Departamento de Ingeniería Electrónica

Guía de Trabajos Prácticos

INFORMATICA I

2025

Introducción y régimen de aprobación

La presente guía de Trabajos Prácticos tiene por objeto llevar a la práctica los contenidos vistos en las clases teóricas. De este modo se espera una realimentación entre la aplicación y la lectura de los diferentes conceptos teóricos que permita desarrollar en el alumno un enfoque metodológico para resolver problemas sencillos de Ingeniería utilizando las diferentes herramientas de software indicadas por la cátedra, y resolviendo los algoritmos planteados en Lenguaje C.

El grado de complejidad irá creciendo a través de los diferentes ejercicios planteados para cada Unidad Temática.

La entrega de cada ejercicio se efectuará sin excepciones en las fechas estipuladas por el docente a cargo del curso.

Las fechas de entrega de los prácticos obligatorios estarán diseñados para que todos los Trabajos Prácticos correspondientes a los contenidos que se incluyen en cada parcial sean revisados por los docentes auxiliares antes del examen. De este modo los alumnos tendrán una devolución con las correcciones de los errores detectados, como forma de realimentación necesaria para el examen parcial.

La no entrega de un ejercicio en la fecha establecida equivale a considerar al alumno Ausente en ese práctico. En consecuencia, se considerará No Aprobado dicho práctico. De acuerdo con el reglamento vigente, la aprobación de los Trabajos Prácticos requiere el 80% de los mismos Aprobados. En el caso de esta guía de Trabajos Prácticos se requiere la aprobación del 80% de los estipulados de Entrega Obligatoria. Tenga en cuenta que la aprobación final del trabajo práctico requerirá una **defensa oral**, en la cual el alumno deberá demostrar el conocimiento adquirido durante su realización.

Formato de presentación

- Los archivos fuentes deben tener en todos los casos los comentarios necesarios para clarificar su lectura.
- Cada subrutina/función, debe llevar un encabezado con la descripción de la operación que realiza, los parámetros que espera como entrada, y los resultados que debe presentar, indicando formato y método de entrega. Para esto, los docentes proveerán al alumno de un template el cual debe ser utilizado en todos los ejercicios realizados.
- En el archivo fuentes principal (nombre_programa.c) deberá haber un comentario que explique claramente las instrucciones detalladas (comandos) para su compilación, linkeo y ejecución.

  IMPORTANTE: El alumno deberá entregar un único archivo comprimido con el nombre y extensión: tp_numero_apellido-legajo.tar.gz (ejemplo: tp_1_martinez-1451932.tar.gz) el cual contendrá **sólo** código fuente junto a una imagen de la terminal mostrando el **correcto funcionamiento** del programa realizado.

Trabajo Práctico N°1

Contenido a evaluar: *Introducción a la lógica estructurada.*

UtMarket

La empresa **Market** lo ha contratado para desarrollar un sistema que permita gestionar el inventario y las ventas de frutas y verduras en sus sucursales. El objetivo del sistema es registrar las ventas diarias, controlar el stock disponible y calcular las ventas acumuladas por empleado, así como los totales al final del día.

El negocio maneja cinco productos: manzanas, naranjas, bananas, lechugas y tomates. A su vez, cada sucursal cuenta con cuatro empleados responsables de registrar las ventas, identificados como Empleado 1, Empleado 2, Empleado 3 y Empleado 4.

El sistema debe permitir realizar las siguientes operaciones: Primero, registrar una venta, lo que implica ingresar el producto vendido, la cantidad (en kilogramos), el precio por kilogramo y el empleado que realizó la venta. Tenga en cuenta que, cada vez que se registra una venta, el sistema debe verificar si hay stock suficiente del producto. Si no lo hay, se debe mostrar un mensaje de error indicando que la operación no se puede completar. Segundo, consultar el stock actual de cada producto, el cual debe estar predefinido al inicio del programa y actualizarse con cada venta registrada. Además, se debe permitir mostrar las ventas acumuladas de cada empleado, tanto en cantidad como en dinero recaudado.

El ingreso de datos se dará hasta que el usuario decide finalizar el programa. En ningún caso se deben permitir cantidades negativas o productos que no estén en la lista. Al finalizar, el sistema debe mostrar un resumen con el monto total de ventas por producto, la recaudación total del día y las ventas acumuladas por empleado.

