Stacks

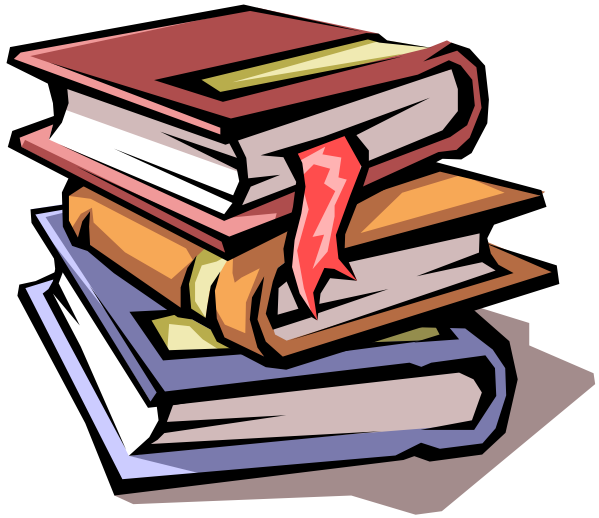
Carlos Delgado Kloos

Dep. Ingeniería Telemática

Univ. Carlos III de Madrid

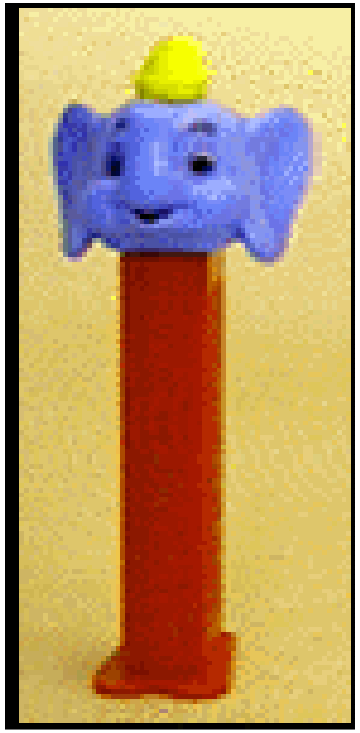
cc BY NC SA

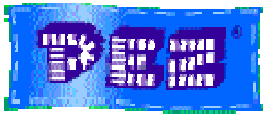
Example



cc BY NC SA
cdk@it.uc3m.es

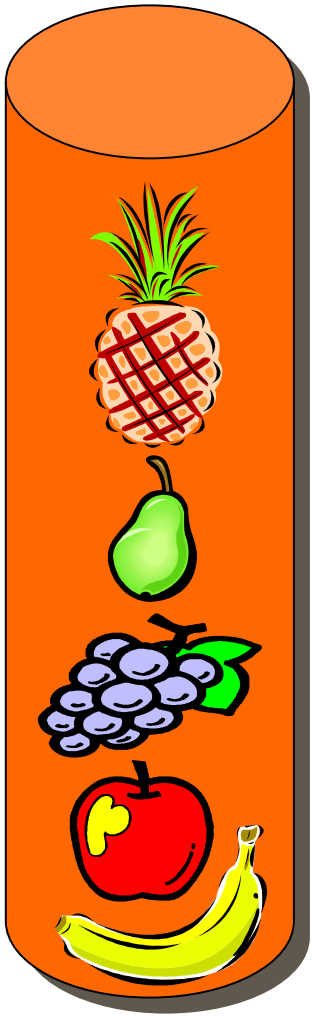
Example





cc BY NC SA

Example



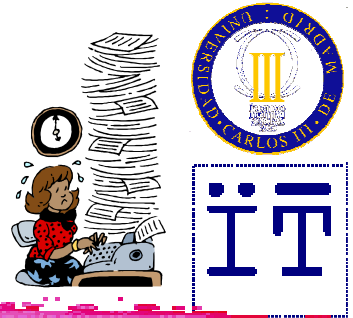
cc BY NC SA
cdk@it.uc3m.es

Characteristics

- Linear structure
- Access on one end both for insertion and extraction

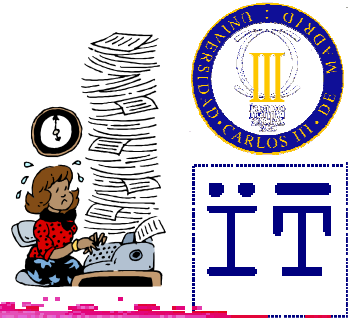
BY NC SA

Main methods



- Insert on one end: **`push(x)`**
- Extract at the same end: **`pop()`**

Example: Check parentheses

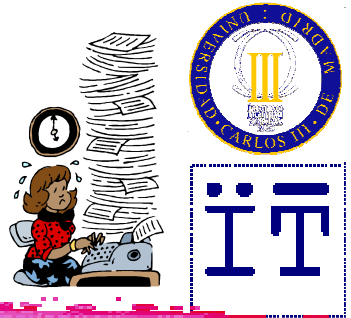


- Good:
  -
  - ()
  - (()(()))

