UCA
Universidad de Cádiz

Introducción a la Programación
Grado en Ingeniería Informática

Teoría - Curso 2015-2016

Contenido 5 – Tipos de Datos

Contenido 5.- Tipos de datos

5.1.- Tipos de datos Estructurados

- Un tipo de dato estructurado o una estructura de datos es un conjunto de datos que se trata como una sola unidad pero que a su vez permite referenciar independientemente sus componentes.
- Las estructuras de datos que estudiaremos están compuestas de datos simples, pero veremos cómo, en ciertas ocasiones, el problema a resolver requiere el uso de estructuras aún más complejas.
- La mayoría de los lenguajes de programación permiten al programador definir estructuras de datos cuyos elementos son a su vez tipos de datos estructurados.
- La elección de la estructura de datos adecuada, dependerá en primer lugar del problema a resolver y en segundo lugar del lenguaje de programación que estemos utilizando.

5.1.- Tipos de datos Estructurados

- Clasificación de las Estructuras de Datos:
  - Atendiendo al momento en que se reserva la memoria:
    - **Estáticas**: El espacio de memoria ocupado por la estructura es fijo, se define en tiempo de compilación y no puede ser modificado durante la ejecución del programa.
    - **Dinámicas**: El espacio de memoria ocupado por la estructura es dinámico, se define en tiempo de ejecución y además puede ser variado durante la ejecución del programa en función de las necesidades del mismo.
  - Atendiendo al tipo de los datos que la forman
    - **Homogéneas**: estructuras que están compuestas por componentes del mismo tipo.
    - **Heterogéneas**: estructuras que están compuestas por elementos de distinto tipo.

5.1.- Tipos de datos Estructurados

- Clasificación de las Estructuras de Datos:
  - Estáticos
    - vector/matriz
    - registro
    - cadena de caracteres
    - fichero (archivo)
  - Dinámicos
    - lista
    - pila
    - árbol
    - Grafo
- Los tipos de datos estructurados los definiremos en la sección de definición de tipos, encabezada por la palabra ***tipo***.

5.2.- Vectores y Matrices

5.2.1.- Vectores o matrices unidimensionales.

- Un vector o matriz unidimensional es un conjunto finito de elementos del mismo tipo, que están almacenados consecutivamente y que pueden ser identificados de forma independiente.

- Definición de un vector:

Tipo

vector [tamaño] **de** <tipo de dato> **:** <identificador_del_tipo_vector>

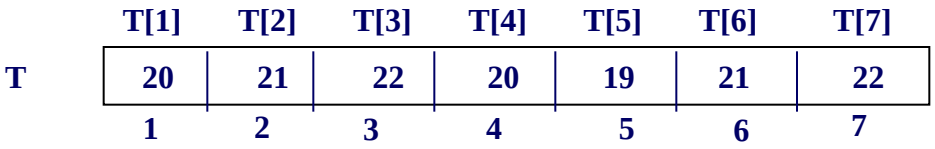
- Cuando no conocemos el número de elementos que tendrá el vector suponemos un tamaño máximo de elementos que podrá contener

5.2.- Vectores y Matrices

5.2.1.- Vectores o matrices unidimensionales.

- Ejemplo:

```
const
    DIAS = 7
tipo
    vector [DIAS] de entero : Temperatura
var
    Temperatura: T
```



- **Nombre o identificador del vector**: **T**
- **Tamaño del vector**: Se fija en el momento de su definición. 7.
- **Índice**: Es el número que identifica las posiciones del vector: [1, 2, 3, 4, 5, 6, 7]. (En algunos lenguajes de programación los índices comienzan en 0 en vez de en 1, por ejemplo en C).
- **Valor almacenado o dato**: [20, 21, 22, 20, 19, 21, 22].
- **Tipo de dato base**: entero
- **Acceso a los elementos**: Accedemos a los elementos a través de su índice utilizando la siguiente notación: vector[posición], donde *vector* es el identificador del vector y entre corchetes aparece la posición a la que se desea acceder.

5.2.- Vectores y Matrices

5.2.1.- Vectores o matrices unidimensionales.

