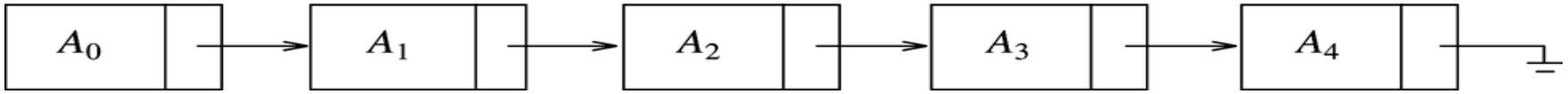
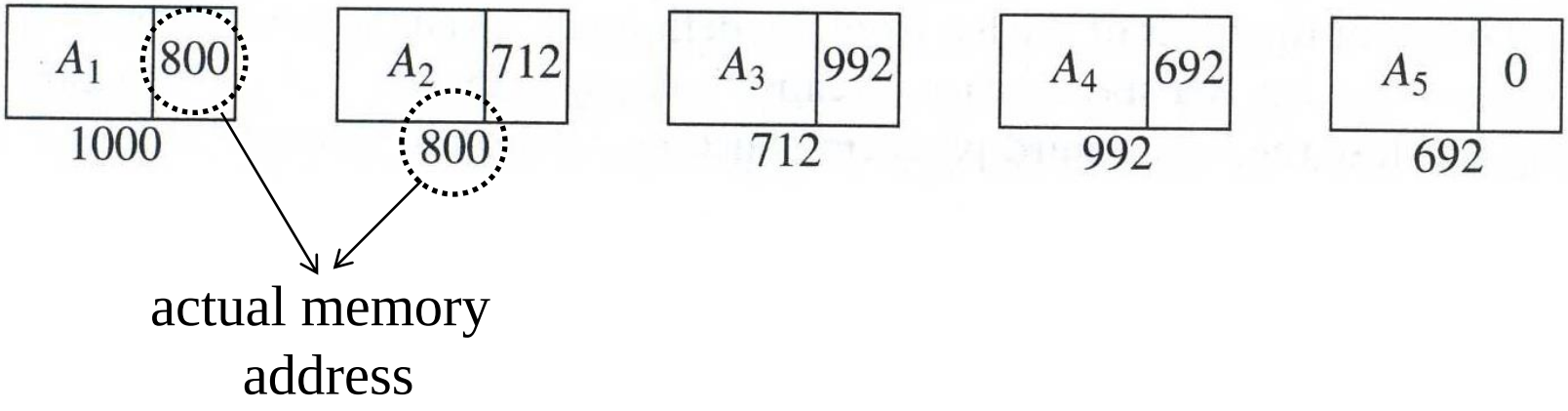


Figure 3.1 A linked list

Figure 3.2 Linked list with actual pointer values



Functions for Lists

- PrintList (L) – print the content
- Insert (X,L,P) – add X at P in the list L
- Delete (X,L) – delete X
- Find (X,L) – return the position of X
- FindPrevious(X,L) – return the position of predecessor of X
- IsEmpty(L), IsLast(L)
- MakeEmpty(L)

```
struct Node
{
  ElementType Element;
  Position Next;
 };

struct Node;
typedef struct Node *PtrToNode;
typedef PtrToNode List;
typedef PtrToNode Position;
```

구현 예제: (Figures 3.6-15)

- Header – sentinel/dummy node

  리스트 생성시 만들어지고, *MakeEmpty*를 실행하기 전까지는 존재하는 부분, 세부 내용은 없고 단지 맨 앞에서 대표역할을 함
- main content/objects (head? tail?)
- empty list – list with only the header–no head, no tail, no content

Find, Delete, Insert

- Carefully analyze the programs in Figures 3.11-13 in the textbook!
- Avoid the *common errors*
- *FindPrevious(X, L)* –looks confusing (if you consider the meaning of the function name.)
- *Insert(X, L, P)* – add after the object pointed by *P;* if the object is the header, the first data element is created.

```
struct Node;
typedef struct Node *PtrToNode;
typedef PtrToNode List;
typedef PtrToNode Position;

List MakeEmpty( List L );
int IsEmpty( List L );
int IsLast( Position P, List L );
Position Find( ElementType X, List L );
void Delete( ElementType X, List L );
Position FindPrevious( ElementType X, List L );
void Insert( ElementType X, List L, Position P );
void DeleteList( List L );
Position Header( List L );
Position First( List L );
Position Advance( Position P );
ElementType Retrieve( Position P );
#endif    /* _List_H */
```

```
/* Place in the
implementation file */
struct Node
{
    ElementType Element;
    Position Next;
};
```

Figure 3.6: Type definitions for linked lists

```
/* Return true if L is empty */

Int IsEmpty( List L )
{
    return L->Next == NULL;
}
```

Figure 3.8: Function to test if a link list is

```
/* Return true if P is the last position in list L */
/* Parameter L is unused in this implementation */

int IsLast( Position P, List L )
{
    return P->Next == NULL;
}
```

Figure 3.9: Function to test if current position is the last in a linked list

```
/* Return Position of X in L; NULL if not found */

Position Find( ElementType X, List L )
{
    Position P;

    P = L->Next;
    while( P != NULL && P->Element != X )
        P = P->Next;

    return P;
}
```

Figure 3.10: *Find* routine

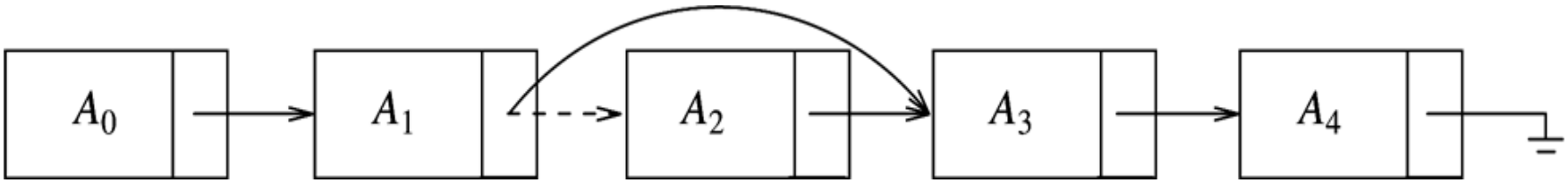


Figure 3.3 Deletion from a linked list

```
/* If X is not found, then Next field of returned */
/* Position is NULL */
/* Assumes a header */

Position FindPrevious( ElementType X, List L )
{
   Position P;

   P = L;
   while( P->Next != NULL && P->Next->Element != X )
       P = P->Next;

   return P;
}
```

