

**Resulta que ...**

Antes de que existieran las computadoras, en una organización era normal tener un registro manual de la información para agilizar los procesos. Por ejemplo, una biblioteca tenía un registro de fichas de sus libros y cada vez que se retiraba o devolvía algún libro se dejaba asentado. Este control con fichas, ocurría en distintas áreas, no sólo en las bibliotecas. Otro ejemplo podría ser el de una empresa pequeña de vuelos comerciales, tenía su fichero de viajes, origen, destino, duración, costo....



Ficha de viaje:

Ficha de viaje:

Ficha de viaje:

Ficha de viaje:

Origen: AEP  
Destino: CNQ  
Duración en minutos: 95  
Precio: 680



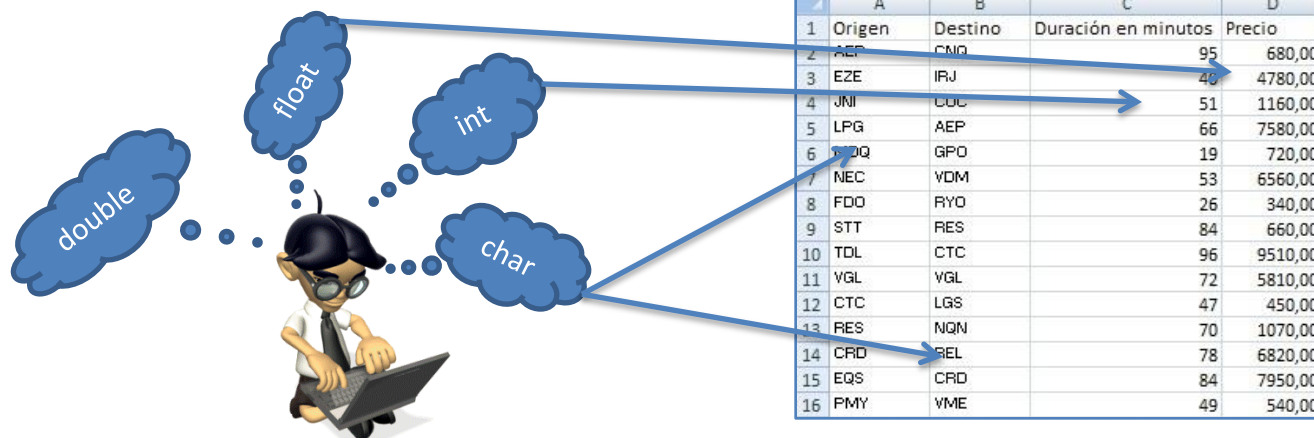
Más tarde las computadoras fueron llegando a empresas y hogares , hubo procesos que se informatizaron, agilizando las tareas y con ello pudieron hacer más tareas para mejorar sus servicios.




	A	B	C	D
1	Origen	Destino	Duración en minutos	Precio
2	AEP	CNQ	95	680,00
3	EZE	IRJ	40	4780,00
4	JNI	COC	51	1160,00
5	LPG	AEP	66	7580,00
6	MDQ	GPO	19	720,00
7	NEC	YDM	53	6560,00
8	FDO	RYO	26	340,00
9	STT	RES	84	660,00
10	TDL	CTC	96	9510,00
11	VGL	VGL	72	5810,00
12	CTC	LGS	47	450,00
13	RES	NGN	70	1070,00
14	CRD	REL	78	6820,00
15	EQS	CRD	84	7950,00
16	PMY	VME	49	540,00



También se popularizaron distintos lenguajes de programación y con los lenguajes, había que manejar **tipos de datos**. Estos tipos de datos son los ofrecidos por el lenguaje o predefinidos, en C son los que conocemos hasta ahora...





	A	B	C	D
	Origen	Destino	Duración en minutos	Precio
1	AEP	CMD	35	680,00
3	EZE	IBJ	40	4780,00
4	JNI	COC	51	1160,00
5	LPG	AEP	66	7580,00
6	MDQ	GPO	19	720,00
7	NEC	VDM	53	6560,00
8	FDO	RYO	26	340,00
9	STT	RES	84	660,00
10	TDL	CTC	96	9510,00
11	VGL	VGL	72	5810,00
12	CTC	LGS	47	450,00
13	RES	NQN	70	1070,00
14	CRD	REL	78	6820,00
15	EQS	CRD	84	7950,00
16	PMY	VME	49	540,00

Pero las tareas y los programas exigen que se opere con muchos tipos de datos. Por ejemplo, en la imagen de la planilla tenemos 4 columnas, cada una con su tipo de datos, el desarrollo del programa requiere todas las declaraciones.

