Unidad 5
ESTRUCTURAS ESTATICAS II (Tablas y Registros)
Desarrollo de la unidad: 36 h

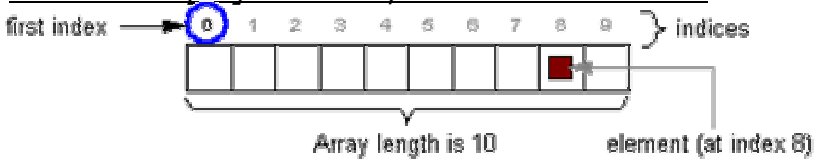
Prácticas :
Ejercicios :
Conceptos:
Implementación de tablas en C, Punteros y tablas, Cadenas de caracteres (string), librería string.h, Matrices de string, parámetros de la función main, registros y uniones, estructuras punteros a registros, estructuras complejas de datos, tablas de registros.

IMPLEMENTACIÓN EN C DE LAS TABLAS

Definición
<tipo base> nombre [tamaño];

Ejemplos:
```
short Tdatos[100];
float  cifras[3];
char mensaje[50];
char tablero[3][3];
```

NOTA MUY IMPORTANTE:
Las tablas en lenguaje C, comienzan con la posición 0, en una tabla definida como *int tabla[10]*, los valores posibles son desde el *tabla[0]* hasta *tabla[9],*
el valor *tabla[10]* no existe, está fuera de la tabla.



Entrada y Salida de Tablas
No podemos leer o mostrar una tabla entera (salvo las cadenas de caracteres que tienen un tratamiento especial) siempre elemento a elemento.

```
scanf("%d", &tabla[i])
printf(" El valor de la posición %d es igual a %d \n", i, tabla[i]);
```

Inicialización de tablas

```
int tabla[5] = {2,30,5,3,9};  Entre llaves con valores separados por comas.
int tabla[2][3]= { {3,5,9}, {10,12,93} };
char vocales [5] = {'a','e','i','o'.'u' };
```

Ejemplos de funciones sobre tablas:

```
void  Leer ( int Tabla[], int tamaño)
void Mostar ( int Tabla[], int tamaño )
int   EstaOrdenada ( int Tabla[], int tamaño )
void  Ordenar ( int tabla[], int tamaño );
```

```
void  CopiaEnOrden ( int tabla1[], int tabla2[], int tamano )
int  BuscarSecuencial   ( int tabla[], int tamaño, int valor )
int  BuscarBinaria   (int tabla[], int tamaño,  int valor)
```

Consultar los ejemplos de la página web.

Punteros y Tablas

Las tablas y los punteros en son tipos de datos muy relacionados. El nombre de una tabla es un puntero a la primera posición

```
int tabla[10];
int *pint;
```

Un puntero puede señalar a cualquier sitio de una tabla.
Significado de los operadores con punteros:

& - Dirección de una variable normal o una posición de una tabla
***** - Contenido de los señalado por un puntero

Ejemplo

```
 pint = & tabla[4]; // Significa que el puntero pint guarde la dirección de  la posición tabla[4]
*pint = 55              // Significa que donde este señalando el puntero se guarde el valor 55
```

La última instrucción sería equivalente a haber ejecutado: tabla[4] = 55,

Equivalencias

pint = tabla, *es lo mismo que :* pint = & tabla[0];
*pint == *tabla, *es lo mismo que,* pint[0] == tabla[0]
*(pint+1) = *(tabla +1*), es o mismo que* tabla[1] == pint[1]
pint++, *Incrementar el puntero es lo mismo que hacer que señale al siguiente posición*
pint == & tabla[1], si anterior mente pint señalaba a tabla[0]

Aritmética de punteros:

Sumar y restar valores a un puntero implica señala a nuevos elementos
Incrementar el uno el valor de un puntero (pin++) no siempre corresponde a incrementar en 1 el valor de la dirección, el valor exacto va a depender del tipo de dato señalado por el puntero.

