Entrada-Salida de a Caracteres y Redirección

Por Prof. Ing. José María Sola & Prof. CC. Jorge Muchnik
20050828
20120328

Abstract

Presenta el tratamiento de la entrada y salida de a caracteres, las funciones estándar *feof*, *ferror*, *fputc* y *fgetc* y la *redirección* de los flujos estándar *stdin*, *stdout* y *stderr*. Aplica los conceptos en dos casos de estudio.

Contenidos

Objetivos

- Comprender los flujos de entrada y de salida estándar: *stdin*, *stdout* y *stderr*.
- Realizar una práctica en la consola de un sistema operativo Unix o Microsoft.
- Conocer los operadores de redirección provistos por la consola.
- Identificar las razones por las cuales puede terminar un flujo: error en la conexión o fin de datos.
- Conocer las funciones de entrada y salida estándar de a caracteres.
- Avanzar en el conocimiento de los flujos de datos en general.
- Conocer dos operadores particulares de ANIS C: condicional (`?:`) y coma (`,`)
- Avanzar en el conocimiento de ANSI C.

Introducción

ANSI C especifica que al momento de iniciar la ejecución de un programa dos *streams* (flujos) de texto[1] están disponibles: `stdin` y `stdout`, uno para entrada y otro para salida.

Los sistemas operativos derivados de Unix y luego los de Microsoft –las dos familias de sistemas operativos más difundidas hoy en día– poseen modelos muy similares en cuanto al manejo de la entrada. Al invocar un programa desde de sus consolas (i.e., líneas de comando), estos sistemas asocian automáticamente su entrada estándar y su salida estándar al teclado y a la pantalla respectivamente. Pero estas consolas permiten *redireccionar* de forma simple la entrada o la salida estándar a otros orígenes o destinos.

Los programas que basan su entrada y salida en `stdin` y `stdout` permiten flexibilidad en su uso de manera simple, y es por eso que es importante que conozcamos los mecanismos que ANSI C provee para tratar estos flujos.

Interacción con la Consola

Vamos a utilizar el programa de la sección 1.5.3 de *[K&R1988]* para demostrar el uso de la consola. Este programa simple informa la cantidad de líneas que recibe por la entrada estándar.

Primero debemos el archivo `ContarLineas.c` con el código fuente del programa, utilizando cualquier editor de texto. Luego lo compilamos con un comando *similar* al siguiente:

```
> cc ContarLineas.c
```

Para ejecutarlo, ingresamos un comando similar al siguiente:

```
> ContarLineas
```

A continuación, el sistema operativo inicia el programa y presenta un cursor, indicando que está a la espera de entrada desde el teclado. En ese momento comienza la interacción.

En este ejemplo de uso vamos a escribir tres líneas (los corchetes indican las teclas o combinación de teclas que debemos ingresar).

```
a    [a Enter]
bc   [b c Enter]
def  [d e f Enter]
[ctrl+Z Enter]
```

En los sistemas Microsoft la combinación `ctrl+Z` y luego la tecla `Enter`, **señaliza que no hay más datos para el flujo de texto de entrada**; los sistemas Unix y sus derivados suelen utilizar `ctrl+D`.

La salida por pantalla esperada es

```
3
```

Es importante notar que el teclado es un dispositivo de entrada con *buffer*, esto implica que a medida que pulsamos las teclas los caracteres se almacenan en una memoria intermedia y el programa *no* procesa la entrada hasta que recibe la indicación tratar los datos del buffer.

La forma de interactuar con la consola varía entre los diferentes sistemas operativos, pero en general el contenido del buffer se envía al programa para que lo procese ante dos situaciones:

1. Al finalizar la línea, ó

2. Al finalizar el texto.

El fin de línea se indica pulsando Enter. Esto tiene dos efectos: primero, agrega al buffer un carácter *nueva línea* y, luego, hace que se envíe su contenido al programa para que lo procese.

Como indicamos previamente, la señalización de que no hay más datos en la entrada varía entre las diferentes consolas, pero en ninguno de los casos la *señal* de fin de flujo se traduce un carácter que reciba el programa.

[1] La definición de flujo de texto y de flujo binario se encuentra en [K&R1988 Appendix B - Standard Library -- B.1 Input and Output: <stdio.h>].

