DIIT Departamento de Ingeniería e Investigaciones Tecnológicas	**TÓPICOS DE PROGRAMACIÓN (3635)**	Universidad Nacional de La Matanza

Guía Complementaria de Actividades Prácticas No Obligatorias

Carrera: Ingeniería Informática

Plan: 2023

Materia: Tópicos de Programación (3635)

Año Lectivo: 2023

Contenido:
Descripción de la materia
Programa
Bibliografía
Cronograma
Guía de Trabajos Prácticos

Docentes:
Álvarez, Erik
Cacho Mendoza, Ariel
Calaz, Ezequiel
Cuesta, Cristian
Ghigo, Paola
González, Giselle
Guatelli, Renata
Jordi, Brian
Martínez, Pablo
Mendoza, Matías
Pan, Néstor
Pezzola, Federico
Soligo, Pablo
Uran Acevedo Jónathan

Descripción de la asignatura

La asignatura “Tópicos de programación”, refuerza los conocimientos adquiridos en asignaturas previas introduciendo a la programación imperativa estructurada soluciones alternativas mediante recursividad. Además presenta el concepto de reusabilidad, a través del diseño de algoritmos independientes del tipo de dato y de una acción particular de forma de crear soluciones realistas. Se presentan estrategias de diseño de algoritmos para mejorar soluciones ineficientes y diferentes implementaciones para un problema en particular, trabajando sobre tipos de datos simples, compuestos, estáticos y dinámicos.

Metodología de enseñanza

Se utilizan metodologías activas de enseñanza, especialmente aprendizaje basado en problemas (ABP). Se presenta cada unidad temática introduciendo los conceptos fundamentales realizando analogías con ejemplos reales, que permite relacionar los contenidos de la materia con las herramientas habituales de trabajo.

Los contenidos de la asignatura se presentan de forma iterativa e incremental, de forma que le permitan al estudiante, construir sus propios procedimientos para resolver una situación problemática, lo que implica que sus ideas puedan verse modificadas y de esta forma siga construyendo nuevos conocimientos.

La materia tiene una fuerte carga práctica. En la misma se resuelven problemas, desarrollando pequeños sistemas, aplicando lo visto en las clases, trabajando de forma colaborativa, simulando un entorno de trabajo real.

Se motiva a los estudiantes en el uso de los foros de la plataforma MIeL y recursos de Teams, para la resolución de dudas, tanto de conceptos teóricos como prácticos, permitiendo desarrollar las capacidades de comunicación y afianzar el uso del lenguaje técnico. Además, la cátedra cuenta con soporte digital de los contenidos, que los estudiantes pueden consultar luego de haber asistido a la clase.

Objetivos de aprendizaje

A través de esta asignatura, el alumno habrá adquirido los conocimientos necesarios y suficientes para estar en condiciones de:

Objetivos Generales:

La materia se desarrolla teniendo en cuenta los siguientes objetivos generales:

- analizar, plantear y resolver situaciones problemáticas.
- organizar y planificar su trabajo.
- hacer transferencia de los conocimientos teóricos a la práctica.
- adquirir la capacidad de trabajar en equipo.
- identificar un problema a partir de una situación problemática presentada.
- diseñar un algoritmo eficiente para la resolución de una situación problemática analizada.
- desarrollar un algoritmo utilizando un lenguaje de programación.
- plantear casos de prueba de forma tal de ver los casos generales y particulares de cada situación problemática planteada.
- expresar los contenidos teóricos de la materia y su vinculación con situaciones de la vida real.
- detectar la fuerte vinculación de esta asignatura con materias de años anteriores y posteriores del plan de carrera.

- integrar grupos de trabajo, potenciando su propio aprendizaje a través de la interacción y cooperación con sus pares.
- utilizar con fluidez el lenguaje técnico relacionado con la materia.

Objetivos Específicos:

- Aplicar los principios de la Programación Estructurada.
- Aplicar el concepto de reusabilidad.
- Manejar con solvencia los diferentes tipos de pasaje de parámetros, tanto de variables como de funciones.
- Aplicar sus conocimientos de algoritmia a colecciones de datos.
- Implementar algoritmos realistas independientes del tipo de dato.
- Diseñar e implementar soluciones que no dependan de un tipo de dato en particular, de forma que puedan utilizarse para diversos tipos de datos y puedan realizar diferentes acciones en función de la acción que reciba como parámetro, administrando correctamente la memoria en tiempo de ejecución.
- Entender, decidir y defender sobre alternativas de implementación conociendo las implicancias de cada una.
- Entender las diferentes estrategias de persistencia y aplicarla en ejemplos concretos.

Contenidos mínimos

Algorítmica. Estructuras de control. Tipos de datos simples y compuestos. (estáticos y dinámicos). Procedimientos y funciones. Concepto de reusabilidad. Estrategias de diseño de algoritmos.

Competencias a desarrollar

Genéricas

- **Competencias tecnológicas**
  - Identificar, formular y resolver problemas de ingeniería.
  - Concebir, diseñar y desarrollar proyectos de ingeniería.
  - Gestionar, planificar, ejecutar y controlar proyectos de ingeniería.
  - Utilizar de manera efectiva las técnicas y herramientas de aplicación en la ingeniería.

- **Competencias sociales, políticas y actitudinales**
  - Desempeñarse de manera efectiva en equipos de trabajo.
  - Comunicarse con efectividad.
  - Actuar con ética, responsabilidad profesional y compromiso social, considerando el impacto económico, social y ambiental de su actividad en el contexto local y global.
  - Aprender en forma continua y autónoma.
  - Actuar con espíritu emprendedor.
  - Capacidad de análisis y síntesis.
  - Toma de decisiones.
  - Razonamiento crítico.

Específicas

- Capacidad de resolución de problemas aplicando conocimientos de matemáticas, ciencias e ingeniería.
- Tener capacidad para realizar la formalización y especificación de problemas reales cuya solución requiere el uso de la informática.

- Capacidad para analizar, diseñar, construir y mantener aplicaciones de forma robusta, segura y eficiente, eligiendo las estructuras de control más adecuadas, administrando de forma forma eficiente los recursos disponibles.

Programa analítico	
Unidad 1	**Arrays** Arrays unidimensionales. Concepto de puntero. Aritmética de Punteros. Manejo de arrays unidimensionales con diferentes notaciones. Arrays numéricos. Cadenas. Funciones de biblioteca. Arrays multidimensionales. Manejo de arrays multidimensionales con diferentes notaciones. Uso de arrays.
Unidad 2	**Tipos de memorias** Tipos de memoria. Memoria de pila. Memoria Heap. Manejo de memoria en tiempo de ejecución. Ventajas y desventajas de cada tipo de memoria.
Unidad 3	**Pasaje de parámetros** Concepto de pasaje de parámetros. Pasaje de parámetros por copia o valor. Pasaje de parámetros por dirección. Punteros a variables. Punteros a funciones. Funciones de biblioteca de ordenamiento y búsqueda. Creación de funciones independientes del tipo de dato. Concepto de funciones map, filter y reduce. Argumentos en el main.
Unidad 4	**Tipo de Dato Abstracto (TDA)** Concepto de Tipo de Dato Abstracto. Creación e implementación de un TDA. Diferencia entre TDA y estructuras de datos.
Unidad 5	**Persistencia de memoria** Archivos binarios y de texto, su creación, modos de acceso, posicionamiento, cierre, eliminación, y funciones relacionadas. Creación de archivos como lotes de prueba. Merge o apareo de archivos.
Unidad 6	**Introducción a la Recursividad** Concepto de recursión. Estructura de funciones recursivas. Diferencias entre recursividad e Iteración.

