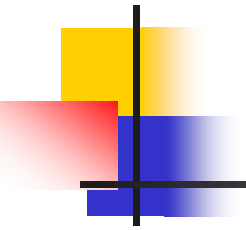
Program Patterns:
Stack and Queue

J.W. Choi

2024



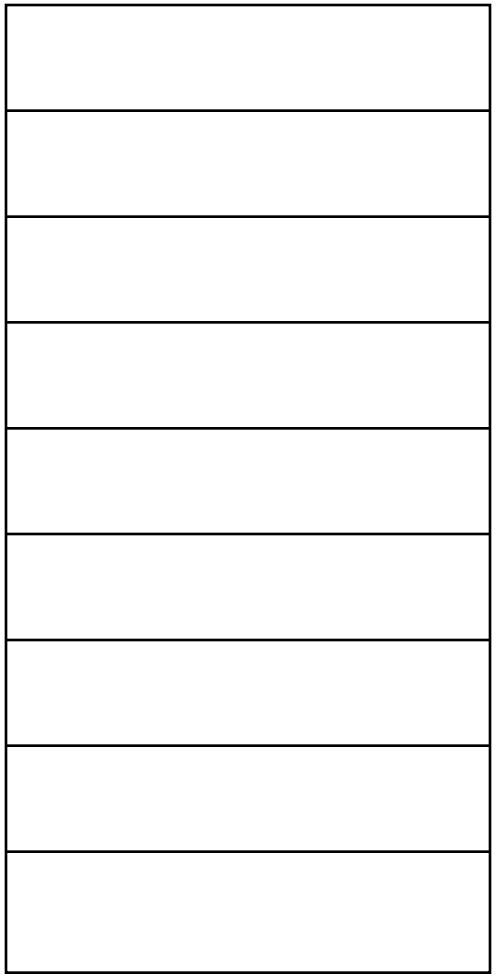
Lists

- Arrays
- Linked Lists
- Stacks
- Queues

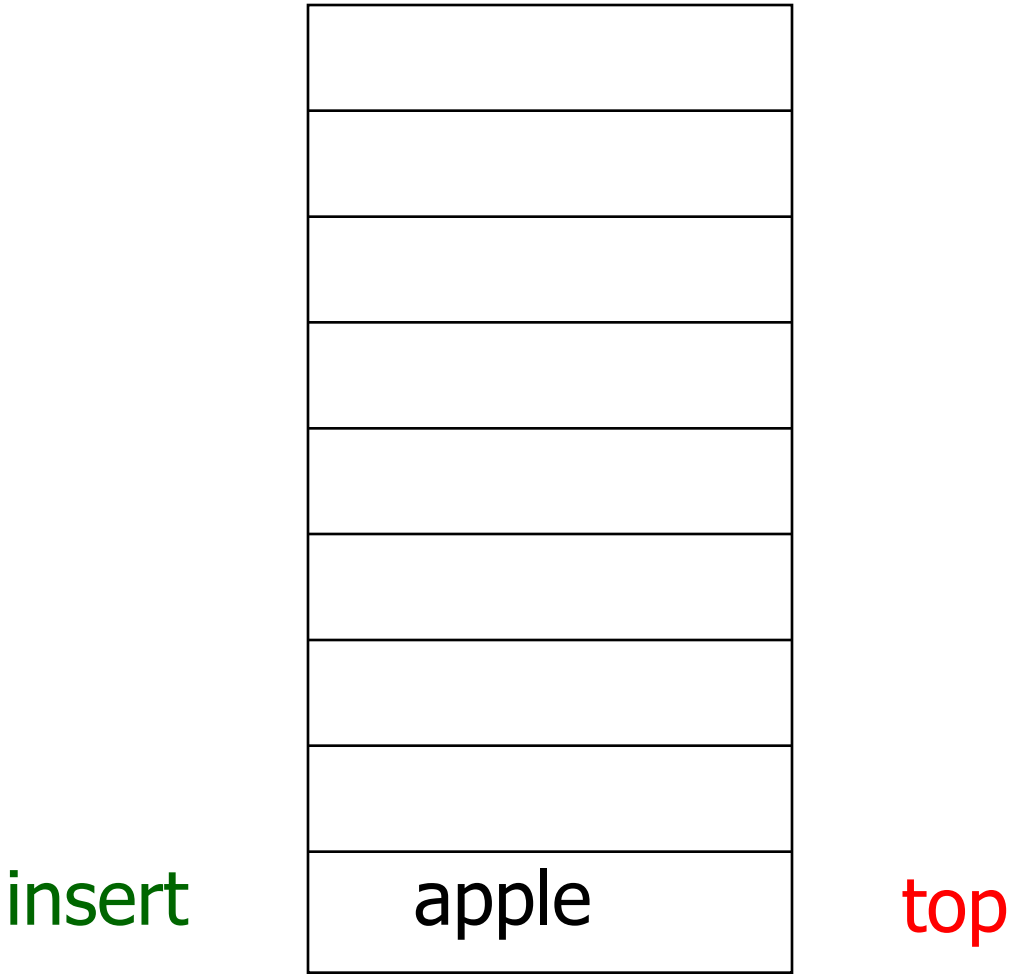
Stack: of Pancakes

Stack

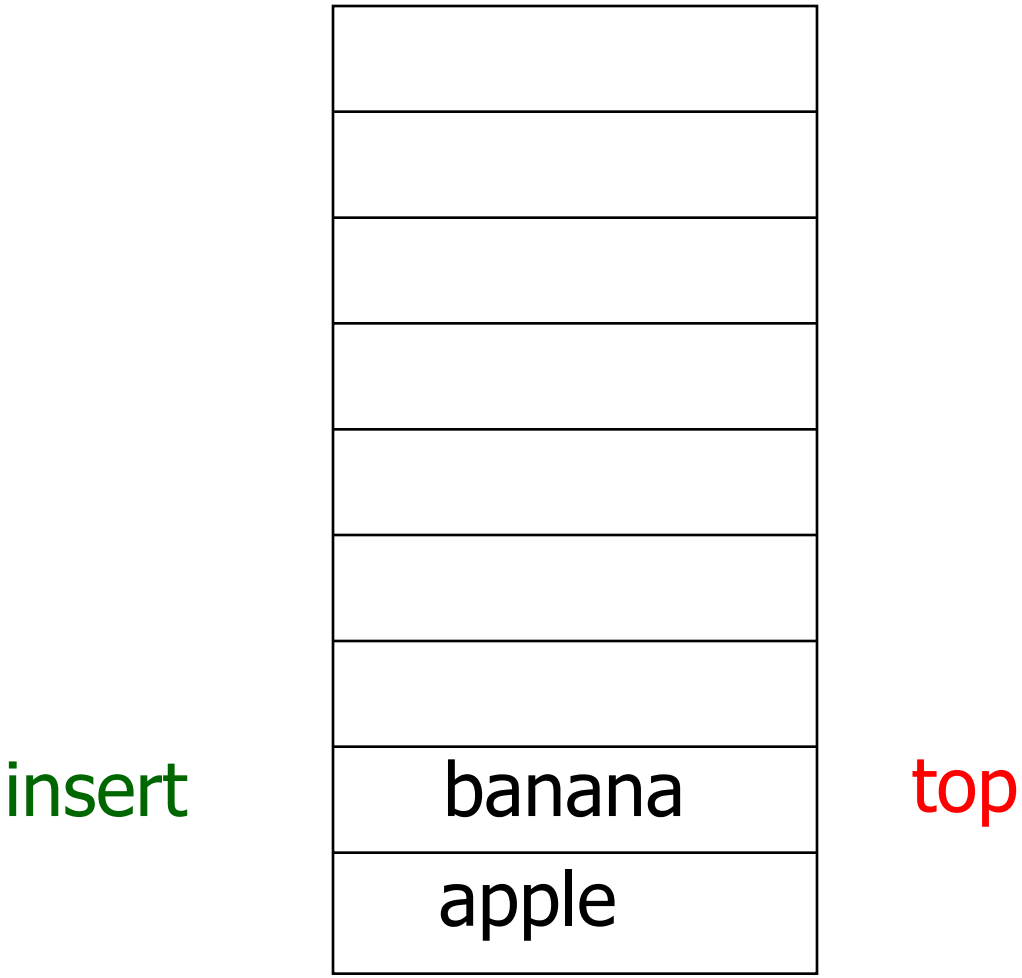
Stack: Concept



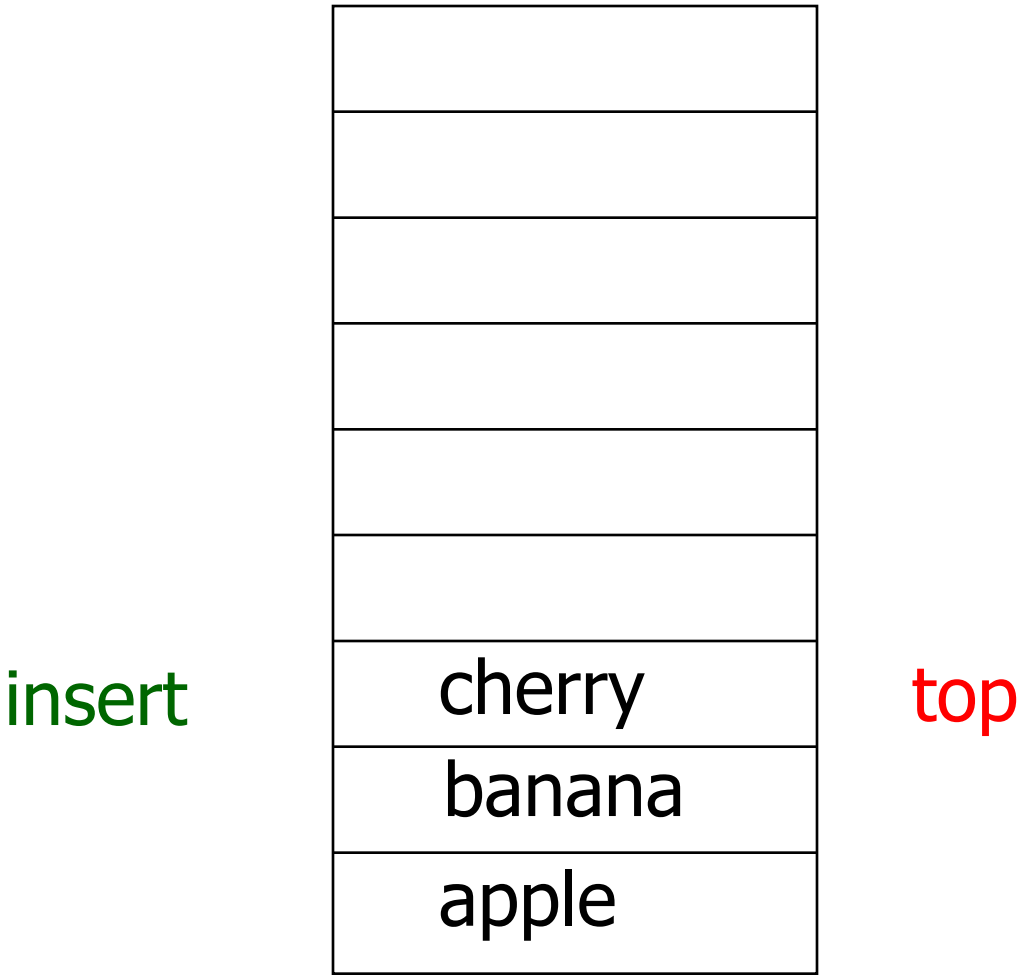
Stack: Concept



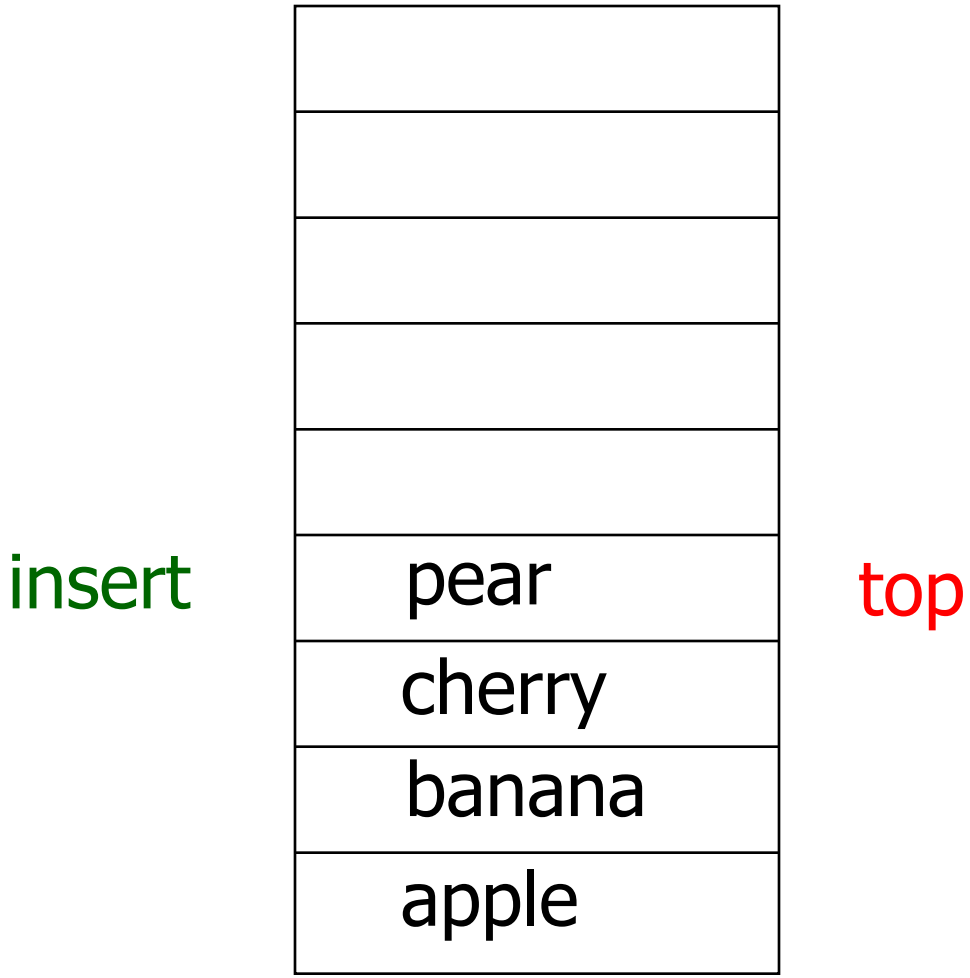
Stack: Concept



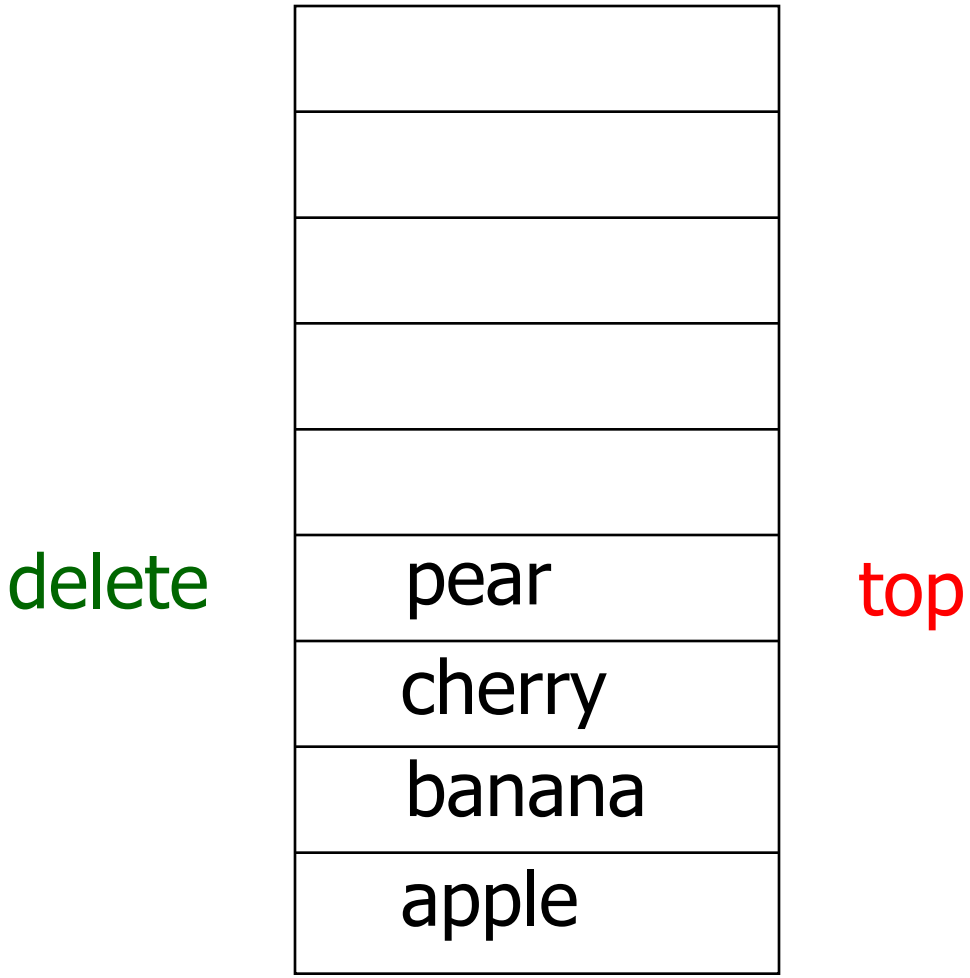
Stack: Concept



Stack: Concept

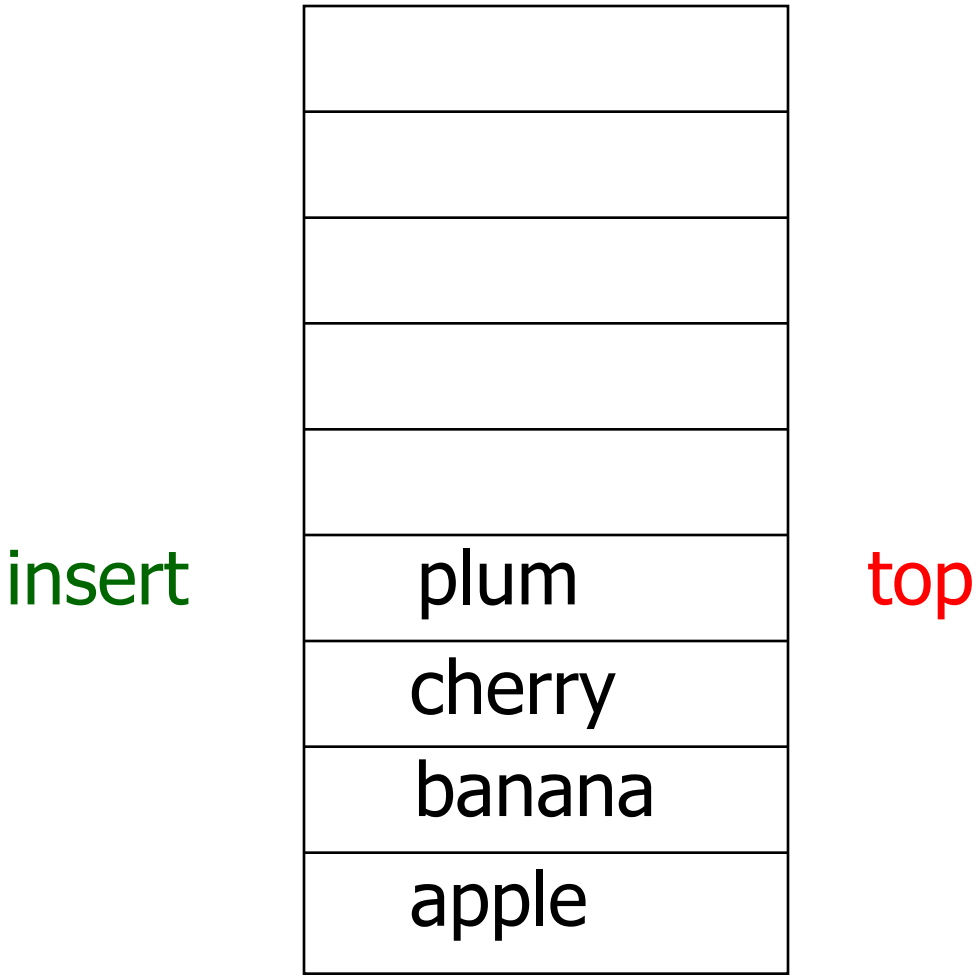


Stack: Concept



Stack: Concept

Stack: Concept

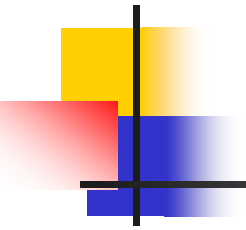


Stack: Properties

- Last in first out (LIFO) ordered list of data
- Data
  - primitive data
    - int, small int, float, double, char, var char, bit
  - complex data
