ESCOLA SUPERIOR DE ENXEÑARÍA INFORMÁTICA

Universidade Vigo

Programación I
Tema 4

Grado en Ingeniería Informática

María José Lado Touriño - Xosé Antón Vila Sobrino

Tema 4. Tipos de Datos Estructurados

1. Arrays
2. Registros
3. Cadenas
4. Punteros

Estructura de Datos

- Nuevos tipos de datos que se definen agrupando tipos de datos simples, o bien otros tipos previamente definidos.
- Tienen identificador común que representa múltiples datos individuales.
- Cada dato individual se puede referenciar independientemente.
- Caracterizados por:
  - Tipos de datos que los componen.
  - Operaciones que se pueden realizar.
- Tipos de organización de estructuras de datos (dependiendo de la forma de almacenamiento en memoria):
  - Estructuras estáticas: el tamaño ocupado en memoria se define antes de la ejecución del programa. No se puede modificar durante ejecución.
    - Arrays, registros, ficheros, archivos.
  - Estructuras dinámicas: sin limitaciones de tamaño. Se pueden construir mediante punteros (soportados en muchos lenguajes).
    - Listas, árboles, grafos.

Tipos de Estructuras de Datos

Homogéneas

- Todos los valores o datos que la forman son del mismo tipo (tipo base).
- Ejemplo: arrays, cadenas.

Heterogéneas

- Los valores o datos que la forman no tienen que ser necesariamente del mismo tipo.
- Ejemplo: registros.

Tipos de Datos Estructurados

- Nuevos tipos de datos que se definen agrupando tipos de datos simples, o otros tipos previamente definidos.
- Se caracterizan por los tipos de datos que los componen y por las operaciones que se pueden realizar.
- El tipo determina la forma de almacenamiento en la memoria.

Tipo base

- Todos los valores agrupados son del mismo tipo.

1. Arrays

- Conjunto finito y ordenado de elementos homogéneos.
- *Ordenado*: se pueden identificar los elementos primero, segundo, tercero,...
- Puede estar compuesto por elementos de tipo carácter, entero, real,...
- Su tamaño debe especificarse en tiempo de compilación:
  - Se emplea constante simbólica.
  - El tamaño no cambia durante la ejecución del programa.
- **Vector**: array en una dimensión.

Vectores

Tipo

- Tipo de dato básico de todos los elementos que forman parte del array.

Elemento

- Cada uno de los datos que forman parte del array.

Subíndice o índice de un elemento

- Posición del elemento en la ordenación del array. Toma valores de 0 a N-1 (N: tamaño del array).

Límite inferior (superior)

- Valor mínimo (máximo) permitido del array.

Rango

- Número de elementos.

Vectores

Almacenamiento

- Los distintos elementos se almacenan en posiciones sucesivas de memoria.
- La dirección de memoria de un elemento cualquiera i (si límite inferior = 0) viene dada por:
  - B + i * S

B: dirección de inicio de la memoria
S: tamaño (número de bytes) en memoria que ocupa cada dato del tipo básico que forma parte del vector

- El elemento i de un array `v` se representa como `v[i]`, teniendo en cuenta que el primer elemento es `v[0]`,y el último `v[N-1]` (N: tamaño del array).

Vectores

Declaración

```
TIPOS
   <nombre_tipo> = array[tamanõ] de <tipo_base>
   <nombre_tipo> = array[li..ls] de <tipo_base>
VARIABLES
   <nombre_vector> : <nombre_tipo>
```

Ejemplo

```
TIPOS
   vector1 = array[5] de Real
   vector2 = array[0..3] de Entero
VARIABLES
   v1 : vector1
   v2 : vector2
```

Vectores

Operaciones

- Implican procesamiento o tratamiento de los elementos individuales del vector.
- Operaciones:
  - Asignación e inicialización.
  - Lectura/Escritura.
  - Recorrido.
  - Actualización.

Vectores

Asignación e inicialización

- De un elemento: con las instrucción de asignación:

```
v1[3] <- 35
```

- De todos los elementos del vector: con estructuras repetitivas (o selectivas):

```
DESDE i <- 0 HASTA 4 HACER
   v1[i] <- i
FIN_DESDE
```

  - Si a todos los elementos del vector se les asigna el mismo valor (por ejemplo 34), está permitido:

```
v1 <- 34
```

Vectores

Lectura/Escritura

- Las instrucciones simples se representan con:

```
LEER (v1[3])
```

- Se suele realizar con estructuras repetitivas.

```
DESDE i <- 0 HASTA 4 HACER
   LEER (v1[i])
   ESCRIBIR (v1[i])
FIN_DESDE
```

- También se pueden utilizar estructuras selectivas.

```
DESDE i <- 0 HASTA 4 HACER
   SI i MOD 2 != 0 ENTONCES
      LEER (v1[i])  // Solo posiciones pares
      ESCRIBIR (v1[i])
FIN_DESDE
```