- Operaciones con vectores:
  - **Asignación**: T[2]← 15

    En algunos lenguajes de programación es posible realizar la asignación completa de un vector. Por ejemplo si A y B son dos vectores del mismo tipo base y tamaño, A ← B copiaría el contenido completo del vector B en el vector A.
  - Recorrido de un vector:

    Lectura de los valores de T desde teclado e inserción en el vector

```
desde i ← 1 hasta DIAS hacer
    escribir (“inserte el elemento”, i)
    leer (T[i])
fin_desde
```

Escritura por pantalla del contenido del vector

```
desde i ← 1 hasta DIAS hacer
    escribir (T[i])
fin_desde
```

5.2.- Vectores y Matrices

5.2.1.- Vectores o matrices unidimensionales.

- Operaciones con vectores:
  - Desplazamiento de los elementos de un vector:

```
……..……
escribir(“Introduzca el valor y la posición dónde se desea insertar:
    ”)
leer (n, pos)
i ← 7
mientras i ≥ pos hacer
    V[i+1]← V[i]
    i ← i - 1
fin_mientras
V[pos]← n
……..……
```

5.2.- Vectores y Matrices

5.2.1.- Vectores o matrices unidimensionales.

- Ejemplo: algoritmo que almacena en un vector de 20 elementos los valores correspondientes a las notas de 20 alumnos de una clase. Además el algoritmo debe calcular la nota media y escribirla por pantalla.

```
Algoritmo media_vector
    const
        MAX = 20
    tipo
        vector [MAX] de real : Nota
  Principal
    var
        Nota : V
        real: media
  inicio
        leer_vector(V)
        media← calcula_media(V)
        escribir (“La media es”, media)
  fin_principal
Fin_algoritmo
```

5.2.- Vectores y Matrices

5.2.1.- Vectores o matrices unidimensionales.

```
//Cabecera: leer_vector(S Nota: V).
//Precondición: V es una variable de tipo Nota.
//Postcondición: inicializa V con los datos introducidos por el usuario.

procedimiento leer_vector(S Nota: V)
var
    entero: i
inicio
    desde i← 1 hasta MAX hacer
        escribir(“Introduzca el elemento: “, i)
        leer(V[i])
    fin_desde
fin_procedimiento
```

5.2.- Vectores y Matrices

5.2.1.- Vectores o matrices unidimensionales.

```
//Cabecera: real calcula_media(E Nota: V)
//Precondición: V es una variable de tipo Nota que debe estar inicializada.
//Postcondición: devuelve la media de todos los elementos del vector.

real funcion calcula_media(E Nota: V)
var
     real: suma, media
inicio
     suma ← 0
     desde i← 1 hasta MAX hacer
          suma← suma + V[i ]
     fin_desde
     media← suma/MAX
     devolver media
fin_funcion
```

5.2.- Vectores y Matrices

5.2.2.- Matrices multidimensionales.

- Una matriz multidimensional es una estructura homogénea, en la cual para hacer referencia a un elemento necesitamos dos o más índices dependiendo de su dimensión.
- Lo más frecuente es utilizar matrices bidimensionales, también denominadas tablas, que tienen dos dimensiones y se referencia cada uno de sus elementos con dos índices.
- Desde un punto de vista conceptual se puede entender una matriz como un vector de vectores, es decir, un vector en el cual cada uno de sus elementos es a su vez otro vector.
- En una matriz bidimensional los elementos ocupan un lugar en una fila y en una columna.

5.2.- Vectores y Matrices

5.2.2.- Matrices multidimensionales.

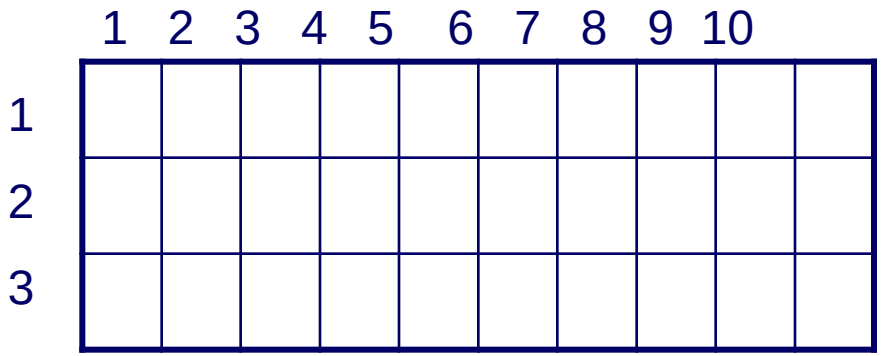
- Definición de una matriz:

tipo

matriz[tam_1,tam_2…tam_n] **de** <tipo de dato>**:**<identificador_del_tipo_matriz>

- Ejemplo:

```
tipo
    matriz [3,10] de entero : notas_asignaturas
var
    notas_asignaturas: M
```



5.2.- Vectores y Matrices

5.2.2.- Matrices multidimensionales.

