Universidad Nacional Autónoma de México
Ingeniería en Computación
Computadoras y programación

Asignación de Software.	**CYP-PRY01**
Responsable:	**M en C Ernesto Peñaloza Romero**

CYP-PRY01

Objetivo: El alumno aplicará los conocimientos adquiridos en el curso para desarrollar un algortimo integral.
Descripción: Un programa que cree un diccionario de palabras y sobre el mismo desarrolle un algortimo heurístico "ud quiso decir".

Objetivos

Al concluir satisfactoriamente esta asignacion, el alumno:

Programará usando:
a) Archivos
b) Arreglos de caracteres
c) Arreglo numericos
d) Decisiones
e) Repeticiones
f) Ordenamientos
g) Búsquedas.

¿Qué hay que hacer?

Hoy en dia, los usuarios esperan que los sistemas "piensen" por ellos. Esto incluye habilidades como la capacidad para predecir lo que un usuario quiere, aún antes de que él mismo lo pida. Y uno de estos algortimos comunes es el que popularizó Google: "Usted quiso decir"

Estos algortimos estan basados en la Distancia de Edición, como se conoce en las ciencias de la computación. El problema consiste en establecer el número de transformaciones que se requiere para convertir una cadena en otra. La idea es que por medio de un conjunto básico de operaciones, podemos transformar una cadena en otra correctamente escrita.

En general se ha podido observar que cuando se escribe mal una palabra, sólo se requiere una transformación para obtener la palabra correcta, es decir, generalmente los errores consisten en la escritura erronea de un solo carácter. Pensemos en *sapato, cavallo, espelusnante, hernesto, peñalosa, inibido*. Todas esas palabras tienen un carácter erroneo como *espeluznante, inhibido o ernesto*, las cuales tienen: un carácter que debe ser sustituido, un carácter faltante o un carácter sobrante.

Del párrafo anterior se obtienen entonces algunas de las operaciones que se pueden realizar para corregir una palabra mal escrita. Estas operaciones fueron propuestas por Vladimir Levenshtein en 1965 y al concepto se le llama Distancia de Levenshtein, posteriormente en un trabajo conjunto con Frederick J. Damerau implementaron la Distancia Damerau-Levenshtein, la cual añade a las operaciones anteriores la trasposición de dos caracteres como en *copoiso* por *copioso.*

¿Cómo funciona entonces el algoritmo predictivo? Para explicarlo observemos un ejemplo

Supongamos que queremos escribir ***Ana*** y en su lugar se escribe la palabra *ans* de entrada. También supongamos un alfabeto constituido por el siguiente conjunto

Letras = { 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,á,é,í,ó,ú}

Entonces la palabra ***ans*** entra a la lista de sugerencias candidatas del algortimo, una vez realizado esto, comenzamos a manipular los caracteres.

La primera operación, es la eliminación de un carácter en la cadena original. Entonces se elimina **a** y queda **ns**, palabra que se añade a la lista de candidatas. Una vez mas, de la cadena original se sustituye el segundo carácter y luego el tercero. La lista de cadenas con las eliminaciones resultantes quedán asi:

ns (eliminando a)
as (eliminando n)
an (eliminando s)

La siguiente operación es la trasposicion de caracteres. Es un intercambio de los caracteres de la cadena original. Cada par de caracteres contiguos debe intercambiarse entre si.

nas (intercambiando n por a)
a**sn** (intercambiando s por n)

A continuación, se sustituye cada carácter de la cadena original por un carácter del alfabeto válido. Se indican a continuacion las primeras tres sustituciones para cada letra de la cadena original.

ans(a->a)	t**n**s	a**i**s	a**é**s	an**p**
bns(b->a)	u**n**s	a**j**s	a**í**s	an**q**
cns(c->a)	v**n**s	a**k**s	a**ó**s	an**r**
dns	w**n**s	a**l**s	a**ú**s	an**s**
ens	x**n**s	a**m**s	ana(a->s)	an**t**
fns	y**n**s	a**n**s	anb(b->s)	an**u**
gns	z**n**s	a**ñ**s	anc(c->s)	an**v**
hns	**á**ns	a**o**s	an**d**	an**w**
ins	**é**ns	a**p**s	an**e**	an**x**
jns	**í**ns	a**q**s	an**f**	an**y**
kns	**ó**ns	a**r**s	an**g**	an**z**
lns	**ú**ns	a**s**s	an**h**	an**á**
mns	a**a**s(a->n)	a**t**s	an**i**	an**é**
nns	a**b**s(b->n)	a**u**s	an**j**	an**í**
ñns	a**c**s(b->n)	a**v**s	an**k**	an**ó**
ons	a**d**s	a**w**s	an**l**	an**ú**
pns	a**e**s	a**x**s	an**m**	
qns	a**f**s	a**y**s	an**n**	
rns	a**g**s	a**z**s	an**ñ**	
sns	a**h**s	a**á**s	an**o**	

