Stacks

- Stack = LIFO (Last In First Out) lists
- Insertion and deletion can be performed in one position, at the end of the list (top).
- Insert ⇔ push
- Delete ⇔ pop
- top: examines the element at the top
- Only the top element is accessible.

Stacks: History

- 1947: Alan Turing

  developed a stack called Reversion Storage (used for subroutine calls, ACE computer)

- 1956: Newell, Simon and Shaw (Rand Corp.)

  IPL language, stack in linked form

- 1957: K. Samuelson and F. Bauer (Germany)

  filled a patent

- 1957: Charles Hamblin (Australia)
- 1958: John McCarthy

  LISP uses a built-in stack

Implementation of Stacks

- Any list implementation works
- list and vector support stack operations
- In some cases it is useful to design faster special-purpose implementations:
  - Linked-list implementation
  - Array implementation

Stacks: Linked List Implementation

- Uses a singly linked list
- push: insert at the front of the list
- pop: delete the element at the front of the list
- top: examines the element at the front of the list

Stacks: Array Implementation

- More popular implementation using vector
- push: push_back
- pop: pop_back
- top: back

Applications of Stacks: Balancing Symbols

- Check if every opening symbol in a string corresponds to a closing symbol.
- Examples: [(….)] - legal

  [(....]) - wrong
- Algorithm: (using a stack)

1. Make an empty stack.
2. Read character until EOF
   a. if (opening symbol) then push it on the stack.
   b. if (closing symbol) then

      if (stack empty) error;

      else pop the stack.

      if (popped symbol != corresponding symbol) then error.
3. if (stack not empty) error

Applications of Stacks: Postfix Expressions

- Postfix notation:
  6 5 2 3 + 8 * + 3 + *
  used to evaluate: ((2+3)*8 + 5 + 3)*6
- Why postfix notation? Avoids explicit precedence rules.
- Evaluation using a stack:

Symbol read		+	8	*	+	3	+	*
	3		8					
	2	5	5	40		3		
	5	5	5	5	45	45	48	
T(N) = ?	6	6	6	6	6	6	6	288

Applications of Stacks: Infix to Postfix Conversion

- Infix expression: a + b*c + (d*e + f)*g
- Postfix expression: a b c * + d e * f + g * +
- Rules:
  - When an operand is read it is placed onto the output
  - Operators and left parentheses are placed on the stack
  - If read a right parenthesis then pop until we encounter a left parenthesis (no output).
  - If read +, * or (then pop entries from stack until we find an entry of lower priority. Exception: never remove a “(“ except when processing a “)”.
  - Priority form lowest to highest: + , *, (
  - If end of input then pop the stack until empty.

Example

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

Symb. read	Stack	Output
a + b	+	a b
* c	* +	a b c
+	+	a b c * +
(d	(+	a b c * + d
* e	* (+	a b c * + d e
+ f	+ (+	a b c * + d e * f
)	+	a b c * + d e * f +
* g	* +	a b c * + d e * f + g
		a b c * + d e * f + g * +

Applications of Stacks: Function Calls

- When calling a function all important information (register values, return address etc.) is saved on the stack then the control is transferred to the new function.
- Information saved => activation record or stack frame
- When function returns the information is restored from the stack
- Problems: Running out of stack space

Queues

- Queue = FIFO (First In First Out) lists
- Insert ⇔ enqueue

  done at the end of the list (called rear)
- Delete ⇔ dequeue

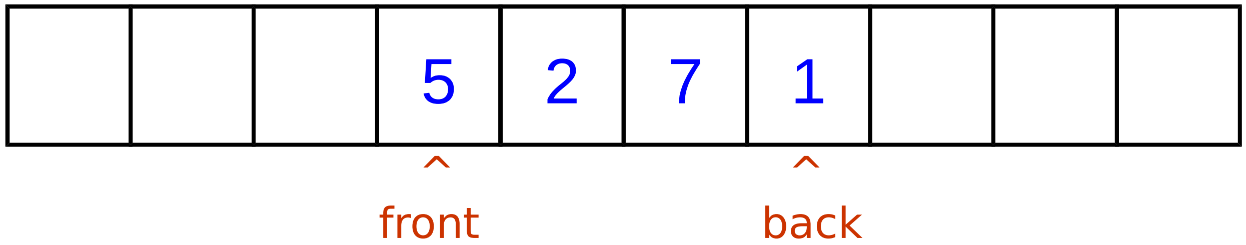
  done at the start of the list (called front)
- Can be implemented using linked lists or arrays.
- Every operation in $O(1)$.

