1. Ejercicios: Sistema de numeración

1. Pasar al sistema decimal el número 101111_2
2. Pasar el número $27{,}025_{10}$ a binario
3. Realiza las siguientes operaciones
    a. 101101+1011
    b. 10001+111
4. Pasa a binario el número $3CB_{16}$
5. Pasa a hexadecimal el número 381_{10}
6. Conversión de binario a decimal:
    a. 101110_2=_____$_{10}$
    b. 000011_2=_____$_{10}$
    c. 101010_2=_____$_{10}$
    d. 111000_2=_____$_{10}$
7. Conversión de decimal a binario:
    a. 64_{10}=_____$_2$
    b. 145_{10}=_____$_2$
    c. 500_{10}=_____$_2$
    d. 111_{10}=_____$_2$
8. Convertir los siguientes números octales a decimales:
    a. 42_8=_____$_{10}$
    b. 376_8=_____$_{10}$
    c. $11{,}11_8$=_____$_{10}$
    d. $37{,}123_8$=_____$_{10}$
9. Convertir los siguientes números decimales a sus octales equivalentes:
    a. $77{,}375_{10}$=_____$_8$
    b. $20{,}515625_{10}$=_____$_8$
    c. $8{,}15625_{10}$=_____$_8$
    d. $44{,}5625_{10}$=_____$_8$
10. Convertir los siguientes números octales a sus binarios equivalentes:
    a. $7{,}5_8$=_____$_2$
    b. $16{,}3_8$=_____$_2$
    c. $20{,}1_8$=_____$_2$
    d. $37{,}6_8$=_____$_2$

11. Convertir los siguientes números binarios a sus equivalentes octales:
    a. 001=_____$_8$
    b. 110=_____$_8$
    c. 111000=_____$_8$
    d. 101100=_____$_8$
12. Convertir los siguientes números hexadecimales a sus decimales equivalentes:
    a. $F,4_{16}$=_____$_{10}$
    b. $D3,E_{16}$=_____$_{10}$
    c. $1111,1_{16}$=_____$_{10}$
    d. EBA,C_{16}=_____$_{10}$
13. Convertir los siguientes nº decimales a sus hexadecimales equivalentes:
    a. $204,125_{10}$=_____$_{16}$
    b. $255,875_{10}$=_____$_{16}$
    c. $631,25_{10}$=_____$_{16}$
    d. $10000,039_{10}$=_____$_{16}$
14. Convertir los siguientes números hexadecimales a sus equivalentes binarios:
    a. B_{16}=_____$_2$
    b. $1C_{16}$=_____$_2$
    c. $1F,C_{16}$=_____$_2$
    d. $239,4_{16}$=_____$_2$
15. Convertir los siguientes números binarios a sus hexadecimales equivalentes:
    a. $1001,111_2$=_____$_{16}$
    b. $110101,011001_2$=_____$_{16}$
    c. $10000,1_2$=_____$_{16}$
    d. $10000000,0000111_2$=_____$_{16}$
16. Convertir los siguientes hexadecimales a sus decimales equivalentes:
    a. C_{16}=_____$_{10}$
    b. $9F_{16}$=_____$_{10}$
    c. $D52_{16}$=_____$_{10}$
    d. $67E_{16}$=_____$_{10}$
    e. $ABCD_{16}$=_____$_{10}$

Ejercitación complementaria

Serie B – primeros códigos

La siguiente ejercitación tiene como objetivo:

- Analizar los enunciados y comprender que es lo que hay que hacer (si no sabemos que es lo que hay que hacer, cualquier cosa que hagamos … será cualquier cosa).
- Conocer las herramientas básicas para implementar nuestras soluciones propuestas.
- Utilizar siempre que sea posible solo datos tipo **int**. En caso de requerirse datos con decimales, puede utilizar datos tipo float o double según considere adecuado.
- Implementar y probar a través de funciones/aplicaciones las soluciones propuestas.

Nociones básicas

Símbolos matemáticos

- Asignación =
- Suma + p./ej.: a = b + c;
- Resta –
- División / p/ej.: a = b / 10;
- Multiplicación *
- Resto de módulo % (*Nos da el resto de la división*)

Símbolos para toma de decisión

- Si es mayor >
- Si es mayor o igual >=
- Si es menor <
- Si es menor o igual <=
- Si son iguales == (doble igual, sino sería una asignación) p/ej.: x == y
- Si son distintos != (Símbolo de exclamación seguido del igual)

Dentro del enunciado se menciona que determinadas funciones deben retornar **verdadero** o **falso** según corresponda, para ello podrán seleccionar el tipo de dato **int**, y retornar **1** cuando es **verdadero** y **0** en caso de **falso**.

Al utilizar este tipo de datos, nos puede ser muy útil el símbolo de negación, el cual cambia una expresión verdadera a Falso y viceversa.

- Negación ! (Símbolo de exclamación)

Notará que muchos ejercicios solicitan la realización de funciones. Para poder probar estas funciones deberá implementar el módulo principal o main, y preparar los datos que requieran las funciones solicitadas para poder ejecutarlas y verificar correcto su funcionamiento (o incorrecto).

Ejercicio 1: Par o impar

Ejercicio 1.a:

Realizar una función que retorne **verdadero** si el número recibido es par y **falso** si es impar.

Ejercicio 1.b:

Realizar una función que retorne **verdadero** si el número recibido es impar y **falso** si es par.

Ejercicio 1.c:

Si ya no lo hizo, modifique alguna de las funciones anteriores (1.a o 1.b) para que realice lo solicitado, en base a la otra función. Es decir, determinar si un número es par usando la función que determina si un número es impar o viceversa.

Ejercicio 2: En rango

Realizar una función que retorne **verdadero**, si el número recibido se encuentra en el rango que va desde 100 a 500, incluyendo los extremos, y **falso** en caso contrario.

Ejercicio 3: Divisibles

Realizar una función que reciba dos números enteros y determine si el primero de los valores es divisible por el segundo. La función debe retornar **verdadero** si es divisible y **falso** en caso contrario.

Ejercicio 4: el cuadrado

Realizar una función que retorne el cuadrado del número recibido.

Ejercicio 5: el cubo

Realizar una función que calcule y retorne el cubo del número recibido.

5.a: implementar la función en el campo de los enteros.

5.b: implementar la función en el campo de los reales.

Ejercicio 6: el rectángulo

Realice una función que calcule el área de un rectángulo a partir de los valores recibidos como parámetros.

Ejercicio 7: el círculo

7.a: Realice una función que calcule el área de un círculo, cuyo radio se recibe como parámetro.

7.b: Modificar la función de modo que, si el radio es negativo, la función retorne -1 (menos uno).

Ejercicio 8: la circunferencia

Realice una función que determine el perímetro de la circunferencia cuyo radio se recibe como parámetro.

Sí el radio recibido es negativo (menor a cero), la función debe retornar -1 (menos uno).

Ejercicio 9: la potencia

Realizar una función que reciba dos números enteros $\mathbf{m}$ y $\mathbf{n}$, y devuelva $\mathbf{m^n}$.

Analice si la función posee o no restricciones y que se podría hacer al respecto.

Ejercicio 10: dentro del rango

Realizar una función que retorne **verdadero**, si el número recibido se encuentra dentro de un rango configurable. Los extremos, no forman parte del rango.

En este contexto, que sean configurables significa que son valores que conforman el rango también se deben enviar a la función.

Ejercicio 11: Los cuadrantes

Realizar una función que determine y devuelva el cuadrante al que pertenece un punto (x;y) recibido como argumento.

Nota: *Definir el comportamiento de la función cuando un punto coincide con alguno de los ejes de coordenadas.*

Ejercicio 12: El mayor

Realizar una función que reciba tres parámetros numéricos, determine cuál es el mayor y lo retorne.

Ejercicio 13: La hipotenusa

Realice una función que calcule la hipotenusa de un triángulo rectángulo.

Ejercicio 14: En rango+

Realizar una función que retorne verdadero, si el número recibido se encuentra dentro del rango recibido como parámetro, incluyendo los extremos, y falso en caso contrario.

Ejercicio 15: El triangulo

Realizar una función que retorne el área de un triángulo a partir de la longitud de sus lados.

En caso de recibir un lado con un valor negativo, transformarlo en positivo y continuar normalmente.

En caso de no poder calcular el área, la función debe retornar -1.

Ejercitación complementaria

Serie C - Ciclos

La siguiente ejercitación tiene como objetivo:

- Analizar los enunciados y comprender que es lo que hay que hacer (si no sabemos que es lo que hay que hacer, cualquier cosa que hagamos … será cualquier cosa).
- Ampliar y profundizar en el conocimiento de herramientas básicas para implementar nuestras funciones.
- Implementar a través de diagramas de flujo las soluciones propuestas.

Notará que muchos ejercicios solicitan la realización de funciones. Para poder probar estas funciones deberá implementar el módulo principal o main, y preparar los datos que requieran las funciones solicitadas para poder ejecutarlas y verificar correcto su funcionamiento (o incorrecto).

En los casos que se solicite la realización de una aplicación, tenga en cuenta el diseño de las funciones que hagan más simple y legible nuestra solución.

Ejercicio 1: El promedio

Realizar una función que calcule y retorne el promedio de una serie de valores ingresados por teclado.

El final del ingreso se debe producir cuando el valor ingresado sea menor o igual al valor recibido como argumento.

Este último valor, el que cierra el proceso de ingreso, no debe ser considerado en el promedio.

Notas: *Contemplar el caso que no haya ingresos válidos.*

Ejercicio 2: Los cuadrantes

Realizar una aplicación que permita el ingreso de varias puntos (x;y) e informe a que cuadrante corresponde cada uno. El proceso finaliza al ingresar el centro de coordenadas (0;0).

Antes de finalizar la aplicación, se debe informar la cantidad de puntos ingresados y cantidad de puntos que corresponden a cada cuadrante.

Analice si la función posee o no restricciones y que se podría hacer al respecto.

Ejercicio 3: Los octantes (cuadrantes en 3D)

Realizar una aplicación que permita el ingreso de varias puntos (x;y;z) en el espacio e informe a que octante corresponde cada uno. El proceso finaliza al ingresar el centro de coordenadas (0;0;0).

Antes de finalizar la aplicación, se debe informar la cantidad de puntos ingresados y cantidad de puntos que corresponden a cada octante.

Analice si la función posee o no restricciones y que se podría hacer al respecto.

Ejercicio 4: factorizar

Para factorizar un número o descomponerlo en factores efectuamos sucesivas divisiones entre sus divisores primos hasta obtener un uno como cociente.

Como ejemplo podemos citar que el 176, se compone de los factores 2; 2; 3; 7 y 11

Se pide entonces, hacer una función que obtenga e imprima los factores de un número recibido como parámetro.

Considerar que el número recibido puede ser negativo.