Planificación de actividades

Semana	Clase	Actividad	Tipo	Duración estimada	Unidad /des
Semana 1 27/03	1	Presentación de los docentes del curso, breve explicación de las pautas generales de la materia. Breve repaso de conceptos adquiridos en “Programación Estructurada Básica”. Trabajar en forma modular. Creación de funciones. Pasaje de parámetros por copia o valor.	Teoría - Práctica en laboratorio o virtual sincrónico	4 hs	3
Semana 2 03/04	2	Breve explicación de cómo trabajar con proyectos y cómo documentar. Edición, compilación, depuración y ejecución de	Teoría - Práctica en laboratorio	4 hs	1 - 3

		programas, pautas de trabajo en laboratorio. Arrays unidimensionales. Concepto de puntero. Aritmética de punteros. Relación entre punteros y vectores. Arrays numéricos. Uso de punteros y aritmética de punteros en lugar del uso de índices para arrays unidimensionales. Pasaje de parámetros por dirección. Variables dimensionadas como parámetros. **Presentación del trabajo práctico.**	o virtual sincrónico		
Semana 3 10/04	3	Arrays de caracteres. Uso de punteros y aritmética de punteros en lugar del uso de índices para arrays de caracteres. Funciones de conversión. Revisión de funciones de biblioteca, reemplazo de las mismas por versiones propias. Estrategias para buscar, contar, etc. palabras en una cadena. Estrategia para resolver una función que permite 'normalizar' una cadena de caracteres.	Teoría - Práctica en laboratorio o virtual sincrónico	4 hs	1
Semana 4 17/04	4	Arrays Multidimensionales. Ejemplificación del uso de arrays bidimensionales. Arrays de punteros. Aritmética de punteros en lugar del uso de índices para arrays bidimensionales. Punteros a punteros. Desarrollar funciones sobre matrices. Recorridos y procesamiento de arrays de más de una dimensión. Argumentos a main.	Teoría - Práctica en laboratorio o virtual sincrónico	4 hs	1
Semana 5 24/04	5	Tipos de memorias. Gestión de memoria dinámica, funciones de solicitud y devolución de memoria. Ejemplificación. Creación de arrays en tiempo de ejecución. Memoria de Pila. Memoria heap.	Teoría - Práctica en laboratorio o virtual sincrónico	4 hs	2
Semana 6 01/05	6	Punteros a funciones. Concepto de punteros a función. Definición de punteros a función. Uso de punteros a función. Función de ordenamiento. Desarrollo de funciones propias que permitan ordenar vectores de cualquier tipo de dato, utilizando diferentes métodos de ordenamiento. Desarrollo de funciones independientes del tipo de dato.	Teoría - Práctica en laboratorio o virtual sincrónico	4 hs	3
Semana 7 08/05	7	Estructuras de datos. Punteros a estructuras de datos. Concepto de archivo. Tipos de archivos. Archivos binarios y de texto, modos de apertura. Archivos binarios. Acceso secuencial. Ejemplificación. Desarrollo de funciones genéricas para operar sobre archivos.	Teoría - Práctica en laboratorio o virtual sincrónico	4 hs	5

Semana 8 15/05	8	Actualización masiva de archivos binarios. Búsqueda de un registro por su clave o por su posición relativa dentro del archivo y actualización del mismo. Desarrollo de funciones genéricas para operar sobre archivos.	Teoría - Práctica en laboratorio o virtual sincrónico	4 hs	3 - 5
Semana 9 22/05	9	Archivos de texto. Archivos de texto de longitud fija y variable. Convertir información de archivos de texto en binarios y viceversa. Ejercitación sobre conversión de archivos.	Teoría - Práctica en laboratorio o virtual sincrónico	4 hs	3 - 5
Semana 10 29/05	10	Proceso de merge. Creación de archivos como lotes de prueba. Ejercitación sobre archivos de diferentes tipos.	Teoría - Práctica en laboratorio o virtual sincrónico	4 hs	3 - 5
Semana 11 05/06	11	Presentación del concepto de TDA. Conveniencia de trabajar con tipos abstractos de datos. Diferencia entre estructura de datos y TDA. Presentación del TDA fecha. Breve explicación de cómo documentar. Armado de un proyecto, pruebas de las distintas operaciones.	Teoría - Práctica en laboratorio o virtual sincrónico	4 hs	4
Semana 12 12/06	12	Ejercitación sobre todos los temas vistos hasta el momento. Consultas sobre el trabajo práctico.	Teoría - Práctica en laboratorio o virtual sincrónico	4 hs	1 - 2 -3 - 4 -5
Semana 13 19/06	13	Parcial 1 Entrega trabajo práctico	Teoría - Práctica en laboratorio o virtual sincrónico	4 hs	1 - 2 -3 - 4 -5
Semana 14 26/06	14	Devolución parcial 1. Defensa del TP. Introducción a la recursividad. Concepto. Elementos característicos de las funciones recursivas. Ventajas y desventajas de implementaciones recursivas vs. iterativas.	Teoría - Práctica en laboratorio o virtual sincrónico	4 hs	6
Semana 15 03/07	15	Recuperatorio Parcial 1 Defensa del TP.	Teoría - Práctica en laboratorio o virtual sincrónico	4 hs	1 - 2 -3 - 4 -5 - 6
Semana 16 10/07	16	Entrega de notas finales. Defensa trabajo práctico	Teoría - Práctica en laboratorio o virtual sincrónico	4 hs	1 - 2 -3 - 4 -5 - 6

Evaluación
El proceso de evaluación consta de: • un examen parcial presencial • un trabajo práctico grupal. La aprobación del trabajo práctico conlleva la aprobación del código entregado y una defensa oral individual del mismo. • un examen recuperatorio presencial. Las evaluaciones podrán ser teórico - prácticas. o únicamente prácticas o teóricas. Las evaluaciones teóricas constan de preguntas para desarrollar y/o preguntas del tipo opción múltiple y/o verdadero / falso, justificando la respuesta o no. Las evaluaciones prácticas constan de la resolución de ejercicios en máquina. Desarrollando el código necesario para resolver el problema solicitado y el lote de pruebas correspondiente.

Primera evaluación	Semana 13	teórico - práctica	4 hs
Aprobación TP Grupal	A partir de la semana 13	teórico - práctica	4 hs
Recuperatorio	Semana 15	teórico - práctica	4 hs

Bibliografía obligatoria [Disponibles en la Biblioteca Leopoldo Marechal, o con acceso digital]

Título	Autor	Editorial	Edición	Año
El Lenguaje de Programación C	Kernighan y Ritchie	Prentice Hall	2.ed.	1991
Cómo Programar en C / C++	Deitel y Deitel	Prentice Hall	2a. ed.	1994
Apuntes de cátedra	Docentes de la cátedra			

Bibliografía complementaria recomendada ([disponible en la Biblioteca Leopoldo Marechal, o con acceso digital]

Título	Autor	Editorial	Edición	Año
Código Limpio	Robert Cecil Martin	Anaya Multimedia		

Otros recursos obligatorios [Videos, enlaces, otros. Incluir una fila por cada recurso]

Nombre	

Otros recursos complementarios [Videos, enlaces, otros. Incluir una fila por cada recurso]

Nombre	
https://pythontutor.com/c.html#mode=edit	Permite visualizar en el navegador lo que la computadora está haciendo paso a paso mientras ejecuta un programa.
https://learngitbranching.js.org/	Una herramienta de visualización Git interactiva para educar y desafiar

Tema: Tipos de datos del lenguaje. Estructuras de control. Estructuras iterativas.

Unidad: Ejercicios de repaso.

Objetivo: Revisión de los principales conceptos introducidos en la materia Programación Estructurada Básica.

Competencia/s a desarrollar:

Genéricas

- **Competencias tecnológicas**
  - Identificación, formulación y resolución de problemas de ingeniería en sistemas de información/informática.
  - Concepción, diseño y desarrollo de proyectos de ingeniería en sistemas de información / informática.
  - Gestión, planificación, ejecución y control de proyectos de ingeniería en sistemas de información / informática.
  - Utilización de técnicas y herramientas de aplicación en la ingeniería en sistemas de información / informática.
- **Competencias sociales, políticas y actitudinales**
  - Desempeño en equipos de trabajo.
  - Comunicación efectiva.
  - Actuación profesional ética y responsable.
  - Aprendizaje continuo.
  - Desarrollo de una actitud profesional emprendedora.

Se espera que el estudiante logre:

Comprensión del requerimiento. Diseñar el algoritmo que resuelva el requerimiento. Implementar el algoritmo. Establecer las condiciones de borde para su funcionamiento. Establecer los lotes de pruebas y sus correspondientes salidas esperadas.