Figure 3.12: *FindPrevious*

```
/* Delete first occurrence of X from a list */
/* Assume use of a header node */

Void Delete( ElementType X, List L )
{
    Position P, TmpCell;
    P = FindPrevious( X, L );

    if( !IsLast( P, L ) )
    {
        TmpCell = P->Next;
        P->Next = TmpCell->Next;
        free( TmpCell );
    }
}
```

Figure 3.11: Deletion routine for linked lists

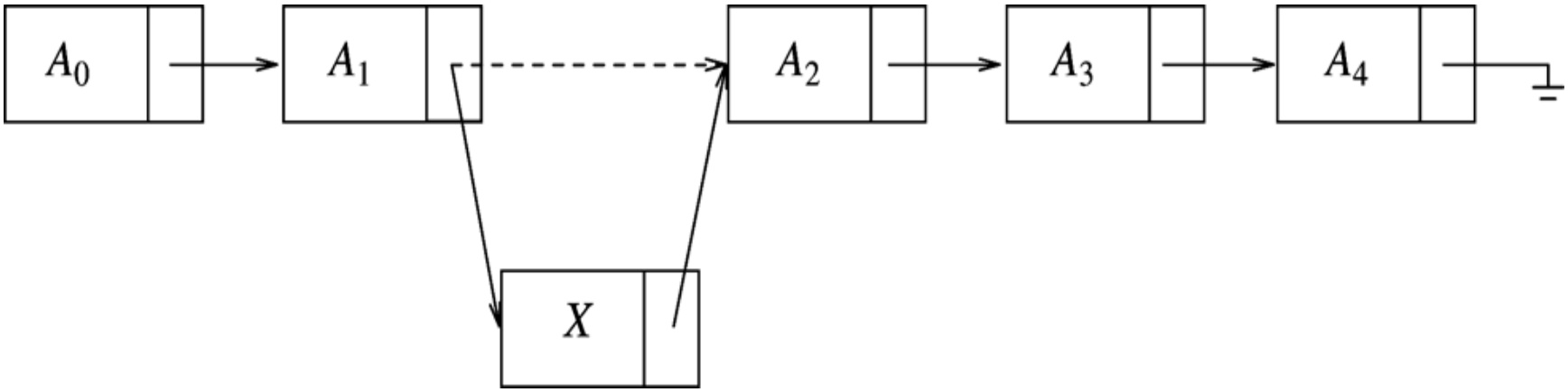


Figure 3.4 Insertion into a linked list

```
/* Insert (after legal position P) */
/* Header implementation assumed */
/* Parameter L is unused in this implementation */

Void Insert( ElementType X, List L, Position P )
{
    Position TmpCell;

    TmpCell = malloc( sizeof( struct Node ) );
    if( TmpCell == NULL )
        FatalError( "Out of space!!!" );

    TmpCell->Element = X;
    TmpCell->Next = P->Next;
    P->Next = TmpCell;
}
```

Figure 3.13: Insertion routines for linked list

```
/* Incorrect DeleteList algorithm */

Void DeleteList( List L )
{
    Position P;

    P = L->Next;
    L->Next = NULL;
    while( P != NULL )
    {
        free( P );
        P = P->Next;
    }
}
```

Figure 3.14: Incorrect way to delete a list

```
/* Correct DeleteList algorithm */

Void DeleteList( List L )
{
    Position P, Tmp;

    P = L->Next;
    L->Next = NULL;
    while( P!= NULL )
    {
        Tmp = P->Next;
        free( P );
        P = Tmp;
    }
}
```

Figure 3.15: Correct way to delete a list

Doubly linked lists

- Doubly linked lists: ptr_to_prev, ptr_to_next
  head & tail: wrap-around

```
struct NodeD;
typedef struct NodeD *PtrToNodeD;
typedef PtrToNodeD ListD;
typedef PtrToNodeD PositionD;

struct NodeD
{ ElementType Element;
  PositionD Next;
  PositionD Prev;
 };
```

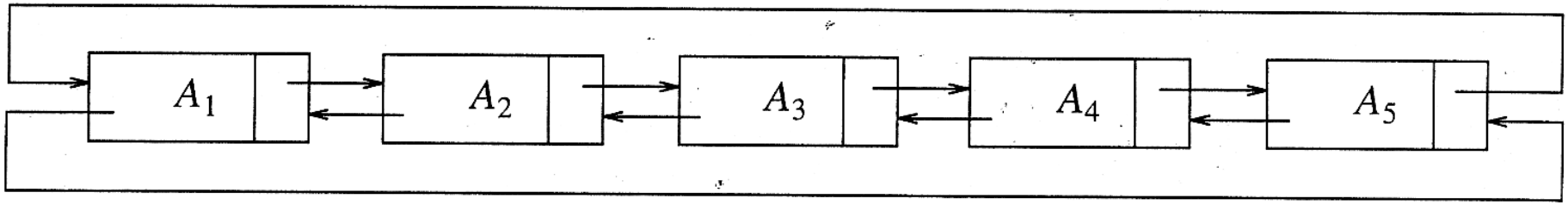


Figure 3.17 A double circularly linked list

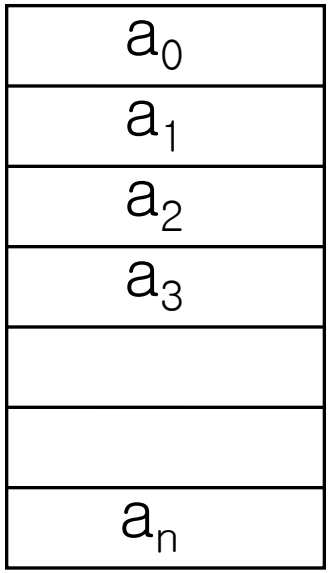
응용 예: Polynomial ADT, Multilists

- 대부분의 계수가 0인 다항식의 저장 및 다항식 간의 계산(합,곱)

- M개의 과목과 N명의 학생의 수강과목 정리, 저장(단 k<<M, k: 학생1인당 평균 수강 과목수)

Application of List

- Array Implementation of Polynomial ADT
  - $a_0 x^0 + a_1 x^1 + a_2 x^2 + a_3 x^3 + \cdots + a_n x^n$



+ : O(min(m,n))
* : O(m x n)

