Guion de prácticas

MPALABRADOS (player)

Marzo 2020

UNIVERSIDAD DE GRANADA

ETSIIT Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación

DECSAI

Metodología de la Programación

DGIM

Curso 2019/2020

Índice

1. Descripción

En esta práctica se va a desarrollar la siguiente capa de la arquitectura, según el plan de trabajo fiijado en el guión de la Práctica 1. En este caso, se va a implementar la clase **player**, según la documentación sobre la misma contenida en el fichero **player.h** (Sección 3.1). .

2. Práctica a entregar

Se deberá duplicar el proyecto de Netbeans de la práctica anterior y realizar los siguientes cambios (en todos ellos aparece la marca **@warning** avisando de las tareas de implementación que están pendientes).

- **player.h**
  Añadirlo al proyecto recién creado.
- **player.cpp**
  Completar la implementación y añadirlo al proyecto.
- Carpeta **tests**
  Eliminar los ficheros de test anteriores y susituirlos por los que están en Prado.
- **main.cpp**
  Sustituir al anterior y completar el código para realizar el siguiente programa.

  1. Leer un string con el ID del lenguaje, según el estándar ISO639-1.
  2. Crear una instancia de la clase **Language** con el anterior ID y mostrar el conjunto de caracteres permitido para ese lenguaje.
  3. Pedir un número entero.
  4. Crear una instancia de la clase **Bag**, inicializar el generador de números aleatorios con el número anterior y definir su contenido en base al lenguaje que se ha declarado anteriormente.
  5. Crear una instancia de la clase **Player** y llenarla por completo con caracteres de la bolsa. Este objeto player deberá estar siempre ordenado de la A a la Z.
  6. Utilizar la función **bestAdvice(...)** para usar el máximo de caracteres de player que forme una palabra válida en el diccionario creado. Si no es posible extraer, al menos una palabra de longitud 2, el programa debe terminar. Si, por el contrario, se encuentra dicha palabra, contenida en la instancia de **Player**, entonces se deben eliminar las letras correspondientes de player y volver a extraer más caracteres de la bolsa para dejar, de nuevo, a player con 7 caracteres, y volver a empezar.

7. Si los caracteres de la bolsa se agotan, entonces el programa debe continuar mientras haya caracteres en player y se puedan seguir sacando palabras válidas del diccionario.
8. El programa deberá de recordar todas las palabras que vayan encontrandose hasta el final del programa e informar del número de palabras que han salido y el número de letras empleadas.

2.1. Configuración de la práctica

La misma que en la práctica anterior

2.2. Ejecución de prueba

ID="EN", ENTERO=16

```
TYPE LANGUAGE: EN
Opening tree file ./languages/EN.tree
Trying to read 30400 words
OK 30400 words read
Opening ./languages/EN.scrabble
OK 26 Scrabble's letter read
ALLOWED LETTERS: EAIONRTLSUDGBCMPFHVWYKJXQZ
TYPE SEED (<0 RANDOM): 16
BAG (16-98) : PFRECRHRAOGDDTMASURWJFIENIMOISNDGBTABLOENTEX
ALGEROTQNVENAKZSUANCOIOYVIEEILEEHSAIOEDYAOUEIWRALIPTUT
PLAYER: CEFHPRR BAG(91)
GENERATING A WORD: PERCH ***

PLAYER: ADFGORR BAG(86)
GENERATING A WORD: ARDOR ***

PLAYER: ADFGMST BAG(81)
GENERATING A WORD: MAST ***

PLAYER: DFGJRUW BAG(77)
GENERATING A WORD: RUG ***

PLAYER: DEFFIJW BAG(74)
GENERATING A WORD: WIFE ***

PLAYER: DFIJMNO BAG(70)
GENERATING A WORD: DIMON ***

PLAYER: DFGIJNS BAG(65)
GENERATING A WORD: FINDS ***

PLAYER: ABBGJLT BAG(60)
GENERATING A WORD: BLAB ***

PLAYER: EGJNOTT BAG(56)
GENERATING A WORD: GOTTEN ***

PLAYER: AEEGJLX BAG(50)
GENERATING A WORD: EAGLE ***
```

```
PLAYER: JNOQRTX BAG(45)
GENERATING A WORD: TORN ***

PLAYER: AEJNQVX BAG(41)
GENERATING A WORD: AVEN ***

PLAYER: JKQSUXZ BAG(37)
GENERATING A WORD: UZ ***

PLAYER: AJKNQSX BAG(35)
GENERATING A WORD: SANK ***

PLAYER: CIJOOQX BAG(31)
GENERATING A WORD: COO ***

PLAYER: IIJQVXY BAG(28)
GENERATING A WORD: VIXI ***

PLAYER: EEIJLQY BAG(24)
GENERATING A WORD: JEIEL ***

PLAYER: AEEHQSY BAG(19)
GENERATING A WORD: EYES ***

PLAYER: ADEHIOQ BAG(15)
GENERATING A WORD: HIDE ***

PLAYER: AAOOQUY BAG(11)
GENERATING A WORD: QUAY ***

PLAYER: AEIOORW BAG(7)
GENERATING A WORD: WORE ***

PLAYER: AAIILOP BAG(3)
GENERATING A WORD: PAIL ***

PLAYER: AIOTTU BAG(0)
GENERATING A WORD: OUTTA ***

PLAYER: I BAG(0)
GENERATING A WORD: -
%%%OUTPUT
LANGUAGE: EN ID: 16
BAG (0):
PLAYER (1): I
23 words and 97 letters found
PERCH - ARDOR - MAST - RUG - WIFE - DIMON - FINDS -
BLAB - GOTTEN - EAGLE - TORN - AVEN - UZ - SANK -
COO - VIXI - JEIEL - EYES - HIDE - QUAY - WORE -
PAIL - OUTTA -
```

