Estructuras de datos tipo Registro

registro.campo

1. Estructuras o Registros.

Antecedentes.

- Los tipos de datos primitivos (variables de tipo entero, real, carácter y lógico) permiten almacenar *"Un DATO de un único tipo"*.
- Por otro lado, las estructuras de tipo array (vectores y matrices) son variables indexadas que almacenan *"Varios DATOS, pero también de un único tipo"*.
- Sin embargo, las estructuras tipo registro, combinan las características de los datos primitivos y de los arrays, es decir: *"Varios DATOS, pero de distintos tipos"*.

Es decir la finalidad de las estructuras de tipo registro es agrupar una o mas variables, generalmente de diferentes tipos, bajo un mismo nombre.

Estructuras:

En los compiladores las estructuras son conocidas como ***registros***. Están formados por miembros **de datos heterogéneos** (campos).

Dependencia funcional: Los Registros almacenan datos relacionados fuertemente entre sí.

Esto es, los datos de un cliente por ejemplo, es un registro que incluye los campos: Codigo, Razon Social, Direccion, Fax y RUC.

Registro crédito

Fig. 1 Jerarquía de registros: Miembros de datos.

Jerarquía y Complejidad. Los registros constan a su vez de entidades de nivel más bajo denominados **miembros de datos** o **campos**, que pueden pertenecer a un tipo de dato primitivo o a datos estructurados complejos como los array e inclusive a otro registro.

Registro empleado

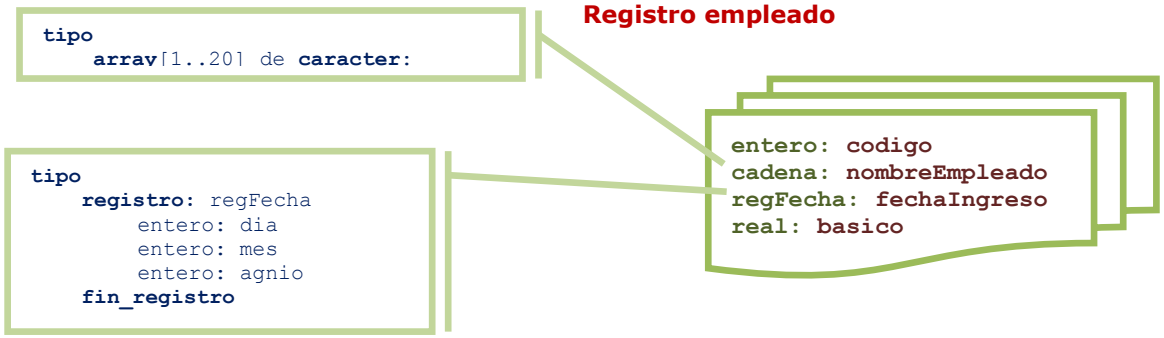


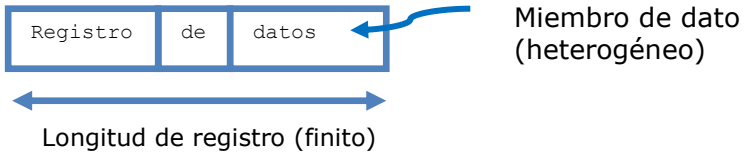
Fig. 2 Complejidad de los registros.

2. Estructuras tipo Registro.

Un registro es una colección de información, normalmente relativa a una entidad en particular (empleados, artículos, proveedores, etc.), que presenta las siguiente características:

Características.

- **Modelo de Datos.** Un registro es una colección de miembros de datos (campos) definidos con un tipo de datos primitivo o estructurado (Regla de Modelo de Datos) y por tanto son heterogéneos.
- **Dependencia Funcional.** Los campos en un registro al estar lógicamente relacionados, pueden ser tratados como una unidad. Por tanto, sus miembros de datos componen un solo bloque de información.
- **Estructuración Finita y Heterogénea.** Los registros organizados en campos se denominan **registros de datos**, cuyos miembros son tipos variados y definidos previamente. Por tanto son Heterogéneos y finitos. Los registros son de longitud fija.



Estructura tipo registro (Record.field)

2. Miembros de datos o Campos.

Los miembros de datos se agrupan en unidades básicas llamados campos (fields). Estos están relacionados por dependencia funcional y definidos por las reglas del modelo de dato.