Existe algún tipo de datos que gestione los casos dónde tengo muchos datos de distinto tipo?... No existe, una de las razones es que no se sabe que tipo y cantidad de datos vamos a gestionar, cada caso es distinto.... Lo bueno es que podemos crearlo nosotros...

# **TIPOS DE DATOS CREADO POR EL PROGRAMADOR**

**(TAMBIÉN LLAMADOS DEFINIDOS POR EL USUARIO)**

## **ESTRUCTURAS**

En C una estructura es una colección de variables que se referencian bajo el mismo nombre. Una estructura proporciona un medio conveniente para mantener junta información que se relaciona.

*Una estructura forma una plantilla que se puede usar para crear variables de estructura. Es un tipo de datos definido por el programador.*

Las variables que forman la estructura son llamados *elementos estructurados*. Generalmente, todos los elementos en la estructura están relacionados lógicamente unos con otros. Por ejemplo, tomando la imagen del ejemplo, el vuelo de origen AEP, tiene un destino que es CNQ, una duración en minutos y un precio. Esa fila es única, es para ese vuelo, y con las siguientes ocurre lo mismo. Al mismo tiempo, todo ese conjunto se refiere a lo mismo, tiene una información determinada.

	A	B	C	D
1	Origen	Destino	Duración en minutos	Precio
2	AEP	CNQ	95	680,00
3	EZE	IRJ	40	4780,00
4	JNI	COC	51	1160,00
5	LPG	AEP	66	7580,00
6	MDQ	GPO	19	720,00
7	NEC	VDM	53	6560,00
8	FDO	RYO	26	340,00
9	STT	RES	84	660,00
10	TDL	CTC	96	9510,00
11	VGL	VGL	72	5810,00
12	CTC	LGS	47	450,00
13	RES	NQN	70	1070,00
14	CRD	REL	78	6820,00
15	EQS	CRD	84	7950,00
16	PMY	VME	49	540,00

El ejemplo se puede representar con una estructura mediante la palabra clave **struct**, así se le indica al compilador que defina un tipo de datos estructura.

Las estructuras nos permiten agrupar valores de tipos diferentes en una misma variable (sería un **tipo compuesto**).

- Una estructura se divide en **campos**.
- Cada campo puede ser de cualquier **tipo**.
- Cada campo de la estructura se identifica por un **nombre**.

Nos permiten crear nuevos tipos de datos en C (**tipos definidos por el usuario**).

Podemos definir diferentes tipos de estructuras en nuestro programa, todas las que necesitemos. Y al definir una estructura, se especifica:

- El número de campos.
- El nombre de los campos.
- El tipo de los campos.

Cada tipo de estructura definido pasa a ser un nuevo **tipo de datos** que podemos utilizar para declarar variables.

La sintaxis de una estructura es:

```
struct <nombre_estructura> {  
    Tipo1  dato1;  
    Tipo2  dato2;  
    ...  
};
```

Las estructuras se declaran después de los #include, fuera y antes del comienzo de la función main(). Una propiedad interesante del tipo de dato estructura es que pueden anidarse otras estructuras.

```
struct punto {  
    int x;  
    int y;  
}; // observa el punto y coma al final.
```

El nombre de la estructura es “punto” y tiene como miembros un par de variables enteras: “x” e “y”.



# **ESTRUCTURAS DECLARACIÓN, VARIABLES E INICIALIZACIÓN**

```
//Ejemplo 1
#include <stdio.h>
#include <string.h>
```

```
// declaro una estructura
struct vuelo {
    char origen[5];
    char destino[5];
    int duracion;
    float precio;
}; // definí un nuevo tipo de datos
```

```
int main() {
```

```
    struct vuelo un_vuelo = {"AEP", "CNQ", 95, 680.00}; /*declaro la variable un_vuelo de
tipo struct vuelo, inicializo los campos*/
    struct vuelo otro_vuelo;
```

```
    strcpy(otro_vuelo.origen,"EZE");
    strcpy(otro_vuelo.destino,"IRJ");
    otro_vuelo.duracion = 40;
    otro_vuelo.precio = 4780;
```

```
    printf("Origen: %s, Destino: %s, Duración: %d, Precio: %.2f\n",
        un_vuelo.origen, un_vuelo.destino, un_vuelo.duracion, un_vuelo.precio);
    printf("Origen: %s, Destino: %s, Duración: %d, Precio: %.2f\n",
        otro_vuelo.origen, otro_vuelo.destino, otro_vuelo.duracion, otro_vuelo.precio);
```

```
    getchar();
    return 0;
}
```

Una vez definida la estructura, pueden hacerse sucesivas declaraciones. Para trabajar con estructuras hace falta conocer sus operadores:

**variable\_estructura.miembro = valor;**

Según esta instrucción se está accediendo a un miembro específico de una estructura y se está modificando de valor.