Descripción de la Actividad:

1- Tiempo estimado de resolución: 7 días.

2- Metodología: Planteo de requerimiento. Comprensión de requisitos y diseño de solución. Diseño de casos de prueba. Determinación de la salida esperada para cada caso de prueba. Codificación de la solución en lenguaje C. Prueba y validación de resultados.

3- Forma de entrega: se indicará qué ejercicios deberán ser resueltos y entregados sobre la plataforma MIeL.

4- Metodología de corrección y feedback al alumno: serán planteadas posibles estrategias de resolución. Serán evacuadas dudas puntuales respecto a la interpretación del ejercicio a resolver y su resolución.

Bibliografía: obligatoria sugerida por la cátedra (ver al inicio de este documento).

Ejercicio 1

El factorial de un número natural incluido el 0, se calcula de la siguiente manera:

$$N! = \begin{cases} 1 & \text{si } N = 0 \\ N.\ (N-1)! & \text{si } N > 0 \end{cases}$$

o sea, N! = N. (N - 1). (N - 2). …. 3. 2. 1

Ejemplo: 5! = 5. 4. 3. 2. 1 = 120

Desarrollar una función para calcular el factorial de un entero.

Ejercicio 2

Dados dos números enteros m y n (m >= n y n >= 0), el número combinatorio se calcula de la siguiente manera:

$$\binom{m}{n} = \frac{m!}{n!\,(m-n)!}$$

Desarrollar una función para calcular el combinatorio m sobre n.

Ejercicio 3

Dado un número entero X y una tolerancia (TOL), puede obtenerse ex mediante la suma de los términos de la serie:

$$e^x = 1 + \left(\frac{x^1}{1}\right) + \left(\frac{x^2}{1\cdot 2}\right) + \left(\frac{x^3}{1\cdot 2\cdot 3}\right) + \left(\frac{x^4}{1\cdot 2\cdot 3\cdot 4}\right) + \ldots$$

El proceso termina cuando se obtiene un término calculado que sea menor que la tolerancia TOL.

Desarrollar una función para calcular el ex , dados X y TOL.

Ejercicio 4

La raíz cuadrada de un número positivo A puede calcularse mediante un proceso iterativo que genera términos según la siguiente fórmula:

$$R_1 = 1$$

$$R_i = \frac{1}{2}\cdot\left(R_{i-1} + \left(\frac{A}{R_{i-1}}\right)\right)$$

El proceso de cálculo se da por terminado cuando la diferencia entre dos términos sucesivos es menor que una cota fijada de antemano.

Desarrollar una función para calcular la raíz cuadrada de X con una tolerancia TOL.

Ejercicio 5

En la serie de Fibonacci, cada término es la suma de los dos anteriores y los dos primeros términos son 1

Serie: 1 1 2 3 5 8 13 21 34 …

Desarrollar una función para determinar si un entero pertenece a la serie de Fibonacci.

Ejercicio 6

Dados X y una tolerancia TOL es posible calcular el seno (x) mediante la suma de los términos de la serie:

$$seno(x) = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \frac{x^9}{9!} - \frac{x^{11}}{11!} + \ldots$$

Este proceso continúa mientras el término calculado (en valor absoluto) sea mayor que la tolerancia. Desarrollar una función que obtenga el seno de X con tolerancia TOL, utilizando dicha serie.

Ejercicio 7

Un número natural es perfecto, deficiente o abundante según que la suma de sus divisores positivos menores que él sea igual, menor o mayor que él. Por ejemplo:

Número	Divisores positivos menores que él	Suma de los divisores	Clasificación
6	1, 2, 3	6	PERFECTO
10	1, 2, 5	8	DEFICIENTE
12	1, 2, 3, 4, 6	16	ABUNDANTE

Desarrollar una función que determine si un número natural es perfecto, deficiente o abundante.

Ejercicio 8

Dados dos números naturales (incluido el cero), obtener su producto por sumas sucesivas.

Ejercicio 9

Dados dos números naturales A y B, desarrollar una función para obtener el cociente entero A/B y el resto. (A puede ser 0; B, no).

Ejercicio 10

Construir un programa que lea un número natural N y calcule la suma de los primeros N números naturales.

Ejercicio 11

Construir un programa que lea un número natural N y calcule la suma de los primeros N números pares.

Ejercicio 12

Construir un programa que lea un número natural N y calcule la suma de los números pares menores que N.

Ejercicio 13

Desarrollar una función que determine si un número natural es primo.

Tema: Arrays unidimensionales. Concepto de puntero. Aritmética de Punteros. Manejo de arrays unidimensionales con diferentes notaciones. Arrays numéricos. Cadenas. Funciones de biblioteca. Arrays multidimensionales. Manejo de arrays multidimensionales con diferentes notaciones. Uso de arrays.

Unidad 1: Arrays.

Objetivo: Comprender los conceptos de array y puntero. Reconocer cuándo deben ser utilizados en función del requerimiento planteado.

Competencia/s a desarrollar:

Genéricas

- **Competencias tecnológicas**
  - Identificación, formulación y resolución de problemas de ingeniería en sistemas de información/informática.
  - Concepción, diseño y desarrollo de proyectos de ingeniería en sistemas de información / informática.
  -
  - Gestión, planificación, ejecución y control de proyectos de ingeniería en sistemas de información / informática.
  - Utilización de técnicas y herramientas de aplicación en la ingeniería en sistemas de información / informática.
- **Competencias sociales, políticas y actitudinales**
  - Desempeño en equipos de trabajo.
  - Comunicación efectiva.
  - Actuación profesional ética y responsable.
  - Aprendizaje continuo.
  - Desarrollo de una actitud profesional emprendedora.

Se espera que el estudiante logre:

Declarar arrays unidimensionales y multidimensionales. Declarar punteros a diferentes tipos de datos. Operar con arrays. Operar con punteros. Establecer las condiciones de borde para la no invasión de memoria. Comprender y utilizar aritmética de punteros. Operar con cadenas de texto.

Descripción de la Actividad:

1- Tiempo estimado de resolución: 3 semanas.

2- Metodología: Planteo de requerimiento. Comprensión de requisitos y diseño de solución. Diseño de casos de prueba. Determinación de la salida esperada para cada caso de prueba. Codificación de la solución en lenguaje C. Prueba y validación de resultados.

3- Forma de entrega: se indicará qué ejercicios deberán ser resueltos y entregados sobre la plataforma MIeL.

4- Metodología de corrección y feedback al alumno: serán planteadas posibles estrategias de resolución. Serán evacuadas dudas puntuales respecto a la interpretación del ejercicio a resolver y su resolución.

Bibliografía: obligatoria sugerida por la cátedra (ver al inicio de este documento).

Ejercicios con vectores (Arreglos unidimensionales).

Ejercicio 1.1
Desarrollar una función que inserte un elemento en un arreglo de enteros, dada la posición de inserción.

Ejercicio 1.2
Desarrollar una función que inserte un elemento en un arreglo de enteros, ordenado en forma ascendente, de forma de no alterar el orden.

Ejercicio 1.3
Desarrollar una función que elimine el elemento que ocupa una determinada posición de un arreglo de enteros.

Ejercicio 1.4
Desarrollar una función que elimine la primera aparición de un elemento determinado de un arreglo de enteros.

Ejercicio 1.5
Desarrollar una función que elimine todas las apariciones de un determinado elemento de un arreglo de enteros.

Ejercicio 1.6
Desarrollar una función que determine si una cadena de caracteres es un palíndromo.

Ejercicio 1.7
Desarrollar una función que devuelva el valor numérico de una cadena de caracteres (asumiendo que los caracteres representan dígitos).

Ejercicio 1.8
Desarrollar una función que cuente el número de apariciones de una palabra dentro de una cadena de texto. Para ello la función recibe como parámetros dos punteros a char. El primero corresponde al texto, el segundo corresponde a la cadena buscada. La función debe retornar el número de ocurrencias. Contemple las condiciones de borde y haga un listado de éstas.

