Pregunta 1

Programe la función *extraerBits* con el siguiente encabezado:

```
typedef unsigned int uint;
uint extraerBits(uint *px, int n);
```

Sea x=**px*. Considere que x se descompone en sus 32 bits: x_{31} … x_0. La función *extraerBits* debe retornar los n bits menos significativos de x, es decir x_{n-1}… x_0, completando con ceros a la izquierda. Además debe extraer esos bits, dejando en **px*: 0…0 x_{31}…x_n. El siguiente es un ejemplo de uso:

```
uint x= 0b1010 01 110;
uint a= extraerBits(&x, 3); // a=0b110, x=0b1010 01
uint b= extraerBits(&x, 2); // b=0b01, x=0b1010
uint c= extraerBits(&x, 4); // c=0b1010, x=0
```

Restricción: Ud. no puede usar los operadores de multiplicación, división o módulo (* / %). Use eficientemente los operadores de bits.

Pregunta 2

Programe la siguiente función:

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

Esta función debe transformar cada espacio en blanco presente en *str* por 3 espacios. El siguiente es un ejemplo de uso:

```
char s[26]= " hola que  tal "; // hay 5 espacios
tripleEspacio(s); // s es "   hola   que      tal   "
```

Observe que el primer y último espacio se transformaron en 3 espacios, al igual que el espacio entre "hola" y "que". Como entre "que" y "tal" habían 2 espacios, estos se transformaron en 6 espacios.

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.

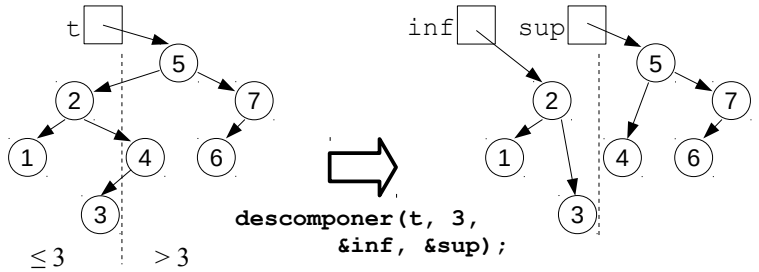
Ayuda: Primero cuente cuantos espacios hay en *str*. Para hacer la transformación recorra el string *str* de atrás hacia adelante. Observe que el resultado debe quedar en la misma área de memoria que recibe como parámetro. Esta tiene el tamaño justo para el resultado.

Pregunta 3

Programe la función *descomponer* declarada de la siguiente manera:

```
typedef struct nodo {
  int x; // La etiqueta
  struct nodo *izq, *der;
} Nodo;
void descomponer(Nodo *t, int z,
                 Nodo **pinf, Nodo **psup);
```

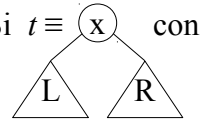
Esta función descompone el árbol de búsqueda binaria t dejando en **pinf* un árbol de búsqueda binaria con todos los nodos de t con etiquetas menores o iguales a z, y en **psup* un árbol de búsqueda binaria con los nodos de t con etiquetas mayores que z. En el siguiente ejemplo de uso el tipo de las variables t, *inf* y *sup* es *Nodo**:



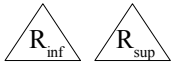
Observe que no se especifica el nodo al cual queda apuntando t.

Restricciones: No puede usar *malloc*. Debe reutilizar los nodos que recibe en t, modificando solo los campos *izq* y *der*. No altere x.

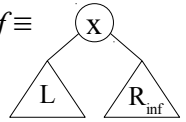
Ayuda: Si t es el ábol vacío, la solución es trivial.

Si $t \equiv$  con $x \le z$:

Descomponer recursivamente R, transformándolo en los subárboles:



Luego transformar t entregando como resultado estos 2 árboles:

inf ≡ 

sup ≡ 

Cuando $z < x$ se debe descomponer recursivamente L. Deduzca el resto.