    - structure
    - list of data
    - multimedia data
- Operations
  - insert, delete
  - check stack empty, check stack full

- insert -> push
- delete -> pop

Stack: Push & Pop



Stack: Applications

- System stack in OS
  - Activation records
    - nested function calls, including recursive function calls
  - Binary expression evaluation
  - Tree data structures (next semester)

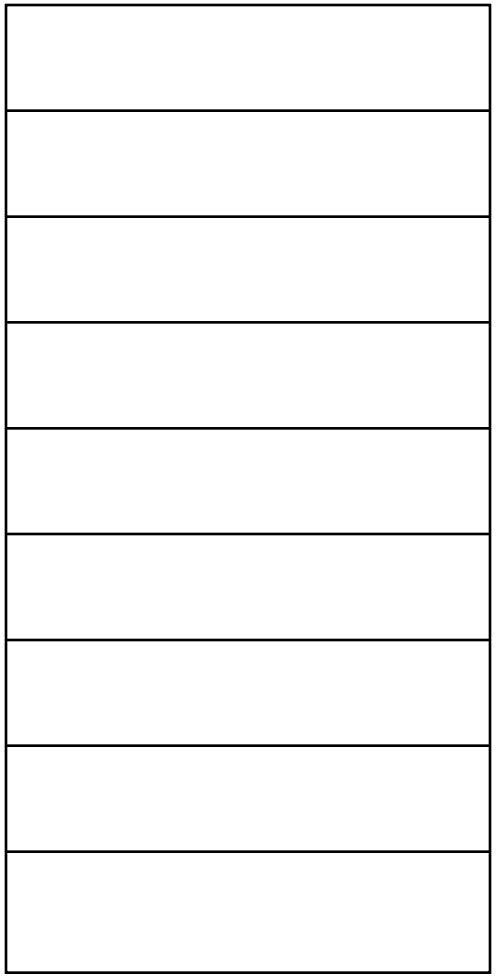
Stack: Applications

- Activation record
  - A structure containing function-call arguments, function's local variables, return address, etc.
  - Automatically created when a function is called.
  - Automatically deleted when the function is terminated.
- (Ex.) "main" calls function "capture",
  - "capture" calls function "analyze",
  - "analyze" calls function "store".