Así por ejemplo:

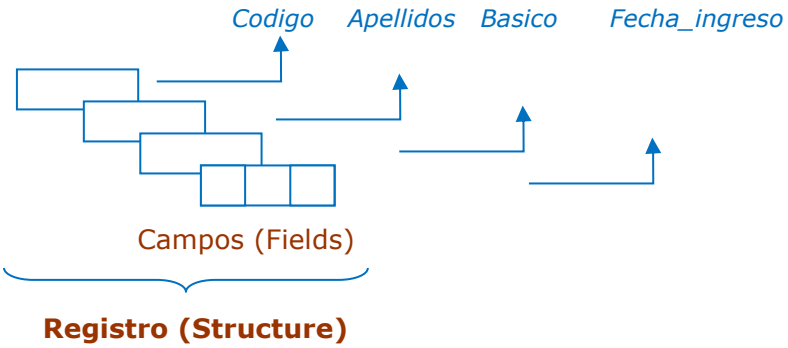
Registro: empleado

código	de empleado de tipo numérico,
nombre	de empleado de tipo cadena,
sueldo	básico de empleado de tipo real,
edad	de tipo entero,
fechaNac	de nacimiento (dia-mes-año), etc.

Algunos de estos miembros podrían ser a su vez estructuras. Asi por ejemplo, La fecha de nacimiento es una estructura con los datos de tipo entero: dia, mes y año en el formato: dd-mm-aa.

Definición.

Es una colección de miembros de datos heterogéneos y longitud fija. Implementan la regla de MODELO de DATOS y Dependencia Funcional. Un miembro de dato o campo esta definido por su identificador, tamaño, y tipo de dato (entero, real, cadena, lógico, array, etc).



Sintaxis: Pseudocodigo.

```
tipo
        registro: <tipo_registro>
                <tipo_dato>: nombre_campo1
                <tipo_dato>: nombre_campo2
                //...
                <tipo_dato>: nombre_campo_n
        fin_registro

var
        <tipo_registro>:  var_tipoRegistro
```

Acceso a los miembros de una estructura de tipo registro: (operador punto)

Sintaxis:

```
var_tipoRegistro . nombre_campo
```

Ejemplos comentados. Representación en pseudocódigo de estructuras tipo registro.

Ejemplo 1. Definir una estructura para registrar los datos, Codigo, monto de crédito y el estado de cancelación del crédito de Clientes.
La definición de la estructura de un registro de clientes, seria:

```
tipo
        registro: regCliente
                entero: id                  //3 digitos entre 100 - 999
                real: montoCredito          //monto de crédito aprobado
                logico: cancelado           //estado crédito verdadero/falso
        fin_registro
var
        regCliente:  cliente
```

Ejemplo 2. Definir registro para los siguientes datos de un Alumno: Identificador, apellidos, nombre y 5 notas. La definición de la estructura de un registro de alumnos, seria.

```
tipo
        array[1..5] de real: arrNota
        array[1..30] de caracter: cadena
tipo
        registro: regAlumno
                entero: id                  //3 digitos autogenerado
                cadena: apellidos           //25 caracteres de ancho
                cadena: nombre              //15 caracteres
                arrNota: nota               //array de 5 notas de practica
        fin_registro
var
        regAlumno:  alumno
```

Ejemplo 3. Definir registro de empleados. Los datos a manejar son: código, nombre, sueldo, edad, fecha de nacimiento, y fecha de ingreso

La definición de la estructura de un registro de empleados, seria:

```
tipo
        array[1..30] de caracter: cadena
tipo
        registro: regFecha
                entero: dia         //2 digitos entre 01-31
                entero: mes         //2 digitos entre 01-12
                entero: agno        //4 digitos entre 1000-9000
        fin_registro
tipo
        registro: regEmpleado
                entero : codigo             //4 digitos
                cadena : nombre             //15 caracteres
                real   : sueldo             //(6,2) 3 enteros, 2 decimales
                entero : edad               //valor entre 18 y 65
                regFecha: fechaNac          //estructura tipo reg.
                regFecha: fechaIng          //estructura tipo reg.
        fin_registro
var
        regRmpleado:  empleado
```

IMPORTANTE

Los miembros de datos o campos de un registro, además de las *reglas de Modelo de Datos* (tipos de datos) deben tener *reglas de Restricción*, para implementar la integridad de datos.
Asi por ejemplo: En una empresa el **número de trabajadores** es un valor discreto y no continuo. Es decir, por regla de Modelo de Datos, existen 24 trabajadores y no 24.5. Esto es el número de trabajadores debe ser definido por modelo de datos como entero. Sin embargo en muchos de los casos no basta con las reglas de modelo de datos para asegurar la integridad de los datos. Las reglas de restricción refuerzan el modelo de datos como en la cantidad de electrodomésticos vendidos por ejemplo, que por modelo de datos debe ser definido como entero y por regla de restricción las ventas deben ser siempre positivas(1 o mas, y no ventas nulas o negativas). Por tanto la regla de restricción a implementar para este campo seria: **cantidad > 0**.

