1º Cuatrimestre 2013 Ejercicio Nº 3 - Word Mangling	Taller de Programación I (75.42) Facultad de Ingeniería Universidad de Buenos Aires

Ejercicio Nº 3 - Word Mangling

Introducción

El word mangling (o alteración de palabras) es una serie de reglas que definen como transformar una palabra en otra a partir de simples modificaciones.

Un ejemplo clásico se encuentra en los generadores de palabras similares a los diccionarios.
Por ejemplo, para generar todas las conjugaciones de verbos se puede partir de un verbo en infinitivo y luego cambiar su raíz (ar, er, ir) por distintas terminaciones, propias de cada tiempo verbal.
Asi, del verbo amar, se obtiene am.ar quien puede originar:

- am.o <-- cambiar **ar** por **o**
- am.e <-- cambiar **ar** por **e**
- am.ará <-- cambiar **ar** por **ará**

Claro está que la serie de reglas esta íntimamente relacionada con las reglas gramaticales del lenguaje y en particular con el tipo de palabra que se usa como punto de partida.
Asi, verbos irregulares no pueden ser procesados de esta manera tan trivial.

En nuestro caso se implementará un módulo de word mangling para ser integrado a un password cracking [1].

Motivación

Cuando se desea administrar cuentas de usuario y contraseñas, no es recomendable almacenar las contraseñas como simples strings pues cualquiera con acceso a la base de datos podría robar las contraseñas de los usuarios.
Así, estas contraseñas son transformadas a strings alfanuméricos conocidos como hashes. Es claro que esta operación **no** debe ser reversible, pues si asi lo fuera, cualquier persona que logre acceder a la base de datos podría robar los hashes y por ser una operaciòn reversible, obtener trivialmente las contraseñas.
Es por eso que las funciones de hash usadas en el almacenado de contraseñas no son reversibles [2].

Si alguien pudiera acceder al listado de hash, podría descubrir las contraseñas almacenadas aun cuando estas fueron almacenadas usando una función de hash no reversible?
Es entonces aquí donde entra en juego el password cracking, un software simple que toma una serie de palabras como "potenciales contraseñas" y le calcula el hash a cada una de ellas.
Luego verifica si dicho hash calculado se encuentra en el listado de hashes de la base de datos.
Si encuentra una coincidencia es probable que dicha palabra, inicialmente como "potencial contraseña", sea efectivamente "la contraseña" para ese hash en particular derrotando así al sistema de protección de contraseñas.

Si bien esto es factible, en la práctica las contraseñas usadas por los usuarios no son meras palabras de uso cotidiano como 'auto' o 'gato' sino que son variantes de estas como:

- 'autoauto123'
- 'HipopotamoHipopotamo'
- 'MiPasswordddddddd'

De aquí la importancia de un módulo que permita de forma simple generar todas esas variaciones.

Descripción

El programa a construir será un aplicativo por consola que leerá desde un archivo una serie de reglas de mangling.
Leerá además desde otro archivo o bien desde la entrada estándar un listado de palabras.
Por cada palabra leída se le aplicará la serie de modificaciones y las nuevas palabras resultantes serán escritas a un archivo de salida o bien directamente a la salida estandar.

Formato de los archivos

Tanto el archivo de palabras de entrada como el de salidas tienen un formato simplificado en donde cada línea contiene una única palabra.

El formato del archivo que contiene las reglas de alteración es ligeramente más complejo pues cada regla finaliza con un punto y coma, pueden haber más de una regla por línea y puede haber una cantidad indeterminada (mayor o igual a 1) de espacios en blancos entre los argumentos de cada regla, entre estos y el nombre de la regla y entre estos y el punto y coma.

Puede considerarse que los archivos de entrada no contienen errores.

Reglas de alteración

Las reglas de modificación o mangling a implementar son las siguientes.

- **uppercase n m**
  - Transforma los caracteres que están en la posición **n** hasta la posición **m** (inclusive) a mayúsculas (para los caracteres alfabéticos, otros caracteres son dejados intactos).
  - Se debe considerar que el texto a procesar esta en codificación ASCII y la *localización* es “C” [4].
  - Las posiciones tiene un estilo C, la posición 0 es el primer caracter, la posición 1 el segundo y así sucesivamente.
  - Además tanto **n** como **m** pueden ser negativos cuya interpretación es, -1 el último caracter, -2 el penúltimo caracter, etc.
  - Nótese que tanto las posiciones **n** como **m** podrían no existir debido a que la palabra es mucho más chica. En dicho caso, si el existe un subrango **n' m'** contenido en el rango **n m**, ese subrango de caracteres deben ser pasados a mayúsculas. Sin no existe ningún subrango, no

