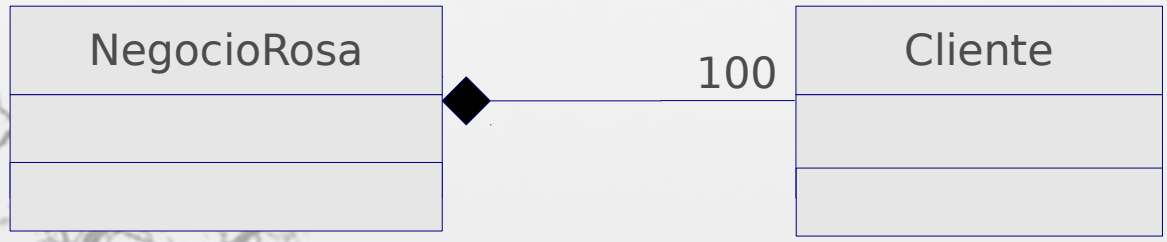
Lección 4:
Vectores estáticos y dinámicos

- Motivación
- Definiciones
- Vectores estáticos
- La clase vector estático
- Búsqueda binaria
- Vectores dinámicos
- Inserción
- Borrado
- Relaciones entre clases de objetos
- ¿Cuando uso vectores?

Motivación

- Rosa vende por Internet abalorios que ella misma fabrica y quiere gestionar el listín de direcciones de sus clientes.
- Lo que pretende hacer es: ordenar, buscar y listar sus clientes.
- Calcula que tendrá unos 100, y elige un vector estático.

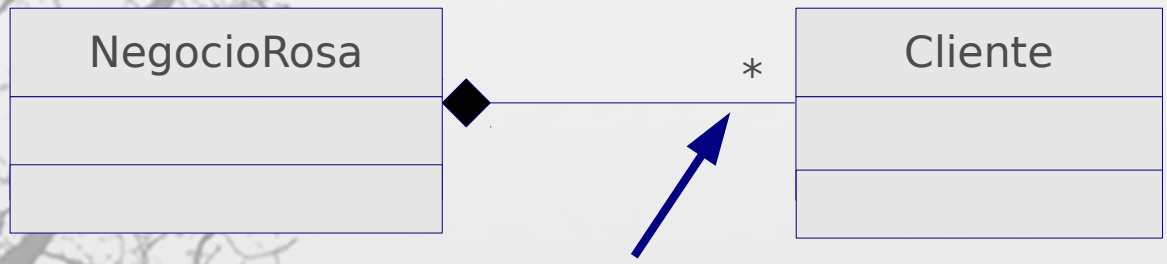



```
class Cliente{
    string nombre;
    string direcc, tfno;
public:
    Cliente(string...);
    ...
}
```

```
Class NegocioRosa{
    Cliente clientes[100];
public:
    NegocioRosa();
    ...
}
```

Motivación

- Al principio le va bien pero al cabo de los meses consigue más clientes y tiene que añadir 50 posiciones más a su vector.
- Se da cuenta que no es la solución, no le gustaría ponerle barreras a su negocio, querría que creciera o decreciera según su volumen de datos.
- Esto lo resolvería un vector dinámico.



El (*) en UML significa muchos, sin una cota máxima

Definiciones

- Un **vector (array)** es una zona contigua de almacenamiento en memoria que contiene objetos de un mismo tipo.
- Un **vector estático** tiene asignado una cantidad fija de memoria y no es capaz de crecer o decrecer en tiempo de ejecución.
- Un **vector dinámico** tiene asignado un espacio inicial que puede crecer o decrecer en función de que lo hagan las necesidades de almacenar datos.

Definiciones

Los vectores se pueden catalogar del siguiente modo:

- Son EEDD lineales y de acceso directo o aleatorio mediante un índice único (en tiempo **O(1)**).
- La inserción al final es también constante.
- Son unidimensionales aunque pueden manejar dos dimensiones convirtiéndose entonces en matrices.
- Están disponibles en la gran mayoría de lenguajes
- Son el contenedor habitual para otras estructuras de datos y adaptadores: pilas, tablas hash, etc.
- No son perfectas: la inserción en posiciones intermedias tiene un coste **O(n)**.

