UNIVERSIDAD DE LAS FUERZAS ARMADAS ESPE

Nombre:Guerrero Jair **NRC:** 29583

Fecha: 11/01/2026

Tema: Ejercicios de Cap. 4

Ejercicio 1

Se desea hacer un programa que almacene datos sobre un determinado colectivo de personas, solicitando el nombre, la edad y la ciudad de residencia. Realice los siguientes apartados:

1. Diseñe una estructura de datos para almacenar la información relativa a cada individuo. Añada un campo adicional que almacene el número de veces que se repite el nombre en el colectivo.

Se ha introducido la definición de un nuevo tipo con la sentencia `typedef`.

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);
```

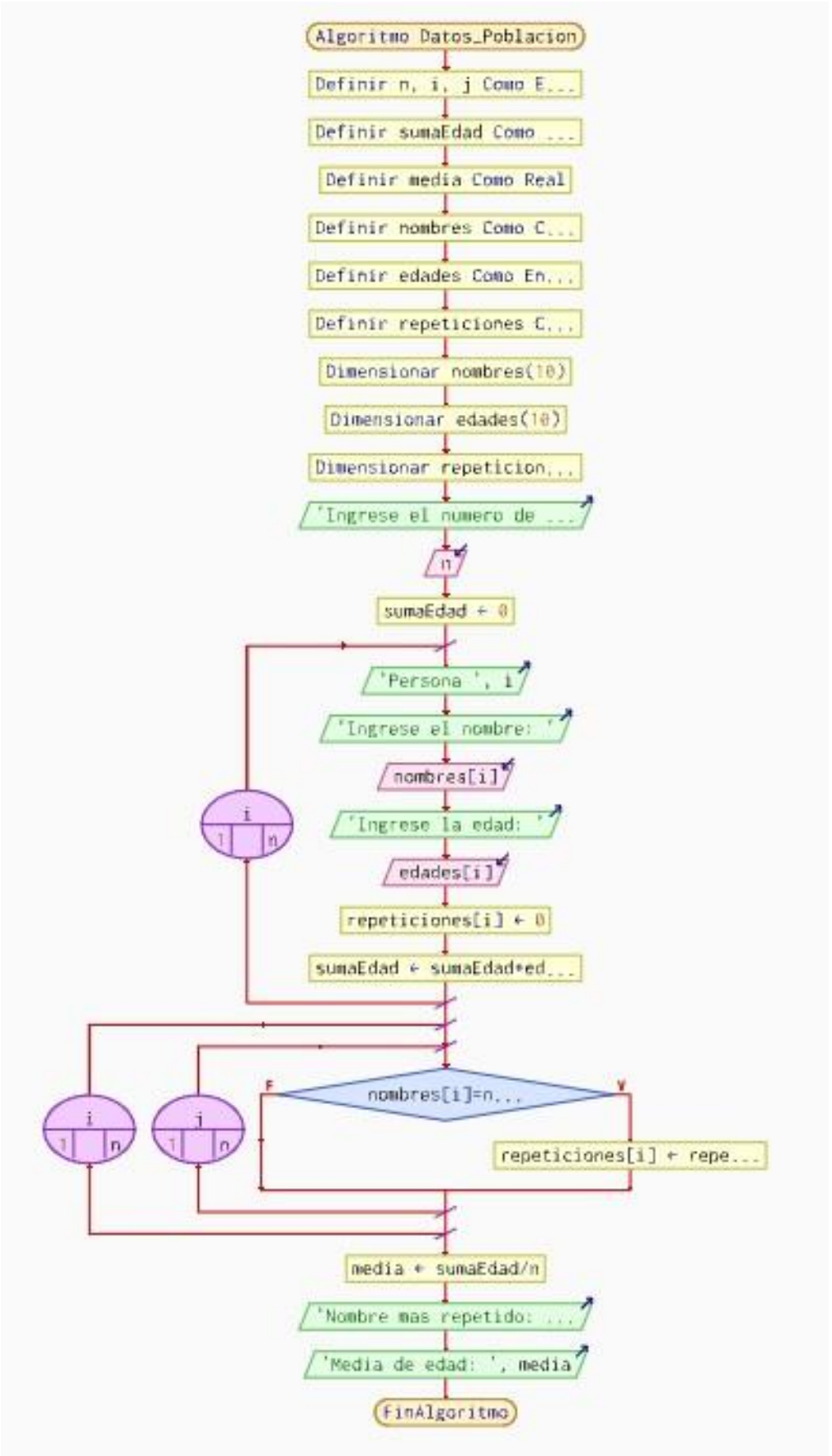
La función retorna un número entero mayor, igual, o menor que cero, apropiadamente según la cadena apuntada por `s1` es mayor, igual, o menor que la cadena `s2`.

2.Tabla de objetos

Objeto	Nombre	Valor	Tipo
Contador del ciclo	i	Variable	Entero

Contador del ciclo secundario	j	Variable	Entero
Número de personas (1–10)	n	Variable	Entero
Máximo número de repeticiones	máx	Variable	Entero
Edad media	media	Variable	Real
Edades de las personas	nombre	Arreglo	Cadena
Ciudad de nacimiento	edad	Arreglo	Entero
Condición de recursividad	ciudad	Arreglo	Cadena
Repeticiones del nombre	rep	Arreglo	Entero
Límite máximo de personas	10	Constante	Entero
Límite mínimo de personas	1	Constante	Entero

3.Diagrama de flujo en PSEINT

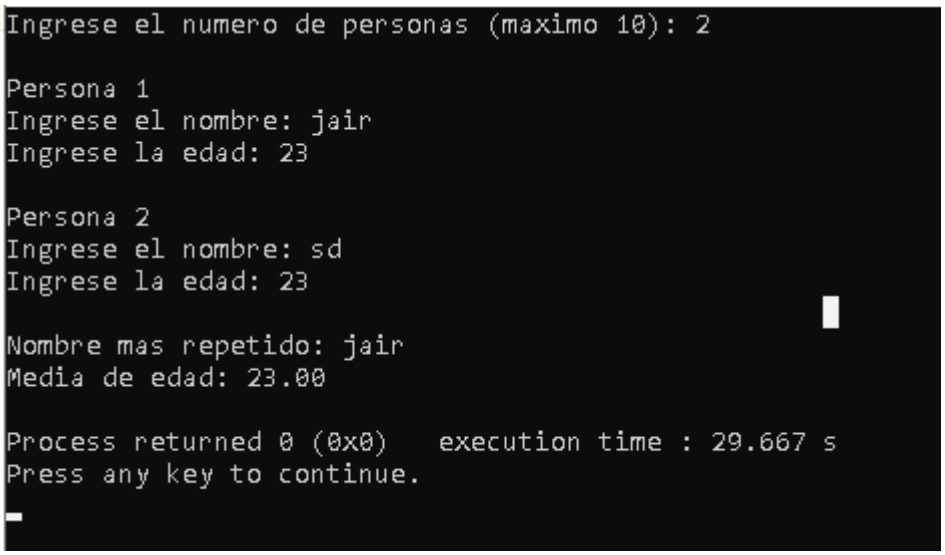


Código en CODEBLOCKS

