TP N°9: Recursividad

La siguiente guía cubre los contenidos vistos en la clase teórica:
16. Recursividad

Antes de empezar con los ejercicios de esta guía se recomienda la resolución de la **Autoevaluación 7** disponible en Campus ITBA, para consolidar los contenidos vistos en la teórica.

En esta práctica ninguna función debe tener más de 10 líneas de código

Ejercicios introductorios

A. Escribir una función "palotes" que reciba un entero `cant` sin signo y envíe en forma recursiva a la salida estándar `cant` "palotes". Si recibió cero, no imprime nada, si recibió 4, la salida debe ser "||||" (las comillas no deben imprimirse)
B. Escribir una versión recursiva para la función `strlen`
C. Escribir una versión recursiva de la función `strchr`

Ejercicio 1

Escribir una función recursiva que reciba como parámetros de entrada un vector de enteros y su dimensión, y que devuelva en su nombre la suma de todos sus elementos.

Ejercicio 2

Escribir la función recursiva **`suma`** que reciba un único parámetro de entrada de tipo integer y devuelva en su nombre la suma de sus dígitos.

Por ejemplo, **suma**(3498) devuelve 24

Ejercicio 3

Escribir una función recursiva **`productoEsc`** que reciba tres parámetros de entrada representando dos vectores de números reales de igual tamaño y su dimensión. La función debe regresar en su nombre el producto escalar de los vectores recibidos.

Ejercicio 4

Se desea escribir funciones que dado un número entero mayor o igual a cero, indiquen si el mismo es par o impar. ¿Qué ocurre si se las define de la siguiente forma?

```
int esPar(unsigned n);
int esImpar(unsigned n);

int
esPar(unsigned n)
{
      return ! esImpar(n);
}

int
esImpar(unsigned n)
{
      return ! esPar(n);
}
```

Ejercicio 5

Idem ejercicio anterior pero con las siguientes funciones. ¿En qué casos funcionan correctamente y en qué casos no?

```
int esPar(unsigned n);
int esImpar(unsigned n);

int
esPar(unsigned n)
{
    if (n = = 0)
        return 1;
    else
        return esImpar(n-1);
}

int
esImpar(unsigned n)
{
    if (n = = 1)
        return 1;
    else
        return esPar(n-1);
}
```

Ejercicio 6

Escribir una función que reciba como parámetro de entrada un string y retorne 1 si el mismo es palíndromo y 0 en caso contrario. Resolver el problema en forma recursiva

Ejercicio 7

Escribir una función recursiva ***busquedaBinaria*** que reciba como únicos parámetros:

- **datos**: un vector de enteros ordenados en forma ascendente.
- **dim**: la dimensión del mismo
- **num**: un número entero

Dicha función debe devolver 1 si **num** es un elemento de **datos** y 0 en caso contrario. La búsqueda del elemento se debe realizar utilizando el algoritmo de ***búsqueda binaria.***

Ejercicio 8

Calcular por medio de una función recursiva que reciba como parámetros de entrada dos números enteros mayores o iguales a cero, la fórmula de ACKERMANN correspondiente a los mismos, de acuerdo a las siguientes reglas:

- ACK (0,N) = N + 1
- ACK (M,0) = ACK (M - 1, 1)
- ACK (M,N) = ACK (M - 1, ACK (M , N - 1))

Ejercicio 9

El método de Newton-Raphson para aproximar la raíz cuadrada de un número positivo se puede calcular aplicando sucesivamente los siguientes pasos un cierto número de veces ITER:

$$\mathbf{valorAproximado} = \frac{\mathbf{valorAnterior} + \frac{\mathbf{X}}{\mathbf{valorAnterior}}}{\mathbf{2}}$$

donde el valor original conveniente para comenzar este ciclo suele ser X/2.

Escribir una función recursiva ***raizCuadrada*** que regrese en su nombre el valor aproximado de la raíz cuadrada de un número, recibiendo tres parámetros de entrada:

- **valorAnterior**: real (representa el valor hasta ahora candidato como raíz cuadrada)
- **iter**: entero que representa la cantidad de iteraciones que deben realizarse para aproximar la raíz cuadrada.
- **x**: real al que se le quiere calcular la raíz cuadrada.

