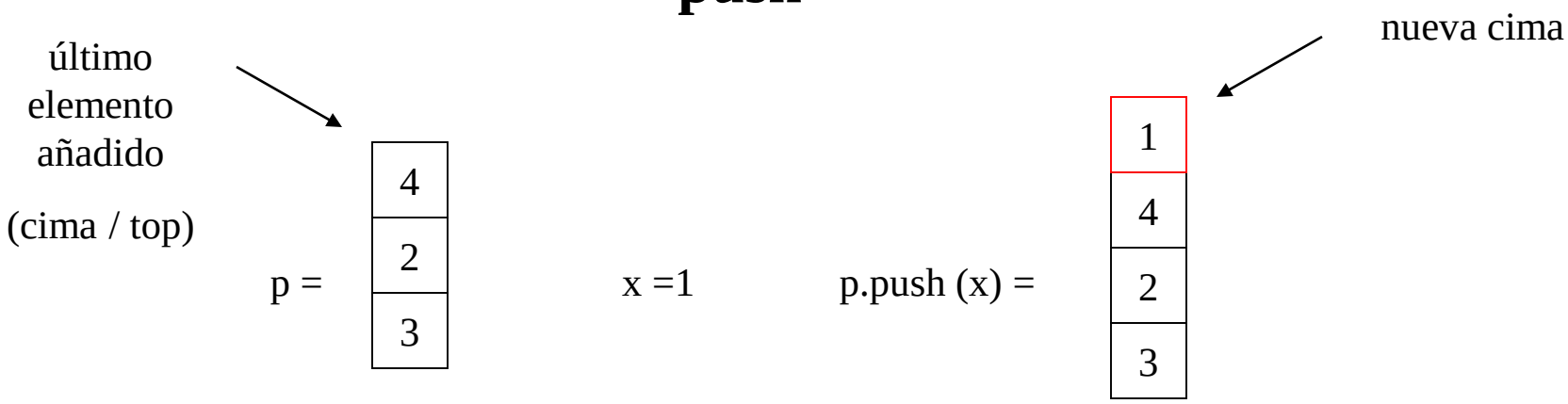
Módulos especiales

- Estructuras lineales:
  - Pilas
  - Colas
  - Listas
- Estructuras arborescentes

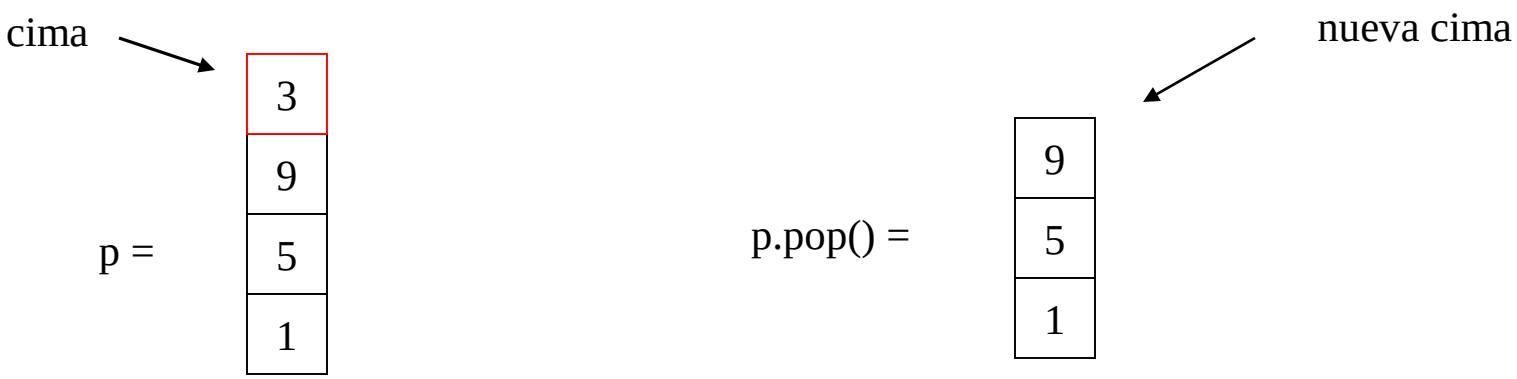
Pilas

- Estructuras lineales: cada elemento puede tener un sucesor como máximo
- Solo se puede consultar o borrar el último elemento añadido (el más nuevo o reciente): estrategia LIFO (last in-first out)
- Su contenido se puede parametrizar: pilas de números, de booleanos, etc.
- Especificación: ver `stack_esp.hh`

push


último
elemento
añadido
(cima / top)
p =
4
2
3
x =1
p.push (x) =
nueva cima
1
4
2
3

pop


cima
p =
3
9
5
1
p.pop() =
nueva cima
9
5
1

Ejemplos de uso de pilas

Suma de una pila de enteros

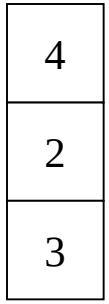
```
int suma_pila_int(stack<int>& p)
/* Pre: p=P */
/* Post: El resultat és la suma dels elements de P */
```

•La operación no podría ser genérica, ya que depende del *tipo* del contenido de la pila (necesita tener definido el operador +)

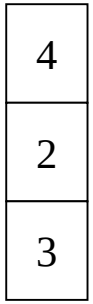
•No dice cómo queda la pila al final de la operación

