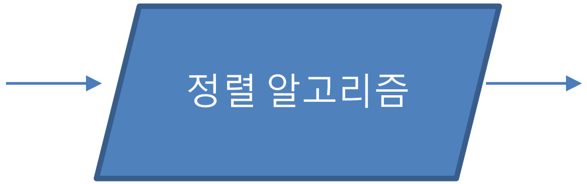
Linear Data Structure

Hyerim Bae
Department of Industrial Engineering, Pusan National University
hrbae@pusan.ac.kr

변수에는 하나의 값만을 담을 수 있다?



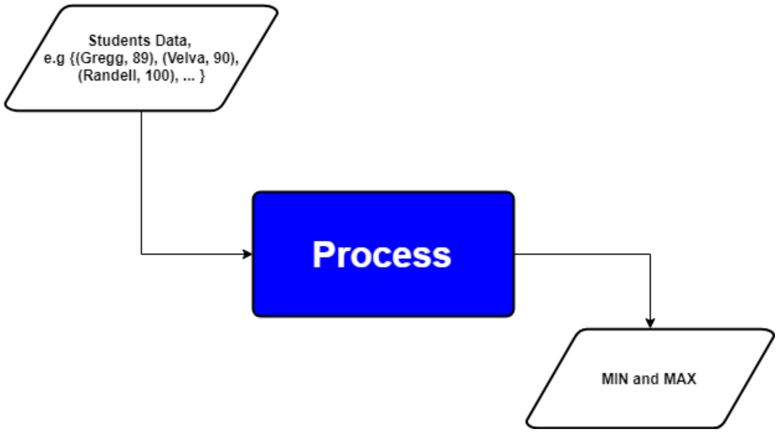
Array

Consider this table. How do we find out the lowest and the highest scores that students have in a programmable way?
How do we store this data into memory using programming languag e like Java?

Student Name	Score
Gregg	89
Velva	90
Randell	100
Vallie	75
Tammi	68
Ima	99
Mina	76
Fleta	88
Shu	89

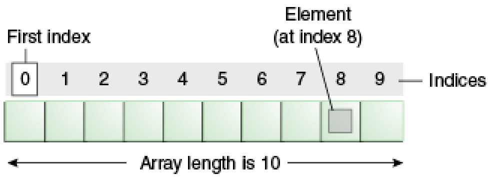
Array

Consider this table. How do we find out the lowest and the highest scores that students have in a programmable way?
How do we store this data into memory using programming languag e like Java?



Array

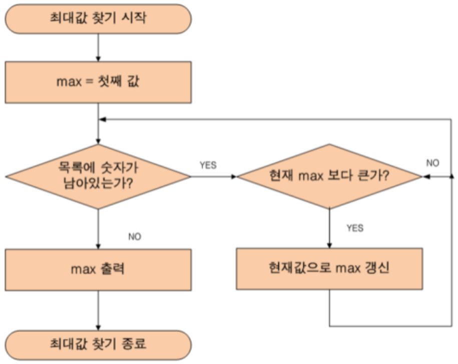
An *array* is a container object that holds a fixed number of values of a single type. The length of an array is established when the array is created. After creation, its length is fixed.

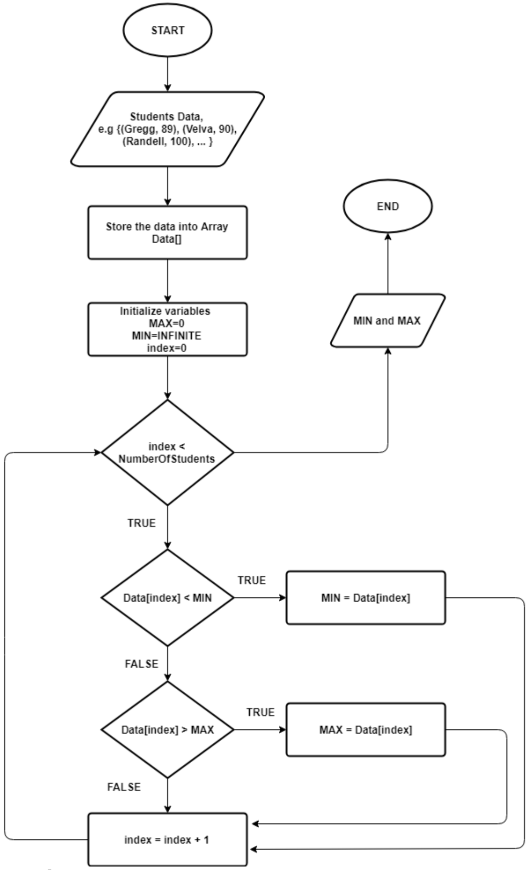


docs.oracle.com

Array

Flowchart of the Solution
using Array to store the data





Array

```java
class Student
{
    public int score;
    public String name;
    Student(String name, int score)
    {
        this.score = score;
        this.name = name;
    }
}
```

```java
public static void main(String[] args) {
    Student[] data;
    data = new Student[8];
    data[0] = new Student("Gregg", 89);
    data[1] = new Student("Velva", 90);
    data[2] = new Student("Randell", 100);
    data[3] = new Student("Vallie", 75);
    data[4] = new Student("Tammi", 68);
    data[5] = new Student("Ima", 99);
    data[6] = new Student("Mina", 76);
    data[7] = new Student("Fleta", 88);

    int MIN = 99999; // assume it as INFINITE
    int MAX = 0;
    for (int index = 0; index < data.length; index++)
{
        if(data[index].score < MIN) {
            MIN = data[index].score;
        } else if (data[index].score > MAX) {
            MAX = data[index].score;
        }
    }

    System.out.println("MIN: " + MIN + ", MAX: " + MAX
);
}
```

Array를 사용하는 장단점

- 장점
  - 원하는 값을 한 번에 접근할 수 있다.
- 단점
  - 한 번 정해진 크기를 바꿀 수 없다.

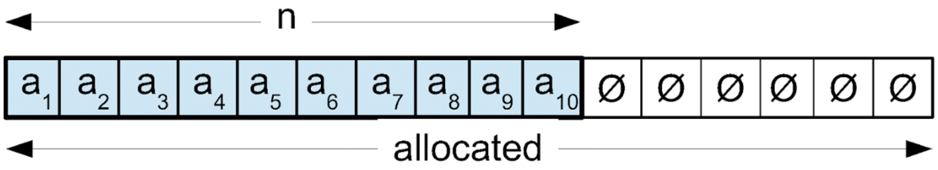
Vector

What if we want dynamically allocate memory just a
s much as we need (not fixed) [with index]?

Vector

The Vector class implements a growable array of objects. Like an array, it contains components that can be accessed using a n integer index.
However, the size of a Vector can grow or shrink as needed to accommodate adding and removing items after the Vector ha s been
created.

docs.oracle.com

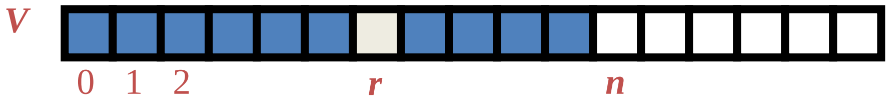

n
a_1 a_2 a_3 a_4 a_5 a_6 a_7 a_8 a_9 a_{10} Ø Ø Ø Ø Ø Ø
allocated

statsbot.co

Vector 는 어떤 기능을 필요로 할까요?