Ejemplo:

Si valorAnterior = 4, iter= 3, x= 8, entonces la función devolverá 2.8333 (los valores sucesivos obtenidos son: 4, 3, 2.833)

Ejercicio 10

Escribir una función recursiva que reciba dos parámetros de tipo string, uno de entrada y el otro de salida. La función debe devolver en el segundo string los caracteres del primero en forma invertida. En la primera invocación de esta función recursiva el segundo string debe contener todos los caracteres en cero y con el suficiente espacio como para almacenar al primero, pero no debe validar estas condiciones.

Ejemplo:

```
char origen[] = "Practica";
char destino[20] = {0};

invierte(origen, destino);
```

Con esta invocación, en **destino** se obtendrá el string **"acitcarP".**

Nota: se asume que el vector de entrada no es el mismo que el de salida

Ejercicio 11

Escribir una función que reciba cuatro parámetros:

- Matrix: parámetro de entrada-salida que representa una matriz cuadrada booleana
- Dim: parámetro de entrada de tipo entero que representa la dimensión de la matriz
- Fila: parámetro de entrada de tipo entero que representa un número de fila de la matriz
- Columna: parámetro de entrada de tipo entero que representa un número de columna de la matriz

La función se encargará de devolver dicha matriz cambiada de la siguiente forma: el elemento de la fila y columna indicada y todos los de las dos diagonales que pasen por él se reemplazarán por su negación booleana, los demás elementos de la matriz quedan igual .

Esta función deberá ser recursiva o deberá invocar a otra función auxiliar recursiva para realizar lo pedido.

Ejemplo:

Si se recibe la siguiente matriz de 5x5 junto a los valores 1 (fila) y 2 (columna)

1	0	1	0	1
0	1	1	1	1
1	0	0	1	0
1	1	1	0	1
0	0	0	0	0

Se debe devolver la matriz cambiada así:

1	1	1	1	1
0	1	0	1	1
1	1	0	0	0
0	1	1	0	0
0	0	0	0	0

Ejercicio 12

Programar una ***función recursiva*** BALANCE que reciba como único parámetro un string constante representando una expresión matemática. Dicha función debe devolver el valor 0 si hay igual cantidad de paréntesis que abren y que cierran, y retorna un valor distinto de cero en caso contrario. **No definir variables ni funciones auxiliares**.

Ejemplos

si la función Balance recibe	*entonces devuelve*
`"56 + ( a – 7 – ( 12 + c ) – t * 678 )"`	**0**
`"56 + ( a – 7 – 12 + c ) – t * 678 )"`	**distinto de 0**
`" + ( a ( 10 + c * 789 ) – 6 / 8 ) + 40"`	**0**
`"5 + 2 – 7 )"`	**distinto de 0**
`")5 + (2 – 7"`	**0**

Ejercicio 13

Escribir una versión recursiva de la función `strrchr`

Ejercicios de Parcial

Ejercicio 14

Similar al ejercicio 12 pero el string contiene únicamente paréntesis y debe retornar cero sólo si los paréntesis están **apareados**. Consideramos que los paréntesis de una expresión están apareados si para cada paréntesis de apertura hay uno de cierre en una posición posterior. **No definir macros ni funciones auxiliares**.

Ejemplos

si la función Balance recibe
"()(()())"
")("
""
"()("
"(()())"
"()())"

Ejercicio 15

En muchos supermercados se ordena la mercadería de manera especial, para capturar la atención del comprador. Un ejemplo concreto es el de ordenar latas en forma de "pirámide", de manera tal que arriba de todo haya una sola lata, debajo 4, luego 9 y así sucesivamente.

Escribir la función **recursiva *piramide*** que regrese toda la información a través de dos parámetros de entrada-salida, a saber:

- ***latas***, que representa la cantidad de latas que no se llegaron a apilar. Inicialmente representa la cantidad total de latas.
- ***altura***, que representa la cantidad de niveles con que se puede construir la pirámide. Inicialmente se encuentra en 0.

El cuerpo de la función ***piramide*** no debe superar las 8 líneas de código
No usar funciones auxiliares

Ejemplos:

- Si se invoca a la función ***pirámide*** con **10 latas** y **altura 0**, al terminar la recursividad el parámetro ***latas*** debe quedar en **5** y ***altura*** en **2**.
- Si se invoca a la función ***pirámide*** con **55 latas** y **altura 0**, al terminar la recursividad el parámetro ***latas*** debe quedar en **0** y ***altura*** en **5**.

- Si se invoca a la función ***pirámide*** con **1 lata** y **altura 0**, al terminar la recursividad el parámetro ***latas*** debe quedar en **0** y ***altura*** en **1**.

