Pregunta 1

Programe la función *descomprimir* con el siguiente encabezado:

```
typedef unsigned int uint;
void descomprimir(uint a[], uint x, int nbits);
```

El número *x* almacena varios enteros sin signo de *nbits* cada uno. Esta función debe extraerlos y almacenarlos en el arreglo *a*. Ejemplo de uso:

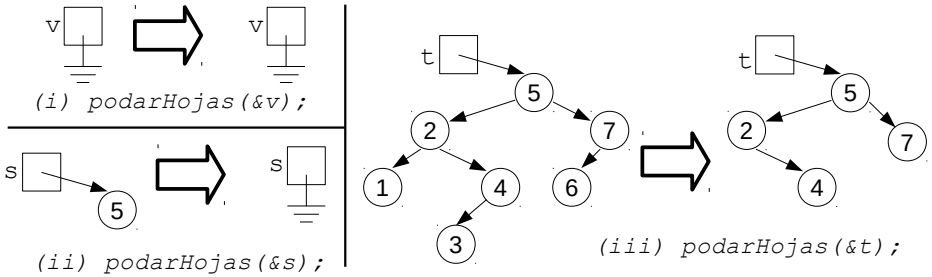
```
int a[8]; descomprimir(a, 0xf31a5, 4); // x en hexadecimal
// a[0]=5 son los 4 bits de menor significancia de x, a[1]=10 son los siguientes
// 4 bits, a[2]=1, a[3]=3, a[4]=15, a[5] hasta a[7] = 0
int b[11]; descomprimir(b, 02573401, 3); // x en octal por el 0
// b[0]=1 son los 3 bits de menor significancia de x, b[1]=0, b[2]=4, ..., b[10]=0
```

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

Pregunta 2

Programe la función *podarHojas* con el encabezado del cuadro de la derecha. Esta función recibe en **pa* un árbol binario y elimina todas sus hojas. Una hoja es un nodo cuyos subárboles izquierdo y derecho están vacíos. Además *podarHojas libera* el espacio que ocupaban esos nodos en memoria. En los 3 ejemplos de la figura el tipo de las variables *v*, *s* y *t* es *Nodo**. El ejemplo (i) muestra que podar las hojas de un árbol vacío lo deja igual. En (ii) el árbol es solo una hoja, al podar sus hojas queda vacío. En el ejemplo (iii) se podan las hojas de un típico árbol.

```
typedef struct nodo {
  int x;
  struct nodo *izq, *der;
} Nodo;
void podarHojas(Nodo **pa);
```



Pregunta 3

Programe la función *elimRep* con encabezado: `void elimRep(char *s);`

Esta función elimina las repeticiones de caracteres consecutivos. Ejemplo de uso:

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

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.

Pregunta 4

Escriba el programa *muestrear* que recibe como parámetros el nombre de un archivo y un número *n*. El programa debe desplegar en la salida estándar una de cada *n* líneas del archivo. Ejemplo de uso:

archivo noms.txt	*Uso de muestrear*
pedro juan diego ana francia patricia renato maria	$ ./muestrear noms.txt 3 pedro ana renato $ ./muestrear noms.txt 4 pedro francia $

Todas las líneas son de 20 caracteres (19 caracteres más el fin de línea '\n'). Debe usar *fseek* para seleccionar qué línea va a leer y *fread* para leerla. No puede leer todo el archivo en memoria ni leer todas las líneas.

Pregunta 5

Se necesita estimar la probabilidad de que al lanzar *n* dados, su suma resulte ser *sum*. Para ayudar en la estimación, la función *favorables* simula *rep* veces el lanzamiento de los *n* dados, retornando la cantidad de casos favorables, es decir los casos en que la suma fue *sum*. La probabilidad se estima como casos favorables / *rep*. Esta es la versión secuencial de *favorables*:

```
int favorables(int n, int sum, int rep) {
  int fav= 0;     // número de casos favorables
  for (int i= 0; i<rep; i++) {
    int s= 0;     // suma de los dados
    for (int j= 0; j<n; j++)
      s += rand_int(1,6); // entrega entero aleatorio en [1,6]
    if (s==sum)   // caso favorable: los dados suman sum
      fav++;
  }
  return fav;
}
```

Programe la función *favorables_par* con el mismo encabezado que *favorables*, pero que usa 8 cores para realizar las *rep* simulaciones. Para lograrlo Ud. debe lanzar 8 threads con *pthread_create*. El thread principal solo se dedica a esperar el resto de los threads y retornar el total de casos favorables. Cada thread invoca la función *favorables* para calcular un octavo de las simulaciones (*rep*/8). Por simplicidad suponga que *rep* es múltiplo de 8.