Vectores

Recorrido

- Acceso secuencial.
- Para introducir datos (LEER) o para visualizarlos (ESCRIBIR).
- Mediante estructuras repetitivas: las variables de control del bucle se emplean como subíndices del vector.
- Recorrido con DESDE:

```
ALGORITMO leer_vector
CONSTANTES
    tam : Entero = 10
TIPO
    vector = array[tam] de Entero
VARIABLES
    v : vector
    i : Entero
INICIO
    DESDE i <- 0 HASTA tam-1 HACER
        LEER(v[i])
    FIN_DESDE
FIN
```

Vectores

Recorrido

- Recorrido con MIENTRAS:

```
ALGORITMO lee_vector
CONSTANTES
    tam : Entero = 10
TIPO
    vector = array[tam] de Entero
VARIABLES
    v : vector
    i : Entero
INICIO
    i <- 0
    MIENTRAS i < tam HACER
        LEER(v[i])
        i <- i + 1
    FIN_MIENTRAS
FIN
```

Vectores

Recorrido

- Recorrido con REPETIR:

```
ALGORITMO lee_vector
CONSTANTES
    tam : Entero = 10
TIPO
    vector = array[tam] de Entero
VARIABLES
    v : vector
    i : Entero
INICIO
    i <- 0
    REPETIR
        LEER(v[i])
        i <- i + 1
    HASTA (i >= tam)
FIN
```

Vectores

Actualización

- Añadir, insertar, borrar.

Añadir

- Consiste en añadir un nuevo elemento al final del vector.
- Únicamente habrá que comprobar si hay memoria suficiente para el nuevo elemento.

```
TIPO
   vector = array[8] de Real
VARIABLES
   v : vector
…… // Suponiendo definidos solo 6 elementos:
   v[6] <- 5.47
   v[7] <- -21.89
```

Vectores

Insertar

- Consiste en insertar un nuevo elemento dentro del vector.
- Necesario realizar desplazamiento para insertar el nuevo elemento (suponiendo que hay espacio en el vector).

```
ALGORITMO insertarElemento
CONSTANTES
     tam : Entero = 10
TIPO
     vector = array[tam] de Entero
VARIABLES
     v : vector
     i : Entero
     numEl : Entero //Número de elementos definidos en v
     pos: Entero // Posición del elemento a insertar
INICIO
     SI numEl < tam ENTONCES
          LEER (pos) //Suponemos pos entero positivo menor que numEl
          DESDE i <- numEl - 1 HASTA pos HACER
               v[i+1] <- v[i]     // Desplazamiento
          FIN_DESDE
          LEER (v[pos])
          numEl <- numEl + 1
     FIN_SI
FIN
```

Vectores

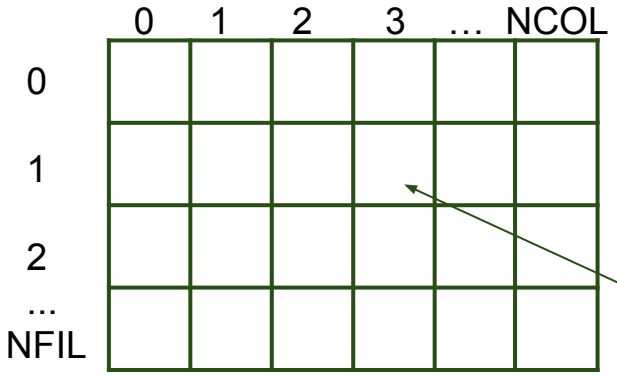
Borrar

- Consiste en eliminar un elemento dentro del vector.
- Siempre se puede hacer (si hay elementos).
- Necesario realizar desplazamiento en sentido contrario a la inserción.

```
ALGORITMO borrarElemento
CONSTANTES
     tam : Entero = 10
TIPO
     vector = array[tam] de Entero
VARIABLES
     v : vector
     i : Entero
     numEl : Entero //Número de elementos definidos en v
     pos: Entero // Posición del elemento a eliminar
INICIO
     SI numEl > 0 ENTONCES
          LEER (pos) //Suponemos pos entero positivo menor que numEl-1
          DESDE i <- pos +1  HASTA numEl - 1 HACER
               v[i-1] <- v[i]     // Desplazamiento
          FIN_DESDE
          numEl <- numEl - 1
     FIN_SI
FIN
```

Matrices

- Array bidimensional, vector de vectores.
- Todos los elementos son del mismo tipo.
- El orden de componentes es significativo.
- Necesarios dos índices para identificar cada elementos del array.
- Tiene dos dimensiones (una por cada subíndice).
- Primer subíndice identifica la fila y segundo la columna.

