Pregunta 1

Programe la siguiente función:

```
char *proxima_palabra(char **pfrase);
```

El parámetro ***pfrase*** apunta a un string de C que contiene palabras separadas por espacios en blanco. Esta función debe retornar la primera palabra de la frase y entregar en ***pfrase*** lo que quedó de la frase. La siguiente tabla muestra ejemplos de uso. Las columnas ***pal*** y ***frase*** muestran los strings a los que deben apuntar los punteros *pal* y *frase* justo después de invocar el código de la primera columna:

Código	***pal***	***frase***
`char *frase= "hola  que tal";` `char *pal= proxima_palabra(&frase);`	"hola"	" que tal"
`pal= proxima_palabra(&frase);`	"que"	" tal"
`pal= proxima_palabra(&frase);`	"tal"	""
`pal= proxima_palabra(&frase);`	NULL	""

Restricciones: No use el operador de subindicación de arreglos [ ], en su lugar use aritmética de punteros; no modifique el string que recibe en ***pfrase*, podría estar en un área de la memoria que es de solo lectura; pida con *malloc* la memoria que necesita la palabra que retorna la función. No necesita invocar *malloc* para el string que entrega en ***pfrase*, puede apuntar a otra parte del mismo string que recibió en ***pfrase*.

Pregunta 2

Parte i.- Se usan listas enlazadas para representar grandes números binarios positivos. Cada nodo contiene un solo bit y se representa mediante la estructura *Bit* definida de la siguiente manera:

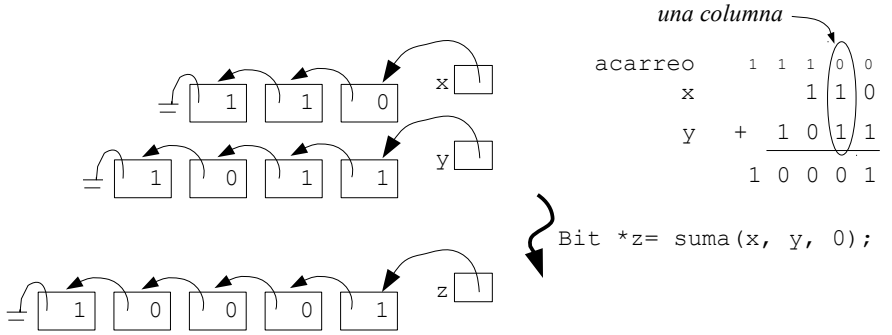
```
typedef struct bit {
  int b; /* 0 o 1 */
  struct bit *prox;
} Bit;
```

El primer nodo de la lista es el bit menos significativo. La lista vacía representa el número 0.

Programe la función:

```
Bit *suma(Bit *x, Bit *y, int acarreo);
```

Esta función recibe 2 números en las listas *x* e *y* más un *acarreo* que solo puede ser 0 o 1. La función retorna una nueva lista que representa el número *x+y+acarreo*. En el siguiente ejemplo de uso, las variables *x*, *y* y *z* son de tipo *Bit**:



En el ejemplo las listas *x* e *y* representan los enteros 6 y 11 respectivamente. El primer nodo de la lista se muestra a la derecha para que sea más fácil leer el número en binario (de izquierda a derecha los bits aparecen de mayor a menor significancia). Al sumar ambos números el resultado es el número *z*.

Observe que los números en binario se suman como los números en decimal: se suman los bits de la misma significancia en ambos números (misma columna) más un acarreo que viene de la columna anterior. Se comienza por la columna de menor significancia. Esto se hace con el siguiente código:

```
int sum= (x==NULL ? 0 : x->b) + (y==NULL ? 0 : y->b) + acarreo;
```

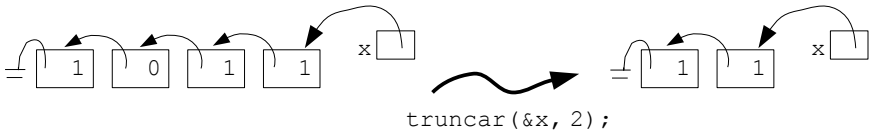
Sum resulta ser un número de 2 bits (entre 0 y 3). El bit menos significativo es el resultado de esa columna. El bit más significativo se usa como acarreo para la siguiente columna.

Restricciones: No use instrucciones de iteración como *while*, *for* o *do ... while*; no use los operadores de multiplicación, división o módulo (* / %). Use recursividad y los operadores de bits.

Parte ii.- Programe la siguiente función:

```
void truncar(Bit **px, int n);
```

Esta función trunca el número ***px* a *n* bits, liberando los nodos de mayor significancia que exceden los *n* bits. En el ejemplo de uso de más abajo, la variable *x* es de tipo *Bit **:



Observe que si se invoca `truncar(&x, 0)`, *x* resulta ser la lista vacía.

No olvide invocar *free* para liberar los nodos sobrantes.

Recomendación: Use recursividad.