

Gestión de la memoria

Profesor Yisheng León



Errores de asignación de memoria

La memoria se reparte entre la pila y el montón

Crece en direcciones opuestas

Al llamar a subprogramas la pila crece

Al crear datos dinámicos el montón crece

Colisión pila-montón

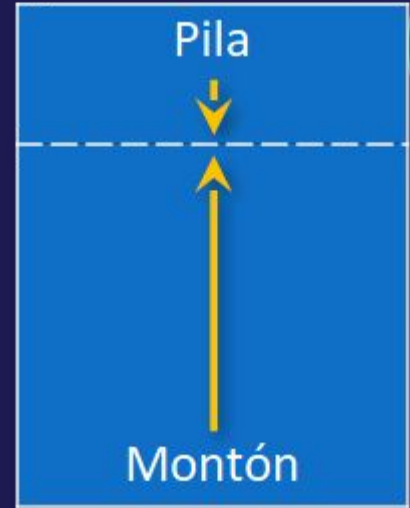
Los límites de ambas regiones se encuentran

Se agota la memoria

Desbordamiento de la pila

La pila suele tener un tamaño máximo establecido

Si se sobrepasa se agota la pila



Gestión de la memoria dinámica

Gestión del montón

Sistema de Gestión de Memoria Dinámica (SGMD)

Gestiona la asignación de memoria a los datos dinámicos

Localiza secciones adecuadas y sigue la pista de lo disponible

No dispone de un *recolector de basura*, como el lenguaje Java

¡Hay que devolver toda la memoria solicitada!

Deben ejecutarse tantos `delete` como `new` se hayan ejecutado

La memoria disponible en el montón debe ser exactamente la misma antes y después de la ejecución del programa

Y todo dato dinámico debe tener algún acceso (puntero)

Es un grave error *perder* un dato en el montón



Errores Comunes

Mal uso de la memoria dinámica I

Olvido de destrucción de un dato dinámico

```
...  
int main() {  
    tRegistro *p;  
    p = new tRegistro;  
    *p = nuevo();  
    mostrar(*p);  
  
    return 0;  
}
```

G++ no indicará ningún error y el programa parecerá terminar correctamente, pero dejará memoria desperdiciada

Visual C++ sí comprueba el uso de la memoria dinámica y nos avisa si dejamos memoria sin liberar

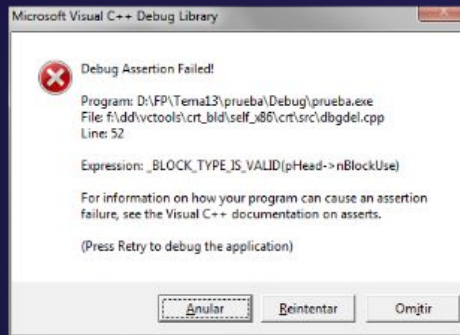
Mal uso de la memoria dinámica II

Intento de destrucción de un dato inexistente

```
...  
int main() {  
    tRegistro *p1 = new tRegistro;  
    *p1 = nuevo();  
    mostrar(*p1);  
    tRegistro *p2;  
    p2 = p1;  
    mostrar(*p2);  
    delete p1;  
    delete p2; ←  
  
    return 0;  
}
```



Sólo se ha creado
una variable



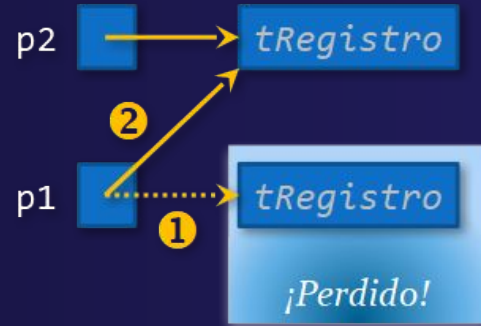
Mal uso de la memoria dinámica III

Pérdida de un dato dinámico

```
...  
int main() {  
    tRegistro *p1, *p2;  
    p1 = new tRegistro(nuevo());  
    p2 = new tRegistro(nuevo());  
  
    mostrar(*p1);  
    p1 = p2;  
    mostrar(*p1);  
  
    delete p1;  
    delete p2;  
  
    return 0;  
}
```

