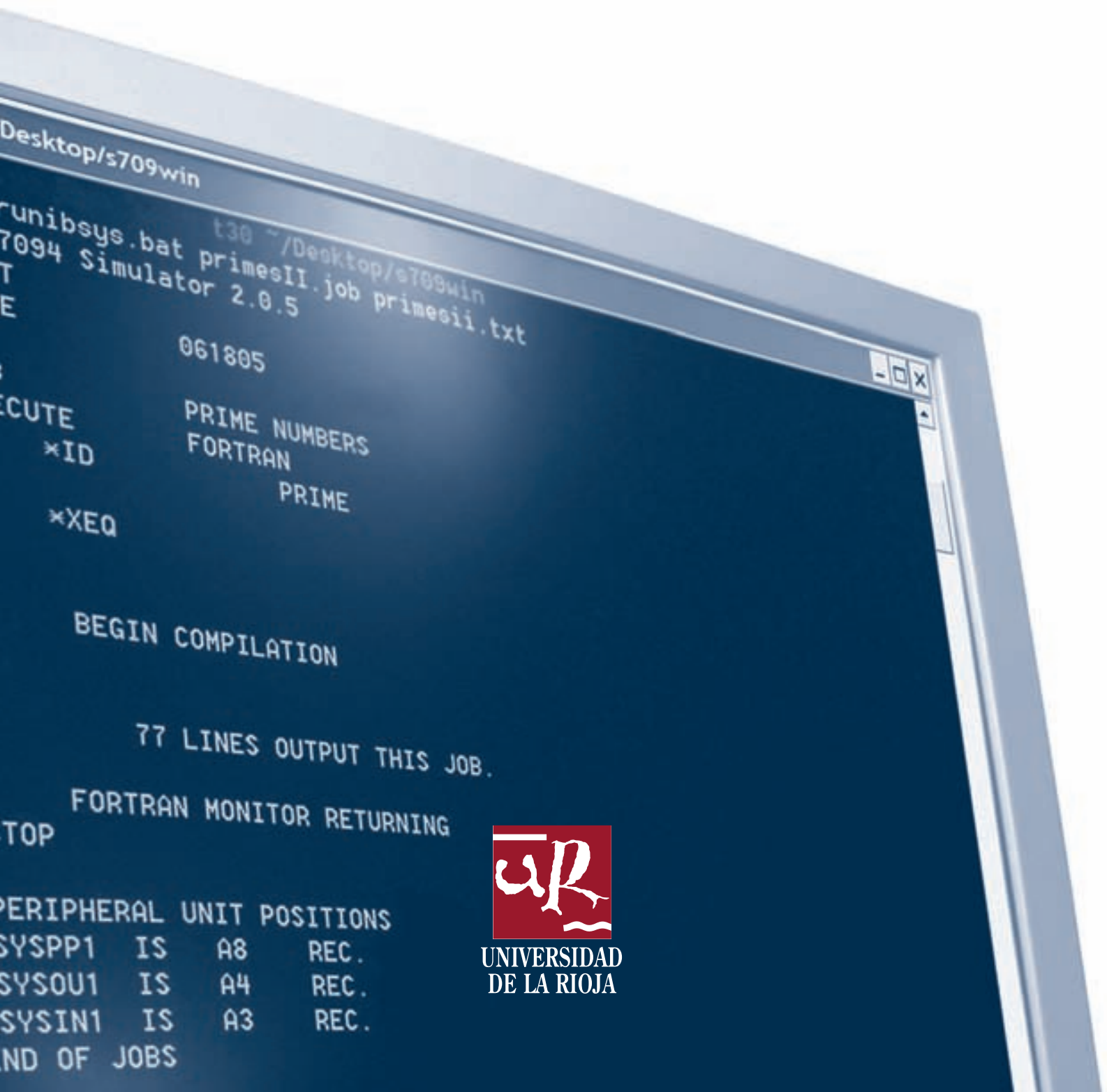


Introducción a la programación en Fortran 90 para Químicos. Prácticas

Pedro Alberto Enriquez Palma
y M^a Pilar Puyuelo García



UNIVERSIDAD
DE LA RIOJA

Introducción a la programación en Fortran 90 para Químicos

Prácticas

MATERIAL DIDÁCTICO
Química
n° 1

Pedro Alberto Enríquez Palma

María Pilar Puyuelo García

Introducción a la programación en Fortran 90 para Químicos

Prácticas

Enrriquez Palma, Pedro Alberto

Introducción a la programación en Fortran 90 para químicos:
Prácticas / Pedro Alberto Enrriquez Palma, María Pilar Puyuelo García.
– [Logroño] : Universidad de La Rioja, Servicio de Publicaciones, 2007
XII, 276 p. : il. ; 30 cm. – (Material didáctico. Química ; 1)
ISBN 978-84-690-3496-5

1. Fortran (Lenguaje de programación). I. Puyuelo García, María Pilar.
II. Universidad de La Rioja. Servicio de Publicaciones
681.3.06

Reservados todos los derechos. No está permitida la reproducción total o parcial de este libro, bajo ninguna forma ni por ningún medio, electrónico o mecánico, ni por fotocopia o grabación, ni por ningún otro sistema de almacenamiento, sin el permiso previo y por escrito de los titulares del Copyright.

