Programación 1

Tema 11

Estructuración agregada de datos

Escuela de
Ingeniería y Arquitectura
Universidad Zaragoza

Índice

- Registros y campos
- Dominio de valores
- Representación externa
- Operaciones
- Problemas y ejemplos

Problema

- Gestionar información relativa a ciudadanos de un determinado lugar:
  - Nombre
  - Apellidos
  - NIF
  - Fecha de nacimiento
  - Estado civil (casado, no casado)
  - Sexo

Registro

- **Registro** o **tupla**
  - Agrupan datos de igual o diferente naturaleza relacionados entre sí
- Para utilizarlos, hay que definir antes un **tipo registro**

Sintaxis

```
/*
 * Definición de un nuevo tipo de dato con estructura de registro:
 */
struct NombreTipo {
    tipo_1 nombre_campo_1;
    tipo_2 nombre_campo_2;
   ...
    tipo_n nombre_campo_n;
};

/*
 * Definición de datos, simples o estructurados, del tipo definido
 * anteriormente:
 */
NombreTipo reg1, reg2;
NombreTipo reg3;
NombreTipo tabla[100];
```

Ejemplos

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

Fecha hoy;
Fecha cumplesClase[50];
```

Ejemplos

```
struct Nif {
    int  dni;    // número del DNI del ciudadano
    char letra; // letra asociada al DNI
};
```

Sintaxis

```
<definiciónTipoRegistro>
  ::= “struct” <identificador> “{”
      { <definiciónCampo> }
      “}” “;”
<definiciónCampo>
  ::= <tipo> <declaraciónSimple>
      { “,” <declaraciónSimple> } “;”
<declaraciónSimple>
  ::= <nombreCampo> [ “=” <expresión> ]
```

Registros

- Dominio de valores
  - Producto cartesiano de los dominios de valores de los campos
- Representación externa
  - Listas (solo en la inicialización)
    - `Fecha hoy = {13, 11, 2018};`
    - `Nif rey = {15, 'S'};`

Registros

- Operadores:
  - "." : operador de selección de campo
    - `hoy.dia = 27;`
    - `hoy.dia++;`
    - `cout << rey.dni << "-" << rey.letra;`
    - `cumplesClase[0].mes = 8;`

Metodología de trabajo con registros en módulos

- Fichero de **interfaz** del módulo
  - Definición del tipo registro
  - Declaraciones de funciones adicionales para trabajar con el tipo
- Fichero de **implementación** del módulo
  - Definiciones de las funciones declaradas en el fichero de interfaz
  - Definiciones de otras funciones auxiliares

Número de identificación fiscal. Interfaz

```
/*
 * Definición del tipo de dato Nif que representa la
 * información del NIF (Número de Identificación
 * Fiscal) de un ciudadano.
 */
struct Nif {
    int dni;        // número del DNI del ciudadano
    char letra;     // letra asociada al DNI anterior
};

...
```

Número de identificación fiscal. Interfaz

```
...

/*
 * Pre:  ---
 * Post: Ha devuelto «true» si y solo si
 *       «nifAValidar» define un NIF válido,
 *       es decir, su letra es la que le corresponde
 *       a su DNI.
 */
bool esValido(const Nif nifAValidar);

...
```

Número de identificación fiscal. Interfaz

```
...

/*
 * Pre:  El valor del parámetro
 *       «nifAEscribir» representa un NIF
 *       válido.
 * Post: Ha escrito «nifAEscribir» en
 *       pantalla, con un formato como
 *       «01234567-L».
 */
void mostrar(const Nif nifAEscribir);
```

Número de identificación fiscal. Implementación

```
#include "nif.h"
#include <iostream>
#include <iomanip>
using namespace std;

const int NUM_LETRAS = 23;
const char TABLA_NIF[NUM_LETRAS]
        = { 'T', 'R', 'W', 'A', 'G', 'M',
            'Y', 'F', 'P', 'D', 'X', 'B',
            'N', 'J', 'Z', 'S', 'Q', 'V',
            'H', 'L', 'C', 'K', 'E' };
```

Número de identificación fiscal.
Implementación

```
/*
 * Pre:  ---
 * Post: Ha devuelto la letra del número
 *       de identificación fiscal que
 *       corresponde a un número de
 *       documento nacional de identidad
 *       igual a «dni».
 */
char letra(int dni) {
    return TABLA_NIF[dni % NUM_LETRAS];
}
```

Número de identificación fiscal.
Implementación

