

Tipos de Datos II. Registros

Muchos problemas requieren trabajar con una serie de datos de diferente naturaleza que se refieren o forman parte de un mismo concepto.

Concepto de registro y campo

Un **registro** es una estructura de datos en la que se almacena una colección finita de elementos, no necesariamente homogéneos (no tienen por qué ser todos del mismo tipo). Por ejemplo la ficha de un cliente. Se dispone para un mismo cliente de una serie de datos tales como el nombre, dirección, código postal, población, provincia, etc., que aparte de ser ítems diferenciados entre sí por la naturaleza de su contenido, traducidos a un lenguaje de programación necesitan de un tipo de datos específico para cada uno. Cada uno de estos datos sería un miembro de un único concepto: la ficha de cliente.

Se puede tener tantos registros como sean necesarios; si se tiene una base de datos de mil clientes, físicamente, en el fichero, habría almacenados mil registros de tipo ficha de cliente.

Cada uno de estos registros tendrá el mismo conjunto de datos rellenados, es decir, los datos personales de cada cliente como nombre, DNI, etc. Cada uno de ellos es un **campo**. Por lo tanto, un registro contiene uno o más campos.

Cada campo dentro de un registro puede estar definido en base a cualquier tipo de dato, incluido el tipo registro. Es decir, un campo puede ser de tipo simple como los enteros o los reales, o de tipo estructurado como los vectores o los registros, y cada uno del tamaño que requiera.

Es evidente que para el campo DNI se necesitan únicamente 9 caracteres, pues para todo cliente de nacionalidad española ése es el número de caracteres que ocupa tal dato. Para datos como el nombre o el domicilio ya es más difícil estimar cuales van a ser las necesidades, puesto que no es lo mismo “Jorge” que “Recesvinto”. Hay que promediar y calcular cual va a ser el tamaño rentable de este campo, que ni desperdicie mucho espacio en blancos ni obligue a abreviar la mayoría de los nombres.

Registros en C

En el lenguaje de programación C, el concepto de registro se identifica con la estructura (`struct`). Una estructura en C es una agrupación de variables bajo un mismo nombre, donde cada variable o campo, es un miembro de la estructura.

Definición de una estructura

El primer paso en la utilización de registros es definir el tipo concreto de estructura. La sintaxis es la siguiente:

```
typedef struct {  
    tipo1 nombreCampo1;  
    tipo2 nombreCampo2;  
    ...  
};
```

```

    tipoN nombreCampoN;
}nombreEstructura;

```

El registro puede tener tantos campos como sea necesario, cada uno con su tipo de datos correspondiente. Ejemplo de registro para almacenar datos de un cliente

```

typedef struct {
    char nombre[50];
    char domicilio[70];
    char codigoPostal[5];
    char localidad[20];
    char provincia[20];
    char dni[10];
    int edad;
    char telefono[10];
}TCliente;

```

La estructura tendrá un tamaño en tiempo de compilación igual a la suma de los tamaños de cada una de las variables declaradas dentro de ella. Nótese que se trata de un tipo de datos definido por el programador. Para poder trabajar con él se deben declarar las variables necesarias de tipo TCliente. Para declarar dos variables de este tipo de datos:

```

TCliente cli1, cli2;


```

Los identificadores de variables miembro de una estructura son locales a ésta, lo que permite que se puedan repetir en estructuras diferentes, aunque no en la misma.

```

typedef struct {
    char nombre[15];
    int edad;
    float v1, v2;
    long float v1,x1;
}TMiRegistro;

```



Procesamiento de registros en C

A la hora de trabajar con un registro, la tarea principal es manejar los campos individuales dentro de cada variable estructura. Se necesita un mecanismo de acceso a los miembros que permita asignarles valores y examinar su contenido.

