FACULTAD DE INFORMÁTICA

Arrays y cadenas de caracteres

SOLUCIONES DEL TALLER

Programación orientada a objetos — Unidad 4

Autor: Luis Hernández Yáñez

FdI UCM

Cuestiones

Sobre los arrays (*resolución individual*)

¿Qué es lo que hace mal el siguiente programa?

```
#include <iostream>
using namespace std;
#define MAX 100

int main() {
  int lista[MAX];
  for(int i = 1; i <= MAX; i++) lista[i] = i;
  for(int i = 1; i <= MAX; i++) cout << lista[i] << endl;

  return 0;
}
```

¡¡¡ Y, sin embargo, parece que funciona !!!
(aunque lo mejor es no fiarse)

- Accede a un elemento inexistente del array. Los índices van de 0 a MAX-1 (ambos inclusive). Sin embargo, ese programa empieza en 1 y termina en MAX. No existe la posición MAX, por lo que no hay tal elemento. Tampoco procesa el elemento en la posición 0.

FdI UCM

Cuestiones

¿Qué es lo que muestra el siguiente programa?

```
#include <iostream>
using namespace std;
#define MAX 10

typedef int Lista[MAX];

void muestra(Lista);
void acumula(Lista);

int main() {
  Lista lista;
  for(int i = 0; i < MAX; i++) lista[i] = i;
  muestra(lista);
  acumula(lista);
  muestra(lista);

  return 0;
}
```

(*continúa*)

FdI UCM

Cuestiones

```
void muestra(Lista array) {
  for(int i = 0; i < MAX; i++) cout << array[i] << " ";
  cout << endl;
}

void acumula(Lista array) {
  for(int i = 0; i < MAX; i++) array[i] += 2;
}
```

- Las funciones reciben siempre los arrays por referencia, por lo que cualquier modificación que sufran dentro de una función, se ve reflejada en el array argumento.
Después de la llamada a **acumula()**, el array ha quedado modificado.

```
0 1 2 3 4 5 6 7 8 9
2 3 4 5 6 7 8 9 10 11
```

Cómo pasa el tiempo (*resolución en grupo*)

Resulta muy habitual que en los programas se tengan que manejar instantes de tiempo (fechas y horas). Por esto, os proporciono dos clases: **Fecha** y **Hora**. La primera implementa el tipo **Fecha** que permite manipular fechas. La segunda implementa el tipo **Hora** que permite manipular instantes de tiempo de un día.

A continuación se muestran las interfaces de esas clases.

Antes de realizar el ejercicio que se plantea tras las definiciones de las clases, jugad un poco con ellas para comprobar cómo funcionan.

La clase **Fecha**

```
// Clase Fecha - Archivo de cabecera "fecha.h"

#ifndef fecha_h // Evitar inclusiones múltiples
#define fecha_h

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

class Fecha {
public:
  Fecha(int = 1, int = 1, int = 1900);
    // Constructor (día, mes, año)
  Fecha(const Fecha&);
  Fecha& operator=(const Fecha&);
  ~Fecha();

  int dia() const;
  int mes() const;  // Accedentes
  int anio() const;
```

(*continúa*)


```
  void dia(int);
  void mes(int);    // Mutadores
  void anio(int);

  long int operator-(Fecha) const;
    // Días transcurridos entre la fecha y la proporcionada
  Fecha operator+(long int) const; // Fecha más días
  Fecha operator-(long int) const; // Fecha menos días
  Fecha& operator+=(long int); // Abreviatura para la suma
  Fecha& operator-=(long int); // Abreviatura para la resta
  Fecha& operator++(); // Día siguiente
  Fecha& operator++(int); // (variante postfija)
  Fecha& operator--(); // Día anterior
  Fecha& operator--(int); // (variante postfija)

  bool operator==(Fecha) const;
  bool operator!=(Fecha) const;
  bool operator<(Fecha) const;   // Operadores relacionales
  bool operator<=(Fecha) const;
  bool operator>(Fecha) const;
  bool operator>=(Fecha) const;
```

