### ESTRUCTURA DE DATOS

Las estructuras son datos ordenados que al contrario que los *arrays* nos permiten agrupar varios tipos de ingresos, se buscara que siempre mantengan algún tipo de relación, aunque sean de distinto tipo, permitiendo manipularlos todos juntos, usando un mismo identificador, o cada uno por separado.

Las estructuras son llamadas también muy a menudo registros, o en inglés *records*, estos datos representan los registros usados en bases de datos. Y siguiendo la misma analogía, cada objeto de una estructura se denomina a menudo campo, o *field*.

Sintaxis:

**struct** [<nombre>]

{

[<tipo> <nombre\_objeto>[,<nombre\_objeto>,...]];

} [<objeto\_estructura>[,<objeto\_estructura>,...];

El nombre de la estructura es un indicador para referirse a la estructura.

Los objetos de estructura son objetos declarados del tipo de la estructura ej.: int, string, char, etc…, y su inclusión es opcional.

En el interior de una estructura, entre las llaves, se pueden definir todos los elementos que consideremos necesarios, del mismo modo que se declaran los objetos.

Las estructuras pueden referenciarse completas, utilizando su nombre como indice, de la misma forma con la que trabajamos las variables que ya conocemos, y también se puede acceder a cada elemento definido en su interior usando el operador de selección (.), punto.

Una vez definida una estructura, se puede usar igual que cualquier otro tipo de C++. Esto significa que se pueden declarar más objetos del tipo de estructura en cualquier parte del programa.

Para ello usaremos la forma normal de declaración de objetos, es decir:

[**struct**] <nombre> <objeto\_estructura>

[,<objeto\_estructura>...];

En C++ la palabra struct es opcional en la declaración de objetos, al contrario de lo que sucede en C, en el que es obligatorio usarla.

Ejemplo:

struct Persona

{

char Nombre[65];

char Direccion[65];

int AnyoNacimiento;

} Fulanito;

Este ejemplo define la estructura ***Persona*** y declara a ***Fulanito*** como un objeto de ese tipo. Para acceder al nombre de *Fulanito*, por ejemplo, para visualizarlo, usaremos la forma:

cout << Fulanito.Nombre;

#### **Funciones en el interior de estructuras**

C++, permite incluir funciones en el interior de las estructuras. Normalmente estas funciones tienen la misión de manipular los datos incluidos en la estructura, y su uso está muy relacionado con la programación orientada a objetos.

Aunque esta característica se usa casi exclusivamente con las clases, como veremos más adelante, también puede usarse en las estructuras. De hecho, en C++, las diferencias entre estructuras y clases son muy tenues.

Dos funciones muy particulares son las de inicialización, o ***constructor***, y el ***destructor***. Veremos con más detalle estas funciones cuando asociemos las estructuras y los punteros.

El *constructor* es una función sin tipo de retorno y con el mismo nombre que la estructura. El *destructor* tiene la misma forma, salvo que el nombre va precedido el símbolo "~".

**Nota:**

para aquellos que usen un teclado español, el símbolo "~" se obtiene pulsando las teclas del teclado numérico 1, 2, 6, mientras se mantiene pulsada la tecla ALT, ([ALT]+126). También mediante la combinación [Atl Gr]+[4] y un espacio (la tecla [4] de la zona de las letras, no del teclado numérico).

Ejemplo:

struct Punto

{

int x, y;

Punto() {x = 0; y = 0;} // Constructor y Definicion de valores.

} Punto1, Punto2;

Forma 2:

struct Punto

{

int x, y;

Punto(); // Declaración del constructor

} Punto1, Punto2;

// Definición del constructor, fuera de la estructura

Punto()

{

x = 0;

y = 0;

