Pregunta 1

Programe las siguientes funciones:

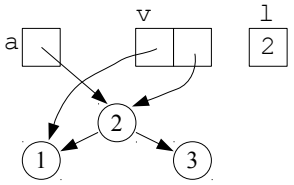
Parte a.- `int pisoMod2alaK(int n, int k)`

Esta función debe entregar el mayor entero múltiplo de 2^k que es menor o igual a *n*. Ejemplo: `pisoMod2alaK(35,4)=32`. Use los operadores de bits. *Hint:* Un entero que es múltiplo de 2^k tiene sus últimos *k* bits en 0 y por lo tanto Ud. tiene que llevar a 0 los últimos *k* bits de n. Piense en cúal es la forma binaria de 2^k-1.

Restricción: Ud. no puede usar la múltiplicación, división o resto (`* / %`).

Parte b.- `int primeros(Nodo *a, int n, Nodo *pvec[])`

Esta función entrega en *pvec* los primeros *n* nodos del árbol de búsqueda binaria *a* al recorrerlo en orden, entregando el número de nodos efectivamente recorridos. Por ejemplo considere el árbol *a* de la derecha. Se ejecuta este código:




```
Nodo *v[2];
int l= primeros(a, 2, v);
```

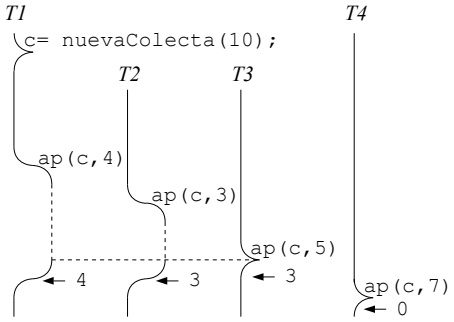
Entonces *v* y *l* deben quedar como se indica en la figura. Si el árbol tiene $m<n$ nodos, *primeros* debe retornar *m* y solo llenar *m* elementos en el arreglo. *Hint:* ¡use recursión!

Pregunta 2

Defina el tipo de datos *Colecta* y programe las siguientes funciones:

```
Colecta *nuevaColecta(double meta);
double aportar(Colecta *c, double monto);
```

La función *nuevaColecta* crea y entrega una colecta para juntar exactamente *meta* euros. La función *aportar* es invocada por múltiples threads para contribuir con *monto* euros. Esta función debe esperar hasta que la suma de los aportes alcance la meta, entregando el monto efectivo aportado. Por ejemplo si faltan 3 euros para alcanzar la meta y un thread aporta 5, entonces *aportar* retorna 3 en ese thread. Si un thread invoca *aportar* cuando la meta ya fue alcanzada, entonces se retorna 0 de inmediato. La figura de la derecha muestra un ejemplo de ejecución con múltiples threads.



Pregunta 3

Considere la siguiente función:

```
typedef double (*RealFun)(void *ptr);
int eval(RealFun f, void *ptr, double *res) {
  *res= (*f)(ptr);
  return 0;
}
```

En donde *RealFun* es el tipo de las funciones que reciben un puntero opaco como parámetro y retornan un *double*.

Se sabe que *f* está mal programada. Si se invoca varias veces con el mismo parámetro de entrada debería entregar el mismo resultado, pero en raras ocasiones falla aleatoriamente entregando un resultado incorrecto o gatillando un *segmentation fault* (señal SIGSEGV).

i.- (3 puntos) Por esta razón se le pide a Ud. reprogramar la función *eval* de tal forma que invoque 2 veces la función *f* en 2 procesos pesados. La función *eval* debe retornar 0 si ambas evaluaciones de *f* terminan exitosamente y ambas entregan el mismo resultado (en este caso se considera que el resultado es confiable). En caso contrario *eval* debe retornar -1.

Metodología: Use *fork* para crear 2 subprocesos y evalúe *f* en cada uno de ellos. Use pipes para que cada subproceso envíe el resultado al padre (un número real de 8 bytes). En el padre lea ambos pipes. Normalmente Ud. leerá 8 bytes (un double) en cada pipe, pero si un proceso termina prematuramente por SIGSEGV, leerá 0 bytes. No olvide enterrar ambos subprocesos.

ii.- (2 puntos) Modifique su solución de tal forma que se use *select* para leer el pipe del primer proceso que envíe su resultado. Si al leer ese pipe se leen 0 bytes, significa que ese proceso terminó por SIGSEGV. En ese caso mate el segundo proceso. No tiene sentido esperar que termine, pues el resultado de *eval* será -1 de todas formas.

iii.- (0,5 puntos) ¿En qué tipo de programas vistos en el curso puede ocurrir que una función que depende únicamente de su parámetro de entrada entregue resultados distintos cuando se invoca 2 veces con el mismo parámetro? No se usa la función *random*.

iv.- (0,5 puntos) Considere que *f* podría en raras ocasiones no terminar nunca con el mismo parámetro con el que normalmente sí termina. Explique en palabras como modificaría su solución para asegurarse que *eval* no se bloquee indefinidamente esperando un resultado que nunca llegará. Considere evaluar 3 veces la función.