FACULTAD DE INFORMÁTICA

Objetos y memoria dinámica

SOLUCIONES DEL 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;
}
```

`p2` NO es un puntero

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;
}
```

Sólo se ha creado un objeto dinámico

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;
}
```

`p1` NO apunta a nada (NULL)

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;
}
```

Se ha perdido el objeto dinámico creado con `p1`

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;
}
```

3 `new` y un sólo `delete`

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;
}
```

Uso y eliminación del objeto dinámico después de eliminarlo

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*)


```
  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.]


```
#ifndef listabib_h
#define listabib_h
#include "publicacion.h"
class Lista {
public:
  Lista();
  ~Lista();
  bool llena() const;
  bool vacia() const;
  int cont() const;
  bool insertar(Publicacion*);
  bool recuperar(int, Publicacion*&) const; // de 1 a cont()
  bool eliminar(int); // la posición, de 1 a cont()
  void mostrar() const;
private:
  enum { MAX = 100 };
  Publicacion *_array[MAX];
  int _cont;
};
#endif
```

>>> listabib.h


```
// Clase Lista (de publicaciones) - Implementación "listabib.cpp"
#include <iostream>
#include <string>
using namespace std;
#include "listabib.h"

Lista::Lista() : _cont(0) {}

Lista::~Lista() {
  for(int i = 0; i < _cont; i++) delete _array[i];
}

bool Lista::llena() const { return _cont == MAX; }

bool Lista::vacia() const { return _cont == 0; }

int Lista::cont() const { return _cont; }

bool Lista::insertar(Publicacion* p) {
  if(_cont == MAX) return false;
  _array[_cont] = p;
  _cont++;
  return true;
}
```

(*continúa*)

Objetos dinámicos

```
bool Lista::recuperar(int pos, Publicacion*& p) const {
  if(pos < 1 || pos > _cont) return false;
  p = _array[pos-1];
  return true;
}

bool Lista::eliminar(int pos) {
  if(pos < 1 || pos > _cont) return false;
  delete _array[pos-1];
  for(int i = pos; i < _cont; i++)
    _array[i-1] = _array[i]; // Ocupamos el hueco dejado
  _cont--;
  return true;
}

void Lista::mostrar() const {
  cout << "Elementos de la lista:" << endl;
  for(int i = 0; i < _cont; i++)
    _array[i]->mostrar();
}
```


Objetos dinámicos

```
// Prueba de la clase Lista (de publicaciones)

#include "articulo.h"
#include "libro.h"
#include "enciclopedia.h"
#include "revista.h"
#include "actas.h"
#include "listabib.h"

int main() {
  Lista lista;

  Articulo *art = new Articulo("Genoma 2", "Londres", "Reino Unido",
                  Fecha(11,4,1995), Persona("", 43, "John", "Maverick"),
                  "BioScience", 134, 161);
  lista.insertar(art);

  Libro *lib = new Libro("Rasputín", "Barcelona", "España",
               Fecha(15,7,1998),
               Persona("223344G", 31, "Juan", "Alan Gil"),
               "76-4235-5478-9", 1, "Ed. Magnus", 231);
  lista.insertar(lib);
```

(*continúa*)

Objetos dinámicos

```
  Enciclopedia* enc = new Enciclopedia("Guía médica", "Madrid", "España",
                      Fecha(1,12,2001),
                      Persona("123867X", 53, "Miguel", "Vegas Sanz"),
                      "78-3245-6665-8", 3, "MisterDoc", 14);
  lista.insertar(enc);

  Revista* rev = new Revista("Faralaes", "Sevilla", "España",
                 Fecha(1,8,1999), 5, 63, "Ed. Lunares");
  lista.insertar(rev);

  Actas* act = new Actas("Proceedings of IWC 2000", "Boston",
               "USA", Fecha(17,3,2001), 5, 63,
               "10th Int. Workshop on Computers");
  lista.insertar(act);

  lista.mostrar();

  Publicacion *p;
  lista.recuperar(1, p); // Ejemplo de recuperación de elemento
  p->mostrar();

  // Los objetos se eliminan al destruir la lista
  return 0;
}
```


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.

La clase **Publicacion** con atributo dinámico

```
// Clase Publicacion - Archivo de cabecera "publicacion.h"
#ifndef publicacion_h // Evitar inclusiones múltiples
#define publicacion_h

#include <iostream>
#include <string>
using namespace std;
#include "fecha.h"

class Publicacion {
public:
  Publicacion(string = "", string = "", string = "",
              Fecha = Fecha());
  // Datos: título, ciudad, país, fecha de edición.
  Publicacion(const Publicacion&);
  Publicacion& operator=(const Publicacion&);
  ~Publicacion();
```

(*continúa*)

La clase **Publicacion** con atributo dinámico

```
  // Accedentes
  string titulo() const;
  string ciudad() const;
  string pais() const;
  Fecha fecha() const;
  // Mutadores
  void titulo(string);
  void ciudad(string);
  void pais(string);
  void fecha(Fecha);
  // Otros métodos
  void leer();
  void mostrar() const;
private:
  string _titulo;
  string _ciudad, _pais;
  Fecha *_fecha;
};