Ejercicio 5: es primo

Realizar una función que determine si un número es primo o no. La función debe retornar **verdadero** si es primo y **falso** en caso contrario.

Ejercicio 6: más primos

Ejercicio 6.1:

Realizar una aplicación que imprima todos los números primos que están entre el 1 y el 1000.

Ejercicio 6.2:

Realizar una aplicación que imprima los primeros 100 números primos.

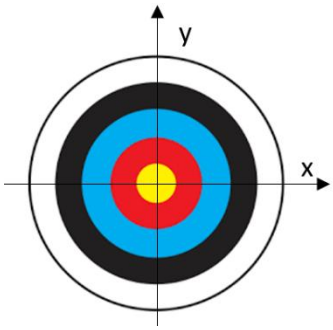
Ejercicio 7: Tiro al blanco

Realizar una función que permita el ingreso por teclado de tres tiros y devuelva el puntaje obtenido.

Al invocarse la función, ésta debe permitir ingresar por teclado tres pares de coordenadas x,y que indican el lugar del impacto de cada tiro. Estos valores pueden ser positivos o negativos dependiendo del cuadrante del impacto y están expresados en milímetros (mm).

Por su parte, el blanco posee 5 zonas concéntricas que de acuerdo a la distancia (d) del impacto al origen se le asigna el siguiente puntaje:

zona				denominación	puntaje
0	<=	d <=	20	Centro	20
20	<	d <=	45	medio interno	10
45	<	d <=	70	medio	5
70	<	d <=	100	medio externo	2
100	<	d <=	120	borde	1
120	<	d		fuera o falla	0



Nota: *Considerar que siempre se ejecutan los 3 tiros.*

Ejercicio 8: Torneo de arquería

Realizar una aplicación para la gestión de un torneo de arquería.

Cada participante se acerca al punto de tiro y se registra por teclado el número de competidor. Realiza los tres tiros y se registra el puntaje obtenido (basado en las reglas definidas en el ejercicio 5).

La competencia finaliza cuando se ingresa el número de participante menor o igual a cero (<=0).

8.1.-

Al final del torneo informar el número del participante ganador (aquel que obtuvo el mayor puntaje) y puntaje obtenido.

8.1.a.- En caso de competidores empardados, considerar que posee mejor puntaje aquel que tiro primero.

8.1.b.- En caso de competidores empardados, considerar que posee mejor puntaje aquel que tiro último.

8.2.-

Al final del torneo informar número del participante y puntaje obtenido, de los tres concursantes que obtuvieron el mejor puntaje.

8.2.a.- En caso de competidores empardados, considerar que posee mejor puntaje aquel que tiro primero.

8.2.b.- En caso de competidores empardados, considerar que posee mejor puntaje aquel que tiro último.

Los competidores pueden volver a tirar siempre y cuando no estén formando parte de la terna ganadora.

Ejercitación complementaria

Serie D – Máquinas de Estados – simples

La siguiente ejercitación tiene como objetivo:

- Analizar los enunciados y comprender que es lo que hay que hacer (si no sabemos que es lo que hay que hacer, cualquier cosa que hagamos … será cualquier cosa).
- Ejercitar la estrategia referida al desarrollo en base a la implementación de Máquina de Estados.
- Implementar a través de diagramas las MdE propuestas.
- Realizar el código en C de estás MdE.

Notas: solo se permite el uso de variables del tipo entero.

Los valores de puerto y pin dentro del documento son referenciales.

Para implementar estos requerimientos, cuenta con los siguientes recursos.

```
#define ON          1
#define OFF         0
typedef unsigned char uint_8t;
typedef unsigned int uint_32t;
```

```
void outDig (uint_8t port, uint_8t pin,uint_8t estado);
```

Permite activar/desactivar una salida determinada

`port/pin`: valores asociados al puerto/pin a controlar

`estado`: ON – activado/ OFF – desactivado

Valor de retorno: no posee

```
int inDig (uint_8t port, uint_8t pin);
```

Permite leer el valor de una entrada digital

`port/pin`: valores asociados al puerto/pin a leer

Valor de retorno: 0 o 1, dependiendo del valor de la entrada

```
int getInp (uint_8t port, uint_8t pin,uint_32t *valor);
```

Permite leer el valor de una entrada analógica -

`port/pin`: valores asociados al puerto/pin a leer

`valor`: ubicación (dirección) en donde se almacenará el dato leído o capturado en la entrada definida

Valor de retorno:

0 – No se posee una lectura válida

1 – En valor se deja el dato capturado de la entrada definida.

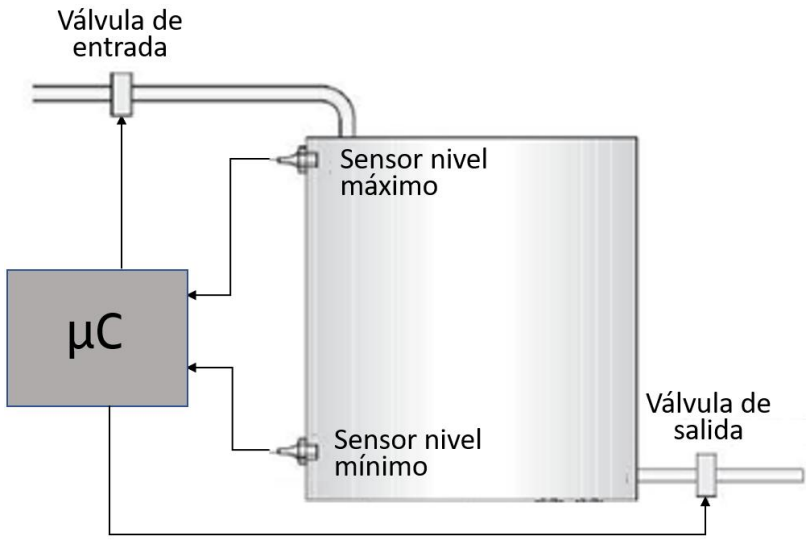
```
int time (NULL);
```

Devuelve la cantidad de segundos transcurridos desde el 1ero de enero de 1970.

Para pensar… como se chequea el funcionamiento de estos códigos.

Ejercicio 1: El tanque de agua

Ejercicio 1.1: Tanque básico



Se debe realizar una MdE que mantenga el tanque de agua lleno, sin que desborde.

El sensor de nivel máximo y mínimo se encuentran conectados a las entradas 0 y 1 respectivamente. Estos sensores están activados cuando detectan agua, y desactivados en caso contrario.

Por su parte, la válvula de entrada y la de salida, se encuentran conectadas a los relés 1 y 2 respectivamente. Al activarse el relé correspondiente se activa la válvula conectada, y esta se desactiva a desactivar el relé.

El funcionamiento básico del tanque es el que sigue:

- La válvula de salida debe estar siempre activada salvo que el sensor de nivel mínimo este sin agua.
- La válvula de entrada debe estar siempre activada salvo que el sensor de nivel máximo detecte agua.
- En caso que el sensor de nivel máximo detecte agua y el mínimo no lo detecte, se debe activar la alarma conectada al relé 3. Esta alarma se debe mantenerse encendida hasta que se reinicie el sistema, o se detecte que el sensor de nivel mínimo detecta agua nuevamente.

Recursos disponibles

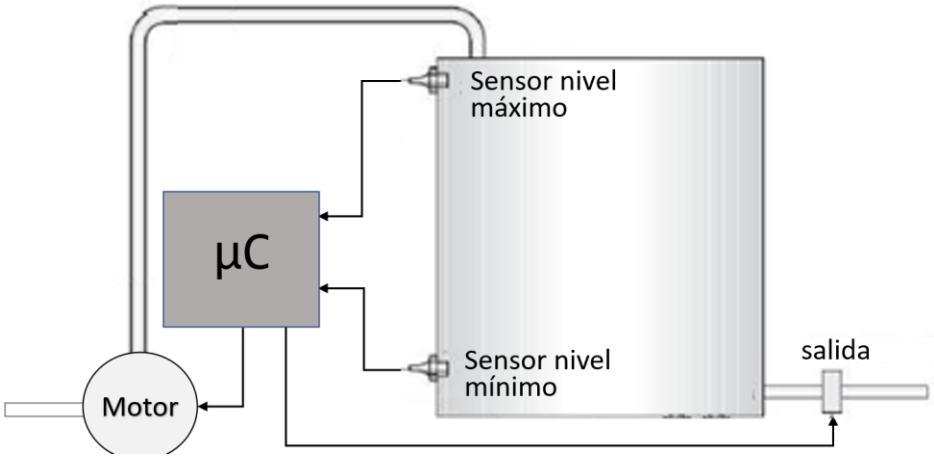
```
#define ENTRADA0    1,26   // Puerto 1, pin 26
#define ENTRADA1    1,24   // Puerto 1, pin 24
#define RELAY1      2,0    // Puerto 2, pin 0
#define RELAY2      0,23   // Puerto 0, pin 23
#define RELAY3      0,21   // Puerto 0, pin 21
```

Se pide:

a) implementar un diagrama de MdE que satisfaga el requerimiento.

b) Realizar el programa en C que represente la máquina de estados del punto a

Ejercicio 1.2: Tanque motorizado



El sistema anterior se modifica, conectando el relé 2 a un motor elevador en lugar de una válvula, lo cual agrega al sistema anterior los siguientes considerandos.

- Cuando se requiere ingresar agua al tanque se debe encender el motor el cual se debe apagar luego de 30 minutos o al detectar que el tanque este lleno.
- Una vez apagado el motor, no se puede encender hasta que hayan transcurrido 10 minutos.
- En caso de estar el sensor de nivel mínimo detectando falta de agua por más de 15 minutos, se debe activar la alarma. Esta se apaga automáticamente al detectar agua en este sensor por más de 1 minuto.

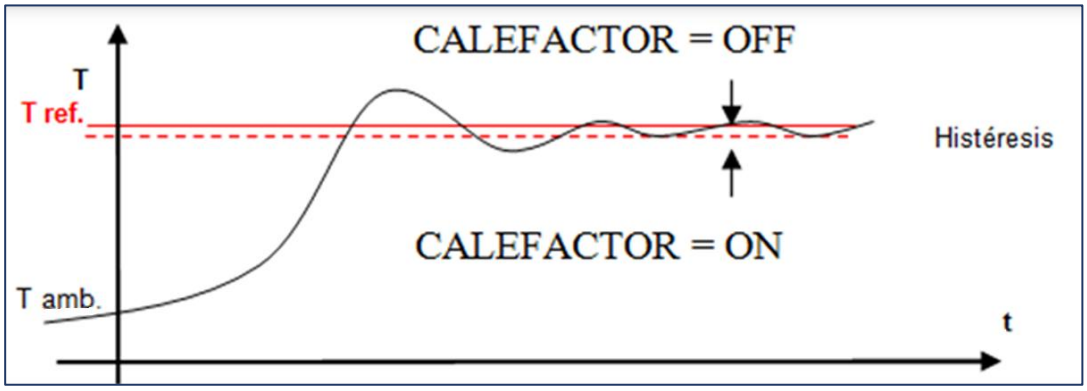
Se pide:

a) implementar un diagrama de MdE que satisfaga el requerimiento.

b) Realizar el programa en C que represente la máquina de estados del punto a

Ejercicio 2: Control de temperatura ON / OFF

Ejercicio 2.1: Control de temperatura básico



Se debe realizar una MdE que mantenga la temperatura del sistema dentro de un rango definido por la temperatura de referencia.