- Para referenciar a un elemento es necesario conocer el índice correspondiente a cada dimensión utilizando la notación M[ind_1][ind_2] … [ind_n], donde n es la dimensión de la matriz. Para el ejemplo anterior, la matriz M tiene dos dimensiones y tamaño 3x10. Para acceder al elemento (*), que se encuentra en la fila 2 y en la columna 3, usamos la notación M[2][3] o M[2,3].
- Operaciones con matrices:
  - Asignación: M[2][3]← 8
  - Recorrido de una matriz:

```
desde i ← 1 hasta 3 hacer
    desde j ← 1 hasta 10 hacer
            M[i][j]← 0
    fin_desde
fin_desde
```

5.2.- Vectores y Matrices

5.2.2.- Matrices multidimensionales.

- Ejemplo: Algoritmo que suma dos matrices de dimensión *f*×*c*, siendo *f* y *c* introducidos por teclado y menores o iguales que 100.

```
Algoritmo suma
const
    MAX = 100
tipo
    matriz [MAX,MAX] de entero : Tabla
Principal
  var
    entero: i,j,f,c
    Tabla: A,B,S
  inicio
    escribir("Introduzca el número de filas y columnas de la matriz: ")
    leer(f,c)
    leer_matriz(A,f,c)
    leer_matriz(B,f,c)
    calcula_suma(A,B,f,c,S);
    escribir_matriz(S)
  fin_principal
Fin_algoritmo
```

5.2.- Vectores y Matrices

5.2.2.- Matrices multidimensionales.

```
//Cabecera: leer_matriz(S Tabla: M, E entero: f, E entero: c).
//Precondición: M es una variable de tipo Tabla, f y c son dos variables enteras que
      deben estar inicializadas.
//Postcondición: M se inicializa a los valores introducidos por el usuario.

procedimiento leer_matriz (S Tabla: M, E entero: f, E entero: c )
var
      entero i, j
inicio
      desde i ← 1 hasta f hacer
            desde j ← 1 hasta c hacer
                escribir("Introduzca el elemento de la posición ", i, j)
                  leer(M[i][j])
            fin_desde
      fin_desde
fin_procedimiento
```

5.2.- Vectores y Matrices

5.2.2.- Matrices multidimensionales.

```
//Cabecera: calcula_matriz(E Tabla: A, E Tabla: B, E entero: fil, E entero: col,
//  S Tabla: C).
//Precondición: A, B y C son matrices de tipo Tabla, fil y col son dos variables
//  enteras que deben estar inicializadas.
//Postcondición: C contiene la matriz suma de A y B.

procedimiento calcula_suma(E Tabla: A, E Tabla: B, E entero: fil, E entero: col, S Tabla: C)
var
      entero i, j
inicio
       desde i ← 1 hasta fil hacer
              desde j ← 1 hasta col hacer
                            C[i][j]← A[i][j]+B[i][j]
              fin_desde
       fin_desde
fin_procedimiento
```

5.2.- Vectores y Matrices

5.2.2.- Matrices multidimensionales.

```
//Cabecera: escribir_matriz(E Tabla: M, E entero: f, E entero: c).
//Precondición: M es una variable de tipo Tabla, f y c son dos variables enteras.
//   Todas las variables deben estar inicializadas.
//Postcondición: Se visualiza el contenido de la matriz.

procedimiento escribir_matriz (E tabla: M, E entero: f, E entero: c )
var
     entero i, j
inicio
     desde i ← 1 hasta f hacer
          desde j ← 1 hasta c hacer
               escribir(M[i][j])
          fin_desde
     fin_desde
fin_procedimiento
```

