UNIVERSIDAD CATÓLICA ANDRÉS BELLO
CIUDAD GUAYANA
FACULTAD DE CIENCIAS Y TECNOLOGÍA
INGENIERÍA INFORMÁTICA
ALGORITMOS Y PROGRAMACIÓN II

FORERO, MARÍA PAULA
FREITAS, VÍCTOR
MORALES, CRISTINA
RODRÍGUEZ, JOSÉ ANDRÉS

# PROYECTO 2:
# DICCIONARIO DE SINÓNIMOS Y ANTÓNIMOS

DOCENTE: LÁREZ, JESÚS

CIUDAD GUAYANA, FEBRERO DE 2022

## ANÁLISIS DEL PROBLEMA

El presente proyecto es un diccionario de sinónimos y antónimos utilizando árboles Trie. Este contiene una entrada, la cual es llamada “input”, y funcionará para recibir los datos ingresados por el usuario. Una vez pedida la entrada se procederá a evaluar el comando indicado.

Esto servirá para analizar qué funciones se deben implementar para cumplir con él. Finalmente, dependiendo del comando, se mostrará por pantalla los sinónimos o antónimos pedidos por el usuario.

El diccionario recibe argumentos para implementar:

- Cargar nombre: para cargar el diccionaio desde el archivo nombre.dic .
- s palabra: para buscar los sinónimos de la palabra.
- a palabra: para buscar los antónimos de la palabra.
- e expresión: para mostrar los sinónimos y antónimos de todas las palabras que comienzan por esa expresión.
- Ayuda: para mostrar los comandos.
- Salir: para salir de la aplicación.