```
#include <stdio.h>

#include <string.h>

#define MAX 10

#define
MAX_NOMBRE 50

int main() {

   int n, i, j;

   int edades[MAX];

   int
repeticiones[MAX];

   int sumaEdad = 0;

   float media;

   char
nombres[MAX][MAX_
NOMBRE];

    printf("Ingrese el
numero de personas
(maximo 10): ");

   scanf("%d", &n);

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

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

       printf("Ingrese el
nombre: ");

       scanf("%s",
nombres[i]);

       printf("Ingrese la
edad: ");

       scanf("%d",
&edades[i]);
```

```
        repeticiones[i] =
0;

        sumaEdad +=
edades[i];

    }

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

        for (j = 0; j < n;
j++) {

            if
(strncmp(nombres[i],
nombres[j],
MAX_NOMBRE) ==
0) {


repeticiones[i]++;

            }

        }

    }

    media =
(float)sumaEdad / n;



    printf("\nNombre
mas repetido: %s\n",
nombres[0]);

    printf("Media de
edad: %.2f\n",
media);



    return 0;

}
```

5.- Pantallazos de la ejecución



- **Conclusiones**

- **Gestión eficiente de datos: Se logró diseñar una herramienta funcional para el registro de información personal, cumpliendo con éxito el cálculo del promedio de edad de los usuarios ingresados.**

