



Universidad de Córdoba
Escuela Politécnica Superior de Córdoba

ESTRUCTURAS DE DATOS
GRADO EN INGENIERÍA INFORMÁTICA
Segundo curso. Segundo cuatrimestre.



Curso académico 2017 – 2018

SEGUNDA PRÁCTICA

PROVINCIA COMPUESTA POR UNA LISTA DE MUNICIPIOS

- **Objetivo**
 - Los municipios se ordenarán por el nombre.
 - Implementar el tipo abstracto de datos **Provincia** utilizando una lista doblemente enlazada y ordenada de municipios
 - No hay municipios repetidos
- **Desarrollo de la práctica número 2**
 - Duración de la práctica 2: tres sesiones de dos horas cada una.
 - **Plazo máximo de entrega**
 - **16:00 horas del 9 de abril de 2018**
 - Se deberá subir un fichero comprimido denominado “**practica-2-usuario.zip**”, donde “usuario” es el *login* de cada estudiante.
 - El fichero comprimido contendrá
 - makefile
 - Doxyfile
 - ficheros hpp
 - ficheros cpp
 - ficheros txt de ejemplo
 - **Observación:**
 - Se debe usar el espacio de nombres de la asignatura: **ed**
- **Evaluación**
 - La calificación de la práctica se basará
 - en la calidad y completitud del trabajo realizado.
 - y en la **defensa presencial de cada estudiante**.
 - **Se valorará**
 - La correcta implementación de las **clases**
 - **Provincia**
 - **ListaDoblementeEnlazadaOrdenadaMunicipios**
 - La codificación de las **funciones auxiliares** del programa principal.
 - El correcto funcionamiento de los programas principales propuestos como ejemplos.
 - principalProvincia.cpp
 - testProvincia.cpp

- La ampliación y mejora del menú del programa principal para añadir más opciones.
 - La documentación del código con doxygen.
 - La claridad del código.
 - El uso de macros de pantalla para mejorar la visualización de la información
 - **Y sobre todo**
 - Un profundo conocimiento de la práctica codificada.
- **Descripción de los ficheros**
 - **Introducción**
 - Se proporciona un fichero comprimido denominado “**practica-2-usuario.zip**” con los ficheros que se describen a continuación
 - El Estudiante debe completar o revisar el código de los ficheros que se indican.
 - **Fichero con una Provincia de ejemplo**
 - Cordoba.txt
 - Este fichero es una modificación del fichero Cordoba.csv descargado desde el Instituto Nacional de Estadística
 - <http://www.ine.es/jaxiT3/Tabla.htm?t=2901>
 - **makefile**
 - Facilita la compilación de los ficheros, la generación de la documentación con doxygen y el borrado de ficheros que ya no sean necesarios (*.o, *~, etc.)
 - **makefile2**
 - Facilita la compilación del segundo programa de prueba
 - testProvincia.cpp
 - **Doxyfile**
 - Fichero de configuración para generar la documentación con doxygen
 - El Estudiante puede modificar este fichero para mejorar su documentación.
 - **principalProvincia.cpp**
 - Programa principal utilizado como ejemplo para comprobar el funcionamiento de la clase Provincia
 - Utiliza macros de pantalla para mejorar la visualización de la información.
 - El Estudiante puede ampliar y mejorar este programa de ejemplo.
 - **testProvincia.cpp**
 - Segundo programa principal utilizado para comprobar la inserción y el borrado en la lista de municipios.
 - **macros.hpp**
 - Fichero con macros para mejorar la visualización de la información en la pantalla.
 - **funcionesAuxiliares.hpp**
 - Incluye los prototipos de funciones auxiliares utilizadas en el fichero **principalProvincia.cpp**
 - **Importante**

