Práctica 2: Ingredientes Alimenticios

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.
Doble Grado en Informática y ADE
Doble Grado en Informática y Matemáticas

Índice de contenido

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 ingredientes alimenticios, y sobre este conjunto realizar una serie de operaciones.

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

- ***Ingrediente*** : Representa un ingrediente alimenticio. Los atributos de un ingrediente es el nombre (puede estar formado por varias palabras) y número de calorías que aporta cada 100 gramos. También se especifica para el alimento los porcentajes de los nutrientes esenciales (hidratos de carbono, proteínas, grasas y fibra) presentes en el alimento. Finalmente se indica el tipo de alimento, por ejemplo: verdura, carne, pescado, etc.
- ***Ingredientes***: Se define como una colección ordenada de ingredientes alimenticios. Dos ordenaciones posibles se definen para este conjunto:
  1. Ordenación por el nombre del alimento
  2. Ordenación por el tipo de alimento y a igualdad se ordena por el nombre.

Se pide desarrollar los tipos de datos abstractos (TDA): ***Ingrediente e Ingredientes.*** Para cada uno de estos tipo de datos abstractos:

1. Dar la especificación. Establecer una definición y el conjunto de operaciones básicas.
2. Determinar la estructura de datos ***tipo rep.***
3. Para la estructura de datos del ***tipo rep*** establecer cual es el invariante de la representación y función de abstracción.
4. Fijado el ***tipo rep*** realizar la implementación de las operaciones.
5. Haciendo uso de test_ingredientes.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 la sección 4.4 se da las operaciones mínimas que debería tener el TDA Ingredientes.

Con respecto al desarrollo de T.D.A ***Ingredientes,*** el alumno usará para su representación el T.D.A : ***vector dinámico.*** El vector dinámico deberá implementarse con una clase plantilla.

4.1. Programa test_ingredientes

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

```
prompt>bin/test_ingredientes datos/ingredientes.txt
```

Este programa recibe un fichero con los ingredientes.

Con estos datos testea los diferentes TDA desarrollados. Si visualizamos el fichero test_ingredientes.cpp el alumno vera cuatro partes diferenciadas:

- ***Sección 1***: En esta sección se comprueba la declaración, lectura, consulta y escritura del TDA ***ingrediente.*** El alumno debería también proporcionar operadores de modificación para cada uno de los atributos de un objeto ingrediente.
- ***Sección 2:*** En esta sección se comprueba la declaración, lectura y escritura del TDA ***ingredientes***. 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 un ingrediente se debe insertar de forma ordenada por nombre y simultáneamente mantener la ordenación por tipo y nombre.
- ***Sección 3:*** Sobre el TDA ingredientes debemos mantener también la ordenación por

tipo de alimento y nombre. En esta sección se comprueba que dicha ordenación se ha realizado correctamente.

- ***Sección 4:*** En esta sección se consulta de información de un ingrediente dando el nombre. Además se borra este alimento y se vuelve a listar los ingredientes. El alumno debe comprobar que tras el borrado la ordenación, tanto por nombre, como por tipo y nombre es correcta.
- ***Sección 5:*** En esta sección se obtiene los tipos diferentes de ingredientes que existen. Además se selecciona solamente los ingredientes de un tipo para a continuación mostrarlos.

4.2. Programa tipos_ingredientes

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

```
prompt>bin/tipos_ingredientes datos/ingredientes.txt Molusco
```

Este programa recibe un fichero con los ingredientes y un tipo de ingrediente en el ejemplo *Molusco*. Este programa en primer lugar muestra todos los tipos de alimentos en nuestro fichero. En el ejemplo la salida debería ser:

```
Los tipos de alimentos son:
Aceite
Azucar
Bebida
Carne
Cereal
Crustaceo
Especias-Condimento
Fruta
Frutos Secos
Huevo
Leche-Derivados
Legumbre
Molusco
Pescado
Procesado
Semillas
Verdura
```

