



Escuela de Ingeniería en Electrónica

Arquitectura de Computadoras

Proyecto de programación en  
ensamblador NASM.  
Ordenamiento y visualización de  
datos

Proyecto 1 de arquitectura de computadoras

Cartago, Costa Rica  
Marzo de 2025

**Autor:**

Josué Marín Vargas

**Profesor:**

Ronny

# Resumen

El presente documento describe el desarrollo de un programa en lenguaje ensamblador NASM para sistemas x86\_64 que permite la lectura de archivos de configuración y de datos, su ordenamiento y futura visualización mediante un histograma textual. Se implementó un sistema que valida correctamente la cantidad de argumentos ingresados al ejecutar el programa, asegurando que se reciban exactamente dos rutas de archivos.

El archivo de configuración es procesado mediante un conjunto de funciones que detectan las líneas correspondientes a parámetros como nota mínima de aprobación, nota de reposición, tamaño de grupos, escala del gráfico y el tipo de ordenamiento (alfabético o numérico).

Para el ordenamiento alfabético se propuso un enfoque en el que se detectan las direcciones de memoria de inicio de cada línea, almacenándose en un arreglo auxiliar. Luego, usando una versión modificada del algoritmo Bubble Sort, se ordenan estas direcciones por el primer carácter de cada línea. Posteriormente, se imprimen secuencialmente según el orden establecido.

Para el ordenamiento numérico, se implementó una recolección de notas desde las líneas del archivo, almacenándolas en un arreglo junto con sus respectivas direcciones de inicio de línea. También se utiliza un algoritmo Bubble Sort para ordenar las notas y se reordenan las direcciones asociadas, permitiendo imprimir las líneas en orden ascendente de calificaciones.

El programa fue diseñado para ser robusto ante cambios en el orden de las líneas del archivo de configuración, ya que cada parámetro es detectado por su encabezado de texto y no por su posición.

**Palabras clave:** ensamblador, NASM, ordenamiento, arquitectura x86\_64, histograma textual.

# Índice general

<b>Resumen</b>	<b>1</b>
<b>1. Presentación del Proyecto</b>	<b>3</b>
<b>2. Diseño del Programa</b>	<b>6</b>
2.1. Método de ejecución . . . . .	6
2.2. Apertura de lectura de archivos . . . . .	7
2.3. Procesamiento de la Configuración . . . . .	8
2.4. Ordenamiento de Datos . . . . .	11
<b>3. Resultados</b>	<b>13</b>
3.1. Independencia en la posición de la líneas del archivo de configuración . . .	13
3.2. Ordenamiento numérico . . . . .	13
3.3. Ordenamiento alfabético . . . . .	14
3.4. Ejemplo de Código Ensamblador . . . . .	15

# Capítulo 1

## Presentación del Proyecto

En la primera parte del curso *Arquitectura de Computadoras* (EL4314), se trabaja en la comprensión de sistemas computacionales de propósito general, como las computadoras personales (laptops y de escritorio) que son utilizadas comúnmente en diversas aplicaciones.

Este primer proyecto tiene como objetivo profundizar en el entendimiento de la arquitectura de este tipo de sistemas, basados en un microprocesador de arquitectura x86 de 64 bits, y utilizando un sistema operativo de la familia Linux.

Para lograrlo, se plantea la creación de una aplicación simple, escrita en lenguaje ensamblador, que permita procesar archivos de datos y generar salidas ordenadas e interpretables.

Además de las habilidades técnicas en arquitectura de sistemas computacionales y software de bajo nivel, el proyecto busca introducir al estudiante en:

- La documentación de requerimientos del sistema.
- El uso de sistemas de control de versiones.
- Buenas prácticas de documentación de código y de soporte al software.

El proyecto se realizará de manera individual.