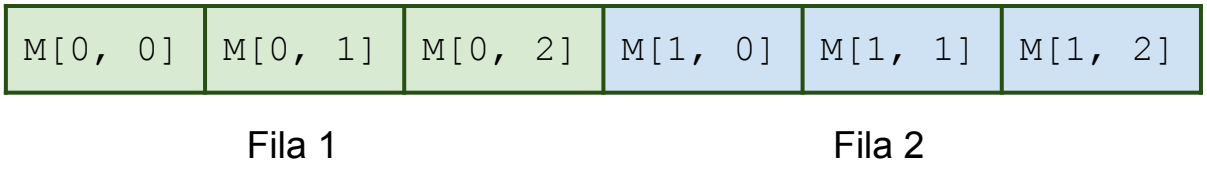


- Tamaño de la matriz (número máximo de elementos): N * M.
- Índice de filas: 0 .. N - 1.
- Índice de columnas: 0 .. M - 1.

Matrices

Almacenamiento

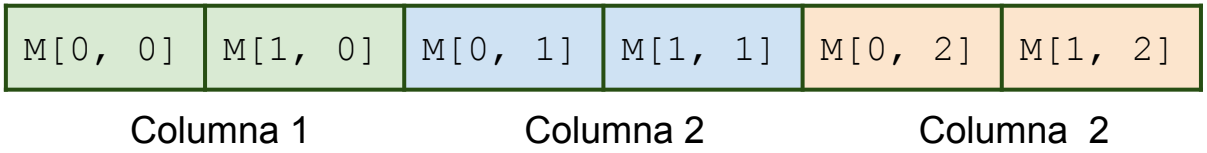
- Memoria de la computadora es lineal: array debe estar linealizado.
- Dos posibles órdenes:
  - Orden de fila mayor: M[2, 3]



    - Dirección de memoria de un elemento i (si límite inferior = 0):

      B + (N * i + j) * S

  - Orden de columna mayor: M[2, 3]



    - Dirección de memoria de un elemento i (si límite inferior = 0):

      B + (M * i + j) * S

B: Dirección de inicio de la memoria
S: Tamaño (número de bytes) en memoria que ocupa cada dato del tipo básico que forma parte del vector
N: número de filas
M: número de columnas

Matrices

Declaración

```
TIPOS
   <nombre_tipo> = array[nfilas, ncols] de <tipo_base>
   <nombre_tipo> = array[nfi..nfs, nci..ncs] de <tipo_base>

VARIABLES
   <nombre_matriz> : <nombre_tipo>
```

Ejemplo

```
TIPOS
   matriz1 = array[5, 5] de Real
   matriz2 = array[0..3, 0..4] de Entero

VARIABLES
   m1 : matriz1
   m2 : matriz2
```

Matrices

Operaciones

- Implican procesamiento o tratamiento de los elementos individuales de la matriz, procesamiento por filas, procesamiento por columnas.
- Operaciones:
  - Asignación e inicialización.
  - Lectura/Escritura.
  - Recorrido.
  - Actualización.

Matrices

Asignación e inicialización

- De un elemento: con las instrucción de asignación:

```
m1[3, 3] <- 35
```

- De todos los elementos de la matriz: con estructuras repetitivas (o selectivas):

```
DESDE i <- 0 HASTA 4 HACER
   DESDE j <- 0 HASTA 4 HACER
      m1[i, j] <- i * j
   FIN_DESDE
FIN_DESDE
```

Matrices

Lectura/Escritura

- Las instrucciones simples se representan con:

```
LEER (m1[3, 3])
```

- Se suele realizar con estructuras repetitivas.

```
DESDE i <- 0 HASTA 4 HACER
   DESDE j <- 0 HASTA 4 HACER
      LEER (m1[i, j])
      ESCRIBIR (m1[i, j])
   FIN_DESDE
FIN_DESDE
```

- También se pueden utilizar estructuras selectivas.

Matrices

Recorrido

- Mediante estructuras repetitivas: las variables de control del bucle se emplean como subíndices del vector.
- Recorrido por filas:

```
ALGORITMO recorrerFilas
CONSTANTES
    nfil : Entero = 10
    ncol : Entero = 8
TIPO
    matriz = array[nfil, ncol] de Entero
VARIABLES
    m : matriz
    i : Entero
    j : Entero
INICIO
    DESDE i <- 0 HASTA nfil HACER
        DESDE j <- 0 HASTA ncol HACER
            LEER(m[i, j])
        FIN_DESDE
    FIN_DESDE
FIN
```

Matrices

Recorrido

- Recorrido por columnas:

```
ALGORITMO recorrerColumnas
CONSTANTES
    nfil : Entero = 10
    ncol : Entero = 8
TIPO
    matriz = array[nfil, ncol] de Entero
VARIABLES
    m : matriz
    i : Entero
    j : Entero
INICIO
    DESDE j <- 0 HASTA ncol HACER
        DESDE i <- 0 HASTA nfil HACER
            LEER(m[i, j])
        FIN_DESDE
    FIN_DESDE
FIN
```