## DISEÑO DE LA SOLUCIÓN

### Algoritmo de la librería “libreria.h”:

```
//Definimos las estructuras

typedef struct Diccionario { //Estructura del diccionario
    struct SoA *A
    struct SoA *S
    struct Diccionario *hijo[26]
    int fdp
    char letra
} Dic

typedef struct SoA{  //Estructura sinónimos y antónimos
  struct SoA *next
  char palabra[]
}SoA

//Inicio de las funciones

SoA *crear_SoA(char *palabra){  //Para crear los sinónimos y antónimos
  SoA* nodo= (SoA*)calloc(1, sizeof(SoA))
  strcpy(nodo->palabra,palabra)
  nodo->next=NULL
  return nodo
}

Función Dic* crear_Dic(char letra) {   //Para crear el árbol Trie
    Dic* node = (Dic*)calloc(1, sizeof(Dic))
    Para (int i=0; i<26; i++){
        node->hijo[i] = NULL
    Fin_para}
    node->fdp = 0

    Si (letra >= 'A' && letra <= 'Z')
        letra = letra - 'A' + 'a'
    Fin_si
    node->letra=letra
    return node
Fin_funcion}
```

```
//Para insertar las palabras en el árbol
Función Dic *insertar_Dic(Dic* raiz, char* palabra,char* palabra2, char tipo){
   Dic* temp = raiz
   int idx=0

   Para (int i=0; palabra[i] != '\0'; i++){

      Si (palabra[i]<'a'){
         idx = (int) palabra[i] - 'A'
      }Sino{
         idx = (int) palabra[i] - 'a'
      Fin_si}
      Si (temp->hijo[idx] == NULL) {
          temp->hijo[idx] = crear_Dic(palabra[i])
       Fin_si}
       temp = temp->hijo[idx]
   Fin_para}

   temp->fdp = 1
   Si (tipo == 'S'){ //Para asignarle los nodos de sinónimos a la palabra
     SoA *newp=crear_SoA(&palabra2[1])
     newp->next=temp->S
     temp->S=newp
   Fin_si}

   Si (tipo == 'A'){ //Para asignarle los nodos de antónimos a la palabra
     SoA *newp=crear_SoA(&palabra2[1])
     newp->next=temp->A
     temp->A=newp
   Fin_si}
   return raiz
Fin_funcion}

//Para buscar la palabra en el árbol
Función Dic *buscar_palabra(Dic* raiz, char *palabra){
   Dic* temp = raiz
   int idx=0

   Para (int i=0; palabra[i+1]!='\0'; i++){
     Si (palabra[i]<'a'){
         idx = (int) palabra[i] - 'A'
```

```
      }Sino{
         idx = (int) palabra[i] - 'a'
       Fin_si}
       Si (temp->hijo[idx] == NULL){
          return NULL
       Fin_si}
       temp = temp->hijo[idx]
    Fin_para}
    Si (temp != NULL && temp->fdp == 1){
       return temp
    Fin_si}
    return NULL
Fin_funcion}

//Para comprobar la expresión regular
Función comprobar(char *expresion, char *palabra){
    regex_t regex
    int value,value2,bol=0

    value = regcomp(&regex,expresion,REG_EXTENDED)
    value2 =regexec(&regex,palabra,0,NULL,0)
    Si (value2==0){
       bol=1
    }Sino{
    Si (value2== REG_NOMATCH){
       bol=0
    Fin_si}

    return bol
Fin_funcion}

//Para imprimir los sinónimos
Función printsinonimos(Dic *nodo){
    SoA* head
    head=nodo->S
    Si (nodo != NULL){
       Para(;nodo->S != NULL;nodo->S=nodo->S->next){
         char *aux=nodo->S->palabra
         int lenght = strlen(aux)
         Si (aux[lenght-1]=='\n'){
            aux[lenght-1] = '\0'
```

```
        Fin_si}
        Escribir("%s",aux)
        Si (nodo->S->next !=NULL){
          Escribir(", ")
         Fin_si}
      Fin_para}
      nodo->S=head
      Escribir("\n")
   Fin_si}
Fin_funcion}

//Para imprimir los antónimos
Función printantonimos(Dic *nodo){
   SoA* head
   head=nodo->A
   Si (nodo != NULL){
     Para(;nodo->A != NULL;nodo->A=nodo->A->next){
       char *aux=nodo->A->palabra
       int lenght = strlen(aux)
       Si (aux[lenght-1]=='\n'){
         aux[lenght-1] = '\0'
       Fin_si}
       Escribir("%s",aux)
       Si (nodo->A->next !=NULL){
         Escribir(", ")
       Fin_si}
     Fin_para}
     nodo->A=head
     Escribir("\n")
   Fin_si}
Fin_funcion}

//Para buscar los sinónimos en el árbol
Función sinonimos(char *palabra, Dic *raiz){
   Dic* temp=raiz
   Dic* nodo=buscar_palabra(temp,palabra)
   printsinonimos(nodo)
Fin_funcion}

//Para buscar los antónimos en el árbol
Función antonimos(char *palabra, Dic *raiz){
```

```
   Dic* temp=raiz
   Dic* nodo=buscar_palabra(temp,palabra)
   printantonimos(nodo)
Fin_funcion}

//Para buscar las palabras que cumplan con la expresión ingresada
Función expresiones(Dic *raiz, char *palabra, int lenght, char *expresion){
   Si (raiz == NULL)
      return
   Fin_si
   strncat(palabra, &raiz->letra, 1)
   lenght++
   palabra[lenght] ='\0'
   Si (raiz->fdp){
      palabra[lenght-1] = '\n'
      Si (comprobar(expresion, palabra)) {
         Escribir("%c%s\n'", toupper(palabra[0], &palabra[1])
         Escribir("s: ")
         printsinonimos(raiz)
         Escribir("a: ")
         printantonimos(raiz)
         Escribir("\n")
      Fin_si}
   Fin_si}

   Para (int i=0; i<26; i++){
      expresiones( raiz->hijo[i], palabra, lenght, expresion )
      palabra[lenght-1] = '\0'
  Fin_para}

   return
Fin_funcion}
```

