En cada ejercicio se evaluará la eficiencia del código, el uso de identificadores significativos, la indentación, escritura correcta de llaves {} y el uso adecuado de la palabra reservada const. Se dispone de dos horas para entregar la prueba y debe realizarse en forma estrictamente individual.

Un método universal para representar hojas de cálculo son los archivos de valores separados por comas (CSV, *comma separated values*). Son archivos de texto donde cada línea representa una fila, y los valores se separan unos de otros utilizando un carácter separador, usualmente la coma (`,`), el punto y coma (`;`) o el tabulador (`\t`). El siguiente es un ejemplo de un archivo CSV.

```
date,product,description,price,stock
10/16/2014,11493,Samsung SSD 256GB,128.99,83
10/17/2014,901,WD 1TB External Hard Drive,69.99,241
10/17/2014,8291,"Apple MacBook Air 11.6, newest version",799.99,48
```

Al abrirlo en algún software de hoja de cálculo, debería verse como la siguiente imagen.

	A	B	C	D	E
1	date	product	description	price	stock
2	10/16/14	11493	Samsung SSD 256GB	128.99	83
3	10/17/14	901	WD 1TB External Hard Drive	69.99	241
4	10/17/14	8291	Apple MacBook Air 11.6, newest version	799.99	48

Sin embargo, si la configuración regional de la máquina o del software donde se abre la hoja de cálculo, no coincide con la configuración regional de quien la produjo, la hoja de cálculo se despliega incorrectamente. En el ejemplo anterior, las fechas se almacenaron en formato `MM/DD/AAAA` pero un usuario hispanohablante las espera en formato `DD/MM/AAAA`, y los precios en lugar de punto decimal deberían tener coma decimal. Dichosamente las aplicaciones de hoja de cálculo pueden hacer estas conversiones, aunque el usuario debe hacerlo manualmente.

Una pequeña empresa recibe y envía diariamente cientos de hojas de cálculo de y hacia diversos proveedores en formato CSV. Algunos proveedores utilizan la misma configuración regional de la empresa, y otros no. Por esto, cambiar la configuración regional en las máquinas locales no solventa el problema. Los empleados abren los archivos con software de hoja de cálculo y realizan los repetitivos pasos de conversión, pero se invierte mucho tiempo en ello. La empresa acude a sus servicios para que les produzca un eficiente programa en línea de comandos, escrito en C ó C++, que haga las conversiones automáticamente. La empresa necesita que el comando se pueda invocar con la siguiente convención:

```
$ csv --help
Usage: csv -is=INSEP -os=OUTSEP -id=INDEC -od=OUTDEC -if=INDATE -of=OUTDATE FILES
Change regional preferences in comma separated value files (CSV). Options:

   -is=INSEP     field separator in input file, default ','
   -os=OUTSEP    field separator for output file, default ';'
   -id=INDEC     decimal separator in input file, default '.'
   -od=OUTDEC    decimal separator for output file, default ','
   -if=INDATE    date format in input file, default MM/DD/YYYY
   -of=OUTDATE   date format for output file, default DD/MM/YYYY
   -W            overwrite input file
```

Al ejecutar el comando siguiente, en el archivo CSV de ejemplo que aparece al inicio del enunciado

```
$ csv -is=',' -os=';' -id='.' -od=',' -if='MM/DD/YYYY' -of='DD/MM/YYYY' -W eg.csv
```

produciría el siguiente resultado:

```
date;product;description;price;stock
16/10/2014;11493;Samsung SSD 256GB;128,99;83
17/10/2014;901;WD 1TB External Hard Drive;69,99;241
17/10/2014;8291;"Apple MacBook Air 11.6, newest version";799,99;48
```

Note que no se deben afectar los valores entre comillas. En general su comando debe hacer lo siguiente. Abrir cada uno de los archivos indicados por la lista `FILES`, y procesarlos línea a línea hasta encontrar el fin de archivo. En cada línea se deben identificar los campos utilizando el separador indicado por el parámetro `-is`. Si el campo es un número y tiene separador decimales (`-id`) se debe reemplazar por el indicado por `-od`. Lo mismo si el campo es una fecha. Finalmente cada campo debe escribirse en el archivo destino separado de los demás por el carácter indicado por `-os`.

