Universidad Tecnológica Nacional

Facultad Regional Buenos Aires

Departamento de Electrónica

Cátedra: Informática I - Plan 95A

GUIA DE TRABAJOS PRACTICOS

Ciclo Lectivo 2011

Indice

T.P. Nº 1. SISTEMAS DE NUMERACIÓN.....5

T.P. Nº 2. CONTROL DE LA PROGRAMACIÓN: 9

T.P. Nº 3. FUNCIONES.....21

T.P. Nº 4. PUNTEROS Y STRINGS.....23

T.P. Nº 5. RECURSIVIDAD.....30

T.P. Nº 6. ARREGLOS.....33

T.P. Nº 7. ESTRUCTURAS UNIONES CAMPOS DE BITS.....47

T.P. Nº 8. ESTRUCTURAS COMPLEJAS. LISTAS.....55

T.P. Nº 9. EJERCICIOS INTEGRADORES 1º NIVEL.....63

T.P. Nº 10. OPERADORES A NIVEL DE BITS Y PORTS.....67

T.P. Nº 11. STREAMS.....72

T.P. Nº 12. PROGRAMACIÓN AVANZADA EN LINUX.....89

T.P. Nº 13. EJERCICIOS INTEGRADORES.....93

Trabajos Prácticos de Informática I

Introducción y Régimen de aprobación

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

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

Cada alumno deberá presentar aquellos ejercicios que lleven la indicación ***Entrega Obligatoria***. La entrega de cada ejercicio se efectuará sin excepciones en las fechas estipuladas en el cronograma de clase que se entregará en la primera clase del ciclo lectivo.

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

La no entrega de un ejercicio en la fecha establecida equivale a considerar al alumno o al grupo **Ausente** en ese práctico. En consecuencia se considerará **No Aprobado** dicho práctico. De acuerdo con el reglamento vigente, la aprobación de los Trabajos Prácticos requiere el 80% de los mismos Aprobados. En el caso de esta guía de Trabajos Prácticos se requiere la aprobación del 80% de los estipulados de ***Entrega Obligatoria***.

Formato de presentación

- Los archivos fuentes deben tener en todos los casos los comentarios necesarios para clarificar su lectura.

- Deben llevar por cada subrutina / función, un encabezado con la descripción de la operación que realiza, los parámetros que espera como entrada, y en que forma y donde entrega sus resultados.

- Como encabezado del programa, debe haber un comentario que explique claramente que hace dicho programa, y las instrucciones detalladas (comandos) para su compilación y linkeo.

Recomendaciones

Esta guía contiene un muy extensa y exhaustiva tira de ejercicios. Se piden en forma obligatoria aquellos que cada año la Cátedra considera esenciales para verificar la adquisición por parte de los alumnos de las competencias que buscan desarrollarse.

No obstante, se considera de suma importancia la resolución de la mayor cantidad posible de ejercicios para garantizar la fluidez de programación que permita resolver en los plazos estipulados los problemas planteados en lso exámenes, los cuales suponen un nivel de práctiac intensivo por parte de los alumnos.

IMPORTANTE:

FORMA DE ENTREGA DE LOS TRABAJOS PRACTICOS

La entrega se realizará por e-mail a los ayudantes del curso.

La no entrega de la versión final completa del TP en la fecha estipulada por parte del alumno se considerará ausente.

T.P. Nº 1.Sistemas de Numeración

Ejercicio 1.1.

Completar la grilla siguiente completando los números equivalentes en las bases numéricas que están vacantes.

Binario	Octal	Decimal	hexadecimal
1010000			
		120	
			3D
1101			
			96
	565		
10100100101			
			BF305A
	766		
-11100111			
	-152.71		
		5634.22809	

Ejercicio 1.2.

Pasar los siguientes números decimales a la base indicada con un error menor o igual al indicado

Número	Base	$Error_{10}<$
0,267	2	0,0010
52,38	2	0,0001
129,64	2	0,1000
163,97	8	0,0001
954,62	16	0,0001

Ejercicio 1.3.

Realizar las siguientes sumas de enteros signados en Ca2:

$$\begin{array}{rrr} 1010_2 & 1001_2 & 1110_2 \\ + & + & + \\ 0101_2 & 0110_2 & 1010_2 \end{array}$$

10110_2	11011_2	10010_2
+	+	+
10101_2	00110_2	10110_2

7354_8	$F1E5_{16}$	3231_{16}
+	+	+
1123_8	$ABC1_{16}$	2123_{16}

Ejercicio 1.4.

Realizar las siguientes restas:

10110_2	10101_2	11010_2
-	-	-
1101_2	10011_2	10111_2

$F91F_{16}$	0334_8	1060_8
-	-	-
0101_{16}	0137_8	1776_8

Ejercicio 1.5.

Escribir en 8 bits, en complemento a 2.

57 13 154 214 243 194

241 163 38 121 157 123

1010_2 $F1_{16}$ 3074_8 1100_2 513_8 37_{16}

Ejercicio 1.6.

Expresar en punto flotante simple precisión los números siguientes expresados en base 10

165,625 -165,625 7564618909631 11659,84375

93,16794382512396283 93,16794382512396284 -9756.609375 0.0625

-6257.234375 0.005859375 -0.005859375 -675.125

Ejercicio 1.7.

Expresar en base 10 los siguientes números dados en formato de Punto Flotante Simple Precisión

35C1F	93700D	ECF	3ED
7A72C	C2E45	39591	4B9DE19F

Ejercicio 1.8.

Expresar en Punto Flotante Doble Precisión los siguientes números expresados en base 10

165,625	-165,625	7564618909631	11659,84375
93,16794382512396283	93,16794382512396284	-9756.609375	0.0625
-6257.234375	0.005859375	-0.005859375	-675.125

Ejercicio 1.9.

Expresar en base 10 los siguientes números que están en formato de punto flotante doble precisión

351CF	937D12	ECFE2	3ED95
7A79C	C2ED5	39591	4B94E19F

Ejercicio 1.10.

Indicar qué número decimal, representa los siguientes números expresados como punto flotante (simple o doble precisión)

$9EC1935F_{16}$	$CD940103_{16}$
$3EAC1000_{16}$	$A\ E8\ F5000_{16}$

Ejercicio 1.11.

Utilizando una "palabra" de 3 bits de ancho, listar todos los números binarios signados y sus equivalentes decimales posibles representables en:

a) Signo y magnitud

b) Complemento a 1

c) Complemento a 2

Ejercicio 1.12.

Utilizando una "palabra" de 4 bits de ancho, listar todos los números binarios signados y sus equivalentes decimales posibles representables en:

a) Signo y magnitud

b) Complemento a 1

c) Complemento a 2

Ejercicio 1.13.

A partir de los resultados de los dos ejercicios previos generalice el rango de valores (en decimal) que puede representarse en cualquier número x de bits dado, utilizando:

a) Signo y magnitud

b) Complemento a 1

c) Complemento a 2

Ejercicio 1.14.

Asuma un computador con un ancho de palabra de 32 bits. En esos 32 bits, deseamos representar el valor 2795.

a) Como se representará el valor decimal del valor 2795?

b) Si el computador soporta ASCII de 8 bits, cual será la representación de la cadena de caracteres 2795?

c) Si el computador soporta BCD empaquetado, cual será la representación del número 2795?

T.P. Nº 2. Control de la Programación:

Estructuras Secuenciales

Ejercicio 2.1.

Escribir un programa que imprima la leyenda “Lenguaje C” en la pantalla

Ejercicio 2.2.

Escribir un programa donde ingresa por teclado una temperatura expresada en grados Celsius. Calcula su valor en grados Kelvin, y muestra por pantalla el valor en ºC y su equivalente en ºK. (Recordar ºC=ºK - 273)

Ejercicio 2.3.

Escribir un programa donde ingresan por teclado dos valores que son almacenados en las variables A y B del mismo tipo. El programa deberá intercambiar los valores de las mismas y mostrar por pantalla los valores ingresados y luego del intercambio.

Ejercicio 2.4.

Suponga que debe escribir un programa para calcular la resistencia total de un circuito en serie. En tal circuito la resistencia total es la suma de todos los valores individuales de las resistencias. Suponer que el circuito consiste en una cantidad de 3 resistores de 56, 33 y 15 Ω respectivamente.

Ejercicio 2.5.

Con los datos del problema 2.4. Haga un programa que calcule el promedio de las resistencias.

Ejercicio 2.6.

Modifique el programa 2.2. para que el mismo transforme y muestre la temperatura ingresada en grados Celsius, en grados Fahrenheit.

(Recordar ºC = 5/9 * [ºF-32])

Ejercicio 2.7.

Escribir un programa que calcule la superficie de un círculo cuyo radio se ingresa por teclado.

Ejercicio 2.8.

Escribir un programa que calcule el tiempo de muestreo de una señal de 250 Hz. Para ello se aplica el teorema de Nyquist: N=1/[2*f], en donde **N** es tiempo de muestreo en segundos y **f** (frecuencia en Hz.))

Ejercicio 2.9.

Modifique el problema 2.4. para que pueda calcular con esos datos, la resistencia total de un circuito en paralelo. (Recordar Rp = 1/[1/R1 + 1/R2 + 1/R3]). La salida deberá aparecer con dos decimales.

Ejercicio 2.10.

Modifique el problema 2.4. para que pueda calcular la resistencia total de un circuito en serie con los siguientes datos. El circuito consiste en una cantidad de 7 resistores de 56 Ω, 5 de 33 Ω y 9 de 15 Ω.

Responda además, las siguientes preguntas:

a) ¿Cuántas salidas requiere este problema de programación?

b) ¿Cuántos datos de entrada tiene el problema?

Ejercicio 2.11.

Pruebe el algoritmo escrito, usando la siguiente muestra de datos: 47KΩ, 680Ω y 2,2MΩ.

Ejercicio 2.12.

Escriba un programa que dibuje un recuadro, en el interior del mismo ubique el nombre de la materia y en el extremo inferior derecho de la pantalla el nombre del alumno.

Selectivas o Decisión

Ejercicio 2.13.

Escribir un programa que lea dos valores que correspondan a los extremos izquierdo y derecho de un intervalo. Luego, se debe dividir dicho intervalo en 4 subintervalos iguales. Si la longitud del intervalo ingresado no fuese múltiplo de 4, se redondeará al primer múltiplo mayor. Se deberá mostrar por pantalla el rango ingresado y cada uno de los subrangos.

Ejercicio 2.14.

Escribir un programa que calcule la diferencia de dos números enteros e informe por medio de un mensaje en pantalla, si la misma es positiva o negativa.

Ejercicio 2.15.

Modificar el programa 2.7. de manera tal que si el dato ingresado no es positivo, presente por pantalla el mensaje "***Dato NO Válido***".

Ejercicio 2.16.

Escribir un programa que calcule la diferencia de dos números enteros e informe si la misma es positiva, negativa o cero.

Ejercicio 2.17.

Escribir un programa que reciba tres valores reales, correspondientes a las longitudes de los lados de un triángulo, y luego informe si el triángulo ingresado es equilátero, isósceles o escaleno.

Ejercicio 2.18.

Modificar el programa anterior para que informe del ingreso de un valor nulo o negativo

Ejercicio 2.19.

Modificar el programa anterior para comprobar que los números ingresados conformen realmente un triángulo.

Ejercicio 2.20.

Modificar el programa anterior para determinar si el triángulo es rectángulo e indique cual de los datos es la hipotenusa.

Ejercicio 2.21.

Se tienen un sensor de temperatura, que registran 10 temperaturas distintas cada uno. Esas temperaturas se ingresan por teclado. Se pide determinar e informar por pantalla:

a) Temperatura promedio.

b) Máxima temperatura registrada.

c) Mínima temperatura registrada.

d) Cantidad de veces que la temperatura se encontró entre los 20 y los 45 ºC.

Ejercicio 2.22.

Elabore un programa donde ingresan dos valores reales y el símbolo de la operación (‘+’, ‘-’, ‘*’, ‘/’). Se deberá presentar por pantalla, los datos ingresados, la operación y el resultado. Si el símbolo utilizado no correspondiera a ninguna de las cuatro operaciones deberá presentar un mensaje de "***Operación NO Válida***". (**El programa deberá resolverse mediante el uso de la estructura switch**).

Ejercicio 2.23.

Modifique el programa anterior de forma tal que si la operación es de división y el valor ingresado como divisor es 0 (cero), presentará por pantalla un mensaje de "***Operación NO Válida***
División por Cero". En dos renglones como muestra el texto. (**El programa deberá resolverse mediante el uso de la estructura switch**).

Ejercicio 2.24.

Modifique el programa anterior de manera que lo que ingrese sean dos números complejos. (**NO se puede emplear la biblioteca cuyas definiciones están en el archivo header <complex.h>. El programador podrá determinar la forma de ingreso de los datos y egreso de la información**).

Ejercicio 2.25.

Escriba un programa donde se ingresa un valor entero y positivo, e informe si el valor es par o impar.

Ejercicio 2.26.

Un programa donde ingresa por teclado un par de valores no nulos X e Y, que representan las coordenadas rectangulares de distintos puntos en el plano. Se pide determinar e informar por pantalla:

a) A cual cuadrante pertenece el punto.

b) Distancia al origen de coordenadas.

Ejercicio 2.27.

Escribir un programa tal que ingresados los coeficientes A, B y C de una ecuación cuadrática, informe sus raíces, en caso de que las mismas sean imaginarias presentarlo adecuadamente. (**NO se puede emplear la biblioteca cuyas definiciones están en el archivo header <complex.h>.**)

Ejercicio 2.28.

Modifique el programa anterior para que informe y resuelva el caso en que A sea 0.

Repetitivas o Iteración

Ejercicio 2.29.

Escribir un programa que imprima la tabla de multiplicación de cualquier número de 0 a 10 ingresado por teclado.

Ejercicio 2.30.

Escribir un programa que realice la sumatoria de los números del 1 al 100.

Ejercicio 2.31.

Escribir un programa que, dado un número N, calcule y presente por pantalla los siguientes valores: 1+0*N; 1+1*N; 1+2*N;... ; 1+10*N.

Ejercicio 2.32.

Escribir un programa que realice la sumatoria del rango de números enteros entre dos valores ingresados por teclado.

Ejercicio 2.33.

Escribir un programa que calcule el promedio de 10 valores numéricos ingresados por teclado.

Ejercicio 2.34.

Escribir un programa que, dado un número N entero y positivo, calcule el factorial de N.

Ejercicio 2.35.

Ingresar un número entero por teclado y determinar si es primo.

Ejercicio 2.36.

Un tirador realiza 20 disparos a un blanco cuyo centro coincide con el origen de coordenadas. Leyendo la abscisa y la ordenada de cada impacto, deberá calcularse el puntaje obtenido. El blanco tiene cuatro zonas que, de acuerdo a la distancia del impacto al origen (R), son:

ZONA	DENOMINACIÓN	PUNTAJE
0 <=R <=1	centro	10
1 < R <= 5	medio	5
5 < R <= 10	externo	1
R > 10	falla	0

Ejercicio 2.37.

Leer 30 números desde teclado, y determinar e informar por pantalla:

a) el promedio de cada uno de los 5 grupos de 6 valores consecutivos

b) el promedio total.

Ejercicio 2.38.

Se lee desde el teclado una serie de N (desconocido), grupos de 5 valores. Los datos finalizan cuando el primer valor de una serie es -1. El programa deberá calcular e informar por pantalla:

a) el promedio de cada uno de los N grupos de 5 valores consecutivos

b) el promedio total.

c) la cantidad de grupos.

Ejercicio 2.39.

Escriba un programa donde ingresa una serie de números enteros y positivos e informe cual es par y cual es impar. El fin del ingreso se da al recibir un 0 que no pertenece a la serie. Al finalizar deberá informar cuantos son par y cuantos impar.

Ejercicio 2.40.

Escriba un programa donde ingresa una serie de números enteros y positivos e informe si la serie se encuentra ordenada en forma estrictamente creciente. El fin del ingreso se da al recibir un 0.

Ejercicio 2.41.

Escriba un programa donde ingresa una serie de números reales. Cuando encuentra un valor mayor que 283, deja de leer e informa cuantos elementos tiene la serie.

Ejercicio 2.42.

Modifique el programa de factorial de manera tal de **NO** utilizar ***for*** para el cálculo y en su lugar utilizar ***while*** o ***do-while***.

Ejercicio 2.43.

Modifique el programa de factorial de manera tal de **NO** deje pasar al área de cálculo hasta que el valor ingresado no se halle entre 0..35.

Ejercicio 2.44.

Escribir un programa que calcule el promedio de una cantidad indeterminada de valores numéricos positivos. El ingreso finaliza al ingresar el valor cero.

Ejercicio 2.45.

Escribir un programa donde ingresa una serie de valores enteros y luego informe la cantidad de valores pares e impares ingresados. El programa finaliza cuando se ingresa 0.

Ejercicio 2.46.

Se ingresan pares de valores numéricos tales que el primero de ellos es un código, que puede ser 1, 2, 3 ó 4 y el segundo es un número real. Se pide determinar e informar:

a) La sumatoria de los números con código 1.

b) La productoria de los que llevan código 2.

c) El promedio de los que llevan código 3.

El programa finaliza cuando se ingresa el par correspondiente al código 4.

Ejercicio 2.47.

Un programa recibe por teclado pares de valores no nulos X e Y, que representan las coordenadas rectangulares de distintos puntos en el plano. Se pide determinar e informar por pantalla:

a) Cantidad de puntos que pertenecen a cada cuadrante.

b) Sumatoria de las distancias al origen de los puntos pertenecientes al primer cuadrante.

El fin de datos se indica con X e Y iguales a cero.