- El estudiante debe completar los comentarios de **doxygen**
- **funcionesAuxiliares.cpp**
 - Código complementario de las funciones auxiliares utilizadas en el programa principal
 - **Importante**
 - El estudiante debe completar este código
- **Provincia.hpp**
 - Definición de la clase Provincia
 - **Importante**
 - El estudiante debe completar este código
- **Provincia.cpp**
 - Código auxiliar de la clase Provincia
 - **Importante**
 - Cada estudiante debe completar este código
- **ListaOrdenadaMunicipiosInterfaz.hpp**
 - Definición de la clase abstracta **ListaOrdenadaMunicipiosInterfaz**
 - **Importante**
 - Cada estudiante debe revisar este código
- **ListaDoblementeEnlazadaOrdenadaMunicipios.hpp**
 - Definición de la clase **ListaDoblementeEnlazadaOrdenadaMunicipios**
 - **Importante**
 - Cada estudiante debe completar este código
- **NodoMunicipioInterfaz.hpp**
 - Definición de la clase abstracta **nodoMunicipioInterfaz**
- **NodoDoblementeEnlazadoMunicipio.hpp**
 - Definición de la clase **NodoDoblementeEnlazadoMunicipio** que hereda de la clase **NodoMunicipioInterfaz**
- **Municipio.hpp**
 - Definición de la clase **Municipio**
- **Municipio.cpp**
 - Código complementario de las funciones de la clase **Municipio**
- **Clases**
 - **Provincia**
 - **Código que debe codificar cada estudiante**
 - **ListaOrdenadaMunicipiosInterfaz**
 - Clase abstracta codificada por el profesor
 - **ListaDoblementeEnlazadaOrdenadaMunicipios**
 - **Clase que debe codificar cada estudiante**
 - **NodoMunicipioInterfaz**
 - Clase abstracta codificada por el profesor

- **NodoDoblementeEnlazadoMunicipio**
 - Clase codificada por el profesor
- **Municipio**
 - Clase codificada por el profesor
- **Especificación de las clases**
 - Clase **Provincia**
 - **Atributos**
 - Nombre de la Provincia
 - Código de la Provincia
 - **Lista de Municipios doblemente enlazada y ordenada por el nombre**
 - **Constructores**
 - Constructor parametrizado
 - *Provincia(nombre: Cadena; codigo: Entero)*
 - Postcondición
 - *getNombre() == nombre*
 - *getCodigo() == codigo*
 - *hayMunicipios() == falso*
 - **Observadores**
 - *Cadena getNombre()*
 - Devuelve el nombre de la provincia
 - *Cadena getCodigo()*
 - Devuelve el código de la provincia
 - *Lógico hayMunicipios()*
 - Indica si la provincia tiene o no municipios
 - *Entero getNumeroMunicipios()*
 - Devuelve el número de municipio de la provincia
 - *Lógico existeMunicipio(nombre: Cadena)*
 - Indica si el municipio con el nombre indicado pertenece a la provincia
 - *Municipio getMunicipio(nombre: Cadena)*
 - Devuelve el municipio con el nombre indicado
 - Precondición
 - *existeMunicipio(nombre) == verdadero*
 - *Entero getTotalHombres()*
 - Devuelve el número total de hombres de la provincia
 - *Entero getTotalMujeres()*
 - Devuelve el número total de mujeres de la provincia
 - *Entero getTotalHabitantes()*
 - Devuelve el número total de habitantes de la provincia
 - Postcondición
 - *valorDevuelto == getTotalHombres() + getTotalMujeres()*
 - **Modificadores**
 - *setNombre(nombre: Cadena)*
 - Asigna un nuevo nombre a la provincia
 - Postcondición
 - *getNombre() == nombre*