- **Compatibilidad de lógica: Se confirmó la validez del diseño algorítmico, demostrando que la lógica estructurada inicialmente en PSeInt es plenamente compatible y funcional al ser trasladada al lenguaje C.**

- **Recomendaciones**

- **Implementación de filtros de seguridad: Se sugiere añadir mecanismos de validación en la entrada de datos para asegurar que el programa sea robusto ante ingresos accidentales de valores no válidos.**

- **Optimización del código: Es recomendable evolucionar hacia el uso de estructuras de datos (struct), lo que permitiría un manejo de la información más organizado, escalable y fácil de mantener.**

-

Ejercicio 2 (Reconocimiento de carácteres)

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 8x8 celdas (véase figura 4.1). Se dispone además de una tabla de estructuras de tipo **"struc 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.

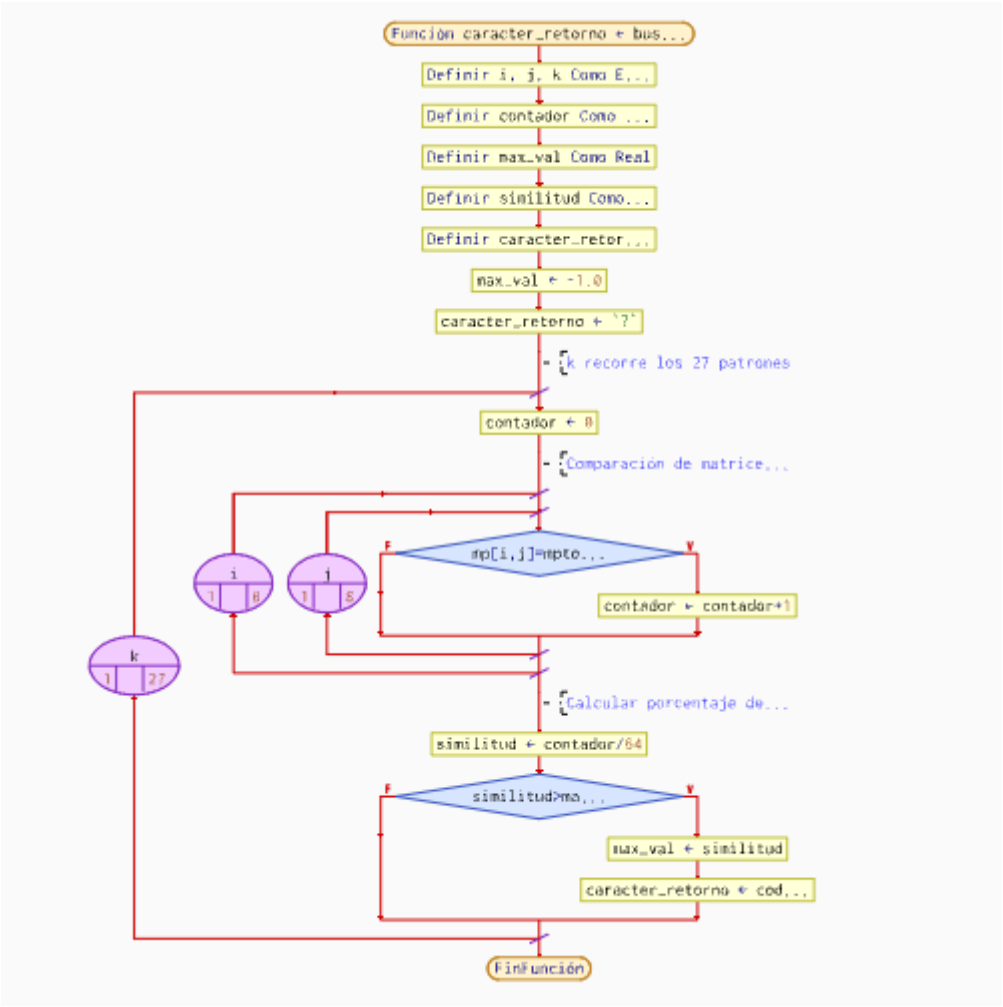
Struc letras
{
Char cod_ASCII; /* Letra a la que corresponde la matriz de puntos*/

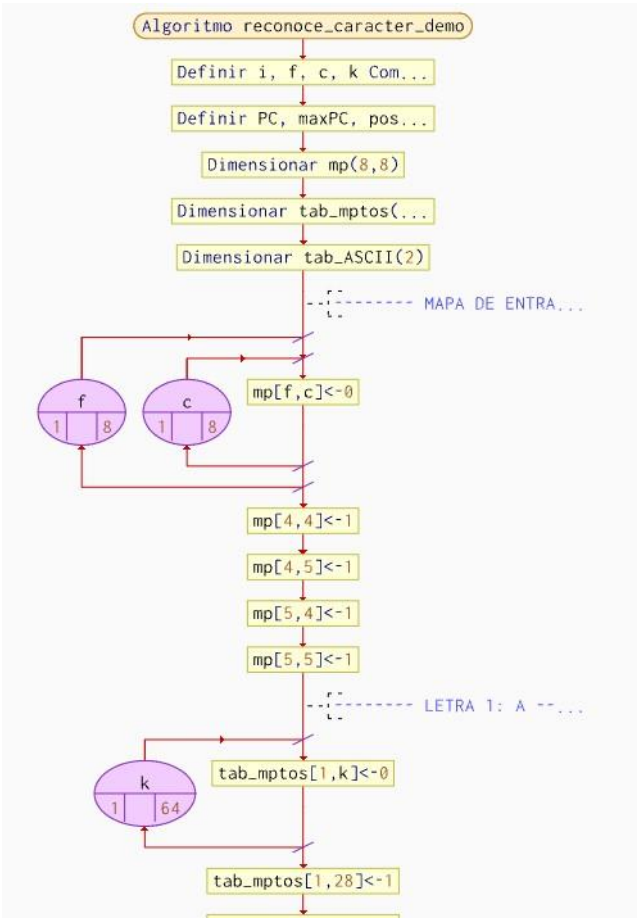
Int mptosd[8][8]; /*Matriz de puntos del carácter*/
};
Struct letras tab_let[27];