Ejercicio 1.9
Desarrollar una función que normalice la cadena de texto que se le pasa como argumento. En este caso, la cadena se encontrará normalizada cuando la primera letra de cada palabra sea mayúscula y las siguientes minúsculas. La cadena normalizada no deberá contener espacios o tabulaciones al inicio o al final. En el caso de que las palabras de la cadena se encuentren separadas por más de un espacio o tabulación, se deberán eliminar los excedentes. Se debe modificar la cadena pasada como argumento. No puede realizar una o más copias locales de la cadena original. Contemple las condiciones de borde y haga un listado de éstas.

Ejercicio 1.10

La siguiente línea **“Nj qemh v ljs kraenkqbres; lj oqe qemh es oqevorme sgn ellhs --Istqt Asdmgj”** ha sido ofuscada para impedir su lectura desplazando desplazado hacia atrás dentro del grupo **"abcdghijkoqtuv"** tantos caracteres como posición tiene en la palabra.

Ejemplo, si la palabra es “hola”

- "h":un lugar atrás porque es la primera letra de la palabra, queda "g"
- "o":dos lugares atrás, queda "j"
- "l":No figura en grupo, no se modifica, queda "l"
- "a":cuatro lugares hacia atrás, queda "q", antes de la “a” está la “v”.

Quedando una vez ofuscada como **“gjlq”.**

- No considere ningún carácter que no figura dentro del grupo
- Una palabra es todo conjunto de uno o mas caracteres que responden a la función **isalpha**
- La frase desofuscada esta en Español.

Ejercicio 1.11

Ingresar por teclado pares de cadenas de caracteres, finalizando el ingreso cuando ambas cadenas sean iguales (las que no deben procesarse). Para cada par, cargar en un array bidimensional, ambas cadenas, respetando cargar primero la más pequeña y luego la mayor, si las longitudes fueran iguales, el orden lo dará la comparación lexicográfica haciendo caso omiso de mayúsculas y minúsculas.

Escriba una función que determine la comparación solicitada invocando a versiones propias de las funciones de biblioteca estándar strlen y strcmpi o strcasecmp (dado que esta no es una función estándar en algunos compiladores tiene otro nombre).

Ejercicios con matrices (Arreglos bidimensionales)

La definición adecuada de los parámetros de las siguientes funciones es parte de la ejercitación.

Ejercicio 1.12

Desarrollar una función para que, dada una matriz cuadrada de reales de orden N, obtenga la sumatoria de los elementos que están por encima de la diagonal principal (excluida ésta). Lo mismo para la diagonal secundaria. Lo mismo incluyendo la diagonal. Lo mismo, con los que están por debajo de la diagonal.

Ejercicio 1.13

Desarrollar una función para que, dada una matriz cuadrada de enteros de orden N, obtenga la traza de la misma (sumatoria de los elementos de la diagonal principal). Lo mismo, pero con la diagonal secundaria.

Ejercicio 1.14

Desarrollar una función que determine si una matriz cuadrada de enteros de orden N es matriz diagonal (ceros en todos sus elementos excepto en la diagonal principal).

Ejercicio 1.15
Desarrollar una función que determine si una matriz cuadrada de enteros de orden N es matriz identidad (matriz diagonal, con unos en la diagonal principal y ceros en los restantes).

Ejercicio 1.16
Desarrollar una función que determine si una matriz cuadrada de enteros de orden N es simétrica.

Ejercicio 1.17
Desarrollar una función para trasponer, in situ, una matriz cuadrada.

Ejercicio 1.18
Desarrollar una función para obtener la traspuesta de una matriz dada.

Ejercicio 1.19
Desarrollar una función para obtener la matriz producto entre dos matrices de enteros.

Ejercicio 1.20
Se dispone de una matriz cuadrada de enteros de orden **N**, donde cada elemento **[i][j]** representa la cantidad de puntos que obtuvo el equipo **i** frente al equipo **j** al fin de un torneo de fútbol (partidos de ida y vuelta) en el que participaron **N** equipos. El sistema de puntuación es: 3 puntos para el ganador del partido y ninguno para el perdedor; 1 punto para cada equipo en caso de empate.
Desarrollar una función que determine si la matriz está correctamente generada.
Desarrollar una función que genere un arreglo de **N** elementos tal que cada elemento **V[k]** contenga la cantidad de puntos obtenidos por el equipo **k**.

Ejercicios ampliados con vectores (Arreglos unidimensionales)

Ejercicio 1.21
Haga una reingeniería de los ejercicios 1.1 a 1.5 para que puedan operar con cualquier tipo de dato (recuerde que los ejercicios originales estaban diseñados para operar con números enteros). Haga uso de memoria estática.

Tema: Tipos de memoria. Memoria de pila. Memoria Heap. Manejo de memoria en tiempo de ejecución. Ventajas y desventajas de cada tipo de memoria.

Unidad 2: Tipos de Memorias.

Objetivo: Comprender los conceptos e implicancias de los diferentes tipos de memoria. Reconocer cuándo deben ser utilizados en función del requerimiento planteado.

Competencia/s a desarrollar:
Genéricas

- **Competencias tecnológicas**
  - Identificación, formulación y resolución de problemas de ingeniería en sistemas de información/informática.
  - Concepción, diseño y desarrollo de proyectos de ingeniería en sistemas de información / informática.
  - Gestión, planificación, ejecución y control de proyectos de ingeniería en sistemas de información / informática.
  - Utilización de técnicas y herramientas de aplicación en la ingeniería en sistemas de información / informática.
- **Competencias sociales, políticas y actitudinales**
  - Desempeño en equipos de trabajo.
  - Comunicación efectiva.
  - Actuación profesional ética y responsable.
  - Aprendizaje continuo.
  - Desarrollo de una actitud profesional emprendedora.

Se espera que el estudiante logre:
Gestionar de manera correcta el uso de memoria dinámica.

Descripción de la Actividad:

1- Tiempo estimado de resolución: 1 semana.

2- Metodología: Planteo de requerimiento. Comprensión de requisitos y diseño de solución. Diseño de casos de prueba. Determinación de la salida esperada para cada caso de prueba. Codificación de la solución en lenguaje C. Prueba y validación de resultados.

3- Forma de entrega: se indicará qué ejercicios deberán ser resueltos y entregados sobre la plataforma MIeL.

4- Metodología de corrección y feedback al alumno: serán planteadas posibles estrategias de resolución. Serán evacuadas dudas puntuales respecto a la interpretación del ejercicio a resolver y su resolución.

Bibliografía: obligatoria sugerida por la cátedra (ver al inicio de este documento).

Ejercicio 2.1
Desarrolle un programa que reserve memoria para almacenar 10 enteros. Luego asigne valores a dichos elementos. Asegúrese de devolver la memoria reservada.

Ejercicio 2.2
Modifique el programa del punto 2.1 para que la cantidad de elementos sea ingresada por teclado.

Ejercicio 2.3
Desarrolle un programa que reserve memoria para almacenar 5 elementos de un tipo de dato creado por Usted. Dicho tipo de dato debe contener al menos un array de char, un entero, un flotante y un char. Luego asigne valores a dichos elementos. Asegúrese de devolver la memoria reservada.

Ejercicio 2.4
Implemente utilizando memoria dinámica las siguientes funciones:

- Una función que retorna una copia de la cadena pasada como parámetro con exactamente la misma cantidad de caracteres de la cadena origen. Ej:

```
char* copiaCadena(const char* origen)
```

- Una función que genere una copia de un elemento pasado como parámetro, este puede ser un tipo primitivo (int,float,char) una estructura o incluso un vector. Ej:

```
void* copiaCosas(void* elemento, unsigned tam)
```

Estas funciones tienen un objetivo estrictamente didáctico, y como puede observar persiguen objetivos similares a las funciones de biblioteca de C pero su implementación es totalmente diferente. ¿Por qué no se recomienda este tipo de implementación en sistemas productivos?

Ejercicio 2.5
Implemente su propia versión de la función de biblioteca memmove, asegurando que los datos se mueven correctamente incluso si hay superposición entre ellos. Verifique que no tiene pérdidas de memoria (memory leaks) .

Tema: Concepto de pasaje de parámetros. Pasaje de parámetros por copia o valor. Pasaje de parámetros por dirección. Punteros a variables. Punteros a funciones. Funciones de biblioteca de ordenamiento y búsqueda. Creación de funciones independientes del tipo de dato. Concepto de funciones map, filter y reduce. Argumentos en el main.