- *setCodigo(numero: Entero)*
 - Asigna un nuevo código a la provincia
 - Postcondición
 - *getCodigo() == numero*
- *insertarMunicipio(municipio: Municipio)*
 - Insertar un municipio en la provincia.
 - Precondición
 - *existeMunicipio(municipio.getNombre()) == falso*
 - Postcondición
 - *existeMunicipio(municipio.getNombre()) == verdadero*
 - *getNumeroMunicipios() = old.getNumeroMunicipios() + 1*
- *borrarMunicipio(nombre: Cadena)*
 - Borra de la provincia el municipio con el nombre indicado
 - Precondición
 - *existeMunicipio(municipio.getNombre()) == verdadero*
 - Postcondición
 - *existeMunicipio(municipio.getNombre()) == false*
 - *getNumeroMunicipios() = old.getNumeroMunicipios() - 1*
- *borrarTodosLosMunicipios()*
 - Borra todos los municipios de la provincia.
 - Postcondición
 - *hayMunicipios() == falso*
- **Función de escritura**
 - *escribirMunicipios()*
 - Escribe por pantalla la información de la provincia
 - Código y nombre de la provincia
 - Datos de cada municipio
- **Operaciones con ficheros**
 - *Lógico cargarFichero(nombre: Cadena)*
 - Carga los datos de una provincia desde un fichero
 - Formato
 - Código de provincia Nombre de provincia
 - *Código postal Nombre; Hombres; Mujeres; (Municipio 1)*
 - *Código postal Nombre; Hombres; Mujeres; (Municipio 2)*
 - Etc.
 - Devuelve verdadero, si se ha hecho la carga; falso, en caso contrario
 - *Lógico grabarFichero(nombre: Cadena)*
 - Graba los datos de una provincia en un fichero
 - Formato
 - Código de provincia Nombre de provincia
 - *Código postal Nombre; Hombres; Mujeres; (Municipio 1)*
 - *Código postal Nombre; Hombres; Mujeres; (Municipio 2)*
 - Etc.
 - Devuelve verdadero, si se ha hecho la grabación; falso, en caso contrario.
- Clase **ListaOrdenadaMunicipiosInterfaz**
 - **Descripción**

- Clase abstracta que especifica los métodos “**virtuales puros**” de una lista de municipios ordenada alfabéticamente por su nombre.
- Los métodos virtuales puros deben ser redefinidos en las clases herederas.
- **Observadores públicos virtuales puros**
 - *bool isEmpty()*
 - Comprueba si la lista está vacía
 - Devuelve verdadero, si la lista está vacía; falso, en caso contrario.
 - *Entero nItems()*
 - Devuelve el número de municipios
 - *Lógico isFirstItem()*
 - Comprueba si el cursor está en el primer nodo
 - Precondición
 - *isEmpty() == false*
 - *Lógico isLastItem()*
 - Comprueba si el cursor está en el último nodo
 - Precondición
 - *isEmpty() == false*
 - *Municipio getCurrentItem()*
 - Obtiene el municipio situado en el nodo actual
 - Precondición
 - *isEmpty() == false*
 - *Municipio getPreviousItem()*
 - Obtiene el municipio situado en el nodo anterior
 - Precondición
 - *isEmpty() == false*
 - *isFirstItem() == false*
 - *Municipio getNextItem()*
 - Obtiene el municipio situado en el nodo siguiente
 - Precondición
 - *isEmpty() == false*
 - *isLastItem() == false*
- **Modificadores públicos virtuales puros**
 - *gotoHead()*
 - Coloca el cursor en el primer nodo
 - Precondición
 - *isEmpty() == false*
 - Postcondición
 - *isFirstItem() == verdadero*
 - *gotoLast()*
 - Coloca el cursor en el último nodo
 - Precondición
 - *isEmpty() == false*
 - Postcondición
 - *isLastItem() == verdadero*
 - *gotoPrevious()*
 - Coloca el cursor en el nodo anterior

- Precondición
 - *isEmpty() == falso*
 - *isFirstItem() == falso*
- gotoNext()
 - Coloca el cursor en el nodo siguiente
 - Precondición
 - *isEmpty() == falso*
 - *isLastItem() == falso*
- Lógico find(ítem:Municipio)
 - Busca el municipio ítem.
 - Si existe, coloca el cursor en el nodo que contiene el ítem.
 - Si no existe, se coloca en el elemento que es mayor que el ítem buscado, o en el último elemento, si todos los elementos de la lista son menores que el elemento buscado
 - Postcondición
 - Si *valorDevuelto == verdadero* entonces *getCurrentItem() == ítem*
 - Si *valorDevuelto == falso* entonces *getCurrentItem > ítem* o *isLastItem() == verdadero*
- insert(ítem:Municipio)
 - Busca el municipio ítem.
 - Si existe, coloca el cursor en el nodo que contiene el ítem.
 - Si no existe, se coloca en el elemento que es mayor que el ítem buscado, o en el último elemento, si todos los elementos de la lista son menores que el elemento buscado
 - Postcondición
 - Si *valorDevuelto == verdadero* entonces *getCurrentItem() == ítem*
 - Si *valorDevuelto == falso* entonces *getCurrentItem > ítem* o *isLastItem() == verdadero*
 - Inserta un Municipio de forma ordenada por el nombre
 - Precondición
 - *find(ítem) == falso*
 - Postcondición
 - *getCurrentItem() == ítem*
 - *nItems() = old.nItems() + 1*
- void remove()
 - Borra el Municipio actual apuntado por el cursor
 - Precondición
 - *isEmpty() == false*
 - Postcondición
 - *nItems() == old.nItems() - 1*
 - Si *old.isFirstItem() == falso* y *old.isLastItem() == falso* entonces *old.getPreviousItem() == getPreviousItem()* y *old.getNextItem() == getCurrentItem()*
 - Si *old.isLastItem() == verdadero* entonces *isEmpty() == verdadero* o *old.getPreviousItem() == getCurrentItem()* y *isLastItem() == verdadero*
 - Si *old.isFirstItem() == verdadero* entonces