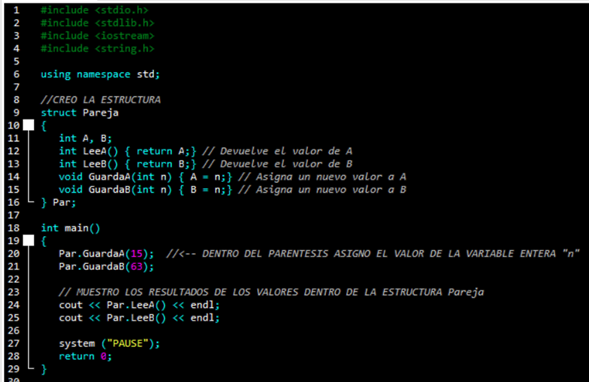
}

Si no usáramos un *constructor*, los valores de *x* e *y* para *Punto1* y *Punto2* estarían indeterminados, contendrían la "basura" que hubiese en la memoria asignada a estas estructuras durante la ejecución. Con las estructuras éste será el caso más habitual, ya que, si necesitamos usar *constructores* para asignar valores iniciales, será mucho más lógico usar clases que estructuras.

Usando *constructores* nos aseguramos los valores iniciales para los elementos de la estructura. Veremos que esto puede ser una gran ventaja, sobre todo cuando combinemos estructuras con punteros.

También podemos incluir otras funciones, que se declaran y definen como las funciones que ya conocemos.

Otro ejemplo:



En este ejemplo podemos ver cómo se define una estructura con dos campos enteros, y dos funciones para modificar y leer sus valores. El ejemplo es muy simple, pero las funciones de guardar valores se pueden elaborar para que no permitan determinados valores, o para que hagan algún tratamiento de los datos.

##### **Vectores de estructuras**

La combinación de las estructuras con los vectores también llamados arrays proporciona una potente herramienta para el almacenamiento y manipulación de datos en archivos.

Ejemplo:

struct Persona

{

char Nombre[65];

char Direccion[65];

int AnyoNacimiento;

} Plantilla[200];

Vemos en este ejemplo lo fácil que podemos declarar el *array* *Plantilla* que contiene los datos relativos a doscientas personas.

Podemos acceder a los datos de cada uno de ellos:

cout << Plantilla[43].Direccion;

O asignar los datos de un elemento de la plantilla a otro:

Plantilla[0] = Plantilla[99];

##### 

##### **Estructuras anidadas**

Tal como se vio con la sentencia “if” también se puden anidar estructuras, con lo cual se obtienen superestructuras muy elaboradas.

Ejemplo:

struct stDireccion

{

char Calle[64];

int Portal;

int Piso;

char Puerta[3];

char CodigoPostal[6];

char Poblacion[32];

};

struct stPersona

{

struct stNombre {

char Nombre[32];

char Apellidos[64];

} NombreCompleto;

stDireccion Direccion;

char Telefono[10];

};

...

En general, no es una buena práctica definir estructuras dentro de estructuras, ya que tienen un ámbito local, y para acceder a ellas se necesita hacer referencia a la estructura externa inicial, sin embargo, es importante comprender como funcionan ya que es el principio de las tablas en las bases de datos.

Por ejemplo para declarar un objeto del tipo *stNombre* hay que utilizar el operador de acceso (::):

stPersona::stNombre NombreAuxiliar;

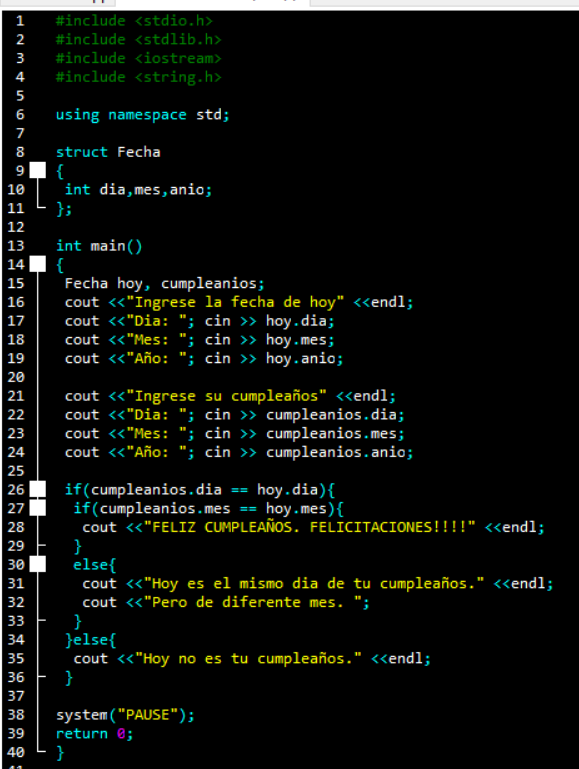
Sin embargo para declarar un objeto de tipo *stDireccion* basta con declararla:

stDireccion DireccionAuxiliar;

