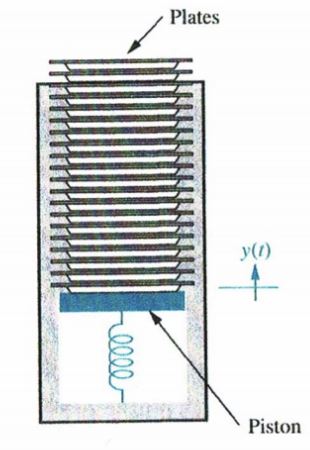
**Stacks**

A *stack* is ordered list in which all insertions and deletions are made at one end. The restrictions on a *stack* imply that if the elements *A, B, C, D , E* are added to the *stack*, in that order, then the first element to be removed/deleted must be *E*. Equivalently, we say that the last element to be inserted into the *stack* will be the first to be removed. For this reason, *stacks* are sometimes referred to as Last In First Out (LIFO) lists. Consequently, a *stack*, also called a last-in first-out (LIFO) system, is a linear list in which insertions and deletions can take place only at one end, called the top. This structure is similar in its operation to a stack of dishes on a spring system, as shown below. Note that new dishes are inserted only at the top of the stack and dishes can be deleted only from the top of the stack.



If we regard a stack as an object, we can represent it using the UML notation as shown below:



**Attributes**

top points to the element, which is on top of a stack, i.e. the element that has been added to a stack last. item represents "things" that are added to a stack. max indicates the maximum capacity of a stack.

**Operations**

create() is responsible for initialising a stack. push() enables an item to be inserted into a stack. pop() removes to delete an item from a stack. getTop() retrieves the item, which is on top of a stack. isEmpty() determines whether or not a stack has any items in it. isFull() determines whether or not a stack is full.

**States**

States of an object constrain the operations applicable to it. This is to ensure that its integrity is maintained. The following states are applicable to a stack.

*Empty****:***A stack can be empty under two circumstances:

1. when it is created
2. when all items stored in it are deleted

*Empty* constrains two operations. Thus, this state must be checked before applying them. They are:

1. top(): it is not possible to retrieve an item when a stack is empty.
2. pop(): it is not possible to remove an item from an empty stack.

***Full:*** A stack can be full under the following circumstance.

1. whenits capacity is exhausted.

*Full* constrains one operation. Thus, this state must be checked before applying it. It is:

1. push(): it is not possible to insert an item when a stack is full.

**Representation of Stacks**

Stacks may be represented in the computer various ways, usually by means of a singly linked list or a linear array. In this lecture note, array representation of stacks will be outlined as shown below:

declare STACK[1..max], top, item

Having declared a stack as a single dimensional array, we will now show the algorithms of each of its operations:

procedure create()

top ← 0

end

procedure push(IN item, INOUT top INOUT STACK[], IN max)

if isFull(IN top, IN max) = true then CALL stackFULL()

else

top ← top + 1

STACK(top) ← item

endif

end

procedure pop(OUT item, INOUT STACK[], INOUT top)

if isEmpty(top) = true then CALL stackEMPTY()

else

item ← STACK(top)

top ← top - 1

endif

end

function getTop(IN top, IN STACK[])

if isEmpty(top) = true then CALL stackEMPTY()

else

return STACK(top)

endif

end

function isEmpty(IN top, IN STACK[])

if top = 0 then return true

else return false

endif

end

function isFull(IN top, IN max)

if top = max then return true

else return false

endif

end

**Examples**

Suppose the following six elements are pushed, in order, onto an empty stack:

AAA, BBB, CCC, DDD, EEE, FFF

TOP TOP

|  |
| --- |
|  |
| FFF |
| EEE |
| DDD |
| CCC |
| BBB |
| AAA |

|  |
| --- |
| AAA |
| BBB |
| CCC |
| DDD |
| EEE |
| FFF |
|  |
|  |

TOP

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| AAA | BBB | CCC | DDD | EEE | FFF |  |

Irrespective of the way a stack is represented as a single dimensional array, its underlying property remains the same. That is that insertions and deletions can occur only at the top of the stack. This means EEE cannot be deleted before FFF is deleted, DDD cannot be deleted before EEE and FFF are deleted, and so on. Consequently, the elements may be popped from the stack only in the reverse order of that in which they were pushed onto the stack.