

## 1. DATOS INFORMATIVOS

Carrera: Electrónica y automatización

Asignatura: Fundamentos de la Programación

Tema : U3

Docente: Jenny Alexandra Ruiz Robalino

Integrantes: Ederson Gualoto

### Problema 3.1.1 Número perfecto

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

$$6=1+2+3=1+2+3=1+2+3.$$

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.

### Prueba de Escritorio

i	¿6 % i == 0?	suma
1	Sí	1
2	Sí	3
3	Sí	6
4	No	6
5	No	6

num	¿Perfecto?	encontrados	Se imprime	i	¿28 % i == 0?	suma
1	No	0	No	1	Sí	1
6	Sí	1	6	2	Sí	3
28	Sí	2	28	4	Sí	7
496	Sí	3	496	7	Sí	14
8128	Sí	4	8128	14	Sí	28

## CODIGO

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

```
int esperfecto(int n)
```

```
{
```

```
    int i, suma = 0;
```

```
    for (i = 1; i < n; i++)
```

```
{
```

```
    if (n % i == 0)
```

```
        suma += i;
```

```
}
```

```
    if (suma == n)
```

```
        return 1;
```

```
    else
```

```
        return 0;
```

```
}
```

```
int main(void)
```

```
{
```

```
    int num = 1;
```



```
int encontrados = 0;

printf("Los 4 primeros numeros perfectos son:\n");

while (encontrados < 4)

{

    if (esperfecto(num))

    {

        printf("%d\n", num);

        encontrados++;

    }

    num++;

}

}
```

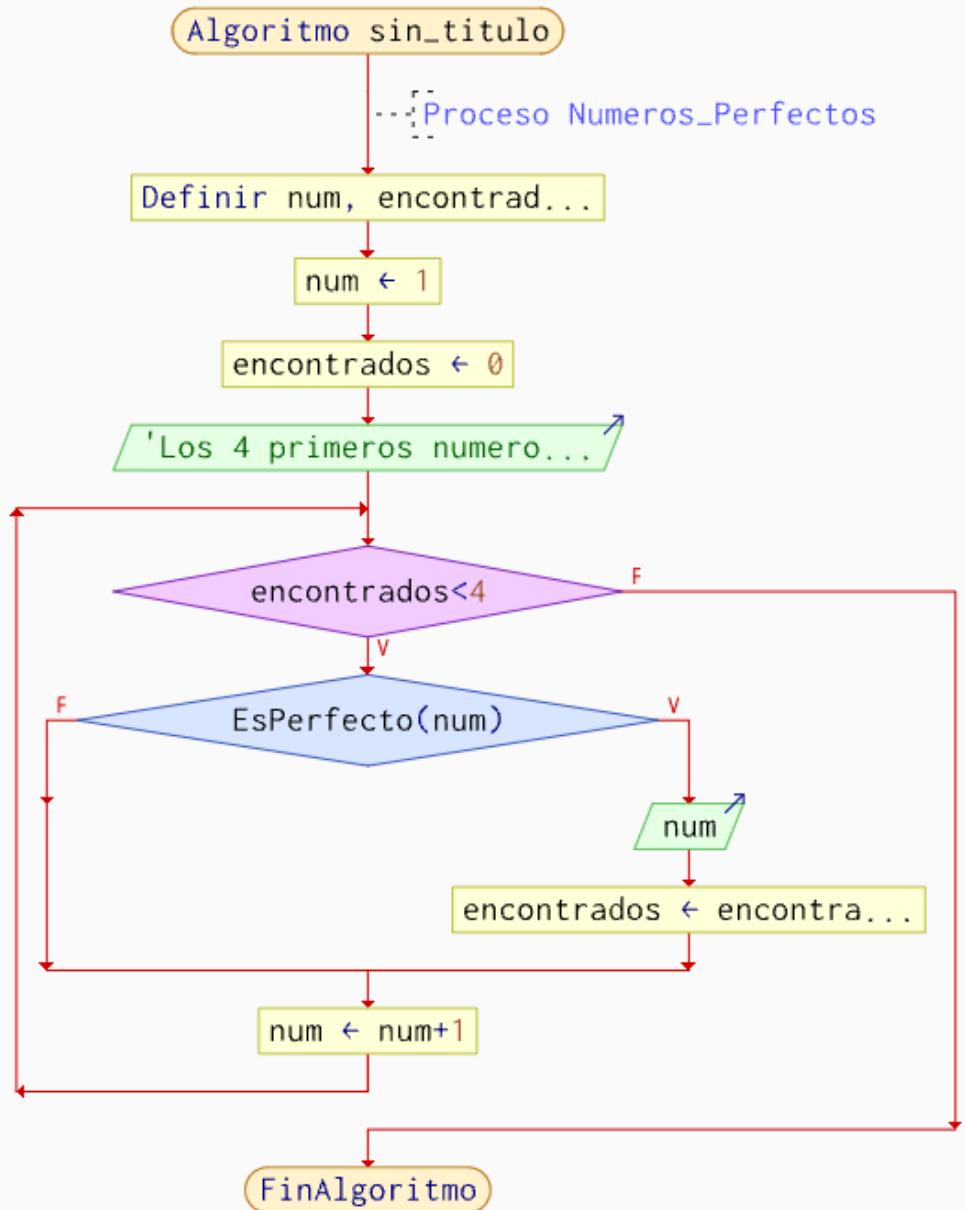
```
C:\Users\jhady\OneDrive\Escr + ▾

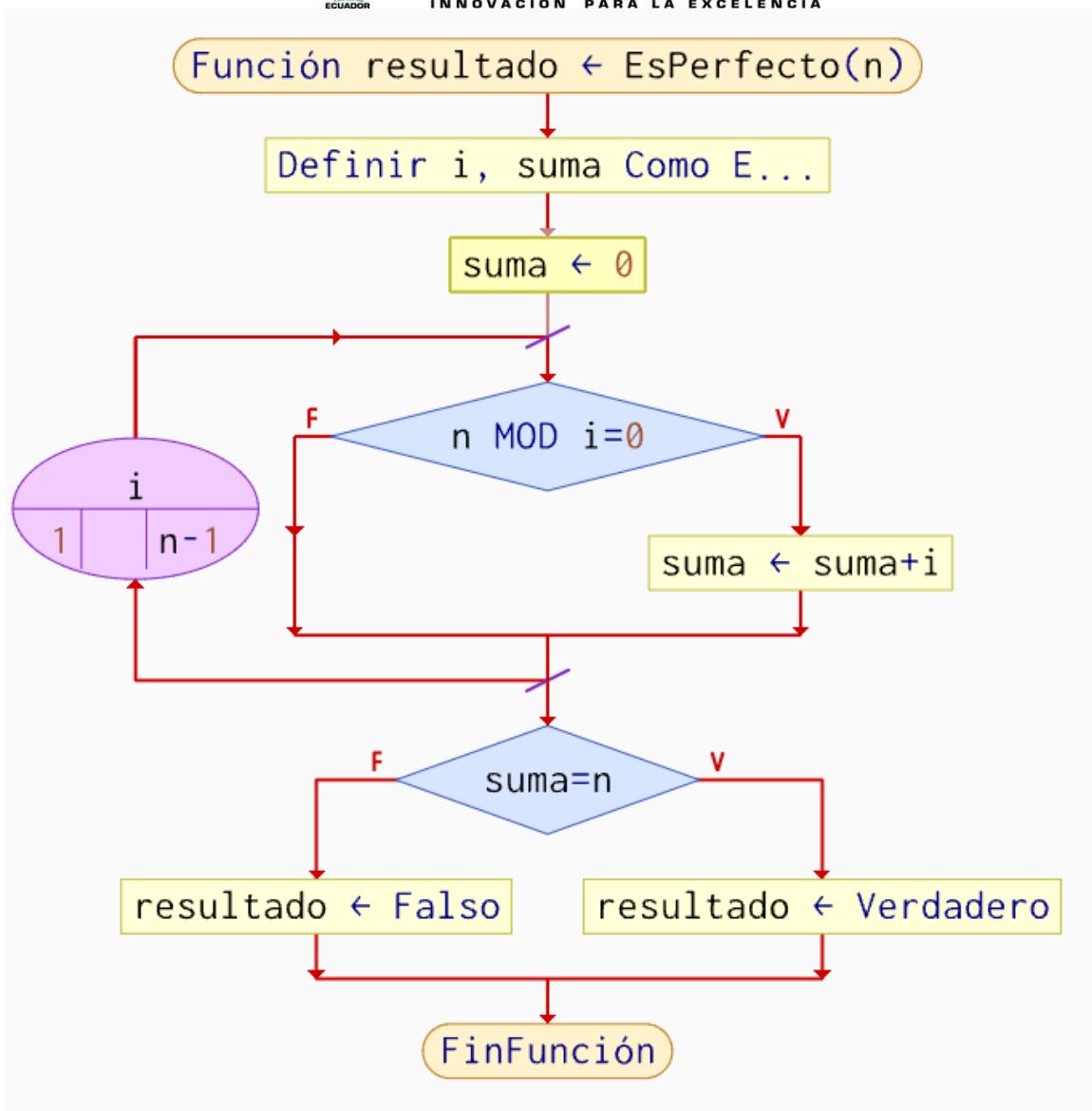
Los 4 primeros numeros perfectos son:
6
28
496
8128

Process returned 0 (0x0)  execution time : 0.186 s
Press any key to continue.
```



