Data Structures: Lists: Stacks and Queues Revisited

YoungWoon Cha
(Slide credits to Won Kim)
Spring 2022

Stack and Queue

Stack: Applications

- Tree data structures (to learn in this course)
- Binary expression evaluation (in a compiler)
- System Stack in OS
  - Activation records
    - nested function calls, including recursive function calls

Implementing a Stack

- Using an Array (global or local)
  - non-circular buffer
  - circular buffer
- Using a Linked List

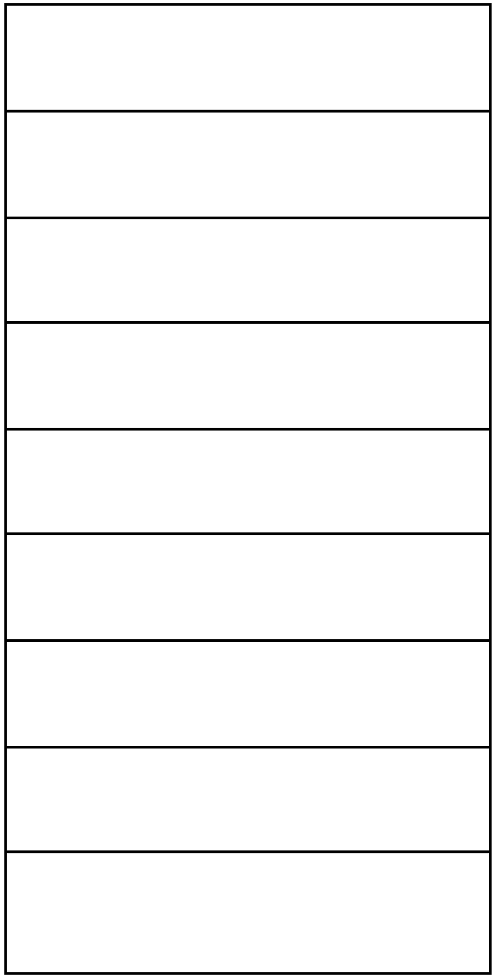
Using a Non-Circular Buffer

- One-Dimensional Array
-     (datatype) stack[stack_size]
- (ex.) char stack[100]

- Variable "Top"
-     initially top = -1 (empty stack)

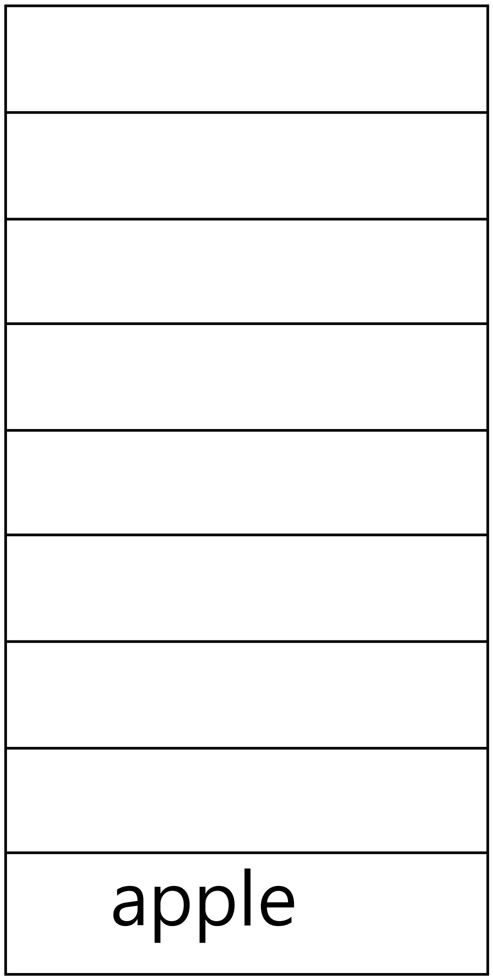
- insert(element) or push(element),
  delete() or pop(),
  stack_full(), stack_empty()

Stack Implementation (Using an Array) (1/12)



top=-1

(2/12)

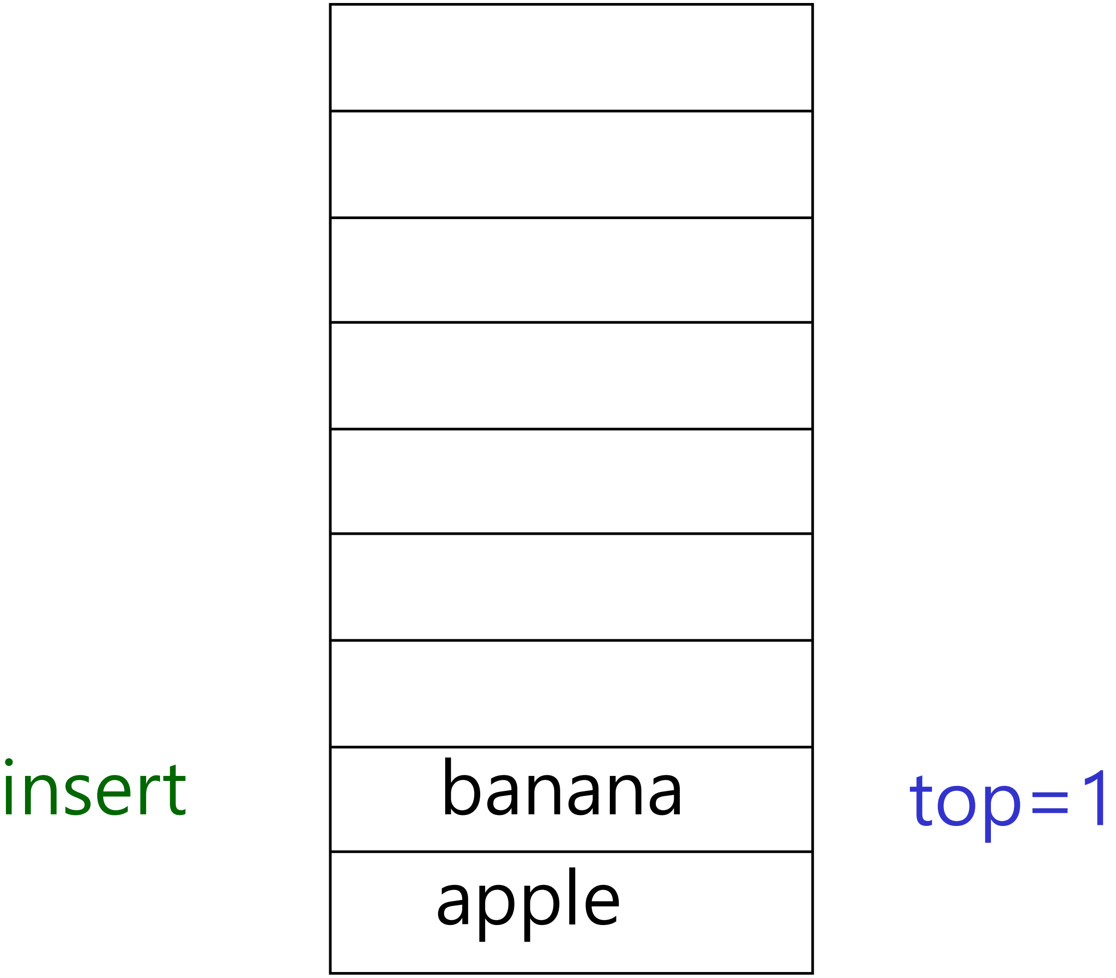


insert

apple

top=0

(3/12)



insert

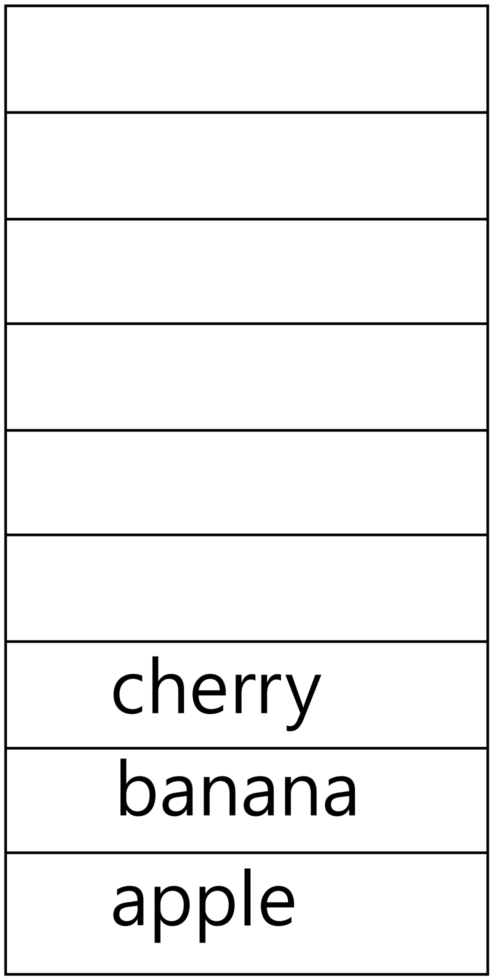
banana

apple

top=1

(4/12)

