Pregunta 1

Programe la función *desplazarDer* con el siguiente encabezado:

```
void desplazarDer(char *s, int n);
```

Esta función desplaza el string *s* en *n* caracteres hacia la derecha agregando espacios en blanco al principio y eliminando los últimos *n* caracteres. Por simplicidad puede considerar que *s* tiene al menos *n* caracteres. Está permitido usar *strlen*. Este es un ejemplo de uso:

```
char s[]= "quien vive";
desplazarDer(s, 3); // s es el string "   quien v" con 3 espacios al inicio
```

Restricciones: 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. Si *L* es el largo del string, Ud. *debe* recorrer los caracteres desde la posición *L-n-1* hacia la posición 0, copiándolos en su posición de destino. Si recorre en orden inverso sobreescribirá incorrectamente caracteres que no ha copiado todavía.

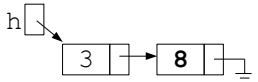
Pregunta 2

Programe la función *extraerUltimo* definida como:

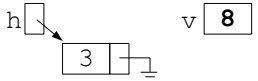
```
typedef struct nodo {
  int x;
  struct nodo *prox;
} Nodo;
void extraerUltimo(Nodo **plista);
```

Esta función recibe en ***plista* una lista simplemente enlazada *no vacía*. Ud. debe extraer el último nodo de la lista y retornar el entero contenido en ese nodo. Se requiere además que Ud. *libere la memoria* ocupada por el nodo extraído. En el siguiente ejemplo de uso, la lista *h* ha sido creada con 2 nodos que almacenan los enteros 3 y 8. Luego se invoca 2 veces la función *extraerUltimo*. La figura muestra los resultados de la función después de cada invocación:

```
Nodo *h= …;
```




```
int v= extraerUltimo(&h);
```




```
int w= extraerUltimo(&h);
```

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

Pregunta 3

Un *left/right mutex* es un tipo de mutex que se puede otorgar completo o por mitades. La implementación de abajo es incorrecta e ineficiente, pero funciona el 99,9% de las veces. Reprograme este código de manera 100% correcta y eficiente. No importa que su solución sufra de hambruna.

```
enum { LEFT= 0, RIGHT= 1 };
int busy[2]= { 0, 0 }; // Ambas mitades libres
```

```
int halfLock() {
  while (busy[LEFT] &&
         busy[RIGHT])
    ; // ¡busy waiting!
  int side= busy[LEFT] ? RIGHT
                       : LEFT;
  busy[side]= 1;
  return side;
}
void halfUnlock(int side) {
  busy[side]= 0;
}
```

```
void fullLock() {
  while ( busy[LEFT] ||
          busy[RIGHT])
    ; // ¡busy waiting!
  busy[LEFT]= busy[RIGHT]= 1;
}
void fullUnlock() {
  busy[LEFT]= busy[RIGHT]= 0;
}
```

Solo un thread puede obtener el mutex completo llamando a *fullLock*. Posteriormente, ese thread devuelve el mutex con *fullUnlock*. Hasta 2 threads pueden obtener cada uno una mitad distinta del mutex por medio de *halfLock*, que entrega LEFT si se otorgó la mitad izquierda o RIGHT si fue la mitad derecha. Más tarde cada thread devuelve su respectiva mitad con *halfUnlock*, especificando cuál fue la mitad que se le había otorgado.

Pregunta 4

Programe la función *maxArreglo* que entrega el máximo valor almacenado en el arreglo *a* de *n* enteros. El cálculo debe hacerse en paralelo usando *p* threads (además del thread que invoca la función). La función tiene el siguiente encabezado:

```
int maxArreglo(int a[], int n, int p);
```

Por simplicidad puede suponer que *n* es múltiplo de *p* y que la constante MININT almacena el entero más negativo.