1

2



Se pierde un dato en el montón
Se intenta eliminar un dato ya eliminado

Mal uso de la memoria dinámica IV

Intento de acceso a un dato tras su eliminación

```
...  
int main() {  
    tRegistro *p;  
    p = new tRegistro(nuevo());  
  
    mostrar(*p);  
    delete p;  
    ...  
    mostrar(*p);  
  
    return 0;  
}
```



p ha dejado de apuntar
al dato dinámico destruido
→ Acceso a memoria inexistente

Arrays de datos dinámicos



Arrays de datos dinámicos

Arrays de punteros a datos dinámicos

```
typedef struct {  
    int codigo;  
    string nombre;  
    double valor;  
} tRegistro;  
typedef tRegistro *tRegPtr;
```

Los punteros ocupan
muy poco en memoria

Los datos a los que apunten
estarán en el montón

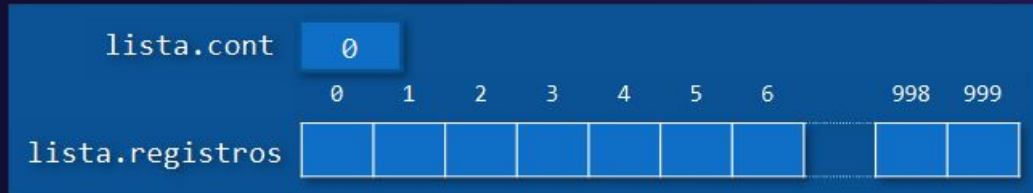
```
const int N = 1000;  
// Array de punteros a registros:  
typedef tRegPtr tArray[N];  
typedef struct {  
    tArray registros;  
    int cont;  
} tLista;
```

Se crean a medida que se insertan

Se destruyen a medida que se eliminan

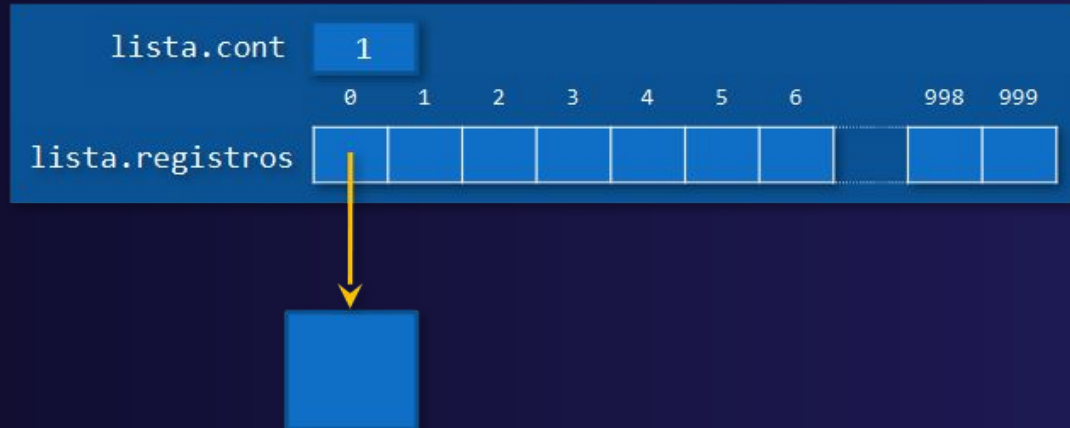
Arrays de datos dinámicos

```
tLista lista;  
lista.cont = 0;
```