isEmpty() == verdadero
o *old.getNextItem() == getCurrentItem()* y *isFirstItem() == verdadero*

○ Clase **ListaDoblementeEnlazadaOrdenadaMunicipios**

▪ **Descripción**

- Esta clase hereda de forma pública de **ListaOrdenadaMunicipiosInterfaz**
- Lista ordenada y doblemente enlazada de elementos de la clase **Municipio**
- Los municipios están ordenados por nombre y no hay municipios con el mismo nombre.
- Se utiliza un cursor para recorrer la lista

▪ **Atributos**

- *_head*: puntero a un nodo del tipo **NodoDoblementeEnlazadoMunicipio**
- *_current*: puntero a un nodo del tipo **NodoDoblementeEnlazadoMunicipio**

▪ **Observadores privados**

- *getHead()*: puntero a **NodoDoblementeEnlazadoMunicipio**
 - Obtiene el puntero al nodo cabeza de la lista
 - Precondición
 - *isEmpty() == falso*
- *getCurrent()*: puntero a **NodoDoblementeEnlazadoMunicipio**
 - Obtiene el puntero al nodo actual de la lista
 - Precondición
 - *isEmpty() == falso*

▪ **Modificadores privados**

- *setHead(head: puntero a NodoDoblementeEnlazadoMunicipio)*
 - Modifica la cabeza de la lista
 - Postcondición
 - *getHead() == head*
- *setCurrent(current: puntero a NodoDoblementeEnlazadoMunicipio)*
 - Modifica el cursor de la lista
 - Postcondición
 - *getCurrent() == current*

▪ **Constructor**

- Constructor sin argumentos
 - **ListaDoblementeEnlazadaOrdenadaMunicipios()**
 - Postcondición
 - *isEmpty() == verdadero*

▪ **Destructor**

- Libera la memoria
 - *~ListaDoblementeEnlazadaOrdenadaMunicipios ()*
 - Postcondición
 - *isEmpty() == verdadero*

▪ **Observadores públicos que se deben redefinir**

- *bool isEmpty()*
 - Comprueba si la lista está vacía
 - Devuelve verdadero, si la lista está vacía; falso, en caso contrario.

- *Entero nItems()*
 - Devuelve el número de municipios
- *Lógico isFirstItem()*
 - Comprueba si el cursor está en el primer nodo
 - Precondición
 - *isEmpty() == falso*
- *Lógico isLastItem()*
 - Comprueba si el cursor está en el último nodo
 - Precondición
 - *isEmpty() == falso*
- *Municipio getCurrentItem()*
 - Obtiene el municipio situado en el nodo actual
 - Precondición
 - *isEmpty() == falso*
- *Municipio getPreviousItem()*
 - Obtiene el municipio situado en el nodo anterior
 - Precondición
 - *isEmpty() == falso*
 - *isFirstItem() == falso*
- *Municipio getNextItem()*
 - Obtiene el municipio situado en el nodo siguiente
 - Precondición
 - *isEmpty() == falso*
 - *isLastItem() == falso*
- **Modificadores públicos que se deben redefinir**
 - *gotoHead()*
 - Coloca el cursor en el primer nodo
 - Precondición
 - *isEmpty() == falso*
 - Postcondición
 - *isFirstItem() == verdadero*
 - *gotoLast()*
 - Coloca el cursor en el último nodo
 - Precondición
 - *isEmpty() == falso*
 - Postcondición
 - *isLastItem() == verdadero*
 - *gotoPrevious()*
 - Coloca el cursor en el nodo anterior
 - Precondición
 - *isEmpty() == falso*
 - *isFirstItem() == falso*
 - *gotoNext()*
 - Coloca el cursor en el nodo siguiente
 - Precondición

