Universidad de Buenos Aires
Facultad de Ingeniería

75.41 – Algoritmos y Programación II

Cátedra Ing. Patricia Calvo

2º Cuatrimestre 2008

Trabajo Práctico

Corrector Ortográfico

Versión 1.0

Índice

1. Enunciado

El objetivo del trabajo práctico es modelar un corrector ortográfico.

1.1 - Definición

Podemos definir que un corrector ortográfico es un proceso o herramienta que ofrece a una aplicación o servicio online la posibilidad de corregir la ortografía de cualquier tipo de texto, ayudando al usuario en su escritura.

Por lo general, los correctores ortográficos vienen incluidos como herramientas en aplicaciones de edición de texto, clientes de e-mails, navegadores, etc. También suelen emplearse en los buscadores de Internet para controlar las palabras que escribe el usuario y así poder hacerle sugerencias.

En síntesis es una aplicación completa y aislada que se puede incluir en otras aplicaciones para mejorar sus usos.

En nuestro trabajo práctico se deberá modelizar para que el Corrector Ortográfico se pueda utilizar como una aplicación independiente o desde otra aplicación mayor.

1.2 - ¿Cómo construir un Corrector Ortográfico?

Nuestro corrector ortográfico tiene que poder corregir palabras de cualquier idioma y dentro de estos, de cualquier tema ¿difícil, no? Para esto vamos a dividir el algoritmo en varias etapas, que son:

1.3 - Proceso

1) Etapa de Filtrado.
2) Etapa de Aprendizaje.
3) Ciclo para las palabras ingresadas que no existen en nuestro diccionario:
    a. Obtención de candidatos
    b. Ponderación de candidatos
    c. Impresión de resultados

1.3.1 - Etapa de filtrado

La etapa de filtrado es muy importante para limitar el error, en esta etapa se aplicará sobre el archivo de texto de entrada, para eliminar los caracteres no válidos.

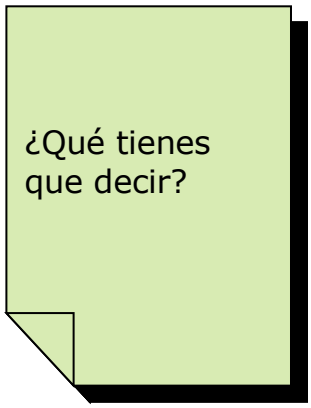
Por ejemplo si nuestro archivo viene de la siguiente forma:

Nuestro texto filtrado sería:

"Hola Mundo" sin el punto, ya que el punto no es una letra del abecedario.

Sin embargo, además de incluir los caracteres "a, b, c, d, e, f, g, h, i, j, k, l, m, n, ñ, o, p, q, r, s, t, u, v, w, x, y, z" tenemos que tener cuidad con algunos caracteres gráficos que si son útiles, como los acentos.

Es decir que si nuestro archivo de entrada es:



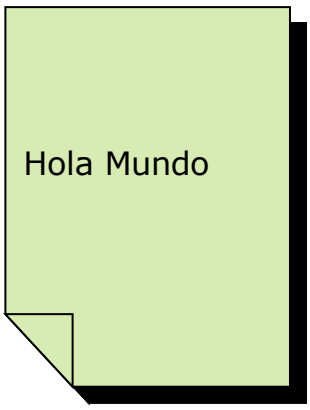
Nuestro texto filtrado sería:

"Qué tienes que decir", agregando a nuestro filtro las letras "á, é, í, ó, ú" como válidas. Este filtro comentado es el filtro básico y sugerido para nuestro Corrector Ortográfico, sin embargo cada grupo podrá modificarlo justificando sus decisiones.