## Diagrama de flujo





### Problema 3.1.2 Números primos.

Realice un programa que resuelva adecuadamente los siguientes apartados:

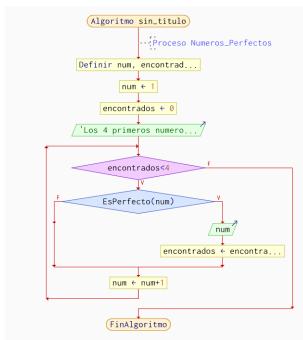
1. Programe la función **esprimo**, que recibe como argumento un número entero, y devuelve un valor **1** si el número es primo y **0** en caso contrario.
2. Almacene en un vector todos los números primos comprendidos entre dos números introducidos por teclado y luego imprima dicho vector.

### Tabla de objetos

i	Primos(i)	Salida de pantalla
0	11	11
1	13	13

2	17	17
3	19	19
4	23	23
5	29	29

## Diagrama de flujo en PSEINT



## Código en CODEBLOCKS

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

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

```
int esprimo(int num) {
    if (num <= 1) return 0;
    if (num == 2) return 1;
    if (num % 2 == 0) return 0;
    for (int i = 3; i <= sqrt(num); i += 2) {
        if (num % i == 0) return 0;
    }
    return 1;
}
```

```
int main() {
```

```
int num1, num2, cont = 0;

int primos[1000];

printf("Ingrese dos números enteros (límite inferior y superior): ");

scanf("%d %d", &num1, &num2);

if (num1 > num2) {

    int temp = num1;

    num1 = num2;

    num2 = temp;

}

for (int i = num1; i <= num2; i++) {

    if (esprimo(i)) {

        primos[cont] = i;

        cont++;

    }

}

printf("\nNúmeros primos entre %d y %d:\n", num1, num2);

for (int i = 0; i < cont; i++) {

    printf("%d ", primos[i]);

}

printf("\n");

return 0;
```

## Pantallazos de la ejecución

```
C:\Users\jhady\OneDrive\Escr X + ▾  
Ingrese dos números enteros (límite inferior y superior):  
15  
  
Números primos entre 1 y 15:  
2 3 5 7 11 13  
  
Process returned 0 (0x0) execution time : 2.648 s  
Press any key to continue.  
|
```

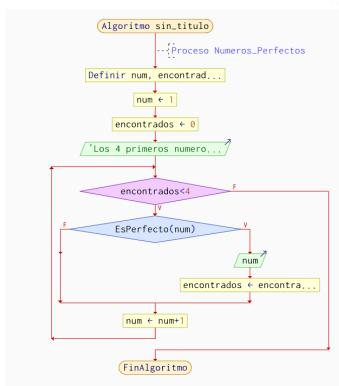
## Problema 3.1.3 Factorial.

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.

### Tabla de objetos

i	Salida de pantalla
0	numero
1	-----
2	0
3	1
4	2
5	3

### Diagrama de flujo en PSEINT



## Código en CODEBLOCKS

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

```
long long calc_fact(int num) {

    if (num < 0) return -1;

    long long resultado = 1;

    for (int i = 1; i <= num; i++) {

        resultado *= i;
    }

    return resultado;
}
```

```
int main() {

    int vec[15] = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 2, 5, 3, 1};

    long long fact[15];

    for (int i = 0; i < 15; i++) {

        fact[i] = calc_fact(vec[i]);
    }
}
```



```
printf("Número | Factorial\n");
printf("-----\n");
for (int i = 0; i < 15; i++) {
    if (fact[i] == -1) {
        printf("%6d | No existe (número negativo)\n", vec[i]);
    } else {
        printf("%6d | %lld\n", vec[i], fact[i]);
    }
}

return 0;
}
```

[Pantallazos de la ejecución](#)

C:\Users\jhady\OneDrive\Escr X + ▾

Número | Factorial

0	1
1	1
2	2
3	6
4	24
5	120
6	720
7	5040
8	40320
9	362880
10	3628800
2	2
5	120
3	6
1	1

Process returned 0 (0x0) execution time : 0.094 s  
Press any key to continue.

### Problema 3.1.4 Factorial recursivo.

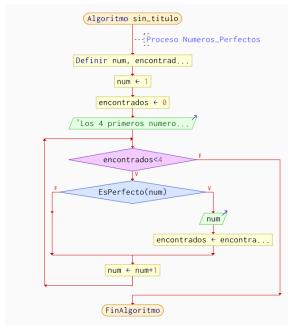
Programe la función **calc\_fact**, que recibe como argumento un número entero, y devuelva el valor de la factorial realizando el cálculo de manera recursiva. A continuación, use esta función en un programa que, dado un vector de 15 números enteros **vec**, calcule un vector **fact** con sus factoriales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

#### Tabla de objetos

i	Salida de pantalla
0	numero

1	-----
2	0
3	1
4	2
5	3

## Diagrama de flujo en PSEINT



## Código en CODEBLOCKS

```
#include <stdio.h>

long long calc_fact(int num) {

    if (num < 0) return -1; // Manejo de números negativos
    if (num == 0 || num == 1) return 1; // Caso base: 0! = 1 y 1! = 1
    return num * calc_fact(num - 1); // Paso recursivo
}

int main() {

    int vec[15] = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, -2, 5, 3, 12};

    long long fact[15];

    for (int i = 0; i < 15; i++) {
        fact[i] = calc_fact(vec[i]);
    }
}
```

```
}

printf("Número | Factorial\n");
printf("-----\n");
for (int i = 0; i < 15; i++) {
    if (fact[i] == -1) {
        printf("%6d | No existe (negativo)\n", vec[i]);
    } else {
        printf("%6d | %lld\n", vec[i], fact[i]);
    }
}

return 0;
}
```

[Pantallazos de la ejecución](#)

```
C:\Users\jhady\OneDrive\Escr X + ▾
Número | Factorial
-----
0 | 1
1 | 1
2 | 2
3 | 6
4 | 24
5 | 120
6 | 720
7 | 5040
8 | 40320
9 | 362880
10 | 3628800
-2 | No existe (negativo)
5 | 120
3 | 6
12 | 479001600

Process returned 0 (0x0)    execution time : 0.083 s
Press any key to continue.
```

#### 4.1. DESARROLLO

Describa aquí el procedimiento, actividades realizadas y resultados obtenidos durante el desarrollo del taller. Use subtítulos, numeraciones o diagramas si es necesario.

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.