- Bad:
  -)(
  - (()
  - ())

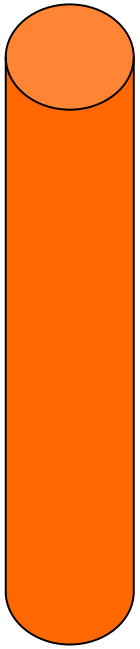
- Rules:
  - Basic:
  - Sequentiation: ()()
  - Nesting: (())

Example: Check parentheses



- Rules:
  - Each time we find a "(" we'll put it in the stack.
  - Each time we find a ")" we'll extract the upper "(" of the stack.
  - The string of parentheses is correct if the stack if empty after having gone through to complete string.

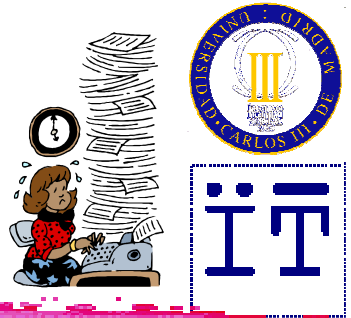
Example: check (()(()()))()

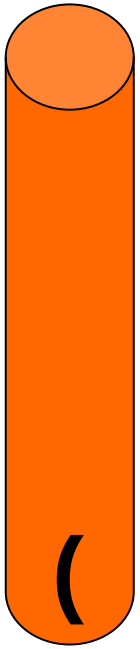


(( ) (( ) ( )) ( ))

BY NC SA
cdk@it.uc3m.es

Example: check (()(()()))

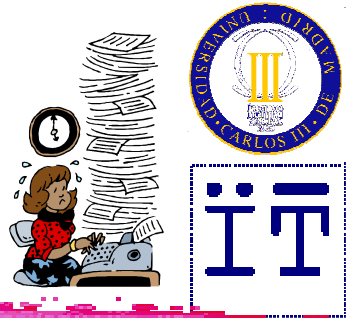


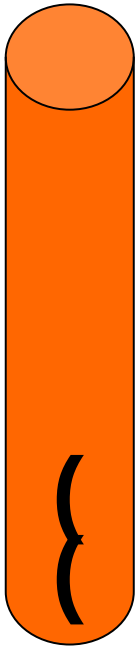


(( ) (( ) ( )) ( ))

BY NC SA

Example: check (()(()()))

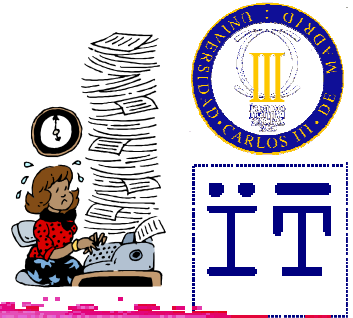


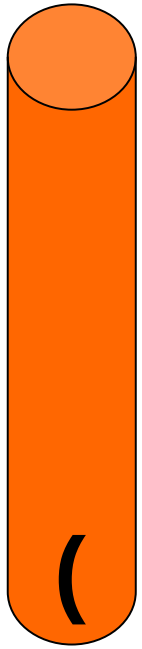


(( ) (( ) ( )) ( ))

BY NC SA

Example: check (()(()()))()

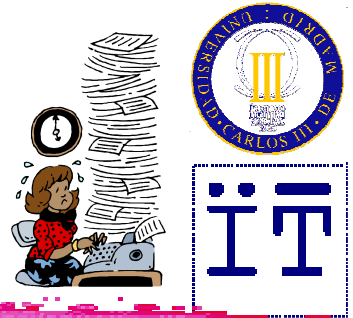


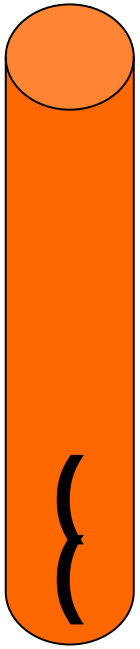


(()(()()))()

BY NC SA

Example: check (()(()()))

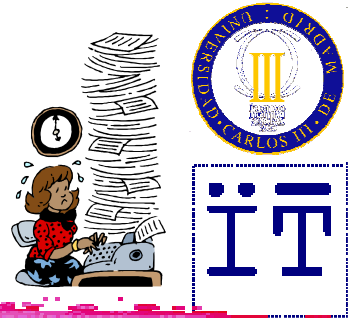


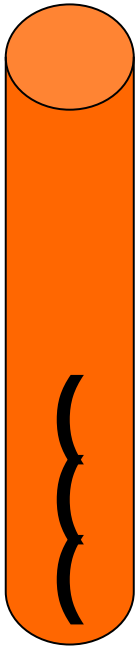


(( ) (( ) ( )) ( ))

