ETSIIT – Universidad de Granada (UGR)

Grado en Ing. Informática – 2ºC – Estructura de Datos

Práctica 2.- Abstracción

INDICE

ETSIIT – Universidad de Granada (UGR)

Grado en Ing. Informática – 2ºC – Estructura de Datos

TDA FRASE

1. ESPECIFICACIÓN

Una instancia c del tipo de dato abstracto Frase es un objeto que mantiene la información de una frase "hecha" en el idioma origen, y todas las posibles traducciones en el idioma destino.

Ejemplo: No worries;Sin problema;No te preocupes

OPERACIONES BÁSICAS

***NOTA:** No son necesarios los Constructores, Destructor y Operador de asignación, ya que uso la clase vector de la STL, y vienen por defecto.

- Consultores :
  - **NumTraducciones :** Devuelve el número de traducciones de la frase.
  - **GetOrigen :** Devuelve la frase origen.
  - **GetDestino :** Devuelve la cadena de las frases traducciones.
- Operadores :
  - **Operadores de E/S :** Se encargan de gestionar la lectura de una frase desde un flujo cualquiera, ya sea un fichero o el mismo teclado. Y de imprimirla en el mismo formato que tienen en los ficheros dados.

*Método adicional : **Push_Back :** Inserta por el final del vector de traducciones una nueva

2. DIFERENTES ED PARA EL *tipo Rep*

1. PRIMERA OPCIÓN

Utilizar,para las traducciones, un vector estático de tipo string, de un tamaño lo suficientemente grande para asegurarnos que nos caben todas las traducciones que vayamos a insertar por el final. Ésta opción no es muy recomendable, ya que estaríamos usando mucha más memoria de la que, en realidad, necesitamos. Además, sería muy ineficiente. Ya que tendríamos que garantizar el buen acceso a posiciones de memoria, y una buena gestión de la capacidad de nuestro vector. Para ello, habría que ayudarse de variables, de tipo entero, una de clase, para el tamaño máximo, y otra para saber cuantas casillas hay ocupadas hasta el momento. Esto podría llevarnos a cometer un error en cualquier momento. Para la frase origen utilizaría una variable de tipo string.

```
class Frase {
private:

  static int TAM = 5;
  int ocupadas;
  string origen;
  string[TAM] cadTraducciones;

  bool Invariante ();
  void MoreSize ();
```

2. SEGUNDA OPCIÓN (bastante buena)

Declararnos un struct, dentro de la clase, al que llamaría Traduc, que contenga 2 datos : un string con el contenido de una traducción y un puntero, de tipo Traduc, apuntando a la traducción siguiente.
Como datos miembro tendría "un string" para la frase "origen" y un puntero, de tipo Traduc, apuntando a la primera traducción. Esta opción es la mejor posiblemente ya que, a priori, no sabemos cuántas traducciones tiene cada frase, y la estructura que estaríamos usando (de celdas enlazadas) nos facilitaría las tareas de inserción y borrado de traducciones, que se harían de manera muy eficiente (O(1)), sin tener que preocuparnos por el exceso de almacenamiento de "basura".
Por contra, en función del programa principal que tenemos que hacer funcionar, nos obliga a hacer recorridos por nuestras traducciones, cuya eficiencia ya pasa a ser O(n), siendo n el

número de traducciones insertadas hasta el momento. Para ello, podríamos facilitarnos la vida añadiendo como dato miembro un entero que sea un contador de las traducciones que tenemos hasta el momento. De no caer en esto último, tendríamos que recorrer toda la cadena de celdas para saber cuántas tenemos. Cosa que para el acceso a ciertas traducciones determinadas es inevitable hacer, el recorrer las anteriores a ella.

```
class Frase {
private:

  struct Traduc {
      string traduccion;
      Traduc *siguiente;
  };

  string origen;
  int NTraduc; //Contador
  Traduc *cadTraducciones;
```

También se podrían añadir algunos métodos sencillos en el struct, para facilitar la implementación del TDA Frase, como un método que devuelva el número de traducciones(añadiendo un contador al struct), o como un operador de acceso a posiciones [].

3. TERCERA OPCIÓN (ELEGIDA)

Utilizar un vector dinámico, para las traducciones. En concreto, la clase vector de la biblioteca STL.
Frente a la opción anterior, es prácticamente igual en cuestión de eficiencia, ya que su tiempo amortizado a la hora de hacer copias es O(n), que es el mismo que las celdas.
Como ventaja frente a la anterior, tiene que el acceso a posiciones es constante y que su implementación es bastante más sencilla, ya que están todas las funciones necesarias para trabajar con el vector implementadas.

```
class Frase {
private:

  string origen;
  vector<string> cadTraducciones;
```

3. TIPO REP ELEGIDO

Para mis datos miembro, únicamente necesito una variable, de tipo string, para la frase en el idioma origen y un vector de la STL, de tipo string, para la cadena de las traducciones.
Con esta implementación no tengo que preocuparme por la gestión de la memoria, porque ya se encarga de ello, internamente, la implementación de la clase vector.

```
class Frase {
private:

  string origen;
  vector<string> cadTraducciones;
```

4. INVARIANTE DE LA REPRESENTACIÓN Y FUNCIÓN DE ABSTRACCIÓN

INVARIANTE DE LA REPRESENTACIÓN

- rep.cadTraducciones.size() >= 1, a excepción del momento de creacion del objeto, que estará vacío completamente
- rep.cadTraducciones[i] != rep.cadTraducciones[j]
  Para todo i,j => $i \neq j$, es decir, no hay traducciones repetidas.