Ejercicio 2.48.

Escribir un programa que lea dos valores que correspondan a los extremos izquierdo y derecho de un intervalo. Luego, se debe dividir dicho intervalo en 4 subintervalos iguales, que se numerarán de 1 a 4. Si la longitud del intervalo no fuese múltiplo de 4, se deberá aplicar el criterio de redondeo que se considere mas adecuado para satisfacer el requerimiento.

A continuación, el programa debe leer números reales, y determinar la cantidad de valores pertenecientes a cada subintervalo. El fin de ingreso termina ingresando un valor igual al extremo derecho del intervalo. Antes de terminar el programa presentará la cantidad de números que cayeron en cada subintervalo.

Ejercicio 2.49.

Escribir un programa que pida el número de legajo (entero de 8 cifras) y la nota de un examen de los alumnos de un curso. El fin de ingreso se indica con legajo igual a cero.

El programa debe informar:

a) Cantidad de alumnos del curso

b) Promedio de las notas

c) Cantidad de alumnos aprobados (aprueban con promedio igual o mayor que 6)

d) Mayor nota.

e) Cantidad de alumnos que obtuvieron la mayor nota.

e) la cantidad de alumnos con cada calificación conceptual, por cada asignatura, según la siguiente tabla:

Calificación Conceptual	Nota numérica
Sobresaliente	10
Muy bueno	8 ó 9
Bueno	6 ó 7
Regular	4 ó 5
Insuficiente	1 a 3

Ejercicio 2.50.

Ingresar un número entero por teclado y determinar si es perfecto. Un número es perfecto cuando la suma de todos sus factores, a excepción de sí mismo, da como resultado el número original.

Ejercicio 2.51.

Escribir un programa que pida el ingreso de un número entero positivo por teclado, y luego busque todos los números perfectos que se encuentren entre 1 y el número ingresado.

Ejercicio 2.52.

Se dice que dos números enteros a y b son "amigos" cuando la suma de los múltiplos de a (a excepción de sí mismo) es b y la suma de los divisores de b (también a excepción de sí mismo) es a. Escribir un programa que investigue las parejas de números amigos existentes entre 1 y un número entero positivo ingresado por teclado.

Ejercicio 2.53.

El valor aproximado del número de Euler, e , se puede obtener con la siguiente fórmula: e = 1 + 1/1! + 1/2! + 1/3! + 1/4! + 1/5! + ...

Escribir un programa que calcule el valor aproximado de *e* mediante un ciclo repetitivo que termine cuando la diferencia entre dos aproximaciones sucesivas difiera en menos de 10^{-9}.

Ejercicio 2.54.

Escribir un programa que permita determinar el máximo y el mínimo de un conjunto de valores ingresados por stdin (sin almacenar la totalidad de los valores leídos). A tal efecto debe ingresarse primero la cantidad esperada de elementos a procesar, y luego el lote de datos de a uno por vez. Finalizado el ingreso de datos, mostrar el resultado por stdout con 3 decimales. Realizar todas las validaciones que considere necesarias.

Ejercicio 2.55.

Escribir un programa que pruebe la efectividad de la función de biblioteca rand(). Comience por inicializar 10 contadores, como cuentacero, cuentauno, cuentados, ..., hasta cuentanueve a cero. Luego genere una gran cantidad de números pseudoaleatorios entre 0 y 9. cada vez que ocurra un 0 se aumenta cuentacero, y así con todos los dígitos decimales. Por último presente en stdout un reporte con el número de ocurrencias de cada valor entrero entre 0 y 9, y el porcentaje de cada ocurrencia.

Ejercicio 2.56.

Modificar el ejercicio 2.27. de forma tal que no permita que A sea 0. El usuario deberá carga tantas veces como sea necesario has que ingrese un valor distinto de cero.

Ejercicio 2.57.

Realizar el ejercicio anterior para el caso de que se ingresen N ecuaciones. Proponga la condición de finalización del programa que considere más adecuada.

Ejercicio 2.58.

Modificar el ejercicio anterior para que el ingreso de ecuaciones finalice cuando 2 de ellas poseen raíces imaginarias.

Ejercicio 2.59.

Modificar el ejercicio anterior para que el ingreso de ecuaciones finalice cuando poseen raíces imaginarias 2 ecuaciones **consecutivas**.

Ejercicio 2.60.

Escribir un programa que invierta los dígitos de un número positivo entero. (Sugerencia: usar operadores módulo, %, y división, /, para ir obteniendo los dígitos uno a uno). El valor se ingresa por teclado.

Ejercicio 2.61.

Escribir un programa que lea del teclado la fecha de nacimiento de una persona y calcular su edad.

Ejercicio 2.62.

Escribir un algoritmo que tome números decimales y los muestre en pantalla en binario. La presentación debe ser por cada línea el valor original y separado por un tabulador el valor convertido a binario.

Ejercicio 2.63.

Escribir un algoritmo que tome de números decimales y los muestre en octal por pantalla.

Ejercicio 2.64.

Escribir un algoritmo que ingrese números decimales y los muestre por pantalla en hexadecimal.

Ejercicio 2.65.

Se tienen 5 sensores de temperatura, que registran 15 temperaturas distintas cada uno. Esas temperaturas se ingresan de a una por vez, primero las 15 del 1er sensor, luego las 15 del segundo, y así sucesivamente.

b) Temperatura promedio detectada por cada sensor.

c) Máxima temperatura registrada por cada sensor.

d) Número de sensor que registró la temperatura máxima.

Ejercicio 2.66.

¿Cómo se almacenan los distintos datos en memoria? Explique como se alocan los distintos bytes que conforman las siguientes declaraciones de variables en la memoria de la PC

```
char val1;

int val2;

float val3;

double val4;
```

Ejercicio 2.67.

Modifique el programa anterior para comprobar que se ingresen únicamente números. Si se ingresa un carácter, el programa debe presentar un mensaje de error informando la situación, descartar el valor ingresado y continuar con su operación normalmente.

T.P. Nº 3. Funciones

Condiciones Generales para resolver el Trabajo Práctico.

Todos los ejercicios deben estar escritos en un archivo fuente que contenga solo la función pedida. Las definiciones que se requieran deben efectuarse en un archivo header que tenga el mismo nombre del fuente C.

Objetivo:Las funciones desarrolladas en esta sección irán luego a bibliotecas de código para ser utilizadas en los futuros Trabajos Prácticos

Ejercicio 3.1.

Escribir una función que reciba dos números reales como argumento y devuelva su MCD claculado mediante el algoritmo de Euclides.

Nota: En lenguaje moderno, el algoritmo se describe como sigue:

1. Dados dos segmentos *AB* y *CD* (con *AB*>*CD*), restamos *CD* de *AB* tantas veces como sea posible. Si no hay residuo, entonces *CD* es la máxima medida común.
2. Si se obtiene un residuo *EF*, éste es menor que *CD* y podemos repetir el proceso: restamos *EF* tantas veces como sea posible de *CD*. Si al final no queda un residuo, *EF* es la medida común. En caso contrario obtenemos un nuevo residuo *GH* menor a EF.
3. El proceso se repite hasta que en algún momento no se obtiene residuo. Entonces el último residuo obtenido es la mayor medida común.

Ejercicio 3.2.

Escriba una función que reciba dos argumentos enteros x e y, y devuelva x^y.

Ejercicio 3.3.

Escribir una función que simule el tiro de un dado.

Ejercicio 3.4.

Escribir una función que simule una mano de truco para la cantidad de jugadores indicada en su argumento (2, 3, 4 ó 6 jugadores). Sólo se pide repartir las cartas.

Ejercicio 3.5.

Realizar funciones que puedan hacer las cuatro operaciones básicas en binario, hexadecimal y octal.

Ejercicio 3.6.

Intente integrar todas las funciones del ítem anterior para lograr una calculadora.

Ejercicio 3.7.

Escribir una función que calcule el factorial de un número natural pasado como parámetro, en forma iterativa. Realizar todas las validaciones que considere necesarias.

Ejercicio 3.8.

a) Defina un tipo de dato "tipodato" a partir del tipo nativo "float" (usando typedef)

b) Defina un tipo enumerativo "tescala" que contenga los símbolos CELSIUS y FAHRENHEIT.

c) Escriba una función que reciba un valor de temperatura en precisión doble, y la escala de temperaturas de destino, y realice la conversión del valor.

Ejercicio 3.9.

¿Es necesario incluir el nombre de los parámetros formales en el prototipo de una función? ¿Qué ventajas puede tener el hacerlo?

T.P. Nº 4.Punteros y strings

Condiciones Generales para resolver el Trabajo Práctico.

Para el caso de los ejercicios que pidan únicamente escribir la función, éstos deben estar escritos en un archivo fuente que contenga solo la función pedida. Las definiciones que se requieran deben efectuarse en un archivo header que tenga el mismo nombre del fuente C.

Objetivo:Las funciones desarrolladas en esta sección irán luego a bibliotecas de código sumándose a las de los Trabajos Prácticos anteriores, y serán utilizadas en los próximos Trabajos Prácticos.

Ejercicio 4.1.

Escriba una función que imprima, byte a byte, los bytes que constituyen una variable long cuya dirección recibe como argumento.

NOTA: Para apreciar mejor el resultado de este programa, se recomienda trabajar con formato hexadecimal

Ejercicio 4.2.

Repita el Ejercicio 3.8. utilizando punteros.

Ejercicio 4.3.

Escribir una función que determine si una a una secuencia de bytes recibida como parámetro está vacía o no. A tal fin, la asumirá como una secuencia en ASCII terminada en '\0' (NULL). El tipo de retorno debe ser bool, conforme al siguiente prototipo:

```
bool emptyString (const char *) ;
```

Ejercicio 4.4.

Escribir una función que reciba como argumento puntero a char, la asuma como una secuencia en ASCII terminada en '\0' (NULL) y devuelva la secuencia invertida.

```
void string_reverse (char *) ;
```

Ejercicio 4.5.

Escribir una función que reciba como argumento puntero a una secuencia de bytes, la asuma como una secuencia en ASCII terminada en '\0' (NULL), calcule su longitud y la retorne de acuerdo al siguiente prototipo:

```
int my_strlen (const char *) ;
```

Ejercicio 4.6.

Escribir una función que reciba un puntero a caracter "s" y un puntero a caracter "t", y copie la “s” en “t”, terminando la cadena con el caracter '\0' (Función strcpy () de la biblioteca <string.h>). El prototipo de la función pedida es:

```
void my_strcpy (char *t, const char *s);
```

Nota: ¿Donde deben tomarse los recaudos para que el puntero destino posea la memoria necesaria?. Si no es en su función especifique donde y como se debe salvar esta situación, mediante un comentario en el encabezado de su programa fuente.

Ejercicio 4.7.

Escribir una función que reciba como argumentos dos punteros a char “t” y “s”, que apunta cada uno a una secuencia de bytes terminados en '/0', y realice la concatenación de la secuencia “s” a continuación de la secuencia “t”, terminando la secuencia de bytes resultante con el caracter '\0' (Función strcat() de la biblioteca <string.h>).

El prototipo de la función pedida es:

```
void my_strcat (char *t, const char *s) ;
```

Nota: ¿Donde deben tomarse los recaudos para que el puntero destino posea la memoria necesaria?. Si no es en su función especifique donde y como se debe salvar esta situación, mediante un comentario en el encabezado de su programa fuente.

Ejercicio 4.8.

Escribir una función que reciba dos punteros a char como argumentos, realice una comparación lexicográfica de las secuencias de bytes finalizadas en '/0' apuntadas por cada uno, y retorne valores positivos, cero y negativos, según corresponda. (Función strcmp () de la biblioteca <string.h>).

El prototipo de la función pedida es:

```
int my_strcmp (const char *t, const char *s) ;
```

Ejercicio 4.9.

Escribir una función que dados dos punteros a caracteres y un número "n" entero recibidos como parámetros, compare lexicográficamente los primeros "n" caracteres a partir de cada puntero, devolviendo un valor positivo, cero o negativo, según corresponda. (Función strncmp() de la biblioteca <string.h>).

El prototipo de la función pedida es:

```
int my_strncmp (const char *s1, const char *s2, int n) ;
```

Ejercicio 4.10.

Escribir una función que reciba como argumento dos punteros a caracter "t" y "s", y una variable entera "n", y copie los primeros "n" caracteres de la cadena "s" sobre la cadena "t", sin terminar la secuencia resultante con el caracter nulo. (Función strncpy () de la biblioteca <string.h>).

El prototipo de la función pedida es:

```
int my_strncpy (char *t, const char *s, int n) ;
```

Nota: ¿Donde deben tomarse los recaudos para que el puntero destino posea la memoria necesaria?. Si no es en su función especifique donde y como se debe salvar esta situación, mediante un comentario en el encabezado de su programa fuente.

Ejercicio 4.11.

Escribir una función que convierta a minúsculas una cadena de caracteres recibida como argumento.

Prototipo: void strlwr (char *);

Ejercicio 4.12.

Escribir una función que convierta a mayúsculas una cadena de caracteres recibida como argumento.

Prototipo: void strupr(char *);

Ejercicio 4.13.

Escribir una función que reciba una cadena de caracteres como argumento, y la convierta a minúsculas o mayúsculas, de acuerdo a un argumento "format", cuyo tipo es case, un tipo enumerativo compuesto por los tokens UPPERCASE y LOWERCASE:

```
typedef enum { UPPERCASE, LOWERCASE } case ;
```

Prototipo de la función pedida: void change_case (char * , case) ;

Nota: Implemente esta función invocando a las dos funciones de los ejercicios previos.

Ejercicio 4.14.

Escribir una función que responda al siguiente prototipo:

void replace (char *s, char nuevo, char viejo) ;

y reemplace en la cadena "s" todas las apariciones del caracter "viejo" por el carácter "nuevo".

Ejercicio 4.15.

Escribir una función que reciba como parámetro una cadena de caracteres que comienza con espacios en blanco, y los elimine desplazando los caracteres útiles hacia la izquierda. (operación "left-trim").

Prototipo: void left_trim (char *) ;

Ejercicio 4.16.

Escribir una función que reciba como parámetro una cadena de caracteres que finaliza con espacios en blanco, y los elimine desplazando los caracteres útiles hacia la izquierda. (operación "right-trim").

Prototipo: void right_trim (char *) ;

Ejercicio 4.17.

Escribir una función que reciba dos cadenas de caracteres denominadas "s1" y "s2" respectivamente, y verifique la existencia de la cadena s2 como subcadena integrante de la s1, retornando un token del tipo enumerativo bool por el nombre de la función (ver función strstr() de la biblioteca <string.h>).

Prototipo: bool my_strstr (char *big, char *sub) ;

Ejercicio 4.18.

Escribir una función que reciba una cadena de caracteres y determine si es un palíndromo (capicúa) o no, retornando el resultado por el nombre. Considerar el caso de longitudes de cadena par e impar.

Prototipo: bool es_palindromo (const char *) ;

Ejercicio 4.19.

Escribir una función que calcule la partes entera y decimal de cualquier número real recibido como argumento, y las retorne al programa invocante. ¿Como se resuelve devolver mas de un resultado?

Ejercicio 4.20.

a) Escribir una función que reciba como parámetros un puntero a una secuencia de doubles y la cantidad de elementos que componen la secuencia, y devuelva al máximo (o el mínimo) valor contenido en la misma.

b) ¿Cómo modificaría la interfaz de la función, para pasarle al módulo un token de tipo enumerativo, que indique cuál de los dos extremos se desea calcular?

Sugerencia:

```
typedef enum { MAXIMO, MINIMO } t_extremo;
```

Ejercicio 4.21.

Escribir una función que reciba un puntero a una secuencia de doubles y su longitud como parámetros, y retorne el promedio aritmético de los valores contenidos en él.

Ejercicio 4.22.

Idem para el desvío estándar.

Nota: reutilice el código desarrollado en el ejercicio anterior.

Ejercicio 4.23.

Escribir una función que convierta un número que representa una cantidad de segundos, a su equivalente en horas, minutos y segundos.

Ejercicio 4.24.

Escribir una función que permita validar fechas recibidas como parámetro desde el programa invocante, mediante un puntero a caracter. Se consideran fechas aquellas comprendidas entre 01/01/1900 y 31/12/2100.

Ejercicio 4.25.

Escribir una función que recibe las coordenadas rectangulares de dos puntos del plano y calcule la distancia entre ellos.

Ejercicio 4.26.

Si un puntero a función es declarado como:

```
double (*pf) (double);
```

e inicializado como:

```
pf = sin;        /* apunta a sin() de la biblioteca math.h */
```

¿Por qué no puede ser declarado simplemente como: void * pf; ?

Ejercicio 4.27.

a)Escribir 3 funciones que, recibiendo una cadena de caracteres como argumento, permitan determinar si la cadena es válida como dirección IP, como dirección de correo electrónico, y como número de tarjeta de crédito.

b) Escribir una función denominada validate_string() que recibiendo una cadena de caracteres y una función de validación (pasada por puntero), determine si la cadena es válida conforme al criterio de validación indicado, retornando en consecuencia true o false por su nombre.

Ejercicio 4.28.

Entrega Obligatoria

Escriba una función que llame al juego de funciones del Ejercicio 4.5. al Ejercicio 4.12. y del Ejercicio 4.14. al Ejercicio 4.17., pero realizando la llamada a las funciones a través de punteros a las mismas.

Ejercicio 4.29.

Analice las diferencias de pasar por referencia o por valor parámetros a una función mediante la observación del stack de memoria.

¿Qué se aloja en el stack cuando se pasan los valores por referencia?

¿Qué se aloja en el stack cuando se pasan los valores por valor?