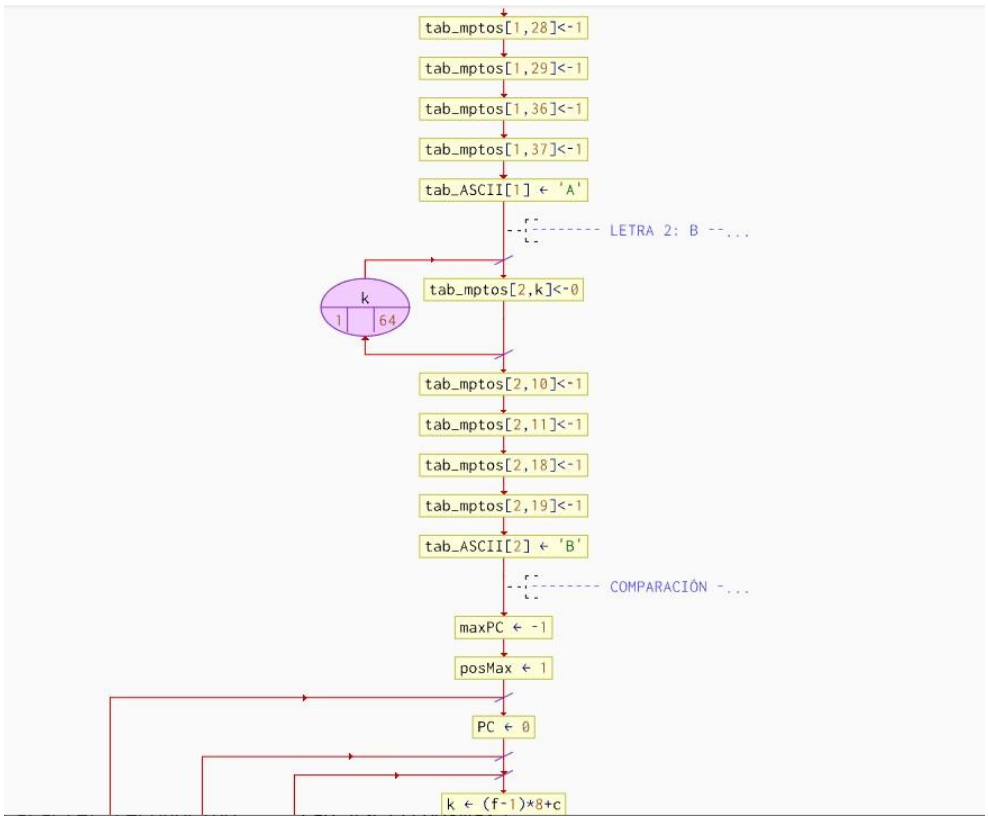
2.- Tabla de Objetos

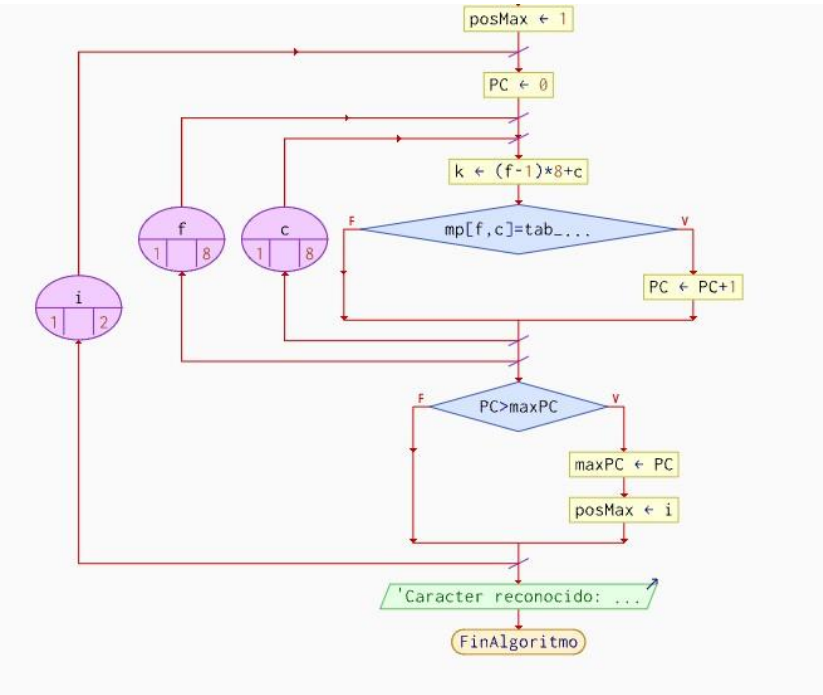
Objeto	Nombre	Valor	Tipo
Contador del ciclo principal	i	Variable	Entero
Contador del ciclo secundario	j	Variable	Entero
Número de personas (1–10)	n	Variable	Entero
Máximo número de repeticiones	max	Variable	Entero
Edad media	media	Variable	Real
Nombres de las personas	nombre	Arreglo	Cadena
Edades de las personas	edad	Arreglo	Entero

3.- Diagrama de flujo




Algoritmo reconoce_caracter_demo
Definir i, f, c, k Com...
Definir PC, maxPC, pos...
Dimensionar mp(8,8)
Dimensionar tab_mptos(...
Dimensionar tab_ASCII(2)
-------- MAPA DE ENTRA...
f
1
8
c
1
8
mp[f,c]<-0
mp[4,4]<-1
mp[4,5]<-1
mp[5,4]<-1
mp[5,5]<-1
-------- LETRA 1: A --...
k
1
64
tab_mptos[1,k]<-0
tab_mptos[1,28]<-1


tab_mptos[1,28]<-1
tab_mptos[1,29]<-1
tab_mptos[1,36]<-1
tab_mptos[1,37]<-1
tab_ASCII[1] ← 'A'
-------- LETRA 2: B --...
k
1
64
tab_mptos[2,k]<-0
tab_mptos[2,10]<-1
tab_mptos[2,11]<-1
tab_mptos[2,18]<-1
tab_mptos[2,19]<-1
tab_ASCII[2] ← 'B'
-------- COMPARACIÓN -...
maxPC ← -1
posMax ← 1
PC ← 0
k ← (f-1)*8+c



4.- Código en CODEBLOCKS

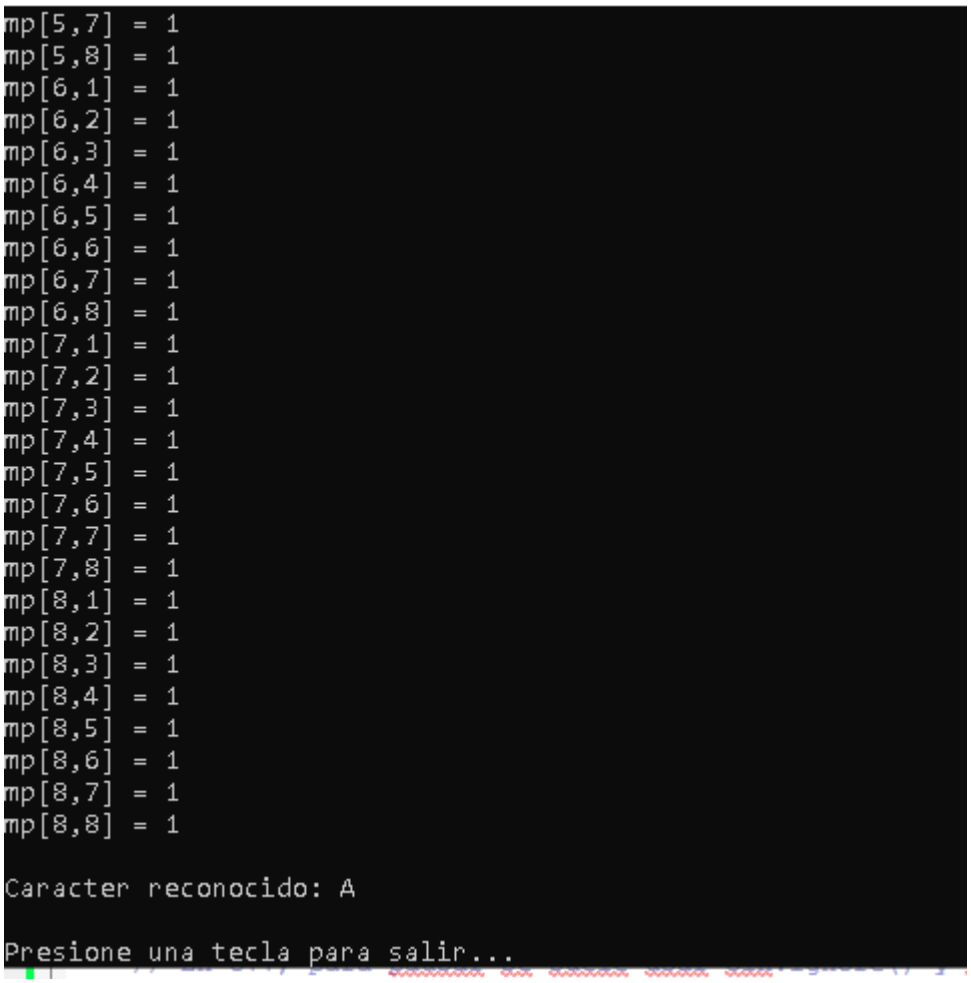
```
#include <iostream>