Se solicita:

1. **Identificación del problema:** Describa cuáles son los datos del problema, las variables necesarias, las entradas, los procesos requeridos y las salidas esperadas.

2. **Diseño del diagrama de flujo de alto nivel:** Diseñe un diagrama de flujo que describa las operaciones principales del sistema, desde el inicio hasta el final, considerando el flujo general de las funciones.

3. **Diseño de diagramas de flujo de detalle:** Elabore diagramas de flujo detallados para cada una de las funciones del menú:
    a. Registrar una venta
    b. Consultar el stock actual
    c. Mostrar ventas acumuladas
    d. Finalizar el programa y calcular los resultados.

Trabajo Práctico N°2

Contenido a evaluar: *Introducción a la programación estructurada, Sistema operativo, Introducción al lenguaje C, Variables, Operadores a nivel de variable, Control de flujo en lenguaje C. Buenas prácticas de programación.*

Simulación interactiva de lanzamientos de cohetes espaciales

La Agencia Espacial Internacional (AEI) está simulando lanzamientos de cohetes para verificar si alcanzan una "órbita segura".

En este trabajo práctico se realizará un programa el cuál sirva de simulador a la agencia a partir de las especificaciones de funcionamiento explicadas a continuación.

Para cada cohete, se ingresarán datos clave, y el programa permitirá realizar hasta tres intentos de lanzamiento (MAX_INTENTOS) antes de declarar un lanzamiento fallido definitivo. Para esto, el usuario deberá ingresar la cantidad de cohetes a lanzar y, para cada uno, la masa en kilogramos (1 tonelada = 1000 kg). Además, se solicitarán los valores mínimos necesarios para que un cohete logre entrar a una órbita segura, que son la velocidad mínima en metros por segundo (m/s) y la altitud mínima en metros.

Durante cada intento de lanzamiento, el programa generará de manera aleatoria la fuerza de empuje, que será un número real entre 500 y 1500 Newtons (N) y el ángulo de lanzamiento, que será un valor real entre 30 y 90 grados. Con estos valores, se calcularán la velocidad y la altitud iniciales del cohete, utilizando las siguientes fórmulas:

$$\text{Velocidad incial} = V_0 = \frac{F[N] * \cos(\varphi)}{M[T]}$$

$$\text{Altitud inicial} = h = \frac{F[N] * \sin(\varphi)}{M[T]}$$

A partir de estos valores, si el cohete alcanza la órbita segura (es decir, si cumple o supera los valores mínimos de velocidad y altitud, el intento será considerado exitoso y no se realizarán más intentos. De lo contrario, si luego de tres intentos no se logrará alcanzar la órbita segura, se declarará como un lanzamiento fallido.

Al final de la simulación, se deberá mostrar el número de intentos, el ángulo de lanzamiento, la fuerza de empuje, la velocidad y la altitud iniciales para cada cohete. Además, se deberá calcular y mostrar el porcentaje de cohetes que tuvieron éxito en alcanzar la órbita segura.

A lo largo de la simulación, además se deberá calcular la energía cinética y la energía potencial del cohete en función de su velocidad inicial ($V_0[m/s^2]$) y altitud inicial ($h[m]$) previamente calculadas. Las fórmulas para estos cálculos son las siguientes:

$$\text{Energía cinética} = \mathrm{E_{cinética}} = \frac{1}{2} * \mathrm{M[Kg]} * {\mathrm{V_0}}^2$$

$$\text{Energía potencial} = \mathrm{E_{potencial}} = \mathrm{M[Kg]} * \mathrm{G} * \mathrm{h}$$

Donde G ($= 9.81\ [\mathrm{m}/s^2]$) es la gravedad.

Con estos valores, se deberá clasificar el rendimiento del cohete en cada intento, el cual también deberá ser informado, según la proporción de energía cinética respecto a la energía total (energía cinética + energía potencial).

Para esto la clasificación se realizará de la siguiente manera (considerando intervalos semi abiertos del modo (valor_extremo_1, valor_extremo_2]:

- Excelente si la energía cinética representa más del 80%.
- Bueno si está entre el 50% y 80%.
- Regular si está entre el 20% y 50%.
- Deficiente si es menos del 20%.

