

Tema 4: Tipos Avanzados de Datos

Oscar Perpiñán Lamigueiro

1 Vectores

2 Cadenas de caracteres

3 Matrices

4 Estructuras

5 Enumeraciones

Vectores en C

Definición

Conjunto de valores numéricos del mismo tipo

Código

```
tipo identificador[dimensión];
```

tipo Tipo de los elementos del vector (int, float, etc.).

identificador Nombre del vector.

dimensión Número de elementos del vector (literal entero).

Ejemplos

```
// Declara un vector llamado miVector compuesto por
// tres elementos de tipo int.
int miVector[3];

// Declara un vector e inicializa todos sus elementos
int miVector[3] = {2, 23, 0};

// Declara un vector e inicializa el primer elemento
// (resto quedan a 0)
int miVector[3] = {2};

// Declara un vector sin dimensión.
// La dimensión queda determinada a partir
// del numero de elementos
int miVector[] = {2, 23, 24};
```

Elementos de un vector

- Se referencian con el nombre del vector seguido de un subíndice entre corchetes.
- El subíndice representa la posición del elemento dentro del vector.
- La **primera posición** del vector tiene el **subíndice 0**.

```
#include <stdio.h>

int main()
{
    int miVector[3];
    miVector[0] = 10;
    miVector[1] = 2 * miVector[0];
    miVector[2] = miVector[0] + miVector[1];

    printf("Posicion 0 = %d\n", miVector[0]);
    printf("Posicion 1 = %d\n", miVector[1]);
    printf("Posicion 2 = %d\n", miVector[2]);
}
```

Acceso a datos de un vectores

```
#include <stdio.h>

int main()
{
    float temp[3];

    printf("Indique tres valores reales.\n");
    scanf("%f %f %f",
          &temp[0], &temp[1], &temp[2]);

    printf("La media de estos valores es: %f\n",
          (temp[0] + temp[1] + temp[2])/3);

    return 0;
}
```

Acceso a datos de un vectores

```
#include <stdio.h>

int main()
{
    float temp[5] = {2.1, 4.9, 0.51, 4.3, 9.01};
    int i;
    // Es común el uso de bucles for para
    // recorrer un vector. Es importante
    // recordar que el primer elemento
    // tiene índice 0.
    for (i = 0; i < 5; i++)
        printf("El elemento %d es %f\n",
            i + 1, temp[i]);
    return 0;
}
```

Operaciones con vectores

Suma de dos vectores

```
#include <stdio.h>
int main()
{
    float v1[5] = {1, 34, 32, 45, 34};
    float v2[5] = {12, -3, 34, 15, -5};
    float v3[5];
    int i;

    for(i = 0; i < 5; i++)
        v3[i] = v1[i] + v2[i];

    printf("Vector3: ");
    for(i = 0; i < 5; i++)
        printf("%f ", v3[i]);
    printf("\n");
    return 0;
}
```


Operaciones con vectores

Multiplicar un vector por una constante

```
#include <stdio.h>
int main()
{
    float v1[5] = {1, 34, 32, 45, 34};
    float v2[5];
    float K = 3.0;
    int i;

    for(i = 0; i < 5; i++)
        v2[i] = K * v1[i];

    for(i = 0; i < 5; i++)
        printf("V1: %f\t V2: %f\n",
            v1[i], v2[i]);

    return 0;
}
```

Vectores de dimensión variable

La dimensión de un vector es un valor constante: **no puede usarse una variable** para definirla.

```
void main()
{
    int miVector[10];
}
```

```
# define N 10
void main()
{
    // Correcto: el precompilador sustituye N
    // por el valor constante 10
    int miVector[N];
}
```

Vectores de dimensión variable

La dimensión de un vector es un valor constante: **no puede usarse una variable** para definirla.

```
void main()
{
    int n = 10;
    // Error de sintaxis:
    // n es una variable
    int miVector[n];
}
```

Vectores de Dimensión Variable

Solución provisional: definir un vector de dimensión suficientemente elevada y emplear sólo un número reducido de elementos.

```
#include <stdio.h>
void main()
{
    int i, n;
    // Definimos un vector de
    // dimension muy grande
    int vect[100];

    printf("Nº datos? ");
    //El usuario debe teclear un n < 100
    scanf("%d", &n);
    //Utilizamos solo las n primeras
    for(i = 0; i < n; i++)
    {
        scanf("%d", &vect[i]);
    }
}
```