Finalmente, se inserta el abecedario completo en cada uno de los espacios. Antes de la **a**, entre la **a** y la **n**,

entre la **ns** y despues de la **s**.
Se muestran en colores las series de inserciones entre cada espacio

aans	vans	amns	ancs	anxs	ansñ
bans	wans	anns	ands	anys	anso
cans	xans	añns	anes	anzs	ansp
dans	yans	aons	anfs	anás	ansq
eans	zans	apns	angs	anés	ansr
fans	áans	aqns	anhs	anís	anss
gans	éans	arns	anis	anós	anst
hans	íans	asns	anjs	anús	ansu
ians	óans	atns	anks	ansa	ansv
jans	úans	auns	anls	ansb	answ
kans	aans	avns	anms	ansc	ansx
lans	abns	awns	anns	ansd	ansy
mans	acns	axns	anñs	anse	ansz
nans	adns	ayns	anos	ansf	ansá
ñans	aens	azns	anps	ansg	ansé
oans	afns	aáns	anqs	ansh	ansí
pans	agns	aéns	anrs	ansi	ansó
qans	ahns	aíns	anss	ansj	ansú
rans	ains	aóns	ants	ansk	
sans	ajns	aúns	anus	ansl	
tans	akns	anas	anvs	ansm	
uans	alns	anbs	anws	ansn	

Muchas de las palabras no tienen significado, por lo tanto se desecharan en el siguiente paso. Pero primero debemos explicar como se detectan las palabras válidas.

Cuando uno abre Google, captura un par de palabras que constituyen las claves para la búsqueda. Google almacena un conjunto de estadísticos sobre la frecuencia de las palabras. Como nosotros no dispondremos de las tablas de frecuencias haremos uso de un archivo muy grande, con el vocabulario que deseamos procesar. Imagimemos un archivo con el siguiente contenido:

"anita lava la tina. Esta es una frase en la que Ana, palabra que por si misma constituye un palindromo (ana, leido al reves tambien dice ana), se transforma en un frase que es un palindromo. Anis no es palindromo."

Al realizar un análisis de este texto encontramos las siguientes frecuencias de las palabras involucradas

al	1	**frase**	2	**que**	3
ana	3	**la**	2	**reves**	1
anis	1	**lava**	1	**se**	1
anita	1	**leido**	1	**si**	1
constituye	1	**misma**	1	**tambien**	1
dice	1	**no**	1	**tina**	1
en	2	**palabra**	1	**transforma**	1
es	3	**palindromo**	3	**un**	3
esta	1	**por**	1	**una**	1

De entre las palabras de este diccionario se buscan las palabras que fueron generadas en la clonacion de palabras. Tras realizar una inspección se observa que solo hay dos en el diccionario (Ana y anis). Ana tiene una frecuencia de 3 y anis tiene una frecuencia de 1. Considerando estos pesos, Ana es la palabra más probable puesto que tiene mayor frecuencia.

Es evidente que la frecuencia requiere de diccionarios grandes, a fin de tener un amayor certidumbre en los dominios.

Especificaciones Técnicas:

El programa se entregará en un archivo: Corrector.cpp. El archivo de cabecera que solo poseerá las funciones prototipo se proporcionará, mientras que el de implementación(.cpp) poseerá el código de las funciones.

El programa se dividirá en las acciones elementales del algortimo

CargaDiccionario: Recibirá el nombre de un archivo y cargará palabra por palabra todas las que encuentre en un par de arreglos, uno conteniendo las palabras, y otro su frecuencia. El resultado deberá venir almacenado en ordenado alfabetico, sin manipulaciones extras a las que pueda proporcionar strmp, es decir, sin considerar las letras acentuadas en su lugar, sino la ordenacion ASCII dada por strcmp.

ClonaPalabras: Toma la palabra a corregir y realiza todas las operaciones (supresion, sustitucion, trasposicion e insercion) a fin de obtener toda la lista de las palabras candidatas. La lista de candidatas deberá ordenarse por orden alfabetico, con las mismas consideraciones que las del diccionario.

ListaCandidatas: Recibe las palabras candidatas, el diccionario y la frecuencia y regresa como resultado un ìndice a las palabras que son candidatas, ordenadas según su peso.

El archivo corrector.h poseerá las siguientes funciones prototipo:

```
void    Diccionario(
        char *  szNombre,                       //Nombre del archivo, desde donde se lee el diccionario
        char    szPalabras[][TAMTOKEN],         //Lista de palabras del diccionario
        int     iEstadisticas[],                //Numero de veces que aparece la palabra en el archivo
        int &   iNumElementos);                 //Numero de elementos


void    ClonaPalabras(
        char *szPalabraLeida,                   // Palabra a clonar
        char szPalabrasSugeridas[][TAMTOKEN],   //Lista final de palabras clonadas
        int & iNumSugeridas);                   //Numero de elementos en la lista

void    ListaCandidatas(
        char szPalabrasSugeridas[][TAMTOKEN],   //Lista de palabras clonadas
        int iNumSugeridas,                      //Lista de palabras clonadas
        char szPalabras[][TAMTOKEN],            //Lista de palabras del diccionario
        int iEstadisticas[],                    //Lista de las frecuencias de las palabras
        int iNumElementos,                      //Numero de elementos en el diccionario
        char szListaFinal[][TAMTOKEN],          //Lista final de palabras a sugerir
        int  iPeso[],                           //Peso de las palabras en la lista final
        int iNumLista);                         //Numero de elementos en la szListaFinal
```