예제

$P_1(X) = 10X^{1000} + 5X^{14} + 1$

$P_2(X) = 3X^{1990} - 2X^{1492} + 11X + 5$

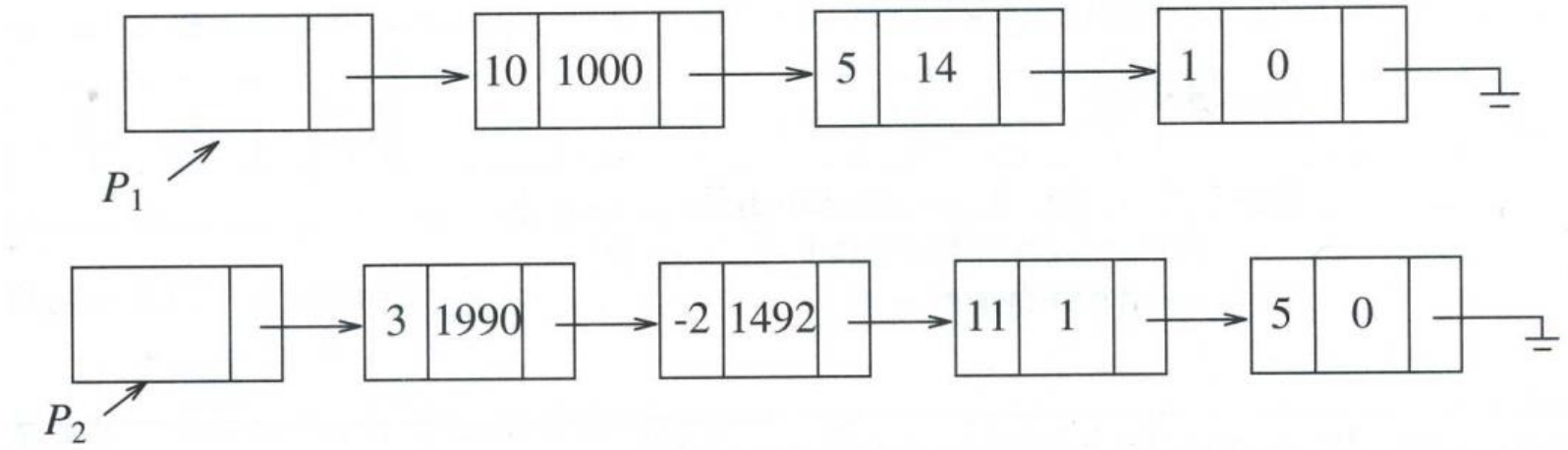


Figure 3.22 Linked list representations of two polynomials

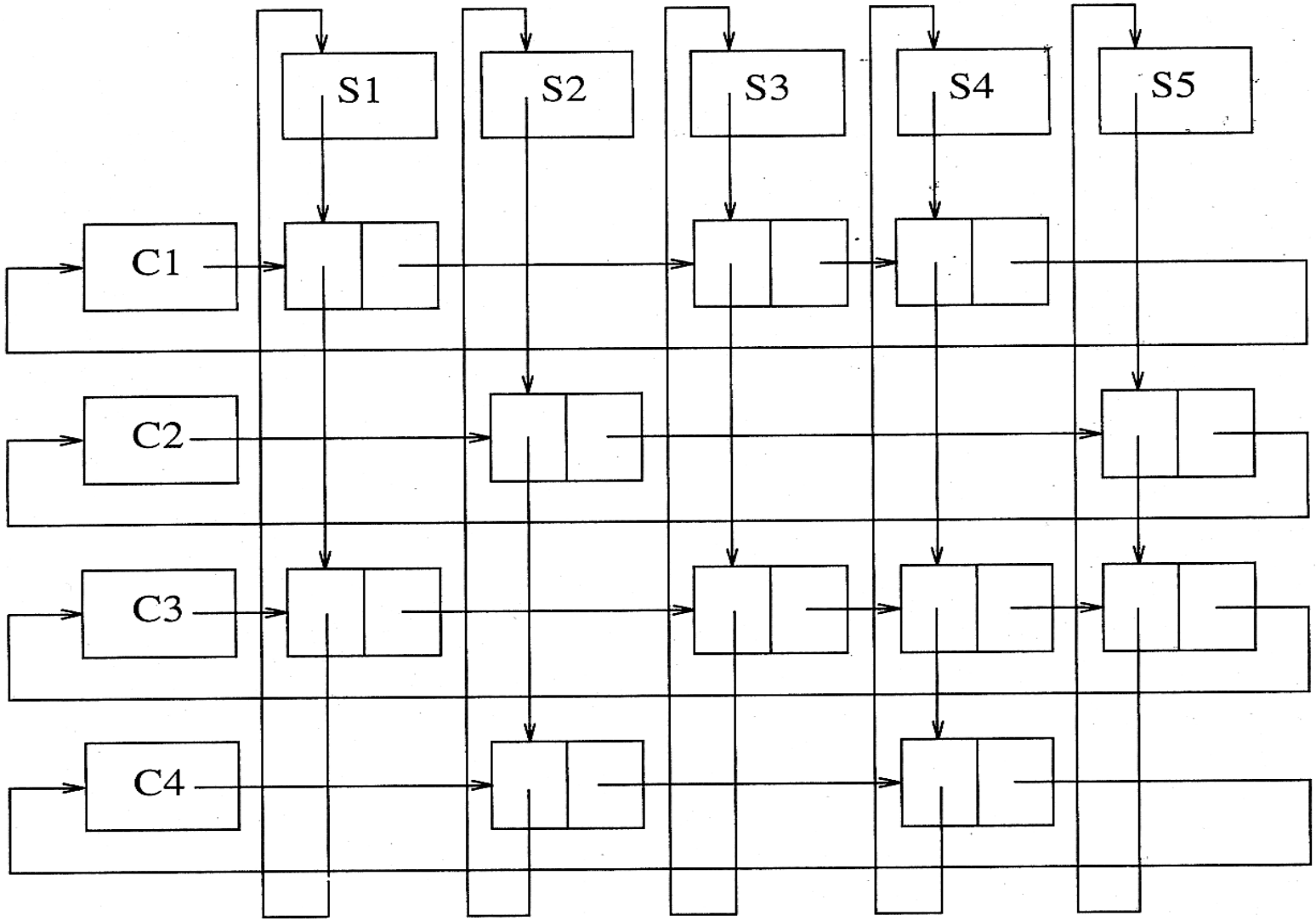


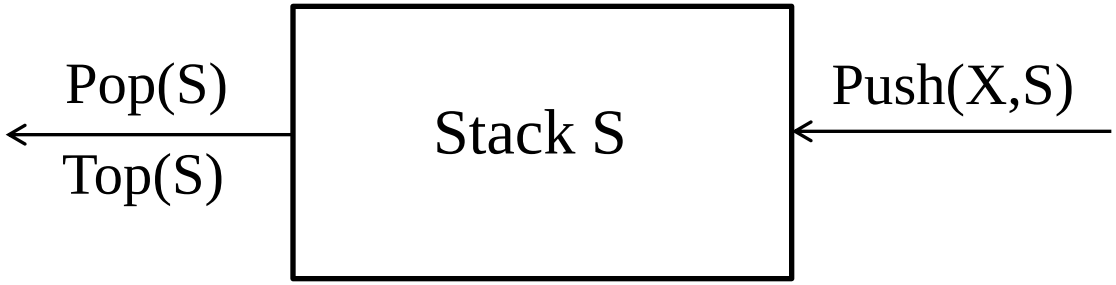
Figure 3.27 Multilist implementation for registration problem

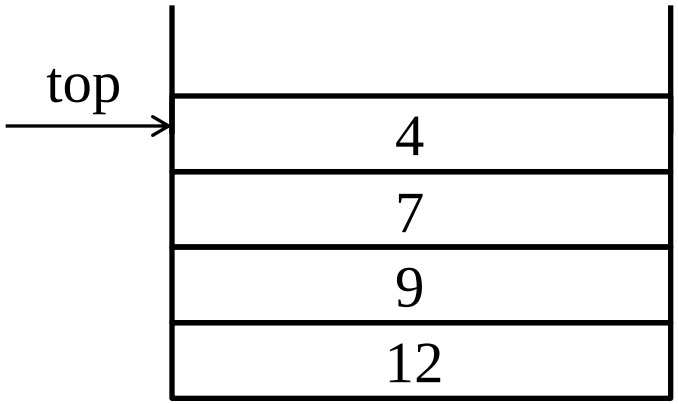
Stacks(pushdown stacks)

- Last-in First-out (LIFO)
  - TOS: top (pointer to the current object)
  - bottom (when empty)

- *Push* (insert), *Top* (read), *Pop*(delete)
- *IsEmpty, IsFull*

Stack Model





Stack 활용

- balancing symbols – 괄호

  (Ex) ((([{ }])))

  { ( ) { [( ) { }] } }

- infix to postfix(prefix) conversion

- function calls – return address 제어

Implementation (by Weiss)

- Linked list를 활용한 stack의 구현
  (how about array?)
- Header – sentinel/dummy node linked list 구현에서와 동일함
- *Push(X,S)* – insert X before the head
- *Pop(S)* – delete the head, then link the next object as a new head.