An Activation Record

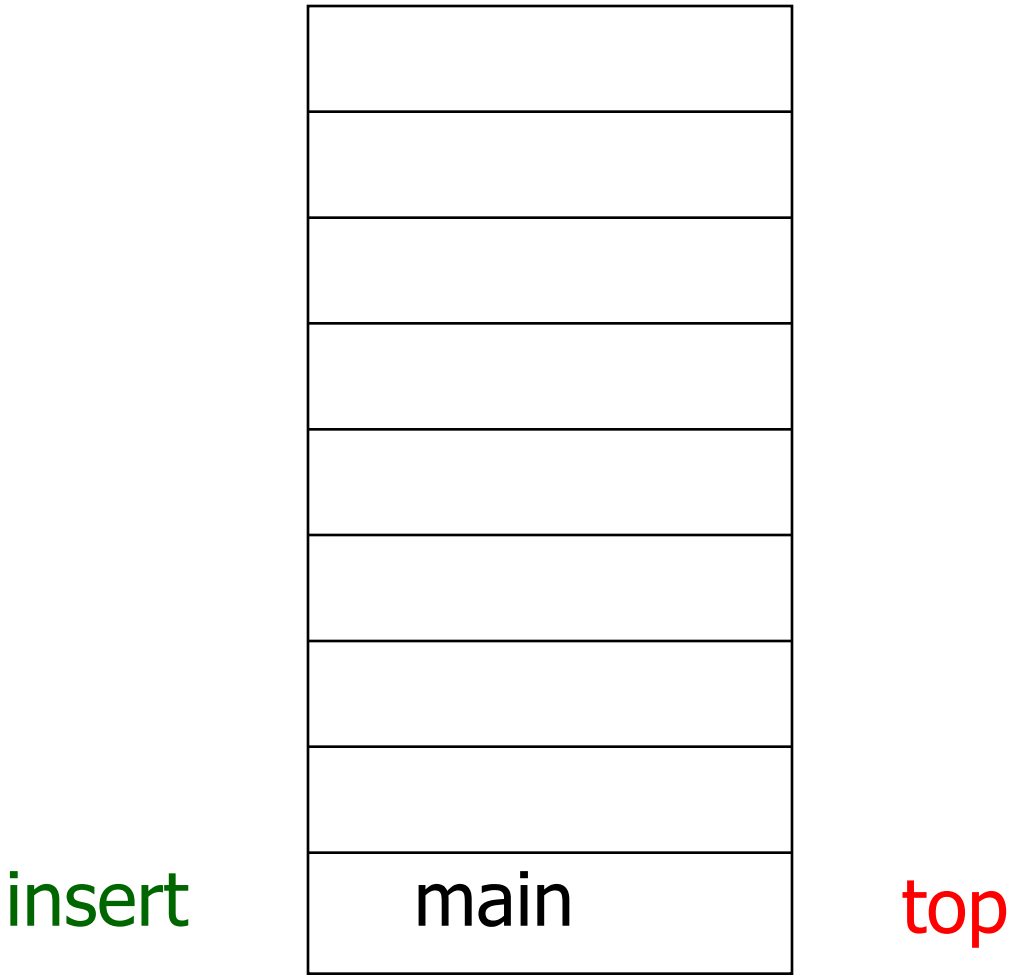
```
struct  activation_record {
    char  name[20];
    …
    local variables…
    …
    int     return_address;
};
```