Unidad 3: Pasaje de Parámetros.

Objetivo: Introducir los conceptos necesarios para dar paso a la programación genérica.

Competencia/s a desarrollar:
Genéricas

- **Competencias tecnológicas**
  - Identificación, formulación y resolución de problemas de ingeniería en sistemas de información/informática.
  - Concepción, diseño y desarrollo de proyectos de ingeniería en sistemas de información / informática.
  - Gestión, planificación, ejecución y control de proyectos de ingeniería en sistemas de información / informática.
  - Utilización de técnicas y herramientas de aplicación en la ingeniería en sistemas de información / informática.
- **Competencias sociales, políticas y actitudinales**
  - Desempeño en equipos de trabajo.
  - Comunicación efectiva.
  - Actuación profesional ética y responsable.
  - Aprendizaje continuo.
  - Desarrollo de una actitud profesional emprendedora.

Se espera que el estudiante logre:
Comprender y explotar los parámetros de una función. Cambiar el comportamiento de una función a través del pasaje de una función como parámetro de la misma. Conocer y explotar los mecanismos de la programación centrada en los algoritmos facilitando su uso de manera independiente de los tipos de datos. Explorar y conocer las principales funciones utilitarias de las bibliotecas stdlib.h y stdio.h. Comprender y utilizar macro reemplazos.

Descripción de la Actividad:
1- Tiempo estimado de resolución: 2 semanas.
2- Metodología: Planteo de requerimiento. Comprensión de requisitos y diseño de solución. Diseño de casos de prueba. Determinación de la salida esperada para cada caso de prueba. Codificación de la solución en lenguaje C. Prueba y validación de resultados.
3- Forma de entrega: se indicará qué ejercicios deberán ser resueltos y entregados sobre la plataforma MIeL.
4- Metodología de corrección y feedback al alumno: serán planteadas posibles estrategias de resolución. Serán evacuadas dudas puntuales respecto a la interpretación del ejercicio a resolver y su resolución.

Bibliografía: obligatoria sugerida por la cátedra (ver al inicio de este documento).

Ejercicio 3.1

Escriba una función que permita desplegar un menú de opciones, devolviendo una opción válida.

Escriba una función que reciba por argumento la dirección de comienzo de un array de float y la cantidad máxima de elementos a ingresar (no utilice subíndices). La función permitirá terminar el ingreso con una condición fijada por el alumno y devolverá la cantidad de elementos ingresados (puede ser cero).

Escriba una función que permita buscar el mínimo elemento de un array de float.

Escriba una función que determine el promedio de los elementos que se encuentran en las posiciones pares de un array de float.

Escriba una función que muestre los elementos de un array de float en orden inverso.

Escriba una función que almacene en un archivo de texto los elementos de un array de float, a razón de un flotante por línea de texto.

Haciendo uso de las funciones anteriores, escriba un programa que al comenzar su ejecución permita el ingreso para un array de float, luego de lo cual muestre un menú de opciones para:

1- Buscar el mínimo elemento,
2- Calcular el promedio de los valores de las posiciones pares,
3- Mostrarlo en orden inverso,
4- Salir.

Consulte de qué modo puede hacer que el programa trabaje con otros tipos de datos (double, long double, int, unsigned, etc.), con mínimas modificaciones.

Ejercicio 3.2

Escriba una función que devuelva en qué dirección de memoria se encuentra un elemento dentro de un array. Si el elemento no se encuentra, debe devolver NULL.

Ejercicio 3.3

Escriba una función que permita el ingreso de una cantidad variable de elementos en un array de enteros int.

Escriba una función que calcule la suma de todos los elementos almacenados en un array de enteros, y su promedio. El promedio (float) debe ser devuelto por la función, y la suma debe ser también devuelta mediante un argumento extra (puntero a long) que recibe la función.

Escriba otra versión de la función anterior, pero devolviendo ambos valores calculados en una variable que responda a una estructura compuesta de un miembro long y un miembro float.

a- escriba un main que utilice la primera y segunda función, y
b- otro main que utilice la primera y la tercera

En ambos casos, la suma y el promedio deben ser mostrados en la función main.

¿Tiene claro que la primera alternativa es mejor que la segunda porque no es necesario el uso de una estructura? En ciertos casos puede ser mejor la segunda alternativa.

Ejercicio 3.4

Existen funciones de conversión (atoi, itoa, atol, etc., declaradas en la biblioteca stdlib.h), que usted debería conocer y recordar. Escriba, compile y ejecute un programa en que haga uso de tales funciones de conversión.

Ejercicio 3.5
En la biblioteca stdio.h hay dos funciones que permiten obtener idénticos (o similares) resultados, se trata de sscanf y sprintf. Escriba, compile y ejecute un programa en que utilice estas funciones.

Ejercicio 3.6
Escriba una macro que:

- redondee un número real al entero más cercano.
- calcule el valor absoluto de un argumento
- retorne la parte entera de un número
- retorne la parte decimal
- verifique si el número es par
- reciba dos argumentos (x,y) verifique si el número “x” es potencia del número “y”
- diga si el argumento es letra
- es dígito
- es mayúscula
- es minúscula
- es blanco
- convierta un caracter a mayúscula
- convierta un caracter a minúscula

Escriba una función que devuelva el menor entre dos enteros que recibe por argumento.
Escriba una macro que cumpla con el mismo cometido.

Ejercicio 3.7
Investigue y utilice las macros max y min de la biblioteca stdlib.h.

Ejercicio 3.8
Escriba una función que intercambie dos enteros que recibe por puntero.
Escriba una macro multilínea que cumpla con el mismo cometido.

Ejercicio 3.9
Dado un arreglo enteros, utilice la función qsort (biblioteca <stdlib>) para ordenarlos de menor a mayor. Repita el ejercicio ordenándolos de mayor a menor.
Para los siguientes ejercicios de ordenamiento o búsqueda, asuma la existencia de:

```
typedef struct
{
      int dni;
      char apellido[20];
      char nombres[30];
      float peso;

} tPersona;
```

Ejercicio 3.10
Dado un arreglo de elementos de tipo tPersona, utilice la función qsort (biblioteca <stdlib>) para ordenarlos por DNI.

Ejercicio 3.11
Dado el arreglo de elementos de tipo tPersona del ejercicio 3.10, utilice la función qsort (biblioteca <stdlib>) para ordenarlos por APELLIDO y NOMBRE.

Ejercicio 3.12
Dado el arreglo de elementos de tipo tPersona del ejercicio 3.10, diseñe e implemente la función:

```
int buscarXdni(const tPersona *p, tPersona *d);
```

El propósito de la función es realizar una búsqueda por DNI de una persona dentro de un arreglo de elementos del tipo persona. En caso de encontrarse el DNI contenido en el campo dni del segundo argumento, se deben completar los demás campos con los valores correspondientes al elemento localizado en el arreglo y además la función debe retornar 1. En el caso de que la búsqueda no haya sido exitosa se debe retornar 0.

Ejercicio 3.13
Idem ejercicio 3.12 buscando por APELLIDO y NOMBRE. Adecúe el nombre de la función respetando la firma y el tipo de valor devuelto.

Ejercicio 3.14
Diseñe e implemente un programa que reciba tres valores desde la línea de comando. Los primeros dos valores deberán ser números enteros. El tercer valor será el tipo de operación (+: sumar, -: restar, *: multiplicar, /: dividir). El programa deberá imprimir el resultado de la operación. Establezca las condiciones de borde. En caso de que se produzcan errores deben informarse por pantalla.

Ejercicio 3.15
Desarrolle una función de intercambio genérica tal que pueda intercambiar 2 bloques de información de manera independiente al tipo de dato. No utilice memoria dinámica.

Ejercicio 3.16
Desarrolle una función genérica que encuentre el menor elemento dentro de un vector. Verifique su funcionamiento encontrando el menor entero, el menor flotante, y el menor alumno (Por DNI, nombre y apellido o promedio). La función retornará la dirección del elemento menor

Ejercicio 3.17
Desarrolle una función genérica que permita insertar de forma ordenada un elemento en un vector. Puede ayudarse, usando como patrón, la versión no genérica desarrollada y probada de la práctica 1 y con las funciones de biblioteca memcpy y memmove.