hay efecto.
    - El parámetro **n** representa el principio, asi como **m** representa el final. Es posible que tanto **n** como **m** marquen valores válidos pero que **n** indique una posición más avanzada que **m** (el final esta antes que el principio). En dichos casos, se deberá considerar que no hay ningún rango efectivo sobre cual aplicar.
- **lowercase n m**
    - Similar a **uppercase** pero transforma los caracteres seleccionados a minúsculas.
- **repeat n m r i**
    - Copia el substring delimitado por **n** y **m**, lo repite **r** veces y lo inserta en la posición **i**.
    - Las mismas observaciones descritas en **uppercase** aplican a los parámetros **n** y **m**.
    - Debe entenderse que si **i** es positivo, debe insertarse **antes** del caracter en la posición **i**
    - Si es negativo, debe insertarse **después** del caracter en la posición **i**.
    - Si **i** es positivo y excede las posiciones válidas de la palabra, se deberá insertar al final, si en cambio es negativo, se deberá insertar al principio.
- **rotate n**
    - Mueve los caracteres **n** lugares hacia la derecha en forma circular. Si **n** es negativo, la rotaciòn se hace hacia la izquierda.
- **insert i mmm**
    - Agrega el string **mmm** en la posición **i**. Las aclaraciones sobre donde insertar son las mismas que se explicaron en la regla **repeat**.
    - Por simplicidad, **mmm** no contiene ningún espacio ni salto de línea ni el caracter ';'.
- **revert i**
    - Revierte los efectos de las modificaciones de las últimas **i** reglas.
    - El parámetro **i** es siempre mayor o igual a 1.
- **print**
    - Emite la palabra actual con los modificaciones realizadas hasta el momento.

Formato de línea de comandos

Por línea de comandos se pasarán el archivo de reglas, el archivo de palabras iniciales y el archivo de palabras alteradas.
Para soportar la comunicación con otros programas, posiblemente un password cracking, se deberá poder interactuar con la entrada y salida estándar.
Para indicar este caso, el o los nombres de los archivos serán “-”. (No aplica al archivo de reglas que siempre es pasado por parámetro)

Ejemplos:

Se lee el archivo reglas.txt, diccionario.txt y se vuelca el contenido en alteraciones.txt
./tp reglas.txt diccionario.txt alteraciones.txt

Mismo ejemplo pero esta vez se lee la entrada estándar.
./tp reglas.txt - alteraciones.txt

El programa interactúa tanto con la entrada como con la salida estándar.
./tp reglas.txt - -

Códigos de retorno

- 0 Sin errores
- 1 Argumentos incorrectos.
- 2 El archivo de entrada no existe o el archivo de salida no pudo ser creado.

Entrada/salida estándar

La entrada estándar contendrá el listado de las palabras a ser modificadas sólo si el “nombre del archivo” corresponde al nombre especial “-” indicado en la línea de comandos.
En caso contrario, no habrá nada en la entrada estándar.
Se aplica lo mismo a la salida estándar.

Ejemplos de ejecución

Se mostrará a continuación los efectos de una serie de reglas. Nótese como los efectos son acumulativos y que las reglas son finalizadas por un ';'.

Ejemplo 1.

Siendo que el archivo de palabras inicial diccionario.txt que contiene solamente la palabra 'password', y dado el archivo reglas.txt (listado más abajo), la ejecución de
./tp reglas.txt diccionario.txt alteraciones.txt

es

Archivo reglas.txt:	*Resultados temporales paso a paso*
uppercase 0 0 ;	password --> Password
repeat 0 -1 1 -1 ;	Password --> PasswordPassword
print ;	(la palabra PasswordPassword es emitida)
insert -1 123 ;	PasswordPassword --> PasswordPassword123
print ;	(la palabra PasswordPassword123 es emitida)

Finalmente, el archivo alteraciones.txt es creado con el siguiente contenido:
PasswordPassword
PasswordPassword123

Ejemplo 2.

