
Escuela de Electrónica

**Arquitectura de computadoras**

Documentación de proyectos

Docente:

Ronny Giovanni Garcia Ramirez

Estudiante:

Jorge Pérez Jiménez 2021061955

Semestre II 2025

Fecha de entrega

Lunes 22 de septiembre

# I. Introducción

El siguiente proyecto presenta el desarrollo de un programa en ensamblador x86 utilizando la herramienta de NASM en el entorno de Linux. Este programa analiza dos archivos de texto. El primero contiene información de inventario que debe ser leída, organizada alfabéticamente y representada mediante un diagrama de barras. El segundo archivo suministra los parámetros de configuración necesarios para la personalización de la visualización gráfica.

La relevancia de este proyecto radica en la demostración práctica de cómo el lenguaje ensamblador, a pesar de su bajo nivel de abstracción, puede ser utilizado para implementar funcionalidades complejas como el procesamiento de archivos, manipulación de cadenas, algoritmos de ordenamiento y generación de interfaces visuales mediante secuencias de control ANSI.

# II. Funcionamiento del código

Este programa desarrollado en ensamblador x86-64 para sistemas Linux tiene como objetivo central administrar un inventario de productos alimenticios, específicamente frutas, organizando sus denominaciones en orden alfabético y creando una representación visual de las existencias mediante gráficos de barras en la terminal de comandos.

La estructura del programa se organiza en tres segmentos fundamentales:

La primera sección corresponde al segmento de datos inicializados (.data), donde se establecen las cadenas constantes requeridas durante la ejecución, incluyendo los nombres de los archivos de entrada y caracteres especiales como el indicador de nueva línea. Esta sección define los elementos de datos que permanecen invariables durante la ejecución del programa.

```
; Sección de datos definidos
section .data
    archivo_config      db "config.ini",0       ; Archivo de parámetros
    archivo_inventario  db "inventario.txt",0   ; Archivo de datos
    salto_linea         db 10                   ; Carácter nueva línea
```

Figura 1: Section .data

Posteriormente, la sección .bss (Block Started by Symbol) se encarga de reservar espacio en memoria para variables cuyo valor inicial se desconoce o no requiere inicialización específica. En esta zona se asignan buffers de memoria de tamaño predefinido para almacenar información dinámica como los contenidos de archivos, parámetros de configuración y resultados intermedios del procesamiento.

```
; Sección de reserva de memoria
section .bss
    memoria_temp    resb 257    ; Almacenamiento temporal para lecturas
    bytes_leidos    resq 1      ; Contador de bytes leídos
    simbolo_barra   resb 8      ; Símbolo utilizado para las barras
    color_texto     resb 4      ; Código color del texto (ANSI)
    color_base      resb 4      ; Código color de fondo (ANSI)

    ; Espacios para nombres de productos
    producto1 resb 32
    producto2 resb 32
    producto3 resb 32
    producto4 resb 32

    ; Almacenamiento para valores numéricos
    valor1 resq 1
    valor2 resq 1
    valor3 resq 1
    valor4 resq 1

    cadena_numero resb 32       ; Conversión temporal números→texto

    ; Buffers para las barras generadas
    grafica1 resb 256
    grafica2 resb 256
    grafica3 resb 256
    grafica4 resb 256

    ; Buffers para combinaciones (producto + barra)
    resultado1 resb 512
    resultado2 resb 512
    resultado3 resb 512
    resultado4 resb 512

    ; Array de referencias a resultados
    referencias resq 4

    ; Buffer unificado para salida final
    salida_ordenada resb 2048
```

Figura 2: Section .bss

La tercera y última sección comprende el segmento de código ejecutable (.text), donde se implementa toda la lógica operativa del programa. Aquí se definen las rutinas auxiliares que permiten realizar operaciones esenciales como manipulación de cadenas de caracteres, conversiones entre formatos numéricos, gestión de archivos del sistema y control del flujo de ejecución.

```
; Código principal
section .text
global _start

_start:
    ; Configurar array de referencias
    lea rax, [rel resultado1]
    mov [referencias+0*8], rax
    lea rax, [rel resultado2]
    mov [referencias+1*8], rax
    lea rax, [rel resultado3]
    mov [referencias+2*8], rax
    lea rax, [rel resultado4]
    mov [referencias+3*8], rax
```

Figura 3: Section .text

La secuencia de ejecución inicia con la apertura y lectura del archivo de configuración (config.ini). Este archivo contiene especificaciones sobre el carácter que constituirá las barras gráficas y los esquemas de color (tanto de fondo como de texto) que se aplicarán mediante secuencias de escape ANSI para la visualización en terminal.

```
simbolo_barra   resb 8      ; Símbolo utilizado para las barras
color_texto     resb 4      ; Código color del texto (ANSI)
color_base      resb 4      ; Código color de fondo (ANSI)
```

Figura 4: Proceso de lectura del archivo de configuración

Una vez extraídos y procesados estos parámetros de configuración, el programa procede a abrir y leer el archivo de inventario (inventario.txt). Cada registro de este archivo contiene un producto seguido del carácter dos puntos y un valor numérico entero que representa la cantidad disponible. El programa está diseñado para procesar un máximo de cuatro líneas, separando en cada una de ellas la denominación del producto y su correspondiente cantidad numérica.

```
memoria_temp    resb 257    ; Almacenamiento temporal para lecturas
bytes_leidos    resq 1      ; Contador de bytes leídos
```

