Sesión 5

Diseño modular en C++ (I)

5.1 Primer caso de estudio: lista de palabras

Retomemos el caso de estudio *"Factor Psi"*, que se encuentra en los apuntes de la asignatura. En esencia, se trata de obtener la frecuencia de la palabra más frecuente de un texto. Para estudiar mejor los resultados, no consideramos el cálculo del porcentaje. El texto consta de una o más palabras separadas por blancos o saltos de línea y no tendrá más de `MAXNUMPAL` palabras distintas, cada una de ellas no mayor de `MAXLONG` caracteres. El texto se introduce por el canal standard de entrada. Podéis hacer las primeras pruebas escribiéndolo por teclado, pero resultará interesante que también lo probéis redireccionando la entrada a un fichero como `ejemplo.txt`.

Ayuda: El módulo `Palabra` está especificado e implementado. Encontraréis el fichero `.hpp` en `INCLUDES_CPP` y el fichero `.o` en `OBJETOS_CPP`. Entre las operaciones del mismo se incluyen las de lectura y escritura de palabras. El criterio para determinar el final del texto puede deducirse de las operaciones del tipo (ver la función `marca`) o bien puede ser definido por el propio usuario.

5.1.1 Ejercicio: programa principal

Implementad el programa principal suponiendo que el módulo `ListaPalabras` también está especificado e implementado en los lugares anteriormente mencionados. Probadlo con los textos que queráis, además de con el fichero `ejemplo.txt`. Controlad el número máximo de palabras distintas que podéis introducir mediante la consultora `long_maxima`.

5.1.2 Ejercicio: juego de pruebas especial

Producid una serie de ficheros que contengan juegos de pruebas especializados en situaciones límite detectables a partir de la especificación de `ListaPalabras`, por ejemplo

- una lista de tamaño 1
- que la palabra de frecuencia máxima sea la primera del texto
- que la palabra de frecuencia máxima sea la última del texto

5.1.3 Módulo `ListaPalabras`

Obtened vuestra propia implementación de `ListaPalabras` y comprobad que el principal sigue funcionando con ella (linkadlo con el `.o` que obtendréis en vuestro propio directorio). Usad los mismos juegos de pruebas que en los ejercicios anteriores.

En la solución de los apuntes, la operación `anadir_palabra` utiliza una función privada para buscar si una palabra está ya en la lista. En vuestro `ListaPalabras.cpp` podéis implementarla de dos maneras.

- Declarándola como una operación más (aunque privada) de la clase: eso obliga a añadir su cabecera a la parte privada del `ListaPalabras.hpp` y a ponerle a su nombre el prefijo `ListaPalabras::` en su cabecera del `ListaPalabras.cpp`.

- Declarándola tal cual, de forma que no sea una operación de la clase. En ese caso, es como una operación de cualquier otra clase o programa. Al no ser de la clase, no tiene un parámetro implícito y dentro de su ámbito de visibilidad no se puede acceder a los campos de la clase.

5.1.4 Ejercicio: otro juego de pruebas

Producid otra serie de juegos de pruebas especializados en situaciones límite relacionadas con elementos específicos de la implementación de `ListaPalabras`. Comprobad que vuestro programa los pasa correctamente.

Algunas situaciones interesantes son:

- que la palabra de frecuencia máxima esté en la última posición ocupada de la lista

- llenar una lista (introducir MAXNUMPAL palabras distintas: usad un MAXNUMPAL pequeño) y, a continuación, añadir una palabra que no esté en la posición MAXNUMPAL

- añadir a una lista llena una palabra que esté en la posición MAXNUMPAL

5.1.5 Ejercicio: las *n* palabras más repetidas

Dado un texto como el del ejercicio anterior y un número *n* mayor que 0, obtener la lista de las *n* palabras más frecuentes, ordenadas de mayor a menor frecuencia. Si el número de palabras distintas del texto es menor o igual que *n*, se obtendrán todas. Si hay varias palabras con la misma frecuencia, tendrán prioridad las palabras que alcanzaron primero dicha frecuencia.

Ejemplo: si *n* es 3 y el texto es `"q w e r r e w q w e q r"`, el resultado deberá ser:

```
w 3
e 3
q 3
```

Suponed que esta operación se utilizará con bastante frecuencia, por lo que no resultará rentable ordenar la estructura cada vez. En vez de eso, modificad la operación `añadir` para que mantenga ordenada la estructura.

Probad todos los casos especiales que podáis identificar.

5.1.6 Material adicional

Os recordamos el programa principal y las especificaciones de los módulos. Cuando os pongáis a implementar la clase `ListaPalabras` usad los apuntes de la asignatura.Tened en cuenta que al traducir los módulos a C++ habrá inevitablemente ciertas modificaciones, por lo que deberéis estudiar cuidadosamente los ficheros `.doc` de la sesión.

```
Programa Frecuencia Máxima;

   Sobre Palabra, ListaPalabras;

   var l: ListaPalabras;
       p: Palabra;
       f: natural;

   l:=crear_lista();
   leer_palabra(p);
   mientras p != ''.'' hacer
      añadir_palabra(l,p);
      leer_palabra(p);
   fmientras;
   f:=max_frec(l)



Modulo ListaPalabras (Especificación);

  Sobre Palabra;

  Constante MAXNUMPAL;

  funcion crear_lista () dev l: ListaPalabras
    /* cierto */
    /* l es una lista vacia */

  accion añadir_palabra(e/s l: ListaPalabras; e p: Palabra)
    /* p esta en l o la longitud de l es < MAXNUMPAL */
    /* l conserva su informacion anterior y si p esta en l
       se incrementa su frecuencia; si no, se añade con frecuencia 1 */
```

```
  funcion longitud (l: ListaPalabras) dev n: nat
    /* cierto */
    /* n es el numero de palabras de l */

  funcion max_frec (l: ListaPalabras) dev f: nat
    /* cierto */
    /* f es la mayor frecuencia de las palabras de l */


Modulo Palabra (Especificación);

  Constante MAXLONG;

  funcion crear_palabra() dev p: Palabra
    /* cierto */
    /* p es una palabra vacia */

  accion añadir_letra(e/s p: Palabra; e char c)
    /* La longitud de p es menor que MAXLONG */
    /* El caracter c se ha añadido al final de p */

  funcion consultar_letra (p: pal; i: natural ) dev c: caracter
    /* 1 <= i <= longitud de p */
    /* c =  caracter i-ésimo de p */

  funcion longitud(p: Palabra) dev l: natural
    /* cierto */
    /* l es el numero de letras de p */

  funcion iguales (p1, p2: Palabra) dev b: booleano
    /* cierto */
    /* b indica si p2 es igual a p2: tienen la misma longitud
     y los mismos caracteres en el mismo orden */
```