Ejemplo;
```
char tablac[10];
char *pc;
short tablas[10];
short *ps;
pc = tablac;
ps = tablas;
pc++; // Si hacemos pc++, el puntero se incrementa en uno, pues un carácter ocupa un byte.
ps++ // Si hacemos ps++, el puntero se incremente en dos pues un entero corto ocupa dos bytes
```

Recorrido de una tabla mediante punteros:

Ejemplo: Calcular la suma de los elementos de una tabla;
```
int tabla[10];
int *pun;
int suma;

pun = tabla;  // Hago que el puntero señale a la primera posición de la tabla.
suma = 0;
for (i=0; i < 10; i++)
   {
    suma = suma + *pun; // Incremento la suma con el valor señalado por el puntero.
    pun++; // Incremento el puntero para que señale al siguiente elemento
   }
```

¡OJO! : El puntero o nombre de una tabla no debe ser modificado.

```
Int tabla[10];
Int *pint;
tabla = pint
tabla++
```

CADENAS DE CARACTERES

La tabla de caracteres es una estructura de datos muy utilizada en programación, por lo que recibe un tratamiento especial. La tabla de caracteres se manejan el nombre de strings o cadenas, pero no todas las tabla de caracteres son strings. Una tabla de caracteres se comportara como una cadena o string si contiene el carácter de terminación de cadena '\0', NULL

char nombre[100];

Entrada y Salida de cadenas:

scanf("***%s***",nombre);
gets(nombre); Almacena un '\0' para indicar fin de cadena
// Ojo en el scanf no hace falta el &
// pues nombre es ya un puntero.
prints("**%s**",nombre);
puts(nombre); Muestra hasta encontrar '\0'

Para leer cadenas es preferible utilizar la función **fgets** pues se indica el tamaño máximo a leer incluido el carácter '\0'.

```
char cadena[5];
puts("Introduzca un texto:")
fgets(cadena, 5 , stdin );
```

Almacena como máximo 4 letra + el '\0'.

Inicialización de cadenas:
char mensaje[10] = "Hola tio" ={ 'H','o','l','a',' ','t','i','o'.'\0','k'}

puts(mensaje); // Muestra **Hola tio**
mensaje[4] = '\0'; // Pongo la marca de final de cadena en la posición 4 de la tabla
puts(mensaje); // Muestra **Hola**

Ojo una tabla sin el carácter de terminación '\0' no es un string y no puede ser tratado como tal:
Ejemplo:
char vocales [5] = {'a','e','i','o','u'};
Si aplicamos cualquier función sobre esta tabla de caracteres el programa fallaría.
Ej,- puts(vocales);

Ejemplo de funciones sobre cadenas:

En la definición de los parámetros es equivalente utilizar: *char cadena[] o char *cadena*.
La misma función se puede implementar manejando la cadena:
Como una tabla *while (cadena[i] ¡= '\0') { i++}*
Como un puntero *while (*cadena ¡= '\0'){ cadena++}*
<<La implementación con punteros suele ser más rápida.>>

```
int  Cuentacadena ( char *cadena )
int  Copiar  ( char *destino, char *origen )
int  Concatenar ( char *cadena1, char *cadena2 )
int  CuentaVocales ( char *cadena  );
int  PonEspaciosAlFinal ( char * cadena, int nespacios);
void PonEspaciosAlPrincipio ( char *cadena, int nespacios)
void QuitarLetra( char *cadena, char letra)
void PonEco ( char *cadena, int veces)
void PonMayuscular ( char *cadena)
int ValorDecimal ( char *cadena)
```

Consultar ejemplos en la Página web.

LIBRERÍA STRING.H

Al ser muy habitual es uso de cadenas de caracteres o string, existe una librería especifica donde están definidas las funciones más habituales en el manejo de cadenas:

#include <string.h>

LISTA DE FUNCIONES MAS COMUNES DE string.h	
int **strlen** (char *cad)	Devuelve el tamaño de la cadena
char * **strcpy** (char *des, char *orig) char * **strncpy**(char *des, char *orig, int num)	Copias en destino la cadena origen Ídem pero sólo copia num caracteres
char * **strcat** (char *des, char *orig) char * **strcat** (char *des, char *orig, int num)	Concatena en la cadena destino el contenido de la cadena origen Ídem pero sólo concatena num caracteres
int **strcmp** (char *cad1, char *cad2) int **strcmp** (char *cad1, char *cad2, int num)	Compara cadenas: Devuelve: 0 Si cad1 = a cad2 >0, Si cad1 es mayor cad2 <0, Si cad1 es menos cad2 Ídem pero sólo compara los num primeros caracteres
char * **strchr** (char *cadena, char letra) char * **strrchr** (char *cadena, char letra)	Devuelve un puntero a la primera posición de la cadena donde aparece la letra indicada. Devuelve un puntero a la última posición de la cadena donde aparece la letra indicada.
char * **strstr** (char *cad1, char *buscada)	Devuelve un puntero que señala el lugar donde aparece en cad1, la cadena buscada.