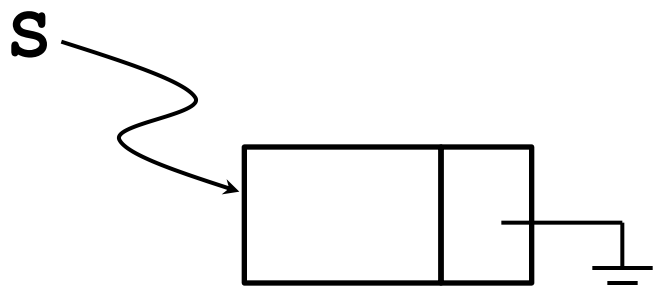
```
struct Node;
typedef struct Node *PtrToNode;
typedef PtrToNode Stack;
int IsEmpty( Stack S );
Stack CreateStack( void );
void DisposeStack( Stack S );
void MakeEmpty( Stack S );
void Push( ElementType X, Stack S );
ElementType Top( Stack S );
void Pop( Stack S );
```

```
struct Node
{
    ElementType Element;
    PtrToNode Next;
};
```

Figure 3.39: Type declaration for linked list implementation of the stack ADT

```
Int IsEmpty( Stack S )
{
    return S->Next == NULL;
}
```

Figure 3.40: Routine to test whether a stack is empty-linked list implementation



```
Stack CreateStack( void )
{
    Stack S;
    S = malloc( sizeof( struct Node ) );
    if( S == NULL )
        FatalError( "Out of space!!!" );
    MakeEmpty( S );
    return S;
}
```

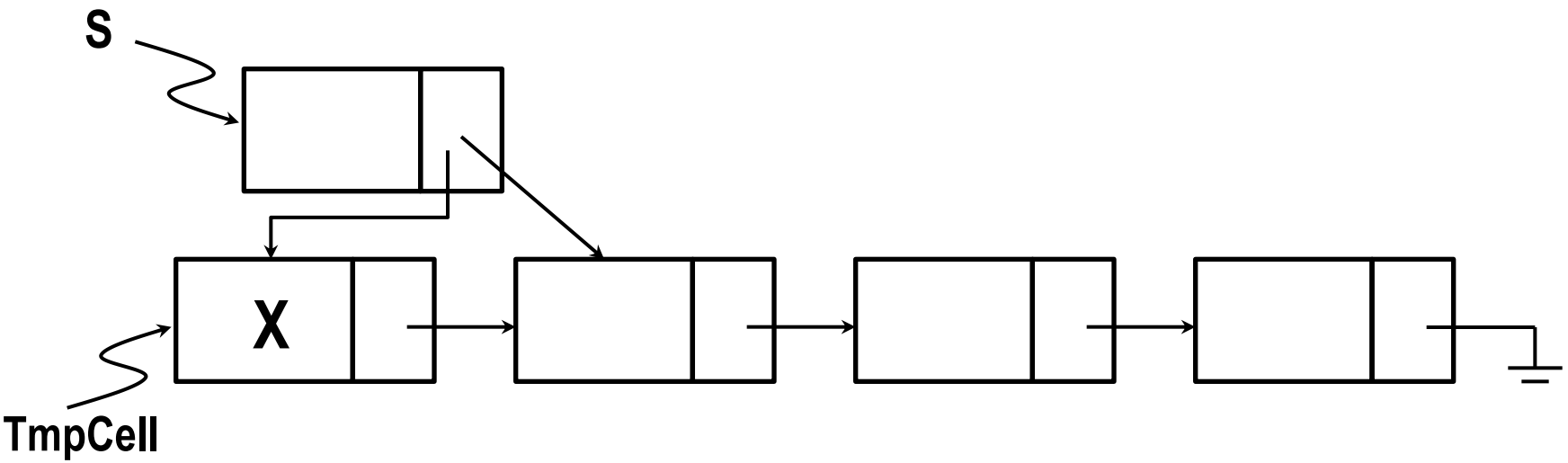
```
Void MakeEmpty( Stack S )
{
    if( S == NULL )
        Error( "Must use CreateStack first" );
    else
        while( !IsEmpty( S ) )
            Pop( S );
}
```

