Práctica 2: Palabras y Sinónimos

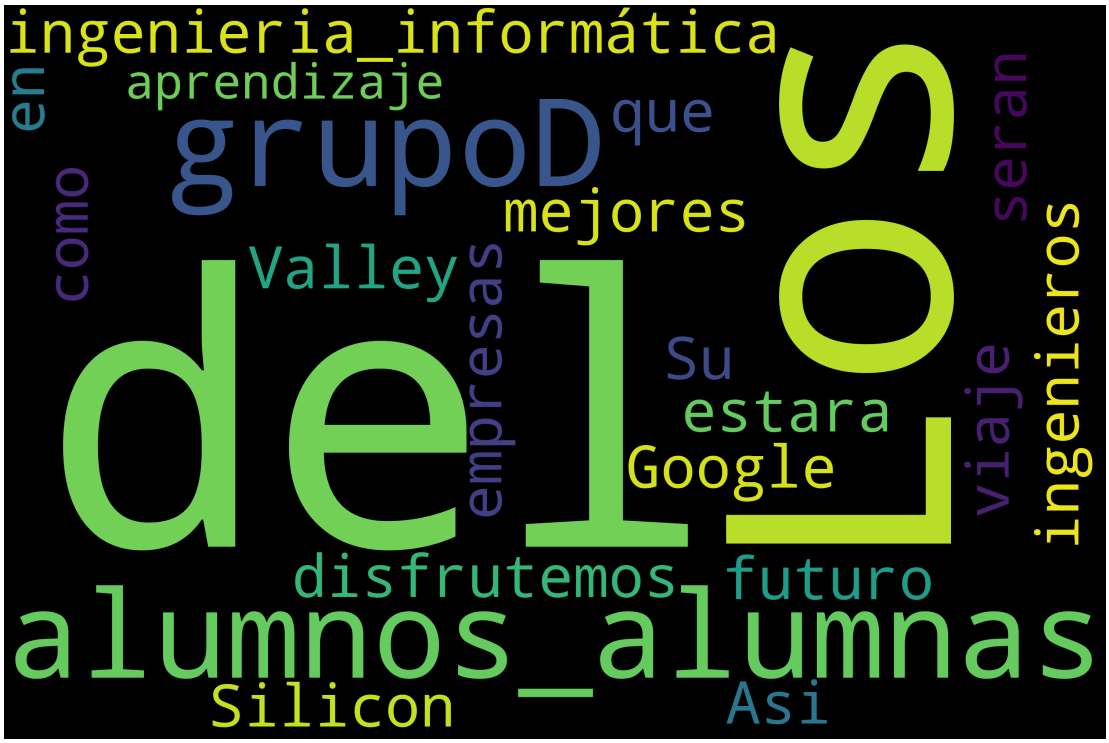
Dpto. Ciencias de la Computación e Inteligencia Artificial
E.T.S. de Ingenierías Informática y de Telecomunicación
Universidad de Granada

Estructuras de Datos

Grado en Ingeniería Informática. Grupo D

Índice de contenido

1. Introducción....3
2. Tipos de datos abstractos....3
   2.1. Selección de operaciones....3
3. Documentación....4
   3.1. Especificación del T.D.A....4
      3.1.1. Definición....4
      3.1.2. Operaciones....4
   3.2. Implementación del T.D.A....5
4. Ejercicio....7
   4.1. Programa test....7
   4.2. Programa tipo_texto....9
   4.3. Fichero con los Palabras....10
   4.4. Módulos a desarrollar....11
5. Práctica a entregar....11



1. Introducción

Los objetivos de este guión de prácticas son los siguientes:

1. Asimilar los conceptos fundamentales de abstracción, aplicado al desarrollo de programas.
2. Documentar un tipo de dato abstracto (T.D.A)
3. Practicar con el uso de doxygen.
4. Profundizar en los conceptos relacionados especificación del T.D.A, representación del T.D.A., función de Abstracción e Invariante de la representación.
5. Entender como implementar funciones y clases plantilla.

Los requisitos para poder realizar esta práctica son:

1. Haber estudiado el Tema 1: Introducción a la eficiencia de los algoritmos
2. Haber estudiado el Tema 2: Abstracción de datos. Plantillas
3. Conocer el contenedor vector dinámico.