La macro TAMTOKEN debera estar definido en el archivo corrector.h como

```
#define TAMTOKEN 50
```

Consideraciones de rendimiento

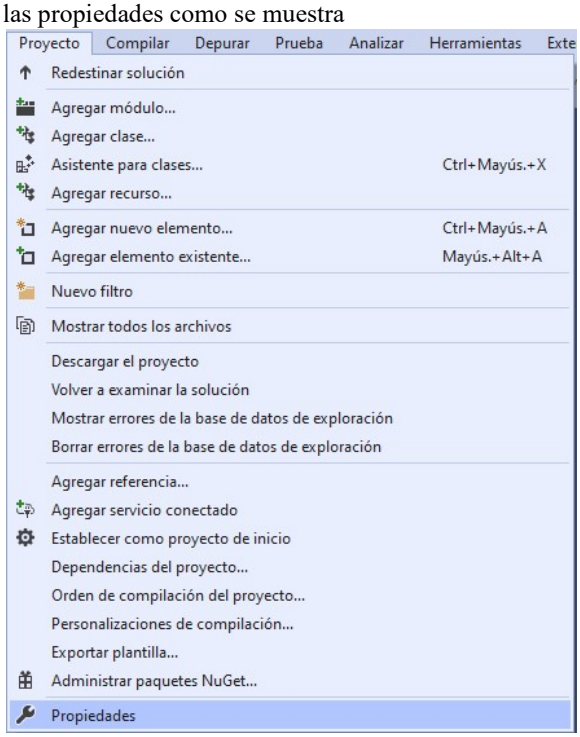
Debe considerar que el archivo de palabras puede contener al menos 50 000 palabras diferentes. Tome eso en cuenta al realizar su programación pues necesitara hacer eficientes las búsquedas y las clonaciones o el evaluador marcará tiempo de ejecucion excesivo y abortará la ejecucion.

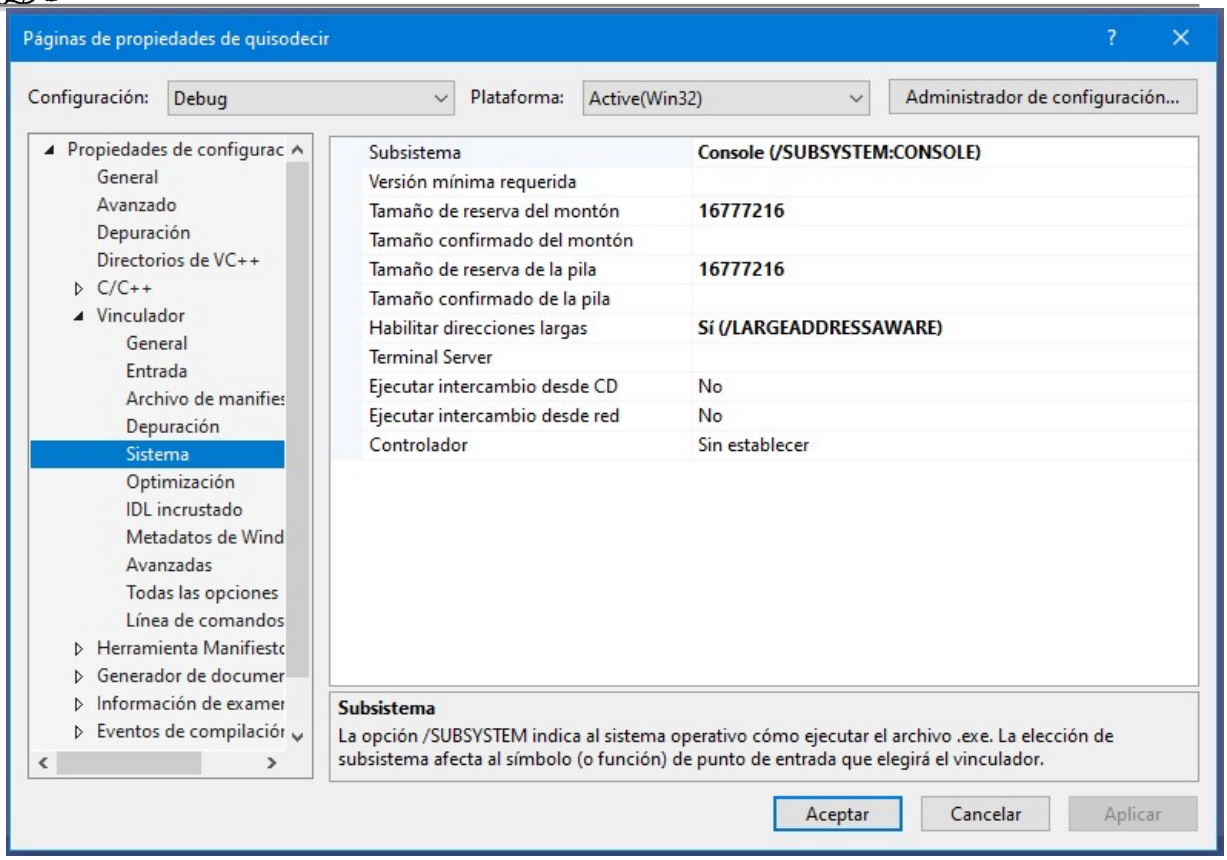
La carga del diccionario del programa de referencia en una maquina Core Duo a 2.92Ghz con 2 Gb de RAM es de 40 segundos. La clonacion y generacion de una lista es de menos de 1 segundo. En esta consideracion, asuma un tiempo máximo de 45 segundos para la carga.