#### 4.1.1 código

```
#include<stdio.h>

#include<stdlib.h>

#include<string.h>

struct poblacion

{

char nombre[100];

int edad;

char ciudad[100];

int rep;

};

typedef struct poblacion pob;

void main(void)

{



int i,j,n=0;
```

```
int max=0;

float media=0;

pob v[10];

while(n<1 || n>10)

{

printf("Introduzca numero de personas\n");

scanf("%d",&n);

}

for(i=0; i<n; i++)

{

system("cls");

printf("Introduzca nombre\n");

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

printf("\nIntroduzca edad\n");

scanf("%d",&v[i].edad);

media+=v[i].edad;

printf("\nIntroduzca ciudad de nacimiento\n");

scanf("%s", v[i].ciudad);

}

media=media/n;

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++;
```

```
}
```

```
for(i=0; i<n; i++)
```

```
{
```

```
if(max<v[i].rep)
```

```
    max=v[i].rep;
```

```
}
```

```
printf("La edad media es %1.2f:\n", media);
```

```
printf("El/los nombres mas frecuentes son:\n");
```

```
for(i=0; i<n; i++)
```

```
{
```

```
if (max==v[i].rep)
```

```
{
```

```
    printf("%s, nacido en %s\n",v[i].nombre,v[i].ciudad);
```

```
}
```

```
}
```

```
}
```

```
"C:\Users\PC-PR\Desktop\Ejer" + ▾
```

Introduzca numero de personas  
10

```
"C:\Users\PC-PR\Desktop\Ejer" + ▾
```

Introduzca nombre  
lizz

Introduzca edad  
31

Introduzca ciudad de nacimiento  
Loja

```
"C:\Users\PC-PR\Desktop\Ejer" + ▾
```

Introduzca nombre  
Juan

Introduzca edad  
19

Introduzca ciudad de nacimiento  
Loja

```
"C:\Users\PC-PR\Desktop\Ejer" + ▾
```

Introduzca nombre  
Lizz

Introduzca edad  
25

Introduzca ciudad de nacimiento  
Cuenca

```
Ejercicio 1
Introduzca nombre
Luis

Introduzca edad
30

Introduzca ciudad de nacimiento
Loja
```

```
Ejercicio 1
Introduzca nombre
Alex
}

Introduzca edad
18

Introduzca ciudad de nacimiento
Quito
i
```

```
Ejercicio 1
Introduzca nombre
Darwin

Introduzca edad
20

Introduzca ciudad de nacimiento
Loja
[]
```

```
Ejercicio 1
Introduzca nombre
Carla

Introduzca edad
21

Introduzca ciudad de nacimiento
Quito
i]
```

```
"C:\Users\PC-PR\Desktop\Ejercicio1.py" + ▾  
  
Introduzca nombre  
Liz  
  
Introduzca edad  
26  
  
Introduzca ciudad de nacimiento  
Tungurahua
```

```
"C:\Users\PC-PR\Desktop\Ejercicio1.py" + ▾  
  
Introduzca nombre  
Ederson  
  
Introduzca edad  
19  
  
Introduzca ciudad de nacimiento  
Quito
```

```
"C:\Users\PC-PR\Desktop\Ejercicio1.py" + ▾  
  
Introduzca nombre  
Alex  
  
Introduzca edad  
24  
  
Introduzca ciudad de nacimiento  
Quito
```

```
C:\ "C:\Users\PC-PR\Desktop\Ejer" X + ▾
La edad media es 23.30:
El/los nombres mas frecuentes son:
Alex, nacido en Quito
Liz, nacido en Tungurahua
Alex, nacido en Quito
Liz, nacido en Guayaquil

Process returned 10 (0xA) execution time : 180.312 s
Press any key to continue.
```

#### 4.1.2 Prueba de escritorio

Persona Nombre Edad Ciudad

1	Alex	18	Quito
2	Liz	26	Guayaquil
3	Alex	24	Quito
4	Carla	21	Quito
5	Darwin	20	Quito

#### Los nombres que se repiten son

Alex 18 Quito

Alex 24 Quito

#### 4.1.3 Tabla de contenido

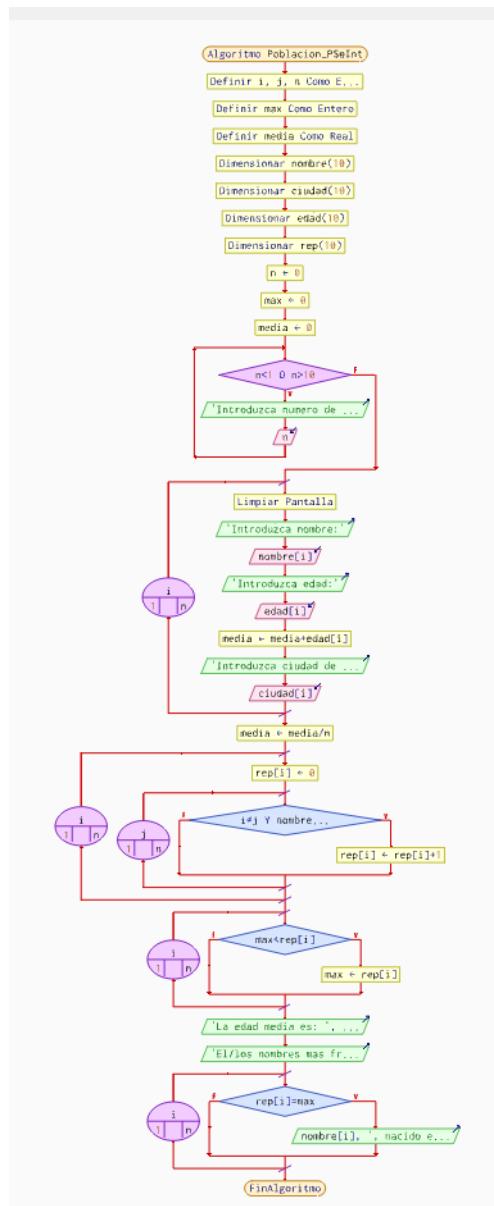
Objeto	Valor	Tipo
Nombre	Edad	Ciudad de nacimiento
Alex	24	Quito
Ederson	19	Quito
Liz	26	Tungurahua
Carla	21	Quito
Darwin	20	Loja
Alex	18	Quito
Luis	30	Loja
Lizz	25	Cuenca
Liz	31	Guayaquil
Juan	19	Loja

En los nombres mas comunes en esta tabla existen 2 que son los que repiten con mayor frecuencia

Alex: ambos nacidos en Quito

Liz: con una variante de ciudad (Guayaquil y Tungurahua)

#### 4.2.4 Diagrama de Flujo



## 4.2 DESARROLLO

Describa aquí el procedimiento, actividades realizadas y resultados obtenidos durante el desarrollo del taller. Use subtítulos, numeraciones o diagramas si es necesario.

**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  $8 \times 8$  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 mp[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  $8 \times 8$  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.*

Para realizar el apartado anterior, debe comparar el mapa de puntos de la letra a identificar ( $mp$ ) con cada uno de los mapas de las 27 letras del alfabeto en  $tab\_let$ , para obtener el carácter más parecido. El carácter más parecido será aquel que maximice el valor  $\frac{PC}{64}$ , donde  $PC$  es el número de puntos del mapa que coinciden, bien a cero o a uno, en los dos caracteres comparados. Es decir,  $\frac{PC}{64}$  toma valores entre 0 y 1. (1 si los dos caracteres fuesen idénticos y 0 si fuesen completamente diferentes).

Ejemplo: La letra a del ejemplo de la figura 4.1, se describe como una estructura de tipo **struct letras** donde el campo **cod\_ASCII** contendrá el valor 'a', y la matriz de enteros **mptos** será la matriz de ceros y unos que se representa.

Este problema se resuelve comparando el mapa  $mp$  con los mapas correspondientes a cada uno de los 27 caracteres almacenados en  $tab\_let$ . Para cada letra del abecedario se calculará el valor  $\frac{PC}{64}$ , guardándose en la variable auxiliar **max** el valor del máximo alcanzado hasta el momento y en la variable **caracter**, el código ASCII de la letra que ha obtenido el máximo. Una vez comparadas las 27 letras se devolverá el código ASCII almacenado en **caracter**.

