

Práctica 8. TIPOS DE DATOS ESTRUCTURADOS

Registros

OBJETIVOS:

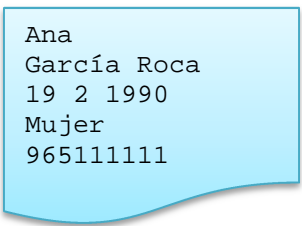
- ✓ Conocer la importancia que tiene el tipo de dato registro (struct en lenguaje C).
- ✓ Declarar, definir y saber utilizar el tipo struct.
- ✓ Diseñar e implementar tipos compuestos mediante la anidación de arrays y registros.

El término registro es un concepto que se maneja continuamente en la vida diaria; y así estamos oyendo frecuentemente frases como: oficina de registros, registros de personal, registro de empresas, registro de estudiantes, etc.

Bajo el punto de vista de la programación el concepto de **registro** es también de gran utilidad; y en principio lo definiremos (después daremos una definición más correcta) como una colección fija de información relativa a un solo objeto, donde unas veces nos referimos a la información como un todo, y otras veces nos referimos a una parte de esa información mediante un nombre. Cada una de estas partes o ítems elementales del registro se denomina **campo**.

Por ejemplo, la información siguiente relativa a una persona:

- 1) nombre
- 2) apellidos
- 3) fecha de nacimiento
- 4) sexo
- 5) número de teléfono



Ana
García Roca
19 2 1990
Mujer
9651111111

Se puede considerar como un registro de cinco componentes o campos.

Es muy importante distinguir muy bien entre los conceptos de registro y array. Diferencias que existen entre ellos:

- los campos de un **registro** pueden tener diferentes tipos de datos, mientras que los elementos de un **array** deben ser todos del mismo tipo.
- los campos de un registro deben seleccionarse por nombre, mientras que los elementos de un array se seleccionan a través de subíndices; y así por ejemplo tendrá sentido referirnos al j-ésimo elemento de un array, mientras que en el caso de un registro no podremos referirnos al j-ésimo campo por su orden o situación dentro del registro.

Como definición más formal de **registro** tendríamos que es una estructura compuesta por un número fijo de componentes, llamados **campos**, donde cada campo viene definido por su **tipo** y su identificador: el **identificador de campo**.

En C se puede definir una estructura o registro con la palabra reservada **struct**.

```
struct idRegistro {
    tipo_elemento campo1;
    tipo_elemento campo2;
    ...
    tipo_elemento campoN;
};
```

Con esta declaración asignamos una **etiqueta** (idRegistro) a la estructura y después declaramos las variables *idVar1*, *idVar2*, mediante la etiqueta anterior.

```
struct idRegistro idVar1, idVar2;
```

Otra posible forma de definir la estructura es haciendo uso de **typedef**.

```
typedef struct {
    tipo_elemento campo1;
    tipo_elemento campo2;
    ...
    tipo_elemento campoN;
} TNombre;
```

Indicamos que el identificador *TNombre* es sinónimo del especificador de estructura anterior dondequiera que ocurra. Posteriormente utilizando el identificador de tipo declaramos dos variables *idVar1*, *idVar2*.

```
TNombre idVar1, idVar2;
```

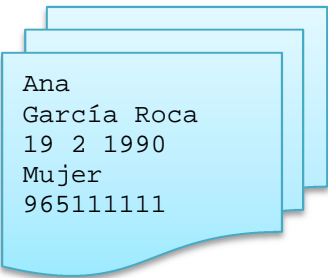
Para acceder a los campos del registro y así operar con ellos, tendremos que **utilizar el operador .** (punto), separando el nombre de la variable del nombre del campo que queremos manipular.

```
idVar1.campo1
```

Resulta muy útil la utilización de arrays de registros en los que cada elemento del array es un registro. Por ejemplo, si queremos utilizar un array de registros para almacenar información de 3 personas:

```
typedef struct{
    char nombre[10];
    char apellidos[20];
    int fecha[3];
    char sexo[8];
    char telefono[9];
}TPersona;

TPersona personas[3];
```



```
Ana
García Roca
19 2 1990
Mujer
965111111
```

