

**UNLaM**

Dto. Ingeniería e Investigaciones Tecnológicas

Carrera:	Ingeniería en Informática
Materia:	plan 09 (1110) Programación
Año Lectivo :	2017
Contenido :	Reglamento Interno de la Cátedra Programa Bibliografía Cronograma Guía de Trabajos Prácticos
Docentes :	Ghigo, Paola Guatelli, Renata Sotelo, Lorena Cacho Mendoza, Ariel Conde, Sergio Fungueriño, Adrián Guillén, Gabriel Guzmán, Gabriel Martínez, Pablo Pan, Néstor Pezzola, Federico Soligo, Pablo Urán Acevedo, Jónatan

**Jefe de Cátedra**      López, Luis

Reglamento, Programa, Bibliografía, Cronograma y Guía de Trabajos Prácticos, basados en el material de cátedra dejado por la Profesora Nélida Matas, anterior jefa de la cátedra.

Trabajo Práctico 1 elaborado por el Profesor Guillermo Módica, siempre presente con nosotros.

## **Reglamento Interno**

### **Organización de la Cátedra**

La cátedra se compone de un Jefe de Cátedra cuya tarea es acompañada por un grupo de Docentes. Este grupo se compone por docentes a cargo del curso, que dictan la teoría de la materia, y otros Docentes que secundan al Docente a Cargo en la complementación de los contenidos de la materia y el dictado de los trabajos prácticos (T. P.), tanto en aula como en laboratorio, control de asistencia y cumplimiento por parte de los alumnos de los T. P..

### **Régimen de cursada**

Se requiere una asistencia no inferior al 75%, en caso de no cumplirse, el alumno queda en condición de ausente.

Además, a fin lograr la internalización de los contenidos de la asignatura, debe cumplir con la realización de los T. P. pautados para la materia y los que adicionalmente o en su reemplazo fijen sus docentes, y la aprobación de los mismos. En el caso de no cumplimiento de sus T. P. no podrá rendir sus evaluaciones parciales hasta que se regularice esta situación.

Una vez cumplido lo anterior, el alumno estará habilitado a rendir sus exámenes parciales.

Estos exámenes parciales reciben la denominación de primer parcial (P. 1) y segundo parcial (P. 2), que se evaluarán en las fechas indicadas (ver cronograma). Además habrá una única instancia de recuperación de parciales es decir recuperatorio único (R.). Las calificaciones obtenidas en instancia de recuperatorio reemplazan a la obtenida en el parcial que se recupera. En el recuperatorio sólo se aplica a los contenidos de un solo parcial.

Un parcial o recuperatorio estará reprobado si la calificación es inferior a 4 (cuatro). Aquellos alumnos que logren 7 (siete) o más puntos en ambos parciales (o su correspondiente recuperatorio), accederán al régimen de promoción por lo que no deberán rendir examen final, teniendo aprobada la materia con el promedio resultante.

El R. siempre será sobre temas del P. 1, ó del P. 2, no de ambos.

Si el P. 1 y el P. 2 tuvieran una calificación inferior a 4 (cuatro) o ausente, en ambos no habrá chance de recuperación. De lo contrario, si el P. 1 ó el P. 2 tuviera una calificación inferior a 4 (cuatro) o ausente, el R. será sobre temas del Parcial no aprobado.

Se pueden utilizar ambas instancias de parcial y la instancia de recuperatorio.

El alumno que sume dos aplazos queda en condición final de reprobado.

El alumno que logre promocionar la materia, es decir con 7 (siete) o más puntos en ambos parciales o su recuperatorio, queda en condición final de aprobado.

El alumno que tenga 4 (cuatro) o más puntos en ambos parciales o su recuperatorio y no logra promocionar, queda en condición de que ha cursado la materia (o desaprobado en la promoción), y deberá rendir examen final, en las fechas dispuestas al efecto.

El alumno que en P. 1 y en P. 2 tenga dos aplazos (menos de cuatro), quedará en condición de reprobado, no pudiendo presentarse a recuperar. Si al recuperar uno de sus parciales obtiene un aplazo, quedará en condición de reprobado (independientemente de que esté recuperando un parcial para promocionar o para quedar regular). En caso de tener sólo un aplazo en P. 1 ó en P 2, y no se presente a recuperar quedará en situación de ausente, independientemente de la calificación de aprobado con cuatro o más puntos del otro parcial.

Las evaluaciones serán corregidos por los docentes del curso y las calificaciones serán comunicadas el siguiente día de clase (siete días después). Este plazo puede prolongarse en el caso del P. 1.

La fecha del P. 1 puede ser cambiada por los docentes del curso comunicándolo a los alumnos con la correspondiente antelación.

La condición académica del alumno deberá quedar cerrada al último día de clases en que cursa la materia salvo disposiciones del Departamento de Ingeniería o de la Universidad.

### **Exámenes finales para alumnos regulares**

El alumno debe estar inscripto en el acta de examen que se recibe del Departamento de Alumnos a través del Departamento de Ingeniería, con lo cual se garantiza que haya cumplido con las condiciones de cursada de la materia y el régimen de correlatividades vigente. Para poder rendir el examen deberá concurrir con su libreta universitaria y/o documento que acredite fehacientemente su identidad.

El tema de examen será preparado por la cátedra y se tomará un único tema a todos los alumnos que se presenten en la misma fecha y podrá ser realizado tanto en laboratorio como en forma escrita, o una combinación de ambos.

Los exámenes serán corregidos por cualquiera de los docentes de la cátedra, y a solicitud del alumno podrán ser revisados por los docentes y en última instancia por el Jefe de Cátedra o quien lo reemplace por ausencia, ejerciendo la función de Presidente de Mesa. Si después de esto el alumno no considerara satisfecho su reclamo, deberá presentar una nota dirigida al Jefe de Cátedra, con lo que el alumno deberá seguir los pasos administrativos que correspondan ante el D.I.I.T..

### **Exámenes finales para alumnos libres**

El alumno que se presente a rendir examen libre, podrá presentarse en las fechas habilitadas por la Universidad y/o el Departamento de Ingeniería.

Con anticipación a la fecha del final (se sugiere un mínimo de tres semanas) deberá comunicarse con el Jefe de Cátedra para acordar un día en que se le encargará un trabajo que deberá presentar una semana antes de la fecha del examen. Este tra-

bajo práctico incluirá la resolución de un ejercicio con `problemáticas similares a los ejercicios de la guía de trabajos prácticos de la materia. Tendrá la característica de un Trabajo Práctico Integrador. Este trabajo deberá ser presentado, defendido y aprobado. Además se hará una evaluación práctica en laboratorio, sin lo cual no podrá pasar a la instancia siguiente. El examen final se tomará el mismo día en horario a acordar con modalidad similar a los alumnos regulares. Este examen no es el mismo con que se evalúa a los alumnos regulares, dado que se requiere que demuestre conocimiento de todas las unidades del programa en sus aspectos teóricos y prácticos.

