Programación 2

Examen de teoría (junio 2022)

31 de mayo de 2022

Instrucciones

- **Duración: 3 horas**
- El fichero del primer problema debe llamarse `ej1.cc`. Para el segundo problema es necesario entregar seis ficheros, llamados `Producto.cc`, `Producto.h`, `Ingrediente.cc`, `Ingrediente.h`, `Receta.cc` y `Receta.h`. Si te falta algún fichero, crea uno vacío. En caso contrario no te dejara hacer la entrega. Pon tu DNI y tu nombre en un comentario al principio de todos los ficheros fuente (incluso de los vacíos)
- La entrega se realizará como en las prácticas, a través del servidor del DLSI (`http://pracdlsi.dlsi.ua.es`), en el enlace `Programación 2`. Puedes realizar varias entregas, aunque sólo se corregirá la última
- En la siguiente página web tienes disponibles las transparencias y libro de la asignatura, así como algunos ficheros de ayuda: `http://www.dlsi.ua.es/asignaturas/p2`
- Se valorará que el código siga la guía de estilo de la asignatura: nombres de variables/funciones, tabulación correcta, estructuración en funciones, código comentado, etc.
- Si el **código no compila**, se restará un punto de la nota final del ejercicio
- Para la resolución de los ejercicios, **solo está permitido el uso de las librerías y funciones vistas en clase** de teoría y prácticas
- Tened el DNI sobre la mesa para poder identificaros en todo momento

Problemas

1. **(5 puntos)**

   La empresa *Boston Dynamics* te ha pedido que realices un programa para convertir sus antiguos ficheros de configuración de robots a un nuevo formato. Los ficheros originales, en formato texto, contienen en cada línea una instrucción de movimiento para el robot con la siguiente información: tipo de movimiento (una cadena de texto), tiempo que estará haciendo ese movimiento (expresado en segundos mediante un número real) y fuerza que hay que darle al motor para ejecutarlo (un número entero).

   Los posibles tipos de movimiento que se pueden dar son cuatro: `jump` (saltar), `walk` (andar), `crouch` (agacharse) y `somersault` (voltereta). En el fichero de texto, cada uno de estos campos viene separado por un punto y coma (`;`). Un ejemplo de fichero de entrada sería el siguiente:

   ```
   walk;2.5;20
   jump;120;30
   walk;90;50
   somersault;10.6;45
   crouch;3.5;8
   ```

   Si el fichero no se puede abrir, se debe mostrar un error por pantalla y finalizar el programa sin hacer nada. El formato del fichero será siempre correcto y nunca estará vacío, pero se debe comprobar que el movimiento introducido sea uno de los cuatro indicados anteriormente, que el tiempo introducido sea mayor que 0 y que la fuerza del motor esté comprendida entre los valores 0 y 100, ambos incluidos. En caso contrario, se ignorará dicha instrucción y se pasará a leer la siguiente. Deberemos llevar un contador del número de instrucciones que hay erróneas, de manera que al terminar de leer el fichero, solo si se ha encontrado alguna instrucción errónea, se muestre por pantalla el número de instrucciones incorrectas encontradas y el porcentaje que eso supone con respecto al total de instrucciones leídas. Por ejemplo:

   ```
   Encontradas 10 instrucciones erróneas de 200 (5%)
   ```

Una vez leído el fichero, se deberá generar la salida en el nuevo formato que nos han pedido. Para ello, el programa deberá crear un fichero binario que contendrá un registro inicial (`struct`) con el nombre del robot (una cadena de texto de 50 caracteres máximo) y su identificador (un número entero). Estos dos valores se le deberán pedir al usuario para que los introduzca por teclado antes de generar el fichero (el identificador será siempre correcto, pero el nombre del robot podría tener más de 50 caracteres y tendrás que recortarlo).

A continuación de este registro se deberán escribir en el fichero binario las acciones leídas del fichero de texto. Estas acciones se guardarán cada una en un registro (`struct`) que permita almacenar el tipo de movimiento (una cadena de texto de 20 caracteres máximo), el tiempo que lo ejecuta (un `float`) y la fuerza del motor (un `int`), como se describió más arriba.

Al igual que con el fichero de entrada, si no se puede abrir el fichero de salida se debe mostrar un error por pantalla y finalizar el programa sin realizar ninguna otra acción.