Matrices

Actualización

- Insertar una fila.

```
ALGORITMO insertarFila
CONSTANTES
     nfil : Entero = 10
     ncol : Entero = 8
TIPO
     matriz = array[nfil, ncol] de Entero
VARIABLES
     m : matriz
     i : Entero
     j : Entero
     numFil : Entero //Número de filas definidas en m
     numCol : Entero //Número de columnas definidas en m
     pos: Entero // Posición de la fila a insertar
INICIO
     SI numFil < nfil ENTONCES
          LEER (pos) //Suponemos pos un entero positivo menor que numFil
          DESDE i <- numFil - 1 HASTA pos HACER
               DESDE j <- 0 HASTA numCol - 1 HACER
                    m[i + 1, j] <- m[i, j] //Desplazamiento
               FIN_DESDE
          FIN_DESDE

          DESDE j <- 0 HASTA nCol -1 HACER
               LEER (m[pos, j])    // Se insertan elementos en la fila pos
          FIN_DESDE
          numFil <- numFil + 1
     FIN_SI
FIN
```

Matrices

Actualización

- Insertar una columna.

```
ALGORITMO insertarColumna
CONSTANTES
     nfil : Entero = 10
     ncol : Entero = 8
TIPO
     matriz = array[nfil, ncol] de Entero
VARIABLES
     m : matriz
     i : Entero
     j : Entero
     numFil : Entero //Número de filas definidas en m
     numCol : Entero //Número de columnas definidas en m
     pos: Entero // Posición de la columna a insertar
INICIO
     SI numCol < ncol ENTONCES
          LEER (pos) //Suponemos pos un entero positivo menor que numCol
          DESDE j <- numCol - 1 HASTA pos HACER
               DESDE i <- 0 HASTA numFil - 1 HACER
                    m[i, j + 1] <- m[i, j] //Desplazamiento
               FIN_DESDE
          FIN_DESDE

          DESDE i <- 0 HASTA numFil -1 HACER
               LEER (m[i, pos])    // Se insertan elementos en la fila pos
          FIN_DESDE
          numCol <- numCol + 1
     FIN_SI
FIN
```

Matrices

Actualización

- Borrar una fila.

```
ALGORITMO borrarFila
CONSTANTES
     nfil : Entero = 10
     ncol : Entero = 8
TIPO
     matriz = array[nfil, ncol] de Entero
VARIABLES
     m : matriz
     i : Entero
     j : Entero
     numFil : Entero //Número de filas definidas en m
     numCol : Entero //Número de columnas definidas en m
     pos: Entero // Posición de la fila a eliminar
INICIO
     SI numFil > 0 ENTONCES
          LEER (pos) //Suponemos pos un entero positivo menor que numFil
          DESDE i <- pos + 1 HASTA numFil - 1 HACER
               DESDE j <- 0 HASTA numCol - 1 HACER
                    m[i - 1, j] <- m[i, j] //Desplazamiento
               FIN_DESDE
          FIN_DESDE
          numFil <- numFil - 1
     FIN_SI
FIN
```

Matrices

Actualización

- Borrar una columna.

```
ALGORITMO borrarColumna
CONSTANTES
     nfil : Entero = 10
     ncol : Entero = 8
TIPO
     matriz = array[nfil, ncol] de Entero
VARIABLES
     m : matriz
     i : Entero
     j : Entero
     numFil : Entero //Número de filas definidas en m
     numCol : Entero //Número de columnas definidas en m
     pos: Entero // Posición de la columna a eliminar
INICIO
     SI numCol > 0 ENTONCES
          LEER (pos) //Suponemos pos un entero positivo menor que numCol
          DESDE j <- pos + 1 HASTA numCol - 1 HACER
               DESDE i <- 0 HASTA numFil - 1 HACER
                    m[i, j - 1] <- m[i, j] //Desplazamiento
               FIN_DESDE
          FIN_DESDE
          numCol <- numCol - 1
     FIN_SI
FIN
```

Arrays Multidimensionales

- Pueden tener N dimensiones.
- Deben poder especificarse los valores de los N subíndices para poder identificar cada elemento del array.

Declaración

```
TIPOS
<nombre_tipo> = array[dim1, dim2, dim3, …, dimN] de <tipo_base>
<nombre_tipo> = array[d1i..d1s, d2i..d2s,..., dNi..dNs] de <tipo_base>

VARIABLES
<nombre_matriz> : <nombre_tipo>
```

Ejemplo

```
TIPOS
array = array[5, 5, 4, 6] de Real
matriz2 = array[0..3, 0..4, 0..7] de Entero

VARIABLES
m1 : array1
m2 : array2
```

Paso de arrays como parámetros

- Los arrays se pueden pasar como parámetros.
- Necesario definir constantes y tipos arrays que se van a pasar.
- Pueden ser parámetros de E, S o E/S.
- Una función puede devolver un array.