#### 4.2.1 código

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

```
#define N 8
```

```
typedef struct {
```

```
    char cod_ASCII;
```

```
    char mpost[N][N];
```

```
} letras;
```

```
int distancia(const char a[N][N], const char b[N][N]) {
```

```
    int d = 0;
```

```
    for (int i = 0; i < N; i++) {
```

```
        for (int 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;

a\_min.cod\_ASCII = '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','#'}

};

for (i = 0; i < N; i++)

for (j = 0; j < N; j++)

a\_min.mpost[i][j] = patron\_a[i][j];

```
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]);

    }

}

d = distancia(mp, a_min.mpost);

if (d == 0)

    printf("\nLa matriz corresponde a la letra 'a' minuscula\n");

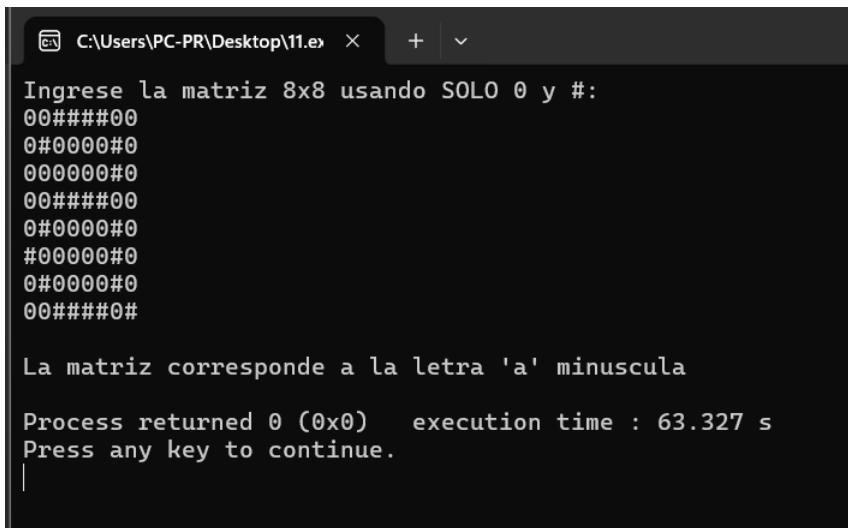
else

    printf("\nLa matriz NO corresponde a la letra 'a' minuscula\n");



return 0;

}
```



```
C:\Users\PC-PR\Desktop\11.exe  +  ▾

Ingrese la matriz 8x8 usando SOLO 0 y #:
00#####00
0#0000#0
000000#0
00#####00
0#0000#0
#00000#0
0#0000#0
00#####0#

La matriz corresponde a la letra 'a' minuscula

Process returned 0 (0x0)  execution time : 63.327 s
Press any key to continue.
```

#### 4.2.2 tabla de contenido

Objeto	Nombre	Valor	Tipo
Valor numerico	0	Constante	Real
Carácter	#	Constante	Carácter

La letra que representara es la “a” minúscula

#### 4.2.3 Prueba de escritorio

- . la ubicación de cada uno de los “#” se ubica puntos estratégicos, lo que hacen formar un carácter o una letra en este caso la “a” minúscula

0 0 ##### 0

0 # 0 0 0 0 # 0

0 0 0 0 0 0 # 0

0 0 ##### 0 0

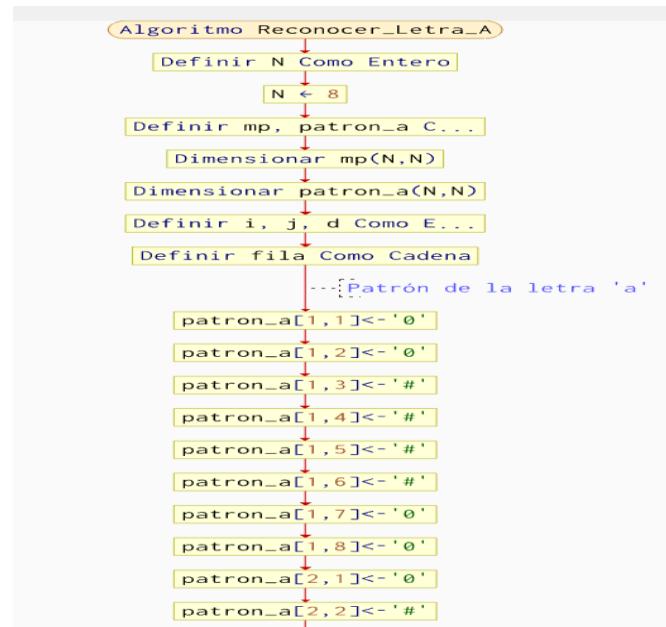
0 # 0 0 0 0 # 0

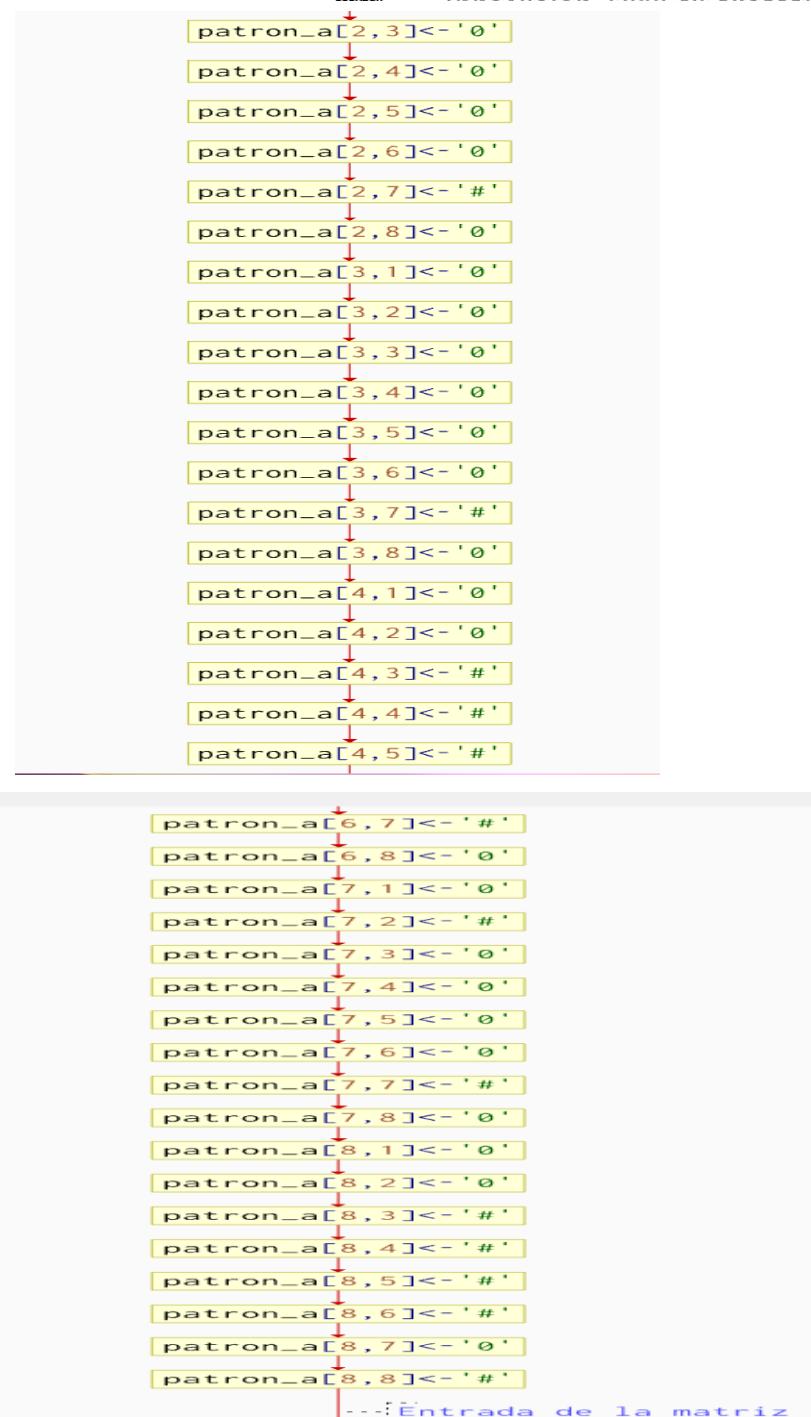
# 0 0 0 0 0 # 0

0 # 0 0 0 0 # 0

0 0 ##### 0 #

#### 4.2.4 diagrama de flujo

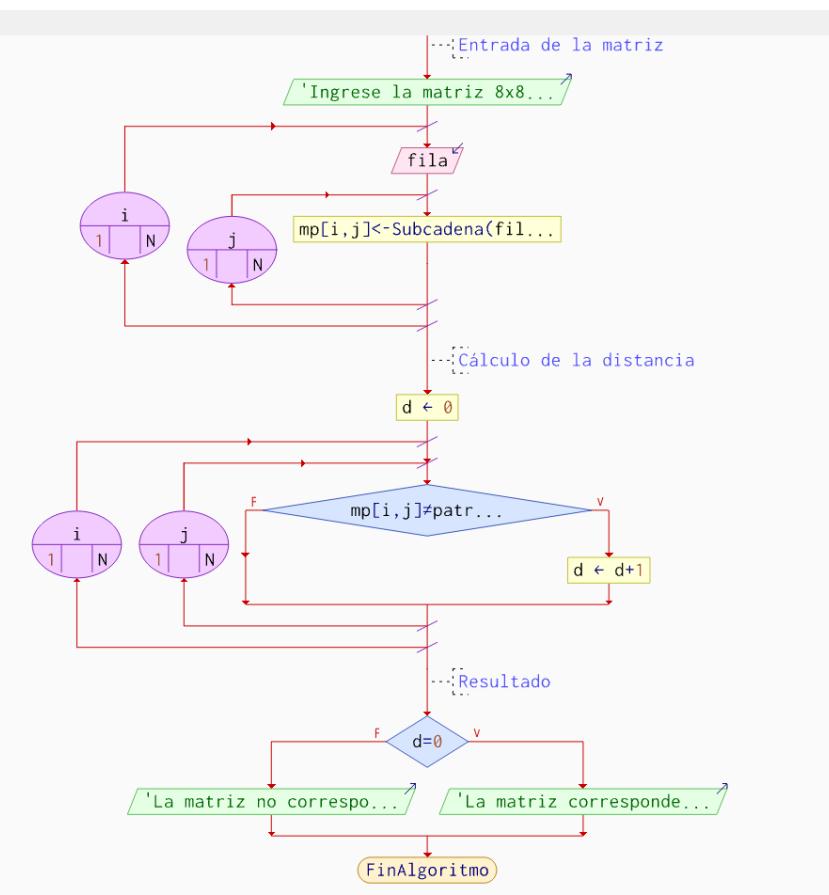