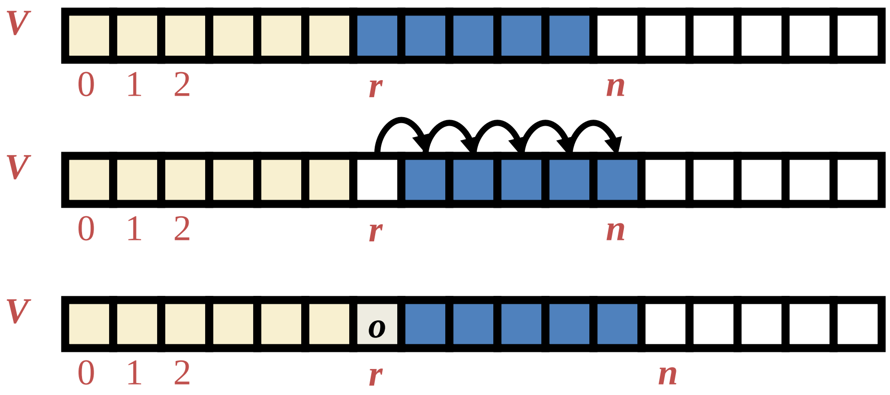
- Use an array ***V*** of size ***N***
- A variable ***n*** keeps track of the size of the vector (number of elements stored)
- Operation ***elemAtRank***(***r***)

벡터를 구현하기 위해서 배열을 쓰면 어떨까?



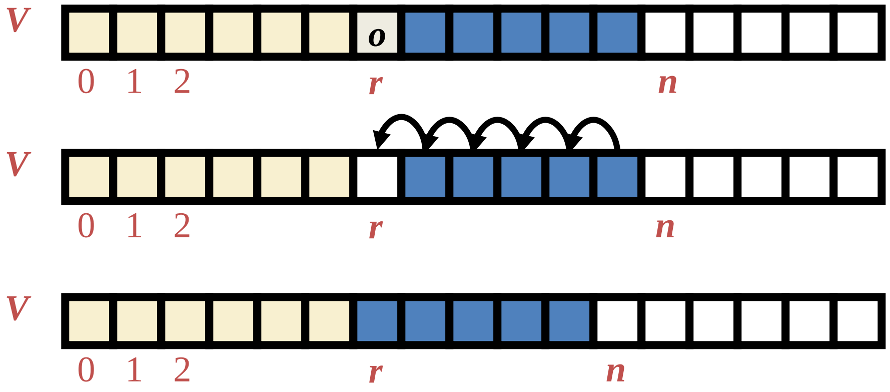
Insertion

- ***insertAtRank***(r, o)
  - r번째 위치에 o를 넣으시오



Deletion

- ***removeAtRank*(*r*)**
  - r번째 있는 원소를 삭제하시오



벡터가 어떤 함수를 가지면 좋을지

- Read
  - elementAt
- Insertion
  - insertAt
- Deletion
  - removeAt
- Replace
  - replaceAt
- 기타...

벡터의 사이즈

- In an insertion operation, when the array is full, instead of throwing an exception, we can replace the array with a larger one
- How large should the new array be?
  1) incremental strategy: 상수 **c** 만큼 늘리기
  2) doubling strategy: 두 배 늘리기

```
Algorithm insertLast(o)
    if size = S.length – 1 then
        A ← new array of
                size …
        for i ← 0 to size do
            A[i] ← S[i]
        S ← A
    size ← size + 1
    S[size] ← o
```

Incremental Strategy Analysis

- We replace the array $\boldsymbol{k} = \boldsymbol{n}/\boldsymbol{c}$ times
- The total time $\boldsymbol{T}(\boldsymbol{n})$ of a series of $\boldsymbol{n}$ push operations is proportional to

$$\boldsymbol{n} + \boldsymbol{c} + 2\boldsymbol{c} + 3\boldsymbol{c} + 4\boldsymbol{c} + \ldots + \boldsymbol{kc} =$$

$$\boldsymbol{n} + \boldsymbol{c}(1 + 2 + 3 + \ldots + \boldsymbol{k}) =$$

$$\underbrace{\boldsymbol{n}}_{\text{Push operation}} + \underbrace{\boldsymbol{ck}(\boldsymbol{k} + 1)/2}_{\text{Copy operation}}$$

- Since $\boldsymbol{c}$ is a constant, $\boldsymbol{T}(\boldsymbol{n})$ is $\boldsymbol{O}(\boldsymbol{n} + \boldsymbol{k}^2)$, i.e., $\boldsymbol{O}(\boldsymbol{n}^2)$
- The amortized time of a push operation is $\boldsymbol{O}(\boldsymbol{n})$

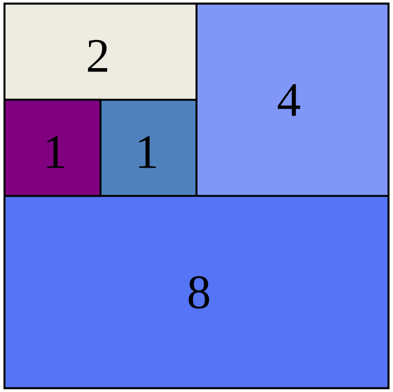
Doubling Strategy Analysis

- We replace the array $\boldsymbol{k} = \log_2 \boldsymbol{n}$ times
- The total time $\boldsymbol{T}(\boldsymbol{n})$ of a series of $\boldsymbol{n}$ push operations is proportional to

$$\boldsymbol{n} + 1 + 2 + 4 + 8 + \ldots + 2^{\boldsymbol{k}} =$$
$$\boldsymbol{n} + 2^{\boldsymbol{k}+1} - 1 = 2\boldsymbol{n} - 1$$

- $\boldsymbol{T}(\boldsymbol{n})$ is $\boldsymbol{O}(\boldsymbol{n})$
- The amortized time of a push operation is $\boldsymbol{O}(1)$

geometric series



벡터의 장단점

- 장점
  - 배열처럼 한 번에 접근
- 단점
  - 원소를 넣고 뺄 때, shifting을 해야함

Vector 사용법

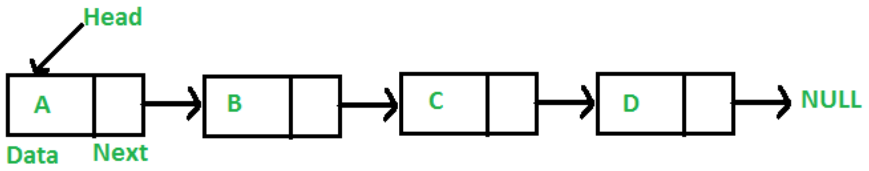
```
public static void main(String[] args) {

    Vector<Student> data = new Vector<Student>();
    data.add(new Student("Gregg", 89));
    data.add(new Student("Velva", 90));
    data.add(new Student("Randell", 100));
    data.add(new Student("Vallie", 75));
    data.add(new Student("Tammi", 68));
    data.add(new Student("Ima", 99));
    data.add(new Student("Mina", 76));
    data.add(new Student("Fleta", 88));

    int MIN = 99999; // assume it as INFINITE
    int MAX = 0;
    for (int index = 0; index < data.size(); index++) {
        if(data.get(index).score < MIN) {
            MIN = data.get(index).score;
        } else if (data.get(index).score > MAX) {
            MIN = data.get(index).score;
        }
    }

    System.out.println("MIN: " + MIN + ", MAX: " + MAX);
}
```

Performance

- In the array based implementation of a Vector
  - The space used by the data structure is $O(n)$
  - ***size***, ***isEmpty***, ***elemAtRank*** and ***replaceAtRank*** run in $O(1)$ time
  - ***insertAtRank*** and ***removeAtRank*** run in $O(n)$ time
- If we use the array in a circular fashion, ***insertAtRank***(0) and ***removeAtRank***(0) run in $O(1)$ time
- In an ***insertAtRank*** operation, when the array is full, instead of throwing an exception, we can replace the array with a larger one

Linked List

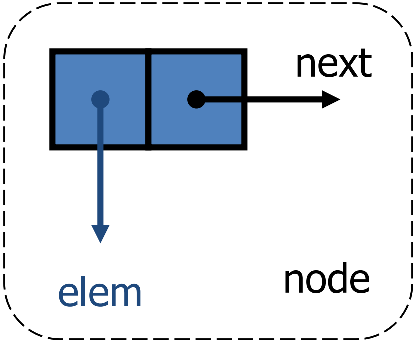
Vector	Linked list
Vector internally uses **dynamic array** to store the elements.	Linked List internally uses **doubly linked list** to s tore the elements.
Manipulation with Vector is **slow** because it inter nally uses array. If any element is removed from t he array, all the bits are shifted in memory.	Manipulation with Linked List is **faster** than Vect or because it uses doubly linked list so no bit shi fting is required in memory.
Vector class can **act as a list** only because it impl ements List only.	Linked List class can **act as a list and queue** bot h because it implements List and Deque interfac es.
Vector is **better for storing and accessing** data.	Linked List is **better for manipulating** data.