Figure 3.41: Routine to create an empty stack-linked list implementation

```
Void Push( ElementType X, Stack S )
{
    PtrToNode TmpCell;
    TmpCell = malloc( sizeof( sturct Node ) );
    if( TmpCell == NULL )
        FatalError( "Out of space!!!" );
    else
    {
        TmpCell->Element = X;
        TmpCell->Next = S->Next;
        S->Next = TmpCell;
    }
}
```

Figure 3.42: Routine to push onto a stack-lined list implementation

Push (X, S)



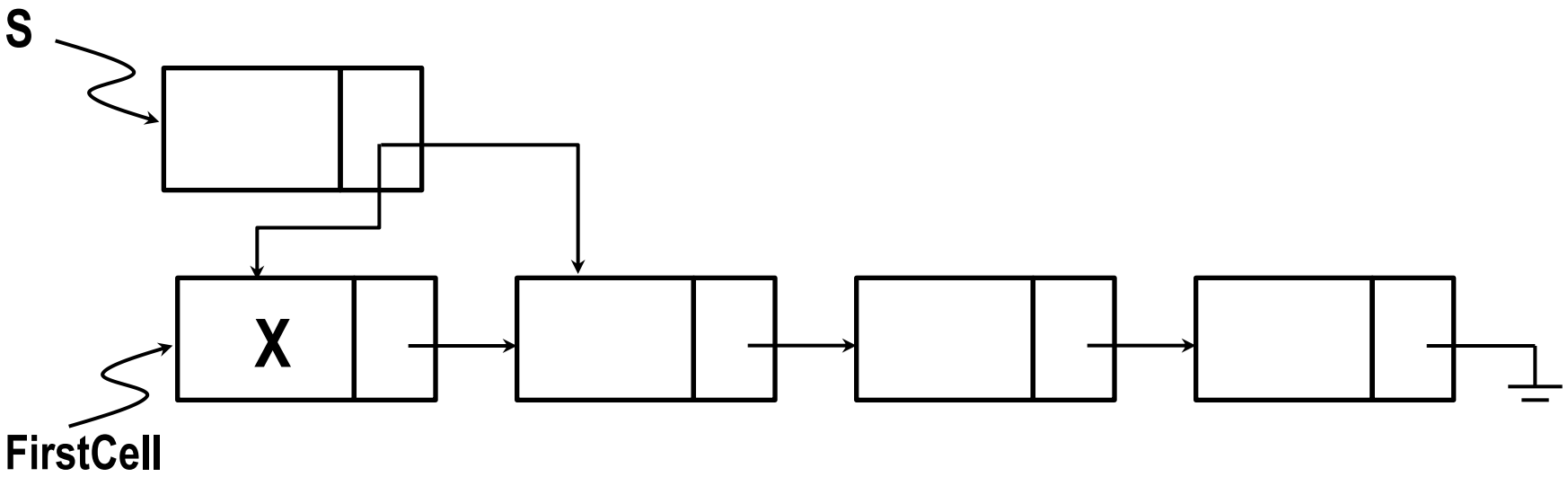
```
ElementType Top( Stack S )
{
    if( !IsEmpty( S ) )
        return S->Next->Element;
    Error( "Empty stack" );
    return 0;
        /* Return value used to avoid warning */
}
```

Figure 3.43: Routine to return top element in a stack-linked list implementation

```
Void Pop( Stack S )
{
    PtrToNode FirstCell;
    if( IsEmpty( S ) )
        Error( "Empty stack" );
    else
    {
        FirstCell = S->Next;
        S->Next = S->Next->Next;
        free( FirstCell);
    }
}
```

Figure 3.44: Routine to pop from a stack-linked list implementation

Pop (S)



응용 예제: 수식 Evaluation

- Prefix, postfix, infix notations
  (Ex) * 3 5, 3 5 *, 3 * 5

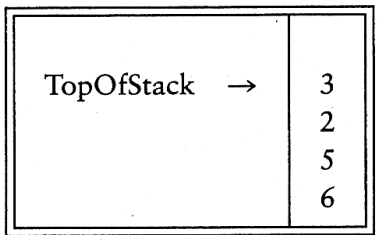
- Postfix: (left-to-right scanning with operations below)

```
if value: push(S, val)
if operator: x = pop(S);
             y = pop(S);
             z = x op y;
             push(S,z)
```

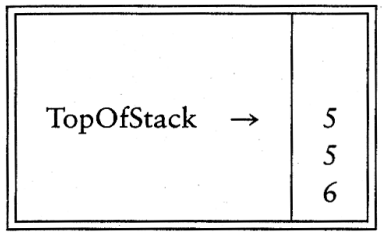
Example: Evaluating expressions

- Infix: operator precedence:
  ( ) > *, / > +, - > logical operators
- left-to-right scanning in compiler
- If no precedence needed- easy to manipulate in evaluation/execution -> postfix/prefix conversion
- Using operator stack

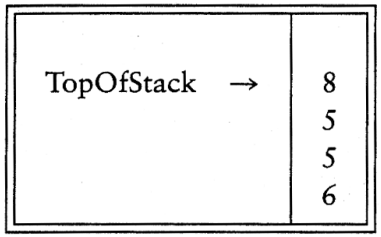
Postfix: 6 5 2 3 + 8 * + 3 + *



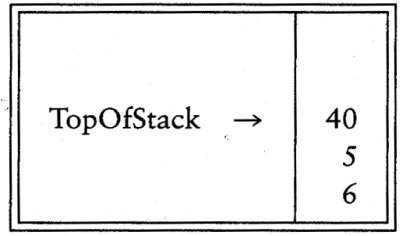
Next a '+' is read, so 3 and 2 are popped from the stack and their sum, 5, is pushed.



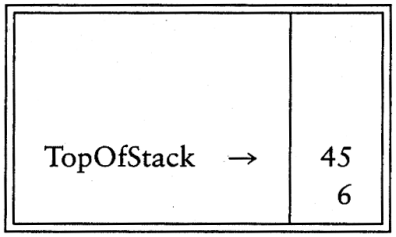
Next 8 is pushed.