Stack of Activation Records

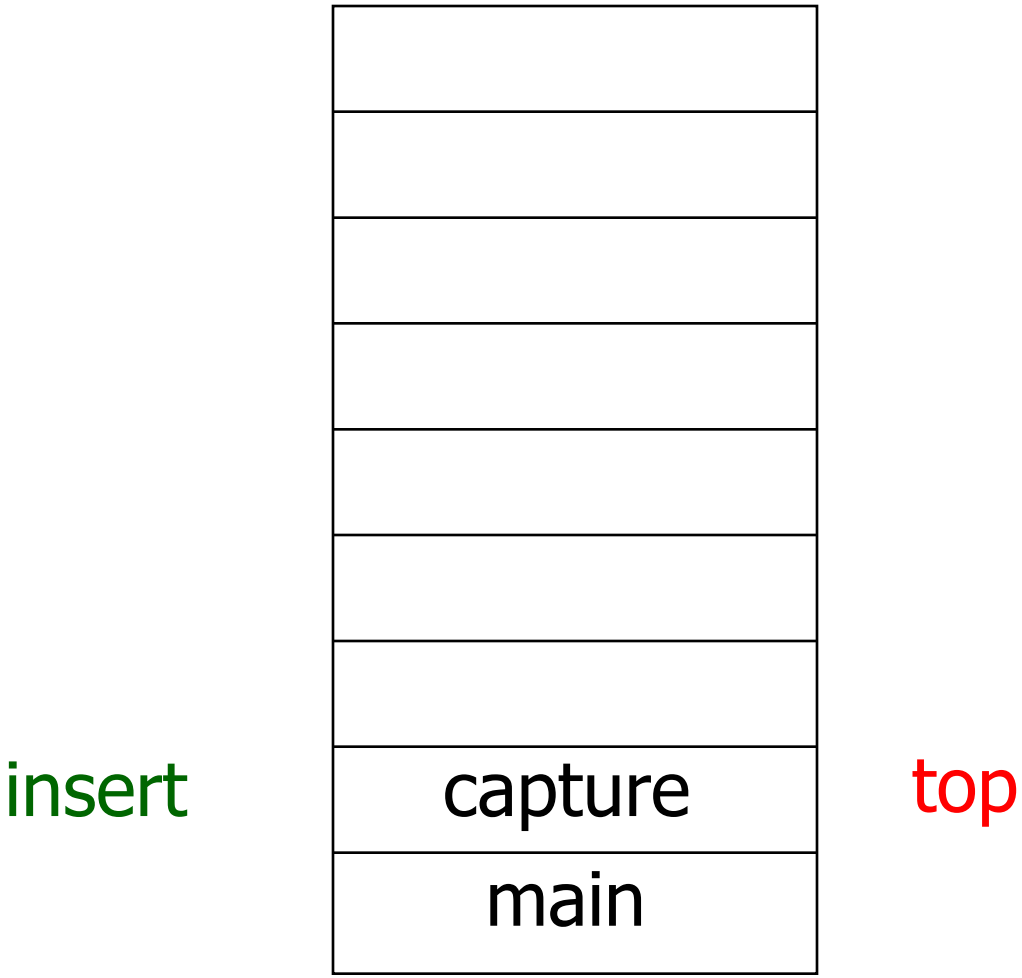


top

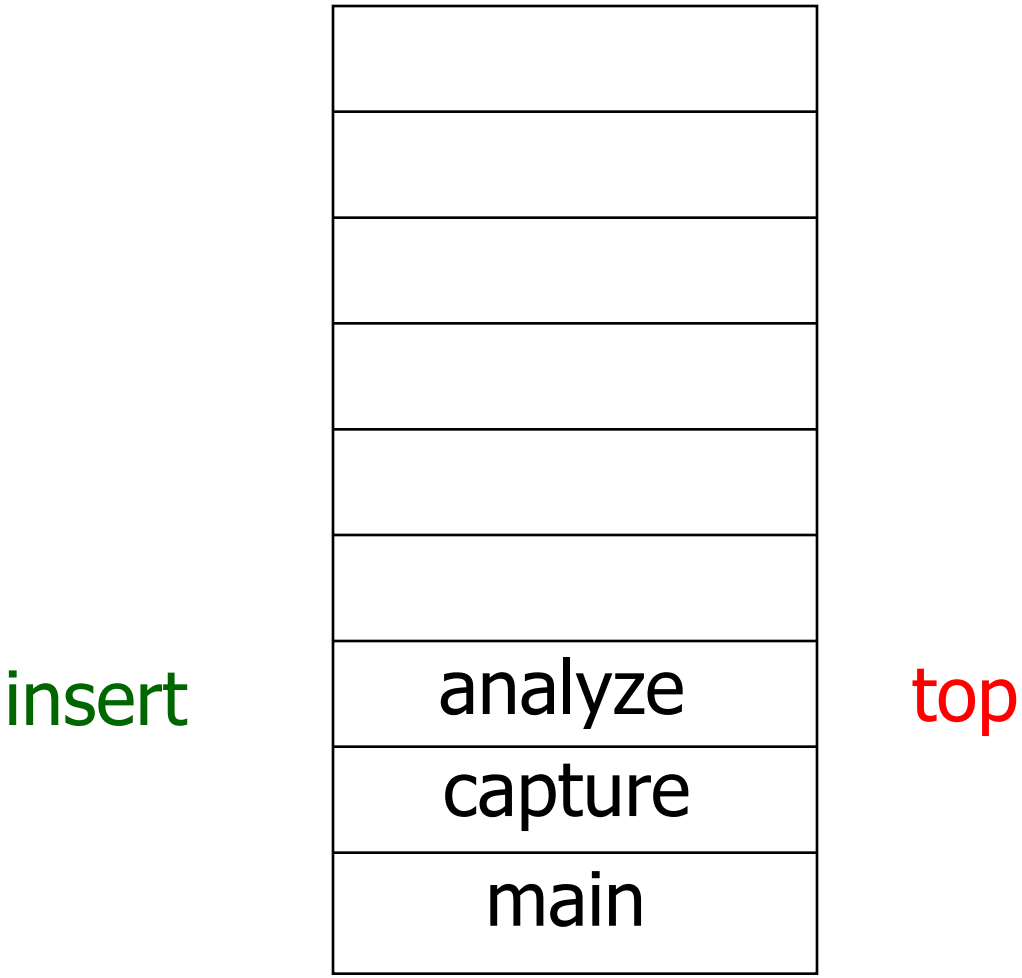
Stack of Activation Records



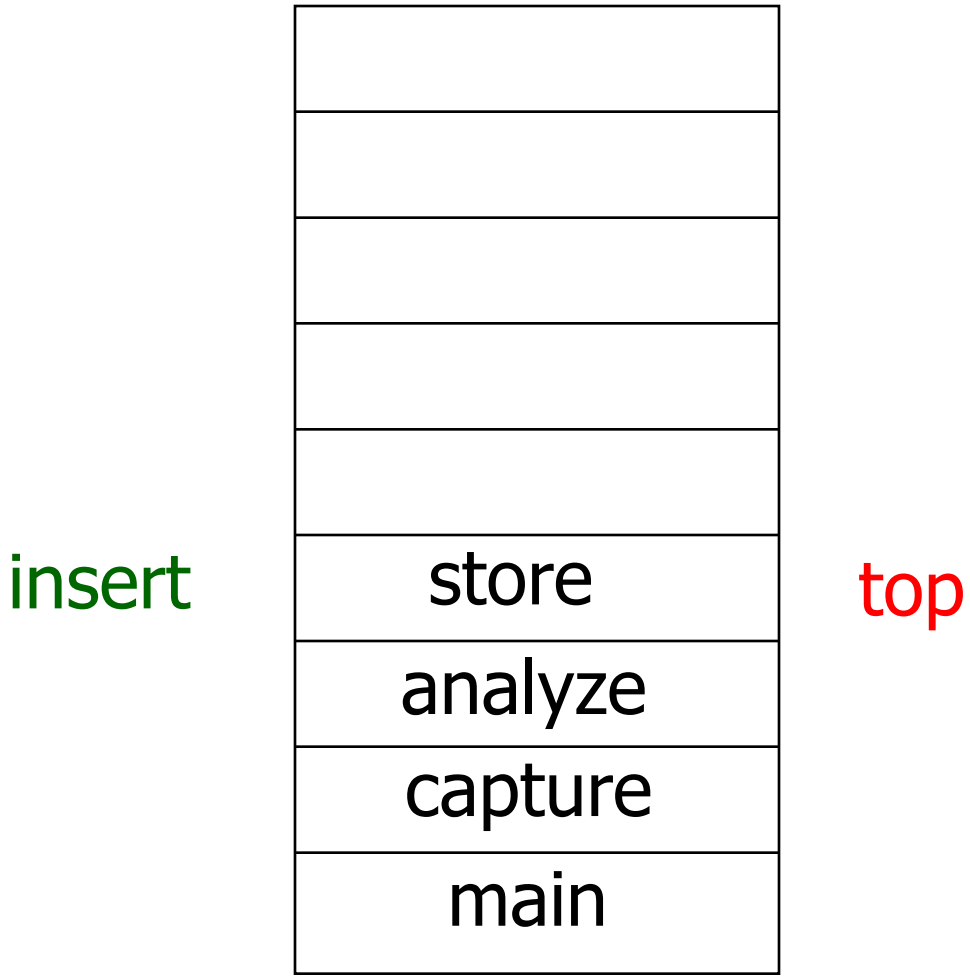
Stack of Activation Records



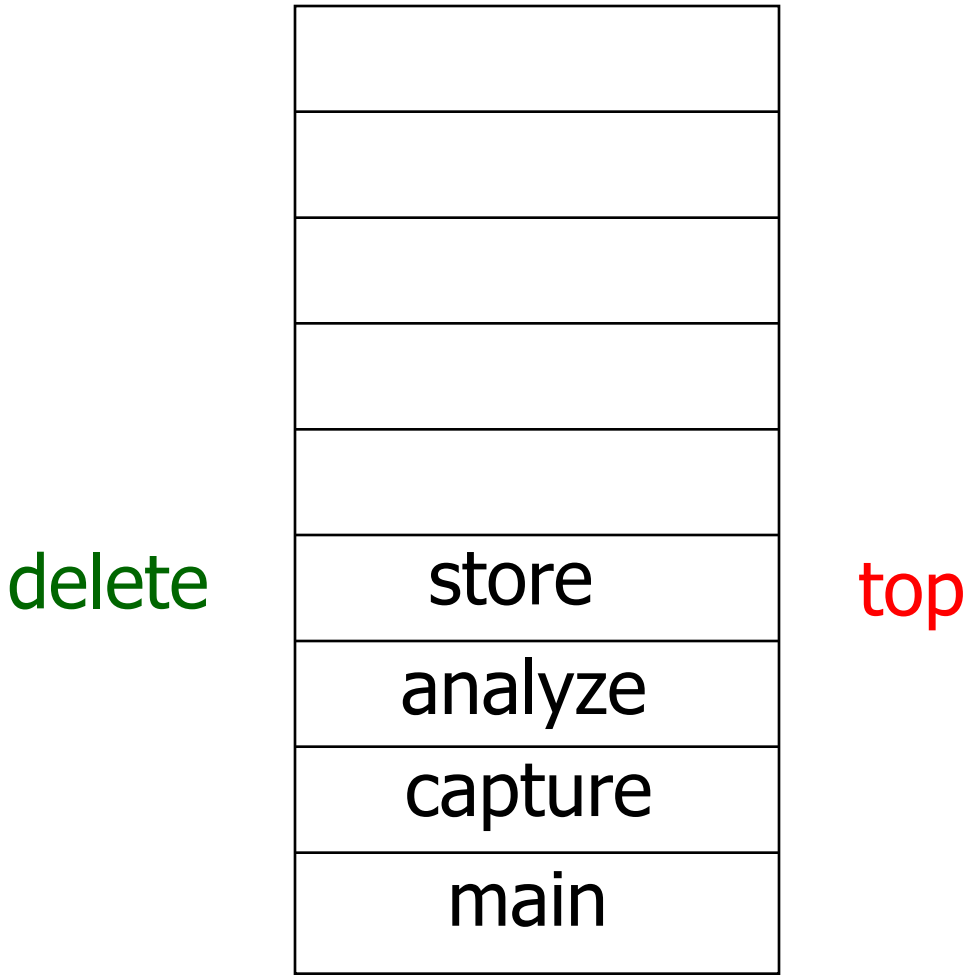
Stack of Activation Records



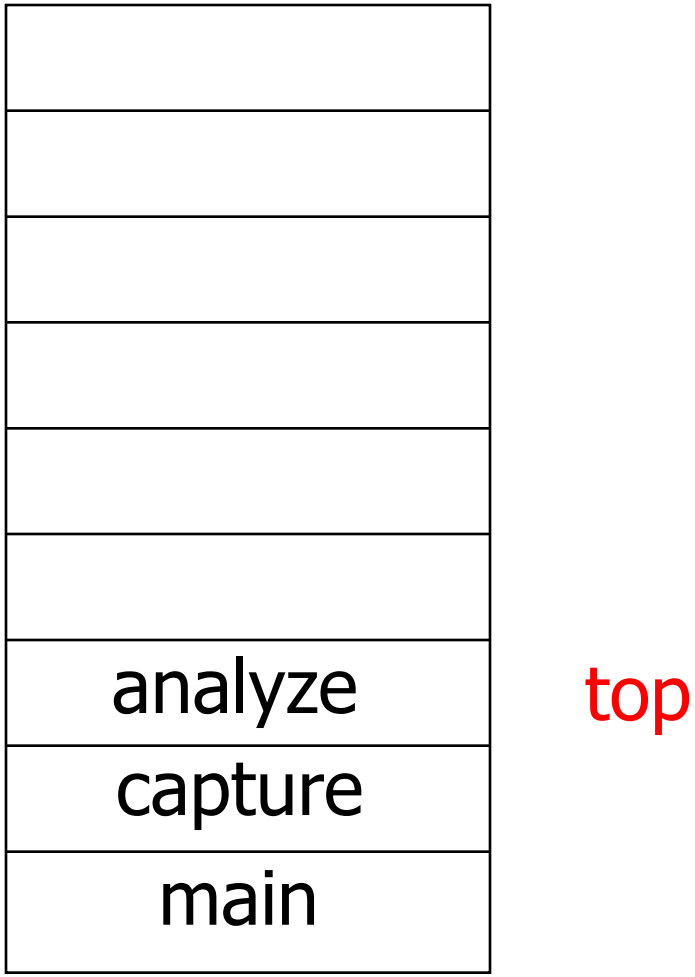
Stack of Activation Records

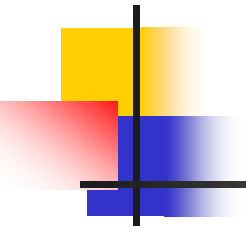


Stack of Activation Records



Stack of Activation Records





Stack in C

One-dimensional array

(datatype) stack[stack_size]

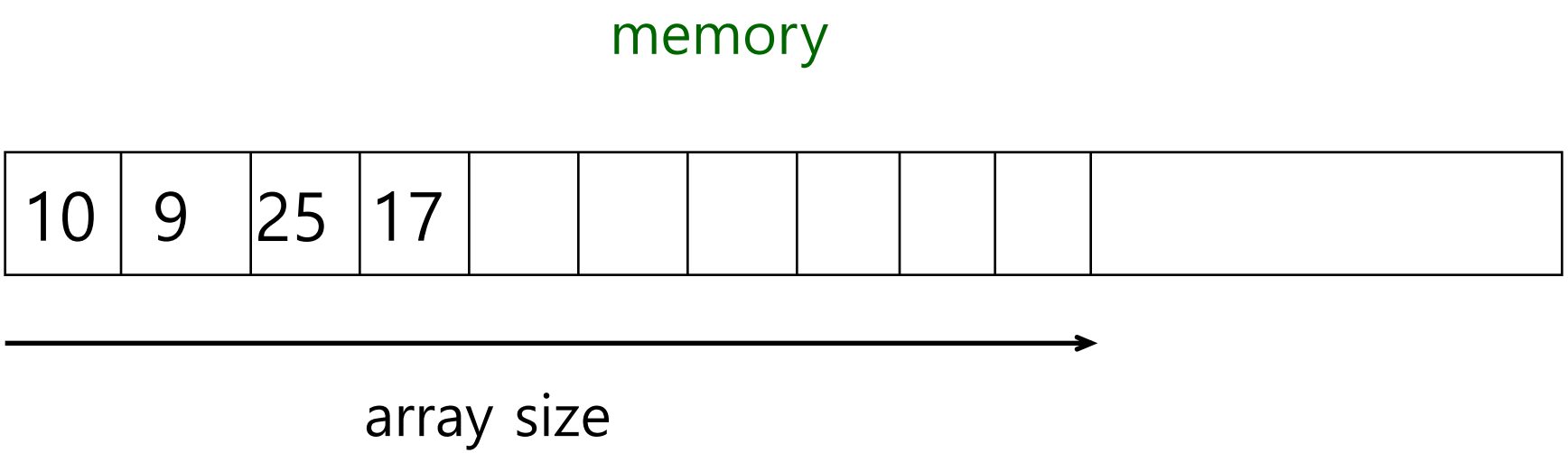
(ex.) char stack[100]

(ex.) struct activation_record stack[100]

