SISTEMAS OPERATIVOS I - Práctica 2 (Parte 2 con soluciones)
Grado en Ingeniería Informática - Escuela Superior de Informática (UCLM))

1. Actividades de Laboratorio

Escriba un programa C estándar para cada uno de los distintos enunciados con la funcionalidad indicada. Salvo que se especifique lo contrario, se entenderá que la entrada y salida del programa corresponderá a la entrada estándar y salida estándar. Se aconseja utilizar la función `scanf` de la biblioteca estándar para la lectura de números enteros o reales de la entrada salvo otra indicación en el enunciado de la actividad.

1. Escriba una función llamada `leerlinea` que tenga dos argumentos, el primero de ellos será un vector de caracteres y el segundo argumento será un entero que indicará el número máximo de caracteres que puede almacenarse en el vector del primer argumento.

   La función debe leer una línea de la entrada estándar y almacenar sus caracteres en el primer argumento de la función. Supóngase que el tamaño máximo que puede tener una línea de la entrada es algo fijo y razonable.

   Además, se va a preparar un archivo de funciones, llamado `util.c`, que pueden resultar útiles en otros ejercicios posteriores. La función `leerlinea` debe formar parte de este archivo. Se debe compilar dicho archivo y obtener el módulo de código objeto `util.o`.

   Por último, escriba un programa que lea líneas de la entrada estándar y copie en la salida estándar únicamente la línea más larga que haya leído. Para ello utilice la función del módulo `util.o`

```c
/* Código en util.c */
int leerlinea(char s[], int lim) {
  int c, i;

  for (i = 0;
       (i < lim - 1) && ((c = getchar()) != EOF && c!= '\n');
       i++)
    s[i] = c;

  if (c == '\n') {
    s[i] = c;
    ++i;
  }
  s[i] = '\0';

  return i;
} /* Fin de código en util.c */

#include <stdio.h>
#include <string.h>
#include "util.h"

#define MAXLINE 1000

int main() {
  int len;  /* Longitud de la línea actual */
  int max;  /* Máxima longitud de las líneas leídas */
  char line[MAXLINE]; /* Línea de la entrada actual */
  char longest[MAXLINE];  /* Línea más larga leída */

  max = 0;

  while ((len = leerlinea(line, MAXLINE)) > 0)
    if (len > max) {
      max = len;
      strcpy(longest, line);  /* Copia la línea actual (line) en longest */
```

```
37        }

39      if (max > 0)  /* Se ha leído al menos una línea */
          printf("%s\n", longest);
41
        return 0;
43    }
```

2. Incorpore una función llamada `miatoi` al archivo `util.c` con la funcionalidad de `atoi` para enteros positivos de la biblioteca estándar de C y escriba un programa que utilice dicha función

```
1   /* Código en util.c */
    int miatoi(char s[]) {
3     int i, n;

5     n = 0;
      for (i = 0; s[i] >= '0' && s[i] <= '9'; i++)
7       n = 10 * n + (s[i] - '0');

9     return n;
    } /* Fin de código en util.c */
11
    #include <stdio.h>
13  #include "util.h"

15  int main() {

17    printf("%d\n", miatoi("358"));

19    return 0;
    }
```

3. Dado el siguiente programa, indique de forma razonada cuántas zonas distintas de memoria utiliza el programa y la justificación del valor de las variables mostrado en la salida

```
    #include <stdio.h>
2
    int vg1;
4   int *vg2;

6   void funcion1(void) {
      int i;
8     int x = 1, y = 2, z[3];
      int *ip;
10
      ip = &x;
12    y = *ip;
      *ip = 0;
14    ip = &z[0];
      for (i = 0; i < 3; i++)
16      *ip++ = i;

18    printf("\nVARIABLES DE funcion1\n");
      printf("   i - Dirección: %p - Valor: %d\n", (void *)&i, i);
20    printf("   x - Dirección: %p - Valor: %d\n", (void *)&x, x);
      printf("   y - Dirección: %p - Valor: %d\n", (void *)&y, y);
22    for (i=0; i<3; i++)
        printf("z[%d] - Dirección: %p - Valor: %d\n", i, (void *)&z[i], z[i]);
24    printf("  ip - Dirección: %p - Valor: %p\n", (void *)&ip, (void *)ip);
    }
26
    void funcion2(void) {
```

```
28      int vf2;

30      printf("\nVARIABLES DE funcion2\n");
        printf("vf2 - Dirección: %p - Valor: %d\n", (void *)&vf2, vf2);
32    }
      int main() {
34      int vm1;

36      printf("VARIABLES GLOBALES\n");
        printf("vg1 - Dirección: %p - Valor: %d\n", (void *)&vg1, vg1);
38      printf("vg2 - Dirección: %p - Valor: %p\n", (void *)&vg2, (void *)vg2);

40      printf("\nVARIABLES DE main\n");
        printf("vm1 - Dirección: %p - Valor: %d\n", (void *)&vm1, vm1);
42
        printf("\nFUNCIONES\n");
44      printf("funcion1 - Dirección %p\n", (void *)funcion1);
        printf("funcion2 - Dirección %p\n", (void *)funcion2);
46      printf("    main - Dirección %p\n", (void *)main);

48      funcion1();
        funcion2();
50
        return 0;
52    }
```