FUNCIÓN DE ABSTRACCIÓN

Un objeto valido rep del TDA Frase representa al valor:

fA(rep) = rep.origen;rep.cadTraducciones[0];...;rep.cadTraducciones[rep.cadTraducciones.size()-1]

TDA CONJUNTO DE FRASES

1. ESPECIFICACIÓN

Una instancia c del tipo de dato abstracto ConjuntoFrases es un objeto que contiene una sucesión de frases leídas desde un fichero, o cualquier entrada posible.

Ejemplo :

```
Not Bad;No esta mal
Not for nothing;No es por nada
Not half;Ya lo creo
Not half;Por supuesto
Not my business;No es asunto mio
...
```

OPERACIONES BÁSICAS

***NOTA:** No son necesarios los Constructores, Destructor y Operador de asignación, ya que uso la clase vector de la STL, y vienen por defecto.

- Consultores :
  - **Esta :** Recorre el fichero buscando en las origen de cada frase y devuelve si el string pasado como parámetro coincide con alguna frase origen.
  - **GetTraducciones :** Dado el campo origen de una frase, que está pasado como parámetro, devuelve la frase completa a la que pertenece.
  - **Contenga :** Devuelve un conjunto de frases que, en cuya frase origen, contenga la subcadena dada como parámetro.
  - **Size :** Devuelve el número de frases que contiene el conjunto.

- Operadores :
  - **Operadores de E/S :** Se encargan de gestionar la lectura de un conjunto de frases desde un flujo cualquiera, ya sea un fichero o el mismo teclado. Y de imprimirlo en el mismo formato que tiene en los ficheros dados.

*Método adicional : **Push_Back :** Inserta por el final del conjunto una nueva Frase, que no este repetida.

2. DIFERENTES ED PARA EL *tipo Rep*

1. PRIMERA OPCIÓN

Meter todo en un único vector estático de Frases. Mala opción, puesto que no sabemos cuantas frases tenemos. Nos restringiría mucho ésta implementación. En términos de eficiencia, estaríamos tardando demasiado a la hora de hacer inserciones ya que, o reservamos mucho espacio, tanto como para que nos sobre (lo que haría que malgastáramos memoria...). Prácticamente tendríamos los mismos problemas que en la primera opción del TDA Frase, ya que el tipo de Rep. Sería muy parecido.

```
class ConjuntoFrases {
private:

  static int TAM = 50;
  int ocupadas;
  Frase[TAM] ConjFrases;

  bool Invariante ();
  void MoreSize ();
```

2. SEGUNDA OPCIÓN

Al igual que la opción 2 del TDA Frase, es el tipo de Representación que usa la estructura de celdas enlazadas. Una opción que sería bastante buena para insertar nuevas Frases, porque su tiempo en el peor de los casos es constante y, además, no necesitaríamos hacer reserva de un gran espacio de memoria. Únicamente iríamos almacenando sobre la marcha, de forma dinámica, el tamaño que fuésemos requiriendo en cada momento. Pero, ojo, tendríamos que tener cuidado a la hora de liberar la memoria cuando ya no estemos utilizándola, ya que nos podríamos ver metidos en el problema de la segmentación, que es aquel en el que nuestro programa intenta acceder a zonas de memoria que, como tenemos mal declaradas, no habíamos reservado, o contienen basura.

```
class ConjuntoFrases {
private:

  struct Info {

    Frase linea;
    Info* Sig;
  };

  Info* Primera;
```

3. TERCERA OPCIÓN (ELEGIDA)

Utilizar un vector dinámico de Frases, basándonos, de nuevo, en la clase vector de la biblioteca STL. Como explico en la opción 3 del TDA Frase, es una de las mejores opciones a tener en cuenta y, en este caso, creo que es la mejor, por lo que voy a contar en el siguiente punto. Únicamente, necesitaremos declarar un vector dinámico de Frases como dato miembro de nuestro TDA Conjunto de Frases.

```
class ConjuntoFrases {
private:

  vector<Frase> ConjF;
```

3. TIPO REP ELEGIDO

Para mis datos miembro, únicamente necesito, como he dicho en el apartado anterior, un único vector de la biblioteca STL, de tipo Frase, para el conjunto con todas las frases.
Con esta implementación no tengo que preocuparme por la gestión de la memoria, porque ya se encarga de ello, internamente, la implementación de la clase vector.
Además, es la mejor elección, a parte de la comodidad en la implementación, por las cuestiones de eficiencia. Ya que un vector es más rápido para las consultas de sus posiciones y, en este caso, únicamente vamos a hacer inserciones por detrás, cuya eficiencia en el caso promedio es O(1).

```
class ConjuntoFrases {
private:

  vector<Frase> ConjF;
```

4. INVARIANTE DE LA REPRESENTACIÓN Y FUNCIÓN DE ABSTRACCIÓN

INVARIANTE DE LA REPRESENTACIÓN

- rep.ConjF[i].GetOrigen() != rep.ConjF[j].GetOrigen(),
  Para todo i,j => $i \neq j$, es decir, no habrá frases repetidas, frases con el mismo campo origen.
- rep.ConjF.size() >= 1. Excepto en el momento de creación de un objeto de tipo ConjuntoFrases, este objeto siempre contendrá, al menos, 1 Frase.

FUNCIÓN DE ABSTRACCIÓN

Un objeto valido rep del TDA Conjunto de Frases representa al valor:

```
fA(rep) = rep.ConjF[0]
          rep.ConjF[1]
          …
          rep.ConjF[i]
          …
          rep.ConjF[rep.ConjF.size()-1]
```