Las variables `otro_vuelo` y `mas_vuelos`, *son declaradas análogamente* como si lo hiciéramos con cualquiera de los tipos conocidos.

```
//Ejemplo 2
#include <stdio.h>
#include <string.h>

struct vuelo {
    char origen[5];
    char destino[5];
    int duracion;
    float precio;
} otro_vuelo, mas_vuelos; // puedo declarar variables aquí

int main() {
    struct vuelo un_vuelo = {"AEP", "CNQ", 95, 680.00};
    // y las uso dentro de mi programa
    strcpy(otro_vuelo.origen, "EZE");
    strcpy(otro_vuelo.destino, "IRJ");
    otro_vuelo.duracion = 40;
    otro_vuelo.precio = 4780;

    printf("Origen: %s, Destino: %s, Duración: %d, Precio: %.2f\n",
        un_vuelo.origen, un_vuelo.destino, un_vuelo.duracion, un_vuelo.precio);
    printf("Origen: %s, Destino: %s, Duración: %d, Precio: %.2f\n",
        otro_vuelo.origen, otro_vuelo.destino, otro_vuelo.duracion, otro_vuelo.precio);
    getchar();
    return 0;
}
```

```
//ejemplo 3
#include <stdio.h>
#include <string.h>
```

**typedef** se puede usar para definir nuevos nombres de datos explícitamente,

```
typedef struct vuelo {
    char origen[5];
    char destino[5];
    int duracion;
    float precio;
} vuelos;

int main() {
    vuelos un_vuelo = {"AEP", "CNQ", 95, 680.00};
    vuelos otro_vuelo;

    strcpy(otro_vuelo.origen, "EZE");
    strcpy(otro_vuelo.destino, "IRJ");
    otro_vuelo.duracion = 40;
    otro_vuelo.precio = 4780;

    printf("Origen: %s, Destino: %s, Duración: %d, Precio: %.2f\n",
        un_vuelo.origen, un_vuelo.destino, un_vuelo.duracion,
        un_vuelo.precio);
    printf("Origen: %s, Destino: %s, Duración: %d, Precio: %.2f\n",
        otro_vuelo.origen, otro_vuelo.destino, otro_vuelo.duracion,
        otro_vuelo.precio);
    getchar();
    return 0;
}
```

# **ANIDAMIENTO DE ESTRUCTURAS**

Las estructuras  
pueden estar anidadas  
unas dentro de otras.

```
//ejemplo 4
#include <stdio.h>
#include <string.h>
```

```
struct fecha{
int dia, mes, anio;
};
```

```
struct vuelo {
    char origen[5];
    char destino[5];
    int duracion;
    float precio;
    struct fecha date;
};
```

```
int main() {
    struct vuelo un_vuelo = {"AEP", "CNQ", 95, 680.00,
    {11,4,2020}};
```

```
    struct vuelo otro_vuelo;
```

```
    struct vuelo tercer_vuelo;
    strcpy(otro_vuelo.origen,"EZE");
    strcpy(otro_vuelo.destino,"IRJ");
```

```
    otro_vuelo.duracion = 40;
```

```
    otro_vuelo.precio = 4780;
```

```
    otro_vuelo.date.dia = 12;
```

```
    otro_vuelo.date.mes = 4;
```

```
    otro_vuelo.date.anio = 2020;
```

```
    printf("Ingrese el vuelo origen\n");
```

```
    gets(tercer_vuelo.origen);
```

```
    printf("Ingrese el vuelo destino\n");
```

```
    gets(tercer_vuelo.destino);
```

```
    printf("Ingrese duración\n");
```

```
    scanf("%d",&tercer_vuelo.duracion);
```

```
    printf("Ingrese el precio\n");
```

```
    scanf("%f",&tercer_vuelo.precio);
```

```
    printf("Ingrese el dia\n");
```

```
    scanf("%d",&tercer_vuelo.date.dia);
```

```
    printf("Ingrese el mes\n");
```

```
    scanf("%d",&tercer_vuelo.date.mes);
```

```
    printf("Ingrese el año\n");
```

```
    scanf("%d",&tercer_vuelo.date.anio);
```

```
    printf("Origen: %s, Destino: %s, Duración: %d, Precio: %.2f,
    Fecha:%d/%d/%d\n",
```

```
        un_vuelo.origen, un_vuelo.destino, un_vuelo.duracion,
        un_vuelo.precio,
```

```
        un_vuelo.date.dia,un_vuelo.date.mes, un_vuelo.date.anio);
```

```
    printf("Origen: %s, Destino: %s, Duración: %d, Precio: %.2f,
    Fecha:%d/%d/%d\n",
```

```
        otro_vuelo.origen, otro_vuelo.destino, otro_vuelo.duracion,
        otro_vuelo.precio,
```

```
        otro_vuelo.date.dia,otro_vuelo.date.mes,
```

```
        otro_vuelo.date.anio);
```

```
    printf("Origen: %s, Destino: %s, Duración: %d, Precio: %.2f,
    Fecha:%d/%d/%d\n",
```

```
        tercer_vuelo.origen, tercer_vuelo.destino,
        tercer_vuelo.duracion, tercer_vuelo.precio,
```