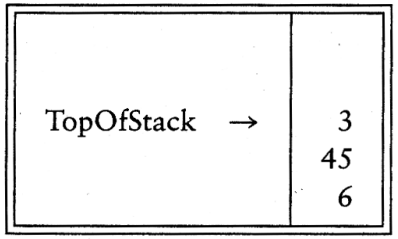
Now a '*' is seen, so 8 and 5 are popped and 5 * 8 = 40 is pushed.



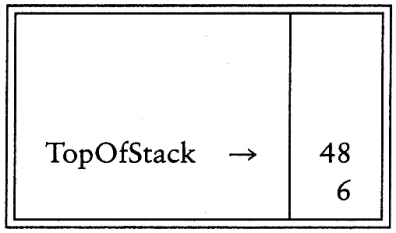
Next a '+' is seen, so 40 and 5 are popped and 5 + 40 = 45 is pushed.



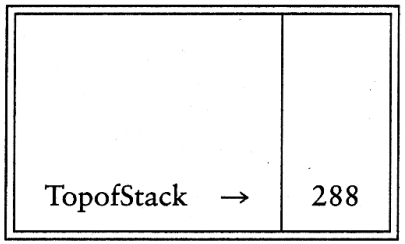
Now, 3 is pushed.



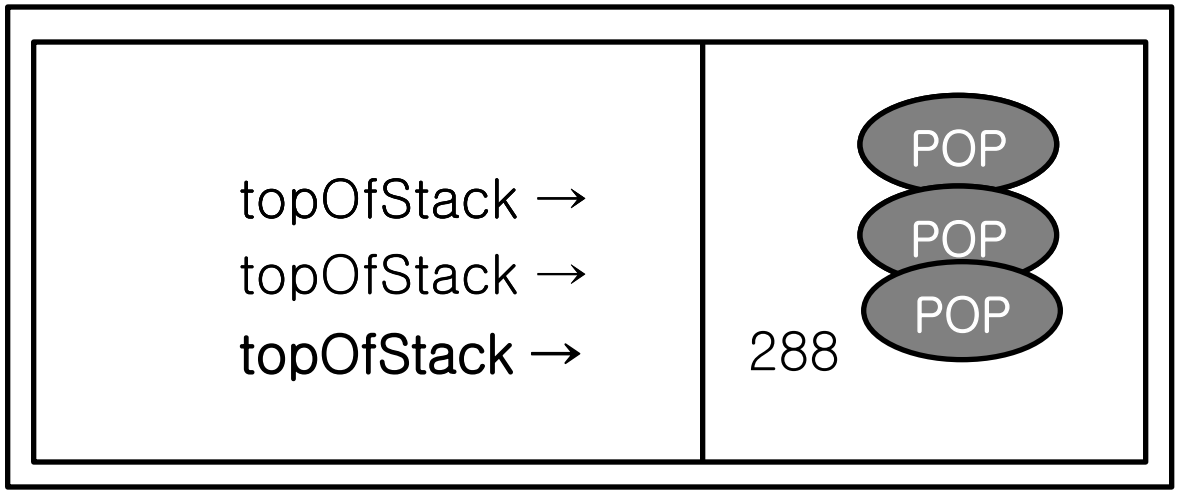
Next '+' pops 3 and 45 and pushes 45 + 3 = 48.



Finally, a '*' is seen and 48 and 6 are popped; the result, 6 * 48 = 288, is pushed.



Postfix Expression



6 5 2 3 + 8 * + 3 + *

Read, '*'
Pop, Pop two integers 48 6
Mul, 6 * 48 = 288
Push, 288

Infix to postfix conversion

BG:expression의 왼쪽에서부터 하나씩 읽어들여 다음을 수행.

(현재 읽은 값은 **op**로 표시함. 만약 수식의 끝에 도달하였으면 **S**가 empty 될 때까지 **Pop**을 반복하고 종료.)

if (op $\notin$ {+,-,*,/, '(', ')'}) then output it.

else /* operator */ .

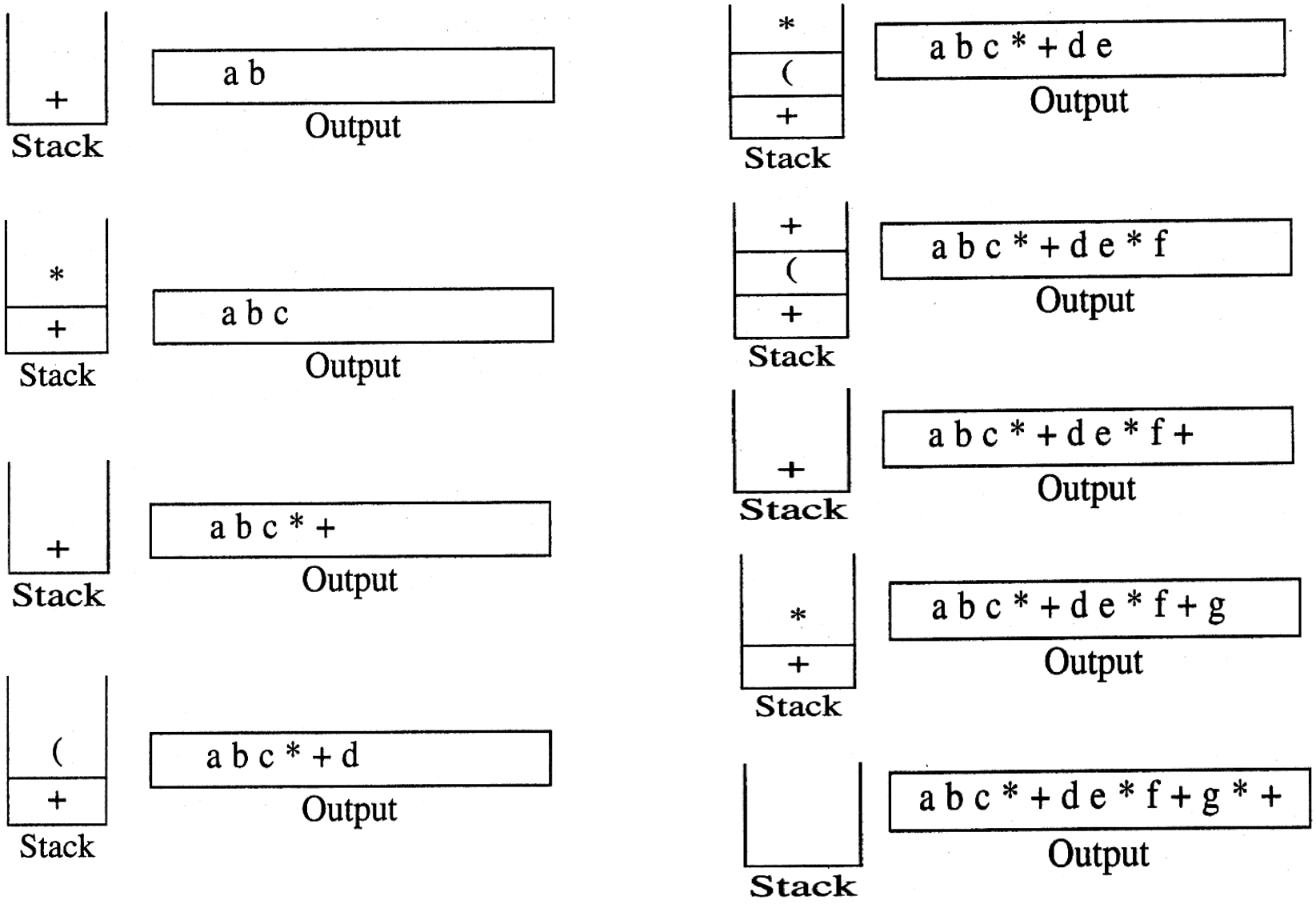
if ((Top(S) = '(') or Top(S) < op) Push(S,op), goto BG.