La salida del programa es:

```
VARIABLES GLOBALES
vg1 - Dirección: 0x804a028 - Valor: 0
vg2 - Dirección: 0x804a024 - Valor: (nil)

VARIABLES DE main
vm1 - Dirección: 0xbf9a28ac - Valor: -1217462284

FUNCIONES
funcion1 - Dirección 0x8048444
funcion2 - Dirección 0x8048569
    main - Dirección 0x8048598

VARIABLES DE funcion1
   i - Dirección: 0xbf9a2870 - Valor: 3
   x - Dirección: 0xbf9a2874 - Valor: 0
   y - Dirección: 0xbf9a2878 - Valor: 1
z[0] - Dirección: 0xbf9a2864 - Valor: 0
z[1] - Dirección: 0xbf9a2868 - Valor: 1
z[2] - Dirección: 0xbf9a286c - Valor: 2
  ip - Dirección: 0xbf9a287c - Valor: 0xbf9a2870

VARIABLES DE funcion2
vf2 - Dirección: 0xbf9a287c - Valor: -1080416144
```

Por el valor de las direcciones de las variables y funciones del programa tenemos las siguientes zonas de memoria:

a) 0x804a024 - 0x804a028 : Zona ocupada por las variables globales del programa

b) 0x8048444 - 0x8048598 : Zona ocupada por el código de las funciones del programa

c) 0xbf9a2870 - 0xbf9a28ac : Zona ocupada por las variables locales a las funciones del programa

- Se debe tener en cuenta que las zonas anteriores se pueden extender más allá de los límites indicados pues estas direcciones indican exclusivamente la dirección ocupada por el primer byte de cada variable o función.
- Se puede observar que la dirección de memoria asignada a la variable `ip` es idéntica a la dirección de la variable `vf2`. Esto es debido a que son variables locales de dos funciones que no coinciden en el tiempo pues primero se ejecuta la función1 y posteriormente la función2.
- Los valores de las variables `vg1` y `vg2` son 0 pues no se han modificado en la ejecución del programa y al ser variables globales tienen un valor inicial implícito de 0. En el caso de `vg2` como es una variable puntero se indica la dirección con valor 0 por (nil).
- Los valores de las variables `vm1` y `vf2` no son predecibles pues no se modifican en el programa y su valor inicial no está preestablecido pues son variables locales de funciones (`main` y `funcion2`).
- El valor de la variable `i` mostrado por la sentencia de la línea 19 es consecuencia de la sentencia for de la línea 15 que en la última asignación le dá un valor de 3 a la variable `i`.
- La variable `x` tiene un valor inicial de 1 pero en la línea 11 al puntero `ip` se le asigna la dirección de la variable `x` y en la línea 13 se asigna a esta variable el valor de 0 a través del puntero `ip`.
- Asimismo, a la variable `y` en la línea 12 se le asigna el valor de la variable apuntada por `ip` que corresponde a la variable `x` antes de la asignación comentada anteriormente. Es decir, cuando tenía el valor inicial de 1 y, por tanto, el valor final de `y` es 1.
- Los valores de la matriz `z` se modifican a través del puntero `ip` que en la línea 14 apunta al primer elemento de `z`. Posteriormente, en la línea 16 con la sentencia *ip++ = i se realiza el postautoincremento del valor de `ip` y se asigna a la variable apuntada por `ip`, es decir, los valores 0, 1 y 2. Por tanto, la sentencia for de la línea 15 recorre la matriz `z` a través del puntero `ip` y asigna los valores 0, 1 y 2. Se debe tener en consideración la aritmética de los punteros pues la expresión ip++ incrementa el valor de `ip` en el tamaño en bytes de un entero que es el tipo de dato apuntado por `ip` e indicado en la declaración `int *vg2`
- Se puede comprobar que el valor final del puntero `ip` corresponde a la dirección de la variable `i` pues la zona de memoria reservada para esta variable `i` está a continuación de la zona de memoria reservada a la matriz `z`. Por este motivo, se debe tener especial cuidado en la manipulación de variables a través de punteros pues se puede alterar otras variables de una forma no evidente.