2. Registros

- Se denominan también estructuras.
- Estructura de datos estática heterogénea.
- Almacena diferentes tipos de datos bajo una misma variable.
- **Campo** o **miembro**: cada uno de los elementos de datos del registro estructura.
- Cada campo es de un determinado tipo (simple o estructurado).
- El nombre de cada campo es único.
- Cada campo aparece en un orden determinado en el registro.
- Para definir el registro es necesario especificar nombre y tipo de cada campo.

Declaración

```
TIPOS
   <nombre_tipo>: REGISTRO
    <nombre_campo1>: <tipo_dato1>
    <nombre_campo2>: <tipo_dato2>
    ……
FIN_REGISTRO

VARIABLES
   <nombre_variable>: <nombre:tipo>
```

Ejemplo

```
TIPOS
    libro = REGISTRO
        titulo : Cadena
        precio : Real
        paginas : Entero
   FIN_REGISTRO

VARIABLES
   miLibro : Libro
```

Operaciones

- Acceso a los miembros.
- Asignación.
- Lectura/Escritura.

Muchas operaciones se podrán hacer en función del tipo de variable y campo a campo: sumar (numéricos, por ejemplo), comparar (numéricos o cadenas), leer/escribir...

Acceso a los miembros

- Mediante el operador '.': registro.campo

```
miLibro.titulo
```

Asignación

- Campo a campo, como cualquier variable:

```
miLibro.precio <- 56.42
```

- De un registro a otro: deben ser del mismo tipo.

```
miLibro : Libro
tuLibro : Libro
tuLibro <- miLibro
```

Lectura/Escritura

- Campo a campo, tanto para lectura como para escritura.

```
LEER(miLibro.titulo)
ESCRIBIR (miLibro.precio, miLibro.paginas)
```

Arrays de registros

- Permiten mantener la integridad de los datos y mejorar la organización.
- Se declaran de forma similar a la declaración de cualquier array de otro tipo de variable.

```
CONSTANTES
    NLibros : Entero = 25
TIPOS
    libro = REGISTRO
        titulo : Cadena
        precio : Real
        paginas : Entero
    FIN_REGISTRO

    conjuntoLibros = array [NLibros] de Entero

VARIABLES
    miConjuntoLibros : conjuntoLibros
    ……
    ESCRIBIR(miConjuntoLibros[0].titulo)
```

Registros anidados

- Algunos de sus campos son, a su vez, registros

```
CONSTANTES
    NLibros : Entero = 25
TIPOS
    fecha = REGISTRO
        dia: Entero
        mes : Entero
        año : Entero
    FIN_REGISTRO
    libro = REGISTRO
        titulo : Cadena
        precio : Real
        paginas : Entero
        fechaEdicion : fecha
    FIN_REGISTRO

VARIABLES
    miLibro : libro
    ……
    LEER (miLibro.fechaEdicion.año)
```

Registros variables (uniones)

- Permiten almacenar diferentes tipos de datos en la misma posición de la memoria.
- Contienen únicamente uno de sus miembros a la vez durante la ejecución del programa.
- Sus miembros pueden ser de cualquier tipo, y pueden contener dos o más tipos de datos.
- El número de bytes empleado para almacenar la unión debe ser suficiente para almacenar el miembro que más memoria ocupa.
- Solo se puede hacer referencia a un miembro (un tipo de dato) a la vez.

Declaración

```
TIPOS
   <nombre_tipo>: REGISTRO_VARIABLE
    <nombre_campo1>: <tipo_dato1>
    <nombre_campo2>: <tipo_dato2>
    ……
FIN_REGISTRO

VARIABLES
   <nombre_variable>: <nombre:tipo>
```

Ejemplo

```
TIPOS
    Volumen = REGISTRO_VARIABLE
        litros : Real
        dcm3 : Real
   FIN_REGISTRO

VARIABLES
   miVolumen : Volumen
```

3. Cadenas

- Secuencia finita de caracteres.
- Formada por cero o más símbolos alfanuméricos y caracteres especiales.
- Juego de caracteres (alfabeto): permite comunicación entre lenguaje de programación y computadora.
- Códigos más utilizados:
  - ASCII.
  - EBCDIC.
  - Unicode.

ASCII

- American Standard Code for Information Interchange.
- Utiliza 7 bits para representar cada carácter: 2^7 (128) caracteres diferentes.
- ASCII ampliado emplea 8 bits: 2^8 (256) caracteres.
- Compuesto de caracteres:
  - Alfabéticos (a, b, ..., z, A, B, ..., Z).
  - Numéricos (0, 1, 2,...,9).
  - Especiales (+, -, *, /, {, }, <,>, …).
  - De control: no imprimibles (funciones de escritura, Separación de archivos,...):
    - DEL: borrar.
    - STX: inicio de texto.
    - LF: avance de línea.
    - FF: avance de página.
    - CR: retorno de carro.