Array Implementation

- Use an array of fixed size in a circular fashion.
- Two variables keep track of the front and rear:

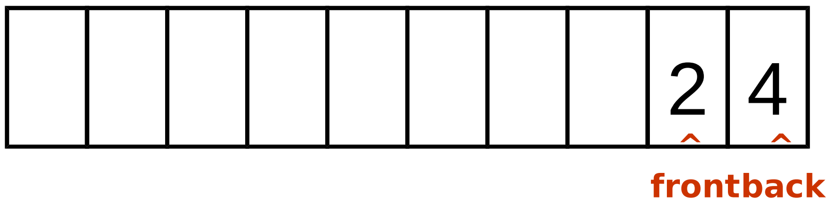
  front = index of the front element

  back = index of the back element
- We keep track of the number of elements in the queue => currentSize

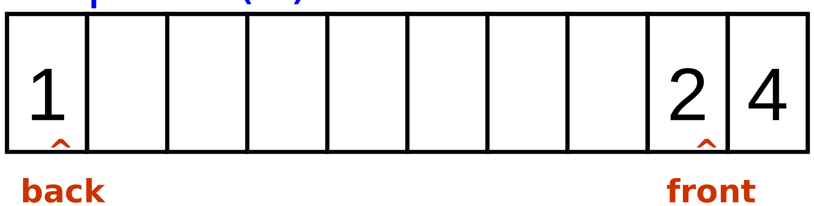


Example

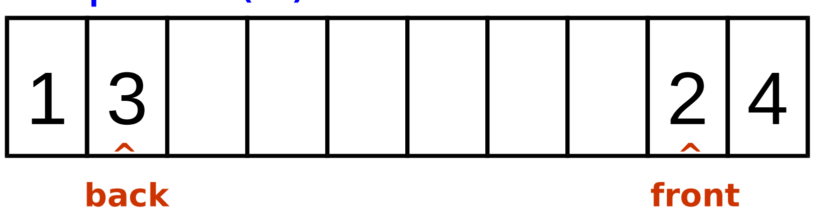
Initial state



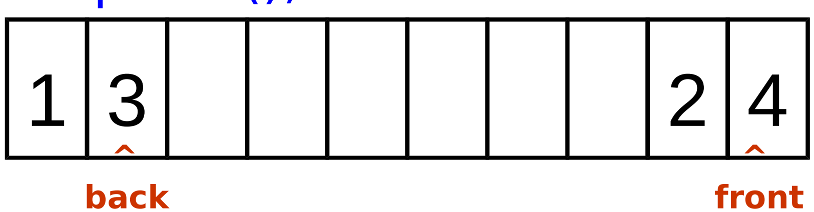
Enqueue(1)



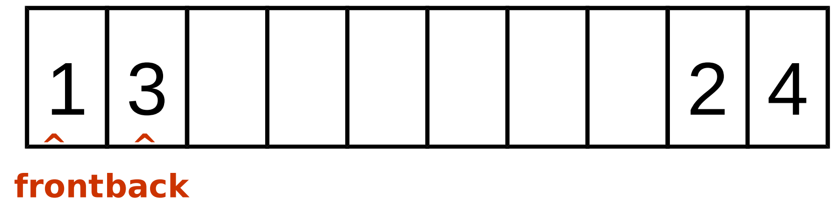
Enqueue(3)



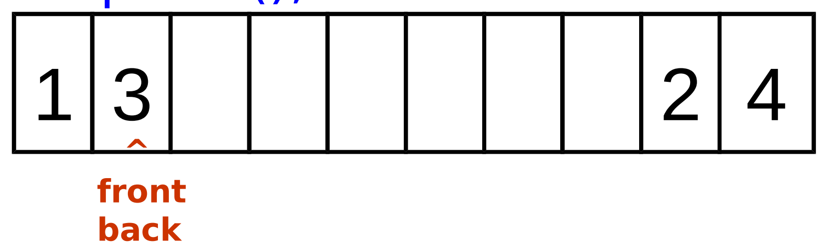
Dequeue(), returns 2



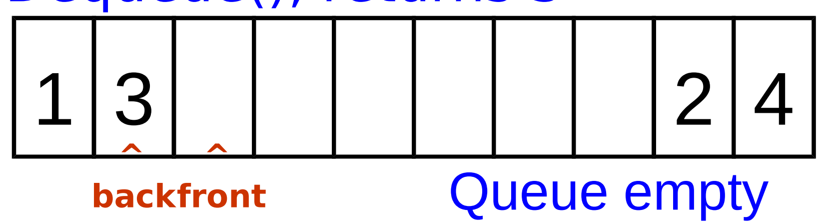
Dequeue(), returns 4



Dequeue(), returns 1



Dequeue(), returns 3



Applications of Queues

- Direct Applications:
  - Waiting lists, bureaucracy
  - Access to shared resources (e.g. printers, servers)
  - Multiprogramming
  - Queue theory, simulations
- Indirect Applications:
  - Auxiliary data structure for algorithms
  - Component of other data structures

Running Time

- Linked Lists:
  - Insert => O(1)
  - Remove => O(1)
  - Find => O(N)
  - Findkth => O(N)
- Stacks:
  - Push => O(1)
  - Pop => O(1)
- Queues:
  - Enqueue => O(1)
  - Dequeue => O(1)