Ejercicio 16

Escribir la función recursiva **orden** que recibe como único parámetro un vector constante de enteros mayores a cero, donde el final del mismo está marcado con el valor -1.

La función debe retornar:

- 1: si el vector está ordenado en forma ascendente (cada elemento es mayor que el anterior)
- -1: si el vector está ordenado en forma descendente (cada elemento es menor al anterior)
- 0 en cualquier otro caso

Si no tiene elementos o tiene un único elemento se considera que no está ordenado.

Ejemplos:

```
int v[] = {2, 1, 2, 8, 20, 12343, -1};
int v2[] = {10, 10, 9, 5, 3, -1};
int resp;

resp = orden(v);         /* resp vale 0 */
resp = orden(v + 1);     /* resp vale 1 */
resp = orden(v2);        /* resp vale 0 */
resp = orden(v2 + 1);    /* resp vale -1 */
resp = orden(v2 + 4);    /* resp vale 0 */
resp = orden(v2 + 5);    /* resp vale 0 */
```

No se pueden usar macros o funciones auxiliares
No se pueden agregar parámetros
No se pueden usar variables static
Esta función puede tener más de 10 líneas de código

Ejercicio 17

Se almacena en un string una serie de palabras (secuencias de letras del alfabeto inglés) y la longitud de cada una de ellas. Cada palabra debería tener entre 0 y 9 caracteres, por lo que después de cada palabra hay un carácter indicando la longitud de la misma ('4' si tiene 4 letras, '8' si tiene 8 letras, etc.).

Escribir una función recursiva **`bienFormado`** que reciba **como único argumento** un string con el formato mencionado y retorne 0 (cero) si está bien formado, y distinto de cero si no.

La función no puede usar otras funciones o macros, salvo `isdigit` e `isalpha`.

Ejemplos:

Los siguientes strings están bien formados : `"abcd4a10bb2", "", "0000", "a1", "abc3", "0a1"`

Los siguientes strings están mal formados: `"a", "1", "a2", "abc2", "abc4", "abc33", "0123", "2aa", "$1", "@@2"`

Ejercicio 18

Escribir una función recursiva **`sumMatch`** que reciba como <u>único parámetro</u> un vector de enteros mayores o iguales a cero, donde la marca de final es un número negativo. La función debe retornar cero si los elementos del vector están "apareados por sumas", y distinto de cero si no, de acuerdo a los siguientes ejemplos:

Para los siguientes vectores la función retorna cero:

```
int v1[]={1,2,3,6,4,3,2,9,1,2,3,-1};     // 1+2+3=6, 4+3+2=9, 1+2=3
int v2[]={1,1,2,4,5,5,14,-1};            // 14=5+5+4, 2=1+1
int v3[]={3,3,-1};                       // 3=3
int v4[]={1,1,2,2,90,90,-1};             // 90=90, 2=2, 1=1
int v5[]={1,0,1,2,2,2,6,-1};             // 6=2+2+2, 1=0+1
int v6[]={-1};
int v7[]={0,-1};
```

Para los siguientes vectores la función retorna distinto cero:

```
int v1[]={1,1,2,3,6,4,3,2,9,1,2,3,-1};
int v2[]={1,1,2,4,5,5,14,10,-1};
int v3[]={4,3,-1};
int v4[]={1,-1};
int v5[]={1,0,1,2,1,2,2,2,-1};
```

<u>No usar variables `static` ni funciones o macros auxiliares</u>

Ejercicios adicionales

Estos ejercicios son de alto nivel, están dirigidos a los que quieren superarse e ir más allá de los alcances de la materia. El nivel de estos ejercicios superan a los ejercicios de parcial.

Ejercicio 19

Escribir una versión recursiva de la función `strpbrk`

Ejercicio 20

Escribir una función `sudokuSolver`, que reciba una matriz de 9x9 de un sudoku incompleto y retorne 1 si tiene solución y 0 si no tiene solución. En caso de tener solución debe completar la matriz con una solución posible (podría haber más de una solución válida). Como estrategia se debe usar un algoritmo de tipo "fuerza bruta", esto es, tratar de llenar cada casilla vacía con todos los números válidos posibles, combinando a su vez con todos los posibles números válidos de las casillas libres restantes.
Se asegura que en cada celda de la matriz recibida hay un número entre 0 y 9, donde 0 indica que la casilla está libre. También se asegura que no hay números repetidos en una misma fila, columna o subcuadrado.