Estructuras

Una estructura es una forma de agrupar un conjunto de datos de distinto tipo bajo el mismo nombre. Por ejemplo, supongamos que queremos guardar los datos correspondientes a un producto.

Producto:
- Código de producto (string [6])
- Descripción (string [30])
- Precio unitario (float)
- Stock (int)
- Stock mínimo (int)
- Rubro (char)
- Estado (int)

Supongamos que necesitamos cargar los datos de 5 productos y resolver una serie de tareas a partir de esa información.
Necesitaríamos una matriz de char de 6x5 para el código, otra de 30x5 para la descripción, arrays de int para el stock, el stock mínimo y el estado, uno de float para el precio y un array de char para el rubro. Sin embargo, gracias a las estructuras lo único que necesitaremos es un array de *producto* de cinco elementos.
En principio, vamos a resolverlo como si no contáramos con una computadora. Necesitaríamos un papel y un lápiz y hacer una tabla como la siguiente:

Código	Descripción	Precio unitario	Stock	Stock mínimo	Rubro	Estado
11111	Pen drive 8GB	$ 125,00	20	5	a	1
22222	Modem-router ADSL2	$ 295,00	5	1	a	1
33333	DVD+R 16x	$ 1,35	150	20	a	1
44444	Calculadora científica	$ 70,00	10	1	b	1
55555	Pila recargable AA	$ 18,00	3	5	b	1

Tabla 1: Modelo de datos

En cada columna se pondría una característica distinta de cada producto y cada fila (exceptuando la primera que no existiría) representaría un registro completo.
Para resolverlo de esta manera tendremos que crear nuestro propio "tipo de dato" que se adecúe a la información que tenemos que almacenar, para ello utilizamos el *struct*.

```
struct Producto{
        char codigo[6];
        char descripcion[30];
        float pu;
        int stock;
        int stockmin;
        char rubro;
        int estado;
};
```

Luego necesitamos declarar variables del tipo de dato Producto dentro de nuestras funciones para poder comenzar a utilizarlos.

```
int main(void){
   struct Producto solo_uno, varios[5], matriz[3][3], *puntero;
   return 0;
}
```

En main se declararon cuatro variables del tipo de dato Producto:

- solo_uno, es una variable simple. Sólo puede almacenar un registro, es decir, un valor para cada uno de los campos de nuestra estructura (equivaldría a una fila de nuestra tabla).
- varios, es un array de 5 elementos. Aquí se podrían almacenar los datos de nuestros cinco productos.
- matriz, es algo más complejo. Ya que representa una matriz de 3 filas y 3 columnas.
- puntero, es un puntero a struct Producto.

Se observa la similitud que hay entre los tipos de datos primitivos (int, char, float...) ya que se pueden declarar punteros, arrays y matrices a partir de nuestro struct definido.
Una vez declarado un struct se utiliza de la siguiente manera:

Variable simple:

```
struct Producto p;
strcpy(p.codigo, "11111");
strcpy(p.descripcion, "Pen drive 8GB");
p.pu = 125.00;
p.stock = 20;
p.stockmin = 5;
p.rubro = 'a';
p.estado = 1;
```

Array:

```
struct Producto v[10];
strcpy(v[0].codigo, "11111");
strcpy(v[0].descripcion, "Pen drive 8GB");
v[0].pu = 125.00;
v[0].stock = 20;
v[0].stockmin = 5;
v[0].rubro = 'a';
v[0].estado = 1;
```

Puntero:

```
struct Producto v, *aptr;
aptr = &v;
strcpy(aptr->codigo, "11111");
strcpy((*aptr).descripcion, "Pen drive 8GB");
aptr->pu = 125.00;
(*aptr).stock = 20;
(*aptr).stockmin = 5;
aptr->rubro = 'a';
aptr->estado = 1;
```

Se puede observar como cuando se trata de un puntero, se pueden utilizar las variables de la siguiente forma:

```
variable_puntero->campo1;
variable_puntero->campoN;
```

```
(*variable_puntero).campo1;
(*variable_puntero).campoN;
```

Mientras que si se trata de una variable simple o un array con corchetes, se utilizan de la siguiente manera:

```
variable.campo1;
variable.campoN;
```

```
array[indice].campo1;
array[indice].campoN;
```

Paso de structs como parámetro a funciones

Al igual que con el resto de las variables, es posible pasar un struct por valor o por dirección. Está sujeto a las mismas restricciones que el resto de las variables de tipos primitivos:

- se puede pasar por valor cuando es una variable simple;
- se tiene que pasar por dirección cuando se trata de un vector, una matriz o cuando se quiere accesar a la dirección de memoria de una variable simple en la función.

