Examen escrito de caracter práctico
Soluciones

Programación para sistemas

Julio 2020

- El presente examen lo componen **dos ejercicios** en los que tendrás que escribir sendos programas C.
- Durante la realización de los programas podrás observar que los dos ejercicios comparten funcionalidad. Te recomendamos que consideres reutilizar código de uno en el otro. No obstante tendrás que **entregar dos ficheros autocontenidos** sin referencia a otros ficheros.
- Las siguientes funciones de las bibliotecas estándares de C son **suficientes** para implementar los dos programas correctamente (usa el manual, `man`, cada una de ellas):
  - De `stdio.h`: `fopen`, `fgets`, y `fprintf` (o `fputs`).
  - De `stdlib.h`: `atoi`, `malloc` (o bien `calloc` o `realloc`), y `free`.
  - De `string.h`: `strdup`.
- **No se corregirá ninguna entrega que no compile** por lo tanto te recomendamos que tengas instalado un compilador de C y que realices las compilaciones de código con las opciones `-Wall -Wextra -Werror`
- En el enunciado podrás encontrar **ejemplos de invocación** de los programas y la **salida esperada** para cada invocación. Te recomendamos que **antes de realizar la entrega** te asegures de que tu implementación **funciona con todos los ejemplos**.

Página intencionalmente en blanco

Ejercicio 1: `intervalo` [6 puntos]

Tu tarea consiste en implementar un programa C denominado `intervalo.c` que lea líneas de texto de un fichero y escriba en la salida estándar sólo las líneas que se encuentran en un intervalo indicado en línea de comandos:

```
$ ./intervalo INICIO FIN [FICHERO]
```

El intervalo es un intervalo cerrado, es decir, se deberán mostrar desde la línea `INICIO` hasta la línea `FIN`, ambas inclusive. Los números o índices de líneas empiezan a contar por 1, es decir, la primera línea es la línea número 1.

`FICHERO` es un parámetro opcional que indica el nombre del fichero del que se leerán las líneas. Si `FICHERO` no aparece, se asume que las líneas se leen de la entrada estándar.

Se espera que tu programa haga cierto control de errores:

- Si el número de argumentos es incorrecto (menor de 2 o mayor de 3) el programa termina con código de error 1 y escribirá en la salida de error el siguiente mensaje:

  salida de error

  ```
  uso: ./intervalo INICIO FIN [FICHERO] (INICIO y FIN enteros positivos)
  ```

- Si `INICIO` o `FIN` no son enteros positivos el programa termina con código de error 2 y escribirá en la salida de error el siguiente mensaje:

  salida de error

  ```
  uso: ./intervalo INICIO FIN [FICHERO] (INICIO y FIN enteros positivos)
  ```

- Si el fichero `FICHERO` no se puede abrir en modo lectura el programa termina con código de error 3 y escribirá en la salida de error el siguiente mensaje (suponiendo que se hace la llamada con `inexistente.txt`):

  salida de error

  ```
  ./intervalo: error al abrir el fichero 'inexistente.txt'
  ```

A continuación puedes encontrar algunos ejemplos de uso incorrecto que tu implementación deberá detectar:

```
$ ./intervalo
$ ./intervalo 1
$ ./intervalo 1 2 3 4
```

```
$ ./intervalo A B
$ ./intervalo 1 B
$ ./intervalo A 2
$ ./intervalo -10 100
$ ./intervalo 10 -100
```

Nota: se puede asumir y no comprobar (en este apartado y en el siguiente), que los caracteres de las líneas son caracteres ASCII, representables en el tipo **`char`** del lenguaje C y que las líneas tienen menos de 2048 caracteres cambio de línea incluído.

Ejemplos de uso

Suponga que `entrada.txt` es un fichero que contiene las siguientes 5 líneas:

`entrada.txt`

```
Esta es la linea 1
y esta es la linea intermedia 2
y esta es la linea intermedia 3
y esta es la linea intermedia 4
y finalmente esta es la linea 5.
```

Se muestran seguidamente ejemplos de llamadas y las salidas correspondientes:

Ejemplo de llamada:

```
$ ./intervalo 2 4 entrada.txt
```

Salida esperada:

```
y esta es la linea intermedia 2
y esta es la linea intermedia 3
y esta es la linea intermedia 4
```

Ejemplo de llamada:

```
$ ./intervalo 3 4 entrada.txt
```

Salida esperada:

```
y esta es la linea intermedia 3
y esta es la linea intermedia 4
```

Ejemplo de llamada:

```
$ ./intervalo 4 4 entrada.txt
```

Salida esperada:

```
y esta es la linea intermedia 4
```

Ejemplo de llamada:

```
$ ./intervalo 4 3 entrada.txt
```

Salida esperada:

No se espera ninguna salida

Ejemplo de llamada:

```
$ ./intervalo 8 1000 entrada.txt
```

Salida esperada:

No se espera ninguna salida

Ejemplo de llamada:

```
$ ./intervalo 4 1000 entrada.txt
```

Salida esperada:

```
y esta es la linea intermedia 4
y finalmente esta es la linea 5.
```