Siendo que el archivo de palabras inicial diccionario.txt contiene:
am
bail

y dado el archivo reglas.txt (listado más abajo), la ejecución de
./tp reglas.txt diccionario.txt alteraciones.txt

es

Archivo reglas.txt:	*Resultados temporales paso a paso (para la primer palabra)*
insert -1 e ;	am --> ame
print ;	(la palabra ame es emitida)
insert -1 mos ;	ame --> amemos
print ;	(la palabra amemos es emitida)
revert 4 ;	(vuelve 4 instrucciones para atrás, resultando en 'am')
insert -1 are ;	am --> amare
print ;	(la palabra amare es emitida)
insert -1 mos ;	amare --> amaremos
print ;	(la palabra amaremos es emitida)

Finalmente, el archivo alteraciones.txt es:
ame
amemos
amare
amaremos
baile
bailemos
bailare
bailaremos

Ejemplo 3.

Dado el archivo reglas.txt (listado más abajo), la ejecución de
echo “password” | ./tp reglas.txt - alteraciones.txt

es (nótese los espacios y saltos de líneas adicionales en reglas.txt)

Archivo reglas.txt:	*Resultados temporales paso a paso*
print ;	(la palabra password es emitida)
rotate 2 ;	password --> rdpasswo
rotate -4 ;	rdpasswo --> sswordpa
rotate 6 ;	sswordpa --> wordpass
print ;	(la palabra wordpass es emitida)
insert 0 qwerty ;	wordpass --> qwertywordpass
revert 2 ;	(vuelve 2 instrucciones para atras, resultando en 'wordpass')
rotate -4 ;	wordpass --> password

Finalmente, el archivo resultante alteraciones.txt es (nótese como las últimas modificaciones se pierden al no haber un **print** al final):
password
wordpass

Ejemplo 4.

Dado el archivo reglas.txt (listado más abajo), la ejecución de
echo “word” | ./tp reglas.txt - -

es

Archivo reglas.txt:	*Resultados temporales paso a paso*
repeat -1 -1 1 -1 ;	word --> wordd
repeat -3 -3 1 -1 ;	wordd --> worddr
repeat -5 -5 1 -1 ;	worddr --> worddro
repeat -7 -7 1 -1 ;	worddro --> worddrow
repeat -9 -9 1 -1 ;	(-9 -9 esta fuera de rango, no hay efecto)
repeat -11 -11 1 -1 ;	(-11 -11 esta fuera de rango, no hay efecto)
print ;	(la palabra worddrow es emitida)
uppercase 0 3 ;	worddrow --> WORDdrow
print ;	(la palabra WORDdrow es emitida)
uppercase 0 30 ;	(0 30 esta parcialmente fuera de rango, resultando en WORDDROW)
print ;	(la palabra WORDDROW es emitida)

Finalmente, por salida estándar se imprime:
worddrow
WORDdrow
WORDDROW

Ejemplo 5.

Dado el archivo reglas.txt (listado más abajo), la ejecución de
echo “word” | ./tp reglas.txt - -

es

Archivo reglas.txt:	*Resultados temporales paso a paso*
insert 0 Q ;	word --> Qword
insert 0 W ;	Qword --> WQword
revert 1 ;	(vuelve 1 instruccion para atras, resultando en 'Qword')
insert 0 E ;	Qword --> EQword
revert 2 ;	(vuelve 2 instruccion para atras, resultando en 'WQword')

Finalmente, por salida estándar no se imprime nada (no hubo ninguna instrucción **print**).

Restricciones

A fin de poder mostrar los conocimientos de la programación orientada a objetos, se deberá implementar al menos una jerarquía de clases, con al menos un método virtual haciendo un uso correcto del polimorfismo, con al menos una llamada a un método de una clase padre.
La jerarquía de clases no podrá tener atributos ni públicos ni protegidos.

No se podrá hacer uso de funciones, salvo la función *main*. Todo deberá estar encapsulado en las clases.

Se deberá hacer uso de la librería estándar *ifstream* o *istream* para leer los archivos. Se deberá usar los operadores >> de extracción de datos formateada de estas clases. [3]

No se podrá usar ninguno de los contenedores estándares STL, salvo std::string.
Tampoco se podrá usar ninguna de las librerías de C, como *fopen* o *malloc*. La idea es que el alumno se familiarice con los objetos y métodos de las clases de C++, especialmente con std::string, std::istream y std::ifstream.

Se deberá implementar un operador a elección.

No se podrá pasar por copia ningún objeto que sea potencialmente grande, como std::string. Deberá ser pasado por referencia, o por referencia constante donde corresponda.

Referencias

[1] http://en.wikipedia.org/wiki/Password_cracking
[2] http://en.wikipedia.org/wiki/Cryptographic_hash_function
[3] http://www.cplusplus.com/reference/
[4] http://docs.oracle.com/cd/E23824_01/html/E26033/glmbx.html