A la hora de ejecutar el programa, éste se puede llamar usando dos parámetros de entrada:

- `-i <fichero_entrada>`: indica el nombre del fichero de entrada en formato texto que queremos leer. Este parámetro es obligatorio
- `-o <fichero_salida>`: indica el nombre del fichero de salida en formato binario que vamos a generar. Este parámetro es opcional. Si no se especifica se le dará por defecto el nombre `output.dat`

Estos dos parámetros pueden aparecer en cualquier orden. Ejemplos de llamadas correctas:

```
$ programa -i atlas.txt -o atlas_new.dat
$ programa -i spot.txt
$ programa -o stretch.dat -i stretch.txt
```

Si se introduce cualquier otro parámetro, se repite alguno de los que son válidos o falta algún nombre de fichero, se debe mostrar un error por pantalla y finalizar el programa sin llevar a cabo ninguna otra acción.

2. **(5 puntos)**

Nos ha llamado el presidente de Worverk para que hagamos un nuevo prototipo del *software* de su robot de cocina. En concreto, debemos desarrollar el módulo que gestiona las recetas integradas. Tras reunirnos con nuestro equipo de desarrollo, diseñamos la aplicación pensando en el paradigma orientado a objetos, tal y como se aprecia en el diagrama UML de la figura que se muestra más abajo. Las clases a implementar, junto a sus métodos, se describen a continuación. **Puedes añadir nuevos métodos y atributos privados a las clases si lo necesitas, pero no públicos**.

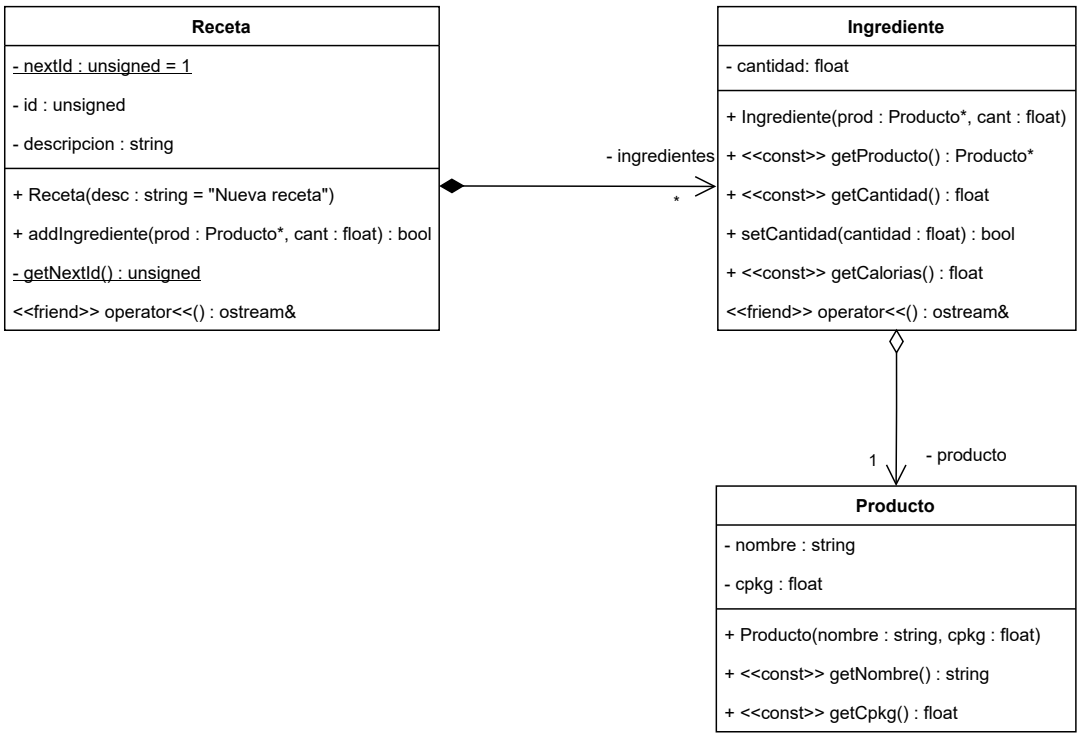
Clase Producto

La clase `Producto` representa un tipo de alimento específico. Almacena el nombre del producto (`nombre`) y la cantidad de kilocalorías por cada kilogramo (`cpkg`). Como métodos, la clase incluye un constructor con parámetros y el *getter* de cada atributo.

