7. Registros (Struct)

7.1. Definición y acceso a los datos

Un **registro** es una agrupación de datos, los cuales no necesariamente son del mismo tipo. Se definen con la palabra “**struct**”.

Para acceder a cada uno de los datos que forman el registro, tanto si queremos leer su valor como si queremos cambiarlo, se debe indicar el nombre de la variable y el del dato (o campo) separados por un punto:

// Introducción a C++, Nacho Cabanes

// Ejemplo 07.01:

// Registros (struct)

#include <iostream>

using namespace std;

int main()

{

struct

{

string nombre;

char inicial;

int edad;

float nota;

} persona;

persona.nombre = "Juan";

persona.inicial = 'J';

persona.edad = 20;

persona.nota = 7.5;

cout << "La edad es " << persona.edad;

return 0;

}

Como es habitual en C++, para declarar la variable hemos indicado primero el tipo de datos (struct { ...} ) y después el nombre que tendrá esa variable (persona).

También podemos declarar primero cómo van a ser nuestros registros, y más adelante definir variables de ese tipo:

// Introducción a C++, Nacho Cabanes

// Ejemplo 07.02:

// Registros (2)

#include <iostream>

#include <string>

using namespace std;

int main()

{

struct datosPersona

{

string nombre;

char inicial;

int edad;

float nota;

};

datosPersona persona;

persona.nombre = "Juan";

persona.inicial = 'J';

persona.edad = 20;

persona.nota = 7.5;

cout << "La edad es " << persona.edad;

return 0;

}

**Ejercicios propuestos:**

* **(7.1.1)** Un struct que almacene datos de una canción en formato MP3: Artista, Título, Duración (en segundos), Tamaño del fichero (en KB). Un programa debe pedir los datos de una canción al usuario, almacenarlos en dicho struct y después mostrarlos en pantalla.

7.2. Arrays de registros

Hemos guardado varios datos de una persona. Se pueden almacenar los de **varias personas** si combinamos el uso de los “struct” con las tablas (arrays) que vimos anteriormente. La sintaxis no es exactamente la misma, y tendremos que añadir la palabra "new" en el momento de reservar espacio. Por ejemplo, si queremos guardar los datos de 100 alumnos podríamos hacer:

// Introducción a C++, Nacho Cabanes

// Ejemplo 07.03:

// Array de registros

#include <iostream>

#include <string>

using namespace std;

int main()

{

struct datosPersona

{

string nombre;

char inicial;

int edad;

float nota;

};

datosPersona \*persona = new datosPersona[50];

for (int i=0; i<5; i++)

{

cout << "Dime el nombre de la persona " << i << endl;

cin >> persona[i].nombre;

}

cout << "La persona 3 es " << persona[2].nombre << endl;

return 0;

}

La inicial del primer alumno sería “alumnos[0].inicial”, y la edad del último sería “alumnos[99].edad”.

**Ejercicios propuestos:**

* **(7.2.1)** Ampliar el programa del ejercicio 7.1.1, para que almacene datos de hasta 100 canciones. Deberá tener un menú que permita las opciones: añadir una nueva canción, mostrar el título de todas las canciones, buscar la canción que contenga un cierto texto (en el artista o en el título). **Nota:** si te parece demasiado complicado, en el apartado 7.4 tienes un ejemplo parecido, desarrollado de principio a fin.
* **(7.2.2)** Un programa que permita guardar datos de "imágenes" (ficheros de ordenador que contengan fotografías o cualquier otro tipo de información gráfica). De cada imagen se debe guardar: nombre (texto), ancho en píxeles (por ejemplo 2000), alto en píxeles (por ejemplo, 3000), tamaño en Kb (por ejemplo 145,6). El programa debe ser capaz de almacenar hasta 700 imágenes (deberá avisar cuando su capacidad esté llena). Debe permitir las opciones: añadir una ficha nueva, ver todas las fichas (número y nombre de cada imagen), buscar la ficha que tenga un cierto nombre.