```

patron_a[4, 5]<- '#'
patron_a[4, 6]<- '#'
patron_a[4, 7]<- '0'
patron_a[4, 8]<- '0'
patron_a[5, 1]<- '0'
patron_a[5, 2]<- '#'
patron_a[5, 3]<- '0'
patron_a[5, 4]<- '0'
patron_a[5, 5]<- '0'
patron_a[5, 6]<- '0'
patron_a[5, 7]<- '#'
patron_a[5, 8]<- '0'
patron_a[6, 1]<- '#'
patron_a[6, 2]<- '0'
patron_a[6, 3]<- '0'
patron_a[6, 4]<- '0'
patron_a[6, 5]<- '0'
patron_a[6, 6]<- '0'
patron_a[6, 7]<- '#'

```



### Problema 4.3 Polígono

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;      /* Coordenada x */
    float y;      /* Coordenada y */
};

struct poligono
{
    int nvert;           /* 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.

### Prueba de Escritorio

Vértice	x	y	i	xi	yi	x(i+1)	y(i+1)	xi·y(i+1)	x(i+1)·yi	Resultado
1	0	0	0	0	0	4	0	0	0	0
2	4	0	1	4	0	4	4	16	0	+16
3	4	4	2	4	4	0	4	16	0	+16
4	0	4								

### CODIGO

```
#include <stdio.h>

#include <stdlib.h>

struct punto

{
```



```
float x; /* Coordenada x */  
float y; /* Coordenada y */  
};
```

```
struct poligono  
{  
    int nvert;      /* Número de vértices */  
    struct punto vert[100]; /* Vector de vértices */  
};
```

```
float area(struct poligono p)  
{  
    float area = 0;  
    int i;  
  
    area = p.vert[p.nvert - 1].x * p.vert[0].y -  
          p.vert[0].x * p.vert[p.nvert - 1].y;  
  
    for (i = 0; i < p.nvert - 1; i++)  
        area += (p.vert[i].x * p.vert[i + 1].y -  
                  p.vert[i + 1].x * p.vert[i].y);  
    return area / 2;  
}
```

```
int main(void)  
{  
    struct poligono p;  
    int i;
```

```
printf("Introduzca el numero de vertices:\n");

scanf("%d", &p.nvert);

printf("Introduzca los vertices de la forma x y:\n");

for (i = 0; i < p.nvert; i++)

{

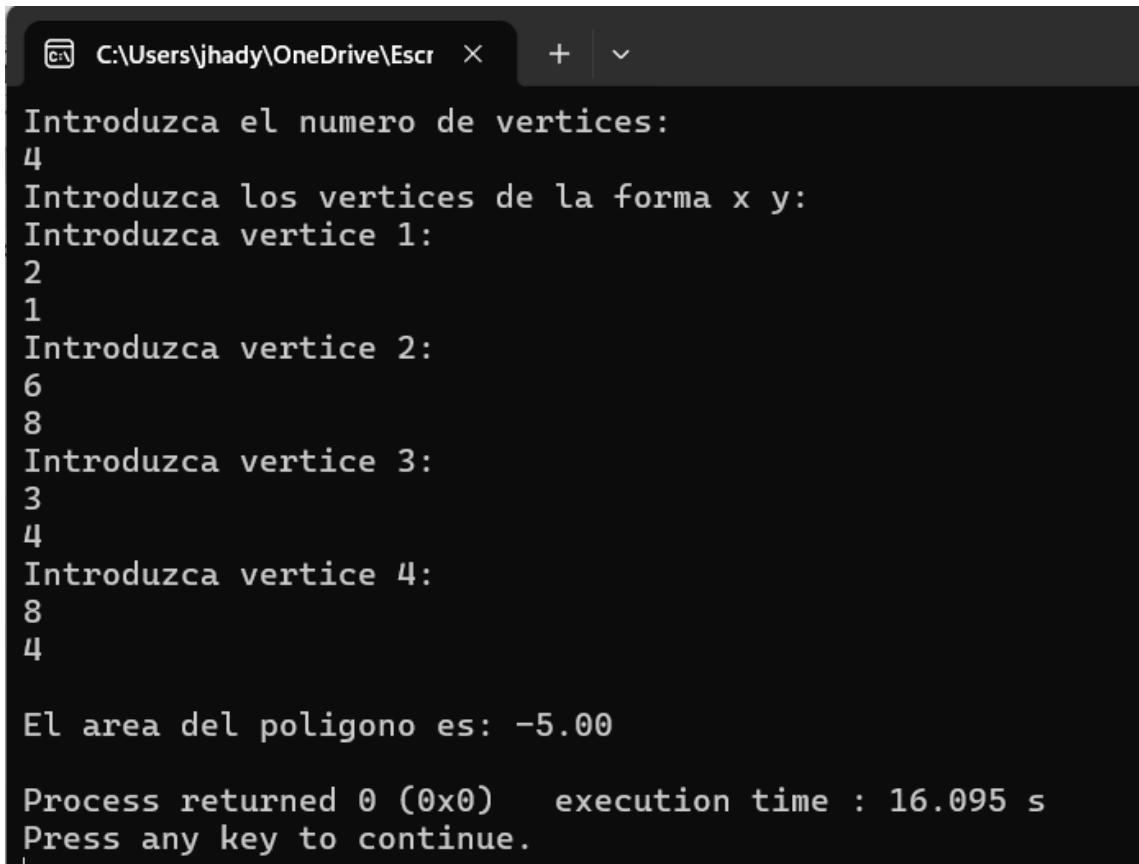
    printf("Introduzca vertice %d:\n", i + 1);

    scanf("%f %f", &p.vert[i].x, &p.vert[i].y);