insert



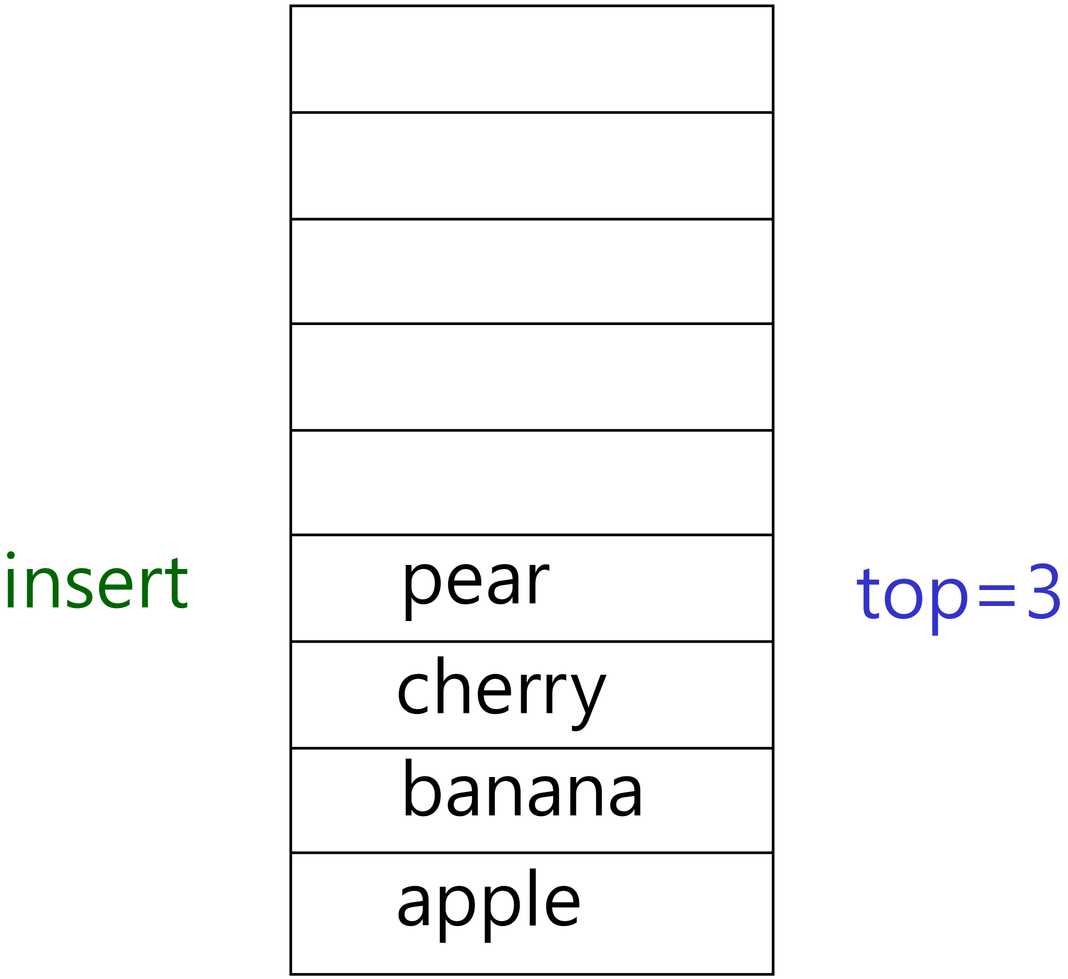
cherry

banana

apple

top=2

(5/12)



insert

pear

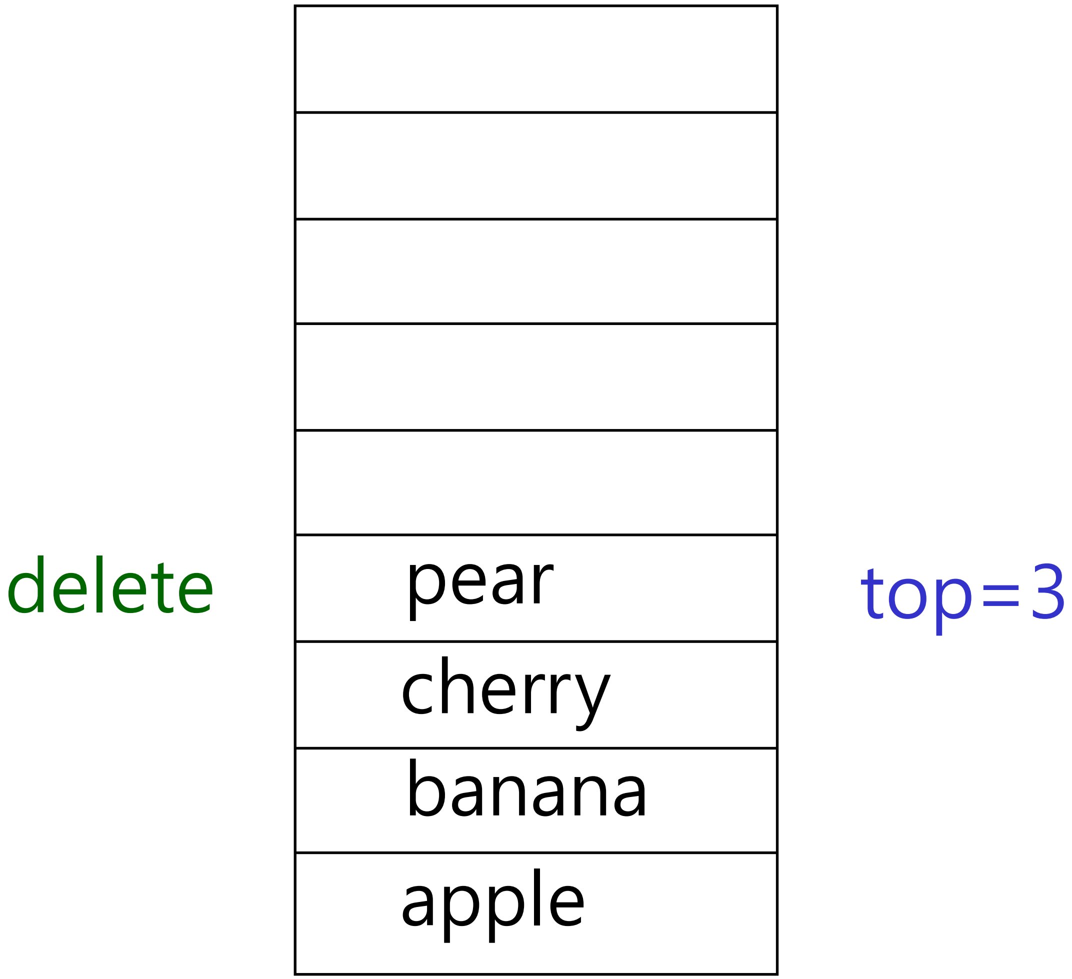
cherry

banana

apple

top=3

(6/12)



delete

pear

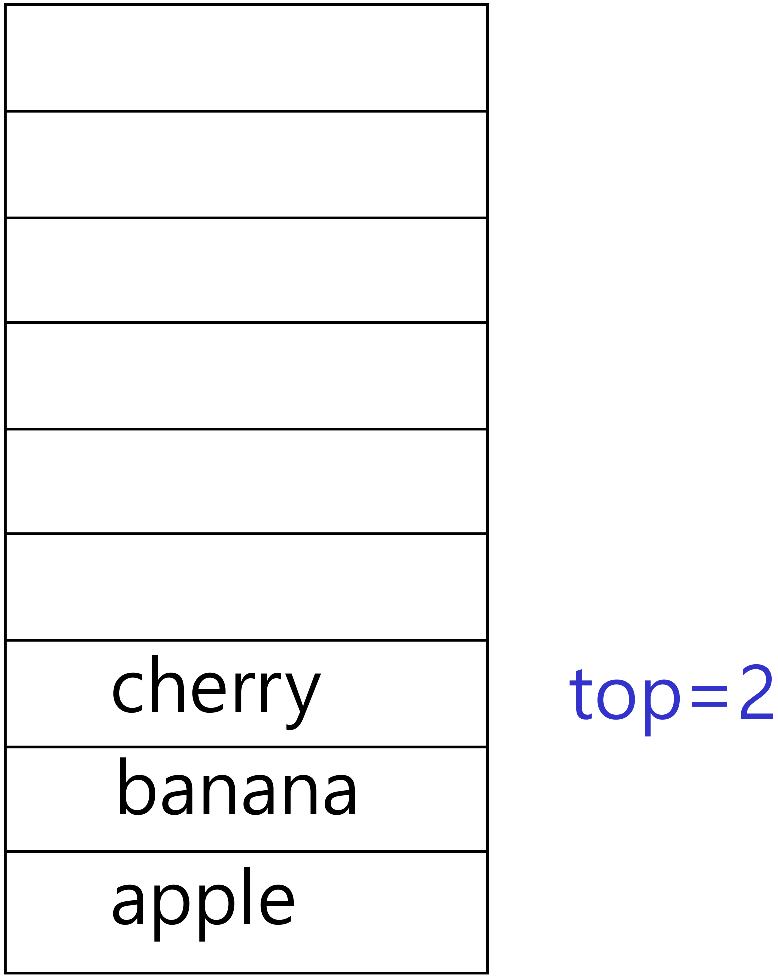
cherry

banana

apple

top=3

(7/12)