Comparación de cadenas	**Asignación de cadenas:**

Para compara cadenas debemos siempre utiliza la función strcmp
char *cad1 =”Pepe”;
char *cad2 = “Pepe”;
 	
Forma incorrecta.
if (cad1 == cad1) // No significa si son iguales ambas cadenas, sino si ambos punteros señalan a la misma posición de memoria, algo que no ocurre. Pues ambos punteros señalan a posiciones de memoria diferentes aunque guarden en este caso el mismo valor.
 	
Forma correcta
if (strcmp(cad1,cad2) == 0)
	Generalmente no se deben asignar, pues perdemos la dirección de lo que esta señalando. En la asignación no se copian los contenidos, sino se copian las direcciones, los dos punteros señalan al mismo sitio.
	
	Ejemplo de uso **correcto:**
	char cad1[20] = “Buenos días”;
	char *p;
	p = cad1; // p Señala al contenido de cad1
	
	Ejemplo de uso **incorrecto.**
	char cad1[20] = “Buenos días”;
	char *p = “Buenas tardes”;
	p = cad1; // p Señala al contenido de cad1 y se pierde la dirección del cadena “Buenas Tardes”, deberíamos haber hecho:
	strcpy(cad2,cad1);

Tablas de cadenas:
Podemos utilizar dos formas una tabla de tablas o una tabla de punteros, siempre que no modifiquemos los valores del las cadenas el tratamiento será similar.

A) // Tabla con cuatro cadenas de 20 caracteres como máximo (incluido el '\0').
```
char  mensajesT [4][20] = { "Hola" ,"Buenos días", "Buenas tardes", "Buenas noches" };
```

B) // Tabla con cuatro punteros a cadenas de tamaño predefinido.
```
char  *mensajesP [4] = { "Hola", "Buenos días", "Buenas tardes", "Buenas noches" };
```

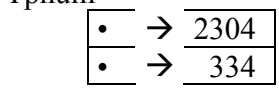
```
puts("TABLAS"); printf("%d\n",sizeof(mensajesT)); // Muestra 4 x 20 -> 80
for (i=0; i < 4 ; i++ )
        puts( mensajesT[i] );

puts("PUNTEROS"); printf("%d\n", sizeof(mensajesP)); // Muestra 4 x 4 -> 16
for (i=0; i < 4 ; i++ )
        puts( mensajesP[i] );
```

¡Ojo con la diferencia entre char mensajesT[5][20] y char *mensajesP[5]

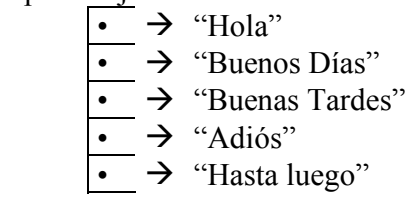
```
//Correcto, cabe la nueva cadena al tener menos de 20 caracteres
strcpy ( mensajeT[0],"Rehola");

// Incorrecto, no cabe la cadena pues sólo tenemos espacios para 4 caracteres + '\0'
strcpy (mensajeP[0],"Rehola");
```

Tablas de punteros y su significado:

```
int   *Tpnum  [2];  // Tabla con dos punteros a enteros
```




```
char  *Tpmensajes[5] ;  // Tabla con cinco punteros a cadenas
```



Introducción al formato <u>int main (int argc, char *argv[])</u>

argc : Guarda el número de argumentos incluyendo el nombre del programa.
char *argv[]; Puntero a una tabla de cadenas con cada uno de los argumentos introducidos en la línea de ordenes del sistema operativo.
int main(: La función devuelve un entero indicando el éxito o el fallo del programa, por convenio, el valor 0 indica que el programa se ha ejecutado con éxito otro valor indica algún tipo de error.

Si introducios la orden siguiente, los valores de los parámetros de la función main serian:
C\>miprograma hola 123

argc = 3
argv

•	→	"miprograma"
•	→	"hola"
•	→	"123"
•	→	NULL

OJO:
Todos los parámetros son cadenas de caracteres por lo tantos el valor decimal 123 debe ser obtenido mediante una función auxiliar:

```
int num;
num = atoi (argv[2]);
// También podemos utilizar:
sscanf(argv[2],"%d",&num);
```

Consulta ejemplos en la web:
1.- Ver los argumentos (mi echo)
2.- Ver los argumentos y mostrarlos al revés (ohce)
3.- Tabla de multiplicar con número pasado como parámetro (mul)
4.- Calculadora en modo línea.