#endif
```


La clase **Publicacion** con atributo dinámico

```
// Clase Publicacion - Implementación "publicacion.cpp"
#include "publicacion.h"

Publicacion::Publicacion(string tit, string c, string p, Fecha f)
  : _titulo(tit), _ciudad(c), _pais(p)
  { _fecha = new Fecha(); *_fecha = f; }

Publicacion::Publicacion(const Publicacion& otro) {
  _titulo = otro._titulo;
  _ciudad = otro._ciudad;
  _pais = otro._pais;
  _fecha = new Fecha();
  *_fecha = *otro._fecha;
}

Publicacion& Publicacion::operator=(const Publicacion& otro) {
  _titulo = otro._titulo;
  _ciudad = otro._ciudad;
  _pais = otro._pais;
  *_fecha = *otro._fecha;
  return *this;
}
```

(*continúa*)

La clase **Publicacion** con atributo dinámico

```
Publicacion::~Publicacion() { delete _fecha; }

string Publicacion::titulo() const { return _titulo; }

string Publicacion::ciudad() const { return _ciudad; }

string Publicacion::pais() const { return _pais; }

Fecha Publicacion::fecha() const { return *_fecha; }

void Publicacion::titulo(string t) { _titulo = t; }

void Publicacion::ciudad(string c) { _ciudad = c; }

void Publicacion::pais(string p) { _pais = p; }

void Publicacion::fecha(Fecha f) { *_fecha = f; }
```

(*continúa*)

La clase **Publicacion** con atributo dinámico

```
void Publicacion::leer() {
  cout << "Titulo: "; getline(cin, _titulo);
  cout << "Lugar de edicion:" << endl;
  cout << "  Ciudad: "; getline(cin, _ciudad);
  cout << "  Pais: "; getline(cin, _pais);
  cin.ignore();
  cout << "Fecha de edicion:" << endl;
  _fecha->leer();
}

void Publicacion::mostrar() const {
  cout << endl;
  cout << "Titulo: " << _titulo << endl;
  cout << "Lugar de edicion: " << _ciudad << " (" << _pais <<
  ")" << endl;
  cout << "Fecha de edicion: " << _fecha->completa() << endl;
}
```


La clase **Volumen** con atributo dinámico

```
// Clase Volumen - Archivo de cabecera "volumen.h"
#ifndef volumen_h // Evitar inclusiones múltiples
#define volumen_h

#include <iostream>
#include <string>
using namespace std;
#include "persona.h"
#include "publicacion.h"

class Volumen : public Publicacion {
public:
  Volumen(string = "", string = "", string = "",
          Fecha = Fecha(), Persona = Persona());
  // Datos: título, ciudad, país, fecha de edición, autor.
  Volumen(const Volumen&);
  Volumen& operator=(const Volumen&);
  ~Volumen();
```

(*continúa*)

La clase **Volumen** con atributo dinámico

```
  // Accedente
  Persona autor() const;
  // Mutador
  void autor(Persona);
  // Otros métodos
  void leer();
  void mostrar() const;
private:
  Persona *_autor;
};

#endif
```


La clase **Volumen** con atributo dinámico

```
// Clase Volumen - Implementación "volumen.cpp"
#include "volumen.h"

Volumen::Volumen(string tit, string c, string p, Fecha f,
  Persona a) : Publicacion(tit, c, p, f)
  { _autor = new Persona(); *_autor = a; }

Volumen::Volumen(const Volumen& otro) : Publicacion (otro) {
  _autor = new Persona();
  *_autor = *otro._autor;
}

Volumen& Volumen::operator=(const Volumen& otro) {
  Publicacion::operator=(otro);
  *_autor = *otro._autor;
  return *this;
}

Volumen::~Volumen() { delete _autor; }
```

(*continúa*)

La clase **Volumen** con atributo dinámico

```
Persona Volumen::autor() const { return *_autor; }

void Volumen::autor(Persona p) { *_autor = p; }

void Volumen::leer() {
  Publicacion::leer();
  cout << "Datos del autor:" << endl;
  _autor->leer();
}

void Volumen::mostrar() const {
  Publicacion::mostrar();
  cout << "Autor: " << _autor->nombreCompleto() << endl;
}
```

Como programa de prueba sirve EXACTAMENTE el mismo de antes.

La implementación de los atributos como atributos dinámicos NO afecta para nada al uso de las clases.

Atributos dinámicos

Clase **Cuenta** con lista de **Movimientos** (*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 **Movimientos**) (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*].

La clase **Cuenta**

```
// Clase Cuenta - Archivo de cabecera "cuenta.h"
#ifndef cuenta_h // Evitar inclusiones múltiples
#define cuenta_h