### **Cambios de Comisión**

Los alumnos al inscribirse para cursar sus materias cumplen un acto administrativo ante el Departamento de Alumnos, con lo cual quedan registrados en qué día y horario cursan, y en base a esta información se generan las Actas de Cursada. Ante motivos justificados para un cambio de comisión deberán realizar el trámite administrativo que corresponda. Una vez hecho lo anterior, deberán presentar el comprobante a todos los docentes involucrados en el cambio de horario o turno. Bajo ningún concepto se aceptarán cambios que no hayan cumplido con estos requisitos.

El docente que acepte un cambio irregular será, junto con el alumno que lo solicitó, responsable de las consecuencias que posteriormente pudiera tener el alumno.

### **Disposiciones Generales**

Las clases perdidas por cualquier circunstancia, hechos del hombre o la naturaleza (incluso feriados), deberán ser recuperadas lo antes posible, según disponibilidad horaria del docente con aviso a los alumnos.

Todos los docentes de la cátedra deben concurrir a las mesas de examen final, de modo acorde a lo dispuesto por las disposiciones de la Universidad y el Departamento. Caso contrario, deberán presentar justificativo ante el Departamento de Ingeniería.

La modalidad para todas las evaluaciones será en el laboratorio de computadoras de la Universidad y las correcciones se harán en el mismo a la mayor brevedad posible. Por lo tanto, el código escrito por el estudiante deberá compilar y ejecutar sin errores en pos de la solución requerida. En caso de no disponer de laboratorios con capacidad suficiente, se podrá hacer la evaluación en forma escrita. Si un examen estuviera escrito en lápiz o la escritura no está clara, no se admitirá ningún tipo de reclamo o revisión sobre el mismo. El alumno que no estuviera conforme con la calificación obtenida en sus parciales o recuperatorios, deberá en primer lugar plantearlo a sus docentes; de no quedar satisfecho con la revisión, deberá entregarle una nota al docente a cargo del curso, la que será elevada al Jefe de Cátedra junto con la evaluación cuestionada. Si la respuesta del mismo no lo satisface, deberá ratificarlo nuevamente por escrito, con lo cual se formará una comisión integrada por docentes de otras comisiones de la cátedra los que resolverán el reclamo. Si con esto aún no queda satisfecho, podrá continuar con el mismo por los canales pertinentes. En el caso de exámenes finales, la solicitud de revisión de nota es en el momento que se comunica la calificación

obtenida. No se aceptarán reclamos posteriores al cierre del acta de examen, debiendo ,en caso de disconformidad, presentar una nota al presidente de mesa y continuar 'per se' el reclamo de no resultar satisfecho.

## **Programa**

### **Unidad 1 Arrays y archivos**

Introducción a la abstracción de datos y ocultamiento de información. Funciones de búsqueda, conversión, ordenamiento, etc.. Manejo de arrays con notación y aritmética de punteros. Recursividad. Funciones de las bibliotecas string.h, ctype.h, math.h, stdlib.h, stdio.h, etc.. Arrays bidimensionales. Arrays de punteros. Punteros a funciones. Funciones de biblioteca qsort, bsearch y lsearch. Archivos binarios y de texto, su creación, modos de acceso, posicionamiento, cierre, eliminación, y funciones relacionadas. Creación de tipos de datos, uniones, enumeraciones, macroreemplazos. Generación de arrays en tiempo de ejecución. Argumentos variables.

### **Unidad 2 Estructura de datos Pila**

Asignación dinámica de memoria vs. asignación estática de memoria. Primitivas para el manejo de Pilas, compatibilidad de primitivas entre la implementación estática y dinámica de Pilas. Su relación con la recursividad.

### **Unidad 3 Estructura de datos Cola**

Asignación dinámica de memoria vs. asignación estática de memoria. Primitivas para el manejo de Colas, compatibilidad entre la implementación estática y dinámica de Colas. Su relación con el "buffer" de teclado.

### **Unidad 4 Estructura de datos Lista**

Primitivas para el manejo de Listas. Creación, inserción, ordenamiento, búsqueda, eliminación, etc., con asignación dinámica de memoria.

Listas circulares, su importancia en la implementación de colas y su parecido y diferencia con la implementación de pilas.

Listas doblemente enlazadas.

### **Unidad 5 Estructura de datos Árbol**

Árbol binario, creación, recorridas EnOrden, PreOrden y PosOrden. Árbol binario de búsqueda, su relación con la búsqueda binaria en arrays. Determinación de altura, y otras funciones.

Árboles AVL y balanceados, determinación.

### **Unidad 6 Introducción a C++ – Programación Orientada a Objetos (POO–OOP)**

Modificaciones menores con respecto al Lenguaje C. Comentarios, punteros constantes, sobrecarga de funciones, parámetros por defecto, parámetros por alias (referencia). Asignación y liberación dinámica de memoria, operadores new y delete. Flujos de entrada/salida, manipuladores.

Programación Estructurada frente a la POO. Arquitectura y evolución del diseño de software. Clases, objetos, métodos y mensajes. Lenguajes que soportan la POO y su evolución.

### **Unidad 7 Programación Orientada a Objetos con C++**

Clases, objetos, métodos de la clase (funciones miembro), funciones amigas. Especificación inline y const. Constructores y destructores,

su ejecución automática. Puntero this. Sobrecarga de operadores. Arrays, punteros y objetos.

**Unidad 8 Herencia**

Conceptos de herencia, y su por qué. Introducción a herencia simple y herencia múltiple. Clases derivadas. Diagramas de Jerarquía. Control de acceso a la clase base. Especificador protected.

**Unidad 9 Excepciones**

Manejo de excepciones, throw, catch, try. Especificación de excepciones. Funciones estándar assert, terminate y unexpected.

**Unidad 10 Introducción al Lenguaje Java.**

Demostración del uso de un Entorno de Desarrollo Integrado para la generación de aplicaciones de consola e interactivas. Tipos de datos operadores, etc..

**Bibliografía****Básica**

El Lenguaje de Programación C – Kernighan y Ritchie – Prentice Hall  
El Lenguaje de Programación C++ – Bjarne Stroustrup – Addison Wesley  
Cómo Programar en C/C++ – Deitel y Deitel – Prentice Hall  
C/C++, Manual de Referencia – Herbert Schildt – Mc Graw Hill  
Apuntes y bibliografía a entregar por la cátedra.