```
/*
 * Pre:  ---
 * Post: Ha devuelto «true» si y solo si
 *       «nifAValidar» define un NIF
 *       válido, es decir, su letra es la
 *       que le corresponde a su DNI.
 */
bool esValido(const Nif nifAValidar) {
    return letra(nifAValidar.dni)
           == nifAValidar.letra;
}
```

Número de identificación fiscal. Implementación

```
/*
 * Pre:  El valor del parámetro «nifAEscribir» representa
 *       un NIF válido.
 * Post: Ha escrito «nifAEscribir» en pantalla, con un
 *       formato como «01234567-L». También ha modificado
 *       el carácter de relleno que utiliza el manipulador
 *       «setw», estableciendo el espacio en blanco.
 */
void mostrar(const Nif nifAEscribir) {
    cout << setfill('0');
    cout << setw(8) << nifAEscribir.dni << "-"
         << nifAEscribir.letra;
    cout << setfill(' ');
}
```

Número de identificación fiscal.
Ejemplo de uso

```
Nif unNifCualquiera;
unNifCualquiera.dni = 1234567;
unNifCualquiera.letra = 'L';
if (esValido(unNifCualquiera)) {
    mostrar(unNifCualquiera);
    cout << endl;
}
```

Número de identificación fiscal.
Ejemplo de uso

```
Nif unNifCualquiera = {1234567, 'L'};
if (esValido(unNifCualquiera)) {
    mostrar(unNifCualquiera);
    cout << endl;
}
```

Horas. Fichero de interfaz horas.h

```
/*
 * Definición del tipo de dato Hora
 * que representa la información de
 * una hora como hora minutos y
 * segundos.
 */
struct Hora {
    int horas, minutos, segundos;
};
```

Horas. Fichero de interfaz horas.h

```
/*
 * Pre:  Los valores de los campos horas,
 *       minutos y segundos del parámetro
 *       «unaHora» son nulos o positivos.
 * Post: Ha modificado el valor de «unaHora»
 *       para que, representando la misma
 *       cantidad de tiempo que al principio,
 *       los campos minutos y segundos
 *       estén en el intervalo [0, 60).
 */
void ajustar(Hora& unaHora);
```

Horas. Fichero de interfaz horas.h

```
/*
 * Pre:  Los valores de los campos horas,
 *       minutos y segundos del parámetro
 *       «unaHora» son nulos o positivos.
 * Post: Ha devuelto la hora correspondiene a
 *       «unaHora», después de que haya
 *       transcurrido un tiempo de «segundos»
 *       segundos.
 */
Hora sumarSegundos(const Hora unaHora,
                   const int segundos);
```

Horas. Fichero de interfaz horas.h

```
/*
 * Pre:  Los valores de los campos horas, minutos y
 *       segundos de los parámetros «horaAnterior» y
 *       «horaPosterior» son nulos o positivos y
 *       «horaAnterior» es cronológicamente anterior
 *       a «horaPosterior».
 * Post: Ha devuelto la cantidad de tiempo
 *       transcurrida entre «horaAnterior» y
 *       «horaPosterior», expresada en segundos.
 */
int calcularTiempoTranscurrido(const Hora horaAnterior,
                               const Hora horaPosterior);
```

Horas. Fichero de interfaz horas.h

```
/*
 * Pre:  ---
 * Post: Ha escrito el valor de
 *       «unaHora» en pantalla, con
 *       el formato "27:59:59" o
 *       "03:04:05".
 */
void mostrar(const Hora unaHora);
```

Horas.
Fichero de implementación horas.cpp

```
#include <iostream>
#include <iomanip>
#include "hora.h"
using namespace std;
```

Horas.
Fichero de implementación horas.cpp

```
/*
 * Pre:  Los valores de los campos horas, minutos y segundos del
 *       parámetro «unaHora» son nulos o positivos.
 * Post: Ha modificado el valor de «unaHora» para que, representando
 *       la misma cantidad de tiempo que al principio, los campos
 *       minutos y segundos estén en el intervalo [0, 60).
 */
void ajustar(Hora& unaHora) {
    unaHora.minutos += unaHora.segundos / 60;
    unaHora.segundos = unaHora.segundos % 60;

    unaHora.horas += unaHora.minutos / 60;
    unaHora.minutos = unaHora.minutos % 60;
}
```

Horas.
Fichero de implementación horas.cpp

