Programación Para Sistemas. Examen C Convocatoria Extraordinaria

- Se debe escribir el nombre, apellidos y DNI en la hoja de respuesta.
- Para indicar su nombre, utilice letra clara, mejor en **mayúsculas**.
- Escriba las respuestas en los recuadros de cada apartado. Solo se entregan las hojas con las respuestas.
- El examen tendrá una duración de **1 hora y media**.

1. Descripción general del problema

Se llama *parser* (del inglés analizador) a un programa (rutina, librería, módulo software, ...) que procesa información, generalmente textual, para dividirla en sus distintas partes útiles y organizarla en algún tipo de estructura de datos. Tu tarea será hacer un programa, *parser*, que procese archivos "ini" y guarde la información en una cadena enlazada en orden inverso[1]. Los archivos "ini" son archivos de texto plano por líneas, con cada línea conteniendo la información de un par `<clave, valor>` . Cada línea puede ser:

```
;Comentario

;Compiladores
AnsiC=gcc
Cpp=g++

;Opciones de compilacion
CompilerOptions=-ansi -Wall -Wextra

;Final
```

- Una línea vacía (se supone que todas las líneas, incluida la última, acaban con salto de línea, `'\n'`).
- Un comentario (el primer carácter debe ser `';'`).
- Un par de cadenas alfanuméricas: una primera cadena, la *clave*, un carácter igual (`'='`) y otra cadena, el *valor*. A la derecha se muestra un ejemplo de este tipo de archivo.

2. Información de referencia

Se **puede usar cualquier función** de la librería estándar de C, no obstante, **las más adecuadas** para la resolución más simple de las preguntas y, por tanto, las recomendadas **son**:

char *strcpy(char *dest, const char *src);	Copia la cadena alfanumérica apuntada por `src`, incluido el byte *null* ('\0') al buffer apuntado por `dest`. La cadena destino debe ser suficientemente grande para recibir la copia.
char *strchr(const char *s, int c);	Retorna un puntero a la primera ocurrencia del carácter `c` en la cadena alfanumérica `s`. Retorna `NULL` si el carácter no se encuentra. En el **segundo apartado se muestra un ejemplo** de uso.
char *fgets(char *s, int size, FILE *stream);	lee hasta un máximo de uno menos que `size` caracteres desde `stream` y los guarda en el *buffer* apuntado por `s`. La lectura se detiene después de un EOF o un carácter nueva línea. Si la nueva línea se lee entonces se guarda en el *buffer*. Se guarda el byte *null* ('\0') después del último carácter guardado. Retorna `s` si se termina con éxito y `NULL` cuando se encuentra un error o cuando se encuentra el final de archivo sin que se haya leído ningún carácter.
void *calloc(size_t nmemb, size_t size);	reserva memoria para un *array* de `nmemb` elementos de tamaño `size`. La memoria se inicializa a 0.
void *malloc(size_t size);	reserva `size` bytes memoria y retorna el puntero a la memoria reservada
void free(void *ptr);	libera el espacio en memoria apuntado por `ptr` que debe haber sido reservado previamente por una llamada a `calloc` (o similares).

[1] Generar la cadena enlazada en orden inverso es mucho más sencillo que generarla en el mismo orden en que aparecen las líneas. Las sucesivas líneas se insertan siempre al inicio de la cadena enlazada.

3. Se pide

Completar el código fuente que se muestra en la hoja de respuestas. La descripción de las funciones a completar se indica a continuación:

1. La función `es_comentario_o_vacia` toma como argumento una cadena alfanumérica (que será el contenido de una línea del archivo), y, retorna 1 (verdadero) si la línea es un comentario o está vacía; en caso contrario retorna 0 (falso).

2. La función `comprobar_linea` toma como argumentos una cadena alfanumérica, `linea` (que será el contenido de una línea del archivo), y un número de línea, `nlinea` y

   a) Retorna 1 (verdadero) si la línea es válida y 0 (falso) en caso contrario. Una línea es válida si cumple que se ha leído completamente (incluido el salto de línea) y contiene el carácter `'='`.

   b) Escribe en el canal de error: el número de línea, una letra 'E', 'C', 'V' o 'N' respectivamente si la línea tiene un error, es un comentario, está vacía o es una línea *normal* y un texto adicional. Para los 2 errores se escribe el error encontrado (es decir: "linea demasiado larga" o "no se encuentra el caracter ="). Los comentarios se escriben tal cuál, pero **omitiendo el carácter punto y coma**. Una línea vacía solamente contiene el carácter `'\n'`. El formato de escritura debe deducirse de los siguientes ejemplos:

   ```
   #2:V: vacia
   #3:C: Compiladores
   #4:N: AnsiC=gcc
   ```

3. La función `comprobar_formato` toma como argumento el nombre (ruta) de un archivo y:

   a) Abre, lee y cierra el archivo cuyo nombre se pasa como argumento.

   b) Lee las líneas utilizado un *buffer* de tamaño dado por la macro `TAM_LINEA`.

   c) Llama a la función `comprobar_linea` con todas las líneas del archivo (o hasta que encuentra una línea con un error).

   d) Retorna 1 (verdadero) si todas las líneas cumplen con la comprobación implementada en `comprobar_linea`.

   e) Retorna 0 (falso) en caso contrario. También devuelve 0 si el archivo no se puede abrir.