Figura 5: Estructura del archivo de inventario

```
; Espacios para nombres de productos
producto1 resb 32
producto2 resb 32
producto3 resb 32
producto4 resb 32
```

Figura 6: Procesamiento de líneas del inventario

Las cantidades obtenidas son fundamentales para la etapa de construcción de barras. Para cada valor, el programa genera una cadena en la que el carácter de barra se repite tantas veces como indique el número leído. Estas cadenas no son simples repeticiones de caracteres, sino que además están enmarcadas por secuencias de escape ANSI, que permiten aplicar colores a la salida en la terminal de Linux. De esta manera, cada barra se distingue tanto por su longitud como por el color que la acompaña.

```
; Buffers para las barras generadas
grafica1 resb 256
grafica2 resb 256
grafica3 resb 256
grafica4 resb 256
```

Figura 7: Generación de barras con secuencias ANSI

Una vez construidas las barras, el programa concatena el nombre del producto con su respectiva barra, formando así una línea completa de salida. Cada producto con su barra

se almacena en buffers separados. Para garantizar una presentación ordenada, se implementa a continuación una rutina de ordenamiento que reorganiza las líneas de manera alfabética.

```
; Buffers para combinaciones (producto + barra)
resultado1 resb 512
resultado2 resb 512
resultado3 resb 512
resultado4 resb 512
```

Figura 8: Concatenación y almacenamiento de resultados

El algoritmo seleccionado para esta tarea de ordenamiento es Bubble Sort, que resulta adecuado y eficiente para conjuntos reducidos de elementos como el manejado en este proyecto. Con las líneas ya organizadas alfabéticamente, se procede a su concatenación en un buffer unificado que finalmente se envía a la salida estándar, presentando así el inventario ordenado en la interfaz de terminal.

```
; Array de referencias a resultados
referencias resq 4

; Buffer unificado para salida final
salida_ordenada resb 2048
```

Figura 9: Algoritmo de ordenamiento e impresión final

## III. Algoritmos utilizados

## III.A Ordenamiento (Bubble Sort)

Se implementa este método de ordenamiento clásico para comparar pares de cadenas de caracteres e intercambiarlas cuando se detecta un desorden alfabético. Su principal ventaja en este contexto radica en la simplicidad de implementación y la adecuación para conjuntos de datos pequeños, como el caso específico de cuatro elementos que maneja este programa.

Funcionamiento:

1. Se realizan múltiples iteraciones sobre el arreglo de punteros outs[ ].

2. En cada ciclo de comparación, se evalúan los elementos consecutivos outs[i] y outs[i+1] mediante la función strcmp.

3. Cuando se detecta que outs[i] es alfabéticamente mayor que outs[i+1], se procede al intercambio de sus punteros.

4. El proceso se repite iterativamente hasta que no se requieran más intercambios, indicando que el arreglo está completamente ordenado.

Este enfoque garantiza que los elementos con menor valor alfabético asciendan gradualmente hacia el inicio del arreglo, similar a burbujas que emergen en un líquido, de ahí la denominación del algoritmo.Inventario.txt

Ejemplo de archivo de entrada (inventario.txt):

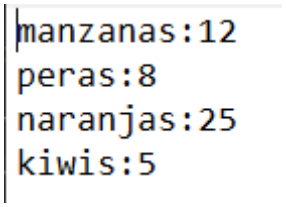


Figura 10: Contenido ejemplo del archivo inventario.txt

Resultado en consola Linux Ubuntu:

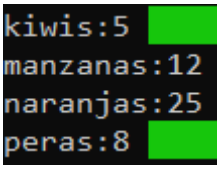


Figura 11: Salida del programa en terminal Linux

## III.II Visualización

El procedimiento para la construcción de las barras visuales implementa las siguientes etapas:

1. Inicia con la secuencia de escape ANSI "ESC[<código_fondo>;<código_texto>m" para establecer los colores.

2. Repite "n" veces el carácter configurado para la barra, donde "n" representa la cantidad del producto.

3. Finaliza con la secuencia "ESC[0m" para restablecer los atributos visuales por defecto.

Por ejemplo, utilizando los parámetros: caracter_barra="#", color_fondo=40 (negro) y color_barra=92 (verde brillante), se genera la secuencia:

\"\\033[40;92m##########\\033[0m\"

Esta combinación produce en terminal una barra de color verde brillante sobre fondo negro, con longitud proporcional al valor numérico representado.

Configuración ejemplo (config.ini):

Figura 12: Parámetros de configuración visual

## IV. Consideraciones técnicas adicionales

El programa incorpora manejo robusto de errores para situaciones excepcionales como fallos en la apertura de archivos o errores de lectura. Para cada operación de entrada/salida, se verifica el código de retorno del sistema y se toman acciones apropiadas, garantizando una terminación controlada en caso de incidencias.

La gestión de memoria se optimiza mediante el uso de buffers de tamaño fijo, calculados para acomodar los casos de uso previstos sin consumir recursos excesivos. Las operaciones sobre cadenas implementan verificación de terminadores nulos para prevenir desbordamientos de buffer.

El uso de secuencias de escape ANSI permite compatibilidad con la mayoría de los terminales modernos de Linux, proporcionando una experiencia visual consistente across diferentes distribuciones y configuraciones de terminal.

## V. Link de GIT HUB

https://github.com/Jorper05/Proyecto-1-Arqui.git