## Descripción del Programa

El programa a desarrollar deberá ser capaz de:

- Leer un archivo de texto con una lista de estudiantes y sus respectivas notas. El formato del archivo será:

```
Nombre <apellido1> <apellido2> nota:[0-100]
```

- Leer un archivo de configuración con el siguiente formato:

Nota de aprobación: [0-100]  
Nota de Reposición: [0-100]  
Tamaño de los grupos de notas: [0-100]  
Escala del gráfico: [0-100]  
Ordenamiento: alfabético/numérico

- Ordenar e imprimir la lista de estudiantes por orden alfabético o por nota.
- Generar e imprimir un histograma textual de las notas, donde:
  - Cada "X" representa un número de estudiantes definido por la escala.
  - Los estudiantes aprobados se representarán con "X" verde.
  - Los que están en el rango de reposición, en naranja.
  - Los reprobados, en rojo.

#### Ejemplo de ejecución:

Programa ruta\_configuracion ruta\_datos

- ruta\_configuracion: Ruta al archivo de configuración.
- ruta\_datos: Ruta al archivo con la lista de estudiantes y sus notas.

## Requisitos para Evaluación

Para ser evaluado, el proyecto debe compilar y ejecutarse correctamente. En caso de que no compile o dé errores fatales, no será evaluado y la nota será cero.

Adicionalmente, se agendará una reunión con el profesor para evaluar individualmente cada trabajo. Si el estudiante no es capaz de explicar su proyecto, también obtendrá una nota de cero.

## Entregables

- **Código fuente:** Debe entregarse en GitHub, adecuadamente documentado y siguiendo un formato uniforme.
- **Código objeto:** Archivos ensamblados del código fuente.
- **Archivos ejecutables:** Versión final o depurada del programa.
- **Reporte de trabajo:** Documento con:
  - Descripción del diseño de software.
  - Principales retos enfrentados y soluciones aplicadas.

- Propuestas de mejora para el programa.
- Conclusiones sobre los ensambladores.
- Referencias bibliográficas.

## Fecha de Entrega

El proyecto debe entregarse el **martes 25 de marzo de 2025**. No se dará tiempo adicional. El proyecto tiene un valor del **10%** de la nota del curso.

### Archivos involucrados:

- **Archivo de configuración:** Define parámetros como nota mínima de aprobación, nota de reposición, tamaño de los grupos de notas, escala del histograma y tipo de ordenamiento.
- **Archivo de datos:** Contiene líneas con nombres de estudiantes y sus notas en formato `Nombre Apellido nota: [0-100]`.

El programa debe ser capaz de leer ambos archivos, procesarlos, ordenarlos según la configuración, e imprimir un histograma visual codificado por colores (verde para aprobado, naranja para reposición, rojo para reprobado).

# Capítulo 2

## Diseño del Programa

### 2.1. Método de ejecución

La manera de ejecutar el programa es de la siguiente manera: `./proyecto1EXE configFile.txt dataFile.txt`

Para archivos en la misma ubicación que el ejecutable, en caso de cambiar la ruta de cada archivo, se debe cambiar cada ruta según corresponda.

Al ejecutar el programa, realiza las siguientes acciones bajo ciertas condiciones:

- Si se ingresa solo una ruta o si se ingresan tres, el programa avisa que no se ingresaron correctamente las rutas.
- Si se ingresa una ruta y el archivo no existe, el programa no funciona y genera errores.

```
1 section .bss
2     ;espacio reservado para las rutas
3     ruta1 resb 256 ;reserva un espacio de 256 caracteres para la
4         ruta1
5     ruta2 resb 256 ;reserva un espacio de 256 caracteres para la
6         ruta2
7
8 _start:
9     mov rax, [rsp]           ; Cargar argc desde la pila
10    cmp rax, 3               ; dos rutas (argc = 3)
11    jne _error_arg          ; Si no es igual a 3, lanza alerta de
12                            ; error y finaliza
13
14    ;obtengo la ruta del archivo 1
15    mov rcx, 0               ; setea un contador en cero para
16                            ; aumentar espacios en la cadena
17    mov rsi, [rsp + 16]      ; Obtener puntero argv[1]
18
19    mov rdi, ruta1           ; Obtengo el puntero de ruta1
20    call _copyStr
```

Listing 2.1: Espacios para guardar las rutas y verificación de ingreso de dos rutas

Mediante la ruta proporcionada en los argumentos de la instrucción de ejecución, permiten abrir los archivos, y copiar el contenido de cada archivo en espacios reservados para dicha información.