#### **Ejemplo Struct 1:**

Este programa utiliza la estructura para ordenar los datos de la FECHA es realmente muy sencillo de comprender y contempla los parámetros mencionados en la teoría.

En principio creamos un struct al que nombraremos como “Fecha” y que posee en su interior 3 datos del tipo entero, para completar el dia / mes y año.-



Se observa en el ejercicio que se crean los objetos “hoy” y “cumpleanios” que heredan la estructura de “fecha” (línea 15 del código).

Luego simplemente utilizamos cin para realizar los ingresos del dia, mes y anio sobre los objetos “hoy” y “cumpleanios”. Finalmente realizamos una comprobación de estructuras comparando los datos “dia” y “mes” de los objetos anteriormente mencionados para definir si hoy es o no el cumpleaños.

A este programa le quedaría bien una comprobación para limitar los ingresos por ejemplo que mes pueda ser (>=1 y <=12) y según el mes tener en consideración la cantidad de días, si quieren pueden realizarlo les será útil para el final.

#### **Ejemplo Struct 2:**

En el siguiente programa vamos a crear una estructura de datos de Alumnos y rellenaremos con ingresos para poder comprender el funcionamiento real de un struct.



Observemos como en este ejemplo:

1) Utilizamos un “define t” (línea 6 del programa) para declarar cuantos registros de estructuras utilizaremos.  
  
2) Realizamos asignaciones a los valores de reg[0] utilizando la función “strcpy” para normalizar datos del tipo char a formato string. Gracias a esto por ejemplo podemos utilizar un nombre compuesto “Eduardo” y “Cesar” (línea 21 al 24 del programa).  
  
3) Iniciamos el primer for con el valor i=1 (línea 27 del programa) ya que la asignación i=0, está realizada arriba.

