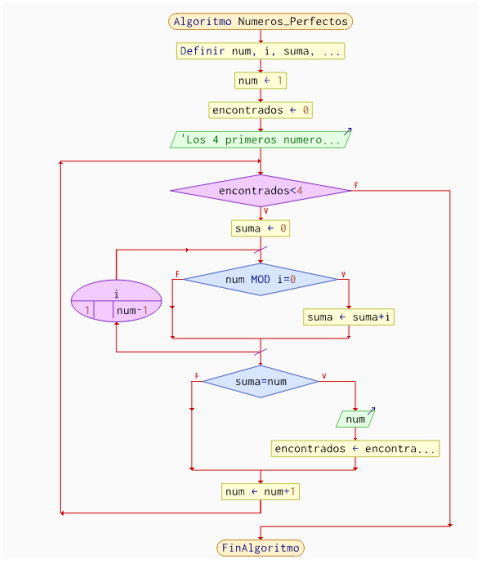
Un número perfecto es un número natural que es igual a la suma de sus divisores propios positivos, sin incluirse él mismo. Por ejemplo, el 6 es un número perfecto, porque sus divisores propios son 1, 2 y 3; y 6=1+2+36 = 1 + 2 + 36=1+2+3. El siguiente número perfecto es 28 = 1 + 2 + 4 + 7 + 14. Se pide: 1. Programe la función esperfecto, que reciba como argumento un número natural y devuelva un valor 1 si el número es perfecto, y 0 en caso contrario. 2. Desarrolle la función principal del programa que calcule los 4 primeros números perfectos y los muestre por pantalla.

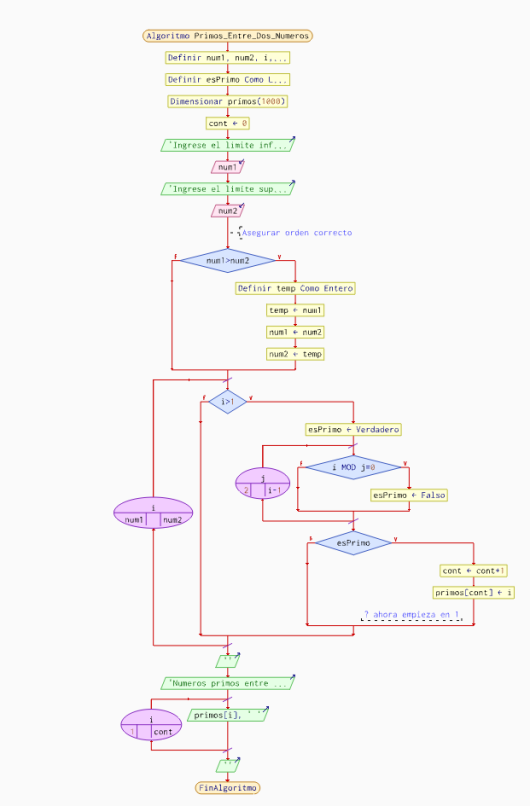
```
Algoritmo Numeros_Perfectos
    Definir num, i, suma, encontrados Como Entero

    num ← 1
    encontrados ← 0

    Escribir "Los 4 primeros numeros perfectos son:"

    Mientras encontrados < 4 Hacer
        suma ← 0

        Para i ← 1 Hasta num - 1 Hacer
            Si num MOD i = 0 Entonces
                suma ← suma + i
            FinSi
        FinPara

        Si suma = num Entonces
            Escribir num
            encontrados ← encontrados + 1
        FinSi

        num ← num + 1
    FinMientras
FinAlgoritmo
```




```
#include <stdio.h>
int esperfecto(int n)
{
    int i, suma = 0;

    para (i = 1;  Yo < n;  Yo++)
    {
        if (n % i == 0)
            suma += i;
    }
    if (suma == n)
        Retorno 1;
    si no,
        retorno 0;
}
int main(void)
{
    int num = 1;
num ¿¿Perfecto?  encontrados Se imprime
1 No 0 No
6 S í 1 6
28 S í 2 28
496 S í 3 496
8128 Sí 4 8128
I ¿28 % i == 0?  SUMA
1 Sí 1
2 Sí 3
4 Sí 7
7 S í 14
14 S í 28
int encontrados = 0;
printf("Los 4 primeros números perfectos son:\n");
mientras (encontrados < 4)
{
}
}
si (esperfecto(num))
{
}
printf("%d\n", num);
encontrados++;
num++;
}
}
```