2.3. Validación de la práctica

Se debe ejecutar la script **doTests.sh** y comprobar que los resultados que aparecen por pantalla coinciden con lo indicado en cada caso de validación.

2.4. Entrega de la práctica

Se deberá ejecutar la script **doZipProject.sh** y subir a Prado, en las fechas que se indican en la temporización de la asignatura, el zip resultante, que está almacenado en la carpeta **.zip**/ del proyecto de Netbeans y siempre se llama **MPPractica.zip**. Para saber si el zip es correcto, seguir los siguientes pasos.

```
lcv:MP$ cd NetBeans/
lcv:NetBeans$ mkdir Test
lcv:NetBeans$ cd Test
lcv:Test unzip <expecificar-ruta>/MPPractica.zip
Archive:  ... MPPractica.zip
...<comprobar-descompresión>...
lcv:Test make clean
lcv:Test ./scripts/doTests.sh
Generating fresh binaries
...<comprobar-compilación>...
Press [RETURN] to continue
Test #1 (mp1920practica2 < tests//XXX.test)
...<comprobar-tests>...

Test #2 (mp1920practica2 < tests//YYY.test)
...<comprobar-tests>...
```

3. Código de la práctica

3.1. Clase Player

```
/**
 * @file player.h
 * @author DECSAI
 * @note To be implemented by students
 */

#ifndef PLAYER_H
#define PLAYER_H
#include <string>

#define MAXPLAYER 7 /// Max number of letters available for the player

/**
 * @class Player
 * @brief Class used to store the letters owned by the player, which is continuosly changing according to
     the evolution of the game.
 * It is stored as a CSTRING, with a \0 delimiter and all the internal operations must be carried out
     EXCLUSIVELY with the functions
 * provided in <cstring> See http://www.cplusplus.com/reference/cstring/ for details about cstrings
 *
 * All the operations in this class must leave the set of letters ordered lexicographically
 *
 **Please note that all the characters are stored in ISO8859 standard and in uppercase**
 */
class Player {
private:
char letters[MAXPLAYER+1]; /// Static vector to store a CSTRING
public:
/**
* @brief Basic constructor and initializer.
*/
Player();
/**
* @brief Returns the number of letters stored.
* @return The number of letters
*/
int size() const;
/**
* @brief Returns the number of letters stored. Although internally this set is stored in a CSTRING, the
     return value must be a STRING
* @return The set of letters
*/
std::string to_string() const;
/**
* @brief Resets the set of letters to 0 components
*/
void clear();
/**
* @brief Query if a given movement can be supported for the existing letters. That is, all the letters in
     the movement, including possible
* repetitions, must be present in the stored letters.
* @param s The string that is queried
* @return @retval true if the move can be supported by the stored letters, @retval false otherwise
*/
bool isValid(const std::string s) const;
/**
* @brief Remove the set of letters of the movement, including possible repetitions, from the set of stored
     letters
* @param m The string that is to be removed
* @return @retval true if the move can be supported by the stored letters, @retval false otherwise
*/
bool extract(const std::string s);
/**
* @brief Adds a set of additional letters to the existing letters whenever there
* is room for them. If the set of additional letters is too large, it does nothing
* @param frombag Set of letters to add
*/
void add(std::string frombag);
};

#endif /* PLAYER_H */
```

Estos métodos se deberán implementar en el fichero **player.cpp**, el cual además, deberá tener implementadas las siguientes funciones auxiliares.

```
/**
 * @brief Removes a position from a cstring
 * @param cstr The cstring
 * @param pos The position
 * @return The cstring is modified
 * @warning To be fully implemented
 */
void removeCString(char *cstr, int pos);

/**
 * @brief Sort a cstring from A to Z
 * @param cstr The cstring
 * @return The cstring is modified
 * @warning To be fully implemented
 */
void sortCString(char *cstr);
```

Más la siguiente función en el **main.cpp**

```
/**
 * @brief Given a string, it returns the longest word in the dictionary
 * that can be found from the characters in the string
 * @param s The string
 * @param l The language
 * @return A string s2 included in s which appears in the dictionary, otherwise it returns ""
 * @warning To be fully implemented. Additional functions are allowed if necessary
 */
string bestAdvice(const string & s, const Language &l);
```