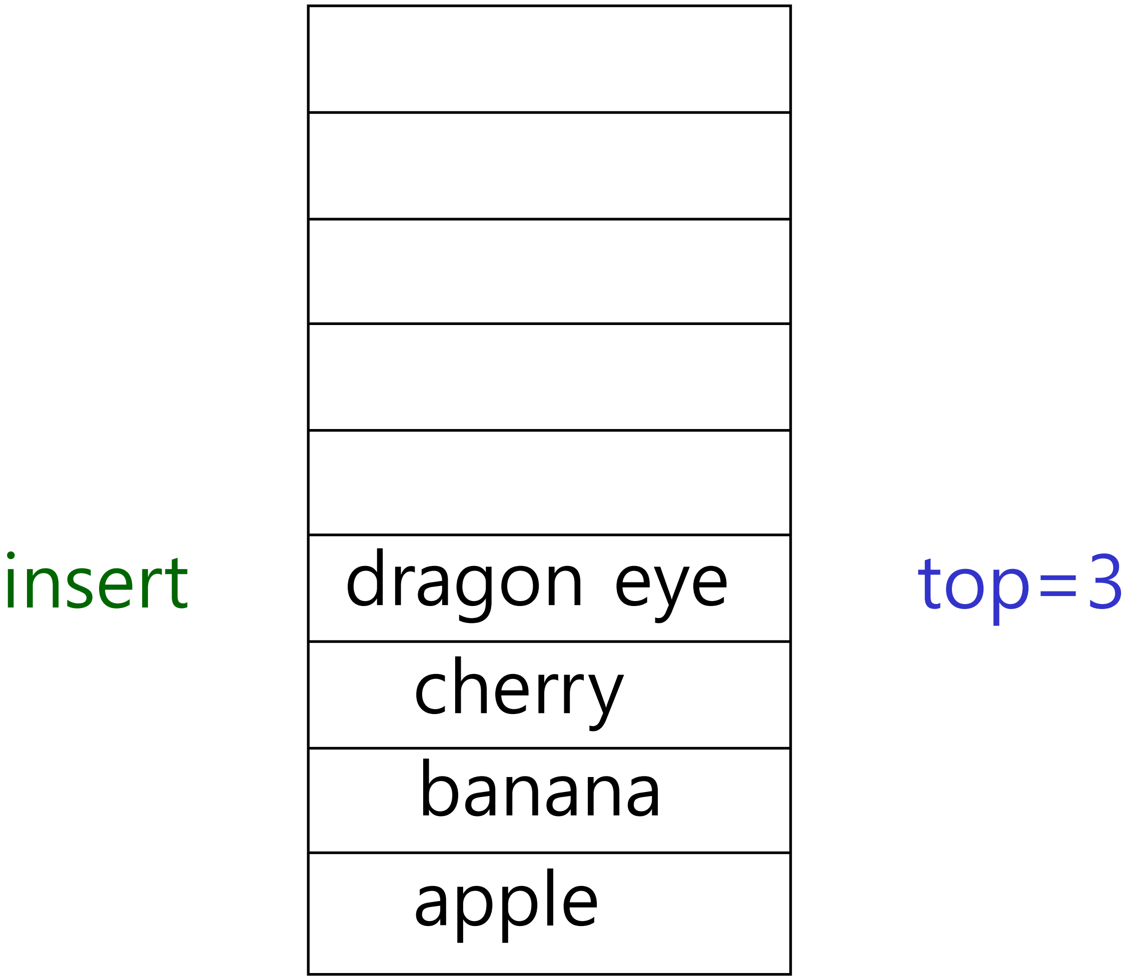
cherry

banana

apple

top=2

(8/12)



insert

dragon eye

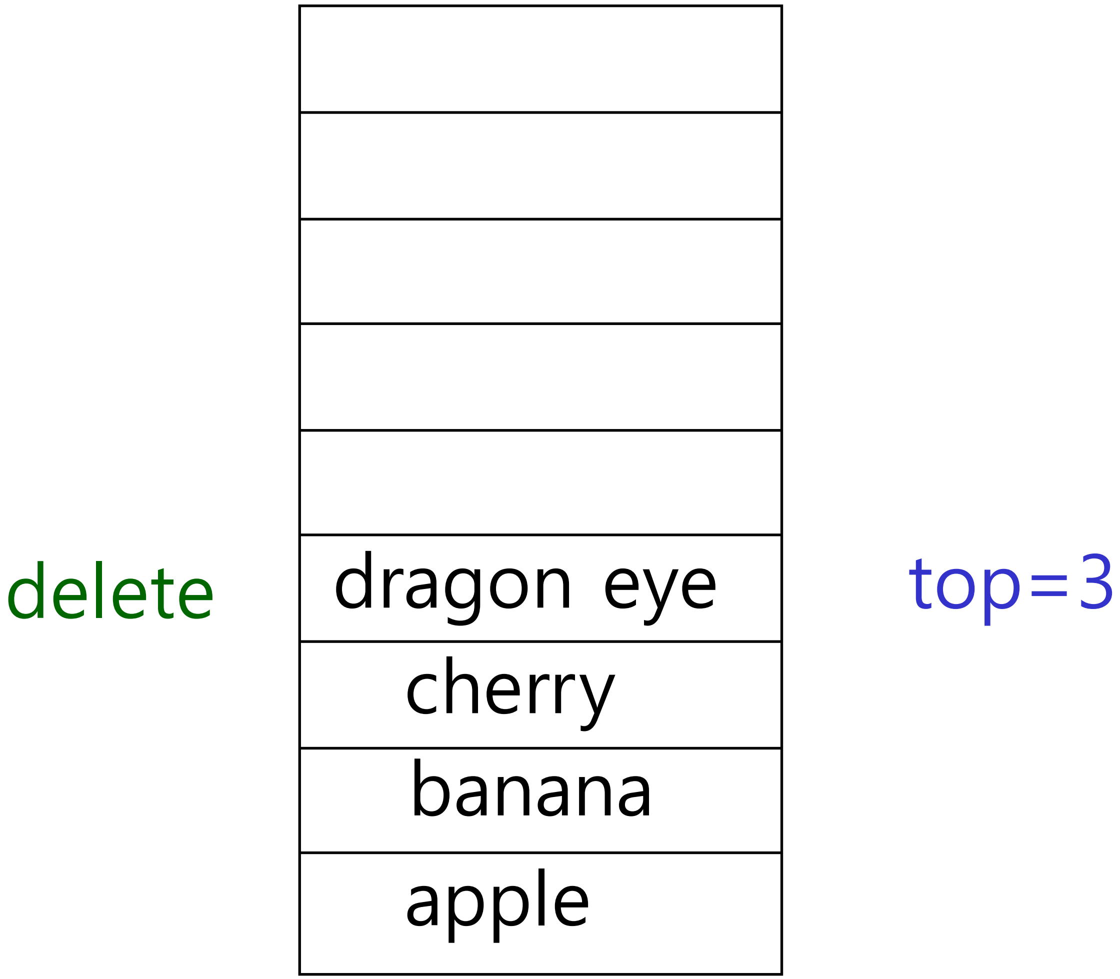
cherry

banana

apple

top=3

(9/12)



delete

dragon eye

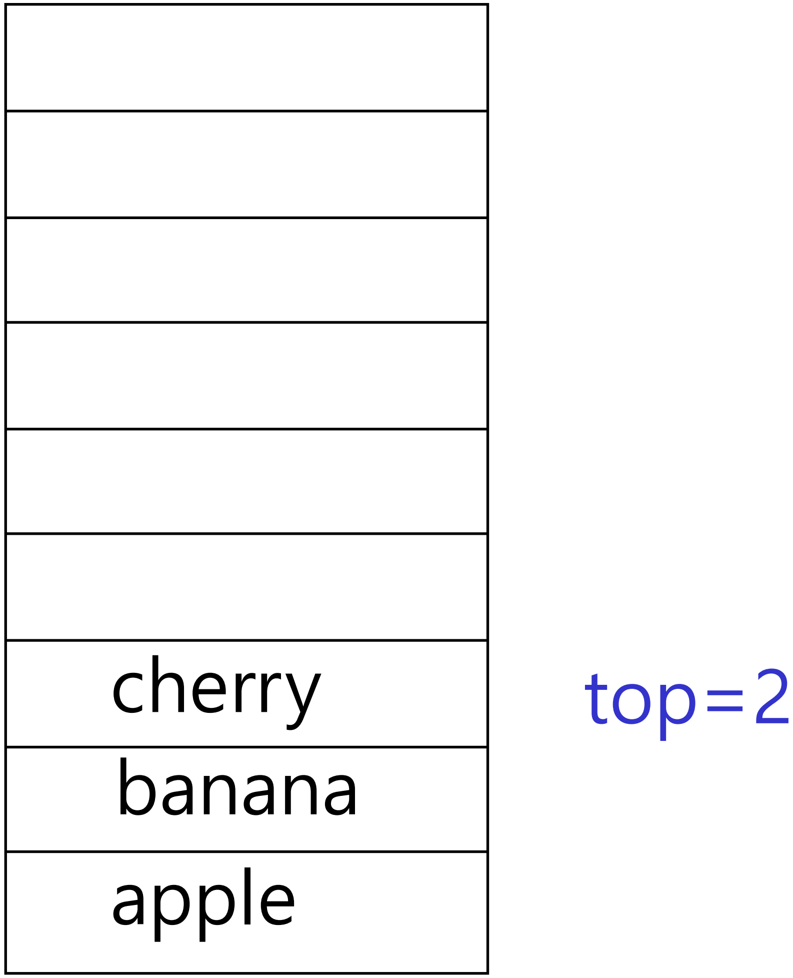
cherry

banana

apple

top=3

(10/12)



