Práctica 8: Registros, Vectores de Registros y Ficheros

Objetivos

- Aprender a definir nuevos tipos de datos compuestos usando el concepto de registro para almacenar una colección de datos heterogéneos en C++.
- Aplicar y manipular vectores de registros a problemas sencillos.
- Conocer cómo crear y utilizar ficheros de tipo texto para almacenar o leer información.
- Aplicar el método de diseño descendente para construir programas modulares.

Parte 1: Registros

Conceptos Básicos

Los registros, también llamados estructuras en C++, son tipos de datos compuestos heterogéneos definidos por el programador, y por tanto agrupan datos de diferentes tipos. Por ejemplo una ficha de un libro que guarda la siguiente información:

- Título
- Autor
- Editorial
- Número de páginas
- Precio

Para almacenar esta información en C++ se puede definir un nuevo tipo de dato `Ficha` del siguiente modo usando la palabra reservada `struct`:

```
struct Ficha
{
   string titulo;
   string autor;
   string editorial;
   unsigned int numPag;
   float precio;
};
```

```
int main()
{
   Ficha a;
   a.titulo = “La casa de los Espíritus”;
   . . .
   return 0;
}
```

La declararación de una variable de tipo `Ficha` se realiza de la manera habitual. A cada uno de los componentes de un registro se denomina **campo**. Los datos título, autor, editorial, numPag y precio son los campos del registro Ficha. Para acceder a cada uno de los campos se utiliza el **operador punto.**

Inicialización de un registro

Una variable de tipo registro puede inicializarse en el momento de su declaración, indicando los valores de los campos separados por comas y encerrados entre llaves:

```
struct Dimensiones
{
   float alto;
   float ancho;
};
```

```
int main()
{
   Dimensiones d = {12.4, 20.8};
   ...
   return 0;
}
```

Como caso aparte, si algún campo es de tipo `string`, no puede inicializarse de la manera anterior. La alternativa es la inicialización campo a campo:

```
int main()
{
   Ficha libro;

   libro.titulo = “El conde de Montecristo”;
   libro.autor = “Alejandro Dumas”;
   libro.editorial = “Aguilar”;
   libro.numPag = 319;
   libro.precio = 29.51;
    . . .
   return 0;
}
```

Asignación entre registros y paso de parámetros a un subprograma

En C++ está permitida la asignación entre registros:

```
Dimensiones a, b;

a.alto = 13.56;
a.ancho = 12.56;
b = a; //b tiene el mismo contenido que a
```

Como consecuencia de esta propiedad, un registro puede pasarse por valor o por referencia como parámetro a una función. Esta propiedad también hace posible que una función pueda devolver un registro como valor de retorno.

Anidamiento de registros

La definición de una estructura puede tener como campo una variable que sea a su vez de tipo registro. Debe prestarse atención al orden de las definiciones.

```
struct Dimensiones
{
   float alto;
   float ancho;
};
struct Ficha
{
   string titulo;
   string autor;
   string editorial;
   unsigned int numPag;
   float precio;
   Dimensiones tamanyo;
};
```

```
Ficha libro;

cout << “El libro mide “ << libro.tamanyo.alto;
cout << " cm de alto.";
```

El acceso a los campos del registro tamanyo se hace usando dos veces el operador punto.

Vectores de registros

Un vector de resgistros es muy útil para implementar colecciones de datos. Por ejemplo, para tener los datos de todos los libros en venta de una librería. En la variable `inventario` se podrían guardar hasta un máximo de 500 fichas.

```
const unsigned int TAM = 500;
typedef Ficha VLibros [TAM];

VLibros inventario;
```

Parte 2: Ficheros de tipo texto

Conceptos Básicos

Todos los programas vistos hasta ahora trabajan con información almacenada en memoria principal que se lee desde teclado o se imprime por pantalla. No obstante, hay situaciones en que esto no es apropiado. Algunas de éstas situaciones son:

- El conjunto de datos con los que se necesita trabajar es demasiado grande para que quepa en la memoria principal.
- Se necesita utilizar datos procedentes de otros programas (editores, hojas de cálculo, bases de datos, etc).
- Interesa mantener la información después de cada ejecución para que pueda ser utilizada por otros programas o aplicaciones.

En estos casos se utilizan ficheros para guardar la información en memoria secundaria (discos duros, memorias USB, etc.).

Definición de Fichero

Es una colección de elementos lógicamente relacionados y almacenados en memoria secundaria. Es un tipo de dato compuesto heterogéneo. A más bajo nivel, un fichero es una secuencia de bits almacenados en un dispositivo externo.