4. Incorpore una función llamada `mistrcpy` al archivo `util.c` con la funcionalidad de `strcpy` de la biblioteca estándar de C y escriba un programa que utilice dicha función

```
/* Código en util.c */
/* Distintas implementaciones de la función mistrcpy */
void mistrcpy1(char destino[], char origen[]) {
  int i;

  i = 0;
  while ((destino[i] = origen[i]) != '\0')
    i++;
}

void mistrcpy2(char *destino, char *origen) {
  while ((*destino = *origen) != '\0') {
    destino++;
    origen++;
  }
}

void mistrcpy3(char *destino, char *origen) {
  while ((*destino++ = *origen++) != '\0') ;
}

/* El modificador const en el argumento origen evita los cambios en la
```

```
    variable apuntada por el puntero origen */
24  void mistrcpy4(char *destino, char *origen) {
      while (*destino++ = *origen++) ; /* Expresión mínima */
26  } /* Fin de código en util.c */

28  #include <stdio.h>
    #include "util.h"
30
    #define MAXSIZE 255
32
    int main() {
34    char cadena[] = "Cadena a copiar."; /* El compilador calcula el tamaño */
      char copia[MAXSIZE];
36
      mistrcpy1(copia, cadena);
38    printf("Versión 1: %s - %s\n", cadena, copia);

40    mistrcpy2(copia, cadena);
      printf("Versión 2: %s - %s\n", cadena, copia);
42
      mistrcpy3(copia, cadena);
44    printf("Versión 3: %s - %s\n", cadena, copia);

46    mistrcpy4(copia, cadena);
      printf("Versión 4: %s - %s\n", cadena, copia);
48
      return 0;
50  }
```

5. Incorpore al archivo `util.c` una función llamada `void reverse(char *s)` que invierta la cadena de caracteres `s` en la propia cadena `s`. Úsela para escribir un programa que invierta su entrada, línea a línea y se envíe a la salida

```
    /* Código en util.c */
2   /* Invierte los caracteres de s */
    void reverse1(char *s) {
4     int i, j;
      char c;
6
      for (i = 0, j = strlen(s)-1; i < j; i++, j--) {
8       c = s[i];
        s[i] = s[j];
10      s[j] = c;
      }
12  }

14  void reverse2(char *s) {
      int i, j;
16    char c;

18    for (i = 0, j = strlen(s)-1; i < j; i++, j--)
        c = s[i], s[i] = s[j], s[j] = c;
20  } /* Fin de código en util.c */

22  #include <stdio.h>
    #include <string.h>
24  #include "util.h"

26  #define MAXLINE 1000

28  int main() {
      char line[MAXLINE];      /* Línea de la entrada actual */
```

```
30
      while (leerlinea(line, MAXLINE) > 0) {
32      line[strlen(line) - 1] = '\0'; /* Se elimina el carácter \n del final */
        reverse1(line); /* También se puede utilizar reverse2 */
34      printf("%s\n", line);
      }
36
      return 0;
38  }
```

6. Incorpore al archivo `util.c` una función llamada `swap` que intercambie el valor de dos variables enteras que pueden estar definidas en cualquier punto del programa. Escriba un programa que utilice dicha función

```
  /* Código en util.c */
2 /* Intercambia el valor dos variables enteras */
  void swap(int *px, int *py) {
4   int temp;

6   temp = *px;
    *px = *py;
8   *py = temp;
  }  /* Fin de código en util.c */
10
  #include <stdio.h>
12 #include "util.h"

14 int main() {
    int x, y;
16
    while (scanf("%d %d", &x, &y) == 2) {
18    printf("Enteros antes del intercambio: %d %d\n", x, y);
      swap(&x, &y);
20    printf("Enteros después del intercambio: %d %d\n", x, y);
    }
22
    return 0;
24 }
```