}

printf("\nEl area del poligono es: %.2f\n", area(p));

}
```



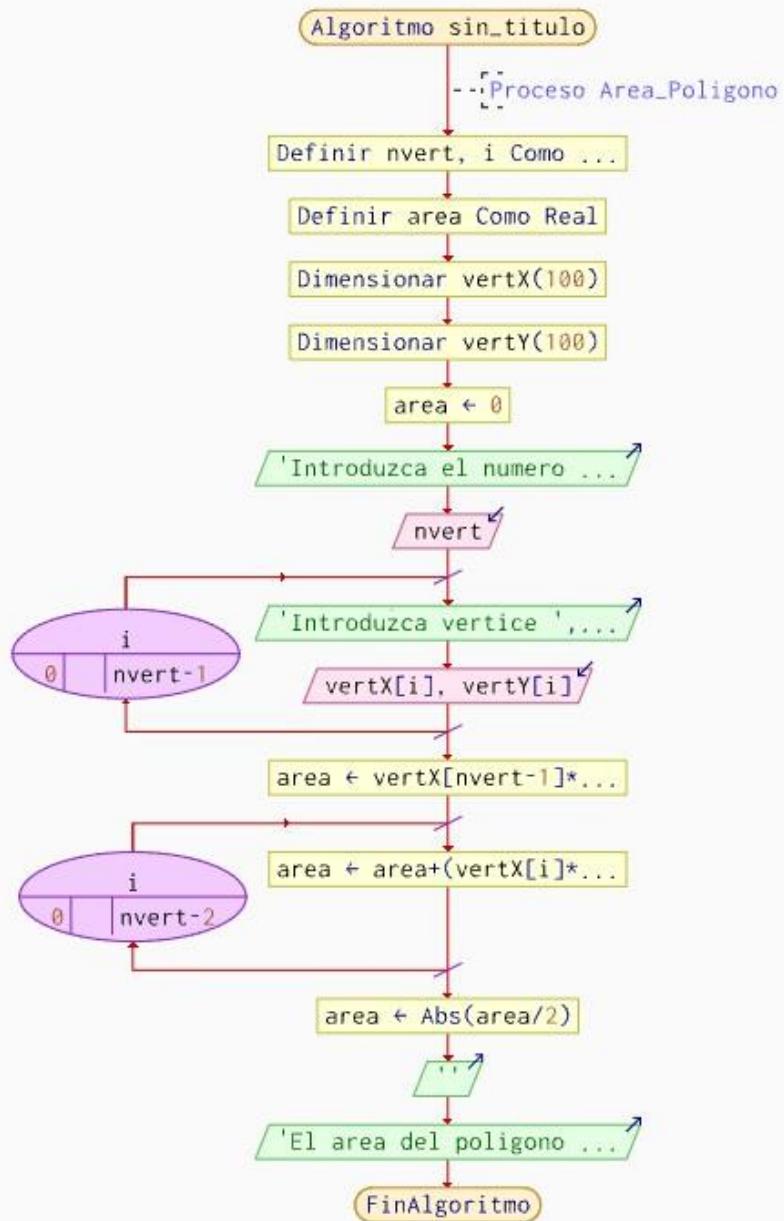
```
C:\Users\jhady\OneDrive\Escr  +  ▾

Introduzca el numero de vertices:
4
Introduzca los vertices de la forma x y:
Introduzca vertice 1:
2
1
Introduzca vertice 2:
6
8
Introduzca vertice 3:
3
4
Introduzca vertice 4:
8
4

El area del poligono es: -5.00

Process returned 0 (0x0)  execution time : 16.095 s
Press any key to continue.
```

## Diagrama de Flujo



### Problema 4.4 Game Over

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
```

### Prueba de escritorio

INICIO	Variable (i)	%*s (i, game_over)	Salida en pantalla
1	1	"G"	G
2	2	"GA"	GA
3	3	"GAM"	GAM
4	4	"GAME"	GAME
5	5	"GAME "	GAME
6	6	"GAME O"	GAME O
7	7	"GAME OV"	GAME OV
8	8	"GAME OVE"	GAME OVE
9	9	"GAME OVER"	GAME OVER

### CODIGO



```
#include <stdio.h>

#include <string.h>

#define N 10

#define M 12

struct mensaje

{

    char game_over[N];

    char se_acabo[M];

};

typedef struct mensaje men;

void prn_cad(men *m, int n)

{

    int i = 0;

    while (i <= n)

    {

        printf("%c", m->game_over[i]);

        i++;

    }

    printf("\n");

    strcpy(m->se_acabo, "INSERT COIN");

}

int main(void)

{

    men fin = {"GAME OVER", ""};

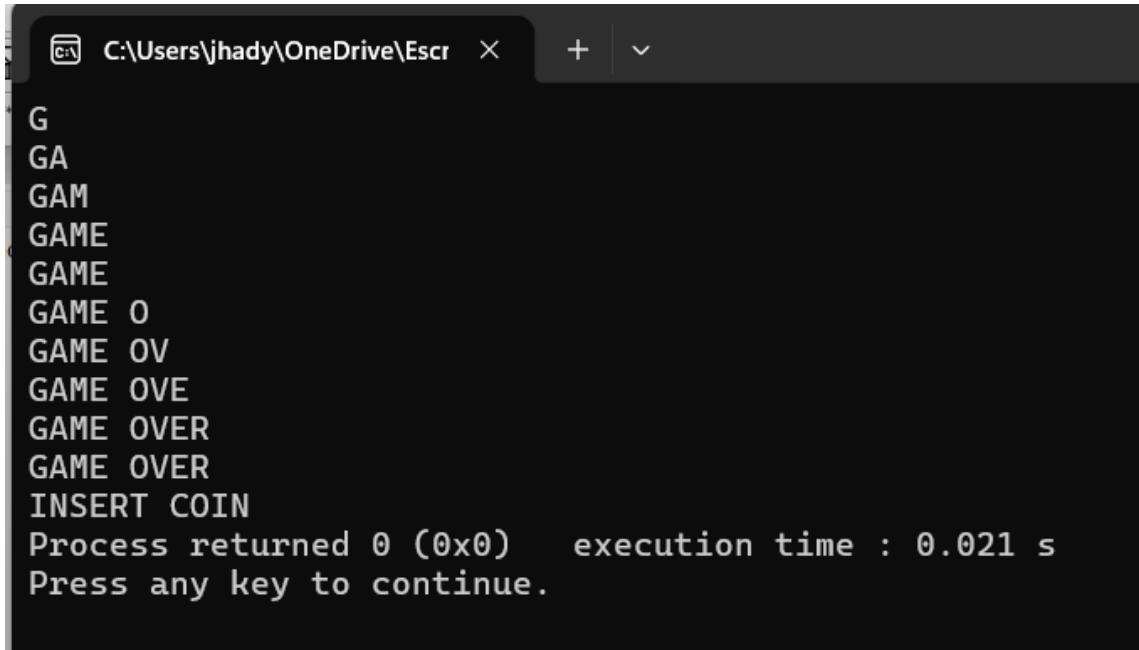
    int i;

    for (i = 0; i < N; i++)

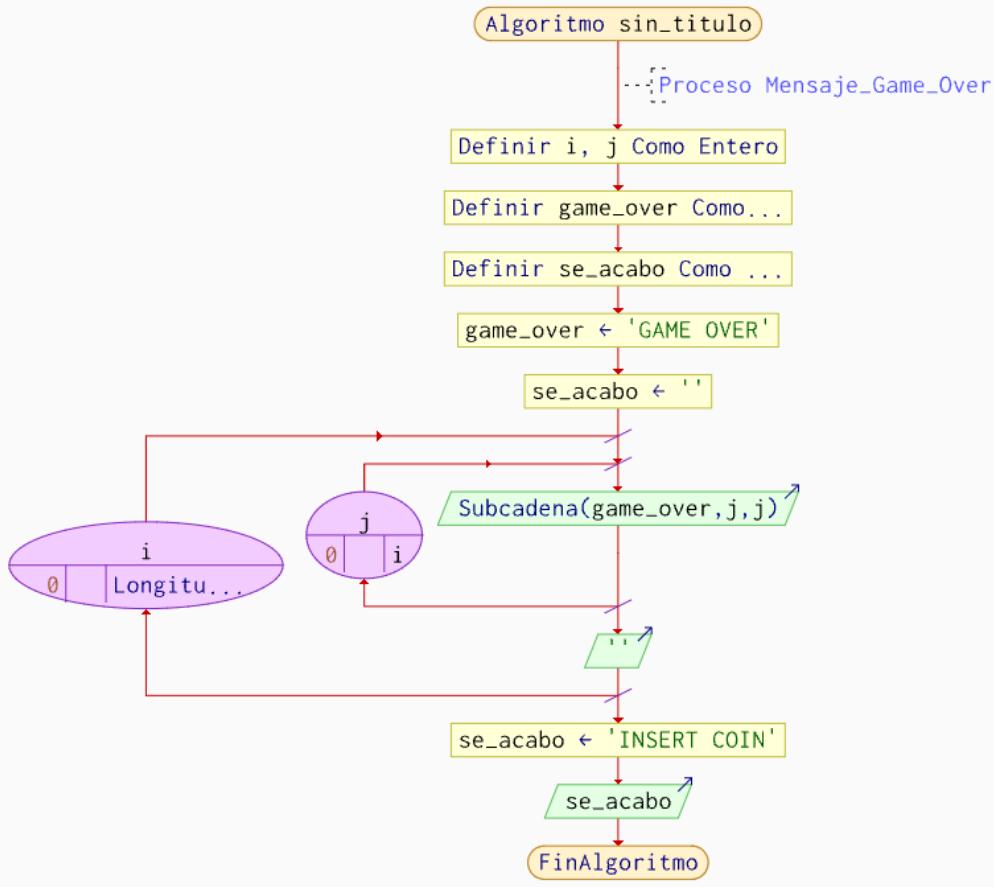
        prn_cad(&fin, i);

}
```

```
printf("%s", fin.se_acabo);  
}
```



```
C:\Users\jhady\OneDrive\Escr + v  
G  
GA  
GAM  
GAME  
GAME  
GAME 0  
GAME OV  
GAME OVE  
GAME OVER  
GAME OVER  
INSERT COIN  
Process returned 0 (0x0) execution time : 0.021 s  
Press any key to continue.
```