**Complementaria**

Cómo Programar en C/C++/Java – Deitel y Deitel – Prentice Hall  
Data Structures and Program Design in C – Kruse, Leung y Tondo – Prentice Hall  
Thinking in C++ – Bruce Eckell – Prentice Hall  
Estructuras de Datos con C y C++ – Langsam, Augenstein y Tenenbaum – Prentice Hall  
Programación Orientada a objetos – Luis Joyanes Aguilar – Mc Graw Hill

**IMPORTANTE:**

Los alumnos de la materia (1110) Programación (plan 09), cursan además un Trabajo de Calificación Profesional, implementado en esta materia como un Taller de Programación del que habrá oferta para hacerlo en diversas plataformas (Delphi, .Net y Java), contando los alumnos con la correspondiente Guía de Trabajos Prácticos y actividades de cada uno de ellos.

**Cronograma** (La primera fecha corresponde a los cursos del primer cuatrimestre y la segunda a los del segundo cuatrimestre, y se refieren a los lunes correspondientes a esa semana).

Clase semana	Actividad
Clase 1 27/03 14/08	<p><b>Aula :</b></p> <p>Presentación de los docentes del curso, breve explicación de las pautas generales de la materia a discutir más exhaustivamente en el siguiente día de clase, una vez que los alumnos hayan leído el presente documento.</p> <p>Breve repaso de conceptos adquiridos en el curso anterior de Programación I, e introducción de nuevos conceptos.</p> <p>La función main, funciones e identificadores, palabras reservadas. Tipos de datos y tipos de variables propios del Lenguaje C. Constantes y constantes literales. Macroreemplazos. Tipos de datos, typedef y struct. Tipos de datos enum. Uniones. Operadores, precedencia de operadores. Control de flujo de ejecución : if, switch, while, for y do while, break; continue y goto. Punteros a una variable, a un array, arrays de punteros, argumentos de main, argumentos de funciones, por valor, por puntero, punteros a estructuras, punteros a arrays de estructuras,</p>
Clase 2 27/03 14/08	<p><b>Aula :</b></p> <p>Terminación de la clase anterior, se concluye con el repaso e introducción de nuevos conceptos.</p> <p>Se evacuarán las consultas derivadas de la lectura por parte de los alumnos del reglamento de la cátedra y del régimen de cursada y promoción.</p> <p><b>Práctica :</b></p> <p>Edición, compilación, depuración y ejecución de programas, pautas de trabajo en laboratorio.</p> <p>T. P. 1, resolución de algoritmos con temas de revisión. Conformación de los grupos de trabajo.</p>
Clase 3 03//04 21/08	<p><b>Aula :</b></p> <p>Ejemplificación del uso de punteros y arrays. Recursividad. Arrays de caracteres. Uso de punteros y aritmética de punteros en lugar del uso de índices para arrays unidimensionales. Funciones de conversión y búsqueda. unión, estructura y tipos de datos.</p> <p><b>Práctica :</b></p> <p>Atención de consultas del T. P. 1.</p>
Clase 4 03//04 21/08	<p><b>Aula :</b></p> <p>Ejemplificación del uso de arrays bidimensionales. Arrays de punteros. Punteros a funciones. y aritmética de punteros en lugar del uso de índices para arrays unidimensionales. Funciones de conversión y búsqueda. unión, estructura y tipos de datos.</p>



**Práctica :**

Evaluación T. P. 1. Introducción del T. P. 2.

Clase 5  
10/04  
28/08

**Aula :**

Archivos binarios y de texto, modos de apertura. Archivos de texto con campos de longitud fija y variable. Ejemplificación.

**Práctica :**

Consultas del T. P. 2.

Clase 6  
10/04  
28/08

**Aula :**

Archivos binarios y de texto. Convertir información de archivos de texto en binarios y viceversa. Ejemplificación. Actualización masiva de archivos binarios. Búsqueda de un registro por su clave o por su posición relativa dentro del archivo y actualización del mismo.

**Práctica :**

Cierre del T. P. 2.

Clase 7  
17/04  
04/09

**Aula :**

Gestión de memoria dinámica, funciones específicas (malloc, calloc, realloc y free). Ejemplificación. Implementación de estructuras dinámicas de datos. Introducción conceptual a estructuras de datos, ejemplos con la estructura Pila y mención de las estructuras Cola y Lista. La Pila con implementación estática vs. dinámica. Importancia del uso de primitivas coherentes entre una implementación y la otra.

**Práctica :**

Evaluación del T. P. 2. Presentación del T. P. 3

Clase 8  
17/04  
04/09

**Aula :**

Implementación estática y dinámica de Pilas. Ejemplificación de primitivas coherentes entre una implementación y la otra.

**Práctica :**

Aplicaciones de Pilas, ejemplos sobre la práctica.

Clase 9  
24/04  
11/09

**Aula :**

Implementación estática y dinámica de Pilas. Ejemplificación de primitivas coherentes entre una implementación y la otra.

**Práctica :**

Aplicaciones de Pilas, ejemplos sobre la práctica.

Clase 10  
24/04  
11/09

**Aula :**

Implementación estática y dinámica de Colas. Ejemplificación de primitivas coherentes entre una implementación y la otra.

**Práctica :**

Aplicaciones de Colas, ejemplos sobre la práctica.

Clase 11  
01/05

**Aula :**

18/09	Implementación estática y dinámica de Colas. Ejemplificación de primitivas coherentes entre una implementación y la otra.
	<b>Práctica :</b>
	Aplicaciones de Colas, ejemplos sobre la práctica.
Clase 12	<b>Aula :</b>
01/05	
18/09	Implementación dinámica de Listas. Ejemplificación de primitivas. Mención de su implementación estática.
	<b>Práctica :</b>
	Aplicaciones de Listas, ejemplos sobre la práctica.
Clase 13	<b>Aula :</b>
08/05	
25/09	Implementación dinámica de Listas. Ejemplificación de primitivas.
	<b>Práctica :</b>
	Aplicaciones de Listas, ejemplos sobre la práctica.
Clase 14	<b>Aula :</b>
08/05	
25/09	Breve presentación conceptual de la estructura de datos Árbol. Evacuación de consultas y dudas, comunicar a los alumnos el día y horario tentativo de la clase de consulta.
	<b>Práctica :</b>
	Evaluación T. P. 3, Pilas, Colas y Listas.
Clase 15	<b>Aula :</b>
15/05	
02/10	Estructura de datos Árbol. Primitivas. Recorridas. Mención de su compatibilidad con la implementación estática. Evacuación de consultas y dudas, confirmar a los alumnos el día y horario de la clase de consulta.
	<b>Práctica :</b>
	Ejercitación a demanda de los alumnos.
Clase 16	<b>Primer parcial :</b>
15/05	
02/10	<b>NOTA :</b>
	En cada comisión se podrá optar por adelantar o postergar una clase esta primer evaluación parcial, de acuerdo con la apreciación que hagan los docentes del avance de los alumnos en los objetivos académicos de la materia.
	Esta evaluación abarcará todos los temas dados con exclusión de Árbol, Listas doblemente enlazadas y Listas circulares, que se evaluará en el segundo parcial.
Clase 17	<b>Aula :</b>
22/05	
09/10	Árbol binario y árbol binario de búsqueda. Mención de árboles semibalanceados (AVL) y balanceados, similitud con búsquedas dicotómicas. Presentación de funciones variadas (búsqueda de la clave de ordenamiento, altura, contar hojas, contar no hojas, contar nodos que cumplen una determinada condición, eliminar árbol, 'podar' ramas, etc.).