Se sabe que la temperatura ambiente es bastante menor a la temperatura de referencia.

Para realizar esta tarea se cuenta con un calefactor conectado a la salida del Relé 1. Al activar este calefactor la temperatura del sistema aumenta, mientras que, al desactivarlo, la temperatura empieza a descender en forma natural.

Para determinar la temperatura del sistema, se cuenta con un sensor de temperatura conectado a la entrada analógica 1. Este sensor da un valor entre 0 y 1024, en donde 0 corresponde a una temperatura de -10°C o menos, y 1024 a una temperatura de 118°C o superior.

La temperatura de referencia es de 80°C y el rango de operación requerido es entre 76 y 80°C.

Recursos disponibles:

```
#define RELAY1 2,0        // Puerto 2, pin 0
#define SENSOR_TEMP 0,6   // Puerto 0, pin 6
#define TEMP_REF_MAX 80   //°C
#define TEMP_REF_MIN 76   //°C
```

Se pide:

a) implementar un diagrama de MdE que satisfaga el requerimiento.

b) Realizar el programa en C que represente la máquina de estados del punto a

Ejercicio 2.2: Control de temperatura ON / OFF temporizado

Se pide qué al sistema anterior, se le agregue un temporizador, y cotas térmicas de seguridad, las cuales deben contemplar las siguientes consideraciones:

- Al iniciar el proceso,
  - lo lógico es que la temperatura esté por debajo de la TEMP_REF_MIN. De ser así, el tiempo para alcanzar el rango de operación no puede ser mayor a 5 minutos.
  - En caso que la temperatura este dentro del rango de operación, +/- 2°C, se debe considerar que el sistema ya se encuentra en operación.
  - En caso que la temperatura exceda los 82°C, se debe asegurar que el calefactor este apagado y esperar que la temperatura ingrese al rango de operación en no más de 6 minutos.
  - Si la temperatura excede la TEMP_CRIT_MAX, se debe asegurar que el calefactor este apagado y encender la alarma del sistema.
- Ingreso a régimen (es cuando la temperatura del sistema ingresa a oscilar dentro del rango definido)
  - Si dentro del tiempo definido al iniciar el proceso, el sistema no se encuentra dentro de la temperatura definida, se debe encender la alarma del sistema. Independientemente de esto, el control de temperatura debe seguir operando.
- En régimen

- Una vez en régimen, la temperatura no debería salir del rango definido por `TEMP_REF_MAX/ TEMP_REF_MIN`. Sin embargo, debido a la inercia del sistema, este valor puede superar estos límites en 2°C sin que esto sea necesariamente un inconveniente.
- Si la temperatura permanece fuera del rango térmico definido por más de 50 segundos, se debe encender la alarma.
- Si la temperatura sale del rango definido por TEMP_CRIT_MAX y TEMP_CRIT_MIN, se debe encender la alarma.

Recursos que se agregan:

```
#define ALARMA 0,23       // Puerto 0, pin 23
#define TEMP_CRI_MAX 90   //°C
#define TEMP_CRI_MIN 68   //°C
```

Se pide:

a) implementar un diagrama de MdE que satisfaga el requerimiento.

b) Realizar el programa en C que represente la máquina de estados del punto a

Ejercicio 2.3: Control de temperatura ON / OFF avanzado

Al sistema anterior se agrega un ventilador el cual estará conectado al relé 3. Este dispositivo debe ayudar a disminuir la temperatura del sistema cuando sea necesario.

Se pide:

a) Defina como será la operatoria de este dispositivo, es decir las condiciones por las cuales se activa y cuales las desactiva.

b) implementar un diagrama de MdE que satisfaga el requerimiento.

c) Realizar el programa en C que represente la máquina de estados del punto b

Ejercitación – Serie E

Pasamos la ubicación (dirección) de variables a las funciones

---

Modificamos varias de las funciones realizadas en las series anteriores, de manera de mejorar sus prestaciones y resolver algunas falencias o limitaciones que se presentaron en sus resoluciones.

Para probar las funciones solicitadas, será necesario que se generen programas con la posibilidad de ingresar los datos por teclado e imprimir los valores devueltos.

En ninguno de los casos las funciones solicitadas deben imprimir información. Los resultados deben imprimirse fuera de la función a partir del valor retornado por estas (salvo se especifique lo contrario)

Las funciones deben utilizar las utilizar las siguientes etiquetas comunes cuando corresponda, las cuales se deben definir en "comun.h"

```
#define SI      1
#define NO      0
#define ERR    -1
```

Ejercicio 1.a: (triángulo)

Realizar una función que a partir de la longitud de los lados recibidos de un triángulo determine:

- si se trata de un triángulo escaleno, isósceles o equilátero.
- El perímetro del triángulo
- El área del triángulo

La función debe retornar si se trata de un triángulo o no y devolver los valores solicitados.

Las funciones deben considerar la recepción de valores negativos o ceros y actuar en consecuencia.

Ejercicio 1.b: (triángulo+)

Modificar la función anterior para que los parámetros a devolver sean opcionales.

Nota: *Al definir el prototipo de una función en C, el mismo se debe cumplir al llamar a dicha función. Esto significa que, si la función debe recibir la ubicación o dirección del perímetro, ¿cómo podemos hacer para que al momento de llamar a la función tengamos la opción de pasar esta dirección o no?*

Ejercicio 2.a: (El máximo, mínimo y promedio)

Realizar una función que determine y devuelva el valor máximo, el valor mínimo y el promedio de una serie de valores ingresados por teclado. La función debe retornar la cantidad de valores ingresados por teclado.

El final del ingreso se debe producir cuando el valor ingresado sea menor a cero (<0), valor que no debe ser tenido en cuenta en los cálculos.

Notas: *Contemplar el caso que no haya ingresos válidos.*

Ejercicio 2.b: (El máximo, mínimo y promedio +)

Modificar la función anterior para que los parámetros a devolver sean opcionales.

Ejercicio 3: (ingreso en rango)

Realizar una función para el ingreso de un dato numérico tipo entero por teclado dentro de un rango el cual se recibe como argumento. Si el valor ingresado se encuentra fuera de este rango, la función debe dar la opción de reingreso y reevaluar el nuevo valor. La cantidad máxima de reintentos es de 3 oportunidades.

La función debe retornar SI, cuando se haya ingresado un valor dentro del rango y en tal caso debe devolver el valor ingresado.

En caso que no se haya ingresado un valor dentro del rango incluyendo los reintentos, la función debe retornar NO.

Notas:

- Los valores que coincidan con los límites del rango deben ser considerados dentro del rango.
- Tener en cuenta que el orden de las cotas no está predefinido, por ejemplo, el primer parámetro que reciba la función podrá ser la cota inferior o superior.

Ejercicio 4.a: (tiro oblicuo)

Realizar una función para determinar la distancia que recorre (según eje horizontal) y altura máxima que alcanza un objeto que es lanzado desde una plataforma con una velocidad y ángulo configurable.

Tanto la velocidad inicial, como su ángulo serán pasados como parámetros a la función.

La aceleración de la gravedad esta especificada a través de la etiqueta ACELERACION_GRAVEDAD como sigue

```
#define ACELERACION_GRAVEDAD 9.81
```

La misma está definida en m/s^2.

La función debe retornar SI y devolver los valores solicitados, cuando los parámetros recibidos son coherentes y permiten calcular los valores solicitados. En caso contrario, la función debe retornar NO.

Ejercicio 5: (temperatura)

Realizar una función para convertir una temperatura de una escala a otra.

Las escalas de temperatura que debe soportar la función son:

C: Centígrados o Celsius

F: Fahrenheit

K: Kelvin

R: Rankine

r: Reamur

La función debe retornar SI y devolver la temperatura en la escala solicitada si la conversión ha sido posible, caso contrario la función debe retornar NO.

Ejercicio 6: (función cuadrática)

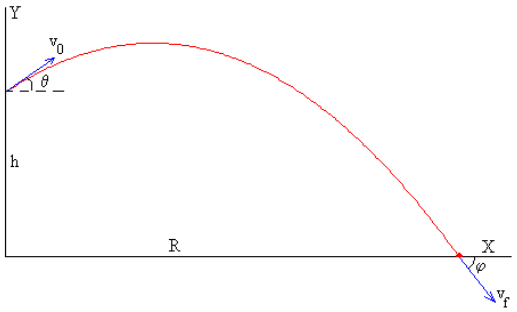
Realizar una función que determine y devuelva las raíces de una ecuación cuadrática $a \cdot x^2 + b \cdot x + c = 0$.

La función deberá recibir como mínimo los coeficientes de la ecuación y retornar la cantidad de raíces reales obtenidas.

La función debe soportar ecuaciones lineales, y contemplar cualquier valor de coeficiente que se pueda recibir.

Ejercicio 7:

Realizar una función para la resolución de ejercicios típico de tiro oblicuo, en donde las incógnitas son la distancia y altura máxima.



Los parámetros que debe recibir la función son:

- Altura inicial (m).
- Velocidad inicial (m/s).
- Angulo inicial.
- Aceleración de la gravedad (m/s^2)

La función debe retornar si las parámetros recibidos son coherentes y las incógnitas se pueden resolver, y devolver en forma opcional los valores calculados.

Ejercitación – Serie F

aritmética a nivel de bits.

Salvo se indique lo contrario, la referencia a una posición dentro de una variable se hará contando desde cero desde el dígito menos significativo.

Realizar las funciones solicitadas en un archivo .c, y el main de la aplicación en otro archivo .c (trabajando con varios archivos fuentes)

Ejercicio 1:

1.a.-

Realizar una función que retorne el valor del *nibble* de una variable de 32 bits, según se lo indique mediante la variable **n** también pasada como parámetro, considerando que el *nibble* cero se corresponde con el *nibble* menos significativo (es decir iniciando a contar desde el *nibble* menos significativo).