Vectores estáticos

- Los vectores estáticos son la solución para manejar datos con un tamaño máximo conocido
- Siempre ocupan el mismo espacio (tamaño físico) aunque no estén llenos (tamaño lógico < tamaño físico).

```
int arr[100];
int nElem = 0;

arr[0] = 77;
arr[1] = 99;
nElem = 2;
for (int i=0; i<nElem; i++) cout << arr[j] << endl;
```

Este código define un array de tamaño físico 100, inserta dos elementos y queda con tamaño lógico igual a 2. Luego se lista su contenido por pantalla.

Vectores estáticos

```
int cbusca = 66;
bool encontrado = false;
for(j=0; j<nElem; j++)
    if(arr[j] == cbusca)
        encontrado = true;
if(encontrado)
    cout << "Encontrado "<< cbusca << endl;
else cout << "No está " << cbusca << endl;
```

Búsqueda secuencial:
Este código busca un el valor 66
El resultado es: "No está 66"

```
#include <algorithm>
...
int main (void){
    int values[] = { 40, 10, 100, 90, 20, 25 };
    sort(values, values+6);
    for (int i=0; i<6; i++) {
       cout << values[i] << " ";
    }
}
```

Este código ordena el array

La clase Vector estático

Ejemplo de definición de vector estático, es mejorable añadiendo excepciones

```
#ifndef VESTATICO_H
#define VESTATICO_H

class Vestatico {
    int tamal;
    int tamaf;
    int *v;

public:
    Vestatico(int tama);
    Vestatico(const Vestatico& orig);
    int lee(int pos){return v[pos];}
    void escribe(int pos, int dato){v[pos] = dato;}
    void ordenar();
    int busca(int dato);
    int busquedaBin(int dato);
    ~Vestatico(){delete []v;}
};

#endif  /* VESTATICO_H */
```

La clase Vector estático

El fichero cpp

```
#include <algorithm>
#include "Vestatico.h"

Vestatico::Vestatico(int tama=1) {
    v = new int[tamal = tamaf = tama];
}

Vestatico::Vestatico(const Vestatico &orig) {
    v = new int[tamaf = orig.tamaf];
    tamal = orig.tamal;
    for (int i=0; i<tamal; i++) v[i] = orig.v[i];
}

void Vestatico::ordenar(){
    sort(v, v+tamal);
}

int Vestatico::busca (int dato){
    for(j=0; j<tamal; j++)
        if(arr[j] == dato) return j;
    return -1;
}
```

Búsqueda binaria

- Un vector debe estar ordenado para que funcione la búsqueda binaria o dicotómica.
- También usada en vectores dinámicos.

```
int Vestatico::busquedaBin(int dato);
  int inf = 0;
  int sup = tamal-1;
  int curIn;
  while(true) {
     curIn = (inf + sup) / 2;
     if(v[curIn] == dato)
        return curIn;
     else if(inf > sup) return -1;
        else {
            if(v[curIn] < dato) inf = curIn + 1;
            else sup = curIn - 1;
        }
  }
}
```

