Cuaderno de prácticas

105000016 - Programación para Sistemas

Grado en Ingeniería Informática

Lenguajes y Sistemas Informáticos e Ingeniería de Software
Facultad de Informática
Universidad Politécnica de Madrid

Curso 2014-2015
Revisión SVN –revision–/ 2014-10-24

Índice

1. Información general para las prácticas

La asignatura se divide en **dos grandes prácticas** de acuerdo con los dos bloques de la asignatura: Bash y C. Cada práctica está compuesta de **una o varias tareas**. Los alumnos realizarán **entregas de cada tarea**.

Las prácticas se realizarán **obligatoriamente en grupo (dos personas)**. La realización de las tareas de las prácticas corresponde al grupo, por lo que la nota será del grupo y no individual. La evaluación de cada tarea se realizará de acuerdo a los puntos siguientes, caso de estar incluidos en la misma:

- **Funcionamiento correcto** de los programas entregados.
- **Calidad del contenido y presentación** de los entregables de la tarea.

1.1. Sistema de entrega

Cada grupo dispondrá en la máquina UNIX del Centro de Cálculo denominada `triqui`, de sus cuentas personales accesibles con la misma identidad y palabra clave que otros sistemas del Centro de Cálculo. Los alumnos podrán utilizar la VPN de la Facultad para acceder a `triqui` y el acceso deberá realizarse a través de SSH.

Para poder realizar la entrega de una tarea, el alumno tendrá que tener los **ficheros a entregar en un directorio determinado** de `triqui`, un directorio que identifica la asignatura, la tarea y el curso al que pertenece. Por ejemplo, para la tarea `tarea-1.1`, los ficheros a entregar deberán estar en el directorio

```
~/pps/tarea-1.1.2014-2015/
```

donde `~` es el directorio personal (`$HOME`) del alumno. Este directorio se crea directamente al desempaquetar el software de apoyo (sección 1.3) en el directorio personal del alumno.

Una vez que los ficheros que se desea entregar se encuentren en dicho directorio, **deberá ejecutarse el siguiente mandato** en `triqui`:

```
entrega.pps tarea-1.1
```

Hay que **asegurarse que la entrega se realiza correctamente**. En caso de error, la utilidad de entrega lo indicaría con un error en el momento de la entrega.

El sistema de entrega, si es el caso, realizará una serie de **pruebas (*tests*) sobre la entrega a una hora determinada**. El resultado de dichos tests será **comunicado por correo electrónico a la cuenta de correo del estudiante en `triqui`**.

1.2. Normas de entrega y presentación

Cada tarea exige la entrega de una serie de *entregables* (ficheros). Cada entregable debe ser incluido en el formato especificado en el enunciado de la tarea, además de los dos ficheros especificados a continuación. La codificación de caracteres de todos los ficheros deberá ser UTF8 por lo que debes configurar el editor de texto que uses para que la codificación de caracteres sea la indicada[1].

Además de los ficheros específicos de cada tarea, cada entrega ha de incluir obligatoriamente los dos ficheros siguientes:

[1] El programa `iconv` convierte entre diferentes codificaciones y casi todos los editores de texto pueden personalizarse para que soporten y generen la codificación deseada.

autores.txt Fichero con los datos de los autores de la tarea. Cada integrante del grupo debe ocupar una línea en dicho fichero. Cada línea presentará varios campos de caracteres separados por el caracter "`:`". Los campos son número de matrícula, primer apellido, segundo apellido y nombre. Veamos un ejemplo:

```
910347:García:de la Torre:María Dolores
930019:Hernado:Pulido:Isabel
```

bitacora.txt Fichero que documenta el material que se entrega y ha de usarse a modo de bitácora del desarrollo de la tarea. Debería contener los siguientes puntos (no necesariamente con esta estructura mostrada):

- Título de la tarea.
- Autores
- Fecha de cada entrega realizada con comentario indicando el nivel de completitud.
- Índice de contenidos con referencias al propio documento.
- Sobre del desarrollo:
  - Descripción general de los trabajos realizados.
  - Definición de términos o conceptos relevantes o nuevos.
  - Problemas encontrados y resolución de los mismos.
  - Conocimientos relevantes adquiridos.
  - Bibliografía y referencias utilizadas, con comentarios sobre su interés.
  - Tiempo dedicado en la realización.
- Comentarios personales positivos o negativos sobre el contenido o método de realización de la tarea.