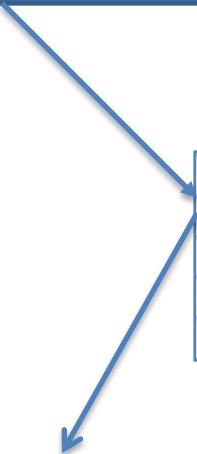
```
        tercer_vuelo.date.dia, tercer_vuelo.date.mes,
        tercer_vuelo.date.anio);
```

```
    getchar();
```

```
    return 0;}
```

# **ESTRUCTURAS** **Y** **ARRAYS**

Ficha de viaje:	Ficha de viaje:	Ficha de viaje:	Ficha de viaje:
Origen: AEP	Origen: EZE	Origen: JNI	Origen: LPG
Destino: CNQ	Destino: IRJ	Destino: COC	Destino: AEP
Duración en minutos: 95	Duración en minutos: 40	Duración en minutos: 51	Duración en minutos: 66
Precio: 680	Precio: 4780	Precio: 1160	Precio: 7580



1	Origen	Destino	Duración en minutos	Precio
2	AEP	CNQ	95	680,00
3	EZE	IRJ	40	4780,00
4	JNI	COC	51	1160,00
5	LPG	AEP	66	7580,00

AEP	EZE	JNI	LPG
CNQ	IRJ	COC	AEP
95	40	51	66
680	4780	1160	7580
0	1	2	3
ABE9C	ABEAE	ABECO	ABED2

Normalmente las estructuras se trabajan con arrays de una o mas dimensiones.



Normalmente las estructuras se trabajan con arrays de una o mas dimensiones

```
//ejemplo 5
#include <stdio.h>
#include <string.h>
struct vuelo {
    char origen[5];
    char destino[5];
    int duracion;
    float precio; };
int main() {
    const int dim=2;
    struct vuelo un_vuelo[dim]; // creo un vector de tipo struct vuelo
    int i;
    for (i = 0; i < dim; i++) {
        printf("Ingrese el vuelo origen\n");
        gets(un_vuelo[ i ].origen);
        fflush(stdin);
        printf("Ingrese el vuelo destino\n");
        gets(un_vuelo[ i ].destino);
        fflush(stdin);
        printf("Ingrese duración\n");
        scanf("%d",&un_vuelo[ i ].duracion);
        fflush(stdin);
        printf("Ingrese el precio\n");
        scanf("%f",&un_vuelo[ i ].precio);
        fflush(stdin); }
    for (i=0; i< dim; i++){
        printf("Origen: %s, Destino: %s, Duración: %d, Precio: %.2f\n",
            un_vuelo[i].origen, un_vuelo[i].destino, un_vuelo[i].duracion, un_vuelo[i].precio);
    }
    getchar();
    return 0;}
```

```
//ejemplo 6
#include <stdio.h>
#include <string.h>
```

```
struct fecha{
int dia, mes, anio;
};
```

```
struct vuelo {
    char origen[5];
    char destino[5];
    int duracion;
    float precio;
    struct fecha date;
};
```

```
int main() {
    const int dim=2;
    struct vuelo un_vuelo[dim];
    int i;

    for (i = 0; i < dim; i++){
        printf("Ingrese el vuelo origen\n");
        gets(un_vuelo[i].origen);
        fflush(stdin);
        printf("Ingrese el vuelo destino\n");
        gets(un_vuelo[i].destino);
        fflush(stdin);
        printf("Ingrese duración\n");
        scanf("%d",&un_vuelo[i].duracion);
        fflush(stdin);
        printf("Ingrese el precio\n");
        scanf("%f",&un_vuelo[i].precio);
        fflush(stdin);
```