Consideraciones de memoria

Un programa como el que se esta solicitando requiere que el compilador solicite recursos muy diferentes a los de un proyecto normal. Por esa razon se debe realizar la compilacion considerando que se debe afectar los valores por defecto del compilador, indicando explicitamente que se utilizará una gran cantidad de memoria.

Abra su proyecto y afecte las propiedades como se muestra





Si aun fuese necesario, puede multiplicar esas cifras por dos, pero debe revisar su instrumentacion pues es posible que este usando memoria extra de manera innecesaria.

Se proporciona un archivo main y un corrector.h con el cual usted puede probar que su instrumentación en corrector.cpp cumple con las firmas solicitadas.

Del mismo modo, se proporciona un proyecto creado con las MFC (Microsoft Fundation Class) en el cual usted puede sustituir su código y verlo funcionar en una aplicación gráfica. Para verlo funcionar abra el directorio de las fuentes y sustituya el archivo corrector.cpp

Otro aspecto fundamental para considerar es que el binario generado para la evaluación no se compilará en modo de depuración (debug), sino en modo de liberación (release). El código estará por tanto ejecutándose sin más validaciones que las que establezca en el mismo. Los errores en tiempo de ejecución no contendrán información específica sobre la condición de error y por tanto no se informará al respecto. Del mismo modo, cualquier tratamiento que el depurador realice y que permita la ejecución en pruebas (como valores iniciales establecidos por el depurador), no existirán en el modo de liberación. Asegúrese de que su código ejecuta en esas condiciones.

Especificaciones de entrega

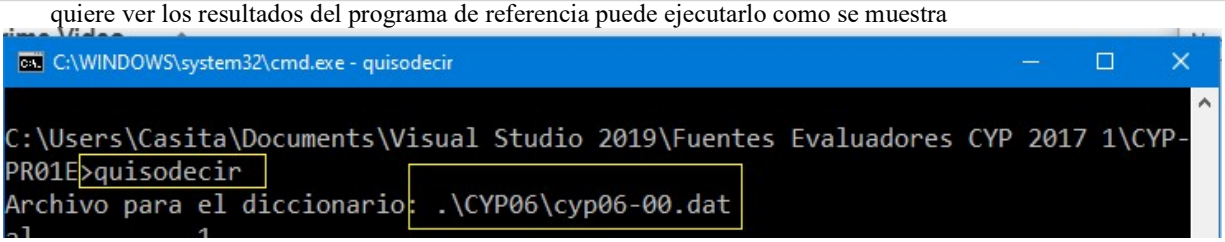
- Solo debe enviar un archivo fuente llamado corrector.cpp. Este se empaquetará en un archivo zip y se enviará para su evaluación a la cuenta de correo indicada en la plataforma.
- El asunto del correo debe ser FESA
- El archivo zip deberá llamarse: CYP06-NUMERODECUENTA-NOMBRESINESPACIOS.zip
- En un archivo texto con el nombre CONCLUSION.TXT escribirá las conclusiones de su asignación. Las conclusiones deberán contener al menos 101 caracteres.
- Su asignación será evaluada automáticamente.
- Dependiendo de la carga en los envíos, la evaluación puede tardar varios minutos en ser calculada. Esta carga se incrementa durante las ultimas 48 horas por lo que la espera puede ser de horas. Considere que esta evaluación tarda casi 5 minutos por alumno en procesarse, si el último día envían todos su asignación, el tiempo para procesar todo el grupo puede ser de 6 horas para un solo envío. Tome sus previsiones