¿Qué se aloja en el stack cuando se pasa una estructura de 10 datos por referencia?

¿Qué se aloja en el stack cuando se pasa una estructura de 10 datos por referencia?

¿Qué se aloja en el stack cuando se pasa un puntero a función?

¿Qué se aloja en el stack cuando se pasa un string?

T.P. Nº 5. Recursividad

Condiciones Generales para resolver el Trabajo Práctico.

Para el caso de los ejercicios que pidan únicamente escribir la función, éstos deben estar escritos en un archivo fuente que contenga solo la función pedida. Las definiciones que se requieran deben efectuarse en un archivo header que tenga el mismo nombre del fuente C.

Objetivo:Las funciones desarrolladas en esta sección irán luego a bibliotecas de código sumándose a las de los Trabajos Prácticos anteriores, y serán utilizadas en los próximos Trabajos Prácticos

Ejercicio 5.1.

Escribir una función recursiva que devuelva la cantidad de dígitos de un número entero.

Ejercicio 5.2.

Escriba una función recursiva que genere a serie de Fibonaci.

Ejercicio 5.3.

Escribir una función recursiva que permita calcular el factorial de un número entero. ¿Conviene realmente la utilización de la versión recursiva, por sobre la iterativa? Justificar debidamente la respuesta.

Ejercicio 5.4.

Escribir una función recursiva que calcule w^k mediante multiplicaciones sucesivas, siendo k un número natural.

Ejercicio 5.5.

Escribir una función recursiva que calcule $z*v$, mediante sumas sucesivas, con z, y v enteros.

Ejercicio 5.6.

Escribir una función recursiva tal que dado un arreglo de números reales permita calcular el mínimo elemento del vector y su posición.

Ejercicio 5.7.

Escribir una función recursiva tal que dado un puntero a una cadena de números reales permita calcular el promedio de sus elementos.

Ejercicio 5.8.

Escribir una función recursiva que dado un número entero positivo calcule su imagen especular. Por ejemplo:

f(345)=543.

Ejercicio 5.9.

Escribir una función recursiva que imprima el contenido de las posiciones pares de una secuencia de números enteros, recibida mediante un puntero al inicio de la misma.

Ejercicio 5.10.

Proponer una función recursiva que recibiendo como parámetros una cadena de dígitos hexadecimales y su longitud, devuelva el valor decimal que representa dicha cadena.

Ejercicio 5.11.

Desarrolle una función recursiva que convierta números de una base a otra. Las bases disponibles serán:

- Binaria
- Octal
- Decimal
- Hexadecimal

Las mismas deberán ser listadas en una enum. La base destino se indicara con una letra

B: Binario

O: Octal

D: Decimal

H: Hexadecimal

T.P. Nº 6. Arreglos

Condiciones Generales para resolver el Trabajo Práctico.

Para el caso de los ejercicios que pidan únicamente escribir la función, éstos deben estar escritos en un archivo fuente que contenga solo la función pedida. Las definiciones que se requieran deben efectuarse en un archivo header que tenga el mismo nombre del fuente C.

El entregable es el conjunto de programas fuente (*.c y *.h), junto con un ***Makefile*** para ser construido el ejecutable mediante la orden ***make*** simplemente.

Objetivo:Las funciones desarrolladas en esta sección irán luego a bibliotecas de código sumándose a las de los Trabajos Prácticos anteriores, y serán utilizadas en los próximos Trabajos Prácticos.

Ejercicio 6.1.

Escriba una función que reciba un vector y la cantidad de elementos del mismo y lo ordene en forma ascendente

Ejercicio 6.2.

Escribir un programa que solicite valores enteros por stdin. Finalizar el ingreso con 0 y considerar 50 elementos como máximo.

Una vez finalizado el ingreso debe invocar a una función que está en un archivo fuente separado, para que imprima un listado de aquellos valores que superen al promedio de todos los valores ingresados.

Prototipo: void show_greater_average (char vector []);

Ejercicio 6.3.

Escribir una función que recibe un arreglo de 10 punteros a float, y un puntero a float (en ese orden), realice las siguientes acciones:

a) sume los diez números a los que hacen referencia los punteros del primer argumento y almacene el resultado en la dirección a la que hace referencia el segundo argumento.

b) presente en pantalla en dos columnas la dirección y el contenido de todas las variables, de entrada y resultado.

Ejercicio 6.4.

Escribir un programa que realice la carga de un vector de enteros. Los datos ingresan por stdin y se cargan en posiciones consecutivas del vector. El ingreso de datos finaliza con un valor negativo, o al alcanzar 500 valores. Una vez ingresados, se debe realizar dos listados: desde el 1er elemento al último, y desde el último al 1ro.

Sugerencia: Usar redirección desde un archivo que contenga los valores, para probar el programa. Utilice el comando more para regular la salida de su programa.

Ejercicio 6.5.

Escribir un programa que ingrese valores por stdin, y cree 4 vectores: uno con valores pares, otro con los impares, el 3ro con los positivos y el último con negativos.

Cada vector será generado por una función diferente. Cada función debe estar escrita en un archivo separado.

Se deberá imprimirlos los cuatro vectores desde otra función también generada en archivo separado, comenzando por el que más elementos tiene. Considerar un máximo de 100 ingresos. Se sale ingresando 0 (cero).

Sugerencia: Usar redirección desde un archivo que contenga los valores, para probar el programa. Utilice el comando more para regular la salida de su programa.

Ejercicio 6.6.

Repita el ejercicio anterior ordenando los vectores de positivos y pares en forma ascendente, y los dos restantes en forma descendente.

Ejercicio 6.7.

Entrega Obligatoria

Se desea ordenar un vector por los métodos pivote, burbuja, burbuja "mejorada" (con detección de orden prematuro) y selección.

Para cada método presentar por stdout:

- Cantidad de comparaciones
- Cantidad de intercambios

Sugerencia: generar el vector en forma aleatoria y poder elegir la cantidad de elementos del mismo.

Condiciones: Se pide escribir un archivo fuente en el que solo existan las cuatro funciones solicitadas, de modo que estas puedan invocarse desde cualquier otro programa que las requiera.

Ejercicio 6.8.

Escribir una función que reciba un arreglo de doubles y su longitud como parámetros, y retorne el promedio aritmético de los valores contenidos en él.

Ejercicio 6.9.

Idem para el desvío estándar.

Nota: reutilice el código desarrollado en el ejercicio anterior.

Ejercicio 6.10.

a) Defina un tipo enumerativo "tmes" con 12 símbolos que representen a cada uno de los meses del año.

b) Declare un arreglo de cadenas de caracteres con los nombres de los 12 meses del año.

c) Escriba una función que realice la traducción de tokens del tipo enumerativo "tmes" a strings, retornando la referencia a la cadena de caracteres correspondiente al módulo invocante.

Ejercicio 6.11.

Escribir una función que reciba un arreglo de N elementos de tipo entero, y su longitud, y que lo retorne cargado con N números aleatorios .

Ejercicio 6.12.

Muestrear una forma de onda es tomar valores instantáneos de la misma para determinados valores de la variable independiente. Cuando se trata de digitalizar una forma de onda es necesario tomar al menos dos muestras por cada ciclo de la misma. (Este concepto es algo mas complejo y general y se tratará mas adelante en nuestra carrera). Es decir que debemos muestrear la forma de onda por lo menos al doble de su máxima frecuencia.

Escribir una función que muestree una forma de onda determinada y almacene las N muestras en un vector de doubles recibido como parámetro, de acuerdo al siguiente prototipo:

```
void sinu_samples ( double values [], int length );
```

En donde length es la longitud del arreglo.

A efectos de no complicar nuestro análisis ya que la forma de onda debe responder a la expresión:

v(t) = Amp *seno (2 * π * Frec * t + Fase)

en donde:

t: variable independiente (tiempo)

Frec: frecuencia de la onda senoidal, en Hz.

Fase: Fase inicial en radianes.

Amp: Amplitud pico de la onda senoidal.

Ejercicio 6.13.

Escribir una función que permita evaluar una función polinómica en un punto, a partir de un vector de coeficientes y su grado recibidos como parámetros. El prototipo de la función pedida es:

```
double PolinomioX (double Coef[], int Grado, double Variable) ;
```

Ejercicio 6.14.

Escribir una función que, reutilizando el código desarrollado en el ejercicio anterior, genere N muestras de una función polinómica de grado G, cuyos coeficientes y grado recibe como parámetros, y complete con las N muestras obtenidas el vector de doubles, que recibe como primer argumento.

El prototipo de la función pedida es el siguiente:

```
void SamplePoli (double Muestras[],  int Cantidad, double Inicio,
        double Fin,  double Coeficientes[], int Grado) ;
```

Ejercicio 6.15.

Para la función cuyo prototipo es:

```
int     print_pupil ( char Legajo [],  char Apellido [], char 1erNombre [],
        char 2doNombre [], int Nota);
```

Se pide compilarla por separado e invocarla desde un programa que a partir de los datos que se obtienen desde stdin, se ocupe de emitir un listado ordenado por Nota. En caso de repetirse la nota, se tendrá en cuenta el Apellido, y si este también se repite, considerar sucesivamente el 1er y 2do nombre. Se considera

una división con 70 alumnos inscriptos máximo. El fin de ingreso se indica con legajo = 9999999.

Ejercicio 6.16.

Proponga una función para ordenar por abecedario teniendo en cuenta las particularidades del idioma español: tildes, diéresis y la vieja y querida letra Ñ.

Ejercicio 6.17.

Suponga que el gerente de una PyME le pide que escriba un programa que calcule el salario semanal de un trabajador, de acuerdo con las siguientes condiciones: un empleado ingresará el nombre del trabajador, el número de horas que ha trabajado y el nivel salarial que tiene el trabajador. El programa deberá calcular el impuesto de Hacienda (se le retiene un 20% del salario bruto) y el impuesto de seguridad social (un 8% del salario bruto).

El programa deberá mostrar en líneas separadas, la siguiente información: el nombre del trabajador, el salario bruto, la cantidad retenida para el pago del impuesto de Hacienda, la cantidad correspondiente al pago del impuesto de la seguridad social y el salario neto del trabajador.

Ejercicio 6.18.

Suponga que trabaja en un videoclub. El encargado quiere que le escriba un programa que calcule el recargo que tienen que pagar los clientes cuando se retrasan en la devolución de películas de acuerdo con las siguientes normas: el alquiler de los videos cuesta 8 pesos al día, que se pagan en el momento de alquilarlos. El periodo de alquiler es de un día. El recargo por retraso es de 4 pesos al día y se abonará al devolver la película. Cuando el cliente entregue la película, un empleado introducirá los siguientes datos: nombre del cliente, título de la película y número de días de retraso (que pueden ser cero). El programa deberá mostrar la siguiente información en líneas separadas: el nombre del cliente, el título de la película y el recargo por retraso.

Ejercicio 6.19.

Escribir una función que inicialice un arreglo de caracteres con una cadena cualquiera y la imprima por pantalla.

Ejercicio 6.20.

Escribir una función que inicialice arreglo de caracteres con una cadena cualquiera y la imprima en orden inverso por pantalla.

Ejercicio 6.21.

a) Definir un tipo enumerativo "tformato" compuesto por los símbolos MAYUSCULAS y MINUSCULAS.

b) Escribir un programa que lea de teclado una cadena de caracteres e invoque a una función externa que la convierta paramétricamente a minúsculas o mayúsculas, de acuerdo a una opción ingresada por el usuario a través del teclado. Con el resultado de la función presentará por pantalla la cadena convertida. Usar tipos enumerativos para la decodificación de la selección ingresada por el usuario.

Ejercicio 6.22.

Escribir una función que reciba dos arreglos cada uno con un número en formato de cadena de caracteres, los convierta a números de un tipo determinado y los compare, devolviendo un entero como resultado de la comparación, -1, si el primero es el mayor de ambos, 0 si son iguales, y 1 si el primero de los argumentos es el menor. Para ello utilizar las funciones atoi() y atof() de la biblioteca estándar. ¿Qué puede concluir del uso de estas funciones? ¿Qué desventajas presentan?

Ejercicio 6.23.

Escribir un programa que reciba un float y lo convierta a una cadena de caracteres, en forma inversa a la función atof(). La función retorna el resultado en un arreglo de caracteres recibido como segundo parámetro.

Ejercicio 6.24.

Reformule la función anterior de acuerdo al siguiente prototipo.

```
char * ftoa (float f);
```

Evalúe la practicidad de trabajar con punteros y arreglos en ambos programas.

Ejercicio 6.25.

Escribir una función que recibe un vector conteniendo la representación ASCII de un número entero positivo en formato decimal, y devuelva su valor en base octal sobre el mismo vector, también en ASCII.

Sugerencia: utilizar la función de biblioteca sprintf ().

Ejercicio 6.26.

Idem para base hexadecimal.

Ejercicio 6.27.

Idem para base binaria.

Ejercicio 6.28.

Escribir una función que reciba como argumento una cadena de caracteres, calcule su longitud y la devuelva en un entero positivo. (Implementación de la función strlen () de la biblioteca <string.h>).

Ejercicio 6.29.

Escribir una función que dada una cadena de caracteres y un arreglo de caracteres con espacio suficiente, copie la cadena en el arreglo, terminando la cadena con el caracter '\0' (implementación de la función strcpy () de la biblioteca <string.h>).

Esta función ya fue escrita utilizando punteros. Comparando ambas implementaciones ¿Cual de las dos soluciones resulta mas adecuada?

Ejercicio 6.30.

Escribir una función que reciba dos arreglos con cadenas de caracteres terminadas en '\0', realice una comparación lexicográfica de las mismas, devolviendo valores positivos, cero y negativos, según corresponda (implementación de la función strcmp () de la biblioteca <string.h>).

Esta función ya fue escrita utilizando punteros. Comparando ambas implementaciones ¿Cual de las dos soluciones resulta mas adecuada?

Ejercicio 6.31.

Escribir una función que convierta a mayúsculas una cadena de caracteres recibida como argumento. (Función strlwr()).

Ejercicio 6.32.

Escribir una función que convierta a minúsculas una cadena de caracteres recibida como argumento. (Función strupper()).

Ejercicio 6.33.

Escribir una función que recibe como argumentos dos cadenas de caracteres, y un entero positivo “n”. La función compara lexicográficamente los primeros "n" caracteres, devolviendo un valor positivo, cero o negativo, según corresponda. (Función strncmp () de la biblioteca <string.h>).

Esta función ya fue escrita utilizando punteros. Comparando ambas implementaciones ¿Cual de las dos soluciones resulta mas adecuada?

Ejercicio 6.34.

Escribir un programa que dada una cadena de caracteres y un arreglo de caracteres con espacio suficiente, copie los primeros "n" caracteres de la cadena sobre el arreglo, sin terminar la cadena con el caracter nulo. (Función strncpy() de la biblioteca <string.h>).

Esta función ya fue escrita utilizando punteros. Comparando ambas implementaciones ¿Cual de las dos soluciones resulta mas adecuada?

Ejercicio 6.35.

Escribir utilizando arreglos las funciones "left-trim" y "right-trim" escritas en la sección correspondiente utilizando punteros. Elabore sus conclusiones acerca del método que resulta mas conveniente de los dos.

Ejercicio 6.36.

Escribir un programa que acepte un número seguido de un espacio y luego una letra. Si la letra que sigue al número es una f, el programa deberá manejar el número introducido como una temperatura en grados Fahrenheit, convertirla en grados Celsius e imprimir un mensaje adecuado de salida. Si la letra que sigue al número es una c, el programa deberá tratar al número como una temperatura en grados Celsius, convertirla a grados Fahrenheit, e imprimir un mensaje adecuado de salida. Si la letra no es ni una f ni una c, el programa deberá imprimir un mensaje que diga que los datos son incorrectos y terminar.

Utilice las funciones de conversión desarrolladas en la sección Punteros de la presente Guía, utilizándolas como funciones externas al presente programa.

Ejercicio 6.37.

Escribir un programa que permita ingresar N números enteros en un arreglo llamado "nros", que calcule la suma total de los números en el arreglo y que despliegue el conjunto de números así como la suma de los mismos.

Ejercicio 6.38.

Escribir un programa que permita calcular el promedio aritmético de una serie de valores ingresados por teclado almacenando la totalidad de los valores leídos). A tal efecto debe ingresarse primero la cantidad esperada de elementos a promediar, y luego el lote de datos de a uno por vez. Finalizado el ingreso de datos, mostrar el resultado por pantalla con 3 decimales. Realizar todas las validaciones que considere necesarias.

¿Es necesario almacenar todos los datos del lote para poder calcular la media aritmética?

Ejercicio 6.39.

Agregar al programa del ejercicio anterior en un archivo fuente separado, una función que calcule además el desvío estándar del conjunto de datos leídos.

Luego mostrar por pantalla el desvío estándar con 2 decimales.

El desvío estándar de un lote muestral se calcula de acuerdo a la expresión:

$$\sigma = \sqrt{\frac{1}{N}\sum_{i=1}^{N}\left(x_i - \bar{x}\right)^2}$$

¿Qué se puede concluir sobre el código escrito para calcular la media aritmética?

Ejercicio 6.40.

En relación a los dos ejercicios anteriores, se sabe que no es necesario almacenar la totalidad de los datos de un lote para realizar el cálculo de la media aritmética:

$$\bar{x} = \frac{1}{N} \cdot \sum_{i=1}^{N} x_i$$

Siendo el desvío estándar del lote muestral calculado de acuerdo a la expresión:

$$\sigma = \sqrt{\frac{1}{N}\sum_{i=1}^{N}\left(x_i - \bar{x}\right)^2}$$

¿Es necesario tener almacenado forzosamente todo el lote de datos, para poder calcular el desvío? Demuestre matemáticamente que no es necesario tener almacenada la totalidad de los datos para obtener estos dos estimadores. Puede pensarse este problema como la lectura de una secuencia de valores, recalculando la media en cada iteración (Actualización de la Media). Lo mismo

sucede con el cálculo del desvío estándar (problema conocido como Actualización del Desvío Estándar, y que es además función de la Media actualizada).

