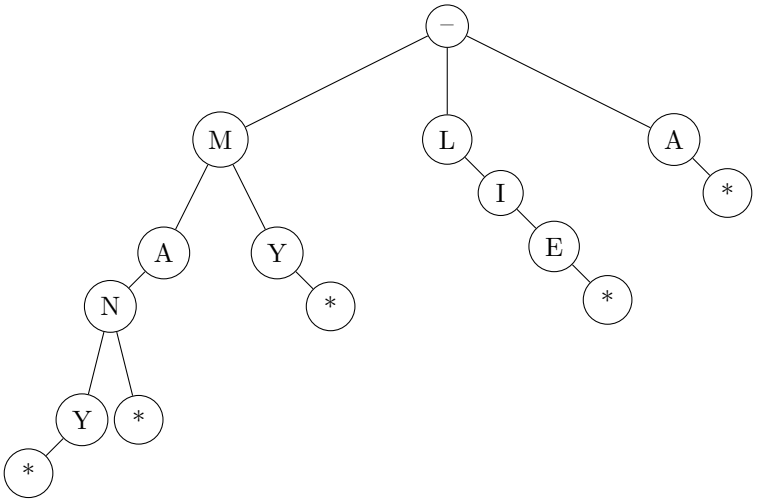
Tarea Nro. 2

IMPORTANTE: Las tareas son **individuales**. Cualquier acción que pueda beneficiar de forma injusta la calificación de su tarea está prohibida, incluyendo la presentación de cualquier componente que no es de su autoría o la facilitación de esto para otros. Es considerado aceptable discutir -*en líneas generales*- los métodos y resultados con sus compañeros, pero **se prohibe explícitamente realizar las tareas en conjunto o compartir código de programación**. **Utilizar código de internet que no es de su autoría (sin citarlo apropiadamente,) también es considerado plagio**. Por último, por favor presten atención en las intrucciones de entrega de la tarea, que pueden incluir políticas de nombre de archivos, codificación y criterios de evaluación.

Contexto de la tarea

Un **trie**, también conocido como **prefix tree**, es una estructura de datos de tipo árbol que, a diferencia de un árbol binario de búsqueda, sus nodos almacenan caracteres y cada recorrido alguna palabra. En un **trie** la cantidad de hijos de cada nodo no está limitada y puede, por ejemplo, tener tantos hijos como el número de letras del alfabeto. Como ya estarán suponiendo, todos los descendientes de un nodo tienen un prefijo común de la palabra asociada con ese nodo, y la raíz del **trie** está asociada con un string vacío (no tiene ningún carácter almacenado).



Si un nodo tiene un hijo que contiene al caracter '`*`', indica que la palabra completada hasta este nodo existe en el trie (en otras palabras, indica que este nodo es también una hoja en este trie). En este ejemplo, recorriendo el árbol, podemos extraer las palabras: `a, lie, my, many` y `man`.

Comúnmente, una estructura **trie** es utilizada para almacenar las palabras contenidas en cierto texto, por ejemplo, en el Diccionario de la Real Academia de la Lengua Española, y luego realizar **autocompletación** mientras un texto se está escribiendo, de manera equivalente a cómo lo hace el doble TAB en una consola Unix, o como ocurre con el teclado en su teléfono móvil.

Descripción del trabajo a realizar

Para esta tarea se pide seguir los siguientes pasos:

1. Prepare el código necesario para poder manipular y manejar estructuras del tipo **trie**. Para esto, defina cada nodo como:

```
// Tamano alfabeto mas caracter de termino de una palabra
#define ALPHABET_SIZE 29

typedef struct mtrie{
   char letter;
   struct mtrie *child [ALPHABET_SIZE];
}TrieNode;
```

2. Construya las siguientes interfaces que le permitan operar con la estructura **trie**:

```
// Funcion que permite ingresar al trie una nueva palabra
void insertWord(TrieNode *tr, char *word);

// Funcion que permite eliminar del trie alguna palabra
void deleteWord(TrieNpde *tr, char *word);

// Funcion que verifica si la palabra word se encuentra o no en el trie
int wordExist(TrieNode *tr, char *word);

// Funcion que permite agregar el caracter c al nodo apuntado por tr
TrieNode *insertNode(TrieNode *tr, char c);
```

3. Sn programa debe permitir leer un texto arbitrario de un archivo cuyo nombre se pasará como argumento por la línea de commando y luego cargar las palabras que ahí se encuentren dentro de una estructura **trie**. Notar que las palabras en el texto podrían estar separadas por espacios, comas, puntos, tabulaciones, etc. o incluso saltos de línea. Deben usar los métodos que hemos visto en clases y las funciones en la librería `string.h` para delimitar las palabras. Al completar este paso, el programa debe imprimir por pantalla un mensaje indicando el número de palabras que se agregaron al trie.

   OJO: este número puede ser different al número total de palabras en el texto, en el cual puede que algunas sean repetidas.

4. A continuación su programa debe interactuar con el usuario permitiendo el ingreso de una cadena de caracteres (string). Como salida el programa debe arrojar todas las palabras del trie que comiencen **exactamente** con esos caracteres. A continuación se da un ejemplo:

```
./elo320_tarea2 textoentrada.txt
Ingrese cadena de caracteres: abr
Palabras sugeridas: abra, abrir, abracadabra
```

donde el archivo `textoentrada.txt` usado en el ejemplo posee la siguiente información:

```
Juncal:Teaching mjescobar$ cat textoentrada.txt
abrir abracadabra abrelata cama foca cabeza manzana pera silla abra lampara
```

Terminando este paso, el programa debe seguir preguntando por nuevas cadenas de caracteres entregadas por el usuario. Si el usuario no ingresa ningún caracter (es decir, si apreta solo ENTER), el programa finaliza.

5. Implemente la función que le permita eliminar el **trie** completamente. Esta función debe ser llamada antes de finalizar el programa. La interfaz de la función es:

```
void destroyTrie (TrieNode* tnode);
```

6. Ejecute su tarea, **correctamente**, en el archivo entregado: `codigodavinci.txt`.

Consideraciones y Formato de entrega

- Utilice comentarios para describir lo que hace en cada etapa de su código.
- El código deberá estar escrito según el estándar de codificación GNU.
- El código escrito debe estar perfectamente *indentado* con tabuladores, no espacios.
- Solamente se aceptarán y revisarán tareas con las siguientes condiciones:
  - Toda tarea debe ser correctamente compilada y ejecutada en el servidor `aragorn`.
  - La tarea se entrega vía AULA dentro del plazo establecido: **miércoles 06 de junio, 23:55hrs**.
  - Para la entrega envíe **un solo archivo** (`.zip`, `.rar`, `.tar.gz`, etc.) que contenga sus archivos `.c` y `.h` utilizados más un archivo `README.txt`, donde se especifique qué es cada archivo de su programa, cómo debe ser compilado y cómo debe ser ejecutado. NO OLVIDE explicar cualquier particularidad que pueda tener su programa.