Departamento de Ingeniería Electrónica

Guía de Trabajos Prácticos

INFORMATICA I

2026

Introducción y régimen de aprobación

La presente guía de Trabajos Prácticos tiene como objetivo llevar a la práctica los contenidos desarrollados en las clases teóricas. De este modo, se busca promover una realimentación constante entre la aplicación práctica y la lectura de los diferentes conceptos teóricos, que permita al alumno desarrollar un enfoque metodológico para la resolución de problemas de Ingeniería. Para ello, se utilizarán las herramientas de software indicadas por la cátedra resolviendo los algoritmos planteados en Lenguaje C.

El grado de complejidad de los ejercicios irá incrementándose progresivamente a lo largo de las distintas Unidades Temáticas.

La entrega de cada ejercicio deberá realizarse, sin excepciones, en las fechas estipuladas por el docente a cargo del curso.

Las fechas de entrega de los trabajos prácticos obligatorios estarán organizadas de manera tal que todos los trabajos prácticos correspondientes a los contenidos evaluados en cada examen sean corregidos por los docentes auxiliares con anterioridad al mismo. De esta forma, los alumnos recibirán una devolución con las correcciones de los errores detectados, como parte del proceso de realimentación necesario para la adecuada preparación del examen.

La no entrega de un ejercicio en la fecha establecida será considerada como **Ausente** en dicho práctico y, en consecuencia, el mismo será clasificado como **No Aprobado**. De acuerdo con el reglamento vigente, la aprobación de los trabajos prácticos requiere la aprobación de al menos el 80% de los mismos.

Asimismo, la aprobación final de los trabajos prácticos podrá requerir una **defensa oral**, en la cual el alumno deberá demostrar el conocimiento adquirido durante el desarrollo de los mismos.

Formato de presentación

- Los archivos fuentes deberán incluir, en todos los casos, los comentarios necesarios que permitan clarificar su lectura y comprensión.
- Cada subrutina o función deberá contar con un encabezado que describa la operación que realiza, los parámetros de entrada y los resultados que presenta, indicando el formato y el método de entrega. Para tal fin, los docentes proveerán al alumno un *template* que deberá ser utilizado obligatoriamente en todos los ejercicios.
- En el archivo fuentes principal (`nombre_programa.c`) deberá incluirse un comentario que explique claramente las instrucciones necesarias (comandos) para la compilación, el linkeo y la ejecución del programa.

  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`). Dicho archivo deberá contener **exclusivamente** el código fuente y una imagen de la terminal que evidencie el **correcto funcionamiento** del programa desarrollado.

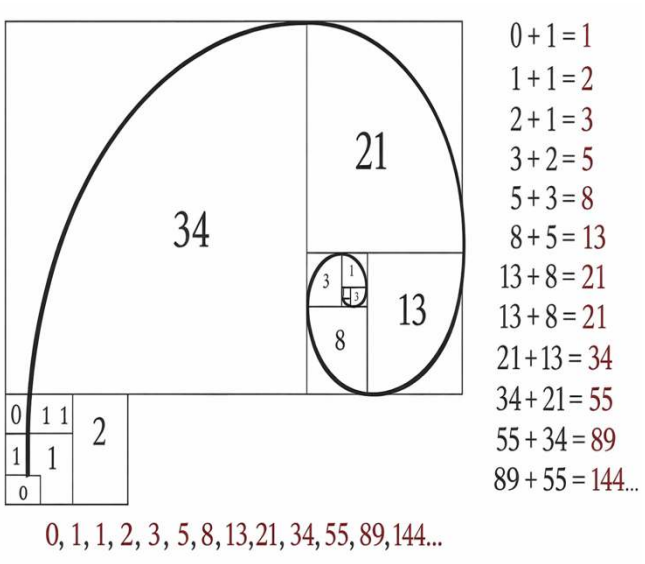
Trabajo Práctico N°1

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

Sucesión de Fibonacci

En matemática, la sucesión de Fibonacci es una secuencia de números enteros conocida desde la antigüedad, atribuida a matemáticos hindúes alrededor del año 1135 y descripta por primera vez en Europa por Leonardo de Pisa, también conocido como Fibonacci.