Algunos ejemplos:

- Valor = **0x12345678** y **n**=0 se debe retornar: **0x08**
- Valor = **0x12345678** y **n**=5 se debe retornar: **0x03**

1.b.-

Ídem anterior, pero considerando que el *nibble* cero se corresponde al *nibble* más significativo (es decir iniciando a contar desde el *nibble* más significativo)

Algunos ejemplos:

- Valor = **0x12345678** y **n**=0 se debe retornar: **0x01**
- Valor = **0x12345678** y **n**=5 se debe retornar: **0x06**

1.c.-

Fusionar las funciones anteriores en una sola, de manera que al momento de invocar la función se deba indicar si se desea trabajar en modo nibble menos significativo o más significativo.

Ejercicio 2:

2.a.-

Realizar una función que modifique una variable de 32 bits, colocando un determinado *nibble* en cero según se lo indique mediante la variable **n**, iniciando desde el nibble menos significativo.

Algunos ejemplos:

- Valor = **0x12345678** y **n**=0 el valor resultante debe ser: 0x1234567**0**
- Valor = **0x12345678** y **n**=5 el valor resultante debe ser: 0x12**0**45678

2.b.-

Modificar el anterior para que en lugar de colocar un cero en el nibble indicado, coloque un valor pasado por parámetro.

Ejercicio 3:

Realizar una función que retorne el valor del bit de una variable según se lo indique mediante el parámetro **bit**.

Ejercicio 4:

4.a.-

Realizar una función que modifique una variable recibida como parámetro, poniendo en 1 el valor del bit de dicha variable según se lo indique mediante el parámetro **`bit`**, también recibida como parámetro.

4.b.-

Realizar una función que modifique una variable recibida como parámetro, poniendo en 0 el valor del bit de dicha variable según se lo indique mediante el parámetro **`bit`**, también recibida como parámetro.

4.c.-

Realizar una función que modifique una variable recibida como parámetro, conmutando el valor del bit de dicha variable según se lo indique mediante la variable **`bit`**, también recibida como parámetro.

Ejercicio 5:

Realizar una función que modifique un bit de una variable no signada de 32 bits, colocando en 0 o en 1 el valor de dicho bit según se especifique.

Ejercicio 6:

Realizar una función que imprima un valor en formato binario.

Ejercicio 7:

Realizar una función que, utilizando aritmética de bits, indique si el valor recibido es par.

Ejercicio 8: (recomendado)

8.a.-

Analizar que hace la siguiente sentencia.

```
a^=b^=a^=b;
```

8.b.-

Antes de ejecutar el código, intente predecir que se imprimirá en el 2do printf.

```
int main(void) {
unsigned char x=0xAA;
unsigned char y=0x55;

    printf(" x:%X - y:%X\n",x,y);
    x^=y^=x^=y;
    printf(" x:%X - y:%X\n",x,y);
  return 0;
}
```

Nota: *dependiendo del compilador la sentencia propuesta (*`x^=y^=x^=y;`*) puede generar un warning.*

8.c.-

Es común que este tipo de sentencia se escriban en una macro. En tal sentido, si la sentencia bajo análisis la definimos en una macro llamada XYZ, el código anterior quedaría:

```
int main(void) {
unsigned char x=0xAA;
unsigned char y=0x55;
```

```
    printf(" x:%X - y:%X\n",x,y);
    XYZ (x,y);
    printf(" x:%X - y:%X\n",x,y);
  return 0;
}
```

Implementar la macro XYZ.

8.d.-

Realizar una función que realice la misma tarea que la macro.

8.e.-

Analice y documente pros y limitaciones de cada método (macro vs función).

8.f.-

Ejecutar el proceso de XOR sucesivos (a^=b^=a^=b;) con variables de punto flotante de simple precisión (float) - es con un poco de trampa -.

Ejercicio 9: (recomendado)

Realizar una función que imprima en pantalla los valores característicos de una variable tipo punto flotante, pudiendo esta ser de simple, doble o cuádruple precisión.

```
void prt_flotantes (const void *pf, int mode);
```

En donde **pf** es el puntero al dato y **mode** hace referencia al tipo de dato apuntado

Ejercicio 10: (desafío)

Determinar si un procesador es del tipo Big endian o Little endian.

Ejercitación – Serie G

Valores aleatorios y variables static

Realizar las funciones solicitadas en un archivo .c, y el main de la aplicación en otro archivo .c

---

Ejercicio 1: (el dado)

Implemente una función que genere y retorne números aleatorios del 1 al 6.

```
int  dado (void);
```

Ejercicio 2: (cubilete)

Implementar una función que imprima en pantalla la resultante de haber lanzado 5 dados.

Ejercicio 3: (cubilete')

Modificar la función anterior de manera de poder realizar una serie de lanzamientos e indicar en cada lanzamiento que dados volver a arrojar y cuáles no.

La indicación de que dados volver a lanzar se debe realizar con una variable de 8bits, a la cual se le debe asignar un bit a cada dado (total 5 bits).

El bit en 1 (uno) significa que el dado no debe ser lanzado, y en 0 (cero) que sí.

Si el parámetro es cero (todos sus bits en cero) significa que se deben arrojar los 5 dados.

La función debe devolver la cantidad de dados arrojados.

Ejercicio 4: (cubilete'')

Agregar a la función anterior la posibilidad de devolver el resultado de la tirada (es decir el valor de todos los dados) en una sola variable.

Nota: Se debe asignar una cantidad de bits y posición dentro de una variable a cada dado. La variable debe ser del tamaño adecuado para albergar los 5 dados.

Ejercicio 5: (jugada)

Realizar que pasándole el valor devuelto por la función del ejercicio 4 (resultado de la tirada), imprima y retorne si se trata de una generala, un poker, full o escalera.

Ejercicio 6: (jugada numérica)

Realizar que pasándole el valor devuelto por la función del ejercicio 4 (resultado de la tirada) y un valor de dado a evaluar, retorne la cantidad de veces que ese dado aparece en la tirada (valor de 0 a 5).

Ejercicio 7: (Solitario)

Realizar una aplicación para jugar a la generala en modalidad obligada (aquella que primero se hace el 1, luego el 2 y así hasta el 6, pasando posteriormente a la escalera, full, poker, generala y en algunos casos generala doble).

Luego de cada turno o jugada se debe imprimir el puntaje obtenido y el acumulado.

Ejercicio 8: (partida)

Modificar la aplicación para permitir jugar a 2 personas en simultáneo.

Finalizado el juego la aplicación debe indicar al participante ganador.

Ejercicio 9: para pensar

Que modificaciones debería considerar para una aplicación en donde los participantes decidan luego de haber realizado los tiros correspondientes a un turno, elija la jugada a anotar.

Que modificaciones debería hacer para permitir una cantidad variable de participantes. Por ejemplo 10 o 3.

Como se podría hacer un torneo.

Ejercicio 10: (punto flotante)

Realizar una aplicación que permita generar un número aleatorio de punto flotante de simlple precisión acotado en rango.

Ejercitación – Serie H

Primeros ejercicios de Arrays

Por cada ejercicio se solicitan dos versiones, una versión con lógica de Arrays y otra versión con aritmética de punteros

Solo las funciones que tengan que interactuar con el usuario (ingreso de datos o presentar información) deben imprimir datos en pantalla. Las demás funciones deben abstenerse de imprimir mensajes en pantalla.

En los casos que solo se solicitan funciones, agregar el soporte correspondiente para probar las mismas.

Es recomendable que las aplicaciones las desarrolle bajo el modo de múltiples archivos fuente

---

Ejercicio 1:

Desarrolle un programa que solicite el ingreso de las calificaciones del primer parcial de los alumnos de Informática I. El ingreso de calificaciones finalizará cuando se ingrese un once (11). Tener en cuenta que las calificaciones deberán ser números comprendidos entre 0 y 10 (0 significa ausente), por lo que se debe validar la carga de datos. Una vez finalizada la carga de datos, se pide computar e imprimir en pantalla los siguientes valores:

a) La cantidad de aprobados, desaprobados y ausentes (nota de aprobación: especificada mediante un define).
b) El valor promedio de las calificaciones (teniendo en cuenta los ausentes).
c) El valor promedio de las calificaciones (sin tener en cuenta los ausentes).
d) La calificación máxima ingresada.
e) La calificación mínima ingresada (sin considerar el ausente).

En donde, el ítem (a), se podrá implementar en una o varias funciones; los ítems (b) y (c), se deberán resolver con una sola función, la cual a través de algún parámetro se le indique si debe calcular (b) o (c). De igual manera se deberá resolver (d) y (e).

Ejercicio 2:

Realizar una función que copie a otro array, una sola instancia de cada valor en el array original, con las siguientes premisas.

- La copia se realiza en el orden que aparecen
- La copia se realiza en forma ordenado ascendente.
- La copia se realiza en forma ordenado descendente.

Ejercicio 3:

Realizar una función que elimine de un array todos los valores pares y los valores negativos

Ejercicio 4:

Realizar una función que elimine de un array todos los valores repetidos

Ejercicio 5:

Realice una función que complete en un array de datos tipo int, los primeros N números primos gemelos.

La función debe retornar la posición de inicio del array.

```
int * ingreso_array ( ----- completar ----);
```

Ejercicio 6:

Realizar una función que reciba un array de elementos enteros y coloque los elementos impares en un 2do array y los pares en un tercer array.

La función deberá informar de alguna manera la cantidad de elementos en cada array.

Ejercicio 7:

Realice una función que permita el ingreso de datos por teclado, con las siguientes condiciones.

- Los valores repetidos deben ser descartados
- Los elementos dentro del array deben introducirse en forma ordenada (ascendente)
- El fin de ingreso se da cuando se ingresa por teclado 3 veces el mismo valor (que ya estaba en el array) o se completó el tamaño del array.
- La función debe retornar la cantidad de elementos ingresados en el array.

Ejercicio 8:

Modificar el ejercicio anterior, de manera que la función se puede invocar en varias ocasiones, ya sea con el array vacío o con datos.

Ejercicio 9:

Realice una función que reciba un array, la cantidad de elementos del array y una condición de impresión parcial/total con las siguientes consideraciones:

```
void prt_array ( int *arr, int cant, int cond);
```

arr: Array a imprimir

cant: cantidad de elementos en el array

cond: Condicional de impresión total parcial

- Los datos se imprimen en 4 columnas.
- Si **cond** es:
    - igual a 0, se deben imprimir el primer valor, el valor central y el último valor del array.
    - Si su valor es mayor a la doceava parte de elementos que posee el array (**cond** > **cant**/12), se debe imprimir todo el array.
    - Cualquier otro valor, se deben imprimir los primeros, últimos **cond** elementos del array y los **cond** elementos que estén en el centro del array, ajustado a la cantidad de columnas (por ejemplo, si **cond**==10, serían 3 columnas y por ende se deberían imprimir 12 valores)

Ejercicio 10: (Arrays asociados)

Realizar una aplicación que maneje un código de producto (int o unsigned int) y un precio (float).

La aplicación debe permitir el ingreso de datos en diferentes ocasiones.

