Pontificia Universidad
JAVERIANA
Colombia

ESTRUCTURAS DE DATOS

PROYECTO: ENTREGA 02

PRESENTADO POR:

SEBASTIÁN CAMILO H. MURCIA SIERRA

JUAN DIEGO ARIAS DURAN

DOCENTE:

ANDREA RUEDA

2024 - 1

ACTA DE EVALUACIÓN

Documento de diseño:

1. *Es probable que para esta entrega sólo fuera necesario un TAD, pero para las siguientes seguramente será necesario contar con un TAD que represente al sistema, con el cual puedan estar conectados los TADs que soportan la funcionalidad de cada componente.*
   - Se implementó el TAD Sistema y se conectaron los componentes del diccionario y los nuevos.
2. *En el diagrama de relación, no hay necesidad de hacerlo para un único TAD. Cuando ya se tengan más TADs, no es necesario incluir los parámetros de las funciones.*
   - Se agregaron los nuevos TADs y se removieron los parámetros dentro de las funciones.
3. *El comando turno no es de esta entrega, ni de este proyecto*
   - Se modificó el nombre erróneo.
4. *Debe haber un esquemático para cada comando, aunque las tareas se parezcan, pero si son comandos diferentes deben hacer algo diferente.*
   - No entendimos bien este punto, en los esquemáticos no hay ninguno igual.
5. *El plan de pruebas debe seguir la tabla vista en el primer ejercicio del curso, y debe estar acompañado de pantallazos que sirvan como evidencia de los resultados obtenidos en cada uno de los casos.*
   - Se modificó el plan de pruebas y se añadieron las capturas.

Código fuente:

1. *El uso de librerías no es adecuado. Los archivos .cxx nunca se incluyen, sino que se compilan junto con el main.*
   - Se modificó el tipo de archivo de .cxx a .hxx.
2. *Es importante revisar la ejecución con otros archivos de prueba, en particular englishWords-3.txt.*
   - Para esta entrega se utilizaron todos los archivos de texto de prueba propuestos (englishWords-[**0-5**].txt).

ANÁLISIS DE ENTRADAS, SALIDAS Y CONDICIONES DE COMANDOS

Comando	**Entrada**	**Salida**	**Condiciones**
inicializar	*nombre_archivo*	"El diccionario *nombre_archivo* se ha inicializado correctamente"	- Archivo existe - Diccionario **normal** no haya sido inicializado - Palabras válidas
iniciar_inverso	*nombre_archivo*	"El diccionario *nombre_archivo* se ha inicializado correctamente"	- Archivo existe - Diccionario **inverso** no haya sido inicializado - Palabras válidas
puntaje	*palabra*	"La *palabra* tiene un puntaje de *puntaje"*	- Palabra existe - Palabra válida
iniciar_arbol	*nombre_archivo*	"El arbol *nombre_archivo* se ha inicializado correctamente"	- Archivo existe - Arbol no haya sido inicializado - Palabras válidas
iniciar_arbol_inverso	*nombre_archivo*	"El arbol inverso *nombre_archivo* se ha inicializado correctamente"	- Archivo existe - Arbol no haya sido inicializado - Palabras válidas
palabras_por_prefijo	*prefijo*	"Las palabras que inician con este prefijo son: [palabras]"	- Arbol haya sido inicializado - Prefijo válido
palabras_por_sufijo	*sufijo*	"Las palabras que terminan con este sufijo son: [palabras]"	- Arbol inverso haya sido inicializado - Sufijo válido
salir	*(No tiene entradas)*	"Saliendo del programa…"	- No hay condiciones

DESCRIPCIÓN DE TADS

TAD Sistema
Conjunto mínimo de datos: ● dicVector, dato de tipo Diccionario, representa los diccionarios creados en vectores. ● dicArbol, dato de tipo Trie, representa el diccionario creado en un árbol.
Comportamiento (operaciones): ● getDiccionario(), retorna un apuntador al diccionario en vectores. ● getArbol(), retorna un apuntador al diccionario en árbol. ● longitudPalabra(palabra), retorna la longitud de una palabra dada. ● puntuarPalabra(palabra), retorna la puntuación de scrabble de una palabra dada.