EBCDIC

- Extended Binary Code Decimal Interchange Code.
- Utiliza 8 bits para representar cada carácter: 2^8 (256) caracteres diferentes.
- Similar a ASCII.
- Incluye también caracteres alfanuméricos y especiales.

Unicode

- Estándar internacional.
- Aplicaciones en Internet y alfabetos internacionales.
- Emplea 2 bytes (16 bits): 2^{16} (65536) caracteres diferentes.

Secuencias de escape

- Representar caracteres que no se pueden escribir desde el teclado.
- Poseen dos partes:
- Carácter escape: símbolo que indica al compilador que se ha de traducir el carácter de modo especial.
- Valor de traducción: los Unicode son los más sencillos:
  - Número hexadecimal de 4 dígitos precedido de letra u.
  - Ejemplos: `'\u00084', '\u000D'`.
- Algunas secuencias de escape:
  - \b: retroceso.
  - \t: tabulación.
  - \n: nueva línea.
  - \r: retorno de carro.

Cadenas de caracteres

- Conjunto de caracteres (incluido el espacio en blanco) que se almacenan en áreas contiguas de la memoria.
- Pueden ser entradas/salidas desde un terminal.
- Longitud de la cadena: número de caracteres que contiene.
  - Cadena vacía o nula: no contiene ningún carácter: longitud cero.
- Representación: comillas dobles “ ” o simples ‘ ’ (dependerá del lenguaje de programación).

Constantes de tipo cadena

- Conjunto de caracteres válidos encerrados entre comillas.

Declaración

```
CONSTANTES
   <nombre_tipo> : Cadena = “texto_de_la_cadena”
```

Ejemplo

```
CONSTANTES
   cad1 : Cadena = “Esta es una cadena constante”
```

Variables de tipo cadena

- Variables cuyo valor es una cadena de caracteres.
- Tendrán una longitud determinada.

Declaración

```
CONSTANTES
    lCadena : Entero = longitud_cadena + 1 (carácter fin de cadena '\0').

VARIABLES
    <nombre_tipo> : Cadena [lCadena]
```

Ejemplo

```
CONSTANTES
    longitud : Entero = 50

VARIABLES
    cad2 : Cadena [longitud]
```

Cada elemento de la cadena se accede igual que cada elemento de un vector: cad[i], donde i entre [0, longitud-1]

Almacenamiento

Cadenas de longitud fija

- Vectores de longitud declarada, con blancos a izquierda o derecha si la cadena no alcanza dicha longitud.

c	a	d	e	n	a		d	e		l	o	n	g	i	t	u	d		f	i	j	a	

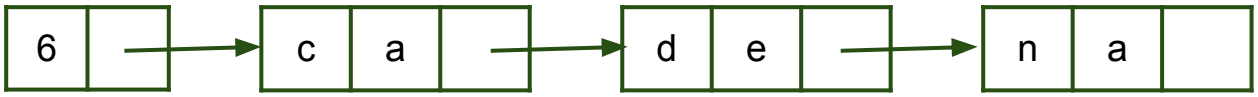
Cadenas de longitud variable con un máximo

- Cadenas de longitud variable con un máximo. Se considera puntero con dos campos (con longitud máxima y actual).
- Se utiliza una marca para indicar el fin de la cadena (‘\0’)

c	a	d	e	n	a		v	a	r	i	a	b	l	e	\0		

Cadenas de longitud indefinida

- Se representan mediante listas enlazadas (estructuras dinámicas, listas unidas mediante punteros).



Instrucciones básicas

- Asignación.
- Entrada/Salida.

Asignación

- El lado derecho de la instrucción debe contener una constante tipo cadena u otra variable del mismo tipo.
- Se emplea directamente `<-` o bien `ASIGNAR`.

Ejemplo

```
CONSTANTES
    longitud : Entero = 50

VARIABLES
    cad2 : Cadena [longitud]
    cad3 : Cadena [longitud]
    ……
    cad2 <- "Esto es una asignación de cadena"
    ASIGNAR (cad3, cad2)
```

Lectura/Escritura

- Se puede realizar en modo carácter.
- Se asigna una cadena de caracteres a una variable tipo cadena.

Ejemplo

```
CONSTANTES
    longitud : Entero = 50
VARIABLES
    cad : Cadena [longitud]

    ……
    LEER (cad)
    ESCRIBIR (cad)
```

Operaciones con cadenas

- Calcular la longitud.
- Comparar.
- Concatenar.
- Extraer subcadenas.
- Buscar.
- Insertar.
- Borrar.
- Reemplazar.
- Convertir en números.