4. La función `separar_clave_valor` toma como argumento la dirección de memoria de una estructura de tipo `nodo_t` (está definida al principio de la hoja de respuestas) y:

   a) Encuentra las posiciones de los caracteres `'='` y `'\n'` en el campo `texto` de la estructura.

   b) Asigna el carácter nulo a las posiciones encontradas.

   c) Asigna la dirección de memoria inmediatamente posterior al lugar en que se encontraba el carácter `'='` al campo `valor`.

5. La función `lista_lineas_inv` toma como argumentos: el nombre (ruta) de un archivo y la dirección de memoria de un entero `p_ncv` y:

   a) Lee el archivo de nombre dado y genera una lista enlazada de estructuras `nodo_t` que almacenan el contenido de las líneas (no comentario o vacías) del archivo en orden inverso, es decir, insertando las líneas siempre al principio, y retorna la dirección de memoria al primer elemento.

   b) Cuenta las líneas que son comentarios o vacías y guarda el valor obtenido en el entero apuntado por `p_ncv`.

6. La función `liberar` toma como argumento una lista enlazada a cuyo primer elemento apunta `las_lineas` y elimina todos los nodos y **libera** la memoria reservada anteriormente sin provocar fugas de memoria.

7. La función `main`:

   a) Comprueba que recibe un único argumento y si no es así termina con un código de error 16.

   b) Utiliza el argumento recibido como el nombre de un archivo para comprobar si el archivo tiene el formato correcto, si no es así termina con un código de error 17.

   c) Llama a la función `lista_lineas` para generar una lista enlazada de estructuras `nodo_t` a partir del argumento.

   d) Muestra ~~el número de líneas leídas~~ y el número de ellas que son comentarios o vacías en el canal de salida estándar.

HOJA DE RESPUESTAS

DNI:	Apellidos, Nombre:

```
#include <stdlib.h>
#include <string.h>
#include <stdio.h>

#define TAM_LINEA 256

struct nodo_linea {
  char texto[TAM_LINEA];
  char *valor;
  struct nodo_linea* next;
};
typedef struct nodo_linea nodo_t;
```

Apartado 1. (1.0 puntos)

```
int es_comentario_o_vacia(char* linea) {
  return linea[0] == ';' || linea[0] == '\n';
}
```

Apartado 2. (1.5 puntos)

```
int comprobar_linea(char* linea, int nlinea) {
  int ok = 1;
  if (strchr(linea, '\n') == NULL) { /* linea demasiado larga */
    ok = 0;
    fprintf(stderr, "#%d:E: linea demasiado larga\n", nlinea);
  } else if (linea[0] == ';') {
    fprintf(stderr, "#%d:C: %s", nlinea, linea + 1);
  } else if (linea[0] == '\n') {
    fprintf(stderr, "#%d:V: vacia\n", nlinea);
  } else if (strchr(linea, '=') == NULL) {
    ok = 0;
    fprintf(stderr, "#%d:E: La linea no contiene '='\n", nlinea);
  } else {
    fprintf(stderr, "#%d:N: %s", nlinea, linea);
  }
  return ok;
}
```

Apartado 3. (1.5 puntos)

```
int comprobar_formato(char* filename) {
  char buffer[TAM_LINEA];
  FILE *g = fopen(filename, "r");
  int ok = 0, i = 0;
  if (g != NULL) {
    ok = 1;
    while ( ok && fgets(buffer, TAM_LINEA, g) != NULL)  {
      ++i;
      ok = comprobar_linea(buffer, i);
    }
    fclose(g);
  }
  return ok;
}
```

Apartado 4. (1.5 puntos)

```
void separar_clave_valor(nodo_t *pnodo) {
  char *igual = strchr(pnodo->texto, '=');
  char *salto_linea = strchr(pnodo->texto, '\n');
  *igual = '\0';
  *salto_linea = '\0';
  pnodo->valor = igual + 1;
}
```

Apartado 5. (2.0 puntos)

```
nodo_t* lista_lineas_inv(char *filename, int *p_ncv) {
  char buffer[TAM_LINEA];
  FILE *g = fopen(filename, "r");
  nodo_t *prim = NULL, *aux;
  *p_ncv = 0;
  if (g != NULL) {
      while (fgets(buffer, TAM_LINEA, g) != NULL) {
        if (es_comentario_o_vacia(buffer))  {
          *p_ncv += 1;
        } else {
          aux = calloc(1, sizeof(nodo_t));
          strcpy(aux->texto, buffer);
          aux->next = prim;
          prim = aux;
        }
      }
    fclose(g);
  }
  return prim;
}

/* Aplica separar_clave_valor a todos los nodo de la lista a cuyo primer elemento apunta las_lineas */
void procesar_lineas(nodo_t *las_lineas)  { ... Codigo Omitido ... }

/* Escribe en el canal de salida los nodos de la lista a cuyo primer elemento apunta las_lineas */
void mostrar_clave_valor(nodo_t *las_lineas)  { ... Codigo Omitido ... }
```

Apartado 6. (1.0 puntos)

```
void liberar(nodo_t *las_lineas) {
  nodo_t *borrar;
  while (las_lineas != NULL) {
    borrar = las_lineas;
    las_lineas = las_lineas->next;
    free(borrar);
  }
}
```

Apartado 7. (1.5 puntos)

```
int main(int argc, char* argv[]) {
  nodo_t *lineas;
  int n_cv;

  if ( argc != 2 ) exit(16);
  if ( !comprobar_formato(argv[1]) ) exit(17);

  lineas = lista_lineas_inv(argv[1], &n_cv);
  fprintf(stderr, "Hay %d lineas de comentario o vacias\n", n_cv);
  procesar_lineas(lineas);
  mostrar_clave_valor(lineas);
  liberar(lineas);
  return 0;
}
```