**Informe: Métricas**

**(Proyecto Organización de computadoras)**

**Introducción:**

Métricas es una aplicación desarrollada en el lenguaje ensamblador de la arquitectura i386 de 32 bits.

El programa tiene la siguiente funcionalidad:

Recibir un texto e imprimir la cantidad de palabras, letras, líneas y párrafos que este contiene.

Puede recibir cero, uno o dos parámetros:

-Si recibe cero parámetros, entonces el texto tiene que ser pasado por consola, y el resultado es recibido también en la consola

-Si recibe un parámetro, este tiene que ser un archivo de texto (conteniendo el texto que quiere ser analizado)

-Si recibe dos parámetros (ambos archivos de texto), el primero tiene que contener el texto a ser analizado y el segundo será el archivo en el cual se va a imprimir el análisis.

Para la creación de este programa se implementaron:

1\_ Modulo cons.asm (constantes del programa)

2\_ Modulo principal main.asm

3\_ Modulo exit.asm (procedimiento para salir del programa)

4\_ Modulo help.asm

5\_ Modulo file.asm

6\_ Modulo parser.asm

7\_ Modulo toString.asm

**Const:**

En este modulo se definen las siguientes constantes:

false = 0

true = 1

stdin = 0

stdout = 1

stderr = 2

no\_err = 0

input\_err = 1

output\_err = 2

unknown\_err = 3

ro\_mode = 0

wo\_mode = 1

rw\_mode = 2

creat\_mode = 0100

buff\_sz = 1000

Además, se declaran las siguientes variables sin inicializar:

on\_file (reservamos 4 bytes para el descriptor del archive de entrada)

out\_file (reservamos 4 bytes para el descriptor de archive de salida)

buffer (reservamos 1000 bytes para el buffer de lectura)

**Main**

En Main se incluyen los archivos: const.asm, exit.asm, file.asm, parser.asm toString.asm y help.asm para utilizar sus procedimientos.

**Procedimientos:**

**global \_start:** Obtiene los argumentos de entrada (si es que hay alguno) y procede dependiendo de cuantos recibió:

Si no recibe argumentos: llama al procedimiento **no\_arg**.

Si recibe dos argumentos: llama al procedimiento **dos\_arg**.

Si recibe un argumento: Primero chequea si el argumento comienza con “- “. En caso de que sea así luego chequea si el siguiente carácter es “h”:

-Si no es “h” termina la ejecución del programa llamando al procedimiento **exit** (en el archivo exit.asm) con un error desconocido (unknown\_err).

-Si es “h” entonces chequea si hay algún carácter mas en el argumento. Si no hay ninguno mas llama al procedimiento **print\_help** (en el archivo help.asm). Si hay otro carácter entonces termina la ejecución del programa llamando al procedimiento **exit**.

**no\_arg:** llama al procedimiento **parse\_stdin** (en el archivo parse.asm).

**dos\_arg:** llama al procedimiento **open\_file** (en el archivo file.asm) para abrir el archivo pasado por parámetro). Luego guarda en la dirección de memoria in\_file, el descriptor del archivo de entrada.  
Llama a **open\_file** nuevamente para abrir el archivo de salida y guarda en la dirección de memoria out\_file, el descriptor del archivo de salida.  
Por último llama al método **parse** (en el archivo parse.asm).

**Exit:**

**Procedimientos:**

**global exit**: Rutina de terminación del programa. Requiere código de terminación en el tope de la pila.

**Help:**

Declara la siguiente variable inicializada que contiene el nombre del archivo que contiene el mensaje de ayuda:

description\_file = “.desc.txt”, 0x0

**Procedimientos:**

**global print\_help:** Imprime el mensaje de ayuda especificado en el archive desc.txt de la siguiente manera:

1\_Abre el archivo desc.txt con **open\_file** (en el archivo file.asm). Luego lee el mensaje de ayuda llamando al sistema operativo. Imprime el mensaje de ayuda llamando al sistema operativo y por último llama al procedimiento **exit** (en el archivo exit.asm).

**File:**

Modulo que se encarga de la apertura y cierre de archivos.

**Procedimientos:**

**global open\_file:** Apertura de archivo. Requiere un puntero al nombre del archivo en EBX. Requiere modo de apertura en el tope de la pila. Guarda el descriptor de archivo en EAX. Se implemento de la siguiente manera:

Guarda la dirección de retorno en EDX. Prepara la llamada al sistema operativo sys\_open (el puntero al nombre del archivo ya tiene que estar almacenado en EBX), luego se establece el modo de apertura y luego la interrupción al sistema operativo. Finalmente restablece la dirección de retorno en EDX.