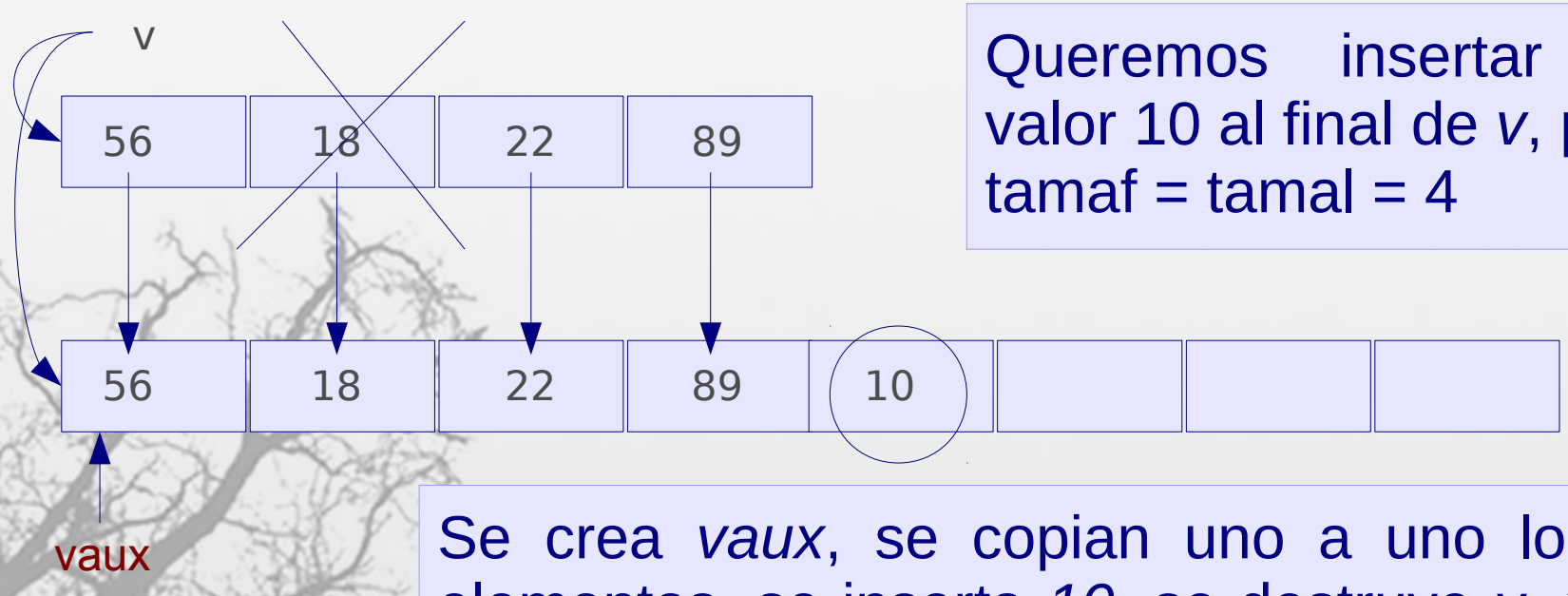
Vectores dinámicos

- Los vectores dinámicos son más versátiles porque adaptan su tamaño físico al tamaño de los datos: crecen y decrecen en tiempo de ejecución.
- La definición es similar:

```
class Vdinamico {
    int tamal,tamaf;
    int *v;
public:
    Vdinamico():
    Vdinamico(const Vdinamico& orig);
    int lee(int pos){ return v[pos]; }
    void escribe(int pos, int dato){ v[pos] = dato; }
    void inserta(int dato, unsigned pos);
    int elimina(unsigned pos);
    void aumenta(int dato); // Inserción por la derecha
    int disminuye(); // Eliminar dato por la derecha
    unsigned tama(){ return tamal; };
    void ordenar();
    int busca(int dato);
    int busquedaBin(int dato);
    ~Vdinamico();
};
```

Inserción en vectores dinámicos

- La función aumenta() inserta un dato al final, pero si no cabe (tamaf=tamal), entonces el vector **crece al doble** para dejar nuevo espacio libre.



Queremos insertar un valor 10 al final de *v*, pero tamaf = tamal = 4

Se crea *vaux*, se copian uno a uno los elementos, se inserta *10*, se destruye *v* y luego éste apunta al mismo lugar que *vaux*

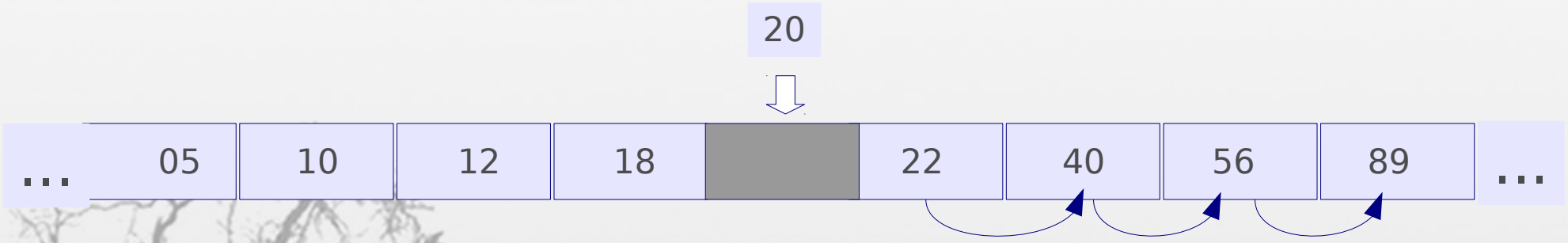
Inserción en vectores dinámicos

- Este proceso es más eficiente que crecer un tamaño constante, adaptándose a la magnitud de los datos.
- El tamaño físico será siempre potencia de 2

```
void Vdinamico::aumenta(int dato){
    if(tamal==tamaf) {
        int *vaux;
        vaux= new int[tamaf=tamaf*2];
        for(int i=0;i<tamal;i++)
            vaux[i]=v[i];
        delete []v;
        v=vaux;
    }
    v[taml++]=dato;
}
```