Ejemplo de llamada:

```
$ seq 1 1000000 | \
./intervalo 500000 500001
```

Salida esperada:

```
500000
500001
```

Solución intervalo

```c
#include<stdio.h>
#include<stdlib.h>
#define MAX_LONG 2048

int main(int argc, char *argv[]) {
  FILE *entrada = stdin;
  char linea[MAX_LONG];
  char *endi, *endf;
  int inicio, fin;
  int i;

  if (argc < 3 || argc > 4) {
    fprintf(stderr,
            "uso: %s INICIO FIN [FICHERO] (INICIO y FIN enteros positivos)\n",
            argv[0]);
    return 1;
  }

  inicio = strtol(argv[1], &endi, 10);
  fin = strtol(argv[2], &endf, 10);
  if (endi == NULL || endf == NULL || inicio < 1 || fin < 1) {
    fprintf(stderr,
            "uso: %s INICIO FIN [FICHERO] (INICIO y FIN enteros positivos)\n",
            argv[0]);
    return 2;
  }

  if (argc == 4) {
    entrada = fopen(argv[3], "r");
    if (entrada == NULL) {
      fprintf(stderr, "%s: error al abrir el fichero '%s'\n", argv[0], argv[3]);
      return 3;
    }
  }

  if (fin < inicio) {
    fclose(entrada);
    return 0;
  }

  /* Saltar hasta inicio o EOF */
  for (i = 1; i < inicio && fgets(linea, MAX_LONG - 1, entrada); i++);

  if (i < inicio) {
    fclose(entrada);
    return 0;
```

```
47    }
48
49    for (; i <= fin && fgets(linea, MAX_LONG - 1, entrada); i++) {
50      fprintf(stdout, "%s", linea);
51    }
52
53    fclose(entrada);
54
55    return 0;
56  }
```

Ejercicio 2: `inverso` [4 puntos]

Tu tarea consiste en implementar un programa C denominado `inverso.c` que se comporte como el programa `intervalo.c` del ejercicio anterior pero que muestre las líneas del fichero de entrada dentro del intervalo en orden inverso.

Se espera que tu programa haga uso de memoria dinámica para almacenar las líneas del fichero de entrada a mostrar antes de escribirlas en salida estándar. Además del control de errores que ya se realizaba en el ejercicio anterior (`intervalo.c`), deberás añadir las siguientes:

- Tu programa deberá liberar la memoria dinámica antes de terminar.
- Si en algún momento surge algún problema al solicitar memoria el programa termina con código de error 8 y escribirá en la salida de error el siguiente mensaje:

salida de error

```
./inverso: no se ha podido solicitar memoria
```

Ejemplos de uso

Se muestran seguidamente ejemplos de llamadas, empleando el fichero `entrada.txt` mencionado en el ejercicio anterior, y las salidas correspondientes:

Ejemplo de llamada:

```
$ ./inverso 2 4 entrada.txt
```

Salida esperada:

```
y esta es la linea intermedia 4
y esta es la linea intermedia 3
y esta es la linea intermedia 2
```

Ejemplo de llamada:

```
$ ./inverso 3 4 entrada.txt
```

Salida esperada:

```
y esta es la linea intermedia 4
y esta es la linea intermedia 3
```

Ejemplo de llamada:

```
$ ./inverso 4 4 entrada.txt
```

Salida esperada:

```
y esta es la linea intermedia 4
```

Ejemplo de llamada:

```
$ ./inverso 4 3 entrada.txt
```

Salida esperada:

No se espera ninguna salida

Ejemplo de llamada:

```
$ ./inverso 8 1000 entrada.txt
```

Salida esperada:

No se espera ninguna salida

Ejemplo de llamada:

```
$ ./inverso 4 1000 entrada.txt
```

Salida esperada:

```
y finalmente esta es la linea 5.
y esta es la linea intermedia 4
```

Ejemplo de llamada:

```
$ seq 1 1000000 | \
./inverso 500000 500001
```

Salida esperada:

```
500001
500000
```

Solución inverso

```c
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#define MAX_LONG 2048

int main(int argc, char *argv[]) {
  FILE *entrada = stdin;
  char linea[MAX_LONG];
  char **lineas;
  int inicio, fin;
  int i, n;

  if (argc < 3 || argc > 4) {
    fprintf(stderr,
            "uso: %s INICIO FIN [FICHERO] (INICIO y FIN enteros positivos)\n",
            argv[0]);
    return 1;
  }

  inicio = atoi(argv[1]);
  fin = atoi(argv[2]);
  if (inicio < 1 || fin < 1) {
    fprintf(stderr,
            "uso: %s INICIO FIN [FICHERO] (INICIO y FIN enteros positivos)\n",
            argv[0]);
    return 2;
  }

  if (argc == 4) {
    entrada = fopen(argv[3], "r");
    if (entrada == NULL) {
      fprintf(stderr, "%s: error al abrir el fichero '%s'\n", argv[0], argv[3]);
      return 3;
    }
  }

  n = fin - inicio + 1;

  if (n < 1) {
    fclose(entrada);
    return 0;
  }

  /* Saltar hasta inicio o EOF */
  for (i = 1; i < inicio && fgets(linea, MAX_LONG - 1, entrada); i++);

```

```
47    if (i < inicio) {
48      fclose(entrada);
49      return 0;
50    }
51
52    lineas = (char **)calloc(sizeof(char *), n);
53
54    if (lineas == NULL) {
55      fclose(entrada);
56      fprintf(stderr, "%s:␣no␣se␣ha␣podido␣solicitar␣memoria\n", argv[0]);
57      return 8;
58    }
59
60    for (i = 0; i < n && fgets(linea, MAX_LONG - 1, entrada); i++) {
61      lineas[i] = strdup(linea);
62    }
63
64    fclose(entrada);
65
66    while(i > 0) {
67      i--;
68      fprintf(stdout, "%s", lineas[i]);
69      free(lineas[i]);
70    }
71
72    free(lineas);
73
74    return 0;
75  }
```