Ejercicio 3.18

Ordenamiento genérico - Desarrolle una función genérica que ordene un vector. Puede ayudarse con las funciones intercambio y buscar menor desarrolladas en ejercicios anteriores e implementar un algoritmo de selección.

Tema: Concepto de Tipo de Dato Abstracto. Creación e implementación de un TDA. Diferencia entre TDA y estructuras de datos.

Unidad 4: Tipo de Dato Abstracto (TDA)

Objetivo: Comprender el concepto y uso de un TDA.

Competencia/s a desarrollar:
Genéricas
- **Competencias tecnológicas**
  - Identificación, formulación y resolución de problemas de ingeniería en sistemas de información/informática.
  - Concepción, diseño y desarrollo de proyectos de ingeniería en sistemas de información / informática.
  - Gestión, planificación, ejecución y control de proyectos de ingeniería en sistemas de información / informática.
  - Utilización de técnicas y herramientas de aplicación en la ingeniería en sistemas de información / informática.
- **Competencias sociales, políticas y actitudinales**
  - Desempeño en equipos de trabajo.
  - Comunicación efectiva.
  - Actuación profesional ética y responsable.
  - Aprendizaje continuo.
  - Desarrollo de una actitud profesional emprendedora.

Se espera que el estudiante logre:
Análisis, diseño e implementación de un TDA acorde a las especificaciones funcionales provistas.

Descripción de la Actividad:

1- Tiempo estimado de resolución: 1 semana.

2- Metodología: Planteo de requerimiento. Comprensión de requisitos y diseño de solución. Diseño de casos de prueba. Determinación de la salida esperada para cada caso de prueba. Codificación de la solución en lenguaje C. Prueba y validación de resultados.

3- Forma de entrega: se indicará qué ejercicios deberán ser resueltos y entregados sobre la plataforma MIeL.

4- Metodología de corrección y feedback al alumno: serán planteadas posibles estrategias de resolución. Serán evacuadas dudas puntuales respecto a la interpretación del ejercicio a resolver y su resolución.

Bibliografía: obligatoria sugerida por la cátedra (ver al inicio de este documento).

Para los siguientes ejercicios con fechas, asuma la existencia de:

```
typedef struct
{
      int dia;
      int mes;
      int anio;
} tFecha;
```

Ejercicio 4.1
Desarrollar una función que determine si una fecha es formalmente correcta.

Ejercicio 4.2
Desarrollar una función que a partir de una fecha obtenga la correspondiente al día siguiente.
Ejercicio 4.3
Desarrollar una función que a partir de una fecha obtenga la que resulte de sumarle N días.

Ejercicio 4.4
Desarrollar una función que a partir de una fecha obtenga la que resulte de restarle N días.

Ejercicio 4.5
Desarrollar una función que a partir de dos fechas obtenga la cantidad de días que hay entre ellas.

Ejercicio 4.6
Desarrollar una función que a partir de una fecha devuelva un entero que representa el día de la semana que le corresponde (0: Domingo; 1: lunes; 2:Martes;…etc.)

Ejercicio 4.7
Implemente un TDA Vector. Debe desarrollar una versión con memoria estática y otra con memoria dinámica. Debe implementar las primitivas:

- crear vector
- vector lleno
- vector vacío
- insertar elemento en orden
- eliminar elemento
- destruir vector

Tenga en cuenta que la primitiva ‘destruir’ debe, según sea el caso, liberar la memoria reservada, o poner la cantidad de elementos en cero.

Tema: Archivos binarios y de texto, su creación, modos de acceso, posicionamiento, cierre, eliminación, y funciones relacionadas. Creación de archivos como lotes de prueba. Merge o apareo de archivos.

Unidad 5: Persistencia de memoria

Objetivo: Comprender el uso de archivos binarios y de texto para la persistencia de datos haciendo uso de las funciones proporcionadas por las bibliotecas de entrada/salida.

Competencia/s a desarrollar:

Genéricas

- **Competencias tecnológicas**
  - Identificación, formulación y resolución de problemas de ingeniería en sistemas de información/informática.
  - Concepción, diseño y desarrollo de proyectos de ingeniería en sistemas de información / informática.
  - Gestión, planificación, ejecución y control de proyectos de ingeniería en sistemas de información / informática.
  - Utilización de técnicas y herramientas de aplicación en la ingeniería en sistemas de información / informática.
- **Competencias sociales, políticas y actitudinales**
  - Desempeño en equipos de trabajo.
  - Comunicación efectiva.
  - Actuación profesional ética y responsable.
  - Aprendizaje continuo.
  - Desarrollo de una actitud profesional emprendedora.

Se espera que el estudiante logre:

Crear y eliminar archivos binarios y de texto. Realizar operaciones de apertura y cierre de archivos binarios y de texto. Realizar lecturas y escrituras sobre archivos binarios y de texto. Realizar actualizaciones sobre archivos binarios.

Descripción de la Actividad:

1- Tiempo estimado de resolución: 3 semanas.

2- Metodología: Planteo de requerimiento. Comprensión de requisitos y diseño de solución. Diseño de casos de prueba. Determinación de la salida esperada para cada caso de prueba. Codificación de la solución en lenguaje C. Prueba y validación de resultados.

3- Forma de entrega: se indicará qué ejercicios deberán ser resueltos y entregados sobre la plataforma MIeL.

4- Metodología de corrección y feedback al alumno: serán planteadas posibles estrategias de resolución. Serán evacuadas dudas puntuales respecto a la interpretación del ejercicio a resolver y su resolución.

Bibliografía: obligatoria sugerida por la cátedra (ver al inicio de este documento).

Ejercicio 5.1

Escriba una función booleana que permita abrir un archivo, mostrando o no un mensaje de error por stdout, según el valor de un argumento.

Escriba una macro multilínea que cumpla con el mismo cometido.

Ejercicio 5.2

Dado un array de char que contiene un texto compuesto por palabras que termina en '.' (o en su defecto en carácter nulo -'\0'-), escriba un programa en que determine e informe:

a- cuál es la primera palabra y cuántas veces se repite en el texto

b- cuántas palabras tiene el texto

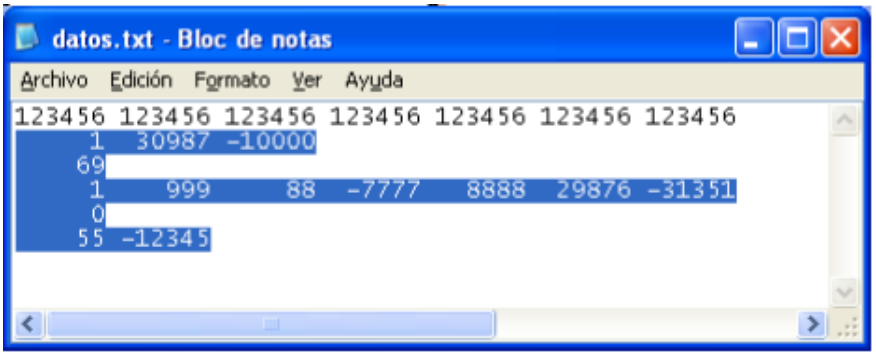
c- longitud de la palabra más larga

El siguiente es un ejercicio compuesto por los ejercicios 5.3, 5.4 y 5.5. Debería resolverlo en forma progresiva.

Ejercicio 5.3

Escriba un programa que genere un archivo de texto ("datos.txt") a partir del ingreso por teclado de números enteros, de modo que en cada línea de texto haya una cantidad variable de cadenas de caracteres que representen tales números. En el archivo debe haber como mínimo la representación de un entero y como máximo de siete. La separación entre estas cadenas que representan números debe ser de al menos un carácter espacio (' ') o a lo sumo cinco, de modo que queden alineados por la derecha al leer el contenido del archivo con un procesador de texto como el 'Notepad' o 'Bloc de Notas'.