Como se observa en 2.1, se extraen las rutas del registro `rsp` y luego se copia el string correspondiente al argumento ingresado, en espacios reservados en el section `.bss`, además se verifica que posea tres argumentos, dos de ellos son las rutas.

## 2.2. Apertura de lectura de archivos

```
1
2 section .bss:
3     buffer resb 256 ; espacio del txt leído
4     _start:        mov rdi, ruta1 ;
5     call _openFile ; abre el archivo 1 con la ruta ingresada en rdi
6     _readConf:
7         ; ~ mov rax, 0 ; llamada al so, leer
8         mov rdi, rbx ; Descriptor, identificador del archivo a leer
9         mov rsi, buffer ; Almacenamos en buffer apartado
10        mov rdx, 256 ; Tamamo del buffer
11        syscall
12        cmp rax, 0 ; Verificar si leyo el archivo
13        jle .error_readcnf ; en caso que no lo lea, se lanza el aviso
14        ret
15
16        .error_readcnf:
17        mov rsi, error_read_msg
18        call _print
19        ret
20
21
22    _openFile:
23
24        mov rax, 2 ; para sys_open
25        mov rsi, 0 ; Modo de apertura (solo lectura)
26        mov rdx, 0 ; Permisos (no necesarios en lectura)
27        syscall
28        cmp rax, 0
29        jl .error_open
30
31
32        mov rbx, rax ; Guardar descriptor en RBX
33        mov rsi, open_check_msg ; imprime que abrio el archivo. ve
34        ; verificar que rsi no afecta a nada relacionado sys open
35        call _print
36        ret
```

Listing 2.2: Código para abrir y copiar el archivo de configuración

En 2.2, se aprecian dos funciones, una para hacer llamar al sistema para que abra la ruta, se traslada la ruta guardada a `rdi` y se realiza la llamada al sistema dentro de `_openfile`, luego, los datos de ese archivo se guardan en un espacio reservado `buffer`, ese contendrá



todo texto relacionado al texto de ese documento, que es la configuración. Podría haberse creado una función que guardara la información, pasando la ruta a un registro antes de llamar la función, pero en este caso existe una función para guardar los datos del archivo de configuración y de datos por separado, el proceso de guardado de datos del archivo con las notas se realiza de la misma manera que para el de configuración, la diferencia es que los datos se guardan en data y usa la función de `_readData`

## 2.3. Procesamiento de la Configuración

En 2.3, se llama a la función `_chargeCnf` y se comparan el texto dentro de buffer contra las cadenas de 8 byte, cuando haya una coincidencia, llamará a la función encargada de extraer el dato de configuración para esa línea correspondiente (`.Conf1`, `.Conf2`, `.Conf3`, `.Cnf4`, `.Cnf5`), cada una de estas funciones, llama a una función conocida como `buscar_corchete`, la que identifica donde está el parámetro de configuración, además de eso, se convierte el texto numérico dentro de los corchetes a entero y se guarda en un espacio reservado para dicho valor y ser empleado posteriormente