Funciones con vectores

Paso por referencia

Cuando un vector se pasa como argumento a una función **no se pasa el vector completo** sino la **dirección de memoria del primer elemento**.

```
void funcion (int vector[], int dimension);
```

Funciones con vectores

```
#include <stdio.h>

void imprime(int v[], int n);

void main()
{
    int v1[3] = {10, 20, 30};
    imprime(v1, 3);
}

void imprime(int v[], int n)
{
    int i;
    for(i = 0; i < n; i++)
        printf("%d\n",v[i]);
}
```

Funciones con vectores: paso por referencia

La función **puede modificar** el contenido de los elementos del vector ya que conoce la dirección de memoria donde están almacenados.

```
#include <stdio.h>

void toy(int vector[]);

void main()
{
    int x[] = {1, 2, 3};
    printf("Antes: %d\n", x[0]);
    toy(x); // ¡Sin asignacion!
    printf("Después: %d\n", x[0]);
}

void toy(int vector[])
{ //Funcion simple que modifica
  //el valor del primer elemento
  vector[0] = 100;
```

Otro ejemplo de funciones con vectores

```
#include <stdio.h>

void fAbs(int vector[], int n);

void main(){
    int datos[5]={-1, 3, -5, 7, -9};
    int i;
    fAbs(datos, 5); //¡Sin asignacion!

    for(i = 0; i < 5; i++) // Mostramos el resultado
        printf("%d ",datos[i]);
}

// La funcion recibe la direccion del primer elemento
void fAbs(int vector[], int n){
    int i;
    for(i = 0; i < n; i++)
        if(vector[i] < 0)
            vector[i] = -vector[i];
}
```


1 Vectores

2 Cadenas de caracteres

3 Matrices

4 Estructuras

5 Enumeraciones

Cadenas de caracteres en C

Definición

Conjunto de caracteres individuales (char)

Código

```
char identificador[dimensión];
```

tipo char

identificador Nombre de la cadena.

dimensión Número de elementos de la cadena (constante entero)
incluyendo el carácter de cierre (\0).

Definición e Inicialización de cadenas

```
// Declara una cadena de 10 caracteres
//( +1 para el cierre)
char cadena[11];

// Declara y asigna contenido
char cadena[5] = "Hola"; // 4 + 1

// Asigna por valores individuales
char cadena[5] = {'H', 'o', 'l', 'a', '\0'}; // 4 + 1

// Asigna por código ASCII
char cadena[5] = {72, 111, 108, 97, 0};
```

Definición e Inicialización de cadenas

```
// Declara una cadena, *no* define dimension
// y asigna contenido
char cadena[] = "Hola";

// Asigna por elementos individuales
char cadena[] = {'H', 'o', 'l', 'a', '\0'}; // 4 + 1;

// Asigna mediante codigo ASCII
char cadena[] = {72, 111, 108, 97, 0};
```

Elementos de una cadena

- Se referencian con el nombre seguido de un subíndice entre corchetes.
- El subíndice representa la posición del elemento dentro de la cadena.
- La **primera posición** tiene el **subíndice 0**.
- La **última posición** es el carácter nulo \0.

```
#include <stdio.h>

int main()
{
    char cadena[5] = "Hola";
    printf("%c \t %c \t %c \t %c \t %c\n",
           cadena[0], cadena[1],
           cadena[2], cadena[3],
           cadena[4]);
    return 0;
}
```

Asignación de valores

Error

```
char cadena[5];  
  
//Error de compilacion  
cadena = "Hola";
```

Solución provisional

Mejor con strcpy de string.h

```
char cadena[5];  
cadena[0] = 'H';  
cadena[1] = 'o';  
cadena[2] = 'l';  
cadena[3] = 'a';  
cadena[4] = '\\0';
```

Lectura y escritura de una cadena

- Usamos el especificador %s con printf y scanf.
- En scanf **debemos** especificar el **límite de caracteres** en el especificador de formato.
- En scanf **no** ponemos & delante del identificador.

```
#include <stdio.h>

int main()
{
    char texto[31];

    printf("Dime algo: \n");
    // Deja de leer cuando detecta un espacio
    // Imponemos el límite de caracteres
    scanf("%30s", texto);
    printf("Has dicho %s", texto);
    return 0;
}
```