Ejercicio 6.41.

Implementar el programa de forma tal que los datos a ingresar sean leídos como cadenas de caracteres y luego convertidos al tipo de dato deseado.

Ejercicio 6.42.

Escribir un programa que cargue en un vector de un tipo determinado (float, double o entero) una serie de valores leídos de stdin, y calcule el máximo y mínimo valor en él contenido, e informe además sus respectivas posiciones dentro del arreglo.

Ejercicio 6.43.

Explicar la siguiente declaración: char cadd[3][10] = { “dia”, “mes”, “año” };

Ejercicio 6.44.

a) Defina un tipo enumerativo "tmes" con 12 símbolos que representen a cada uno de los meses del año.

b) Declare un arreglo de cadenas de caracteres con los nombres de los 12 meses del año.

c) Escriba un fragmento de código que a partir del contenido de una variable de tipo tmes, imprima por pantalla la descripción del mes (traducción).

d) Idem c) pero guardando sobre una cadena de caracteres la denominación del mes, en formato:

MM (<denominación>) .

Sugerencia: utilizar la función de biblioteca sprintf ().

Ejercicio 6.45.

Modificar el ejercicio anterior para incluir un diccionario de meses, que permita presentar los nombres de los meses en español, inglés, y un segundo idioma extranjero. La decisión del idioma a utilizar será tomada en tiempo de compilación.

Sugerencia: utilizar compilación adicional.

Ejercicio 6.46.

Idem para los días de la semana.

Ejercicio 6.47.

Entrega Obligatoria

Una matriz de nxm elementos es simétrica, si y solo si:

- es una matriz cuadrada (m = n), y
- $a_{ij} = a_{ji}$ para todo i, j =1,2,3,4,...,n.

donde a_{ij} representa el elemento que está en la fila *i*-ésima y en la columna *j*-ésima de *A*.

En base a lo anterior, escribir una función que reciba una matriz cuadrada de enteros y su dimensión, y determine si es una matriz simétrica retornando el resultado mediante una variable booleana.

Comprobar su correcto funcionamiento contrastando el resultado del algoritmo con matlab.

Ejercicio 6.48.

Entrega Obligatoria

En Álgebra Lineal, la traza de una matriz cuadrada de *nxn* se define como la suma de los elementos de su diagonal principal.

Es decir, $t(A)=a_{11} + a_{22} + a_{33} + ... + a_{nn}$.

En base a lo anterior escribir una función que reciba una matriz cuadrada de doubles y su dimensión, y retorne el valor de su traza.

Comprobar su correcto funcionamiento contrastando el resultado del algoritmo con matlab.

Ejercicio 6.49.

Entrega Obligatoria

Escribir una función que reciba una matriz cuadrada de doubles y su dimensión, y retorne el valor de su determinante.

Comprobar su correcto funcionamiento contrastando el resultado del algoritmo con matlab.

Ejercicio 6.50.

Escribir una función que recibe como argumentos una matriz de enteros de N x M, y los enteros N y M, y obtenga la transpuesta de dicha matriz.

Comprobar su correcto funcionamiento contrastando el resultado del algoritmo con matlab.

Ejercicio 6.51.

Entrega Obligatoria

Escribir una función que reciba dos matrices de dimensiones NxM y MxP, y cuatro enteros dando filas y columnas de cada matriz, y realice el producto de las dos matrices, devolviendo un puntero a la matriz resultado, y chequeando si las dimensiones de ambas matrices permiten efectuar el producto de las mismas.

Comprobar su correcto funcionamiento contrastando el resultado del algoritmo con matlab.

Ejercicio 6.52.

Escribir un programa que recibe como argumento un arreglo bidimensional de números enteros, y dos enteros que definen sus dimensiones, y devuelva un entero que represente el máximo valor contenido en el mismo.

Ejercicio 6.53.

Explicar el significado de la siguiente declaración:

```
char ( * ( * ufa ()) []) ();
```

Ejercicio 6.54.

a)Escribir una función que, respondiendo al siguiente prototipo:

```
status sort (void *[], size_t, ??????? );
```

ordene una tabla, conforme a un criterio de comparación de registros pasado como puntero a función sobre el tercer argumento.

b) Escribir el prototipo completo de la función.

c) Escribir una posible función de comparación, e invocar a la función (desarrollada en el punto a).

Ejercicio 6.55.

a) Escribir una función que, respondiendo al siguiente prototipo:

```
status print_plain_strings(FILE *, string[], size_t);
```

imprima las cadenas de caracteres pasadas como segundo argumento, sobre un stream de texto previamente abierto pasado como primer argumento.

b) Idem, pero imprimiendo el arreglo de cadenas en forma tabular con formato de página web (HTML).

c) Idem, pero imprimiendo el arreglo de cadenas sobre un archivo XML.

d) Si se define tres estrategias de impresión como plain, html y xml, correspondientes a las funciones de a), b) y c), escribir una función genérica:

```
status print_strings (FILE *, string[], size_t, status (*)(FILE *,string[],size_t));
```

que permita imprimir un arreglo de cadenas en múltiples formatos.

e) Definir un tipo enumerativo fmt con los símbolos: FMT_PLAIN_TEXT, FMT_HTML y FMT_XML.

f) Reescribir la función del punto d) si el nuevo prototipo es:

```
status print_strings2(FILE *, string[], size_t, fmt);
```

pero utilizando la construcción switch.

g) Comparar las implementaciones de los puntos d) y f).

h) Si se agregara en un futuro una nueva estrategia de impresión, ¿cómo resultaría el impacto en estas dos funciones? ¿Cuál es más conveniente?

i) Conformar un arreglo de punteros a función que contenga las múltiples estrategias de impresión.

j) Escribir un programa invocable en línea de órdenes que utilizando lo desarrollado en d), f) y g), imprima los argumentos en línea de órdenes en formato y archivo indicados en los argumentos.

Ejercicio 6.56.

El código secreto. La PC debe generar un código de cuatro cifras (de 0 a 9) que se generan en forma aleatoria. Las cuatro cifras deben ser distintas. El usuario debe adivinar dicho número, proponiendo un código de cuatro cifras por vez. Por cada intento, la PC debe informar cuantas cifras están bien (cifra correcta en la posición correcta) y cuantas regular (cifra correcta pero en posición incorrecta). Con esta información, el usuario debe adivinar el código elegido por la máquina en un máximo de 10 intentos.

Ejercicio 6.57.

Intente elaborar una estrategia que permita a la PC adivinar un código propuesto por Ud. Ahora es la PC quien propone el número, y Ud. quien informa la cantidad de cifras con bien y regular que arriesga la máquina.

Ejercicio 6.58.

Con los dos programas anteriores, cree un juego completo, donde se puedan jugar varios partidos contra la máquina.

T.P. Nº 7.Estructuras Uniones Campos de bits

Condiciones Generales para resolver el Trabajo Práctico.

Para el caso de los ejercicios que pidan únicamente escribir la función, éstos deben estar escritos en un archivo fuente que contenga solo la función pedida. Las definiciones que se requieran deben efectuarse en un archivo header que tenga el mismo nombre del fuente C.

El entregable es el conjunto de programas fuente (*.c y *.h), junto con un ***Makefile*** para ser construido el ejecutable mediante la orden ***make*** simplemente.

Objetivo:Las funciones desarrolladas en esta sección irán luego a librerías de código sumándose a las de los Trabajos Prácticos anteriores, y serán utilizadas en los próximos Trabajos Prácticos

Ejercicio 7.1.

Escriba una función que reciba dos valores enteros (dividendo y divisor) y que devuelva cociente y resto a través de una estructura

Ejercicio 7.2.

Escriba una función que reciba la hora con formato de 24 hs. y la devuelva en el formato AM/PM.

Ejercicio 7.3.

Ídem anterior, pero recibiendo la hora en formato AM/PM, y retornando en sistema de 24 hs.

Ejercicio 7.4.

Escriba una función que reciba un ángulo expresado en radianes y lo devuelva en formato sexagesimal.

Ejercicio 7.5.

Escriba un programa que imprima, byte a byte, los bytes que constituyen una variable long. Utilice Uniones para su implementación.

NOTA: Para apreciar mejor el resultado de este programa, se recomienda trabajar con formato hexadecimal

Ejercicio 7.6.

Escribir funciones que, mediante el uso de estructuras, realicen lo siguiente:

a) Convertir coordenadas rectangulares en polares

b) Convertir coordenadas polares en rectangulares

c) Realizar suma, resta, producto, división, potencia y radicación de números complejos

d) Agregar la posibilidad de expresar los ángulos en grados o radianes.

Ejercicio 7.7.

Entrega Obligatoria

Escribir un programa que cargue un vector de estructuras de tipo:

```
struct datos {
        long legajo;
        char apellido[31];
        char nombre[31];
        };
```

El ingreso de datos se hará en base a una función con el siguiente prototipo:

void carga(struct datos *);

Los datos se ingresan en un vector utilizando la función carga. La condición de fin es legajo = 0. Escriba para esto otra función, que reciba la estructura y devuelva 1 (uno) si se cumple la condición de fin, o un 0 (cero) en caso contrario. El prototipo es:

int fin(struct dato)

Una vez generado el vector, se deberá ordenarlo en forma creciente por apellido. Para ello, debe también utilizar una función para la cual Ud. debe proponer el prototipo. Finalmente, el vector ordenado debe ser guardado, registro por registro, en el archivo binario legajos.dbf, en el directorio raíz de una disquetera. Al utilizar las funciones propuestas, Ud. notará que, al ingresar legajo=0 para salir del programa, la función carga() lo fuerza a ingresar un apellido y nombre

que luego serán descartados. Sugiera una forma de solucionar esto, sin modificar el prototipo de la función carga().

Ejercicio 7.8.

Definir una estructura de datos que permita representar el Tiempo, es decir horas, minutos y segundos.

Desarrollar las funciones de manipulación del Tiempo tales como

a) CreaTime: genera un estructura del tipo Tiempo inicializado en 0 horas, 0 minutos y 0 segundos.

b) ShowTime: imprime las horas, minutos y segundos en el siguiente formato hh:mm:ss

c) SetTime: Recibe las horas minutos y segundos como argumentos y setea los valores en los campos de la estcrutura

Ejercicio 7.9.

Definir una estructura de datos que represente una fecha y el conjunto de funciones de manipulación de una fecha siguiente:

a) crearFecha: función que devuelve una fecha inicializada en alguna fecha particular,

b) mostrarFecha: función que imprime una fecha que recibe,

c) copiarFecha: función que recibe una fecha y devuelve la misma en otra variable,

d) sigDia: función que aumenta una fecha en un día,

e) diaAnt: función que disminuye una fecha en un día.

Ejercicio 7.10.

Construir una estructura de datos que represente a un automóvil. Un automóvil es una estructura que tiene los siguientes atributos:

- numeroDominio: el número de patente,
- numeroMotor: número que trae grabado el motor,
- marca: marca del automóvil
- modelo: el modelo de automóvil

- tamañoMotor: número que indica la potencia del motor.
- color: el color de la carrocería

Escribir las siguientes funciones para manipular un automóvil como el definido en el ejercicio anterior:

a) crearAuto: genera una instancia de tipo Automóvil con valores de inicialización adecuados para cada uno de sus atributos.

b) cambiarColor: modifica el valor del atributo color de un automóvil que recibe

c) mostrarAuto: imprime todos los atributos de un automóvil en particular

d) clonarAuto: es una función que reproduce un automóvil en otra variable de tipo Automóvil (función clone())

Ejercicio 7.11.

Escribir un programa que permita al usuario generar una lista de automóviles, los muestre ordenados según el número de dominio, permita modificar cualquiera de los atributos de un automóvil, ofrezca la posibilidad de agregar un nuevo automóvil a la lista, de eliminar un automóvil a la lista, permita determinar cuál es el automóvil de mayor potencia presente en la lista.

Ejercicio 7.12.

Definir un tipo de dato basado en estructuras que permita almacenar la información de una tarjeta de crédito:

Nº de tarjeta, nombre del titular, fecha de expiración.

Ejercicio 7.13.

En función de las estructuras ya definidas en ejercicios previos, definir un tipo de dato basado en estructuras anidadas que permita almacenar la siguiente información de un cliente:

- Nombre y apellido.
- ID de cliente.
- Fecha de alta.
- Fecha de última modificación.
- Domicilio particular.
- Domicilio laboral.

- Nº de tarjeta de crédito.

Ejercicio 7.14.

Idem para estructuras referenciadas.

Ejercicio 7.15.

Escribir un programa que permita manejar información sobre Estudiantes. Se necesita conocer el padrón (identificador único para cada Estudiante), cinco calificaciones en punto flotante, fecha de ingreso a la Facultad, Carrera en la que están inscriptos. Las funciones para manipular un Estudiante son:

a) crearEstudiante: genera un Estudiante con un número de padrón que recibe.

b) agregarCalificacion: agrega una nota de las cinco notas que puede tener un estudiante.

c) calcularPromedio: devuelve el promedio de las notas que pertenecen a un Estudiante.

Ejercicio 7.16.

Escribir un programa invocable por línea de órdenes que obtenga la fecha actual del sistema y la presente por stdout para los siguientes formatos:

a) AAAAMMDD

b) AAAAMMDD HhmmSS

c) HH:mm:SS del DD/MM/AAAA

Ayuda: usar la biblioteca <time.h>

Ejercicio 7.17.

Modificar el ejercicio anterior para que el formato de salida pueda ser informado al programa por medio de comandos en línea de órdenes (sysdate).

Ejercicio 7.18.

Escribir un programa invocable en línea de comandos que reciba dos fechas, y presente por stdout la diferencia entre ellas, en una unidad de tiempo indicada también como argumento en línea de comandos (timediff).

Ejercicio 7.19.

Entrega Obligatoria

Definir el Tipo de Dato "Dirección IP", que permita trabajar con direcciones IP de equipos de red que utilicen el protocolo TCP/IP.

El Tipo Abstracto Dirección IP puede ser definido como una colección finita y ordenada de cuatro elementos de tipo Octeto y un elemento de tipo Delimitador.

El tipo Octeto representa al conjunto de números enteros comprendidos en el intervalo [0 , 255].

El tipo Delimitador representa a un símbolo separador de elementos de tipo Octeto. Es único dentro de toda la colección, y es utilizado solo a los efectos de su representación textual.

Las posibles operaciones sobre los datos son:

- Crear una dirección IP.
- Destruir una dirección IP.
- Setear un Octeto determinado de la colección.
- Obtener un Octeto determinado de la colección.
- Setear el Delimitador de la colección.
- Obtener el Delimitador de la colección.
- Obtener una representación textual a partir de la dirección IP, como una sarta de símbolos Ci , con:

  Ci = { '0', '1', '2', '3', '4', '5', '6', '7', '8', '9', '.' }

En cuanto a la implementación, se desea disponer de funciones que respondan a los siguientes prototipos:

```
ip*          crear_direccion_ip ( octeto o1, octeto o2, octeto o3, octeto o4 );

void         destruir_direccion_ip ( ip );

ip*          establecer_delimitador_ip ( ip , delimitador );

delimitador  obtener_delimitador_ip ( ip );

octeto       obtener_octeto ( ip , int );

ip*          establecer_octeto_ip ( ip , octeto );

string       direccion_ip_a_string ( ip );
```

```
bool        es_local_host ( ip );

bool        es_broadcast ( ip );
```

Definir todos los tipos utilizados en la implementación del Tipo de Dato Dirección IP.

Encapsular toda la implementación de este Tipo de Dato en una biblioteca cuyo header esté contenido en un archivo denominado "ip.h".

Ejercicio 7.20.

Entrega Obligatoria

Definir un Tipo de Dato "Complex" que permita operar con números complejos. Implementar en una biblioteca, las funciones suma, resta, producto, cociente, y conjugado.

Ejercicio 7.21.

Entrega Obligatoria

Implementar un Tipo de Dato "Vector" que incorpore en forma segura y conveniente las funcionalidades de un arreglo estático estándar de punteros a datos, utilizando memoria dinámica.

Ejercicio 7.22.

Se tienen la siguiente estructura:

```
struct datos {
        long cod_art;
        int cantidad;
        char[26] descripción;
        };
```

Escriba un programa que realice lo siguiente:

- Cargue datos en memoria, correspondientes a dicha estructura (fin de datos: cod_art = 0)
- Ordene en forma creciente por cantidad, usando vector de punteros

Ejercicio 7.23.

Escribir una función que reciba los coeficientes de una función polinómica de 2do grado, expresada como $y = a.x^2 + b.x + c$ y devuelva las raíces a través de una estructura. Considere tanto raíces reales como imaginarias.

Ejercicio 7.24.

Escribir una función o programa que pida el ingreso de los coeficientes de una función polinómica de orden n y, a continuación, calcule sus raíces. Utilice el método que prefiera, aunque se sugieren lo siguientes: bisección, interpolación lineal, o método de Graeffe.

Intente que los coeficientes ingresen como argumentos de la función, implementando una función de parámetros variables.

Ejercicio 7.25.

Escriba un programa que implemente la integración de una función a través del método del trapecio.

Contemple la posibilidad de modificar la precisión del resultado.

Ejercicio 7.26.

Dado un conjunto de pares ordenados correspondientes a una función, escriba una función en C que estime la recta que mejor se adecue a los pares ordenados.

