Celia Arias Martínez
Lucía Salamanca López

Práctica Final

Introducción

La finalidad de esta práctica ha sido elaborar tres ejecutables, *letras*, *testdiccionario* y *cantidad_letras*.
Para ello hemos necesitado implementar cuatro módulos:

- Letra
- ConjuntoLetras
- BolsaLetras
- Diccionario

Letra

El TDA Letra es un objeto formado por tres elementos:

- Un `char` que representa el carácter de la letra
- Un `int` que representa la cantidad de veces que puede aparecer
- Un `double` que representa la puntuación de la letra

Hemos hecho que `Puntuacion` sea `double` en lugar de `int` porque utilizamos un objeto de clase `ConjuntoLetras` para calcular la frecuencia absoluta y relativa de cada letra en el diccionario, y la frecuencia relativa -que la guardamos en el campo `Puntuacion`- tiene que ser de tipo `double`.

En la clase `Letra` tenemos dos constructores: el de por defecto y uno a partir de un `char`, un `int` y un `double`. También tenemos tres métodos que modifican los elementos de la clase y tres métodos que observan dichos elementos.

También tenemos un operador `<` que devuelve un booleano indicando si una letra es menor que otra léxicamente y dos operadores `>>` y `<<` para leer y escribir en el flujo de entrada o salida.

ConjuntoLetras

El TDA ConjuntoLetras es un `set<Letra>`.

Vamos a comentar los métodos más relevantes dentro de esta clase:

```
set<Letra>::iterator BuscarLetra(char letra) const;
```

Su función es devolver un iterador apuntando al tipo de dato `Letra` cuyo campo `carac` coincida con el `char` letra. Para ello recorremos el `ConjuntoLetras` que lo invoca y comparamos cada `Letra` con `letra`.

```
bool leerDeFichero(const char *fichero);
```

Su función es modificar el `ConjuntoLetras` que lo invoca para que tenga los valores guardados en el fichero `fichero`. Hace uso del operador `>>`.

```
int Puntuacion(string palabra);
```

Su función es asignar una puntuación a una palabra dependiendo de los valores guardados en el campo `Puntuación` para cada una de sus letras. Para ello recorremos cada letra de la palabra, buscamos la letra en `ConjuntoLetras` y si la letra existe sumamos la puntuación de esa letra a la puntuación total.

```
pair<int,set<string> > MejoresPalabras(const set<string> & palabras, char modo)
```

Este método busca entre las palabras que le pasamos en el `set<string> palabras` y devuelve solo las mejores según el modo que le pasemos. Para ello diferenciamos dos casos: si `modo` es `P` o si es `L`. Si es `P` recorremos el `set<string>` y si la puntuación de la la palabra es mayor que el entero guardado en el `pair<int,set<string> >` borramos el `set<string>` del `pair` e insertamos la palabra y la nueva puntuación. Si es igual solo insertamos la palabra. Si `modo` es `L` hacemos lo mismo pero fijándonos en la longitud de cada palabra.

```
bool salvarFrecuenciasAFichero(const char *fichero);
```

Este método guarda el objeto ConjuntoLetras que lo invoca en el fichero pasado como parámetro.

```
void CalcularPorcentaje();
```

Este método calcula el porcentaje que le corresponde a cada `char` según el valor `cantidad` de cada uno de ellos y lo guarda en el campo `Puntuación` de

`Letra`. Para ello primero recorremos el `set<Letra>` y calculamos el total de frecuencias entre todas las letras. Después volvemos a recorrer el `set<Letra>` y calculamos el porcentaje que le corresponde a cada letra. Necesitamos crear un `ConjuntoLetras` auxiliar porque no podríamos borrar e insertar en el `ConjuntoLetras` que invoca al método pues lo estamos recorriendo en ese momento. También necesitamos crear una `Letra` auxiliar porque por defecto `set<Letra>` es de tipo constante.

BolsaLetras

El TDA BolsaLetras es un `multiset<char>`.

Lo utilizamos para guardar todos los char de `ConjuntoLetras` tantas veces como indique su campo `cantidad`.

Vamos a comentar los principales métodos de la clase.

```
multiset<char> seleccionaAleatorio(int cantidad);
```

Su función es seleccionar de forma aleatoria tantos elementos como indique `cantidad` de la `BolsaLetras` que lo invoca. Para ello creamos un `multiset<char>` donde vamos guardando los elementos seleccionados y en un bucle generamos los elementos aleatorios, teniendo en cuenta que si el número sale repetido hay que coger otro, para no tener repeticiones.

```
BolsaLetras(const ConjuntoLetras & conjun)
```

Constructor de `BolsaLetras` dado un `ConjuntoLetras`. Recorre el `ConjuntoLetras` e inserta el campo `caracter` de cada `Letra` tantas veces como indique el campo `cantidad`.

```
bool leerDeFichero(const char *fichero)
```

Modifica el objeto `BolsaLetras` que lo invoca según los valores guardados en `fichero`.