Cada línea del archivo de configuración es identificada por un string de 8 byte, ("ota de a", "ota de R", "amaño de ", "scala de", "rdenamen") de modo que cada línea tiene un string clave diferente a cualquiera de las otras líneas, esto permite que el orden de las líneas del archivo de configuración sean intercambiables verticalmente. Una vez identificado el tipo de configuración, se extraen los números entre corchetes usando funciones personalizadas como `buscar_corchete` y `_str2int`. (no se agregadas en este reporte pero que existen en el código)

```
1
2 section .bss
3     ; espacio reservado para configuracion
4     notaAp resb 1 ; reseva un espacio de 2 byte para el valor nota
5         aprobada, cnf1
6     notaRe resb 1 ; reserva un espacio para el valor de nota
7         reprobada, cnf2
8     sizeGr resb 1 ; para tamaño de grupos, cnf3
9     escala resb 1 ; tamaño de la escala, cnf4
10    ord resb 1 ; para guardar si es alfanumerico o alfabetico
11        , cnf5
12
13 _chargeCnf:
14     mov bl, [rcx]
15     mov rbx, [rcx]
16     add rcx, 1
17
18     cmp bl, 0
19     je .fin_chargeCnf
20
21     mov r8, [line1]
22     cmp r8, rbx
23     je .Cnf1
24
25     mov r8, [line2]
```

```

23     cmp r8, rbx
24     je .Cnf2
25
26     mov r8, [line3]
27     cmp r8, rbx
28     je .Cnf3
29
30     mov r8, [line4]
31     cmp r8, rbx
32     je .Cnf4
33
34     mov r8, [line5]
35     cmp r8, rbx
36     je .Cnf5
37
38     jmp _chargeCnf
39
40 .fin_chargeCnf:
41     ret
42
43 .Cnf1:
44     mov r9, 0
45     call buscar_corchete
46     call _str2int
47     mov [notaAp], al
48     mov r10, defline
49     mov [num_temp], r10
50     jmp _chargeCnf
51
52 .Cnf2:
53     mov r9, 0
54     call buscar_corchete
55     call _str2int
56     mov [notaRe], al
57     mov r10, defline
58     mov [num_temp], r10
59     jmp _chargeCnf
60
61 .Cnf3:
62     mov r9, 0
63     call buscar_corchete
64     call _str2int
65     mov [sizeGr], al
66     mov r10, defline
67     mov [num_temp], r10
68     jmp _chargeCnf
69
70 .Cnf4:
71     mov r9, 0
72     call buscar_corchete
73     call _str2int
74     mov [escala], al
75     mov r10, defline
76     mov [num_temp], r10

```

```

77         jmp _chargeCnf
78
79 .Cnf5:
80     mov r9, 0
81     call buscar_corchete
82     mov bl, [num_temp]
83     cmp bl, 97
84     je salto_conf5
85
86     mov al, 0
87     mov [ord], al
88     mov r10, defline
89     mov [num_temp], r10
90     jmp _chargeCnf
91
92     salto_conf5:
93     mov al, 1
94     mov [ord], al
95     mov r10, defline
96     mov [num_temp], r10
97     jmp _chargeCnf
98
99 buscar_corchete:
100     mov bl, [rcx]
101     add rcx, 1
102     cmp bl, 91
103     je .corchete_enc
104     jmp buscar_corchete
105
106 .fin_buscar_corchete:
107     add rcx, 1
108     ret

```

Listing 2.3: Extracción de datos de configuración

En 2.2, se aprecia la función que imprime cada línea ordenada, iniciando desde la primera letra hasta que detecta un salto de línea.

```

1  imprimir_letra:
2      mov rax, 1                ; syscall: write
3      mov rdi, 1                ; File descriptor stdout
4      mov rsi, r12
5      mov rdx, 1                ; Longitud de 1 byte
6      syscall
7
8      mov al, [r12]
9      cmp al, 10                ; Verificar si es el terminador
10     ; nulo ('\n')
11     je fin_imprimir_letra      ; Si es '\n', terminamos
12
13     inc r12
14     jmp imprimir_letra        ; Repetir el proceso
15
16 fin_imprimir_letra:
17     ret

```

## 2.4. Ordenamiento de Datos

Para el ordenamiento alfabético, se empleó un vector de direcciones llamado `line_addrs`, donde se almacenó la dirección de la primer letra de cada línea del archivo de datos, para saber cuando la línea terminaba, se empleó comparación por salto de línea. Teniendo las direcciones en un vector, solo restaba ordenar por orden alfabético, dado que ya los caracteres `ascii` tiene un orden por valor entero, se logró comparar los valores de cada letra. Cuando ya se estuvieran ordenadas las direcciones, se manda a imprimir a partir de esa letra y las contiguas a esa en las direcciones superiores hasta topar con un salto de línea, y así se procedió con cada dirección, sin necesidad de copiar todas las líneas de nuevo.