A continuación muestra todos los ingredientes de tipo *Molusco,* en nuestro ejemplo son:

```
Los ingredientes del tipo Molusco son:
Alimento (100 gramos);Calorias (Kcal.);Hidratos de Carb.;Proteinas;Grasas;Fibra;Tipo
Almeja / Chirla;50;0;11;0.9;0;Molusco
Berberechos Cocidos;48;0;11;0;0;Molusco
Bigaros Cocidos;135;5;26;1;0;Molusco
Caracol Terrestre;80;0;16;1;0;Molusco
Mejillones;66;0;12;2;0;Molusco
Ostras;70;5;9;1;0;Molusco
Percebes;59;0;14;0;0;Molusco
Pulpo;57;2;11;1;0;Molusco
Sepia;79;0;16;1;0;Molusco
Vieiras;70;0;16;0;0;Molusco
```

Sobre este conjunto de ingredientes muestra la media, desviación, máximo y mínimo sobre los atributos calorías, hidratos de carbono, proteínas, grasas y fibra. En nuestro ejemplo la salida

debería ser:

```
Estadistica_____________________________
Tipo de Alimento Molusco
Promedio +-Desviacion
Calorias        Hidratos de Carb      Proteinas      Grasas  Fibra
71.4+-23.6144  1.2+-1.98997   14.2+-4.6      0.79+-0.597411 0+-0
Máximos Valores
Calorias (Alimento)  Hidratos de Carb (Alimento)  Proteinas (Alimento)  Grasas(Alimento)  Fibra (Alimento)
135 Bigaros Cocidos   5 Bigaros Cocidos         26 Bigaros Cocidos       2 Mejillones   0 Almeja / Chirla

Minimos Valores
Calorias (Alimento)   Hidratos de Carb (Alimento)  Proteinas (Alimento)   Grasas (Alimento) Fibra(Alimento)
48  Berberechos  0     Almeja / Chirla              9     Ostras         0      Berberechos 0  Almeja / Chirla
     Cocidos                                                                    Cocidos
```

Cuando se muestra los valores máximos y mínimos de los diferentes atributos se da también en que alimento se obtiene ese máximo o mínimo. Por ejemplo el alimento de tipo Molusco con menor calorías son los “Berberechos Cocidos”.

4.3. Fichero con los ingredientes

Para poder probar nuestro programa usaremos un fichero compuesto de una serie de líneas. Cada línea se corresponde con la información de un ingrediente

```
Alimento (100 gramos);Calorias (Kcal.);Hidratos de Carb.;Proteinas;Grasas;Fibra;Tipo
Ketchup;98;24;2;0;0;Procesado
Salmon;182;0;18;12;0;Pescado
Cereales Cornflakes;368;85;9;2;11;Cereal
Gallo;81;0;17;1;0;Carne
Queso Emmental;377;0;29;29;0;Leche-Derivados
Cerezas;47;12;0;2;0;Fruta
Gatorade;39;10;0;0;0;Bebida
Salsa Ketchup;98;24;2;0;0;Procesado
Esparragos Enlatados;14;1;2;0;0;Verdura
Endibias;11;1;2;0;0;Verdura
Harina Trigo, Panificada;337;75;11;1;3;Cereal
Nuez Brasil;617;4;12;62;9;Frutos Secos
Gofio Millo;377;83;6;5;0;Cereal
Pavo;107;0;22;2;0;Carne
Mantequilla;740;0;0;82;0;Aceite
Yogur Liquido;78;11;3;2;0;Leche-Derivados
Cacao Polvo;357;11;20;24;38;Semillas
Arenque Ahumado;205;0;26;11;0;Pescado
….
```

El fichero contiene:

1. En la primera línea es un comentario. Indicando que atributos tendrá cada uno de los alimentos.
2. A continuación en cada línea viene la información de cada ingrediente. Cada atributo de un ingrediente se encuentra separada por “;”. Los atributos son

- Nombre de ingrediente (puede tener más de una palabra)
- Calorías por cada 100 gramos del ingrediente
- Porcentaje de hidratos de carbono
- Porcentaje de proteínas
- Porcentaje de grasas
- Porcentaje de Fibra
- Tipo de ingrediente

4.4. Módulos a desarrollar.

EL alumno tendrá que desarrollar los siguientes módulos como mínimo: 1) El módulo asociado a ***VD (vector dinámico)*** (***VD***.cpp y ***VD***.h), este modulo será una clase plantilla como se ha visto en clase 2) ***Ingrediente*** (ingrediente.cpp y ingrediente.h); 3) ***Ingredientes*** (ingredientes.h y ingredientes.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 **tipos_ingredientes** (que implementará en el fichero tipos_ingredientes.cpp).