En C++ un fichero es simplemente **un canal de flujo (`stream`)** que se puede abrir para entrada (fichero de entrada), para salida (fichero de salida) o para entrada-salida (fichero de entrada-salida). Este último no lo veremos en este curso.

El lenguaje C++ soporta archivos de texto y archivos binarios. Los primeros almacenan datos como caracteres separados por espacios, retornos de carro o tabuladores. Los segundos almacenan los bits de forma directa. Un fichero binario tiene un tamaño mucho menor del que tendría el fichero texto en general. La biblioteca que proporciona las funciones y operadores para el manejo de ficheros es **`#include <fstream>.`**

Declaración, Apertura y Cierre de Ficheros

Los ficheros se pueden abrir para leer su contenido o para escribir en ellos. Nunca se lee y escribe simultáneamente.

Declaración de una variable tipo **fichero de escritura**:	`ofstream f;`
Declaración de una variable de tipo **fichero de lectura**:	`ifstream f;`

Apertura	`f.open (nombre);  // Apertura del fichero`
	
	`f.open (nombre,ios::app); /* Apertura para añadir datos al final */`
	
	`f.open (nombre, ios::binary);  /* Apertura de un fichero binario */`

Cierre	`f.close ();  // Cierre del fichero`

Si no se especifican rutas, Dev-C++ busca y crea los ficheros en el directorio en el que se está ejecutando el programa. Si se desea leer o guardar los ficheros en otro directorio es necesario indicar el camino completo `f.open ("c:\\tmp\\prueba\\datos.txt").`

Al abrir un fichero cuyo nombre es almacenado en una variable de tipo `string` hay que realizar una conversión usando el método `c_str()`.

```
string nombreFich;
ifstream f;
cout << "Introduce el nombre del fichero a analizar: ";
cin >> nombreFich;
```

```
f.open (nombreFich.c_str () );
f >> num;
while ( !f.eof () )
   f >> num;
f.close ();
```

Detección de errores

- La función `f.eof()` devuelve `true` si se ha alcanzado el final del fichero y `false` en otro caso. Requiere una lectura adelantada para que devuelva un valor de verdad actualizado.
- La función `f.fail()` devuelve `true` si existe un error en una operación asociada al fichero.
- La función `f.is_open()`devuelve `true` si el fichero se ha abierto correctamente.

Antes de empezar a leer o escribir en un fichero es necesario verificar que la operación de apertura se ha realizado con éxito.

```
ifstream f;

f.open("ejemplo.txt");
if ( f.fail() )
   cout << "Error en la apertura del fichero " << endl;
else
{
   . . .
   f.close();
}
```

Ficheros de tipo texto

La lectura y escritura de información se realiza mediante los operadores de entrada y de salida '**>>**' y '**<<**'.

Ejemplo de escritura:

```
ofstream f;
int dato = 4;

f.open ("nombrefich.txt");

f << dato;

f.close ();
```

Ejemplo de lectura:

```
ifstream f;
int dato;

f.open ("nombrefich.txt");

f >> dato;

f.close ();
```

Para leer o escribir un carácter también se pueden utilizar los métodos `put` y `get`:

Ejemplo de escritura:
```
char c = 'a';
ofstream f;

f.put (c);
```

Ejemplo de lectura:
```
char c;
ifstream f;

f.get (c);
```

El operador >> lee hasta que encuentra un espacio en blanco. Para leer cadenas que contengan espacios en blanco se puede utilizar la función `getline`. Lee toda la línea hasta que se encuentre un salto de línea.

```
string s;
ifstream f;
getline (f, s);
```

No es posible almacenar o leer un registro en un fichero de tipo texto. La lectura o escritura se debe realizar campo a campo:

```
struct Complejo
{
    int real;
    int imag;
};
```

```
ofstream f;
Complejo c;

f << c.real; // CORRECTO
f << c.imag;

f << c; // INCORRECTO
```

Para leer todos los datos de un fichero de texto separados por espacios en blanco es necesario utilizar un bucle. Dos opciones de implementación pueden ser:

```
while (fich >> dato)
// false cuando se encuentra el final
{
   ...
}
```

```
fich >> num;
while (!fich.eof () )
{
   fich >> num;
   . . .
}
```