Para el ordenamiento numérico, se emplearon dos vectores, uno llamado `notas`, y de nuevo, `line_addrs`, en `notas` se almacenaban los valores enteros de la notas y en `line_addrs` la dirección de la primer letra de la línea donde esa nota se encuentra ubicada. Para el ordenamiento, se ordenó con base al los valores de `notas`, pero como hay una relación de posición directa de las direcciones de las letras con las notas, entonces, cada movimiento en `notas`, se hace también en `line_addrs`, cuando ya estuviera ordenado `notas`, las direcciones de la primer letra de cada línea también lo estaría, así que de nuevo se vuelve a imprimir de la dirección de cada letra y las letras contiguas en posiciones superiores.

En 2.5, se aprecian dos funciones, `sort_numeric` y `sort_alpha`, una es para el ordenamiento para el ordenamiento numérico y la otra para el ordenamiento alfabético.

```

1  sort_numeric:
2      mov rcx, [line_count]          ; Numero total de elementos
3      dec rcx                        ; Necesitamos n-1 pasadas
4      cmp rcx, 0
5      jle sort_numeric_done          ; Si hay 0 o 1 elemento, no se ordena
6
7  sort_numeric_outer:
8      mov rdi, 0
9      mov rsi, rcx                    ; Comparaciones por pasada
10
11  sort_numeric_inner:
12      mov rax, [notas + rdi * 8]
13      mov rbx, [notas + rdi * 8 + 8]
14      cmp rax, rbx
15      jbe sort_numeric_no_swap
16
17      ; Intercambiar notas[rdi] y notas[rdi + 1]
18      mov rax, [notas + rdi * 8]
19      mov rbx, [notas + rdi * 8 + 8]
20      mov [notas + rdi * 8], rbx
21      mov [notas + rdi * 8 + 8], rax
22
23

```

```

24     mov rax, [line_addrs + rdi * 8]
25     mov rbx, [line_addrs + rdi * 8 + 8]
26     mov [line_addrs + rdi * 8], rbx
27     mov [line_addrs + rdi * 8 + 8], rax
28
29 sort_numeric_no_swap:
30     inc rdi
31     dec rsi
32     jnz sort_numeric_inner
33
34     loop sort_numeric_outer
35
36 sort_numeric_done:
37     ret
38
39
40 sort_alpha:
41     mov rcx, [line_count]
42     dec rcx                                ; Se requieren n-1 pasadas
43     cmp rcx, 0
44     jle sort_alpha_done
45
46 sort_alpha_outer:
47     mov rdi, 0                            ; Indice del vector
48     mov rsi, rcx
49
50 sort_alpha_inner:
51     mov rax, [line_addrs + rdi * 8]
52     mov rbx, [line_addrs + rdi * 8 + 8]
53     mov dl, [rax]
54     mov dh, [rbx]
55
56     cmp dl, dh
57     jbe sort_alpha_no_swap                ; Si ya estan ordenadas,
                                           saltar swap
58
59     ; Intercambiar direcciones
60     mov [line_addrs + rdi * 8], rbx
61     mov [line_addrs + rdi * 8 + 8], rax
62
63 sort_alpha_no_swap:
64     inc rdi
65     dec rsi
66     jnz sort_alpha_inner
67
68     loop sort_alpha_outer
69
70 sort_alpha_done:
71     ret