Calcular la longitud

- Número de caracteres de la cadena.
- Función `longitud(cadena)`.
  - Argumento: cadena.
  - Resultado: valor numérico entero.

```
Entero FUNCIÓN longitud(E Cadena: cad)
VARIABLES
   i : Entero
INICIO
   i <- 0
   MIENTRAS (cad[i] <> '\0') HACER
      i <- i + 1
   FIN_MIENTRAS
   i <- i - 1
   DEVOLVER (i)
FIN
```

Comparar

- Criterios se basan en orden numérico del código.
- Función `comparar(cadena1, cadena2)`.
  - Argumento: dos cadenas.
  - Resultado: valor lógico.

Igualdad

- Dos cadenas a y b de longitudes l1, l2, son iguales si:
  - l1 = l2 (mismo número de caracteres).
  - Cada carácter de a, a[i], es igual a su correspondiente de b, b[i].
- Se comparan de forma sucesiva los caracteres correspondientes de las dos cadenas, hasta llegar al final de las cadenas sin encontrar dos caracteres diferentes.

Desigualdad

- Se emplean los operadores <, <=, >, >=, <>.
- Se comparan de forma sucesiva los caracteres correspondientes de las dos cadenas, hasta encontrar dos caracteres diferentes, momento en el cual se puede parar.

Comparar

```
Lógico FUNCIÓN comparar(E Cadena: cad1, cad2)
VARIABLES
    i : Entero
    iguales : Lógico
INICIO
    i <- 0
    SI longitud(cad1) <> longitud(cad2) ENTONCES
        iguales <- FALSO
    SI_NO
        MIENTRAS  i <> '\0' HACER
            SI cad1[i] = cad2[i] ENTONCES
                iguales <- VERDADERO
            SI_NO
                iguales <- FALSO
            FIN_SI_NO
            i <- i + 1
    FIN_SI
    DEVOLVER (iguales)
FIN
```

Concatenar

- Reunión de varias cadenas de caracteres en una sola, conservando el orden de caracteres de cada una de ellas.
- Símbolos utilizados: +, //, &, o
- Procedimiento `concatenar(cadena1, cadena2, cadRes)`.
  - Argumento: tres cadenas.

```
PROCEDIMIENTO concatenar(E Cadena : cad1, cad2 ; Cadena; S cadena : cadRes)
VARIABLES
    i : Entero
    l1 : Entero
    l2 : Entero
INICIO
    l1 <- longitud(cad1)
    l2 <- longitud(cad2)
    SI l1 + l2 <= longitud(cadRes) ENTONCES
        ASIGNAR (cadRes, cad1)
        DESDE i <- 0 HASTA l2 - 1 HACER
            cadRes[l1 + i] <- cad2[i]
        FIN_DESDE
        cadRes[l1 + l2] <- '\0' // Fin de cadena en la última posición
    FIN_SI
FIN
```

Extraer subcadenas

- Subcadena: parte específica de una cadena.
- Procedimiento `subcadena(cadIni, posIni, nCar, subCad)`.
  - Argumentos: cadIni (cadena inicial), posIni (posición inicial no nula), nCar (número de caracteres a extraer), subCad (subcadena extraída).

```
PROCEDIMIENTO subcadena(E Cadena : cadIni; E Entero : posIni, nCar; E/S Cadena : subCad)
VARIABLES
     i : Entero
     fin : Entero
     cadAux: Cadena
INICIO
     subCad <- “”
     lon <- longitud(cadIni)
     posFin <- posIni + nCar -1 // Posición final tras extraer subcadena
     SI posFin > lon - 1 ENTONCES  //Si sobrepasamos en posición la longitud de cadIni
          posFin <- lon
     FIN_SI
     DESDE i <- posIni HASTA posFin - 1 HACER
          concatenar(subCad, cadIni[i], cadAux)
          subCad <- auxCad
     FIN_DESDE
FIN
```

Buscar

- Localizar una determinada cadena dentro de otra mayor, si existe.
- Buscar la posición en que aparece un determinado carácter o secuencia de caracteres en un texto.
- Función `posicion(cadIni, cadBuscada)` (o `indice(cadIni, cadBuscada)`).
  - Argumento: dos cadenas.
  - Resultado: valor entero.
    - >= 1: posición del primer carácter de la primera coincidencia de subcadena.
    - = 0: subcadena vacía, o no aparece.