No puede haber dos elementos con el mismo código de producto.

Informar según se requiere:

- Valor promedio de los precios
- Listados de los valores por sobre o por debajo del promedio
- Permitir modificar el precio de un producto.
- Listado de los 3 productos más costosos y más económicos (no se permite usar ordenamiento de arrays).

Ejercitación – Serie H +

Ejercicios de Arrays II

Búsqueda y ordenamiento en Arrays

Conceptos que se agregan en esta serie: manejo avanzado de Arrays.

Está prohibido el uso de variables globales.

Utilizar los algoritmos de ordenamiento clásicos como Intercambio o Burbujeo.

Implementar las aplicaciones y funciones complementarias que permita visualizar el resultado de las funciones realizadas.

Esta guía se enfoca en obtener un manejo avanzado de Arrays e indirecciones.

Ejercicio 1:

Realizar una función que permita determinar si un array de números enteros (por ejemplo, int) está o no ordenado. Y en caso de estarlo, indique el sentido de ordenamiento (ascendente, descendente).

Considerar:

- Arrays vacíos (es decir con cantidad cero)
- Arrays con un solo dato
- Arrays con valores repetidos (incluso todos los valores del array podrían ser iguales)

Para pensar y definir:

En estos casos, ¿está el array ordenado o no? seguramente no está desordenado, pero no sabría si esta ordenado. Pero si esta ordenado, ¿sería ascendente o descendente?

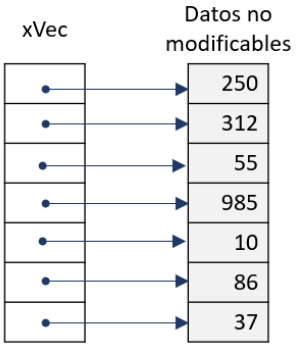
Ejercicio 2:

Modificar el punto anterior para trabajar con Arrays de strings.

Ejercicio 3:

Realizar una función que permita imprimir los datos que están siendo apuntados por un array de punteros a datos entero.

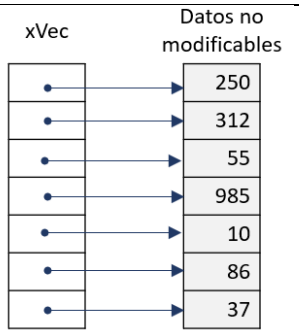
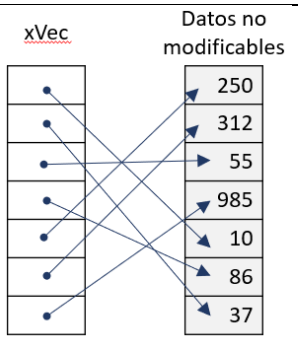
En base a la figura, lo que se estará recibiendo el array xVec, y lo que deberemos imprimir son los datos no modificables.



Ejercicio 4:

Realizar una función que permita ordenar un array de punteros a datos entero, considerando que los elementos a los que se está apuntando, no se pueden modificar (por ejemplo, están en área de código, o en memoria tipo ROM).

Basados en el gráfico del ejercicio anterior, deberíamos tener:

Array antes del ordenamiento	Array luego del ordenamiento
	

Si luego de este ordenamiento llamásemos a la función del ejercicio anterior, deberíamos ver la salida de los datos ordenada: 10; 37; 55; 86; 250; 312; 985

Ejercicio 5:

Realizar una función que permita ordenar un array de strings. Considerar que los strings en sí, están en área de código, y por ende no se pueden modificar.

- .

Ejercitación – Serie I

Strings

Conceptos que se agregan en esta serie: Arrays como cadena de caracteres conocidos habitualmente como Strings.

Preferentemente utilice lógica de punteros para desarrollar sus funciones

---

Para probar las funciones solicitadas, será necesario generar programas y funciones con la posibilidad de ingresar los datos e imprimir los resultados según corresponda.

Salvo se especifique, las funciones solicitadas NO deben imprimir ningún tipo de dato o información. Si fuese necesario imprimir algún resultado, estos deberán imprimirse por fuera de la función a partir del valor retornado.

Recuerden que no está permitido el uso de variables globales.

Ejercicio 1:

Implemente y pruebe su propia versión de las siguientes funciones declaradas en string.h, antecediendo al nombre de las mismas el prefijo **my_**.

```
int strcmp (const char *s1, const char *s2);
int strncmp(const char *s1, const char *s2, size_t n);
```

Respete sus prototipos y utilice el man para saber cómo se debe comportar cada una de ellas.

Para verificar el funcionamiento de las funciones realizadas, ejecute el siguiente código, el cual deberá tener la misma salida que se visualiza luego del mismo.

```
#include <stdio.h>
int main()
{
char s1[]="abcdefghaij";
char s2[20]="ABZ", *sx="por las dudas.";
int aux,i;
    for (i=0; i<=3;i++) {
        switch(i){
            case 1: sx=s1; break;
            case 2: sx="abcde"; break;
            case 3: sx=s2; break;
        }
        printf ("\nComparamos s1: %s\n\ty sx: %s \n",s1,sx);
        aux= my_strcmp(s1,sx);
        if (!aux)
            printf (" --> son iguales\n",sx);
        else{
            printf (" -- s1 es %s que sx",(aux<0)?"menor":"mayor");
            printf ("\ny si solo comparamos las primeras 4 letras\n");
            if (my_strncmp(s1,sx,4))
                printf (" --> tambien son distintos\n",sx);
            else
                printf (" --> son coincidentes\n",sx);
        }
    }
return 0;
}
```

```
Comparamos s1: abcdefghaij
       y sx: por las dudas.
 -- s1 es menor que sx
y si solo comparamos las primeras 4 letras
 --> tambien son distintos
Comparamos s1: abcdefghaij
       y sx: abcdefghaij
 --> son iguales
Comparamos s1: abcdefghaij
       y sx: abcde
 -- s1 es mayor que sx
y si solo comparamos las primeras 4 letras
 --> son coincidentes
Comparamos s1: abcdefghaij
       y sx: ABZ
 -- s1 es mayor que sx
y si solo comparamos las primeras 4 letras
 --> tambien son distintos
```

Ejercicio 2:

Implemente y pruebe su propia versión de las siguientes funciones declaradas en string.h, antecediendo al nombre de las mismas el prefijo **my_**.

```
size_t strlen(const char *s);
char *strcpy(char *dest, const char *src);
char *strncpy(char *s1, const char *s2, size_t n);
char *strcat(char*s1, const char *s2);
char *strncat(char*s1, const char *s2, size_t n);
```

Respete sus prototipos y utilice el man para saber cómo se debe comportar cada una de ellas.

Está prohibido el uso de cualquier función preexistente.

Para verificar el funcionamiento de las funciones implementadas, ejecute el siguiente código, el cual deberá tener la misma salida que se visualiza luego del mismo.

```
#include <stdio.h>
#define ABZ "ABC-XYZ"

int main()
{
char s1[80]="abcdefghaijklmnopqrstuvwxyz";
char s2[80];
char *s3="1234567890-";

    printf ("Los largos de s1 y s3 son: %d y %d\n",my_strlen(s1), my_strlen(s3));

    my_strcpy(s2,s1);
    my_strcat(s2,s3);
    my_strncat(s1,s3,4);
    printf ("En s1 tengo:[%s]\ny en s2:[%s]\n",s1,s2);

    my_strncpy (s1+10,ABZ, my_strlen(ABZ));
    printf ("Ahora en s1 tengo:[%s]\n ",s1);
return 0;
}
```

```
Los largos de s1 y s3 son: 27 y 11
En s1 tengo:[abcdefghaijklmnopqrstuvwxyz1234]
y en s2:[abcdefghaijklmnopqrstuvwxyz1234567890-]
Ahora en s1 tengo:[abcdefghaiABC-XYZqrstuvwxyz1234]
```

Ejercicio 3:

Implemente y pruebe su propia versión de las siguientes funciones declaradas en string.h, antecediendo al nombre de las mismas el prefijo **my_**.

```
char *strchr(char *s, int c);
char *strstr(const char *s1, const char *s2);
```

Respete sus prototipos y utilice el man para saber cómo se debe comportar cada una de ellas.

Está prohibido el uso de cualquier función preexistente, a excepción de las implementas en puntos anteriores.

Para verificar el funcionamiento de las funciones realizadas, ejecute el siguiente código, el cual deberá tener la misma salida que se visualiza luego del mismo.

```
#include <stdio.h>
void informamos (const char *s1,const char *s2)
{
    if (s2) // o s2!=NULL
    {
        printf ("\nEncontrado en la posicion %d",s2-s1);
        printf ("\ny a partir de ahi tenemos: %s",s2);
    }
    else
        printf ("\nNo encontrado en el texto");
}
int main()
{
char *s1="Estamos desarrollando funciones";
char *sx, s2[]="carro", aux;
    printf ("s1: %s",s1);
    for (aux='e'; aux<='h'; aux++)
    {
        printf ("\nBuscamos el caracter:%c",aux);
        sx=my_strchr(s1,aux);
        informamos (s1,sx);
    }
    printf ("\nAhora buscamos el texto: %s",s2);
    sx=my_strstr(s1,s2);
    informamos (s1,sx);
    printf ("\nAhora buscamos el texto: %s",s2+1);
    sx=my_strstr(s1,s2+1);
    informamos (s1,sx);
return 0;
}
```

```
s1: Estamos desarrollando funciones
Buscamos el caracter:e
Encontrado en la posicion 9
y a partir de ahi tenemos: esarrollando funciones
Buscamos el caracter:f
Encontrado en la posicion 22
y a partir de ahi tenemos: funciones
Buscamos el caracter:g
No encontrado en el texto
Buscamos el caracter:h
```

```
No encontrado en el texto
Ahora buscamos el texto: carro
No encontrado en el texto
Ahora buscamos el texto: arro
Encontrado en la posicion 11
y a partir de ahi tenemos: arrollando funciones
```

Ejercicio 4:

Implemente y pruebe su propia versión de la siguiente función declarada en string.h, antecediendo al nombre de la misma el prefijo **my_**.

```
size_t strspn(const char *s1, const char *s2);
```

Respete su prototipo y utilice el man para saber cómo se debe comportar cada una de ellas.

Está prohibido el uso de cualquier función preexistente, a excepción de las implementas en puntos anteriores.