Inserción en vectores dinámicos

- Insertar el final de un vector tiene un tiempo constante, pero insertar en otra posición con *inserta()* es tiempo lineal porque en el peor de los casos hay que abrir un hueco de tamaño *tamal*




```
...
for(unsigned i=tamal-1;i>=pos;i--){
    v[i+1]=v[i];
}
v[pos]=t;
...
```

este código de inserta() introduce dato en la posición pos

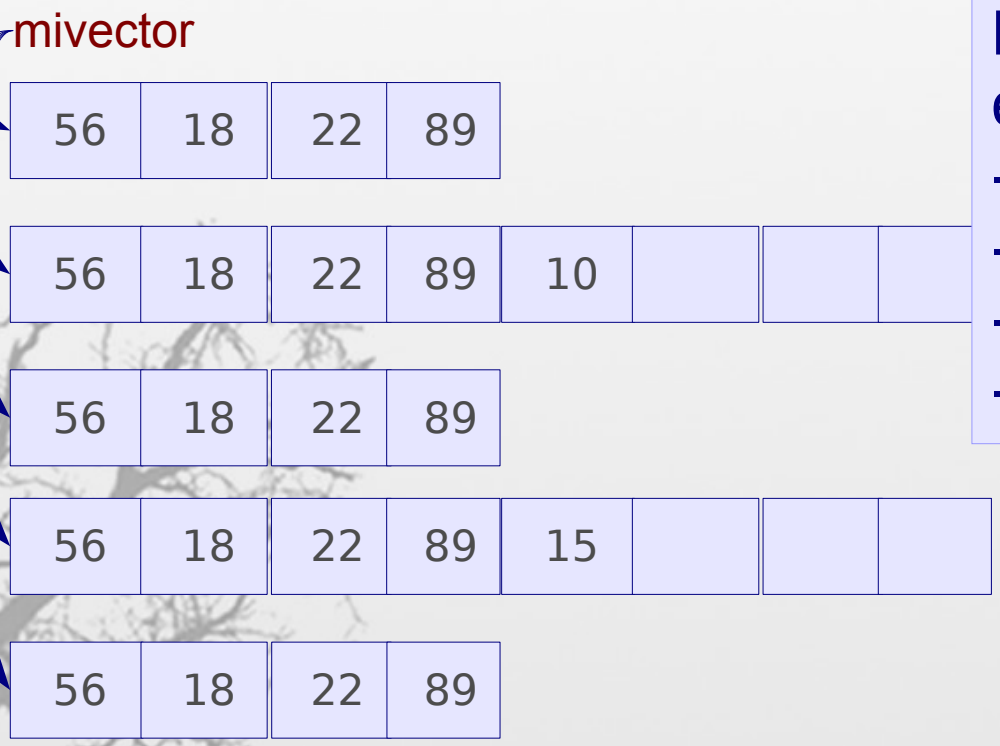
Borrar datos en vectores dinámicos

- El borrado en la posición final también es constante.
- Si el vector sufre muchos borrados se debe disminuir su tamaño a la mitad.

```
int Vdinamico::disminuye(){
  if(tamal*3<tamaf) {
     tamaf=tamaf/2;
     int *vaux = new int[tamaf];
     for(unsigned i=0;i<tamal;i++){
        vaux[i]=v[i];
     };
     delete []v;
     v=vaux;
  }
  return v[--taml];
}
```

Borrar datos en vectores dinámicos

- La reducción del tamaño físico se hace cuando el tamaño lógico es la tercera parte del tamaño físico.