Será importante elegir una buena representación para el TDA Ingredientes para que la doble ordenación sea eficiente. Con doble ordenación queremos decir que los ingredientes se deberán mantener ordenadas:

1. Por nombre
2. Por tipo y a igualdad de tipo por nombre.

4.4.1. Módulo Ingredientes

Este módulo está dedicado a la especificación e implementación del TDA Ingredientes Este TDA es una colección de objetos ingrediente.

Las operaciones mas relevantes de este T.D.A son:

- Constructor por defecto
- Insertar y Borrar un ingrediente. Estas operaciones tienen que preservar el orden tras su ejecución. Recordar que el orden que tenemos en Ingredientes es doble 1) Solamente por nombre 2) Por tipo y nombre.
- Operadores de consulta:
  - Obtener el ingrediente en la posición *i* según el orden del nombre.
  - Obtener la información de un ingrediente dando su nombre
  - Obtener todas los ingredientes de un tipo
  - Sobrecargas el operator [] para realizar consultas
- Operadores de modificación: modificar el ingrediente de la posición **i** por otro ingrediente
- Además se deberá sobrecargar los operadores de escritura y lectura en flujos.

Para representar este módulo el ***tipo rep*** es un vector dinámico implementado como template. De forma que el modulo Ingrediente podría seguir el siguiente esquema inicial que habrá que perfilar.

```
template <class T>
class VD{
    T *datos;
    int n;
    int reservados;
…
}

//Ingredientes

class ingredientes{
    VD<ingrediente> datos;

…..

…
}
```

En función de que **tipo rep** estéis usando debéis implementar las anteriores operaciones comentadas. Puede incluirse otras si lo veis necesario.

4.5. Fichero para probar los TDA ingrediente e ingredientes

En el directorio del material que os damos tenéis el fichero test_ingredientes.cpp