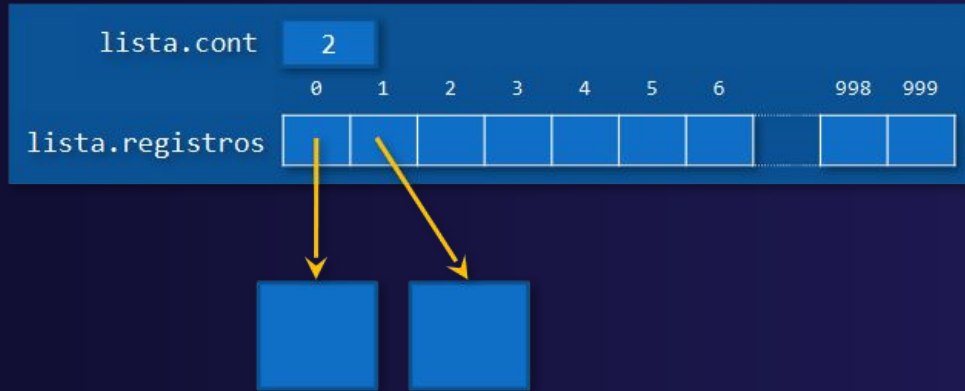
Arrays de datos dinámicos

```
tLista lista;  
lista.cont = 0;  
lista.registros[lista.cont] = new tRegistro(nuevo());  
lista.cont++;
```



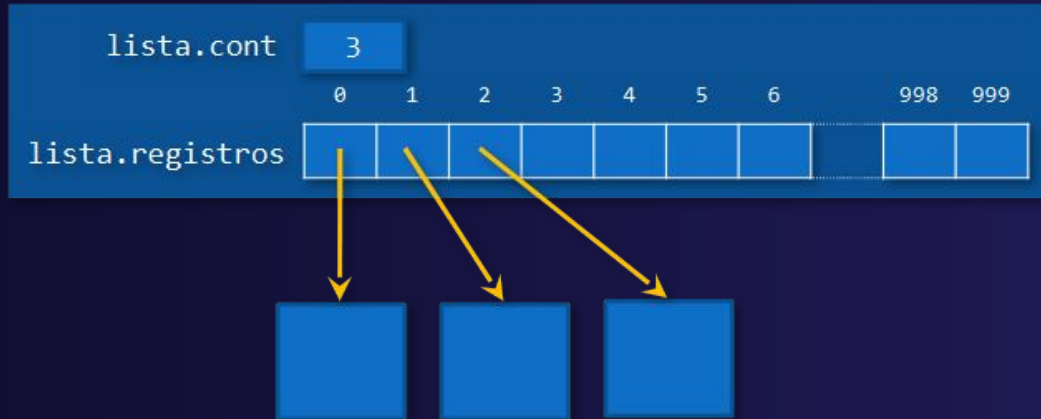
Arrays de datos dinámicos

```
tLista lista;  
lista.cont = 0;  
lista.registros[lista.cont] = new tRegistro(nuevo());  
lista.cont++;  
lista.registros[lista.cont] = new tRegistro(nuevo());  
lista.cont++;
```