5.3.- Cadenas de caracteres

- Una cadena de caracteres es una secuencia de caracteres consecutivos
- Existen lenguajes de programación que incorporan el tipo cadena como uno de sus tipos básicos, denominándolo generalmente *string*.
- Otros lenguajes no incorporan esta posibilidad y el programador debe definir y utilizar las cadenas de caracteres como vectores de tipo carácter.
- Una constante de cadena será cualquier secuencia de caracteres encerrada entre comillas dobles
- Es posible definir cualquier dato de este tipo, tanto variables como constantes.
- La longitud de una cadena es el número de caracteres que contiene.
- La cadena vacía es aquella que no contiene caracteres, y por tanto su longitud es cero, y se nota “” .
- Definición: **var**
  cadena: cad

5.3.- Cadenas de caracteres

- Operaciones con cadenas de caracteres
- **Hay lenguajes que proporcionan multitud de funciones de manejo de cadenas de caracteres, como por ejemplo búsqueda de un carácter en una cadena, conversión entre cadenas y números o inserción y borrado de caracteres en una cadena.**
  - **Asignación**: Podemos asignar a una variable de tipo cadena una constante de cadena.

    cad← “Hola a todos”
  - **Lectura/escritura**: Podemos utilizar las operaciones de lectura y escritura con cadenas de caracteres.

    leer(cad)

    escribir(cad)
  - **Cálculo de la longitud**: Esta función devuelve el número de caracteres que contiene la cadena.

    cad← “hola”

    n ← **longitud**(cad)

    En este ejemplo en n se almacenará 4.

5.3.- Cadenas de caracteres

- Operaciones con cadenas de caracteres
  - **Comparación**: para realizar comparaciones entre cadenas de caracteres utilizaremos el orden lexicográfico.
  - **Concatenación de cadenas**: Consiste en unir varias cadenas en una. Algunos lenguajes de programación utilizan símbolos concretos para concatenar cadenas (‘+’) y otros lenguajes utilizan una función que permite concatenar cadenas: una función del tipo *cadena concatena (cad1,cad2)* que devuelve la concatenación de cad1 y cad2:

Ejemplo:

```
var
        cadena: cad1,cad2, cad3
.............
cad1← “hola”
cad2← “¿qué tal?”
cad3←concatena(cad1,cad2)   ó   cad3←cad1+cad2
escribe(cad3)
.............
```

El ejemplo escribiría “Hola¿qué tal?”

5.4.- Registros

- Es una estructura de datos formada por un conjunto de elementos que van a contener información relativa a un mismo objeto.
- Los elementos que constituyen un registro se llaman ***campos***, y cada campo puede ser de un tipo diferente.
- En la definición del registro se especifica el identificador de cada campo y el tipo al que pertenece, que puede ser cualquiera de los definidos, incluidos los tipos compuestos
- Definición:

tipo

registro*: nombre_reg*

tipo 1: id_campo 1

tipo 2: id_campo 2

. . .

tipo n: id_campo n

fin_registro

5.4.- Registros

- **Ejemplo:**

  Para representar la información relativa a un estudiante matriculado en una asignatura concreta, se puede emplear un registro con los campos: nombre, apellidos, edad, DNI y nota.

```
tipo
registro: reg_estudiante
        cadena: apellidos, nombre
        entero: edad
        entero: dni
        entero: nota
fin_registro
Principal
  var
    reg_estudiante: estudiante
```

- Para seleccionar el campo *identificador_campo* del registro *nombre_reg* se utiliza el formato *nombre_reg.identificador_campo.*

5.4.- Registros

- Se puede realizar la asignación entre dos registros completos siempre que sean del mismo tipo registro
- Cada campo del registro puede usarse en cualquier tipo de expresión aritmética, lógica o de tipo carácter; puede ser asignado; puede ser pasado como argumento en llamadas a subalgoritmos o incluso ser devuelto como resultado de una función.
- En resumen, con los campos de un registro se pueden realizar las mismas operaciones que se suelen llevar a cabo sobre una variable del mismo tipo.
- Es frecuente definir un vector donde cada uno de sus componentes sea de un tipo registro. Ejemplo:

```
Const
  N=100
tipo
  registro: reg_estudiante
 cadena: apellidos, nombre
  entero: edad, dni, nota
  fin_registro
  vector [N] de reg_estudiante : Testudiantes
Principal
var
   TEstudiantes: estudiantes
```

Para acceder al D.N.I. del décimo estudiante del vector se utilizaría estudiantes[10].dni.