FICHERO test_ingredientes.cpp

```
1.  #include <iostream>
2.  #include "ingrediente.h"
3.  #include "ingredientes.h"
4.  #include <fstream>
5.  using namespace std;
6.  void MuestraParametros(){
7.
8.      cout<<"1.- Dime el nombre del fichero con los ingredientes"<<endl;
9.  }
10.
11. int main(int argc, char *argv[])
12. {
13.  if (argc!=2){
14.      MuestraParametros();
15.      return 0;
16.  }
17. /*****************************************************************************************/
18. //SECTION 1: Test sobre la lectura de un ingrediente
19. //Ingrediente debe tener operadores de consulta y de modificacion por cada parametros
20. //ademas de sobrecarga de lectura y escritura
21. string nf =argv[1];
```

```
22.     ifstream f(nf);
23.     if (!f){
24.       cout<<"No puedo abrir "<<nf<<endl;
25.       return 0;
26.     }
27.     //Quitamos la primera linea
28.     string linea;
29.     getline(f,linea);
30.
31.     ingrediente i;
32.     f>>i;
33.     cout<<"Leido ingrediente:"<<endl;
34.     cout<<"Nombre " <<i.getNombre()<<endl;
35.     cout<<"Calorias "<<i.getCalorias()<<endl;
36.     cout<<"Hc "<<i.getHc()<<endl;
37.     cout<<"Proteinas "<<i.getProteinas()<<endl;
38.     cout<<"Grasas "<<i.getGrasas()<<endl;
39.     cout<<"Fibra "<<i.getFibra()<<endl;
40.     cout<<"Tipo "<<i.getTipo()<<endl;
41.     cout<<endl<<"Ahora probamos la sobrecarga de salida:\t"<<i<<endl;
42.
43.     cout<<"\n Pulsa una tecla para continuar...."<<endl;
44.     cin.get();
45.     /*****************************************************************************************/
46.     //SECTION 2: Test sobre el objeto Ingredientes. En primer lugar comprobamos que la
47.    // sobrecarga de entrada/ salida esta bien. Y por lo tanto la operación de insertar
48.     //Ponemos el puntero del ficheo al principio
49.     f.seekg(0);
50.     ingredientes all_ingre;
51.     cout<<"Lectura de todos los ingredientes"<<endl;
52.     f>>all_ingre;
53.
54.     //Comprobamos que hemos hecho bien la lectura
55.     cout<<"Los ingredientes ordenados por nombre..."<<endl<<endl;
56.     cout<<all_ingre<<endl;
57.     cout<<"Pulse una tecla para continuar..."<<endl<<endl;
58.     cin.get();
59.     /*****************************************************************************************/
60.     //Section 3: Sobre ingredientes comprobamos  que la indexacion por tipo funciona
61.      cout<<"Imprimos por tipo "<<endl;
62.      all_ingre.ImprimirPorTipo(cout);
63.      cout<<endl<<"Pulse una tecla para continuar..."<<endl<<endl;
64.      cin.get();
65.
66.     /*****************************************************************************************/
67.     //SECTION 4: Sobre ingredientes comprobamos consultar por nombre, size y borrar
```

```
68.        cout<<endl<<endl;
69.        cout<<"El numero de ingredientes son "<<all_ingre.size()<<endl;
70.        cout<<"Dime un nombre de ingrediente:";
71.        string n;
72.        getline(cin,n);
73.        ingrediente ing=all_ingre.get(n);
74.        if (ing.getNombre()!="Undefined"){
75.          cout<<"Información del ingrediente "<<ing<<endl;
76.          cout<<"Pulse una tecla para continuar...."<<endl;
77.          cin.get();
78.
79.          //borra por nombre del ingrediente
80.          all_ingre.borrar(n);
81.          cout<<"Tras el borrado de "<<ing.getNombre()<<" los ingredientes son:"<<endl<<all_ingre;
82.          cout<<"Numero de ingredientes tras el borrado son "<<all_ingre.size()<<endl;
83.          cout<<"Pulse una tecla para continuar ...."<<endl;
84.          cin.get();
85.          cout<<"Ahora los vemos ordenados por tipo"<<endl;
86.          all_ingre.ImprimirPorTipo(cout);
87.         }
88.         else{
89.                cout<<"El ingrediente "<<n<<" no aparece "<<endl;
90.
91.         }
92.        cout<<endl<<"Pulse una tecla para continuar..."<<endl<<endl;
93.        cin.get();
94.        /*********************************************************************************************/
95.
96.        //SECTION 5: Obtiene los tipos diferentes y los  ingredientes de un tipo concreto
97.         VD<string> tipos=all_ingre.getTipos();
98.         cout<<"Los tipos de alimentos son:"<<endl;
99.         for (int i=0;i<tipos.size(); ++i){
100.          cout<<tipos[i]<<endl;
101.        }
102.        cout<<"Pulse una tecla para continuar "<<endl;
103.        cin.get();
104.
105.        string tipo="Molusco";
106.        ingredientes ingre_tipo=all_ingre.getIngredienteTipo(tipo);
107.        cout<<"Los ingredientes de tipo "<<tipo<<" son: "<<endl<<ingre_tipo<<endl;
108.
109.    }
```

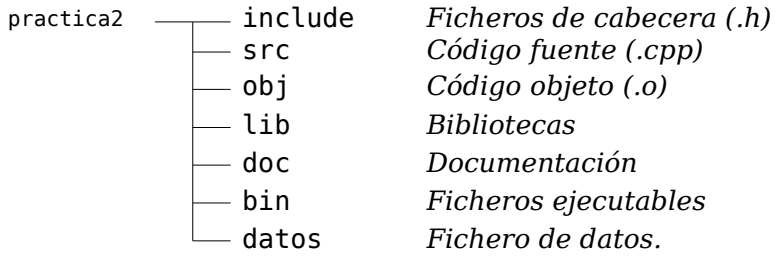
FIN de FICHERO test_ingredientes.cpp

Este código debe funcionar con los TDA desarrollados, y ademas el alumno/a no modificará el código.

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*:*



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.