Redirección de los Flujos Estándar

Existen situaciones donde quisiéramos que el origen del flujo de entrada estándar—`stdin`—no sea el teclado, si no, un archivo. Esta *redirección* la indicamos desde la línea de comando anexando un símbolo de menor (**<**) y el nombre del archivo.

En este ejemplo, calculamos la cantidad de líneas que el propio código fuente de `ContarLineas.c`, posee:

```
> ContarLineas < ContarLineas.c
18
```

Adicionalmente, si deseásemos que la salida del programa deje de ser la pantalla y pase a ser un archivo, por ejemplo `Out.txt`, simplemente debemos indicarlo con un símbolo mayor (>) y el nombre del archivo.

```
> ContarLineas < ContarLineas.c > Out.txt
```

El anterior comando redirecciona la salida estándar al nuevo archivo `Out.txt`, que lo crea con una única línea:

```
18
```

Si el archivo existiese, su contenido se destruye y se reemplaza por la salida del programa.

El contenido se puede visualizar de diferentes formas, a continuación se indican las dos más comunes:

1. Desde los *shells* visuales (e.g. *Explorer, KDE, Finder*) se hace doble clic sobre ícono del archivo `Out.txt` o se indica *Open* desde el *menú contextual* (accedido por el botón secundario). El programa asociado a ese tipo de archivos (e.g. *Notepad, KEdit*) se abrirá mostrando el contenido del archivo.
2. Desde la línea de comandos:
   Microsoft: `type Out.txt`.
   Unix: `cat Out.txt`.

Asimismo, la salida de un programa puede ser la entrada de otro, este control de flujo de datos se logra mediante la canalización utilizando los "*pipes*" (tuberías) que proveen la mayoría de las consolas. El carácter que indica la conexión de la "tubería" es |.

Como ejemplo, el siguiente comando hace que la salida de contar la cantidad de líneas que hay en el archivo fuente actúe como la entrada a contar líneas.

```
> ContarLineas < ContarLineas.c | ContarLineas
```

Sabemos que `ContarLineas < ContarLineas.c` da como resultado la línea `18\n`. Luego, ese resultado se utiliza como la entrada para la segunda invocación a `ContarLineas`, así que el resultado del comando es `1\n`.

En resumen, el símbolo menor (<) indica que el flujo `stdin` se *redirecciona*, en vez de estar asociado con el teclado, se asocia con el archivo indicado a continuación (en el ejemplo, `ContarLineas.c`). Mientras que el símbolo mayor (>) indica que el flujo `stdout` se *redirecciona*, en vez de estar asociado con la pantalla, se asocia con el archivo indicado a continuación (en el ejemplo, `ContarLineas.c`). El *pipe* (|) permite *entubar* la salida de un programa y utilizarla como entrada a otro programa.

En la documentación de cada sistema operativo hay más información detallada sobre la línea de comando y sus opciones.

La redirección de la entrada y la salida estándar, así como los *pipes* entre entrada y salida de programas se tratan en *[K&R1988 Chapter 7 - Input and Output -- 7.1 Standard Input and Output]*.

Caso de Estudio 1 – Promedio de Líneas

Problema

Calcular la longitud promedio de las líneas de un texto.

Solución

La solución se basa en **construir el programa ANSI C *Promedio*** que lea y que cuente los caracteres del *stream* (flujo, corriente) de entrada estándar (`stdin`), cuente los caracteres de nueva línea (`'\n'`), y escriba en el stream de salida estándar (`stdout`) el promedio de caracteres por línea. `Promediar.c` es un programa simple y está basado en el ejemplo de *[K&R1988 Chapter 1 - A Tutorial Introduction -- 1.5.3 Line Counting]*.