Las estructuras anidadas en un array de estructuras, se trabajan de la misma forma.

```
printf("Ingrese el dia\n");
scanf("%d",&un_vuelo[i].date.dia);
fflush(stdin);
printf("Ingrese el mes\n");
scanf("%d",&un_vuelo[i].date.mes);
fflush(stdin);
printf("Ingrese el año\n");
scanf("%d",&un_vuelo[i].date.anio);
fflush(stdin);
    }
    for (i=0; i< dim; i++){
        printf("Origen: %s, Destino: %s, Duración: %d, Precio: %.2f,
Fecha:%d/%d/%d\n",
            un_vuelo[i].origen, un_vuelo[i].destino, un_vuelo[i].duracion,
            un_vuelo[i].precio,
            un_vuelo[i].date.dia,un_vuelo[i].date.mes,
            un_vuelo[i].date.anio);
    }

    getchar();
    return 0;
}
```

```
//ejemplo 7
#include <stdio.h>
#include <string.h>
```

Se puede declarar un array o más (si son necesarios) dentro de la estructura

```
struct fecha{
int dia, mes, anio;
};
```

```
struct vuelo {
    char origen[5];
    char destino[5];
    int duracion;
    float precio;
    struct fecha date;
    float combustible[3];
};
```

```
int main() {
    const int dim=2;
    struct vuelo un_vuelo[dim];
    int i, j; // agrego un índice para recorrer el vector de combustible
```

```
    for (i = 0; i < dim; i++){
        printf("Ingrese el vuelo origen\n");
        gets(un_vuelo[i].origen);
        fflush(stdin);
        printf("Ingrese el vuelo destino\n");
        gets(un_vuelo[i].destino);
        fflush(stdin);
        printf("Ingrese duración\n");
        scanf("%d",&un_vuelo[i].duracion);
        fflush(stdin);
        printf("Ingrese el precio\n");
        scanf("%f",&un_vuelo[i].precio);
        fflush(stdin);
```

```
        printf("Ingrese el dia\n");
        scanf("%d",&un_vuelo[i].date.dia);
        fflush(stdin);
        printf("Ingrese el mes\n");
        scanf("%d",&un_vuelo[i].date.mes);
        fflush(stdin);
        printf("Ingrese el año\n");
        scanf("%d",&un_vuelo[i].date.anio);
        fflush(stdin);
        for ( j = 0; j < 3; j++){
            printf("Ingrese la cantidad de combustible del viaje %d\n", j + 1);
            scanf("%f",&un_vuelo [ i ].combustible[ j ]);
            fflush(stdin);
        }
    }

    for (i=0; i< dim; i++){
        printf("Origen: %s, Destino: %s, Duracion: %d, Precio: %.2f, Fecha:%d/%d/%d\n",
            un_vuelo[i].origen, un_vuelo[i].destino, un_vuelo[i].duracion,
            un_vuelo[i].precio,
            un_vuelo[i].date.dia,un_vuelo[i].date.mes,
            un_vuelo[i].date.anio);
        for ( j = 0; j < 3; j++){
            printf("Combustible de este trayecto gastado en el dia: %d, %.2f\n", j+1, un_vuelo[i].combustible[j]);
        }
    }

    getchar();
    return 0;
}
```

# ESTRUCTURAS Y FUNCIONES

```
//ejemplo 8
#include <stdio.h>
#include <string.h>
```

```
struct fecha{
int dia, mes, anio;
};

struct vuelo {
    char origen[5];
    char destino[5];
    int duracion;
    float precio;
    struct fecha date;
    float combustible[3];
};
```

```
void carga_vuelo(struct vuelo v[], int d){
int i,j;
for (i = 0; i < d; i++){
    printf("Ingrese el vuelo origen\n");
    gets(v[i].origen);
    fflush(stdin);
    printf("Ingrese el vuelo destino\n");
    gets(v[i].destino);
    fflush(stdin);
    printf("Ingrese duración\n");
    scanf("%d",&v[i].duracion);
    fflush(stdin);
    printf("Ingrese el precio\n");
    scanf("%f",&v[i].precio);
    fflush(stdin);
    printf("Ingrese el dia\n");
    scanf("%d",&v[i].date.dia);
    fflush(stdin);
```