Referencia a un campo

Para hacer referencia a uno de los campos de una estructura se utiliza el identificador del registro seguido de un punto y del identificador del campo correspondiente. Por ejemplo, para acceder al campo edad de la variable cli1 del tipo TCliente definido anteriormente:

```

cli1.edad;

```

- ✓ ¿Qué hace el siguiente fragmento de código?

```
typedef struct{
    int dia;
    int mes;
    int anyo;
}TFecha;

int main(){
    TFecha fec;

    fec.dia=12;
    fec.mes=9;
    fec.anyo=2017;
}
```

Define un tipo de estructura llamada TFecha, en el main se declara una variable de dicho tipo llamada fec y se le asigna valor a sus tres campos.

- ✓ Si en lugar de asignarle valor a fec directamente en código, queremos pedirle el valor al usuario, ¿cómo se haría?

```
cout << "Introduce el día: ";
cin >> fec.dia;
cout << "Introduce el mes: ";
cin >> fec.mes;
cout << "Introduce el año: ";
cin >> fec.anyo;
```

El punto es un operador de máxima precedencia por lo que será prioritario sobre cualquier otro tipo de operador. Si la estructura contuviera un campo de tipo registro (con otros subcampos asociados) se utiliza una secuencia de operadores punto.

Registros anidados

Un campo de un registro puede a su vez ser otro registro. El siguiente ejemplo muestra un registro en el que uno de los campos del registro TSocio es a su vez un registro formado por dos campos:

```
typedef struct {
    int numero;
    char letra;
}TNif;

typedef struct {
    TNif nif;
    char nombre[30];
}TSocio;
```

- ✓ Dados los tipos definidos arriba (TNif, TSocio) y la siguiente declaración de variables:

```
int main(){

    TSocio soc;

}
```

¿Cómo se puede asignar la letra G, el número 21341200 y el nombre Pepe a la variable soc?

```
soc.nif.letra='G';
soc.nif.numero=21341200;
strcpy(soc.nombre, "Pepe");
```

- ✓ Dada la siguiente definición de un registro

```
typedef struct{
    char autor[15];
    char titulo[30];
    int edad;
    bool prestado;
}TLibro;
```

¿Qué muestra en pantalla este fragmento de código?

```
cout << TLibro.edad;
```

Nada porque da un error de compilación ya que TLibro no es una variable, es un tipo. Hay que declarar una variable de dicho tipo: TLibro reg; asignarle algún valor reg.edad=30; y luego ya se puede mostrar su valor en pantalla cout << reg.edad;

- ✓ Dado el siguiente código

```
typedef struct{
    char autor[20];
    char titulo[30];
    bool prestado;
}TLibro;
```

```
int main(){
    TLibro lib;
```

¿Qué hace la instrucción strcpy(lib.autor, "Pedro García Soto")

Asigna el valor Pedro García Soto al campo autor del registro lib

El array de estructuras

Una primera aproximación al proceso de estructuras es el uso de un array de registros. Suponer una empresa que posee cien clientes: se puede declarar una variable de tipo vector donde cada una de sus variables fuera una ficha de un cliente en particular.

```
TCliente listaClientes[100];
```

Se han declarado 100 registros de cliente agrupados en forma de array bajo el nombre listaClientes.

Otro ejemplo en este caso de un array de productos:

```
typedef struct {
    int    codigo;
    float  precio;
}Tproducto;

int main(){
    TProducto prods[50];
}
```

Para acceder a un campo de un registro que está incluido en un array hay que utilizar el nombre del vector, seguido de unos corchetes con la posición del registro al que se quiere acceder, el operador punto y el

nombre del campo. Por ejemplo, para acceder al campo edad del cliente que ocupa la posición 20 en el array de clientes definido anteriormente:

```
listaClientes[20].edad
```

Para acceder al nombre del cliente número 77 de la lista de clientes:

```
listaClientes[77].nombre
```

- ✓ Dada el siguiente fragmento de código

```
typedef struct {  
    int codigo;  
    char nombre[15];  
    float precioMes[12];  
} TProducto;  
  
int main(){  
    TProducto prods[50];  
}
```