BY NC SA
cdk@it.uc3m.es

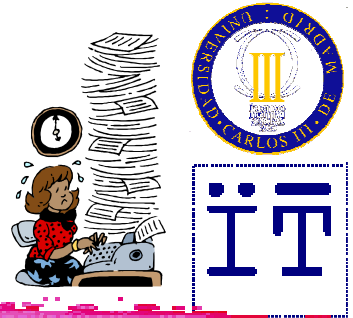
Example: check (()(()()))

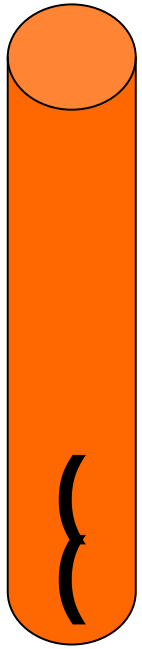




(( ) (( ) ( )) ( ))

Example: check (()(()()))

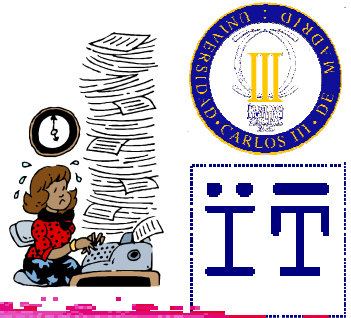


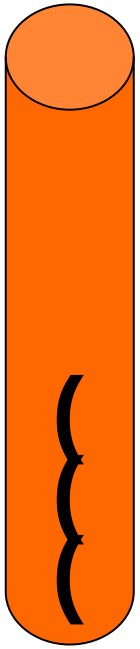


(( ) ((( ) ( )) ( ))

BY NC SA

Example: check (()(()()))

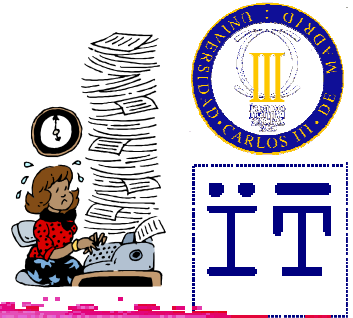


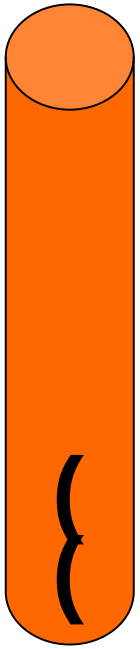


(()(()(())))

BY NC SA

Example: check (()(()()))

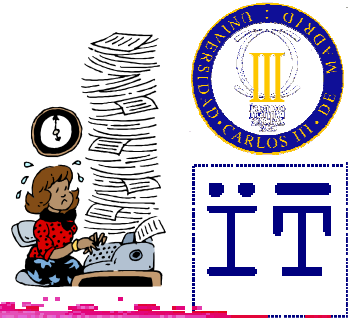


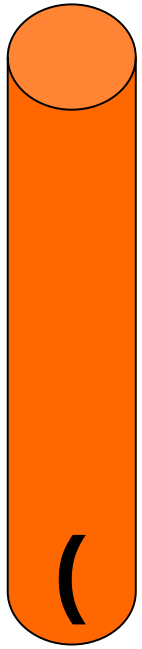


(()(()()))

BY NC SA

Example: check (()(()()))

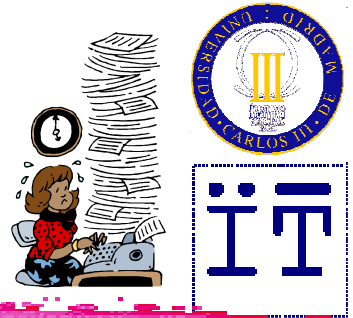


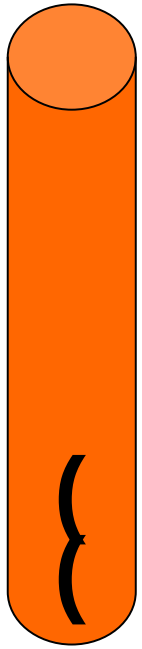


(()(()()))

BY NC SA

Example: check (()(()()))

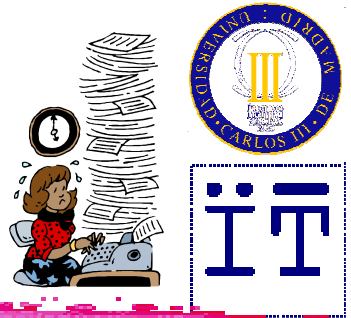


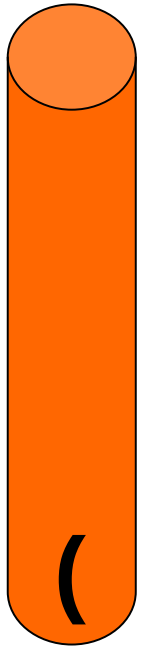


(()(()(())))