7. Copie en una única línea de la salida estándar los argumentos de la línea de ordenes

```
  /* Versión 1 */
2 #include <stdio.h>

4 int main (int argc, char *argv[]) {
    int i;
6
    for (i = 1; i < argc; i++)
8     printf("%s%s", argv[i], (i < argc-1) ? " " : "");

10  printf("\n");

12  return 0;
  }
14
  /* Versión 2 */
16 #include <stdio.h>

18 int main (int argc, char *argv[]) {
    while (--argc)
20    printf("%s%s", *++argv, (argc > 1) ? " " : "");
```

```
22      printf("\n");

24      return 0;
    }
```

8. Copiar el contenido de los ficheros indicados en la línea de ordenes en la salida estándar. Si no se especifica ningún argumento se debe copiar la entrada en la salida. Es decir, la funcionalidad del comando **cat** [<*archivo*>]*

```
1   #include <stdio.h>

3   void filecopy (FILE *ifp, FILE *ofp);

5   int main (int argc, char *argv[]) {
      FILE *fp;
7
      if (argc == 1) /* Sin argumentos, se copia la entrada en la salida */
9       filecopy(stdin, stdout);
      else
11      while (--argc > 0)
          if ((fp = fopen(*++argv, "r")) == NULL) {
13          /* Se envía el mensaje a la salida de error estándar */
            fprintf(stderr, "Error al abrir %s\n", *argv);
15          return 1;
          }
17        else {
            filecopy(fp, stdout);
19          fclose(fp);
          }
21
      return 0;
23  }

25  void filecopy (FILE *ifp, FILE *ofp) {
      int c;
27
      while ((c = getc(ifp)) != EOF)
29      putc(c, ofp);
    }
```

9. La salida del programa será una copia de las líneas de la entrada que contengan un patrón indicado en su argumento. El patrón consistirá exclusivamente en una secuencia de caracteres sin ningún metacarácter. El programa debe tener dos opciones con la sintaxis habitual de Unix (con el carácter inicial `'-'` y en cualquier combinación), `'x'` para copiar las líneas que no contengan el patrón y `'n'` para indicar al inicio de cada línea de salida con el nº de línea de la entrada seguido del carácter `':'` y la línea de entrada. Es decir, una funcionalidad similar al comando `grep` con las opciones `'v'` y `'n'`

```
  #include <stdio.h>
2 #include <string.h>
  #include "util.h"
4
  #define MAXLINE 1000
6
  int main (int argc, char *argv[]) {
8   int c;
    char line[MAXLINE];     /* Última línea leída */
10  long lineno = 0;        /* Nº de líneas leídas */
    int except = 0;         /* 0=No imprimir líneas con el patrón */
12  int number = 0;         /* 0=No imprimir el nº de línea */
    int found = 0;          /* Nº de líneas impresas */
14
    while (--argc > 0 && (*++argv)[0] == '-')
```

```
16        while ((c = *++argv[0]))
            switch (c) {
18            case 'x':
                except = 1;
20              break;
              case 'n':
22              number = 1;
                break;
24            default:
                fprintf(stderr, "Opción ilegal %c\n", c);
26              argc = 0;
                found = -1;
28              break;
            }
30
      if (argc != 1)
32        fprintf(stderr, "Uso: -x -n patron\n");
      else
34        while (leerlinea(line, MAXLINE) > 0) {
            lineno++;
36          if ((strstr(line, *argv) != NULL) != except) {
              if (number)
38              printf("%ld:", lineno);
              printf("%s", line);
40            found++;
            }
42        }

44    return found;
    }
```

10. El programa debe calcular el valor máximo, el valor mínimo y la media aritmética de una serie de números enteros obtenidos de la entrada. La salida del programa serán tres líneas con el siguiente formato:

```
Máximo: <valor máximo>
Mínimo: <valor mínimo>
Media: <valor medio>
```