Si un intento no supera la altitud mínima en metros para entrar en órbita, el cohete recibirá una penalización que disminuirá su rendimiento en un 5% en el siguiente intento.

Importante: Para la resolución de este ejercicio, utilice, al menos, dos estructuras de repetición distintas. Indique, como comentario (al final del archivo entregable) la forma de compilar, linkear y ejecutar. Además, deberá adjuntarse una imagen de la terminal con el programa ejecutando.

Ayuda: Implemente la generación de números aleatorios usando la función *srand* para asegurar la aleatoriedad de los valores de empuje y ángulo. Las funciones trigonométricas *cos* y *sin* serán necesarias para los cálculos de velocidad y altitud inicial.

Trabajo Práctico N°3

Contenido a evaluar: *Funciones en lenguaje C, Sistemas de numeración, Operadores a nivel de bit y campos de bit.*

Desarrollo de un Sistema de Cifrado y Descifrado de Mensajes para Dispositivos IoT

Como parte del grupo de sistemas de comunicaciones en una empresa especializada en dispositivos IoT (*Internet of Things*), se le solicita desarrollar un programa para garantizar la seguridad de la comunicación entre dispositivos. El objetivo es cifrar y descifrar mensajes de texto de manera sencilla utilizando una clave, para garantizar la seguridad de los datos transmitidos, de longitud variable de hasta 256 bits.

El programa debe ser capaz de cifrar y descifrar mensajes de texto de un máximo de 255 caracteres. Cada carácter debe cifrarse mediante una clave de 8 bits proporcionada por el usuario o generada de manera aleatoria. Para el cifrado, se utilizará el operador XOR para mezclar el mensaje con la clave, y además, se realizará un desplazamiento de 2 posiciones hacia la izquierda como paso adicional para aumentar la seguridad del proceso de cifrado.

El programa debe cumplir con las siguientes condiciones: debe mostrar el mensaje original, el mensaje cifrado en formato hexadecimal y el mensaje descifrado. El desarrollador debe implementar al menos dos funciones: cifrar y descifrar. Además, es necesario realizar todas las validaciones necesarias para asegurar el correcto funcionamiento del programa.

Ejemplo:

Entrada	Mensaje: “Hola” Clave: 0b10101010 (170 en decimal)
Salida	Mensaje Original: "Hola" Mensaje Cifrado: 0xd6 0xdb 0xca 0xd2 Mensaje Descifrado: "Hola"

Importante: El alumno debe entregar los archivos main.c , funciones.c y funciones.h.
Ninguna función realizada, incluyendo el main, puede superar las 35 líneas de código.

Trabajo Práctico N°4

Contenido a evaluar: *Arreglos en C, memoria dinámica (parte I). Argumentos del main.*

Gestión de Usuarios en Base de Datos

Desarrollar un programa que permita ingresar usuarios a una base de datos. El programa recibirá como argumento una lista de usuarios junto con sus respectivas fechas de nacimiento con el siguiente formato:

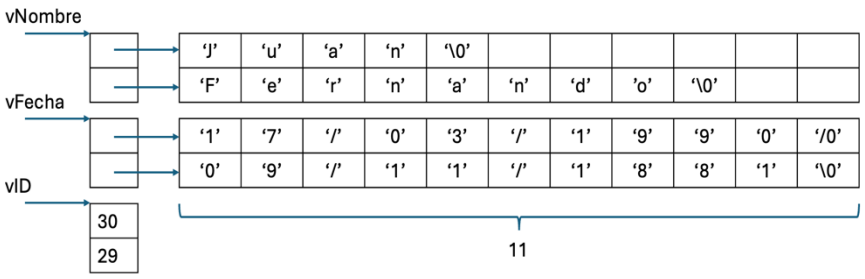
Juan-17/03/1990 Fernando-9/11/1881 Hector-11/9/1890

El objetivo del programa es procesar esta información y almacenarla en los siguientes arreglos:

- **vNombre**: Nombres de los usuarios (tipo char, 11 caracteres como máximo). Vector de punteros a string.
- **vFecha**: Fechas de nacimiento de los usuarios (tipo char, 11 caracteres). Vector de punteros a string.
- **vID**: Número identificatorio (tipo int). Vector de enteros. El ID de cada usuario se obtiene sumando todas las cifras de su fecha de nacimiento. Por ejemplo, para la fecha 17/03/1990, el ID sería 1 + 7 + 0 + 3 + 1 + 9 + 9 + 0 = 30.