Ejemplos:

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

struct Producto{
        char codigo[6];
        char descripcion[30];
        float pu;
        int stock;
        int stockmin;
        char rubro;
        int estado;
};

void mostrarProducto(struct Producto);
void cargarProducto(struct Producto *);

int main(void){
   struct Producto p;
   cargarProducto(&p);
   mostrarProducto(p);
   return 0;
}

void cargarProducto(struct Producto *prod){
   strcpy(prod->codigo, "11111");
```

```
   strcpy(prod->descripcion, "Pen drive 8GB");
   prod->pu = 125.00;
   prod->stock = 20;
   prod->stockmin = 5;
   prod->rubro = 'a';
   prod->estado = 1;
}
void mostrarProducto(struct Producto p){
   cout << "CODIGO: " << p.codigo << endl;
   cout << "DESCRIPCION: " << p.descripcion << endl;
   cout << "PRECIO: " << p.pu << endl;
   cout << "STOCK: " << p.stock << endl;
   cout << "STOCK MINIMO: " << p.stockmin << endl;
   cout << "RUBRO: " << p.rubro << endl;
   cout << "ESTADO: " << p.estado;
}
```

T04CF01.cpp

Como puede verse en el ejemplo anterior, se puede enviar a una función una variable de tipo struct por valor o por dirección. También podemos retornar de una función una variable de tipo struct o un puntero a una variable de tipo struct.

```
void funcion(struct algunaEstructura variable){
}
void funcion(struct algunaEstructura *puntero){
}
struct algunaEstructura funcion(void){
}
struct algunaEstructura * funcion(void){
}
```

Ejercicio de ejemplo:

Se dispone de una serie de datos correspondientes a cinco productos (ver Tabla 1), se pide un programa que permita cargarlos, determinar el producto más caro, mostrar un producto a partir de su código, mostrar los productos que su stock esté por debajo del stock y los productos que pertenezcan a un rubro.

```
#include <iostream>
using namespace std;
struct Producto{
        char codigo[6];
        char descripcion[30];
        float pu;
        int stock;
        int stockmin;
        char rubro;
        int estado;
};
const int CANT_PRODS = 5;
void mostrarProducto(struct Producto);
```

```
void cargarProductos(struct Producto *);
struct Producto obtenerProductoxCodigo(struct Producto *, char *);
void mostrarProductosxRubro(struct Producto *, char);
void mostrarProductoStockMinimo(struct Producto *);
struct Producto obtenerProductoMasCaro(struct Producto *);

int main(void){
   struct Producto p[CANT_PRODS], aux;
   char rubro, codigo[6];
   cargarProductos(p);
   aux = obtenerProductoMasCaro(p);
   cout << endl << "El producto mas caro es: " << aux.descripcion << " que cuesta $" << aux.pu <<
" c/u." << endl << endl;
   cout << "Productos con stock debajo del stock minimo: " << endl;
   mostrarProductoStockMinimo(p);
   cout << "Productos pertenecientes a un rubro: ";
   cout << endl << "Ingrese rubro: ";
   cin >> rubro;
   mostrarProductosxRubro(p, rubro);
   cout << "Producto por codigo: ";
   cout << endl << "Ingrese codigo: ";
   cin >> codigo;
   aux = obtenerProductoxCodigo(p, codigo);
   mostrarProducto(aux);
   return 0;
}

void cargarProductos(struct Producto *prod){
   int i;
   for(i=0; i<CANT_PRODS; i++){
      cout << "Codigo: ";
      cin >> prod[i].codigo;
      cin.ignore();
      cout << "Descripcion: ";
      cin.get(prod[i].descripcion, 30);
      cout << "Precio unitario: ";
      cin >> prod[i].pu;
      cout << "Stock: ";
      cin >> prod[i].stock;
      cout << "Stock minimo: ";
      cin >> prod[i].stockmin;
      cout << "Rubro: ";
      cin >> prod[i].rubro;
      cout << "Estado: ";
      cin >> prod[i].estado;
   }
}
void mostrarProducto(struct Producto p){
   cout << "CODIGO: " << p.codigo << endl;
   cout << "DESCRIPCION: " << p.descripcion << endl;
   cout << "PRECIO: " << p.pu << endl;
   cout << "STOCK: " << p.stock << endl;
```

```
    cout << "STOCK MINIMO: " << p.stockmin << endl;
    cout << "RUBRO: " << p.rubro << endl;
    cout << "ESTADO: " << p.estado;
}
struct Producto obtenerProductoxCodigo(struct Producto *v, char *cod){
    int i;
    for(i=0; i<CANT_PRODS; i++){
        if(strcmp(v[i].codigo, cod) == 0){
            return v[i];
        }
    }
    struct Producto err;
    return err;
}
void mostrarProductosxRubro(struct Producto *v, char r){
    int i;
    for(i=0; i<CANT_PRODS; i++){
        if(v[i].rubro == r){
            mostrarProducto(v[i]);
            cout << endl << endl;
        }
    }
}
void mostrarProductoStockMinimo(struct Producto *v){
    int i;
    for(i=0; i<CANT_PRODS; i++){
        if(v[i].stock < v[i].stockmin){
            cout << "Producto " << v[i].codigo << endl;
        }
    }
}
struct Producto obtenerProductoMasCaro(struct Producto *v){
    int i, pos=0;
    for(i=1; i<CANT_PRODS; i++){
        if(v[i].pu > v[pos].pu){
            pos = i;
        }
    }
    return v[pos];
}
```

T04CF02.cpp