La sucesión comienza con los valores 0 y 1, y cada término posterior se obtiene como la suma de los dos términos anteriores. A estos valores, también llamados coeficientes, se los denomina **números de Fibonacci** (ver figura).



Ejemplos:

n = 5	0 1 1 2 3
n = 7	0 1 1 2 3 5 8

Se solicita desarrollar un algoritmo que permita al usuario ingresar dos valores enteros positivos distintos, los cuales definirán los extremos de un intervalo numérico. Uno de ellos representará el límite inferior y el otro el límite superior, independientemente del orden en que sean ingresados por el usuario. En caso de que ambos valores ingresados sean iguales o uno u ambos valores sean negativos, el programa deberá solicitar nuevamente el ingreso de los datos, repitiendo el proceso hasta que ambos valores ingresados sean válidos.

Una vez ingresados dos valores válidos, el programa deberá calcular y mostrar por pantalla la suma de todos los coeficientes de la sucesión de Fibonacci cuyos valores se encuentren comprendidos en el intervalo $[extremo_1, extremo_2]$, inclusive.

Ejemplo: Para los valores de entrada **4 y 23**, se considerarán para la suma los coeficientes de la sucesión de Fibonacci **5, 8, 13 y 21** mostrando por pantalla **"La suma es 47"**.

Ayuda: Antes de comenzar con el desarrollo del algoritmo, se recomienda identificar y describir los datos del problema, las variables necesarias, las entradas, los procesos requeridos y las salidas esperadas.

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 de una máquina expendedora de billetes

Se propone desarrollar un programa que simule la operación de extracción de dinero de un cajero automático. El programa deberá solicitar al usuario el ingreso de una **cantidad de dinero a extraer**, expresada como un **valor real positivo**, y calcular la **cantidad de billetes y monedas necesarias** para alcanzar dicho monto.

El cajero automático únicamente puede entregar **billetes de $10 y $5**, y **monedas de $1, $0.50 y $0.25**, asumiendo que dispone de una cantidad ilimitada de cada denominación. En todos los casos, el cajero deberá **priorizar la entrega de la mayor denominación posible**.

Ejemplos:

- Al extraer **$30**, se entregarán **3 billetes de $10**, y no 6 billetes de $5.
- Al extraer **$4**, se entregarán **4 monedas de $1**.

Tenga en cuenta que, en caso de no poder alcanzar exactamente el monto ingresado por el usuario con las denominaciones disponibles, deberá **entregar una moneda adicional de $0.25**.

El programa deberá mostrar por pantalla la **cantidad de billetes y/o monedas entregadas**, comenzando por las denominaciones de mayor valor. Finalmente, si el monto total entregado resulta **mayor** al solicitado, deberá **informar el excedente entregado** antes de finalizar su ejecución.

Ejemplos con posible formato de salida:

[Máquina] Cuánto desea retirar? [Usuario] $31.25 [Máquina] 3 billetes de $10 [Máquina] 1 moneda de $1 [Máquina] 1 moneda de $0.25	[Máquina] Cuánto desea retirar? [Usuario] $31.26 [Máquina] 3 billetes de $10 [Máquina] 1 moneda de $1 [Máquina] 2 monedas de $0.25 [Máquina] Se devolvieron $31.50 ($0.24 adicionales por falta de cambio)	[Máquina] Cuánto desea retirar? [Usuario] -$5 [Máquina] No puede ingresar un número negativo

Trabajo Práctico N°3

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

Cifrado y Descifrado de Mensajes para Dispositivos IoT

Como parte del grupo de sistemas de comunicaciones de una empresa especializada en dispositivos IoT (*Internet of Things*), se le solicita desarrollar un programa que permita **cifrar y descifrar mensajes de texto**, utilizando una clave para garantizar la seguridad de la información transmitida entre dispositivos.

El programa deberá ser capaz de cifrar y descifrar mensajes de texto de **longitud variable**, con un máximo de **255 caracteres**. Cada carácter del mensaje deberá cifrarse utilizando una **clave de 8 bits**, la cual podrá (a) ser ingresada manualmente por el usuario, o (b) ser generada de manera aleatoria por el programa.