Los valores máximo y mínimo deberán escribirse como enteros mientras que el valor medio se escribirá con dos cifras decimales

```
1   #include <stdio.h>

3   int main (int argc, char *argv[]) {
      int valor;          /* Último entero leído */
5     int min, max;       /* Entero mínimo y máximo actual */
      int leidos = 0;     /* Nº de enteros leídos */
7     float suma = 0.0;   /* Suma de los enteros leídos */

9     /* Se detiene el bucle en EOF o lectura de no entero */
      while (scanf("%d", &valor) == 1) {
11      ++leidos;
        suma += valor;
13      if (leidos == 1)  /* Primero entero leído */
            max = min = valor;
15      else {
          if (valor > max) max = valor;
17        if (valor < min) min = valor;
        }
19    }
```

```
21      if (leidos > 0) {
          printf("Máximo: %d\n", max);
23        printf("Mínimo: %d\n", min);
          printf("Media: %.2f\n", (suma / leidos));
25      }

27      return 0;
    }
```

11. El programa debe calcular todas las raíces de polinomios de segundo grado. La entrada será una secuencia de líneas que contendrá una serie de números reales que, tomados de tres en tres, se interpretarán como los coeficientes a, b y c de cada polinomio de la forma ax^2+bx+c. La salida del programa debe consistir en una línea por cada polinomio leído con el siguiente formato:

    ```
    Raíces de ax^2+bx+c : x1 x2
    ```

    Si x1 o x2 son valores reales se escribirán con tres cifras decimales y si son números complejos de la forma R+Ii donde R e I se escribirán con tres cifras decimales

```
  #include <stdio.h>
2 #include <stdlib.h>
  #include <math.h>
4
  int main (int argc, char *argv[]) {
6   float a, b ,c;       /* Coeficientes del polinomio */
    float d;             /* Discriminante del polinomio */
8   float real, imag;    /* Parte real e imaginaria de las raíces */

10  /* Se detiene el bucle en EOF o lectura distinta a 3 reales */
    while (scanf("%f %f %f", &a, &b, &c) == 3) {
12    d = b*b - 4*a*c;  /* Valor del discriminante */
      if (d >= 0)
14      printf("Raíces de ax^2+bx+c : %.3f %.3f\n", (-b + sqrt(d))/2*a,
          (-b - sqrt(d))/2*a);
16    else {
        real = -b/2*a;
18      imag = sqrt(abs(d))/(2*a);
        printf("Raíces de ax^2+bx+c : %.3f+%.3fi %.3f-%.3fi\n", real, imag, real, imag);
20    }
    }
22
    return 0;
24 }
```

12. Construir un programa llamado **formato** que imprima en la salida estándar el contenido de una serie de archivos cuyos nombres se pasan en la línea de órdenes. El programa admitirá las siguientes opciones:

    ```
    -m indicará que la salida deberá escribirse toda en minúsculas
    -M indicará que la salida deberá escribirse toda en mayúsculas
    ```

    Se podrá usar a lo más una de las opciones anteriores. Si no se indica ninguna de las anteriores la salida deberá hacerse copiando la entrada como esté (mayúscula o minúscula). Adicionalmente podrá tenerse una opción -n, donde n es un entero mayor que cero. Si se usa esta opción, n será el interlineado que deba usarse (1 indica espacio simple, 2 doble espacio, etc.) Si no está presente la opción anterior se imprimirá a espacio simple. La sintaxis de la línea de órdenes será