BY NC SA

Example: check (()(()()))

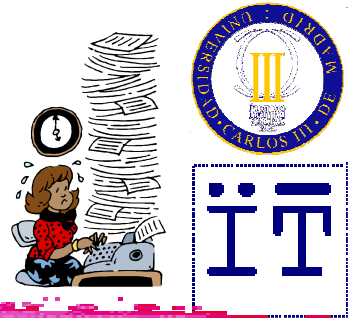


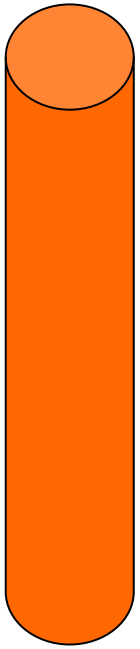


(()(()(()))())

BY NC SA

Example: check (()(()()))

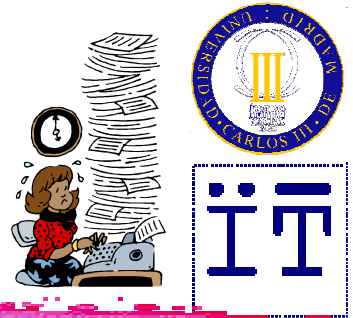


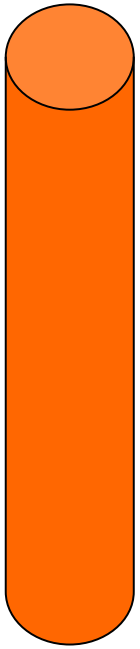


Correct: We have completed the string and the stack is empty

(()(()(())))

Example: check ([]{()<>}())





Correct: We have completed the string and the stack is empty

~~([]{()<>}())~~

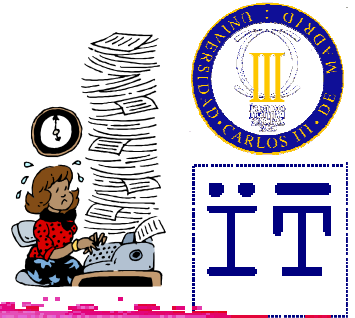
cc BY NC SA

Example: HTML

- **hello**
  - ([)]
  - Correct with HTML 1.0-4.0
  - Incorrect with XHTML
- **hello**
  - ([])
  - Correct with HTML 1.0-4.0
  - Correct with XHTML

cc BY NC SA

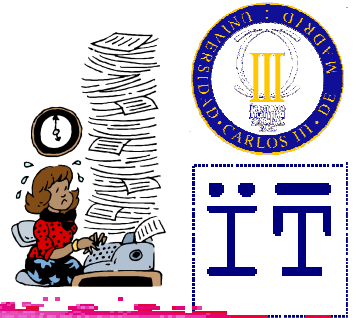
Stack interface

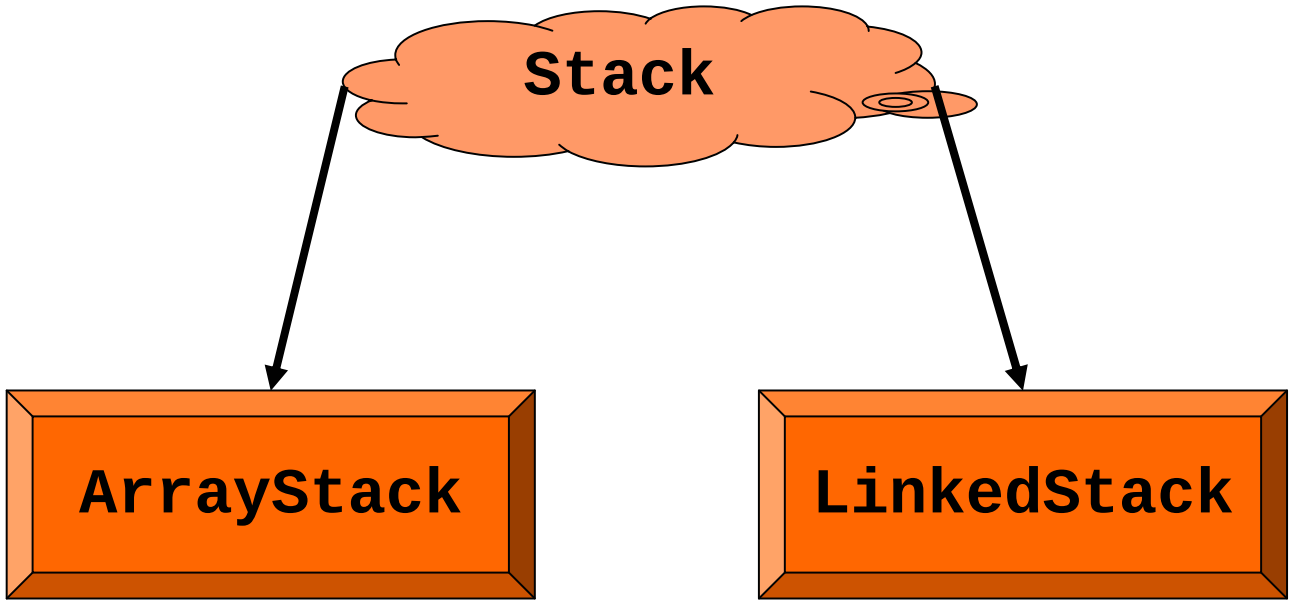


```
public interface Stack {
  public void push(Object o)
    throws StackOverflowException;
  public Object pop()
    throws EmptyStackException;
  public Object top()
    throws EmptyStackException;
  public int size();
  public boolean isEmpty();
}
```