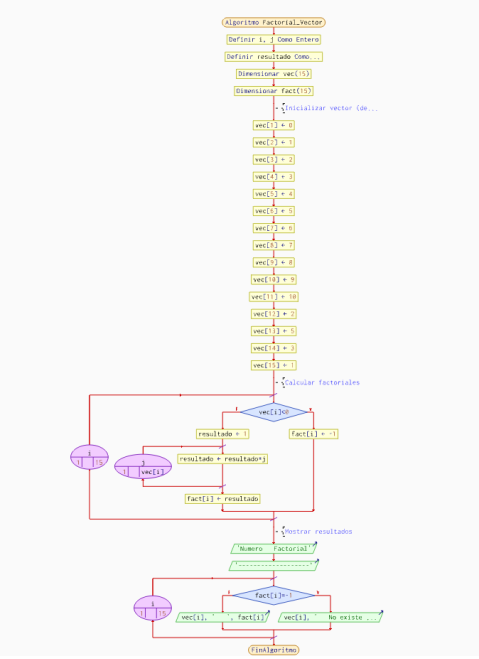
Problema 3.1.2 Números primos. Realice un programa que resuelva adecuadamente los siguientes apartados: 1. Programe la función esprimo, que recibe como argumento un número entero, y devuelve un valor 1 si el número es primo y 0 en caso contrario. 2. Almacene en un vector todos los números primos comprendidos entre dos números introducidos por teclado y luego imprima dicho vector.

```
Algoritmo Primos_Entre_Dos_Numeros
    Definir num1, num2, i, j, cont Como Entero
    Definir esPrimo Como Logico

    Dimension primos[1000]
    cont ← 0

    Escribir "Ingrese el limite inferior: "
    Leer num1
    Escribir "Ingrese el limite superior: "
    Leer num2

    // Asegurar orden correcto
    Si num1 > num2 Entonces
        Definir temp Como Entero
        temp ← num1
        num1 ← num2
        num2 ← temp
    FinSi

    Para i ← num1 Hasta num2 Hacer
        Si i > 1 Entonces
            esPrimo ← Verdadero

            Para j ← 2 Hasta i - 1 Hacer
                Si i MOD j = 0 Entonces
                    esPrimo ← Falso
                FinSi
            FinPara

            Si esPrimo Entonces
                cont ← cont + 1
                primos[cont] ← i    // ? ahora empieza en 1
            FinSi
        FinSi
    FinPara

    Escribir ""
    Escribir "Numeros primos entre ", num1, " y ", num2, ":"

    Para i ← 1 Hasta cont Hacer
        Escribir Sin Saltar primos[i], " "
    FinPara

    Escribir ""
FinAlgoritmo
```



Programe la función calc_fact, que recibe como argumento un número entero, y devuelva el valor de la factorial. A continuación, use esta función en un programa que, dado un vector de 15 números enteros vec, calcule un vector fact con sus factoriales y lo muestre por pantalla.

```
Algoritmo Factorial_Vector
    Definir i, j Como Entero
    Definir resultado Como Entero

    Dimension vec[15]
    Dimension fact[15]

    // Inicializar vector (desde 1)
    vec[1] ← 0
    vec[2] ← 1
    vec[3] ← 2
    vec[4] ← 3
    vec[5] ← 4
    vec[6] ← 5
    vec[7] ← 6
    vec[8] ← 7
    vec[9] ← 8
    vec[10] ← 9
    vec[11] ← 10
    vec[12] ← 2
    vec[13] ← 5
    vec[14] ← 3
    vec[15] ← 1

    // Calcular factoriales
    Para i ← 1 Hasta 15 Hacer
        Si vec[i] < 0 Entonces
            fact[i] ← -1
        SiNo
            resultado ← 1
            Para j ← 1 Hasta vec[i] Hacer
                resultado ← resultado * j
            FinPara
            fact[i] ← resultado
        FinSi
    FinPara

    // Mostrar resultados
    Escribir "Numero   Factorial"
    Escribir "-------------------"

    Para i ← 1 Hasta 15 Hacer
        Si fact[i] = -1 Entonces
            Escribir vec[i], "   No existe (numero negativo)"
        SiNo
            Escribir vec[i], "   ", fact[i]
        FinSi
    FinPara
FinAlgoritmo
```