Arrays de datos dinámicos

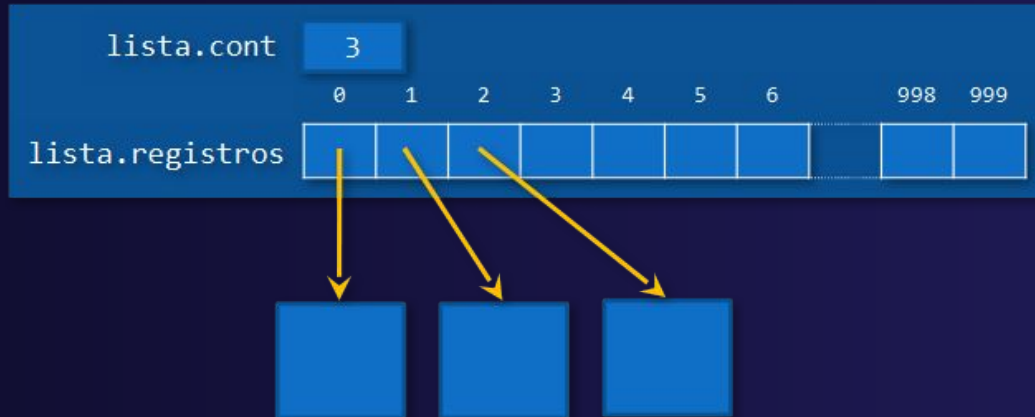
```
tLista lista;  
lista.cont = 0;  
lista.registros[lista.cont] = new tRegistro(nuevo());  
lista.cont++;  
lista.registros[lista.cont] = new tRegistro(nuevo());  
lista.cont++;  
lista.registros[lista.cont] = new tRegistro(nuevo());  
lista.cont++;
```



Arrays de datos dinámicos

Los registros se acceden a través de los punteros (operador ->):

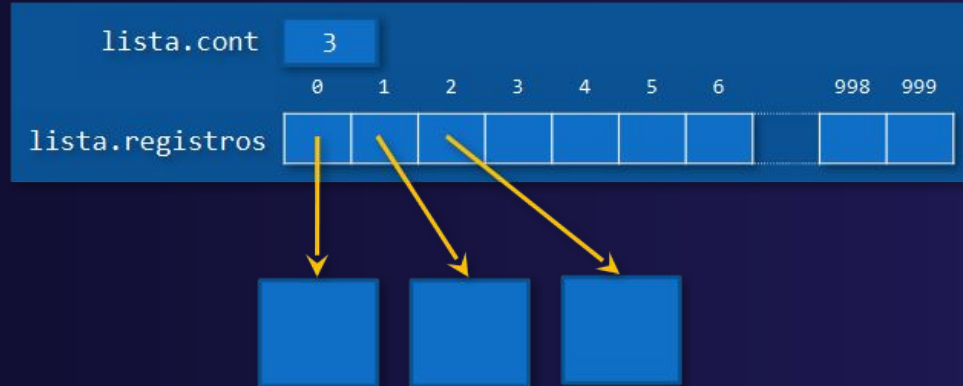
```
cout << lista.registros[0]->nombre;
```



Arrays de datos dinámicos

No hay que olvidarse de devolver la memoria al montón:

```
for (int i = 0; i < lista.cont; i++) {  
    delete lista.registros[i];  
}
```



```
#ifndef lista_h  
#define lista_h  
#include "registro.h"
```

```
const int N = 1000;  
const string BD = "bd.dat";  
typedef tRegPtr tArray[N];  
typedef struct {  
    tArray registros;  
    int cont;  
} tLista;
```

```
void mostrar(const tLista &lista);  
void insertar(tLista &lista, tRegistro registro, bool &ok);  
void eliminar(tLista &lista, int code, bool &ok);  
int buscar(const tLista &lista, int code);  
void cargar(tLista &lista, bool &ok);  
void guardar(const tLista &lista);  
void destruir(tLista &lista);
```

```
#endif
```