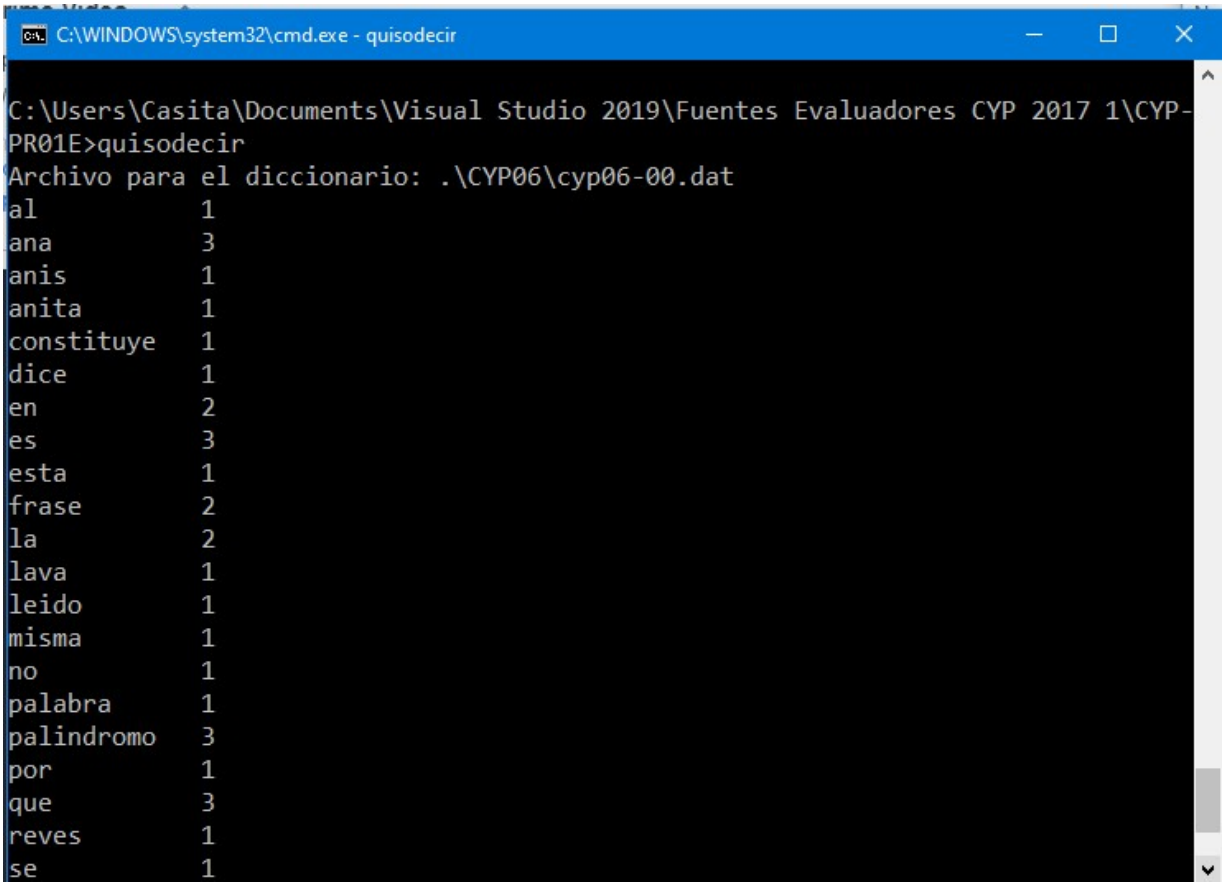
FAQ

- ¿Es un proyecto individual? Si. Se revisará la originalidad del código de cada solución, tanto por una herramienta, como de manera visual.
- ¿Que clase de pruebas tendrá mi programa? El programa será sometido a pruebas en cada una de las funciones
  - La funcion `Diccionario` recibirá distintos archivos, incluyendo un archivo vacio, La biblia y Don Quijote, tras lo cual se revisarà el diccionario generado, el número de elementos y el número de apariciones de cada palabra. Su evaluacion global valdra el 30% de la calificación.
  - La función `ClonaPalabras` recibirá distintas palabras y se compararán las palabras generadas contra la referencia, tanto en tipo como en número. Su evaluacion global valdra el 30% de la calificación.
  - La función `ListaCandidatas` recibirá distintos diccionarios y palabras clonadas y se comparará la lista de candidatas, en número o orden. Su evaluacion global valdra el 30% de la calificación.

- ¿Cómo determino que es una palabra? Considere que cualquier secuencia de caracteres separada por espacios, tabuladores, saltos de línea, coma, punto y coma, paréntesis y retorno de carro es una palabra. Esto hace que ***arr[10]*** sea considerada una palabra. Inclusive ***y=10*z+20*** sea tambien considerada una palabra si esta se encuentra como se muestra: sin espacios en blanco u otros signos de puntacion ya indicados. ***(anita, come).*** se divide en solo dos palabras: ***anita*** y ***come.***
- ¿Cuál es el alfabeto por utilizar? 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, á, é, í, ó, ú.
- ¿Debo realizar una función especial que ordene los acentos en las posicion de las vocales sin acentuar? No. Deje que strcmp ordene conforme el código de la máquina en donde se compile sin hacer ningun tratamiento especial.
- ¿Cómo debo ordenar caracteres especiales no alfabeticos? Deje que strcmp los gestione. No realice tratamiento alguno.
- ¿Las palabras a predecir serán siempre palabras con caracteres alfanumericos? Si, se garantiza que las palabras serán alfabéticas y que existiran en el diccionario.
- ¿Puedo mandar más de un archivo? No, solo debe mandar corrector.cpp
- ¿Qué debo hacer si necesito funciones que no están en el corrector.h? incluyalas en corrector.cpp de manera que no necesite declarar las funciones prototipo.
- ¿Que sucederá si necesito más memoria y debo modificar los valores sugeridos de memoria en las propiedades del proyecto? El proyecto se garantiza que será compilado con esos valores, pero no se proporcionará memoria extra a lo especificado (16Mb).
- Al igual que en las ocasiones anteriores, debe descomprimirse el paquete con los archivos comprimidos en donde se encontrará a) el programa de referencia quisodecir.exe y b) el conjunto de archivos de prueba. Si

quiere ver los resultados del programa de referencia puede ejecutarlo como se muestra