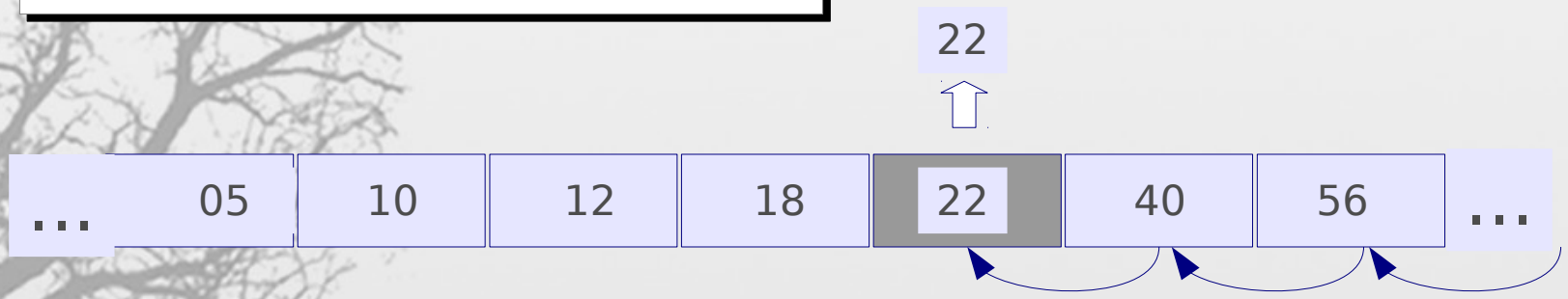


Esperar a un 1/3 evita esto:
- inserta 10, crece al doble
- borra 10, decrece
- inserta 15, aumenta
- borra 15, decrece

Borrar datos en vectores dinámicos

- El borrado en posiciones intermedias se considera operación en tiempo lineal.
- En realidad borrar significa sobreescribir posiciones
- Se reduce el tamaño a la mitad si **tamal*3<tamaf**

```
for(unsigned i=pos;i<taml;i++){
    v[i]=v[i+1];
};
taml--;
```

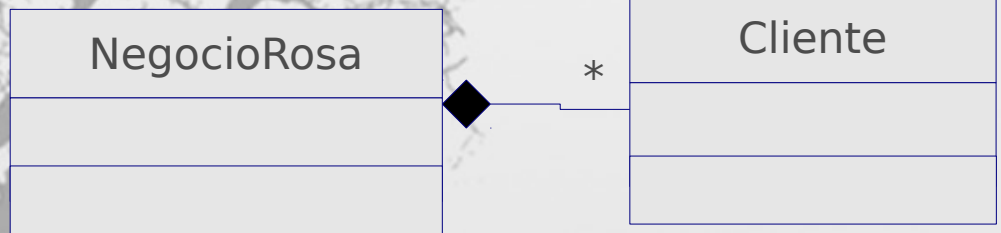


Relaciones entre clases de objetos

Las **composiciones** pueden implementarse de dos modos: (1) insertando objetos dentro del vector o (2) insertando un puntero a dicho objeto.

En cualquier caso la clase contenedora crea y destruye los objetos.

Recordemos el negocio de Rosa




```
class NegocioRosa{
    Cliente *cli;
    int tamal;
public:
    NegocioRosa(int n){
        cli = new Cliente[n];
        ...
    }
    ~NegocioRosa(){
        delete []cli;
    }
...
};
```

Relaciones entre clases de objetos: composición

En las composiciones implementadas mediante punteros, la clase contenedora debe crear y destruir los objetos que contiene.

La clase NegocioRosa debe crear (new) y destruir (delete) los objetos Cliente.

```
class NegocioRosa{
    Cliente **cli;
    int tamal;
public:
    NegocioRosa(int n){
        cli = new Cliente*[n];
        ...
    }
    void nuevoCli(Cliente &c);
    ~NegocioRosa();
};
```

```
void NegocioRosa::nuevoCli(Cliente &c)
{
   cli[tamal++] = new Cliente(c);
}

NegocioRosa::~NegocioRosa(){
    for (int i=0; i<tamal; i++)
        delete cli[i];
    delete []cli;
}
```