#### 

#### 

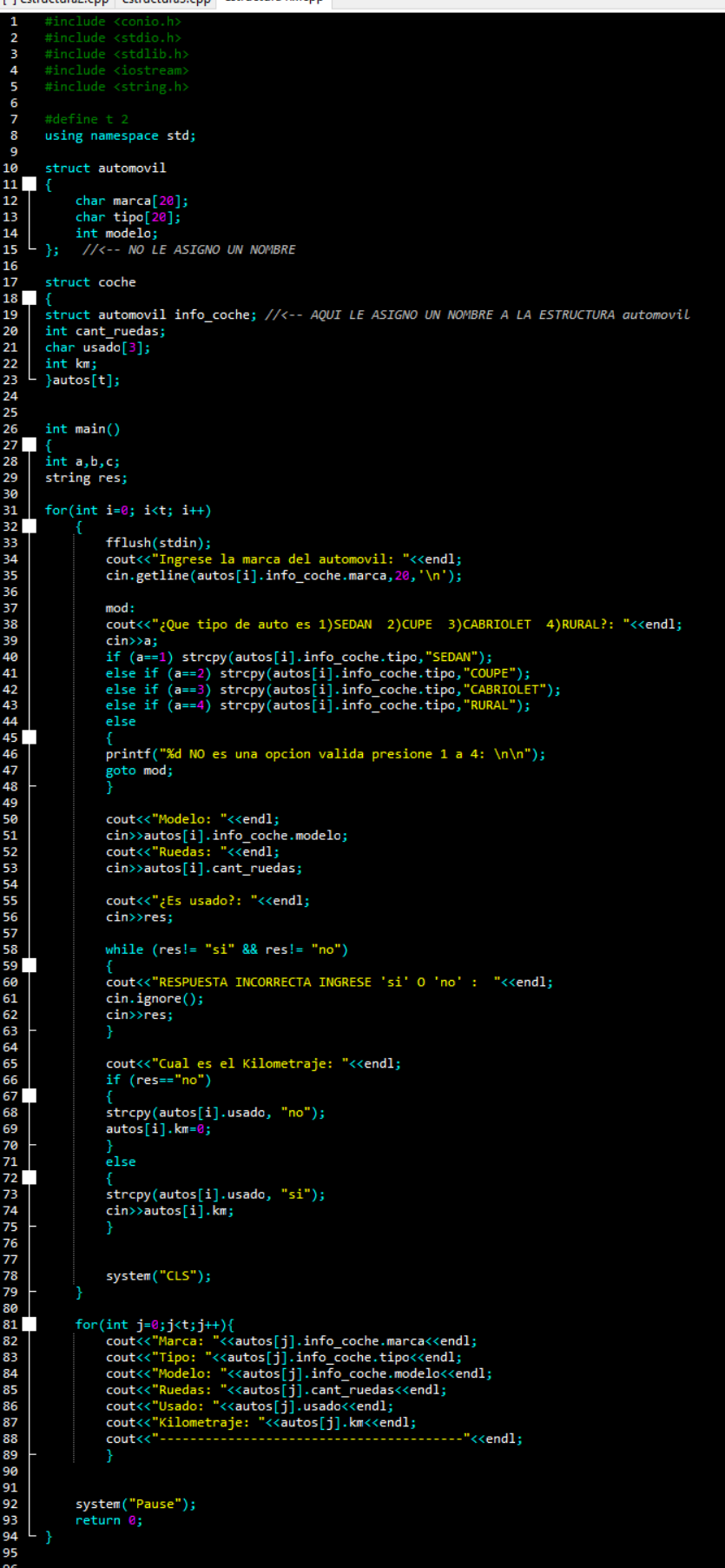
#### **Ejemplo Struct 2 BIS:**

Como todo buen programa siempre hay mejoras que se pueden realizar, en este caso utilizaremos la función cin para ingresar los datos y aprovecharemos la ventaja que tiene getline que me permite capturar el ingreso completo por teclado (esto significa tomar también los espacios entre palabras) y asignárselos a una variable. Entonces cin.getline asigna al registro reg[i].nombre y reg[i].apellido las líneas completas de texto (por ejemplo doble nombre o doble apellido) admitiendo entonces colocar nombres y apellidos compuestos de hasta 20 caracteres máximos. (línea de código 30 y 35 del programa).



#### **Ejemplo Struct 3:**

El siguiente programa utiliza un struct automóvil para completar la ficha de información de “t” cantidad de autos.



En si la interpretación del programa no representa un mayor desafío, utilizaremos una estructura anidada de “automóvil” dentro de la estructura “coche”. Si es interesante la interpretación de las líneas 40 a 43 de código, donde según la opción numérica que se seleccione (de 1 a 4) le asigna un tipo al auto. También es bueno observar como se utiliza el recurso en la línea de código 66 para verificar que si el auto “NO” es usado entonces automáticamente le asigna 0 km. y solo pregunta los kilometrajes si “SI” es usado.

F,{3fb84f89-7305-4af3-b116-5f1de43758ce}{101},3.125,3.125

Se deberia ver un resultado similar al de la imagen.

#### **Ejemplo Struct 4:**

En este último programa volveremos a utilizar las estructuras anidadas, esta vez de forma entrelazada es decir en este caso tendremos una estructura “Fecha”, una estructura “Persona (contiene fecha)” y una estructura “Empleado (contiene a persona y por lo tanto contiene a fecha)”.

Nótese algo interesante la estructura “Persona” puede tener un objeto también llamado “persona” al cual directamente se le asignan los datos de (“nombre”, edad, peso…….).

Finalmente tenemos al objeto “trabajador” que pertenece a la estructura “Empleado” y que como se observa contiene todos los datos de las estructuras (persona / fecha / empleado) así se conforma:

{“Anibal”, 19, 60, (persona) // 22, 7, 1983 (fecha) // 62830 (empleado)}