**Práctica :**

Entrega y revisión de calificaciones del primer parcial. Presentación del T. P. 4

Clase 18  
22/05  
09/10

**Aula :**

Finalización del tratamiento de árboles binarios. Explicación conceptual de listas doblemente enlazadas (estas primitivas serán implementadas por los alumnos). Explicación conceptual de listas circulares, justificación e importancia para la implementación de colas dinámicas en listas circulares, similitud y diferencia con la implementación de pilas en listas circulares.

**Práctica :**

Ejercitación a demanda de los alumnos (T.P. 4).

Clase 19  
29/05  
16/10

**Aula :**

Introducción a la Programación Orientada a Objetos. Su importancia en el diseño de sistemas y la programación. Diferencias con la programación modular convencional. Conceptos de Clases, atributos, métodos y mensajes; privacidad de los atributos. Herencia, diagramas de herencia. El Lenguaje C++ como lenguaje orientado a objetos. Modificaciones menores en C++ respecto a C : comentarios, especificador const, sobrecarga de funciones, parámetros por defecto, referencias, operadores new y delete, flujos de entrada/salida, manipuladores.

**Práctica :**

Evaluación del T.P. 4. Presentación del T.P. 5.

Clase 20  
29/05  
16/10

**Aula :**

Continuación de la clase anterior. Explicación de clases sencillas y ejemplificación de sus atributos miembro y métodos, constructores, otros métodos de la clase, necesidad del destructor, miembros de información static, funciones inline, puntero this.

**Práctica :**

Entrega y revisión de calificaciones del primer recuperatorio.  
Ejercitación a demanda de los alumnos (T.P. 5).

Clase 21  
05/06  
23/10

**Aula :**

Continuación de la clase anterior. Métodos y sobrecarga de operadores, funciones amigas.

**Práctica :**

Ejercitación a demanda de los alumnos (T.P. 5).

Clase 22  
05/06  
23/10

**Aula :**

Sobrecarga de operadores, casos diversos.

**Práctica :**

Ejercitación a demanda de los alumnos (T.P. 5).

Clase 23  
12/06

**Aula :**

30/10	Conceptos de herencia, simple y múltiple. El por qué y la necesidad de la herencia. Ejemplos con herencia simple.
	<b>Práctica :</b>
	Ejercitación a demanda de los alumnos (T.P. 5).
Clase 24	<b>Aula :</b>
12/06	
30/10	Continuación clase anterior. Ejemplos con herencia simple. Especificador protected.
	<b>Práctica :</b>
	Ejercitación a demanda de los alumnos (T.P. 5).
Clase 25	<b>Aula :</b>
19/06	
06/11	Herencia múltiple. Ejemplos con herencia múltiple.
	<b>Práctica :</b>
	Ejercitación a demanda de los alumnos (T.P. 5).
Clase 26	<b>Aula :</b>
19/06	
06/11	Herencia múltiple. Polimorfismo. funciones virtuales. Clases abstractas. Manejo de excepciones: try, throw y catch.
	<b>Práctica :</b>
	Ejercitación a demanda de los alumnos (T.P. 5).
Clase 27	<b>Aula :</b>
26/06	
13/11	Consultas y problemas diversos.
	<b>Práctica :</b>
	Evaluación T.P. 5.
Clase 28	<b>Aula :</b>
26/06	
13/11	Segunda evaluación parcial.
	<b>Práctica :</b>
	Consultas a demanda de los alumnos.
Clase 29	<b>Aula :</b>
03/07	
20/11	Introducción al lenguaje Java. Programa 'Hola Mundo' a modo consola y a modo ventana interactiva utilizando los recursos Swing.
	Cálculo de factorial, combinatorio, productoria de un entero como métodos de una clase.
	<b>Práctica :</b>
	Resolución de ejercicios sobre lo visto.
	Entrega de calificaciones, revisión, atención de consultas.
Clase 30	<b>Aula :</b>
03/07	
20/11	Profundización en el lenguaje, creación de clases Factorial, Combinatorio y Productoria.
	Su uso.
	Ejemplos varios.

**Práctica :**

Resolución de ejercicios sobre lo visto.

Clase 31  
10/07  
27/11**Aula :**

Recuperatorio.

**Práctica :**

Consultas a demanda de los alumnos.

Clase 32  
10/07  
27/11**Aula :**Entrega de calificaciones, revisión, atención de consultas.  
Firma de libretas y exhibición del acta de cursada.**Inicio 1er Cuatrimestre: 27/03 -****Fin 1er Cuatrimestre: 15/07 -****Inicio 2do. Cuatrimestre: 14/08****Fin 2do Cuatrimestre: 02/12**

## Guía de Trabajos Prácticos

### Trabajo Práctico 1

#### Ejercicio 1

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

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

o sea,  $N! = N \cdot (N - 1) \cdot (N - 2) \cdot \dots \cdot 3 \cdot 2 \cdot 1$

Ejemplo:  $5! = 5 \cdot 4 \cdot 3 \cdot 2 \cdot 1 = 120$

Desarrollar una función para calcular el factorial de un entero.

#### Ejercicio 2

Dados dos números enteros m y n ( $m \geq n$  y  $n \geq 0$ ), 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  $e^x$  mediante la suma de los términos de la serie :

$$e^x = 1 + \left( X^1 / 1 \right) + \left( X^2 / (1 * 2) \right) + \left( X^3 / (1 * 2 * 3) \right) + \left( X^4 / (1 * 2 * 3 * 4) \right) + \dots$$

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

Desarrollar una función para calcular el  $e^x$ , 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 :

$$R_1 = 1$$

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

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:

$$\text{seno}(x) = x - x^3 / 3! + x^5 / 5! - x^7 / 7! + x^9 / 9! - x^{11} / 11! + \dots$$

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.

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

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

**Ejercicio 14**

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

**Ejercicio 15**

Desarrollar una función que a partir de una fecha obtenga la correspondiente al día siguiente.

**Ejercicio 16**

Desarrollar una función que a partir de una fecha obtenga la que resulte de sumarle N días.

**Ejercicio 17**

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

**Ejercicio 18**

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

**Ejercicio 19**

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 20**

El método de multiplicación rusa de dos números naturales, consiste en lo siguiente:

Se divide sucesivamente por 2 (división entera) a uno de sus factores hasta obtener 1.

Paralelamente, se multiplica sucesivamente por 2 al otro factor.

La suma de éstos últimos números obtenidos que se correspondan con números impares obtenidos en las divisiones, es el producto buscado (Se consideran los factores originales para la suma correspondiente).