Relaciones entre clases de objetos: composición

Lo más habitula será utilizar la clase vector como patrón, Vdinamico<T>

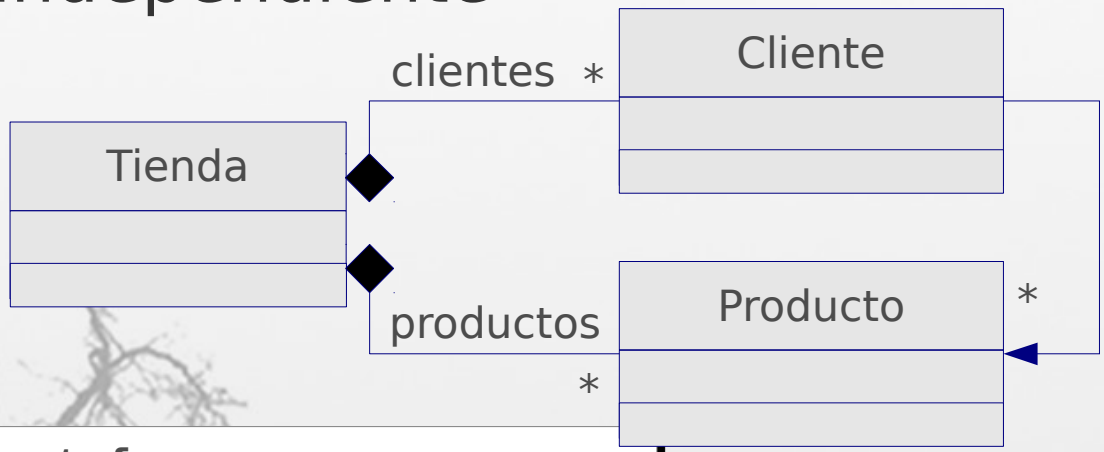
Modificamos la clase NegocioRosa para que maneje Vdinamico<T>, pero debe crear igualmente los objetos Cliente; la destrucción del array la realiza el destructor de Vdinamico<T>

```
class NegocioRosa{
    Vdinamico<Cliente*> cli;

public:
    NegocioRosa(int n):cli(n){}
    void nuevoCli(Cliente &c);
    ~NegocioRosa();
};
```

```
void NegocioRosa::nuevoCli(Cliente &c)
{
   cli.aumenta(new Cliente(c));
}

NegocioRosa::~NegocioRosa(){
    for (int i=0; i<tamal; i++)
        delete cli[i];
}
```

Relaciones entre clases de objetos

Las **asociaciones** se implementan siempre a través de punteros a objetos ya existentes, cuyo ciclo de vida es independiente




```
class Cliente{
    Producto *listaCompra[];
    string nombre, apellidos, nif;
    int nproductos;
public:
    Cliente();
    nuevoProd(Producto *p);
...
}
```

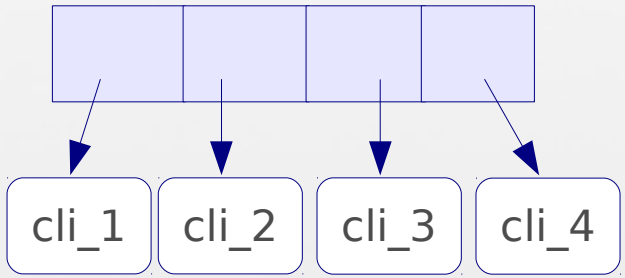
```
Cliente::nuevoProd(Producto *p){
    listaCompra[nproductos++]=p;
}
Cliente::~Cliente(){
    delete []listaCompra;
}
```

No se borran los productos

Relaciones entre clases de objetos: asociación

- Las asociaciones implementadas con la clase Vdinamico<T> se manejan instanciándolas siempre a punteros
- Los productos se pasan como punteros a objetos ya creados

```
class Cliente{
    Vdinamico<Producto*> listaCompra;
    string nombre, apellidos, nif;
    int nproductos;
public:
    Cliente();
    nuevoProd(Producto *p);
...
}
```




```
Cliente::nuevoProd(Producto *p){
    listaCompra.aumenta(p);
}

Cliente::~Cliente(){}
```

No se borran los productos ni el vector

¿Cúando uso vectores?

Los vectores son las estructuras de datos más simples y fáciles de manejar. Se aconseja usarlas:

- Si el tamaño de los datos es pequeño.
- Si puedo acceder mediante índices enteros: O(1)
- Si las inserciones/borrados son por el final: O(1)
- Realizo búsquedas binarias en un vector ordenado O(*log* n) y apenas hago inserciones O(n).

Por tanto no son siempre la mejor opción:

- Cuando haya que insertar en cualquier posición
- Haya grandes masas de datos localizables por clave y que sufren altas/bajas/modificaciones.

De ahora en adelante

Los vectores son EEDD muy utilizadas por ser fáciles de manejar, pero tienen los inconvenientes citados anteriormente:

- Utilizaremos listas enlazadas para mejorar el tiempo de inserción en posiciones intermedias
- Utilizaremos contenedores asociativos como árboles y tablas hash para mejorar las búsquedas de datos por clave