#include <string>


using namespace std;

char busca_caracter(int mp[8][8], int mptosd[27][8][8], string cod_ascii[27]) {

    int i, j, k;

    int contador;

    float max_val = -1.0;

    string caracter_encontrado = "?";


    for (k = 0; k < 27; k++) {

        contador = 0;

        for (i = 0; i < 8; i++) {

            for (j = 0; j < 8; j++) {

                if (mp[i][j] == mptosd[k][i][j]) {
```

```
                contador++;
            }
        }
    }
    float similitud = (float)contador / 64.0;

    if (similitud > max_val) {
        max_val = similitud;
        caracter_encontrado = cod_ascii[k];
    }
  }
  return caracter_encontrado[0];
}

int main() {
  int mp[8][8];
  int mptosd[27][8][8];
  string cod_ascii[27];
  int i, j, k;
  char letra;
  for (k = 0; k < 27; k++) {
    cod_ascii[k] = "?";
    for (i = 0; i < 8; i++) {
      for (j = 0; j < 8; j++) {
        mptosd[k][i][j] = 0;
      }
```

```
    }
}
cod_ascii[0] = "A";
mptosd[0][0][2] = 1; mptosd[0][0][3] = 1; mptosd[0][0][4] = 1; mptosd[0][0][5] = 1;
mptosd[0][1][1] = 1; mptosd[0][1][6] = 1;
mptosd[0][2][1] = 1; mptosd[0][2][6] = 1;
mptosd[0][3][1] = 1; mptosd[0][3][2] = 1; mptosd[0][3][3] = 1; mptosd[0][3][4] = 1;
mptosd[0][3][5] = 1; mptosd[0][3][6] = 1;
mptosd[0][4][1] = 1; mptosd[0][4][6] = 1;
mptosd[0][5][1] = 1; mptosd[0][5][6] = 1;
mptosd[0][6][1] = 1; mptosd[0][6][6] = 1;

cout << "RECONOCIMIENTO DE CARACTERES (8x8)" << endl;
cout << "Ingrese el mapa (0 o 1):" << endl;

for (i = 0; i < 8; i++) {
    for (j = 0; j < 8; j++) {
        cout << "mp[" << i + 1 << "," << j + 1 << "] = ";
        cin >> mp[i][j];
    }
}
letra = busca_caracter(mp, mptosd, cod_ascii);

cout << "\nCaracter reconocido: " << letra << endl;
cout << "\nPresione una tecla para salir...";
```

```
    cin.ignore();

    cin.get();



    return 0;