2. *Realice el programa principal que declare una tabla de estructuras de dimensión 10 para almacenar la información sobre personas (se ha supuesto que el número de personas no será mayor de 10). A continuación debe pedir por teclado el número de personas para introducir y después los datos de cada una de ellas. Tras ello, calcule el nombre que se repita más en los datos introducidos y la media de edad de todas las personas.*

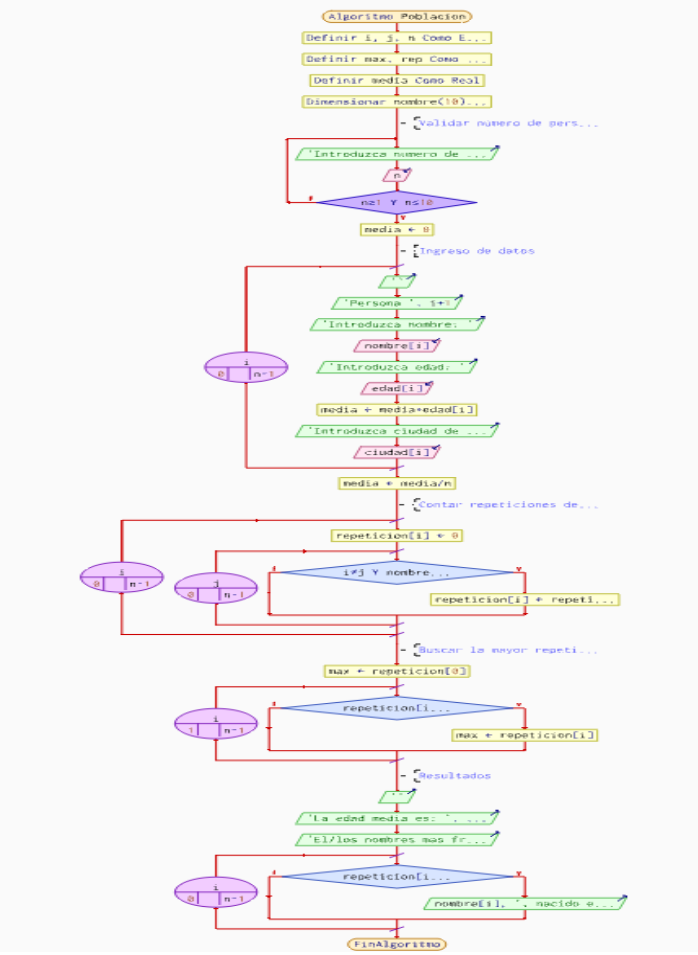
   Se puede usar la función de librería:

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

   La función retorna un número entero mayor, igual, o menor que cero, apropiadamente según la cadena apuntada por s1 es mayor, igual, o menor que la cadena s2.

Tras la lectura de todos los datos de personas se calcula el número de veces que se repite el nombre, actualizándolo en el campo *rep*, y almacenando en la variable *max* el mayor número de veces que se repite hasta el momento un nombre.

```
Proceso Poblacion

    Definir i, j, n Como Entero
    Definir max, rep Como Entero
    Definir media Como Real

    Dimension nombre[10], ciudad[10], edad[10], repeticion[10]

    // Validar número de personas
    Repetir
        Escribir "Introduzca numero de personas (1 a 10): "
        Leer n
    Hasta Que n ≥ 1 Y n ≤ 10

    media ← 0

    // Ingreso de datos
    Para i ← 0 Hasta n-1 Hacer
        Escribir ""
        Escribir "Persona ", i + 1

        Escribir "Introduzca nombre: "
        Leer nombre[i]

        Escribir "Introduzca edad: "
        Leer edad[i]
        media ← media + edad[i]

        Escribir "Introduzca ciudad de nacimiento: "
        Leer ciudad[i]
    FinPara

    media ← media / n

    // Contar repeticiones de nombres
    Para i ← 0 Hasta n-1 Hacer
        repeticion[i] ← 0
        Para j ← 0 Hasta n-1 Hacer
            Si i ≠ j Y nombre[i] = nombre[j] Entonces
                repeticion[i] ← repeticion[i] + 1
            FinSi
        FinPara
    FinPara

    // Buscar la mayor repetición
    max ← repeticion[0]
    Para i ← 1 Hasta n-1 Hacer
        Si repeticion[i] > max Entonces
            max ← repeticion[i]
        FinSi
    FinPara

    // Resultados
    Escribir ""
    Escribir "La edad media es: ", media
    Escribir "El/los nombres mas frecuentes son:"

    Para i ← 0 Hasta n-1 Hacer
        Si repeticion[i] = max Entonces
            Escribir nombre[i], ", nacido en ", ciudad[i]
        FinSi
    FinPara

FinProceso
```




