Pregunta 1

Programe la función *elimRep* con encabezado:

```
void elimRep(char *s);
```

Esta función elimina las repeticiones de caracteres consecutivos rellenando con espacios en blanco al comienzo. Ejemplo de uso:

```
char s[] = "aaa11b*bbaaaa0";   // de largo 14
elimRep(s); // s es "        a1b*ba0" de largo 14
```

Restricciones: 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.

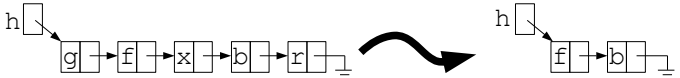
Metodología obligatoria: Use *strlen* para obtener un puntero hacia el último carácter del string *s*. Luego retroceda en el string eliminando los caracteres repetidos. Finalmente rellene el comienzo del string con espacios en blanco.

Pregunta 2

Programe eficientemente la función *mediaLista* que elimina el primer, tercer, quinto, etc. elemento de una lista simplemente enlazada. Debe tener el siguiente encabezado:

```
typedef struct nodo {
  char c;
  struct nodo *prox;
} Nodo;
void mediaLista(Nodo **ph);
```

En el siguiente ejemplo de uso el tipo de *h* es *Nodo**:



mediaLista(&h);

Requisitos: No puede usar *malloc*, debe reutilizar los nodos que recibe en la lista y debe liberar con *free* la memoria utilizada por los nodos eliminados.

Pregunta 3

Múltiples threads comparten un recurso único. Los threads se agrupan en categoría 0 y categoría 1. Un thread de categoría *cat* solicita el uso exclusivo del recurso invocando *ocupar(cat)*, y lo devuelve llamando a *desocupar()*, sin parámetros. El recurso debe ser otorgado de manera alternada: un thread de categoría 0, luego de categoría 1, etc., pero puede ser otorgado consecutivamente a threads de la misma categoría si no hay threads pendientes de la otra categoría. Los threads de la misma categoría pueden ser atendidos en cualquier orden. La siguiente solución es incorrecta pero funciona bien más del 99,9% de los casos.

```
int ocup= 0, cat_ult=1, pend[2]= {0, 0};
```

```
void ocupar(int cat) {
  pend[cat]++;
  while ( ocup || (cat==cat_ult &&
                   pend[!cat]!=0) )
    ;          // note que !0 es 1 y !1 es 0
  pend[cat]--;
  ocup= 1;
  cat_ult= cat;
}
void desocupar() {
  ocup= 0;
}
```

Reprograme este código de manera que sea correcto y eficiente.

Pregunta 4

Programe la función *buscarLlavePar* que busca la llave asociada a un valor en un árbol binario de búsqueda. La búsqueda se debe hacer en paralelo usando *p* procesos pesados, para que esta función se ejecute eficientemente en una máquina con *p* cores. La función tiene el siguiente encabezado:

```
typedef struct nodo {
  int k, v;
  struct nodo *izq, *der;
} Nodo;
int buscarLlavePar(Nodo *a, int valor, int *pk, int p);
```

Si existe un nodo en el árbol *a* cuyo campo *v* es igual a *valor*, *buscarLlave* debe retornar 1 y entregar el campo *k* en **pk*. En caso contrario la función debe retornar 0. La búsqueda tiene que ser exhaustiva porque los valores están desordenados en el árbol.

Métodología obligatoria: Si la búsqueda no concluye en la raíz de *a* y *p>1*, use *fork* para crear un nuevo proceso pesado. El proceso padre busca *valor* recursivamente en el subárbol izquierdo de *a* usando *p/2 procesos pesados*. El proceso hijo busca *valor* recursivamente en el subárbol derecho de *a* usando *p-p/2* procesos pesados, entregando el resultado a través de un *pipe*. Suponga que el árbol está razonablemente equilibrado. Al final de la búsqueda Ud. habrá invocado *p-1* veces *fork* para crear *p-1* nuevos procesos pesados, además del proceso pesado en donde se invocó *buscarPar*. Este problema es similar al ordenamiento con quicksort usando procesos pesados visto en clase de cátedra.