Para verificar el funcionamiento de la función desarrollada, ejecute el siguiente código, el cual deberá tener la misma salida que se visualiza luego del mismo.

```
#include <stdio.h>
int main()
{
   char s1[] = "Hola a todos";
   char *sx = "al Ho";
   char *sy = "alHo";
   char *sz = "axHo";

   printf( "s1=[%s] \n", s1 );
   printf( "spn_sx: %d  sx=[%s]\n", my_strspn(s1, sx), sx);
   printf( "spn_sy: %d  sy=[%s]\n", my_strspn(s1, sy), sy);
   printf( "spn_sz: %d  sz=[%s]\n", my_strspn(s1, sz), sz);
return 0;
}
```

```
s1=[Hola a todos]
spn_sx: 7  sx=[al Ho]
spn_sy: 4  sy=[alHo]
spn_sz: 2  sz=[axHo]
```

Ejercicio 5:

Implemente y pruebe su propia versión de la siguiente función declarada en string.h, antecediendo al nombre de la misma el prefijo **my_**.

```
char *strpbrk(const char *str1, const char *str2);
```

Respete su prototipo y utilice el man para saber cómo se debe comportar cada una de ellas.

Está prohibido el uso de cualquier función preexistente, a excepción de las implementas en puntos anteriores.

Para verificar el funcionamiento de la función desarrollada, ejecute el siguiente código, el cual deberá tener la misma salida que se visualiza luego del mismo.

```
int main()
{
   char s1[] = "Hola a todos, hoy es ...";
   char *sx = "xyz123";
   char *sy = "bd .";
   char *sz = "bruc";

   printf( "s1=[%s] \n", s1 );
   printf( "sx=[%s] : %s\n", sx,my_strpbrk(s1,sx));
   printf( "sy=[%s] : %s\n", sy,my_strpbrk(s1,sy));
   printf( "sz=[%s] : %s\n", sz,my_strpbrk(s1,sz)); // antiguamente esta línea hubiese hecho caer a la aplicación
```

```
return 0;
}
```

```
s1=[Hola a todos, hoy es ...]
sx=[xyz123] : y es ...
sy=[bd .] :  a todos, hoy es ...
sz=[bruc] : (null)
```

Ejercicio 6:

Desarrollar las siguientes funciones

```
char *my_strLower (char *str1, const char *str2);
char *my_strnLower(char *str1, const char *str2,int n);
char *my_strUpper (char *str1, const char *str2);
char *my_strnUpper(char *str1, const char *str2,int n);
```

Las cuales copiarán el contenido de **s2** a **s1** transformando todos los caracteres a minúscula (Lower) o a mayúscula (Upper), según corresponda.

En todos los casos, la función devolverá el valor de **s1**.

En los casos de `my_strnLower` y `my_strUpper`, se transferirán solo **n** caracteres, y se agregará el finalizador de strings ('\0').

Por ejemplo, dado el siguiente valor de str2, "El dos (2), es Menor que el 5 (cinco)", tendremos las siguientes salidas en cada caso:

`puts (my_strLower (s1,str2));`	el dos (2), es menor que el 5 (cinco)
`puts (my_strnLower(s1,str2,10));`	el dos (2)
`puts (my_strUpper (s1,str2));`	EL DOS (2), ES MENOR QUE EL 5 (CINCO)
`puts (my_strnUpper(s1,str2,9));`	EL DOS (2

Ejercicio 7:

7a.- Desarrollar una función que invierta el sentido del texto de forma tal que quede de atrás hacia adelante.

```
int my_strInvert (char *str1);
```

Por ejemplo, si se recibe el texto "Hola mundo", la salida deberá ser "odnum aloH".

La función debe retornar la cantidad de caracteres del string, y debe soportar recibir NULL como parámetro, en cuyo caso retornará 0 (cero).

7b.- Realice una función que reciba un strings y devuelva el carácter que más veces se repite. En caso que haya más de un carácter que cumpla esta condición, se debe retornar aquel que posea el mayor código ASCII.

Ejercicio 8:

8.a.- Desarrollar una función para reemplazar un determinado carácter de un texto por otro.

Por ejemplo, si tenemos el texto: "Hola, ¿cómo están todos?"

Y se solicita cambiar la letra 'o' por la 'U', el texto debería quedar: "HUla, ¿cómU están tUdUs?"

El valor de retorno será la cantidad de cambios realizados, en este caso: 4.

La función debe soportar recibir NULL como parámetro, en cuyo caso retornará 0 (cero).

Los cambios se realizan sobre el mismo string.

8.b- Ídem al punto a, pero para reemplazar cadenas de caracteres.

Ejercicio 9:

Basado en el las funciones realizadas en el punto 1, realice las modificaciones necesarias para que las funciones de comparación no sean case sensitive. Es decir que Alfa sea igual a alfa o ALFA.

```
int my_strxcmp (const char *s1, const char *s2);
int my_strxncmp(const char *s1, const char *s2, size_t n);
```

Ejercicio 10:

Realice una aplicación que permita ingresar varios nombres y que finalizado el mismo, imprima:

- El nombre que este primero según orden alfabético
- El nombre que este último según orden alfabético

El fin de ingreso se da cuando dentro del texto ingresado exista un número.

El ingreso debe permitir incluir espacios como separación de nombres.

La evaluación de primero/último no debe ser case sensitive

El largo máximo de ingreso queda a criterio del desarrollador.

Serie J

Argumentos del main o Arrays de strings

Conceptos que se agregan en esta serie: Configuración de una aplicación a través de parámetros de consola.

Como ya se sabe, está prohibido el uso de variables globales.

Dentro de esta guía se permite el uso de las funciones que provee el lenguaje, como ser `string.h`. Respecto a este último archivo de cabecera, se recomienda que una vez se haya resuelto la guía reemplacen las funciones del `string.h` por las funciones desarrolladas en la serie H y evalúen el comportamiento de las aplicaciones (no debería haber cambios).

Introducción:

Uno de los usos de los argumentos del **`main`**, entre tantos otros, es el de configurar algún comportamiento de nuestra aplicación. Para ello, muchas veces se suelen ingresar parámetros de un carácter antecedidos de un guion '-'.

Se solicita hacer una aplicación que detecte estos parámetros ingresados por consola, actualice las variables asociadas a estos parámetros e imprima sus valores.

Si se llegase a ingresar un parámetro sin este guion o alguno de los parámetros ingresados no estuviese dentro de la lista de parámetros válidos, se debe imprimir un mensaje de error y terminar la aplicación.

Ejercicio 1:

Implementar una aplicación que permita configurar los siguientes parámetros

Parámetros unarios

Variable interna	Tipo	Valor por omisión	Parámetro	Impacto
Paridad	char	0 (par)	-npar	Pone a paridad en 1 (impar)
Borde	char	0 (no)	-brd	Pone a borde en 1 (si)
profundidad	char	1 (si)	-npf	Pone a profundidad en 0 (no)

Parámetros con valores

Variable interna	Tipo	Valor por omisión	Parámetro	Valores posibles
color_letra	char *	"blanco"	-cl	Texto
color_fondo	char *	"negro"	-cf	Texto
Brillo	int	25	-b	entre 0 y 100
contraste	int	33	-c	entre 0 y 100
Alto	int	12	-at	entre 4 y 30
ancho	int	5	-an	entre 4 y 10

Sí un parámetro es repetido tomar como válido el último. Imprimir todos los parámetros de manera de corroborar el resultado del ingreso.

En caso de un parámetro inválido, informar y detener la ejecución del programa.

Ejercicio 2:

Modificar el ejercicio anterior para permitir configurar a los parámetros unarios independientemente del valor por omisión.

Parámetros unarios

Variable interna	Tipo	Valor por omisión	Parámetro	Impacto
Paridad	char	0 (par)	-npar	Pone a paridad en 1 (impar)
			-par	Pone a paridad en 0 (par)
Borde	char	0 (no)	-brd	Pone a borde en 1 (si)
			-nbrd	Pone a borde en 0 (no)
profundidad	char	1 (si)	-npf	Pone a profundidad en 0 (no)
			-pf	Pone a profundidad en 1 (si)

Sí se ingresan parámetros contrapuestos, tomar como válido el último ingresado.

Ejercicio 3:

Modificar el ejercicio anterior para que los parámetros de color (tipo Texto) solo puedan tomar alguno de los siguientes valores: azul, rojo, verde, celeste, blanco, negro, gris, rosa, amarillo, marrón y turquesa

Ejercicio 4: Ejercicio Adicionales:

- Soportar la opción -h o --h (help) e imprima la ayuda (listado de los parámetros).
- Que haya parámetros que sean mandatorios (que deban ser ingresados sí o sí) y otros opcionales.
- Que los colores puedan ser ingresados en mayúsculas o minúsculas
- Que los colores puedan ser ingresados como texto o su equivalente numérico.
- Que los colores puedan aceptar nombres compuestos como ser verde claro.
- Que algunos parámetros acepten un nombre abreviado y otro extenso (por ejemplo -b / -brillo)
- Que no sea necesario el símbolo guion (-) para definir los parámetros

Existen muchas opciones más que utilizan a diario, como ser los parámetros de los comandos gcc, cp, cd, rm, etc. Queda a su libre albedrío buscar más opciones y practicar.

Ejercitación – Serie K

Memoria dinámica

Conceptos que se agregan en esta serie: Manejo de memoria en forma dinámica, uso del Heap y consideraciones generales respecto al manejo de recursos (liberarlos cuando ya no se requieren).

Como ya se sabe, está prohibido el uso de variables globales.

Ejercicio 1:

Realizar una aplicación para gestionar el nombre de personas junto a su número de legajo.

Cada persona tendrá su nombre y apellido, junto a un legajo.

Tanto el nombre como el apellido pueden contener varias palabras separada por espacios.

La aplicación debe permitir ingresar datos, consultar y eliminar bajo las siguientes condiciones. Estas acciones se pueden repetir varias veces durante la ejecución de la aplicación.

Detalles a considerar:

- Se desconoce la cantidad máxima de datos a ingresar.
- Cada vez que se inicia un proceso de ingreso (o reingreso), la aplicación debe consultar al usuario, cuantos datos quiere ingresar (***uso de realloc – ver el man***).
- Más allá de contar con una cantidad máxima definida, se debe poder interrumpir el ingreso antes de alcanzar dicha cantidad.
- No debe haber legajos repetidos. En caso de querer ingresar un legajo que ya se encuentre ingresado, se debe negar su ingreso e informar al usuario la situación.
- En caso de ingresar un usuario con el mismo nombre y apellido, alertar al usuario y consultarle si quiere continuar con el ingreso (independiente de si se ingresó con minúsculas o mayúsculas).
- El largo máximo de un nombre y de un apellido nunca serán mayor a 80 caracteres (cada uno de ellos). Solo debe aceptar caracteres alfabéticos, espacios y apóstrofes entre otros. De existir más de un espacio de separación entre las palabras que conforman el nombre o el apellido, el excedente se debe eliminar.
- La memoria utilizada para almacenar estos datos debe ajustarse a la cantidad necesaria.
- Para eliminar un usuario, se debe ingresar su legajo.
- En caso que luego de un proceso de ingreso o eliminación, la capacidad de los bloques de memoria posea espacio para más de 20 datos disponibles, el sistema debe reajustar los tamaños de los bloques (***uso de realloc – ver el man***).
- En caso de error al solicitar memoria, debe informarle al usuario esta condición y continuar con la aplicación normalmente.