Ejercicio Resuelto 1. Definir las estructuras de datos necesarias para almacenar en memoria la información de 76 alumnos. De cada alumno se almacenarán los siguientes datos:

nombre y apellidos	→	45 caracteres
domicilio	→	45 caracteres
asignatura1	→	3 caracteres
nota1	→	real
asignatura2	→	3 caracteres
nota2	→	real
.		.
.		.
.		.
asignatura10	→	3 caracteres
nota10	→	real

```
#include <iostream>
using namespace std;

// Tamaño de las cadenas de caracteres del código y del nombre
const int kTAMCOD = 3;
const int kTAMCAD = 45;

const int kNUMASIG = 10; // Número de asignaturas
const int kNUMALUM = 76; // Número de alumnos

typedef struct {
    char asig[kTAMCOD];
    float nota;
} TAsignatura;

typedef struct {
    char nombre[kTAMCAD];
    char domicilio[kTAMCAD];
    TAsignatura notas[kNUMASIG];
} TFichaAlumno;

// Tipo de datos que define un array de fichas de alumnos
typedef TFichaAlumno TAlumnos[kNUMALUM];
```

Ejercicio Resuelto 2. Realizar un programa en C que guarde información de 30 alumnos. De cada alumno leeremos su nombre, su fecha de nacimiento, su fecha de ingreso en la escuela y su nota media:

```
#include <iostream>
using namespace std;

const int kTAMCAD = 45;
const int kALUMNOS = 30;

// Estructura fecha
typedef struct {
    int dia, mes, anyo;
} TFecha;

// Estructura ficha de alumnos
typedef struct {
    char nombre[kTAMCAD];
    TFecha fechaNac;
    TFecha fechaIngreso;
```

```
        int    notaMedia;
    } TFichaAlumno;

// Declaración del tipo TAlumnos. Será un array de fichas de alumnos
typedef TFichaAlumno TAlumnos[kALUMNOS];

// Cabecera de las funciones.
void leeAlumnos(TAlumnos alumnos);

//Programa principal
main()
{
    // alumnos será el array de fichas de alumnos
    TAlumnos alumnos;

    leeAlumnos(alumnos);
}

// Función que nos permite limpiar el buffer de entrada.
// Lo utilizaremos después de leer un número entero,
// para evitar que quede el retorno de carro en el buffer.
void limpiarBuffer()
{
    while(cin.get() != '\n');
}

// Función para leer una fecha
void leeFecha(TFecha &fecha)
{
    cout << "Introduzca el día: ";
    cin >> fecha.dia;
    cout << "Introduzca el mes: ";
    cin >> fecha.mes;
    cout << "Introduzca el año: ";
    cin >> fecha.anyo;
    limpiarBuffer();
}

// Función para leer la ficha de un alumno.
void leeFichaAlumno(TFichaAlumno &fichaAlumno)
{
    cout << "Introduzca el nombre del alumno: ";
    cin.getline(fichaAlumno.nombre, kTAMCAD);

    cout << "Introduzca la fecha de nacimiento\n";
    leeFecha(fichaAlumno.fechaNac);

    cout << "Introduzca la fecha de Ingreso\n";
    leeFecha(fichaAlumno.fechaIngreso);

    cout << "Introduzca la nota del alumno: ";
    cin >> fichaAlumno.notaMedia;
    limpiarBuffer();
}

// Funcion para leer los datos de todos los alumnos
void leeAlumnos(TAlumnos alumnos)
{
    int i;

    for (i= 0; i < kALUMNOS; i++)
```

```

    {
        cout << "Introduzca datos del alumno " << i << "\n";
        leeFichaAlumno(alumnos[i]);
    }
}