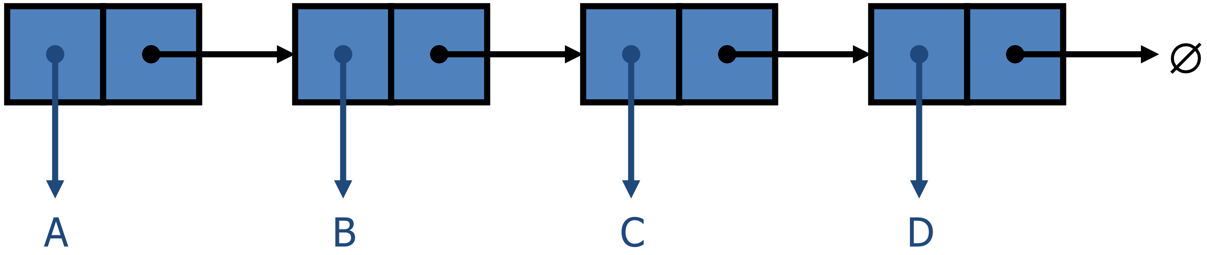


geeksforgeeks.org

Singly Linked List (§ 4.4.1)

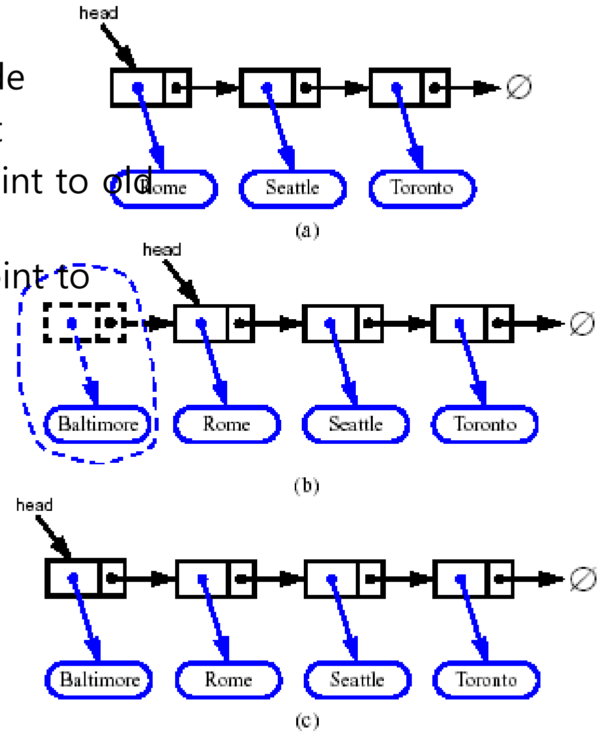
- A singly linked list is a concrete data structure consisting of a sequence of nodes
- Each node stores
  - element
  - link to the next node





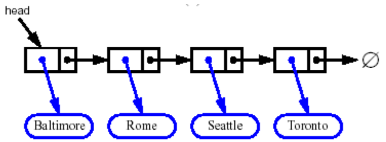
Inserting at the Head

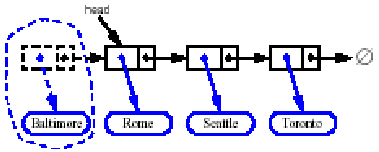
1. Allocate a new node
2. Insert new element
3. Have new node point to old head
4. Update head to point to new node

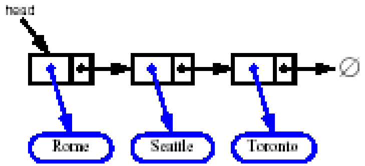


Removing at the Head

1. Update head to point to next node in the list
2. Allow garbage collector to reclaim the former first node

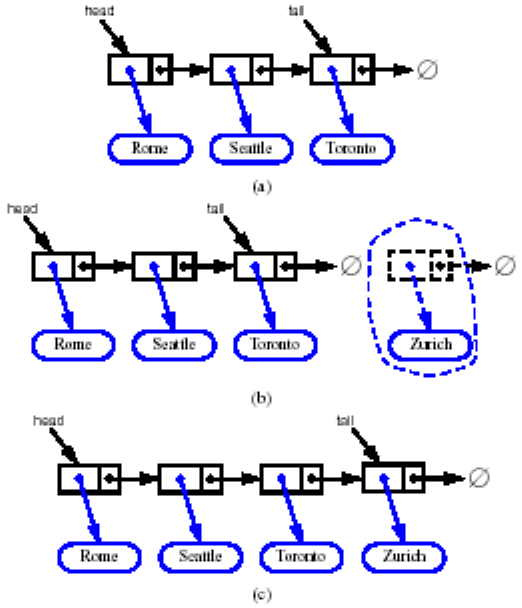






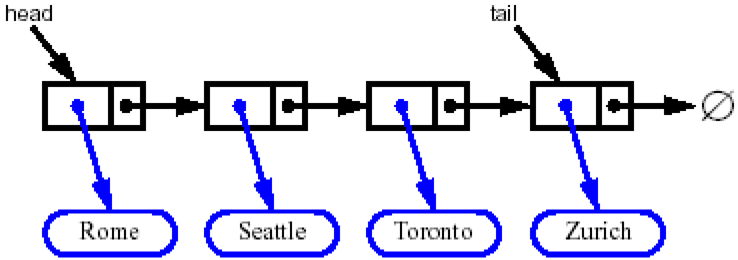
Inserting at the Tail

1. Allocate a new node
2. Insert new element
3. Have new node point to null
4. Have old last node point to new node
5. Update tail to point to new node



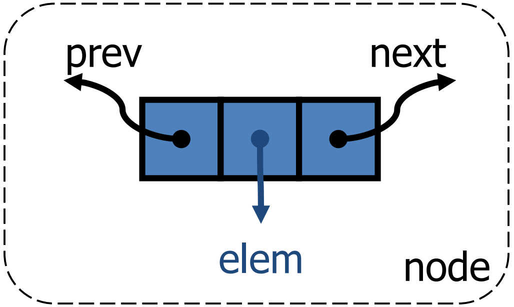
Removing at the Tail

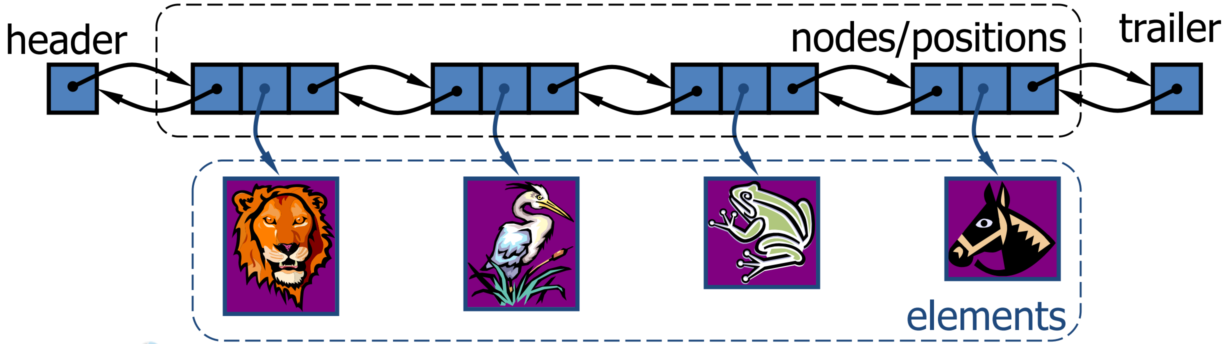
- Removing at the tail of a singly linked list is not efficient!
- There is no constant-time way to update the tail to point to the previous node



Doubly Linked List

- A doubly linked list provides a natural implementation of the List ADT

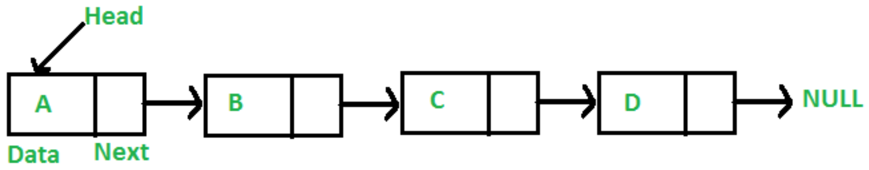




벡터와의 차이

- 벡터는 인덱스를 사용하고, 리스트는 포인터를 사용한다.