Lectura de una cadena con espacios

- scanf con %s termina de leer cuando recibe un espacio o salto de línea.
- Para leer cadenas de caracteres que incluyan espacios se emplea el identificador %[^\n]

```
#include <stdio.h>

int main()
{
    char texto[31];

    printf("Dime algo: \n");
    // Deja de leer cuando detecta un salto de línea
    // o al alcanzar el límite de caracteres
    scanf("%30[^\n]", texto);
    // En printf seguimos usando %s
    printf("Has dicho %s\n", texto);
    return 0;
}
```


Recorrido por los elementos

- El bucle `while` es el más indicado, usando el carácter nulo para terminar:

```
#include <stdio.h>

int main()
{
    char cadena[5] = "Hola";
    int i = 0;
    printf("Los caracteres son:\n");
    while (cadena[i] != '\0')
    {
        printf("%c \t", cadena[i]);
        i++;
    }
    return 0;
}
```

Recorrido por los elementos

- También se puede usar un bucle for (equivalencia entre for y while)

```
#include <stdio.h>

int main()
{
    char cadena[5] = "Hola";
    int i = 0;
    printf("Los caracteres son:\n");
    for(i = 0; cadena[i] != '\0'; i++)
    {
        printf("%c \t", cadena[i]);
    }
    return 0;
}
```

Ejemplo: pasar a mayúsculas

```
#include <stdio.h>

void main()
{
    char cadena[5] = "Hola";
    // Distancia entre A y a
    int inc = 'A' - 'a';
    int i = 0;
    // Recorremos la cadena
    while(cadena[i] != '\0')
    { // Si el caracter es letra minusculta
        if (cadena[i] >= 'a' && cadena[i] <= 'z')
            //sumamos la distancia para pasar a
            //mayuscula
            cadena[i] += inc;
        i++;
    }
    printf("%s\n", cadena);
}
```

Librería string.h

La librería string.h incluye numerosas funciones dedicadas a cadenas de caracteres:

```
#include <string.h>
```

Longitud de una cadena `strlen`

Paso a mayúsculas `_strup`

Copiar cadenas `strcpy`

Concatenar cadenas `strcat`

Comparación de cadenas `strcmp`

Longitud de una cadena :: strlen

- strlen devuelve un entero con el número de caracteres.

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

void main()
{
    char palabra[21];
    int longitud;
    printf("Introduce una palabra: ");
    scanf("%20s", palabra);
    longitud = strlen(palabra);
    printf("Esta palabra tiene %d caracteres\n",
        longitud);
}
```

Paso a mayúsculas :: _strup

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

void main()
{
    char nombre[100];
    printf("Introduce tu nombre: ");
    scanf("%s", nombre);
    //;Sin asignacion!
    _strupr(nombre); //atencion al guion inicial
    printf("En mayusculas %s\n", nombre);
}
```

Copiar cadenas :: strcpy

Con strcpy tenemos una solución óptima para la **asignación de contenido**.

```
#include <stdio.h>
#include <string.h>
int main()
{
    char s1[50], s2[50];
    strcpy(s1, "Hello World!");
    strcpy(s2, s1);
    printf("%s\n", s2);
    return 0;
}
```

La cadena receptora debe tener espacio suficiente: *los caracteres sobrantes serán eliminados*.

Concatenar cadenas :: strcat

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

int main()
{
    char nombre_completo[50];
    char nombre[ ] = "Juana";
    char apellido[ ] = "de Arco";
    // Copiamos por tramos:
    // Primero el nombre
    strcpy(nombre_completo, nombre);
    // A continuacion un espacio
    strcat(nombre_completo, " ");
    // Finalmente el apellido
    strcat(nombre_completo, apellido);
    printf("El nombre completo es: %s.\n",
           nombre_completo);
    return 0;
}
```


Comparación de cadenas :: strcmp

- Si las dos cadenas son iguales entrega un 0.