    formato [-<n>] [-m] [-M] [<*archivo*>]$^+$

```c
/* Versión 1 */
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <ctype.h>

int main (int argc, char *argv[]) {
  int c, i;
  int opcion_m = 0;        /* 0=No cambiar a minúsculas */
  int opcion_M = 0;        /* 0=No cambiar a mayúsculas */
  int opcion_n = 1;        /* Nº de líneas de separación */
  FILE *fp;

  while (--argc > 0 && (*++argv)[0] == '-')
    while ((c = *++argv[0]))
      switch (c) {
        case 'm':
          if (opcion_M != 0) {
            fprintf(stderr, "Opciones incompatibles -m -M\n");
            argc = 0;
          }
          else
            opcion_m = 1;
          break;
        case 'M':
          if (opcion_m != 0) {
            fprintf(stderr, "Opciones incompatibles -m -M\n");
            argc = 0;
          }
          else
            opcion_M = 1;
          break;
        default:
          if (isdigit(c)) {
            opcion_n = atoi(argv[0]);
            argv[0] =  argv[0] + strlen(argv[0]) - 1; /* Fin del argumento */
            if (opcion_n < 1) {
              fprintf(stderr, "Opción ilegal -%s\n", argv[0]);
              argc = 0;
            }
          }
          else {
            fprintf(stderr, "Opción ilegal %c\n", c);
            argc = 0;
          }
          break;
      }

  if (argc < 1) {
      fprintf(stderr, "Uso: [-<n>] [-m] [-M] <archivos>\n");
      return EXIT_FAILURE; /* Código estándar de fallo definido en stdlib.h */
  }
  else {
    while (argc-- > 0) {
      if ((fp = fopen(*argv++, "r")) == NULL) {
        fprintf(stderr, "Error al abrir %s", *argv);
        return EXIT_FAILURE;
      }
      while ((c = fgetc(fp)) != EOF) {
        if (opcion_m)
          c = tolower(c);
        if (opcion_M)
```

```
            c = toupper(c);
64          if (c != '\n')
              putchar(c);
66          else
              for (i = 0; i < opcion_n; i++)
68              putchar('\n');
          }
70        fclose(fp);
        }
72      return EXIT_SUCCESS; /* Código estándar de éxito definido en stdlib.h */
    }
74  }

76  /* Versión 2 */
    #include <stdio.h>
78  #include <string.h>
    #include <stdlib.h>
80  #include <ctype.h>

82  int main (int argc, char *argv[]) {
      char c;                   /* Carácter de opción */
84    int i, j;
      int opcion_m = 0;         /* 0=No cambiar a minúsculas */
86    int opcion_M = 0;         /* 0=No cambiar a mayúsculas */
      int opcion_n = 1;         /* Nº de líneas de separación */
88    FILE *fp;

90    for (i = 1; i < argc; i++) { /* Recorrido por los argumentos */
        if (argv[i][0] != '-')
92        break; /* El argumento no es una opción y salimos del bucle */
        else
94        for (j = 1; (c = argv[i][j]) != '\0'; j++) {
            switch (c) {
96            case 'm':
                if (opcion_M != 0) {
98                fprintf(stderr, "Opciones incompatibles -m -M\n");
                  argc = 0;
100             }
                else
102               opcion_m = 1;
                break;
104           case 'M':
                if (opcion_m != 0) {
106               fprintf(stderr, "Opciones incompatibles -m -M\n");
                  argc = 0;
108             }
                else
110               opcion_M = 1;
                break;
112           default:
                if (isdigit(c)) {
114               opcion_n = atoi(&argv[i][j]);
                  if (opcion_n < 1) {
116               fprintf(stderr, "Opción ilegal -%s\n", argv[i]);
                    argc = 0;
118               }
                }
120             else {
                  fprintf(stderr, "Opción ilegal %c\n", c);
122               argc = 0;
                }
124             break;
```

```
            }
126         }
      }
128   if (i == argc) {
        fprintf(stderr, "Uso: [-<n>] [-m] [-M] <archivos>\n");
130     return EXIT_FAILURE;
      }
132   else {
        for (; i < argc; i++) { /* Recorrido por la lista de archivos */
134       if ((fp = fopen(argv[i], "r")) == NULL) {
            fprintf(stderr, "Error al abrir %s", *argv);
136         return EXIT_FAILURE;
          }
138       while ((c = fgetc(fp)) != EOF) {
            if (opcion_m)
140           c = tolower(c);
            if (opcion_M)
142           c = toupper(c);
            if (c != '\n')
144           putchar(c);
            else
146           for (j = 0; j < opcion_n; j++)
                putchar('\n');
148       }
          fclose(fp);
150     }
        return EXIT_SUCCESS;
152   }
    }
```

13. Construir un programa llamado **frecuencia** que imprima en la salida estándar el nº de veces que aparece las distintas palabras de la entrada que empiecen por un carácter alfabético y el nº de la última línea donde apareció cada palabra. Se considerarán como separadores de palabras los caracteres blancos, los tabuladores horizontales y los caracteres de nueva línea. El programa debe calcular la frecuencia de cualquier número de palabras de la entrada y usar la mínima memoria posible. El formato de la salida consistirá en una línea por palabra distinta y en el mismo orden que aparecen en la entrada con la siguiente estructura:

    `<palabra> <frecuencia> <nº de la última línea>`

```
1   #include <stdio.h>
    #include <string.h>
3   #include <stdlib.h>
    #include <ctype.h>
5
    #define MAXPALABRA 256
7
    struct tnodo {
9     char *palabra;  /* Cadena de caracteres de la palabra */
      int frecuencia; /* Número de ocurrencias en la entrada */
11    int linea;      /* Línea de la última ocurrencia */
      struct tnodo *sig;  /* Siguiente nodo de la lista */
13  };

15  /* Obtiene la siguiente palabra, devuelve su longitud o EOF si no hay palabra*/
    /* Actualiza el nº de líneas leídas */
17  int obtenerpalabra(char *palabra, int limite, int *numlinea) {
      int c;
19    int i=0; /* Índice de recorrido por palabra */
```

```
21      if (limite < 2) return EOF; /* Espacio insuficiente en palabra */

23      /* Descarta los separadores iniciales */
        while ((c = getchar()) != EOF) {
25        if (c == '\n') (*numlinea)++;
          if (c != ' ' && c != '\n' && c != '\t') break;
27      }

29      if (c == EOF) return EOF; /* No hay más palabras en la entrada */

31      palabra[i++] = c;

33      /* Lectura del resto de caracteres de la palabra */
        while ((c = getchar()) != EOF) {
35        if (c == ' ' || c == '\n' || c == '\t') break; /* Fin de palabra */
          else {
37          palabra[i++] = c;
            if (i == (limite -1)) break; /* Fin de espacio en palabra */
39        }
        }
41
        if (c == '\n') ungetc(c, stdin); /* Se devuelve el último '\n' */
43      palabra[i] = '\0';

45      return i;
    }
47
    /* Crea un nodo para una palabra nueva */
49  struct tnodo *crearnodo(const char *palabra) {
      struct tnodo *ptr;
51    char *cad;

53    ptr = malloc(sizeof(struct tnodo));
      cad = malloc(strlen(palabra) + 1);
55    if ((ptr == NULL) || (cad == NULL)) {
        fprintf(stderr, "Error en asignación de memoria\n");
57      exit(1);
      }
59    ptr->palabra = cad;
      strcpy(ptr->palabra, palabra);
61    ptr->frecuencia = ptr->linea = 0;
      ptr->sig = NULL;
63
      return ptr;
65  }

67  /* Se actualiza/incorpora la palabra */
    /* Se devuelve un puntero a la primera palabra de la lista */
69  struct tnodo *sumarlista(struct tnodo *lista, const char *palabra, int numlinea) {
      struct tnodo *ptr, *ant;
71
      if ((lista == NULL)) {/* Lista vacia */
73      ptr = crearnodo(palabra);
        lista = ptr;
75    }
      else {
77      for (ptr = ant = lista; ptr != NULL; ant = ptr, ptr = ptr->sig)
          if (! strcmp(ptr->palabra, palabra)) break; /* Palabra encontrada */
79
        if (ptr == NULL) { /* Fin de lista sin encontrar palabra */
81        ptr = crearnodo(palabra); /* Palabra nueva */
          ant->sig = ptr; /* Enlace de la palabra nueva al final de la lista */
```

```
83          }
        }
85
        (ptr->frecuencia)++; /* Nueva ocurrencia de la palabra */
87      ptr->linea = numlinea; /* Línea de la última ocurrencia de la palabra */

89      return lista;
    }
91
    /* Se imprime en la salida la lista de palabras */
93  void imprimirlista(struct tnodo *lista) {
        struct tnodo *ptr;
95
        for (ptr = lista; ptr != NULL; ptr = ptr->sig)
97          printf("%s %d %d\n", ptr->palabra, ptr->frecuencia, ptr->linea);
    }
99
    int main() {
101     char palabra[MAXPALABRA]; /* Última palabra leída */
        struct tnodo *lista = NULL; /* Lista de palabras */
103     int numlinea=1; /* Línea actual */

105     while (obtenerpalabra(palabra, MAXPALABRA, &numlinea) != EOF) {
            if (isalpha(palabra[0])) /* Palabras con carácter inicial alfabético */
107             lista = sumarlista(lista, palabra, numlinea);
        }
109
        imprimirlista(lista);
111     return 0;
    }
```