2. Tipos de datos abstractos.

Los tipos de datos abstractos son nuevos tipos de datos con un grupo de operaciones que proporcionan la única manera de manejarlos. De esta forma, debemos conocer las operaciones que se pueden usar, pero no necesitamos saber

- la forma en como se almacena los datos ni
- cómo se implementan las operaciones.

2.1. Selección de operaciones

Una tarea fundamental en el desarrollo de un T.D.A. es la selección del conjunto de operaciones que se usarán para manejar el nuevo tipo de dato. Para ello, el diseñador deberá considerar los problemas que quiere resolver en base a este tipo, y ofrecer el conjunto de operaciones que considere más adecuado. Las operaciones seleccionadas deben atender a las siguiente exigencias:

1. Debe existir un conjunto mínimo de operaciones para garantizar la abstracción. Este conjunto mínimo permite resolver cualquier problema en el que se necesite el T.D.A.
2. Las operaciones deben ser usadas con bastante frecuencia.
3. Debemos considerar que el tipo de dato sufra en el futuro modificaciones y conlleve también la modificación de las operaciones. Por lo tanto un número muy alto de operaciones puede conllevar un gran esfuerzo en la modificación.

Por otro lado las operaciones seleccionadas pueden clasificarse en dos conjuntos:

1. Fundamentales. Son aquellas necesarias para garantizar la abstracción. No es posible prescindir de ellas ya que habría problemas que no se podrían resolver sin acceder a la parte interna del tipo de dato. A estas funciones también se le denominan operaciones **primitivas.**
2. No fundamentales. Corresponden a las operaciones prescindibles ya que el usuario podría construirlas en base al resto de operaciones.

3. Documentación

El objetivo fundamental de un programador es que los T.D.A. que programe sean reutilizados en el futuro por él u otros programadores. Para que esta tarea pueda llevarse a cabo los módulos donde se materializa un T.D.A. deben de estar bien documentados. Para llevar a cabo una buena documentación de T.D.A. se deben crear dos documentos bien diferenciados:

1. **Especificación**. Es el documento donde se presentan las características sintácticas y semánticas que describen la parte pública (la parte del T.D.A. visible a otros módulos). Con este documento cualquier otro módulo podría usar el módulo desarrollado y además es totalmente independiente de los detalle internos de construcción del mismo.
2. **Implementación.** Corresponden al documento que presenta las características internas del módulo. Para facilitar futuras mejoras o modificaciones de los detalles internos del módulo, es necesario que la parte de implementación sea documentada.

3.1. Especificación del T.D.A

En este caso vamos a especificar un tipo de dato junto con el conjunto de operaciones. Por lo tanto en la especificación de T.D.A. aparecerán dos partes:

1. **Definición.** En esta parte deberemos definir el nuevo tipo de dato abstracto, así como todos los términos relacionados que sean necesarios para comprender el resto de la especificación.
2. **Operaciones**. En esta parte se especifican las operaciones, tanto sintáctica como semánticamente.

3.1.1. Definición

Se dará una definición del T.D.A en lenguaje natural. Para ello el T.D.A se distinguirá como una nueva clase de objetos en la que cualquier instancia de esta nueva clase tomará valores (dominio) en un conjunto establecido. Por ejemplo si queremos definir un *Racional* diremos:

> ***Una instancia f del tipo de dato abstracto Racional es un objeto del conjunto de los números racionales, compuesto por dos valores enteros que representan, respectivamente, numerador y denominador. Lo representamos num/den.***

3.1.2. Operaciones

En esta parte se realiza una especificación de las operaciones que se usarán sobre el tipo de dato abstracto que se está construyendo. Cada una de estas operaciones representarán una función y por lo tanto se hará la especificación con los siguientes items:

1. Breve descripción. Que es lo que hace la función. Usando en doxygen la sentencia @brief.
2. Se especifican cada uno de los parámetros de la función. Por cada parámetro se especificará si el parámetros se modifica o no tras la ejecución de la función. Usando en doxygen el comando @param.
3. Las condiciones previas a la ejecución de la función (precondiciones) que deben cumplirse para un buen funcionamiento de la función. Usando en doxygen la sentencia @pre.
4. Que devuelve la función. En doxygen usaremos el comando @return
5. Las condiciones que deben cumplirse trás la ejecución de la función (postcondiciones). Usando en doxygen el comando @post.