Buscar

```
Entero FUNCIÓN posicion(E Cadena : cadIni, cadBuscada)
VARIABLES
     i, j, k : Entero
     encontrado : Entero
INICIO
     i <- 0
     encontrado <- 0 //Inicialmente se considera que no hay ocurrencia

     MIENTRAS (cadIni[i] <> '\0') y (encontrado <> 1) HACER
          j <- 0
          k <- i
          MIENTRAS (cadIni[k] = cadBuscada[j]) y (cadBuscada <> '\0') HACER
               j <- j + 1
               k <- k + 1
          FIN_MIENTRAS
          SI cadBuscada[j] = '\0') ENTONCES
               encontrado <- 1
          SI_NO
               i <- i + 1
          FIN_SI
          SI encontrado = 0 ENTONCES
               i <- i - 1
          FIN_SI
     FIN_MIENTRAS
     DEVOLVER(i)
FIN
```

Insertar

- Necesario indicar la posición donde se quiere insertar.
- A partir de funciones y procedimientos ya implementados.
- Procedimiento `insertar(cadIni, cadIns, posicion)`
    - Argumentos: `cadIni` (donde se va a insertar), `cadIns` (lo que se va a insertar), `posicion` (en la que se va a insertar).

```
PROCEDIMIENTO insertar(E Cadena : cadIni; E/S Cadena : cadIns; E pEntero : posicion)
VARIABLES
    cadAux1 : Cadena //Misma longitud que la cadena a insertar
    cadAux2 : Cadena //Misma longitud que la cadena a insertar
INICIO
    subcadena (cadIni, 0, posicion, cadAux1)
    concatenar (cadIni, cadIns, cadAux)
    subcadena (cadIni, posicion+1, longitud (cadIns) - posicion, cadAux2)
    concatenar (cadAux1, cadAux2, cadIni)

FIN
```

Borrar

- Permite borrar partes de una cadena.
- A partir de funciones y procedimientos ya implementados.
- Procedimiento `borrar(cadIni, posicion, nCar)`
- Argumentos: `cad1` (donde se va a borrar), `posicion` (en la que se va a borrar), `nCar` (número de caracteres a borrar).

```
PROCEDIMIENTO borrar(E/S Cadena : cadIni; E Entero : posicion, nCar)
VARIABLES
    cadAux1 : Cadena //Misma longitud que número de caracteres a borrar
    cadAux2 : Cadena //Misma longitud que número de caracteres a borrar

INICIO
    subcadena (cadIni, 0, posicion, cadAux1)
    subcadena (cadIni, posicion + nCar, longitud(cadIni) - (posicion + nCar), cadAux2)
    concatenar (cadAux1, cadAux2, cadIni)
FIN
```

Reemplazar

- Sustituir en una cadena la primera ocurrencia de una subcadena por otra.
- A partir de funciones y procedimientos ya implementados.
- Procedimiento `reemplazar(cadIni, cadAnt, cadNue)`
  - Argumentos: `cadIni` (donde se va a reemplazar), `cadAnt` (subcadena antigua a reemplazar), `cadNue` (subcadena nueva que reemplaza).

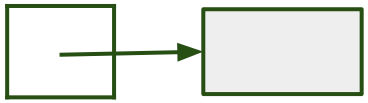
```
PROCEDIMIENTO reemplazar(E/S Cadena : cadIni; E cadena : cadAnt, cadNue)

VARIABLES
    p : Entero

INICIO
    pos <- posicion(cadIni, cadAnt);
    borrar(cadIni, pos, longitud(cadAnt));
    insertar(cadIni,cadNue,pos);
FIN
```

Convertir en números

- Convierte un número en una cadena y viceversa.
- Función `valor(cadena)`
  - Argumento: cadena (que se va a convertir).
  - Resultados: entero (resultado de la conversión).
- Función `cad(valor)`
  - Argumento: entero (que se va a convertir).
  - Resultados: cadena (resultado de la conversión).

4. Punteros

- Variable que almacena la dirección de memoria de una variable dinámica.
- Representación gráfica:



Declaración

```
TIPOS
    punt = puntero_a <tipo_dato>

VARIABLES
    p : punt
```

Ejemplo

```
TIPOS
    pentero = puntero_a Entero

VARIABLES
    p : pentero
```

Punteros

- El tipo de dato puede ser simple o estructurado.
- Operaciones:
  - Inicialización:
    - p <- nulo no apunta a ninguna variable.
  - Comparación:
    - p = q, p <> q.
  - Asignación:
    - p <- q.
  - Crear variables dinámicas:
    - reservar(p): reserva espacio en memoria para p.
  - Eliminar variables dinámicas:
    - liberar(p): libera espacio ocupado por p.

Variable dinámica: variable simple o estructura de datos sin nombre y creada en tiempo de ejecución.

Referencias

Joyanes Aguilar, L.; Rodríguez Baena, L.; Fernández Azuela, M. (2008). Fundamentos de programación. McGraw-Hill.

García-Bermejo Giner, J.R. (2008). Programación estructurada en C, Pearson Prentice Hall.

Joyanes Aguilar, L.; Zahonero Martínez, I. (2007). Programación en C : metodología, algoritmos y estructura de datos.