}
```

5.- Pantallazos de la ejecución



6.- Conclusiones y recomendaciones

Conclusiones

- **Eficacia del reconocimiento:** El sistema demuestra ser capaz de identificar caracteres mediante un análisis comparativo de matrices de bits, validando el uso de mapas de puntos como método de detección.

- **Organización de la información:** El empleo de estructuras de datos multidimensionales facilita una gestión ordenada y lógica de los patrones de referencia y las entradas del usuario.

Recomendaciones

- **Optimización de patrones:** Se sugiere refinar y ampliar la resolución de los mapas de caracteres para incrementar el porcentaje de coincidencia y reducir el margen de error.

- **Escalabilidad del sistema:** Es recomendable integrar el algoritmo con dispositivos de entrada de datos reales (como sensores o archivos de imagen) para potenciar su utilidad en entornos prácticos.

Ejercicio 3 (Polígono)

Un polígono es una figura geométrica cerrada delimitada por segmentos rectos (aristas). En este ejercicio se usa la estructura tipo para almacenar la información de un polígono, donde en vez de almacena sus aristas se almacenan sus vértices, como se describe a continuación:

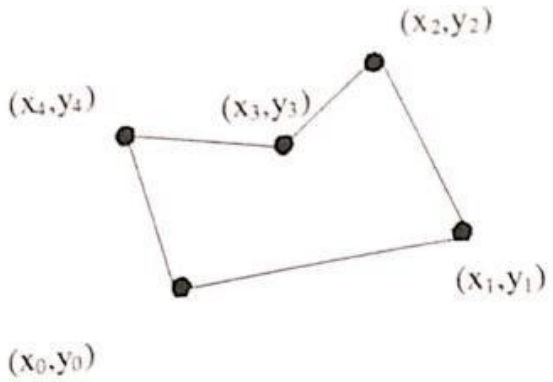


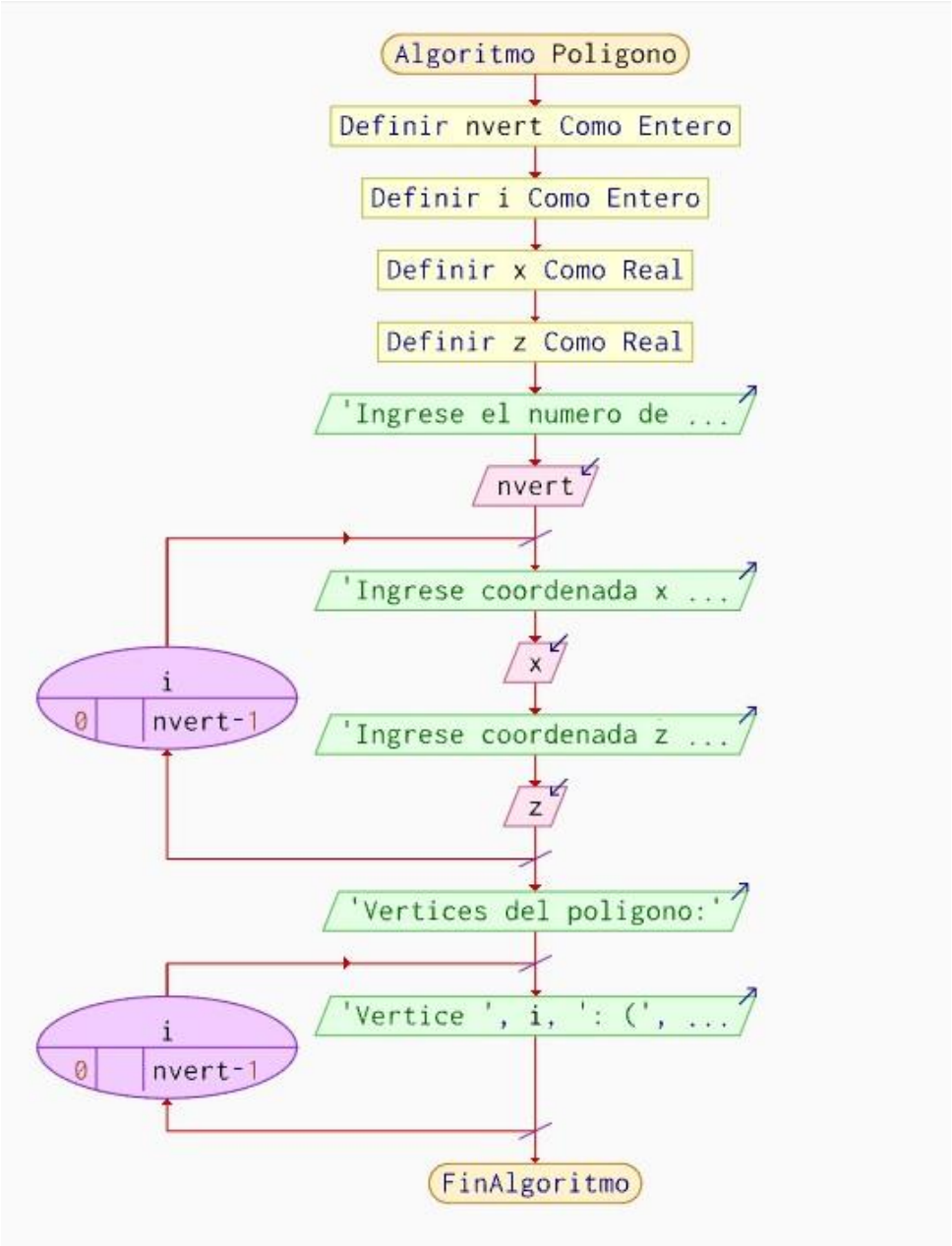
Figura 4.2: Polígono

2.- Tabla de Objetos

Objeto	Nombre	Valor	Tipo
Número de vértices	nvert	Variable	Entero
Contador de ciclos	i	Variable	Entero
acumulador del cálculo del área	areaTotal	Variable	Real
coordenadas X de los vértices	p	Arreglo	Real

coordenadas Y de los vértices	q	Arreglo	Real

3.- Diagrama de flujo

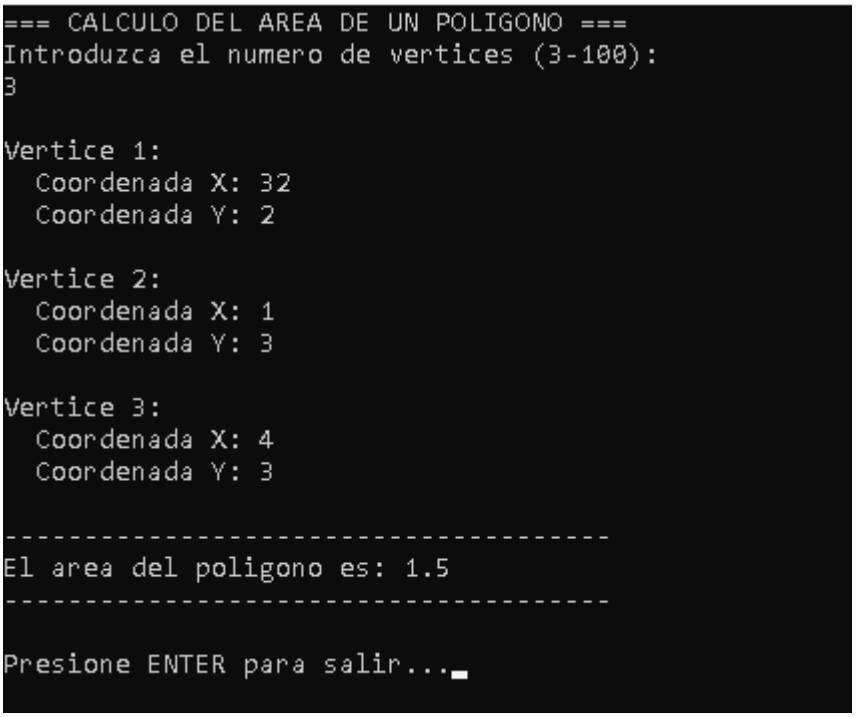


4.- Código en CODEBLOCKS