리스트에 값을 어떻게 넣을까?



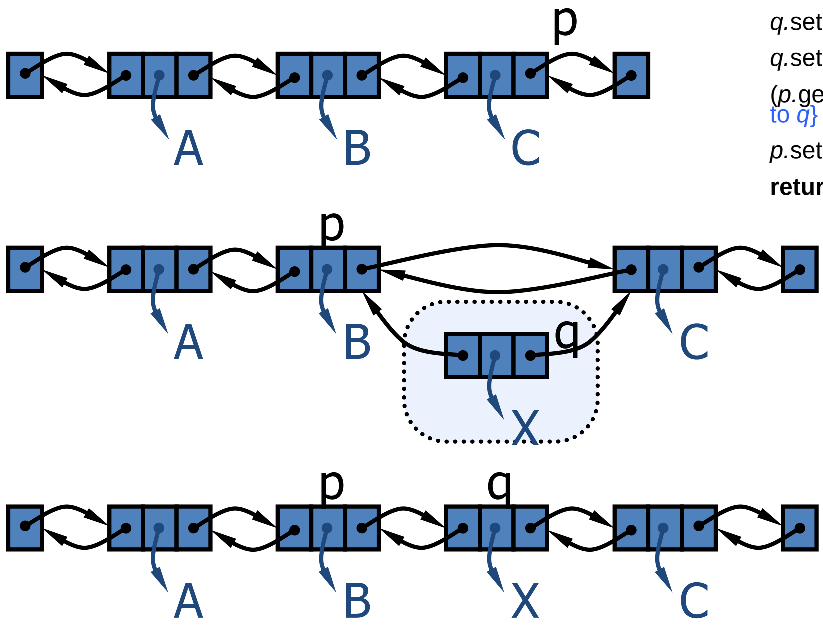
List에 사용되는 함수들

- Accessor
  - first(), last(), prev(p), next(p)
- Updat
  - replace(p, e), insertBefore(p,e), insertAfter(p, e), insertFirst(e), insertLast(e)
- 기타:
  - size(), isEmpty()

Insertion

- insertAfter(p, X)

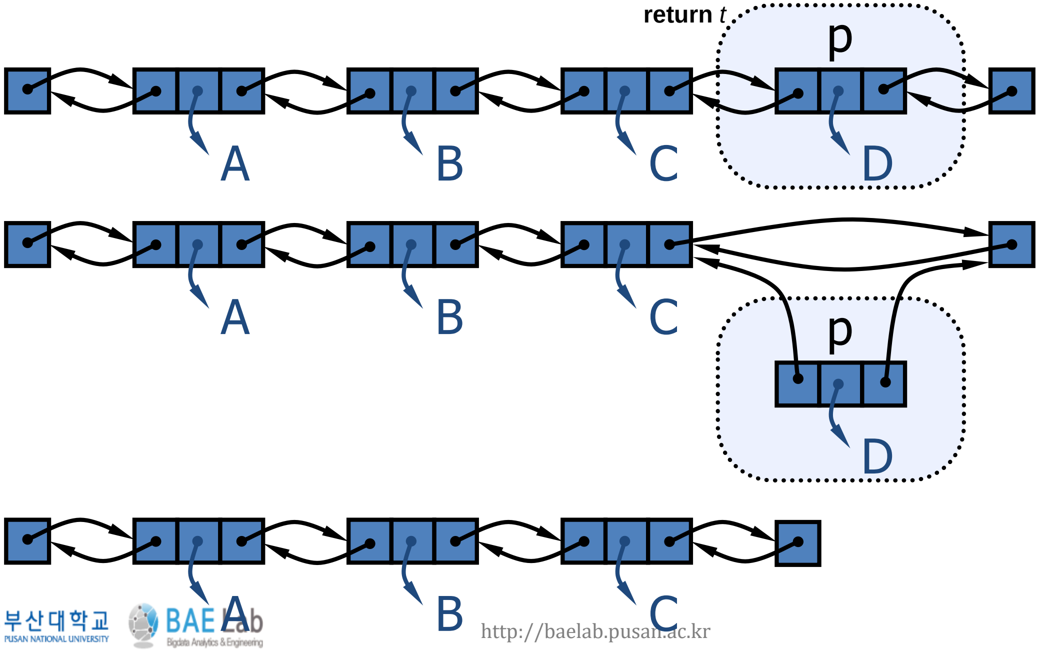
```
Algorithm insertAfter(p,X):
    Create a new node q
    q.setElement(X)
    q.setPrev(p)  {link q to its predecessor}
    q.setNext(p.getNext())          {link q to its successor}
    (p.getNext()).setPrev(q)          {link p's old successor to q}
    p.setNext(q)  {link p to its new successor, q}
    return q        {the position for the element X}
```



Deletion

- remove(p)

```
Algorithm remove(p):
    t = p.element {a temporary variable to hold the return value}
    (p.getPrev()).setNext(p.getNext())        {linking out p}
    (p.getNext()).setPrev(p.getPrev())
    p.setPrev(null)                           {invalidating the position p}
    p.setNext(null)
    return t
```

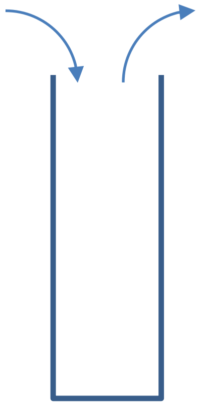


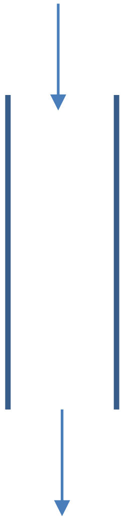
Performance

- In the implementation of the List ADT by means of a doubly linked list
  - The space used by a list with $\boldsymbol{n}$ elements is $\boldsymbol{O}(\boldsymbol{n})$
  - The space used by each position of the list is $\boldsymbol{O}(1)$
  - All the operations of the List ADT run in $\boldsymbol{O}(1)$ time
  - Operation element() of the
    Position ADT runs in $\boldsymbol{O}(1)$ time

스택(Stack) 과 큐(Queue)

- FILO vs. FIFO

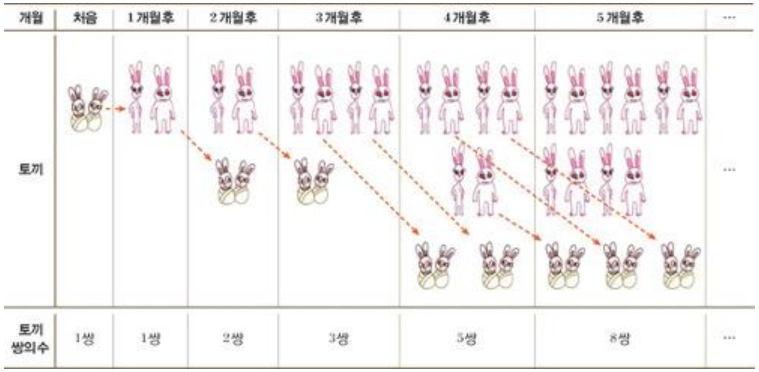




Stack

How do we compute Fibonacci number by using stack ?

