## 1.3 Tipos de datos elementales, operadores y comandos utilitarios

- **Tipos de datos elementales**

M es fundamentalmente un lenguaje para cálculo matricial. Todos los datos que maneja son matrices, pudiendo también trabajar con vectores y escalares, pero considerando a éstos casos particulares de matrices: un vector fila es una matriz de una fila; un vector columna, es una matriz de una columna; un escalar es una matriz de una fila y una columna.

El lenguaje M, en general, trabaja con datos numéricos, utilizando, por defecto, para su almacenamiento doble precisión (tipo de dato `double`), es decir, manejando 8 bytes de memoria para cada dato, con 15 cifras significativas en punto flotante. También puede trabajar con otros tipos de datos como enteros, reales de simple precisión, datos lógicos, cadenas de caracteres y con tipos de datos más avanzados: hipermatrices, estructuras, matrices de celdas y clases y objetos.

Combinando el tipo de dato por defecto (`double`) junto con datos lógicos y de carácter podría realizarse casi cualquier programa, por ello, son los tipos de datos que manejaremos en este curso.

- **Datos numéricos double**

Las variables reales en doble precisión (tipo `double`), son las variables por defecto que maneja el lenguaje M. Utilizan 8 bytes en memoria para almacenar el dato. No hay que hacer ninguna declaración, simplemente se usan. Los valores máximos y mínimos que se pueden almacenar son `1.7977e+308` y `2.2251e-308`. Sin embargo, la precisión de los cálculos no es superior a 15 cifras significativas.

- **Datos de tipo lógico**

Es habitual al trabajar con cualquier lenguaje de programación que aparezcan datos de contenido lógico (*true* o *false*). Estos datos se manejan en lenguaje M con los valores 1 o true (cierto) y 0 o false (falso).

Lo normal es que estos datos se generen automáticamente como resultado de ciertas operaciones.

Si un valor no lógico se tiene que evaluar como lógico se hará la transformación de la siguiente manera:

- Cualquier dato distinto de 0 equivale a cierto (1).
- Cualquier dato igual a cero equivale a falso.

- **Cadenas de caracteres**

Además de datos numéricos es necesario manejar texto. M puede definir variables que contengan cadenas de caracteres, para ello las cadenas de texto se deben delimitar entre apóstrofos o comillas simples. En el siguiente ejemplo se define la variable `s` que contiene un texto.

```
s = 'Esto es una cadena'
```

M maneja la cadena como una matriz de una fila (vector fila) distinguiendo cada carácter como si fueran distintas componentes del vector. Así, en este caso tenemos un vector de 18 componentes.

Veamos un fragmento de programa en el que se puede observar la necesidad de entrecomillar un valor de carácter para diferenciarlo del nombre de una variable.

```
a=7;

v=a;

w='a'
```

Fíjense que la variable `v` contiene el valor 7 y la variable `w` el carácter '`a`'.

- **Otros Operadores**

— **Operadores relacionales y de igualdad**

Relacionan dos valores numéricos y dan como resultado un valor lógico (cierto (1) o falso (0)). Se muestran en la siguiente tabla:

| | |
|---|---|
| Menor que | < |
| Mayor que | > |
| Menor o igual que | <= |
| Mayor o igual que | >= |
| Equivale a | == |
| Distinto de | ~= |

Ejemplos de utilización de estos operadores:

```
>> u=3.5;

>> v=7.3;

>> w=7.3;

>> u<v

ans =

     1

>> v>w

ans =

     0
```

```
>> v>=w

ans =

     1

>> u~=v

ans =

     1

>> v==w

ans =

     1
```

— **Operadores lógicos**

Se aplican a valores lógicos resultando otro valor lógico. Son los siguientes:

- Operador lógico OR: se representa con el carácter |. Se utiliza entre dos valores lógicos dando como resultado cierto, si ambos o uno de ellos son ciertos. Sólo si los dos son falsos resulta falso. La función equivalente es `or(A,B)`.

| **C=A\|B** | | |
|---|---|---|
| **A** | **B** | **C** |
| 1 | 1 | 1 |
| 1 | 0 | 1 |
| 0 | 1 | 1 |
| 0 | 0 | 0 |

  Ejemplo: si la calificación de un examen se guarda en la variable `nota`, la expresión que resulta cierta si la nota es errónea es: `nota<0|nota>10`.