```
#include <iostream>

#include <vector>

#include <cmath> //
Para la función abs()


using namespace std;


int main() {

   int nvert;

   double areaTotal = 0;

   cout << "===
CALCULO DEL AREA DE
UN POLIGONO ===" <<
endl;

   do {

      cout << "Introduzca
el numero de vertices
(3-100): ";

      cin >> nvert;


      if (nvert < 3 ||
nvert > 100) {

         cout << "Error:
un poligono debe tener
entre 3 y 100 vertices."
<< endl;

      }

   } while (nvert < 3 ||
nvert > 100);

   vector<double>
p(nvert);

   vector<double>
q(nvert);

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

      cout << "\nVertice "
<< i + 1 << ":" << endl;

      cout << "
Coordenada X: ";
```

```
        cin >> p[i];
        cout << "
Coordenada Y: ";
        cin >> q[i];
    }
    for (int i = 0; i < nvert
- 1; i++) {
        areaTotal += (p[i] *
q[i + 1] - p[i + 1] * q[i]);
    }
    areaTotal += (p[nvert
- 1] * q[0] - p[0] *
q[nvert - 1]);


    areaTotal =
abs(areaTotal) / 2.0;


    cout << "\n----------
----------" <<
endl;
    cout << "El area del
poligono es: " <<
areaTotal << endl;
    cout << "----------
----------" <<
endl;


    cout << "\nPresione
ENTER para salir...";
    cin.ignore();
    cin.get();


    return 0;
}
```

5.- Pantallazos de la ejecución



6.- Conclusiones y recomendaciones

Conclusiones

1. El uso de estructuras permite representar correctamente un polígono mediante el almacenamiento de sus vértices.
2. La implementación en C facilita el manejo organizado de datos geométricos usando arreglos y ciclos.

Recomendaciones

1. Validar que el número de vértices no exceda el tamaño del arreglo definido.
2. Mantener el código simple y bien comentado para facilitar su comprensión y mantenimiento.

Ejercicio 4 (Game Over)

Se pretende hacer un pequeño juego de palabras por ordenador en lenguaje C que muestra en pantalla los caracteres que apareen en la figura 4.3 de fin de juego.

Considere la siguiente estructura:

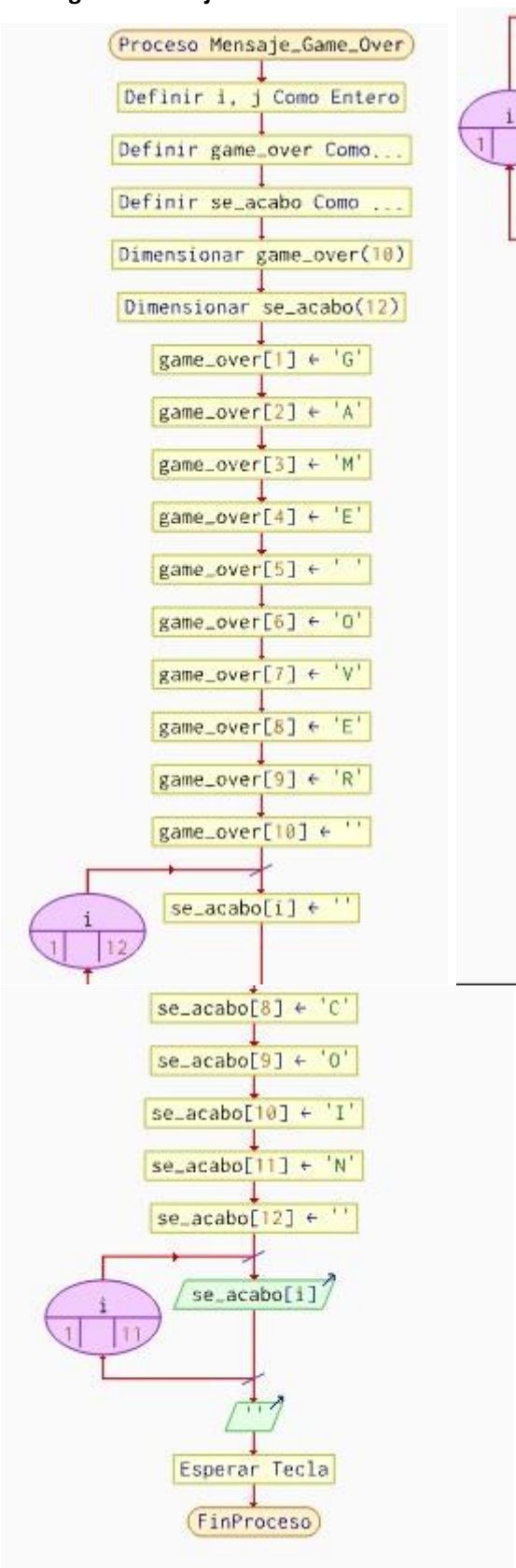
#define N 10
#define M 12
Struct mensaje
{

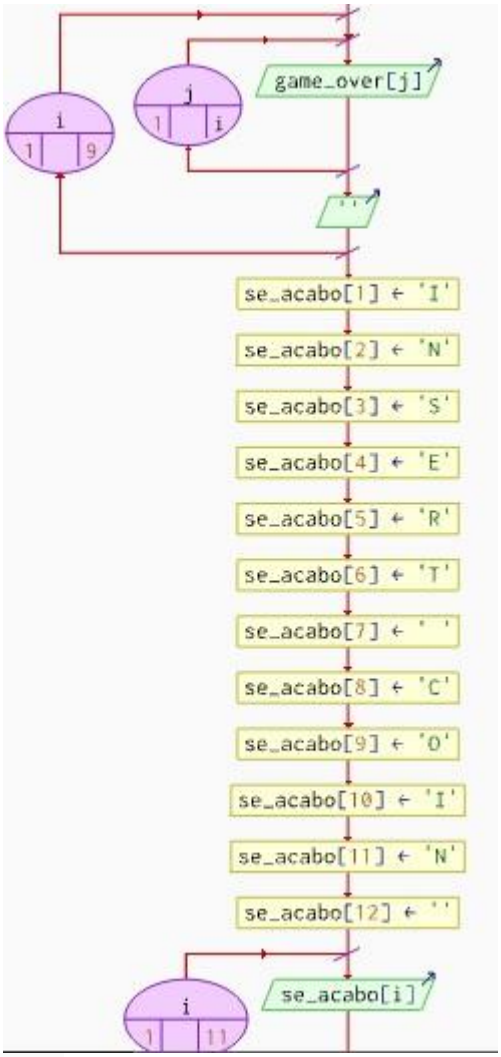
Char game_over[N];
Char se_acabo[M];
};
Typedef struct mensaje men;

2.- Tabla de Objetos

Objeto	Nombre	Valor
Contador1	i	Variable
Contador2	j	Variable
Mensaje principal	Game_over	Variable
Mensaje final	Se_acabo	Variable
Tamaño mensaje G. O	10	Constante
Tamaño mensaje I. C	12	Constante

3.- Diagrama de flujo



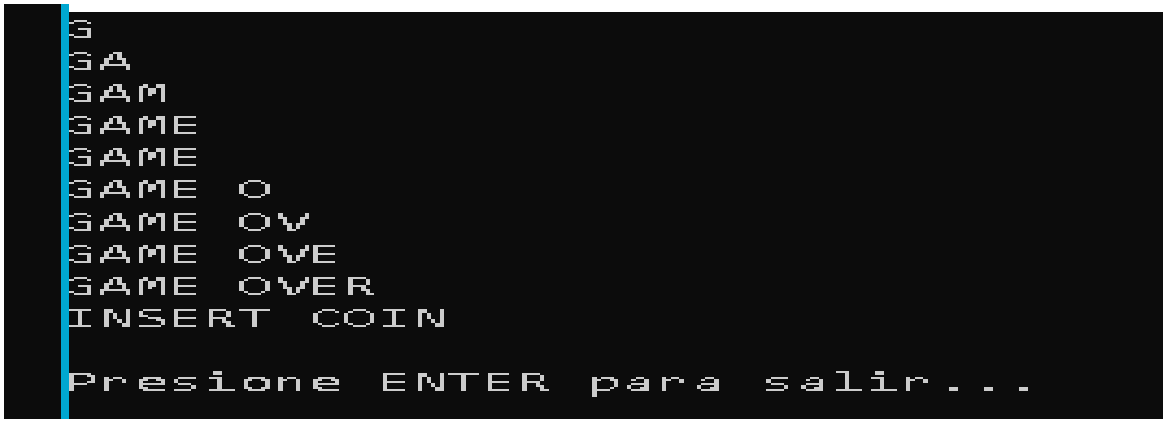


4.- Código en CODEBLOCKS