7.3. Estructuras anidadas

Podemos encontrarnos con un registo que tenga varios datos, y que a su vez ocurra que uno de esos datos esté formado por varios datos más sencillos. Para hacerlo desde C++, incluiríamos un “struct” dentro de otro, así:

// Introducción a C++, Nacho Cabanes

// Ejemplo 07.04:

// Registros anidados

#include <iostream>

#include <string>

using namespace std;

struct fechaNacimiento

{

int dia;

int mes;

int anyo;

};

struct datosPersona

{

string nombre;

char inicial;

struct fechaNacimiento diaDeNacimiento;

float nota;

};

int main()

{

datosPersona persona;

persona.nombre = "Ignacio";

persona.inicial = 'I';

persona.diaDeNacimiento.mes = 8;

persona.nota = 7.5;

cout << "La nota es " << persona.nota;

return 0;

}

**Ejercicios propuestos:**

* **(7.3.1)** Ampliar el programa del ejercicio 7.2.1, para que el campo duración se almacene como minutos y segundos, usando un struct anidado que contenga a su vez estos dos campos.

7.4. Ejemplo completo

Vamos a hacer un ejemplo completo que use tablas (“arrays”), registros (“struct”) y que además manipule cadenas.

La idea va a ser la siguiente: Crearemos un programa que pueda almacenar datos de hasta 1000 ficheros (archivos de ordenador). Para cada fichero, debe guardar los siguientes datos: Nombre del fichero (max 40 letras), Tamaño (en KB, número de 0 a 2.000.000.000). El programa mostrará un menú que permita al usuario las siguientes operaciones:

1- Añadir datos de un nuevo fichero  
2- Mostrar los nombres de todos los ficheros almacenados  
3- Mostrar ficheros que sean de más de un cierto tamaño (por ejemplo, 2000 KB).  
4- Ver todos los datos de un cierto fichero (a partir de su nombre)  
5- Salir de la aplicación (como todavía no sabemos almacenar los datos, éstos se perderán).

No debería resultar difícil. Vamos a ver directamente una de las formas en que se podría plantear y luego comentaremos alguna de las mejoras que se podría (incluso se debería) hacer.

Una opción que podemos a tomar para resolver este problema es la de contar el número de fichas que tenemos almacenadas, y así podremos añadir de una en una. Si tenemos 0 fichas, deberemos almacenar la siguiente (la primera) en la posición 0; si tenemos dos fichas, serán la 0 y la 1, luego añadiremos en la posición 2; en general, si tenemos “n” fichas, añadiremos cada nueva ficha en la posición “n”. Por otra parte, para revisar todas las fichas, recorreremos desde la posición 0 hasta la n-1, haciendo algo como

for (i=0; i<=n-1; i++) { ... más órdenes ...}

o bien algo como

for (i=0; i<n; i++) { ... más órdenes ...}

El resto del programa no es difícil: sabemos leer y comparar textos y números. Sólo haremos dos consideraciones:

* No se comportará correctamente si los textos (nombre del fichero, por ejemplo) contienen espacios, porque aún no sabemos leer textos con espacios.
* Hemos limitado el número de fichas a 1000, así que, si nos piden añadir, deberíamos asegurarnos antes de que todavía tenemos hueco disponible.

Con todo esto, nuestro fuente quedaría así:

// Introducción a C++, Nacho Cabanes

// Ejemplo 07.05:

// Array con muchos struct y menu para manejarla

#include <iostream>

#include <string>

using namespace std;

struct tipoDatos

{

string nombreFich; // Nombre del fichero

long tamanyo; // El tamaño en bytes

};

int numeroFichas=0; // Número de fichas que ya tenemos

int i; // Para bucles

int opcion; // La opcion del menu que elija el usuario

string textoTemporal; // Para pedir datos al usuario

int numeroTemporal;

int main()

{

tipoDatos \*fichas = new tipoDatos[1000];

do

{

// Menu principal

cout << endl;

cout << "Escoja una opción:" << endl;

cout << "1.- Añadir datos de un nuevo fichero" << endl;

cout << "2.- Mostrar los nombres de todos los ficheros" << endl;

cout << "3.- Mostrar ficheros que sean de mas de un cierto tamaño" << endl;

cout << "4.- Ver datos de un fichero" << endl;

cout << "5.- Salir" << endl;

cin >> opcion;