cc BY NC SA

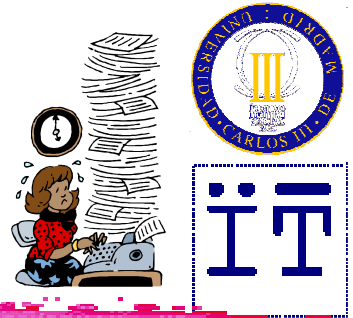
One interface and several implementations

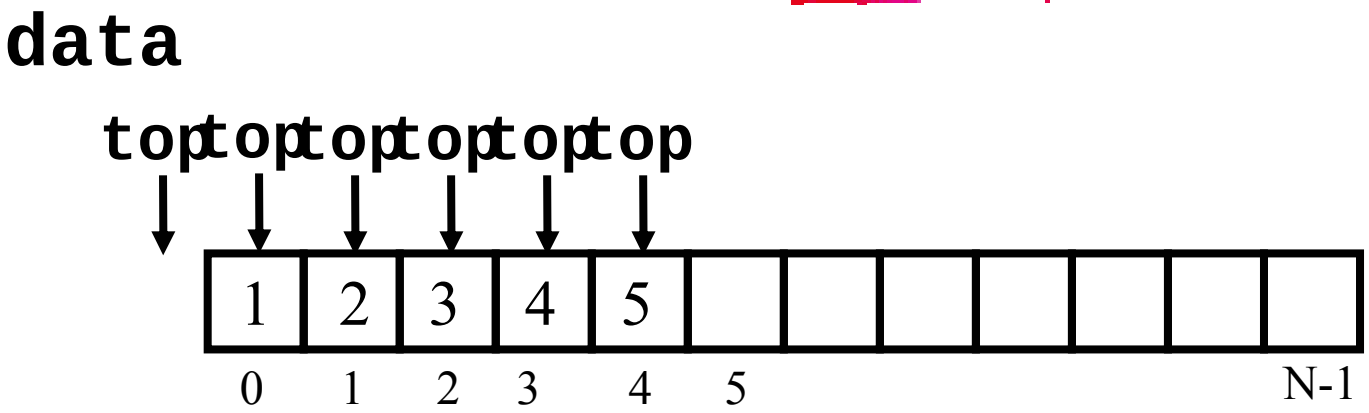




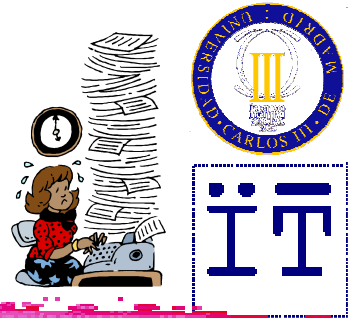
cc BY NC SA

Array-based implementation





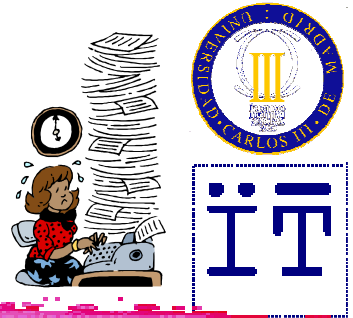
Array-based implementation



```
public class ArrayStack implements Stack {
  public static final int DEFAULT_CAPACITY = 1000;
  private int capacity;
  private Object data[];
  private int top = -1;
  public ArrayStack() {
    this(DEFAULT_CAPACITY);
  }
  public ArrayStack(int capacity) {
    this.capacity = capacity;
    data = new Object[capacity];
  }
```

cc BY NC SA

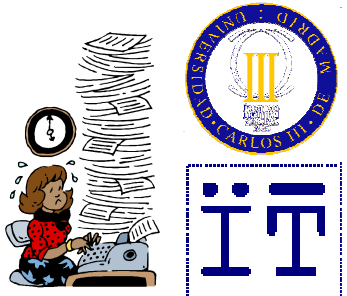
Array-based implementation



```
public int size() {
  return (top + 1);
}
public boolean isEmpty() {
  return (top < 0);
}
public Object top()
  throws EmptyStackException {
  if (top == -1)
    throw new EmptyStackException("empty");
  return data[top];
}
```