3. Operaciones sobre registros.

Asignaciones de Lectura y escritura de registros.

Pseudocodigo

- 3.1. La lectura o entrada de datos del registro seria mediante el procedimiento:

```
llamar_a leerRegistro(empleado)
```

y su implementación:

```
procedimiento leerRegistro(S reg_empleado: emp )
var
inicio
       leer(emp.codigo)
       leer(emp.nombre)
       leer(emp.sueldo)
       leer(emp.edad)
       leer(emp.fechaNacimiento.dia)
       leer(emp.fechaNacimiento.mes)
       leer(emp.fechaNacimiento.agno)
fin_procedimeinto
```

- 3.2. La escritura o consulta del contenido del registro seria mediante el procedimiento:

```
llamar_a escribirRegistro(empleado)
```

y su implementación:

```
procedimiento escribirRegistro(E reg_empleado: emp )
var
inicio
       escribir(emp.codigo)
       escribir(emp.nombre)
       escribir(emp.sueldo)
       escribir(emp.edad)
       escribir(emp.fechaNacimiento.dia)
       escribir(emp.fechaNacimiento.mes)
       escribir(emp.fechaNacimiento.agno)
fin_procedimeinto
```

- 3.2. Asignar una estructura a otra estructura, utilizando el operador de asignación:

```
estructuraDestino ← estructuraOrigen   //asignacion
```

y su implementación:

```
var
       reg_empleado: empleado, trabajador
inicio
       llamar_a leerRegistro(empleado)
       trabajador ← empleado
fin
```

IMPORTANTE

Cuando se asigna una estructura a otra estructura se copian uno a uno todos los miembros de la estructura fuente a la estructura destino. Esto es, se duplica la estructura. Observe que para que sea posible la asignación los registros deben tener la misma definición (registros del mismo tipo).

Ejemplos comentados.

Problema 1. Se necesita diseñar una ficha o etiqueta para una agenda de contactos de los cuales se tiene la siguiente información: Apellido contacto y teléfono celular.

Analisis.

Los miembros de datos para cada contacto se pueden organizar en una estructura como la siguiente:

```
Registro agenda
        Apellido contacto       cadena      //20 caracteres de ancho
        Telefono celular        entero      //9 digitos
```

Especificaciones de E/S

Entrada: miembros de datos en estructura tipo registro
Salida : Ficha o etiqueta de contacto

Pseudocodigo

```
Algoritmo. Ficha de contactos
tipo
        array[1..30] de caracter: cadena
tipo
        registro: regContacto
                cadena: apellido    //apellido paterno de contacto
                entero: tlfCelular  //9 digitos (enteros positivos)
        fin_registro
var
        regContacto: contacto, agenda

inicio

        llamar_a leerRegistro(contacto)
        agenda ← contacto  //asignar estructura
        llamar_a escribirRegistro(agenda)
fin


//definir leerRegistro
procedimiento leerRegistro(S regContacto: reg)
var
        const TITULO="Ingrese contacto y Tlf: "
inicio
                escribir(TITULO)
                leer(reg.apellido)
                leer(reg.tlfCelular)
fin_procedimeinto


//definir escribirRegistro
procedimiento escribirRegistro(E regContacto: reg)
var
        const TITULO="Datos Contacto"
inicio
        escribir(TITULO)
        escribir(reg.apellido)
        escribir(reg.tlfCelular)
fin_procedimeinto
```

Problema 2. Se necesita calcular el promedio semestral de un alumno del cual se conoce: Codigo, Apellido y sus tres notas: PP (promedio practicas), ExaPar. (examen parcial) y ExaFin. (examen final). Presentar su ficha académica con una anotación de aprobado o desaprobado según su promedio.