Diccionario

El TDA BolsaLetras es un `set<string>`.

Lo utilizamos para guardar todas las palabras que hay en un diccionario.

Los principales métodos son:

```
vector<string> PalabrasLongitud(int longitud);
```

Devuelve las palabras del diccionario cuyo tamaño sea igual a `longitud`. Para ello recorremos el `Diccionario` y guardamos las palabras que coincidan en un `vector<string>`, que luego devolveremos en el return.

```
bool Esta(string palabra);
```

Devuelve true si la palabra está en el diccionario y false en caso contrario. Para ello utilizamos el método `count` de `set`.

```
set<string> SacarPalabras(const multiset<char> & seleccionadas);
```

Devuelve el conjunto de palabras del diccionario que se pueden formar dadas unas letras determinadas. Para ello recorremos el objeto `Diccionario` estudiando solo las palabras que sean más pequeñas que el tamaño de `seleccionadas`. Creamos una copia de `seleccionadas` y recorremos esas palabras `char` a `char`. Buscamos cada `char` en `seleccionadas`: si está lo borramos y seguimos con la siguiente letra, si no está salimos del bucle. Al terminar cada iteración si el booleano `fin`, que determina si hemos salido forzosamente del bucle, es `false` añadimos la palabra al `set<string> resultado`.

```
void buscarFrecuenciaLetras(ConjuntoLetras & conjunto ) const;
```

Calcula la cantidad de veces que aparece una letra en el `Diccionario` y la guarda en el apartado `Cantidad` del `ConjuntoLetras`. Para ello recorremos el diccionario y cada palabra de ese diccionario. Si esa letra aparece en `ConjuntoLetras` sumamos +1 a su `Cantidad`. Hemos tenido que crear una `Letra` auxiliar porque por defecto `set<Letra>` es constante, por lo que en su lugar hemos creado una letra igual, hemos cambiado su `Cantidad`, hemos borrado la letra antigua e insertado la nueva.

Ejecutables

testdiccionario

En este programa leemos de un fichero un diccionario. Cuando hemos cargado el diccionario le pedimos al usuario una longitud para así mostrar por pantalla las palabras con dicha longitud incluidas en el diccionario.

En segundo lugar se le pide al usuario una palabra para comprobar si dicha palabra se encuentra en el diccionario.

cantidad_letras

Para hacer funcionar este programa debemos pasarle como argumento un diccionario, un fichero con Letras permitidas y el nombre del fichero de salida. Este programa lee un diccionario y a partir de él cuenta la cantidad de veces que se encuentra una letra en el diccionario. Como resultado genera un archivo de salida en el que podemos ver el carácter, la frecuencia absoluta y la frecuencia relativa del diccionario pasado por argumento.

Para ello, cuando el programa lee el ConjuntoLetras pasado por argumento, pone los valores de cada Letra a 0 (la cantidad y la puntuación). Seguidamente calcula las frecuencias absolutas de las letras con el método buscarFrecuenciaLetras de Diccionario y calcula la frecuencia relativa con el método CalcularPorcentaje de ConjuntoLetras. Finalmente guarda este ConjuntoLetras en un fichero.

letras

Podríamos decir que este es el programa "principal" de la práctica. Se trata de un juego parecido a "Cifras y Letras". Con un determinado número de letras el jugador debe formar una palabra, intentando conseguir la mayor puntuación posible. Después la máquina muestra sus mejores respuestas.

Los parámetros necesarios para ejecutar el programa son:

- **Un diccionario**: en él se encuentran las palabras válidas para ser admitidas como respuesta.
- **Un fichero letras.txt**: en dicho fichero, además de los carácteres, también está la cantidad de veces que las letras se encuentran en la bolsa (parecido a la bolsa donde se encuentran las fichas del Scrabble) y la puntuación.
- **Número de letras**: número de letras del que posee el jugador para formar la palabra.
- **Modalidad del juego (L o P)**: si la modalidad es L la puntuación de la palabra será su longitud, en cambio si la modalidad es P, la puntuación vendrá dada por la puntuación de cada letra (esta puntuación se encuentra en el fichero de letras permitidas).

Cuando el programa lee el Diccionario y el ConjuntoLetras, crea una BolsaLetras a partir del conjunto y selecciona las letras de forma aleatoria con las que el jugador va a poder formar la palabra. Después se las muestra al usuario y éste ha de escribir su respuesta, luego comprueba si dicha palabra está formada por las letras seleccionadas(función EstaSeleccionados) y si se encuentra en el diccionario leído, si es así muestra la puntuación de la palabra según el modo elegido.

Ahora pasa a ser el turno de la máquina, con el método de ConjuntoLetras MejoresPalabras selecciona las respuestas con mayor puntuación según el modo elegido y luego las muestra por pantalla.

Por último se le pregunta al usuario si desea seguir jugando, dependiendo de la respuesta se vuelve a repetir el juego explicado anteriormente.