1.3.2 - Etapa de aprendizaje

Para posibilitar que el algoritmo corrector funcione con cualquier idioma o tema, vamos a utilizar un diccionario que será la fuente de palabras correctas. Como base utilizaremos cualquier libro electrónico (llamado comúnmente EBook) en formato de texto plano.
Este archivo, luego de pasar por la etapa de filtrado, será el que contiene las palabras válidas para tomas como diccionario.
Si este archivo de texto esta escrito en castellano, nuestro corrector será en castellano, si esta escrito en varios idiomas, corregirá varios idiomas. Si habla de medicina, tendremos un diccionario con dicho tema, y así podremos adaptar nuestro vocabulario según el archivo de entrada.
El algoritmo de aprendizaje se correrá una única vez sobre el archivo de entrada, dando como resultado una estructura, la cual se leerá las veces que sea necesaria para ejecutar las correcciones.

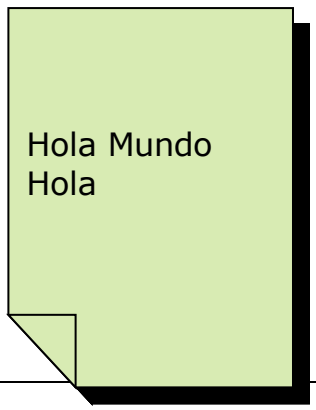
Por ejemplo, si el archivo de texto es:



Nuestro diccionario contendrá las palabras:

a) Hola
b) Mundo

Ahora bien, si nuestro texto de entrada es de la forma:



Nuestro diccionario contendrá las mismas palabras:

a) Hola
b) Mundo

Pero en este caso tenemos una connotación distinta para la palabra Hola. Para mejorar nuestro diccionario vamos a ponderar cada palabra con la cantidad de apariciones de la misma, quedando:

a) Hola (2 apariciones)
b) Mundo (1 aparición)

Y para dar un paso más sobre nuestro diccionario, mostramos el caso siguiente:

Debido a que C++ es "Case Sensitive" esto daría un resultado para nuestro diccionario como el siguiente:

a) Hola (1 aparición)
b) Mundo (1 aparición)
c) Hola (1 aparición)

Para corregir esto, pasaremos todo nuestro diccionario a minúsculas, quedando:

a) hola (2 apariciones)
b) mundo (1 aparición)

Este es el algoritmo básico para nuestro diccionario ponderado, cada grupo deberá modificarlo para que responda mejor para ciertos casos, por ejemplo que las palabras además de estar ponderadas por la cantidad de apariciones,

también lo estén por la cantidad de silabas, por ejemplo en el archivo de entrada "que queso", al buscar la palabra mal escrita "ques", el resultado será "queso".

1.3.3 - Ciclo de búsqueda

a) Búsqueda de candidatos.

En nuestro ejemplo si queremos corregir la palabra "ola" sobre nuestro diccionario, primero vamos a tener que buscar una serie de candidatos, que serán:

1.3.3.1 - Borrado de una letra:

Borramos cada una de las letras, y todas las palabras resultantes se insertan en el conjunto de candidatos, quedando:
la
oa
ol

1.3.3.2 - Alteraciones de una letra:

Las alteraciones consisten en cambiar la primera letra de la palabra por cada una de las letras del abecedario, luego la segunda letra de la palabra y así con cada una de las posiciones de la palabra, quedando:

ala
oaa
ola
bla
oba
olb
cla
oca
olc
dla
oda
old
ela
oea
ole
fla
ofa
olf
gla
oga
olg
hla

oha
olh
ila
oia
oli
jla
oja
olj
kla
oka
olk
lla
ola
oll
mla
oma
olm
nla
ona
oln
ola
ooa
olo
pla
opa
olp
qla
oqa
olq
rla
ora
olr
sla
osa
ols
tla
ota
olt
ula
oua
olu
vla
ova
olv
wla

owa
olw
xla
oxa
olx
yla
oya
oly
zla
oza
olz

1.3.3.3 - Inserción de una letra:

Las alteraciones consisten en agregar en la primera posición de la palabra cada una de las letras del abecedario, luego la segunda letra de la palabra y así con cada una de las posiciones de la palabra, quedando:

aola
oala
olaa
olaa
bola
obla
olba
olab
cola
ocla
olca
olac
dola
odla
olda
olad
eola
oela
olea
olae
fola
ofla
olfa
olaf
gola
ogla
olga
olag
hola
ohla
olha
olah
iola
oila
olia
olai
jola
ojla
olja

olaj
kola
okla
olka
olak
lola
olla
olla
olal
mola
omla
olma
olam
nola
onla
olna
olan
oola
oola
oloa
olao
pola
opla
olpa
olap
qola
oqla
olqa
olaq
rola
orla
olra
olar
sola
osla
olsa
olas
tola
otla
olta
olat
uola
oula
olua
olau

vola
ovla
olva
olav
wola
owla
olwa
olaw
xola
oxla
olxa
olax
yola
oyla
olya
olay
zola
ozla
olza
olaz

La suma de todas estas palabras es el conjunto de candidatos posibles para la corrección.

b) Ponderación de candidatos

Ahora buscamos cada palabra de este conjunto en nuestro diccionario y obtenemos su ponderación, es decir la cantidad de apariciones. Si la palabra del punto a) no aparece en nuestro diccionario, descartamos la misma, quedándonos solamente con las palabras que si tengan sentido.

El resultado de esto, ordenado por ponderación descendiente, son las palabras del resultado para sugerir al usuario su modificación.

2 - Objetivos del TP

Primera Parte:

1) Diseñar la solución al problema planteado.
2) Realizar un diagrama con la solución y las relaciones entre las clases.

Segunda Parte:

3) Implementar la solución al problema en C++.
4) Modificar el algoritmo de Ponderación para que arroje mejores resultados.
5) Utilizar la clase Cronometro para medir los tiempos de la aplicación con cada una de las estructuras en el Diccionario Ponderado: vector, lista. Sacar conclusiones.

Tercera Parte:

6) Diseñar la clase Árbol.
7) Implementar la clase Árbol.
8) Utilizar la estructura Árbol en la estructura del Diccionario Ponderado y comparar los tiempos con los tiempos de las demás estructuras.
9) Escribir las conclusiones.

3 - Apéndice A

Clase Cronometro:

La clase cronometro mide los tiempos que transcurren en determinados intervalos.
Las secuencias de medición pueden ser:

a)
Constructor();
Parar();

Acumula el tiempo transcurrido entre que se ejecuto el constructor y se llamo el método parar.

b)
Iniciar();
Parar();

Acumula el tiempo transcurrido entre que se ejecuto el método iniciar y se llamo el método parar.

c)
Iniciar();
Pausar();
Continuar();
Parar();

Acumula el tiempo transcurrido entre que se ejecuto el método iniciar y se llamo el método parar, más el tiempo transcurrido en la llamada al método continuar y la segunda llamada al tiempo parar.

