

DOCUMENTACIÓN
PROYECTO N°2
LENGUAJE ENSAMBLADOR
ORGANIZACIÓN DE
COMPUTADORAS
2017

Comisión 13
Enrique, Cristian Facundo - 89498
Rodríguez, Marcelo - 93554

Proyecto de Assembler

OBJETIVOS

El objetivo de este proyecto es, a partir de la consigna dada por el profesor, la implementación de un programa de nombre "enum", en el lenguaje de programación Assembler (bajo nivel) en la arquitectura de Linux i386, haciendo uso de las llamadas al sistema provistas por el sistema operativo GNU/Linux, incurriendo en la experiencia de programar con muchísimas menos herramientas abstractas que las presentes al programar con un lenguaje de alto nivel. La función del programa 'enum' es enumerar cada una de las líneas de un archivo de texto parametrizado.

ORGANIZACIÓN - ESTRUCTURA

El proyecto se divide en tres partes fundamentales que se combinan con el objetivo de cumplir con los requerimientos del enunciado:

- Determinación y captura de la cantidad de parámetros ingresados por el usuario, así como los parámetros propiamente dichos.
- Obtención de la información necesaria para manipular los datos ingresados.
- Procesamiento de los datos y salida al usuario a través de la consola (por pantalla) o mediante un archivo de salida

DESARROLLO

Como método de resolución del proyecto, se optó por comenzar verificando la cantidad de parámetros ingresados, y la correctitud de los mismos. Luego se procedió a leer la entrada carácter por carácter, hasta encontrar saltos de línea. Cada salto de línea se registró en un contador. Luego, se procedió a mostrar las líneas leídas anteriormente, con su correspondiente número de línea a la izquierda, siendo esta muestra a través de la consola o por medio de un archivo de salida.

DECISIONES DE DISEÑO

- Debido a la dificultad que impone el lenguaje, se optó por dividir el programa en sub rutinas más simples, en pos de simplificar los problemas a resolver mediante rutinas menos complejas.
- Se optó por la claridad por encima de la eficiencia, utilizando etiquetas significativas en el código, así como también gran cantidad de comentarios.
- A la hora del manejo de situaciones anormales, el valor de la situación de terminación del programa quedará alojado en la variable EBX, que luego podrá ser consultada por consola. Los valores para su identificación son los siguientes:

EBX	Detalle
0	Terminación normal.
1	Terminación anormal por error en el archivo de entrada.
2	Terminación anormal por error en el archivo de salida.
3	Terminación anormal por otras causas.

- Para el caso de que una línea este vacía, se decidió contarla igual que una línea con caracteres
- Para el caso de que la salida sea por pantalla, se optó por leer el archivo e ir mostrando por pantalla a medida que lee.
- En el caso de guardar en archivo, hace lo mismo, va leyendo, contando y escribiendo.

DESCRIPCIÓN DE LAS RUTINAS IMPLEMENTADAS

- **_start:** Inicia la ejecución del programa. Se encarga de leer los parámetros y realizar los ajustes necesarios dentro del programa para decidir qué rutina debe ejecutarse.
- **evaluar:** Controla cantidad de parámetros ingresados para definir que función utilizar, casos de mostrar ayuda o comenzar a leer el archivo de entrada.
- **errorParametros:** Realiza una terminación anormal, error causado por argumentos inválidos.
- **write:** Escribe en el archivo de salida.
- **open:** Asigna el modo lectura al archivo abierto.
- **abrirArchivoEntrada:** Realiza la apertura del archivo de lectura y si es necesario llama a crearArchivoSalida.
- **crearArchivoSalida:** Crea el archivo de salida.
- **modoAyuda:** Muestra por pantalla la ayuda y termina el programa de manera normal.
- **errorArchivoEntrada:** Terminación anormal por error en archivo de entrada.
- **errorArchivoSalida:** Terminación anormal por error en archivo de salida.
- **terminar:** Muestra "Cantidad de líneas: ".
- **success:** Realiza una terminación normal del programa.
- **analizarCaracter:** Lee de a un carácter y controla si se llegó al fin del archivo.
- **nuevaLinea:** Incrementa el contador de líneas.
- **finDeArchivo:** Cierra el archivo de entrada.
- **procesar:** Apila un carácter delimitador en la pila del sistema.
- **apilar:** Convierte la cantidad de líneas que se encuentra en número hexadecimal a decimal y apila el resultado.
- **desapilar:** Desapila el resultado de apilar y la muestra o imprime.
- **escribirSeparador:** Muestra " ": " ".