// Hacemos una cosa u otra según la opción escogida

switch(opcion)

{

case 1: // Añadir un dato nuevo

if (numeroFichas < 1000) // Si queda hueco

{

cout << "Introduce el nombre del fichero: ";

cin >> fichas[numeroFichas].nombreFich;

cout << "Introduce el tamaño en KB: ";

cin >> fichas[numeroFichas].tamanyo;

numeroFichas++; // Y tenemos una ficha más

}

else // Si no hay hueco para más fichas, avisamos

cout << "Máximo de fichas alcanzado (1000)!" << endl;

break;

case 2: // Mostrar todos

for (i=0; i<numeroFichas; i++)

cout << "Nombre: " << fichas[i].nombreFich

<< "; Tamaño: " << fichas[i].tamanyo

<< "Kb" << endl;

break;

case 3: // Mostrar según el tamaño

cout << "¿A partir de que tamaño quieres que te muestre? ";

cin >> numeroTemporal;

for (i=0; i<numeroFichas; i++)

if (fichas[i].tamanyo >= numeroTemporal)

cout << "Nombre: " << fichas[i].nombreFich

<< "; Tamaño: " << fichas[i].tamanyo

<< " Kb" << endl;

break;

case 4: // Ver todos los datos (pocos) de un fichero

cout << "¿De qué fichero quieres ver todos los datos?";

cin >> textoTemporal;

for (i=0; i<numeroFichas; i++)

if (fichas[i].nombreFich == textoTemporal)

cout << "Nombre: " << fichas[i].nombreFich

<< "; Tamaño: " << fichas[i].tamanyo

<< " Kb" << endl;

break;

case 5: // Salir: avisamos de que salimos

cout << "Fin del programa" << endl;

break;

default: // Otra opcion: no válida

cout << "Opción desconocida!" << endl;

break;

}

} while (opcion != 5); // Si la opcion es 5, terminamos

return 0;

}

Funciona, y hace todo lo que tiene que hacer, pero es mejorable. Por supuesto, en un caso real es habitual que cada ficha tenga que guardar más información que sólo esos dos apartados de ejemplo que hemos previsto esta vez. Si nos muestra todos los datos en pantalla y se trata de muchos datos, puede ocurrir que aparezcan en pantalla tan rápido que no nos dé tiempo a leerlos, así que sería deseable que parase cuando se llenase la pantalla de información (por ejemplo, una pausa tras mostrar cada 25 datos). Por supuesto, se nos pueden ocurrir muchas más preguntas que hacerle sobre nuestros datos. Y además, cuando salgamos del programa se borrarán todos los datos que habíamos tecleado, pero eso es lo único “casi inevitable”, porque aún no sabemos manejar ficheros.

Ejercicios propuestos:

* **(7.4.1)** Un programa que pida el nombre, el apellido y la edad de una persona, los almacene en un “struct” y luego muestre los tres datos en una misma línea, separados por comas.
* **(7.4.2)** Un programa que pida datos de 8 personas: nombre, dia de nacimiento, mes de nacimiento, y año de nacimiento (que se deben almacenar en una tabla de structs). Después deberá repetir lo siguiente: preguntar un número de mes y mostrar en pantalla los datos de las personas que cumplan los años durante ese mes. Terminará de repetirse cuando se teclee 0 como número de mes.
* **(7.4.3)** Un programa que sea capaz de almacenar los datos de 50 personas: nombre, dirección, teléfono, edad (usando una tabla de structs). Deberá ir pidiendo los datos uno por uno, hasta que un nombre se introduzca vacío (se pulse Intro sin teclear nada). Entonces deberá aparecer un menú que permita:
  + Mostrar la lista de todos los nombres.
  + Mostrar las personas de una cierta edad.
  + Mostrar las personas cuya inicial sea la que el usuario indique.
  + Salir del programa  
    (lógicamente, este menú debe repetirse hasta que se escoja la opción de “salir”).
* **(7.4.4)** Mejorar la base de datos de ficheros (ejemplo 07.05) para que no permita introducir tamaños incorrectos (números negativos) ni nombres de fichero vacíos.
* **(7.4.5)** Ampliar la base de datos de ficheros (ejemplo 07.05) para que incluya una opción de búsqueda parcial, en la que el usuario indique parte del nombre y se muestre todos los ficheros que contienen ese fragmento.
* **(7.4.6)** Ampliar la base de datos de ficheros (ejemplo 07.05) para que se pueda borrar un cierto dato (habrá que mover hacia atrás todos los datos que había después de ese, y disminuir el contador de la cantidad de datos que tenemos).
* **(7.4.7)** Mejorar la base de datos de ficheros (ejemplo 07.05) para que se pueda modificar un cierto dato a partir de su número (por ejemplo, el dato número 3). En esa modificación, se deberá permitir al usuario pulsar Intro sin teclear nada, para indicar que no desea modificar un cierto dato, en vez de reemplazarlo por una cadena vacía.
* **(7.4.8)** Ampliar la base de datos de ficheros (ejemplo 07.05) para que se permita ordenar los datos por nombre. Para ello, deberás buscar información sobre algún método de ordenación sencillo, como el "método de burbuja" (en el siguiente apartado tienes algunos), y aplicarlo a este caso concreto.

7.5. Ordenaciones Simples

Es muy frecuente querer ordenar datos que tenemos en un array. Para conseguirlo, existen varios algoritmos sencillos, que no son especialmente eficientes, pero son fáciles de programar. La falta de eficiencia se refiere a que la mayoría de ellos se basan en dos bucles for anidados, de modo que en cada pasada quede ordenado un dato, y se dan tantas pasadas como datos existen, de modo que para un array con 1.000 datos, podrían llegar a tener que hacerse un millón de comparaciones.

Existen ligeras mejoras (por ejemplo, cambiar uno de los for por un while, para no repasar todos los datos si ya estaban parcialmente ordenados), así como métodos claramente más efectivos, pero más difíciles de programar, alguno de los cuales veremos más adelante.

Veremos tres de estos métodos simples de ordenación, primero mirando la apariencia que tiene el algoritmo, y luego juntando los tres en un ejemplo que los pruebe:

**Método de burbuja**

(Intercambiar cada pareja consecutiva que no esté ordenada)

Para i=1 hasta n-1

Para j=i+1 hasta n

Si A[i] > A[j]

Intercambiar ( A[i], A[j])

(Nota: algunos autores hacen el bucle exterior creciente y otros decreciente, así:)

Para i=n descendiendo hasta 1

Para j=2 hasta i

Si A[j-1] > A[j]

Intercambiar ( A[j-1], A[j])

**Selección directa**

(En cada pasada busca el menor, y lo intercambia al final de la pasada)

Para i=1 hasta n-1

menor = i

Para j=i+1 hasta n

Si A[j] < A[menor]

menor = j

Si menor <> i

Intercambiar ( A[i], A[menor])

**Inserción directa**

(Comparar cada elemento con los anteriores -que ya están ordenados- y desplazarlo hasta su posición correcta).

Para i=2 hasta n

j=i-1

mientras (j>=1) y (A[j] > A[j+1])

Intercambiar ( A[j], A[j+1])

j = j - 1

(Es mejorable, no intercambiando el dato que se mueve con cada elemento, sino sólo al final de cada pasada, pero no entraremos en más detalles).

**Ejercicios propuestos:**

* **(7.5.1)** Un programa que pida al usuario 5 números reales y los muestre ordenados, usando el método de la burbuja.
* **(7.5.2)** Un programa que cree un array de 7 números enteros y lo ordene con cada uno de estos tres métodos, mostrando el resultado de los pasos intermedios.

Bibliografía.

Aprende a Programar.com

URL: <http://www.aprendeaprogramar.com/mod/resource/view.php?id=495>