En caso de que dos usuarios tengan el mismo ID (es decir, si sus fechas de nacimiento suman el mismo valor), solo se ingresará al primer usuario. Además, se debe tener en cuenta los formato de fecha de nacimiento posible a ser considerados: xx/xx/xxxx (formato completo), x/xx/xxxx (un solo dígito en el día) y x/x/xxxx (un solo dígito tanto en el día como en el mes). En la imagen se muestra un ejemplo para dos usuarios.

Una vez que los usuarios sean procesados e ingresados en el sistema, el programa debe mostrar por pantalla la lista de usuarios ordenada de forma ascendente según su ID. Además, debe informar la cantidad de usuarios que no pudieron ser ingresados correctamente debido a IDs repetidos, así como la cantidad de usuarios que fueron ingresados con éxito. En caso de que no se ingrese ningún usuario, el programa debe indicar que no hubo registros ni errores.



Ejemplo:

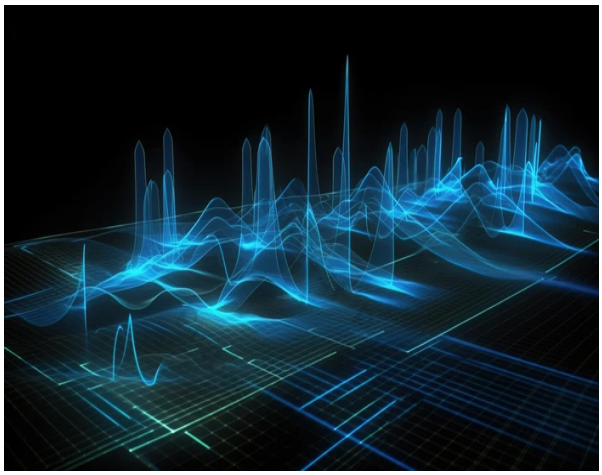
Ejemplo 1	Ejecución	./tp4
	Salida	Usuarios cargados al sistema: 0 Usuarios no cargados al sistema: 0
Ejemplo 2	Ejecución	./tp4 Juan-17/03/1990 Fernando-9/11/1881 Hector-11/9/1890
	Salida	ID: 29 – Nombre: Fernando (Fecha: 9/11/1881) ID: 30 – Nombre: Juan (Fecha: 17/03/1990) Usuarios cargados al sistema: 2 Usuarios no cargados al sistema: 1

Tener en cuenta: La cantidad de usuarios a ingresar es desconocida hasta el momento de ejecutar el programa.

Trabajo Práctico N°5

Contenido a evaluar: *Memoria dinámica (parte II). Puntero a función.*

Procesamiento digital de señales



En el campo de la Ingeniería Electrónica, el procesamiento de señales desempeña un papel fundamental en numerosas aplicaciones, como el tratamiento de audio, el análisis de señales biomédicas y los sistemas de comunicación, entre otros. Una estrategia eficiente en estos casos para diseñar sistemas modulares es el uso de punteros a función. En este trabajo práctico, se deberá desarrollar un sistema de procesamiento de señales que permita aplicar diferentes transformaciones a un conjunto de muestras de una señal ingresadas por el usuario.

El programa debe permitir almacenar una señal discreta (valores reales) en un arreglo y procesarla mediante distintas operaciones seleccionadas a través de un menú interactivo. Tenga en cuenta que cada transformación deberá implementarse como una función independiente.

Operaciones disponibles:

- **Atenuación:** Reduce la amplitud de la señal multiplicando cada muestra por un factor menor a 1.
- **Amplificación:** Aumenta la amplitud de la señal multiplicando cada muestra por un factor mayor a 1.
- **Filtrado de ruido:** Suaviza la señal aplicando un filtro de media móvil. En este caso, cada muestra se reemplaza por el promedio de sus valores vecinos, excepto la primera y la última que permanecerán sin cambios.