```
/*
 * Pre:  Los valores de los campos horas, minutos y segundos del
 *       parámetro «unaHora» son nulos o positivos.
 * Post: Ha devuelto la hora correspondiene a «unaHora», después
 *       de que haya transcurrido un tiempo de «segundos»
 *       segundos.
 */
Hora sumarSegundos(const Hora unaHora, const int segundos) {
    Hora resultado = unaHora;
    resultado.segundos += segundos;
    ajustar(resultado);
    return resultado;
}
```

Horas.
Fichero de implementación horas.cpp

```
/*
 * Pre:  Los valores de los campos horas, minutos y segundos de
 *       los parámetros «horaAnterior» y «horaPosterior» son
 *       nulos o positivos y «horaAnterior» es cronológicamente
 *       anterior a «horaPosterior».
 * Post: Ha devuelto la cantidad de tiempo transcurrida entre
 *       «horaAnterior» y «horaPosterior», expresada en
 *       segundos.
 */
int calcularTiempoTranscurrido(const Hora horaAnterior,
                               const Hora horaPosterior) {
    return segundosTotales(horaPosterior)
         - segundosTotales(horaAnterior);
}
```

Horas.
Fichero de implementación horas.cpp

```
/*
 * Pre:  Los valores de los campos horas, minutos y
 *       segundos del parámetro «unaHora» son nulos
 *       o positivos.
 * Post: Ha devuelto la cantidad de tiempo
 *       correspondiente a «h» expresada en
 *       segundos.
 */
int segundosTotales(const Hora h) {
    return h.horas * 3600 + h.minutos * 60
         + h.segundos;
}
```

Horas.
Fichero de implementación horas.cpp

```
/*
 * Pre:  ---
 * Post: Ha escrito el valor de «unaHora» en pantalla, con
 *       el formato "27:59:59" o "03:04:05".
 */
void mostrar(const Hora unaHora) {
    cout << setfill('0');
    cout << setw(2) << unaHora.horas << ":"
         << setw(2) << unaHora.minutos << ":"
         << setw(2) << unaHora.segundos;
    cout << setfill(' ');
}
```

Horas. Ejemplo de uso

```
Hora h1 = {0, 106, 39};
ajustar(h1);
mostrar(h1);
cout << endl;

Hora h2 = sumarSegundos(h1, 3600);
mostrar(h2);
cout << endl;

cout << calcularTiempoTranscurrido(h1, h2)
     << endl;
```

Ciudadanos

- Gestionar información relativa a ciudadanos de un determinado lugar:
  - Nombre
  - Apellidos
  - NIF
  - Fecha de nacimiento
  - Estado civil (casado, no casado)
  - Sexo

Ciudadano. Interfaz

```
#include "nif.h"
#include "fecha.h"

/*
 * Longitudes máximas del nombre y
 * los apellidos de un ciudadano
 */
const int MAX_LONG_NOMBRE = 24;
const int MAX_LONG_APELLIDOS = 24;
```

Ciudadano. Interfaz

```
/*
 * Definición del tipo de dato Ciudadano que
 * representa la información relevante de un
 * ciudadano: nombre y apellidos, número de
 * identificación fiscal, fecha de nacimiento,
 * estado civil y sexo
 */
 struct Ciudadano {
      char nombre[MAX_LONG_NOMBRE];
      char apellidos[MAX_LONG_APELLIDOS];
      Nif nif;
      Fecha nacimiento;
      bool estaCasado;
      bool esMujer;
};
```

Ciudadano. Interfaz

```
/*
 * Pre:  ---
 * Post: Ha asignado a «cadena» el nombre
 *       completo del ciudadano «c».
 */
void nombreCompleto(const Ciudadano c,
                    char cadena[]);

/*
 * Pre:  ---
 * Post: Ha mostrado los datos del
 *       ciudadano «c» en la pantalla.
 */
void mostrar(const Ciudadano c);
```

Ciudadano. Implementación

```
#include <cstring>
#include <iostream>
#include "ciudadano.h"
using namespace std;

/*
 * Pre:  ---
 * Post: Ha asignado a «cadena» el nombre completo del
 *       ciudadano «c».
 */
void nombreCompleto(const Ciudadano c, char cadena[]) {
    strcpy(cadena, c.nombre);
    strcat(cadena, " ");
    strcat(cadena, c.apellidos);
}
```

Ciudadano. Implementación