Análisis.
Los miembros de datos se pueden definir en una estructura como la siguiente:

```
Registro alumno
        Codigo          entero              //3 dígitos
        Apellido        cadena              //20 caracteres de ancho
        array de notas con 3 elementos
                PP      real                //2.1 (2 enteros 1 decimal)
                EP      real                // en escala vigesimal
                EF      real                //y en el rango: 0-20
```

La entrada por teclado provee los datos a la estructura, a partir de los cuales las notas son procesadas para el cálculo del promedio.

$$\text{Promedio} = \frac{\Sigma n}{3}$$

Especificaciones de E/S

Entradas: miembros de datos de tipo registro
Salidas : promedio semestral con anotación (Aprobado/Desaprobado) en ficha académica.

Pseudocodigo

```
Algoritmo. Ficha academica
tipo
       array[1..20] de caracter: cadena
       array[1..3] de real: arrNotas
tipo
       registro regAlumno
              entero: codigo      //identificador
              cadena: apellido    //apellido paterno
              arrNotas: notas     //vector de 3 notas(PP, EP, EF)
       fin_registro
var
       regAlumno: alumno

inicio

       llamar_a leerRegistro(alumno)
       llamar_a escribirRegistro(alumno)
fin



//Definir leerRegistro
procedimiento leerRegistro(S regAlumno: alu)
var
       const TITULO="Datos alumno"
inicio
       escribir(TITULO)
       escribir("Codigo   : "): leer(alu.codigo)
       escribir("Apellido: "): leer(alu.apellido)
       escribir("PPract   : "): leer(alu.notas[1])
       escribir("ExPar    : "): leer(alu.notas[2])
       escribir("ExFin    : "): leer(alu.notas[3])
fin_procedimiento
```

```
procedimiento escribirRegistro(E regAlumno: alu )
var
       const TITULO="Registro academico"
       entero: n
       real:   suma, prom
       cadena: anotación
inicio
              anotación ← "[Desaprobado]"
              escribir(TUTULO)
              escribir("Codigo  : ", alu.codigo)
              escribir("Apellido: ", alu.apellido)
              escribir("Notas PP: ", alu.nota.PP)
              escribir("      EP: ", alu.nota.EP)
              escribir("      EF: ", alu.nota.EF)
              //calcular promedio
              desde n ← 1 hasta 3 hacer
                     suma ← suma + alu.nota[n]
              fin_desde
              prom ← suma / 3
              si prom >= 10.5 entonces
                     anotación ← "[Aprobado]"
              fin_si
              escribir(anotación, " Promedio: ", prom)
fin_procedimiento
```

3.2. Implementación en Código c++

```
/* Name: Ficha_empleado.cpp
   Author: J.Medianero.A
   Description:   Operaciones de Lectura / Escritura y
   Acceso a miembros de estructura de registros     */
#include <cstdlib>
#include <iostream>

struct regEmpleado{
       int codigo;
       char nombreEmp[20];
       float sueldo;
       };

//declaraciones prototipo
void leerRegistro(struct regEmpleado *);
void escribirRegistro(struct regEmpleado);

using namespace std;

int main(int argc, char* argv[]){
      struct regEmpleado empleado, trabajador;

      //llamar_a
      leerRegistro(&empleado);
      //asignar registro del mismo tipo
      trabajador = empleado;
      //llamar_a
      escribirRegistro(trabajador);

      system("PAUSE");
      return EXIT_SUCCESS;
}//:~
```

```
/* OBS. La sintaxis C++, para el acceso a miembros de datos de una estructura
 *      cuando esta es un parámetro de referencia, utiliza el operador
 *      flecha: -> (formado por el guion y el signo mayor que)
*/

//definir rutina: leerRegistro
void leerRegistro(struct regEmpleado *emp){
    cout << "Codigo: "; cin >> emp->codigo;
    cout << "Nombre: "; cin >> emp->nombre;
    cout << "Sueldo: "; cin >> emp->sueldo;
}



//definir procedimeinto escribirRegistro
void escribirRegistro(struct regEmpleado emp){
     cout << setw(4);                  //establece ancho de 4 posiciones
     cout << emp.codigo << " ";
     cout << setw(20);                 //establece ancho de 20 posiciones
     cout << emp.nombre << " ";
     cout << setw(6);                  //establece ancho de 6 posiciones
     //activa formato de coma flotante con dos decimales
     cout << fixed << setprecision(2);
     cout << emp.sueldo << endl;
}
```

Siguiente tema:

4. arrays de estructuras de tipo Registro.

Definir arrays de Registros. (Aplicar arrays sobre registros).