Ingrese los enteros con una variable short int (note que el rango de las mismas pertenece al intervalo [-32768, 32767]). Vea e interprete qué sucede cuando ingresa números fuera de ese rango.



Note que:

- los caracteres de separación sólo están entre números.
- la primera línea se ha indicado para visualizar la alineación de los números.
- en el archivo debe haber al menos una línea con un número y otra con siete.

Utilice el generador de números pseudo aleatorios para determinar cuántos números se almacenan por línea de texto, y además para que determine cuántas líneas se almacenarán una vez cumplida la condición de una línea con un número y otra con siete.

Ejercicio 5.4
Escriba una función que determine si una cadena de caracteres que representa a un número, es decir compuesta por los caracteres que representan dígitos, recibida por argumento:

- es capicúa (es obvio),
- es múltiplo de 5 (el último dígito es 0 ó 5),
- es múltiplo de 6 (es par -termina en '0', '2', '4', '6', '8'; y la suma de sus dígitos es múltiplo de 3),
- es múltiplo de 11 (la suma de los dígitos de posiciones pares, y la suma de los dígitos de posiciones impares es múltiplo de 11),
- es mayor que '100' (o cualquier otra cadena representando un número entero cualquiera, p. ej.: '-42').

Escriba una función que valide si todos los caracteres de una cadena representan un número dentro del intervalo de los: short int.

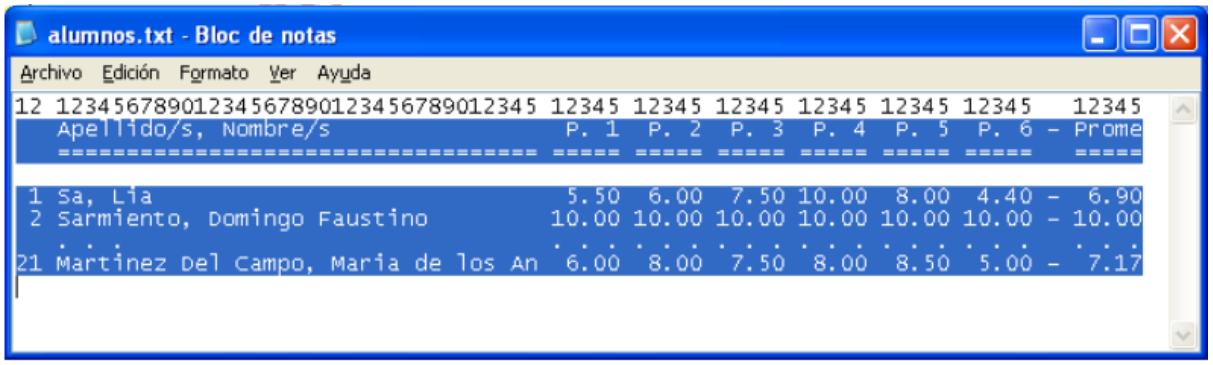
Ejercicio 5.5
Leyendo (sólo una vez) un archivo de texto como el del <ej.: 5.3> y utilizando las funciones del <ej.: 5.4>, y otras al efecto, determinar:

- cuántos son múltiplo de 5,
- cuántos son múltiplo de 6,
- cuántos son múltiplo de 11, y
- generar un archivo con los que sean mayores que '100' (o cualquier otro número recibido por argumento en la línea de comando).

Ejercicio 5.6
Escriba un programa que le permita ingresar en arrays bidimensionales los apellidos y nombres de alumnos y las seis calificaciones obtenidas en cada parcial. Haga uso de una función que resuelva el ingreso, y devuelva cuántos se cargaron. Se garantiza que la cantidad de alumnos es menor que 100.
El array para las calificaciones tendrá una fila y una columna extra, en las que se calculará, haciendo uso de funciones al efecto:

- el promedio de calificaciones de cada alumno, que se acumula en la columna extra que le corresponde a cada alumno. Esta función debe ser invocada repetidamente con la dirección de cada fila del array.
- el promedio general de los alumnos para cada evaluación, y el promedio general.

Al terminar el programa debe generar una salida impresa (en archivo de texto), que debería verse así:



Note que:

- debe numerar las líneas de detalle,
- debe colocar un título indicativo, subrayarlo, y dejando un renglón en blanco (vacío), comenzar el listado de alumnos (hasta 21 por página),
- la numeración es consecutiva en las distintas páginas,
- los campos impresos deben quedar encolumnados,
- los promedios deben informarse en la última hoja,
- la primera línea se ha indicado para visualizar la alineación de los campos.

Ejercicio 5.7

Se dispone de dos archivos binarios: <empleados> y <estudiantes>.

Cada registro del primer archivo contiene los campos:

- <dni>, <apellido>, <nombre> y <sueldo>

en tanto que los del segundo:

- <dni>, <apellido>, <nombre> y <promedio>.

Ambos archivos están ordenados alfabéticamente por <apellido> / <nombre> / <dni>.

Ambos archivos deben ser leídos sólo una vez, y no deben ser almacenados en arrays. El sueldo es double y el promedio es float.

Escriba un programa que, leyendo ambos archivos, actualice el sueldo de aquellos empleados que tengan un registro de estudiante con un promedio superior a 7, en un 7,28%.

Ejercicio 5.8

Escriba un programa que genere un archivo de texto de varias líneas, y en cada línea una o varias palabras (se considera palabra, cualquier carácter que no responde a lo indicado por la función de biblioteca isspace, ni es coma, ni punto, ni ... (use su criterio). La separación entre palabras puede ser de uno o varios de estos caracteres. La primera palabra en una línea de texto puede estar precedida por más de uno de estos caracteres de separación, y lo mismo puede suceder después de la última palabra.

Escriba otro programa que agregue nuevas líneas de texto al archivo creado por el programa.

Escriba un programa que haciendo uso de su propia versión de strstr, busque en todo el archivo la subcadena recibida en la línea de comando (cualquiera sea esta), e informe en qué línea y en qué posición dentro de la línea la encuentra. Puede no encontrarla, encontrarla una única vez, o varias veces en la misma o distintas líneas del archivo.

Ejercicio 5.9
Leyendo el texto del archivo del ejercicio anterior, informar la cantidad de palabras que cumplen con cada una de las siguientes condiciones:

- están formadas por una sola letra,
- están formadas por una cantidad par de letras,
- comienzan con 'n',
- comienzan con un prefijo (o subcadena) determinado ingresado por el operador,
- tienen más de tres vocales,
- comienzan y terminan con vocales,
- contienen dígitos,
- sólo están formadas por dígitos,
- son palíndromos.

Ejercicio 5.10
Resuelva el <ej.: 5.7> con archivos de texto con campos de longitud fija.
Ídem, pero con archivos de texto con campos de longitud variable.
Note que deberá generar un nuevo archivo auxiliar de empleados con otro nombre, para al terminar eliminar (unlink) el archivo original y renombrar (rename) el auxiliar.

Ejercicio 5.11
Ingrese hasta un máximo de 1000 caracteres (o hasta que se ingrese el carácter '.'. Mostrar por pantalla el carácter inmediato posterior
(p. ej.: "a bgxj" -> "b!chyk"; "Zapato" -> "[bqbup"). Grabar en un archivo aquellas palabras con más de una determinada cantidad de letras que se deben pedir al operador al comienzo del proceso.