Notas: se trata de una aplicación elemental del método de aproximación de los cuadrados mínimos No se pide representación gráfica (aunque, si lo desea, ¡puede intentarlo!). Sólo se piden los coeficientes de la recta.

Ejercicio 7.27.

Dado un conjunto de pares ordenados correspondientes a una función, escriba una función en C que estime la parábola que mejor se adecue a los pares ordenados

Notas: se trata de una aplicación elemental del método de aproximación de los cuadrados mínimos No se pide representación gráfica (aunque, si lo desea, ¡puede intentarlo!). Sólo se piden los coeficientes de la parábola.

T.P. Nº 8. Estructuras complejas. Listas

Condiciones Generales para resolver el Trabajo Práctico.

Para el caso de los ejercicios que pidan únicamente escribir la función, éstos deben estar escritos en un archivo fuente que contenga solo la función pedida. Las definiciones que se requieran deben efectuarse en un archivo header que tenga el mismo nombre del fuente C.

El entregable es el conjunto de programas fuente (*.c y *.h), junto con un ***Makefile*** para ser construido el ejecutable mediante la orden ***make*** simplemente.

Objetivo:Las funciones desarrolladas en esta sección irán luego a librerías de código sumándose a las de los Trabajos Prácticos anteriores, y serán utilizadas en los próximos Trabajos Prácticos.

Ejercicio 8.1.

Utilizando las funciones malloc(), realloc() y free(), realice un programa en el que se ingrese cíclicamente por stdin la cantidad en bytes de memoria que se desea reservar o liberar. El programa deberá reservar o liberar de un solo bloque de memoria y preguntar por un nuevo valor. Si el valor es positivo, se reserva más memoria para ese mismo bloque. Si es negativo, se libera ese valor en bytes memoria. El programa debe finalizar (indicando lo sucedido) cuando:

- El valor ingresado es cero, indicando cuánta memoria ha quedado reservada antes de finalizar el programa.
- El resultado de reservar y liberar memoria equivale a cero.
- No hay suficiente espacio de memoria disponible para reservar.

Nota: Al finalizar el programa, no olvide liberar cualquier espacio de memoria que haya quedado reservado.

Ejercicio 8.2.

Para la siguiente estructura de datos:

```
struct sData {
        char Nombre[30];
        char Apellido[30];
        unsigned char Edad;
}
```

Crear una estructura **struct sNode** que permita agrupar muchos elementos del tipo struct sData en una lista simplemente enlazada.

Ejercicio 8.3.

Repetir el ejercicio anterior creando una estructura **struct dNode** para una lista doblemente enlazada.

Ejercicio 8.4.

Repetir el ejercicio anterior para una estructura **struct pNode** de lista del tipo pila. Si encuentra similitudes con algo hecho anteriormente, recuerde que puede crear nuevos tipos de variable con **typedef**, sin tener que definir nuevamente lo mismo.

Ejercicio 8.5.

Realizar una copia exacta (estructura y datos) de los parámetros recibidos por línea de comando en un espacio de memoria distinto. Se requiere inicializar y reservar los espacios de memoria, crear el puntero al vector de char pointers, llenar los punteros, copiar los datos. No debe en ningún caso reservar más memoria de la que necesita.

Una vez hecho esto debe presentar la cantidad de memoria empleada por stdout y esperar que el usuario pulse una tecla para salir del programa liberando antes la memoria utilizada.

Ejercicio 8.6.

Mejorar las estructuras de los ejercicios anteriores (sNode, dNode, pNode) para que acepten "cualquier" tipo de variable como datos del nodo (no confundir datos con los punteros que arman la lista).

Nota: Recuerde que un void pointer reserva un puntero para cualquier tipo de datos, y sólo hace falta realizar un cast en el momento de usarlo.

Ejercicio 8.7.

Utilizando la estructura sNode del ejercicio anterior, implemente las siguientes funciones:

void **s_insert**(struct sNode * List, void * Data);

void **s_delete**(struct sNode * List, struct sNode * Node);

La función s_insert agrega un elemento de Datos a la Lista indicada.

La función s_delete elimina el Nodo indicado de la Lista.

Nota: No olvide liberar la memoria reservada para ese nodo.

Ejercicio 8.8.

Utilizando la estructura dNode del ejercicio anterior, implemente las siguientes funciones:

void **d_insert** (struct dNode * List, void * Data);

void **d_delete**(struct dNode * Node);

La función d_insert agrega un elemento de Datos a la Lista indicada. Retorna el puntero al elemento resultante.

La función d_delete elimina el Nodo indicado.

Nota: No olvide liberar toda la memoria reservada para ese nodo.

Ejercicio 8.9.

Utilizando la estructura pNode del ejercicio anterior, implemente las siguientes funciones:

void **push**(struct pNode ** Stack, void * Data);

void * **pop**(struct pNode ** Stack);

La función push agrega un elemento de datos al tope de la pila, y modifica el puntero al stack para que apunte a ese último elemento (que es el nuevo comienzo de la lista)

La función pop retorna un puntero a los datos, y modifica el puntero al stack para redefinir el comienzo de la pila (que ahora tiene un elemento menos).

Nota: No olvide liberar la memoria correspondiente en este caso.

Ejercicio 8.10.

Se tiene la siguiente estructura de datos:

struct sData {int r1, r2; };

Se pide un programa que genere indefinidamente pares de números aleatorios r1 y r2 (entre 0 y 100, utilizando las funciones rand() y srand()). Los pares de datos deberán ser contenidos en una lista simplemente enlazada. La generación de pares de números finaliza cuando r1+r2 es menor a 30. Presentar todos los pares de números generados en stdout.

Ejercicio 8.11.

Para el ejercicio anterior, se desea agregar dos funciones: una que ordene la lista de pares de números por el valor de r1 (de menor a mayor), y otra para presentar la lista resultante por stdout.

Ejercicio 8.12.

Repetir los dos ejercicios anteriores con una lista doblemente enlazada.

Nota: Observe la diferencia de complejidad en el algoritmo de ordenamiento.

Ejercicio 8.13.

Utilizando las funciones de pila implementadas anteriormente, leer las líneas de un archivo de texto cualquiera, y guardar el contenido en reverso (la última línea al principio, la primer línea al final) en otro archivo. El tamaño máximo de cada línea es 255 caracteres. Se desconoce cuántas líneas en total tiene el archivo.

Ejercicio 8.14.

Repetir el ejercicio anterior sólo para las líneas que comiencen con la letra "A" (mayúscula o minúscula). El resto de las líneas pueden descartarse.

Ejercicio 8.15.

Leer el contenido de las líneas de un archivo de texto, cargarlas en un vector de punteros a char (similar al de parámetros del main), ordenar las líneas de forma ascendente, y guardar el archivo ordenado. El tamaño máximo de cada línea es 255 caracteres. Se desconoce cuántas líneas en total tiene el archivo.

Ejercicio 8.16.

Se dispone de un archivo de datos (partidas.db) que almacena la cantidad de productos que entregaron ciertos fabricantes en cada partida, con el siguiente formato separado por espacios, un registro por línea.

COD.FABRICANTE NRO.PARTIDA CANTIDAD

Donde:

- COD.FABRICANTE: char [20], sin espacios
- NRO.PARTIDA: char [10], sin espacios
- CANTIDAD: unsigned char

Se pide procesar el archivo y obtener el siguiente resultado (resultado.db)

COD.FABRICANTE CANT.PARTIDAS CANT.TOTAL

En base al siguiente criterio

- CANT.PARTIDAS es la cantidad total de partidas para cada fabricante.

- CANT.TOTALes la cantidad total de productos que entregó el fabricante en todas las partidas.
- Se desconoce cuántos registros tiene el archivo de entrada. Puede tener varios registros para un mismo fabricante, variando solamente el nro de partida y la cantidad.

Ejercicio 8.17.

Implemente una función **s_count** que devuelva la cantidad de nodos existentes en una lista simplemente enlazada. Realice la misma función **d_count** para una lista doblemente enlazada. Reutilice tódo el código que pueda.

Ejercicio 8.18.

Basándose en las funciones s_insert y s_delete creadas anteriormente, implemente **c_insert** y **c_delete** para crear una lista del tipo **circular**, en la que el último nodo apuntará al primero como elemento siguiente.

Ejercicio 8.19.

Basándose en las funciones realizadas anteriormente, implemente una función **c_count** que cuente los nodos de una lista circular.

Ejercicio 8.20.

Entrega Obligatoria

Se dispone de un archivo de texto (datos.db) el cual contiene un número indeterminado de líneas de longitud máxima 255 caracters. Se desconoce la longitud total del archivo. Se pide:

- Leer todas las líneas del archivo y colocar cada una en un nodo de una lista circular.
- Avanzar un número aleatorio de posiciones en la lista utilizando al función rand(). Al ser circular, no importa la cantidad de elementos de la lista, pues al llegar al final se vuelve a comenzar desde el principio.
- Elminar el nodo de la posición resultante.
- Imprimir la lista resultante.
- Hacer una pausa y repetir desde el paso 2 hasta que la lista esté vacía.

Ejercicio 8.21.

Se dispone de dos archivos de texto (data1.db y data2.db) que almacena los movimientos de una cuenta bancaria, con el siguiente formato por cada línea:

FECHA MONTO

- El formato de **fecha** es un Unix Timestamp (la cantidad de segundos que pasaron desde el 01/01/1970). El tipo es unsigned int.
- El **monto** puede ser positivo (en el caso de un crédito en la cuenta) o negativo (en el caso de un débito de la cuenta).

Se pide **consolidar** esos dos archivos en uno solo (resultados.db) con el siguiente formato:

FECHA DEBITO CREDITO SALDO

Con el siguiente criterio:

- El orden deberá respetarse de menor a mayor (la fecha más antigua va primero)
- Para una misma fecha exacta, sumar los débitos y los créditos respectivamente.
- En el archivo de resultados, tanto débito como crédito deben ser números positivos.
- El saldo inicial es cero.
- El saldo es el resultado del saldo de la fecha inmediata anterior, más los créditos, menos los débitos.
- Se desconoce la cantidad de registros que poseen los dos archivos de entrada.

Ejercicio 8.22.

Realizar una función **s_copy** que reciba un puntero a una lista simplemente enlazada, y realice una copia de la misma en memoria. La copia no debe tener nada en común con la lista original, salvo los datos, por lo tanto debe crear nuevos punteros, nodos y espacios de memoria.

Al finalizar, devuelve el puntero a la nueva lista. Puede hacer uso de las funciones ya desarrolladas que crea convenientes.

Ejercicio 8.23.

Repetir el ejercicio anterior para función **d_copy** que cree una copia de una lista doblemente enlazada. Puede hacer uso de las funciones ya desarrolladas que crea convenientes.

Ejercicio 8.24.

Implementar las siguientes funciones que deben unir dos listas simplemente o doblemente enlazadas, respectivamente.

void s_join(struct sNode *, struct sNode *);

void d_join(struct dNode *, struct dNode *);

Ejercicio 8.25.

Implementar las siguientes funciones que deben separar una lista (simplemente o doblemente enlazada, respectivamente) en dos listas distintas que comenzarán a partir de cada uno de los elementos pasados como parámetros de la función.

void s_split(struct sNode *, struct sNode *);

void d_split(struct dNode *, struct dNode *);

Ejercicio 8.26.

Implementar una función que realice el intercambio (swap) de dos nodos de una lista simplemente enlazada.

void s_swap(struct sNode * List, struct sNode * node1, struct sNode * node2);

Ejercicio 8.27.

Leer todas las líneas de un archivo de texto cualquiera, almacenarlas en la lista simplemente enlazada, ordenarlas alfabéticamente, dividir la lista a la mitad y guardar cada lista resultante en dos archivos distintos. Convenientemente, deberá hacer uso de todas las funciones desarrolladas anteriormente que crea necesarias.

Ejercicio 8.28.

Realizar una función que reciba un puntero a una lista simplemente enlazada y convierta esa lista en doblemente enlazada, devolviendo el puntero a la nueva lista. Debe liberar toda la memoria que sea necesario, ya que esta es una operación de conversión, no de copia. Todo lo relacionado a la lista simplemente enlazada original debe eliminarse.

Ejercicio 8.29.

Entrega Obligatoria

Compile todas las funciones desarrolladas para manipulación de listas simple y doblemente enlazadas y pila, en Ejercicio 8.7., Ejercicio 8.8., Ejercicio 8.9., Ejercicio 8.17., Ejercicio 8.18., Ejercicio 8.19., Ejercicio 8.22., Ejercicio 8.23., Ejercicio 8.24., Ejercicio 8.25.,y , en una biblioteca.

Escriba un programa genérico para probarlas mostrando claramente el resultado en stdout.

Sugerencias: Los lotes de prueba pueden estar en sendos archivos lote***xx***.db, en donde ***xx*** identifica a cada tipo de lote para la prueba a realizar, los cuales se redireccionan a stdin al momento de probar cada función determinada. Tenga en cuenta liberar la memoria requerida a medida que no la utiliza. Use redirección de stdout si la longitud de la presentación excede la cantidad de datos a mostrar por stdout.

T.P. Nº 9. Ejercicios Integradores 1º Nivel

Condiciones Generales para resolver el Trabajo Práctico.

Para el caso de los ejercicios que pidan únicamente escribir la función, éstos deben estar escritos en un archivo fuente que contenga solo la función pedida. Las definiciones que se requieran deben efectuarse en un archivo header que tenga el mismo nombre del fuente C.

El entregable es el conjunto de programas fuente (*.c y *.h), junto con un ***Makefile*** para ser construido el ejecutable mediante la orden ***make*** simplemente.

Objetivo:Las funciones desarrolladas en esta sección irán luego a librerías de código sumándose a las de los Trabajos Prácticos anteriores, y serán utilizadas en los próximos Trabajos Prácticos.

Consideraciones adicionales: Buena parte del TP se realizará con apoyo de una biblioteca externa para manipulación de imágenes y video: OpenCV. El objetivo es utilizar este recurso para acceder a los archivos de imagenes en cualquier formato, o a un archivo de video avi, o a la webcam, de manera simple. Una vez hecho ésto, manipular la información de manera directa accediendo a la estructura IplImage en donde dejan las funciones de la biblioteca el bitmap del archivo de imagen, o el bitmap de un frame de video, según corresponda.

Ejercicio 9.1.

Utilice la función cvLoadImage de la librería OpenCV, para abrir archivos con imágenes. Mediante el uso de esta función, manipule los miembros la librería IplImage y presente en stdout los siguientes atributos de la imagen

- Dimensiones en pixeles (ancho por alto)
- Profundidad en bits de color

Ejercicio 9.2.

Modifique el programa anterior para que presente en una ventana la imagen original y la imagen invertida. Utilice las funciones cvCreateImage y cvShowImage, para crear y mostrar la imagen, cvCloneImage para copiar la original y luego manipule IplImage->imageData para invertir los pixeles.

El algoritmo es:

IplImage->imageData = 255 - IplImage->imageData

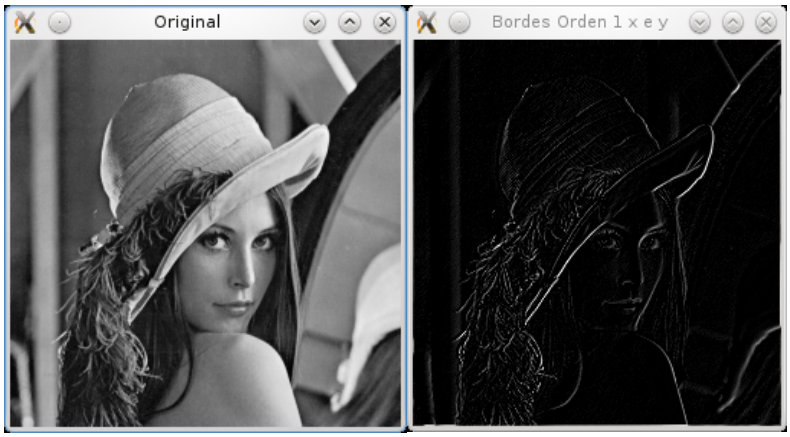
Para salir esperará que el usuario pulse una tecla.

Ejercicio 9.3.

Utilice la función cvCaptureFromCAM0 para modificar el ejemplo anterior a fin de mostrar la imagen de una webcam en negativo.

Ejercicio 9.4.

Con las funciones de manipulación de imágenes de OpenCV ya descriptas, abra un archivo de una imagen cualquiera, preséntela en una ventana. Cree otra ventana que se ubique en la posición 100,100 de la pantalla, y en ella presente una imagen que solo tenga los bordes de la imagen original.



Una imagen es una matriz de puntos, que representa los valores de la función de iluminación respecto de un espacio de dos dimensiones. Un borde es una variación fuerte de color entre un pixel y sus vecinos. ¿Que operación representa la velocidad de variación de una función? La derivada!!

Para obtener los bordes deberá evaluar la derivada de cada pixel respecto del de su derecha y del que tiene inmediatamente debajo. Establezca un valor de umbral para la derivada y si al menos una de las dos derivadas supera dicho umbral el pixel es un borde. En tal caso se reemplaza por 0xFF. En cualquier otro caso se lo reemplaza por 0x00.

Sugerencia: Una imagen color tiene por cada pixel cuatro bytes consecutivos. IplImage->imageData los guarda como Azul Verde Rojo y AlfaChannel (si la imagen lo tiene). Deberá calcular las derivadas en cada color.

Ejercicio 9.5.

Aplique el mismo algoritmo para una secuencia de video, que provenga de un archivo avi. Utilice para capturar los frames cvCaptureFromAVI.

Ejercicio 9.6.

