ESCUELA POLITÉCNICA SUPERIOR

INFORMÁTICA – CURSO 2024-25

PRÁCTICA 6: ESTRUCTURAS DE DATOS

HASTA AHORA...

En prácticas anteriores se ha aprendido:

- La estructura principal de un programa en C: la función main y las librerías.
- Variables y Constantes: con sus respectivos tipos, usos y funcionamiento.
- Funciones printf/scanf: muestra y obtención de información por pantalla/teclado.
- Condicionales if/else: ejecución de un programa en árbol.
- Programación estructurada en funciones: división en subproblemas y reusabilidad.
- Programación iterativa utilizando las instrucciones while y for.
- Almacenamiento de información a gran escala: vectores.
- Caso especial de vectores: cadenas de caracteres.

OBJETIVOS

Hasta ahora, las variables de distinto tipo (char, int, etc.) eran tratadas de forma independiente, aunque la información que contienen esté relacionada entre sí. En esta sesión, se expondrá cómo un conjunto de variables de distinto tipo pueden agruparse dentro de un nuevo tipo de datos más complejo (estructuras de datos) permitiendo así una mayor organización de la información que almacena el programa.

En esta práctica, se persigue que el alumno aprenda a organizar la información de la realidad que modela de una forma más estructurada mediante estos nuevos tipos de datos compuestos. Para ello el alumno deberá aprender a determinar qué variables deben formar parte de la estructura de datos, cómo declararla y cómo acceder a ella para leer y/o escribir la información que contiene.

PRERREQUISITOS

Para un mejor aprovechamiento, ANTES DE LA SESIÓN DE PRÁCTICA el alumno deberá:

- Leer de forma comprensiva el contenido teórico del apartado ‘Estructura de datos’ prestando especial atención a los 2 ejemplos que se exponen (trabajaremos con ellos).
- Trabajar los 3 primeros ejercicios de este documento y el ejercicio 7.

Como material de apoyo se recomienda la lectura de:

- Apartado 6.3 del documento ‘Aprenda lenguaje ANSI C como si estuviera en primero’
- Apartado 4.8.1 del libro ‘Fundamentos de Informática para Ingeniería Industrial’ (tema 4)

Ambos recursos se encuentran disponibles en la plataforma de enseñanza virtual.

ESTRUCTURA DE DATOS

Descripción

Las estructuras (*struct* en inglés) son tipos compuestos de datos. Es decir, son tipos de datos constituidos por un conjunto de otros tipos de datos (no necesariamente tipos simples) con el fin de aglutinar en un único tipo toda la información del objeto de la realidad que se quiere gestionar. Cada uno de los tipos de datos que forman parte de una estructura se denomina *campo*. De esta forma, cuando se declara una estructura de datos, cada campo que posee la estructura actúa como si fuese una variable más del programa pero que sólo es accesible a través de la estructura que la contiene (podría decirse que una variable de tipo estructura contiene a su vez otras variables "internas" denominadas *campos*).

Declaración de una estructura de datos: las estructuras de datos se definen al principio del programa (generalmente antes de las variables globales) y su sintaxis es la siguiente:

```
struct <nombre_estructura>
{
     <tipo1> <nombre_campo1>;
     <tipo2> <nombre_campo2>;
          ...
     <tipoN> <nombre_campoN>;
};
```

Declaración de variables del tipo de la estructura de datos: Una vez definida la estructura de datos, la declaración de variables del nuevo tipo de datos se realiza de forma similar a cualquier tipo de datos simple pero añadiendo al principio la palabra reservada `struct`. Su sintaxis es la siguiente:

```
struct <nombre_estructura> <Lista_Variables>;
```

Acceso a los campos que forman la estructura: para acceder a un campo de una variable de tipo estructura se utiliza el operador punto '.' de la siguiente forma:

```
<nombre_variable>.<nombre_campo1>
```

Ejemplo 1:

Para ejemplificar lo anterior, suponga un programa para el diseño de planos en el que resulta imprescindible guardar información sobre un punto del plano.

Para trabajar con puntos de plano se podría definir una estructura que guarde las dos componentes 'x' e 'y' que forman un punto en el espacio 2D:

```
struct Punto
{
     float x;
     float y;
};
```

Declarar variables del tipo `Punto`, por ejemplo, las variables `Origen` y `Destino`, sería:

```
struct Punto Origen;
struct Punto Destino;
```

Puede verse un ejemplo simple de cómo acceder a los campos que forman la estructura durante la inicialización de variables:

```
main()
{
     struct Punto p;

     p.x = 0;
     p.y = 10.5;
}
```

Una estructura puede contener campos de cualquier tipo, no sólo tipos simples, por lo que podemos definir campos cuyo tipo sean a su vez otra estructura.

Como ejemplo, suponga que el programa de diseño de planos requiere trabajar con triángulos, puesto que un triángulo puede estar definido por 3 puntos (denominados vértices), podría basarse en la estructura *Punto* para declarar una nueva estructura denominada *Triangulo* que contenga los 3 vértices que definen a un triángulo:

```
struct Triangulo
{
     struct Punto vertice1;
     struct Punto vertice2;
     struct Punto vertice3;
};
```

Por tanto, *Triangulo* es una estructura que contiene campos de la estructura *Punto*. Un ejemplo de acceso a estructuras de estructuras sería el siguiente: mostrar por pantalla las coordenadas del primer vértice:

```
int main()
{
     struct Triangulo tglo;
     printf("(%f, %f) ",tglo.vertice1.x, tglo.vertice1.y);
}
```

Ejemplo 2: Diseño de un programa para la gestión de las calificaciones de una asignatura.

