

Organización de Computadoras  
Proyecto N°1  
Programación en Lenguaje C

Integrantes:

Rodríguez, Marcelo. LU: 93554

Enrique, Cristian Facundo. LU: 89498

**TDA's IMPLEMENTADOS: prototipos de funciones y sus comportamientos.****TDA Lista**

1. tLista crearLista(): crea una lista vacía y la devuelve. Se reserva el espacio en memoria correspondiente a la lista y se setean los campos de la misma en nulo, siendo el tamaño inicial igual a 0.
2. void l\_insertar(TLista lista, TPosicion pos, TElemento elem): inserta un elemento en la posición anterior a una dada. Recibe como argumentos a la lista lista que se desea modificar, la posición próxima siguiente a la cual se va a realizar la inserción, y el nuevo elemento. Se busca el nodo recibido como parámetro, se crea un nuevo nodo con el elemento recibido, y se inserta inmediatamente antes a pos. Finalmente, se actualiza la estructura.
3. void l\_eliminar(TLista lista, TPosicion pos): Elimina el elemento de una posición dada de la lista. Se reciben como argumentos la lista y la posición que se desea remover. Se busca en la lista el nodo equivalente al recibido, se elimina y se libera el espacio de memoria de dicho nodo. Finalmente se actualiza la estructura.
4. TPosicion l\_primera(TLista lista): devuelve la primera posición de la lista lista.  
Se recibe como parámetro la lista lista. Se devuelve el primer elemento. Si la lista está vacía, entonces dicho puntero será nulo (POS\_NULA).
5. TPosicion l\_ultima(TLista lista): devuelve la última posición de la lista lista.  
Se recibe como parámetro la lista lista. Se devuelve el último elemento. Si la lista está vacía, entonces dicho puntero será nulo (POS\_NULA).
6. TPosicion l\_anterior(TLista lista, TPosicion pos): devuelve la posición anterior a una dada.  
Éste método recibe una lista y la posición de la cual se desea conocer la celda anterior.  
Devuelve un puntero nulo si el nodo recibido no tiene un elemento siguiente. En caso contrario, el método retorna la posición correspondiente.
7. TPosicion l\_siguiente(TLista lista, TPosicion pos): devuelve la posición siguiente a una dada.  
Éste método recibe una lista y la posición de la cual se desea conocer el siguiente nodo.  
Devuelve un puntero nulo si el nodo recibido no tiene un elemento siguiente. En caso contrario, el método retorna la posición correspondiente.
8. TElemento l\_recuperar(TLista lista, TPosicion pos): devuelve el elemento correspondiente a una determinada posición de la lista. Recibe como parámetro una lista y una posición. Retorna el puntero al elemento de la posición recibida.
9. int l\_size(TLista lista): responde con la cantidad de elementos de la lista. Recibe la lista, y retorna la cantidad de elementos de la misma.

**TDA ListaOrdenada**

1. TListaOrdenada crear\_lista\_ordenada(int (\*f)(void \*,void \*)) : Crea una lista ordenada en base a un Tlista, el orden depende exclusivamente de la función f. Retorna una lista ordenada. Recibe un puntero a una función que compara elementos. Reserva memoria para el tipo TListaOrdenada
2. int lo\_insertar(TListaOrdenada lista, TElemento elem): Agrega un elemento en la posición correspondiente de la lista, de modo que la misma quede siempre ordenada de forma ascendente. Recibe una lista ordenada y el elemento a insertar. Recorre la lista hasta encontrar la ubicación final. En cuanto a la reserva de memoria lo hace para la nueva celda.
3. int lo\_eliminar(TListaOrdenada lista, TPosicion pos) :Elimina el elemento de una posición de la lista. Reacomoda la lista adecuadamente al eliminar en posiciones intermedias. Libera el espacio de memoria reservado por lo\_insertar.
4. int lo\_size(TListaOrdenada lista) :Retorna la cantidad de elementos de la lista.
5. TPosicion lo\_primera(TListaOrdenada lista) : Retorna la primera posición de la lista. Se recibe como parámetro la lista lista. Se devuelve el primer elemento. Si la lista está vacía, entonces dicho puntero será nulo (POS\_NULA).
6. TPosicion lo\_ultima(TListaOrdenada lista) : Retorna la última posición de la lista. Se recibe como parámetro la lista lista. Se devuelve el último elemento. Si la lista está vacía, entonces dicho puntero será nulo (POS\_NULA).
7. TPosicion lo\_siguiente(TListaOrdenada lista, TPosicion pos) : Retorna la posición siguiente de pos. Éste método recibe una lista y la posición de la cual se desea conocer el siguiente nodo. Devuelve un puntero nulo si el nodo recibido no tiene un elemento siguiente. En caso contrario, el método retorna la posición correspondiente.