Código Fuente de Promediar.c

Esta es una implementación de `Promediar.c` que intenta resolver el problema.

```
/* ¿Informa la longitud promedio de las líneas?
 * por José María Sola
 * 20100906
 */

#include <stdio.h> /* getchar EOF feof perror printf */
#include <stdlib.h> /* EXIT_SUCCESS */

int main(void) {
    int nl;  /* la cantidad de lineas */
    int nc;  /* la cantidad de caracteres */
    int c;   /* el caracter leido */

    for(nl = 0, nc = 0; ( c = getchar()) != EOF;  ++nc)
        if(c == '\n')
            ++nl;

    if( !feof(stdin) )
        perror("No se pudo seguir leyendo de la entrada debido a un error");

    printf("Longitud promedio: %.1f\n", nc / (float)nl );

    return EXIT_SUCCESS;
}
```

Preguntas y Ejercicios

1. Explique la primer *expresión* de la sentencia `for` de esta función `main`. ¿Qué significa la coma? ¿Es un operador? ¿Qué otra expresión equivalente existe?
2. ¿Por qué son necesarios los paréntesis para el expresión `c=getchar()`? ¿Qué ocurriría si no los usamos?
3. Describa la *semántica* y la *pragmática* de la sentencia `if` que está a continuación de la sentencia `for`. *[K&R1988 7.6 Error Handling - Stderr and Exit]*.
4. Describa la función `perror`. *[K&R1988 Appendix B - Standard Library -- B.1.7 Error Functions]*.
5. Reemplace la *expresión* `!feof(stdin)` por una equivalente. Ayuda: `ferror`. Explique la semántica de `feof` y de `ferror`. ¿Las expresiones `!feof(stdin)` y `ferror(stdin)` son mutuamente excluyentes? ¿Que pragmatica tiene en este programa?
6. Explique el formato `%.1f` *[K&R1988 Chapter 7 - Input and Output -- 7.2 Formatted Output - printf]*.
7. ¿Por qué se aplica un *casteo* a la expresión `nl`?
8. Ejecute el programa con el teclado como entrada y la pantalla como salida.
9. Ejecute el programa utilizando como entrada al archivo con el programa fuente, `Promediar.c`, y como salida a `estadisticas.txt`.
10. ¿Este programa funciona correctamente para cualquier entrada? Ayuda: expresión condicional *[K&R1988 Chapter 2 - Types, Operators and Expressions -- 2.11 Conditional Expressions]*.
11. Analice si la expresión que calcula el promedio es *precisa*. ¿Cuenta la cantidad de caracteres correctamente?
12. ¿Qué cambios se deben hacer al programa para que también informe la cantidad de líneas y la cantidad de caracteres en la entrada?
13. Considerando las respuestas a 10, 11 y 12, escriba un nuevo programa: `Promediar2.c`.

Caso de Estudio 2 – Convertir a Mayúsculas

Problema

Convertir a mayúsculas un texto en castellano escrito con el alfabeto *OEM/DOS 8-bits*[2].

Solución

La solución se basa en **construir el programa ANSI C *Convetir*** que lea caracteres del *stream* (flujo, corriente) de entrada estándar (`stdin`), los convierta con la función `GetComoMayuscula`, y los escriba en el stream de salida estándar (`stdout`), de a uno por vez.

Pregunta a. ¿Es aplicable la función `toupper`? Justifique.

Comprobación de la solución – correcto funcionamiento del programa

Para probar el programa se debe construir un archivo-dato de prueba (e.g., `in.txt`) utilizando un editor de texto (e.g., *Notepad* [*Bloc de Notas*], *TextPad*, *Notepad++*, *Notepad2*, *vi*, *KEdit*) con pocas líneas pero que contengan todos los casos que permitan aseverar que el programa funciona correctamente. El alfabeto que se utiliza para escribir el archivo de texto dato debe ser compatible con el que utiliza por el programa ejecutable[3][4].

Diseño de la Solución

El programa debe utilizar las funciones estándar `getchar` y `putchar`, y la constante simbólica (*macro*) `EOF`, todos definidos en la Biblioteca Estándar de ANSI C y declarados en el encabezado `<stdio.h>`.

Para abstraer el manejo de los caracteres del alfabeto castellano se debe **especificar e implementar la función** `GetComoMayuscula` que oculte el tratamiento particular para algunos caracteres especiales del alfabeto castellano (eñe, diéresis, tildes).

La ***especificación*** de la función debe ser mediante una *función matemática partida*, indicando dominio e imagen.