Una vez completada la operación de procesamiento, el programa deberá mostrar en pantalla la señal original, la transformación aplicada junto con su factor (si corresponde) y la señal resultante. El usuario podrá procesar los datos de manera acumulativa, aplicando múltiples transformaciones sucesivas, hasta que decida finalizar la ejecución del programa.

Importante: Tenga en cuenta que no se conoce el número de muestras a ingresar. Para garantizar una implementación **modular** y **flexible**, se deberá utilizar un **puntero a función** definido del siguiente modo:

```
int(*process_sig)(const float *in_signal, int tam_in, float factor, float
                  **out_signal, int *tam_out);
```

La función apuntada por process_sig deberá devolver **1** si la operación se realizó correctamente y **-1** en caso de error.

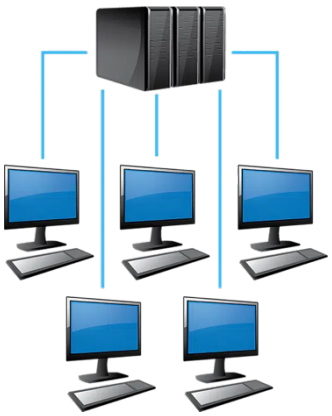
Parámetros de la función:

- **in_signal:** Señal de entrada. Se recomienda generar un vector de valores aleatorios (donde en dicho caso, el número de muestras debe ser solicitado)
- **tam_in:** Cantidad de muestras en la señal de entrada.
- **factor:** Factor de multiplicación utilizado en las funciones de "atenuación" y "amplificación". En el caso del **filtro de media móvil**, este parámetro debe tomar el valor constante 99.
- **out_signal:** Señal de salida procesada.
- **tam_out:** Tamaño de la señal de salida.

Trabajo Práctico N°6

Contenido a evaluar: *Memoria dinámica (parte III). Estructuras. Listas.*

Gestión de conexiones en un servidor



Se pide desarrollar un programa que simule la gestión de conexiones en un servidor, utilizando una lista enlazada. En este sistema, cada conexión aceptada se almacena en un nodo de la lista, con la restricción de que solo puede haber hasta **5 conexiones simultáneas**. Cada nodo de la lista representará una conexión y deberá contener los siguientes datos:

- **Dirección IP:** Formato xxx.xxx.xxx.xxx, donde cada xxx es un número entero entre 0 y 255.
- **Número de puerto:** Valor entero entre 1 y 32000.
- **ID de conexión:** Número aleatorio entre 1 y 20 el cual simula un identificador único de la conexión.

El programa funcionará mediante un menú interactivo que permitirá al usuario gestionar las conexiones. Para aceptar una nueva conexión, el usuario deberá ingresar una dirección IP y un puerto en el formato: **xxx.xxx.xxx.xxx:yyy**, donde la primera parte representa la IP y la segunda el número de puerto. Estos valores deberán ser extraídos y almacenados en los campos correspondientes a un nodo de la estructura de datos autoreferenciada. En caso de que el número máximo de conexiones permitidas ya esté ocupado, la solicitud deberá ser rechazada.

El sistema, también deberá permitir, en todo momento, la visualización del estado actual de las conexiones activas, mostrando para cada una su identificador, la dirección IP y el puerto asignado. Si el usuario desea cerrar una conexión específica, podrá hacerlo proporcionando el identificador de esta, tras lo cual el nodo correspondiente deberá ser eliminado de la lista. Finalmente, al seleccionar la opción de salida, el programa deberá cerrar las conexiones activas liberando toda la memoria utilizada.

Nota: Para garantizar claridad en la implementación, se requiere el uso de constantes simbólicas para definir valores como el límite máximo de conexiones, rango de puertos permitidos, intervalo de valores para la generación aleatoria de los identificadores, entre otras.