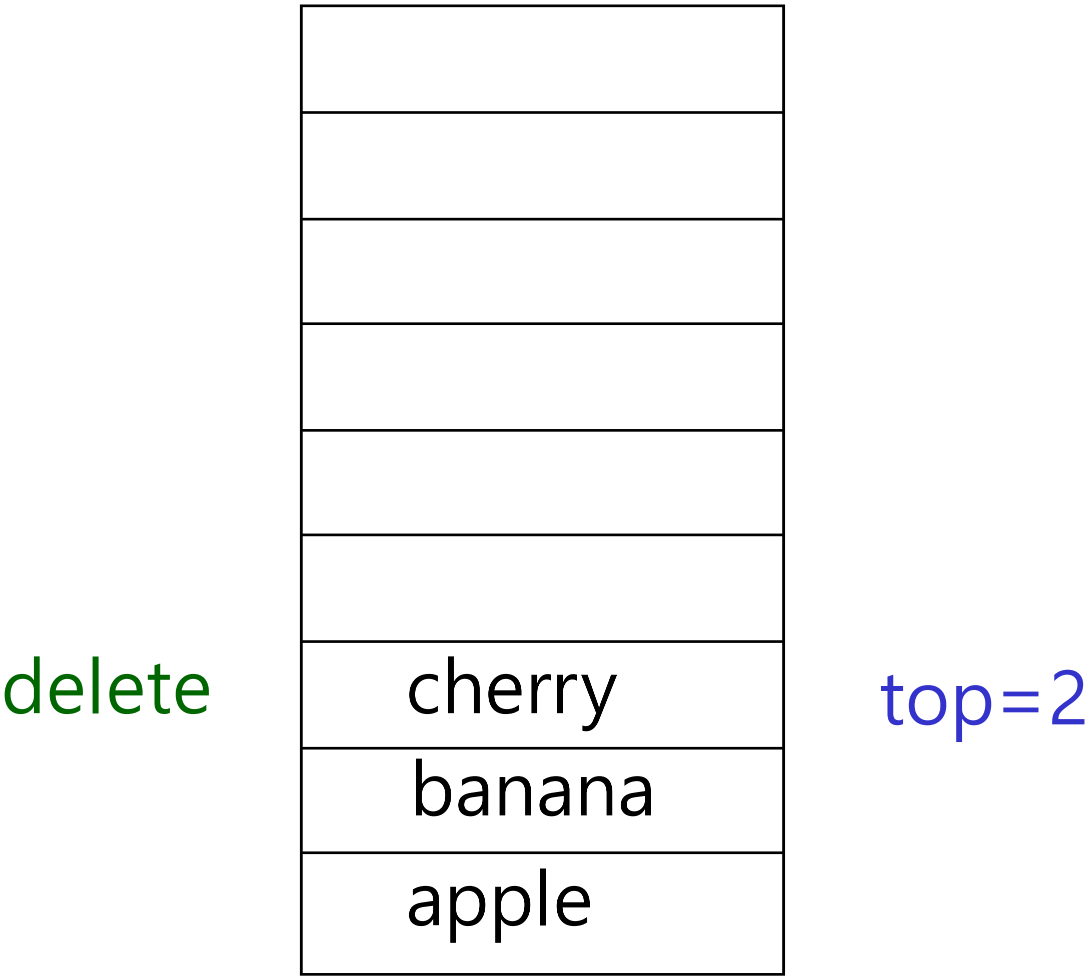
cherry

banana

apple

top=2

(11/12)



delete

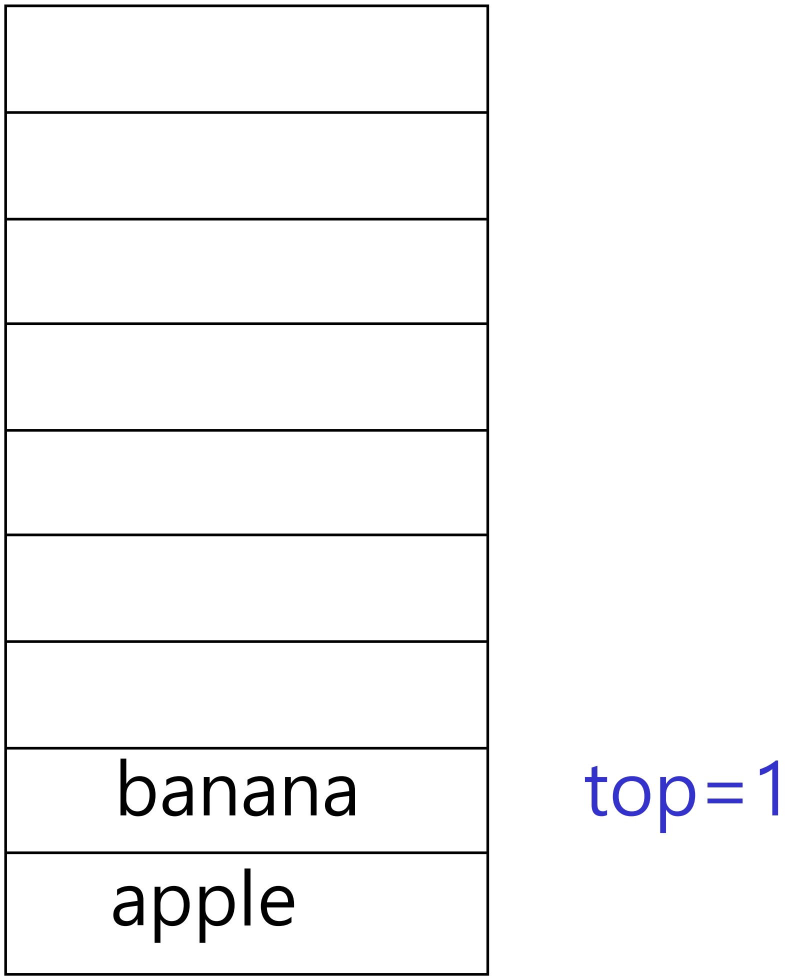
cherry

banana

apple

top=2

(12/12)



Queue

Implementing a Queue

- Using an Array (global or local)
  - non-circular buffer
  - circular buffer
- Using a Linked List

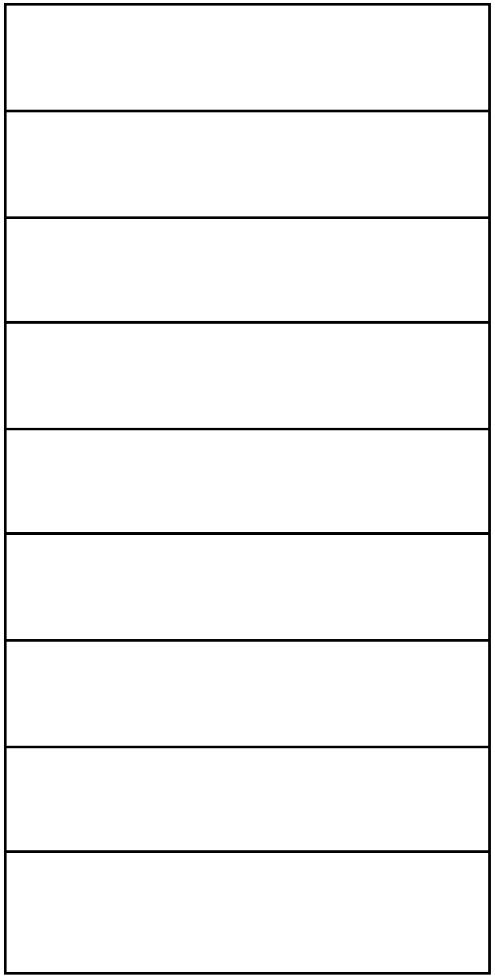
Using a Non-Circular Buffer

- One-Dimensional Array
- (datatype) queue[queue_size]

- Variable “Front”
- Variable “Rear”
- initially front = rear = -1 (empty queue)