Ejemplo : 35 x 8

35	8	8, 16 y 256 se corresponden con impares (35, 17 y 1). $8 + 16 + 256 = 280$
17	16	
8	32	
4	64	
2	128	
1	256	

**Ejercicio 21**

Desarrollar una función para obtener la parte entera de un número real.

**Ejercicios con vectores (Arreglos unidimensionales).****Ejercicio 22**

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

**Ejercicio 23**

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

**Ejercicio 24**

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

**Ejercicio 25**

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

**Ejercicio 26**

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

**Ejercicio 27**

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

**Ejercicio 28**

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

**Ejercicios con matrices (Arreglos bidimensionales)**

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

**Ejercicio 29**

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 30**

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 31**

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 32**

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 33**

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

**Ejercicio 34**

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

**Ejercicio 35**

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

**Ejercicio 36**

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

**Ejercicio 37**

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  $k$ .

## **Trabajo Práctico 2**

### **Ejercicio 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.

y que antes de terminar la ejecución del programa grabe los elementos del array en un archivo de texto.

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 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**

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 primer 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 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 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 6**

Escriba una macro que redondee un número real al entero más cercano.

Escriba una función que devuelva el menor entre dos enteros que recibe por argumento.

Escriba una macro que cumpla con el mismo cometido.

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

Escriba una función que intercambie dos enteros que recibe por puntero.

Escriba una macro multilínea que cumpla con el mismo cometido.

#### **Ejercicio 7**

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 8**

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 primer 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 ejercicio 9, 10 y 11**

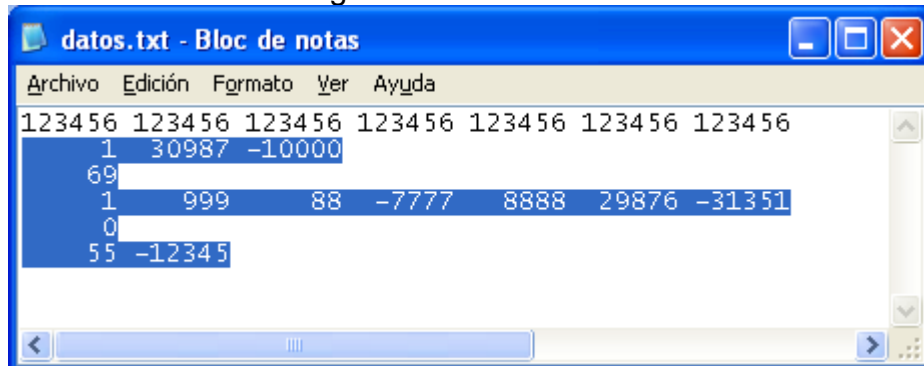
Debería resolverlo en forma progresiva.

#### **Ejercicio 9**

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 me-

nos 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'.

Ingresa 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 primer 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 10

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 representa un número dentro del intervalo de los `short int`.

### Ejercicio 11

Leyendo (sólo una vez) un archivo de texto como el del <ej. : 9> y utilizando las funciones del <ej. : 10>, 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 12**

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.

alumnos.txt - Bloc de notas

Archivo Edición Formato Ver Ayuda

12	12345678901234567890123456789012345	12345	12345	12345	12345	12345	12345	12345	12345	12345
	Apellido/s, Nombre/s	P. 1	P. 2	P. 3	P. 4	P. 5	P. 6	-	Prome	
	=====	=====	=====	=====	=====	=====	=====	=====	=====	
1	Sa, Lía	5.50	6.00	7.50	10.00	8.00	4.40	-	6.90	
2	Sarmiento, Domingo Faustino	10.00	10.00	10.00	10.00	10.00	10.00	-	10.00	
21	Martínez Del Campo, Maria de los An	6.00	8.00	7.50	8.00	8.50	5.00	-	7.17	

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 13**

Se dispone de dos archivos binarios : <"empleados"> y <"estudiantes">. Cada registro del primer archivo contiene los campos :

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

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 14**

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 mas 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 `strcmp` o `strcasecmp` (dado que esta no es una función estándar en algunos compiladores tiene otro nombre).

**Ejercicio 15**

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 primer 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 16**

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 17**

Resuelva el <ej. : 13> 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 18**

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" -> "[bqbpup]"). 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 19**

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 :

archivoPara el	conversión	los dígitos se transforman										ejemplos		
		0	1	2	3	4	5	6	7	8	9	157	901	975
		A	B	C	D	E	F	G	H	I	J	BFH	JAB	JHF
		K	L	M	N	O	P	Q	R	S	T	LPB	TKL	TRP
		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 (unicamente 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 20**

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 21**

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 de 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">`.

## **Trabajo Práctico 3**

### **Ejercicio 1**

Acorde con lo visto en clases, desarrolle la implementación estática de **Pila**, creando sus archivos "**pila.h**" y "**pila.c**", deje estos archivos en un subdirectorio "**./estatica/**". Para que resulte de uso más general el tipo de dato para la información declárelo en otro archivo junto con los prototipos de las funciones que permiten ingresar, mostrar, etc. esa información.

### **Ejercicio 2**

Ídem para la implementación dinámica de **Pila**, creando sus archivos "**pila.h**" y "**pila.c**", en un subdirectorio "**./dinamica/**". Para la información, utilice lo mismo del punto anterior.

### **Ejercicio 3**

La información de los puntos anteriores deberá constar de un código de producto, descripción, proveedor (alfanuméricos de 7, 15 y 15 respectivamente), fechas de compra y de vencimiento, cantidad, precios de compra y de venta. Los desarrollos de las funciones para el manejo de la información deben estar en sus propios archivos fuente.

### **Ejercicio 4**

Escriba un programa que al comenzar lea (si lo puede abrir) un archivo de binario <"**datos**">, y lo cargue en una pila con implementación estática de memoria. El archivo debe ser cerrado al terminar la carga de la pila.

A continuación, y valiéndose de una función de menú, que permita cargar más información en la pila, ver la información del tope de la pila, sacar de la pila, salir del menú. Al salir del menú, se terminará de cargar el archivo con la información que aún quede en la pila.

Si el archivo resultara vacío, deberá ser eliminado.

Pruebe repetidamente el programa, hasta que logre que el archivo quede con información.

### **Ejercicio 5**

Ídem anterior pero con implementación dinámica de memoria.

### **Ejercicio 6**

Ídem [Ejercicio 1] para la estructura **Cola**.

### **Ejercicio 7**

Ídem [Ejercicio 2] para la estructura **Cola**.

### **Ejercicio 8**

Ídem [Ejercicio 4] pero cargando en una **Cola** con implementación estática.

### **Ejercicio 9**

Ídem [Ejercicio 4] pero cargando en una **Cola** con implementación dinámica.