Ejemplo:

1. Aceptar conexión

Ingrese la dirección y puerto (Formato: xxx.xxx.xxx.xxx:yyy):
- 192.168.1.10:8080

1. Aceptar conexión

Ingrese la dirección y puerto (Formato: xxx.xxx.xxx.xxx:yyy):
- 10.0.0.5:3000

3. Ver conexiones

ID: 12 | IP: 129.168.1.10 | PUERTO: 8080
ID: 7 | IP: 10.0.0.5 | PUERTO: 3000

2. Cerrar conexión

Ingrese ID conexión a cerrar:
- 12

3. Ver conexiones

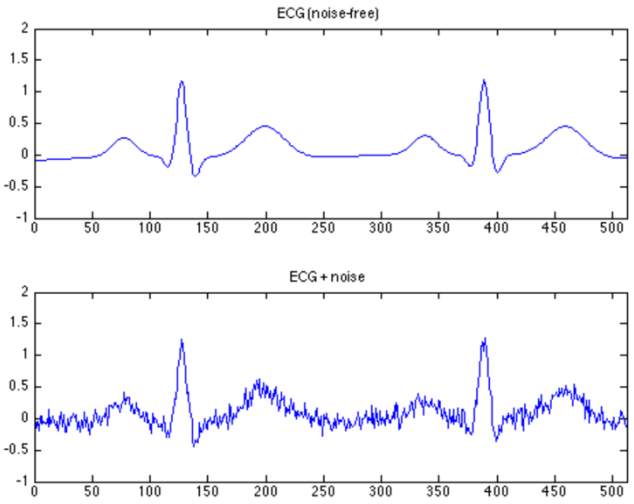
ID: 7 | IP: 10.0.0.5 | PUERTO: 3000

4. Salir

Trabajo Práctico N°7

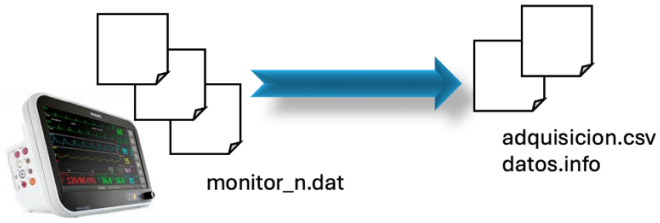
Contenido a evaluar: *Streams y señales. Archivos*

Adquisición de señales



Como ingenieros electrónicos, es común enfrentarnos al desafío de adquirir datos provenientes del mundo real antes de poder analizarlos o procesarlos. Esto requiere el uso de sensores y dispositivos asociados a la adquisición de señales. En este trabajo práctico, simularemos un sistema de adquisición de señales de electrocardiografía (ECG) que permite procesar y almacenar datos adquiridos de forma periódica. Es decir, este programa será capaz de adquirir señales simuladas utilizando un archivo de datos, muestrearlas según una frecuencia determinada, almacenarlas en un archivo de salida y generar un reporte estadístico con información relevante sobre los datos adquiridos.

Para simular el monitor que adquiere los datos, se utilizarán archivos con el formato monitor_n.dat, donde n es un número entero que identifica cada archivo (por ejemplo, monitor_1.dat, monitor_2.dat, etc.). Cada archivo contendrá las muestras de la señal como valores enteros dispuestos en forma de columna, representando la lectura continua de un monitor médico en tiempo real.



Modo de funcionamiento del programa

El programa deberá recibir dos parámetros a través de la línea de comandos (argumentos del main):

1. **Nombre del archivo de entrada:** El nombre del archivo (monitor_n.dat) que contiene las muestras de la señal en formato numérico, con un valor por línea.
2. **Frecuencia de muestreo (Fs):** Un número entero que indica la cantidad de muestras a tomar del archivo en un segundo. Este valor deberá estar en el rango de 40 a 120 Hz. (40 a 120 muestras en un segundo)

Una vez iniciado el programa deberá leer el archivo de entrada y extraer las muestras a la frecuencia de muestreo indicada. Para realizar esta tarea, el programa utilizará la señal **SIGALRM** para realizar adquisiciones cada 1 minuto. En cada intervalo de 1 minuto, el programa deberá leer y almacenar en un archivo de salida las muestras adquiridas durante ese periodo. Es decir, en caso de seleccionar una frecuencia de muestreo (Fs) de 100, se deberán tomar del archivo de entrada N muestras donde N es 100 (muestras por segundo) * 60 (segundos).

Una vez iniciado, el programa debe realizar las siguientes acciones:

1. **Adquisición periódica de datos:** Utilizando la señal **SIGALRM**, el programa deberá realizar la adquisición cada minuto. En cada intervalo, se tomarán N muestras del archivo de entrada, donde: $\mathbf{N = Fs * 60}$.