Ejercicio 2:

Modificar el punto anterior, de manera que cuando se tenga que hacer un ingreso (o reingreso), el usuario no tenga que ingresar la cantidad de datos que desea entrar.

Ejercicio 3:

Modificar el punto anterior de manera que cada vez que se tenga que solicitar memoria, esta petición se realice optimizando los tiempos de procesamiento del S.O., esto es, no pedir un solo bloque, sino que en su lugar pedir memoria para poder ingresar varios datos. Este valor debe estar declarado en la etiqueta QTTY_MEMO_ALLOC.

Ejercicio 4:

Modificar a la aplicación realizada de manera tal que tanto el nombre como el apellido se ingresen en forma directa en la memoria dinámica, considerando utilizar solo la capacidad necesaria de memoria.

Ejercicio 5:

Modificar el ejercicio anterior, para permitir eliminar un registro por nombre, por apellido o por nombre y apellido. En caso de existir más de uno, se debe dar la opción de eliminar uno en particular (por ejemplo, a través del legajo) o todos.

Ejercicio 6: **Liberar recursos**

Agregar al anterior la opción de borrar todos los elementos.

Ejercicio 7: **Prueba operativa**

7.1.-

Modificar la aplicación obtenida en el punto anterior de manera de contar con un set de datos que permita evaluar el comportamiento de la aplicación sin ingresar datos por teclado.

El set de datos, debería probar/testear todas las opciones de la aplicación.

Se debe considerar que, en el paso previo a la finalización de la aplicación, no queden recursos tomados.

7.2.-

Meter el código de la aplicación generada dentro de un while (1), de manera que su ejecución sea perpetua.

Una forma simple de hacer esto, es modificar el nombre de la función main por main_2, y generar un nuevo main que llame al anterior.

Por ejemplo:

```
int main ()
{
     while (1)
        main_2();
return 0;
}
```

o

```
int main (int argc, char **argv, char ** env)
{
     while (1)
        main_2(argc, argv, env);
return 0;
}
```

Puede verificar el estado de la memoria desde otro terminal, por ejemplo, con los comados top o free entre otros. Si bien la ocupación del procesador podría subir drásticamente, la ocupación de memoria debería permanecer dentro de un rango estable.

7.3.-

Introducir en el código a través de compilación condicional, condiciones controladas de fallo para verificar el comportamiento de la aplicación ante estas condiciones.

Como mínimo considerar:

- Error al pedir memoria para nuevos registros
- Error al pedir memoria para el nombre y/o apellido.

Ejercitación – Serie K+

Memoria dinámica

Además de poder desarrollar nuestras funciones y aplicaciones, en muchos casos tendremos que intervenir en códigos de terceros, e implementar nuestras modificaciones.

En los siguientes ejercicios/códigos se encontrarán con algunos inconvenientes que deberán corregir, sean estos de memoria dinámica u otro tema.

Los códigos acá presentados pueden copiarse o descargarse del aula virtual (serie K+ codes)

Ejercicio 1:

El siguiente código presenta varias fallas, además de imprimir valores incorrectos y no finalizar en forma correcta. Se solicita identificar estos errores y corregirlos.

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

typedef struct {
  int code, cant;
  char *desc;
} data_t;

int setData(data_t *pd, int cod, int qtt, char *str);
void prtData(const data_t *pd);
void freeData(const data_t *pd);

int main() {
  data_t da, db, dc;

  setData(&da, 2456, 365, "Colector de base abstracta");
  setData(&db, 0056, 126, "Bromuro de sustrato metalico A o B (56:A / 57:B)");

  if (!da.desc || !db.desc) {
    fprintf(stdout, "Error grave de memoria\n");
    fprintf(stdout, "La aplicación finaliza\n");
    return 1;
  }

  dc = db;
  dc.code++;
  dc.cant++;

  fprintf(stdout, "----------------------------\n");

  prtData(&da);
  prtData(&db);
  prtData(&dc);
```

```
  fprintf(stdout, "----------------------------\n");

  freeData(&da);
  freeData(&db);
  freeData(&dc);
  return 0;
}

int setData(data_t *pd, int cod, int qtt, char *str) {
  int aux = strlen(str) + 1;
  pd->code = cod;
  pd->cant = qtt;

  if ((pd->desc = (char *)malloc(aux * sizeof(char)))) {
    strcpy(pd->desc, str);
    return 1;
  }
  return 0;
}

void prtData(const data_t *pd) {
  fprintf(stdout, "code: %5d | cant: %4d ", pd->code, pd->cant);
  fprintf(stdout, "| %s\n", pd->desc);
}

void freeData(const data_t *pd) { free(pd->desc); }
```

Ejercicio 2:

Por favor, revisar el siguiente código que parecería tener algunos inconvenientes.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

typedef struct {
  int len, size;
  char *str;
} data_t;

int setData(data_t *pd, char *str);
int addData(data_t *pd, char *str);
void freeData(const data_t *pd);

int main() {
  data_t db = {0, 0, NULL};
  data_t da;
  int aux = 0;

  aux = setData(&da, "Buenos días,\n");
  aux += addData(&da, "Este es un codigo que presenta algunos problemas.");
  aux += addData(&da, "Te animas a encontrarlos y corregirlos.\n");
  aux += addData(&da, "Quizas debas replantear parte de la solución.\n");
  if (aux != 4) {
    fprintf(stdout, "Error de memoria. Chau!!\n");
    return 1;
  }

  fprintf(stdout, "----------------------------\n");
```

```
  fprintf(stdout, ":--> %s\n", da.str);
  fprintf(stdout, "----------------------------\n");

  aux = 0;
  aux = addData( &db, "Bien, en este caso estamos iniciando con un addData en form
directa\n");
  aux += addData(&db, "Podemos hacer esto?.\n");
  aux += addData(&db, "Parecería que si, pues esto no se cuelga, por lo que debería
estar bien.\n");
  aux += addData(&db, "Si. parece que esta perfecto!!!.\n");
  if (aux != 4) {
    fprintf(stdout, "Error de memoria. Chau!!\n");
          free(da.str);
    return 1;
  }
  fprintf(stdout, ":--> %s\n", db.str);
  fprintf(stdout, "----------------------------\n");
  aux = setData(&da, "Estamos llegando al final de codigo,\n");
  aux += addData(&da, "Quien dijo que este codigo presenta algunos problemas.");
  aux += addData(&da, "Por lo que veo anda perfecto.\n");
  aux += addData(&da, "o quizás no? tendra eso que le llaman \"vicios ocultos\".\n");
  if (aux != 4) {
    fprintf(stdout, "Error de memoria. Chau!!\n");
    return 1;
  }
  fprintf(stdout, ":--> %s\n", da.str);
  fprintf(stdout, "----------------------------\n");

  freeData(&da);
  freeData(&db);
  return 0;
}

#define SIZE_BLOCK 64
int setData(data_t *pd, char *str) {
  int aux = strlen(str) + 1;
  aux = SIZE_BLOCK * ((aux / SIZE_BLOCK) + 1);

  if ((pd->str = (char *)malloc(aux * sizeof(char)))) {
    strcpy(pd->str, str);
    pd->len = strlen(str);
    pd->size = aux;
    return 1;
  }
  return 0;
}

int addData(data_t *pd, char *str) {
  int lx = strlen(str);
  int sz;
  if (pd->size > pd->len + lx) {
    strcat(pd->str, str);
    pd->len += lx;
    return 1;
  }

  sz = pd->len + lx;
  sz = SIZE_BLOCK * ((sz / SIZE_BLOCK) + 1);
```

```
  if ((pd->str = (char *)realloc(pd->str, sz * sizeof(char)))) {
    strcat(pd->str, str);
    pd->len += lx;
    pd->size = sz;
    return 1;
  }
  return 0;
}

void freeData(const data_t *pd) { free(pd->str); }
```

Ejercicio 3:

El siguiente código puede genera inconvenientes en el sistema, o bien, el sistema puede interrumpir su funcionamiento. Lo invito a que analice lo que hace la función getAllocNombre, y corrija los errores de la aplicación.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define SIZE_BUFF 1000
char *getAllocNombre() {
  char *nombresFemeninos[] = {
      "María",     "Ana",       "Laura",     "Luisa",    "Sofía",     "Lucía",
      "Elena",     "Isabel",    "Carmen",    "Rosa",     "Patricia",  "Teresa",
      "Raquel",    "Natalia",   "Julia",     "Andrea",   "Clara",     "Victoria",
      "Eva",       "Sara",      "Paula",     "Beatriz",  "Monica",    "Olga",
      "Marta",     "Carolina",  "Mónica",    "Silvia",   "Diana",     "Gloria",
      "Susana",    "Lorena",    "Yolanda",   "Adela",    "Lidia",     "Antonia",
      "Fabiola",   "Camila",    "Manuela",   "Eloisa",   "Rocio",     "Liliana",
      "Celia",     "Lourdes",   "Rosario",   "Elisa",    "Alicia",    "Clotilde",
      "Esperanza", "Milagros",  "Constanza", "Renata",   "Gabriela",  "Adriana",
      "Marina",    "Eugenia",   "Fernanda",  "Rita",     "Josefina"};
  char *nombresMaculinos[] = {
      "Néstor",    "Juan",      "Carlos",    "Luis",     "Pedro",     "José",
      "Miguel",    "Alejandro", "Javier",    "Fernando", "David",     "Manuel",
      "Antonio",   "Francisco", "Pablo",     "Jorge",    "Rafael",    "Diego",
      "Daniel",    "Roberto",   "Alberto",   "Santiago", "José Luis", "Andrés",
      "Ricardo",   "Lorenzo",   "Ángel",     "Víctor",   "Enrique",   "Héctor",
      "Guillermo", "Eduardo",   "Ignacio",   "Óscar",    "Emilio",    "Hugo",
      "Félix",     "Tomás",     "Marcos",    "Raúl",     "Felipe",    "Nicolás",
      "Armando",   "Rogelio",   "Alfredo",   "Gustavo",  "Samuel",    "Ernesto",
      "Rodrigo",   "Esteban",   "Sergio",    "Adrián",   "Abel",      "Benjamín",
      "Arturo",    "César",     "Leonardo",  "Mariano",  "Mauricio"};
  char **s;
  char *sRet;
  int sz, i,cant;
  char sBuff[SIZE_BUFF];

  if (rand() % 2) {
    s = nombresFemeninos;
    sz = sizeof(nombresFemeninos) / sizeof(*nombresFemeninos);
  } else {
    s = nombresMaculinos;
    sz = sizeof(nombresMaculinos) / sizeof(*nombresMaculinos);
  }
```