Se obtienen por stdin diversos valores correspondientes a una curva de respuesta de un sistema de control de velocidad de un motor.

Se requiere calcular la mejor curva de interpolación posible empleando el método de cuadrados mínimos. El resultado debe ser una secuencia de coordenadas x,y mucho que sea 20 veces mas precisa que la recibida como mínimo.

Utilizando la función *cvPolyLine* de la biblioteca Opencv dibuje la curva interpolada, junto con los puntos originales recibidos para apreciar la interpolación.

Los puntos pueden ser dibujados como un pequeño rectángulo de dimensiones mínimas mediante la función *cvReactangle* de la biblioteca OpenCV.

Ejercicio 9.7.

Escriba un algoritmo que reciba como argumento una string terminada en NULL, que corresponde a una función, y dos números reales que representen los extremos de un intervalo, y calcule sus raíces, aplicando un algoritmo basado en el Método de la bisección.

Ejercicio 9.8.

Repita el ejercicio anterior, pero aplicando el Método de las aproximaciones sucesivas.

Ejercicio 9.9.

Repita el ejercicio anterior, pero aplicando el Método de Newton.

Ejercicio 9.10.

Repita el ejercicio anterior, pero aplicando el Método de la secante.

Ejercicio 9.11.

Repita el ejercicio anterior, pero aplicando el Método de Steffensen

Ejercicio 9.12.

Repita el ejercicio anterior, pero aplicando el Método de la falsa posición

Ejercicio 9.13.

Escriba un algoritmo que reciba una matriz de 2 x M como argumento, en donde las filas corresponden a la variable independiente y a los valores de una función de esa variable en un intervalo dado. El algoritmo debe:

a. Calcular la función de interpolación por medio del polinomio de Lagrange

b. Por medio de la función OpenCV *cvRectangle* dibuje punto a punto la función interpolada con la máxima precisión que le sea posible, y dibuje los puntos de la matriz en otro color a modo de referencia.

Ejercicio 9.14.

Repita el ejercicio anterior utilizando con el polinomio de Newton.

Ejercicio 9.15.

Repita el ejercicio anterior utilizando el polinomio de Hermite.

Ejercicio 9.16.

Escriba un algoritmo que reciba como argumento una string terminada en NULL, que corresponde a una función, y dos números reales que representen los extremos de un intervalo, y calcule su integral entre esos dos puntos. Aplique el método de interpolación polinomial.

Ejercicio 9.17.

Repita el ejercicio anterior aplicando la regla del trapecio.

Ejercicio 9.18.

Repita el ejercicio anterior aplicando la regla de Simpson.

T.P. Nº 10.Operadores a nivel de Bits y ports.

Condiciones Generales para resolver el Trabajo Práctico.

Para el caso de los ejercicios que pidan únicamente escribir la función, éstos deben estar escritos en un archivo fuente que contenga solo la función pedida. Las definiciones que se requieran deben efectuarse en un archivo header que tenga el mismo nombre del fuente C.

El entregable es el conjunto de programas fuente (*.c y *.h), junto con un ***Makefile*** para ser construido el ejecutable mediante la orden ***make*** simplemente.

Objetivo:Las funciones desarrolladas en esta sección irán luego a librerías de código sumándose a las de los Trabajos Prácticos anteriores, y serán utilizadas en los próximos Trabajos Prácticos.

Por otra parte para aquellos ejercicios que pidan acceso a puertos de E/S recordar que el mismo se puede realizar mediante la invocación previa a ioperm para su habilitación, y una vez finalizados los acceso a los ports, invocar ioperm nuevamente nuevamente para su deshabilitación. Por otra parte los programas involucrados deben ejecutarse con privilegios de root. Utilice el comando sudo para tal fin.

Ejercicio 10.1.

Escribir una función que reciba un carácter y lo devuelva negando los bis que estén en orden par (los impares no se modifican).

Ejercicio 10.2.

Entrega Obligatoria

Escribir una función que reciba un carácter y lo devuelva con sus bits invertidos (El MSB pasa a ser el LSB y viceversa).

Ejercicio 10.3.

Repita el escribiendo la función utilizando recursividad.

Ejercicio 10.4.

Se dispone de un hardware externo que nos provee información por un puerto de un byte mapeado en la dirección de E/S 0x300. Escribir un programa que lea

dicho port y calcule el promedio de todos los valores leídos que presenten paridad par. Se sabe además que el port actualiza sus datos cada 10 mseg.

Duerma al proceso sin consumir CPU cada vez que no necesite acceder.

El programa finaliza con CTRL-C.

Ejercicio 10.5.

Se dispone de un hardware externo que nos entrega la información recogida por él en el port 0x303. Escribir un programa que explore el puerto continuamente separando las muestras al menos 70 milisegundos y que se detenga cuando se reciban 4 0xFF consecutivos. Cuando se detenga la exploración se debe informar por stdout el valor mínimo, el valor máximo y el promedio de los valores leídos.

Ejercicio 10.6.

Se pide escribir un programa que muestre en pantalla la frecuencia cardiaca de un paciente, expresada en pulsaciones/minuto. La señal cardiaca proveniente del medidor ingresa a través del port 0x310, que toma el valor 0xFF cada vez que se produce un latido. Permanece en este estado durante 10 mseg, y vuelve a 0x00, en espera del siguiente latido. La frecuencia cardíaca debe calcularse como el promedio de las 8 últimas mediciones, y actualizarse en pantalla cada 5 segundos.

Ejercicio 10.7.

En una PC se reciben tramas desde otra PC remota conectada al puerto 0x297, con la siguiente información:

- N datos de tipo real (doble precisión)
- Como último elemento de cada trama se recibe un carácter ASCII con la operación a realizar (´S´= sumatoria, ´P´= promedio, ´M´= máximo)

- La información proveniente del puerto 0x297 se recibe byte a byte y cada uno de estos bytes deberá almacenarse en un vector en el orden en que llega, incluyendo la operación a realizar.

- Por el puerto 0x298, se recibe un byte con un 1 en el bit más alto, los 7 bits restantes indican el total de los N datos a recibir en esa trama. Este byte indica cuándo llegó una trama de datos y el byte permanecerá sin alteración hasta que no haya mas datos a recibir, cambiando automáticamente el bit mas alto a cero al finalizar la recepción del ultimo byte de la trama.

- Finalizada la recepción de los valores de cada trama, deberá invocarse una función que realice la operación determinada con ellos (suma, promedio u

obtención del máximo), a la cual se le pasara como parámetro el puntero a los datos.

- El cálculo de la sumatoria, el promedio y el máximo se implementará mediante funciones. - El resultado de la operación (un long double) se deberá reenviar por el puerto 0x299 mediante una función que utilice punteros.

- Lo descrito anteriormente se realizará continuamente sin detener en ningún momento el proceso de recepción y proceso de los datos hasta que se presione la tecla "Q" (mayúscula).

Notas: No se permite el uso de variables globales. Las funciones devuelven el valor del resultado (de la sumatoria, promedio o máximo). El vector de recepción se deberá dimensionar para recibir el total de los datos para el peor de los casos posibles junto con la operación a realizar. Este vector se reutilizará cada vez que llegue una nueva partida de datos. Asimismo, el vector podrá formar parte de un tipo de dato creado por el programador, para facilitar la resolución del programa

Ejercicio 10.8.

Para un programa que hará uso del puerto 0x310 de una PC se necesita disponer de funciones que permitan colocar a nivel lógico "1" ó "0" un bit determinado dentro de un byte a ser presentado por ese puerto. Se pide:

a) Escribir una función llamada "Set()", que obedezca al siguiente prototipo:

```
unsigned char Set( unsigned char Datos, int Linea);
```

que reciba un byte sobre la variable "Datos" y el número de línea que será forzada a nivel lógico "1", devolviendo por el nombre el resultado de la operación.

Ejemplo: Si en "Datos" se recibe 1000 0001 y Linea=3, se debe retornar el byte 1000 1001

b) Dar un ejemplo de invocación de la función.

c) Idem a) pero para una función llamada "Clear()" que coloque a nivel lógico "0" la línea indicada, y devuelva por el nombre el resultado de la operación. Ejemplo: Si en "Datos" se recibe 111 1100 y línea = 3, se debe devolver el byte

111 0 111.

d) Dar un ejemplo de invocación de la función. ACLARACION: La línea 0 se corresponde con el bit menos significativo (LSB) del byte, y la linea 7 con el bit más significativo (MSB) del byte.

Ejercicio 10.9.

Escribir una función de acuerdo al siguiente prototipo:

int beep (int tiempo);

tiempo es un valor medible en segundos, e indica durante el tiempo que debe dejarse sonar el parlante de la PC de acuerdo a como esté programado.

Debe setear los dos bits menos significativos del port 0x61 para activar el parlante (comienza a sonar el "beep") y resetear el bit menos significativo para apagarlo (deja de sonar el "beep").

Si la función activa exitosamente el parlante retorna 0, de otro modo un valor positivo.

Ejercicio 10.10.

Entrega Obligatoria

Escribir una función de acuerdo al siguiente prototipo:

int nota (int octava, int nota, int tempo);

Octava, va de 0 a 7 (de acuerdo a las octavas de un piano).

Nota es un valor correspondiente a una constante de enumeración en donde se define un valor para cada nota de la primer columna de la tabla adjunta

Tempo es la duración en mseg. de la nota musical enviada al timer de sonido de la PC.

En la siguiente tabla se tiene la frecuencia en Hertz, que corresponde a cada nota de cada octava.

	Octava 0	Octava 1	Octava 2	Octava 3	Octava 4	Octava 5	Octava 6	Octava 7
DO	16,35	32,70	65,41	130,81	261,63	523,25	1046,50	2093,00
DO#	17,32	34,65	69,30	138,59	277,18	554,37	1108,74	2217,46
RE	18,35	36,71	73,42	146,83	293,66	587,33	1174,66	2349,32
RE#	19,45	38,89	77,78	155,56	311,13	622,25	1244,51	2489,02
MI	20,60	41,20	82,41	164,81	329,63	659,26	1328,51	2637,02
FA	21,83	43,65	87,31	174,61	349,23	698,46	1396,91	2793,83
FA#	23,12	46,25	92,50	185,00	369,99	739,99	1479,98	2959,96
SOL	24,50	49,00	98,00	196,00	392,00	783,99	1567,98	3135,96
SOL#	25,96	51,91	103,83	207,65	415,30	830,61	1661,22	3322,44
LA	27,50	55,00	110,00	220,00	440,00	880,00	1760,00	3520,00
LA#	29,14	58,27	116,54	233,08	466,16	923,33	1864,66	3729,31
SI	30,87	61,74	123,47	246,94	493,88	987,77	1975,53	3951,07

Tener en cuenta que la fórmula de conversión de frecuencias viene dada por la siguiente expresión que contempla la frecuencia de oscilación del cristal piezoeléctrico conectado a la entrada del PIT (Programmable Interval Timer)

cuenta_timer = 1.193.180 / frecuencia.

El valor a programar para la nota es un valor de 16 bits que se escribirá en el port 0x42, a razón de un byte a la vez comenzando por el byte menos significativo.

T.P. Nº 11. Streams

Condiciones Generales para resolver el Trabajo Práctico.

Para el caso de los ejercicios que pidan únicamente escribir la función, éstos deben estar escritos en un archivo fuente que contenga solo la función pedida. Las definiciones que se requieran deben efectuarse en un archivo header que tenga el mismo nombre del fuente C.

El entregable es el conjunto de programas fuente (*.c y *.h), junto con un ***Makefile*** para ser construido el ejecutable mediante la orden ***make*** simplemente.

Objetivo:Las funciones desarrolladas en esta sección irán luego a librerías de código sumándose a las de los Trabajos Prácticos anteriores, y serán utilizadas en los próximos Trabajos Prácticos

Ejercicio 11.1.

Escriba un programa que lea el archivo de texto que recibe por línea de comandos y determine:

- Cantidad total de palabras
- Cantidad de veces que aparece la palabra "diodo"

Ejercicio 11.2.

Escribir un programa que genere un archivo de texto con los caracteres que el operador ingrese por stdin.

Sugerencia: Revise que secuencia de teclas representa EOF para stdin.

Ejercicio 11.3.

El archivo binario ~/info1/BIPOLAR.DAT contiene una rudimentaria base de datos de transistores bipolares. La base tiene registros de cinco campos:

- Nombre del transistor: alfabético – 10 caracteres
- Tipo: alfabético de tres caracteres
- hFE: valores enteros entre 50 y 420
- Potencia: valores reales entre 10mW y 12W

- Tensión de ruptura: valores enteros entre -130V y +130V

Se pide un programa que permita realizar consultas a la base de datos. Luego, el programa pedirá el nombre del transistor, y toda la información relevante del mismo se informará por pantalla.

La búsqueda debe hacerse en forma binaria, mediante una función con el siguiente prototipo:

struct TR buscar (struct TR * , char *, int);

NOTA: struct TR es el tipo correspondiente a la estructura que UD. utilizará para organizar la información en el archivo. Si le asigna otro nombre téngase en cuenta en el prototipo.

Ejercicio 11.4.

Escriba un programa que permita copiar archivos de cualquier tipo. Llámelo ***copy***. Recibe por línea de comandos el path del archivo fuente en primer lugar y luego separado por uno o mas espacios o tabs, el path de destino. Compruebe que ambos argumentos están disponibles, y de otro modo no prosiga con la copia e informe al usuario el modo correcto de invocación de ***copy***.

Uso: copy <path archivo origen> <path archivo destino>

Ejercicio 11.5.

Cuando es necesario enviar muchos parámetros por línea de comandos, generalmente es más cómodo contenerlos en un archivo de configuración. Realizar la siguiente función:

```
int read_parameters( char * filename, char ** );
```

La función recibe el nombre del archivo que contiene todos los parámetros (uno por cada línea), y un char ** donde se cargará la dirección inicial del vector de punteros (“argv”). El valor de retorno será la cantidad de parámetros leídos del archivo (“argc”).

Al retornar de la función, el uso de esos parámetros (mediante argc y argv) deberá poderse realizar de la misma forma que al leer los parámetros de línea de comandos.

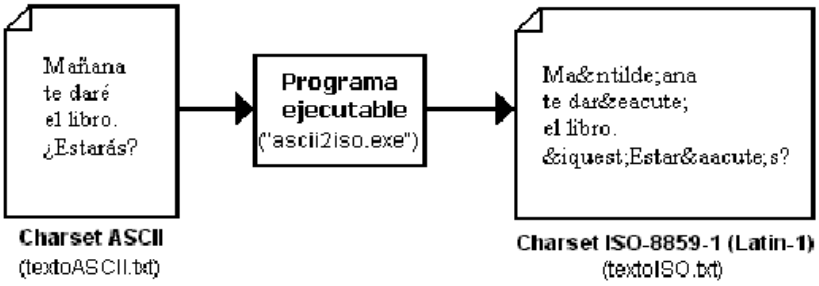
Ejercicio 11.6.

Escribir un programa que lea una cadena de caracteres ingresada por stdin y la imprima por stdout. Para ello utilizar las funciones de biblioteca scanf() con formato "%s", gets() y fgets(). Comparar los resultados y explicar la conveniencia de utilizar cada una de ellas, y bajo qué circunstancias.

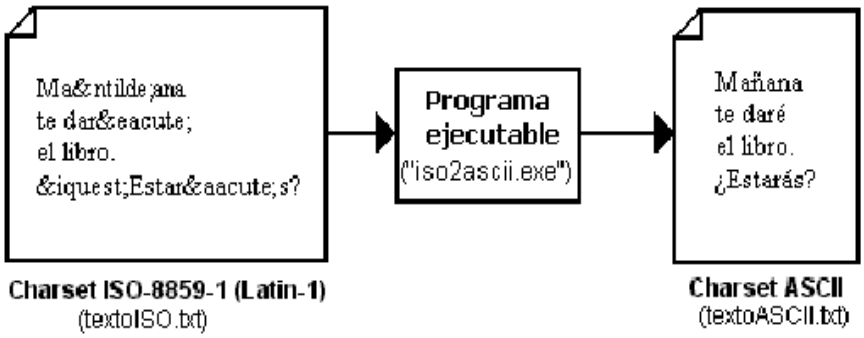
Ejercicio 11.7.

Construir los siguientes dos (2) programas:

Encoder: convertirá un texto compuesto por símbolos del alfabeto ASCII (caracteres) a otro idéntico en contenidos pero compuesto por símbolos del alfabeto ISO-8859-1.



Decoder: convertirá un texto compuesto por símbolos del alfabeto ISO-8859-1 a otro idéntico en contenidos pero compuesto por símbolos (caracteres) del alfabeto ASCII.



El programa Encoder, materializado sobre un archivo ejecutable denominado "ascii2iso" deberá ser invocado de la siguiente forma:

cat textoASCII.txt | ascii2iso>textoISO.txt

El programa Decoder, materializado sobre un archivo ejecutable denominado "iso2ascii" deberá ser invocado de la siguiente forma:

cat textoISO.txt | iso2ascii>textoASCII.txt

en donde "textoISO.txt" y "textoSCII.txt" son dos archivos de texto usados como ejemplo, conteniendo símbolos de los alfabetos ISO-8859-1 y ASCII, respectivamente.

Ejercicio 11.8.

Entrega Obligatoria

Escribir un programa que utilizando las funciones ya desarrolladas e incluidas en una librería, convierta un archivo de texto a minúsculas o mayúsculas, dependiendo de la selección hecha por el usuario:

Uso: chgcase <path archivo origen> <path archivo destino> <formato>

en donde el formato puede ser:

-u : a mayúsculas (UPPERCASE)

-l : a minúsculas (LOWERCASE)

Ejercicio 11.9.