**TDA Trie**

1. `TTrie crear trie()` : Crea un árbol trie y lo devuelve. Reserva memoria para el tipo `Ttrie` y para el nodo raíz, el cual queda con rotulo nulo (`ELE_NULO`)

2. `int tr_insertar(TTrie tr, char* str)` :

Inserta un string en el trie, inicializando el valor de contador asociado en uno. En caso de que el string ya se encuentre representado en el trie, aumenta el valor del contador asociado a dicho string en una unidad. Recibe como parámetros un trie y una cadena de caracteres. Comienza a insertar desde los hijos de la raíz, si la primera letra ya existe continúa por sus hijos, así hasta llegar al final de la palabra, si en algún momento no está insertado un caracter lo inserta, creando un nuevo nodo y reservando la memoria correspondiente. Si la palabra ya estaba ingresada aumenta en 1 el contador de esta.

3. `int tr_pertenece(TTrie tr, char* str)` : Chequea si un string pertenece a un trie. Recibe un trie y una cadena de caracteres. La búsqueda de la cadena consiste en ir desplazándose por los hijos de los nodos que contengan el caracter correspondiente en ese nivel.

4. `int tr_recuperar(TTrie tr, char* str)` : Recupera el entero asociado a un string en un trie. Recibe un trie y una cadena de caracteres. Primero realiza una búsqueda igual que la función `tr_pertenece` solo que esta vez en el último nodo almacenado correspondiente a esa palabra se detendrá y retornará el entero asociado.

5. `int tr_size(TTrie tr)` : Retorna la cantidad de elementos del trie.

6. `int tr_eliminar(TTrie tr, char* str)`: Elimina el string `str` del trie `tr`. Liberando la memoria utilizada.

Comienza borrando desde la hoja del árbol (última letra) y termina en la primera letra, en su recorrido para llegar a la primera letra puede toparse con un nodo a borrar que tenga hijos y no pueda ser borrado, en ese caso se detiene. Cada vez que se elimina un nodo se libera la memoria guardada para el `TNodo` y para la lista ordenada de hijos que almacena este nodo.

## DECISIONES DE DISEÑO

### **TDA Lista**

Se siguen las pautas de diseño establecidas en el enunciado del proyecto: cada uno de los elementos de la lista son punteros tipo TElemento y la lista es simplemente enlazada sin nodo centinela. Las celdas mantienen un elemento y un enlace a la siguiente. La lista a su vez, mantiene un enlace al primer nodo y un valor entero correspondiente a su tamaño.

Una lista se considera vacía si el valor que indica el tamaño es un cero.

### **TDA Lista Ordenada**

Hace uso del TDA Lista, en base a esta ordenada sus elementos. A su vez contiene un valor entero que almacena su tamaño.

### **TDA Trie**

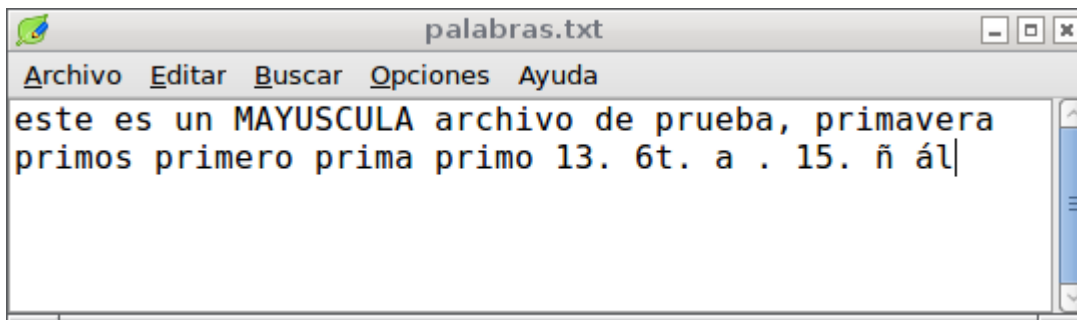
Se siguen las pautas de diseños provistas por la catedra que indicaban que los nodos del árbol tienen como rotulo un caracter (char), y almacena adicionalmente un contador de tipo entero (int) Además el trie debe implementarse manteniendo referencia a un nodo raíz, y considerando cada nodo del árbol como una estructura que mantiene referencia al padre y una lista ordenada de nodos como hijos. Se hace un uso de memoria dinámica, muy eficiente.