### **Ejercicio 10**

Resuelva el cálculo de la suma de dos números enteros de muchos dígitos (30 o muchos más) haciendo uso de dos **pilas** en las que almacena sólo los dígitos. Tenga en cuenta que debe utilizar una tercera **pila** en la que irá cargando los resultados parciales.

Compruebe que obtiene idénticos resultados con ambas implementaciones de **Pila** (estática y dinámica)

### **Ejercicio 11**

Resuelva la simulación de la cola de espera en un cajero automático. Suponga que cada cliente demora en el mismo un tiempo aleatorio de 1, 3 ó 5 minutos con igual probabilidad, y que los clientes llegan al mismo de a uno, con intervalo de arribo aleatorio de 1, 5 ó 9 minutos, con igual probabilidad.

La simulación termina después que la cola queda vacía cinco veces.

### **Ejercicio 12**

Se dispone de un archivo como el generado en el [Ejercicio 4]. Se desea generar dos nuevos archivos : en <"**datos1**"> con los registros cuya clave comience o termine con un carácter representativo de un dígito, pero en orden contrario al del archivo original, valiéndose de una **Pila**; y en <"**datos2**"> en el mismo orden en que estaban grabados.

El proceso debe ser resuelto con una única lectura del archivo de entrada, mostrando por pantalla cada registro leído, para luego a la vez que se genera cada archivo de salida, mostrar qué se graba en cada uno.

### **Ejercicio 13**

De acuerdo con lo visto en clases, desarrolle la implementación dinámica de **Lista**, creando sus archivos "**lista.h**" y "**lista.c**", deje estos archivos en un subdirectorio "**./dinamica/**". Para que resulte de uso más general el tipo de dato para la información declárelo en otro archivo junto con los prototipos de las funciones que permiten ingresar, mostrar, etc. esa información.

### **Ejercicio 15**

Utilizando una lista, ordene el archivo <"**datos1**"> valiéndose de una inserción en orden, y el archivo <"**datos2**"> valiéndose de una función que inserte al comienzo de la lista para luego ordenar la lista. Los archivos deben resultar ordenados por fecha de vencimiento, y a igualdad del mismo por proveedor y clave.

### **Ejercicio 16**

Implemente un programa que le permita cargar el archivo <"datos"> en una Lista dinámica. En la información a cargar no se debe incluir el nombre del proveedor.

Pruebe las siguientes alternativas :

a- Insertar los nuevos nodos al final de la lista, salvo que la clave ya estuviera cargada y la descripción coincide, con lo cual se acumula la cantidad, se retiene la última fecha de compra y la fecha de vencimiento mas vieja, el mayor precio de compra y de venta; si la descripción no coincidiera, se genera un nuevo nodo. Eliminar **todos** los nodos cuya clave está más de una vez, mostrando su información por pantalla y grabándolos en un archivo de texto <"errores2"> (tienen distinta descripción). Ordenar la lista resultante, y luego grabar esta información en un nuevo archivo binario <"depurado">.

b- Insertar los nuevos nodos ordenados por clave y a igualdad de clave por descripción, salvo que esta clave compuesta coincida con lo que se procede a acumular igual que antes. Eliminar **todos** los nodos cuya clave está más de una vez, mostrando su información por pantalla y grabándolos en un archivo de texto <"errores2"> (tienen distinta descripción). Grabar esta información en un nuevo archivo <"depurado2">.

### Ejercicio 17

Implemente y pruebe las siguientes funciones de listas en versiones para listas ordenadas/no-ordenadas :

Función iterativa (**buscar\_cla**) que busque por una clave en una lista y recupere la información del nodo con esa clave, devolviendo un indicador de éxito/fracaso en su cometido, y eliminando o no el nodo encontrado, según lo indique un argumento extra.

Ídem, recursiva.

Función iterativa (**contar\_cla**) que busque y devuelva la cantidad de veces que encontró la clave, y eliminando o no los nodos encontrados, según lo indique un argumento extra.

Ídem, recursiva.

Función iterativa (**buscar\_cla\_n**) que busque por una clave en una lista y recupere la información de la ocurrencia n de la clave indicada.

Ídem, recursiva.

### Ejercicio 18

Dado un archivo binario (tal como el <"datos"> antes indicado), proceda a ordenarlo valiéndose de dos pilas.

### Ejercicio 19

Genere una matriz poco densa de FIL filas por COL columnas (con muchos ceros). A partir de la información cargada en la matriz, genere una lista simplemente enlazada con miembros de información para la fila, columna y valor (sólo si este es distinto de cero). Valiéndose de un menú que permita :

a- el ingreso de fila y columna e informe el valor correspondiente, buscándolo en la lista.

b- el ingreso de una fila, y muestre los elementos de la fila.

c- el ingreso de una columna, y muestre los elementos de la columna.

d- que muestre a partir de la lista los elementos de la matriz.

Tenga en cuenta para los puntos [a-], [b-] y [d-] que debe disponer de la lista ordenada de modo que la búsqueda sea óptima.

## Ejercicio 20

Se dispone de un archivo de texto con el número y el nombre de cada agrupación que se presenta en las elecciones para elegir los congresales de una asociación civil sin fines de lucro; y además de un archivo binario en el que se almacenó el número de agrupación, de distrito y de región (un registro por cada voto electrónico emitido).

Se requiere un proceso que a partir de la lectura de ambos archivos almacene en arrays bidimensionales los nombres de las agrupaciones y el total de votos obtenidos por distrito.

A partir de los arrays, genere una lista (con inserción en orden), a fin de poder mostrar, al final del proceso, los nombres de las tres agrupaciones que obtienen mayor cantidad de votos para cada distrito.

En todo momento en la lista sólo deben quedar a lo sumo las tres agrupaciones ganadoras para cada distrito, con nombre de agrupación (sólo los primeros 25 caracteres para el ordenamiento alfabético) y los votos obtenidos por la agrupación en qué distrito y el total de votos obtenidos en el país.

Ejemplo del archivo de texto (sus campos son de longitud variable y NO hay un carácter especial de separación de campo). El número de agrupación no tiene ninguna relación de orden ni de correlatividad. El nombre puede tener más de los 25 caracteres indicados.

```
1028Celeste y Blanca
4Verde
125Unión por Todos y Para Todos
... ..
```

El archivo binario responde a la siguiente información:

<b>nagrup</b>	número de agrupación	entero	cualquiera hasta cuatro dígitos
<b>region</b>	número de región	entero	de 1 a 9, no utilizado en este proceso
<b>distri</b>	número de distrito	entero	suponga de 1 a 20

Contemple los distintos errores que se pueden producir al leer el archivo de texto (que no exista, que esté vacío, que el número o el nombre de la agrupación no existan, que el mismo número o nombre se repitan, etc.) e infórmelos. Suponga que hay un máximo de 25 agrupaciones, y controle no excederlas.