El mecanismo de cifrado y descifrado deberá implementarse utilizando el **operador lógico XOR**, aplicándolo entre cada carácter del mensaje y la clave. Además, el programa deberá cumplir con las siguientes especificaciones:

- Mostrar por pantalla el **mensaje original**, el **mensaje cifrado** expresado en **formato hexadecimal** y el **mensaje descifrado**, verificando que este último coincida con el mensaje original.
- Implementar, como mínimo, las siguientes funciones: `cifrar()` y `descifrar()`.

Realice todas las **validaciones necesarias** para asegurar el correcto funcionamiento del programa (longitud del mensaje, rango de la clave, opciones ingresadas por el usuario, etc)

Ejemplo:

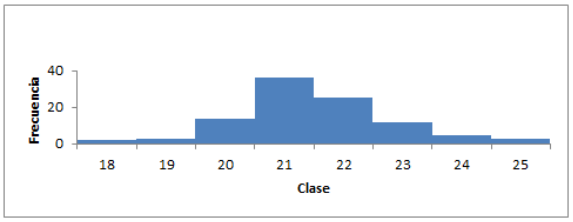
Entrada	Mensaje: "`Hola`" Clave: `0b10101010` (170 en decimal)
Salida	Mensaje Original: "`Hola`" Mensaje Cifrado: `0xd6 0xdb 0xca 0xd2` Mensaje Descifrado: "`Hola`"

Importante: Debe entregarse los archivos main.c , funciones.c y funciones.h. Ninguna función realizada, incluyendo la función main, puede superar las 35 líneas de código.

Trabajo Práctico N°4

Contenido a evaluar: *Arreglos en C*

Histograma



En estadística, un **diagrama de frecuencias, o histograma**, es una representación gráfica de una variable cuantitativa discreta mediante barras, donde en el eje horizontal se representan los valores que puede tomar la variable y la altura de cada barra es proporcional a la frecuencia de aparición de dichos valores (cantidad de repeticiones en el conjunto de datos). Este tipo de diagrama (ver imagen), aplicando a un conjunto de valores como:

$$\{\mathbf{18}, \mathbf{18}, \mathbf{19}, \mathbf{19}, \mathbf{19}, \mathbf{20}, \mathbf{20}, \mathbf{20}, \dots, \mathbf{21}, \dots, \mathbf{22}, \mathbf{22}, \dots, \mathbf{23}, \mathbf{23}, \dots, \mathbf{24}, \mathbf{24}, \dots, \mathbf{25}, \mathbf{25}, \mathbf{25}\}$$

Permite visualizar la distribución de la muestra y obtener una visión general de su tendencia.

En este ejercicio se solicita desarrollar un programa que permita obtener un **diagrama de frecuencias de letras** (de `a` a `z` y de `A` a `Z`) a partir de **palabras ingresadas por teclado**. Tenga en cuenta que la cantidad de palabras a ingresar **no se conoce a priori**. El ingreso de datos deberá finalizar cuando el usuario ingrese la palabra especial **`\fin`**.

El programa deberá contabilizar la frecuencia de aparición de cada letra y **mostrar por pantalla todas las letras posibles junto con su respectiva frecuencia**.

Ayuda: Con el objetivo de facilitar la resolución del ejercicio:

- No se requiere realizar validaciones sobre los caracteres ingresados por el usuario.
- Cada palabra podrá tener como máximo **15 caracteres**, pertenecientes al alfabeto del idioma inglés.

Ejemplo: Palabras ingresadas por el usuario: Hola Mi nombre es FERNANDO \fin

[Máquina] Ingreses las palabras a analizar [Usuario] Hola [Usuario] Mi [Usuario] nombre [Usuario] es [Usuario] FERNANDO [Máquina] [a] : 1 [Máquina] [b] : 1 ----------→	[Máquina] [e] : 2 [Máquina] [i] : 1 [Máquina] [l] : 1 [Máquina] [m] : 1 [Máquina] [n] : 1 [Máquina] [o] : 2 [Máquina] [r] : 1 [Máquina] [s] : 1 ----------→	[Máquina] [A] : 1 [Máquina] [D] : 1 [Máquina] [E] : 1 [Máquina] [F] : 1 [Máquina] [H] : 1 [Máquina] [M] : 1 [Máquina] [N] : 2 [Máquina] [O] : 1 [Máquina] [R] : 1

Importante: Debe entregarse los archivos main.c, funciones.c y funciones.h. Ninguna función realizada, incluyendo la función main, puede superar las 35 líneas de código.