### **Aplicación**

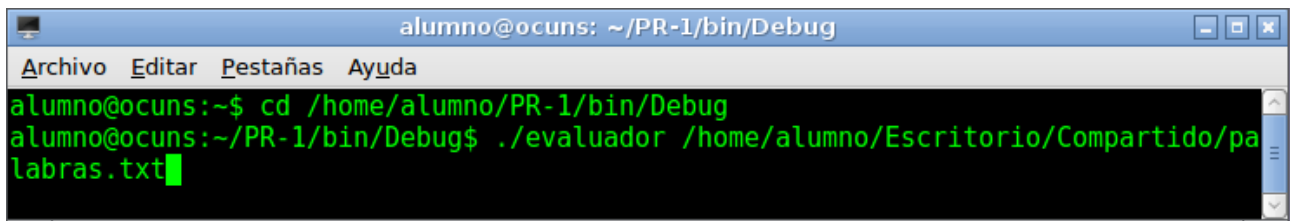
Para la programación de la aplicación principal, se usó un árbol trie, el cual almacena las palabras que se leen del archivo de texto. En las funciones que fueron necesario se usaron recorridos preorden y en salir, se hace uso de la liberación de memoria que se reserva cuando se lee el archivo y almacena las palabras. Se hace uso de todas las estructuras antes mencionadas de forma indirecta, ya que directamente se trabaja sobre el trie, sus nodos y los hijos de éstos.

## Ejemplo de aplicación del programa

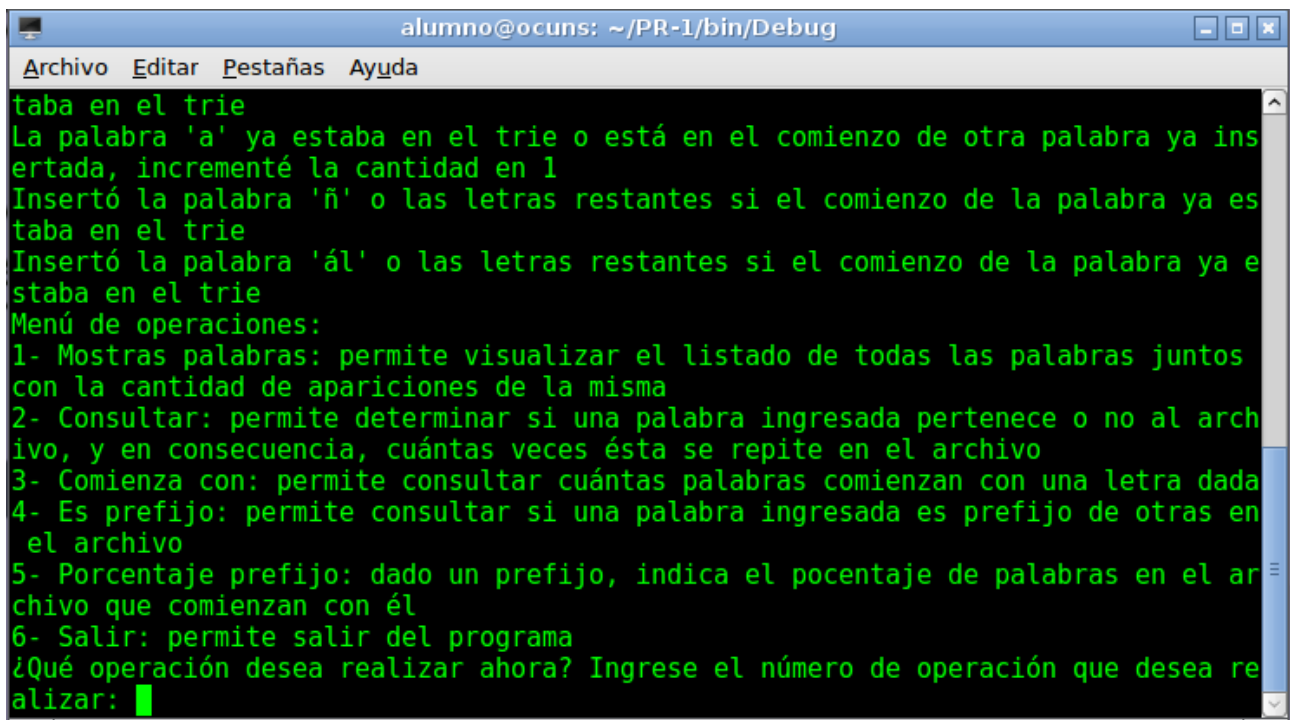
Supongamos que tenemos un archivo de texto “palabras.txt” que contiene:



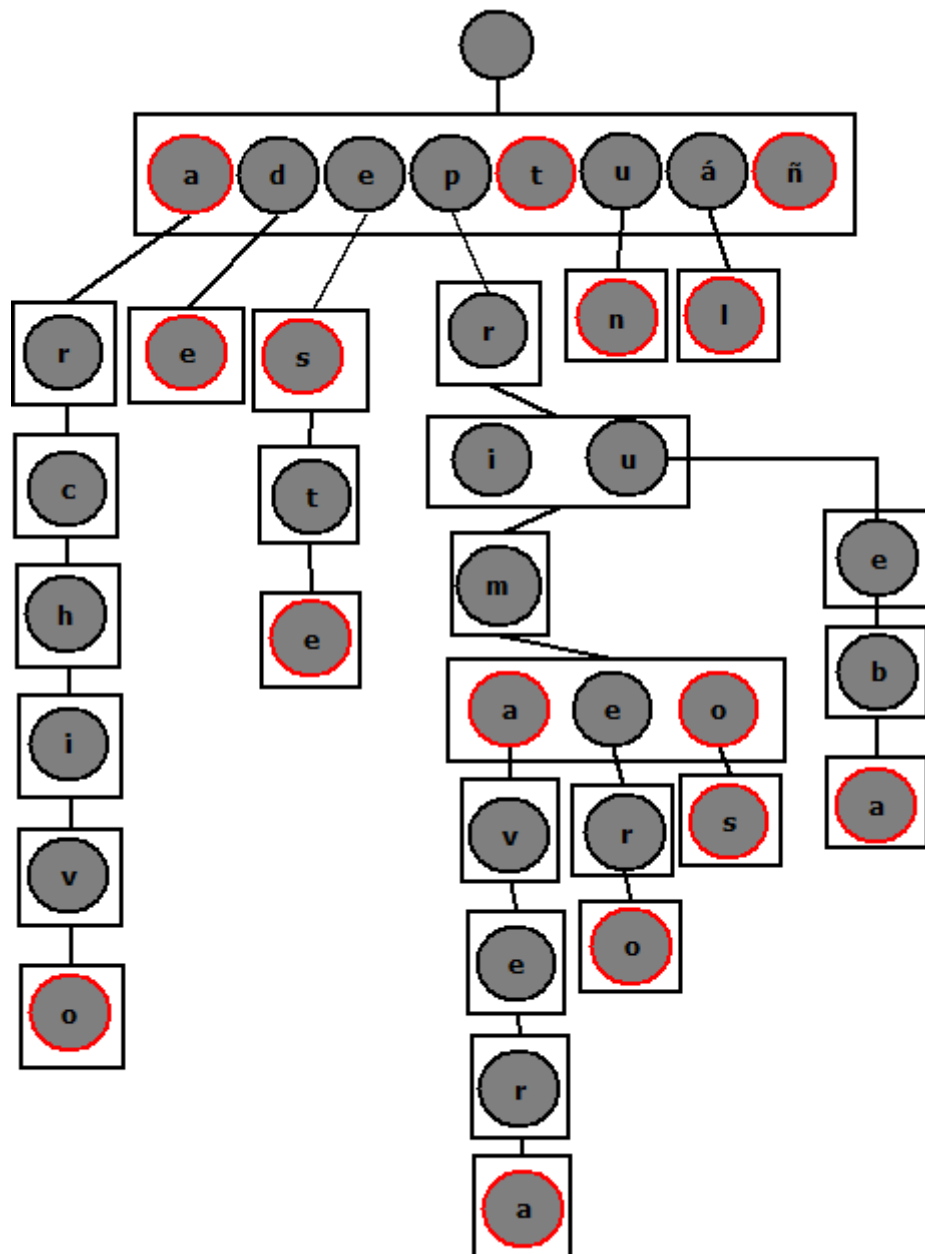
Para ejecutar el programa, hacemos:



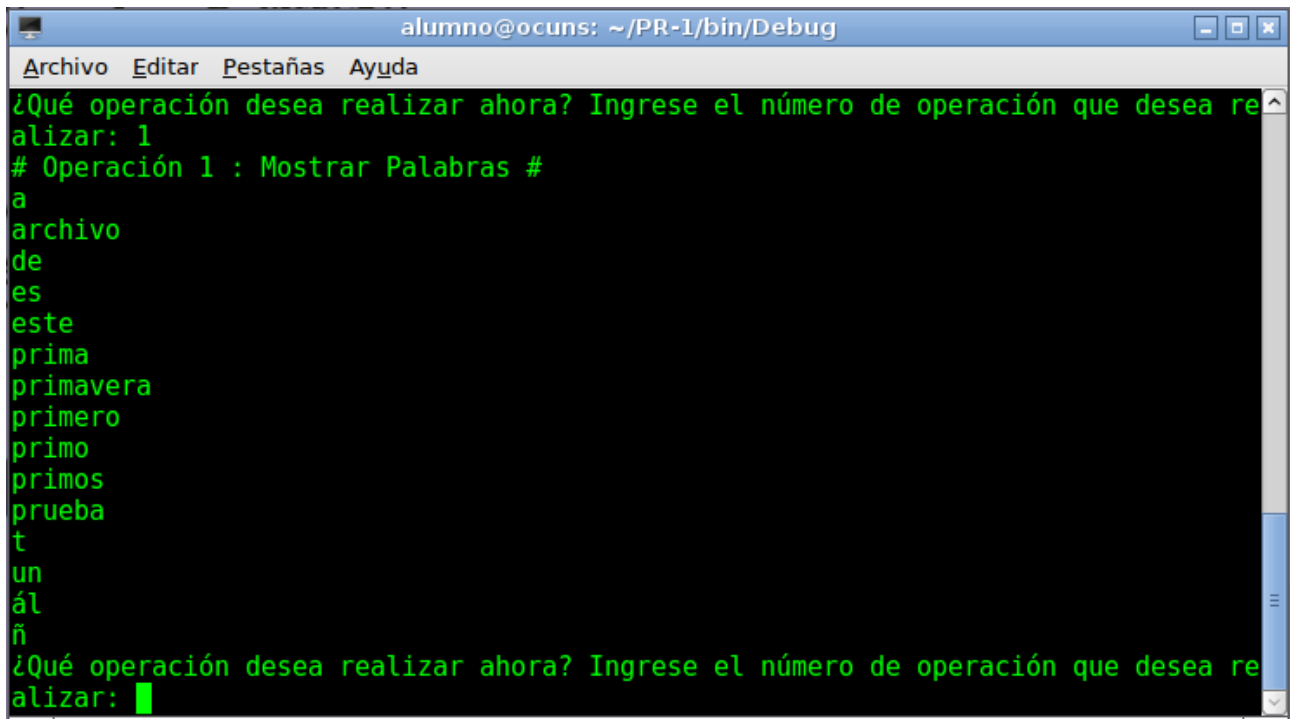
Una vez ejecutado el programa, se cargan las palabras del archivo en un trie y se muestra el siguiente menú de operaciones:



De manera interna para manejar el archivo de texto se utiliza un trie. En este caso, el trie quedará conformado de la siguiente manera (los contornos rojos indican que la cantidad de la palabra es 1):