A continuación se brinda la clase Cronometro para dar una posible forma de implementación, la misma puede ser modificada y corregida según el criterio del grupo.

```
/*****************************************************************************
* Algoritmos y Programación II - 75.41 *
* Cátedra Ing. Patricia Calvo *
* Facultad de Ingeniería - Universidad de Buenos Aires *
*****************************************************************************/
/* TDA Cronometro
* Archivo : Cronometro.h
* Versión : 1.0
*/
#ifndef __CRONOMETRO_H__
#define __CRONOMETRO_H__
#include <time>
#include <sstream>
/*****************************************************************************/
/* Definiciones de Tipos de Datos */
/*--------------------------------*/
/* Definición del TDA Cronometro */
class Cronometro{
      private:
/*****************************************************************************/
/* Definicion de Atributos */
/*-------------------------*/
/* El atributo inicio guarda el tiempo en que se inicia el cronometro*/
            clock_t inicio;
/* El atributo cronometro guarda la suma de los intervalos de tiempo entre que se
inicia el cronometro y se pausa, o se inicia y se detiene*/
            long contador;
/* El atributo pausado guarda el estado actual del cronometro, si esta pausado
o no*/
            bool pausado;
/*****************************************************************************/
/* Definicion de Primitivas */
/*--------------------------*/
/*
pre : ninguna.
post: Crea un Cronometro inicializado en cero.
*/
public:
      Cronometro(){
            iniciar();
      }
/*
pre : el cronometro debe haber sido creado con el constructor.
post: Borra todos los tiempos acumulados y estable el instante en que se
comienza a contar el tiempo.
*/
      void iniciar(){
            contador = 0;
            inicio = clock();
            this->pausado = false;
      }
/*
pre : el cronometro debe haber sido creado con el constructor.
post: Acumula el tiempo transcurrido desde la creacion, inicio o continuar
(lo ultimo que haya pasado).
*/
      void pausar(){
            clock_t fin;
            fin = clock();
            contador += fin - inicio;
            this->pausado = true;
      }
/*
pre : el cronometro debe haber sido creado con el constructor.
post: Estable el instante a partir del cual se cuenta el tiempo.
```

```
*/
	void continuar(){
		if (this->pausado){
			inicio = clock();
			this->pausado = false;
		}
	}
/*
pre : El cronometro debe haber sido creado con el constructor.
post: Finaliza la cuenta y acumula todos los tiempos transcurridos.
*/
	void parar(){
		if (!this->pausado){
			this->pausar();
			this->pausado = true;
		}
	}
/*
pre : El cronometro debe haber sido creado con el constructor.
post: Devuelve una leyenda con los milisegundos que estan acumulados.
*/
	std::string toString(){
		std::stringstream convertidor;
		convertidor << this->contador;
		return "Transcurrieron " + convertidor.str() + " milisegundos";
}
/*
pre : El cronometro debe haber sido creado con el constructor.
post: Devuelve la cantidad de milisegundos que estan acumulados.
*/
	long getTiempoTranscurrido(){
		return this->contador;
	}
};
#endif /* __CRONOMETRO_H__ */
```

4 - Normas de Entrega

4.1 - Carpeta

La carpeta deberá constar **únicamente** de las carátulas brindadas por la cátedra. Sólo las carátulas deben ser impresas, el resto de los elementos de cada entrega deberán proporcionarse en formato digital.

Las carátulas de las entregas irán permaneciendo en la carpeta, y en la entrega final se firmará la aprobación del TP en la última de ellas, la cual es el comprobante para poder firmar la cursada de la materia.

Tanto el código fuente, la documentación y el informe deberán entregarse en formato digital (**NO impreso**).

4.2 - Informe

1) *Carátula*: carátula provista por la cátedra.
2) *Índice*
3) *Enunciado*: enunciado completo del trabajo práctico.
4) *Diseño*
   a) *Estructura*: estructura general de la solución (*). Estrategias adoptadas. Enumeración de los TDAs identificados. Interacción entre los TDAs. Esquemas y diagramas que documenten el diseño de la solución propuesta.
   b) *TDAs*: Definición de todos los TDAs desarrollados por el grupo.
      i) Tipo
      ii) Descripción
      iii) Axiomas
      iv) Primitivas
      v) Precondiciones
      vi) Postcondiciones
      vii) Consideraciones de implementación (*)
   c) *Estructuras dinámicas* (*): Análisis de las estructuras dinámicas utilizadas para el desarrollo del trabajo práctico.
      i) Estructuras dinámicas utilizadas
      ii) Descripción del problema que resuelven
      iii) Consideraciones de implementación (*)

d) *Consideraciones adicionales* (*)

5) *Manual de usuario* (*): Pasos a seguir para utilizar la aplicación. Debe ser simple y completo.

6) *Código Fuente*: código fuente completo, ordenado convenientemente de acuerdo a algún criterio elegido por el grupo. Incluir al comienzo de esta sección una lista a modo de índice con el nombre y objetivo de cada archivo.
   a) Entorno de desarrollo: nombre del compilador y del IDE utilizado.
   b) Índice de archivos
   c) Fuentes

7) Pruebas

   a) Prueba unitaria para cada TDA: código de un programa de prueba **simple y concreto**, en el cual se muestre el uso del TDA y su correcto funcionamiento.
   b) Pruebas integrales de la aplicación (*): corrida válida de la aplicación completa, especificando los datos de entrada utilizados y los datos de salida obtenidos.