Supongamos que queremos especificar en nuestra clase *Racional* la función *Comparar* que

compara un Racional con el Racional que apunta this. Una posible especificación de esta función sería.

```
/**
 * @brief Compara dos racionales
 * @param r racional a comparar
 * @return Devuelve 0 si este objeto es igual a r,
 *              <0 si este objeto es menor que r,
 *              >0 si este objeto es mayor que r
 */
bool comparar(Racional r);
```

Un ejemplo donde se usa una precondición es en la función *asignar* que se le asigna al Racional apuntado por this unos valores concretos para el numerador y denominador. En esta especificación cabe resaltar que si el nuevo denominador es 0 se estará violando las propiedades que deben mantenerse para una correcta instanciación de un objeto de tipo Racional.

```
/**
 * @brief Asignación de un racional
 * @param n numerador del racional a asignar
 * @param d denominador del racional a asignar
 * @return Asigna al objeto implícito el numero racional n/d
 * @pre d debe ser distinto de cero
 */
void asignar(int n, int d);
```

3.2. Implementación del T.D.A.

Para implementar un T.D.A., es necesario en primer lugar, escoger una representación interna adecuada, una forma de estructurar la información de manera que podamos representar todos los objetos de nuestro tipo de dato abstracto de una manera eficaz. Por lo tanto, debemos seleccionar una estructura de datos adecuada para la implementación, es decir, un tipo de dato que corresponda a esta representación interna y sobre el que implementamos las operaciones. A éste tipo escogido (la estructura de datos seleccionada), se le denomina ***tipo rep.***

Para nuestro T.D.A *Racional* las estructuras de datos posibles para representar nuestro ***tipo rep*** podrían ser por ejemplo:

- Un vector de dos posiciones para almacenar el numerado y denominador

```
class Racional{
private:
   int r[2];
....
}
```

- Dos enteros que representen el numerador y denominador respectivamente

```
class Racional{
private:
int numerador;
int denominador;
….
}
```

De entre las representaciones posibles en el documento debe aparecer la estructura de datos escogida para el ***tipo rep***. Esta elección debe formalizarse mediante la especificación de la función de abstracción. Esta función relaciona los objetos que se pueden representar con el ***tipo rep*** y los objetos del tipo de dato abstracto. Las propiedades de esta función son:

- Parcial, todos los valores de los objetos del ***tipo rep*** no se corresponden con un objeto del tipo abstracto. Por ejemplo valores de numerador=1 y denominador =0 no son valores válidos para un objeto del tipo Racional.
- Todos los elementos del tipo abstracto tienen que tener una representación.
- Varios valores de la representación podrían representar a un mismo valor abstracto. Por ejemplo {4,8} y {1,2} representan al mismo racional.

Por lo tanto en la documentación podemos incluir esta aplicación para indicar el significado de la representación. Esta aplicación tiene dos partes:

1. Indicar exactamente cual es el conjunto de valores de representación que son válidos, es decir, que representen a un tipo abstracto. Por tanto, será necesario establecer una condición sobre el conjunto de valores del tipo *rep* que nos indique si corresponden a un objeto válido. Esta condición se denomina invariante de la representación.

$$f_{inv}: rep \dashrightarrow booleanos$$

2. Indicar para cada representación válida cómo se obtiene el tipo abstracto correspondiente, es decir la función de abstracción.

$$f_{abs}: rep \dashrightarrow A$$

Un invariante de la representación es "invariante" porque siempre es cierto para la representación de cualquier objeto abstracto. Por tanto, cuando se llama a una función del tipo de dato se garantiza que la representación cumple dicha condición y cuando se devuelve el control de la llamada debemos asegurarnos que se sigue cumpliendo. En nuestro ejemplo el T.D.A Racional en la documentación de la implementación incluiríamos lo siguiente:

```
class Racional {
 private:
/**
  * @page repRacional Rep del TDA Racional
  * @section invRacional Invariante de la representación
  * El invariante es \e rep.den!=0
  *  * @section faRacional Función de abstracción
  * Un objeto válido @e rep del TDA Racional representa al valor
  * (rep.num,rep.den)
  */
  int num; /**< numerador */
  int den; /**< denominador */
 public:
””””
```

4. Ejercicio

El objetivo en este ejercicio es mantener la información de un conjunto de palabras, y sobre este conjunto realizar una serie de operaciones.