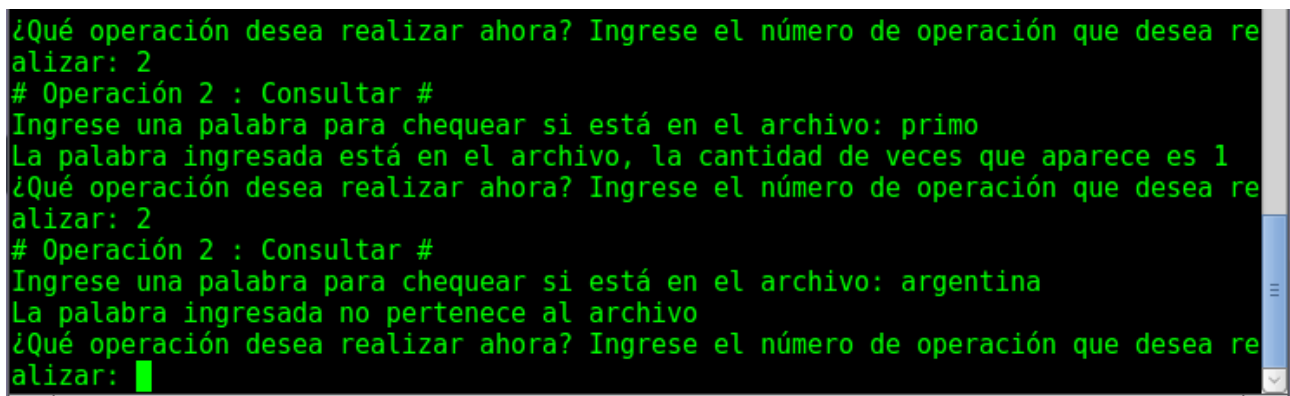


Para ver todas las palabras que hay en el archivo (número de operación 1):



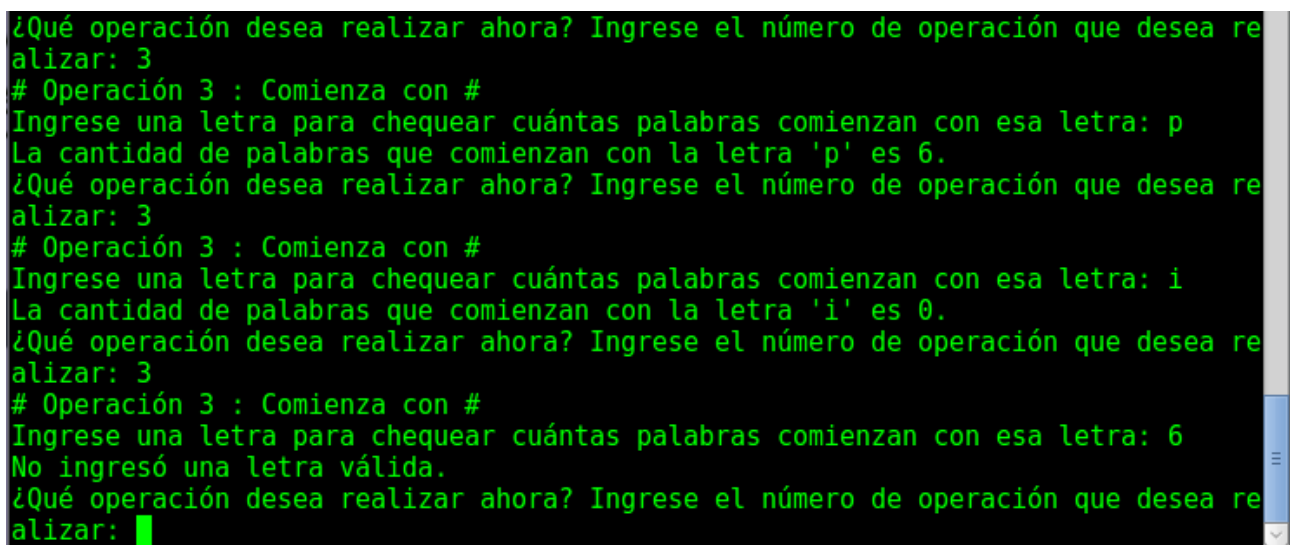
```
alumno@ocuns: ~/PR-1/bin/Debug
Archivo  Editar  Pestañas  Ayuda
¿Qué operación desea realizar ahora? Ingrese el número de operación que desea re
alizar: 1
# Operación 1 : Mostrar Palabras #
a
archivo
de
es
este
prima
primavera
primero
primo
primos
prueba
t
un
ál
ñ
¿Qué operación desea realizar ahora? Ingrese el número de operación que desea re
alizar: █
```