Trabajo Práctico N°5

Contenido a evaluar: *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:

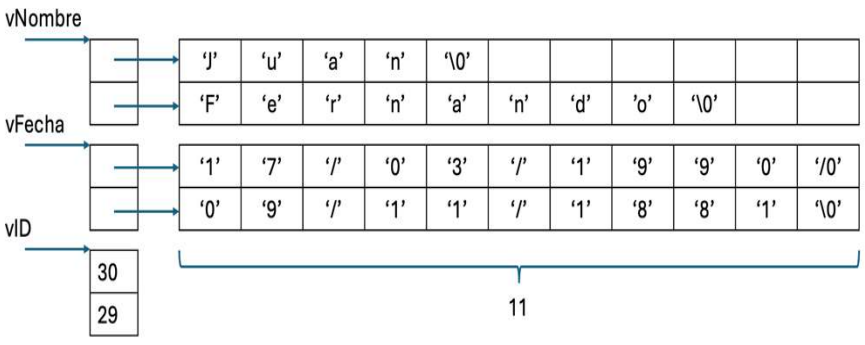
Juan-17/03/1990 Fernando-9/11/1881 Hector-11/9/1890

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

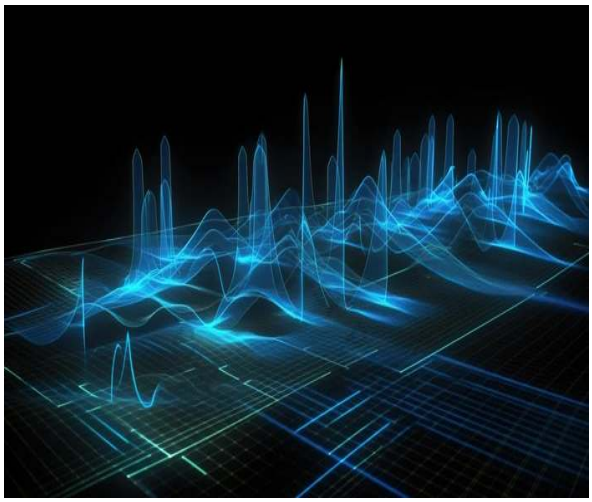
Tener en cuenta: La cantidad de usuarios a ingresar es desconocida hasta el momento de ejecutar el programa. Recuerde verificar ID disponible previo pedido de memoria.

Importante: Debe entregarse los archivos main.c, funciones.c y funciones.h. Ninguna función realizada, incluyendo la función main, puede superar las 35 líneas de código.

Trabajo Práctico N°6

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 rol fundamental en múltiples aplicaciones, tales como el tratamiento de audio, el análisis de señales biomédicas y los sistemas de comunicación. Una estrategia eficiente para el diseño de sistemas **modulares y flexibles** en este contexto 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 discreta ingresada por el usuario.

El programa deberá:

- Almacenar una señal discreta compuesta por **valores reales** en un arreglo dinámico.
- Permitir al usuario procesar dicha señal mediante distintas operaciones seleccionadas a través de un **menú interactivo**.
- Implementar cada transformación como una **función independiente**, accesible a través de un puntero a función.
- Permitir la aplicación **acumulativa** de múltiples transformaciones sobre la señal, hasta que el usuario decida finalizar la ejecución del programa.

Operaciones disponibles: Las transformaciones que deberá implementar el sistema son las siguientes:

- **Atenuación:** Reduce la amplitud de la señal multiplicando cada muestra por un factor **menor que 1**.
- **Amplificación:** Aumenta la amplitud de la señal multiplicando cada muestra por un factor **mayor que 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 muestra que deberán **permanecer sin cambios**.

