Pregunta 1

Parte a.- Programe la siguiente función:

```
void espaciar(char *str);
```

Esta función debe agregar un espacio después de cada caracter en str. El siguiente es un ejemplo de uso:

```
char s[9]= "hola";
espaciar(s); // s es "h o l a "
```

Restricciones: No use el operador de subindicación de arreglos [ ] ni su equivalente **(p+i)*, use aritmética de punteros. No puede pedir memoria adicional (por ejemplo usando *malloc* o declarando un arreglo de caracteres).

Ayuda: Para hacer el espaciado recorra el string de atrás hacia adelante. Observe que el resultado debe quedar en la misma área de memoria que recibe como parámetro. Esta tiene el tamaño justo para el resultado.

Parte b.- Programe la función *cortar* definida como:

```
typedef struct nodo {
  char *pal;
  struct nodo *prox;
} Nodo;
void cortar(Nodo **plista, char *pal, Nodo **pres);
```

Esta función recibe en **plista* una lista simplemente enlazada en donde cada nodo almacena una palabra. Las palabras de la lista están ordenadas alfabéticamente. El segundo parámetro *pal* es una palabra que señala un punto de corte para la lista enlazada. Ud. debe cortar la lista dejando en **plista* la porción de la lista que antecede alfabéticamente a *pal* y en **pres* el resto de la lista. En el siguiente ejemplo de uso, la lista *h* ha sido creada con 5 nodos que almacenan las palabras "a", "b", "c", "d" y "e". Luego se invoca *cortar* como se indica en este código:

```
Nodo *h= …;
Nodo *r;
cortar(&h, "c", &r);
```

La siguiente figura muestra el cambio que produce la invocación de cortar en los punteros *h*, *r* y la lista enlazada.

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

Pregunta 2 (60%)

Parte i.- Programe la función:

```
char *ultimaDireccionValida(char *ptr);
```

Esta función debe entregar la última dirección válida que se puede leer a partir de *ptr*. Para calcularla lea el caracter apuntado por *ptr* e incremente *ptr* en 1. Repita esta lectura e incremento indefinidamente hasta que se produzca el *segmentation fault*. Capture la señal *SIGSEGV*. Resguarde *ptr* en una variable global. El último valor de *ptr* menos 1 es la última dirección válida.

Parte ii.- Un pub posee un único baño que debe ser compartido por damas y varones. El baño es amplio y admite un número ilimitado de personas. El problema consiste en evitar que las damas se encuentren con los varones dentro del baño. Las damas son representadas por threads que solicitan entrar al baño invocando la función *entrarDama* y notifican su salida llamando a *salirDama*, análogamente los threads varones invocan *entrarVaron* y *salirVaron*. Las funciones *entrarDama* y *entrarVaron* deben esperar cuando en el baño se encuentran personas del sexo opuesto. La siguiente es una solución incorrecta e ineficiente para la programación de estas 4 funciones:

```
int damas= 0, varones=0;
```

```
void entrarVaron() {
  while (damas>0)
    ;
  varon++;
}

void salirVaron() {
  varon--;
}
```

```
void entrarDama() {
  while (varones>0)
    ;
  damas++;
}

void salirDama() {
  damas--;
}
```

A pesar de todo esta solución funciona correctamente el 99,9% de las veces. Reprograme el código de más arriba de manera correcta y eficiente. No importa que su solución sufra de hambruna.

Ayuda: el código de más arriba está casi bueno, no cambie nada, solo agregue el código faltante.