한 쌍의 토끼는 매달 암수 한 쌍의 새끼를 낳으며 새로 태어난 토끼도 태어난 지 두 달 후 부터 매달 한 쌍씩의 암수 새끼를 낳는다고 한다. 그러면 농장 주는 1년이 지난 후 모두 몇 쌍의 토끼를 갖게 될까?



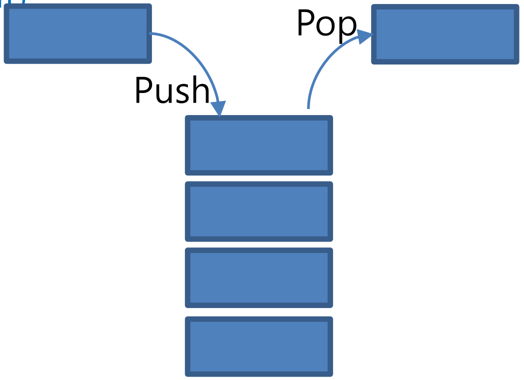
개월	처음	1개월후	2개월후	3개월후	4개월후	5개월후	…
토끼							…
토끼 쌍의수	1쌍	1쌍	2쌍	3쌍	5쌍	8쌍	…

(이미지 출처 : 네이버)

Stack

A stack is a basic data structure that can be logically thought as linear structur e represented by a real physical stack or pile, a structure where insertion and deletion of items takes place at one end called top of the stack. The basic concept can be illustrated by thinking of your data set as a stack of plates or books where you can only take the top item off the stack in order to remove things from it.

LIFO (Last In First Out)



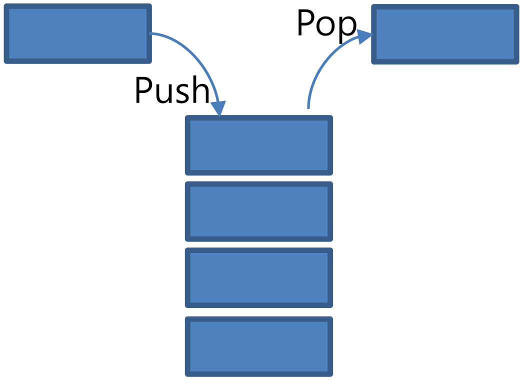
Stack

Main stack operations:

- push(object): inserts an element
- object pop(): removes and returns the last inserted element

Auxiliary stack operations:

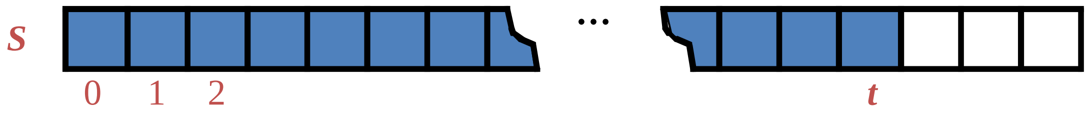
- object top(): returns the last inserted element without removing it
- integer size(): returns the number of elements stored
- boolean isEmpty(): indicates whether no elements are stored



Array-based Stack

- A simple way of implementing the Stack ADT uses an array
- We add elements from left to right
- A variable keeps track of the index of the top element

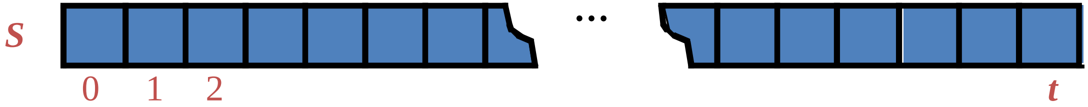
```
Algorithm size()
    return $t + 1$

Algorithm pop()
    if isEmpty() then
        throw EmptyStackException
    else
        $t \leftarrow t - 1$
        return $S[t + 1]$
```



Array-based Stack (cont.)

- The array storing the stack elements may become full
- A push operation will then throw a FullStackException
  - Limitation of the array-based implementation
  - Not intrinsic to the Stack ADT

```
Algorithm push(o)
    if $t = S.length - 1$ then
        throw FullStackException
    else
        $t \leftarrow t + 1$
        $S[t] \leftarrow o$
```

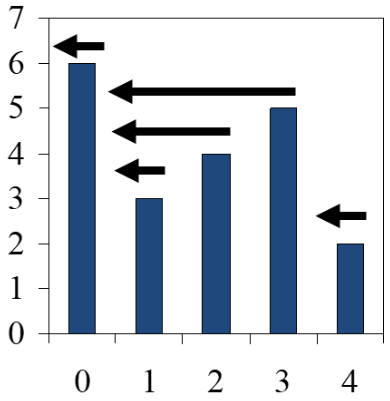


스택 사용예

```
Algorithm ParenMatch(X,n):
Input: An array X of n tokens, each of which is either a grouping symbol, a
variable, an arithmetic operator, or a number
Output: true if and only if all the grouping symbols in X match
Let S be an empty stack
for i=0 to n-1 do
    if X[i] is an opening grouping symbol then
            S.push(X[i])
    else if X[i] is a closing grouping symbol then
            if S.isEmpty() then
                    return false {nothing to match with}
            if S.pop() does not match the type of X[i] then
                    return false {wrong type}
if S.isEmpty() then
    return true {every symbol matched}
else
    return false {some symbols were never matched}
```

스택예제: Computing Spans

- We show how to use a stack as an auxiliary data structure in an algorithm
- Given an an array ***X***, the span ***S***[***i***] of ***X***[***i***] is the maximum number of consecutive elements ***X***[***j***] immediately preceding ***X***[***i***] and such that ***X***[***j***] ≤ ***X***[***i***]
- Spans have applications to financial analysis
  - E.g., stock at 52-week high



X	6	3	4	5	2
S					

Quadratic Algorithm

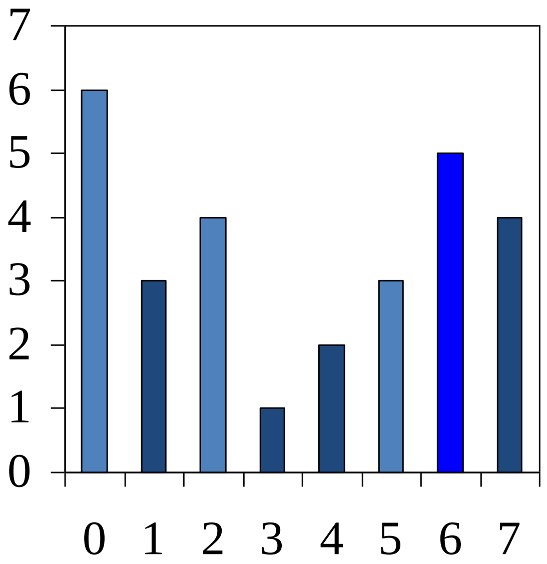
```
Algorithm spans1(X, n)                                              #
    Input array X of n integers
    Output array S of spans of X
    S ← new array of n integers                                     $n$
    for i ← 0 to n − 1 do                                           $n$
        s ← 1                                                       $n$
        while ($s \le i$) ∧ ($X[i - s] \le X[i]$)                   $1 + 2 + \ldots + (n - 1)$
            s ← s + 1                                               $1 + 2 + \ldots + (n - 1)$
        S[i] ← s                                                    $n$
    return S                                                        $1$
```

X	6	3	4	5	2
S					

- Algorithm ***spans1*** runs in $O(n^2)$ time

Computing Spans with a Stack

- We keep in a stack the indices of the elements visible when "looking back"
- We scan the array from left to right
  - Let i be the current index
  - We pop indices from the stack until we find index j such that $X[i] < X[j]$
  - We set $S[i] \leftarrow i - j$
  - We push x onto the stack



Linear Algorithm

- Each index of the array
  - Is pushed into the stack exactly one
  - Is popped from the stack at most once
- The statements in the while-loop are executed at most ***n*** times
- Algorithm ***spans2*** runs in ***O***(***n***) time