registro.h con el tipo puntero:

```
typedef tRegistro *tRegPtr;
```

```
void insertar(tLista &lista, tRegistro registro, bool &ok) {  
    ok = true;  
    if (lista.cont == N) {  
        ok = false;  
    }  
    else {  
        lista.registros[lista.cont] = new tRegistro(registro);  
        lista.cont++;  
    }  
}
```

```
void eliminar(tLista &lista, int code, bool &ok) {  
    ok = true;  
    int ind = buscar(lista, code);  
    if (ind == -1) {  
        ok = false;  
    }  
    else {  
        delete lista.registros[ind];  
        for (int i = ind + 1; i < lista.cont; i++) {  
            lista.registros[i - 1] = lista.registros[i];  
        }  
        lista.cont--;  
    }  
}
```

```
int buscar(const tLista &lista, int code) {  
    // Devuelve el índice o -1 si no se ha encontrado  
    int ind = 0;  
    bool encontrado = false;  
    while ((ind < lista.cont) && !encontrado) {  
        if (lista.registros[ind]->codigo == code) {  
            encontrado = true;  
        }  
        else {  
            ind++;  
        }  
    }  
    if (!encontrado) {  
        ind = -1;  
    }  
    return ind;  
}  
  
void destruir(tLista &lista) {  
    for (int i = 0; i < lista.cont; i++) {  
        delete lista.registros[i];  
    }  
    lista.cont = 0;  
}  
...
```

```

#include <iostream>
using namespace std;
#include "registro.h"
#include "lista.h"

int main() {
    tLista lista;
    tLista lista;
    bool ok;
    bool ok;
    cargar(lista, ok);
    cargar(lista, ok);
    if (ok) {
        mostrar(lista);
        mostrar(lista);
        destruir(lista);
    }
    return 0;
}
return 0;
}

```

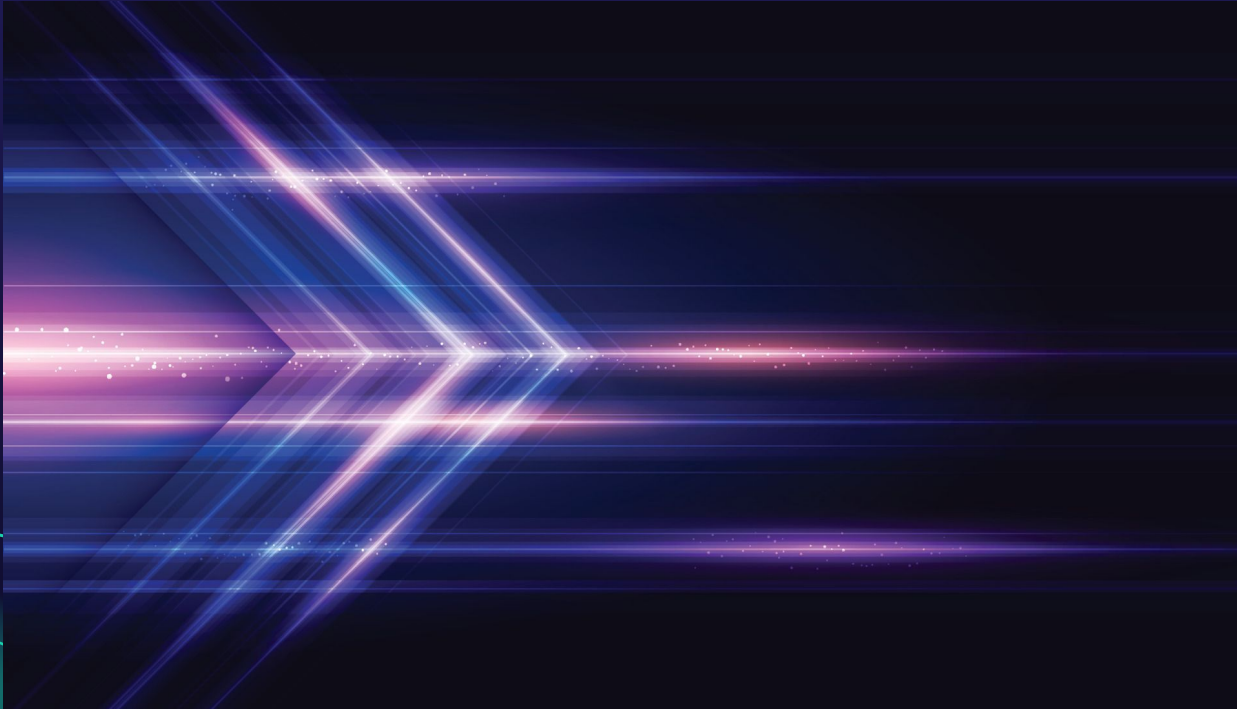
Elementos de la lista:

```

-----
12345 - Disco duro - 123.59 euros
324356 - Placa base core i7 - 234.50 euros
2121 - Multipuerto USB - 15.00 euros
54354 - Disco externo 500 Gb - 95.00 euros
112341 - Procesador AMD - 132.95 euros
66678325 - Marco digital 2 Gb - 78.99 euros
600673 - Monitor 22" Nisu - 154.50 euros