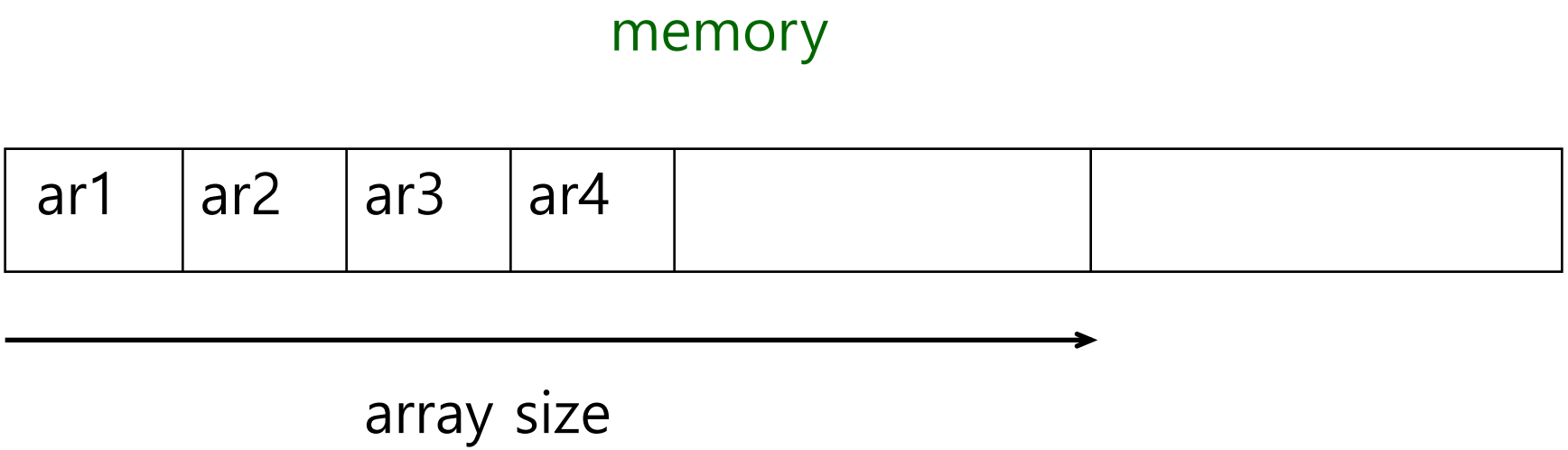
Variable “top”

initially top = -1 (empty stack)

Stack: Memory Allocation



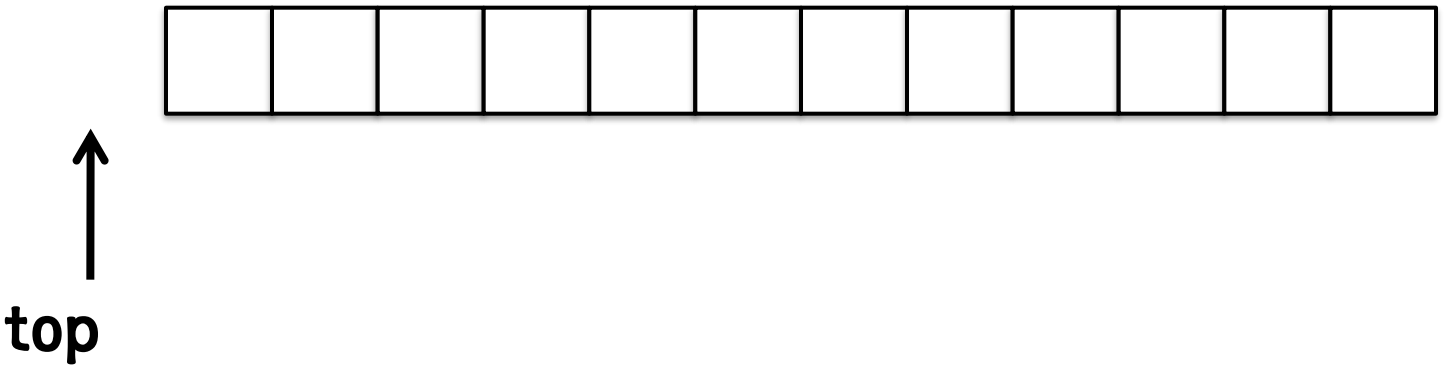
Stack: Memory Allocation



Using a Stack

Evaluating a postfix expression

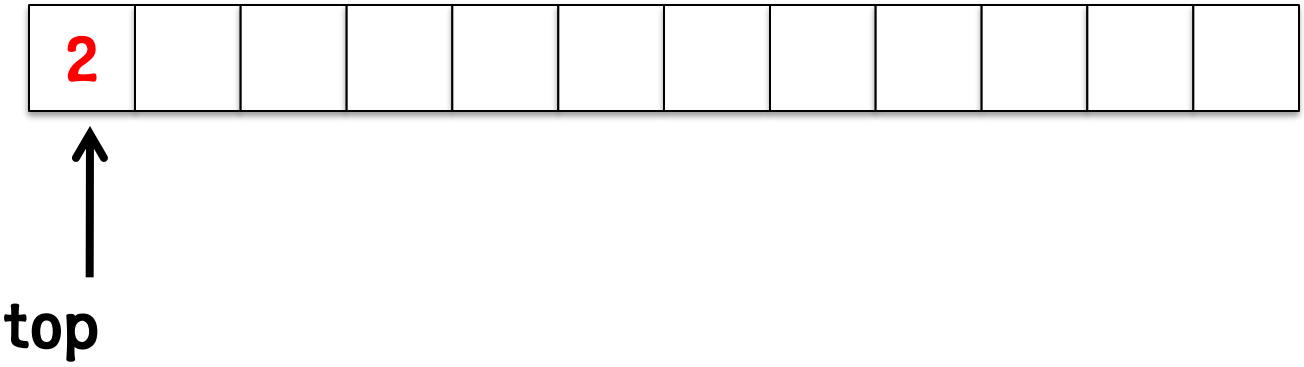
- Example postfix expression: 2 3 4 * 2 * +
- Create a stack of size larger than the length of the expression
- top (cursor) should be set to -1 initially



(1/8)

- 2 3 4 * 2 * +

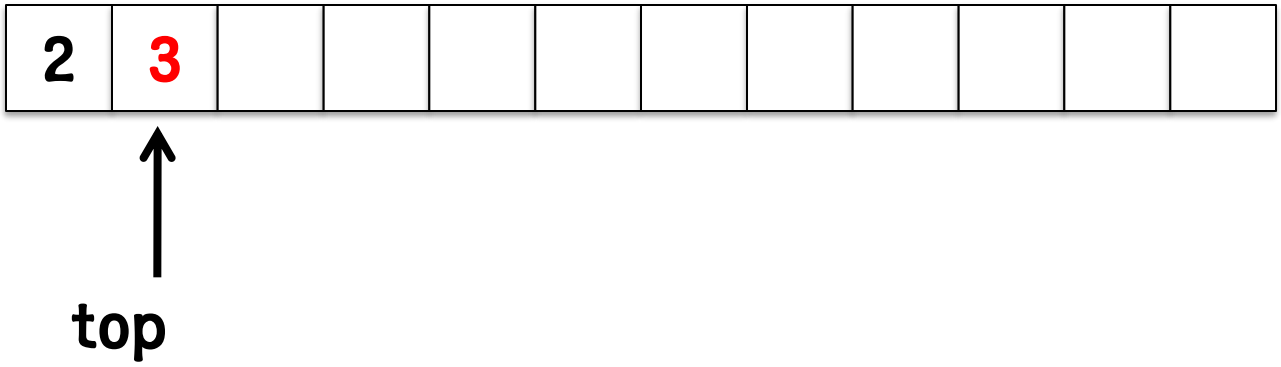
- Read the first item.
- If it is an integer
  - move top to the next position.
  - insert the integer in the stack at the top position.



(2/8)

- 2 3 4 * 2 * +

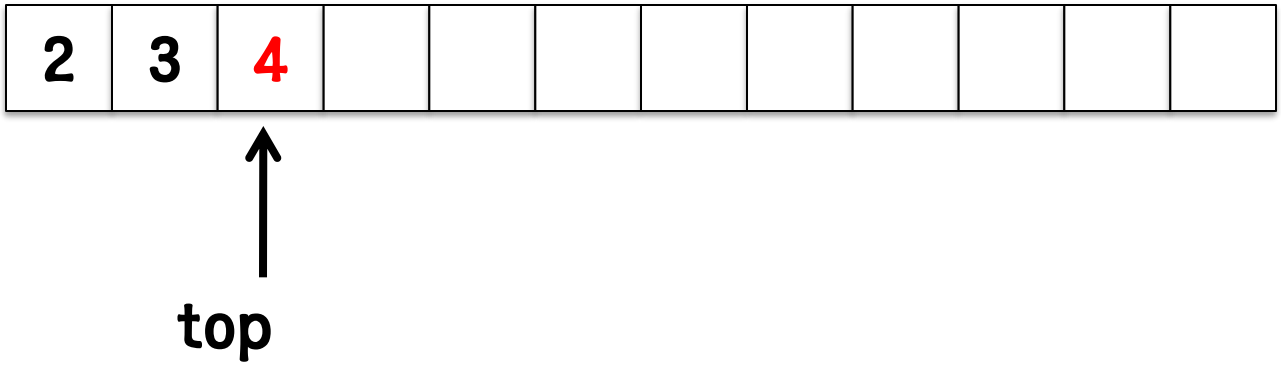
- Read the next item.
- If it is an integer
  - move top to the next position.
  - insert the integer in the stack at the top position.



(3/8)

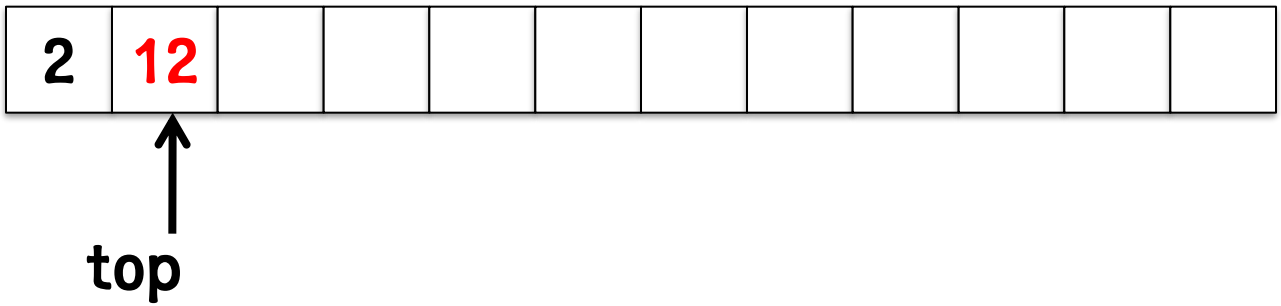
- 2 3 4 * 2 * +

- Read the next item.
- If it is an integer
  - move top to the next position.
  - insert the integer in the stack at the top position.