Algoritmo del programa “main.c”:

```
//ESTRUCTURA DEL MAIN

   Dic *raiz= crear_Dic('\0')
   int flag=0,pos,i,j
   char input[30],cadena[80],palabra[30],palabra2[30],tipo,*nombre
   FILE *f
   f=fopen(“nombre.dic”,"rt")  //Se abre en modo lectura el archivo del diccionario
         Mientras (!feof(f)){
            Si (!feof(f)){
               fgets(cadena,80,f)
               tipo=cadena[0]
               Para(i=2; cadena[i] != ' '; i++){
                     strncat(palabra, &cadena[i], 1)
                Fin_para}
               palabra[i] = '\0'
               strcpy(palabra2, &cadena[i++])
               insertar_Dic(raiz,palabra,palabra2,tipo)
               memset(palabra, 0, 30)
            Fin_si}
         Fin_mientras}
      fclose(f)

      Si (argc>1) flag=1 // 0=Menú interactivo // 1=Operando desde el terminal//
   Si (!flag) Escribir ("Bienvenido al programa, ingrese el comando (si no conoce
los comandos escriba 'ayuda')\n")
   Fin_si

   Si (argv[1]==NULL){
      argv[1]=" "
   Fin_si}

  // Bucle de input de usuario
   Mientras (1){
      Si (¡flag){
         Escribir(">")
         fgets(input,30,stdin)
      Fin_si

//Cargar el archivo manualmente
         Si (strstr(input,"cargar ") != NULL){
            Dic *aux = crear_Dic('\0')
            raiz = aux
            nombre=&input[7]
            nombre[strlen(nombre)-1] = '\0'
```

```
        f=fopen(nombre,"rt")
        Mientras (!feof(f)){
            Si (!feof(f)){
                fgets(cadena,80,f)
                tipo=cadena[0]
                Para(i=2; cadena[i] != ' '; i++){
                      strncat(palabra, &cadena[i], 1)
                Fin_para}
                palabra[i] = '\0'
                strcpy(palabra2, &cadena[i++])
                insertar_Dic(raiz,palabra,palabra2,tipo
                memset(palabra, 0, 30);
            Fin_si}
        Fin_mientras}
        fclose(f)

  //Para sinónimos
    }Sino Si (input[0] == 's' && input[1] == ' ' || strcmp(argv[1], "s")==0){
        Si (!flag) sinonimos(&input[2],raiz)
        Sino{
            strcpy(input,argv[2])
            strncat(input, " ", 2)
            sinonimos(input, raiz)
            return 0
        Fin_si}
        Escribir ("\n")

  //Para antónimos
    }Sino Si (input[0] == 'a' && input[1] == ' ' || strcmp(argv[1], "a")==0){
        Si (!flag) antonimos(&input[2],raiz)
        Sino{
            strcpy(input,argv[2])
            strncat(input, " ", 2)
            antonimos(input, raiz)
            return 0
        Fin_si}
        Escribir ("\n")

//Para las expresiones
    }Sino Si (input[0] == 'e' && input[1] == ' ' || strcmp(argv[1], "e")==0){
        memset(palabra, 0, 30)
        Si(!flag){
            input[strlen(input)-1]='\0'
            expresiones(raiz,palabra,0,&input[2])
        }Sino{
            strcpy(input, argv[2])
            strncat(input, " ", 2)
```

```
            input[strlen(input)-1]='\0'
            expresiones(raiz,palabra,0,input)
            return 0
        }

    //Para el comando ayuda
        }Sino Si (strcmp(input,"ayuda")==10){
            Escribir ("\nlas entradas posibles son: \n\n")
            Escribir ("cargar nombre - carga el diccionario desde el archivo
nombre.dic\n")
            Escribir ("s palabra – busca los sinónimos de la palabra ingresada\n")
            Escribir ("a palabra – busca los antónimos de la palabra ingresada\n")
            Escribir ("e expresión – muestras los sinónimos y antónimos de todas las
palabras que comienza con expresión\n")
            Escribir ("salir – sale de la aplicación\n\n")

    //Para salir del programa
        }Sino Si (strcmp(input,"salir")==10){
            return 0
        Fin_si}

    Fin_mientras}
    return 0
```

## DETALLES DE LA IMPLEMENTACIÓN

Detalles de la implementación en la librería “libreria.c”:

Se define al inicio la estructura “Diccionario” para la creación del árbol Trie, y la estructura “SoA” para los sinónimos y antónimos de las palabras.

Función crear_SoA: Para crear las listas enlazadas de sinónimos y antónimos.