Escribir un programa de comando en línea de órdenes que cuente la cantidad de líneas de un archivo de texto. Consultar la documentación del comando wc de Linux en las man pages, en particular para la opción: wc -l .

Ejercicio 11.10.

Escribir una función permita imprimir datos directamente sobre la impresora, a través del stream stdprn.

Ejercicio 11.11.

Escribir un programa que permita numerar las líneas de un archivo de texto:

Uso: numerator <numero máximo> <archivo de salida>

Ejercicio 11.12.

Escribir un programa tal que lea cadenas con gets() y con fputs() las escriba en un archivo de nombre “prueba”. Para que finalice el programa, hay que ingresar una línea en blanco.

Ejercicio 11.13.

Escriba un programa que lea todas y cada una de las cadenas generadas por el programa del ejercicio anterior.

Ejercicio 11.14.

Escriba un programa que reemplace por una tabulación grupos de entre 2 y 5 espacios consecutivos en un archivo.

Ejercicio 11.15.

Escribir un programa de comandos en línea de órdenes que genere un archivo de texto con información de clientes morosos (saldo negativo) contenida en un archivo (entrada). El archivo de entrada contiene una secuencia de estructuras de longitud desconocida, del siguiente tipo:

```
typedef struct {
                    char calle[MAX_LARGO];
                    int numero;
                    int piso,
                    char depto;
                    } StDomicilio;
typedef struct {
                    char CBU[MAX_LARGO];
                    double saldo;
                    char categoria;
                    } StCuentaBancaria;
typedef struct {
                    char razonsocial [MAX_LARGO];
                    StDomicilio domicilio;
                    StCuentaBancaria cuenta;
                    }StCliente;
```

El programa debe ser invocado como:

morosos <archivo de datos> <archivo a generar>

y el formato de una línea del archivo de texto a generar debe ser el siguiente, con valores delimitados por pipes ("|"):

<Razon Social> | <CBU> | <Saldo> \n

Ejemplo:

Juan Perez|123456789|-43.50

María Fernandez|258654|-7.23

José Bianchi|124674881|-192.47

Ejercicio 11.16.

Entrega Obligatoria

Escribir un programa de comandos en línea de órdenes que permita extraer un campo contenido en un archivo de texto de tipo CSV (valores separados por comas), que sea invocado como:

extract <archivo de entrada> <número de campo> <archivo de salida>

Ejemplo: Si el archivo de entrada, "in.txt" está compuesto de los siguientes campos:

Juan, Perez, Viamonte 1566 3ºH, 4559-9281

Ana, Solís, Argerich 3144 PB, 4583-8567

al invocar al programa como:

extract in.txt 3 out.txt

se habrá de generar un segundo archivo de texto, "out.txt", con el siguiente contenido:

Viamonte 1566 3ºH

Argerich 3144 PB

Se debe validar los argumentos del programa e informar al usuario en caso de error. Debe contemplarse los casos en que el archivo esté vacío, y el que no exista el campo seleccionado.

Nota: Para la resolución del ejercicio puede suponerse que cada renglón del archivo de texto no contiene más de 16K caracteres, incluidos todos los delimitadores y el retorno de carro.

Ejercicio 11.17.

Escribir un programa que permita generar un lote de números aleatorios sobre un archivo de texto, de acuerdo a la siguiente forma de uso:

random <cantidad> <mínimo> <máximo> <archivo destino>

en donde <cantidad> es el tamaño del lote, <mínimo> y <máximo> los extremos del intervalo al que pertenecen los números generados, y <archivo destino> el archivo destino donde han de volcarse las muestras.

Ejercicio 11.18.

Escribir un programa de comando en línea de órdenes que genere un archivo CSV con las muestras de una función senoidal de la forma:

$$S(t) = A \cdot \sin(2 * \Pi * f * t + \varphi)$$

La invocación del programa debe responder a:

sample <forma > <cantidad> <amplitud> <frecuencia> <inicio> <fin> <fase> <archivo> <delimitador>

en donde:

<forma>	forma de onda, cadena literal "SENOIDAL".
<cantidad>	Cantidad de muestras a generar.
<amplitud>	Amplitud de pico de la senoidal.
<frecuencia>	frecuencia de la senoidal.
<inicio>	Valor inicial de la variable independiente.
<fin>	Valor final de la variable independiente.
<archivo>	Path del archivo CVS de destino.
<delimitador>	Secuencia de caracteres delimitadores (ej. |, "|", <coma>, <TAB>, etc.).

Comprobar el contenido del archivo resultante con una hoja de cálculo (importación de archivos CSV).

Ejercicio 11.19.

Escribir un programa que convierta un archivo de texto de tipo CSV (comma-separated values) a formato HTML.

Uso: csv2html <input file> <output file> <delimitter> <web page title>

En donde <delimitter> es el carácter delimitador del arvchivo CSV y <web page title> el título a darle a la página HTML resultante. La idea es visualizar los campos del archivo como una tabla de datos.

Sugerencia: Lea un simple fuente de HTML que contenga una tabla para ver su estructura.

Ejercicio 11.20.

Escribir un programa que lee un archivo de texto e imprime en pantalla las primeras n líneas que recibe en la línea de comandos. Si n es mayor que el número de líneas presentes en el archivo de entrada, imprime hasta la última que encuentre en el archivo de entrada.

Ejercicio 11.21.

Escribir un programa similar al del ejercicio anterior, que imprime las últimas n líneas leídas desde el archivo de entrada.

Nota: Comparar este ejercicio y el anterior con los comandos head y tail de Linux (consultar las man pages)

Ejercicio 11.22.

Escribir un programa que concatena dos archivos de texto cuyos nombres recibe desde la línea de comandos.

Ejercicio 11.23.

Escribir un programa que busque una cadena de caracteres que recibe desde la línea de comandos, en un archivo de texto y determine si esa cadena existe o no. En caso de encontrarse dicha cadena, también deberá informar en qué línea se encuentra.

Ejercicio 11.24.

Documente los siguientes prototipos de funciones, indicando el significado de los parámetros formales, del tipo de dato que devuelve y cómo se invoca:

fgetc(), ungetc(), puts(), fseek(), ftell(), freopen(), fcloseall(), y rewind().

Nota: Mientras ya estará abriendo las man pages para copiar y pegar, cumplimos en aclarar que este TP requiere que lo explique con sus propias palabras.

Ejercicio 11.25.

Un archivo llamado "polar.dat" contiene las coordenadas polares necesarias en un programa de gráficas. Actualmente este archivo contiene los siguientes datos:

Distancia (en cm.)	Ángulo (grados)
2,00	45.0
6,00	30.0
10,00	45.0
4,00	60.0
12,00	55.0
8,00	15

Escribir un programa que cree este archivo.

Ejercicio 11.26.

Escribir un programa que lea el archivo creado en el ejercicio anterior y cree otro de nombre "xycord.dat". Las entradas al nuevo archivo deben contener las coordenadas rectangulares que corresponden a las coordenadas polares en el archivo "polar.dat".

Las coordenadas polares se convierten a rectangulares mediante las ecuaciones:

x = r*coseno(t)

y = r*seno(t)

donde r es la coordenada de distancia y t es el equivalente en radianes a la coordenada del ángulo en el archivo

"polar.dat."

Ejercicio 11.27.

Escribir un programa invocable en línea de órdenes que permita restaurar un archivo de texto transmitido por FTP en modo binario en vez de modo texto. En UNIX cada línea de texto finaliza con 0xA, mientras que en Windows se finaliza con 0x0D 0x0A

a) Para ser restaurado en Unix

b) Para ser restaurado en Windows

Ejercicio 11.28.

Una librería quiere automatizar su sistema de control de inventario. Las condiciones que debe reunir el sistema de control de inventario, son las siguientes:

a) El sistema debe estar funcionando mientras la librería esté abierta. Cuando la librería abra por la mañana, el programa leerá el inventario desde el archivo de inventario, "entrada.dat". Cuando, al finalizar el día, se cierre la librería, el inventario actualizado deberá grabarse en el mismo archivo.

b) Cuando la librería reciba un envío de libros, el inventario deberá actualizarse bien, mediante el aumento del número de libros en depósito o bien mediante la entrada de un nuevo libro, si aún no está contenido en el inventario.

c) Cuando los libros estén agotados, el inventario deberá actualizarse mediante el borrado de la entrada, pero sólo cuando el número de libros en reserva sea cero.

d) Cada vez que un libro sea vendido, el inventario deberá actualizarse.

e) El propietario quiere tener un listado completo del inventario, ordenado alfabéticamente por el título de los libros.

f) También quiere poder mostrar la información bibliográfica referente a un libro en concreto que haya en el inventario.

g) El programa debe generar una lista de los títulos de todos lo libros que haya en el inventario e indicar cuántas copias de cada libro se venden cada día.

En las reuniones han acordado la información referente a cada libro que se guardará en el inventario. La información siguiente es la que se almacenará en el archivo de inventario "entrada.dat" para cada libro:

- Título (50 caracteres como máximo)
- Apellido del autor (30 caracteres como máximo)
- Nombre del autor(20 caracteres como máximo)
- Precio
- Editorial (30 caracteres como máximo)
- ISBN
- Fecha de copyright
- Número de ejemplares disponibles
- Estado (1 para libros que pueden solicitarse a la editorial y 0 para los libros agotados en la editorial y que no se reimprimen)

Ejercicio 11.29.

Escribir una función que permita leer al dispositivo de audio del computador. El prototipo es:

int DSPread (int fd, struct DSPparams , int TimeRead, char *buffer, size_t tam);

fd es el file descriptor obtenido mediante la función open ().

La estructura *DSPparams*, debe ir en un archivo header complementario del fuente de la función, y debe como mínimo contener los siguientes elementos:

int SampleSize: Tamaño de la muestra de audio con que se desea leer el dispositivo

int SampleRate: Velocidad de muestreo a la que se desea configurar el dispositivo de audio

int Channels: Cantidad de canales que se desea trabajar (1 o 2).

La función debe:

a) Configurarlo con los parámetros recibidos en la estructura DSPparams

b) Leer audio durante el tiempo estipulado en el argumento TimeRead. El valor de este parámetro se mide en segundos.

c) Guardar las muestras leídas en *buffer* (cuarto parámetro de la función), considerando como tamaño del mismo el parámetro *tam*.

d) retornar la cantidad de muestras leídas (es decir la cantidad de elementos del tamaño SampleSize), o -1 en caso de error.

Ejercicio 11.30.

Escribir una función que escriba en el dispositivo de audio del computador.

El prototipo es:

void DSPwrite (int fd, struct DSPparams, char *buffer, size_t tam, int flag);

fd es el file descriptor obtenido mediante la función open ().

La estructura *DSPparams*, es la misma del caso anterior. Si este argumento contiene NULL, no debe ser tomado en cuenta y se trabajará con los parámetros configurados en el dispositivo.

La función debe:

a) Configurar el dispositivo con los parámetros recibidos en la estructura *DSPparams*, siempre que este argumento sea diferente a NULL.

b) Escribir en el dispositivo la información mediante los parámetros *tam*, y *buffer*.

c) En la escritura manejar el bloqueo del dispositivo de audio mediante el parámetro *flag*. Si es flag es 0 no bloquea el dispositivo durante la escritura.

d) retornar la cantidad de bytes escritos, o -1 en caso de error.

Ejercicio 11.31.

Utilizando las funciones de los dos ejercicios anteriores desarrolle una aplicación ejecutable desde la consola que permita leer audio estéreo con 15KHz de

máxima frecuencia, y tamaño de muestra de 16 bits, y lo reproduzca en el parlante de su computador.

Ejercicio 11.32.

Tome el código del ejercicio anterior, y modifíquelo para que la reproducción de audio se lleve a cabo introduciendo una demora de algunos microsegundos a uno de los canales. Mediante enum trabaje el canal en donde se desea introducir la demora. La cantidad de milisegundos se pasa por línea de comandos. Ejemplo

$ dplay R 120

reproducirá la entrada de audio aplicando al canal derecho (R de Rigth) una demora de 120 mseg.

$ dplay L 200

reproducirá la entrada de audio introduciendo al canal izquierdo (L de left) una demora de 200 mseg.

Ejercicio 11.33.

Desarrolle una función que abra un archivo de audio en formato wav PCM e imprima en pantalla todos sus parámetros. El encabezado contiene el siguiente formato.

Nombre del campo	Tamaño [Bytes]	Offset [Bytes]	Endian	Descripción
ChunkID	4	0	Big	Se almacenan los caracteres RIFF
ChunKSize	4	4	Little	Es el tamaño del archivo menos 8 bytes = 4 + (8 + SubChunk1Size) + (8 + SubChunk2Size)
Format	4	8	Big	Se almacenan los caracteres WAVE
SubChunk1ID	4	12	Big	Se almacenan los caracteres fmt
SubChunk1Size	4	16	Little	Vale 16 para PCM
AudioFormat	2	20	Little	Para PCM = 1
NumChannels	2	22	Little	Cantidad de canales Mono = 1; Stero = 2, etc
SampleRate	4	24	Little	Frecuencia de muestreo
ByteRate	4	28	Little	SampleRate*NumChannels*BitPerSample/8
BlockAlign	2	32	Little	Cantidad de bytes por muestra de todos los canales = NumChannels*BitsPerSample/8
BitsPerSample	2	34	Little	Cantidad de bits por muestra:

				8 = 8 bits; 16 = 16 bits
SubChunk2ID	4	36	Big	Se almacenan los caracteres data
SubChunk2Size	4	40	Little	Cantidad de bytes de audio. NumSamples*'NumChannels'*bitpersample/8
Data	SubChunk2Size	44	Little	Las muestras de audio

A continuación se muestra el parseo de un archivo wav.

52	49	46	46	60	54	13	00	57	41	56	45
ChunkID				ChunkSize				Format			
“RIFF”				1266784Bytes				“WAVE”			

66	6d	74	20	10	00	00	00	01	00	02	00
SubChunk1ID				SubChunk1Size				AudioFormat		NumChannel	
“fmt “				16Bytes				PCM		2	

44	AC	00	00	10	B1	02	00	04	00	10	00
SampleRate				ByteRate				BlockAlign		BitPerSample	
44100Hz				67285464Bytes/s				4Bytes		16Bits	

64	61	74	61	3C	54	13	00
SubChunk2ID				SubChunk2Size			
“data”				126674:8Bytes			

Notas:

- Si las muestras son de 8 bits se guarda en el rango de 0 a 255.
- Las muestras de 16 bits son almacenadas en complemento a dos.
- Las muestras para dos canales están ordenadas de la siguiente manera

Muestra0	Muestra0	Muestra1	Muestra1	Muestra2	Muestra2
Canal0	Canal1	Canal0	Canal1	Canal0	Canal1

Ejercicio 11.34.

Escriba un programa que utilizando la función anterior y la de escritura del dispositivo de audio, reproduzca por el sistema de sonido del computador un archivo wav.

Ejercicio 11.35.

Realice una función que tome como entrada un archivo wav PCM stereo e invierta los canales.

Ejercicio 11.36.

Desarrolle una función que tome como entrada un archivo wav stereo, sume ambos canales y genere un archivo wav mono.

Ejercicio 11.37.

Desarrolle una función que abra un archivo de audio wav mono y modifique la frecuencia de muestreo del mismo a la mitad, generando como salida un archivo wav mono.

Ejercicio 11.38.

Desarrolle una FIFO de N elementos para datos del tipo int. Dicha FIFO debe presentar la siguiente interfaz.

- struct fifo * fifoCreate (int N);
  - Descripción: Crea la FIFO.
  - Parámetros: Cantidad de elementos de la FIFO.
  - Retorna: Puntero a la estructura de control de la FIFO. En caso de error devuelve NULL.
- int fifoWrite (struct fifo *p, int *dataPtr, int dataSize);
  - Descripción: Escribe datos en la FIFO.
  - Parámetros:
    - Puntero a la estructura de control de la FIFO.
    - Puntero a los datos a escribir en la fifo.
    - Cantidad de elementos a escritos.
  - Retorna: La cantidad de elementos escritos en la FIFO.

    O retorna -1 si la cantidad de elementos es mayor al tamaño definido cuando se la creó

- int fifoRead (struct fifo *p, int *dataPtr, int dataSize);
  - Descripción: Lee datos de la FIFO.
  - Parámetros:
    - Puntero a la estructura de control de la FIFO.
    - Puntero al buffer donde colocar los datos leídos.
    - Cantidad de elementos a leer.
  - Retorna: La cantidad de elementos leídos en la FIFO

    O retorna 0 si está vacía, y -1 si hubo cualquier otro error.
- int fifoDestroy (struct fifo *p);
  - Parámetros:
  - Puntero a la estructura de control de la FIFO.
  - Retorna:
  - Cero en caso de éxito, -1 en el caso opuesto.

Nota: No debe usar ***mkfifo***. La estructura de control de la FIFO debe contener todo lo necesario para el manejo de la misma.

Ejercicio 11.39.

Tomando como base el ejercicio anterior, desarrolle una FIFO de N bytes para elemento de tamaño no definido.

Ejercicio 11.40.

Entrega Obligatoria

Escriba un programa que invoque a la función de lectura del dispositivo de audio y envíe las muestras leídas al FIFO ubicado en ~/Info1/conector.

Escriba un segundo programa que tome las muestras de ese FIFO y las envíe a la salida de audio correspondiente.

Ejercicio 11.41.

Desarrollar una aplicación que ejecutada en diferentes terminales del mismo equipo Linux por dos usuarios permita a estos establecer un chat, sencillo. Esto es cada extremo escribe de a una vez y luego pasa a modo escucha. Hasta que no reciba nada del otro extremo no estará habilitado para escribir nuevamente.