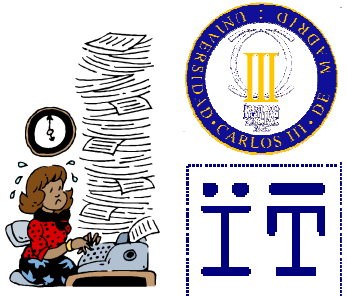
cc BY NC SA

Array-based implementation



```
public void push(Object o)
  throws StackOverflowException {
  if (top == capacity - 1)
    throw new StackOverflowException();
  data[++top] = o;
}
```

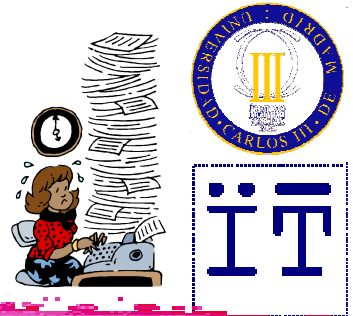
Array-based implementation

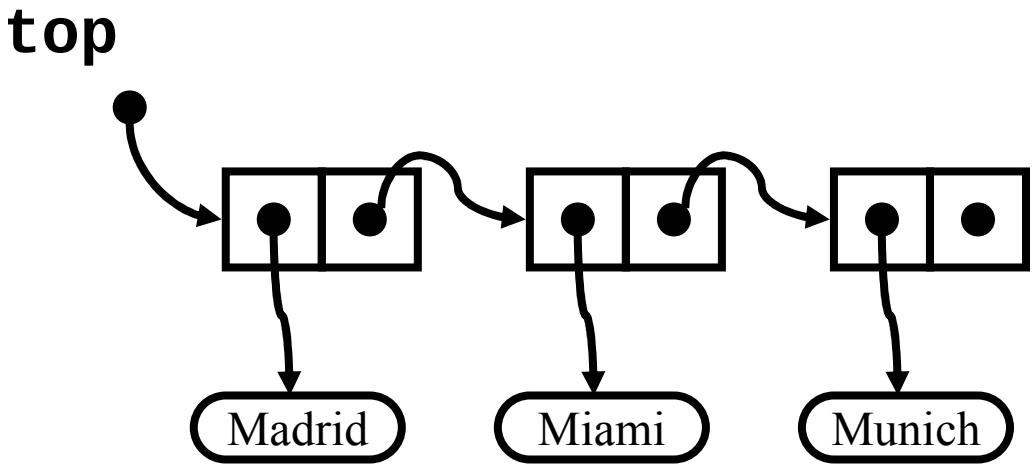


```
public Object pop()
   throws EmptyStackException {
   Object o;
   if (top == -1)
      throw new EmptyStackException();
   o = data[top];
   data[top--] = null;
   return o;
}
```

cc BY NC SA

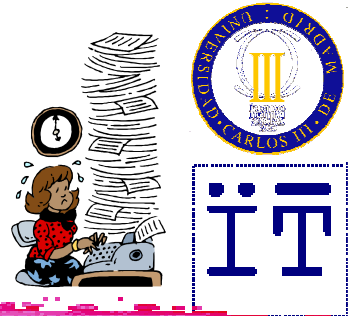
Implementation based on linked lists





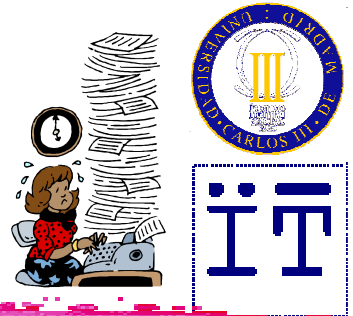
cc BY NC SA

Implementation based on linked lists



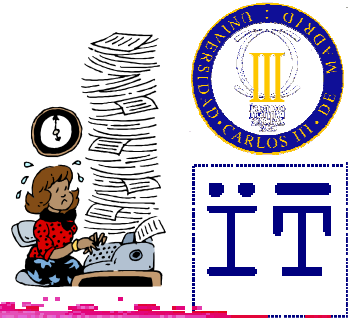
```
class Node {
  private Object info;
  private Node next;
  public Node(Object info, Node next) {
    this.info = info;
    this.next = next;
  }
  void setInfo(Object info) {this.info = info;}
  void setNext(Node next) {this.next = next;}
  Object getInfo() {return info;}
  Node getNext() {return next;}
}
```