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

#define MAX 10
#define STR 100

typedef struct {
    char nombre[STR];
    int edad;
    char ciudad[STR];
    int rep;
} Poblacion;

int main() {

    int i, j, n;
    int maxRep = 0;
    float media = 0;

    Poblacion v[MAX];

    /* Validar número de personas */
    do {
        printf("Introduzca numero de personas (1 a 10): ");
        scanf("%d", &n);
    } while (n < 1 || n > 10);

    /* Ingreso de datos */
    for (i = 0; i < n; i++) {

        printf("\nPersona %d\n", i + 1);

        printf("Introduzca nombre: ");
        scanf("%s", v[i].nombre);

        printf("Introduzca edad: ");
        scanf("%d", &v[i].edad);
        media += v[i].edad;

        printf("Introduzca ciudad de nacimiento: ");
        scanf("%s", v[i].ciudad);
    }

    media = media / n;

    /* Contar repeticiones de nombres */
    for (i = 0; i < n; i++) {
        v[i].rep = 0;
        for (j = 0; j < n; j++) {
            if (i != j && strcmp(v[i].nombre, v[j].nombre) == 0) {
                v[i].rep++;
            }
        }
    }

    /* Buscar máxima repetición */
    maxRep = v[0].rep;
    for (i = 1; i < n; i++) {
        if (v[i].rep > maxRep) {
            maxRep = v[i].rep;
        }
    }

    /* Resultados */
    printf("\nLa edad media es: %.2f\n", media);
    printf("El/los nombres mas frecuentes son:\n");

    for (i = 0; i < n; i++) {
        if (v[i].rep == maxRep) {
            printf("%s, nacido en %s\n", v[i].nombre, v[i].ciudad);
        }
    }

    return 0;
}
```

Problema 4.2 Reconocimiento de caracteres.

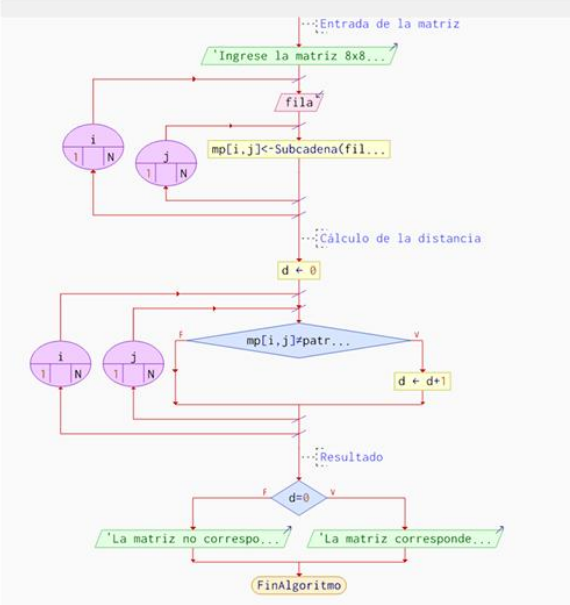
Se pretende escribir un programa para reconocer caracteres a partir de un mapa de puntos. El mapa de puntos describe la forma de un carácter como una matriz de unos y ceros de 8×8 celdas (véase figura 4.1). Se dispone además de una tabla de estructuras de tipo `struct letras`*, que puede suponer convenientemente creada e inicializada, y que contiene la descripción de las 27 letras del alfabeto, tal como se describe en el ejemplo.*

```
struct letras
{
   char cod_ASCII;  /* Letra a la que corresponde la matriz  de puntos*/
   int mptos[8][8];  /* Matriz de puntos del carácter */
};
struct letras tab_let[27];
```

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

Figura 4.1: Representación del carácter *a* minúscula

Escriba la función `busca_caracter`*, la cual recibe una matriz de 8 × 8 enteros que contiene el mapa de puntos de la letra que se quiere identificar (parámetro mp) y la tabla de estructuras de tipo letras que contiene la descripción de las 27 letras del alfabeto (*`tab_let`*). La función debe devolver el código ASCII del carácter que más se parezca.*



```c
#include <stdio.h>

#define N 8

typedef struct {
    char cod_ASCII;
    char mpost[N][N];
} letras;