Este documento, para ser de calidad, debería cumplir los siguientes requisitos:

- Presentar una estructura lógica en sus contenidos.
- Estar convenientemente formateada, para facilitar su lectura.
- Tratar con claridad y con suficiencia cada tema abordado.
- NO incluir código fuente ni el contenido de otros entregables.

1.3. Código fuente de apoyo

El material de apoyo para la realización de las tareas que así lo requieran puede obtenerse del directorio

```
~pps/pub
```

donde `~pps` es el directorio de la asignatura. En dicho directorio encontrará ficheros cuyo nombre identifica la tarea a la que pertenecen. Descomprima y desempaquete dichos ficheros en su directorio personal con el siguiente mandato:

cat `~pps/pub/`*idtarea*`.2014-2015.tar.gz | tar xvfz -`

mandato equivalenta a este otro:

`$ tar xvfz ~pps/pub/idtarea.2014-2015.tar.gz`

y donde *idtarea* representa el identificador de la tarea de la que se quiera extraer el código de apoyo.

1.4. Recomendaciones generales

Edición de programas y documentación

Tanto para la preparación de los programas como para la preparación de la documentación **se exige el uso de editores de texto plano**. En UNIX disponemos de algunos *clásicos*:

- *nano*: Un editor sencillo muy fácil de manejar.
- *vi*: El editor "estándar" de UNIX.
- *emacs*: El editor de GNU que posee una gran potencia.

Por supuesto, existen editores sencillos con interfaz gráfica y coloreado de programas son *gedit*, *Scite*, *Kate*. También existen entornos gráficos de desarrollo de software cuya utilidad se hace más patente en grandes proyectos. Algunos bastante utilizados y no demasiado complejos pueden ser *Kdeveloper*, *Anjuta* o *Code::Blocks*.

Control de versiones

Para facilitar el trabajo en equipo se puede utilizar un sistema de control de versiones. En `triqui` es posible utilizar *Subversion* o *Git*.

Manual

En caso de duda sobre cualquier mandato o servicio del sistema operativo UNIX utilice el mandato *man*, para obtener ayuda.

Pensar antes de actuar

Antes de empezar una tarea, lea detalladamente:

- El enunciado de la misma para obtener una idea clara de qué es concretamente lo que se le pide.
- El material de lectura obligatoria. En caso de duda consulte el material recomendado y con los profesores.

Antes de entregar una tarea, verifique concienzudamente:

- El correcto funcionamiento de la misma.
- Que cumple las Normas de Presentación indicadas a continuación.
- Que incluye todos los entregables listados en la descripción de la tarea, además de los dos ficheros obligatorios `autores.txt` y `bitacora.txt`.

1.5. Protección frente a copias

Un objetivo fundamental del proyecto es conseguir que el alumno, con su realización, obtenga una experiencia que le será muy útil en su vida profesional. Persiguiendo esta meta fundamental, prestaremos especial interés en detectar los intentos de copia total o parcial de las tareas del proyecto.

Para evitar problemas de confidencialidad y posibles copias, deberá tenerse especial cuidado en proteger las cuentas de accesos no deseados. Para ello se deben seguir las siguientes pautas:

- Asegúrese de que su palabra clave es suficientemente *fuerte* como para que un programa (mediante un diccionario) o una persona no puedan deducirla fácilmente: escoja una palabra clave que tenga caracteres especiales para que la protección sea más efectiva. No le diga su palabra clave a nadie.
- Proteja el directorio home de su cuenta contra todo tipo de acceso de cualquier otro usuario usando el mandato `chmod 700 $HOME`.
- Cambie la mascara de creación de ficheros para proteger cualquier fichero que cree en su cuenta. Para ello añada al final del fichero `.bashrc` el mandato **`umask`** `077`.
- No saque listados si no puede esperar a recogerlos.
- Si sospecha que le han robado un listado o fichero con la solución de alguna tarea, comuníquelo al profesor de la misma utilizando correo electrónico (mandato mail) y, **muy importante**, adjunte una copia de la versión extraviada.