El paso de parámetros a una función es similar al de un arrays de tipo predefinido sólo que en el encabezado de la definición de la función hay que agregar la palabra struct y el nombre que le dimos

```
printf("Ingrese el mes\n");
scanf("%d",&v[i].date.mes);
fflush(stdin);
printf("Ingrese el año\n");
scanf("%d",&v[i].date.anio);
fflush(stdin);
for ( j = 0; j < 3; j++){
    printf("Ingrese la cantidad de combustible del viaje %d\n", j + 1);
    scanf("%f",&v[i].combustible[j]);
    fflush(stdin);
} } }
```

```
void emito_vuelo(struct vuelo v[], int d){
int i,j;
for (i=0; i< d; i++){
    printf("Origen: %s, Destino: %s, Duracion: %d, Precio: %.2f,
Fecha:%d/%d/%d\n",
        v[i].origen, v[i].destino, v[i].duracion, v[i].precio,
        v[i].date.dia, v[i].date.mes, v[i].date.anio);
    for ( j = 0; j < 3; j++){
        printf("Combustible de este trayecto gastado en el dia: %d,
%.2f\n", j+1, v[i].combustible[j]);
    } } }
```

```
int main() {
const int dim=2;
struct vuelo un_vuelo[dim];
printf("Comienzo de programa...\n");
carga_vuelo(un_vuelo, dim);
printf("Emisión de los datos cargados...\n");
emito_vuelo(un_vuelo, dim);
printf("Fin de programa...\n");
getchar();
return 0; }
```

```
/*ejemplo 9*/  
#include <stdio.h>  
#include <string.h>
```

```
struct fecha{  
int dia, mes, anio;  
};
```

```
struct vuelo {  
char origen[5];  
char destino[5];  
int duracion;  
float precio;  
struct fecha date;  
float combustible[3];  
};
```

```
void emite_reg(struct vuelo reg){  
int j;  
printf("Origen: %s, Destino: %s, Duracion: %d, Precio: %.2f,  
Fecha:%d/%d/%d\n",  
reg.origen, reg.destino, reg.duracion, reg.precio,  
reg.date.dia, reg.date.mes, reg.date.anio);  
for ( j = 0; j < 3; j++){  
printf("Combustible de este trayecto gastado en el dia:  
%d, %.2f\n", j+1, reg.combustible[j]);  
}  
}
```

```
void emito_vuelo(struct vuelo v[], int d){  
int i,j;  
for (i=0; i< d; i++){  
emite_regv[i];  
}  
}
```

```
void carga_reg(struct vuelo * ptr_reg){  
int j;  
printf("Ingrese el vuelo origen\n");  
gets(ptr_reg->origen);  
fflush(stdin);  
printf("Ingrese el vuelo destino\n");  
gets(ptr_reg->destino);  
fflush(stdin);  
printf("Ingrese duraci3n\n");  
scanf("%d",&ptr_reg->duracion);  
fflush(stdin);  
printf("Ingrese el precio\n");  
scanf("%f",&ptr_reg->precio);  
fflush(stdin);  
printf("Ingrese el dia\n");  
scanf("%d",&ptr_reg->date.dia);  
fflush(stdin);  
printf("Ingrese el mes\n");  
scanf("%d",&ptr_reg->date.mes);  
fflush(stdin);  
printf("Ingrese el a3o\n");  
scanf("%d",&ptr_reg->date.anio);  
fflush(stdin);  
for ( j = 0; j < 3; j++){  
printf("Ingrese la cantidad de combustible del viaje %d\n", j + 1);  
scanf("%f",&ptr_reg->combustible[j]);  
fflush(stdin);  
}  
}
```

```
void carga_vuelo(struct vuelo v[], int d){  
int i,j;  
char op;  
for (i = 0; i < d; i++)  
carga_reg(&v[i]);  
}
```

Podemos modificar registro del struct pasando como argumento ese dato por direcci3n, el operador -> sirve para acceder a los miembros de una estructura con un puntero. Tambi3n puede ser por ejemplo(\*ptr\_reg).campo)

```

/*ejemplo 10*/
#include <stdio.h>
#include <string.h>

struct fecha{
int dia, mes, anio;
};

struct vuelo {
    char origen[5];
    char destino[5];
    int duracion;
    float precio;
    struct fecha date;
    float combustible[3];
};

void emite_reg(struct vuelo reg){
int j;
printf("Origen: %s, Destino: %s, Duracion: %d, Precio: %.2f,
Fecha:%d/%d/%d\n",
    reg.origen, reg.destino, reg.duracion, reg.precio,
    reg.date.dia, reg.date.mes, reg.date.anio);
for ( j = 0; j < 3; j++){
    printf("Combustible de este trayecto gastado en el dia:
%d, %.2f\n", j+1, reg.combustible[j]);
}
}

void emito_vuelo(struct vuelo v[], int d){
int i,j;
for (i=0; i< d; i++){
    emite_reg(v[i]);
}
}

