Fundamentos de Programación
Sesión 3

Actividades a realizar en casa

Actividad: Resolución de problemas.

Resolved los siguientes problemas de la relación 1 y 2. Recordad que, antes del inicio de esta sesión en el aula de ordenadores, hay que subir las soluciones a PRADO.

De la relación 1:
11 (Precisión y desbordamiento)
12 (Intercambiar tres variables)
15 (Pasar de mayúscula a minúscula)
16 (Pasar de carácter a entero)
18 (Expresiones lógicas)

De la relación 2:
1 (Valor por encima o por debajo de la media)
3 (Ver si dos números se dividen)

Actividades a realizar en las aulas de ordenadores

Estructuras o registros (struct)

En esta sesión se presentará un tipo de dato compuesto (es decir, que se construye a partir de otros tipos de datos) heredado de C: las estructuras. Éstas sirven para organizar un conjunto de tipos de datos que representan las características de una entidad. Nos servirán como introducción al concepto de clase (Tema 3 de teoría).

Las estructuras o registros son tipos de dato compuestos que se definen a partir de otros tipos. Simplemente consisten en agrupar varios tipos de datos para formar un tipo de dato complejo (que representan las características de un objeto o entidad a tratar en nuestro algoritmo). Cada uno de los datos que lo componen se llama *campo* (o *miembro,* o *dato miembro)* de la estructura.

Por ejemplo, si queremos implementar un programa para trabajar con los DNI's de personas podríamos considerar una estructura DNI, que tiene dos campos: un entero (int) y un carácter (char).

La sintaxis genérica para definir una estructura es la siguiente:

```
struct identificador_struct{
        Tipo_dato1     identificador_campo1;
        Tipo_dato2     identificador_campo2;
        …
        Tipo_datoN     identificador_campoN;
};
```

Entonces, la estructura DNI se debería definir como se especifica en la Figura 1.

```
struct DNI{
    int numero;
    char letra;
};
```

Figura 1. Definición de la estructura DNI

Una vez que se ha definido esta estructura, DNI pasa a considerarse como cualquier otro tipo de dato básico. Entonces, podemos implementar sentencias de declaración de datos, asignación de valores, entrada y salida de datos, etc.

```
DNI alumno1={12345678,'T'};
DNI alumno2;
alumno2=alumno1;
cout << alumno2.letra;
```

Figura 2. Algunas sentencias con variables de tipo DNI

Veamos ahora un ejemplo básico de uso de las estructuras. En PRADO, dentro de la carpeta de prácticas se encuentra el fichero 3_salario.cpp. Este programa muestra por pantalla el empleado de cierta empresa que cobra más, entre dos empleados insertados por el usuario. Cread una carpeta 3_salario en vuestra carpeta local (dentro de U:\FP) y copiad este fichero dentro de la misma. Desde el Explorador de Windows, haced doble click sobre el fichero para abrirlo con Orwell Dev-C++. Debería aparecer un programa como el de la Figura 3.

```
1  /* Programa para calcular que empleado tiene mayor salario */
2
3  #include<iostream> // inclusion de la libreria de E/S
4  #include<string> // inclusion de la libreria para tratamiento de cadenas de car.
5
6  using namespace std; // espacio de nombres estandar
7
8  // definicion de la estructura que representa a un empleado
9  struct Empleado{
10     string nombre;
11     int salario;
12 };
13
14 int main(){
15     Empleado empleado1; // Declaracion del primer dato Empleado
16     Empleado empleado2; // Declaracion del primer dato Empleado
17     string empleado_mayor_salario; // Declaración de dato string para almacenar salida
18
19     // Insercion de datos por parte del usuario
20     cout << "\nIntroduce el nombre del primer empleado: ";
21     cin >> empleado1.nombre;
22     cout << "\nIntroduce el salario del primer empleado: ";
23     cin >> empleado1.salario;
24     cout << "\nIntroduce el nombre del segundo empleado: ";
25     cin >> empleado2.nombre;
26     cout << "\nIntroduce el salario del segundo empleado: ";
27     cin >> empleado2.salario;
28
29     // Calculo de la solucion
30     if ( empleado1.salario > empleado2.salario )
31         empleado_mayor_salario = empleado1.nombre;
32     else
33         empleado_mayor_salario = empleado2.nombre;
34
35     // Salida de datos
36     cout << "\nEl empleado que cobra más es: " << empleado_mayor_salario << "\n\n";
37 }
```

Figura 3. Programa que implementa el cálculo del empleado con mayor salario.

Compila el programa y comprueba si tiene un funcionamiento correcto.

- Lo primero que debemos darnos cuenta es que la definición de una estructura termina siempre con un ; después de la } que cierra la definición (línea 12 en el programa de la

```
8  // definicion de la estructura que representa a un empleado
9  struct Empleado{
10     string nombre;
11     int salario;
12 };
```

Figura 3). Un error muy común es olvidarse de este punto y coma. Elimina el punto y coma de la línea 12 y compila para conocer el error. Vuelve a escribirlo para que el programa no presente errores de compilación

- Observa también la posición en el fichero de la definición de la estructura. Antes de utilizar el tipo de dato Empleado (como en una declaración de variables, por ejemplo) es necesario que el compilador "conozca" qué es Empleado (recordad que el compilador va leyendo en orden las líneas de código). En nuestro código colocaremos las definiciones de nuestras estructuras (en general de los tipos de datos definidos por nosotros) entre el espacio de nombres y la función main. Entonces, un fichero .cpp debería tener una organización como la siguiente:

```
/* Descripción del programa
contenido en el fichero */

#include<..> // Bibliotecas

using namespace std; // espacio de nombres

struct Dato{   // tipos de datos definidos por el usuario
...
};

int main(){
        // declaración constantes
        // declaración variables

        // operaciones

        // salida de datos
}
```

Corta la definición de Empleado en el fichero 3_salario.cpp y pégala después de la función main. Observa el error de compilación que se produce y vuelve a colocar la definición en su sitio.