BLOQUE DE EJERCICIOS

Ejercicio1 (ejercicio1.cpp): Abre el fichero ejercicio1.cpp. El tipo de dato `Cliente` permite guardar la información de un cliente de un banco: su DNI, el saldo y la fecha de apertura de la cuenta. El tipo de dato `VectorC` permite almacenar cinco clientes.

```
struct Fecha
{
   unsigned int dia;
   unsigned int mes;
   unsigned int anyo;
};

struct Cliente
{
   unsigned int dni;
   float saldo;
   Fecha f;
};

const unsigned int TAM = 5;
typedef Cliente VectorC [TAM];
```

Implementa las siguientes funciones:

1. Una función **LeerClientes** que lea la información de cinco clientes desde el teclado y la almacene en un vector de clientes que se le pasará como parámetro.
   `void LeerClientes (VectorC);`

2. Una función **SaldoMedio** que calcule el saldo medio de los clientes y lo devuelva como valor de retorno. Tiene como dato de entrada el vector con los clientes.
   `float SaldoMedio (const VectorC);`

3. Un procedimiento **MostrarClientesVIP** que pasándole el vector de clientes y un saldo, muestre en pantalla toda la información de los clientes que superan ese saldo en su cuenta.
   `void MostrarCientesVIP (const VectorC, float);`

4. Escribe el programa en C++ de forma que se llame a la función leer para inicializar el vector, calcule el saldo medio de los clientes y se muestre toda la información de los clientes con saldo superior a 20.000 € y de nuevo con saldo superior a 50.000 €.

5. Con GDB (al igual que en la sesión 6) comprueba que los elementos del vector se almacenan de forma consecutiva en memoria. Obtén el tamaño en bytes de un Cliente y del vector de Clientes. Usa la función `sizeof()` para comprobar que tu respuesta es correcta.

   `https://en.cppreference.com/w/cpp/language/sizeof`

Ejercicio2 (ejercicio2.cpp): Escribe un programa en C++ que muestre por pantalla una tabla con los valores de la función x^2 y $\sqrt{x}$ en el intervalo [1,100], en incrementos de 10 en 10. La tabla se debe guardar en un archivo llamado `datos.txt` con el siguiente formato.

```
--------------------------------
x             x^2      sqrt(x)
--------------------------------
0             0           0.0
10            100         3.16
. . .
100           10000       10.0
```

Ejercicio3 (ejercicio3.cpp): El código EAN-13 (European Article Number) contiene 13 dígitos y es el código de barras más comúnmente utilizado. Los dos primeros dígitos identifican el país de origen del producto, los siguientes cinco corresponden con el fabricante, los cinco siguientes con el identificador de producto y el último es un dígito de control para comprobar que el código es válido.

Cálculo del Dígito de Control: Se suman los dígitos de las posiciones impares, el resultado se multiplica por 3 y se le suman los dígitos de las posiciones pares. Se busca la decena inmediatamente superior y se le resta el resultado anterior. El valor resultante es el dígito de control. Si es 10, el dígito de control será cero. Por ejemplo:

Código EAN 13	8	4	1	2	5	8	4	5	1	2	5	4	1

Paso 1: Suma de los dígitos que ocupan las posiciones pares: 8 + 1 + 5 + 4 + 1 + 5 = 24

Paso 2: Suma de los dígitos que ocupan las posiciones impares: 4 + 2 + 8 + 5 + 2 + 4 = 25

Paso 3: Multiplica por 3 la suma de los dígitos impares: 25 * 3 = 75

Paso 4: Suma al valor obtenido anteriormente la suma de los números pares: 24 + 75 = 99

Paso 5: Redondea el valor obtenido a la decena inmediatamente superior, 100

Paso 6: El dígito de control es el valor obtenido de restar a la decena superior la suma total: 100 – 99 = 1

Escribe una función **esEAN** que se le pase como parámetro de entrada una cadena de caracteres que representa un código de barras y devuelva si es o no correcto. Ayuda: el código ASCII del carácter '0' es 48.

Implementa un programa en C++ donde se lea desde el fichero `EAN13.txt` cada uno de los códigos de barras y usando la función anterior se verifique si son correctos o no. Se debe crear un fichero con los códigos válidos y otro con los no válidos (Ficheros `EAN13_Validos.txt` y `EAN13_NoValidos.txt`).

Ejercicio4 (ejercicio4.cpp): Abre el fichero ejercicio4.cpp. Completa el programa. Este programa debe permitir gestionar una agenda de teléfonos que se guarda en un fichero.