El programa referencia es el quisodecir. En cuanto inicie, pide el nombre del archivo en el cual se basará el diccionario. Cuando termine de procesar mostrará la lista completa de palabras detectadas y su frecuencia. Después de eso le preguntará por una palabra a corregir. Cuando desee salir del programa digite la palabra **fin**

- En el directorio diccionario del empacado se proporcionan los diccionarios generados por los archivos CYP*.dat que se usan en la evaluación con el fin de que puedas comparar la referencia contra tus resultados. Recuerda que si deseas grabar la salida de tu programa para poder compararlo usa

C:\MiDirectorioDelProyecto\quisodecir > miArhivoDeSalida.txt

Donde **MiDirectorioDelProyecto** debe ser la ruta en donde este probando y **miArhivoDeSalida.txt** es el

archivo en donde se escribirá la salida.

- ¿Cuál es la escala?: estilo 20%, correccion 70%, conclusiones 10%

- ¿Cómo se resulve este problema? La estrategia mas importante de desarrollo consiste en dividir el problema es pequeñas secciones que sean resolubles de manera unitaria. Esto significa dividir el problema en problemas más pequeños que sean fácilmente resolubles.

  Lo siguiente solo es una guía, es decir, una sugerencia de lo que hay que hacer y de ninguna manera es obligatorio seguirla.

  El problema indica que debemos dividir el problema en los siguientes subproblemas
  - Cargar el diccionario
  - Generar las palabras
  - Buscar las palabras en el diccionario

  La carga del diccionario es el proceso por medio del cual se lee un archivo con distintas palabras separadas por los caracteres ya indicados.

  Cargar diccionario

  Este proceso a su vez se divide en dos procesos separados

  - Leer archivo
  - Ordenar palabras

  La lectura del archivo la podemos estructurar de la siguiente manera

  Leer archivo
  Abrir el archivo
  Hasta que no se encuentre el fin de archivo
  Leer una palabra hasta el proximo espacio, tabulador, salto de linea o salto de carro.
  De la cadena leída, eliminar signos de puntuación
  Convertir la cadena en letras minúsculas
  Buscar la cadena entre las cadenas del diccionario
  Si la cadena no existe en el diccionario
  Añadirla al diccionario
  Sino
  Incrementar la frecuencia de esa palabra
  Fin si
  Fin Hasta

  El proceso de la generacion de palabras puede ser expresado con los siguientes pasos

  Generar palabras

  Añadir la palabra a la lista generada
  Para cada letra en la palabra
  Generar la palabra sin el carácter i-ésimo
  Agregar la palabra generada a las candidatas
  Fin para

Para cada par de letras en la palabra
    Generar la palabra intercambaindo el caracter i-ésimo con el i-ésimo+1
    Agregar la palabra generada a las candidatas

Fin para

Para cada una de las letras de la palabra
    Para cada una de las letras del alfabeto
        Cambia la letra i-ésima de las palabras por la letra del alfabeto
        Agregar la palabra generada a las candidatas

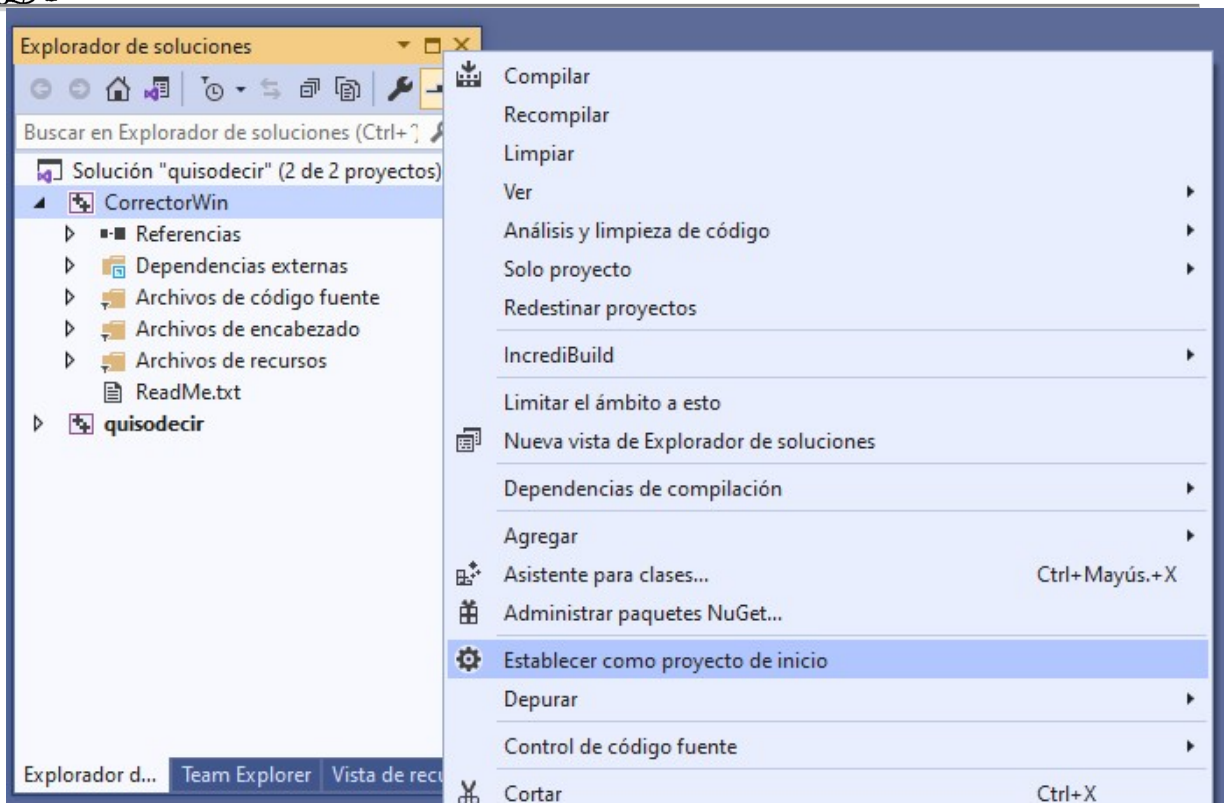
    Fin para
Fin para
Inserta el alfabeto al final de la palabra original