### 13. Laboratorio propuesto (entregable práctico)

Objetivo: implementar el CRUD completo y probarlo con al menos 10 registros.

1. Crear el proyecto en Code::Blocks y pegar el código en un solo archivo main.c.

2. Ejecutar y agregar 3 estudiantes. Verificar que se cree estudiantes.txt y que cada estudiante quede en una línea.

#### Fundamentos de Programación

Página 10

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#define ARCHIVO_EST "estudiantes.txt"
```

```
typedef struct {
  char id[20];
  char apellidos[50];
  char nombres[50];
  int edad;
} Estudiante;
```



```
// Elimina el '\n' que deja fgets si existe.
void limpiarNuevaLinea(char *s) {
    s[strcspn(s, "\n")] = '\0';
}

// Lee una cadena con etiqueta, usando fgets.
void leerCadena(const char *etiqueta, char *dest, int tam) {
    printf("%s", etiqueta);
    fgets(dest, tam, stdin);
    limpiarNuevaLinea(dest);
}

// Lee un entero de forma robusta usando fgets + strtol.
int leerEntero(const char *etiqueta) {
    char buf[50];
    long val;
    char *endptr;
    while (1) {
        printf("%s", etiqueta);
        fgets(buf, sizeof(buf), stdin);
        val = strtol(buf, &endptr, 10);
        if (endptr != buf) { // hubo conversión
            return (int)val;
        }
        printf("Entrada inválida. Intente de nuevo.\n");
    }
}

// Convierte una línea "id;apellidos;nombres;edad" a estructura Estudiante.
// Retorna 1 si pudo parsear, 0 si la línea es inválida.
int parsearEstudiante(const char *linea, Estudiante *e) {
    // %[^;] = lee hasta encontrar ;
    int n = sscanf(linea, "%19[^;];%49[^;];%49[^;];%d",
                   e->id, e->apellidos, e->nombres, &e->edad);
    return (n == 4);
}

int existeid(const char *idBuscado) {
    FILE *f = fopen(ARCHIVO_EST, "r");
    if (f == NULL) return 0; // si no existe el archivo, no hay IDs guardados
    char linea[200];
    Estudiante e;
    while (fgets(linea, sizeof(linea), f)) {
        if (parsearEstudiante(linea, &e) && strcmp(e.id, idBuscado) == 0) {
            fclose(f);
            return 1;
        }
    }
    fclose(f);
    return 0;
}
```



```
void agregarEstudiante() {
    Estudiante nuevo;
    leerCadena("Ingrese ID: ", nuevo.id, sizeof(nuevo.id));

    if (existId(nuevo.id)) {
        printf("Ya existe un estudiante con ese ID.\n");
        return;
    }
    leerCadena("Ingrese apellidos: ", nuevo.apellidos, sizeof(nuevo.apellidos));
    leerCadena("Ingrese nombres: ", nuevo.nombres, sizeof(nuevo.nombres));
    nuevo.edad = leerEntero("Ingrese edad: ");
    FILE *f = fopen(ARCHIVO_EST, "a");
    if (f == NULL) {
        printf("Error: no se pudo abrir el archivo.\n");
        return;
    }
    fprintf(f, "%s;%s;%s;%d\n", nuevo.id, nuevo.apellidos, nuevo.nombres,
            nuevo.edad);
    fclose(f);
    printf("Estudiante agregado correctamente.\n");
}

void listarEstudiantes() {
    FILE *f = fopen(ARCHIVO_EST, "r");
    if (f == NULL) {
        printf("No hay datos aún (archivo inexistente).\n");
        return;
    }
    char linea[200];
    Estudiante e;
    printf("\n%-10s %-20s %-20s %-5s\n", "ID", "APELIDOS", "NOMBRES", "EDAD");
    printf("-----\n");
    while (fgets(linea, sizeof(linea), f)) {
        if (parsearEstudiante(linea, &e)) {
            printf("%-10s %-20s %-20s %-5d\n", e.id, e.apellidos, e.nombres,
                   e.edad);
        }
    }
    fclose(f);
}

int buscarPorId(const char *idBuscado, Estudiante *salida) {
    FILE *f = fopen(ARCHIVO_EST, "r");
    if (f == NULL) return 0;
    char linea[200];
    Estudiante e;
    while (fgets(linea, sizeof(linea), f)) {
        if (parsearEstudiante(linea, &e) && strcmp(e.id, idBuscado) == 0) {
            *salida = e;
        }
    }
}
```



```
fclose(f);
return 1;
}
}
fclose(f);
return 0;
}
void consultarEstudiante() {
char id[20];
Estudiante e;
leerCadena("Ingrese ID a buscar: ", id, sizeof(id));
if (buscarPorId(id, &e)) {
printf("Encontrado: %s %s (Edad: %d)\n", e.nombres, e.apellidos,
e.edad);
} else {
printf("No se encontró el ID.\n");
}
}
void actualizarEstudiante() {
char idObjetivo[20];
leerCadena("Ingrese ID a actualizar: ", idObjetivo, sizeof(idObjetivo));
FILE *f = fopen(ARCHIVO_EST, "r");
if (f == NULL) {
printf("No existe el archivo.\n");
return;
}
FILE *temp = fopen("tmp.txt", "w");
if (temp == NULL) {
fclose(f);
printf("No se pudo crear archivo temporal.\n");
return;
}
char linea[200];
Estudiante e;
int actualizado = 0;
while (fgets(linea, sizeof(linea), f)) {
if (parsearEstudiante(linea, &e) && strcmp(e.id, idObjetivo) == 0) {
// Solicitar nuevos datos
leerCadena("Nuevos apellidos: ", e.apellidos, sizeof(e.apellidos));
leerCadena("Nuevos nombres: ", e.nombres, sizeof(e.nombres));
e.edad = leerEntero("Nueva edad: ");
fprintf(temp, "%s;%s;%s;%d\n", e.id, e.apellidos, e.nombres,
e.edad);
actualizado = 1;
} else {
// Copia la línea original (si está bien formateada, mejor reescribirla)
if (parsearEstudiante(linea, &e)) {
```