```

Arrays dinámicos



Creación y destrucción de arrays dinámicos

Array dinámico: array que se ubica en la memoria dinámica

Creación de un array dinámico:

```
tipo *puntero = new tipo[dimensión];
```

```
int *p = new int[10];
```

Crea un array de 10 *int* en memoria dinámica

Los elementos se acceden a través del puntero: `p[i]`

Destrucción del array:

```
delete [] p;
```

```
#include <iostream>
using namespace std;
const int N = 10;
```

```
int main() {
    int *p = new int[N];
    for (int i = 0; i < N; i++) {
        p[i] = i;
    }
    for (int i = 0; i < N; i++) {
        cout << p[i] << endl;
    }
    delete [] p;
    return 0;
}
```



¡No olvides destruir el array dinámico!

```
...  
#include "registro.h"  
  
const int N = 1000;  
  
// Lista: array dinámico (puntero) y contador  
typedef struct {  
    tRegPtr registros;  
    int cont;  
} tLista;  
  
...
```

```
void insertar(tLista &lista, tRegistro registro, bool &ok) {  
    ok = true;  
    if (lista.cont == N) {  
        ok = false;  
    }  
    else {  
        lista.registros[lista.cont] = registro;  
        lista.cont++;  
    }  
}
```

No usamos `new`

Se han creado todo
el array al cargar

```
void eliminar(tLista &lista, int code, bool &ok) {  
    ok = true;  
    int ind = buscar(lista, code);  
    if (ind == -1) {  
        ok = false;  
    }  
    else {  
        for (int i = ind + 1; i < lista.cont; i++) {  
            lista.registros[i - 1] = lista.registros[i];  
        }  
        lista.cont--;  
    }  
} ...
```

No usamos `delete`

Se destruye todo
el array al final

```
int buscar(tLista lista, int code) {
    int ind = 0;
    bool encontrado = false;
    while ((ind < lista.cont) && !encontrado) {
        if (lista.registros[ind].codigo == code) {
            encontrado = true;
        }
        else {
            ind++;
        }
    }
    if (!encontrado) {
        ind = -1;
    }
    return ind;
}

void destruir(tLista &lista) {
    delete [] lista.registros;
    lista.cont = 0;
}

...
```

```
void cargar(tLista &lista, bool &ok) {
    ifstream archivo;
    char aux;
    ok = true;
    archivo.open(BD.c_str());
    if (!archivo.is_open()) {
        ok = false;
    }
    else {
        tRegistro registro;
        lista.cont = 0;
        lista.registros = new tRegistro[N];
        archivo >> registro.codigo;
        while ((registro.codigo != -1) && (lista.cont < N)) {
            archivo >> registro.valor;
            archivo.get(aux); // Saltamos el espacio
            getline(archivo, registro.nombre);
            lista.registros[lista.cont] = registro;
            lista.cont++;
            archivo >> registro.codigo;
        }
        archivo.close();
    }
}
```

Mismo programa principal que el del array de datos dinámicos
Pero incluyendo `listaAD.h`, en lugar de `lista.h`