Ejercicio 11.42.

Entrega Obligatoria

Utilizando la función de librería provista por la cátedra para abrir una conexión por red modifique el programa anterior para que el chat pueda hacerse entre usuarios con sesión en equipos diferentes pero vinculados por una red de datos.

Ejercicio 11.43.

Usando la misma función de librería del ejercicio anterior desarrolle, un programa que pueda transmitir un archivo al computador. El path del archivo y la dirección IP de destino se le especifican por línea de comandos.

El receptor guarda los archivos recibidos en la ruta /download/. Si en esa dirección ya existe un archivo con ese nombre, el archivo destino llevará el mismo nombre del que se transmite, al que se le añade un número de secuencia: "001", "002", etc, según cuantas versiones haya del mismo en el directorio de recepción (se busca con ello evitar pisar archivos que no sean duplicados).

Ejercicio 11.44.

Entrega Obligatoria

Utilizando la función provista por la cátedra para abrir una conexión, la función DSPread escrita en el Ejercicio 4.35, y la función DSPWrite escrita en el Ejercicio4.36, escriba un programa que opere de acuerdo a las siguientes especificaciones.

El programa se ejecuta con el siguiente comando

streamer [modo] [ip]

modo puede tomar dos valores: ***tx*** o ***rx***.

Si ***modo*** es ***tx***, abre una conexión con el nodo especificado en el parámetro ***ip***, utilizando la función provista por la cátedra, lee audio desde el dispositivo a 15KHz de ancho de banda, con un tamaño de muestra de 16 bits, y estéreo, y lo transmita por red al nodo remoto.

Si ***modo*** es ***rx,*** entonces abre la conexión con el nodo remoto especificado en el parámetro ***ip***, y espera por ella audio asumiéndolo con Ancho de Banda 15 KHz., tamaño de muestra 16 bits, y estéreo, y lo escriba en el dispositivo de audio.

Utilice el puerto de conexión 5466.

Ejercicio 11.45.

Modifique el programa anterior para que antes de comenzar la transmisión el extremo transmisor, le indique al receptor mediante intercambio de mensajes los parámetros de audio (Frecuencia de Muestreo, estéreo/mono, y tamaño de muestra).

El receptor deberá programar al dispositivo de audio, y si dicha acción no arrojó errores, le indicará mediante un mensaje al extremo transmisor que está listo para recibir audio. A partir de este evento, comienza la transmisión.

Si la programación del dispositivo de audio en el extremo receptor dio errores, entonces deberán suspender la transmisión enviando por consola los mensajes correspondientes, y cerrando la conexión mediante la función de librería provista por la Cátedra.

T.P. Nº 12. Programación Avanzada en Linux

Condiciones Generales para resolver el Trabajo Práctico.

Para el caso de los ejercicios que pidan únicamente escribir la función, éstos deben estar escritos en un archivo fuente que contenga solo la función pedida. Las definiciones que se requieran deben efectuarse en un archivo header que tenga el mismo nombre del fuente C.

El entregable es el conjunto de programas fuente (*.c y *.h), junto con un ***Makefile*** para ser construido el ejecutable mediante la orden ***make*** simplemente.

Objetivo:Las funciones desarrolladas en esta sección irán luego a librerías de código sumándose a las de los Trabajos Prácticos anteriores, y serán utilizadas en los próximos Trabajos Prácticos.

Ejercicio 12.1.

Escriba un programa que ejecute un ciclo infinito sin consumir CPU del sistema y que no pueda ser interrumpido por medio de CTRL-C.

Ejercicio 12.2.

Modificar al programa anterior para que realice las siguientes actividades:

a. Presentar en stdout su número de proceso y la fecha y hora de su inicio, con el formato aaaa-mm-dd hh:mm:ss,mseg.

Por ejemplo: Soy el PID Nº 27509, creado en 2011-05-04 15:46:16,397

b. Cada vez que reciba la señal SIGUSR1 presente en stdout la fecha y hora actuales, en el mismo formato del punto a.

c. Termine cuando recibe una señal SIGUSR2, imprimiendo en stdout "Terminado por señal SIGUSR2", seguido del timestamp.

Ejercicio 12.3.

Entrega Obligatoria

Modifique el programa anterior para que al recibir SIGUSR1, en lugar de escribir el timestamp en stdout, cree un proceso mediante la syscall fork() que haga esa actividad, y termine.

Ejercicio 12.4.

Cuando ejecuta el programa anterior, visualice en una señal el estado del proceso en ejecución verificando que sus procesos hijos creados a consecuencia de SIGUSR2 terminen correctamente. Si los procesos siguen estando preste atención a su estado. Para ello el comando ps -elf | grep [nombre del proceso] le será de mucha ayuda.

Intercepte la señal SIGCHLD y escriba en ella el código para resolver este problema. Ahora sí su programa funcionará correctamente.

Ejercicio 12.5.

Modificar el programa anterior para que finalice automáticamente sin no recibe señales SIGUSR1 durante 30 segundos.

Ejercicio 12.6.

Entrega Obligatoria

Se necesita monitorear desde una única función datos una conexión de red, un Named pipe, y stdin. Los datos pueden llegar en cualquier orden, y deben ser recuperados ni bien se reciben.

A medida que se leen las entradas se loguean respectivamente en ./netwok.log, ./NamedPipe.log, ./stdin.log.

El programa termina al pulsarse CTRL-C

Ejercicio 12.7.

Modifique el programa anterior para esperar cada una de las entradas en un thread diferente.

Ejercicio 12.8.

Escriba un programa que pueda ejecutarse en diferentes computadores, y que permita un chat entre los usuarios.

Cada usuario puede escribir cuanto desee sin esperar a recibir mensajes del otro extremo. Para ello utilizará un thread para controlar la escritura y otro thread para recibir datos.

Ejercicio 12.9.

Modifique el programa anterior para que cuando se reciben datos, la presentación de estos espere que se complete la escritura local de datos si esta está en curso.

Sugerencia. Exclusión mutua.

Ejercicio 12.10.

Entrega Obligatoria

Tome cualquiera de los ejercicios resueltos que multiplican dos matrices y paralelice el algoritmo de multiplicación mediante el uso de threads.

Ejercicio 12.11.

Un proceso debe leer datos en formato ASCII por dos conexiones de red diferentes. Los datos se almacenan en dos archivos separados, y se desea llevar la cuenta de los caracteres recibidos por cada una. Se desea comparar tres resoluciones posibles:

a. En un único proceso utilizando select () para monitorear ambas conexiones.

b. Creando una segunda instancia del proceso original en ejecución mediante la syscall fork (). Intentar en ambos procesos utilizar la misma variable, que ha sido definida en el proceso original, para acumular en cada caso la cantidad de caracteres. ¿Funciona? ¿O se mezclan los datos?¿Porque?

c. Utilizando threads. Uno por cada conexión. Intentar en ambos threads utilizar la misma variable, que ha sido definida en el proceso original, para acumular en cada caso la cantidad de caracteres. ¿Funciona? ¿O se mezclan los datos? ¿Porque?

Ejercicio 12.12.

Entrega Obligatoria

Escriba un programa que cree un buffer de BUFFER_SIZE bytes, y una cantidad máxima de MAX_THREADS. Cada thread, t_0, t_1, t_2, … , t_i, …, $t_{(MAX\_THREADS-1)}$, y escribirá cada *i* segundos, el número *i* en la primer posición libre del buffer, presentando en pantalla "Escribí *i* en la posición *p*!!". Cada thread finaliza cuando detecta la condición de buffer full. El proceso termina cuando todos los threads lo hayan hecho ya.

Ejercicio 12.13.

Se dispone de un archivo índice que contiene por cada línea el path de un archivo de texto. Escribir un programa que lea ese archivo, genere una lista doblemente enlazada en la que cada nodo tenga la descripción del path del archivo, y tres variables enteras a calcular para cada uno: Cantidad de líneas, cantidad de palabras y cantidad de caracteres. A continuación debe crear un thread por cada nodo que se ocupará de cacular las tres variables restantes de

ese archivo y almacenarlas en la lista.
Una vez finalizados **todos** los threads, el programa generará un reporte por stdout con los resultados. Antes de ello propondrá al usuario el criterio de ordenamiento, por nro. de nodo, por tamaño de cada archivo en caracteres, por cantidad de líneas, o por cantidad de palabras.

Ejercicio 12.14.

Repita el ejercicio de detección de bordes de la imagen visto en el TP Nº 9, procesando cada fila de la imagen con un thread diferente.

Evalúe el tiempo de respuesta de ambos algoritmos y extraiga conclusiones al respecto.

Ejercicio 12.15.

Se desea recibir por conexiones de red información desde equipos remotos. Para ello se pide escribir un programa que espere indefinidamente pedidos de conexión por el port TCP 12345.

Por cada pedido creará un thread para atenderlo y dicho thread creará un nodo de una lista doblemente enlazada, creada por el programa principal, en el que escribirá los siguientes datos:

Dirección IP de destino (en formato punto, es decir, 210.23.7.134, por ejemplo)

Mensaje recibido

Debe evitarse accesos concurrentes a lista para evitar que se pisen las referencias entre nodos.

T.P. Nº 13. Ejercicios Integradores

Ejercicio 13.1.

Una curtiembre necesita desarrollar el software para una máquina que mida el área de los cueros que han finalizado el proceso de curtido.

El modelo más común de estas máquinas consiste básicamente en una cinta transportadora que desplaza los cueros, uno por uno, a una velocidad constante. La medición en sí se realiza en base a 16 sensores ópticos ubicados en una hilera perpendicular al desplazamiento de los cueros, por debajo de los mismos. El sistema recibe una potente iluminación desde arriba y cada sensor detecta o no la presencia de luz. Si no hay luz, significa que el cuero está pasando por allí. Como cada sensor corresponde a un área parcial del cuero a ser medido, la superficie total del cuero se obtendrá sumando todas las áreas parciales. A este proceso se lo denomina medición de área por integración gráfica.

El objetivo principal de la medición es clasificar los cueros en 3 clases: chicos (área menor a 2,3 m^2), medianos (área entre 2,3 y 3,6 m^2) y grandes (área mayor a 3,6m^2).

La velocidad de desplazamiento de la cinta es de 0,9 m/min, y cada sensor cubre una longitud de 12,5 cm. La hilera de sensores envía el estado de cada uno de los mismos a través de los ports de una PC, a partir del 0x310. Cada sensor se corresponderá con un bit que se pondrá en "0" cuando detecte luz, y en "1" cuando no la detecte. Todos los sensores de la hilera envían la información de su estado cada 3 segundos.

Una vez puesto en marcha el sistema, con la recepción del valor 0Fh en el port 0x312, todos los sensores están en "0", y la medición comenzará cuando cualquiera de ellos se ponga en "1"

(porque un cuero comenzó a pasar sobre la hilera de sensores). Se considera finalizado el paso de un cuero cuando, luego de 3 lecturas consecutivas, todos los sensores están en "0". El fin del proceso completo se produce cuando el port 0x312 recibe el valor 7Fh.

Se pide desarrollar el programa en lenguaje C que realice la medición según las indicaciones dadas, y además determine e informe:

- Cantidad de cueros de cada clase

- promedio de todos los cueros

Ejercicio 13.2.

(Tema de Parcial) Se tiene la siguiente esctructura:

```
struct datos_taller {
        unsigned int numero;
        unsigned char encendida;
        unsigned int rpm;
        };
```

Escriba una función cuyo prototipo es:

struct datos_taller maquinas(unsigned char);

El carácter recibido por la función maneja la siguiente información:

Bit 0 a 2	número de máquina (de 1 a 8).
Bit 3	apagada (0) o encendida (1).
Bits 4 a 7	velocidad (en RPM) de cada máquina.

La función debe devolver los 3 datos de funcionamiento (es decir, número de máquina, si está encendida o no, y velocidad (en caso de estar prendida) a través de la estructura indicada en el prototipo.

Ejercicio 13.3.

Utilice la función del ejercicio anterior para escribir un programa que recibirá los datos de las máquinas de un taller a través del port 0x300. El programa debe determinar en tiempo real (es decir, mientras se van recibiendo los datos):

a) Cantidad de máquinas apagadas

b) Velocidad promedio de las máquinas encendidas

La información recibida por el port se actualiza como mínimo cada 0,5 segundos.

El programa finaliza pulsando la tecla "X" (mayúscula o minúscula). ¡Cuidado! Recuerde que no se puede interrumpir al programa en espera de la intervención del usuario.

Ejercicio 13.4.

(Tema de final) Se tiene un archivo maestro con los datos de los clientes de una agencia de remises. Los registros de datos de dicho archivo presentan la siguiente estructura: Teléfono (campo clave), Apellido y nombre, Dirección, Cantidad de viaje realizados e Importe acumulado. La ruta y nombre de dicho archivo es D:\REMIS\MAESTRO.DAT

Se pide un programa en lenguaje C que presente un menú con las siguientes opciones:

a) VIAJES: en esta opción se registra cada uno de los viajes realizados, guardándolos en un archivo de novedades, cuya ruta y nombre es D:\REMIS\NUEVOS.DAT. La estructura de este archivo es idéntica a la del archivo maestro, excepto que el importe ahora no es un acumulado, sino el correspondiente a cada viaje. Se debe verificar en el archivo maestro la cantidad de viajes de cada cliente, ya que cada 10 viajes realizados, el siguiente es gratis. En este caso, el importe a registrar será $0.

b) ACTUALIZACIÓN: esta opción actualizará el archivo maestro con los datos registrados en el de novedades. Se actualizan los campos correspondientes a cantidad de viajes e importe acumulado. En caso de que sea un nuevo cliente, se agregará el nuevo registro. Además, después de la actualización se borrará el archivo de novedades.

c) SALIR: Esta opción finalizará el programa

NOTA: Tenga en cuenta que la búsqueda se hará a través del campo clave teléfono.

Ejercicio 13.5.

(Tema de final) Después de denodados esfuerzos, el Servicio de Inteligencia de Chinlandia ha logrado desencriptar la clave de los mensajes secretos que envían los espías de Tibecia, su legendaria nación rival. Los programadores chinlandeses han logrado captar una transmisión espía y mediante un ingenioso dispositivo lograron que la misma ingrese, byte a byte, por el puerto 0x320 de una vieja PC abandonada en la embajada de Iowandia. Consideraremos que la transmisión es continua, ya que el dispositivo contraespía transmite constantemente. La clave para desencriptar los mensajes ingresa a través del puerto 0x330, procediéndose de la siguiente manera.

a) Se sabe que comienza un mensaje válido cuando tres bytes seguidos del mensaje tienen el valor F5.

b) A partir de allí se lee un byte de la clave por cada byte del mensaje

c) La clave consiste en las siguientes reglas:

- Si todos los bits de orden par de la clave están en “0”, forzar los bits de orden 3 y 1 del mensaje a “1”
- Si los bits 5 y 4 de la clave están en “1”, invertir todos los bits del mensaje.
- Si el valor de la clave es impar, intercambio los 4 bits de más peso por los de menos peso del mensaje.

- En el resto de los casos, el byte mensaje se descarta.

d) El mensaje termina cuando tres bytes desencriptados consecutivos toman el valor 5F

Cuando se desencripta un mensaje completo, el programa dará la opción de guardarlo en disco como archivo de texto. La ruta será C:\CHINLANDIA\ y el nombre lo elige el usuario. Si Ud. cursó la materia antes de 2006, en vez de esto, retransmita el mensaje desencriptado por el puerto 0x340. Después de cualquiera de las dos opciones, el sistema estará en condiciones de recibir y desencriptar un mensaje nuevo.

NOTAS:

- Recuerde: el mensaje a desencriptar llega por el port 0x320, y la clave por el 0x330.
- Para asegurar la sincronización de los puertos, tenga en cuenta que los datos se mantienen durante 250 mseg.
- Debe incluir al menos una función con el siguiente prototipo: char desencriptar(char, char);
- Puede salir del programa en cualquier momento presionando la tecla "X" (mayúscula o minúscula), pero sin "cortar" la ejecución del programa.

Ejercicio 13.6.

(Tema de final) Una conocida casa de empanadas con servicio de delivery quiere disponer de un sistema para optimizar la gestión de pedidos.

En primer lugar, se cuenta con una base de datos con los siguientes datos:

- Teléfono (se utilizará como código de cliente)
- Nombre
- Dirección
- Cantidad de pedidos acumulados

Cuando un cliente llama, se solicita en primer lugar su número de teléfono. Si dicho teléfono ya está registrado se procede a ingresar su pedido; en caso contrario, se procede al alta del mismo en la base de datos, solicitando nombre y dirección.

A continuación, tanto si ya era cliente como si fue dado de alta en ese llamado, se toma nota de su pedido. Para ello se ingresará el gusto elegido (los gustos disponibles son carne, pollo, jamón y queso y humita) y la cantidad de empanadas de cada uno. Decida Ud. como implementar este ingreso. Tenga en

cuenta que esta información NO FORMARÁ PARTE de la base de datos, aunque se incrementará en 1 (uno) la cantidad de pedidos acumulados.

Una vez que la operadora termina de ingresar el pedido, se emitirán por el port 0x330 una serie de 7 (siete) bytes con la información relativa a cada pedido, codificada de la siguiente manera:

- 1er byte: AAh
- 2do byte: Número de pedido (comienza en 0 y se incrementa en uno por cada nuevo pedido)
- Bytes 3ro a 6to: Se transmite un byte por cada gusto. Los dos primeros bits de cada uno de estos bytes corresponden al gusto, mientras que los seis restantes corresponden a la cantidad de empanadas de ese gusto
- 7mo byte: FAh

NOTA: El programa termina con número de teléfono 0 (cero)