```

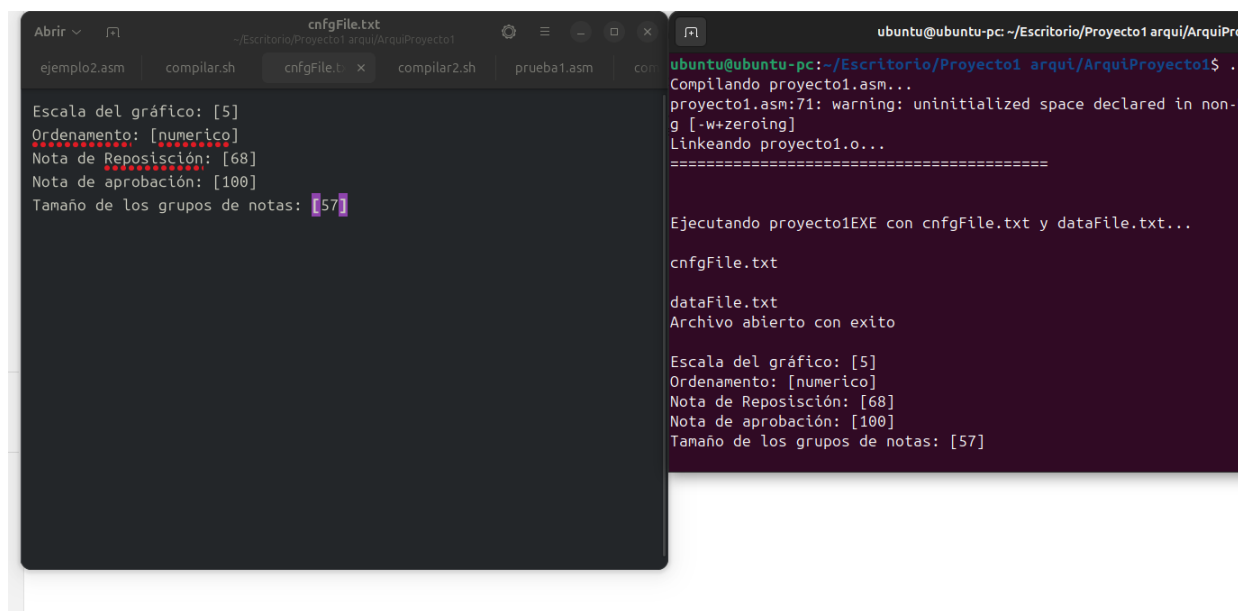
Listing 2.5: Algoritmo de ordenamiento

# Capítulo 3

## Resultados

### 3.1. Independencia en la posición de la líneas del archivo de configuración

En la Fig. 3.1, se muestra en consola como el programa lee el primer acomodo



The image shows a code editor window on the left and a terminal window on the right. The code editor displays the contents of a file named `cnfgFile.txt`, which contains the following configuration data:

```
Escala del gráfico: [5]
Ordenamiento: [numerico]
Nota de Reposición: [68]
Nota de aprobación: [100]
Tamaño de los grupos de notas: [57]
```

The terminal window shows the output of the program. It starts with the command `./proyecto1` being executed. The output indicates that the program is compiling `proyecto1.asm`, linking it to `proyecto1.o`, and then executing the resulting `proyecto1EXE` with `cnfgFile.txt` and `dataFile.txt` as input. The output of the execution matches the configuration data in the code editor:

```
Ejecutando proyecto1EXE con cnfgFile.txt y dataFile.txt...
cnfgFile.txt
dataFile.txt
Archivo abierto con exito
Escala del gráfico: [5]
Ordenamiento: [numerico]
Nota de Reposición: [68]
Nota de aprobación: [100]
Tamaño de los grupos de notas: [57]
```

Figura 3.1: Primer orden en las líneas del archivo de configuración.

En la Fig. 3.2, se muestra en consola como el programa lee el primer acomodo

### 3.2. Ordenamiento numérico

Los resultados del ordenamiento numérico se aprecian en la Fig. 3.3.

