Tema 10:
Tipos de datos definidos por el usuario

Indice

1. Introducción

- En un lenguaje de programación es posible crear nuevos tipos de datos según la necesidad de un problema concreto
- Estos tipos pueden ser
  - Registro
  - Unión
  - Enumeración
- Problema donde es necesario usar registros: Se desea obtener un listado de los nombres de los alumnos, si es o no repetidor y su nota en el examen, ordenado por nota.

2. Registros

- Un registro es una agrupación de variables de igual o distinto tipo bajo un mismo nombre. Esto permite tener unida una serie de información relacionada
- Las variables que forman un registro se llaman **campos del registro.**

PSEUDOCODIGO	LENGUAJE C

REGISTRO Alumno
  nombre: CADENA[30]
  repetidor: CARÁCTER
  nota: REAL
FIN REGISTRO Alumno
	struct Alumno
	{
	  char nombre[30];
	  char repetidor;
	  float nota;
	};

Registros

- Importante: se está dando de alta un nuevo tipo de dato
- En Pseudocódigo este código se colocaría antes del programa en un apartado titulado TIPOS
- En C el código se colocaría después de las directivas y antes de los prototipos de las funciones
- Declarar una variable de tipo registro

PSEUCODIGO **LENGUAJE C**

alum: REGISTRO Alumno struct Alumno alum;

2.1. Elementos de un registro

- Para acceder a los campos de un registro se usa el operador . (punto)

PSEUDOCODIGO	**LENGUAJE C**
strcpy (alum.nombre, "PEPE") alum.repetidor ← 'S' alum.nota ←8	strcpy (alum.nombre, "PEPE"); alum.repetidor = 'S'; alum.nota = 8;

- A diferencia de vectores y matrices un registro puede copiarse directamente en otro (siempre que sean del mismo tipo)

```
struct Alumno r1, r2;
…
r1= r2;
```

2.2. Arrays de registros

- Normalmente un registro no se utiliza solo sino que se usa para almacenar información de varios elementos

PSEUDOCODIGO	LENGUAJE C

clase: ARRAY[TAM] DE REGISTRO Alumno
INICIO
PARA i DE 1 A TAM
ESCRIBIR "Nombre del alumno",i
gets(clase[i].nombre)
….
FIN PARA
	struct Alumno clase[TAM];
	
	for (i=0; i<TAM; I++)
	{
	printf("Nombre del alumno %d",i);
	gets(clase[i].nombre);
	fflush(stdin);
	}

2.3. Registros con arrays

- **Puede ocurrir que un campo de un registro sea un array**

```
REGISTRO Alumno2
        notas: ARRAY[NTRIM] DE REAL
        nombre: CADENA[TAM]
        repetidor: CARACTER
FIN Alumno2
```

- **Para acceder a los elementos**

```
REGISTRO Alumno2   r;
ESCRIBIR "Introduce nombre del alumno"
gets( r.nombre)
PARA i DE 1 A 3
     ESCRIBIR "Introduce nota", i, " del alumno", r.nombre
     LEER r.nota[i]
FIN PARA
```

2.4. Registros con registros

- Un registro puede incluir como campo otro registro

TIPOS

```
REGISTRO Direccion
   calle: CADENA[30]
    bloque: ENTERO
    piso: ENTERO
    letra: CARÁCTER
    ciudad: CADENA[30]
FIN Direccion

REGISTRO Cliente
  nombrecli: CADENA[30]
  dircli: REGISTRO Direccion
  edad: ENTERO
FIN Cliente
```

2.6. Paso de registros a funciones

- Por defecto el paso de registros a funciones es por copia (a menos que usemos punteros a registros para pasarlo por valor).
- Es posible que una función devuelva un registro completo.
- Ver ej1registros.cpp
- OJO: Un array de registros sí se pasa por referencia.

2.6. Punteros a registros

- Es posible declarar un puntero a tipo registro

PSEUDOCODIGO	LENGUAJE C
palumno: PUNTERO A REGISTRO Alumno	struct Alumno * palumno;

- Acceder a los campos de una estructura usando un puntero, puede usarse un nuevo operador

(*palumno).nota= 8;

es equivalente a

palumno -> nota= 8;

3. Enumeraciones

- Una enumeración es un tipo de dato q especifica todos los valores válidos que una variable de este tipo puede tomar
- A cada valor posible se le asigna un número a partir del 0.

PSEUCODIGO

ENUMERADO estadocivil { soltero, casado, viudo, separado}

LENGUAJE C

enum estadocivil {soltero, casado, viudo, separado};

4. Union

- Es una estructura de datos del lenguaje C que permite utilizar la misma zona de memoria para almacenar tipos de datos distintos en diferentes momentos.

```
union tipo_nota_nss {
                        int nota;
                        char nss[12]);
                        }
```

- En la union se reserva espacio para almacenar el tipo de dato más grande de los campos que tiene.

5. Renombrar tipos

- La sentencia typedef del lenguaje C nos permite renombrar tipos

Ejemplo:

```
typedef int entero;
typedef  struct {
                    float sueldo;
                    char nombre[TAM];
                    char dirección[TAM];
                } Empleado;
```

6. Combinación de estructuras estáticas

- Todas las estructuras estáticas pueden combinarse entre sí (vectores, tablas, registros, enumeraciones, uniones…)
- Es muy importante para resolver un problema saber escoger la estructura de datos adecuada y la organización de los datos dentro de la estructura.

7. Bibliotecas y archivos de cabecera

Archivos de cabecera

- Son archivos de extensión .h se utilizan para describir la interfaz del programa.
- En el se incluyen los #define, #include, tipos, y prototipos de las funciones.
- No es obligatorio para el funcionamiento del programa pero el estándar ANSI recomienda su uso

Bibliotecas o Librerías

- Es un conjunto de funciones enlazadas entre sí que podrán ser utilizadas posteriormente por otros programas
- En una librería no hay función principal (main) ni tampoco se genera un archivo ejecutable
- Cuando se compile y linke se generará un archivo .lib con el código objeto de las funciones.