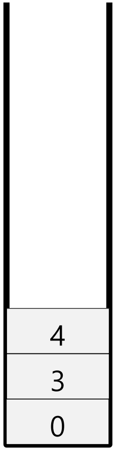
```
Algorithm spans2(X, n)                          #
    S ← new array of n integers                 n
    A ← new empty stack                         1
    for i ← 0 to n − 1 do                       n
        while (¬A.isEmpty() ∧
                X[A.top()] ≤ X[i] ) do          n
            A.pop()                             n
        if A.isEmpty() then                     n
            S[i] ← i + 1                        n
        else
            S[i] ← i − A.top()                  n
        A.push(i)                               n
    return S                                    1
```

X	6	3	4	5	2
S	1	1	2	3	1

```
Algorithm spans2(X, n)                          #
    S ← new array of n integers              n
    A ← new empty stack                          1
    for i ← 0 to n − 1 do                          n
        while (¬A.isEmpty() ∧
                X[A.top()] ≤ X[i] ) do         n
            A.pop()                                    n
        if A.isEmpty() then                        n
            S[i] ← i + 1                               n
        else
            S[i] ← i − A.top()                      n
        A.push(i)                                      n
    return S                                            1
```

1) i=0, X[0]=6, A is empty!
2) i=1, X[1]=3, A is not empty! X[0]>X[1]
3) i=2, X[2]=4, A is not empty! X[1]<=X[2]
   A.pop()
4) i=3, X[3]=5, A is not empty! X[2]<=X[3]
   A.pop()
5) i=4, X[4]=2, A is not empty! X[3]>X[4]

X	6	3	4	5	2	7
S	1	1	2	3	1	?



S[4]=4-3, A.push(4)

S[3]=3-0, A.push(3)

S[0]=1, A.push(0)

If we have X[5] = 7?

Applications of Stack

- Direct applications
  - Page-visited history in a Web browser
  - Undo sequence in a text editor
  - Chain of method calls in the Java Virtual Machine

- Indirect applications
  - Auxiliary data structure for algorithms
  - Component of other data structures

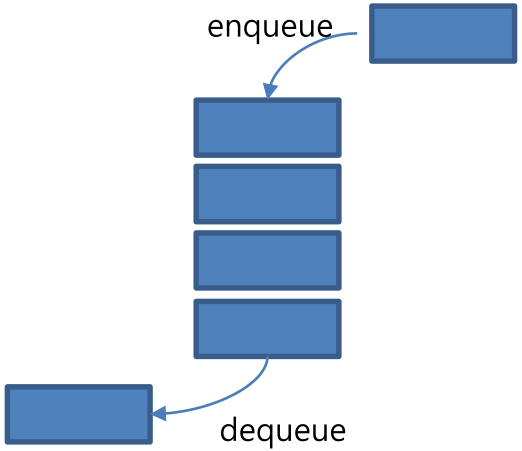
Performance and Limitations of Stack

- Performance
  - Let ***n*** be the number of elements in the stack
  - The space used is ***O***(***n***)
  - Each operation runs in time ***O***(1)

- Limitations
  - The maximum size of the stack must be defined a priori and cannot be changed
  - Trying to push a new element into a full stack causes an implementation-specific exception

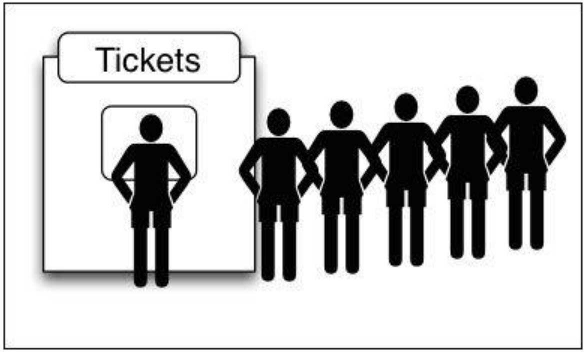
Queue

A queue is a basic data structure that is used throughout programming.
You can think of it as a line in a grocery store.
The first one in the line is the first one to be served. Just like a queue.
A queue is also called a FIFO (First In First Out) to demonstrate the way it accesses data.



Applications of Queue

- Direct applications
  - Waiting lists, bureaucracy
  - Access to shared resources (e.g., printer)
  - Multiprogramming
- Indirect applications
  - Auxiliary data structure for algorithms
  - Component of other data structures



Queue

Operations on Queue Java

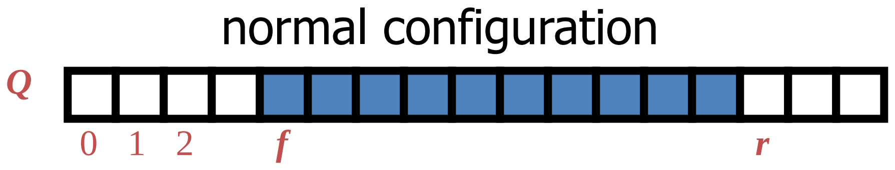
- enqueue(object)
- dequeue()
- front()
- size()
- isEmpty()

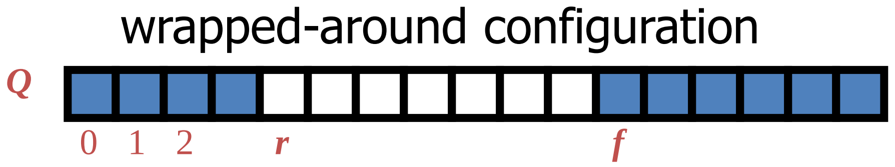
LAST IN
LAST OUT
ONE WAY
FIRST IN
FIRST OUT

tutorialspoint.com

Array-based Queue

- Use an array of size ***N*** in a circular fashion
- Two variables keep track of the front and rear
  - ***f*** index of the front element
  - ***r*** index immediately past the rear element
- Array location ***r*** is kept empty

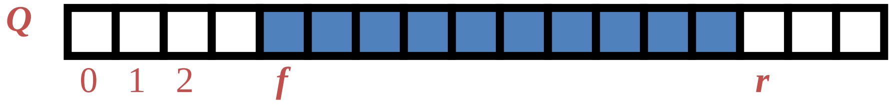


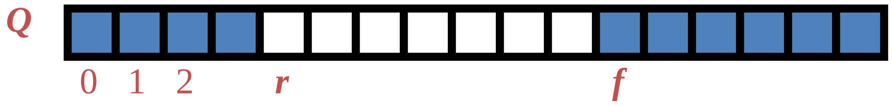


Queue Operations

- We use the modulo operator (remainder of division)

```
Algorithm size()
    return $(N - f + r) \bmod N$

Algorithm isEmpty()
    return $(f = r)$
```

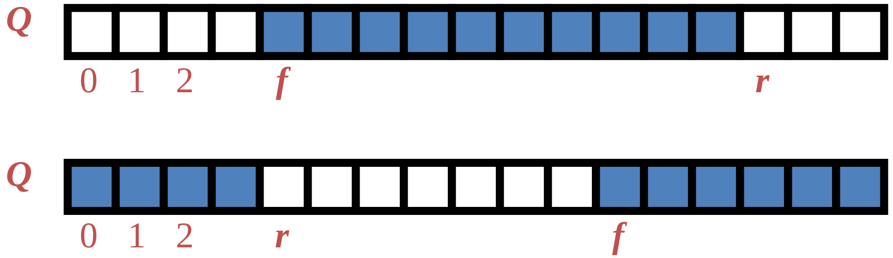




Queue Operations (cont.)

