Pregunta 1

Programe eficientemente la función *elimUltPal* con el siguiente encabezado:

*void elimUltPal(char *s);*

Esta funciona elimina la última palabra en *s* desplazando el resto del texto al final de *s*. Las palabras están delimitadas por uno o más espacios en blanco. Los siguientes son ejemplos de uso:

```
char s[] =          "hola   que tal   ";
elimUltPal(s); // s es "      hola   que "  (el gris indica lo que se
elimUltPal(s); // s es "          hola   "   borrará)
elimUltPal(s); // s es "                 "
elimUltPal(s); // s es "                 "
```

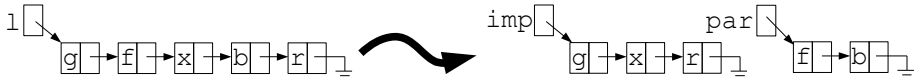
Restricción: No use el operador de subindicación de arreglos [ ] ni su equivalente **(p+i)*, use aritmética de punteros. No puede pedir memoria con *malloc* ni declarar arreglos.

Pregunta 2

Programe eficientemente la función *dividir* que descompone una lista en 2 sublistas. Debe tener el siguiente encabezado:

```
typedef struct nodo {
  char c;
  struct nodo *prox;
} Nodo;
void dividir(Nodo *lista,
             Nodo **pimpares, Nodo **ppares);
```

Esta función deja en **pimpares* una lista compuesta por el primer, tercer, quinto, etc. nodo en *lista*. En **ppares* quedan los nodos segundo, cuarto, sexto, etc. de *lista*. Ambas listas deben terminar con el nodo *NULL*. En el siguiente ejemplo de uso las variables *l*, *imp* y *par* son de tipo *Nodo**:



dividir(l, &imp, &par);

Restricción: No puede usar *malloc*. Debe reutilizar los nodos que recibe en *lista*.

Ayuda: si usa recursión puede llegar una solución con muy pocas líneas.

Pregunta 3

Los filósofos deben ocupar un baño que tiene capacidad para una sola persona. Se debe evitar que ingresen 2 filósofos simultáneamente. Antes de ingresar al baño un filósofo llama a *ocupar( )*. Esta función espera si el baño está ocupado. Al salir invoca *desocupar( )*. La siguiente implementación de estas funciones es correcta pero lamentablemente cuando hay múltiples filósofos en espera del baño, entran en desorden porque *wait* no garantiza un orden de atención específico:

```
pthread_mutex_t m= PTHREAD_MUTEX_INITIALIZER;
pthread_cond_t c= PTHREAD_COND_INITIALIZER;
int ocupado= 0;
```

```
void ocupar() {
  lock(&m);
  while (ocupado)
    wait(&c, &m);
  ocupado= 1;
  unlock(&m);
}
```

```
void desocupar() {
  lock(&m);
  ocupado= 0;
  broadcast(&c);
  unlock(&m);
}
```

Reprograme esta solución de modo que los filósofos ocupen el baño por orden de llegada. Por ejemplo si mientras se ocupa el baño llegan los filósofos F2, F4 y F1, en ese orden, al desocuparse debe entrar primero F2, después entra F4 y finalmente ingresa F1.

Ayuda: Use un protocolo similar al que usan farmacias, isapres, bancos, etc. Al llegar, un cliente toma un número. Un visor anuncia el número que se está atendiendo ahora. Cuando se termina de atender un cliente, el visor se incrementa en 1. Un cliente sabe que es su turno cuando el número del visor coincide con su propio número.

Pregunta 4

Programe la función *buscarPar* que busca en paralelo un entero en una árbol binario no ordenado. La función tiene el siguiente encabezado:

```
typedef struct nodo {
  int x;
  struct nodo *izq, *der;
} Nodo;
int buscarPar(Nodo *a, int z, int p);
```

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.

Si la búsqueda no concluye en la raíz de *a* y *p>1*, su solución debe buscar *z* recursivamente en el subárbol izquierdo de *a* usando *p/2* threads. En paralelo debe buscar *z* recursivamente en el subárbol derecho 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.