Función crear_Dic: Para crear las listas enlazadas de los nodos que corresponden a cada una de las letras del abecedario.

Función insertar_Dic: Para cargar las palabras en el árbol Trie, recorriendo los nodos de las letras para formarla. También carga los sinónimos y antónimos de cada palabra de igual forma.

Función bucar_palabra: Para recorrer el árbol y sus nodos hasta encontrar la palabra requerida.

Función comprobar: Para validar la expresión regular ingresada por el usuario.

Función printsinonimos: Para recorrer los nodos, concatenar las letras, almacenar los valores de los sinónimos encontrados e imprimirlos por pantalla.

Función printantonimos: Para recorrer los nodos, concatenar las letras, almacenar los valores de los antónimos encontrados e imprimirlos por pantalla.

Función sinonimos: Para recorrer el árbol en busca de la palabra ingresada por el usuario, y mandar a imprimir los sinónimos.

Función antonimos: Para recorrer el árbol en busca de la palabra ingresada por el usuario, y mandar a imprimir los antónimos.

Función expresiones: Para verificar las palabras del árbol que cumplan con la expresión regular ingresada, y mandar a imprimir por pantalla los sinónimos y antónimos de las mismas.

## Detalles de la implementación en el programa “main.c”:

Se hará uso de las siguientes librerías: “stdio.h”, “stdlib.h”, “string.h”, “regex.h”, “ctype.h” y “libreria.h”.

Modo de operación en terminal UNIX: Se crea una bandera, donde si su valor es 0 se usará el menú interactivo y si es 1 se operará desde la terminal.

Bucle de input del usuario: Se declara una variable tipo char llamada “input”, la cual obtendrá el valor que ingrese el usuario por pantalla.

Si el valor de input es “cargar”, se abrirá en modo lectura el archivo de extensión “.dic” ingresado por el usuario, del cual se tomarán las palabras y sus sinónimos y antónimos.

Si el valor de input es “s”, se utilizará la función “sinonimos” de la librería “libreria.h”.

Si el valor de input es “a”, se utilizará la función “antonimos” de la librería “libreria.h”.

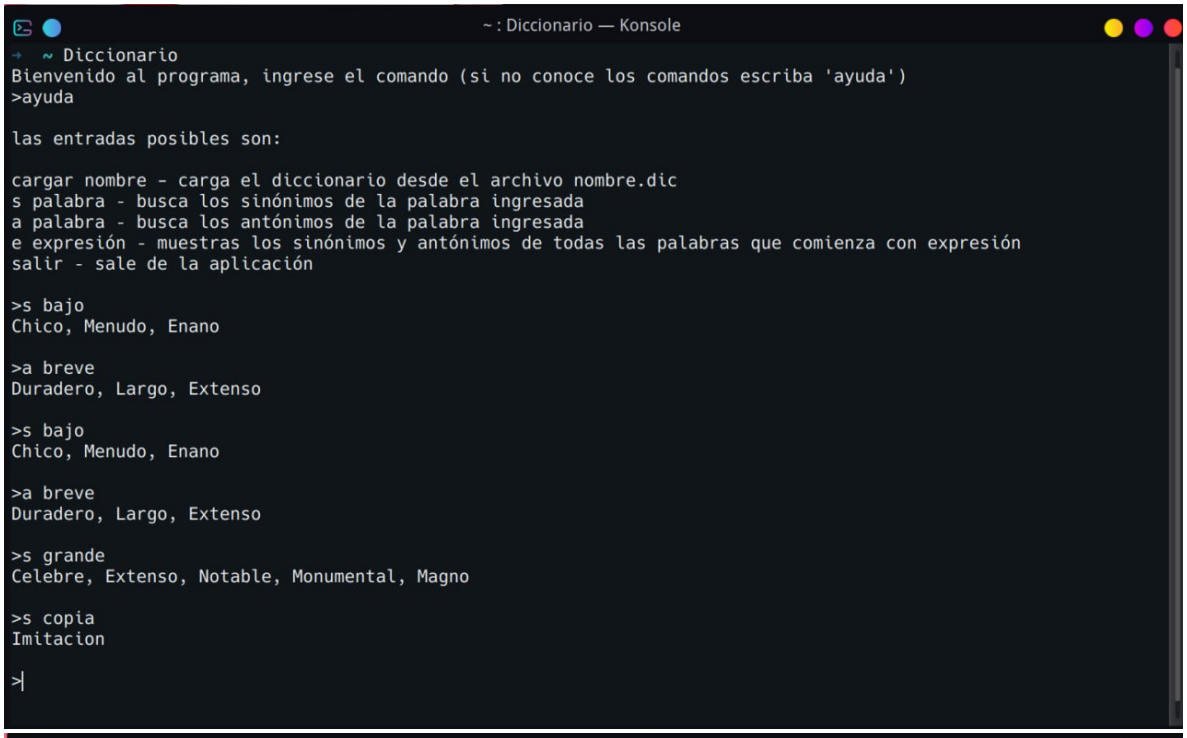
Si el valor de input es “e”, se utilizará la función “expresiones” de la librería “librería.h”.