Evaluación

1. [15%] Procesa y reacciona adecuadamente a los parámetros de invocación. Puede asumir para efectos del examen que el usuario siempre provee todos los parámetros en el mismo orden. No es necesario imprimir ayuda del programa.
2. [25%] El programa abre cada archivo provisto al final de la lista de parámetros, o reporta en el error estándar en caso de no poderlos abrir. Lee líneas de cada archivo en orden y las procesa. Cierra archivos tan pronto como se dejen de utilizar.
3. [10%] Hace un adecuado uso de la memoria. Trabaja a nivel de línea. Si utiliza `fgets()` maneja adecuadamente cambios de línea al final de las cadenas devueltas. Si utiliza memoria dinámica no provoca fugas de memoria.
4. [25%] Divide la línea en campos. Convierte los separadores de campos por los deseados en el archivo destino. No modifica separadores que aparecen dentro de valores entrecomillas.
5. [25%] Procesa cada campo adecuadamente. Actualiza el orden de los campos en las fechas. Actualiza el separador de decimales si el campo es un número real. Deja el valor del campo intacto en culaquier otro caso.
6. [10% **Opcional**] Haga que su programa pueda leer archivos que contengan líneas de cualquier longitud. Para esto debe llamar repetidamente a `fgets()` incrementando el tamaño del *buffer* hasta que encuentre el final de línea.
7. [10% **Opcional**] Haga que el programa pueda sobrescribir los archivos origen con la opción `-w`, la cual se debe especificar antes del nombre del primero de los archivos.

Material de referencia

- `FILE * fopen( const char * filename, const char * mode );`
  Abre el archivo cuyo nombre es indicado por `filename` utilizando uno de los modos: `"r"` (sólo lectura, el archivo debe exisitr), `"w"` (escritura, será sobrescrito con uno vacío si existe), `"a"` (añadir al final, se creará si no existe). Se debe agregar una `"b"` al modo si se quiere trabajar con arcivos binarios. Retorna un puntero hacia una estructura que representa el archivo. Declarada en `<stdio.h>`.
- `int fclose( FILE * stream );`
  Cierra el archivo. Retorna 0 en caso de éxito, `EOF` en caso de fallo. Declarada en `<stdio.h>`.
- `int remove( const char * filename );`
  Elimina el archivo cuyo nombre es dado por parámetro. Retorna 0 en caso de éxito. Declarada en `<stdio.h>`.
- `int rename( const char * oldname, const char * newname );`
  Renombra el archivo o directorio con nombre `oldname` a `newname`. Si el archivo destino existe, podría ser sobrescrito. Retorna 0 en caso de éxito. Declarada en `<stdio.h>`.
- `char * fgets( char * str, int num, FILE * stream );`
  Lee caracteres de `stream` y los aloja en `str` hasta encontrar un cambio de línea (`'\n'`) (el cual es incluido en `str`) o llenar la capacidad máxima del `str` indicada por `num` caracteres; lo que ocurra primero. Retorna `str` en caso de éxito, o `NULL` en caso de fallo. Declarada en `<stdio.h>`.
- `int fputs( const char * str, FILE * stream );`
  Escribe los caracteres de la cadena `str` en el archivo `stream` hasta encontrar el final de cadena (`'\0'`). Retorna `EOF` en caso de fallo. Declarada en `<stdio.h>`.
- `int fprintf( FILE * stream, const char * format, ... );`
  Imprime la cadena con formato `format` en el archivo `stream`. Por cada especificador `%` se espera un parámetro opcional. Algunos especificadores son: `%c` (carácter), `%s` (cadena de caracteres terminada en `'\0'`), `%i` (entero con signo), `%u` (entero sin signo), `%f` (flotante con decimales), `%%` (el carácter `%`). Retorna la cantidad de caracteres escritas en el archivo en caso de éxito, o un número negativo en caso de fallo. Declarada en `<stdio.h>`.
- `int fgetc( FILE * stream );`
  Lee y retorna el carácter actual donde se encuentra el cursor del archivo `stream`. Retorna `EOF` en caso de error. Declarada en `<stdio.h>`.
- `int fputc( int ch, FILE * stream );`
  Escribe el carácter `ch` en la posición actual de `stream`. Retorna el mismo carácter `ch` en caso de éxito o `EOF` en caso de error. Declarada en `<stdio.h>`.
- `char * strcpy( char * destination, const char * source );`
  Copia todos los caracteres de la cadena `source` en `destination` incluyendo el fin de cadena (`'\0'`). Asume que `destination` tiene suficiente memoria para copiar a `source`. Retorna `destination`. Declarada en `<string.h>`.
- `char * strcat( char * destination, const char * source );`
  Agrega `source` al final de `destination`. Asume que `destination` tiene suficiente memoria adicional para almacenar la concatenación de las dos cadenas. Retorna `destination`. Declarada en `<string.h>`.