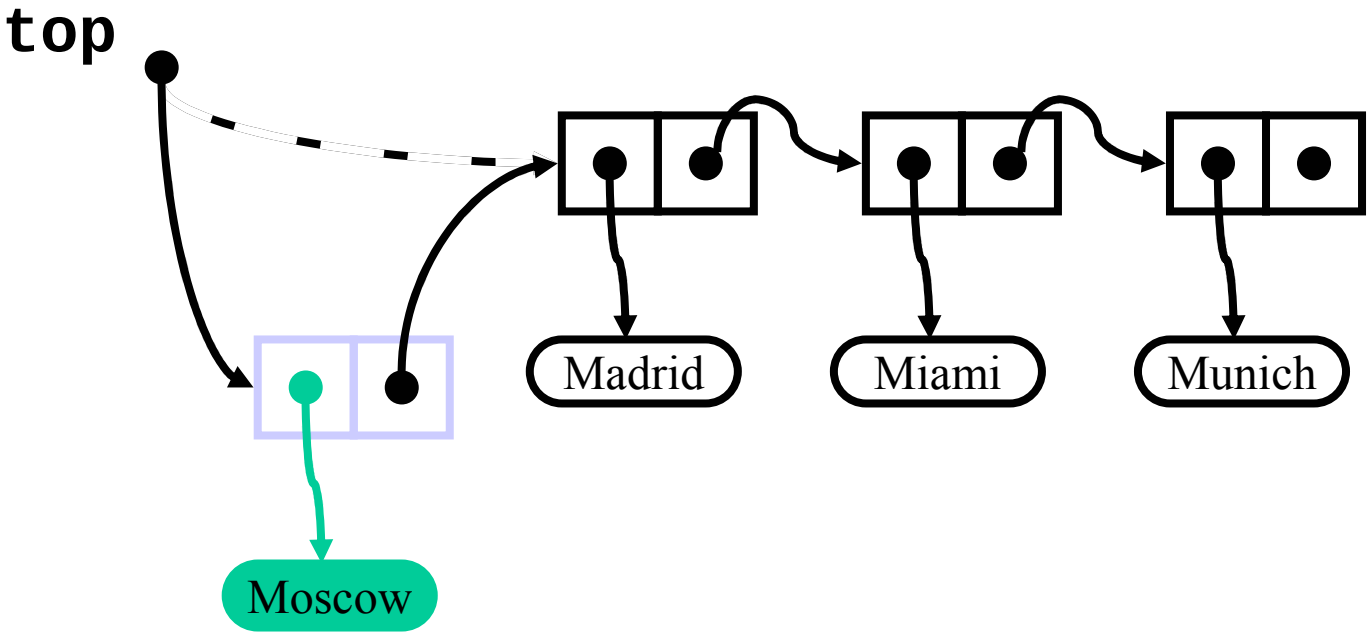
Implementation based on linked lists



```
public class LinkedStack implements Stack {
  private Node top;
  private int size;
  public LinkedStack() {
    top = null;
    size = 0;
  }
  public int size() {
    return size;
  }
  public boolean isEmpty() {
    return (top == null);
  }
```

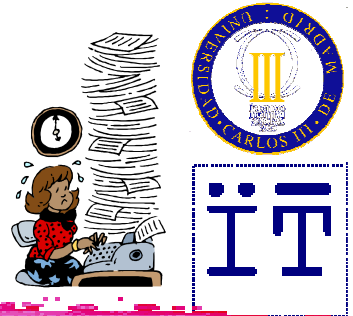
Insertion (push)





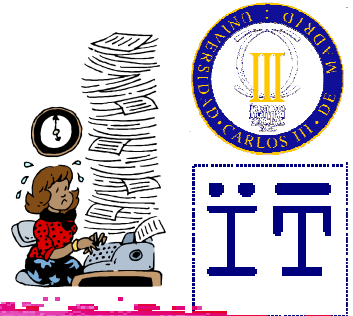
cc BY NC SA

Implementation based on linked lists



```
public void push(Object info) {
   Node n = new Node(info, top);
   top = n;
   size++;
}
```