5.4.- Registros

- Un registro también puede tener campos que sean vectores o incluso otros registros, como podemos observar en el siguiente ejemplo:

```
const
  N=100
tipo
  registro: reg_fecha
        entero: dia
        cadena: mes
        entero: año
  fin_registro
  registro: reg_estudiante
        cadena: apellidos, nombre
        entero: edad
        entero: dni
        entero: nota
        reg_fecha: nacimiento
  fin_registro
   vector [N] de reg_estudiante : Testudiantes
Principal
  var
   TEstudiantes: estudiantes
```

Si queremos conocer el mes de la fecha de nacimiento del estudiante *i*, el acceso al registro en el cuerpo del algoritmo sería *estudiantes[i].nacimiento.mes*.

5.5.- Ficheros

- Estructura de datos no acotada y almacenada en memoria masiva o soporte externo.
- Ventajas:
  - La información es permanente.
  - La limitación es la del soporte.
- Para trabajar con un fichero almacenado en memoria masiva definiremos una variable tipo fichero en memoria principal a la que le asociaremos dicho fichero en la operación de apertura del mismo.
- Tipos de ficheros:
  - **Ficheros secuenciales**: se accede a los elementos secuencialmente (son los que estudiaremos). Los ficheros o archivos disponen de un *indicador de posición* que señala la posición del siguiente elemento que será accedido dentro del fichero.
  - **Ficheros de acceso directo**: se puede acceder a los elementos directamente por su posición

5.5.- Ficheros

- Declaración de un fichero:

tipo

archivo de tipo: tipo_fich

var

tipo_fich: id_var

Ejemplo:

tipo

archivo de entero: tipo_fich

Principal

var

tipo_fich: fich_notas_act, fich_notas_ant

Se ha definido un tipo fichero que va a corresponder a ficheros cuyos elementos son enteros, y en la sección *var* se han declarado dos variables, que serán posteriormente asociadas a ficheros mediante la función de apertura.

5.5.- Ficheros

- Apertura de un fichero:

La función *abrir()* asocia una variable de tipo fichero con un archivo almacenado en memoria secundaria. El formato de la operación de apertura es el siguiente:

abrir(var_fichero, modo, nombre_fichero)

- ***nombre_fichero***, es el nombre del fichero que se encuentra almacenado en memoria masiva
- ***var_fichero*** es la variable, definida previamente a la que se le asociará el fichero físico. A partir de este momento cualquier referencia al fichero se hará a través de esta variable.
- ***modo*** indica el modo de acceso al fichero, es decir, si la operación que se desea realizar es de lectura o de escritura.

5.5.- Ficheros

- Apertura de un fichero:

- Modo Escritura: El sistema comprueba si existe un fichero con ese nombre. Si el fichero no existe entonces se crea uno vacío con ese nombre situando el indicador de posición al principio del fichero y cada vez que se realice una operación de escritura el indicador va avanzando a la siguiente posición.

- Modo Lectura: El indicador de posición o cursor se situará al inicio del mismo y avanzará a la posición siguiente cada vez que se realice una operación de lectura. Un fichero abierto para lectura no puede ser modificado.

5.5.- Ficheros

- Manejo de ficheros:

  - Escribir:

    escribir *(var_fichero, elemento)*

  - Leer:

    leer *(var_fichero, elemento)*

  - Comprobar fin de fichero:

    feof *(var_fichero)*

  - Cerrar fichero

    cerrar *(var_fichero)*

5.5.- Ficheros

Ejemplo: Algoritmo que lee caracteres y los escribe en un fichero. Posteriormente lee del fichero para escribir en pantalla.

```
Algoritmo lee_escribe_fichero
    tipo
       archivo de caracteres: tipo_fich
    Principal
    var
       tipo_fich: fichero
    inicio
      lectura (fichero)
      escritura(fichero)
    Fin_principal
    fin_algoritmo
```

5.5.- Ficheros

```
//Cabecera: lectura(E/S tipo_fich f)
//Precondición:
//Postcondición: escribe en f los caracteres introducidos por el usuario.
Procedimiento lectura (E/S tipo_fich f)
var
   carácter : c
inicio
  abrir (f, “escritura”, “fichero_caracteres.txt”)
  escribir(“introduzca los caracteres. Caracter ‘*’ para terminar”)
  mientras (leer(c) != ‘*’) hacer
        escribir(f,c)
  fin_mientras
  cerrar(f)
fin_procedimiento
```