- *isEmpty() == falso*
 - *isLastItem() == falso*
- *Lógico find(ítem:Municipio)*
 - Busca el municipio *ítem*.
 - Si existe, coloca el cursor en el nodo que contiene el *ítem*.
 - Si no existe, se coloca en el elemento que es mayor que el ítem buscado, o en el último elemento, si todos los elementos de la lista son menores que el elemento buscado
 - *Postcondición*
 - Si *valorDevuelto == verdadero* entonces *getCurrentItem() == ítem*
 - Si *valorDevuelto == falso* entonces *getCurrentItem() > ítem* o *isLastItem() == verdadero*
- *insert(ítem:Municipio)*
 - Inserta un Municipio de forma ordenada por el nombre
 - *Precondición*
 - *find(ítem) == falso*
 - *Postcondición*
 - *getCurrentItem() == ítem*
 - *nItems() = old.nItems() + 1*
- *void remove()*
 - Borra el Municipio actual apuntado por el cursor
 - *Precondición*
 - *isEmpty() == falso*
 - *Postcondición*
 - *nItems() = old.nItems() - 1*
 - Si *old.isFirstItem() == falso* y *old.isLastItem() == falso* entonces *old.getPreviousItem() == getPreviousItem()* y *old.getNextItem() == getCurrentItem()*
 - Si *old.isLastItem() == verdadero* entonces *isEmpty() == verdadero* o *old.getPreviousItem() == getCurrentItem()* y *isLastItem() == verdadero*
 - Si *old.isFirstItem() == verdadero* entonces *isEmpty() == verdadero* o *old.getNextItem() == getCurrentItem()* y *isFirstItem() == verdadero*
- **Nuevo modificador público**
 - *removeAll()*
 - Borra todos los nodos de la lista
 - *Postcondición*
 - *isEmpty() == verdadero*
- Clase **NodoMunicipioInterfaz**
 - **Descripción**
 - Clase abstracta que especifica los métodos “**virtuales puros**” de un nodo doblemente enlazado
 - Los métodos virtuales puros deben ser redefinidos en las clases herederas.
 - **Destructor (método virtual)**

- *~NodoMunicipioInterfaz()*
 - Libera la memoria del nodo
 - **Observador público virtual puro**
 - *getItem(): Municipio*
 - Devuelve el campo informativo (municipio) del nodo
 - **Modificador público virtual puro**
 - *setItem(ítem: Municipio)*
 - Modifica el campo informativo (municipio) del nodo
 - Postcondición
 - *getItem() == ítem*
- Clase **NodoDoblementeEnlazadoMunicipio**
- **Descripción**
 - Clase que hereda de forma pública de la clase abstracta **NodoMunicipioInterfaz**
 - **Atributos**
 - *_ítem:*
 - campo informativo de tipo **Municipio**
 - *_previous:*
 - Enlace al nodo anterior
 - Puntero de tipo **NodoDoblementeEnlazadoMunicipio**
 - *_next:*
 - Enlace al nodo siguiente
 - **Puntero de tipo NodoDoblementeEnlazadoMunicipio**
 - **Constructor**
 - Constructor parametrizado
 - *NodoDoblementeEnlazadoMunicipio(ítem: Municipio
previous: puntero de tipo NodoDoblementeEnlazadoMunicipio,
next: puntero de tipo NodoDoblementeEnlazadoMunicipio)*
 - Postcondición
 - *getItem() == ítem*
 - *getPrevious() == previous*
 - *getNext() == next*
 - **Destructor**
 - *~NodoDoblementeEnlazadoMunicipio()*
 - Libera la memoria del nodo
 - **Observador público que se debe redefinir**
 - *getItem(): Municipio*
 - Obtiene el campo informativo (municipio) del nodo
 - **Nuevos observadores públicos**
 - *getPrevious(): puntero a NodoDoblementeEnlazadoMunicipio*
 - Obtiene el puntero al nodo anterior
 - *getNext(): puntero a NodoDoblementeEnlazadoMunicipio*
 - Obtiene el puntero al nodo siguiente
 - **Modificador público que se debe redefinir**
 - *setItem(ítem: Municipio)*
 - Modifica el campo informativo (municipio) del nodo