1. Leer la agenda desde un fichero, “agenda.txt”.
2. Insertar una nueva persona en la agenda. La persona es antes leída desde teclado.
3. Borrar una persona de la agenda. Se lee desde teclado el nombre a eliminar. **Al eliminar del vector, no dejar huecos.** Es necesario reorganizar el vector.
4. Modificar el teléfono de una persona conocido su nombre. Se lee desde el teclado su nombre y el nuevo teléfono.
5. Volcar a un fichero nuevo la información actual de la agenda, ”agendanueva.txt”.
6. Mostrar en pantalla la información actual de la agenda.
0. Salir.

La agenda tiene capacidad para **1000 contactos como máximo**. Los tipos de datos definidos y el prototipo de las funciones a implementar son:

```
struct Persona
{
  string nombre;
   string telefono;
   string email;
};

const unsigned int MAX = 1000;
typedef Persona VPersona [MAX];

unsigned int Menu ();

//lee el vector de personas desde un fichero y rellena el tamaño real del vector
void LeerAgendaFich (ifstream &, VPersona, unsigned int &);

//lee los datos de una persona desde teclado
Persona LeerPersonaTeclado (void);

//inserta una persona al final del último elemento e incrementa el tamaño
void InsertarPersonaAgenda (VPersona, unsigned int &, Persona);

//elimina una persona conocido su nombre y decrementa el tamaño
void EliminarPersonaAgenda (VPersona, unsigned int &, string);

//busca la persona por el nombre y modifica el teléfono.
//vector, tamaño actual, nombre de la persona a modificar, nuevo teléfono.
void ModificarPersonaAgendaTel(VPersona, unsigned int, string, string);

//guarda el contenido de la agenda a fichero
void GuardarAgendaFich(ofstream &, const VPersona, unsigned int);

//muestra el contenido de la agenda en pantalla
void MostrarAgendaPantalla(const VPersona, unsigned int);
```

Ejercicio5 (ejercicio5.cpp): En un fichero de texto `empleados.txt` se tiene almacenada la información de **a lo sumo 1000 empleados.** De cada uno se almacena: nombre, puesto que ocupa, fecha de contratación, si es fijo y un vector con su nómina de los 12 meses del año 2021. Por ejemplo:

```
Juan García Sanz
Analista programador
1 8 2014
FIJO
3201.4 3102.1 3305.2 3003.5 3204.6 3715.1 3216.0 3117.4 3228.1 3109.5 3120.7 3001.9
Irene González Alegría
Jefa Departamento Recursos Humanos
10 12 2018
FIJA
3802.4 3901.1 3706.4 3903.1 3874.1 3816.3 3618.1 3517.4 3681.2 3699.2 3520.6 3812.5
. . .
```

Implementa los siguientes tipos de datos y funciones:

1. El tipo de dato Empleado y un tipo de dato vector de Empleados.
2. Una función que lea del fichero los datos de los empleados y los almacene en un vector.
3. Una función que devuelva **el empleado** más antiguo.
4. Una función que reciba un vector de empleados y genere un nuevo fichero con los mismos datos donde el salario se ha incrementado un 3%.
5. Un programa en C++ que: i) lea la información del archivo, ii) obtenga el empleado más antiguo en la empresa y muestre toda su información; iii) genere el nuevo fichero `empleadosNuevo.txt` con el salario incrementado y iv) genere un fichero HTML `empleados.html` cuya estructura sea la siguiente:

```
struct Fecha
{
  //completar
};

struct Empleado
{
      //completar
};

const unsigned int TAM = 1000;
typedef Empleado Vector [TAM];

void LeerVector (ifstream &, Vector, unsigned int &);
Empleado EmpleadoMasAntiguo (const Vector, unsigned int);
void VolcarFich (ofstream &, const Vector, unsigned int);
```

Fichero HTML

```
<html>
<body>
<h2>Lista de empleados:</h2>
<p>
<b>Nombre:</b>Juan García Sanz<br>
<b>Puesto:</b>Analista Programador<br>
<b>Salario Total:</b>38325.5<br>
</p>
<p>
<b>Nombre:</b>Irene González Alegría<br>
<b>Puesto:</b>Jefa Departamento Recursos Humanos<br>
<b>Salario Total:</b>44852.4<br>
</p>

...
</body>
</html>
```

IMPORTANTE:

- Todos los programas deben seguir la Guía de Estilo e incluir los comandos de Doxygen para generar la documentación.
- Subir al Aula Virtual los archivos .cpp de uno en uno. Sólo los sube un miembro de la pareja.
- Fecha de entrega: durante la sesión de vuestro grupo de lab.