FACULTAD DE INFORMÁTICA

Objetos y memoria dinámica

TALLER

Programación orientada a objetos — Unidad 8

Autor: Luis Hernández Yáñez

Problemas con punteros

Punteros a objetos dinámicos (*resolución individual*)

Los siguientes programas no manejan adecuadamente los punteros. Descubre qué es lo que hacen mal y por qué.

Mal uso de punteros nº 1:

```
#include "persona.h"

int main() {
  Persona *p1, p2;
  p1 = new Persona();

  p2 = p1;
  p2->mostrar();
  delete p1;

  return 0;
}
```

Mal uso de punteros nº 2:

```
#include "persona.h"

int main() {
  Persona *p1 = new Persona();
  p1->mostrar();
  Persona *p2;
  p2 = p1;
  p2->mostrar();
  delete p1;
  delete p2;

  return 0;
}
```


Problemas con punteros

Mal uso de punteros nº 3:

```
#include "persona.h"

int main() {
  Persona *p1 = NULL;
  Persona *p2;
  p2 = new Persona();

  p1->mostrar();
  p2->mostrar();

  delete p1;
  delete p2;

  return 0;
}
```

Mal uso de punteros nº 4:

```
#include "persona.h"

int main() {
  Persona *p1, *p2;
  p1 = new Persona();
  p2 = new Persona();

  p1->mostrar();
  p1 = p2;
  p1->mostrar();

  delete p1;
  delete p2;

  return 0;
}
```


Problemas con punteros

Mal uso de punteros nº 5:

```
#include "persona.h"

int main() {
  Persona *p1;
  p1 = new Persona();
  p1->mostrar();

  p1 = new Persona();
  p1->mostrar();

  p1 = new Persona();
  p1->mostrar();
  delete p1;

  return 0;
}
```

Mal uso de punteros nº 6:

```
#include "persona.h"

int main() {
  Persona *p1;
  p1 = new Persona();

  p1->mostrar();
  delete p1;

  p1->mostrar();
  delete p1;

  return 0;
}
```

Objetos dinámicos

Otra vez la lista de **Publicacion**es (*resolución en grupo*)

Se trata de modificar la clase **Lista** (de **Publicacion**es) del taller anterior para que guarde objetos dinámicos en lugar de objetos estáticos como hacía entonces.

La interfaz deseada para la lista es la siguiente:

```
class Lista {
public:
  Lista();
  ~Lista();
  bool llena() const;
  bool vacia() const;
  int cont() const;
  bool insertar(Publicacion*);
  bool recuperar(int, Publicacion*&) const;
    // la posición, de 1 a cont()
```

En la Unidad 9 veremos cómo desarrollar constructores de copia y operadores de asignación para listas polimórficas

(*continúa*)

Objetos dinámicos

```
  bool eliminar(int);
    // la posición, de 1 a cont()
  void mostrar() const;
private:
  enum { MAX = 100 };
  Publicacion* _array[MAX];
  int _cont;
};
```

Hay que probar la clase exhaustivamente en una función **main()**.

[Para los ejercicios de este taller, y en adelante, activad la opción CodeGuard en Project | Options, con el fin de detectar los malos usos de la memoria dinámica.]

Atributos dinámicos

Seguimos con la lista de **Publicacion**es (*resolución en grupo*)

Modificad las clases de la jerarquía de clases de publicaciones para la biblioteca de forma que los atributos de las clases que hemos creado nosotros (**Fecha** y **Persona**) sean atributos dinámicos.

Probad de nuevo la clase **Lista** para comprobar estas modificaciones.

Atributos dinámicos

Clase **Cuenta** con lista de **Movimiento**s (*resolución en grupo*)

Se trata de desarrollar una clase **Cuenta** con los siguientes atributos:

- **_cliente**: puntero a **Persona** (atributo dinámico)
- **_fecha**: puntero a **Fecha** (atributo dinámico)
- **_saldo**: un **double**
- **_movs**: puntero a **Lista** (lista de **Movimiento**s) (atributo dinámico)

Además de accedentes y mutadores para los tres primeros atributos, la clase **Cuenta** debe disponer de un método **nuevo(Movimiento*)** que permita insertar un movimiento en la lista **_movs**. La lista mantendrá los últimos 15 movimientos. [*Pero todavía no sabemos cómo actualizar el saldo.*]

Habrá un método **mostrar()** que muestre los datos básicos de la cuenta y otro **extracto()** que además de los datos básicos también muestre la lista de los últimos movimientos.

De momento no habrá constructor de copia ni operador de asignación [*en la Unidad 9 veremos cómo construirlos*].