- Postcondición
 - *getItem() == ítem*
- **Nuevos modificadores públicos**
 - *setPrevious(previous: puntero a NodoDoblementeEnlazadoMunicipio)*
 - *Modifica el puntero al nodo anterior*
 - Postcondición
 - *getPrevious() == item*
 - *setNext(next: puntero a NodoDoblementeEnlazadoMunicipio)*
 - *Modifica el puntero al nodo siguiente*
 - Postcondición
 - *getNext() == item*
- Clase **Municipio**
 - **Atributos**
 - Nombre
 - Código postal
 - Número de Hombres
 - Número de Mujeres
 - **Constructores**
 - Constructor parametrizado con valores por defecto
 - *Municipio(nombre: Cadena = “”, codigoPostal: Entero = 0, hombres: Entero = 0, mujeres: Entero = 0)*
 - Postcondición
 - *getNombre() == nombre*
 - *getCodigoPostal() == codigoPostal*
 - *getHombres() == hombres*
 - *getMujeres() == mujeres*
 - Constructor de copia
 - *Municipio(Municipio const &objeto)*
 - Postcondición
 - *getNombre() == objeto.getNombre()*
 - *getCodigoPostal() == objeto.getCodigoPostal()*
 - *getHombres() == objeto.getHombres()*
 - *getMujeres() == objeto.getMujeres()*
 - **Observadores públicos**
 - *getNombre(): Cadena*
 - *Obtiene el nombre del municipio*
 - *getCódigoPostal(): Entero*
 - *Obtiene el código postal del municipio*
 - *getHombres(): Entero*
 - *Obtiene el número de hombres del municipio*
 - *getMujeres(): Entero*
 - *Obtiene el número de mujeres del municipio*
 - *getHabitantes(): Entero*
 - *Obtiene el número de habitantes del municipio*
 - Postcondición

- $valorDevuelto == getHombres() + getMujeres()$
- **Modificadores públicos**
 - *setNombre(v: Cadena)*
 - Modifica el nombre del municipio
 - Postcondición
 - $getNombre() == v$
 - *setCodigoPostal(v: Entero)*
 - Obtiene el código postal del municipio
 - Postcondición
 - $getCodigoPostal() == v$
 - *setHombres(v: Entero)*
 - Obtiene el número de hombres del municipio
 - Postcondición
 - $getHombres() == v$
 - *setMujeres(v: Entero)*
 - Obtiene el número de mujeres del municipio
 - Postcondición
 - $getMujeres() == v$
- **Operadores**
 - **Operador de igualdad**
 - *Lógico operador == (municipio: Municipio)*
 - Compara los nombres de dos municipios
 - Postcondición
 - $valorDevuelto == (getNombre() == municipio.getNombre())$
 - **Operador de asignación**
 - *Municipio operador = (municipio: Municipio)*
 - Devuelve el municipio actual modificado con los atributos de otro municipio
 - Postcondición
 - $getNombre() == municipio.getNombre()$
 - $getCodigoPostal() == municipio.getCodigoPostal()$
 - $getHombres() == municipio.getHombres()$
 - $getMujeres() == municipio.getMujeres()$
 - **Operador de comparación “menor que”**
 - *Municipio operador < (municipio: Municipio)*
 - Compara si el municipio actual es lexicográficamente anterior a otro municipio.
 - Postcondición
 - $valorDevuelto == (getNombre() < municipio.getNombre())$
- **Funciones de lectura y escritura**
 - *leerMunicipio()*
 - Lee desde el teclado los atributos de un municipio.
 - *escribirMunicipio()*
 - Escribe por pantalla los atributos de un municipio.

- **Funciones externas a la clase Municipio**
 - **Sobrecarga del operador de entrada**
 - Lee desde el flujo de entrada los atributos del municipio
 - Prototipo de C++
 - `istream &operator>>(istream &stream, Municipio &m);`
 - **Sobrecarga del operador de salida**
 - Escribe en el flujo de salida los atribtos del municipio:
 - Prototipo de C++
 - `ostream &operator<<(ostream &stream, Municipio const &m);`