```
#include <iostream>
#include <string>

using namespace std;

int main() {

   int i, j;

   string game_over[10];
   string se_acabo[12];

   game_over[0] = "G";
   game_over[1] = "A";
   game_over[2] = "M";
   game_over[3] = "E";
   game_over[4] = " ";
   game_over[5] = "O";
   game_over[6] = "V";
   game_over[7] = "E";
   game_over[8] = "R";
   game_over[9] = "";

   for (i = 0; i < 12; i++) {
      se_acabo[i] = "";
   }
   for (i = 0; i < 9; i++) {
      for (j = 0; j <= i; j++)
{
         cout <<
game_over[j];
      }
      cout << endl;
   }
   se_acabo[0] = "I";
```

```
   se_acabo[1] = "N";
   se_acabo[2] = "S";
   se_acabo[3] = "E";
   se_acabo[4] = "R";
   se_acabo[5] = "T";
   se_acabo[6] = " ";
   se_acabo[7] = "C";
   se_acabo[8] = "O";
   se_acabo[9] = "I";
   se_acabo[10] = "N";
   se_acabo[11] = "";

   for (i = 0; i < 11; i++) {
      cout <<
se_acabo[i];
   }
   cout << endl;
   cout << "\nPresione
ENTER para salir...";
   cin.get();

   return 0;
}
```

5.- Pantallazos de la ejecución



6.- Conclusiones y recomendaciones

Conclusiones

- **Animación lógica y secuencial:** Mediante la implementación de ciclos anidados, se logró una presentación dinámica y progresiva de la información, emulando la estética clásica de las interfaces de videojuegos (*retro gaming*).
- **Gestión eficiente de cadenas:** El uso de arreglos estructurados en C++ demuestra ser una técnica eficaz para administrar y manipular secuencias de caracteres de forma ordenada y accesible.

Recomendaciones

- **Seguridad en la gestión de memoria:** Es fundamental supervisar rigurosamente la dimensión de los arreglos y sus índices para prevenir desbordamientos (*buffer overflow*) o accesos a direcciones de memoria inválidas.
- **Depuración y traza del código:** Se aconseja realizar pruebas de escritorio o ejecuciones controladas línea por línea para asegurar que la representación visual en consola sea exacta y libre de errores lógicos.