```
/*
 * Pre:  ---
 * Post: Ha mostrado los datos del ciudadano «c» en la pantalla.
 */
void mostrar(const Ciudadano c) {
    char marcaGenero = 'o';
    if (c.esMujer) {
        marcaGenero = 'a';
    }

    char cadenaNombreCompleto[MAX_LONG_NOMBRE + MAX_LONG_APELLIDOS];
    nombreCompleto(c, cadenaNombreCompleto);
    cout << "Ciudadan" << marcaGenero << ": " << cadenaNombreCompleto
         << endl;
    cout << "NIF: "; mostrar(c.nif); cout << endl;
    cout << "Nacid" << marcaGenero << " el "; mostrar(c.nacimiento);
    cout << endl;
    if (c.estaCasado) {
        cout << "Casad" << marcaGenero << endl;
    }
    else {
        cout << "Solter" << marcaGenero << endl;
    }
}
```

Ciudadano. Ejemplos de utilización

```
Ciudadano rey;
strcpy(rey.nombre, "Felipe");
strcpy(rey.apellidos, "Borbón Grecia");
rey.nif.dni = 15;
rey.nif.letra = 'S';
rey.nacimiento.dia = 30;
rey.nacimiento.mes = 1;
rey.nacimiento.agno = 1968;
rey.esMujer = false;
rey.estaCasado = true;
mostrar(rey); cout << endl;
```

Ciudadano. Ejemplos de utilización

```
Ciudadano reinaEmerita
          = { "Sofia", "Grecia Dinamarca",
              {11, 'B'}, {2, 11, 1938},
              true, true
            };
mostrar(reinaEmerita);
cout << endl;
```

Ciudadano. Ejemplos de utilización

```
Ciudadano reinaEmerita
          = { "Sofia",          // nombre
              "Grecia Dinamarca",  // apellidos
              {11, 'B'},        //  NIF
              {2, 11, 1938}, // fecha nacimiento
              true,             // esMujer
              true              // estaCasada
            };
mostrar(reinaEmerita);
cout << endl;
```

Ciudadano. Ejemplos de utilización

```
Ciudadano princesa = {"Leonor",
                      "Borbón Ortiz"};
princesa.nif = {16, 'Q'};
princesa.nacimiento = {19, 6, 2014};
princesa.esMujer = true;
princesa.estaCasado = false;
mostrar(princesa);
cout << endl;
```

Registros

- Permiten a los programadores trabajar con los datos del nuevo tipo conociendo cómo se ha representado el nuevo tipo.
- Esto significa que el efecto de cualquier cambio en la representación del tipo puede extenderse **a todo** el código que haga uso del tipo.

Registros

- ¿Qué pasaría en el código de un proyecto grande si cambiamos la forma de representar las fechas?
- ¿Y si cambiáramos la forma de representar a los ciudadanos?

Registro fecha. Alternativa 1

```
struct Fecha {
    int dia, mes, agno;
};
```

Registro fecha. Alternativa 2

```
struct Fecha {
    int aaaammdd;
};
```

Registro fecha. Alternativa 3

```
const int INDICE_DIA = 0;
const int INDICE_MES = 1;
const int INDICE_AGNO = 2;

struct Fecha {
    int campos[3];
};
```

Registro Ciudadano. Alternativa 1

```
struct Ciudadano {
      char nombre[MAX_LONG_NOMBRE];
      char primerApellido[MAX_LONG_APELLIDOS];
      char segundoApellido[MAX_LONG_APELLIDOS];
      Nif nif;
      Fecha nacimiento;
      bool estaCasado;
      bool esMujer;
};
```

Registro Ciudadano. Alternativa 2

```
struct Ciudadano {
      char nombre[MAX_LONG_NOMBRE];
      char primerApellido[MAX_LONG_APELLIDOS];
      char segundoApellido[MAX_LONG_APELLIDOS];
      Nif nif;
      Fecha nacimiento;
      int agnosMatrimonio;
      bool esMujer;
};
```

Registro Ciudadano. Alternativa 3

```
struct Ciudadano {
      char nombre[MAX_LONG_NOMBRE];
      char primerApellido[MAX_LONG_APELLIDOS];
      char segundoApellido[MAX_LONG_APELLIDOS];
      Nif nif;
      Fecha nacimiento, matrimonio;
      int agnosMatrimonio;
      bool esMujer;
};
```

Clases y objetos en C++

- Permiten representar la misma información que los registros.
- *Encapsulamiento:* permiten de forma cómoda ocultar sus campos o *atributos* fuera del módulo en el que se definen.
- *Comportamiento:* requieren de la definición de un mayor número de funciones *(métodos)* para gestionar la información que contienen.
- Facilitan el cambio en la representación de un tipo de datos, minimizando el impacto en el código externo al módulo afectado.