ALGORITMOS DE SUB RUTINAS

- **_start:**
Obtengo la cantidad de argumentos a través de ECX
Si la cantidad de argumentos es mayor a 3:
 Salto a subrutina de error de argumentos

Si la cantidad de argumentos es menor a 2:
 Salto a subrutina de error de argumentos
 Salto a subrutina evaluar.

- **evaluar**

Salida:

Obtengo el nombre del programa en EBX
 Obtengo la cantidad de argumentos en EBX
 Si el argumento no comienza con “-”
 Salto a abrir archivo de entrada
 Si comienza con “-” y le sigue “h” salta a mostrar ayuda.
 En otros caso salta a subrutina error de argumentos.

- **errorParametros**

Salida: EBX Estado de terminación del programa.
 Realiza una terminación anormal del programa.

- **write**

Entrada: ECX caracteres a mostrar, EDX largo de caracteres a mostrar.
Salida: EAX =bytes escritos, EBX descriptor_salida
 Muestra por pantalla o escribe en el archivo de salida.

- **open**

Entrada: EBX nombre del archivo de entrada
Salida: EAX = 5
 Abre el archivo de entrada en modo lectura.

- **abrirArchivoEntrada**

Entrada: EBX nombre del archivo de entrada. ECX cantidad de argumentos.
Salida: descriptor_entrada descriptor entrada. ESI = 1
 Salta a subrutina open para abrir el archivo de entrada y regresa, en caso de producirse un error salta a la subrutina error en archivo de entrada.
 Asigna el descriptor de entrada a su variable.
 Contador de lineas = 1
 Si la cantidad de argumentos es 3
 Salta a la subrutina crearArchivoSalida
 Si no salta a la subrutina procesar

- **crearArchivoSalida**

Entrada: EAX descriptor salida.
Salida: EBX archivo de salida.
 Obtengo en EBX el archivo de entrada
 Obtengo en EBX el archivo de salida.
 Crea el archivo de salida.
 Y controla que se haya realizado exitosamente, caso contrario salta a errorArchivoSalida.
 Asigna a descriptor_salida el descriptor de salida.
 Salta a la subrutina procesar.

- **modoAyuda**

Salida: ECX mensaje de ayuda, EDX largo del mensaje de ayuda.
 Muestra el mensaje de ayuda por pantalla.
 Realiza una terminación normal del programa.

- **errorArchivoEntrada**
Salida: EBX Estado de terminación del programa.
Realiza una terminación anormal del programa.
- **errorArchivoSalida**
Salida: EBX Estado de terminación del programa.
Realiza una terminación anormal del programa.
- **terminar**
Salida: ECX "Cantidad de líneas: ", EDX longitud del mensaje en ECX.
Muestra por pantalla o escribe en el archivo de salida el contenido de ECX.
Salta a la subrutina procesar.
- **success**
Salida: ECX salto de línea, EDX longitud del salto.
Muestra por pantalla o escribe en el archivo un salto de línea.
Realiza una terminación normal del programa.
- **analizarCaracter**
Entrada: EDI fin de archivo
Si es fin de archivo
 salta a success
Lee caracter a caracter de la línea a procesar en ese momento
Si llega al fin de archivo
 Salta a fin de archivo
Salta a write y regresa para mostrar o escribir el carácter actual.
Si llega a fin de línea
 Aumenta el contador de líneas
 Salta a analizarCaracter.
- **nuevaLinea**
Entrada: ESI contador de líneas.
Salida: ESI contador de líneas actualizado.
Incrementa el contador de líneas procesadas en 1.
Salta a la subrutina procesar.
- **finDeArchivo**
Salida: EDI = 1.
Cierra el archivo de entrada.
Salta a subrutina terminar
- **procesar**
Entrada: ESI cantidad de líneas.
Apila un carácter delimitador(\$).
- **apilar**
Entrada: EAX cantidad de líneas.

Salida: EDX cantidad de lineas en decimal.
 Agarra dígito por dígito y suma 48 a cantidad de lineas para mostrarlo por pantalla o escribirlo.

Y apila el dígito
 Sino de procesar los digitos
 Salta a apilar

- **desapilar**

Desapila el dígito y lo muestra por pantalla o escribe en el archivo saltando a write
 Si termino de desapilar los digitos
 Salta a escribir separador
 Salta a desapilar

- **escribirSeparador**

Entrada: EDI fin de archivo

Salida: ECX separador, EDX largo separador.