Si el valor de input es “ayuda”, se imprimirán por pantalla todos los comandos posibles que puede ingresar el usuario, junto con sus usos.

Si el valor de input es “salir”, se detendrá la ejecución del programa.

## CORRIDAS DE EJEMPLOS

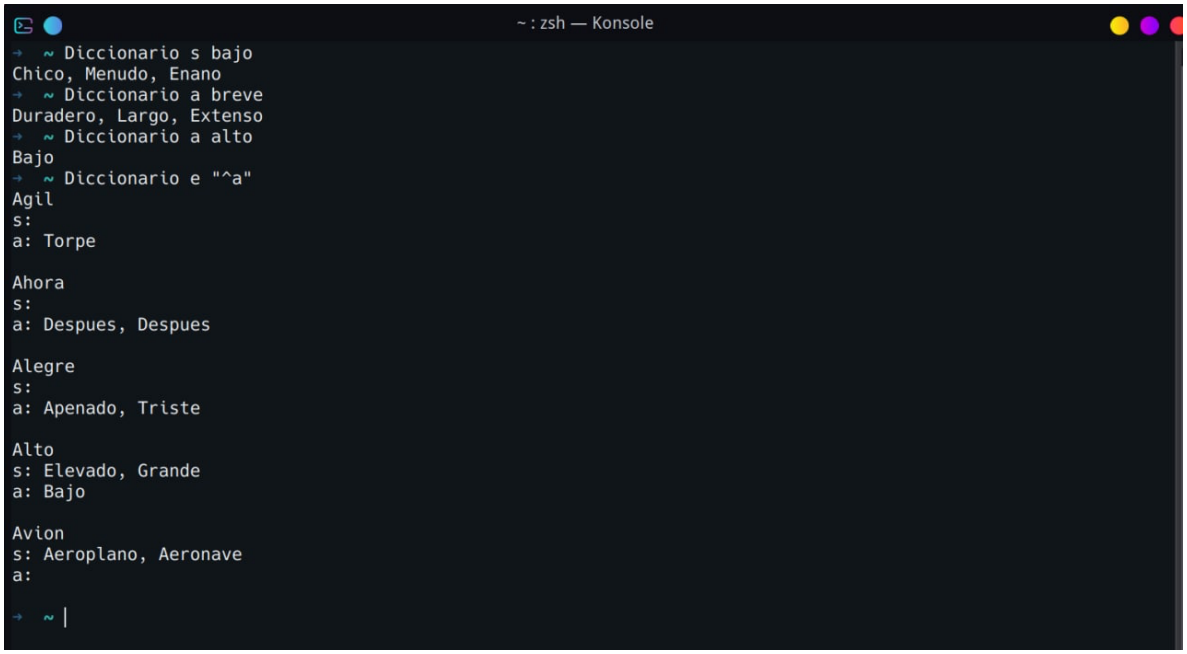
Modo interactivo desde el intérprete de comandos del Sistema de Linux:

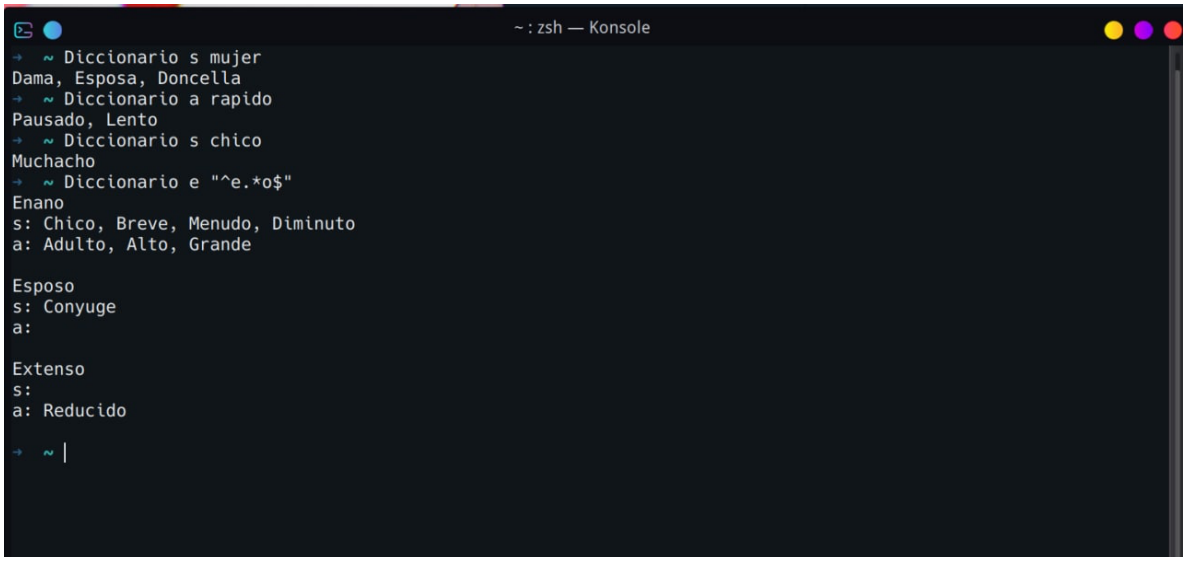




~ : Diccionario — Konsole
~ Diccionario
Bienvenido al programa, ingrese el comando (si no conoce los comandos escriba 'ayuda')
>e [b]
Bajo
s: Chico, Menudo, Enano
a:
Barco
s: Buque, Embarcacion, Navio
a:
Breve
s:
a: Duradero, Largo, Extenso
Bueno
s:
a: Malo
Doble
s: Copia
a:
Hombre
s: Individuo, Humano, Varon, Marido
a:
Pibe
s: Crio, Enano
a:
>

## Uso desde la línea de comando de Linux:



## PROGRAMA FUENTE

Código fuente de la librería “libreria.h”:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <regex.h>

//DEFINIMOS LAS ESTRUCTURAS
typedef struct Diccionario {
    struct SoA *A;
    struct SoA *S;
    struct Diccionario *hijo[26];
    int fdp;
    char letra;
} Dic;

typedef struct SoA{
    struct SoA *next;
    char palabra[];
}SoA;

//FUNCIONES DE LA LIBRERIA//

//Funcion crear los nodos de sinonimos y antonimos
SoA *crear_SoA(char *palabra){
    SoA* nodo= (SoA*)calloc(1, sizeof(SoA));
    strcpy(nodo->palabra,palabra);
    nodo->next=NULL;
    return nodo;
}

//Funcion crear los nodos del arbol Trie
Dic* crear_Dic(char letra) {
    Dic* node = (Dic*)calloc(1, sizeof(Dic));
```