Con tal fin vamos a desarrollar varios tipos de datos abstractos:

- ***Palabra*** : Representa una palabra de un diccionario. Los atributos de una palabra son el valor de la palabra (puede estar formado por varias palabras). El tipo de la palabra y una secuencia de palabras sinónimas. Un ejemplo podría ser:

  Tipo: *adverb*

  Valor de Palabra*: apresuradamente*

  Sinónimos*: a prisa, al trote, con prontitud, deprisa, ejecutivamente,…*

  Siendo el tipo "adverb" (adverbio). Los sinónimos deberá almacenarse de forma ordenada.

- ***Palabras***: Se define como una colección ordenada de palabras pertenecientes a un idioma. La ordenación por defecto será por valor de palabra y a igualdad de valor de palabra ordenado por tipo. No obstante podremos obtener un objeto de tipo Palabras ordenada por tipo y a igualdad de tipo por valor de palabra.

Se pide desarrollar los tipos de datos abstractos (TDA): ***Palabra y Palabras.*** Para cada uno de estos tipo de datos abstractos:

1. Dar la especificación. Establecer una definición de los T.D.A.
2. Para la estructura de datos propuesta del ***tipo rep*** establecer cual es el invariante de la representación y función de abstracción.
3. Fijado el ***tipo rep*** realizar la implementación de las operaciones.
4. Haciendo uso de test.cpp probar los tipos de datos abstractos desarrollados. Este fichero viene en el material dado al alumno/a. Es importante que este fichero **no se modifique**. Por lo tanto debemos estudiar que funciones o métodos son necesarios para nuestros tipos de datos para que este fichero pueda compilarse.

Como guía para llevar a cabo estos puntos se puede observar el T.D.A Racional dado en el material. En el material (ficheros palabra.h y palabras.h) se da las operaciones mínimas que debería tener los TDA ***Palabra y Palabras***.

Con respecto al desarrollo de T.D.A ***Palabra y Palabras,*** el estudiante usará para su representación el T.D.A ***vector dinámico.*** El T.D.A vector dinámico se da implementado en el material como una clase plantilla.

4.1. Programa test

El estudiante creará el programa ***test*** que se ejecuta desde la línea de órdenes de la siguiente forma:

```
prompt>bin/test datos/dic_spanish.txt
```

Este programa recibe un fichero con el conjunto de palabras.

Con estos datos se prueba los diferentes TDA desarrollados. Si visualizamos el fichero test.cpp el estudiante vera seis partes diferenciadas:

- ***Sección 1***: En esta sección se comprueba la declaración, lectura, consulta y escritura del T.D.A ***Palabra.***
- ***Sección 2:*** En esta sección se comprueba la declaración, lectura y escritura del T.D.A

Palabras. La lectura hará uso de los métodos de inserción. Hay que tener en cuenta que los ficheros de entrada no están ordenados. Y por lo tanto cada vez que se inserta una palabra se debe insertar de forma ordenada por valor de palabra y a igualdad por tipo. Además se comprobara otras operaciones como numero de elementos y borrado.

- ***Sección 3:*** Generamos otro conjunto de palabras, con las mismas palabras, pero ahora ordenado por tipo y a igualdad de tipo por nombre.
- ***Sección 4:*** En esta sección se obtiene varias informaciones sobre el conjunto de palabras. En primer lugar se obtiene el conjunto de tipos diferentes en el conjunto de palabras. Este conjunto de tipos lo obtenemos en un vector dinámico de forma ordenada. En segundo lugar se obtiene las palabras de un determinado tipo, para ello obtenemos un nuevo objeto Palabras. Y finalmente obtenemos los tipos de una determinada palabra. Una palabra puede tener diferentes tipos por ejemplo la palabra ***"sol"*** tiene como tipos "relations" (p.e "Mi hermano es un sol") o puede tener el tipo "geography" en el sentido del astro Sol.
- ***Sección 5:*** En esta sección se comprueba si dos palabras dadas por el usuario y con un tipo concreto son sinónimas.