(*continúa*)


```
  bool bisiesto() const;
  string diaSemana() const; // Devuelve el día de la semana
  string completa() const;
    // Devuelve la fecha en formato largo:
    // <día de la semana> <día> de <nombre del mes> de <año>
    // Por ejemplo: sabado 1 de abril de 2000
  string reducida() const;
    // Devuelve la fecha en formato corto:
    // <día>/<mes>/<año> (día y mes con 2 dígitos)
    // Por ejemplo: 01/04/2000
  void leer();

private:
  // No es necesario conocer esta sección para usar la clase
};

#endif
```

Composición de tipos

La clase **Hora**

```
// Clase Hora - Archivo de cabecera "hora.h"

#ifndef hora_h // Evitar inclusiones múltiples
#define hora_h

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

class Hora {
public:
   Hora(int = 0, int = 0, int = 0);
     // Constructor (horas, minutos, segundos)
  Hora(const Hora&);
  Hora& operator=(const Hora&);
  ~Hora();
  Hora(long int); // Otro constructor:construye la hora
    // desde la medianoche pasados los segundos indicados.
```

(*continúa*)

Composición de tipos

```
void horas(int);
void minutos(int);   // Mutadores
void segundos(int);

int horas() const;
int minutos() const; // Accedentes
int segundos() const;

long int operator-(Hora) const;
  // Segundos transcurridos entre la hora y la proporcionada
Hora operator+(long int) const; // Hora más segundos
Hora operator-(long int) const; // Hora menos segundos
Hora& operator+=(long int); // Abreviatura para la suma
Hora& operator-=(long int); // Abreviatura para la resta
Hora& operator++(); // Un segundo más
Hora& operator++(int); // (versión postfija)
Hora& operator--(); // Un segundo menos
Hora& operator--(int); // (versión postfija)
```

(*continúa*)

Composición de tipos

```
  bool operator==(Hora) const;
  bool operator!=(Hora) const;
  bool operator<(Hora) const;    // Operadores relacionales
  bool operator<=(Hora) const;
  bool operator>(Hora) const;
  bool operator>=(Hora) const;

  string hora() const; // Devuelve la hora en una cadena
  void leer();

private:
  // No es necesario conocer esta sección para usar la clase
};

#endif
```


Composición de tipos

Si queremos trabajar en los programas con instantes históricos, necesitamos las dos clases anteriores, ya que se ha de conocer la hora concreta de la fecha concreta en que se produce cada evento.

Aunque podríamos mantener la fecha y la hora por separado, resulta mucho más adecuado crear otra clase, **FechaHora**, que se encargue de implementar los instantes históricos. Así, por ejemplo, cuando se incremente la hora en un segundo y llegue a ser 00:00:00, se pasará automáticamente al día siguiente.

Así pues, lo que se pide es desarrollar esa nueva clase **FechaHora**. A continuación se muestra la interfaz de la clase que se quiere que implementéis.
De lo que os tenéis que encargar vosotros es de todo lo que falta.

Algunas indicaciones

- ✓ Necesitaréis una constante que indique el número de segundos por día (86.400). Será un atributo de clase (compartido).
- ✓ Al sumar (restar) un número de segundos hay que calcular el número de días completos y quedarse con los segundos restantes, para luego proceder a sumar (restar) los días a la fecha y sumar (restar) los segundos restantes a la hora.
- ✓ Al incrementar (decrementar) hay que tener en cuenta que se puede cambiar de fecha.


```
// Clase FechaHora - Archivo de cabecera "fechahora.h"
#ifndef fechahora_h // Evitar inclusiones múltiples
#define fechahora_h

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

class FechaHora {
public:
  FechaHora(); // Constructor predeterminado
  FechaHora(const FechaHora&); // Constructor de copia
  FechaHora& operator=(const FechaHora&); // Op. asignación
  ~FechaHora(); // Destructor

  // Otros constructores:
  FechaHora(Fecha);
  FechaHora(Hora);
  FechaHora(Fecha, Hora);
```