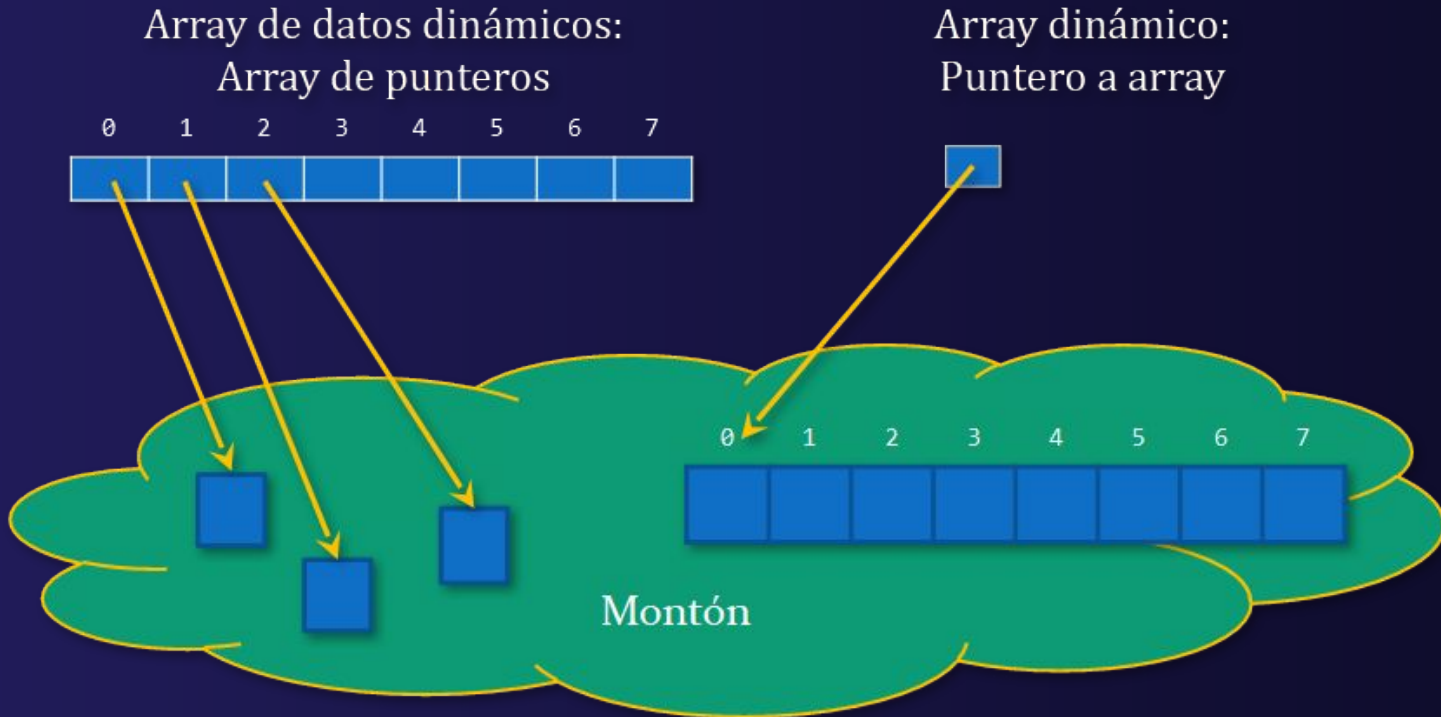
```
Elementos de la lista:
```


```
-----  
12345 - Disco duro - 123.59 euros  
324356 - Placa base core i7 - 234.50 euros  
2121 - Multpuerto USB - 15.00 euros  
54354 - Disco externo 500 Gb - 95.00 euros  
112341 - Procesador AMD - 132.95 euros  
66678325 - Marco digital 2 Gb - 78.99 euros  
600673 - Monitor 22" Nisu - 154.50 euros
```


Arrays dinámicos vs. arrays de dinámicos

Array de datos dinámicos: Array de punteros a datos dinámicos

Array dinámico: Puntero a array en memoria dinámica





Hasta luego
nos vemos en la próxima clase