- Operador lógico AND: se representa con el carácter &. Se utiliza entre dos valores lógicos dando como resultado *cierto* sólo si ambos son *ciertos*. Si uno o los dos son *falsos* el resultado es *falso*. La función equivalente es `and(A,B)`.

| **C= A&B** | | |
|---|---|---|
| **A** | **B** | **C** |
| 1 | 1 | 1 |
| 1 | 0 | 0 |

| | | |
|---|---|---|
| 0 | 1 | 0 |
| 0 | 0 | 0 |

Ejemplo: si la calificación de un examen se guarda en la variable `nota`, la expresión que resulta cierta si la nota es igual a notable es: `nota<9&nota>=7`

- Operador lógico OR EXCLUSIVO: se utiliza con la sintaxis `xor(A,B)`, resultando cierto cuando A o B son ciertos, pero no ambos.

| **C=xor(A,B)** | | |
|---|---|---|
| **A** | **B** | **C** |
| 1 | 1 | 0 |
| 1 | 0 | 1 |
| 0 | 1 | 1 |
| 0 | 0 | 0 |

Ejemplo: supongamos que un alumno realiza dos pruebas cuyas calificaciones se guardan en las variables `nota1` y `nota2`, y se ofrece una recuperación a los alumnos que hayan suspendido sólo una de las dos, la expresión que resulta cierta en ese caso sería: `xor(nota1<5,nota2<5)`

- Operador lógico NOT: se representa con el carácter `~`. Actúa sobre el valor lógico situado a su derecha, resultando el valor lógico contrario a éste. La función equivalente es `not(A)`. Suele utilizarse para hacer referencia a una condición contraria a la que se esté estudiando.

| **C=~A** | |
|---|---|
| **A** | **C** |
| 1 | 0 |
| 0 | 1 |

Ejemplo: si la calificación de un examen se guarda en la variable `nota`, la expresión que resulta cierta si la nota es suspensa es: ~(`nota>=5)`

- Operadores lógicos BREVES (`&&`) y (`||`) : son realmente el AND y el OR pero permiten simplificar la operación. Veamos las diferencias:

`A&B`: estudia siempre las condiciones A y B.

`A&&B`: estudia la condición A, si es cierta estudia la B, pero si es falsa ya no estudia la B, porque el resultado final se conoce que es falso.

`A|B`: estudia siempre las condiciones A y B.

`A||B`: estudia la condición A, si es falsa estudia la B, pero si es cierta ya no estudia la B, porque el resultado final se conoce que es cierto.

Esto además de simplificar los cálculos se puede utilizar para evitar posibles errores. Por ejemplo, en la expresión

```
c= a~=0&&b/a>5,
```

si el valor de la variable `a` es cero, ya no se desarrolla la segunda expresión y así se evita el error que supondría ésta (división entre cero).

— **Prioridad de operadores**

En la tabla siguiente se muestra el orden de prioridad todos los operadores conocidos hasta el momento. Se han ordenado de mayor a menor prioridad (de arriba hacia abajo).

| Operador | Símbolos | A igual prioridad se sigue el orden de |
|---|---|---|
| Paréntesis, or exclusivo | `( ), xor( )` | |
| No lógico | **~** | Derecha a izquierda |
| Potencia | ^ | Izquierda a derecha |
| Multiplicación, división | * , / , \ | Izquierda a derecha |
| Suma y resta, y cambio de signo (-) | + , - | Izquierda a derecha |
| Operador : | : | Izquierda a derecha |
| Relacionales | < , < = , > , > = | Izquierda a derecha |
| Igualdad | = =, **~** = | Izquierda a derecha |
| Y lógico | & | Izquierda a derecha |
| O lógico | \| | Izquierda a derecha |
| Asignación | = | Derecha a izquierda |

- **Comandos utilitarios en el manejo del lenguaje M**

— **Almacenamiento del texto de la ventana workspace**

Si se desea almacenar en un fichero de texto las entradas que se van a teclear en la ventana de comandos y las salidas ofrecidas por el programa, lo más sencillo es utilizar el comando `diary`.

Tecleando

```
diary nombrefichero
```

se crea el fichero con el nombre indicado en la ubicación actual almacenándose en él el contenido que se inserte en la ventana *workspace* a partir de ese momento. Si no se indicara nombre de fichero, se crearía el archivo de nombre *diary* para tal fin. Si en un momento de la sesión se quiere desactivar la grabación en el fichero, se escribirá

```
diary off
```

cuando quiera reanudarse la grabación, se tecleará

```
diary on
```

También se puede pasar del estado *on* al *off*, o viceversa, escribiendo simplemente `diary`.