```
    for (int i=0; i<26; i++){
        node->hijo[i] = NULL;
    }
    node->fdp = 0;
    if (letra >= 'A' && letra <= 'Z')
        letra = letra - 'A' + 'a';

    node->letra=letra;
    return node;
}

//Funcion insertar las palabras en el arbol
Dic *insertar_Dic(Dic* raiz, char* palabra,char* palabra2, char tipo){
    Dic* temp = raiz;
    int idx=0;

    for (int i=0; palabra[i] != '\0'; i++){
        if (palabra[i]<'a'){
            idx = (int) palabra[i] - 'A';
        }else
            idx = (int) palabra[i] - 'a';

        if (temp->hijo[idx] == NULL) {
            temp->hijo[idx] = crear_Dic(palabra[i]);
        }
        temp = temp->hijo[idx];
    }
    temp->fdp = 1;
    if (tipo == 'S'){
        SoA *newp=crear_SoA(&palabra2[1]);
        newp->next=temp->S;
        temp->S=newp;
    }
    if(tipo == 'A'){
        SoA *newp=crear_SoA(&palabra2[1]);
        newp->next=temp->A;
```

```
        temp->A=newp;
    }
    return raiz;
}

//Funcion buscar palabras en el arbol
Dic *buscar_palabra(Dic* raiz, char *palabra){
    Dic* temp = raiz;
    int idx=0;

    for(int i=0; palabra[i+1]!='\0'; i++){
        if (palabra[i]<'a'){
            idx = (int) palabra[i] - 'A';
        }else
            idx = (int) palabra[i] - 'a';

        if (temp->hijo[idx] == NULL)
            return NULL;

        temp = temp->hijo[idx];
    }
    if (temp != NULL && temp->fdp == 1)
        return temp;
    return NULL;
}

//Funcion comprobar la expresion regular
int comprobar(char *expresion, char *palabra){
    regex_t regex;
    int value,value2,bol=0;

    value = regcomp(&regex,expresion,REG_EXTENDED);
    value2 =regexec(&regex,palabra,0,NULL,0);
    if (value2==0){
        bol=1;
    }
```

```
    else
    if (value2== REG_NOMATCH){
        bol=0;
    }
    return bol;
}

//Funcion imprimir sinonimos
void printsinonimos(Dic *nodo){
    SoA* head;
    head=nodo->S;
    if (nodo != NULL){
        for(;nodo->S != NULL;nodo->S=nodo->S->next){
            char *aux=nodo->S->palabra;
            int lenght = strlen(aux);
            if(aux[lenght-1]=='\n')
                aux[lenght-1] = '\0';

            printf("%s",aux);
            if (nodo->S->next !=NULL)
                printf(", ");
        }
        nodo->S=head;
        printf("\n");
    }
}

//Funcion imprimir antonimos
void printantonimos(Dic *nodo){
    SoA* head;
    head=nodo->A;
    if (nodo != NULL){
        for(;nodo->A != NULL;nodo->A=nodo->A->next){
            char *aux=nodo->A->palabra;
            int lenght = strlen(aux);
            if(aux[lenght-1]=='\n')
```

```
            aux[lenght-1] = '\0';

        printf("%s",aux);
        if (nodo->A->next !=NULL)
            printf(", ");
    }
    nodo->A=head;
    printf("\n");
  }
}

//Funcion buscar sinonimos
void sinonimos(char *palabra, Dic *raiz){
  Dic* temp=raiz;
  Dic* nodo=buscar_palabra(temp,palabra);
  printsinonimos(nodo);
}

//Funcion buscar antonimos
void antonimos(char *palabra, Dic *raiz){
  Dic* temp=raiz;
  Dic* nodo=buscar_palabra(temp,palabra);
  printantonimos(nodo);
}

//Funcion expresiones regulares
void expresiones(Dic *raiz, char *palabra, int lenght, char *expresion){
  if (raiz == NULL) return;
  char* imprimir;
  strncat(palabra, &raiz->letra, 1);
  lenght++;
  palabra[lenght] ='\0';
  if (raiz->fdp){
    if (comprobar(expresion, palabra)) {
      printf("%c%s\n", toupper(palabra[0]), &palabra[1]);
      printf("s: ");
```