Ejercicio 5.12
Escriba una función:

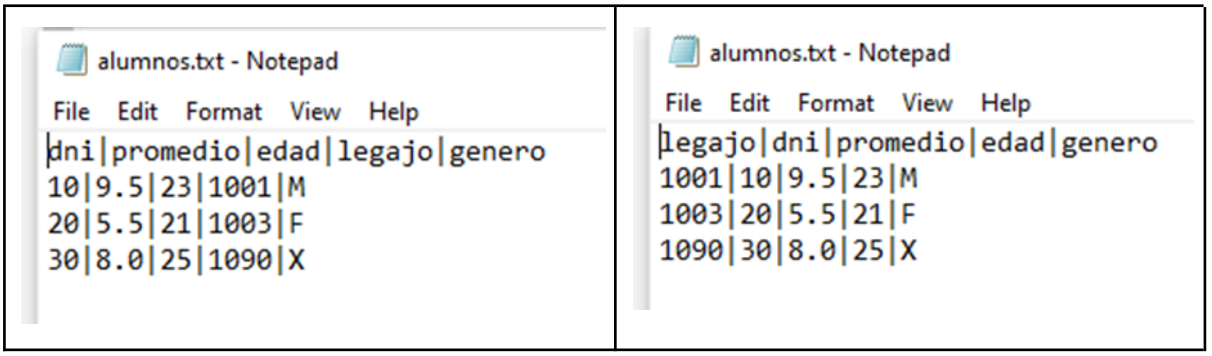
- que le permita mostrar el contenido de un archivo binario de enteros cuyo nombre recibe por argumento. Cada registro se mostrará separado del siguiente por un espacio en blanco.
- que le permita mostrar el contenido de un archivo de texto cuyo nombre recibe por argumento. Cada registro se mostrará separado del siguiente por un espacio en blanco.
- que genere un archivo binario (<"archalea">) con 90 números al azar de tres cifras (función estándar rand de la biblioteca stdlib.h).
- que divida en tres nuevos archivos binarios (<"archal1">, <"archal2"> y <"archal3">) el archivo anterior, tomado los primeros 30 para el primer archivo, los siguientes 30 para el segundo y los restantes para el tercero.
- que transforme en archivo de texto cada uno de los archivos del punto anterior (<"archal1.txt">, <"archal2.txt"> y <"archal3.txt">).
- que transforme los dígitos de cada número de los archivos del punto anterior en letras, generando los archivos de texto (<"archtex1.txt">, <"archtex2.txt"> y <"archtex3.txt">) de modo que a los dígitos del '0' al '9' les correspondan los caracteres de la 'A' a la 'J' para el primero, de la 'K' a la 'T' para el segundo y de la 'U' a la 'D' para el tercero. O sea:

		los dígitos se transforman										Ejemplos		
	Conversión	0	1	2	3	4	5	6	7	8	9	157	901	975
Para el archivo	1ro	A	B	C	D	E	F	G	H	I	J	BFH	JAB	JHF
	2do	K	L	M	N	O	P	Q	R	S	T	LPB	TKL	TRP
	3ro	U	V	W	X	Y	Z	A	B	C	D	VZB	DUV	DBZ

- que lea los archivos del punto anterior, y haciendo uso de la función <qsort> los ordene en forma creciente (únicamente aquí es necesario leer todo el archivo en un array). Se generan los archivos (<"archord1.txt">, <"archord2.txt"> y <"archord3.txt">).
- que genere un único archivo ordenado a partir de los archivos del punto anterior, manteniendo el ordenamiento ascendente. Se genera el archivo (<"archord.txt">).

Escriba un programa que haciendo uso de cada una de las funciones (desde la tercera hasta la última), genere un archivo binario con 90 números al azar, lo separe en tres archivos, genere tres archivos de texto, etc.. Utilice las dos primeras funciones para visualizar el avance del proceso. Antes de terminar el proceso, utilice una función de menú para consultar si se desean eliminar los archivos intermedios.
Resuelva las funciones extra necesarias para una mejor estructuración del programa.

Ejercicio 5.13
Se tiene un archivo de texto con los campos "legajo, dni, promedio, edad, genero", todos los campos son numéricos y no hay blancos entre ellos. La primera línea indica el orden de los campos en el archivo de texto y este puede cambiar entre distintos archivos, todas las combinaciones son posibles. Ej:



Desarrolle una función que genere un archivo binario de registros de la estructura tAlumno que se muestra al pie a partir de cualquier orden de campos en los archivos de texto. Utilice todas las técnicas aprendidas hasta la fecha para evitar poner una condición para cada orden de campos posible, recuerde que la cantidad de combinaciones es el factorial de la cantidad de campos, volviendo inconveniente esta estrategia

```c
typedef struct{
    int dni;
    int legajo;
    float promedio;
    int edad;
    char genero;
}tAlumno;
```

Tenga presente que debe usar punteros a función para la resolución del ejercicio.

Tema: Concepto de recursión. Estructura de funciones recursivas. Diferencias entre recursividad e Iteración.

Unidad 6: Introducción a la Recursividad

Objetivo: Comprender y utilizar algoritmos y funciones recursivas. Determinar si el requerimiento soporta una solución basada en recursión.

Competencia/s a desarrollar:
Genéricas

- **Competencias tecnológicas**
  - Identificación, formulación y resolución de problemas de ingeniería en sistemas de información/informática.
  - Concepción, diseño y desarrollo de proyectos de ingeniería en sistemas de información / informática.
  - Gestión, planificación, ejecución y control de proyectos de ingeniería en sistemas de información / informática.
  - Utilización de técnicas y herramientas de aplicación en la ingeniería en sistemas de información / informática.
- **Competencias sociales, políticas y actitudinales**
  - Desempeño en equipos de trabajo.
  - Comunicación efectiva.
  - Actuación profesional ética y responsable.
  - Aprendizaje continuo.
  - Desarrollo de una actitud profesional emprendedora.

Se espera que el estudiante logre:
Comprender la estructura de una función recursiva. Comprender la necesidad de la condición de corte de la recursividad. Detectar la condición de corte de la recursividad en el requerimiento. Interpretar la evolución de la pila de llamadas. Diseñar, implementar y probar funciones recursivas.

Descripción de la Actividad:
1- Tiempo estimado de resolución: 1 semana.
2- Metodología: Planteo de requerimiento. Comprensión de requisitos y diseño de solución. Diseño de casos de prueba. Determinación de la salida esperada para cada caso de prueba. Codificación de la solución en lenguaje C. Prueba y validación de resultados.
3- Forma de entrega: se indicará qué ejercicios deberán ser resueltos y entregados sobre la plataforma MIeL.
4- Metodología de corrección y feedback al alumno: serán planteadas posibles estrategias de resolución. Serán evacuadas dudas puntuales respecto a la interpretación del ejercicio a resolver y su resolución.

Bibliografía: obligatoria sugerida por la cátedra (ver al inicio de este documento).

Ejercicio 6.1

Escriba una función recursiva que:

- calcule el factorial de un número entero.
- muestre el contenido de un array de char.
- ídem anterior, mostrando en orden inverso.
- ídem anterior, devolviendo la suma de los caracteres que representan dígitos.
- muestre el contenido de un array de enteros en orden inverso, devolviendo la suma de todos los elementos.
- ídem anterior, devolviendo la suma de los pares.
- Ídem anterior, devolviendo la suma de los que están en posiciones pares.
- Escriba versiones recursivas de las funciones de biblioteca <strlen>, <strchr> y <strrchr>.

Ejercicio 6.2

El triángulo de Tartaglia es una forma sencilla de calcular los coeficientes de la potencia de un binomio. Valiéndose sólo de un array de enteros <unsigned short>, muestre por pantalla los primeros 19 juegos de coeficientes (potencias cero a 18).

Potencia	Coeficientes								
0	1								
1	1	1							
2	1	2	1						
3	1	3	3	1					
4	1	4	6	4	1				
5	1	5	10	10	5	1			
6	1	6	15	20	15	6	1		
7	1	7	21	35	35	21	7	1	
8	1	8	28	56	70	56	28	8	1
…	…								
18	…								

Compruebe que el siguiente juego de coeficientes (el que corresponde a la potencia 19) excederá el rango de almacenamiento de un entero corto. Diseñe nuevamente su cálculo para que se detenga cuando se dé esta condición, y no muestre los coeficientes en que se produce el problema. En los siguientes ejemplos falta omitir los coeficientes uno y las potencias cero.

$$(a+b)^0 = 1.a^0.b^0$$
$$(a+b)^1 = 1.a^1.b^0 + 1.a^0.b^1$$
$$(a+b)^3 = 1.a^3.b^0 + 3.a^2.b^1 + 3.a^1.b^2 + 1.a^0.b^3$$
$$(a+b)^6 = 1.a^6.b^0 + 6.a^5.b^1 + 15.a^4.b^2 + 20.a^3.b^3 + 15.a^2.b^4 + 6.a^1.b^5 + 1.a^0.b^6$$

Para ver mejor la tabla genere el archivo <tartaglia.txt>.