(4/8)

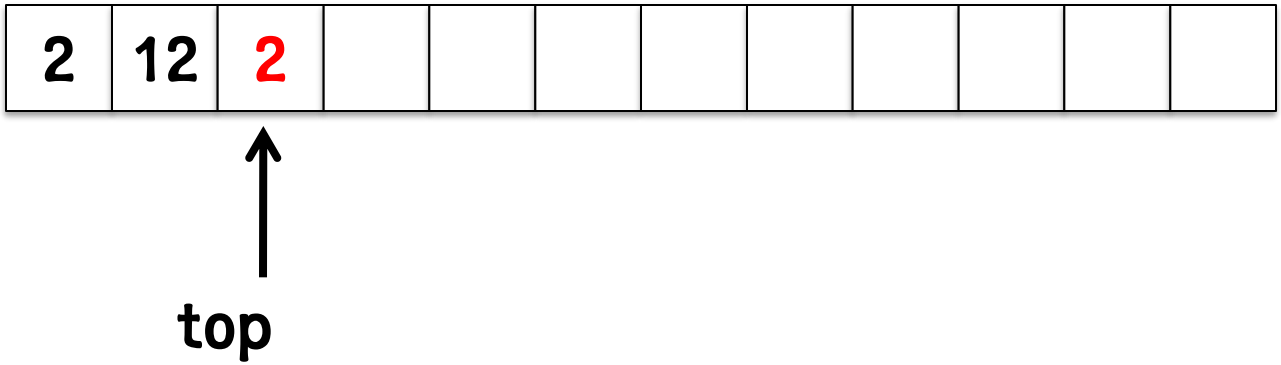
- 2 3 4 * 2 * +
- If it is an operator
  - get two integers from the stack (the one at the top position and the previous one), and apply the operation.
  - In this case, 3 * 4. The result is 12.
  - Insert the result in the stack and let top point to the newly inserted number.



(5/8)

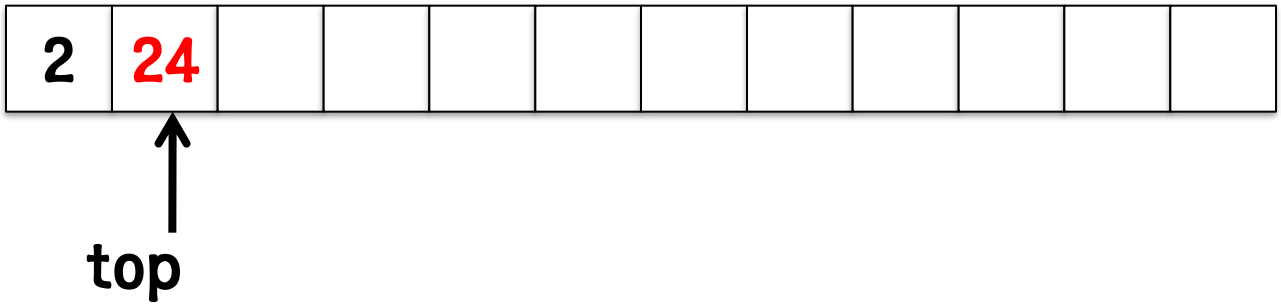
- 2 3 4 * 2 * +

- Read the next item.
- If it is an integer
  - move top to the next position.
  - insert the integer in the stack at the top position.



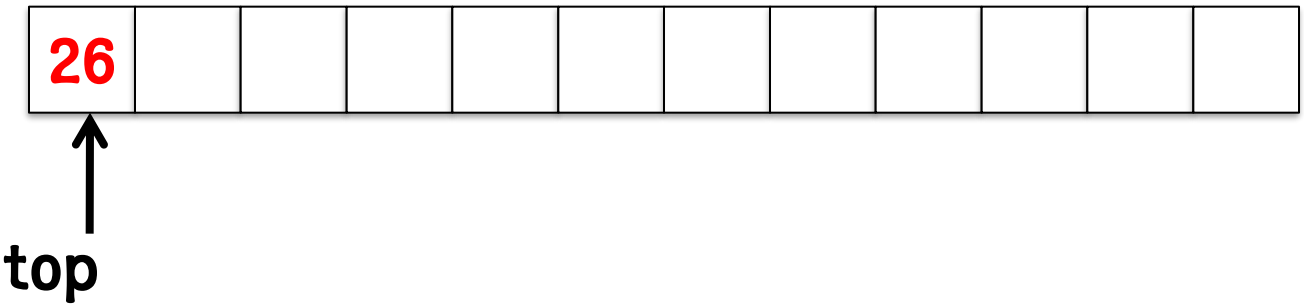
(6/8)

- 2 3 4 * 2 * +
- If it is an operator
  - get two integers from the stack (the one at the top position and the previous one), and apply the operation.
  - (In this case, 12 * 2. The result is 24.)
  - insert the result in the stack and let top point to the newly inserted number.



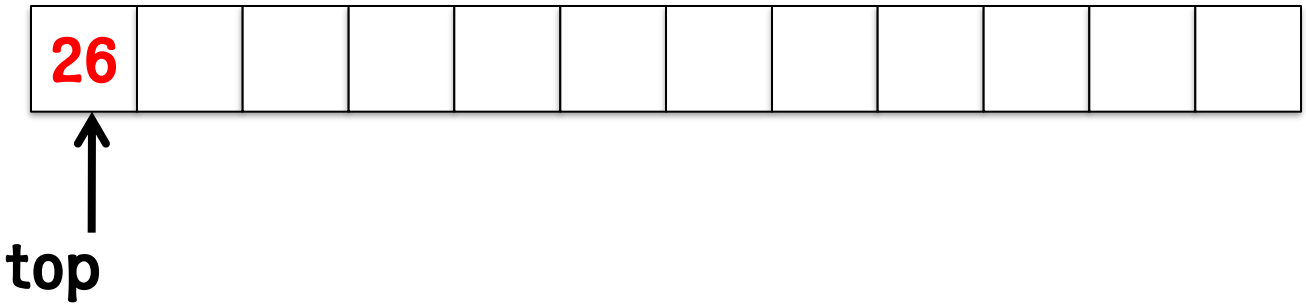
(7/8)

- 2 3 4 * 2 * +
- If it is an operator
  - get two integers from the stack (the one at the top position and the previous one), and apply the operation.
  - (In this case, 2 + 24. The result is 26.)
  - insert the result in the stack and let top point to the newly inserted number.



(8/8)

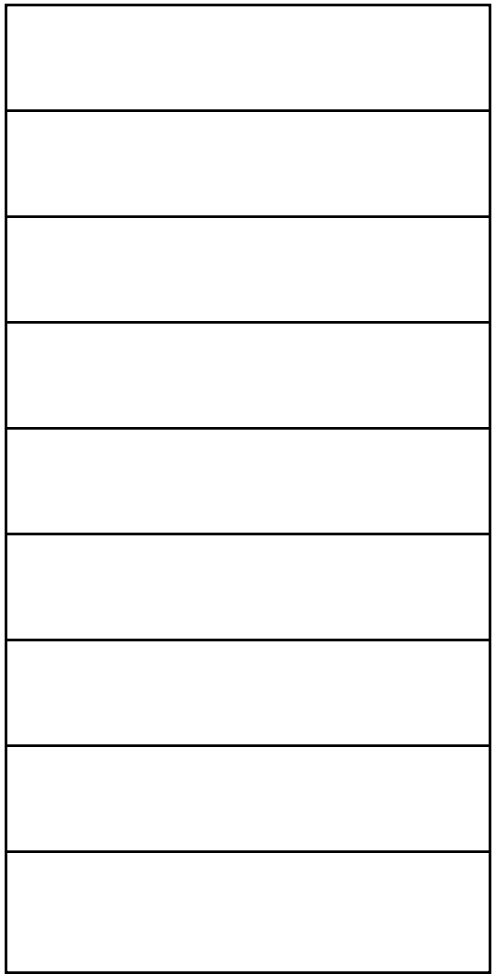
- When the expression is finished, a number will be left in the first position of the stack.
- It is the result of calculating the postfix expression.