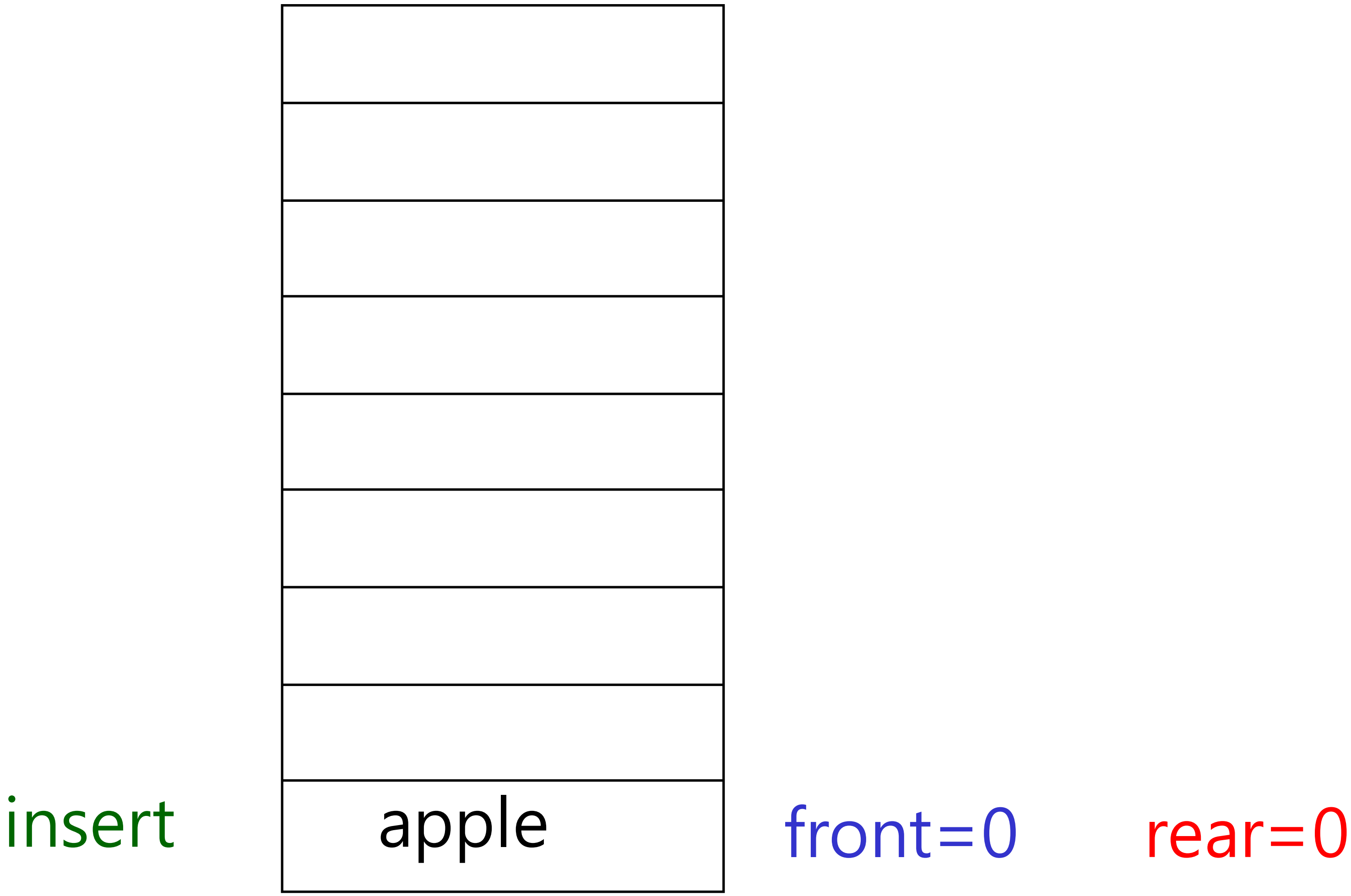
- insert(element) or enqueue(element),
  delete() or dequeue(),
  queue_full(), queue_empty()

Queue Implementation (Using an Array) (1/12)



front=-1 rear=-1

(2/12)



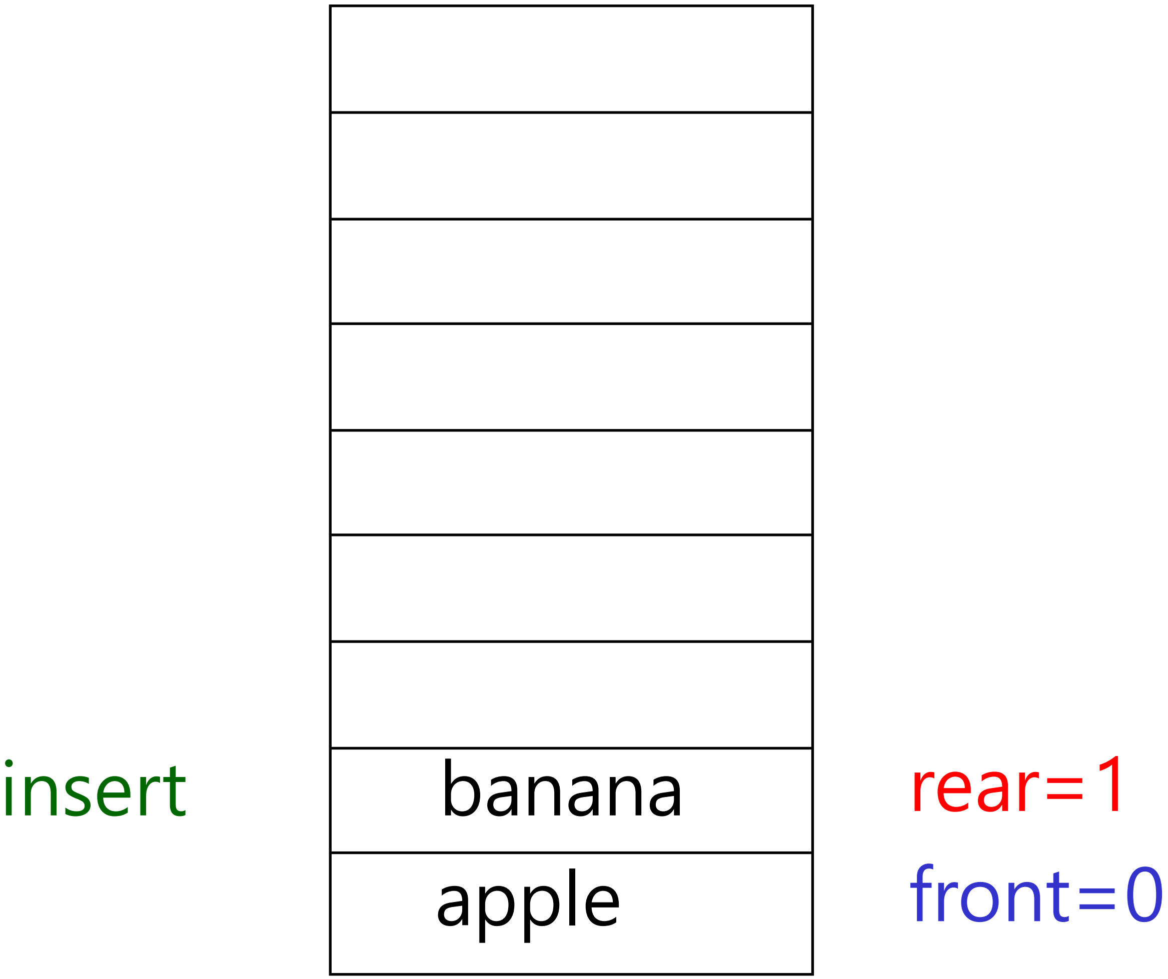
insert

apple

front=0

rear=0

(3/12)