- Operation enqueue throws an exception if the array is full
- This exception is implementation-dependent

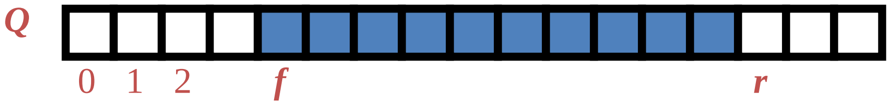
```
Algorithm enqueue(o)
    if size() = N – 1 then
        throw FullQueueException
    else
        Q[r] ← o
        r ← (r + 1) mod N
```

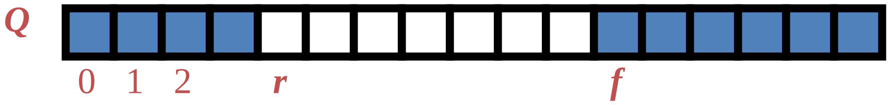


Queue Operations (cont.)

- Operation dequeue throws an exception if the queue is empty
- This exception is specified in the queue ADT

```
Algorithm dequeue()
    if isEmpty() then
        throw EmptyQueueException
    else
        $o \leftarrow Q[f]$
        $f \leftarrow (f + 1) \bmod N$
        return $o$
```

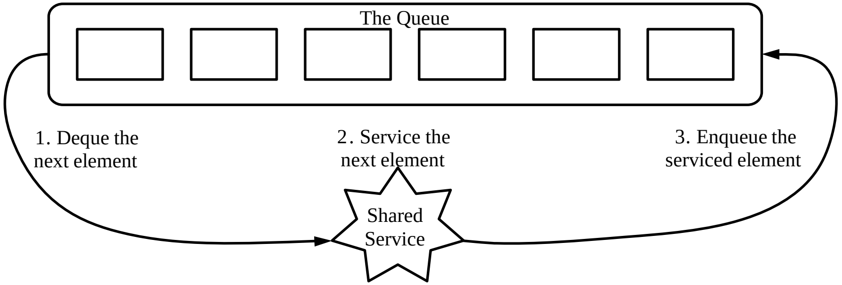




Queue

We can implement a round robin scheduler using a queue, Q, by r epeatedly performing the following steps:

1. e = Q.dequeue()
2. Service element e
3. Q.enqueue(e)


The Queue
1. Deque the next element
2. Service the next element
3. Enqueue the serviced element
Shared Service

Comprehensive example

Motivational Problem (Rope Intranet, Google Code Jam 2010 Round 1)

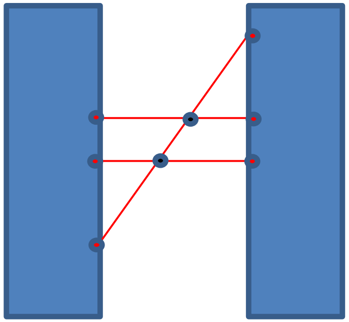
A company is located in two very tall buildings. The company intranet connecting the buildings consists of many w ires, each connecting a window on the first building to a window on the second building.

You are looking at those buildings from the side, so that one of the buildings is to the left and one is to the right. The windows on the left building are seen as points on its right wall, and the windows on the right building are seen as points on its left wall. Wires are straight segments connecting a window on the left building to a window o n the right building.

Noticed that no two wires share an endpoint (in other words, there's at most one wire going out of each window). However, from your viewpoint, some of the wires intersect midway. You've also noticed that exactly two wires meet at each intersection point.

On the below picture, the intersection points are the black circles, while the windows are the red circles.

How many intersection points do you see?



Comprehensive example

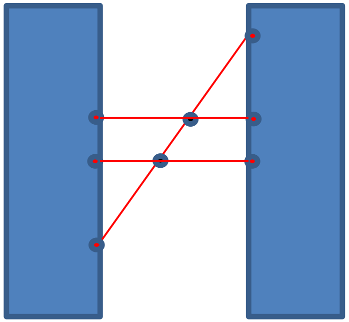
Input
The first line of the input containing an integer **N**, denoting the number of wires.

The next N lines each describe one wire with two integers A_i and B_i. These describe the windows that this wire connects: A_i is the height of the window on the left building, and B_i is the height o
f the window on the right
building.

Output
The number of intersection points you see.

Example Input
3
1 10
5 5
7 7

Example Output
2



Comprehensive example

Observation: Two lines, (A_i, B_i) and (A_j, B_j), intersect each other where

$$(A_i > A_j \text{ AND } B_i < B_j) \text{ OR } (A_i < A_j \text{ AND } B_i > B_j)$$

Example solution which using only native variables.

Assuming the input only consists of 3 wires.

```
        public static void main(String[] args) {
                int A1, A2, A3;
                int B1, B2, B3;

                Scanner in = new Scanner(System.in);

                A1 = in.nextInt(); B1 = in.nextInt();
                A2 = in.nextInt(); B2 = in.nextInt();
                A3 = in.nextInt(); B3 = in.nextInt();

                int NumberOfIntersectionPoints = 0;

                if((A1 > A2 && B1 < B2) || (A1 < A2 && B1 > B2))
                                NumberOfIntersectionPoints = NumberOfIntersectionPoints + 1;

                if((A1 > A3 && B1 < B3) || (A1 < A3 && B1 > B3))
                                NumberOfIntersectionPoints = NumberOfIntersectionPoints + 1;

                if((A3 > A2 && B3 < B2) || (A3 < A2 && B3 > B2))
                                NumberOfIntersectionPoints = NumberOfIntersectionPoints + 1;

                System.out.println(NumberOfIntersectionPoints);
        }
```

What if the input size (number of wires) is large ?

Comprehensive example

Fixed si
ze

```
public static void main(String[] args) {

   int N;
   int[] A = new int[1000];
   int[] B = new int[1000];

   Scanner in = new Scanner(System.in);

   N = in.nextInt();

   for(int i=0; i<N; i++) {
      A[i] = in.nextInt();
      B[i] = in.nextInt();
   }

   System.out.println(getNumberOfIntersectionPoints(N, A, B));
}
```

```
public static int getNumberOfIntersectionPoints(int N, int A[], int B[]) {
   int NumberOfIntersectionPoints = 0;

   for(int i=0; i<N; i++) {
      for(int j=i+1; j<N; j++) {
         if((A[i] > A[j] && B[i] < B[j]) || (A[i] < A[j] && B[i] > B[j]))
            NumberOfIntersectionPoints = NumberOfIntersectionPoints + 1;
      }
   }

   return NumberOfIntersectionPoints;
}
```

Comprehensive example

What if we want dynamically allocate memory just as m uch as we need (not fixed) ?

Comprehensive example

"Rope Intranet" solution using Vector

```
public static void main(String[] args) {

   int N;
   Vector<Integer> A = new Vector<Integer>();
   Vector<Integer> B = new Vector<Integer>();

   Scanner in = new Scanner(System.in);

   N = in.nextInt();

   for(int i=0; i<N; i++) {
      A.add(in.nextInt());
      B.add(in.nextInt());
   }

   System.out.println(getNumberOfIntersectionPoints(N, A, B));
}
```

```
public static int getNumberOfIntersectionPoints(int N, Vector<Integer> A, Vector<Integer> B) {
   int NumberOfIntersectionPoints = 0;

   for(int i=0; i<N; i++) {
      for(int j=i+1; j<N; j++) {
         if((A.get(i) > A.get(j) && B.get(i) < B.get(j)) || (A.get(i) < A.get(j) && B.get(i) > B.get(j)
))
            NumberOfIntersectionPoints = NumberOfIntersectionPoints + 1;
      }
   }

   return NumberOfIntersectionPoints;
}
```

Comprehensive example

Motivational Problem (Broken Keyboard a.k.a. Beiju Text, uva.onlinejudge.org)

You're typing a long text with a broken keyboard. Well it's not so badly broken. The only problem with the keyboard is that sometimes the "home" key or the "end" key gets automatically pressed (internally). You're not aware of this issue, since you're focusing on the text and did not even turn on the monitor! After you finished typing, you can see a text on the screen (if you turn on the monitor).
In Chinese, we can call it Beiju. Your task is to find the Beiju text.

Input
There are several test cases. Each test case is a single line containing at least one and at most 100,000 letters, underscores and two special characters '[' and ']'. '[' means the "Home" key is pressed internally, and ']' means the "End" key is pressed internally. The input is terminated by end-of-file (EOF).