Yo sugiero, como siempre ir realizando una funcion a la vez, una línea de pseudodigo a la vez. Si empiezan de inmediato, muy pronto descrubrirán si su estrategia es correcta o tal vez descubran una menor estrategia de programación. Si incian lo mas pronto posible, tendrán tiempo más que suficiente para corregir o reescribir el código que sea necesario

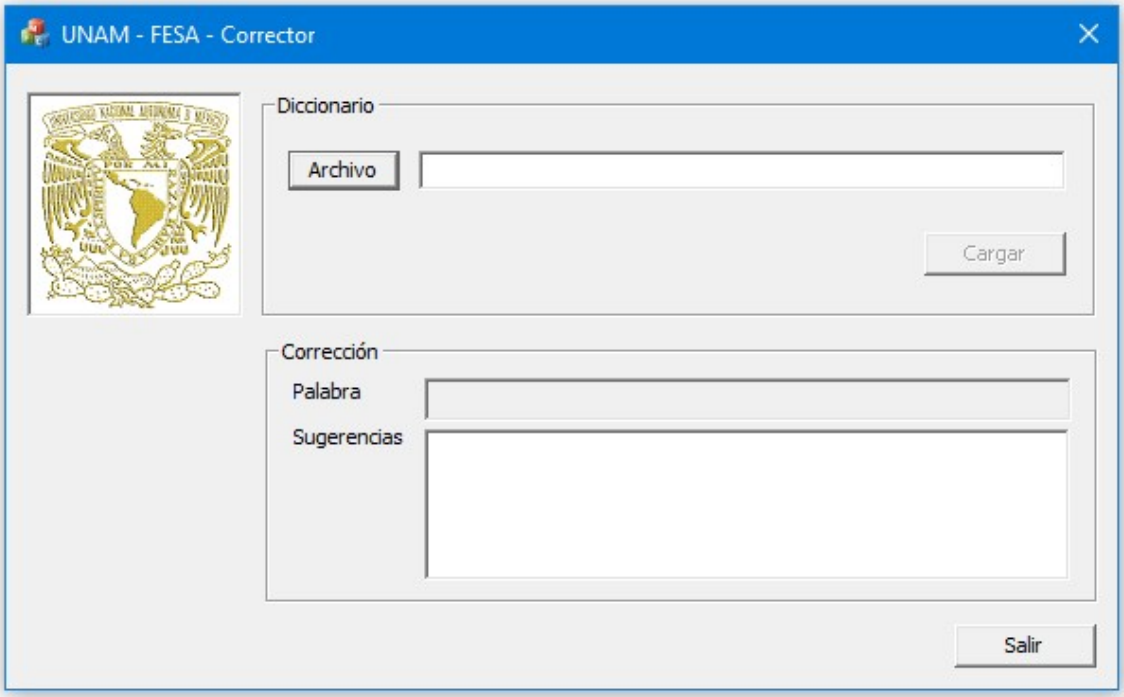
Probar el proyecto

Se provee de un proyecto en donde pueden colocar su código fuente. Este proyecto muestra el programa principal en modo texto y ya tiene un cascaron en donde han de poner sus funciones.

Del mismo modo se provee de una interfaz gráfica en donde pueden probar la eficiencia de su programación. No necesitan capturar nada. En cuanto quede su versión en modo texto simplemente se cambia el proyecto de inicio y en ese momento compilará la versión gráfica.



La ventana que genera es la siguiente



Lo primero que pide la ventana es el archivo desde donde se generará el diccionario. Tal operación puede tardar. En cuanto quede cargado se habilitará el campo para capturar una palabra y de inmediato aparecen los resultados en la lista de la parte inferior (Sugerencias).