1.6. Tutorías, consultas y notificaciones

La mayoría de las consultas deberán resolverse en las clases de trabajo en grupo, puesto que permiten resolver de forma eficiente dudas o cuestiones que afecten a más de un grupo.

Los horarios de tutorías estarán expuestos en el tablón y/o en la hoja web de la asignatura. Los alumnos deberán respetar los horarios de tutorías.

Las notificaciones a los alumnos se harán mediante notas en el tablón de anuncios, plataforma *Moodle* o mediante correo electrónico, por lo que los alumnos deberán estar atentos a dichos medios de comunicación.

2. Práctica 0: Formación de grupo

La primera práctica consiste en la formación de grupo. Para ello los alumnos deberán decidir quién será su pareja para realizar el resto de las prácticas. Una vez elegido el compañero o compañera de grupo habrá que realizar una única tarea: el alta de grupo.

2.1. `tarea-0.1`: Alta de grupo

Ficheros a entregar

`autores.txt`: según normas de presentación en 1.2.

3. Práctica 1: Unix shell

Esta práctica consiste en realizar un *script* en Bash que le permitirá afianzar sus conocimientos sobre este lenguaje. El *script* implementará un supuesto mecanismo de entrega automática de prácticas similar al que se usa en esta asignatura. Las ideas principales son:

- El *script* a desarrollar entenderá que cada *práctica* tiene un *identificador de práctica*.
- El *script* utilizará el *identificador de práctica* para conocer la información relacionada con dicha *práctica*: fecha límite de entrega, ficheros a entregar y directorio de entrega. Información que habrá que extraer de un fichero con el mismo nombre que el *identificador de práctica*.
- La labor principal del *script* será copiar los ficheros apropiados de un directorio (del supuesto usuario del *script*) a otro directorio (directorio de entrega).

3.1. `tarea-1.1`: *Script* `minientrega.sh`

El script deberá ser ejecutado mediante el siguiente mandato:

```
minientrega.sh ID_PRACTICA
```

`ID_PRACTICA` es una ristra de caracteres que representa el hipotético nombre de una práctica (por ejemplo `prac1`, `prac7`, `bucles`, etc.).

El *script* comprobará que el *identificador de práctica* (`ID_PRACTICA`) es válido, comprobará que la entrega no está fuera de plazo, comprobará que existen los ficheros exigidos para la *práctica* identificada por ese *identificador de práctica* y finalmente copiará todos los ficheros a un directorio adecuado. El *script* devolverá 0 como código de error para indicar que todo funcionó correctamente o un código de error distinto de 0 indicando qué error se produjo. El *script* no esperará ninguna información a través de la entrada estándar ni provocará la emisión de mensajes en la salida estándar ni en la salida de error excepto los propios de la ayuda (ver la sección "Opciones de los programas" en este documento) o de la invocación incorrecta (en este caso, sin argumentos o con más de uno). Más detalles:

- Si el *script* ha sido invocado sin argumentos o con más de uno se informará al usuario con *exit status* 64 y un mensaje de dos líneas en la salida de error siguiendo el formato mostrado por este ejemplo:

  ```
  $ minientrega.sh
  minientrega.sh: Error(EX_USAGE), uso incorrecto del mandato. "Success"
  minientrega.sh+ <<descripción detallada del error>>
  ```

- La variable de entorno `MINIENTREGA_CONF` debe estar definida antes de ejecutar el *script*. Dicha variable contiene el nombre de un directorio en el que existen varios ficheros de configuración de *prácticas*. En caso de que la variable no represente un directorio legible se informará al usuario con *exit status* 64 (**Nota:** vuestro script no debe definir dicha variable).
- Cada nombre de fichero de configuración de *práctica* (`${MINIENTREGA_CONF}/*`) que sea legible es un *identificador válido de práctica*. Si el *identificador de la práctica* (`ID_PRACTICA`) no es válido se informará al usuario con un *exit status* 66.
- Cada fichero en `${MINIENTREGA_CONF}/*` define las siguientes variables de entorno en formato compatible con Bash:

- `MINIENTREGA_FECHALIMITE`: contiene la fecha de entrega límite de entrega de la *práctica* identificada por el *identificador de práctica* `ID_PRACTICA`. La fecha vendrá codificada siguiendo el formato *Año-Mes-Día* (ejemplo: el 18 de octubre de 2011 se condifica como `2011-10-18`). Se informará al usuario si el contenido no es una fecha correcta o si la entrega se está realizando fuera de plazo con un *exit status* 65.
- `MINIENTREGA_FICHEROS`: contiene la lista de ficheros que deben ser entregados y que deben estar en el directorio de trabajo desde el que se ejecuta el *script*. En caso de que alguno de los ficheros no exista o no sea legible se informará al usuario con un *exit status* 66.
- `MINIENTREGA_DESTINO`: directorio de destino en el que se debe crear un directorio de entrega para la *práctica* identificada por el *identificador de práctica* `ID_PRACTICA`. El nombre del directorio a crear será el nombre del usuario que ejecuta el *script* (`${USER}`). En caso de que el directorio de destino no exista o el directorio de entrega no pueda crearse se informará al usuario con un *exit status* 73.

Opciones de los programas

La única opción que debe soportar el *script* es la solicitud de ayuda:

`-h` | **`--help`**
: El programa debe emitir por la **salida estándar** un texto breve indicando qué hace el programa y qué parámetros admite. A continuación, el programa debe terminar correctamente, esto es, con valor de terminación cero. El texto a emitir deberá seguir el siguiente formato:

```
$ minientrega.sh -h
minientrega.sh: Uso: <<describrir esquema de llamada>>
minientrega.sh: <<describir en una línea lo que hace>>
```

Ficheros de apoyo

En el fichero comprimido con los ficheros de apoyo de la práctica se puede encontrar una pequeña estructura de directorios que contiene un ejemplo de configuración de *práctica*. Para su uso debería ser suficiente la lectura del fichero `COMO.txt`.

Debe utilizarse el fichero de configuración `conf/prac1` como ejemplo para vuestras propias pruebas. Para ello es necesario que se respete el formato de uso del entrecomillado y de la fecha.

Ficheros a entregar

`autores.txt` y `bitacora.txt`: ficheros siguiendo las normas de presentación en 1.2.

`minientrega.sh`: el *script* incluyendo comentarios.

`man-minientrega.txt`: manual de usuario del *script*.

`test-minientrega.txt`: fichero de texto con una descripción de los tests realizados para validar el *script*.

Consulte el Manual

`man 1 bash`
Familiarizarse con las principales subsecciones de la página de manual de Bash.

`man 1 test`
Para realizar comprobaciones sobre ficheros, expresiones aritméticas, strings, etc.

`man 1 [`
Idéntico a `test`.

`man 1 date`
Para obtener la fecha actual en diferentes formatos.

`help if`
Control de flujo `if` en Bash.

`help [[`
Ejecuta una orden condicional en Bash.

`help for`
Control de flujo `for` en Bash.

`help source`
Para ejecutar órdenes de un fichero en el shell actual.

Evaluación

La entrega de los ficheros es obligatoria. La valoración total de la práctica será de 100 puntos.
Se pueden realizar pruebas adicionales a las ya realizadas mediante el tester.

Los profesores podrían pedir una entrevista con los componentes de los grupos si lo consideraran necesario. Dichas entrevistas durarían entre 5 y 10 minutos con cada grupo durante la cuál se realizarían algunas preguntas sobre el diseño del *script* y el conocimiento del mismo.

La puntuación final de la práctica viene dada por el resultado de las pruebas multiplicado por un factor, que depende de la calidad de la documentación (bitacora.txt, man-minientrega.txt, test-minientrega.txt) incluído el fichero fuente minientrega.sh . Esto puede suponer perder hasta un 20 % de la nota de las pruebas.

4. Práctica 2: Programación C

Se desea que realice una serie de pequeños programas en C que le permitirán afianzar sus conocimientos sobre este lenguaje. En las siguientes secciones se describirá en detalle cada uno de los programas a realizar.

4.1. Instrucciones

En la presente sección se recogen ciertos aspectos del comportamiento general, que serán de aplicación a todos los programas a realizar.

Identificación de los programas

Cada programa será identificado por el nombre que recibirá su ejecutable (ej. "`secuencia`"). Cada programa derivará de un único fichero fuente en C del el mismo nombre, pero con extensión "`.c`" (ej. "`secuencia.c`").