5.5.- Ficheros

```
//Cabecera: escritura (E tipo_fich f)
//Precondición:
//Postcondición: lee todo el contenido de f y lo escribe en pantalla.
Procedimiento escritura (E tipo_fich f)
var
   carácter : c
inicio
  abrir (f, “lectura”, “fichero_caracteres.txt”)
  mientras (no fdf(f)) hacer
        leer(f,c)
        escribir(c)
  fin_mientras
  cerrar(f)
fin_procedimiento
```

5.6.- Tipos enumerados y subrango

5.6.1.- Tipos enumerados

tipo

id_tipo_enumerado = { lista_elementos }

Ejemplo:

```
tipo
   asignatura = {Ingles, Algebra, IP}
   semaforo = {verde, amarillo, rojo}
   sexo = {Hombre, Mujer}
```

- **Cuando se realiza la definición del tipo, implícitamente se le asocia una ordenación al conjunto: el orden en que se listan los identificadores. Es decir, el primer elemento de la lista ocupa la posición 1, el siguiente la posición 2, y así sucesivamente hasta el último, que ocupa la posición N, suponiendo que el tipo conste de N elementos enumerados. Por tanto, pueden aplicarse los operadores relacionales al tipo enumerado.**
- **Las variables de un tipo enumerado no pueden ser leídas o escritas con las funciones *leer()* y *escribir()* que hemos estudiado.**
- **El uso de tipos enumerados permite mejorar la legibilidad de los algoritmos y reducir posibles errores, ya que se restringe el conjunto de valores que se puede asignar a las variables.**

5.6.- Tipos enumerados y subrango

5.6.1.- Tipos enumerados

Ejemplo: **algoritmo que simula un semáforo.**

```
Algoritmo Cruzar_Semaforo
tipo
      semaforo = { verde, ambar, rojo}
Principal
var
   semaforo: señal
 inicio
    escribir(“Cuidado al cruzar un semáforo: ”)
    desde señal ← verde hasta rojo hacer
            mostrar_semaforo(señal)
    fin_desde
    escribir(”Hasta luego.”)
  Fin_principal
Fin_algoritmo
```

5.6.- Tipos enumerados y subrango

5.6.1.- Tipos enumerados

Ejemplo: **algoritmo que simula un semáforo.**

```
//Cabecera: mostrar_semaforo(E semaforo: s).
//Precondición: s es una variable de tipo semaforo, debe estar
//inicializada.
//Postcondición: Se muestra al usuario las indicaciones de lo que
//debe hacer en cada caso, según el valor de la variable de entrada.

procedimiento mostrar_semaforo(E semaforo: s)
inicio
   según_sea (s) hacer
   rojo:  escribir("El semáforo está en rojo. Los coches deben parar.")
    ambar: escribir("Los coches deben ir frenando.")
    verde: escribir("Los coches pueden pasar.")
   fin_según
fin_procedimiento
```

5.6.- Tipos enumerados y subrango

5.6.2.- Tipos subrango

- Contiene un rango de valores de otro tipo existente, ya sea predefinido o definido por el usuario.
- Se puede definir un tipo subrango para un subconjunto del tipo entero, o bien para un subconjunto de valores de un tipo enumerado o subrango.
- No se definen nuevos valores, sino un subconjunto consecutivo de valores válidos de un tipo de dato ya definido. Al tipo ya existente, a partir del cual se define el tipo subrango, se le denomina tipo base
- El tipo subrango permite mayor legibilidad de los algoritmos o programas y evita errores, ya que se indica el rango de valores que una variable puede tomar y no se permite asignar un valor fuera del rango.
- Estas características no son soportadas por todos los lenguajes.
- Definición:

Tipo

id_tipo_subrango = lim_inf .. lim_sup

5.6.- Tipos enumerados y subrango

5.6.2.- Tipos subrango

- **Ejemplo:**

```
tipo
   semana = { Lunes, Martes, Miercoles, Jueves, Viernes, Sabado, Domingo}
   horas_semanales = 0..40                  // tipo subrango del tipo entero
   dias_laborables = Lunes .. Viernes       // subrango del tipo enumerado semana
   digitos = 0 .. 9
Principal
var
   dias_laborables: dia
   horas_semanales: hora
   digitos: dig
```