#include <iostream>
#include <string>
using namespace std;
#include "persona.h"
#include "fecha.h"
#include "listamov.h"

class Cuenta {
public:
  Cuenta(Persona = Persona(), Fecha = Fecha(), double = 0);
  // En la siguiente unidad veremos cómo implementar
  // el constructor de copia y el operador de asignación.
  ~Cuenta();
  Persona cliente() const;
  void cliente(Persona);
```

(*continúa*)

La clase **Cuenta**

```
  Fecha fecha() const;
  void fecha(Fecha);
  double saldo() const;
  void saldo(double);
  void nuevo(Movimiento*);
  void mostrar() const;
  void extracto() const;
private:
  Persona *_cliente;
  Fecha *_fecha;
  double _saldo;
  Lista *_movs;
};

#endif
```

La clase **Cuenta**

```
// Clase Cuenta - Implementación "cuenta.h"
#include "cuenta.h"

Cuenta::Cuenta(Persona cliente, Fecha fecha, double saldo)
  : _saldo(saldo) {
  _cliente = new Persona(cliente);
  _fecha = new Fecha(fecha);
  _movs = new Lista();
}

Cuenta::~Cuenta() {
  delete _cliente;
  delete _fecha;
  delete _movs;
}

Persona Cuenta::cliente() const { return *_cliente; }

void Cuenta::cliente(Persona p) { *_cliente = p; }
```

(*continúa*)

La clase **Cuenta**

```
Fecha Cuenta::fecha() const { return *_fecha; }

void Cuenta::fecha(Fecha f) { *_fecha = f; }

double Cuenta::saldo() const { return _saldo; }

void Cuenta::saldo(double s) { _saldo = s; }

void Cuenta::nuevo(Movimiento* m) {
  _movs->insertar(m);
  // No sabemos todavía cómo actualizar el saldo
}
```

(*continúa*)

La clase **Cuenta**

```
void Cuenta::mostrar() const {
  cout << endl;
  cout << "Cliente: " << _cliente->nombreCompleto() << " ("
       << _cliente->nif() << "), " << _cliente->edad() << "
   años" << endl;
  cout << "Cuenta abierta el " << _fecha->completa() << endl;
  cout << "Saldo inicial: " << _saldo << " eur" << endl;
}

void Cuenta::extracto() const {
  mostrar();
  _movs->mostrar();
}
```


La clase **Lista** de **Movimientos**

```
// Clase Lista (de movimientos) - Cabecera "listamov.h"
#ifndef listamov_h
#define listamov_h
#include "movimiento.h"
class Lista {
public:
  Lista();
  // El constructor de copia y el operador de
  // asignación se añadirán en la siguiente unidad
  ~Lista();
  bool llena() const;
  bool vacia() const;
  void insertar(Movimiento*);
  bool recuperar(int, Movimiento*&) const;
  void mostrar() const;
private:
  enum { MAX = 15 };
  Movimiento* _array[MAX];
  int _cont;
};
#endif
```

La clase **Lista** de **Movimientos**

```
// Clase Lista (de movimientos) - Implementación "listamov.cpp"
#include <iostream>
#include <string>
using namespace std;
#include "listamov.h"

Lista::Lista() : _cont(0) {}

Lista::~Lista() {
  for(int i = 0; i < _cont; i++) delete _array[i];}

bool Lista::llena() const { return _cont == MAX; }

bool Lista::vacia() const { return _cont == 0; }

void Lista::insertar(Movimiento* p) {
  if(_cont == MAX) {
    delete _array[0];
    for(int i = 0; i < MAX - 1; i++) _array[i] = _array[i+1];
    _array[MAX-1] = p;
  }
```

(*continúa*)

La clase **Lista** de **Movimientos**

```
  else {
    _array[_cont] = p;
    _cont++;
  }
}

bool Lista::recuperar(int pos, Movimiento*& p) const {
  if(pos < 1 || pos > _cont) return false;
  p = _array[pos-1];
  return true;
}

void Lista::mostrar() const {
  cout << endl;
  for(int i = 0; i < _cont; i++)
    _array[i]->mostrar();
  cout << endl;
}
```


Prueba de la clase **Cuenta**

```
#include <iostream>
#include <string>
using namespace std;

#include "fechahora.h"
#include "efectivo.h"
#include "transferencia.h"
#include "nomina.h"
#include "recibo.h"
#include "cajero.h"
#include "listamov.h"
#include "cuenta.h"

int main() {
  Cuenta cc(Persona("223344G", 25, "Javier", "Vegas Vegas"),
            Fecha(23, 11, 2002), 100);
  FechaHora fh(Fecha(29,11,2002),Hora(10,36,59));
  cc.nuevo(new Efectivo(125, fh));
```

(*continúa*)

Prueba de la clase **Cuenta**

```
  fh = fh + 90012; // Un tiempo después
  cc.nuevo(new Transferencia(325.60, fh));

  fh = fh + 70235; // Un tiempo después
  cc.nuevo(new Cajero(50, fh));

  fh = fh + 215014; // Un tiempo después
  cc.nuevo(new Nomina(1025.73, fh));

  fh = fh + 100001; // Un tiempo después
  cc.nuevo(new Recibo(120, fh));

  cc.extracto();

  return 0;
}
```