(*continúa*)


```
  // Accedentes y mutadores:
  Fecha fecha() const;
  Hora hora() const;
  void fecha(Fecha);
  void hora(Hora);
  // Operadores aritméticos:
  long int operator-(FechaHora) const;
  // Segundos entre la FechaHora y la proporcionada
  long int operator-(Hora) const;
  // Segundos entre la hora y la proporcionada
  long int operator-(Fecha) const;
  // Días entre la fecha y la proporcionada
  FechaHora operator+(long int) const; // + segundos
  FechaHora operator-(long int) const; // - segundos
  FechaHora& operator++(); // Un segundo más
  FechaHora& operator--(); // Un segundo menos
  // Operadores relacionales:
  bool operator==(FechaHora) const;
  bool operator!=(FechaHora) const;
  bool operator<(FechaHora) const;
```

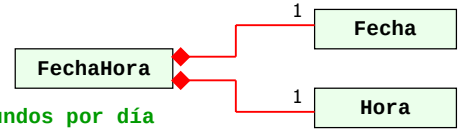
(*continúa*)


```
  bool operator<=(FechaHora) const;
  bool operator>(FechaHora) const;
  bool operator>=(FechaHora) const;

  // Otros métodos:
  long int aDias(long int); // Segundos a días
  long int aSegundos(long int); // Días a segundos
  string diaSemana() const;
  string completa() const;
    // Cadena con la fecha completa seguida de la hora
  string reducida() const;
    // Cadena con la fecha reducida seguida de la hora
  void leer();

private:
  Fecha _fecha;
  Hora _hora;
  static long int SPD; // Segundos por día
};

#endif
```



Composición de tipos

```
// Clase FechaHora - Implementación "fechahora.cpp"
#include "fechahora.h"

long int FechaHora::SPD = 86400; // Atributo de clase

FechaHora::FechaHora() { }

FechaHora::FechaHora(const FechaHora& otra) {
  _fecha = otra._fecha;
  _hora = otra._hora;
}

FechaHora& FechaHora::operator=(const FechaHora& otra) {
  _fecha = otra._fecha;
  _hora = otra._hora;
  return *this;
}

FechaHora::~FechaHora() { }

// Otros constructores:

FechaHora::FechaHora(Fecha f) : _fecha(f) { }
```

(*continúa*)

Composición de tipos

```
FechaHora::FechaHora(Hora h) : _hora(h) { }

FechaHora::FechaHora(Fecha f, Hora h) : _fecha(f), _hora(h) { }

// Accedentes y mutadores:

Fecha FechaHora::fecha() const { return _fecha; }

Hora FechaHora::hora() const { return _hora; }

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

void FechaHora::hora(Hora h) { _hora = h; }

// Operadores aritméticos:

long int FechaHora::operator-(FechaHora otra) const {
// Segundos entre la FechaHora y la proporcionada
  return (_fecha - otra._fecha) * SPD + (_hora - otra._hora);
}

long int FechaHora::operator-(Hora otra) const {
// Segundos entre la hora y la proporcionada
  return _hora - otra;
}
```

(*continúa*)

Composición de tipos

```
long int FechaHora::operator-(Fecha otra) const {
// Días entre la fecha y la proporcionada
  return _fecha - otra;
}

FechaHora FechaHora::operator+(long int seg) const {
// + segundos
  FechaHora fh;
  fh._fecha = _fecha + seg / SPD; // incremento en días
  seg = seg % SPD; // segundos restantes
  fh._hora = _hora + seg;
  if(fh._hora < _hora) fh._fecha++; // ¿cambio de día?
  return fh;
}

FechaHora FechaHora::operator-(long int seg) const {
// - segundos
  FechaHora fh;
  fh._fecha = _fecha - seg / SPD; // decremento en días
  seg = seg % SPD; // segundos restantes
  fh._hora = _hora - seg;
  if(fh._hora > _hora) fh._fecha--; // ¿cambio de día?
  return fh;
}
```

(*continúa*)

Composición de tipos

```
FechaHora& FechaHora::operator++() {
// Un segundo más
  _hora++;
  if(_hora == Hora(0,0,0)) _fecha++; // cambio de fecha
  return *this;
}

FechaHora& FechaHora::operator--() {
// Un segundo menos
  if(_hora == Hora(0,0,0)) _fecha--; // cambio de fecha
  _hora--;
  return *this;
}

// Operadores relacionales:

bool FechaHora::operator==(FechaHora otra) const {
  return (_fecha == otra._fecha) && (_hora == otra._hora); }
```