El modo de trabajo debería ser el siguiente:

1. En primer lugar desarrollar el T.D.A Palabra. En el directorio ***include*** del material se ha dado el fichero *palabra.h* donde se puede ver la representación del TDA junto con las cabeceras de las operaciones propuestas para el T.D.A Palabra. El alumno/a deberá aquí documentar el T.D.A. Para ello dará una especificación del T.D.A junto con la función de abstracción e invariante de representación. Ahora cada una de las operaciones deberá documentarse. Para generar la documentación se usará ***doxygen***. En el directorio ***src*** crearemos el el fichero *palabra.cpp* que contendrá la implementación de las operaciones indicadas en *palabra.h*. Compilar el fichero palabra.cpp para obtener palabra.o. Para ello podemos ejecutar ***make obj/palabra.o.*** Si la compilación no genera ningún fallo pasamos al paso 2.
2. En el fichero test.cpp comentar todo para dejar solamente la sección 1 que testea el T.D.A Palabra. Para ello comentar el resto de secciones y comentar ***#include "palabras.h".*** Ahora ejecutar ***make bin/test1.*** Si compila bien ahora probamos el funcionamiento ejecutando ***bin/test1 datos/dic_spanish.txt.*** Si todo funciona bien pasamos al paso 3.
3. En este paso desarrollaremos el T.D.A Palabras. En el directorio ***include*** del material se ha dado el fichero *palabras.h* donde se puede ver la representación del TDA junto con las cabeceras de las operaciones propuestas para el T.D.A Palabras. El alumno deberá aquí documentar el T.D.A. Para ello dará una especificación del T.D.A junto con la función de abstracción e invariante de representación. Ahora cada una de las operaciones deberá documentarse. Para generar la documentación se usará ***doxygen***. En el directorio ***src*** crearemos el fichero *palabras.cpp* que contendrá la implementación de las operaciones indicadas en *palabra.h*. Compilar el fichero palabras.cpp para obtener palabras.o. Para ello podemos ejecutar ***make obj/palabras.o.*** Si la compilación no genera ningún fallo pasamos al paso 4.
4. En el fichero test.cpp descomentamos ***#include "palabras.h".*** A continuación vamos descomentando las siguientes secciones. Se propone descomentar una sección, compilar y ejecutar. Si todo va bien pasamos a la siguiente sección. Así hasta completar todas las secciones.
5. Para generar la documentación es necesario que los fichero ***.h*** contengan la documentación doxygen. Una vez completada se podrá ejecutar ***make documentacion.*** Si todo va bien en el directorio doc debe haber creado un directorio html. Visualizar en un navegador el fichero ***index.html*** y comprobar el resultado obtenido.

4.2. Programa tipo_texto

El alumno creará el programa ***tipo_texto*** que se ejecuta desde la línea de órdenes de la siguiente forma:

```
prompt>bin\tipo_texto.exe datos\texto5.txt datos\dic_spanish.txt
```

Este programa recibe un fichero con un texto y un *conjunto de palabras*. Este programa en primer lugar muestra todos los tipos de palabras en nuestro conjunto de palabras. En el ejemplo la salida debería ser:

```
Los tipos de alimentos son:
abstract
adjective
adjective-basic
adjective-people
adverb
anatomy
animal
art
article
business
city
clothes
color
communication
conjunction
container
conversation
country
device
drink
education
environment
family
feeling
food
fruit
furniture
geography
house
kitchen
language
material
mathematics
medicine
nature
number
object
physics
preposition
profession
pronoun
relations
science
society
sports
time
tool
transport
vegetable
verb
verb-basic
weather
```

En el ejemplo el texto de entrada es el siguiente:

```
Ingredientes:
400 gr de harina.
1 vaso de vino blanco.
1 pizca de sal.
Abundante aceite.
Azucar.

Relleno:
4 cucharadas de chocolate en polvo.
500 ml de leche.
4 cucharadas de azucar.
2 cucharadas de fecula de maiz.

Preparacion:
Ponga la harina en un recipiente, dele forma volcan. Vierta en el centro el vino
mezclado con un vaso de aceite y sal y trabaje hasta obtener una masa homogenea.
Prepare el relleno: caliente la leche (reservando un vasito) con el azucar y el
chocolate y, cuando rompa a hervir, incorpore la fecula disuelta en la leche
reservada y remueva hasta que espese; paselo a un plato y deje que se enfrie.
Extienda la masa y cortela en tiras; distribuya el relleno sobre la masa, forme las
empanadillas y frialas en abundante aceite bien caliente; rebocelas en azucar y
sirva.
```

Sobre este texto se obtiene la frecuencia con la que aparece cada tipo (solamente muestra los tipos con una frecuencia mayor que cero):

```
Tipo abstract aparece 2                Tipo material aparece 1
Tipo adverb aparece 8                  Tipo mathematics aparece 1
Tipo art aparece 1                     Tipo number aparece 2
Tipo article aparece 10                Tipo physics aparece 3
Tipo color aparece 1                   Tipo preposition aparece 21
Tipo conjunction aparece 4             Tipo pronoun aparece 10
Tipo container aparece 4               Tipo science aparece 1
Tipo drink aparece 5                   Tipo verb-basic aparece 1
Tipo food aparece 10
```