Argumentos de los programas

Todo programa es invocado escribiendo el nombre del ejecutable correspondiente. A la derecha de este nombre (y separado por blancos o tabuladores) podemos escribir otros textos, que denominamos argumentos. Estos argumentos estarán disponibles para ser usados por el proceso que se ejecuta (el programa en ejecución).

Si el programa fue escrito en C, los argumentos podrán ser accedidos a través de los dos primeros parámetros formales de la función principal:

```
main(int argc, char*argv[], char*envp[])
```

La manera en que un programa utiliza sus argumentos puede ser muy variada, pero generalmente se utilizan para parametrizar su propio comportamiento.

A continuación se describe la sintaxis empleada en la descripción de los argumentos de los programas.

`-x`
: Esta es la notación corta, un signo menos ('-') y una letra (en este caso una 'x'). Se indica una opción del programa.

`--help`
: Esta es la notación larga, dos signos menos ('--') y una palabra (en este caso '`help`'). Se indica una opción del programa.

`-t` *segundos*
: Las opciones se usan para condicionar el comportamiento del programa. Algunas opciones pueden indicar cómo interpretar el siguiente argumento como parámetro del programa.

`[arg]`
: Mediante esta notación se indica una parte opcional que puede aparecer o no.

`arg...`
: Mediante esta notación se indica una parte que puede aparecer una o varias veces.

Opciones de los programas

De forma general, todos los programas que se pide que usted desarrolle debe admitir la siguiente opción.

`-h` | `--help`
: Todo programa debe reconocer el caso en que sea invocado con esta opción como único argumento. El usuario está solicitando ayuda. El programa debe emitir por la **salida estándar** un texto breve indicando qué hace el programa y qué parámetros admite. A continuación, el programa debe terminar correctamente, esto es, con valor de terminación cero.

Valores de terminación de los programas

No existe acuerdo en el significado de los valores de terminación exceptuando que todo programa que termina correctamente, lo hace con el valor de terminación cero (`man 3 exit`).

No obstante, para esta práctica, intentaremos adoptar el intento de estandarización realizado por BSD con `<sysexits.h>`.

Todo programa debe realizar ciertas comprobaciones sobre la corrección de sus argumentos, datos de entrada, servicios que utiliza, etc. Si alguna comprobación falla, debe emitir por el **estándar error** un mensaje de error de dos líneas con el siguiente formato:

```
"%s: Error(%s), %s.\n"
"%s+ %s.\n"
```

Cada línea irá prefijada con el nombre del programa que la emite. La primera indicará el código simbólico del error y una descripción genérica del mismo. La segunda línea presentará una descripción detallada de la razón de dicho error.

Ejemplo

```
secuencia: Error(EX_USAGE), uso incorrecto del mandato. "Success"
secuencia+ El signo de "paso" no permite recorrer el intervalo en ...
```

A continuación, el programa debe terminar con el valor de terminación indicado.

Recomendaciones generales

- Allá donde se le sugiera que consulte el manual, consúltelo. Le ahorrará mucho tiempo.
- Respete el tipo del texto (mayúsculas y minúsculas) allá donde, en este documento, se presente un formato, un nombre de fichero, de variable, etc.
- Sea sumamente estricto con los formatos de entrada y salida mencionados en este documento. Sólo se podrá evaluar correctamente su práctica si se ajusta a los formatos indicados.

4.2. Evaluación

La entrega de los ficheros es obligatoria. La valoración total de la práctica será de 100 puntos.

Se pueden realizar pruebas adicionales a las ya realizadas mediante el tester.

Los profesores podrían pedir una entrevista con los componentes de los grupos si lo consideraran necesario. Dichas entrevistas durarían entre 5 y 10 minutos con cada grupo durante la cuál se realizarían algunas preguntas sobre el diseño de los *programas* y el conocimiento de los mismos.

La puntuación final de la práctica viene dada por el resultado de las pruebas multiplicado por un factor, que depende de la calidad de la documentación (bitacora.txt y demás ficheros pedidos) incluídos los ficheros fuente. Esto puede suponer perder hasta un 20 % de la nota de las pruebas.