/* Función para calcular la distancia entre dos matrices */
int distancia(const char a[N][N], const char b[N][N]) {
    int d = 0;
    int i, j;

    for (i = 0; i < N; i++) {
        for (j = 0; j < N; j++) {
            if (a[i][j] != b[i][j]) {
                d++;
            }
        }
    }
    return d;
}

int main() {

    char mp[N][N];
    letras a_min;
    int i, j, d;

    /* Código ASCII */
    a_min.cod_ASCII = 'a';

    /* Patrón de la letra 'a' */
    char patron_a[N][N] = {
        {'0','0','#','#','#','#','0','0'},
        {'0','#','0','0','0','0','#','0'},
        {'0','0','0','0','0','0','#','0'},
        {'0','0','#','#','#','#','0','0'},
        {'0','#','0','0','0','0','#','0'},
        {'#','0','0','0','0','0','#','0'},
        {'0','#','0','0','0','0','#','0'},
        {'0','0','#','#','#','#','0','#'}
    };
```

```c
};

/* Copiar patrón a la estructura */
for (i = 0; i < N; i++) {
    for (j = 0; j < N; j++) {
        a_min.mpost[i][j] = patron_a[i][j];
    }
}

/* Ingreso de la matriz */
printf("Ingrese la matriz 8x8 usando SOLO 0 y #:\n");
for (i = 0; i < N; i++) {
    for (j = 0; j < N; j++) {
        scanf(" %c", &mp[i][j]);
    }
}

/* Comparación */
d = distancia(mp, a_min.mpost);

if (d == 0) {
    printf("\nLa matriz corresponde a la letra 'a' minuscula\n");
} else {
    printf("\nLa matriz NO corresponde a la letra 'a' minuscula\n");
}

return 0;
```

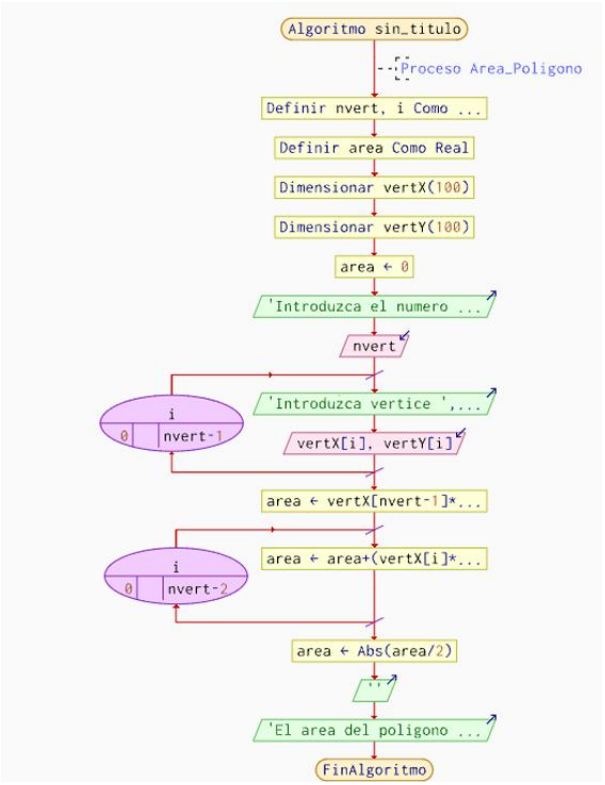
```
Proceso CompararLetraA
    Definir i, j, d Como Entero
    Definir mp, patronA Como Caracter
    Dimension mp[8,8]
    Dimension patronA[8,8]
    patronA[0,0] ← "0"
    patronA[0,1] ← "0"
    patronA[0,2] ← "#"
    patronA[0,3] ← "#"
    patronA[0,4] ← "#"
    patronA[0,5] ← "#"
    patronA[0,6] ← "0"
    patronA[0,7] ← "0"

    patronA[1,0] ← "0"
    patronA[1,1] ← "#"
    patronA[1,2] ← "0"
    patronA[1,3] ← "0"
    patronA[1,4] ← "0"
    patronA[1,5] ← "0"
    patronA[1,6] ← "#"
    patronA[1,7] ← "0"

    patronA[2,0] ← "0"
    patronA[2,1] ← "0"
    patronA[2,2] ← "0"
    patronA[2,3] ← "0"
    patronA[2,4] ← "0"
    patronA[2,5] ← "0"
    patronA[2,6] ← "#"
    patronA[2,7] ← "0"