Clase Ingrediente

La clase `Ingrediente` representa un ítem a añadir a la receta. Cada `Ingrediente` tiene agregado un objeto `Producto`. También incluye la cantidad en kilogramos (`cantidad`) de dicho producto que se debe añadir a la receta. La clase `Ingrediente` debe implementar los siguientes métodos:

- `Ingrediente`: constructor que recibe el producto asociado y la cantidad a considerar. Si no se indica ningún producto (el puntero es `NULL`) o la cantidad es menor o igual a 0, el constructor debe lanzar una excepción
- `getProducto`: debe devolver el puntero al producto asociado
- `getCantidad`: debe devolver la cantidad de kilogramos del ingrediente
- `setCantidad`: recibe una cantidad en kilogramos del ingrediente. Si la cantidad es mayor a 0, debe actualizar el atributo `cantidad` y devolver `true`. En caso contrario, el método debe devolver `false` sin modificar el atributo



- `getCalorias`: devuelve la cantidad de kilocalorías que aporta este ingrediente (calculada teniendo en cuenta la cantidad de ingrediente y el número de calorías por kilo del producto)

Además, debe definirse el operador de salida. Por ejemplo, para un objeto `Ingrediente` asociado a un `Producto` con nombre "Patata", que incluye una cantidad de medio kilogramo, se mostraría por pantalla:

```
Patata (0.5 kg.)
```

Al final de la cadena no debe incluirse el salto de línea.

Clase Receta

La clase `Receta` representa las recetas que incluye el robot de cocina. Cada receta está compuesta por una serie de ingredientes. Además, cada objeto de esta clase almacena un código numérico que lo identifica (`id`) y la descripción de la receta (`descripcion`). Esta clase también almacena, de manera común para todos los objetos, el siguiente identificador que debe considerarse cuando se incluya una nueva receta (`nextId`).

Sus métodos funcionan de la siguiente forma:

- `Receta`: constructor que crea una receta con el nombre indicado por parámetro. Además, asigna al objeto el siguiente identificador, almacenado en `nextId`
- `addIngrediente`: recibe un producto y la cantidad del mismo que se debe añadir a la receta. El método debe crear el objeto de tipo `Ingrediente` e incluirlo en la lista de ingredientes. Si el objeto no se puede crear (ver constructor de `Ingrediente`), debe devolver `false`. Se debe usar una estructura `try/catch` para capturar la posible excepción lanzada por el constructor. Además, si había ya un ingrediente en la lista con el mismo producto, no debe incluirse nuevamente sino sobrescribir la cantidad del que ya estaba incluido. Si no ha habido ningún problema, el método devolverá `true`
- `getNextId`: devuelve el siguiente identificador válido e incrementa el valor del atributo `nextId` en una unidad

Además, debe definirse el operador de salida. Este operador comenzará escribiendo la palabra "Receta" seguida por un espacio en blanco y la descripción de la receta. Tras ello, se proporcionará una lista enumerada de

ingredientes junto con su cantidad. Finalmente, se debe mostrar la cantidad de calorías totales. Por ejemplo, asumiendo una receta de tortilla de patatas, que incluye medio kilo de patatas (770 kilocalorías por kilo) y 100 gramos de huevo (1550 kilocalorías por kilo), el operador debe producir la siguiente cadena:

```
Receta Tortilla de patatas
1: Huevo (0.1 kg.)
2: Patata (0.5 kg.)
Total kilocalorías: 540
```

Programa principal

Junto al enunciado se proporciona un fichero de ejemplo `main.cc`, así como un `makefile` para hacer la compilación. Este programa contiene el siguiente código:

```
int main(){
  Producto *tomate=new Producto("Tomate",180);
  Producto *patata=new Producto("Patata",770);
  Producto *huevo=new Producto("Huevo",1550);

  Receta ensalada("Ensalada murciana");
  ensalada.addIngrediente(patata,0.5);
  ensalada.addIngrediente(tomate,1);
  cout << ensalada << endl;

  Receta tortilla("Tortilla de patatas");
  tortilla.addIngrediente(huevo,0.1);
  tortilla.addIngrediente(patata,0.5);
  cout << tortilla << endl;
}
```

Al ejecutar este programa se debería producir la siguiente salida:

```
Receta Ensalada murciana
1: Patata (0.5 kg.)
2: Tomate (1 kg.)
Total kilocalorías: 565

Receta Tortilla de patatas
1: Huevo (0.1 kg.)
2: Patata (0.5 kg.)
Total kilocalorías: 540
```