```

```

struct vuelo carga_reg(){
struct vuelo reg;
int j;
    printf("Ingrese el vuelo origen\n");
    gets(reg.origen);
    fflush(stdin);
    printf("Ingrese el vuelo destino\n");
    gets(reg.destino);
    fflush(stdin);
    printf("Ingrese duraci3n\n");
    scanf("%d",&reg.duracion);
    fflush(stdin);
    printf("Ingrese el precio\n");
    scanf("%f",&reg.precio);
    fflush(stdin);
    printf("Ingrese el dia\n");
    scanf("%d",&reg.date.dia);
    fflush(stdin);
    printf("Ingrese el mes\n");
    scanf("%d",&reg.date.mes);
    fflush(stdin);
    printf("Ingrese el a3o\n");
    scanf("%d",&reg.date.anio);
    fflush(stdin);
    for ( j = 0; j < 3; j++){
        printf("Ingrese la cantidad de combustible del viaje %d\n", j + 1);
        scanf("%f",&reg.combustible[j]);
        fflush(stdin);
    }
return reg; }

void carga_vuelo(struct vuelo v[], int d){
int i,j;
char op;
for (i = 0; i < d; i++)
    v[i] = carga_reg();
}

```

En este ejemplo se leen los datos de ingreso en una funci3n que retorna el registro completo a la posici3n del vector

```

#include <stdio.h> //ejemplo 11
#include <string.h>
struct fecha{
int dia, mes, anio; };
struct vuelo {
    char origen[5];
    char destino[5];
    int duracion;
    float precio;
    struct fecha date;
    float combustible[3]; };

// void emito_vuelo(struct vuelo v[], int d){ }
void modif_varias(float * pre_o_comb){
printf("Ingrese el nuevo valor:\n");
scanf("%f",pre_o_comb);
fflush(stdin);
printf("Campo modificado...\n");
}
void carga_vuelo(struct vuelo v[], int d){
int i,j;
char op;
for (i = 0; i < d; i++){
    printf("Ingrese el vuelo origen\n");
    gets(v[i].origen);
    fflush(stdin);
    printf("Ingrese el vuelo destino\n");
    gets(v[i].destino);
    fflush(stdin);
    printf("Ingrese duración\n");
    scanf("%d",&v[i].duracion);
    fflush(stdin);
    printf("Ingrese el precio\n");
    scanf("%f",&v[i].precio);
    fflush(stdin);

```

```

printf("Precio Ingresado %.2f\n", v[i].precio);
printf("\nSe equivocó en el ingreso del precio?\nPulse cualquier
teclar para seguir o '*' para modificar...\n");
    scanf("%c",&op);
    fflush(stdin);
    if (op == '*') modif_varias(&v[i].precio);
    printf("Ingrese el dia\n");
    scanf("%d",&v[i].date.dia);
    fflush(stdin);
    printf("Ingrese el mes\n");
    scanf("%d",&v[i].date.mes);
    fflush(stdin);
    printf("Ingrese el año\n");
    scanf("%d",&v[i].date.anio);
    fflush(stdin);
    for ( j = 0; j < 3; j++){
        printf("Ingrese la cantidad de combustible del viaje %d\n", j + 1);
        scanf("%f",&v[i].combustible[j]);
        fflush(stdin);
        printf("\nSe equivocó en el ingreso de la cantidad de
combustible?\nPulse cualquier teclar para seguir con el programa
o '*' para modificar...\n");
        scanf("%c",&op);
        fflush(stdin);
        if (op == '*') modif_varias(&v[i].combustible[j]);
    } } }

```

Podemos modificar un campo del struct pasando como argumento ese dato por dirección



## Ventajas que ofrecen las estructuras:

- Agrupar una serie de datos bajo un nombre. Esto nos permite tener una visión más estructurada y lógica del programa y sus datos.
- Mejor organización del código, más expresividad y en consecuencia un mejor código.
- Han resuelto el problema de almacenar datos de distintos tipos en un mismo *array*.
- Acorta el tamaño del código para hacer asignaciones de estructuras completas. Por ejemplo: `un_vuelo = otro_vuelo`, donde ambas tipo `struct vuelo` y pueden ser pasadas por parámetro.
- Puede ser pasado como parámetro sólo un campo.

# ENUMERACIONES

Existe un tipo especial de variables denominadas variables enumeradas o simplemente enumeraciones. Se caracterizan por poder adoptar valores entre una selección de constantes enteras denominadas enumeradores, cuyos valores son establecidos en el momento de la declaración del nuevo tipo. Como se ha señalado, son enteros y (una vez establecidos) de valor constante, razón por la que se los denomina también constantes de enumeración.

