Pregunta 1

El archivo de texto *cola.txt* almacena una cola de prioridad en el formato que muestra la columna de la izquierda de la siguiente tabla:

archivo *cola.txt*	**Ejemplo de uso**	***cola.txt* después**
4 pedro 8 **juan 2** diego 7 ximena 4 ~~ana 1~~	$ **./extraer cola.txt** juan $	**3** pedro 8 **ximena 4** diego 7 ~~ximena 4~~ ~~ana 1~~

La primera línea del archivo tiene 4 caracteres más el \n (cambio de línea) y corresponde a la cantidad de elementos almacenados en la cola. Cada una del resto de las líneas tiene 16 caracteres más el \n. Los primeros 10 corresponden a un texto almacenado en la cola y los 6 restantes a su prioridad. En el ejemplo la mejor prioridad es 2. El archivo podría contener 0, 1 o más líneas sobrantes al final (de 16+1 caracteres) debido a elementos extraídos, pero solo al final. Su contenido es irrelevante.

Programe el comando *extraer* que recibe como único argumento el nombre del archivo que contiene la cola y extrae de ella el texto con la mejor prioridad, mostrándolo en pantalla. Para la extracción Ud. debe (i) leer secuencialmente todos los elementos de la cola, buscando el texto con mejor prioridad; (ii) mostrarlo en pantalla (recuerde que al mostrar un string con *printf*, ese string debe terminar con \0); (iii) usar *fseek* para actualizar el tamaño de la cola; y (iv) usar *fseek* para mover el último elemento de la cola a la línea que queda libre.

La columna del centro de la tabla de arriba muestra un ejemplo de uso y la columna de la derecha muestra el contenido de *cola.txt* después de la extracción. Observe que el *i*-ésimo elemento de la cola está en la posición *4+1+i*(16+1)* en el archivo (partiendo con *i≡0*). Decida Ud. que hacer cuando la cola está vacía. Está prohibido almacenar todo el contenido de la cola en la memoria.

Ayuda: API para manejar archivos y otras funciones de utilidad:

```
FILE  *fopen(char *nom_arch, char *modo);   // modo es "r+" para lect/escr
size_t  fread(char *buf, size_t ancho_item, int n_item, FILE *file);
size_t  fwrite(char *buf, size_t ancho_item, int n_item, FILE *file);
int  fseek(FILE *file, long despl, int w);   // w==SEEK_SET
int  fclose(FILE *file);
```

```
int n= atoi("25  \n");    // n es 25
char s[6], d[6];
sprintf(s, "%-4d\n", n); // imprime en un string, no en un archivo,
 // %-4d significa mostrar entero en 4 caracteres, alineado a la izquierda,
 // s es el string "25  \n" de largo 5
bcopy(d, s, 4);           // copia 4 bytes a partir de s en d
```

Pregunta 2

Parte a.- En aplicaciones de reproducción de video como *youtube* y *netflix* cuando se vacía el buffer la acción no es esperar a leer un solo cuadro de video, si no que más bien la reproducción queda en pausa hasta que el buffer esté completamente lleno. Es preferible hacer pocas pausas, aunque largas, que hacer muchas pausas de un solo cuadro.

Modifique la funciones *put* y *get* del buffer de manera que se logre el comportamiento de *youtube* y *netflix*. No olvide que el video termina cuando se deposita NULL en el buffer. El buffer nunca se llenará si el video ya terminó. Puede agregar campos a *Buffer* si los necesita. Esta era la implementación de *put* y *get*:

```
Cuadro *get(Buffer *b) {
  lock(&b->m);
  while (b->cnt==0)
    wait(&b->cond, &b->m);
  Cuadro *cuadro=
          b->array[b->out];
  b->out= (b->out+1) %
           b->size;
  b->cnt--;
  broadcast(&b->cond);
  unlock(&b->m);
  return cuadro;
}
```

```
void put(Buffer *b,
         Cuadro *cuadro) {
  lock(&b->m);
  while (b->cnt==b->size)
    wait(&b->cond, &b->m);
  b->array[b->in]= cuadro;
  b->in= (b->in+1) %
         b->size;
  b->cnt++;
  broadcast(&b->cond);
  unlock(&b->m);
}
```

Parte b.- La función de abajo es una implementación del algoritmo *quicksort* para ordenar un arreglo de enteros. Paralelice la función *quicksort* para una máquina dual-core.

```
void quicksort(int a[], int i, int j) {
  if (i<j) {
    int h= particionar(a, i, j);
    quicksort(a, i, h-1);
    quicksort(a, h+1, j);
} }
```

Restricción: Ud. puede crear a lo más un solo thread.