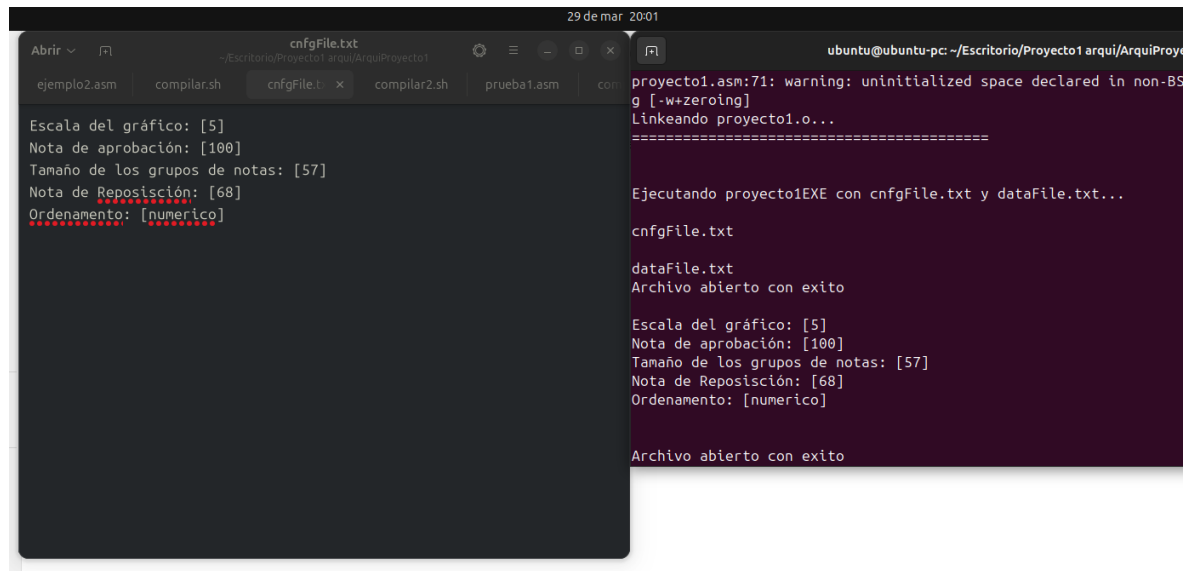


Figura 3.2: Segundo orden en las líneas del archivo de configuración.