¿Cómo se le puede asignar el valor 123 al código del tercer producto?

```
prods[2].codigo=123;
```

¿Y asignar el valor 45 al precio en el mes de agosto del quinto producto?

```
prods[4].precioMes[7]=45;
```

¿Y asignar el valor 100 a los 12 precios del quinto producto?

```
for (i=0; i<12; i++)  
    prods[4].precioMes[i]=100;
```

¿Y asignar el valor 125 a los 12 precios de todos los productos?

```
for (j=0; j<49; j++)  
    for (i=0; i<12; i++)  
        prods[j].precioMes[i]=125;
```

Trabajando con registros

En C se está permitida la asignación directa entre variables estructura:

```
typedef struct {  
    int    codigo;  
    float  precio;  
} TProducto;  
  
int main(){  
    TProducto p1, p2;  
  
    p1.codigo = 3;  
    p1.precio = 34.8;  
    p2 = p1;  
}
```

Registros como parámetros: Paso de estructuras a módulos

Hay que distinguir primero entre miembros de estructura y una estructura completa. A la hora de pasar un campo de una estructura como argumento de una función, como variable que es, se puede hacer por referencia o por valor.

Pasando un campo por valor

Dado el siguiente tipo de registro definido:

```
typedef struct {
    bool prestado;
    char autor [30];
    char titulo [50];
} TLibro ;

void miPrueba(bool); //prototipo
```

En el siguiente ejemplo hay creada una variable llamada libro de tipo TLibro. El módulo miPrueba recibe como parámetro un campo del registro:

```
int main(){
    TLibro libro;
    ...
    miPrueba(libro.prestado);
    ...
}
```

Pasando un campo por referencia

En el siguiente ejemplo se pasa un campo del registro por referencia:

```
void miPrueba2(bool &); //prototipo

int main(){
    TLibro libro;
    ...
    miPrueba2(libro.prestado);
    ...
}
```

Pasando una estructura

Aunque lo mismo se puede aplicar a las estructuras completas, el paso por valor está desaconsejado debido al sistema que utiliza el compilador. El paso por valor implica la copia de la variable a transmitir en la zona de variables de la función, por lo que la eficiencia en ocupación de memoria y tiempo del procedimiento de apilar y desapilar copias de estructuras completas (que poseen varios campos que pueden ser más o menos complejos) puede resentirse alarmantemente según los casos. Si se pretende optimizar la ejecución del programa lo mejor es optar por el paso por referencia, teniendo cuidado de no modificar el contenido de la estructura.

Por valor

```
void otraPrueba (TLibro );

int main(){
    TLibro libro;
    ...
}
```

```
otraPrueba(libro);
...
```

Por referencia

```
void otraPrueba (TLibro &);

int main(){
    TLibro libro:
    ...
    otraPrueba(libro);
    ...
}
```

El estándar ANSI reconoce la posibilidad de devolver una estructura completa como resultado de una función mediante la sentencia `return()`.

Ejemplo 1 de programa empleando registros

- Diseña la estructura de datos adecuada para representar 20 productos teniendo en cuenta que de cada producto interesa almacenar código, precio, nombre y unidades.
- Implementa el módulo que rellena los datos de los productos
- Diseña una función que calcule el valor monetario de los 20 productos almacenados.

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

const int KMAX=20;

typedef char TCadena[15];

typedef struct{
    int cod;
    float precio;
    TCadena nombre;
    int unidades;
}TPProducto;

typedef TPProducto TAlmacen[KMAX];

void rellena (TAlmacen );
void rellenaProducto (TPProducto &);
float calculaValor (TAlmacen );

//Introduce los datos de los 20 productos
void rellena (TAlmacen alm){
    int i;

    for (i=0;i<KMAX; i++)
        rellenaProducto(alm[i]);
}

//Pide al usuario los datos de un producto
void rellenaProducto (TPProducto &prod){
    cout << "Introduce el código ";
    cin >> prod.cod;
    cout << "Introduce el precio ";
    cin >> prod.precio;
```