else if (Top(S) >= op) Pop(S) & output repeatedly
until (Top(S) < op);
Push (S,op), goto BG.

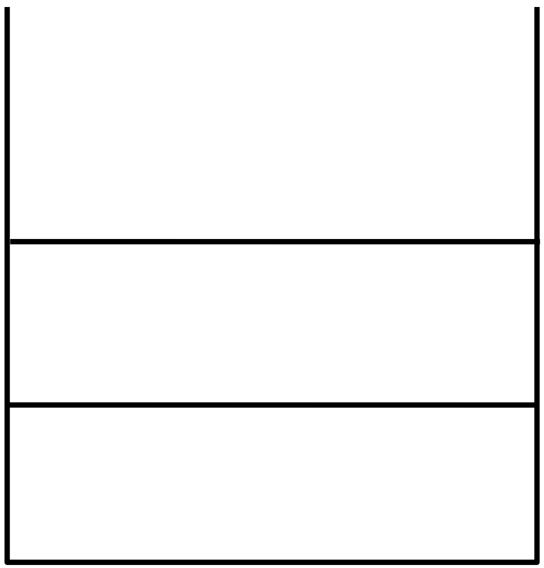
else if (op='(') Push(op), goto BG.

else if (op=')') Pop(S) and output it until ')' is popped,
goto BG.

Infix: a + b * c + (d * e + f) * g



Infix to Postfix Conversion



Input is empty,

Read a,

Read), stack is emptied until)

Read +, pushed on stack

Read b

a b c * + d e * f + g * +

Stack

Output

Convert the infix expression :

a + b * c + (d * e + f) * g

Correct answer :

a b c * + d e * f + g * +

Quiz - Exercise

- Show how to convert the following infix expr into postfix expr

(a + b) ^ c – d + (e – f * g) ^ h + i

- Show how to evaluate the following expr using a stack.

2 6 3 + * 2 1 1 + 2 ^ * 3 * +

Quiz # 1 (9/25)

- Show how to convert the following infix expr into postfix expr

  (a + b * c) ^ d – e * (f – g * b) ^ h * i

- Show how to evaluate the following expr using a stack.

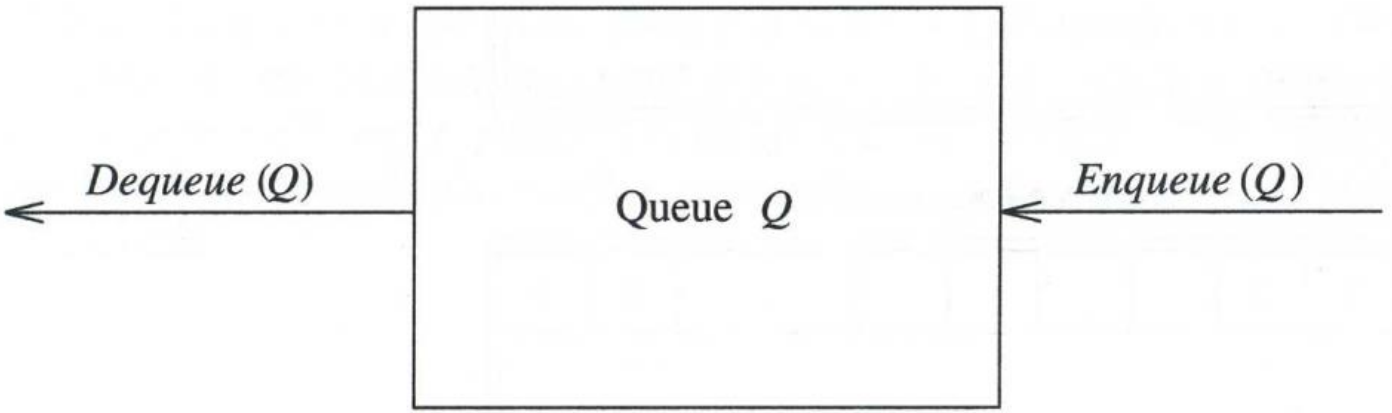
  2 3 + 7 * 4 8 3 2 * – 3 ^ * –

Queues

- First-in First-out (FIFO)
  예) 줄서기 (대기/순서 관리)
- enqueue, dequeue
  추가(삽입)-front, 삭제-back
- Job scheduling/ work assignment
- How to modify the Singly Linked List to implement the queues?