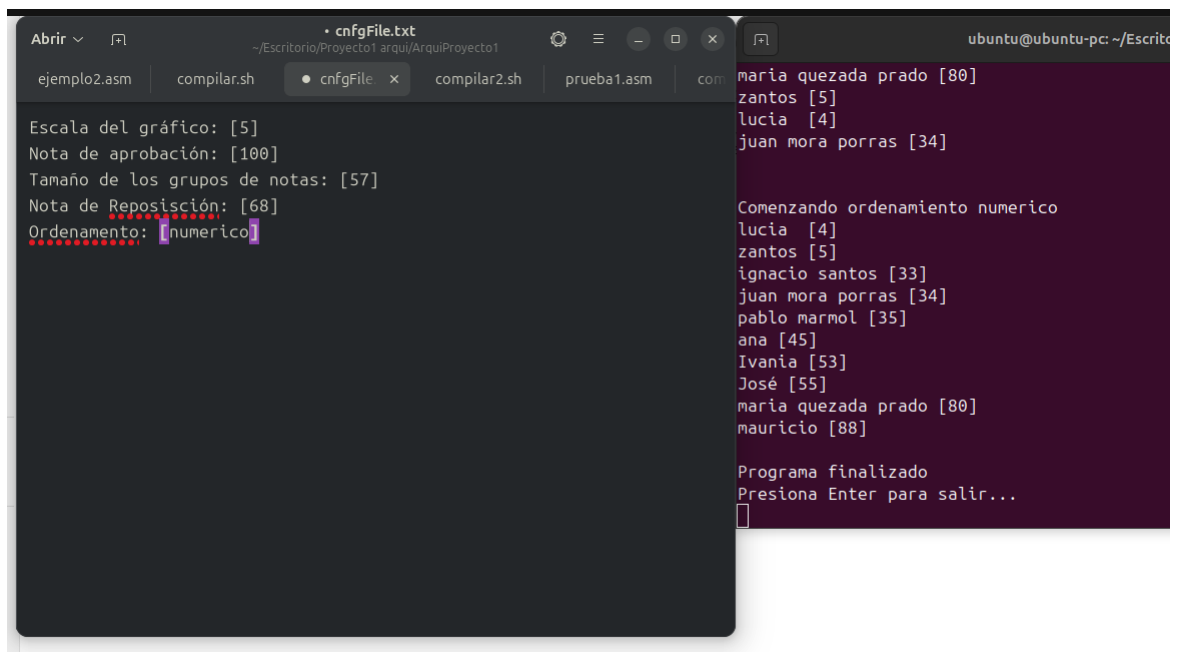


Figura 3.3: Resultado ordenamiento numérico

### 3.3. Ordenamiento alfabetizo

Los resultados del ordenamiento alfabético se aprecian en la Fig. 3.4.

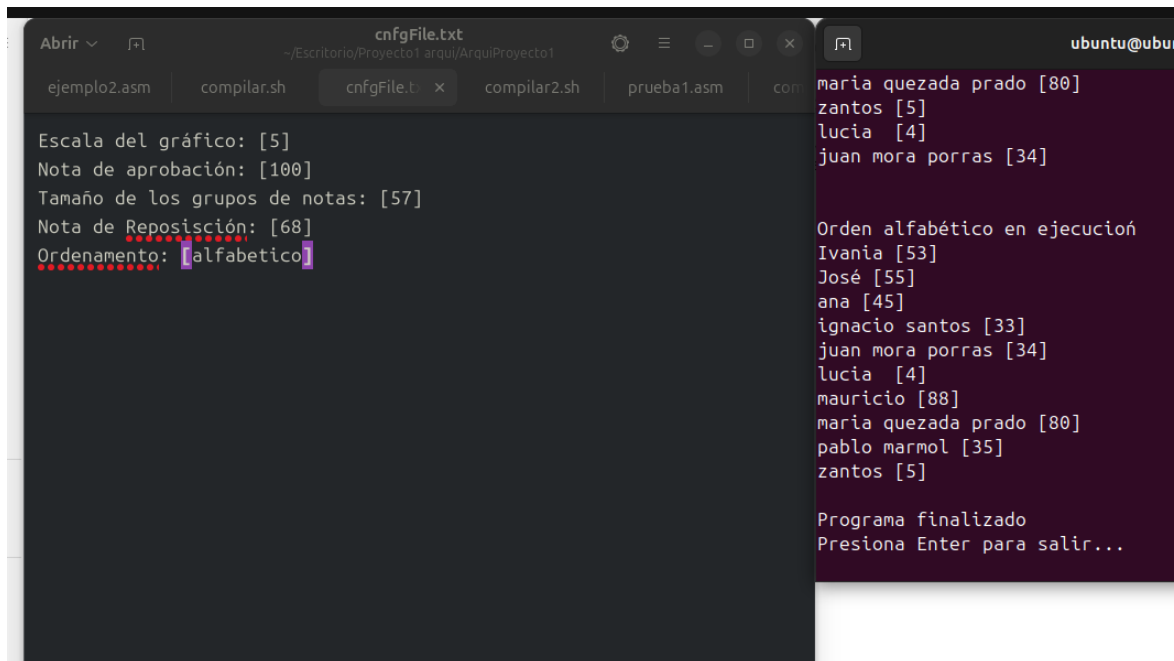


Figura 3.4: Resultado ordenamiento alfabetizo

### 3.4. Ejemplo de Código Ensamblador

A continuación se muestra un fragmento del código ensamblador implementado en el proyecto (ver Listado 3.1):

```

1 section .data
2     msg db 'Hola mundo', 0Ah
3
4 section .text
5     global _start
6
7 _start:
8     mov eax, 4
9     mov ebx, 1
10    mov ecx, msg
11    mov edx, 10
12    int 0x80
13
14    mov eax, 1
15    xor ebx, ebx
16    int 0x80

```

Listing 3.1: Fragmento de código NASM para imprimir mensaje