La ***implementación*** de la función debe manejar los caracteres especiales del alfabeto castellano considerando los valores de la tabla OEM/DOS 8-bits; por ejemplo, para tratar la letra eñe minúscula (**ñ**) se debe usar la constante entera `164`. Mientras que para el manejo de los caracteres comunes, no deben usarse valores de la tabla, simplemente deben utilizarse las constante carácter; por ejemplo la letra A mayúscula (**A**) se debe manejar como la constante carácter `'A'` y no como la constante entera `65`.

Pregunta b. Hay dos variantes de implementación de la función: las que invocan a `toupper` y las que no. ¿Cuál implementación es más eficiente la implementación? Justificar.

Pregunta c. ¿Qué diferencias se generan si se modifica el programa `Convertir.c` para que utilice `fgetc(stdin)` en vez de `getchar()`, y `fputc(c, stdout)` en vez de `putchar(c)`? Probar y justificar.

[2] Para las consolas Microsoft, derivadas de DOS, el alfabeto generalmente utilizado es el conocido como *OEM/DOS 8-bits* o mal llamado *ASCII extendido*; este alfabeto puede ser cambiado mediante *Code Pages*. El Code Page más utilizado es el 437. Para más información, leer http://en.wikipedia.org/wiki/Code_page_437 y http://en.wikipedia.org/wiki/Character_encoding.

[3] Los sistemas operativos y sus aplicaciones pueden definir el alfabeto que utilizan, esto puede traer diferencias o comportamientos no esperados en la ejecución. Para más información, leer http://en.wikipedia.org/wiki/Code_page.

[4] En el caso de las aplicaciones visuales de Windows, el alfabeto generalmente utilizado es el 1252. Para más información leer: http://en.wikipedia.org/wiki/Windows-1252 y http://www.microsoft.com/globaldev/reference/cphome.mspx

Bibliografía

- *[K&R1988]*
  Chapter 1 - A Tutorial Introduction -- 1.5.3 Line Counting
  Chapter 2 - Types, Operators and Expressions -- 2.11 Conditional Expressions
  Chapter 7 - Input and Output -- 7.1 Standard Input and Output
  Chapter 7 - Input and Output -- 7.6 Error Handling - Stderr and Exit
  Chapter 7 - Input and Output -- 7.2 Formatted Output – printf
  Appendix B - Standard Library -- B.1 Input and Output: <stdio.h>
  Appendix B - Standard Library -- B.1.7 Error Functions

Entrega

Los puntos a resolver en este escrito deben presentarse siguiendo las siguientes pautas.

Forma

El trabajo debe ser presentado en **hojas A4**, **abrochadas** en la esquina superior izquierda, **todas numeradas** al pie con el formato "**Hoja n de m**".

Cada archivo con códigos fuente debe comenzar con un **comentario encabezado**, que actuará como carátula, con todos los datos del equipo de trabajo: **curso**; **legajo**, **apellido y nombre** de cada uno de los integrantes del equipo y **fecha de última modificación**.

Los listados de los archivos de códigos fuente, archivos dato, archivos resultado y las salidas de los programas y de los procesos de traducción deben ser impresos con Fuente de Ancho Fijo (e.g., `Courier New`, Lucida Console).

Estructuración de la Presentación

Caso de Estudio 1
1.1.1 a 1.1.13. Resolución de ejercicios.
1.2 Listado del código fuente de `Promediar2.c`.
1.3 Listado del archivo dato.
1.4 Listado del archivo resultado.
1.5 Captura impresa de la salida del proceso de traducción del compilador. Cualquier *warning* (mensaje de advertencia) o mensaje de error en tiempo de compilación debe ser incluido, copiado textualmente, junto con su número de línea y nombre de archivo.
1.6 Captura impresa de la salida de las ejecuciones del programa.
Caso de Estudio 2
2.1 Respuesta a pregunta 1.
2.2 Especificación de la función GetComoMayuscula.
2.3 Listado del código fuente de `Convertir.c`.
2.4 Listado del archivo dato.
2.5 Listado del archivo resultado.
2.6 Captura impresa de la salida del proceso de traducción del compilador. Cualquier *warning* (mensaje de advertencia) o mensaje de error en tiempo de compilación debe ser incluido, copiado textualmente, junto con su número de línea y nombre de archivo.
2.7 Captura impresa de la salida de las ejecuciones del programa.
2.8 Respuesta pregunta 2.

Tiempo

A definir en el curso.—