Pregunta 1

Programe la siguiente función: *unsigned int* ***max01****(unsigned int x);*

Esta función entrega en los 6 bits menos significativos del valor retornado el máximo número de bits consecutivos en 0 encontrados en *x*, y en los 26 bits más significativos el máximo número de bits en 1 consecutivos encontrados en *x*. Ejemplo de uso:

```
unsigned int r= max01(0b00110011000011001111100011110001);
int max0s=r&0b111111, max1s=r>> 6; // max0s=4 y max1s=5
```

Restricciones: Ud. no puede usar los operadores de multiplicación, división o módulo (* / %). Use eficientemente los operadores de bits.

Metodología obligatoria: Desplace sucesivamente *x* en 1 bit hacia la izquierda y calcule el *y bit a bit* de todos esos valores. Observe en el ejemplo de la derecha (con *x*=00111011 y enteros de 8 bits) que después de 3 desplazamientos el resultado del *y bit a bit* es 0. Por lo tanto el número máximo de 1s es 3. Siempre será el número de desplazamientos requeridos para que el *y* sea 0. ¡Créame! Para el máximo de 0s, proceda de manera similar solo que use el *o bit a bit* y al desplazar rellene con un 1 a la derecha. Termine cuando el *o* sea -1 (0xffffffff).

```
  00111011
  01110110
  11101100
  11011000
& 00000000
```

Pregunta 2

Programe la siguiente función: *void* ***espaciar****(char *str);*

Esta función debe agregar un espacio después de cada caracter en *str*. El siguiente es un ejemplo de uso:

```
char s[9]= "hola";
espaciar(s); // s es "h o l a "
```

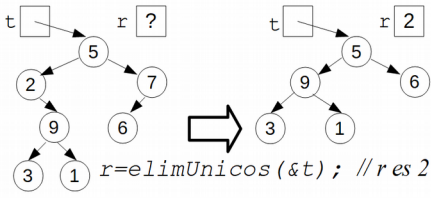
Restricciones: No use el operador de subindicación de arreglos [ ] ni su equivalente **(p+i)*, use aritmética de punteros. No puede pedir memoria adicional usando *malloc* o declarando un arreglo de caracteres.

Ayuda: Para hacer el espaciado recorra el string de atrás hacia adelante. Observe que el resultado debe quedar en la misma área de memoria que recibe como parámetro. Esta tiene el tamaño justo para el resultado.

Pregunta 3

Programe la función *elimUnicos* con el encabezado del cuadro de la derecha. Esta función recibe en **pa* un árbol binario (no ordenado) y elimina todos los nodos que tengan un solo hijo. Debe retornar el número de nodos que fueron eliminados. En el ejemplo de la figura de la derecha el tipo de *t* es *Nodo** y de *r* es *int*. El nodo con etiqueta 2 es eliminado porque su único hijo es el nodo de etiqueta 9. Por la misma razón se elimina el nodo de etiqueta 7. Los nodos 3, 1 y 6 no se eliminan porque no tienen hijos.

```
typedef struct nodo {
  int x;
  struct nodo *izq, *der;
} Nodo;
int elimUnicos(Nodo **pa);
```



Metodología obligatoria: Sea *a= *pa*. El caso en que *a* es el árbol vacío es trivial. Elimine recursivamente los nodos que tienen un solo hijo del subárbol izquierdo y del subárbol derecho. Si ahora *a* tiene un solo nodo hijo, haga que **pa* apunte hacia ese único nodo y libere la memoria ocupada por el nodo *a*. Calcule Ud. eficientemente el número de nodos eliminados.

Pregunta 4

Escriba el programa *rotar.c* que recibe como parámetros el nombre de un archivo y una secuencia de números de líneas l_1, l_2, l_3, … l_n. El programa debe mover la línea l_1 a la línea l_2, la línea l_2 a l_3, … y finalmente l_n a l_1. Ejemplo de uso:

línea	*archivo noms.txt*	*línea*	*Resultado de: ./rotar noms.txt 2 5 0 4*	
0	pedro	0	patricia	Observe que la línea 2, diego, pasa a la línea 5, en donde estaba patricia, que pasa a la línea 0, en donde estaba pedro, que pasa a la línea 4, en donde estaba francia, que pasa a la línea 2.
1	juan	1	juan	
2	diego	2	francia	
3	ana	3	ana	
4	francia	4	pedro	
5	patricia	5	diego	

Todas las líneas son de 10 caracteres (9 caracteres más el fin de línea '\n'). Debe usar *fseek* para seleccionar una línea y *fread* para leerla o *fwrite* para escribirla. No puede leer todo el archivo en memoria ni leer todas las líneas. Note que debe programar la función *main*. En el archivo adjunto *rotar.c.plantilla* hay ayuda para obtener los parámetros de *argc* y *argv*.

Pregunta 5

Programe la siguiente función: *int* ***buscarPar****(Nodo *a, int z, int p);*

Esta función busca en paralelo usando *p* threads el entero *z* en el árbol binario *a* no ordenado. La estructura *Nodo* es la misma de la pregunta 3. Esta función debe entregar 1 si existe un nodo en el árbol *a* cuya etiqueta *x* es igual a *z*. 0 en caso contrario. La búsqueda tiene que ser exhaustiva porque el árbol no es un árbol de búsqueda binaria. Por ejemplo si se busca 1 en el árbol del ejemplo de la pregunta 3, el resultado debe ser 1, pero si se busca 8, el resultado es 0.

Use la siguiente metodología: El caso en que la búsqueda concluye en la raíz de *a* es trivial. De lo contrario, si *p* es 1, busque secuencialmente *z* en *a* invocando *buscar(a, z),* que está definida en *test-p5.c* en los archivos adjuntos. Para el caso en que *p>1*, su solución debe crear un nuevo thread en el que debe buscar *z* recursivamente en el subárbol derecho de *a* usando *p/2* threads. En el thread original debe buscar *z* recursivamente en el subárbol izquierdo de *a* usando *p-p/2* threads. Suponga que el árbol está razonablemente equilibrado. Al final se habrán utilizado *p* threads para hacer la búsqueda (incluyendo el thread original).