Declaración del tipo de datos: puesto que el propósito del programa es gestionar las calificaciones, podría definirse en primer lugar una estructura de datos que mantenga la información referente a un alumno:

```
struct Alumno
{
     char nombre[50];
     char apellidos[50];
     char DNI[10];
     float calTeoria;
     float calPractica;
};
```

Declaración de la variable: a la hora de trabajar con los alumnos, nos interesa mantenerlos a todos juntos bajo una variable indivisible. Así pues, esa variable podría ser un vector (véase la práctica 5) de alumnos (en nuestro caso, la clase tiene el número fijo de 20 alumnos):

```
struct Alumno clase[20];
```

Acceso a la estructura: como primer ejemplo, se muestra una función de cómo añadir la calificación a un alumno concreto de la clase; del cual se proporciona su dni como una cadena de caracteres (para contener el número y la letra a la vez) y la calificación a ponerle:

```
void ponerNota(char dni[], float notaTeo, float notaPra)
{
   int i, enc=0;
   for(i=0; i<20 && !enc; i++)
   {
        if(strcmp(dni, clase[i].DNI)==0)
        {
             enc=1;
             clase[i].calTeoria=notaTeo;
             clase[i].calPractica=notaPra;
        }
   }
}
```

EJERCICIOS:

Ejercicio 1: Con la definición de Punto utilizada en el ejemplo 1, realice lo siguiente:

- **a)** Cree dos variables de dicho tipo en la función principal.
- **b)** Asígneles dos valores para x e y del primer punto, introduciendo los mismos por teclado.
- **c)** Realice una copia de los valores del primer punto a los del segundo (copiando x y, posteriormente y).
- **d)** Compruebe que se ha realizado correctamente la copia, imprimiendo las coordenadas del segundo punto por pantalla.

Ejercicio 2: A partir de la estructura Punto (ejemplo 1), defina un programa que calcule la distancia entre dos puntos. Hágalo paso a paso:

- **a)** En primer lugar, cree dentro de la función principal dos variables de tipo punto.
- **b)** Introduzca los valores de las coordenadas de ambas variables desde teclado.
- **c)** Calcule la distancia entre dichos puntos, haciendo uso de la distancia euclídea entre ellos. Muestre el resultado por pantalla.
- **d)** Por último, encapsule dicha funcionalidad dentro de una función independiente, para ello tendrá que implementar una función. Ésta recibirá dos puntos, calculará la distancia entre ambos y la devolverá. En la función principal se hará la llamada a la misma, y la muestra del resultado. El prototipo de la función es:

```
float calculaDistancia (struct Punto p1, struct Punto p2);
```

Ejercicio 3: A partir de la estructura Triangulo (ejemplo 1), defina un programa que calcule el perímetro de un triángulo. Para ello, realice lo siguiente:

- **a)** Cree una variable del tipo triángulo en la función principal e inicialice los valores de todos los vértices.
- **b)** Calcule el perímetro del triángulo, haciendo uso de la función creada en el "Ejercicio 2". Para ello, deberá llamar a la función que calcula la distancia entre dos puntos para cada par de puntos del triángulo y sumar todas esas distancias. Muestre el resultado por pantalla.

e) Por último, encapsule dicha funcionalidad dentro de una función independiente. La función recibe por parámetros un triángulo y devuelve el perímetro del mismo (atendiendo al sumatorio de la distancia entre todos sus puntos). Incluya la llamada a la función dentro del main, pasándole el triángulo definido anteriormente. Finalmente, mostrará el resultado de la llamada a la función. El prototipo de la función es:

```
float calculaPerimetro (struct Triangulo t);
```

Ejercicio 4: Defina una nueva estructura denominada Polígono capaz de almacenar hasta 50 puntos. Para ello, recuerde que los campos de una estructura no tienen por qué ser de tipos básicos (Nota: podría ser un vector de puntos). Además, introduzca un segundo campo a su estructura, de tipo entero, para indicar cuántos puntos (de los 50) están siendo utilizados para una variable concreta. De esta forma, la estructura Polígono admitirá cualquier polígono desde tres puntos (menos de tres no forman un polígono) hasta 50.

Ejercicio 5: Implemente una función que devuelva el perímetro de un Polígono recibido por parámetros. No sería más que calcular la distancia entre cada par de puntos situados en posiciones adyacentes. Nota: tenga cuidado con el primer y último punto.

Ejercicio 6: A partir de la estructura Alumno, realice lo siguiente:

a) Cree un vector de alumnos y rellene el vector con datos ficticios. Hágalo de forma estática (sin meter los datos por teclado) para poder utilizarlo a continuación.

b) Muestre por pantalla las notas de teoría y prácticas de todos alumnos, así como la nota final calculada de la media aritmética ponderada (60% y 40% respectivamente). Deberá mostrar:

```
  Apellidos         Nombre            Teoría   Práctica   Final
-------------------------------------------------------------------
  Rodríguez López   Fulanito             10         9       9.6
```

c) Calcule y muestre por pantalla el porcentaje de alumnos aprobados (según la ponderación anterior).

d) Muestre la posición en el vector del alumno con mejor nota (utilizando la ponderación anterior).

e) Muestre la posición en el vector del alumno con peor nota (utilizando la ponderación anterior).

Ejercicio 7: La información de todos los empleados de la empresa DATASYSTEM está almacenada en una variable, que no es más que un vector de estructuras. En cada posición de dicho vector, se encuentra la información concerniente a un `empleado`. La información con la que se cuenta de cada empleado es: nombre (cadena de caracteres), sexo (cadena de caracteres) y sueldo (real). Realice lo siguiente:

a) Suponiendo que hay diez empleados en la empresa, cree todas las estructuras necesarias y/o variables globales que necesite.

b) Rellene los datos de los empleados.

c) Cree un programa que muestre por pantalla los datos del empleado con mayor y menor salario.