```
#include <stdio.h>
#include <string.h>
void main()
{
    char color[] = "negro";
    char respuesta[11];
    do // El bucle se repite mientras
        { // las cadenas *no* coincidan
            printf("Adivina un color: ");
            scanf ("%10s", respuesta);
        } while (strcmp(color, respuesta) != 0);
    printf(";Correcto!\n");
}
```

Comparación de cadenas :: strcmp

- Si hay diferencias, es positivo si el valor ASCII del primer carácter diferente es mayor en la cadena 1.

```
#include <stdio.h>
#include <string.h>
int main()
{
    char s1[] = "abcdef";
    char s2[] = "abCdef";
    char s3[] = "abcdff";
    int res;
    res = strcmp(s1, s2);
    printf("strcmp(s1, s2) = %d\n",
           res);
    res = strcmp(s1, s3);
    printf("strcmp(s1, s3) = %d\n",
           res);
    return 0;
}
```

Funciones y cadenas

Una función acepta una cadena como argumento: **paso por referencia** (igual que un vector).

```
#include <stdio.h>

void imprime(char cadena[]);

void main() {
    char saludo[]="Hola";
    imprime(saludo);
}

void imprime(char cadena[]) {
    int i=0;
    while(cadena[i]!='\0') {
        printf("%c", cadena[i]);
        i++;
    }
    printf("\n");
}
```

- 1 Vectores
- 2 Cadenas de caracteres
- 3 Matrices**
- 4 Estructuras
- 5 Enumeraciones

Matrices

Una matriz es un conjunto de valores del mismo tipo (int, char, float, etc.), de dos o más dimensiones

```
tipo identificador[dimension_1][dimension_2] ... [dimension_n];
```

tipo Tipo de los elementos de la matriz.

identificador Nombre de la matriz.

dimensión_n Dimensión n-ésima de la matriz.

Ejemplo

```
// Crea una matriz de datos enteros, llamada  
// tabla, de dos dimensiones y 9 elementos.  
int tabla[3][3];
```

Elementos de una matriz

- Se referencian con el nombre de la matriz seguido de tantos subíndices, entre corchetes, como dimensiones tenga la matriz.

```
#include <stdio.h>
void main (){
    int matriz[2][2]; // Matriz 2 x 2
    int fila, columna;
    // Inicializacion de elementos
    matriz[0][0] = 1;
    matriz[0][1] = 2;
    matriz[1][0] = 3;
    matriz[1][1] = 4;
    // Recorre matriz con un bucle for anidado
    for(fila = 0; fila < 2; fila++) {
        for(columna = 0; columna < 2; columna++)
            printf("%d\t", matriz[fila][columna]);
        printf("\n\n");
    }
}
```

Inicialización de una matriz

- Los elementos de una matriz pueden iniciarse en el momento de la declaración.

```
#include <stdio.h>
void main()
{
    int matriz[2][3] = // Matriz 2 x 3
    {
        {10, 20, 30}, // 1a fila
        {40, 50, 60} // 2a fila
    };
    int fil, col;
    // Recorremos con bucle anidado
    for(fil = 0; fil < 2; fil++){
        for(col = 0; col < 3; col++){
            printf(" %d\t",matriz[fil][col]);
            printf("\n\n");
        }
    }
}
```

Inicialización de una matriz

- Los elementos de una matriz pueden iniciarse en el momento de la declaración.

```
#include <stdio.h>
int main()
{
    // Matriz de dos filas, tres columnas
    int matriz[2][3] = {10, 20, 30, 40, 50, 60};
    int fil, col;
    // Recorremos con bucle anidado
    for(fil = 0; fil < 2; fil++)
    {
        for(col = 0; col < 3; col++)
            printf(" %d\t",matriz[fil][col]);
        printf("\n\n");
    }
    return 0;
}
```


Operaciones con matrices: suma

```
#include <stdio.h>
int main()
{
    int i,j;
    int m1[2][3] = {1, 2, 3, 4, 5, 6};
    int m2[2][3] = {4, 5, 12, 23, -5, 6};
    int m3[2][3]; // Matriz resultado
    // Realiza la suma con bucle anidado
    for(i = 0; i < 2; i++) // Filas
        for(j = 0; j < 3; j++) // Columnas
            m3[i][j] = m1[i][j] + m2[i][j];
    // Imprime resultado con bucle anidado
    for(i = 0; i < 2; i++) // Filas
    {
        for(j = 0; j < 3; j++) // Columnas
            printf("%d\t",m3[i][j]);
        printf("\n");
    }
}
```

Funciones con matrices

- Una matriz siempre se pasa por referencia: no se pasa la matriz completa sino la dirección del primer elemento.

