Pregunta 1

Programe la función: *void* ***porciento****(char *s)*. Esta función reemplaza en *s* todas las secuencias de caracteres *o/o* por *%*. Ejemplo de uso:

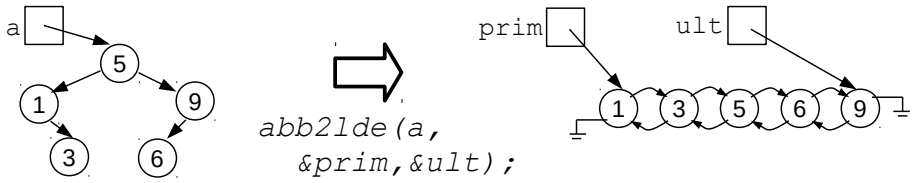
```
char s[] = "el 10o/o del 10o/o es 1o/o o/";   // de largo 29
porciento(s); // s es "el 10% del 10% es 1% o/" de largo 23
```

Restricciones: No puede usar las funciones de manejo de strings como *strcmp*, *strncmp*, *strcpy*, *strlen*, etc. No use el operador de subindicación de arreglos [ ] ni su equivalente **(p+i)*, use aritmética de punteros. No puede pedir memoria adicional con *malloc* ni declarar arreglos. Necesitará usar punteros adicionales.

Pregunta 2

```
typedef struct nodo {
  int x;
  struct nodo *izq, *der;
} Nodo;
void abb2lde(Nodo *a,
  Nodo *pprim, Nodo **pult);
```

Programe la función ***abb2lde*** que transforma el árbol binario de búsqueda *a* no vacío en una lista doblemente enlazada ordenada, entregando en **pprim* la dirección del primer nodo de la lista y en **pult* la dirección del último nodo. El encabezado de *abb2lde* y tipo de cada nodo es el de arriba. ***La transformación debe tomar tiempo O(n)***, siendo *n* la cantidad de nodos de *a*. En el siguiente ejemplo de uso, el tipo de *a*, *prim* y *ult* es *Nodo**:

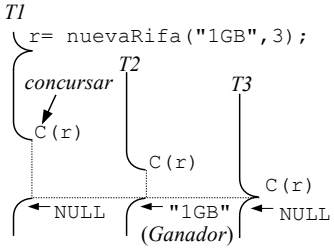


Metodología obligatoria: No puede usar *malloc*. Debe reutilizar los nodos del árbol *a*. El campo *izq* pasa a ser el nodo previo en la lista enlazada y *der* el nodo siguiente. Ignore el caso en que *a* es nulo. Si el subárbol izquierdo de *a* es nulo, el primer nodo de la lista es *a*. Si no es nulo, transforme recursivamente el subárbol izquierdo, obteniendo una lista parcial con nodos desde *primIzq* hasta *ultIzq*. Si el subárbol derecho de *a* es nulo, el último nodo de la lista es *a*. Si no es nulo, transfórmelo recursivamente obteniendo otra lista parcial con nodos que van de *primDer* a *ultDer*. Use astutamente *primIzq*, *ultIzq*, *primDer* y *ultDer* para enlazar la lista completa en tiempo constante: primero debe ir la lista parcial izquierda, luego el nodo *a* y finalmente la lista parcial derecha. No olvide asignar **pprim* y **pult*.

Pregunta 3

```
Rifa *nuevaRifa(void *premio,
                int p);
void *concursar(Rifa *r);
void destruirRifa(Rifa *r);
```

Múltiples threads concursan por ganar el premio en una rifa *r*. Ud. debe definir el tipo *Rifa* y programar las funciones de la derecha. Cuando un thread invoca *concursar(r)*, debe esperar hasta que en total *p* threads llamen a *concursar(r)*. Cuando llega el último de los threads Ud. debe elegir aleatoriamente un ganador entre los *p* threads. En el thread ganador, *concursar* debe retornar el *premio* y en el resto de los threads debe retornar nulo. Para elegir el ganador asigne a cada thread en espera un identificador entre 0 y *p-1*. El thread ganador será el que tenga como identificador *random( ) % p*. La figura de la derecha muestra un ejemplo de uso.



Pregunta 4

La función *int menores(double a[ ], int n, double x)* es dada y recibe un arreglo *a* de tamaño *n*. Esta función mueve al inicio del arreglo *a* todos los elementos de *a* que son menores que *x*, respetando el orden en que aparecían en *a*, y retorna la cantidad de elementos menores que *x*. Está programada en el archivo *test-fork.c* en los archivos adjuntos. Este es un ejemplo de uso:

```
double a[5]= {5.2, 4.1, 7.3, 8.2, 1.6};
int k= menores(a, 5, 5.2); // k es 2, a[0] es 4.1 y a[1] es 1.6
```

Programe la función *int* ***menoresPar****(double a[ ], int n, double x)* con la misma funcionalidad pero en paralelo para un computador dual core.

Método obligatoria: Use *fork* para crear un proceso hijo que se comunica con el proceso padre por medio de un pipe. Sea *h=(n+1)/2*. En el proceso hijo determine los menores que *x* desde *a[h]* hasta *a[n-1]*. Para estos efectos se le permite invocar la función *menores*. Considere que encuentra *kh* elementos menores que *x*. Envíe por el pipe *kh* y los elementos menores encontrados. En el proceso padre determine los menores que *x* desde *a[0]* hasta *a[h-1]*. Considere que encuentra *kp* elementos menores y por lo tanto quedan almacenados en *a[0]* hasta *a[kp-1]*. Lea *kh* del pipe y reciba los menores encontrados por el hijo a partir de *a[kp]* (use la función *leer*). La cantidad de menores que *x* en todo el arreglo *a* es *kh+kp*. Este problema es similar al quicksort para computadores dual core visto en la cátedra del jueves 10 de junio.