Una vez completada la operación de procesamiento, el programa deberá mostrar en pantalla **la señal original**, la **transformación aplicada** (incluyendo el factor utilizado, cuando corresponda) y la **señal resultante** luego del procesamiento.

Importante: Dado que **no se conoce a priori la cantidad de muestras** a ingresar, el programa deberá utilizar memoria dinámica. Para garantizar una implementación modular, se deberá emplear un puntero a función definido de la siguiente manera:

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

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

Parámetros de la función:

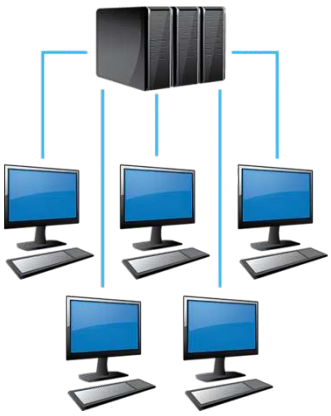
- **in_signal**: Señal de entrada. *Se recomienda generar un vector de valores aleatorios; en este caso, el número de muestras deberá ser solicitado al usuario.*
- **tam_in**: Cantidad de muestras en la señal de entrada.
- **factor**: Factor de multiplicación utilizado en las funciones de **atenuación** y **amplificación**. Para el filtro de **media móvil**, este parámetro deberá tomar el valor constante `99`.
- **out_signal**: Puntero a la señal de salida procesada.

Importante: Debe entregarse los archivos main.c, funciones.c y funciones.h. Ninguna función realizada, incluyendo la función main, puede superar las 35 líneas de código.

Trabajo Práctico N°7

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

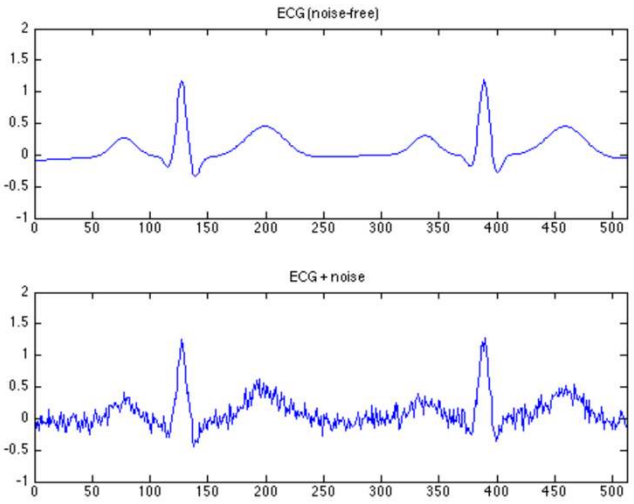
ID: 7 | IP: 10.0.0.5 | PUERTO: 3000

4. Salir

Trabajo Práctico N°8

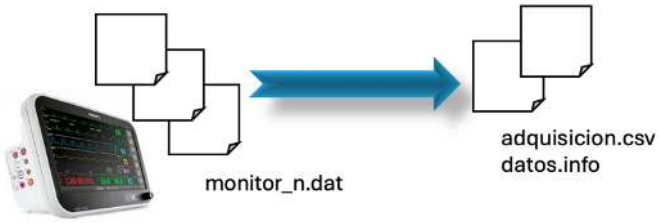
Contenido a evaluar: *Streams y señales. Archivos*

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.

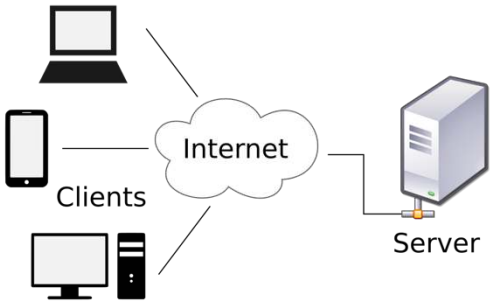
Importante: Tenga en cuenta que el programa debe poder manejar archivos grandes, por lo que debe ser implementado mediante un control eficiente de la memoria.

Trabajo Práctico N°9

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.