Contemple los distintos errores que se pueden producir al leer el archivo de votos (que no exista, que esté vacío, que el número de agrupación no coincida con alguna de las agrupaciones leídas del archivo de texto, que el número de distrito no esté en el rango posible, es decir la cantidad de columnas con que diseñó el array de votos, que la región no esté en el rango fijado, etc.), e infórmelos.

Tenga en cuenta que para un distrito puede haber los siguientes resultados, con  $v_p > v_s > v_t > \dots$  (restante cantidad de votos),

$v_p$	una	una	una	dos	tres o más
$v_s$	una	una	dos ó más	una o más	

<b>v<sub>t</sub></b>	<b>una</b>	<b>una o más</b>			
----------------------	------------	------------------	--	--	--

salvo que hubiera sólo una o dos agrupaciones para ese distrito.

### Ejercicio 21

Se ha hecho una encuesta de aceptación del público de una serie de productos y se desea generar un informe en que se muestren los tres productos con más aceptación por cada una de las zonas. Al efecto se dispone de un archivo de texto en el que se ha almacenado un código de identificación del producto y la denominación del mismo. Con los resultados de la encuesta, por cada producto elegido por cada encuestado se ha generado un archivo binario en que se almacenó el código de identificación del producto, el código del encuestador y la zona dónde habita el encuestado.

Ejemplo del archivo de texto (sus campos son de longitud variable y el carácter de separación de campo es ~). El primer campo es el código de identificación del producto (alfanumérico de hasta 8 caracteres) y el segundo es la denominación de los productos encuestados.

K1028~Balizas Verde Agua

A5~Cereales Rellenos de Dulce con Chispas de Chocolate

F125~Cable de Yeso

... ..

El archivo binario responde a la siguiente información:

<b>prod</b>	<b>código de identificación del producto</b>	alfanumérico	cualquiera, de hasta 8 caracteres
<b>encu</b>	<b>código del encuestador</b>	entero	de 1 a 197
<b>zona</b>	<b>zona</b>	entero	suponga de 1 a 20

El jefe de programadores ha decidido que se deben resolver las siguientes funciones:

Una función que lea el archivo de texto y almacene en arrays bidimensionales los códigos de identificación de los productos y sus denominaciones (**prod** y **deno**). Haga uso de una función que se encargue de separar y validar estos campos (que no estén vacíos, que el **prod** no exceda los 8 caracteres, etc.).

Una función que se encargue de leer (del archivo binario) y acumular las preferencias del público.

Una función que a partir de los arrays invoque a una función que se encargue de insertar en orden en una lista de modo de poder hacer un informe con los cinco productos que cuentan con mayor aceptación en cada zona, de modo que quede ordenado por zona, cantidad de encuestados que lo eligen, y denominación del producto. Haga las consideraciones correspondientes al ejercicio anterior para determinar los cinco (o más) productos preferidos.

## **Trabajo Práctico 4**

### **Ejercicio 1**

Valiéndose de las primitivas de Árbol vistas en clases, escriba un programa que permita cargar información en un árbol binario de búsqueda. Esta información estará compuesta de : legajo, apellido-y-nombre y cargo (alfanuméricos de 10, 35 y 15 caracteres respectivamente), fecha de alta y fecha de baja (la fecha de alta no puede faltar a diferencia de la de baja).

Valiéndose de un menú :

Pruebe las primitivas de CargarArbol en ambas versiones (recursiva e iterativa).

Implemente las funciones que recorren el árbol, mostrando la información de sus nodos en las tres formas de recorrido (EnOrden, PreOrden y PosOrden).

Implemente la función que determina la altura del árbol.

Implemente funciones que :

a- muestre los nodos hoja.

b- muestre los nodos no-hoja.

c- muestre los nodos que sólo tienen hijo por izquierda.

d- muestre los nodos que tienen hijo por izquierda.

e- determine si es un árbol semi-balanceado (AVL).

f- determine si es un árbol balanceado.

g- elimine todos los nodos de un árbol.

h- 'pode' las ramas de un árbol de modo que no supere una altura determinada.

i- 'pode' las ramas de un árbol de una altura determinada o inferior.

Al terminar el programa, genere un archivo con la información del árbol, haciendo uso de una función que lo recorre en pre-orden.

### **Ejercicio 2**

Valiéndose de un árbol en el que sólo almacena la clave y el número de registro que le corresponde en el archivo, escriba un programa que intente abrir un archivo, y si lo puede abrir, haga la carga del árbol. La información a tratar es la del ejercicio anterior.

A continuación, haciendo uso de un menú, que permita :

a- agregar nuevos registros de información (siempre que la clave no exista en el árbol), agregando el registro al final del archivo y su clave y número de registro en el árbol.

b- que permita ingresar la clave, para buscarla en el árbol y con el número de registro muestre la información del archivo.

c- que recorriendo el árbol, muestre la información de los registros del archivo en orden.

d- ídem en pre orden.

e- ídem en pos orden.

f- ídem [b-], pero para asignar la fecha de baja.

Al terminar el programa, debe generar un nuevo archivo ordenado por la clave.

### **Ejercicio 3**

Escriba un programa que le permita verificar que el archivo ha quedado ordenado.



**Ejercicio 4**

Resuelva las siete primitivas de Pila y Cola implementada en lista circular con asignación dinámica de memoria.

**Ejercicio 5**

Resuelva las primitivas de lista doblemente enlazada, teniendo en cuenta que para la inserción en la lista doblemente enlazada hay tres variantes : tener la dirección del primer nodo de la lista, del último de la lista, y del último insertado (cuando se hace una carga ordenada).

Resuelva la inserción al comienzo, al final y en orden por una clave.

Resuelva el ordenamiento de la lista.

Resuelva la búsqueda por la clave, con eliminación o no del nodo, recuperando la información en caso de encontrarse la clave y teniendo en cuenta que hay tres variantes (se tiene la dirección del primer nodo de la lista, del último, o del último tratado).

Escriba un programa que le permita comprobar el correcto funcionamiento de sus funciones.

**Ejercicio 6**

Investigue y resuelva una estrategia que permita visualizar cómo queda cargado un árbol binario. Al efecto, y por sencillez del diseño, suponga que la única información es una clave numérica entera de a lo sumo dos.

## Trabajo Práctico 5

### Ejercicio 1

Escriba, pruebe y saque conclusiones con un programa en el que declara una clase Punto con miembros privados enteros (`coordX`, `coordY`) que representen las coordenadas de un punto en la pantalla (suponga una resolución de 1024x768).

En primer lugar, sólo declare y desarrolle (abreviado en adelante con **DyD**) para esta clase un método mostrar que muestre por pantalla las coordenadas del punto en el formato (`x`, `y`). Pruebe el programa con algunos objetos creados por defecto y otro objeto creado por copia. Identifique los 'warning' que le da su compilador.

DyD el constructor por defecto que inicialice las coordenadas del punto representando el centro de la pantalla. Verifique el funcionamiento del constructor de copia generado por el compilador. Saque conclusiones sobre la necesidad del constructor por defecto y del constructor de copia.