Para consultar por palabras particulares para chequear si están en el archivo (número de operación 2):



```
alumno@ocuns: ~/PR-1/bin/Debug
Archivo  Editar  Pestañas  Ayuda
¿Qué operación desea realizar ahora? Ingrese el número de operación que desea re
alizar: 2
# Operación 2 : Consultar #
Ingrese una palabra para chequear si está en el archivo: primo
La palabra ingresada está en el archivo, la cantidad de veces que aparece es 1
¿Qué operación desea realizar ahora? Ingrese el número de operación que desea re
alizar: 2
# Operación 2 : Consultar #
Ingrese una palabra para chequear si está en el archivo: argentina
La palabra ingresada no pertenece al archivo
¿Qué operación desea realizar ahora? Ingrese el número de operación que desea re
alizar: █
```

Para consultar cuántas palabras comienzan con una letra particular (número de operación 3):



```
alumno@ocuns: ~/PR-1/bin/Debug
Archivo  Editar  Pestañas  Ayuda
¿Qué operación desea realizar ahora? Ingrese el número de operación que desea re
alizar: 3
# Operación 3 : Comienza con #
Ingrese una letra para chequear cuántas palabras comienzan con esa letra: p
La cantidad de palabras que comienzan con la letra 'p' es 6.
¿Qué operación desea realizar ahora? Ingrese el número de operación que desea re
alizar: 3
# Operación 3 : Comienza con #
Ingrese una letra para chequear cuántas palabras comienzan con esa letra: i
La cantidad de palabras que comienzan con la letra 'i' es 0.
¿Qué operación desea realizar ahora? Ingrese el número de operación que desea re
alizar: 3
# Operación 3 : Comienza con #
Ingrese una letra para chequear cuántas palabras comienzan con esa letra: 6
No ingresó una letra válida.
¿Qué operación desea realizar ahora? Ingrese el número de operación que desea re
alizar: █
```



Para chequear si alguna palabra es prefijo de otra en el archivo (número de operación 4):

```
¿Qué operación desea realizar ahora? Ingrese el número de operación que desea re
alizar: 4
# Operación 4 : Es prefijo #
Ingrese una palabra para chequear si es prefijo de otras: pri
La palabra 'pri' es prefijo de otras.
¿Qué operación desea realizar ahora? Ingrese el número de operación que desea re
alizar: este
No ingresó un número, ingrese un número de operación: 4
# Operación 4 : Es prefijo #
Ingrese una palabra para chequear si es prefijo de otras: este
La palabra 'este' está pero no es prefijo de otras.
¿Qué operación desea realizar ahora? Ingrese el número de operación que desea re
alizar: █
```

Para consultar el porcentaje de un prefijo en las palabras del archivo (número de operación 5):

```
alumno@ocuns: ~/PR-1/bin/Debug
Archivo Editar Pestañas Ayuda
# Operación 5 : Es prefijo #
Ingrese una palabra para chequear si es prefijo de otras y en que porcentaje de
palabras aparece: p
La palabra 'p' es prefijo de otras.
Y aparece en un 40.00 por ciento de palabras
¿Qué operación desea realizar ahora? Ingrese el número de operación que desea re
alizar: 5
# Operación 5 : Es prefijo #
Ingrese una palabra para chequear si es prefijo de otras y en que porcentaje de
palabras aparece: pre
La palabra 'pre' no es prefijo de ninguna.
¿Qué operación desea realizar ahora? Ingrese el número de operación que desea re
alizar: 5
# Operación 5 : Es prefijo #
Ingrese una palabra para chequear si es prefijo de otras y en que porcentaje de
palabras aparece: primo
La palabra 'primo' es prefijo de otras.
Y aparece en un 6.00 por ciento de palabras
¿Qué operación desea realizar ahora? Ingrese el número de operación que desea re
alizar: 5
# Operación 5 : Es prefijo #
Ingrese una palabra para chequear si es prefijo de otras y en que porcentaje de
palabras aparece: primos
La palabra 'primos' está pero no es prefijo de otras.
¿Qué operación desea realizar ahora? Ingrese el número de operación que desea re
alizar: █
```

Por último, para salir del programa (número de operación 6):

```
alumno@ocuns: ~/PR-1/bin/Debug
Archivo Editar Pestañas Ayuda
¿Qué operación desea realizar ahora? Ingrese el número de operación que desea re
alizar: 6
Operación realizada con éxito
alumno@ocuns:~/PR-1/bin/Debug$ █
```