Si en EDI tengo 1 salta a subrutina success.

Muestra en pantalla o escribe en el archivo de salida un separador (espacio).

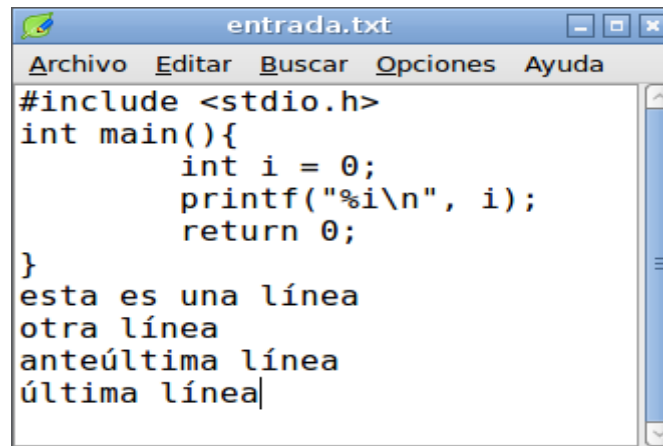
Salta a la subrutina analizarCaracter.

LIMITACIONES DEL PROGRAMA

- Por cómo está implementado, no considera el caso de archivo de entrada vacío
- Muestra el modo ayuda con cualquier cadena de caracteres seguida del -h. Por ejemplo: ./enum -hdyafs7c será lo mismo que ./enum -h
- Cuando falla por error de parámetros, o en error de apertura de archivo de entrada se podría mostrar un mensaje de advertencia. En ebx, queda el número pedido pero no muestra ningún mensaje.

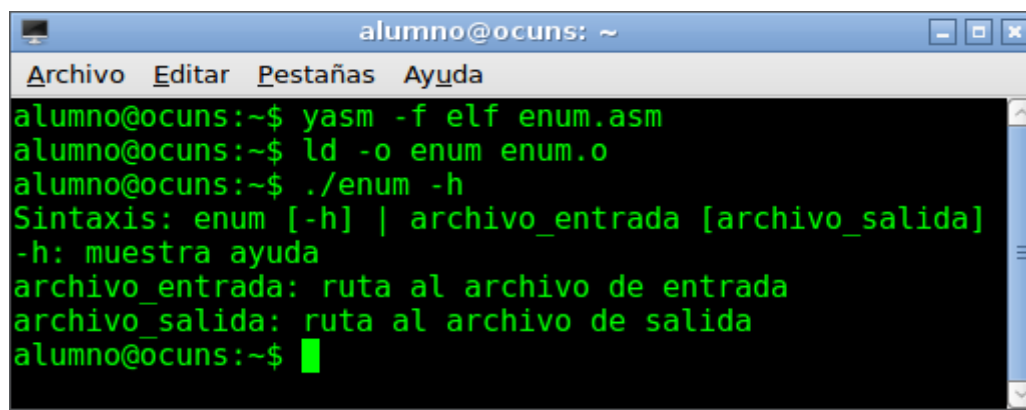
EJEMPLO DE APLICACIÓN DEL PROGRAMA

Teniendo un archivo de texto “entrada.txt” en el mismo directorio donde está el código del programa, el archivo con el siguiente contenido:



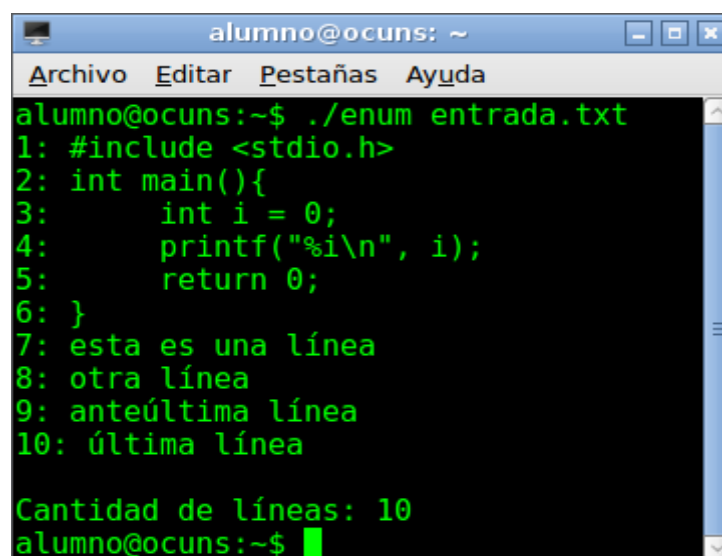
```
#include <stdio.h>
int main(){
    int i = 0;
    printf("%i\n", i);
    return 0;
}
esta es una línea
otra línea
anteúltima línea
última línea|
```