TAD Diccionario
Conjunto mínimo de datos: ● palabras, vector de cadena de caracteres, representa las palabras en el diccionario. ● palabrasInversas, vector de cadena de caracteres, representa las palabras del diccionario escritas al inverso.
Comportamiento (operaciones): ● obtenerPalabras(), retorna las palabras. ● verificarPalabra(palabra), recibe una palabra y valida que no contenga guiones, mayúsculas, números y signos de puntuación. ● agregarPalabra(palabra), recibe una palabra para agregarla al diccionario. ● iniDiccionario(diccionario), recibe el diccionario y guarda las palabras que contiene en el vector llamado palabras.

- iniDiccionarioInverso(diccionario), recibe el diccionario, invierte las palabras que contiene y las almacena en el vector llamado palabrasInversas.
- puntuarPalabra(palabra), recibe una palabra, verifica que exista tanto en diccionario normal como inverso, posterior, calcula el puntaje de la palabra y retorna el puntaje.
- printDiccionario(), imprime las palabras.
- printDiccionarioInverso(), imprime las palabras inversas.

TAD Trie

Conjunto mínimo de datos:

- root, apuntador a dato de tipo TrieNode, representa la raiz del arbol.

Comportamiento (operaciones):

- clear(), limpia la memoria del árbol recursivamente.
- insert(), inserta una palabra en el árbol.
- search(), busca una palabra en el árbol.
- verificarPalabra(palabra), recibe una palabra y valida que no contenga guiones, mayúsculas, números y signos de puntuación.
- agregarPalabra(palabra), recibe una palabra para agregarla al diccionario.
- iniArbol(diccionario), recibe el nombre del archivo e inserta las palabras en el árbol.
- iniArbolInverso(diccionario), recibe el nombre del archivo e inserta las palabras, luego de invertirlas, en el árbol.
- recogerPalabras(nodo, palabra, resultados),recoge y añade al resultado de manera recurrente las palabras que extienden el prefijo dado desde un nodo específico del trie.
- palabrasPrefijo(prefijo), ubica todas las palabras posibles a construir a partir de ese prefijo.

- palabrasSufijo(sufijo), ubica todas las palabras posibles a construir que terminan con ese sufijo.

TAD TrieNode

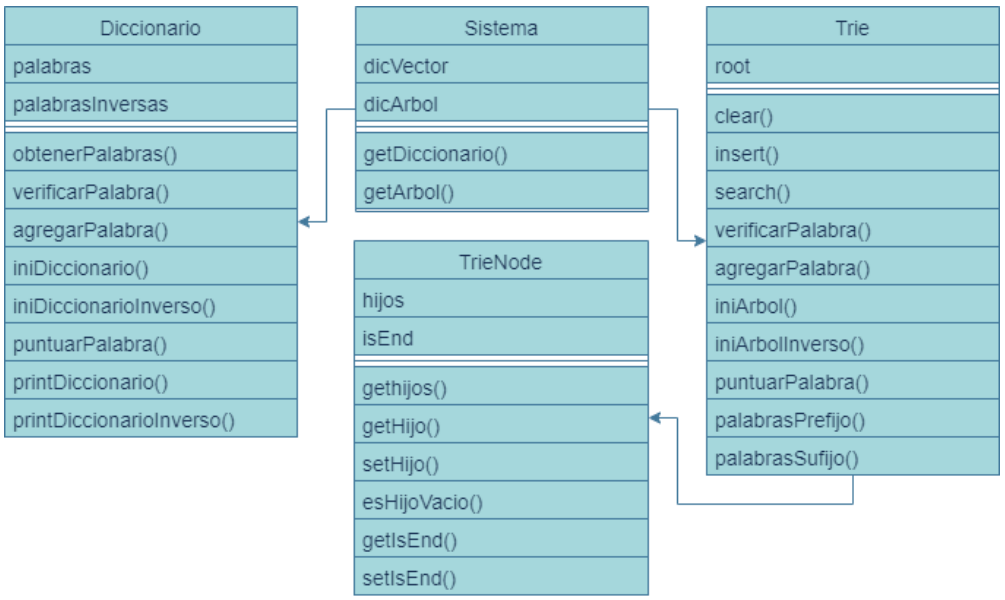
Conjunto mínimo de datos:

- hijos, vector de apuntadores a datos tipo TrieNode, representa los 26 espacios para cada letra del abecedario.
- isEnd, dato booleano, representa si ha llegado al final de la palabra.