```

fprintf(temp, "%s;%s;%s;%d\n", e.id, e.apellidos, e.nombres,
e.edad);
}
}
}
fclose(f);
fclose(temp);
remove(ARCHIVO_EST);
rename("tmp.txt", ARCHIVO_EST);
if (actualizado) printf("Registro actualizado.\n");
else printf("No se encontró el ID.\n");
}
void eliminarEstudiante() {
char idEliminar[20];
leerCadena("Ingrese ID a eliminar: ", idEliminar, sizeof(idEliminar));
FILE *f = fopen(ARCHIVO_EST, "r");
if (f == NULL) {
printf("No existe el archivo.\n");
return;
}
FILE *temp = fopen("tmp.txt", "w");
if (temp == NULL) {
fclose(f);
printf("No se pudo crear archivo temporal.\n");
return;
}
char linea[200];
Estudiante e;
int eliminado = 0;
while (fgets(linea, sizeof(linea), f)) {
if (parsearEstudiante(linea, &e)) {
if (strcmp(e.id, idEliminar) != 0) {
fprintf(temp, "%s;%s;%s;%d\n", e.id, e.apellidos, e.nombres,
e.edad);
} else {
eliminado = 1;
}
}
}
fclose(f);
fclose(temp);
remove(ARCHIVO_EST);
rename("tmp.txt", ARCHIVO_EST);
if (eliminado) printf("Registro eliminado.\n");
else printf("No se encontró el ID.\n");
}
int menu() {

```

```
char opcion[10];
printf("\n==== CRUD Estudiantes (TXT) ====\n");
printf("1. Agregar\n");
printf("2. Listar\n");
printf("3. Consultar por ID\n");
printf("4. Actualizar\n");
printf("5. Eliminar\n");
printf("0. Salir\n");
printf("Seleccione una opción: ");
fgets(opcion, sizeof(opcion), stdin);
return atoi(opcion);
}
int main() {
int op;
do {
op = menu();
switch (op) {
case 1: agregarEstudiante(); break;
case 2: listarEstudiantes(); break;
case 3: consultarEstudiante(); break;
case 4: actualizarEstudiante(); break;
case 5: eliminarEstudiante(); break;
case 0: printf("Saliendo...\n"); break;
default: printf("Opción inválida.\n");
}
} while (op != 0);
return 0;
}
```

3. Ejecutar y agregar 3 estudiantes. Verificar que se cree estudiantes.txt y que cada estudiante quede en una línea.

```
==== CRUD Estudiantes (TXT) ====
```

1. Agregar
2. Listar
3. Consultar por ID
4. Actualizar
5. Eliminar
0. Salir

```
Seleccione una opci n: 2
```

ID	APELLIDOS	NOMBRES	EDAD
L00457013	Tapia	Darwin	20
L00456713	Chuquimarpa	Alex	20
L00457895	Gualoto	Ederson	20

1. 3. Probar la validaci n de ID duplicado (intentar agregar el mismo ID dos veces).

```
==== CRUD Estudiantes (TXT) ====
```

1. Agregar
2. Listar
3. Consultar por ID
4. Actualizar
5. Eliminar
0. Salir

```
Seleccione una opci n: 3
```

```
Ingr se ID a buscar: L00456713
```

```
Encontrado: Alex Chuquimarpa (Edad: 20)
```

```
==== CRUD Estudiantes (TXT) ====
```

1. Agregar
2. Listar
3. Consultar por ID
4. Actualizar
5. Eliminar
0. Salir

```
Seleccione una opci n: 3
```

```
Ingr se ID a buscar: L00456083
```

```
No se encontr  el ID.
```

2. 4. Listar y verificar el formato de tabla en consola.

```
==== CRUD Estudiantes (TXT) ====
```

1. Agregar
2. Listar
3. Consultar por ID
4. Actualizar
5. Eliminar
0. Salir

Seleccione una opción: 4

Ingresé ID a actualizar: L00456713

Nuevos apellidos: Pablo

Nuevos nombres: Jose

Nueva edad: 25

Registro actualizado.

```
==== CRUD Estudiantes (TXT) ====
```

1. Agregar
2. Listar
3. Consultar por ID
4. Actualizar
5. Eliminar
0. Salir

Seleccione una opción: 2

ID	APELLIDOS	NOMBRES	EDAD
L00457013	Tapia	Darwin	20
L00456713	Pablo	Jose	25
L00457895	Gualoto	Ederson	20

3. 5. Consultar por ID existente y por uno inexistente.



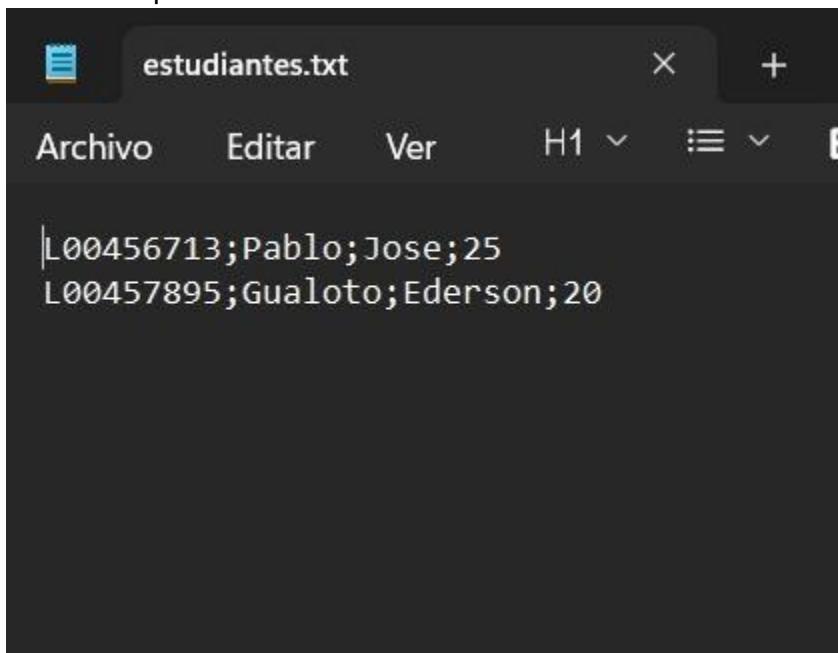
4. 6. Actualizar un estudiante y verificar que los cambios persistan al listar de nuevo.
  5. 7. Eliminar un estudiante y verificar que ya no aparece.

```
==> CRUD Estudiantes (TXT) ==>
1. Agregar
2. Listar
3. Consultar por ID
4. Actualizar
5. Eliminar
0. Salir
Seleccione una opci n: 5
Ingrese ID a eliminar: L00457013
Registro eliminado.

==> CRUD Estudiantes (TXT) ==>
1. Agregar
2. Listar
3. Consultar por ID
4. Actualizar
5. Eliminar
0. Salir
Seleccione una opci n: 2
```

ID	APELLIDOS	NOMBRES	EDAD
L00456713	Pablo	Jose	25
L00457895	Gualoto	Ederson	20

6. 8. Abrir estudiantes.txt con un editor de texto y comprobar que el formato sigue siendo ";" por campos.



Extensión (opcional): Implementar una “baja lógica” agregando un campo estado (A/I) y omitir en listados los registros inactivos.