    patronA[3,0] ← "0"
    patronA[3,1] ← "0"
    patronA[3,2] ← "#"
    patronA[3,3] ← "#"
    patronA[3,4] ← "#"
    patronA[3,5] ← "#"
    patronA[3,6] ← "0"
    patronA[3,7] ← "0"

    patronA[4,0] ← "0"
    patronA[4,1] ← "#"
    patronA[4,2] ← "0"
    patronA[4,3] ← "0"
    patronA[4,4] ← "0"
    patronA[4,5] ← "0"
    patronA[4,6] ← "#"
    patronA[4,7] ← "0"
```

```
    patronA[5,1] ← "0"
    patronA[5,2] ← "0"
    patronA[5,3] ← "0"
    patronA[5,4] ← "0"
    patronA[5,5] ← "0"
    patronA[5,6] ← "#"
    patronA[5,7] ← "0"

    patronA[6,0] ← "0"
    patronA[6,1] ← "#"
    patronA[6,2] ← "0"
    patronA[6,3] ← "0"
    patronA[6,4] ← "0"
    patronA[6,5] ← "0"
    patronA[6,6] ← "#"
    patronA[6,7] ← "0"

    patronA[7,0] ← "0"
    patronA[7,1] ← "0"
    patronA[7,2] ← "#"
    patronA[7,3] ← "#"
    patronA[7,4] ← "#"
    patronA[7,5] ← "#"
    patronA[7,6] ← "0"
    patronA[7,7] ← "#"

    Escribir "Ingrese la matriz 8x8 (0 o #):"
    Para i ← 0 Hasta 7
        Para j ← 0 Hasta 7
            Leer mp[i,j]
        FinPara
    FinPara

    d ← 0
    Para i ← 0 Hasta 7
        Para j ← 0 Hasta 7
            Si mp[i,j] ≠ patronA[i,j] Entonces
                d ← d + 1
            FinSi
        FinPara
    FinPara

    Si d = 0 Entonces
        Escribir "La matriz corresponde a la letra a minuscula"
    Sino
        Escribir "La matriz NO corresponde a la letra a minuscula"
    FinSi

FinProceso
```

https://onlinegdb.com/lgQA3Qwb5

Un polígono es una figura geométrica cerrada delimitada por segmentos rectos (aristas). En este ejercicio se usa la estructura tipo polígono para almacenar la información de un polígono, donde en vez de almacenarse sus aristas se almacenan sus vértices, como se describe a continuación: struct punto { float x; float y; }; struct poligono { int nvert; / Coordenada x */ /* Coordenada y */ /* Nro de vértices */ struct punto vert[100]; /* Vector de vértices */ }; De este modo, un polígono de la figura 4.2 quedaría representado mediante una estructura tipo struct poligono, cuyo miembro contiene el número de vértices del polígono (5), y vert es un vector de estructuras que contiene las coordenadas de los vértices con dimensión 100.*

```c
#include <stdio.h>
#include <math.h>

#define MAX_VERTICES 100

/* ===== ESTRUCTURAS ===== */
struct punto {
    float x;
    float y;
};

struct poligono {
    int nvert;              // Número de vértices
    struct punto vert[MAX_VERTICES]; // Vector de vértices
};

/* ===== FUNCIONES ===== */

// Función para leer los vértices del polígono
void leerPoligono(struct poligono *p) {
    printf("Ingrese el numero de vertices del poligono (máx %d): ", MAX_VERTICES);
    scanf("%d", &p->nvert);

    if (p->nvert < 3 || p->nvert > MAX_VERTICES) {
        printf("Número de vértices inválido.\n");
        return;
    }

    for(int i = 0; i < p->nvert; i++) {
        printf("Vértice %d - coordenada x: ", i+1);
        scanf("%f", &p->vert[i].x);
        printf("Vértice %d - coordenada y: ", i+1);
        scanf("%f", &p->vert[i].y);
    }
}

// Función para imprimir los vértices del polígono
void mostrarPoligono(struct poligono p) {
    printf("\nLos vértices del poligono son:\n");
    for(int i = 0; i < p.nvert; i++) {
        printf("Vértice %d: (%.2f, %.2f)\n", i+1, p.vert[i].x, p.vert[i].y);
    }
}