TIPOS ABSTRACTOS DE DATOS (TAD)

Utilizando tablas y cadenas de caracteres podemos trabajar con tipos de datos que el C no implementa directamente. Estos nuevos tipos tendrán su representación interna y un conjunto de operaciones particulares.
Ejemplos:

- Enteros muy grandes Implementados con cadenas de caracteres:
  char num [100] ="120393948849590300032039400023";
  // En num podemos almacenar valores enteros hasta con 99 digitos.
  Operaciones suma, resta, multiplicación, división.

- Polinomios implementados con tablas de enteros: suma, resta, multiplicación, división.
  $5x^3 +3x +1$
  int polinomio[4] = {5,0,3,1};

- Conjuntos implementados con tablas de valores lógicos: (O esta incluido, 1 no está)

  Unión, Intersección, Pertenece, Conjunto Vacío, Inclusión.

Los Tipos Abstractos de Datos son el origen de la programación Orientada a Objeto implementada por lenguajes modernos como (C++, Java, Delphi)

MANEJO DE REGISTROS

Definición

Un conjunto finito de elementos de distintos tipos a los que se accede mediante un nombre de campo que lo identifica.

Representación

Nombre:	Pepe Rodríguez
Edad	22
Nota	8.50
Grupo	'C'

```
REGISTRO
   carácter  Nombre[40]
   entero  Edad
   real Altura
   carácter grupo
FIN REGISTRO   FICHA
FICHA  F1, F2
```

Definición de registros en C

```
struct Ficha{
   char nombre[40];
   int   edad;
   float nota;
   char  grupo;
 } ;

// Definición de variables tipo estructura ficha;
struct Ficha F1,F2;
```

```
typedef struct {
   char nombre[40];
   int   edad;
   float nota;
   char  grupo;
 } Ficha;

// Definición de variables tipo estructura ficha;
Ficha F1,F2;
```

Operaciones:
Casi ninguna global "sólo asignación":

```
F1 = F2;
```

Al igual que las tablas hay que operar con sus elementos individualmente: leer, escribir, modificar un campo.
Ejemplo:

```
scanf( "%s" , F1.nombre);
printf(" Primera ficha: %s,%d,%f \n",F1.nombre,F1.edad,F2.grupo);
F1.edad = 12;
if ( F2.grupo == 'D')
   {
   F2.nota = 0;
   }
```

Inicialización: Un valor para cada campo, separado por comas.

```
Ficha FichaPrimera = { "Ana María", 23, 8.5, 'B'};
```

Punteros a registros:

```
Ficha *pf1;
Ficha  F1,F2;

pf1 =  & F1;
F2 =   * pf1;
// Es equivalente a haber hecho:
F2 = FI;
```

¿Como acceder a los elementos de un registro señalado por un puntero?
(*pf1).edad = 22; o **pf1->edad = 22**;

En Resumen
registro**.**campo = valor;
pun_registro**->**campo = valor;

// Si queremos que una función modifique el contenido de un registro hay que pasarle un puntero a ese registro. Parámetro de salida → Paso por referencia.

```
void RellenarRegistro ( Ficha *pf )
{
printf("Nombre:");
gets(pf->nombre);
printf("Edad:");
scanf("%d",& pf->edad);
....
}
void MostrarRegistro ( Ficha F)
{
printf("Nombre: %s \n",F.nombre);
…
```

```
void main ( )
{
 Ficha F1;

  // Paso por referencia
 RellenarRegistro ( & F1 );

 // Paso por valor
 MostrarRegistro ( F1);

}
```

Tablas de registros:
Es habitual agrupar los registros en una tabla para manejarlos conjuntamente.

Ejemplo:

```
Ficha  TablaFicha[10]; // Una tabla que almacena 10 registros.
// Calculo de la media
media = 0;
for (i=0;i<10;i++)
   {
  media = media + TablaFichar[i].nota;
 }
media = media / 10;
```

Ejemplo a realizar: Gestión de un almacén de Registros
1.- Añadir un nuevo Registro
2.- Mostrar Registros
3.- Modificar Registros
3.- Borrar Registro
4.- Listado de Registros
5.- Salir

UNIONES

Similares a los registros, pero con todos sus campos sobrepuestos, en las mismas posiciones de memoria. Su uso no es muy común, aunque para algunas aplicaciones de bajo nivel como interfaz de comunicaciones con el hardware suele ser bastante utilizadas.