- Una vez definido el tipo de dato Empleado, este se puede utilizar como cualquier tipo de dato básico. Por ejemplo, podemos **declarar variables** (líneas 14 y 15) de este nuevo tipo de dato.

```
14 int main(){
15     Empleado empleado1; // Declaracion del primer dato Empleado
16     Empleado empleado2; // Declaracion del primer dato Empleado
```

La declaración de una variable de tipo struct tiene como efecto la reserva de memoria **consecutiva** para almacenar cada uno de los tipos de datos de los campos.

- A la hora de inicializar una variable Empleado, podemos inicializar todos los campos de una vez. Pero esto debe hacerse **en la misma sentencia en la que se declara la variable**. La sintaxis, en el ejemplo, es la siguiente:

```
Empleado empleado1 = { “Maria” , 2000 } ;
```

  En general, se escriben, entre llaves y separados por comas, los campos de la estructura en el mismo orden en que se listan en la definición.
  Comenta las líneas 20, 21, 22 y 23, e inicializa la variable declarada en la línea 15 con los escrito en el recuadro anterior. Compila y comprueba el funcionamiento.
  Para comprobar que esta inicialización “de un tirón” sólo es posible hacerla en la misma sentencia en la que se declara la variable, cambia la nueva línea 15 por dos sentencias: una donde se declara la variable

```
Empleado empleado1;
```

  y otra donde se le dan los valores

```
empleado1 = { “Maria” , 2000 };
```

  Compila para observar el error que produce. Devuelve el código a su forma original (como está en PRADO).

- Otra forma de inicializar todos los datos miembros de una sola vez es utilizar **el operador de asignación** (=). Este copia campo a campo los datos almacenados. Vuelve a comentar las líneas 20, 21, 22 y 23, y añade las líneas

```
Empleado empleado3 = { “Pepito” , 1500 } ;
empleado1=empleado3;
```

  después de la declaración de la variable empleado2. Compila y comprueba el funcionamiento del programa. Devuelve el código a su forma original (como está en PRADO).

- Cada campo de una variable de tipo struct puede tratarse como una variable independiente del tipo de dato correspondiente. En este ejemplo, como si manejáramos una variable de tipo string y otra de tipo entero (int). Para acceder a cada dato miembro se utiliza el **operador punto** (.), seguido del nombre del campo. Esto podemos utilizarlo:
  - En una sentencia de entrada de datos, para darle valores a los campos (líneas 21, 23, 25 y 27).

```
18
19      // Insercion de datos por parte del usuario
20      cout << "\nIntroduce el nombre del primer empleado: ";
21      cin >> empleado1.nombre;
22      cout << "\nIntroduce el salario del primer empleado: ";
23      cin >> empleado1.salario;
24      cout << "\nIntroduce el nombre del segundo empleado: ";
25      cin >> empleado2.nombre;
26      cout << "\nIntroduce el salario del segundo empleado: ";
27      cin >> empleado2.salario;
28
```

De igual manera, en sentencias de salida de datos, para mostrar el valor de un campo por consola. Añade (no utilices cortar y pegar desde el pdf) las siguientes líneas después de la línea 27.

```
cout << "El primer empleado se llama " << empleado1.nombre << "\n";
cout << "El segundo empleado se llama " << empleado2.nombre << "\n";
```

Compila y ejecuta para observar el resultado.

- Dentro de una expresión, para asignar valores a otra variable. Añade al final del programa (no utilices copiar y pegar desde el pdf)

```
int media = ( empleado1.salario + empleado2.salario ) / 2;
cout << "El salario medio es " << media << "\n";
```

Compila y ejecuta para observar el resultado.

- O bien, directamente, para asignar valores a un campo (por ejemplo, para inicializar una variable de la estructura campo a campo). Añade después de la línea 23

```
empleado1.nombre = "Maria";
empleado2.salario = 2000;
```

Compila y ejecuta el programa.

- Como caso particular, una estructura puede ser un dato miembro de otra estructura. Considera la estructura DNI definida anteriormente. En el archivo original 3_salario.cpp:
  - Escribe la definición de DNI encima de la de Empleado.
  - Añade un campo más a la definición de empleado: DNI doc_id;
  - Observa que la definición de DNI debe estar colocada en el código **ANTES** de la de Empleado. El compilador lee en orden el fichero y debe "conocer" todas las palabras.
  - Para acceder al número y letra del DNI de un empleado **debemos utilizar dos veces el operador punto**. Añade, después de la petición al usuario del salario del segundo empleado, el siguiente código

```
cout << "\nIntroduce el número de DNI del primer empleado: ";
cin >> empleado1.doc_id.numero;
cout << "\nIntroduce la letra de DNI del primer empleado: ";
cin >> empleado1.doc_id.letra;
cout << "\nIntroduce el número de DNI del segundo empleado: ";
cin >> empleado2.doc_id.numero;
cout << "\nIntroduce la letra de DNI del segundo empleado: ";
cin >> empleado2.doc_id.letra;
```

No es necesario utilizar paréntesis en las sentencias con doble punto, el compilar lee de izquierda a derecha. Aún así, podríamos haber puesto, por ejemplo,

```
cin >> (empleado1.doc_id).numero;
```

Actividad: Resolución de problemas.

Resolved los siguientes ejercicios de la relación 1:

31 (TipoPunto)
32 (CuentaCD)
35 (Fecha)