Para ejecutar el programa y que muestre la ayuda:



```
alumno@ocuns: ~
Archivo Editar Pestañas Ayuda
alumno@ocuns:~$ yasm -f elf enum.asm
alumno@ocuns:~$ ld -o enum enum.o
alumno@ocuns:~$ ./enum -h
Sintaxis: enum [-h] | archivo_entrada [archivo_salida]
-h: muestra ayuda
archivo_entrada: ruta al archivo de entrada
archivo_salida: ruta al archivo de salida
alumno@ocuns:~$
```

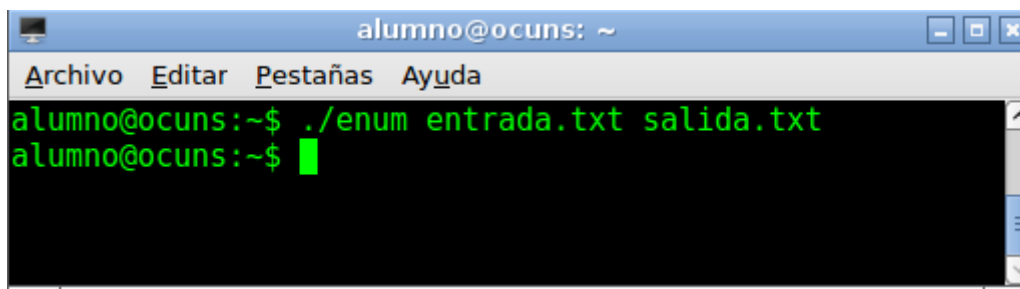
Para ejecutar el programa y que muestre el resultado por consola:



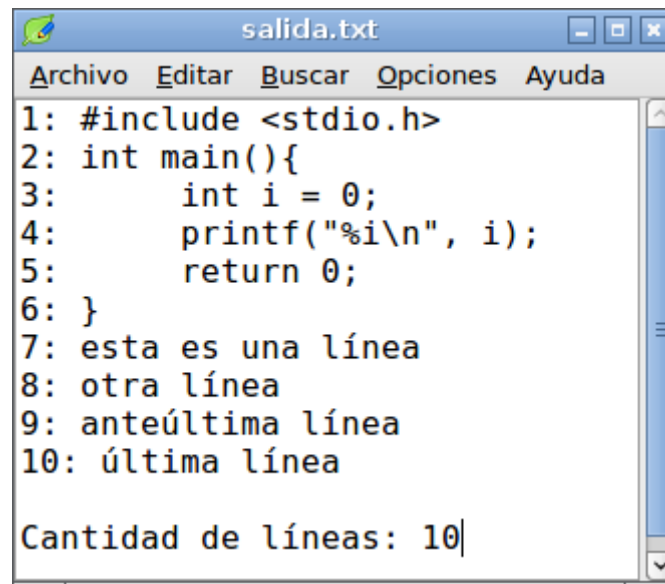
```
alumno@ocuns: ~
Archivo Editar Pestañas Ayuda
alumno@ocuns:~$ ./enum entrada.txt
1: #include <stdio.h>
2: int main(){
3:     int i = 0;
4:     printf("%i\n", i);
5:     return 0;
6: }
7: esta es una línea
8: otra línea
9: anteúltima línea
10: última línea

Cantidad de líneas: 10
alumno@ocuns:~$
```

Por último, para ejecutar el programa y que el resultado se genere en un archivo de salida:



A terminal window titled 'alumno@ocuns: ~' with a menu bar containing 'Archivo', 'Editar', 'Pestañas', and 'Ayuda'. The terminal shows the command `./enum entrada.txt salida.txt` being executed, followed by a prompt `alumno@ocuns:~$` and a green cursor.



A text editor window titled 'salida.txt' with a menu bar containing 'Archivo', 'Editar', 'Buscar', 'Opciones', and 'Ayuda'. The editor displays the following C code:

```
1: #include <stdio.h>
2: int main(){
3:     int i = 0;
4:     printf("%i\n", i);
5:     return 0;
6: }
7: esta es una línea
8: otra línea
9: anteúltima línea
10: última línea
```

At the bottom, the text 'Cantidad de líneas: 10' is displayed with a cursor at the end.