La declaración:

```
enum estado { MALO =0, REGULAR =1, BUENO =2, EXTRA =3 };
```

La sentencia anterior declara estado como un tipo de variable de enumeración. Los miembros de esta clase pueden adoptar los valores indicados y son representados por los nemónicos:

MALO, REGULAR, BUENO y EXTRA.

Estas cuatro constantes son los enumeradores del nuevo tipo.

La declaración:

```
enum dias { DOM, LUN, MAR, MIE, JUE, VIE, SAB } diaX;
```

establece un tipo enum al que se identifica por dias; las variables de este tipo pueden adoptar un conjunto de seis valores enteros 0, 1, 2, 3, 4, 5, 6 (enumeradores) representados por los nemónicos DOM, LUN,...,SAB. Además se define una variable enumerada diaX de este tipo.

La declaración:

```
enum modelo { ULT =-1, BW40=0, C40, BW80, C80, MON =7 };
```

define un tipo enum al que identificamos por la etiqueta modelo; las variables de este tipo pueden adoptar 6 valores (-1, 0, 1, 2, 3 y 7) que se identifican con los nemónicos: ULT, BW40, C40, BW80, C80 y MON.

```

/* Ejemplo de enum */
#include <stdio.h>
#include <stdlib.h>

enum color{
    blanco, amarillo, rojo, verde, azul, marron, negro
}

int main(){
    enum color color_coche;

    printf("Ingrese el color del coche, valores enteros de 0 a 6\n");
    scanf("%d", &color_coche);

    while ((color_coche>=0)&&(color_coche<=6)){

        switch(color_coche){
            case blanco: printf("El coche es de color blanco\n"); break;
            case amarillo: printf("El coche es de color amarillo\n"); break;
            case rojo: printf("El coche es de color rojo\n"); break;
            case verde: printf("El coche es de color verde\n"); break;
            case azul: printf("El coche es de color azul\n"); break;
            case marron: printf("El coche es de color marron\n"); break;
            case negro: printf("El coche es de color negro\n"); break;
        }
        printf("Ingrese el color del coche\n");
        scanf("%d", &color_coche);
    }
    getchar();
    return 0;
}

```

```

Ingrese el color del coche, valor entero de 0 al 6
5
El coche es de color marron
Ingrese el color del coche
4
El coche es de color azul
Ingrese el color del coche
3
El coche es de color verde
Ingrese el color del coche

```

**UNIONES**

Una unión es un tipo de datos derivado, como una estructura, con miembros que comparten el mismo espacio de almacenamiento. **Una variable de tipo unión puede contener (en momentos diferentes) objetos de diferentes tipos y tamaños.**

Las uniones proporcionan una forma de manipular diferentes tipos de datos dentro de una sola área de almacenamiento.

*En cualquier momento una unión puede contener un máximo de un objeto debido a que los miembros de una unión comparten el espacio de almacenamiento.*

Una unión se declara con el mismo formato de un struct. Primero declaramos el tipo unión y luego declaramos variables de ese tipo.

```

/* Un ejemplo de unión */
#include <stdio.h>

/* definición de la unión numero */
union numero {
    int x;
    double y;
}; /* fin de la unión numero */

int main(){
    union numero valor; /* define la variable de unión */

    valor.x = 100; /* coloca un entero dentro de la unión
    */
    printf( "%s\n%s\n%s%d\n%s%f\n\n",
        "Coloca un valor en el miembro entero",
        "e imprime ambos miembros.",
        "int: ", valor.x,
        "double:\n", valor.y );

    valor.y = 100.0; /* coloca un double dentro de la
    misma unión */
    printf( "%s\n%s\n%s%d\n%s%f\n",
        "Coloca un valor en el miembro flotante",
        "e imprime ambos miembros.",
        "int: ", valor.x,
        "double:\n", valor.y );

```

```

printf( "sizeof(valor.x):%d\n",sizeof(valor.x));
printf( "sizeof(valor.y):%d\n",sizeof(valor.y));
printf( "sizeof(valor):%d\n",sizeof(valor));
return 0; /* indica terminación exitosa */

```

```

} /* fin de main , fuente: Cómo programar en C, C++ y
Java, Deitel y Deitel*/

```

Coloca un valor en el miembro entero  
e imprime ambos miembros.

```

int:    100
double:
0.000000

```

Coloca un valor en el miembro flotante  
e imprime ambos miembros.

```

int:    0
double:
100.000000
sizeof(valor.x):4
sizeof(valor.y):8
sizeof(valor):8

```

**FIN**  
FIN