Output
For each case, print the Beiju text on the screen.

Sample Input
```
This_is_a_[Beiju]_text
[[]][][]Happy_Birthday_to_PNU
```

Sample Output
```
BeijuThis_is_a__text
Happy_Birthday_to_PNU
```

Comprehensive example

"Broken keyboard" solution using Vector

"Home" key

"End" key

```
public static void main(String[] args) throws IOException {

    BufferedReader br=new BufferedReader(new InputStreamReader(System.in));
    String s;

    s=br.readLine();

    Vector<Character> value = new Vector<Character>();

    int pointerLocation=0;
    for (char c : s.toCharArray()) {
        if (c=='[') pointerLocation=0;
        else if (c==']') pointerLocation=value.size();
        else value.add(pointerLocation++,c);
    }

    StringBuilder sb=new StringBuilder();
    for (char c : value) sb.append(c);
    System.out.println(sb.toString());
}
```

Comprehensive example

"Broken keyboard" solution using Linkedlist

```
public static void main(String[] args) throws IOException {

    BufferedReader br=new BufferedReader(new InputStreamReader(System.in));
    String s;

    s=br.readLine();

    Linkedlist<Character> value = new Linkedlist<>();

    int pointerLocation=0;
    for (char c : s.toCharArray()) {
        if (c=='[') pointerLocation=0;
        else if (c==']') pointerLocation=value.size();
        else value.add(pointerLocation++,c);
    }

    StringBuilder sb=new StringBuilder();
    for (char c : value) sb.append(c);
    System.out.println(sb.toString());
}
```

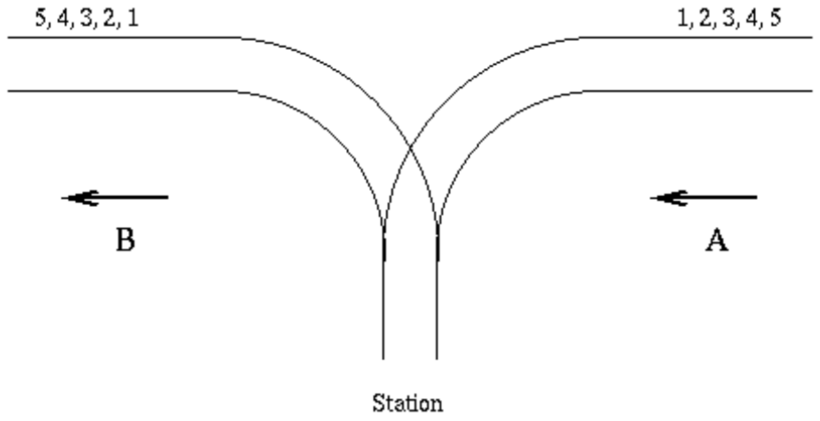
"Home" key

"End" key

Comprehensive example

Motivational Problem (Rails, uva.onlinejudge.org)

There is a famous railway station in PopPush City. Country there is incredibly hil
ly.
The station was built in last century. Unfortunately, funds were extremely limite
d that time. It was possible to establish only a surface track. Moreover, it turned
out
that the station could be only a dead-end one (see picture) and due to lack of
available
space it cou



Comprehensive example

The local tradition is that every train arriving from the direction A continues in the direction B with coaches reorganized in some way. Assume that the train arriving from the direction A has N ≤ 1000 coaches numbered in increasing order 1, 2, . . . , N. The chief for train reorganizations must know whether it is possible to marshal coaches continuing in the direction B so that their order will be a1.a2, . . . , aN . Help him and write a program that decides whether it is possible to get the required order of coaches. You can assume that single coaches can be disconnected from the train before they enter the station and that they can move themselves until they are on the track in the direction B. You can also suppose that at any time there can be located as many coaches as necessary in the station. But once a coach has entered the station it cannot return to the track in the direction A and also once it has left the station in the direction B it cannot return back to the station.

Input

The input file consists of 2 lines. The first line of the block there is the integer N described above. Second line is permutation of 1, 2, . . . , N.

Output

A line of the output file contains 'Yes' if it is possible to marshal the coaches in the order required on th
e
corresponding line of the input file. Otherwise it contains 'No'.

Sample Input Output

```
5
1 2 3 4 5
Yes

5
5 4 1 2 3
No
```

Comprehensive example

Rails Solution using Stack

```java
public static void main(String[] args) throws NumberFormatException, IOException {

   Scanner in = new Scanner(System.in);

   int N;
   N = in.nextInt();
   int[] target = new int[N];
   for (int i = 0; i < N; i++) {
      target[i] = in.nextInt();
   }

   Stack<Integer> station = new Stack<Integer>();
   int targetIndex = 0;
   for (int current = 1; current <= target.length; current++) {
      station.push(current);
      while (!station.empty() && station.peek() == target[targetIndex] && targetIndex <= target.length) {
         station.pop();
         targetIndex++;
      }
   }

   if (station.empty()) {
      System.out.println("Yes");
   } else {
      System.out.println("No");
   }
}
```

Comprehensive example

Motivational Problem (Ferry Loading III, uva.onlinejudge.org)

Before bridges were common, ferries were used to transport cars across rivers. River ferries, unlike their Larger cousins, run on a guide line and are powered by t he river's current. Cars drive onto the ferry from one end, the ferry crosses the river, and the cars exit from the othe r end of the ferry.

There is a ferry across the river that can take **n** cars across the river in **t** minutes and return in **t** minutes.

A car may arrive at either river bank to be transported by the ferry to the opposite bank.
The ferry travels continuously back and forth between the banks so long it is carrying a car or there is
at least one car waiting at either bank. Whenever the ferry arrives at one of the banks,
it unloads its cargo and loads up to n cars that are waiting to cross. If there are more than n, t hose that have been waiting the longest are loaded. If there are no cars waiting on either bank , the ferry waits until one
arrives, loads it (if it arrives on the same bank of the ferry), and crosses the river. At what time does each car reach the other side of the river?

Comprehensive example

Input

Input consists of **n**, **t**, **m**.
m lines follow, each giving the arrival time for a car (in minutes since the beginning of the day), a nd the bank at which the car arrives ('left' or 'right').

Output

Output one line per car, in the same order as the input, giving the time at which that car is un loaded at the opposite bank. You may assume that 0 < n, t, m ≤ 10000. The arrival times are s trictly nondecreasing. The ferry is initially on the left bank. Loading and unloading time may b e considered to be 0.

Sample Input	Sample Output
2 10 10	
0 left	10
10 left	30
20 left	30
30 left	50
40 left	50
50 left	70
60 left	70
70 left	90
80 left	90
90 left	110

Comprehensive example

Ferry Loading IV Solution

```
public static void main(String[] args) throws IOException
{
    Scanner scanner = new Scanner(System.in);

    int L, M;
    final int LEFT = 0;
    final int RIGHT = 1;

    L = scanner.nextInt();
    M = scanner.nextInt();

    L *= 100;
    Queue<Integer>[] cars = new Queue[2];
    cars[LEFT] = new LinkedList<>();
    cars[RIGHT] = new LinkedList<>();

    for(int i=0; i<M; i++) {
        int length;
        String bank;

        length = scanner.nextInt();
        bank = scanner.next();

        if(bank.equals("left"))
            cars[LEFT].add(length);
        else
            cars[RIGHT].add(length);
    }
```

Comprehensive example

Ferry Loading IV Solution

```
    int currentBank = LEFT;
    int numberTimesCrossRiver = 0;

    while (! (cars[LEFT].isEmpty() && cars[RIGHT].isEmpty())) {

        int totalLength = 0;
        while (! cars[currentBank].isEmpty() && totalLength + cars[currentBank].peek() <= L) {
            totalLength += cars[currentBank].peek();
            cars[currentBank].remove();
        }

        numberTimesCrossRiver++;
        if(currentBank == LEFT) currentBank = RIGHT;
        else currentBank = LEFT;
    }

    System.out.println(numberTimesCrossRiver);
}
```