Práctica 11: Algoritmos Voraces (Greedy). Cambio de Monedas

Objetivo

El objetivo principal de esta práctica consiste en desarrollar un ejemplo de Algoritmos Voraces (*Greedy*) como es el Problema de Cambio de Monedas. Para ello, se detalla a continuación las clases STL necesarias, los requisitos funcionales mínimos y opcionales, ejemplos de prueba, y los criterios de evaluación.

Clases STL necesarias

Para poder desarrollar más fácilmente el algoritmo del cambio de monedas, se puede hacer uso de las siguientes clases de la STL (*Standard Template Library*):

```
#include <vector>;
#include <list>;

std::vector<T>;
std::list<T>
```

También se puede hacer uso de cualquier otra clase STL que suponga una mejora significativa en el desarrollo del algoritmo. Para más información de dichas clases, se puede consultar la siguiente web: cppreference.com. Asimismo, se pueden usar las nuevas características del estándar C++11 del compilador g++, que se activan con la orden de compilación:

```
$ g++ -std=c++11 ...
```

Requisitos funcionales mínimos

Para superar la práctica, los requisitos funcionales mínimos exigidos son:

- Dada una cierta cantidad *n* (número real), el algoritmo debe devolver el conjunto de monedas correcto cuya suma total sea igual al valor de *n*, así como, el número total de monedas, usando el conjunto de monedas de curso legal en la Eurozona (2€, 1€, 50¢, 20¢, 10¢, 5¢, 2¢, 1¢)
- Dentro del esquema de estrategia voraz, el mejor candidato (moneda) será el valor más alto factible que queda en el conjunto de candidatos (referenciado en presentación del tema como "Algoritmo para el cambio de monedas"). Es decir, se deberá explorar el conjunto de candidatos desde la posición de la última moneda seleccionada, y no desde el principio, moneda a moneda.
- El programa realizado ha de ejecutarse indicando en la línea de comandos la cantidad *n* (número real). Debe emitir como resultado el conjunto de monedas y número total de monedas en la solución.

Ejecución y devolución de resultados:

```
$ monedas <n>
Solución: <conjunto_de_moneda>
Total monedas: <número_total_de_monedas>
```

Requisitos funcionales opcionales

Dentro de la evaluación global de la práctica, se valorarán positivamente los siguientes requisitos optativos:

- Devolver el cambio contando el número de monedas de cada tipo. Por ejemplo, si *n* = 7,98€, el cambio devuelto será *S* = {3×2€, 1€, 50¢, 2×20¢, 5¢, 2¢, 1¢}.
- Considerar tanto monedas como billetes (5€, 10€, 20€, 50€, 100€, 200€, 500€). Para resolver estos tipos de problemas habrá de llamarse al programa con la opción `-b`. ¿Cambia en algo el esquema del algoritmo original al incluir los billetes en el conjunto de candidatos?
- Diseñar un nuevo algoritmo más eficiente, con orden de complejidad O(*m*) siendo *m* en número de tipos de monedas (referenciado en presentación del tema como "Algoritmo alternativo para el cambio de monedas"). Para resolver estos tipos de problemas habrá de llamarse al programa con la opción `-o`

Ejemplos de prueba

Se puede probar la aplicación desarrollada usando los siguientes ejemplos:

Solución básica: *n* = 7,43€ → *S* = {2€, 2€, 2€, 1€, 20¢, 20¢, 2¢, 1¢}, con el número total de monedas igual a 8:

```
$ monedas 7.43
Solución: 2€, 2€, 2€, 1€, 20¢, 20¢, 2¢, 1¢
Total monedas: 8
```

Solución más elaborada: *n* = 5,34€ → *S* = {2×2€, 1€, 20¢, 10¢, 2×2¢}, con el número total de monedas igual a 7:

```
$ monedas 5.34
Solución: 2×2€, 1€, 20¢, 10¢, 2×2¢
Total monedas: 7
```

Solución más elaborada incluyendo también billetes: *n* = 425,34€ → *S* = {2x200€, 20€, 5€, 20¢, 10¢, 2×2¢}, con el número total de monedas igual a 8:

```
$ monedas -b 425.34
Solución: 2x200€, 20€, 5€, 20¢, 10¢, 2×2¢
Total billetes o monedas: 8
```

Solución más elaborada con algoritmo eficiente: n = 5,34€ → S = {2×2€, 1€, 20¢, 10¢, 2×2¢}, con el número total de monedas igual a 7 (resultado no varía respecto a sin -o):

```
$ monedas -o 5.34
Solución: 2×2€, 1€, 20¢, 10¢, 2×2¢
Total monedas: 7
```

Evaluación

Se evaluará positivamente los siguientes aspectos:

- Funcionamiento correcto de la modificación solicitada: si no se consigue realizar la modificación la práctica será valorada con un cero.
- Presentación en el laboratorio: el grado de funcionamiento de la práctica, y si desarrolla requisitos opcionales o mejoras significativas.
- Código subido en la tarea correspondiente a la práctica: buen diseño, estructura de clases y limpieza.

Teniendo superados los aspectos anteriormente descritos, una evaluación orientativa sería la siguiente:

- Si se desarrolla el algoritmo y se muestra la solución simple: hasta 5 puntos.
- Si se desarrolla una solución más elaborada (contando las monedas de cada tipo): hasta 7 puntos.
- Si se desarrolla un algoritmo más eficiente, que además proporciona el número de monedas de cada tipo de forma directa, valorándose la sencillez (en longitud y comprensión) del diseño propuesto: hasta 10 puntos.

Profesor responsable

Si tienen alguna duda más sobre la práctica, el enunciado o la corrección, pueden hacerla al correo electrónico del profesor Patricio García Báez.