```
  sBuff[0] = 0;
  cant = 1 + rand() % 4;
  for (i = 0; i < cant; i++) {
    strcat(sBuff, s[rand() % sz]);
    strcat(sBuff, " ");
  }
  sBuff[strlen(sBuff) - 1] = 0;
  sRet = (char *)malloc(SIZE_BUFF * sizeof(char));
  strcpy(sRet, sBuff);
  return sRet;
}

#define SIZE_TEST 100000
int main() {
  char **s;
  int i;

  while (1) {
    s = (char **)malloc(SIZE_TEST * sizeof(char *));
    if (!s) {
      printf("Error de Memoria");
      return 1;
    }
    for (i = 0; i < SIZE_TEST; i++) {
      s[i] = getAllocNombre();
    }

    for (i = 0; i < 5; i++)
      printf("%s\n", s[i]);

    free(s);
  }
  return 0;
      }
```

Ejercitación – Serie L

Variables, punteros y áreas de memoria

En forma simplificada podemos decir que una aplicación esta compuesta por 4 tipos de memoria a saber:

- Área de código
- Área de datos
- Stack
- Heap

La primera de éstas, el área de código, en los sistemas operativos actuales se comporta como memoria de solo lectura. Mientras que las tres restantes, son de lecto/escritura.

Adicionalmente podemos mencionar la dirección 0 (cero), la cual dentro de linux es una dirección en la cual no está permitido acceder ni para leer ni escribir. Es buena práctica utilizar este valor para inicializar punteros - NULL es simplemente una definición de (void*) 0).

Esta guía se enfoca a entender en que área de memoria se almacenan los diferentes datos o variables, y que consideraciones hay que tener al momento de utilizar los diferentes bloques de memoria.

Ejercicio 1:

1.a.- Defina en forma simple las 4 áreas de memoria antes mencionadas, incluya dentro de la definición para que se usan.

1.b.- Cuando y/o para qué se inicializa una variable tipo puntero en NULL.

Ejercicio 2:

Dado el siguiente segmento de código, indique donde en que se almacenan los diferentes elementos o variables declaradas, y en el caso de punteros, a donde apuntan.

```
01  #define MAX 20
02  volatile int cnt;
03  int CANT=25;
04  void imprimir (int *px,int cc)
05  {
06  static int cnt=0;
07  int i;
08      printf("\nvalores: ");
09      for (i=0;i<cc;printf("%d | ",*(px+i++)),cnt++);
10      printf("\ncounter: %d\n",cnt);
11  }
12  void genera (int *p, int cc)
13  {
14      int i;
15      for (i=0;i<cc;i++)
16      {
17          cnt+=3;
18          p[i]=cnt;
19      }
20  }
```

```
22  int main(void)
23  {
24  int arr[MAX];
25  int *p1;
26      genera (arr,MAX);
27      imprimir (arr,MAX);
28
29      p1=arr+3;
30      imprimir (p1,MAX-3);
31
32      p1=&CANT;
33      imprimir (p1,1);
34
35      return 0;
36  }
```

2.a.- De las líneas 01 a 03

2.a.1.- MAX

2.a.2.- cnt

2.a.3.- CANT

2.b.- Dentro de la función imprimir (líneas 04 a 11).

2.b.1.- px y *px

2.b.2.- cc

2.b.3.- cnt

2.b.4.- i

2.b.5.- (px+i) y *(px+i)

2.c.- Dentro de la función genera (líneas 12 a 20).

2.c.1.- p y *p

2.c.2.- cc

2.c.3.- p[i]

2.c.4.- cnt

2.d.- Dentro de la función main (líneas 22 a 36).

2.c.1.- arr , *arr y arr[0]

2.c.2.- p1 y *p1 entre las líneas 25 y 28

2.c.3.- p1 y *p1 entre las líneas 29 y 31

2.c.4.- p1 y *p1 entre las líneas 32 y 34

2.c.5.- CANT y &CANT

Ejercicio 3:

Dado el siguiente segmento de código, indique donde en que se almacenan las diferentes variables, a donde apuntan y cuanta memoria tienen reservada en cada caso.

```
01  void imprimir (const char *s)
02  {
03  static int cc=0;
04      cc++;
05      printf ("%02d - %s\n",cc,s);
06  }
07
08  int main(void)
09  {
10  char *ps1= "Informatica I R1093";
11  char ps2[]="Informatica I R1093";
12  char ps3[40];
13  char *ps4;
14
15      imprimir (ps1);
16      imprimir (ps2);
17
18      strcpy (ps3,"Informatica I R1093");
19      imprimir (ps3);
20
22      ps4=(char *) malloc (40*sizeof(char));
23      if (ps4)
24      {
25          strcpy (ps4,"Informatica I R1093");
26          imprimir (ps4);
27          free (ps4);
28      }
29      else
30          printf ("Error en obtención de memoria\n");
31  return 0;
32  }
```

3.a.- Dentro de la función main (líneas 04 a 11).

3.a.1.- ps1 y *ps1

3.a.2.- ps2 y *ps2

3.a.3.- ps3 y *ps3

3.a.4.- ps4 y *ps4 desde línea 13 a 20.

3.a.5.- ps4 y *ps4 en línea 23.

3.a.6.- ps4 y *ps4 desde línea 25 a 26.

3.a.7.- ps4 y *ps4 en línea 30.

3.b.- Dentro de la función imprimir (líneas 01 a 06).

3.b.1.- cc

3.b.2.- s y *s

Ejercicio 4:

Dado el siguiente segmento de código, indique donde en que se almacenan las diferentes variables, a donde apuntan y cuanta memoria tienen reservada en cada caso.

```
01  #define MAX 5
02
03  void imprimir (char **psx,int cc)
04  {
05  int i;
06      printf("Listado:\n-------\n ");
07      for (i=0;i<cc;printf("%s\n",*(psx+i++)));
08      printf("\n");
09  }
10
11  int main(int argc, char **argv)
12  {
13  char *ps1[]={"Liliana","Juan","Silvia","Mario","Malena"};
14  char *ps2[MAX];
15  char *s1, s2[40];
16
17      imprimir (ps1,sizeof ps1/sizeof *ps1);
18      imprimir (argv,argc);
19
20      if ((s1=(char *) malloc (40*sizeof (char)) )==NULL)
22      {
23          printf ("Error en reserva de memoria\n");
24          return 0;
25      }
26      strcpy (s1,"Informatica I");
27      strcpy (s2,"R1093");
28
29      ps2[0]="Materia";
30      ps2[1]=s1;
31      ps2[2]=s2;
32      ps2[3]=*(argv+argc-1);
33      ps2[4]=ps1[2];
34
35      imprimir (ps2,MAX);
36      free (s1);
37      return 0;
38  }
```

4.a.- Dentro de la función main.

4.a.1.- ps1, *ps1 o ps1[0] y **p1 o *ps1[0]

4.a.2.- ps2

4.a.3.- ps2[i] con i desde 0 a 4, en las diferentes partes de la función main. Hacer referencia a los números de línea para indicar a donde apuntan en cada caso.

4.a.4.- argc, argv y *argv

4.b.- Dentro de la función imprimir (líneas 03 a 09).

4.b.1.- psx y los correspondientes *(psx+i) si correspondiese

4.b.2.- cc

Ejercicio 5:

Si bien los siguientes segmentos de código no poseen errores de sintaxis, pueden presentar fallas en su ejecución. Identifique cuales de estos códigos poseen falla y describa cual es el inconveniente.

5.a.-

```
char *ps1;
    strcpy (ps1,"Buenos Aires");
```

5.b.-

```
char ps2[]="Catamarca";
    strcpy (ps2,"Salta");
```

5.c.-

```
char *ps3;
    ps3=(char *) malloc (100);
    strcpy (ps3,"San Luis, San Juan y Mendoza");
```

5.d.-

```
char ps4[50]="La Pampa y La Rioja";
    printf ("%s",ps4);
```

5.e.-

```
char *ps5="Formosa";
char ps6[50]="Formosa";
int a;
    a=strcmp (ps5,ps6);
```

5.f.-

```
char *ps7="Entre Rios, Corrientes y Misiones";
char ps8[50];
    strcpy (ps8,ps7);
```

5.g.-

```
char *ps9="Santa Cruz";
    strncpy (ps9,"Chubut",6);
    ps9[6]=0;
```

5.h.-

```
char *px1="Rio Negro y Neuquen";
char px2[20];
    strcpy (px2,px1);
```

5.i.-

```
char *px3="Tierra del Fuego";
char px4[20];
    strncpy (px4,px3,6);
    printf ("%s\n",px4);
```

5.j.-

```
char *px5;
    px5=(char *) malloc (CANT_DATA);
    if (!px5)
    {
        xxxxxx;
        return xxxx;
    }
```

```
    ...
    ...

    px5=(char *) realloc (px5,2*CANT_DATA);
    if (!px5)
    {
        xxxxxx;
        xxxxxx;
        return xxxx;
    }
```

5.k.-

```
char *px6="Salta y Jujuy";
    px6=(char *) realloc (px6,CANT_DATA);
```

5.l.-

```
char *px7="Salta y Jujuy";
char *px8;
    px7=(char *) malloc (CANT_DATA);
    if (!px7)
    {
        xxxxxx;
        return xxxx;
    }
    ...

   px8=px7;
    px7=(char *) realloc (px7,2*CANT_DATA);
    if (!px7)
    {
        xxxxxx;
        px7=px8;
    }

    ...
```

Ejercicio 6:

Si bien las siguientes funciones no poseen errores de sintaxis (a lo sumo algún warning), pueden presentar fallas en su ejecución. Identifique cuales de estas funciones poseen falla y describa cual es el inconveniente.

6.a.-

```
char * func_1 ()
{
char *s1="Balvanera";
return s1;
}
```

6.b.-

```
char * func_2 ()
{
char s2[]="Caballito";
return s2;
}
```

6.c.-

```c
#define TXT3 "Colegiales"
char * func_3 ()
{
char *s3;
    s3=(char*) calloc (strlen (TXT3)+1,sizeof(char));
    strcpy (s3,TXT3);
return s3;
}
```

6.d.-

```c
#define TXT4 "Congreso"
char * func_4 ()
{
char *s4;
    s4=(char*) calloc (strlen (TXT4+1),sizeof(char));
    if (s4)
        strcpy (s4,TXT4);
return s4;
}
```

6.e.-

```c
#define TXT5 "Palermo"
char * func_5 ()
{
char *s5;
    s5=(char*) calloc (strlen (TXT5)+1,sizeof(char));
    if (s5)
    {
        strcpy (s5,TXT5);
        ...
        free (s5);
    }
return s5;
}
```

6.f.-

```c
char * func_6 (int size)
{
char *s6;
    s6=(char*) calloc (size,sizeof(char));
return s6;
}
```