(*) Sólo en los casos en que corresponda.

4.3 - Formato

La entrega se realizará en formato digital exclusivamente.

Cada uno de los grupos deberá entregar un archivo comprimido (***.zip***) que contenga el conjunto de archivos que componen la entrega formal del Trabajo Práctico. Dentro de este archivo comprimido el material deberá estructurarse de acuerdo al siguiente esquema de directorios:

1) /src
   fuentes completos
   instrucciones necesarias para compilar la aplicación (leeme.txt)
2) /datos
   archivos de entrada con datos de prueba
3) /doc
   informe y documentación adicional

La ausencia de cualquiera de estos elementos implicará la desaprobación del Trabajo Práctico.

4.4 - Entrega

Todas las entregas se deberán mandar por mail a

To: algo2-fiuba-owner@gruposyahoo.com.ar

Subject: TP N Grupo X

Attachment: TPN-GrupoX.zip

Siendo N el número de entrega y X el número de grupo.

Las fechas límite para cada entrega son inamovibles. El incumplimiento de las mismas implicará la desaprobación del Trabajo Práctico.

Dentro de las 12 horas de recibido el mensaje el ayudante asignado enviará la confirmación. Cada grupo debe reenviar el mail hasta recibir la confirmación.

5 - Normas de Evaluación

5.1 - Diseño

Se tendrá en cuenta el diseño de la solución, las estructuras utilizadas, optimización de las mismas, ideas propias del grupo, ideas novedosas, simplificaciones cuando corresponda, descripción del diseño y su justificación. Elegancia de la entrega.

5.2 - Implementación

Se tendrá en cuenta la claridad de los códigos, los comentarios, uso de las estructuras vistas en clase, manejo de punteros, uso de los TDAS dados y creación de nuevos TDA a conveniencia de la solución encontrada al problema.

5.3 – Entrega

Cumplimiento de la entrega en tiempo y forma.

5.4 - Corrección

El ayudante asignado al grupo corregirá en conjunto el informe, la documentación y la implementación del Trabajo Práctico. Todos estos componentes son requeridos para todas las entregas. Las normas de entregas provistas por la cátedra deben ser cumplidas a los efectos de alcanzar la etapa de corrección.

En caso de que el ayudante lo considere, se podrá tomar una evaluación oral sobre el o los Trabajos Prácticos, para analizar la participación de cada uno de los integrantes.

6 - Carátula

Las carátulas debe ser entregadas como primer hoja en los Informes del Trabajo Práctico.

Universidad de Buenos Aires – Facultad de Ingeniería

75.41 Algoritmos y Programación II

Cátedra Ing. Patricia Calvo

TP1

Cuatrimestre	

Ayudante	

Grupo	

Apellido, Nombre	**Padrón**	**e-mail**

Entrega			
Fecha		**Nota**	
Observaciones			
	Firma		

Universidad de Buenos Aires – Facultad de Ingeniería

75.41 Algoritmos y Programación II

Cátedra Ing. Patricia Calvo

TP2

Cuatrimestre	

Ayudante	

Grupo	

Apellido, Nombre	**Padrón**	**e-mail**

1º Entrega			
Fecha		**Nota**	
Observaciones			
	Firma		

2º Entrega			
Fecha		**Nota**	
Observaciones			
	Firma		

	Universidad de Buenos Aires – Facultad de Ingeniería **75.41 Algoritmos y Programación II** *Cátedra Ing. Patricia Calvo*	**TP3**

Cuatrimestre	

Ayudante	

Grupo	

Apellido, Nombre	**Padrón**	**e-mail**

1º Entrega			
Fecha		**Nota**	
Observaciones			
	Firma		

2º Entrega			
Fecha		**Nota**	
Observaciones			
	Firma		