```

    cout << "Introduce el nombre ";
    cin >> prod.nombre;
    cout << "Introduce las unidades ";
    cin >> prod.unidades;
}

//Calcula el valor de los productos almacenados
float calculaValor (TAlmacen alm){
    int i;
    float valor;

    for (i=0;i<KMAX; i++)
        valor = valor + alm[i].precio * alm[i].unidades;

    return (valor);
}

int main(){
    TAlmacen almacen;

    rellena(almacen);
    cout << "Valor de los productos: " << calculaValor(almacen);
    return 0;
}

```

Ejemplo 2 de programa empleando registros

- Diseña la estructura de datos adecuada para representar datos de 20 clientes teniendo en cuenta que de cada cliente interesa almacenar DNI, nombre y email.
- Implementa el módulo que permite dar de alta un cliente.
- Implementa el módulo que permite dar de baja un cliente
- Implementa el módulo que permite modificar el email de un cliente.

```

#include <iostream>
using namespace std;

const int KMAX=20;

typedef char TDNI[10];
typedef char TCadena[15];

typedef struct{
    TDNI dni;
    TCadena nombre;
    TCadena email;
}TCliente;

typedef TCliente TOficina[KMAX];

//Muestra el menu con las opciones del programa
char menu(){
    char op;

    do{
        cout << "1. Alta de cliente\n";
        cout << "2. Baja de cliente\n";
        cout << "3. Modificar email de un cliente\n";
    }
}

```



```
        cout << "4. Salir de programa\n";
        cout << "Opción: ";
        cin >> op;
    }while (op<'1' || op>'4');

    return (op);
}

//Pide al usuario los datos de un producto
void alta (TOficina ofi, int &num){
    int i;

    cout << "Introduce el DNI ";
    cin >> ofi[num].dni;
    cin.get();          //para limpiar el buffer
    cout << "Introduce el nombre ";
    cin.getline(ofi[num].nombre, 15);
    cout << "Introduce el email ";
    cin >> ofi[num].email;
    num++;              //se incrementa el contador de registros
}

//Elimina el registro que quiere el usuario
void baja (TOficina ofi, int &num){
    int i, k;
    TDNI dni;

    cout << "Introduce el DNI del cliente a dar de baja ";
    cin >> dni;
    //buscar ese DNI
    i=0;
    while (i<num && strcmp(dni, ofi[i].dni)!=0)
        i++;
    if (i<num){          //lo ha encontrado
        //Hay que desplazar los registros
        if (i<num-1){    // no está en la última posición
            for (k=i;k<num-1; k++)
                ofi[k]=ofi[k+1];
        }
        num--;          //se decrementa el contador de registros
    }
    else                 // no lo ha encontrado
        cout << "CLIENTE NO EXISTENTE\n";
}

//Modifica el registro que quiere el usuario
void modifica (TOficina ofi, int num){
    int i;
    TDNI dni;

    cout << "Introduce el DNI del cliente a modificar ";
    cin >> dni;
    //buscar ese DNI
    i=0;
    while (i<num && strcmp(dni, ofi[i].dni)!=0)
        i++;
    if (i<num){ //lo ha encontrado
        cout << "Introduce el nuevo email ";
        cin >> ofi[i].email;
```

```
        cout << "Datos de cliente modificado\n";
        cout << "DNI " << ofi[i].dni << endl;
        cout << "Nombre " << ofi[i].nombre << endl;
        cout << "email " << ofi[i].email << endl;
    }
    else // no lo ha encontrado
        cout << "CLIENTE NO EXISTENTE\n";
}

int main(){
    TOficina ofi;
    int num;
    char op;

    num=0;
    do{
        op=menu();
        switch (op){
            case '1': alta (ofi, num);
                       break;
            case '2': baja (ofi, num);
                       break;
            case '3': modifica(ofi, num);
                       break;
            case '4': cout << "FIN DE PROGRAMA\n";
                       }
        }while (op!='4');

    return 0;
}
```