4.3. Documentación común a entregar

Para su desarrollo, esta práctica se divide en tres tareas (`tarea-2.`n donde n será 1, 2 ó 3 para indicar cada tarea) con fecha de entrega independiente. Los siguientes ficheros son de entrega obligatoria para cada tarea:

`autores.txt`
: Descrito en la sección 1.2.

`bitacora.txt`
: Descrito en la sección 1.2.

`auxiliar.c`
: Este fichero fuente podrá ser utilizado para contener funciones auxiliares, útiles para varios de los programas. Si no va a utilizar este fichero, debe dejarlo tal cual lo encuentre.

`auxiliar.h`
: Este fichero de cabecera deberá contener los prototipos de las funciones auxiliares definidas en `auxiliar.c`. Los programas que deseen utilizar dichas funciones deberán incluir este fichero de cabecera. Si no va a utilizar este fichero, debe dejarlo tal cual lo encuentre.

4.4. `tarea-2.1`

Ficheros extras a entregar

Con cada entrega parcial de esta tarea se recogerán además de los ficheros indicados en la sección 4.3:

`delreves.c`
: Fichero fuente correspondiente al programa `delreves`. Aunque no se vaya a realizar este programa, este fichero deberá existir.

4.4.1. Programa `delreves`

```
delreves [ fichero... ]
```

Puede recibir cualquier número de nombres de fichero de tipo `char *`.

Descripción

Este programa procesa cada fichero en el orden indicado (o en su defecto, la **entrada estándar**) leyendo cada línea del mismo (considerando un máximo de 2048 caracteres por línea) y emite por su **salida estándar** dichas líneas previa inversión del contenido de las mismas (contenido idéntico pero leído de derecha a izquierda).

Valores de Terminación

EX_OK Terminación correcta.

EX_NOINPUT El fichero "FICHERO"[2] no puede ser leído.

Ejemplos

```
$ delreves -h
delreves: Uso: delreves [ fichero... ]
delreves: Invierte el contenido de las líneas de los ficheros (o de la entrada).
$ echo "abcdefg" | delreves
gfedcba
$ cat fich1
uno
dos
tres
cuatro
$ cat fich2
A
Be
Ce
De
E
$ delreves fich1 fich2
onu
sod
sert
ortauc
A
eB
eC
eD
E
$
```

Consulte el Manual

man 3 fprintf
Para emitir mensajes con formato.

man 3 fopen
Para abrir ficheros.

man 3 fgets
Para leer líneas de texto.

man 3 fputs
Para escribir líneas de texto.

[2] FICHERO representa el nombre real del fichero.

man 3 fclose
Para cerrar los ficheros abiertos.

4.5. tarea-2.2

Ficheros extras a entregar

Con cada entrega parcial de esta tarea se recogerán además de los ficheros indicados en la sección 4.3:

secuencia.c
Fichero fuente correspondiente al programa `secuencia`. Aunque no se vaya a realizar este programa, este fichero deberá existir.

pruebas.sh
Shell script que sea capaz de evaluar la funcionalidad de los programas realizados. Deberá seguir el modelo dado para `ejemplo.c`.

pruebas.txt
Documento que cuente de forma más completa y legible la especificación de las pruebas planteadas y efectivamente implementadas en `pruebas.sh`.

4.5.1. Programa secuencia

```
secuencia [ hasta [ desde [ paso ]]]
```

Todos estos argumentos contienen un valor de tipo real y el valor por defecto para ellos es: `hasta = 10`, `desde = 1` y `paso = 1`.

Descripción

Este programa genera por su **salida estándar** la secuencia de los números que van desde el valor `desde`, hasta el valor `hasta` (sin superarlo), avanzando en el sentido y al paso indicados por `paso`.

Este programa genera un número por línea con el siguiente formato: `"\t %g\n"` (sin espacios).

Entorno

Utiliza la variable de entorno `MAX_OUTPUT` para controlar que, en ningún caso, se produzcan más de esta cantidad de números. Si esta variable no existe o no contiene un valor entero no negativo, se debe asumir el valor por defecto de 100.

Valores de Terminación

EX_OK Terminación correcta.

EX_USAGE El parámetro `"paso"` no es un número real válido.

EX_USAGE El parámetro `"paso"` no puede valer 0.