```
void funcion (int matriz[3][3], int nFil, int nCol);
```

- Se puede omitir el número de filas pero **no** el número de columnas.

```
void funcion (int matriz[][3], int nFil, int nCol);
```

```
//Error de sintaxis
```

```
void funcion (int matriz[][], int nFil, int nCol);
```

Ejemplo de función con matrices

```
#include <stdio.h>

void imprime_matriz(int M[][2], int f, int c);

void main()
{
    int tabla[2][2] = {{1,2}, {3,4}};
    imprime_matriz(tabla, 2, 2);
}

void imprime_matriz(int M[][2], int f, int c)
{
    int i, j;
    for(i = 0; i < f; i++) {
        for(j = 0; j < c; j++)
            printf("%d ", M[i][j]);
        printf("\n");
    }
}
```

Ejemplo de función con matrices (2)

```
#include <stdio.h>

void absMatriz(int M[][2], int f, int c);

void main()
{
    int tabla[2][2] = {{-1,2}, {-3,4}};
    printf("Antes: %d\n", tabla[0][0]);
    absMatriz(tabla, 2, 2); // ¡Sin asignacion!
    printf("Después: %d\n", tabla[0][0]);
}

void absMatriz(int M[][2], int f, int c)
{
    int i, j;
    for(i = 0; i < f; i++)
        for(j = 0; j < c; j++)
            if (M[i][j] < 0)
                M[i][j] = -M[i][j];
}
```

- 1 Vectores
- 2 Cadenas de caracteres
- 3 Matrices
- 4 Estructuras**
- 5 Enumeraciones

Estructuras en C

Permiten almacenar valores de diferentes tipos bajo un mismo identificador.

```
struct identificador
{
    tipo_1 comp_1;
    tipo_2 comp_2;
    ...
    tipo_n comp_n;
};
```

identificador Nombre de la estructura

tipo_n Tipo de datos del componente comp_n.

comp_n Componente n-ésimo de la estructura.

Estructuras con typedef struct

Permiten usar estructuras (u otros tipos) sin necesidad de usar la palabra clave struct.

```
typedef struct
{
    tipo_1 comp_1;
    tipo_2 comp_2;
    ...
    tipo_n comp_n;
} identificador;
```

Ejemplo con struct

```
struct contacto
{
    char nombre[30];
    int telefono;
    int edad;
};

void main()
{
    struct contacto person1;
}
```


Ejemplo con typedef struct

```
typedef struct
{
    char nombre[30];
    int telefono;
    int edad;
} contacto;

void main()
{
    contacto person1;
}
```

Inicialización de valores en estructuras

Si no se especifica el identificador de cada componente la asignación se realiza en orden

```
typedef struct {  
    char nombre[50];  
    char apellidos[50];  
    int matricula;  
} ficha;  
  
void main ()  
{  
    ficha alumno1 = {"Yo", "Soy Aquel", 1234};  
}
```

Inicialización de valores en estructuras

Con el identificador de cada componente se puede asignar en cualquier orden

```
typedef struct {  
    char nombre[50];  
    char apellidos[50];  
    int matricula;  
} ficha;  
  
void main ()  
{  
    ficha alumno1 = {.apellidos = "Soy Aquel",  
                     .matricula = 1234,  
                     .nombre = "Yo"};  
}
```

Asignación de valores en estructuras

```
typedef struct {  
    int day;  
    int month;  
    int year;  
} date;  
  
void main ()  
{  
    date d1, d2, d3;  
    // Asignacion por componentes  
    d1.day = 31;  
    d1.month = 12;  
    d1.year = 1999;  
    // Asignacion con el operador cast  
    d2 = (date) {1, 1, 2000};  
    // Asignacion por copia  
    d3 = d1;  
}
```

Asignación de cadenas en estructuras

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

typedef struct {
    char nombre[50];
    char apellidos[50];
    int matricula;
} ficha;

void main ()
{
    ficha alumno1, alumno2, alumno3;
    // Para asignar cadenas usamos strcpy
    strcpy(alumno1.nombre, "Yo");
    strcpy(alumno1.apellidos, "Soy Aquel");
    alumno1.matricula = 1234;
}
```