Comportamiento (operaciones):

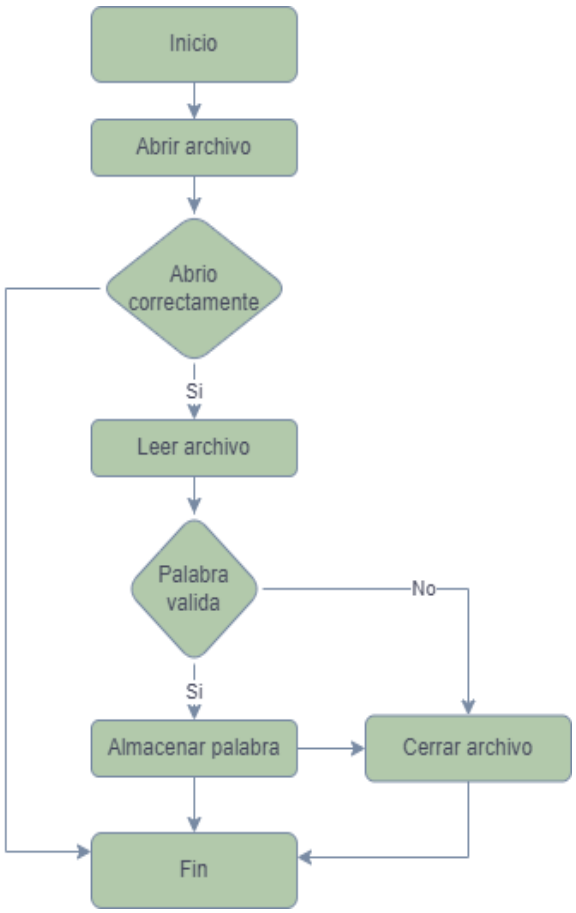
- gethijos(), obtiene el vector de hijos.
- getHijo(int index), obtiene el hijo de la posición dada.
- setHijo(int index), inicializa el hijo de la posición dada.
- esHijoVacio(int index), retorna si el hijo está inicializado o no.
- getIsEnd(), obtiene el estado del dato isEnd.
- setIsEnd(bool estado), cambia el estado del dato isEnd al proporcionado.