Ejemplos.

```
union tojunto {
   char letra;
   int número;
   char mensaje[20];
   int   datos[3];
}

union  uvariable {
       char texto[20];
       int  valores[4];
}
```

Si aplicamos la función **sizeof**() sobre una unión nos devolverá el tamaño del campo que ocupe más espacio en memoria. Si la variable es un registro, el valor devuelto corresponderá a la suma del tamaño de todos los campos.

Permiten manejar registros con estructura variable. Ejemplo:

```
// Registro con estructura variable:
struct  diversa {
      int clave;
      char tipo;
      union variable resto;
} R1, R2;
// Ejemplo de uso:
```

```
R1.clave = 10;
R1..resto.valores[2] = 0;
R2.clave = 6;
strcpy (R2.resto. texto, ” Marciano”);
```

REGISTROS CON TAMAÑO DE CAMPOS DE BITS

Son registros donde se indica al compilador cual va a ser el tamaño exacto, en número de bits, de alguno de los campos. Se utilizan como estructura de datos, normalmente, como interfaz con dispositivos hardware: tarjetas, controladores, puertos de comunicaciones, etc.

Ejemplo:

```
struct  registroIO {
   unsigned int  estado  :3;  // Tres bits
                         :5;  // Cinco bits sin uso
   unsigned int  dirIn    :4;
   unsigned int  dirOut :4;
   unsigned char dato;  // Variable normal char (1 byte, 8 bits)
}
// Se operan igual que campos nomales, aunque hay que tener presente su capacidad limitada.
registroIO.estado = 2;
registroIO.dirOut = 0x07;
registroIO.dato = ‘A’;
OperarTarjetaRM8923  ( &registroIO )
```

ESTRUCTURAS COMPLEJAS DE DATOS:

En programación es habitual tener que mezclas varios tipos de datos para representar problemas complejos. De tal forma, podemos tener: tablas de registros, registros que contienen tablas, Puntero a registros, tablas de punteros a tablas de registros, etc. Podemos combinar todas las estructuras de datos de la forma que queramos. A la hora de manejarlas hay que tener presente que cual es el tipo de dato que estamos usando: un puntero, una tabla, un campo, para no provocar errores de compilación o de ejecución.

Hay que tener especial atención con los avisos o *warning* que nos ofrece el compilador, pues el Lenguaje C permite una gran liberta al programador y puede aceptar operaciones que en la mayoría de los casos serian ilógicas, pero para no limitar al programador, se permiten.

Ejemplo: Posibles estructuras para tratar dibujos formados por líneas de colores.

```
define MAXPUNT  10 // Máximo de puntos que puede tener una línea.
typedef  enum {BLANCO, NEGRO, ROJO, VERDE, AMARILLO, MARRON, AZUL, GRIS} COLORES;

/* Estructura compuesta con dos campos simples */
struct SPunto {
            short x;
            short y;
              };

/* Estructura compuesta con campos simples y una tabla de MAXPUNT estructuras de puntos */
struct SLinea {
                COLORES color;
               short numpuntos;
               struct SPunto pun[MAXPUNT];
               };

/* Estructura de un dibujo que contiene una línea */
struct SDibujoA {
         char nombre[20]; // Nombre del dibujo
          struct SLinea linea;
          } da;

/* Estructura de un dibujo que contiene una tabla con 5 líneas */
struct SDibujoB {
         char nombre[20];
           struct SLinea Tlineas[5];
            }db;

/* Estructura de un dibujo que guarda una tabla con un número no definido
 de punteros a lineas */
struct SDibujoC {
         char nombre[20];
                struct SLinea *Tplineas[];
                }dc;

// Ejemplo de USO: Cambiar la coordenada x del tercer punto de la primera línea.
  da.linea.pun[2].x        = 140;
  db.Tlineas[0].pun[2].x   = 140;
  dc.TPlineas[0]->pun[2].x = 140;
```

Podemos definir igualmente
struct SDibujoB Plano[10];
Donde la variable Plano seria una tabla con 10 dibujos de tipo struct SDibujoB.