Acceso a componentes de una estructura

```
#include <stdio.h>

typedef struct {
    char nombre[50];
    char apellidos[50];
    int matricula;
} ficha;

void main ()
{
    ficha alumno;
    printf("Nombre:");
    scanf("%s", alumno.nombre);
    printf("Apellidos:");
    scanf("%s", alumno.apellidos);
    printf("Numero de matricula:");
    scanf("%d", &alumno.matricula);
}
```

Estructuras dentro de estructuras

- Una estructura puede contener otras estructuras.

```
typedef struct
{
    int d, m, a;
} fecha;
```

```
typedef struct
{
    char nombre[50];
    char apellidos[50];
    int matricula;
    fecha fNacimiento;
} ficha;
```

Estructuras dentro de estructuras

```
ficha alumno1, alumno2;  
  
alumno1.fNacimiento.d = 31;  
alumno1.fNacimiento.m = 12;  
alumno1.fNacimiento.a = 1999;  
  
alumno2.fNacimiento = (fecha){1, 1, 2000};
```


Vector de estructuras

A partir de una estructura previamente definida se pueden generar vectores basados en esa estructura.

```
#include <stdio.h>

typedef struct
{
    int day;
    int month;
    int year;
} date;

void main()
{
    date fechas[3] = { // Vector de 3 fechas
        {1, 1, 1999},
        {31, 12, 2000},
        {15, 5, 1980}
    };
}
```

Vector de estructuras

La asignación de valores sigue las mismas reglas que para vectores de tipos simples (mediante []).

```
#include <stdio.h>

typedef struct
{
    int day;
    int month;
    int year;
} date;

void main()
{
    date fechas[3]; // Vector de 3 fechas
    fechas[1].day = 1;
    fechas[2] = (date) {31, 12, 1999};
    fechas[3] = fechas[2];
}
```

Funciones y estructuras

- Una función acepta estructuras (**paso por valor**).

```
#include <stdio.h>
#include <math.h>
//Definicion de estructura
typedef struct {
    float real, imaginaria;
} complejo;
//Funcion que acepta una estructura
float modulo(complejo c);

void main(){
    complejo comp={1, 3};
    printf("El modulo es: %f\n", modulo(comp));
}

//Implementacion de la funcion
float modulo(complejo c){
    return sqrt(c.real * c.real +
               c.imaginaria * c.imaginaria);
}
```

Funciones y estructuras

- Una función puede devolver una estructura

```
#include <stdio.h>

typedef struct {
    float real, imaginaria;
} complejo;

//Funcion que devuelve estructura
complejo conjugado(complejo c);

void main(){
    complejo comp1 = {1, 3}, comp2;
    comp2 = conjugado(comp1);
    printf("%f", comp2.imaginaria);
}

complejo conjugado(complejo c){
    return (complejo) {c.real, -c.imaginaria};
}
```

- 1 Vectores
- 2 Cadenas de caracteres
- 3 Matrices
- 4 Estructuras
- 5 Enumeraciones**

Enumeraciones en C

Definición

Con `enum` se pueden definir tipos de datos enteros que tengan un rango limitado de valores, y darle un nombre significativo a cada uno de los posibles valores.

Código

```
enum nombre_enum {lista_de_valores};
```

Ejemplo

```
enum dia{ //valores enteros: 0 al 6
    lunes, martes, miercoles, jueves, viernes, sabado, domingo
};
```

Ejemplo (1)

```
#include <stdio.h>
enum dia{ //valores enteros: 0 al 6
    lunes, martes, miercoles, jueves, viernes, sabado, domingo
};

int main()
{
    enum dia hoy, manana;
    hoy = lunes;
    manana = hoy + 1;
    printf("%d\n", hoy);
    printf("%d\n", manana);
    return 0;
}
```

Ejemplo (2)

```
#include <stdio.h>
enum dia{ //valores enteros: 1 en adelante
    lunes = 1, martes, miercoles, jueves, viernes, sabado, domingo
};

int main()
{
    enum dia hoy, manana;
    hoy = lunes;
    manana = hoy + 1;
    printf("%d\n", hoy);
    printf("%d\n", manana);
    return 0;
}
```