TRABAJO PRÁCTICO N° 4

Deben entregarse por grupos en la entrega correspondiente vía Campus los archivos .c correspondientes a los ejercicios indicados.

1. Predecir la salida del siguiente programa. Luego ejecutarlo línea por línea con **gdb**, viendo el valor del arreglo *magic* en cada paso para verificar lo predicho.

```
#include <stdio.h>

int main (void)
{
   int magic[3], i=1;
   magic[2] = 2;
   magic[i] = 1;
   magic[i-1] = 0;
   printf("Size of magic: %d\n", sizeof(magic));
   for (i=1 ; i<=3 ; i++)
   {
         printf("i: %d; magic[i]: %d\n", i, magic[i]);
   }
   return 0;
}
```

2. Predecir la salida del siguiente programa. Luego ejecutarlo línea por línea con **gdb**, viendo el valor del arreglo *arr* en cada paso para verificar lo predicho.

```
#include <stdio.h>

int main(void)
{
   int arr[5];
   int ind = 1;
   arr[0] = 5;
   arr[ind+1] = -6;
   arr[4] = 7;
   arr[ind++] = 1;
   arr[++ind] = -3;
   ++(arr[2]);

   for (ind=0 ; ind<5 ; ind++)
   {
      printf("arr[%d]: %d\n", ind, arr[ind]);
   }
   return 0;
}
```

3. Escribir una función que reciba del usuario una cantidad arbitraria de números y calcule el promedio. ¿Qué argumentos debe recibir la función y de que tipo deben ser para poder implementarla?

4. Escribir un programa que pregunte al usuario su nombre, y luego imprima en pantalla el mensaje *"Hola <nombre>."*

5. Escribir una función que pase un string a mayúsculas.

6. Escribir una función que calcule el producto de una matriz de tres columnas por un vector de tres filas. El prototipo de la función requerida es el siguiente:

   ```
   void prm3(double vr[], double M[][3], double v[], int filas);
   ```

   donde filas es el número de filas de la matriz M. La función realiza el producto de la matriz M por el vector v y el resultado lo coloca en el vector vr. La matriz y la cantidad de filas deben ser ingresadas por teclado.

7. Escribir una función que calcule el determinante de una matriz de 2x2.

8. Escribir una función que calcule el determinante de una matriz de 3x3.

9. Escribir una función que calcule la inversa de una matriz de 3x3.

10. Predecir la salida del siguiente programa. ¿Es el resultado esperado? Si hay errores, corregirlos.

```
#include<stdio.h>
#define SQR(x)(x*x)

int main(void)
{
    int a, b=3;
    a = SQR(b+2);
    printf("%d\n", a);
    return 0;
}
```

11. La siguiente función pasa un carácter a mayúscula.

```
char pasar_a_mayuscula(char c)
{
    if(c >= 'a' && c <= 'z')
    {
        return c - 'a' + 'A';
    }
    else
    {
        return c;
    }
}
```

Escribir una macro que realice la misma tarea.

12. Escribir una macro parametrizada llamada *DIST* que calcule la distancia (valor mayor o igual a cero) entre dos números.

Ejemplo: la distancia entre los números 3 y 7 es 4; y la distancia entre los números 7 y 3 también es 4.

13. Predecir la salida del siguiente programa. Luego compilar y ejecutar para verificar la salida del mismo, y explicarla.

```
#include <stdio.h>

int main()
{
    int a = 3, b = 5;

    #if a == b
        printf("Son iguales!\n");
    #endif

    return 0;
}
```

14. **[ENTREGAR] Juego de la vida**

El juego de la vida, inventado en 1970 por John Conway in 1970, es un simulador de células autómatas. El juego consiste de un mundo 2D que se extiende infinitamente en todas las direcciones, dividido en celdas que representan una célula. Cada célula puede estar únicamente en uno de dos estados: viva o muerta, en cada una de las generaciones. El juego consiste de una serie de reglas que describen cómo estas células evolucionan de generación en generación.

Estas reglas calculan el estado de una célula para la próxima generación como función del estado de las células colindantes (vecinas) en la generación actual. En un mundo de dos dimensiones, estas son las 8 celdas que se encuentran vertical, horizontal o diagonalmente adyacentes a la misa. Las reglas se pueden resumir como:

a) Una célula viva con menos de dos vecinos muere;
b) una célula con más de tres vecinos también muere;
c) una célula viva con exactamente dos o tres vecinos sobrevive;
d) una célula muerta con exactamente tres vecinos obtiene vida nuevamente.

Figura 1: Ejemplo

Para este ejercicio, vamos a implementar el Juego de la vida, con la restricción menor de que nuestro mundo es finito. A su vez, asumiremos que los vecinos más allá de los límites de nuestro mundo se encuentran muertos (y nunca son actualizados). El estado inicial estará hardcoded (es decir, estará fijado en el código fuente del programa).

Algunos requisitos del programa a realizar:

- El tamaño del mundo debe encontrarse en dos constantes, por ejemplo, ANCHO y ALTO. Debe poder cambiarse el valor de las mismas y recompilar el código, y el mismo debe funcionar.
- El programa debe devolver un error de compilación definido por ustedes si las constantes ANCHO o ALTO son menores o iguales a cero.
- El programa debe devolver un warning de compilación definido por ustedes si ANCHO o ALTO son mayores a 50, o si su producto es mayor a 1000, advirtiendo que la salida será difícil de ver.
- El programa debe mostrar el estado actual del mundo en pantalla, y luego preguntar al usuario cuantas generaciones desea avanzar. Cuando el usuario ingresa un número y presiona enter, se calcula el nuevo estado del mundo, se presenta en pantalla, y se vuelve a preguntar.
- Si el usuario presiona enter sin ingresar ningún número, se debe avanzar una generación.
- Si el usuario ingresa el carácter ‘q’, se debe salir del programa.
- Cualquier funcionalidad extra que deseen agregar se tomará en cuenta en la nota del trabajo práctico.