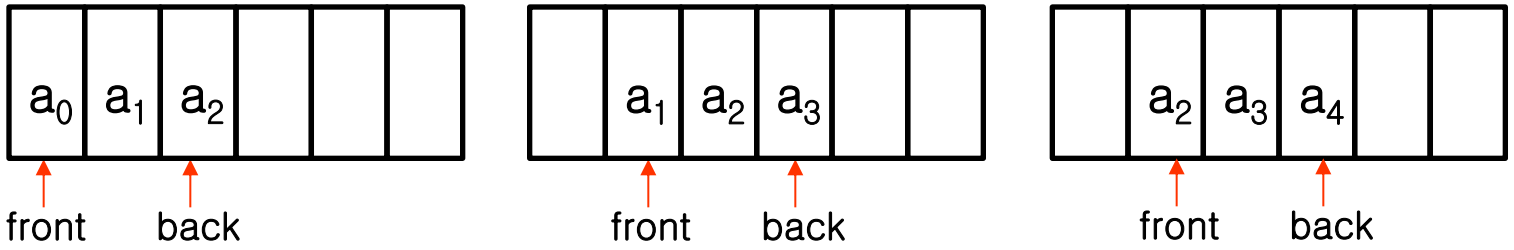
Queue Model

Figure 3.56 Model of a queue



Implementation of Queue

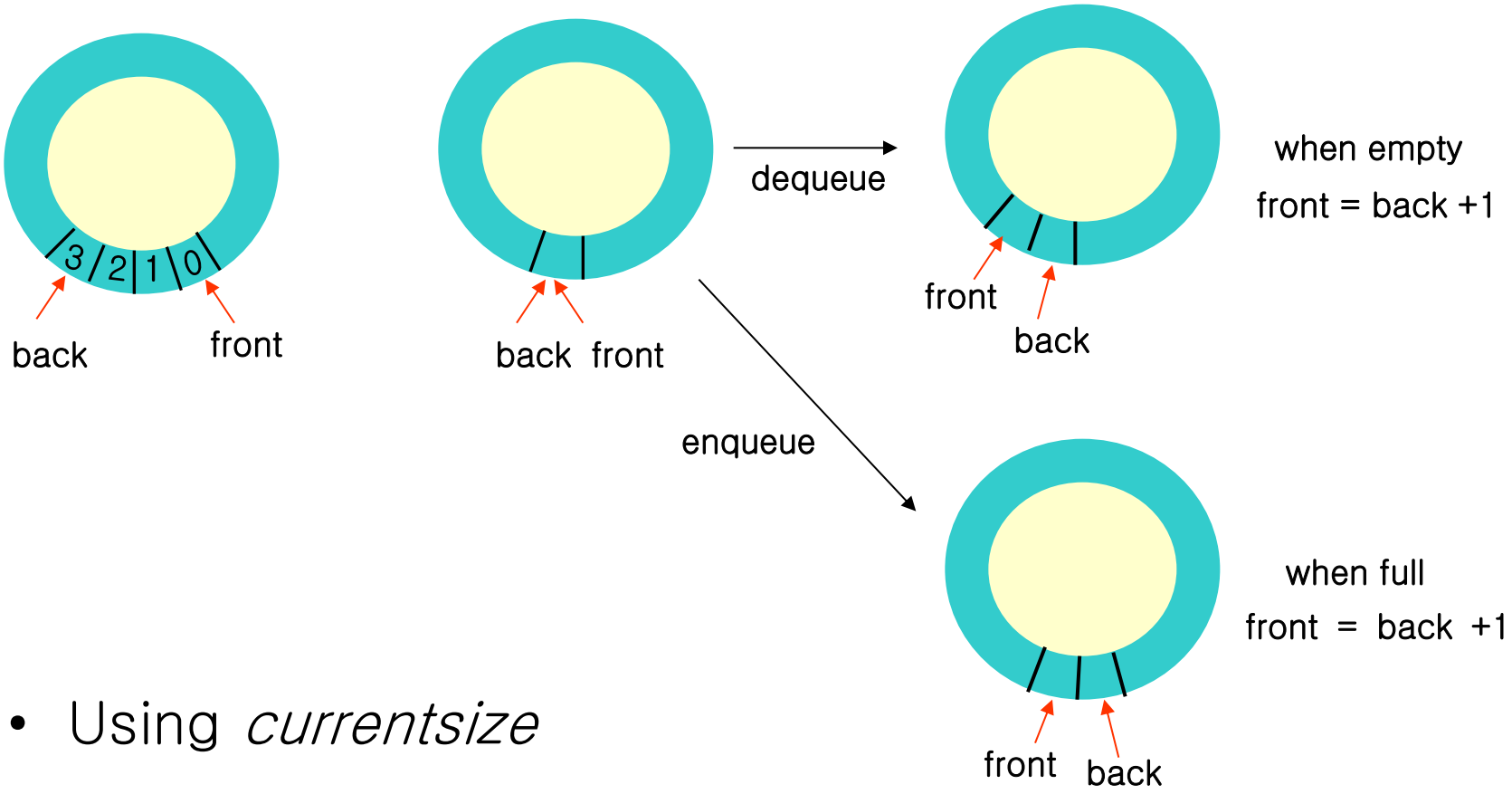
- Array implementation #1



enqueue(x) : back ← back +1; arr[back] = x;
dequeue( ) : front ← front +1;

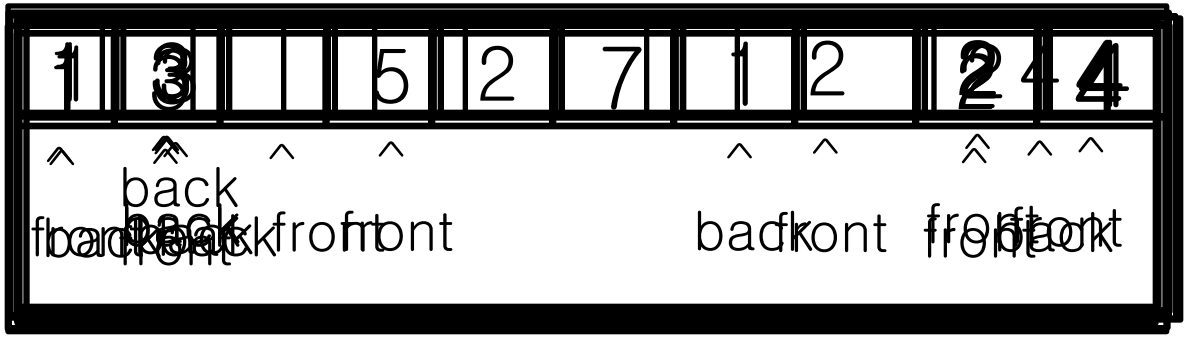
Implementation of Queue

- Array implementation #2



- Using *currentsize*

After dequeue, which Returns 3



For each queue data structure, we keep an array, and the positions *front* and *back*, which represent the ends of the queue. Also , keep track of the number of elements that are actually in the queue, *currentsize*

Implementation of Queue

- Pointer – Singly linked list