**global close\_file:** Cierre de archivo. Requiere descriptor del archivo en el tope de la pila.

**Parser:**

Crea las siguientes variables inicializadas:

tmp\_file = “/tmp/.tmp.txt”, 0x0 (nombre del archivo temporal para analizar stdin)

Crea las siguientes variables sin inicializar:

cnt\_letra (reserva 8 bytes para el contador de letras)

cnt\_palabra (reserva 8 bytes para el contador de palabras)

cnt\_linea (reserva 8 bytes para el contador de líneas)

cnt\_parrafo (reserva 8 bytes para el contador de párrafos)

soy\_palabra (reserva 1 byte para el flag para controlar si está escribiendo una palabra)

soy\_parrafo (reserva 1 byte para el flag para controlar si está escribiendo un párrafo)

**Procedimientos:**

**global parse:**

Inicializa los contadores y los flags en cero.  
 Lee el texto del archivo y lo almacena en al buffer (con sys\_read)  
 Si no leyó ningún carácter llama al procedimiento **toString** (en el archivo toString.asm)  
 Guarda la cantidad de caracteres leídos en la pila  
 Pone el offset del buffer en cero  
 Por cada carácter guardado:  
 si llego al fin del texto llama a **toString** (termina el bucle)  
 analiza si el carácter es una letra llama al procedimiento **es\_letra** si no es una letra llama al procedimiento **no\_es\_letra**llama al procedimiento **no\_se\_me\_ocurre\_nada**

**No\_se\_me\_ocurre\_nada**: Revisa si tiene que seguir analizando el buffer actual o leer otra porción del archivo

**Es\_letra:**

Incrementa en 1 el contador de letras  
 llama al procedimiento **comprobar\_si\_sumo\_palabra**

**Comprobar\_si\_sumo\_palabra:**

Si [soy\_palabra] es true (estoy escribiendo una palabra)  
 llama al procedimiento **comprobar\_si\_sumo\_parrafo**  
 si es false  
 incrementa el contador de palabras en 1  
 [soy\_palabra] 🡨 true  
llama al procedimiento **comprobar\_si\_sumo\_parrafo**

**Comprobar\_si\_sumo\_parrafo:**

Si [soy\_parrafo] es true (estoy escribiendo un párrafo)  
 llama al procedimiento no\_se\_me\_ocurre\_nada  
 si es false  
 incrementa el contador de parrafos en 1  
 [soy\_parrafo]🡨 true

**no\_es\_letra:**

[soy\_palabra]🡨 false  
 comparo si el carácter es ‘\n’ (salto de línea)  
 si es un salto de línea llama al procedimiento **salto\_de\_linea** si no es un salto de línea llama al **procedimiento no\_se\_me\_ocurre\_nada**

**salto\_de\_linea:**

[soy\_parrafo]🡨 false (ya no estoy escribiendo un párrafo)  
 incrementa el contador de líneas en 1  
 llama al procedimiento **no\_se\_me\_ocurre\_nada**

**global parse\_stdin:**

Abre un archivo temporal para guardar el input del usuario  
 Cierra el archivo temporal y lo abre en modo RW  
 Lee el input del usuario y cuando se llena el buffer lo escribe en el archivo  
 se obtiene el archivo cono in\_file y llama al procedimiento **parse**

**toString:**

Crea las siguientes variables inicializadas:

letras = “Letras: “  
letras\_len = longitud de “letras”

palabras = “ Palabras: “  
palabras\_len = longitud de “palabras”

lineas = “ Lineas: “  
lineas\_len = longitude de “lineas”

parrafos = “ Parrafos= “  
parrafos\_len = longitud de “parrafos”

entr = 0xa

Crea las siguientes variables sin inicializar:

buffer\_letras (reserva 6 bytes)

buffer\_palabras (reserva 6 bytes)

buffer\_lineas (reserva 6 bytes)

buffer\_parrafos (reserva 6 bytes)

**Procedimientos:**

**toString:**

Usamos AX para obtener la cantidad de letras/palabras/líneas/párrafos

EBX es un contador que lleva la cuenta de la cantidad de dígitos de AX

ESI mantiene la dirección del buffer de letras/palabras/líneas/párrafos

llamamos a **obtener\_string** que:

;1°: guarda cada uno de los dígitos de AX en la pila

;2°: hace un jmp al procedimiento  **guardar\_en\_buffer**

;3°: **guardar\_en\_buffer** hace pop a cada uno de los dígitos en la pila

;4°: mientras 'popea' estos dígitos les suma 0x30 para obtener el código ASCII

;5°: los guarda en su correspondiente buffer

;6°: una vez guardado en el buffer correspondiente retorna

;7°: volvemos a hacer lo mismo con los otros contadores (4 veces en total)

Finalmente imprime todos los buffers llamando al procedimiento **imprimir\_buffers**  y una vez finalizado llama a exit (en el archivo exit.asm)