   Estas muestras se almacenarán en un archivo de salida llamado adquisición.csv con el siguiente formato:

   - La primera columna corresponde al **timestamp** (marca de tiempo) el cual indica el momento en que se realizó el proceso de adquisición (momento en el cual se comienza a guardar las muestras en el archivo adquisición.csv). Este valor será el mismo para todas las muestras correspondientes al mismo período de adquisición.
   - La segunda columna debe contener los valores de las muestras adquiridas.

2. **Manejo de señales externas:** El programa debe ser capaz de manejar señales externas, tales como la señal de interrupción (**SIGINT**) y la señal de suspensión (**SIGSTP**). En ambos casos, el programa deberá imprimir un mensaje indicando que la señal fue recibida pero no se tomará ninguna acción al respecto a ella y continuará su ejecución normalmente.
3.
4. **Generación de un reporte estadístico**: Al finalizar la adquisición de la señal (es decir, una vez que se hayan leído todas las muestras del archivo de entrada), el programa deberá abrir y en caso de no existir crear un archivo adicional (datos.info). Este archivo contendrá un resumen estadístico de los datos adquiridos, con la siguiente información:

   - **Valor máximo:** El valor más alto entre todas las muestras adquiridas.
   - **Valor mínimo:** El valor más bajo entre todas las muestras adquiridas.
   - **Valor medio:** El promedio de todas las muestras adquiridas.
   - **Frecuencia de muestreo:** El valor de Fs utilizado durante la adquisición.
   - **Cantidad de muestras:** El número total de muestras adquiridas durante la ejecución del programa.

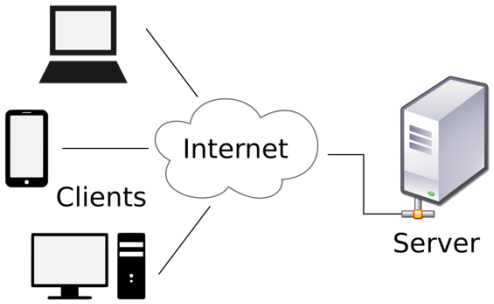
Importante: Tenga en cuenta que el programa debe poder manejar archivos grandes, por lo que debe ser implementado mediante un control eficiente de la memoria.

Trabajo Práctico N°8

Contenido a evaluar: *Multiprocesamiento. Threads. Redes.*

Comunicación Cliente-Servidor con Multiusuarios usando Threads

Dado el excelente trabajo realizado previamente en la gestión de conexiones mediante estructuras autoreferenciadas (Trabajo práctico 6), ahora se nos ha encargado el desarrollo de un **sistema de comunicación cliente-servidor con soporte para múltiples usuarios**. En esta nueva versión, el servidor deberá ser capaz de aceptar hasta cinco conexiones simultáneas y recibir mensajes en tiempo real de los clientes conectados.



Cada cliente se conectará automáticamente al servidor utilizando un puerto disponible, sin necesidad de ingresarlo manualmente *(se recomienda el uso de estructuras de repetición para encontrar un puerto libre)*. Una vez establecida la conexión, el servidor mostrará en pantalla un mensaje indicando la nueva conexión, incluyendo un identificador aleatorio (generado en el servidor) y la dirección IP. Cuando un cliente envíe un mensaje, el servidor deberá imprimirlo en pantalla. Si el cliente decide cerrar su conexión, el servidor también deberá notificar su desconexión.

SERVIDOR:

El servidor gestionará las conexiones y la recepción de mensajes mediante **threads**, permitiendo el manejo concurrente de múltiples clientes. Si el límite de cinco conexiones activas se alcanza, cualquier cliente adicional que intentara conectarse deberá recibir un mensaje de rechazo.

CLIENTE:

El cliente podrá enviar mensajes al servidor de forma continua mientras la conexión esté activa. Una vez que el usuario decida cerrar la aplicación cliente, la conexión se cerrará, y el servidor notificará que el cliente ha finalizado su sesión.

Importante: En este ejercicio no hay especificaciones adicionales acerca de la implementación. Considere utilizar todas las buenas prácticas de programación vistas en clase y utilizadas a lo largo de esta guía de trabajos prácticos.