DyD los métodos constructores:

- con un sólo parámetro (se asigna directamente a la componente `coordX`, pero a la componente `coordY` teniendo en cuenta la relación de  $1024/768 = 4/3$ ).
  - con dos parámetros (uno para cada componente)
- y utilícelos en su programa. Además verifique como trabaja la asignación múltiple entre puntos, mostrándolos antes y después de la asignación (`p1 = p2 = p3`).

Observación: si la/las coordenada/s de estos constructores no corresponden a puntos de la pantalla considere que se trata de una matriz esférica, es decir que el punto  $(-1, -1) \Rightarrow (1022, 767)$  y  $(1024, 768) \Rightarrow (0, 0)$ .

DyD el destructor de la clase, de modo que manifieste que se está ejecutando. Dado que es totalmente innecesario un destructor para una clase como esta, declare una etiqueta (p.ej.: `#define DEPURAR`) y directivas de compilación condicional (p.ej.: `#ifdef DEPURAR ... #endif`), de modo que cuando se elimine la declaración de la etiqueta el destructor no sea compilado en el programa. Este destructor podría mostrar los miembros de información (las coordenadas) del objeto que se destruye, además de la dirección de memoria del objeto que se está destruyendo (puntero `this`). Agregue además con directivas de compilación condicional para mostrar en los constructores la misma información que en el destructor y pruebe nuevamente el programa. compruebe en qué orden se destruyen los objetos.

DyD un constructor parametrizado con ambos parámetros con valores por defecto. Verifique los errores de compilación, y corrijalos 'comentando' el código. Verifique si puede crear un objeto con tan sólo el segundo argumento, sin el primero. Aunque innecesarios para los objetos de esta clase, DyD el constructor de copia y la sobrecarga del operador de asignación, y verifique que funcionan correctamente. Una vez que lo haya hecho controle con sus docentes la correcta escritura de los argumentos (utilizando `const` y `&`). No deje de utilizar las directivas de compilación condicional para estos métodos.

DyD las sobrecargas de los operadores << y >> para mostrar/ingresar las coordenadas de un punto.

Suponga que incrementar/decrementar un punto consiste en sumarle/restarle 1 a cada componente. DyD las sobrecargas de (++) --) pre/pos incremento/decremento, verificando su correcto funcionamiento.

DyD la suma y diferencia de dos puntos. Suponga que para la suma se adicionan las componentes y para la resta se las resta.

DyD los operator += y -=.

## **Ejercicio 2**

Escriba una clase **Hora** con tres miembros enteros para almacenar las horas, minutos y segundos. Resuelva los métodos indispensables para crear objetos de la clase **Hora**, mostrar la hora, incrementar y decrementar en un segundo (++) -- con pre/pos incremento/decremento), calcular (+ -) la suma/resta o asignar (+= -=) la suma/resta entre dos horas, sumar/restar un entero que represente una cantidad (positiva o negativa) de segundos, sumar/restar (+ - += -=) una hora con un objeto/con un tipo de dato hora. Desarrolle los métodos de comparación (== !=) y relacionales (< > <= >=) entre horas y entre hora y tipo de dato hora.

## **Ejercicio 3**

Ídem anterior, pero con la estrategia de almacenar sólo un entero largo con la cantidad de segundos que represente la hora.

## **Ejercicio 4**

Escriba una clase **Fecha** con tres miembros enteros para el día, mes y año. Resuelva los métodos indispensables para crear objetos de la clase **Fecha**, mostrar la fecha, incrementar y decrementar en un día (++) -- con pre/pos incremento/decremento), calcular (+ -) o asignar (+= -=) la suma/resta entre dos fechas, sumar/restar un entero que represente una cantidad (positiva o negativa) de días, sumar/restar (+ - += -=) una fecha con un objeto/con un tipo de dato fecha. Desarrolle los métodos de comparación (== !=) y relacionales (< > <= >=) entre fechas y entre fecha y tipo de dato fecha.

## **Ejercicio 5**

Ídem anterior, pero con la estrategia de almacenar sólo un entero largo con la cantidad de días que representen la fecha.

**Ejercicio 6**

Escriba y pruebe una clase **Cadena**, que permita contener y manejar una cadena de caracteres (con asignación dinámica de memoria). Declare y desarrolle :

- los constructores por defecto, parametrizado y de copia,
- el destructor,
- los métodos que permiten el ingreso de una cadena por teclado (<<) y mostrarla por pantalla (>>),
- el de asignación (**operator =**), los de comparación (**operator ==** y **operator !=**), los relacionales (> >= < <=),
- el de concatenación (+), y los **operator +=** en sus distintas versiones.

**Ejercicio 7**

Escriba una clase **PuntoDeColor** que herede de las clases **Punto** y **Cadena**.

**Ejercicio 8**

Escriba una clase **PuntoColor** cuyos miembros de información sean de la clase **Punto** y de la clase **Cadena**

**Ejercicio 9**

Escriba una clase **Persona** con atributos de información privados para apellido/s y nombre/s (con asignación dinámica), domicilio (con asignación dinámica), localidad (con asignación estática), fecha de nacimiento, D. N. I., sexo y fecha de nacimiento. Resuelva los constructores por defecto, parametrizados en varias versiones (p.ej.: intercambiando el orden de los argumentos), y constructor de copia. resuelva los operadores de asignación, inserción en el flujo de salida (<<) y extracción del flujo de entrada (>>).

**Ejercicio 10**

Heredando de la clase **Persona** cree la clase **Empleados** extendiendo la información privada con cargo (con asignación estática), sector (con asignación dinámica) y fecha de ingreso y fecha de baja y los mismos métodos del punto anterior.

Ídem anterior para crear la clase **Alumnos**, extendiendo la información privada con carrera, fechas de ingreso, egreso y del último examen aprobado, y cantidad de materias aprobadas.

**Ejercicio 11**

Genere una clase **DatosEmpleo** y otra clase **DatosAlum** con la misma información y métodos del punto anterior.

Heredando de la clase **Persona** y **DatosEmpleo** genere la clase **Empleados** con los mismos métodos antes resueltos.

Ídem con **Persona** y **DatosAlum** para la clase **Alumno2**.

**Ejercicio 12**

Utilizando la clase **Fecha** y la clase **Cadena** volver a resolver los tres puntos anteriores, pero esta vez con miembros de información de la clase **Cadena** (para el apellido/s y nombre/s, domicilio, localidad, cargo, sector, etc.) y de la clase **Fecha** (para nacimiento, alta y baja, etc.).

**Ejercicio 13**

Utilizando las clases **Empleados** y **Alumnos** que heredan de la clase **Persona** cree la clase **AlumnoEmpleado**. Escriba una función que reciba la dirección de un objeto de cualquiera de las clases (base o derivadas). En esta función haga uso de los métodos.