(*continúa*)

Composición de tipos

```
bool FechaHora::operator!=(FechaHora otra) const {
  return (_fecha != otra._fecha) || (_hora != otra._hora); }

bool FechaHora::operator<(FechaHora otra) const {
  return (_fecha < otra._fecha) ||
         (_fecha == otra._fecha) && (_hora < otra._hora); }

bool FechaHora::operator<=(FechaHora otra) const {
  return (_fecha < otra._fecha) ||
         (_fecha == otra._fecha) && (_hora <= otra._hora); }

bool FechaHora::operator>(FechaHora otra) const {
  return (_fecha > otra._fecha) ||
         (_fecha == otra._fecha) && (_hora > otra._hora); }

bool FechaHora::operator>=(FechaHora otra) const {
  return (_fecha > otra._fecha) ||
         (_fecha == otra._fecha) && (_hora >= otra._hora); }

// Otros métodos:

long int FechaHora::aDias(long int seg) { return seg / SPD; }
// Transforma esos segundos en días (completos)
```

(*continúa*)

Composición de tipos

```
long int FechaHora::aSegundos(long int dias)
{ return dias * SPD; } // Transforma esos días en segundos

string FechaHora::diaSemana() const {
// Devuelve el nombre del día de la semana
  return _fecha.diaSemana();
}

string FechaHora::completa() const {
// Ejemplo: "4 de noviembre de 2002 17:35:12"
  return _fecha.completa() + " " + _hora.hora();
}

string FechaHora::reducida() const {
// Ejemplo: "04/11/2002 17:35:12"
  return _fecha.reducida() + " " + _hora.hora();
}

void FechaHora::leer() {
  _fecha.leer();
  _hora.leer();
}
```


En el mundo real

Para una biblioteca (*resolución en grupo*)

Crea una clase **Libro** que modele la información que se mantiene en una biblioteca sobre cada libro: **título**, **autor** (usa la clase **Persona**), **ISBN**, **páginas**, **edición**, **editorial** , **lugar** (ciudad y país) y **fecha** de edición (usa la clase **Fecha** anterior). La clase se debe ajustar a la Forma canónica ortodoxa y proporcionar los siguientes servicios: accedentes y mutadores, método para leer la información y método para mostrar la información. Este último método mostrará la información del libro con este formato:

```
Título: C++ Cómo programar
2a. edición
Autor: Deitel, Harvey M.
ISBN: 0-13-528910-6
Prentice Hall, Méjico (Méjico), martes 16 de noviembre de 1999
1130 páginas
```

Prueba la clase en una función **main()**.

En el mundo real

```
// Clase Libro - Archivo de cabecera "libro.h"
#ifndef libro_h // Evitar inclusiones múltiples
#define libro_h

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

class Libro {
public:
  Libro();
  Libro(const Libro&);
  Libro& operator=(const Libro&);
  ~Libro();
  Libro(string, Persona, string, int, int,
        string, string, string, Fecha);
  // Datos: título, autor, ISBN, páginas, edición, editorial,
  //        ciudad, país, fecha de edición.
```

(*continúa*)

En el mundo real

```
  string titulo() const;
  Persona autor() const;
  string isbn() const;
  int paginas() const;
  int edicion() const;
  string editorial() const;
  string ciudad() const;
  string pais() const;
  Fecha fecha() const;

  void titulo(string);
  void autor(Persona);
  void isbn(string);
  void paginas(int);
  void edicion(int);
  void editorial(string);
  void ciudad(string);
  void pais(string);
  void fecha(Fecha);
```

(*continúa*)

En el mundo real

```
  void leer();
  void mostrar() const;

private:
  string _titulo;
  Persona _autor;
  string _isbn;
  int _paginas;
  int _edicion;
  string _editorial;
  string _ciudad, _pais;
  Fecha _fecha;
};

#endif
```

(*continúa*)