- **Diagrama de relación entre TADs:**

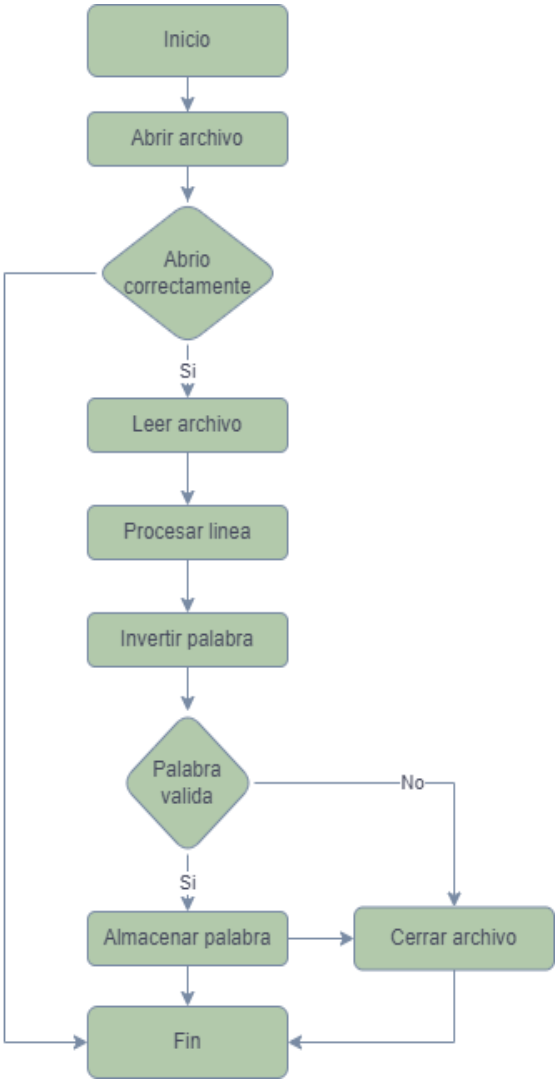


ESQUEMÁTICOS DE OPERACIONES

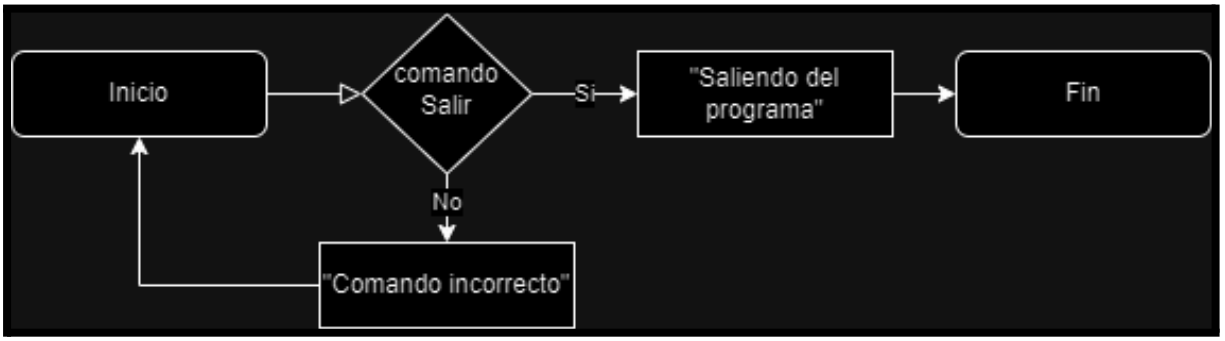
inicializar:



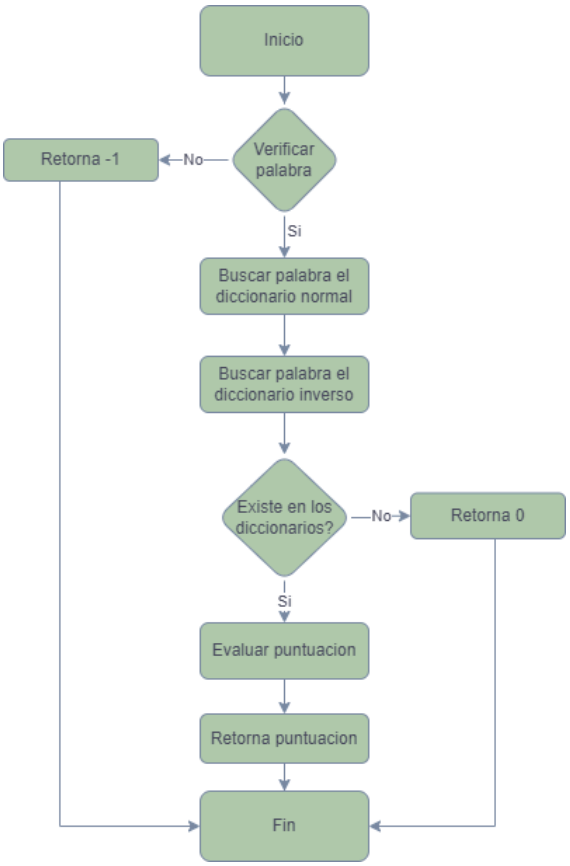
iniciar_inverso:



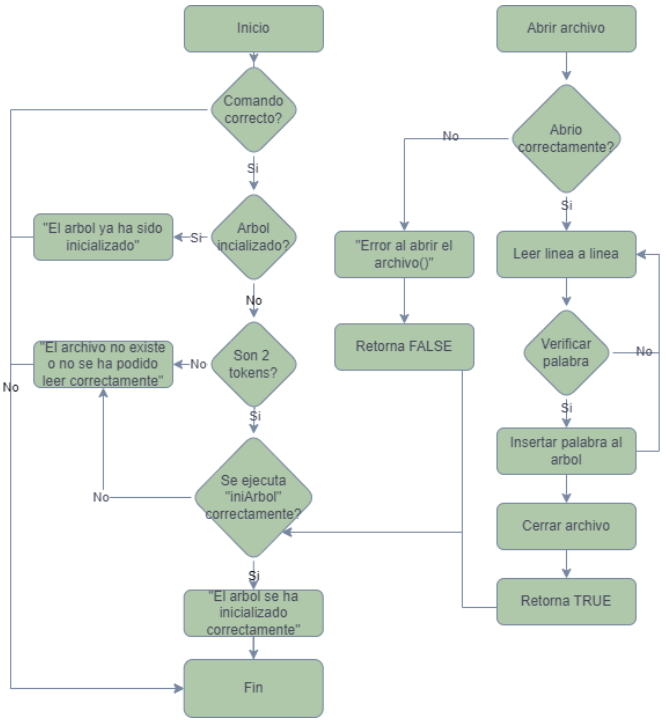
salir:



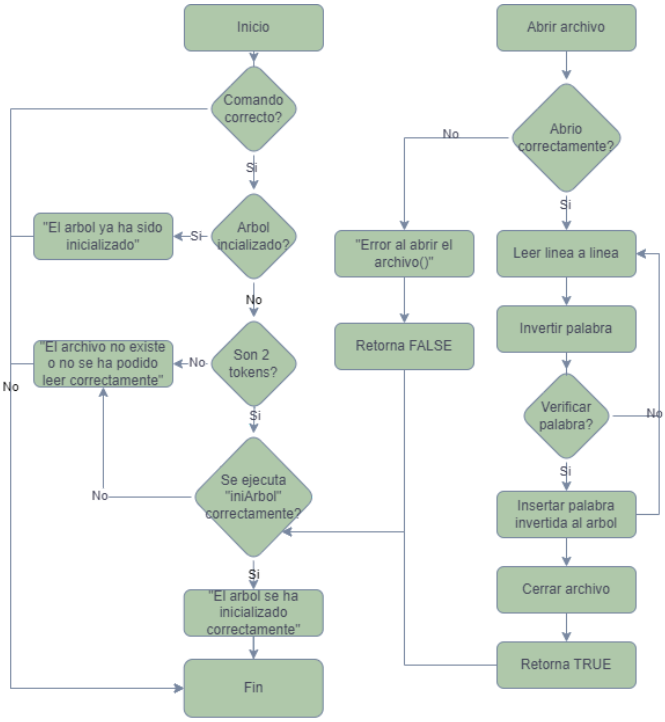
puntaje:



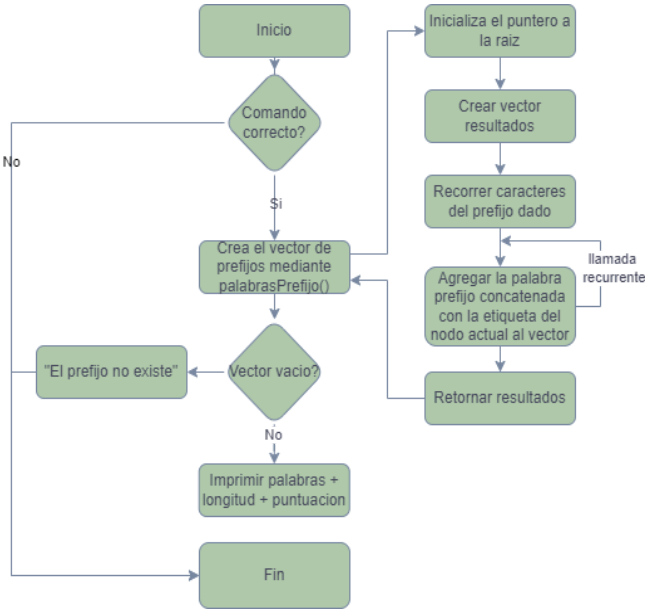
iniciar_arbol:



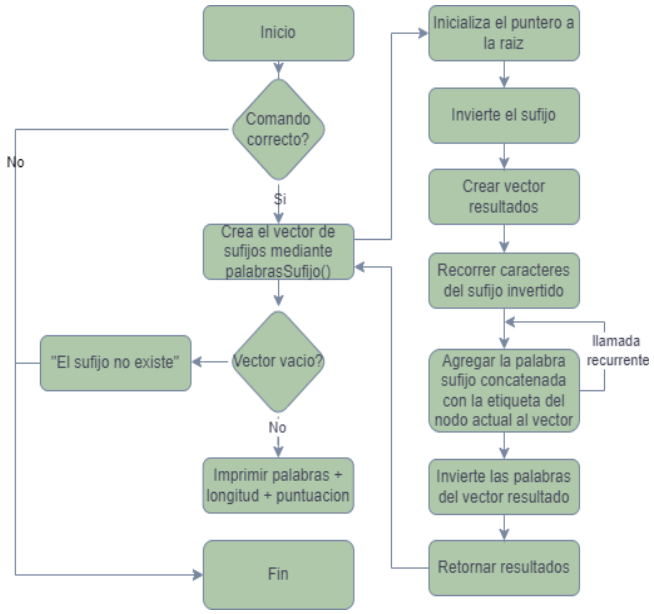
iniciar_arbol_inverso:



palabras_por_prefijo:



palabras_por_prefijo:



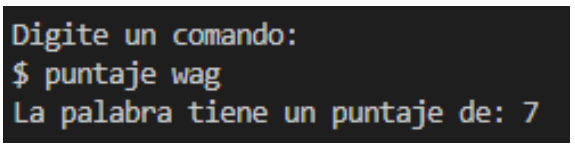
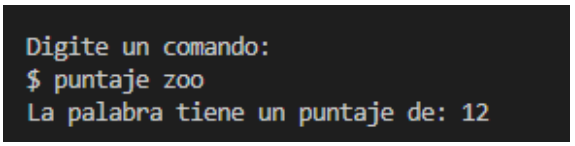
PLAN DE PRUEBAS

Plan de pruebas: Comando puntaje			
Caso	**Entradas**	**Resultado esperado**	**Resultado obtenido**
Prueba 1	abandon	"La palabra ***abandon*** tiene un puntaje de 10"	"La palabra ***abandon*** tiene un puntaje de 10"
Prueba 2	wag	"La palabra ***wag*** tiene un puntaje de 7"	"La palabra ***wag*** tiene un puntaje de 7"
Prueba 3	zoo	"La palabra ***zoo*** tiene un puntaje de 12"	"La palabra ***zoo*** tiene un puntaje de 12"

Ejecución:

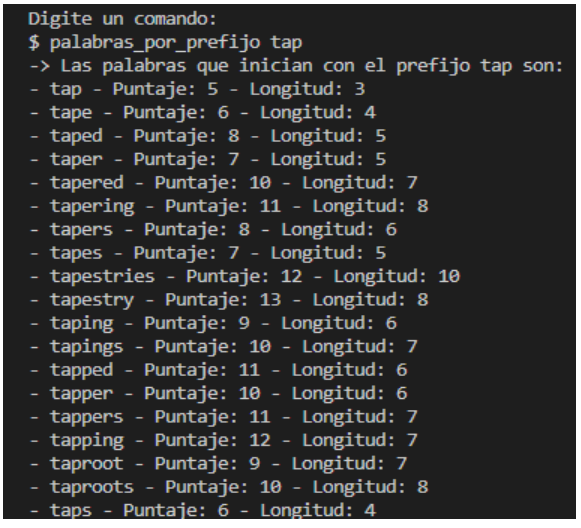
Descripción de caso	Evidencia de ejecución

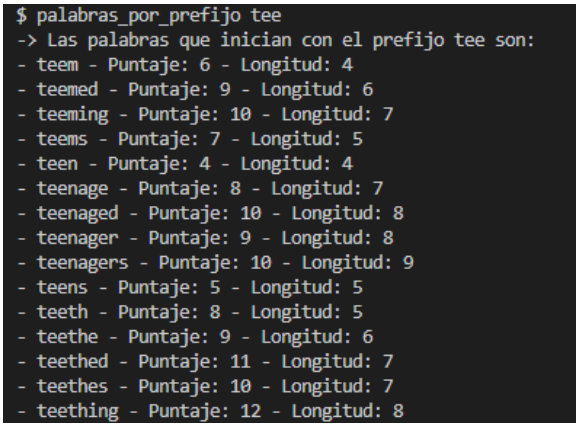
1: Prueba 1	`Digite un comando:`
	`$ puntaje abandon`
	`La palabra tiene un puntaje de: 10`

2: Prueba 2	
3. Prueba 3	

Plan de pruebas: Comando palabras_por_prefijo			
Caso	**Entradas**	**Resultado esperado**	**Resultado obtenido**
Prueba 1	tap	"Las palabras que inician con este prefijo son: *[palabras + longitud + puntuación]*"	"Las palabras que inician con este prefijo son: *[palabras + longitud + puntuación]*"
Prueba 2	tee	"Las palabras que inician con este prefijo son: *[palabras + longitud + puntuación]*"	"Las palabras que inician con este prefijo son: *[palabras + longitud + puntuación]*"
Prueba 3	wor	"Las palabras que inician con este prefijo son: *[palabras + longitud + puntuación]*"	"Las palabras que inician con este prefijo son: *[palabras + longitud + puntuación]*"

Ejecución:

Descripción de caso	**Evidencia de ejecución**
1: Prueba 1	

2: Prueba 2	
3. Prueba 3