```

Ejercicio Resuelto 3. Realizar en C un subprograma que permita ordenar la estructura de datos del ejercicio resuelto 2 por notas de menor a mayor. Realizar otro subprograma que permita ordenar dicha estructura por orden alfabético, utilizando funciones de cadenas vistas en la práctica anterior.

```

// Subprograma para ordenar la lista de alumnos por notas.
// Utilizamos el algoritmo de selección directa.
void ordenaAlumnosporNota(TAlumnos alumnos)
{
    int i, j, minimo, posicion_minimo;
    TFichaAlumno fichaAlumno;

    for (i = 0; i < kALUMNOS-1; i++)
    {
        posicion_minimo = i;
        for (j=i+1; j < kALUMNOS; j++)
        {
            if (alumnos[j].notaMedia < alumnos[posicion_minimo].notaMedia)
                posicion_minimo = j;
        }

        fichaAlumno = alumnos[posicion_minimo];
        alumnos[posicion_minimo] = alumnos[i];
        alumnos[i] = fichaAlumno;
    }
}

// Subprograma para ordenar la lista de alumnos por orden alfabético.
// Utilizamos el algoritmo de selección directa.
void ordenaAlumnosporOrdenAlfabetico(TAlumnos alumnos)
{
    int i, j, minimo, posicion_minimo;
    TFichaAlumno fichaAlumno;

    for (i = 0; i < kALUMNOS-1; i++)
    {
        posicion_minimo = i;
        for (j=i+1; j < kALUMNOS; j++)
        {
            if (strcmp(alumnos[j].nombre, alumnos[posicion_minimo].nombre) < 0)
                posicion_minimo = j;
        }

        fichaAlumno = alumnos[posicion_minimo];
        alumnos[posicion_minimo] = alumnos[i];
        alumnos[i] = fichaAlumno;
    }
}

// Nota: para usar la función strcmp habrá que hacer un include de la
librería <string.h>

```

Ejercicio Resuelto 4. Implementa un programa en lenguaje C que permita gestionar las pociones de un taller de magia. ¡Los magos están desesperados con tantas pociones! Debes ayudarlos para que no se equivoquen. Se desea almacenar para cada poción:

- Número de la poción: se rellenará automáticamente empezando en 0.
- Utilidad de la poción: “amor”, “exámenes”, “salud”, etc.
- Ingredientes que lleva: puede contener un número variable de ingredientes, como máximo 4, los cuales se codificarán mediante un identificador entero, de la siguiente forma:

Identificador	Ingrediente
1	Bigote de gato
2	Escama de dragón
3	Diente de diablo
4	Veneno de serpiente

- Cantidad de la poción que hay en el frasco (medida en ml).
- Si se ha terminado la poción del frasco o no.

El programa tendrá un menú con las siguientes opciones:

1. **Introducir una nueva poción.** El almacén de los brujos es reducido, por lo que como máximo podrán almacenar 100 pociones. Características:
 - a. El número de poción no se pedirá al usuario, se escribirá automáticamente (basándote en la posición del array), empezando en 0.
 - b. Inicialmente una poción está como “no terminada”, como es lógico.
2. **Utilizar poción,** sabiendo su número, basándonos en:
 - a. Ten en cuenta que el número de poción está relacionado con la posición en el array, no es necesario que la busques por todo el array.
 - b. Se pedirá la cantidad que se desea utilizar (siempre que sea menor que la cantidad que queda en el frasco) y se decrementará del frasco. Si se llegase a terminar la poción, marcar una poción como “terminada”.
3. **Listar las pociones que todavía quedan** (las que no se han terminado).
4. **Finalizar la ejecución del programa.**

La ejecución del programa consistirá en ir seleccionando cualquiera de las 3 primeras opciones del menú, en cualquier orden y número de veces, hasta que se elija la opción 4, en cuyo caso el programa finalizará.

```
#include <iostream>
#include <string.h>
using namespace std;

//*****
// DECLARACION DE CONSTANTES
//*****
const int KMAX=25;
const int KPOC=200; //máximo de pociones
const int KTAM=4; //Ingredientes disponibles

//*****
// DECLARACION DE TIPOS
//*****
typedef char TCadena[KMAX];
typedef int TVector[KTAM];

//Registro que representa la información de una poción
typedef struct{
    int cod;
    TCadena util;
    int numIngr;
    TVector ingr;
    int cantidad;
```

```

    bool terminada;
}TPocion;

//Array de registros para almacenar la información de 200 pociones
//como máximo
typedef TPocion TAlmacen[KPOC];

//*****
// PROTOTIPOS DE LOS MODULOS
//*****
void rellenaPoc(TAlmacen pociones, int &pos);
void limpiabuffer();
int menu();
void listado(TAlmacen pociones, int pos);
void muestrapocion(TPocion pocion);
void utilizaPocion(TAlmacen pociones, int pos);

/*****
main()
*****/
main(){
    TAlmacen pociones;
    int pos, op;

    pos=0;
    cout << "TALLER DE MAGIA\n\n";
    do{
        op=menu();
        switch (op){
            case 1: rellenaPoc(pociones, pos);
                    break;
            case 2: utilizaPocion(pociones, pos);
                    break;
            case 3: listado(pociones, pos);
                    break;
        }
    }while (op!=4);
}

/*****
MODULO QUE SE CORRESPONDE CON LA OPCION 2: UTILIZACION DE UNA POCION
*****/
void utilizaPocion(TAlmacen pociones, int pos){
    int i, num, cantidad;

    do{
        cout << "Numero de pocion ";
        cin >> num;
    }while (num>pos);

    do{ //Hay que validar que la cantidad sea menor o igual que la
disponible
        cout << "Cantidad a utilizar (max " << pociones[num].cantidad <<
") ";
        cin >> cantidad;
        if (cantidad > pociones[num].cantidad)
            cout << "\nNo hay pocion suficiente\n";
    }while (cantidad >pociones[num].cantidad);

    pociones[num].cantidad=pociones[num].cantidad-cantidad;
}