Queue

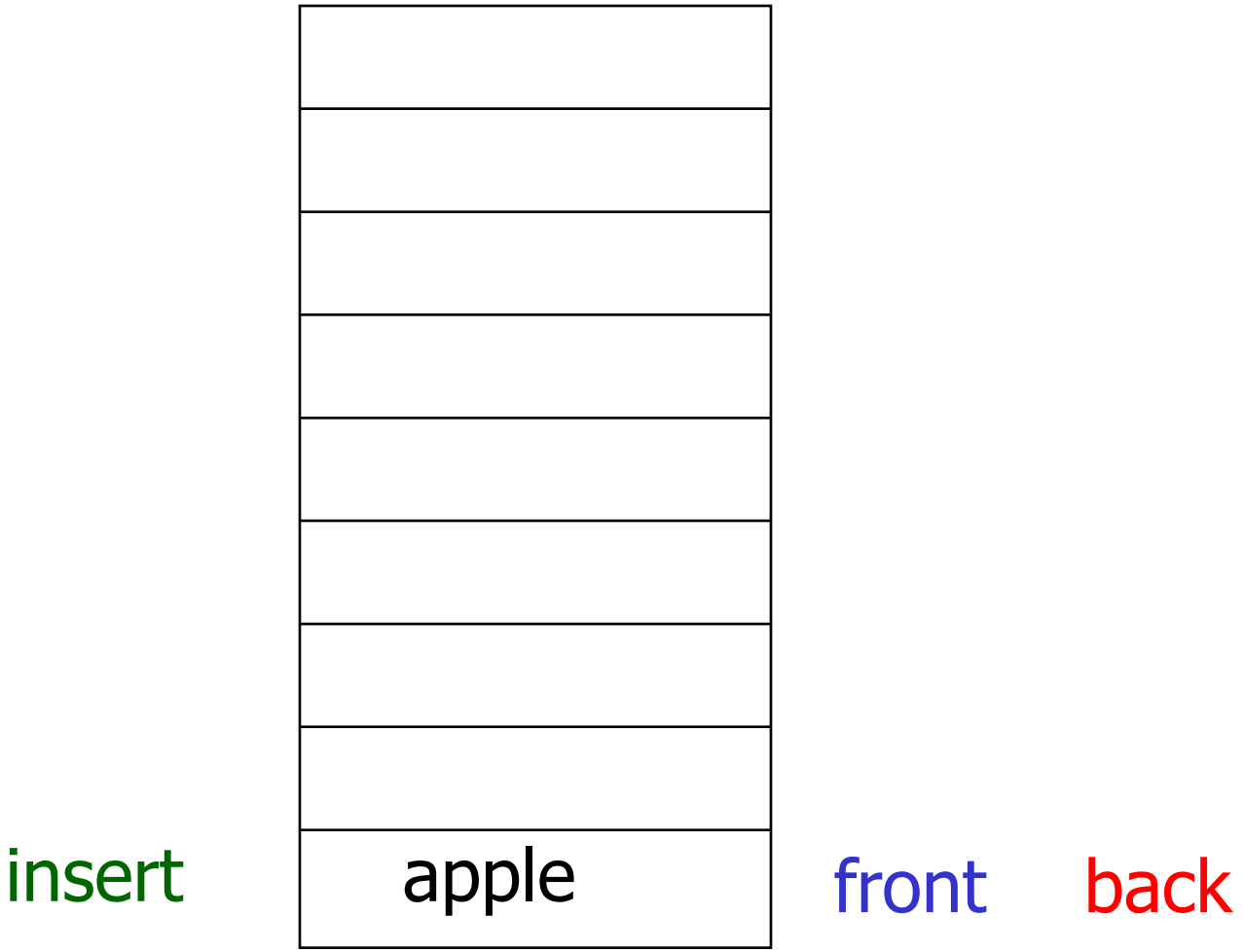
Queue

Queue: Concept

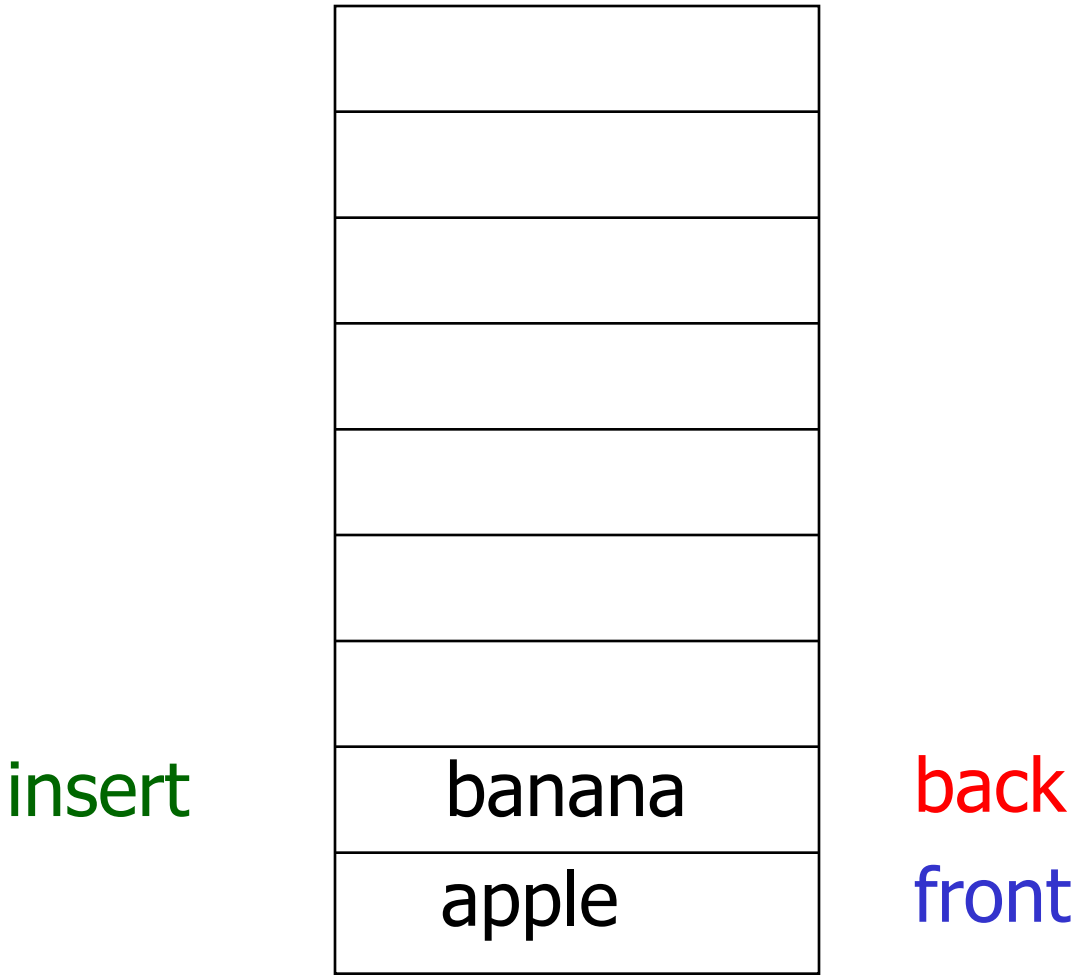


front back

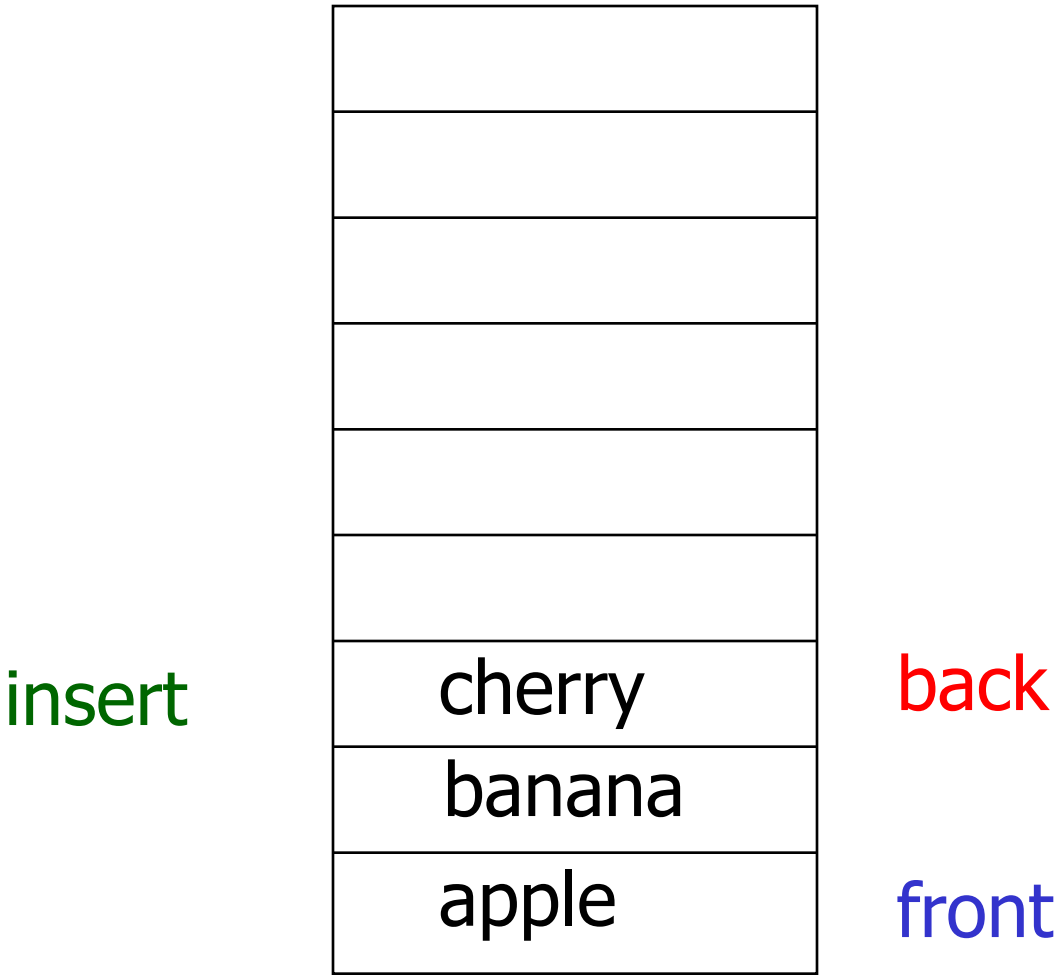
Queue: Concept



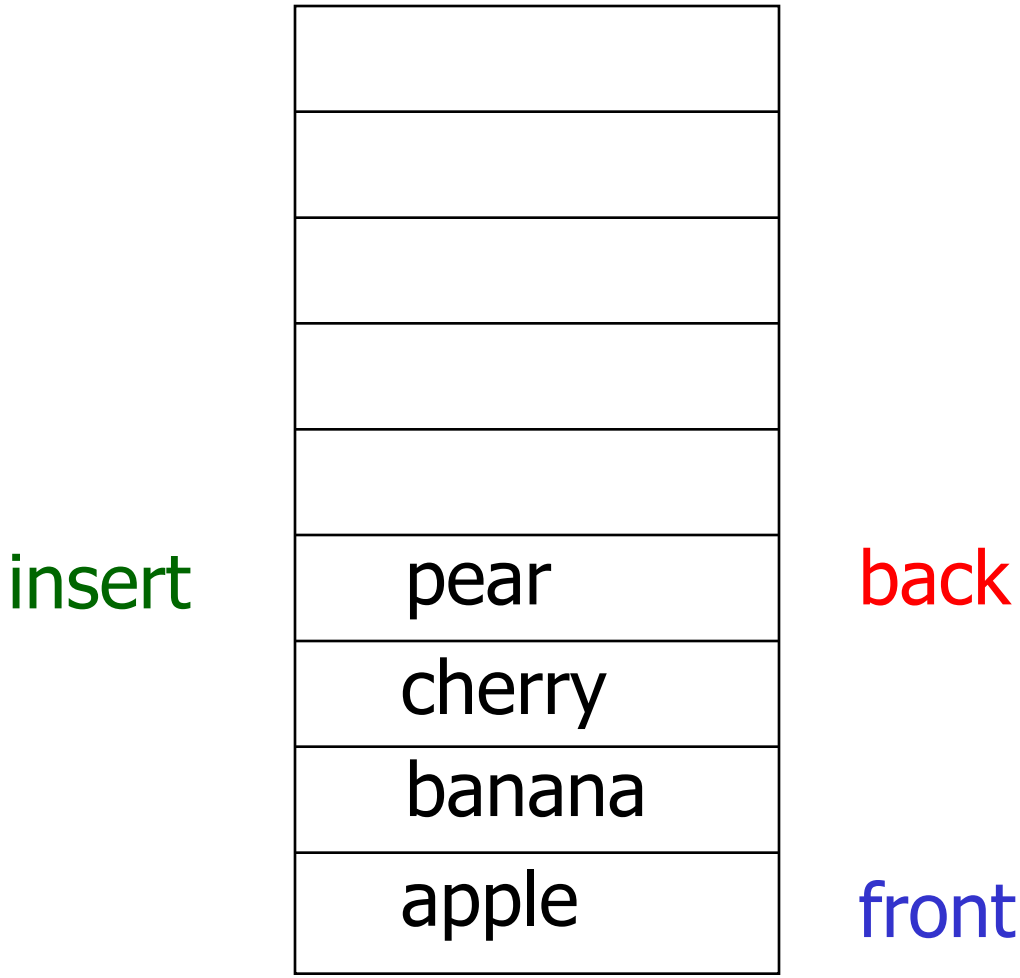
Queue: Concept



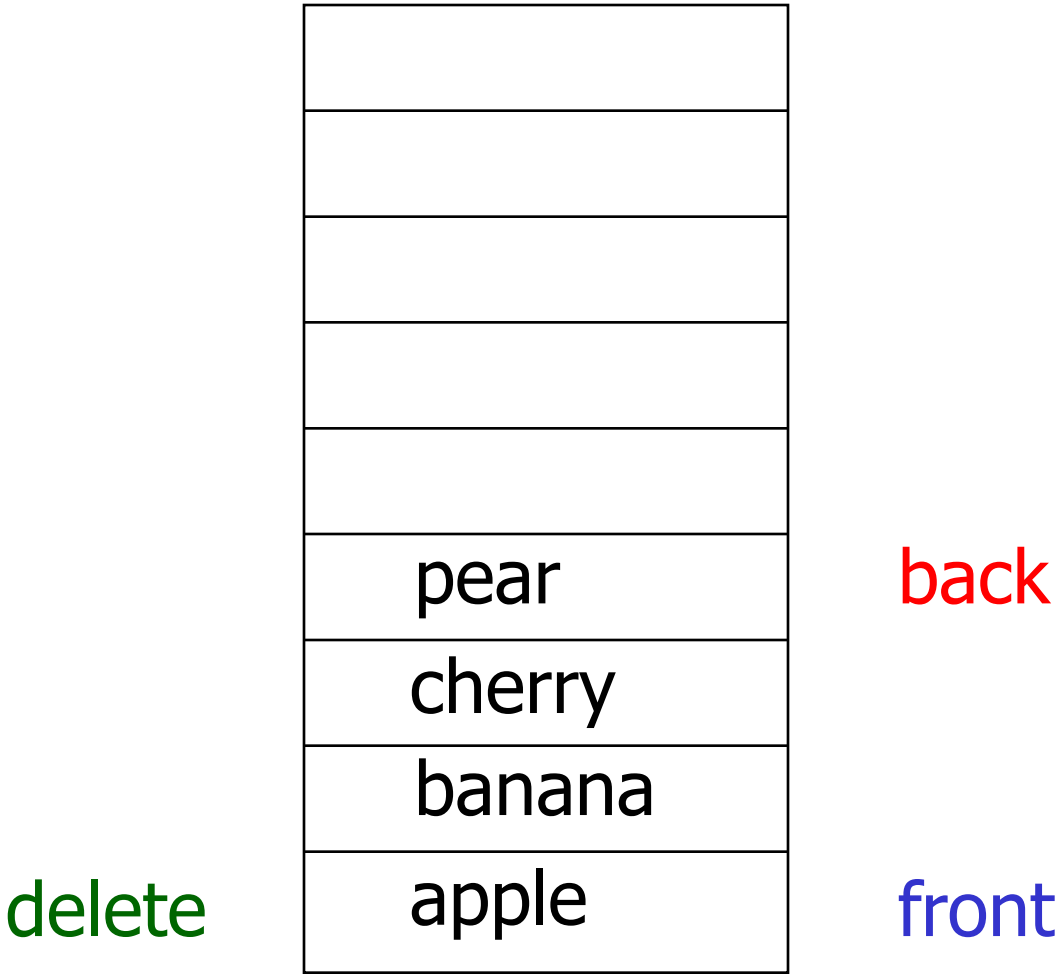
Queue: Concept



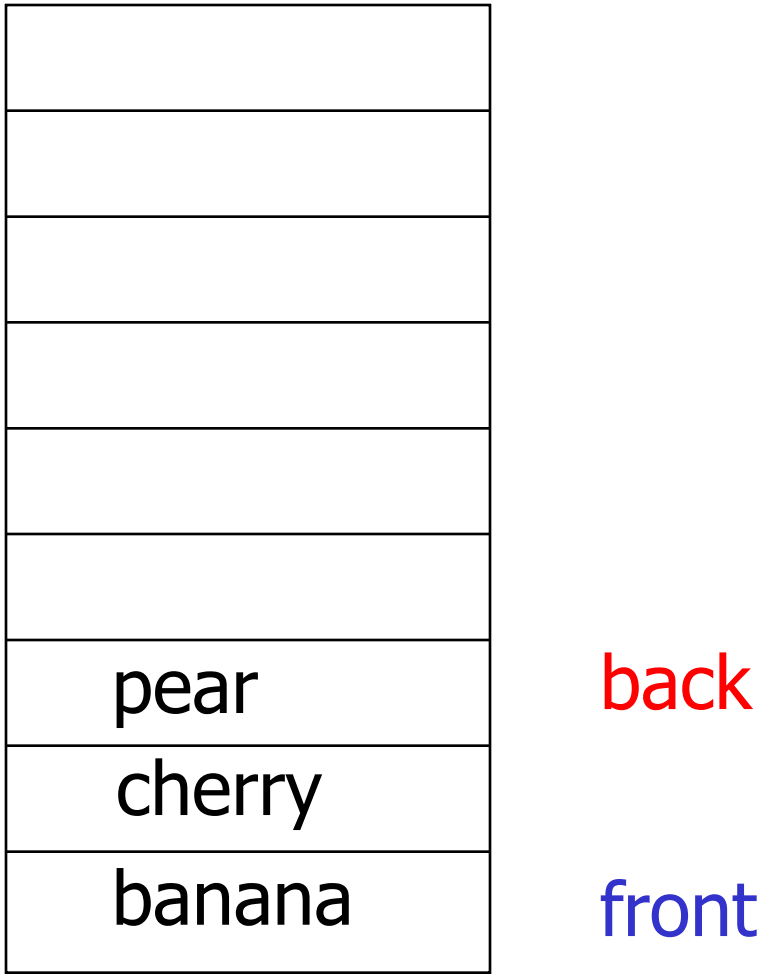
Queue: Concept



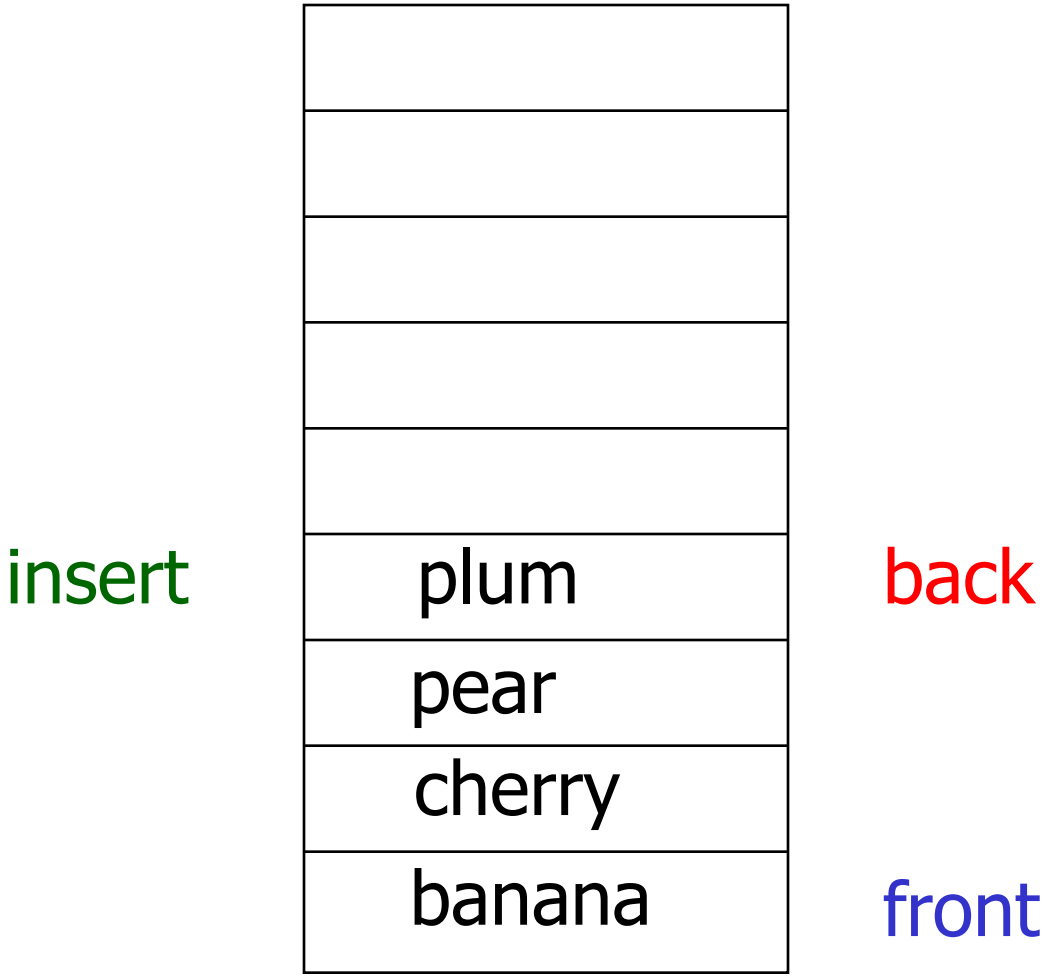
Queue: Concept



Queue: Concept

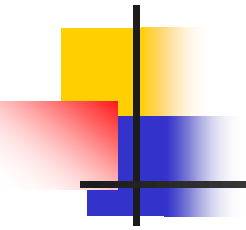


Queue: Concept



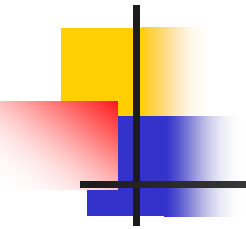
Queue: Properties

- First in first out (FIFO) ordered list of data
- Data
  - Primitive data
    - int, small int, float, double, char, var char, bit
  - Complex data
    - structure
    - list of data
    - multimedia data
- Operations
  - insert, delete
  - check queue empty, check queue full
- insert -> append, add
- Append to the back
- Delete from the Front



Queue: Applications

- Communication
  - Message queue
- OS scheduling
  - Process queue
- …



Queue in C

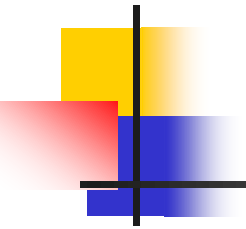
One-dimensional array

(datatype) queue[queue_size]

Variable “front”

Variable “back”

initially front = back = -1 (empty queue)



Lab 1

- Implement the stack using an array. (push, pop, isEmpty, isFull)
- Notes
  - Create a global array (size=100)