insert

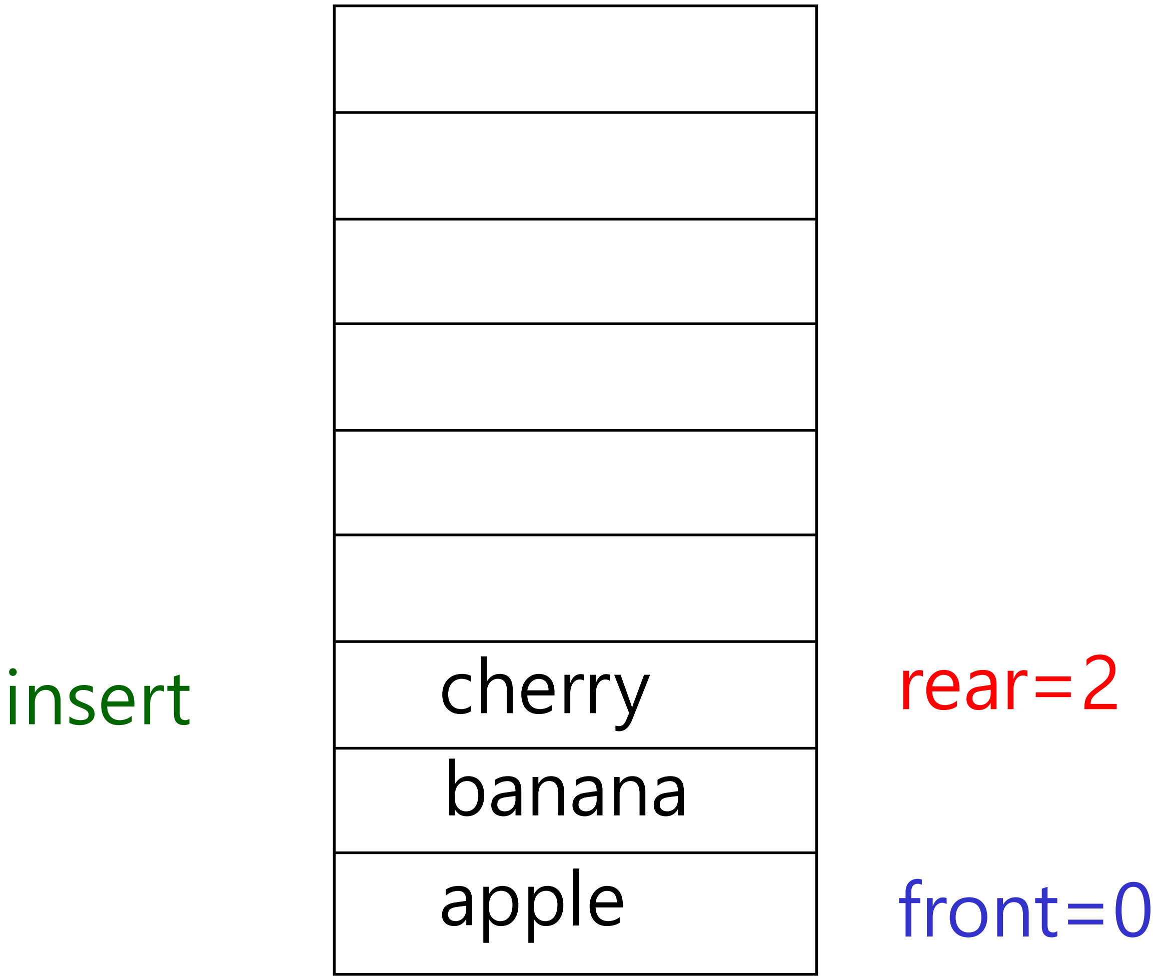
banana

apple

rear=1

front=0

(4/12)



insert

cherry

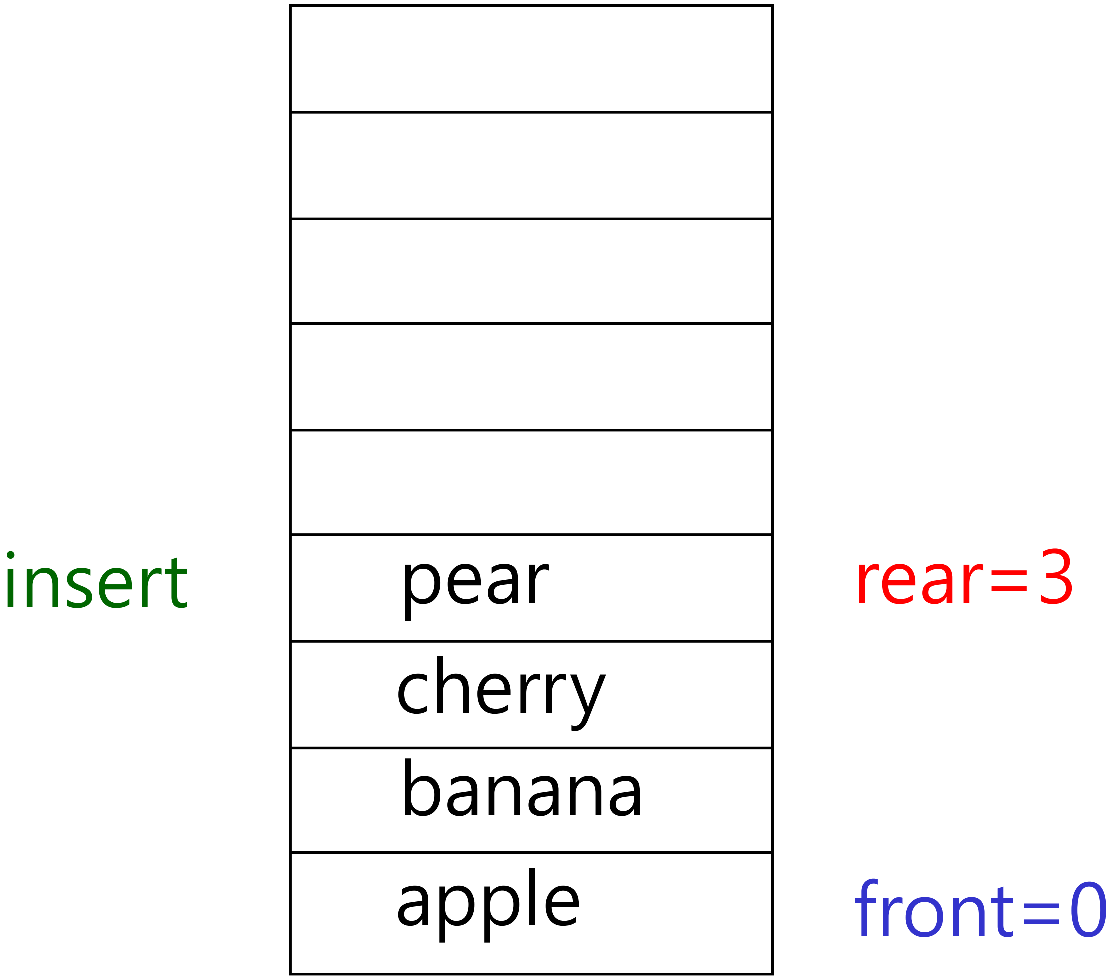
banana

apple

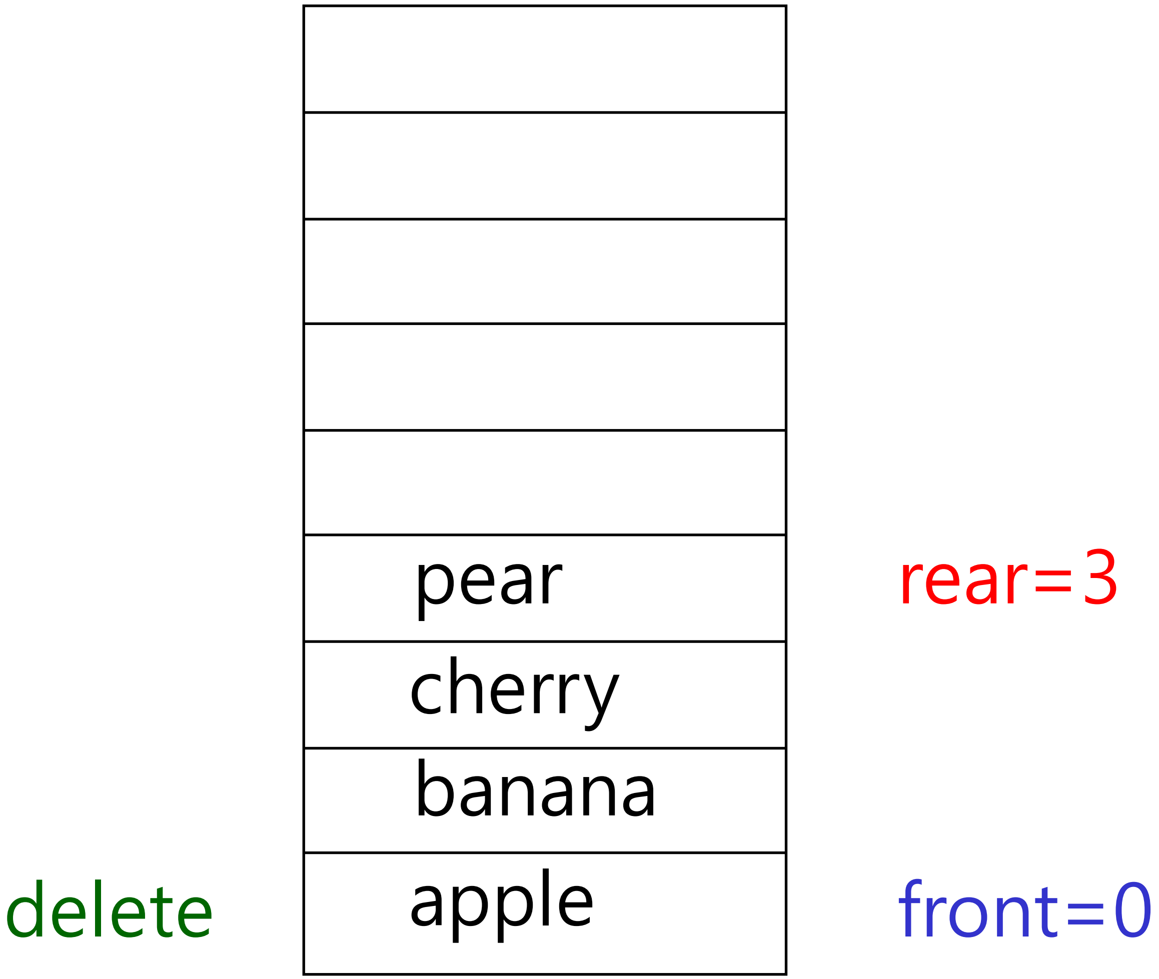
rear=2

front=0

(5/12)