```

```

    if (pociones[num].cantidad==0){
        cout << "\n;La pocion "<< pos << "se ha terminado!\n";
        pociones[num].terminada=true;
    }
}

/*****
MODULO QUE SE CORRESPONDE CON LA OPCION 1: INTRODUCCIÓN DE UNA POCION
El parámetro pos se pasa por referencia para controlar la posición en
la que hay que introducir la poción. Cuando se introduce una poción
hay que incrementar pos.
*****/
void rellenaPoc(TAlmacen pociones, int &pos){
    int i;

    pociones[pos].cod=pos;
    cout << "\nIntroduce la utilidad (amor, exámenes, salud, etc) ";
    cin.getline(pociones[pos].util, KMAX);

    cout << "\nIntroduce el numero de ingredientes (max 4) ";
    cin >> pociones[pos].numIngr;
    limpiabuffer();
    cout << "Ingredientes disponibles: 1: Bigote de gato, 2. Escama de
dragon, 3. Diente de diablo, 4. Veneno de serpiente" << endl;
    for (i=0; i<pociones[pos].numIngr; i++){
        do{ //Hay que validar que el ingrediente esté entre 1 y 4
            cout << "Introduce ingrediente (1..4): ";
            cin >> pociones[pos].ingr[i];
            if (pociones[pos].ingr[i]<1 || pociones[pos].ingr[i]>4)
                cout << "\nNo existe ese ingrediente\n";
        }while (pociones[pos].ingr[i]<1 || pociones[pos].ingr[i]>4);
    }
    cout << "\nIntroduce cantidad ";
    cin >> pociones[pos].cantidad;
    limpiabuffer();
    pociones[pos].terminada=false;
    pos++;
}

/*****
LIMPIAR BUFFER
*****/
void limpiabuffer(){
    while (cin.get() != '\n');
}

/*****
MODULO PARA MOSTRAR EL MENU DE OPCIONES
*****/
int menu(){
    int op;

    cout << "\nMENU DE OPCIONES\n";
    cout << "1. Introducir pocion\n";
    cout << "2. Utilizar pocion\n";
    cout << "3. Listar pociones\n";
    cout << "4. Salir\n";
    do{
        cout << "Introduce opcion ";
        cin >> op;
        limpiabuffer();
    }
}

```



```

    }while (op<1 || op>4);
    return op;
}

//*****
// MODULO PARA MOSTRAR EL LISTADO DE POCIONES
//*****
void listado(TAlmacen pociones, int pos){
    int i;

    cout << "LISTADO DE POCIONES\n";
    for (i=0; i<pos; i++){
        if (pociones[i].terminada==false)
            muestrapocion(pociones[i]);
    }
}

//*****
// MODULO PARA MOSTRAR LOS DATOS DE UNA POCION
//*****
void muestrapocion(TPocion pocion){
    int j;

    cout << "*****\n";
    cout << "Numero: " << pocion.cod << endl;
    cout << "Utilidad: " << pocion.util << endl;
    cout << "Ingredientes: ";
    for (j=0; j<pocion.numIngr; j++){
        switch (pocion.ingr[j]){
            case 1: cout << "Bigote de gato ";
                    break;
            case 2: cout << "Escama de dragon ";
                    break;
            case 3: cout << "Diente de diablo ";
                    break;
            case 4: cout << "Veneno de serpiente ";
                    break;
        }
    }
    cout << "\nCantidad: " << pocion.cantidad;
    cout << "\n*****\n";
}

```