Pregunta 1

Un diccionario representado como un árbol binario de búsqueda está almacenado en un archivo binario (no es un archivo de texto). Cada nodo del árbol binario está almacenado en una cantidad variable de bytes contiguos en el archivo en el siguiente formato:

campo:	*izq*	*der*	*kl*	*vl*	*key*	*val*
	312	123	5	17	perro	mascota lamebotas
tamaño en bytes:	*4*	*4*	*1*	*1*	*kl (5)*	*vl (17)*

Los primeros 4 bytes corresponden a un entero en binario que almacena la posición en el archivo del nodo hijo izquierdo del árbol binario. Contiene 0 si el subárbol izquierdo está vacío. Los siguientes 4 bytes corresponden a la posición del nodo derecho. Luego viene un byte con el largo *kl* en bytes de la llave y un byte *vl* con el largo del valor asociado. Termina el nodo con los *kl* bytes de la llave seguidos de *vl* bytes del valor. El nodo raíz del arbol, si no está vacío, se encuentra al inicio del archivo. El siguiente código permite leer la raíz del archivo:

```
FILE *in= ...;
int izq, der;
unsigned char kl, vl;
fseek(in, 0, SEEK_SET);
fread(&izq, sizeof(izq), 1, in); // posicion nodo izquierdo
fread(&der, sizeof(der), 1, in); // posicion nodo derecho
fread(&kl, 1, 1, in);            // largo de la llave
fread(&vl, 1, 1, in);            // largo del valor asociado
char key[kl], val[vl];
fread(key, kl, 1, in);           // arreglo para la llave
fread(val, vl, 1, in);           // arreglo para el valor
```

En el ejemplo, para leer el nodo izquierdo debe usar un código similar posicionándose con *fseek* en 312 bytes (el valor de *izq*) desde el inicio del archivo. Cuidado porque *key* y *val* no son strings (no terminan en 0).

Programe la función: *int main(int argc, char *argv[ ])* para el comando *./buscar*. Este comando recibe como parámetros (i) el nombre de un archivo que almacena un diccionario en el formato descrito más arriba y (ii) una llave. El programa debe buscar la llave en el diccionario y mostrar el valor asociado. Ejemplos de uso:

```
$ ./buscar mascotas.bin perro
mascota lamebotas
$ ./buscar mascotas.bin gato
miaumifero peludo
```

Restricciones: No puede leer el archivo completo en memoria. Debe usar *fseek* y *fread* para leer el mínimo número de nodos necesarios para encontrar la llave en el archivo. Como es un arbol binario de búsqueda, todas las llaves del subárbol izquierdo de *n* son menores lexicográficamente que la llave en *n* y todas las del subárbol derecho son mayores. No necesita validar errores excepto el caso en el que la llave no está en el diccionario.

Pregunta 2

Parte a.- La función *f* de la derecha está programada en assembler Risc-V. Considere que se invoca *f* recibiendo en *a1* el entero 5 y en *a0* la dirección *d* de un arreglo de 4 enteros con los valores 3, 8, 1 y 0. La siguiente tabla muestra la ejecución de la función hasta la 2da ejecución de *lw*.

```
f:
   mv      a4,a0
.L1:
   lw      a5,0(a4)
   addi    a4,a4,4
   bge     a5,a1,.L2
   sw      a5,0(a0)
   addi    a0,a0,4
.L2:
   bne     a5,zero,.L1
   ret
```

Instrucción	*a0*	*a1*	*a4*	*a5*	*arreglo d*
	d	5			3 8 1 0
`mv a4,a0`			*d*		
`lw a5,0(a4)`				3	
`addi a4,a4,4`			*d+4*		
`bge a5,a1,.L2`		no salta			
`sw a5,0(a0)`					**3** 8 1 0
`addi a0,a0,4`	*d+4*				
`bne a5,zero,.L1`		salta			
`lw a5,0(a4)`				8	

Prosiga llenando la tabla con la ejecución de *addi a4,a4,4* hasta que se ejecute la instrucción *ret*.

Parte b.- Traduzca la función de la derecha a assembler Risc-V. Optimice el código en assembler para reducir la cantidad de instrucciones.

```
int g(int *a, int n, int x) {
  int i=0, j=n-1, h=(i+j+1)/2;
  while (i<=j && a[h]!=x) {
    if (a[h]<x) i= h+1;
    else j= h-1;
    h= (i+j+1)/2;
  }
  return i<=j ? 1 : 0;
}
```