En el mundo real

```
// Clase Libro - Implementación "libro.cpp"

#include "libro.h"

Libro::Libro() : _paginas(0), _edicion(1) { }
// Los demás atributos se inicializan automáticamente
// (son objetos de otras clases - string, Persona o Fecha)

Libro::Libro(const Libro& otro) {
  _titulo = otro._titulo;
  _autor = otro._autor;
  _isbn = otro._isbn;
  _paginas = otro._paginas;
  _edicion = otro._edicion;
  _editorial = otro._editorial;
  _ciudad = otro._ciudad;
  _pais = otro._pais;
  _fecha = otro._fecha;
}
```

(*continúa*)

En el mundo real

```
Libro& Libro::operator=(const Libro& otro) {
  _titulo = otro._titulo;
  _autor = otro._autor;
  _isbn = otro._isbn;
  _paginas = otro._paginas;
  _edicion = otro._edicion;
  _editorial = otro._editorial;
  _ciudad = otro._ciudad;
  _pais = otro._pais;
  _fecha = otro._fecha;
  return *this;
}

Libro::~Libro() { }

Libro::Libro(string tit, Persona per, string isbn, int pag,
             int edic, string edit, string c, string p, Fecha f)
 : _titulo(tit),_autor(per), _isbn(isbn), _paginas(pag),
   _edicion(edic), _editorial(edit), _ciudad(c), _pais(p),
   _fecha(f) { }
```

(*continúa*)

En el mundo real

```
string Libro::titulo() const { return _titulo; }
Persona Libro::autor() const { return _autor; }
string Libro::isbn() const { return _isbn; }
int Libro::paginas() const { return _paginas; }
int Libro::edicion() const { return _edicion; }
string Libro::editorial() const { return _editorial; }
string Libro::ciudad() const { return _ciudad; }
string Libro::pais() const { return _pais; }
Fecha Libro::fecha() const { return _fecha; }

void Libro::titulo(string t) { _titulo = t; }
void Libro::autor(Persona a) { _autor = a; }
void Libro::isbn(string i) { _isbn = i; }
void Libro::paginas(int p) { _paginas = p; }
void Libro::edicion(int e) { _edicion = e; }
void Libro::editorial(string e) { _editorial = e; }
void Libro::ciudad(string c) { _ciudad = c; }
void Libro::pais(string p) { _pais = p; }
void Libro::fecha(Fecha f) { _fecha = f; }
```

(*continúa*)

En el mundo real

```
void Libro::leer() {
// El autor y la fecha se habrán leído en otro sitio
  cout << "Título: "; getline(cin, _titulo);
  cout << "I.S.B.N.: "; getline(cin, _isbn);
  cout << "Número de páginas: "; cin >> _paginas;
  cout << "Número de edición: "; cin >> _edicion; cin.ignore();
  cout << "Editorial: "; getline(cin, _editorial);
  cout << "Lugar de edición:" << endl;
  cout << "  Ciudad: "; getline(cin, _ciudad);
  cout << "  País: "; getline(cin, _pais);
}

void Libro::mostrar() const {
  cout << endl << "Título: " << _titulo << endl;
  cout << _edicion << "a. edición" << endl;
  cout << "Autor: " << _autor.apellidos() << ", "
       << _autor.nombre() << endl;
  cout << "ISBN: " << _isbn << endl;
  cout << _editorial << ", " << _ciudad << " (" << _pais
       << "), " << _fecha.completa() << endl;
  cout << _paginas << " páginas" << endl;
}
```


En el mundo real

```
// Prueba de la clase Libro
#include <iostream>
#include <string>
using namespace std;
#include "persona.h"
#include "fecha.h"
#include "libro.h"

int main() {
  Libro libro1;
  libro1.mostrar(); // Inicialización automática de atributos
  Libro libro2("C++: Cómo programar",
               Persona("", 35, "Harvey M.", "Deitel"),
               "0-13-528910-6", 1130, 2, "Prentice Hall",
               "Méjico", "Méjico", Fecha(16,11,1999));
  libro2.mostrar();
  Libro libro3; libro3.leer();
  Persona per; per.leer();
  libro3.autor(per);
  Fecha f; f.leer();
  libro3.fecha(f);
  libro3.mostrar();
  return 0;
}
```