```
            printsinonimos(raiz);
            printf("a: ");
            printantonimos(raiz);
            printf("\n");
        }
    }
    for (int i=0; i<26; i++){
        expresiones( raiz->hijo[i], palabra, lenght, expresion );
        palabra[lenght-1] = '\0';
    }
    return;
}
```

Código fuente del programa “main.c”:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <regex.h>
#include <ctype.h>
#include "libreria.h"

int main(int argc, char *argv[]){

    Dic *raiz= crear_Dic('\0');
    int flag=0,pos,i,j;
    char input[30],cadena[80],palabra[30],palabra2[30],tipo,*nombre;
    FILE *f;
    f=fopen("nombre.dic","rt"); //Se abre en modo lectura el archivo del
diccionario
        while (!feof(f)){
            if (!feof(f)){
                fgets(cadena,80,f);
                tipo=cadena[0];
                    for(i=2; cadena[i] != ' '; i++){
                        strncat(palabra, &cadena[i], 1);
                    }
                palabra[i] = '\0';
                strcpy(palabra2, &cadena[i++]);
                insertar_Dic(raiz,palabra,palabra2,tipo);
                memset(palabra, 0, 30);
            }
        }
    fclose(f);
    if(argc>1) flag=1; // 0=Menú interactivo // 1=Operando desde el terminal//
    if(!flag) printf("Bienvenido al programa, ingrese el comando (si no conoce
los comandos escriba 'ayuda')\n");

    if(argv[1]==NULL){
        argv[1]=" ";
    }

    // Bucle de input de usuario
    while(1){
```

```
if(!flag){
    printf(">");
    fgets(input,30,stdin);
}

//Cargar el archivo manualmente
if(strstr(input,"cargar ") != NULL){
    Dic *aux = crear_Dic('\0');
    raiz = aux;
    nombre=&input[7];
    nombre[strlen(nombre)-1] = '\0';
    f=fopen(nombre,"rt");
    while (!feof(f)){
        if (!feof(f)){
            fgets(cadena,80,f);
            tipo=cadena[0];
            for(i=2; cadena[i] != ' '; i++){
                  strncat(palabra, &cadena[i], 1);
            }
            palabra[i] = '\0';
            strcpy(palabra2, &cadena[i++]);
            insertar_Dic(raiz,palabra,palabra2,tipo);
            memset(palabra, 0, 30);
        }
    }
    fclose(f);

//Para sinónimos
}else if(input[0] == 's' && input[1] == ' ' || strcmp(argv[1], "s")==0){
    if(!flag) sinonimos(&input[2],raiz);
    else{
        strcpy(input,argv[2]);
        strncat(input, " ", 2);
        sinonimos(input, raiz);
        return 0;
    }
    printf("\n");

//Para antónimos
}else if(input[0] == 'a' && input[1] == ' ' || strcmp(argv[1], "a")==0){
    if(!flag) antonimos(&input[2],raiz);
```

```
            else{
                strcpy(input,argv[2]);
                strncat(input, " ", 2);
                antonimos(input, raiz);
                return 0;
            }
            printf("\n");

        //Para las expresiones
        }else if(input[0] == 'e' && input[1] == ' '){
            memset(palabra, 0, 30);
            input[strlen(input)-1]='\0';
            expresiones(raiz,palabra,0,&input[2]);

        //Para el comando ayuda
        }else if (strcmp(input,"ayuda")==10){
            printf("\nlas entradas posibles son: \n\n");
            printf("cargar nombre - carga el diccionario desde el archivo
nombre.dic\n");
            printf("s palabra – busca los sinónimos de la palabra ingresada\n");
            printf("a palabra – busca los antónimos de la palabra ingresada\n");
            printf("e expresión – muestras los sinónimos y antónimos de todas
las palabras que comienza con expresión\n");
            printf("salir – sale de la aplicación\n\n");

        //Para salir del programa
        }else if (strcmp(input,"salir")==10){
            return 0;
        }
    }
    return 0;
}
```

## CÓDIGO DEL MAKEFILE

```
CC=gcc
CFLAGS= -g -lm
Diccionario: main.o libreria.o
	$(CC) $(CFLAGS) -o Diccionario main.o

main.o: main.c libreria.h
	$(CC) $(CFLAGS) -c main.c

libreria.o: libreria.c libreria.h
	$(CC) $(CFLAGS) -c libreria.c

clean:
	rm -rf *.o
```

## ENLACE AL REPOSITORIO

https://github.com/Noralgorithm/AyP_PR0Y3CT0_2