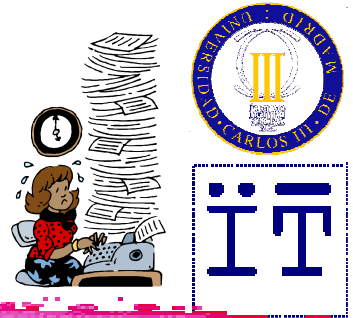
Implementation based on linked lists

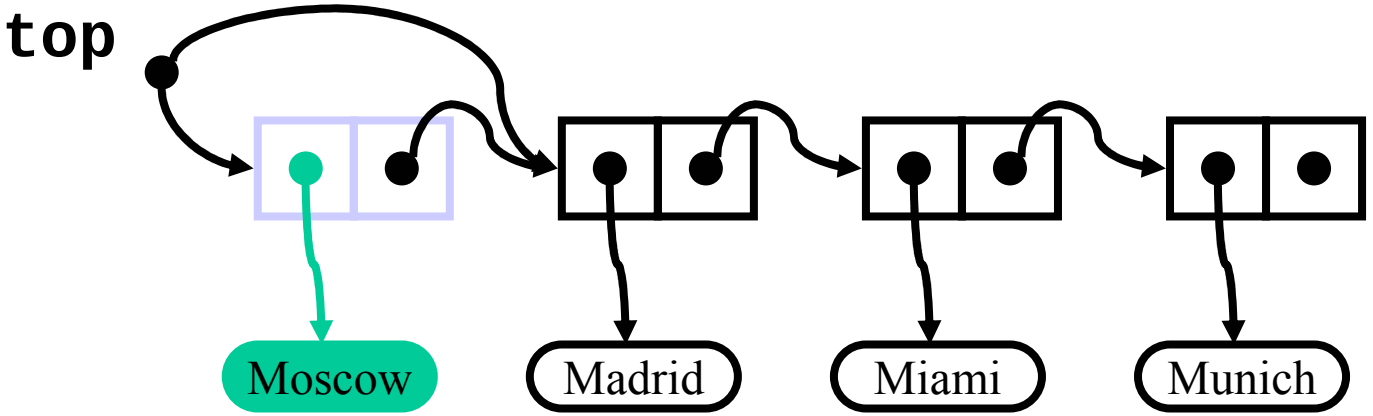


```
public Object top()
   throws EmptyStackException {
   if (top == null)
      throw new EmptyStackException();
   return top.getInfo();
}
```

cc BY NC SA

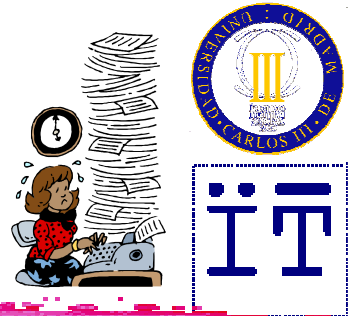
Extraction (pop)





BY NC SA

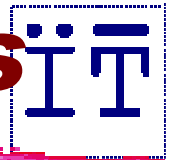
Implementation based on linked lists



```
public Object pop()
   throws EmptyStackException {
   Object info;
   if (top == null)
      throw new EmptyStackException();
   info = top.getInfo();
   top = top.getNext();
   size--;
   return info;
}
```

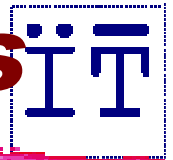
cc BY NC SA

Arrays vs. Linked lists



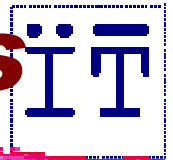
- Advantages of arrays:
  - Efficient use of memory when the number of elements is known before creation
  - Fast random access (to any position)
- Disadvantages of arrays:
  - Data needs to be moved in insertions, extractions, concatenations, etc.
  - Static size (resizing is possible but requires movement of data)
  - Needs contiguous memory

Arrays vs. Linked lists



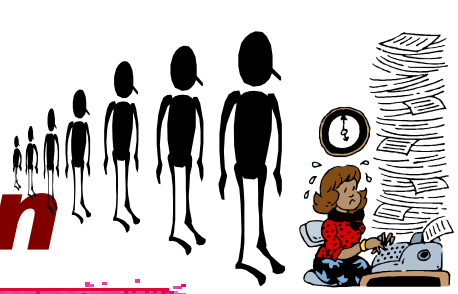
- Advantages of linked lists:
  - Insertions, extractions, concatenations, partitions without movement of data
  - Dynamic size
  - No need of contiguous memory
- Disadvantages of linked lists:
  - Slow access to random positions
  - Memory overhead to store nodes

Arrays vs. Linked lists



- Advantages of arrays:
  - Efficient use of memory when the number of elements is known before creation
  - Fast random access (to any position)
- Disadvantages of arrays:
  - Data needs to be moved in insertions, extractions, concatenations, etc.
  - Static size (resizing is possible but requires movement of data)
  - Needs contiguous memory

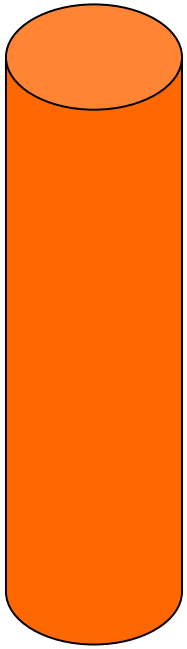
Stacks and recursion



```
public static long fac (int n) {
  if (n <= 1)
     return 1;
   else
     return n * fac(n - 1);
  }
```

Execution

- `fac(4)`
- `4*fac(3)`
- `4*(3*fac(2))`
- `4*(3*(2*fac(1)))`
- `4*(3*(2*1))`
- `4*(3*2)`
- `4*6`
- `24`



cc BY NC SA