// Función para calcular el perímetro del polígono
float calcularPerimetro(struct poligono p) {
    float perimetro = 0.0;
    for(int i = 0; i < p.nvert; i++) {
        int siguiente = (i + 1) % p.nvert; // Se conecta el último con el primero
        float dx = p.vert[siguiente].x - p.vert[i].x;
        float dy = p.vert[siguiente].y - p.vert[i].y;
        perimetro += sqrt(dx*dx + dy*dy); // distancia entre dos puntos
    }
    return perimetro;
}

/* ===== PROGRAMA PRINCIPAL ===== */
int main() {
    struct poligono miPoligono;

    // Leer vértices
    leerPoligono(&miPoligono);

    // Mostrar los vértices
    mostrarPoligono(miPoligono);

    // Calcular e imprimir perímetro
    float perimetro = calcularPerimetro(miPoligono);
    printf("\nEl perimetro del poligono es: %.2f\n", perimetro);

    return 0;
}
```



https://onlinegdb.com/plyEXsx6a

Se pretende hacer un pequeño juego de palabras por ordenador en lenguaje C que muestre en pantalla los caracteres que aparecen en la figura 4.3 de fin de juego. Considere la siguiente estructura: #define N 10 #define M 12 struct mensaje { char game_over[N]; char se_acabo[M]; }; typedef struct mensaje men; (En la figura 4.3 se muestra en pantalla el siguiente efecto visual:)

G

GA

GAM

GAME

GAME

GAME O

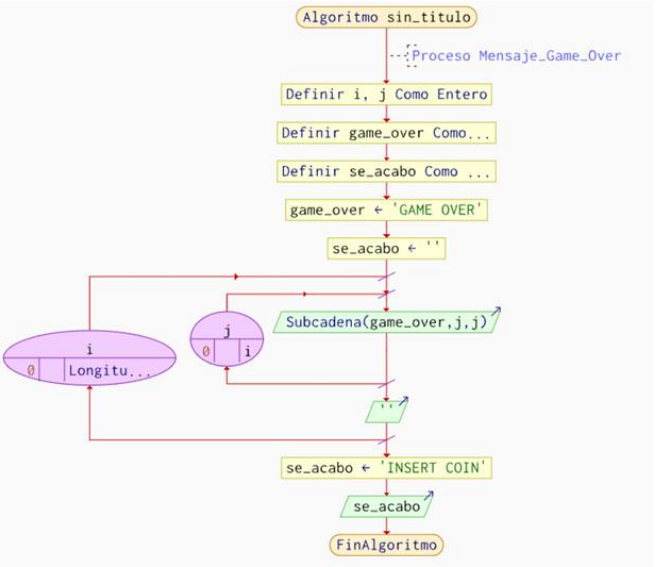
GAME OV

GAME OVE

GAME OVER

INSERT COIN

```
Algoritmo GameOver_PSeInt

    Definir game_over, se_acabo Como Cadena
    Definir i Como Entero

    // Inicializar cadenas
    game_over = "GAME OVER"
    se_acabo = ""

    // Imprimir progresivamente "GAME OVER"
    Para i = 1 Hasta Longitud(game_over)
        Escribir Subcadena(game_over, 1, i)
    FinPara

    // Asignar mensaje final
    se_acabo = "INSERT COIN"

    // Mostrar mensaje final
    Escribir se_acabo

FinAlgoritmo
```




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

#define N 10
#define M 12

// Estructura mensaje con dos cadenas
struct mensaje {
    char game_over[N];
    char se_acabo[M];
};

typedef struct mensaje men;

// Función que imprime los primeros n caracteres de game_over
void prn_cad(men *m, int n) {
    int i = 0;
    while (i < n) {  // imprime hasta n-1
        printf("%c", m->game_over[i]);
        i++;
    }
    printf("\n"); // salto de línea al final
}

int main(void) {
    // Inicializar estructura
    men fin;
    strcpy(fin.game_over, "GAME OVER");
    strcpy(fin.se_acabo, "");  // inicializamos vacío

    // Imprimir progresivamente "GAME OVER"
    for (int i = 1; i <= strlen(fin.game_over); i++) {
        prn_cad(&fin, i);
    }

    // Asignar "INSERT COIN" a se_acabo
    strcpy(fin.se_acabo, "INSERT COIN");

    // Mostrar mensaje final
    printf("%s\n", fin.se_acabo);

    return 0;
}
```

https://onlinegdb.com/kDJ8Fe4RH