El programa entrega los resultados en el orden especificado por el arreglo `szListaFinal.`

Del mismo modo se entrega un ejecutable completamente funcional en modo gráfico y en modo texto. El programa en modo texto entrega los resultados intermedios, los cuales se pueden capturar en un archivo usando la indireccion en la linea de comandos del sistema operativo

Quisodecir > salida.txt

Toda la salida se redirigirá al archivo salida.txt, el cual se puede abrir con Word.

Sugerencias

- NUNCA PIERDAS DE VISTA QUE ES LO QUE QUIERES LOGRAR CON TU PROYECTO. No se trata de mostrar que sabes programar mucho, ni que has investigado más que nadie. Se trata de aplicar los conocimientos de la materia. No pierdas tiempo valioso tratando de solucionar problemas que no se van a presentar (como tratar de detectar si el archivo es UNICODE o UTF-8, o peleando con una determinada característica del lenguaje, si existe una forma mas sencilla de hacerlo: ¡hazlo!
- Anota en una libreta todas las ideas que no funcionaron y porqué. Esto te dará experiencia en la elaboración de proyectos más grandes y te ayudará a crear tu propia filosofía de desarrollo de sistemas. Lleva una bitácora de errores. Esto te ayudara a evitarlos en el futuro.

- Crea las funciones que necesites, solo no las expongas o metas en otros archivos.
- Envía todos los datos por parámetros a las funciones.
- No uses variables globales.
- Cada vez que una función trabaje, respalda el proyecto en otro directorio o en USB. Si te equivocas, siempre podrás regresarte a la versión anterior. ***Recuerda que los discos duros también fallan.***
- No te preocupes demasiado por la presentación. Cuando el proyecto ya funcione, tendrás tiempo para preocuparte por ella.
- Llena de prints tu codigo si el programa no hace lo que quieres. Sacale la verdad sobre lo que realmente esta haciendo. Usa las directivas del procesador para controlar cuando se encienden o apagan. Y no olvides apagarlas antes de entregar al evaluador
- Desde el principio diseña un pseudocódigo general, de muy alto nivel, de la aplicación. También haz un plan de la arquitectura (la forma como se conectan los datos, se comunican las funciones, la estructura del archivo, etc) Esto te dará un panorama general de lo que debes contemplar para llevar a buen éxito el proyecto.
- Sobre la base de tu diseño y de las especificaciones, elabora un plan de trabajo REALISTA, que contemple tus otras materias. Cuida de especificar que funciones serán codificadas, en que fechas.
- Cumple tu plan de trabajo. Si te permites atrasarte esto va en perjuicio de tu calificación. Recuerda que el proyecto tiene un porcentaje sobre la calificación final. Destina un horario que puedas cumplir para realizarlo. Sé constante
- No te confíes. La complejidad del proyecto suele estar regida por el número de módulos de los que consta el proyecto. Si se tienen pocos, la complejidad es poca, pero si se tienen muchos, la complejidad crece a un ritmo de

  $$\text{Complejidad} = (\text{numero de módulos})^{(\text{número de módulos con los que interactua})} * \text{número de líneas en módulo.}$$
- Se estima que el proyecto requiere 60 horas de un programador que genera 60 líneas de código bien depuradas al día. A eso hay que sumarle las horas necesarias para el análisis (Otras 45).
- La experiencia muestra que se requieren tres semanas de trabajo constante para obtener resultados **aceptables**.
- No inicien a escribir una sola línea de código si no tienen idea de cómo funcionará el proyecto. Reúnanse en equipos para discutir posibles formas de hacerlo trabajar. La arquitectura no es el código, así que, aunque varios equipos tengan la misma idea, seguramente lo instrumentarán diferente. Pero el proyecto es individual y se usaran algoritmos de Google para detectar plagios

Debes recordar que esta clase de proyectos tienen por finalidad aplicar los conocimientos que has adquirido hasta el día de hoy. No se trata de hacer programas perfectos. Con estos proyectos adquieres la experiencia necesaria para realizar proyectos. Pero solo la adquirirás si y solo si, te das la oportunidad de adquirirla. Programar una aplicación real es enfrentar la teoría aprendida, contra problemas específicos. Nos enfrenta a problemas que nunca habíamos imaginado, y nos obliga a **crear** soluciones, buscar en libros y darnos cuenta de nuestras deficiencias. Algunos problemas no serán correctamente resueltos solo con los conocimientos de esta materia, pero si lo meditamos profundamente, en materias sucesivas encontraremos soluciones más eficientes. Un proyecto que funciona es la prueba fehaciente, palpable de que estas aprendiendo cosas útiles y reales. Te dará seguridad y aguijoneará tu imaginación. También servirá para que en otras materias tengas antecedentes sólidos de lo que te quieren decir, que lo motiva y cual es la solución más correcta. También te mostrará porque los productos comerciales son así y no de otra manera Pero al final no olvides lo que es importante: Proyectos como este, te prepararán para ser un ingeniero útil a la sociedad que hoy tanto lo necesita y por supuesto, serás un ingeniero con un trabajo en el cual te puedas sentir realizado.