(6/12)



pear

rear=3

cherry

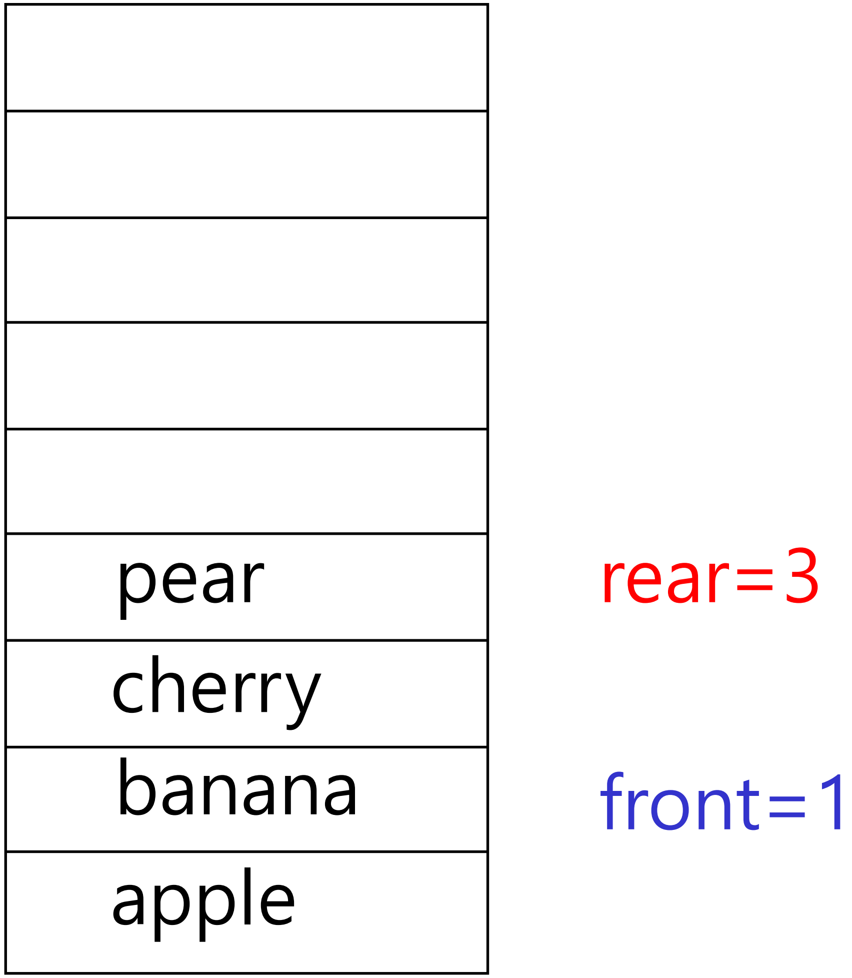
banana

delete

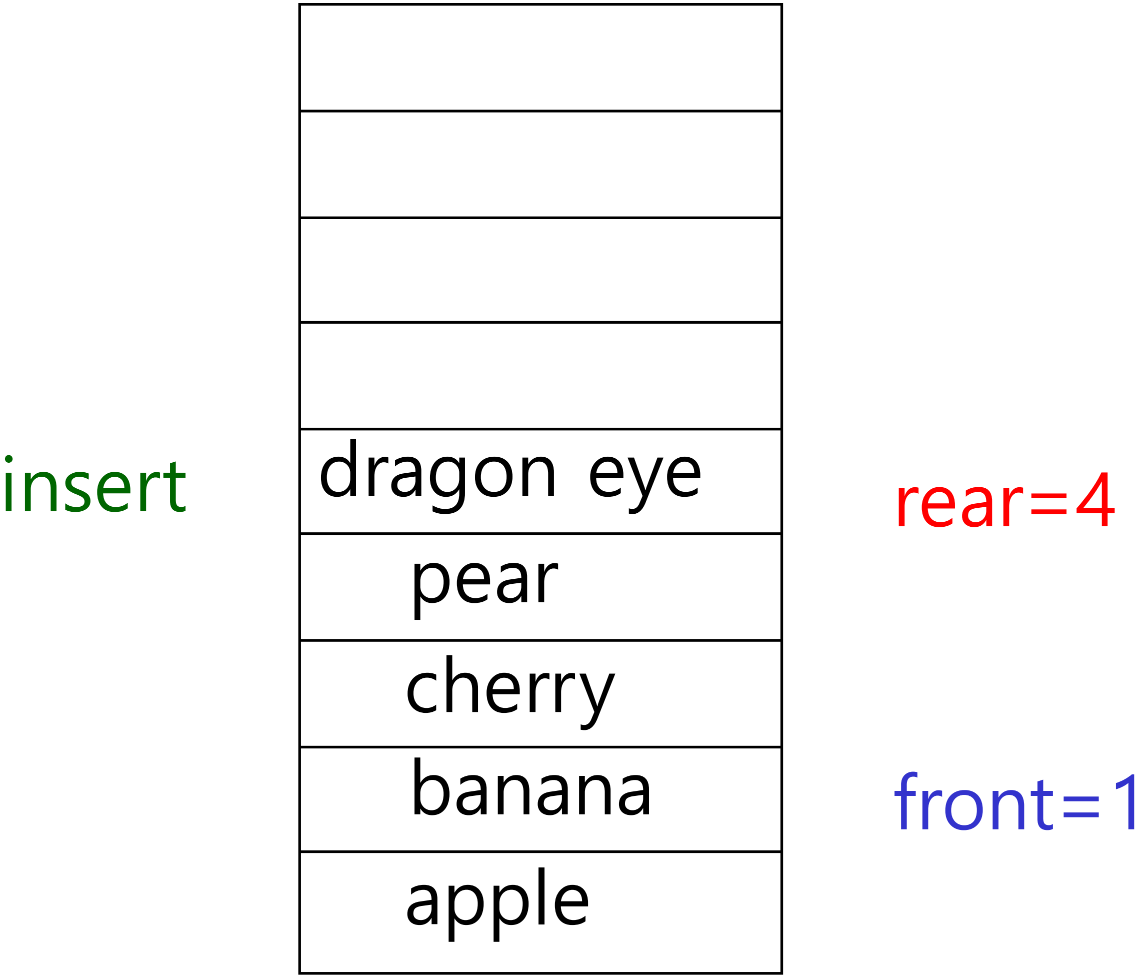
apple

front=0

(7/12)



(8/12)



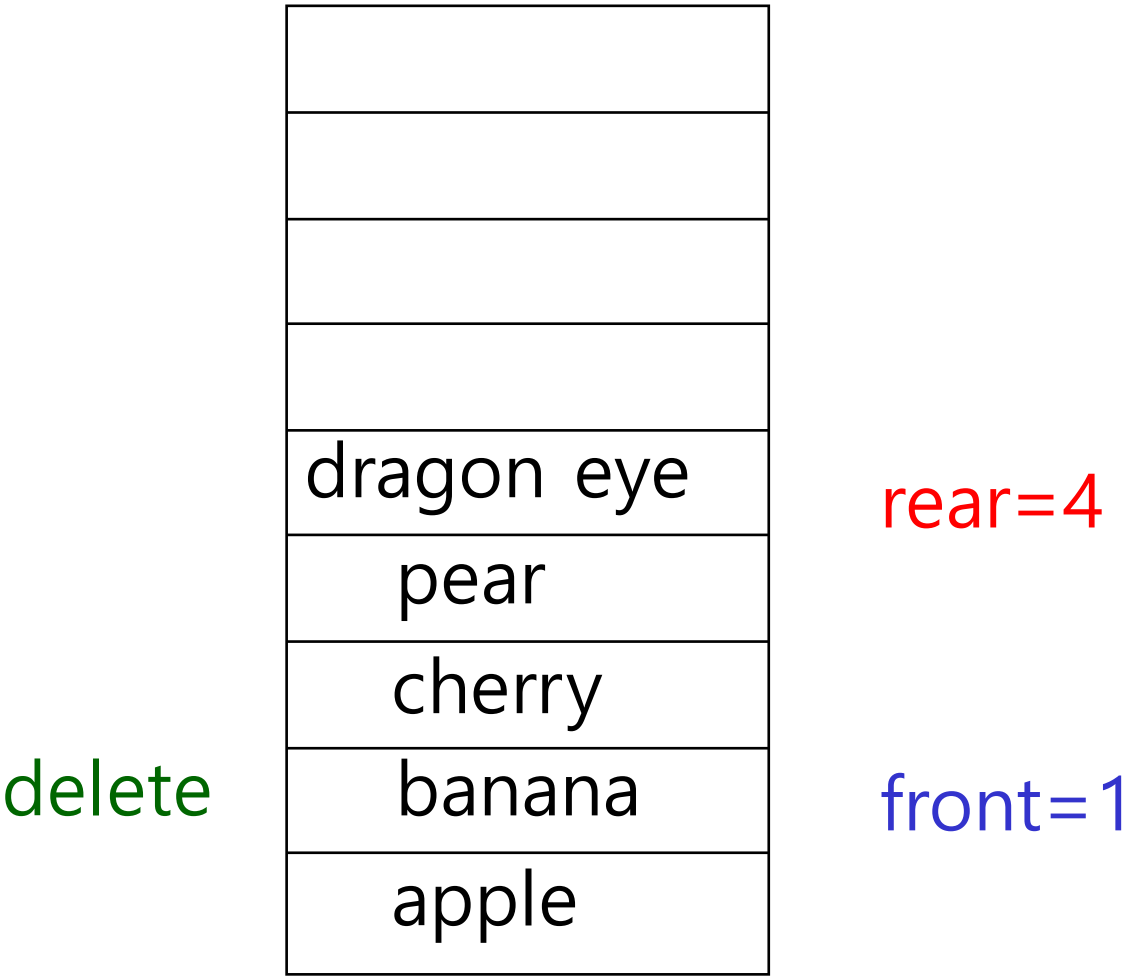
insert

dragon eye
pear
cherry
banana
apple

rear=4

front=1

(9/12)