EX_USAGE El parámetro `"desde"` no es un número real válido.

EX_USAGE El parámetro `"hasta"` no es un número real válido.

EX_USAGE El signo de "paso" no permite recorrer el intervalo en el sentido "desde" a "hasta".

EX_USAGE El número de argumentos no es correcto.

EX_NOPERM Se intentó superar el límite de salida establecido por MAX_OUTPUT.

Ejemplos

```
$ secuencia -h
secuencia: Uso: secuencia [ hasta [ desde [ paso ]]]
secuencia: Genera la secuencia de números en el intervalo y paso indicados.
$ secuencia
        1
        2
        3
        4
        5
        6
        7
        8
        9
        10
$ secuencia 5
        1
        2
        3
        4
        5
$ secuencia 5 3
        3
        4
        5
$ secuencia -0.9 2 -0.5
        2
        1.5
        1
        0.5
        0
        -0.5
$ secuencia 10-1 1
secuencia: Error(EX_USAGE), uso incorrecto del mandato. "Success"
secuencia+ El parámetro "hasta" no es un número real válido.
$ secuencia 1 2 3 4
secuencia: Error(EX_USAGE), uso incorrecto del mandato. "Success"
secuencia+ El número de argumentos no es correcto.
$ secuencia 10 1 -1
secuencia: Error(EX_USAGE), uso incorrecto del mandato. "Success"
secuencia+ El signo de "paso" no permite recorrer el intervalo en el sentido "desde"
$ secuencia 105
        1
```

```
        2
        ...
        100
secuencia: Error(EX_NOPERM), permiso denegado. "Success"
secuencia+ Se intentó superar el limite de salida.
$
```

Consulte el Manual

`man 3 fprintf`
Para emitir mensajes con formato.

`man 3 strtod`
Para extraer el `double` contenido en una tira de caracteres.

`man 3 strtol`
Para extraer el `long` contenido en una tira de caracteres.

`man 3 getenv`
Para acceder al valor de una variable de entorno.

4.6. `tarea-2.3`

Ficheros extras a entregar

Con cada entrega parcial de esta tarea se recogerán además de los ficheros indicados en la sección 4.3:

`bocabajo.c`
Fichero fuente correspondiente al programa `bocabajo`. Aunque no se vaya a realizar este programa, este fichero deberá existir.

`pruebas.sh`
Shell script que sea capaz de evaluar la funcionalidad de los programas realizados. Deberá seguir el modelo dado para `ejemplo.c`.

`pruebas.txt`
Documento que cuente de forma más completa y legible la especificación de las pruebas planteadas y efectivamente implementadas en `pruebas.sh`.

4.6.1. Programa `bocabajo`

```
bocabajo [ fichero... ]
```

Puede recibir cualquier número de nombres de fichero de tipo `char *`.

Descripción

Este programa procesa cada fichero en el orden indicado (o en su defecto, la **entrada estándar**) leyendo cada línea del mismo (considerando un máximo de 2048 caracteres por línea) y emite por su **salida estándar** dichas líneas pero en orden inverso, es decir, la primera línea del primer fichero deberá ser la última mostrada, y la primera mostrada la última del último fichero.

Valores de Terminación

EX_OK Terminación correcta.

EX_NOINPUT El fichero "FICHERO" no puede ser leído.

EX_OSERR No se pudo ubicar la memoria dinámica necesaria.

Ejemplos

```
$ bocabajo -h
bocabajo: Uso: bocabajo [ fichero... ]
bocabajo: Invierte el orden de las líneas de los ficheros (o de la entrada).
$ cat fich1
uno
dos
tres
cuatro
$ cat fich2
A
Be
Ce
De
E
$ bocabajo fich1 fich2
E
De
Ce
Be
A
cuatro
tres
dos
uno
$
```

Consulte el Manual

man 3 fprintf
Para emitir mensajes con formato.

man 3 fopen
Para abrir ficheros.

man 3 fgets
Para leer líneas de texto.

man 3 fputs
Para escribir líneas de texto.

man 3 fclose
Para cerrar los ficheros abiertos.

`man 3 malloc`
: Para ubicar memoria dinámica.

`man 3 strdup`
: Replicar una tira de caracteres en memoria dinámica.