4.3. Fichero con los Palabras

Para poder probar nuestros programas usaremos un fichero compuesto de una serie de líneas. Cada línea se corresponde con la información de una palabra

```
Español
#TIPO;PALABRA;SINONIMOS(separadas por ;)
country;China;
country;Estados Unidos;
country;Italia;
country;Japon;
country;Reino Unido;
country;Reino de España;
country;Republica Checa;
country;Republica Federal de Alemania;
country;Republica Francesa;
country;Republica de la India;
preposition;a;
adverb;a lo largo de;junto con;
adverb;junto con;a lo largo de;
adverb;a traves de;dentro de;por;
adverb;dentro de;a traves de;por;
adverb;por;a traves de;dentro de;
adverb;a veces;
adverb;abajo;
adjective;abajo;debajo;
adjective;debajo;abajo;
verb;abandonar;dejar;
verb;dejar;abandonar;
animal;abeja;
adjective-basic;abierto;
verb-basic;abonar;pagar;
verb-basic;pagar;abonar;
time;abril;
verb;abrillantar;
verb-basic;abrir;
adjective;absoluto;completo;perfecto;total;
….
```

El fichero contiene:

1. La primera línea nos indica el idioma del conjunto de palabras
2. La siguiente linea es un comentario, que se inicia con el carácter '#'.
3. A continuación en cada línea viene la información de cada palabra. Cada atributo de un palabra se encuentra separada por ";". Los atributos son
    - Tipo de palabra
    - Valor de la palabra (puede ser una palabra simple o compuesta)
    - A continuación puede aparecer (no siempre) una lista de sinónimos separados por ';'

En el material que se aporta en el directorio datos ejemplos de estos ficheros son

dic_spanish.txt, dic_english.txt or dic_french.txt

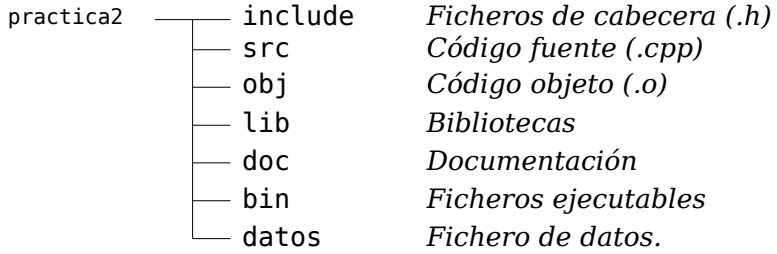
4.4. Módulos a desarrollar.

EL alumno tendrá que desarrollar los siguientes módulos como mínimo: 1) ***Palabra*** (palabra.cpp y palabra.h); 2) ***Palabras*** (palabras.h y palabras.cpp). Estos módulos tienen que tener la documentación suficiente para que cualquier usuario pueda usarlo sin problemas. Para poder documentar los módulos usaremos doxygen. Además creará el programa **tipo_textos** (que implementará en el fichero tipo_texto.cpp). En el material que se aporta tiene ya el modulo vector dinámico (VD.h y VD.cpp). Tiene la representación y operaciones de Palabra y Palabras (en los fichero palabra.h y palabras.h respectivamente).

5. Práctica a entregar

El alumno deberá empaquetar todos los archivos relacionados en el proyecto en un archivo con nombre "practica2.*tgz*" y entregarlo antes de la fecha que se publicará en la página web de la asignatura (en PRADO). Tenga en cuenta que no se incluirán ficheros objeto, ni ejecutables, ni la carpeta datos. Es recomendable que haga una "limpieza" para eliminar los archivos temporales o que se puedan generar a partir de los fuentes.

El alumno debe incluir el archivo *Makefile* para realizar la compilación. Tenga en cuenta que los archivos deben estar distribuidos en directorios*:*



Para realizar la entrega, en primer lugar, realice la limpieza de archivos que no se incluirán en ella, y sitúese en la carpeta superior (en el mismo nivel de la carpeta "*practica2*") para ejecutar:

```
prompt% tar zcv practica2.tgz practica2
```

tras lo cual, dispondrá de un nuevo archivo practica2.tgz que contiene la carpeta practica2 así como todas las carpetas y archivos que cuelgan de ella.