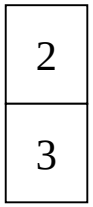
•La primera versión que veremos será recursiva

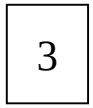
```
int suma_pila_int(stack<int>& p)
/* Pre: p=P */
/* Post: El resultat és la suma dels elements de P */
{
  int ret;
  if (p.empty()) ret = 0;
  else{
    int aux = p.top();
    p.pop();
    ret = suma_pila_int(p) + aux;
  }
  return ret;
}
```

p = 

suma (p) = 9

suma (

) = 4+ suma (

) = 6+ suma (

)= 9+ suma (p_vacia) = 9+0 = 9

Versión iterativa:

•si pasamos el parámetro por valor, garantizamos que la pila no se destruye (la operación recibe una copia)

•si hiciéramos eso en la versión recursiva, tendríamos mucha ineficiencia (una copia en cada llamada)

```
int suma_pila_int(stack<int> p)
/* Pre: p=P */
/* Post: El resultat és la suma dels elements de P */
{
  int ret = 0;
  while (not p.empty()) {
    ret += p.top();
    p.pop();
  }
  return ret;
}
```

Más ejemplos básicos de uso de pilas

- Búsqueda de un elemento en una pila
- Determinar si dos pilas son iguales
- Sumar una valor k a todos los elementos de una pila

`(ver fichero ejemplos_pila.cc)`

Ejemplos avanzados de uso de pilas

• Comprobar si una expresión aritmética/booleana está parentizada correctamente

(([()])) se apilan los abiertos y se desapilan cuando llegan los correspondientes cerrados, hasta que llegue uno que no coincida

• Simulación de la recursividad mediante iteraciones (ejemplo: recorrido de un árbol binario por niveles)

• Evaluar una expresión aritmética/booleana dada en notación polaca inversa

(a * (b + c)) ➔ a b c + *

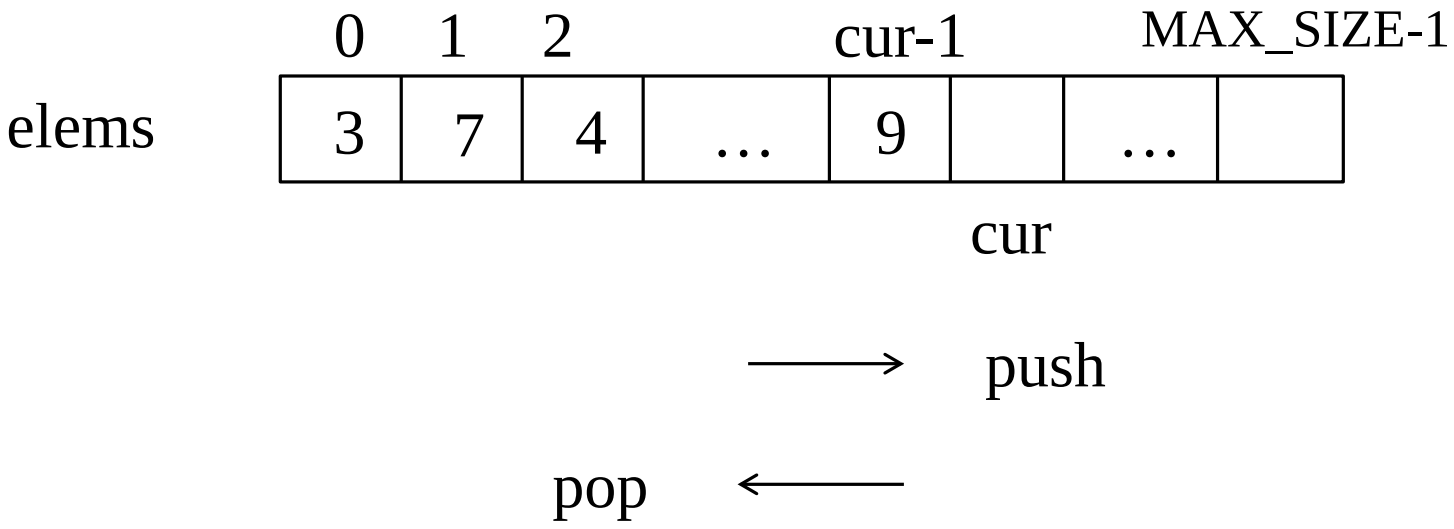
a * b + c ➔ a b * c +

Ejemplos de implementación de pilas

- Con un vector estático, particularizando el contenido (`stack_int.hh, stack_int.cc`)
- Con un vector estático, genérica (`stack.hh`)
- Lo mismo, con un vector de tamaño variable (`push_back`)
- Con punteros (lo veremos más adelante)
- Por adaptación de estructuras más complejas (`STL: deque`)

Ejemplos de implementación de pilas

Si implementamos la pila con un vector, guardamos sus elementos en orden cronológico desde la posición 0 hasta cur-1



Independencia de la implementación

Aunque sepamos que la pila se implementa con un vector,
al usar la clase no podemos hacer cosas como

`p[i]=14` ni `p.elems[i]=14`

Tampoco podemos contar con una operación que vuelque
la pila en un vector

```
vector<T> volcar_elementos(const stack<T> &p);
```