  - Create a global integer variable top (initial value -1)
  - Create 2 tests that will exercise all 3 functions.

Lab 1 : pseudo code

bool isFull()

if(스택의 원소 수==size) return TRUE;

else return FALSE;

bool isEmpty()

if(스택의 원소 수==0) return TRUE;

else return FALSE;

void push(item)

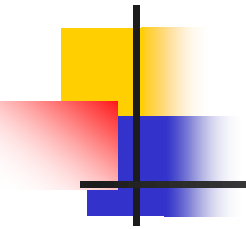
if(isFull()) return OVERFLOW_ERROR;

else 스택의 맨 위에 item 추가;

int pop()

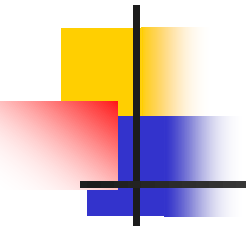
if(isEmpty()) return UNDERFLOW_ERROR;

else 스택의 맨 위의 원소를 제거하여 반환;



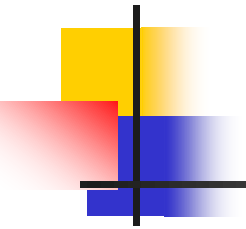
Lab 2

- Implement the stack using a linked list. (push, pop, isEmpty)
- Notes
  - Each node has an integer key and next struct pointer.
  - Create each node using malloc.
  - When a node is deleted, free the memory.
  - Create 2 tests that will exercise all 4 functions.



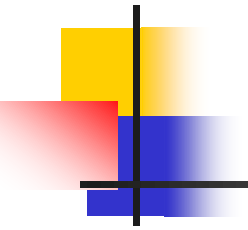
Lab 3

- Implement the Queue using an array. (insert, delete, isEmpty, isFull)
- Notes
  - Create 2 tests that will exercise all 3 functions.



Lab 4

- Implement the Queue using an Linked List. (insert, delete, isEmpty, isFull)
- Notes
  - Each node has an integer key and next struct pointer.
  - Create each node using malloc.
  - When a node is deleted, free the memory.
  - Create 2 tests that will exercise all 3 functions.



End of Class