— **Almacenamiento de variables en ficheros binarios**

Cuando se finalice una sesión de trabajo, para evitar perder los datos obtenidos (variables) se pueden guardar éstos (con sus valores) dentro de un fichero binario, que puede ser cargado en otro momento para continuar con el trabajo anterior. Esta operación se puede realizar utilizando el comando `save`. A continuación se ejemplifican diferentes opciones de utilización.

Si se quieren guardar todas las variables del espacio de trabajo en el fichero de nombre *guardavariables*, se utilizaría:

```
save guardavariables
```

Si sólo interesa guardar algunas variables, por ejemplo, las variables de identificadores `w` y `z`, se procede así:

```
save guardavariables w z
```

Para cargar de nuevo las variables guardadas en un fichero binario se utiliza el comando `load` de la forma siguiente:

```
load guardavariables
```

— **Otros comandos útiles**

– Comentarios en el código

Los comentarios en el código fuente de un programa se añaden para hacer el código más entendible al programador de cara a futuras utilizaciones o a compartir el código con terceros. Se trata de texto que se incrusta en el código pero que es ignorado por el compilador o intérprete.

En lenguaje M, para insertar un comentario debemos preceder éste por el símbolo `%`. En el siguiente ejemplo, la primera línea es un comentario, también aparece un comentario después de la instrucción de la tercera línea.

```
% radio y altura del cilindro

r=7.3;h=25.4;

a=pi*r^2*h; %volumen del cilindro
```

- Continuación de una sentencia en otra línea

En ocasiones la escritura de una sentencia queda demasiado larga y conviene continuarla en la línea posterior. Para ello se debe terminar la línea que continuará en la siguiente con tres puntos. Veamos un ejemplo:

```
>> b=7;c=6;

>> x=b+c^3-c*b...

+4/b

x =

  181.5714
```

Sin embargo no se permite realizar este proceso en una cadena de caracteres. El siguiente ejemplo produciría un error:

```
>>x='No es posible continuar un texto...

en la siguiente línea'
```

- Eliminación de variables del espacio de trabajo

Si se desea eliminar todas las variables del espacio de trabajo, se utiliza el comando `clear`. Para eliminar únicamente algunas variable se emplea el mismo comando seguido de los nombres de las variables a eliminar. Por ejemplo la siguiente instrucción borra las variables `a`, `b` y `c`.

```
clear a b c
```

- Limpieza de texto de la ventana de comandos

Para eliminar todo el texto escrito en la ventana de comandos se utiliza el comando **`clc`**. La ejecución de esta orden no afecta a las variables de la sesión de trabajo (la ventana *workspace* sigue manteniendo las variables).

- Directorio de trabajo

Para conocer la ruta al directorio de trabajo actual se utiliza el comando `pwd`.

Para obtener un listado de los ficheros que existen en la carpeta actual, se utiliza el comando `dir`.

Para cambiar de carpeta de trabajo se utiliza el comando:

```
cd  nombrenuevacarpeta.
```

- Utilización de la ayuda

Para obtener ayuda sobre la utilización de un comando o función determinada se utiliza la sentencia:

```
help  nombredelcomando
```

Para buscar un texto que esté contenido en la primera línea de comentario de los programas se utiliza el comando `lookfor` de la forma:

```
lookfor  'texto a buscar'
```

- Tiempo que tarda un cálculo en ser efectuado

Cuando se implementa un algoritmo en un lenguaje de programación es habitual que se pueda obtener la solución esperada mediante diferentes códigos fuente. Una forma de comparar los códigos y elegir cuál resulta más conveniente es calcular el tiempo de ejecución. El lenguaje M cuenta con las funciones `tic` y `toc` que trabajan juntas y calculan el tiempo transcurrido entre la ejecución de la primera y la segunda.

`tic`**:** activa un contador temporal en segundos que finaliza al utilizar el comando `toc`.

`toc`**:** devuelve el tiempo transcurrido en segundos desde que se activó el contador con `tic`.

Veamos unos ejemplos que muestran dos formas de utilización. En ambas se almacena el tiempo de ejecución en la variable `tiempo`:

```
tic

suma=0;

for i=1:1000

    suma=suma+i;

end

tiempo=toc
```

```
inicio=tic

suma=0;

for i=1:1000

    suma=suma+i;

end

tiempo=toc(inicio)
```