dragon eye

pear

cherry

banana

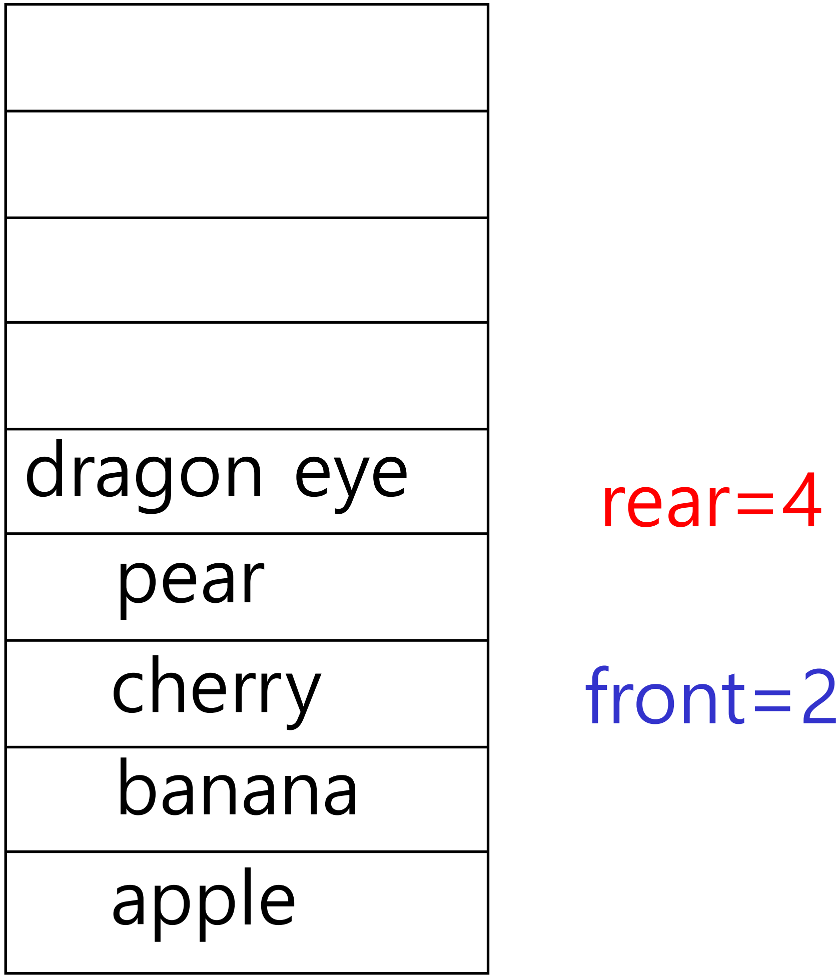
apple

rear=4

delete

front=1

(10/12)



dragon eye

pear

cherry

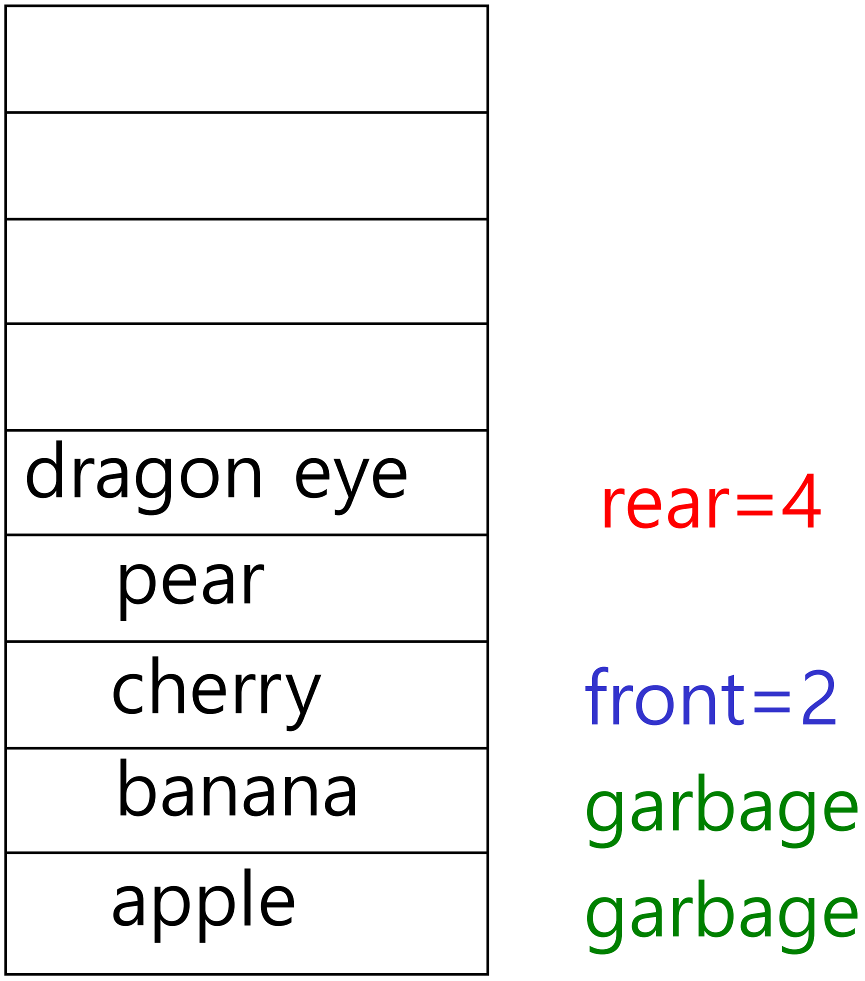
banana

apple

rear=4

front=2

(11/12)



dragon eye

pear

cherry

banana

apple

rear=4

front=2

garbage

garbage

(12/12) wasted space !!

peach	rear=8
apricot	front=7
melon	garbage
orange	
dragon eye	
pear	…
cherry	
banana	
apple	garbage

How to Reuse Space?

peach	rear=8
apricot	front=7
melon	garbage
orange	
dragon eye	
pear	…
cherry	
banana	
apple	garbage

Exercise

Stack Exercise

Queue Exercise

Lab 1

Software Development Process

- Understand All the Requirements
- Plan
  - Development, Testing, Documentation
- Basic Design
- Implement
  - detailed design, code
  - test (code review, test suite)
- Document

Principles of Good Coding

- Follow All the Requirements
- Design a Good Structure
  - divide work into independent and reusable functions
- Make It Easy to Read
  - structure, (variable, function) naming, layout (spacing)
  - function (and inline) comments
- Make It Efficient
  - minimum (instructions, CPU time, memory use)
- Make It Error-Free
  - defensive coding (check for errors)

Principles of Good Testing

- Check All the Requirements
- Do Manual Code Inspection
  - (same as checking PPT, report, exam answers before submitting)
- Create a Test Plan
  - test scenarios (e.g., sequence of push and pop)
  - test environment (e.g., reduce the data structure size – if array size is 1000, for test purpose, set it to 10)
- Create a Test Suite
  - test cases, and golden (correct) result set
- Document and Save the Test Plan and Test Suite

Lab 1-1 (10 points)

- Implement a Stack Program for a (non-Circular) Integer Stack of size 10
- 4 functions, using an array of size 10
  - push (int)
  - int pop ()
  - int stack_full ()
  - int stack_empty ()
- Test the Stack Program
  - Write the main function to exercise the 4 functions

Implementing Stack Operations

- Do not use pointers to call functions
- (for testing) Use scanf, printf only in “main”
- Use defensive coding
  - push
    - call stack_full before “push”
  - pop
    - call stack_empty before “pop”

Function Comments

- push
  - description: appends data to the stack
  - input: data to append (the stack is a global structure)
  - output: none
- pop
  - description: removes data from the stack
  - input: none
  - output: data on top of the stack

Lab 1-2 (10 points)

- Implement a Queue Program for a (non-Circular) Integer Queue of size 10
- 4 functions, using a global array of size 10
  - enqueue (int)
  - int dequeue ()
  - int queue_full ()
  - int queue_empty ()
- Test the Queue Program
  - Write the main function to exercise the 4 functions

Submit to the CyberCampus

- # Assignment 1
  - A single pdf file containing the source code and the result screen capture

Notes About Point Deductions

- Even if the code runs, points will be deducted for
  - inadequate comments
  - not following the spec
  - poor program structure
  - poor readability of the result screen
  - needless renaming of such standard terms as "push", "pop", "front", "rear", etc.

End of Class