© Pedro Alberto Enrriquez Palma

María Pilar Puyuelo García

Universidad de La Rioja. Servicio de Publicaciones

Edita: Universidad de La Rioja. Servicio de Publicaciones

Diseño de portada: Universidad de La Rioja. Servicio de Comunicación

ISBN 978-84-690-3496-5

Impreso en España - Printed in Spain

Prefacio

Este libro está dirigido a los alumnos y profesores de la asignatura *Estadística y Programación Aplicada a la Química* que se imparte en el primer cuatrimestre del segundo curso de la Licenciatura en Química de la Universidad de La Rioja, y cuya docencia uno de los autores (PAE) inició en el curso 94/95, habiéndose mantenido desde entonces el programa docente y las características generales del diseño instruccional. A lo largo de los años, el material didáctico utilizado en la parte de la asignatura dedicada a la programación, desarrollada utilizando el lenguaje Fortran, se ha ido mejorando con las aportaciones de otros docentes y de los propios alumnos, adaptándose también a las rápidas innovaciones en los recursos informáticos disponibles en la universidad. En el año 2007, la primera versión del lenguaje Fortran cumplirá 50 años y hemos querido aprovechar la ocasión para hacer una nueva revisión del material docente y publicarlo en formato electrónico.

En la citada asignatura, la introducción a la programación está diseñada en forma de sesiones prácticas de aula informática semanales, de dos horas de duración cada una, completándose un total de 3 créditos de docencia presencial. En estas sesiones, el aprendizaje por parte de los alumnos se orienta utilizando el material docente incluido en este libro. Está constituido por diez unidades didácticas, llamadas Prácticas, que corresponden aproximadamente con las sesiones prácticas del curso. Estas Prácticas están escritas como un texto-guía para el estudiante, asumiendo que la docencia es presencial y el instructor supervisa la actividad. Aunque al inicio de cada sesión el profesor hace una breve introducción a los conceptos de la Práctica correspondiente y explica algún algoritmo particularmente difícil, a continuación cada alumno continúa con su aprendizaje siguiendo las pautas del material docente y a su propio ritmo, permitiéndose un solapamiento de Prácticas en la clase. Por otra parte, el material didáctico está diseñado de acuerdo con la evaluación de la actividad, que se realiza mediante un examen escrito en el que se han de escribir fragmentos de programas o programas que ejecuten determinadas tareas. Este libro no está pensado como un texto para que un alumno sin experiencia en lenguajes de programación aprenda de forma autónoma a programar en Fortran90. Si este es el caso, le recomendamos que consulte algunos de los manuales reseñados en el apéndice C.

En esta versión de las Prácticas del curso, además de revisarse la redacción de las Prácticas, se han incluido más ejemplos y ejercicios, se han incluido preguntas de autoevaluación y tres apéndices con soluciones a ejercicios seleccionados, propuestas de microproyectos y material de consulta. Las prácticas 1 a 8 introducen los elementos básicos del Fortran 90. En la práctica 1 se explica la estructura general de un programa en Fortran 90, cómo editarlo, cómo compilarlo y ejecutarlo, y cómo escribir programas que realicen cálculos aritméticos sencillos. La práctica termina con una breve explicación del uso de texto como datos de un programa en Fortran. En la práctica 2 se introducen las funciones intrínsecas y se da una explicación más detallada de las instrucciones de lectura y escritura. La práctica 3 se inicia con las variables y constantes lógicas, los operadores lógicos y las reglas de construcción de expresiones

Prefacio

de comparación y de lógica booleana. Una vez introducidas las expresiones lógicas, se estudian las instrucciones de ejecución condicionada y transferencia de control (IF simple, bloques IF) que permiten controlar el flujo del programa. En las prácticas 4 y 5 se introducen los bucles DO WHILE y DO iterativos que permiten la repetición de manera controlada de partes del código. El estudio del uso básico de estos bucles se completa con el estudio de las instrucciones CYCLE y EXIT que permiten interrumpir la ejecución de una iteración o la ejecución de estos bucles de manera selectiva. Las prácticas 6 y 8 están dedicadas al estudio de las variables subindicadas o arreglos que se utilizarán para el almacenamiento de listas y tablas de datos, matrices y vectores, y al uso de algunas funciones intrínsecas de Fortran 90 diseñadas para este tipo de variables. La práctica 7 se dedica a plantear dos ejercicios largos en los que se aplican los conocimientos adquiridos a la escritura de un programa para calcular estadísticos muestrales y un programa de ajuste de mínimos cuadrados a una recta.

Las prácticas 9 y 10 corresponden a una introducción a la programación modular. En éstas se introducen los módulos FUNCTION, que permiten definir funciones de usuario externas, y los módulos SUBROUTINE, que permiten definir subrutinas. En esta parte del curso, el alumno aprende a crear sus propios módulos y la formación adquirida puede servirle para utilizar otro material disponible en Internet para sus programas.

El libro termina con tres apéndices. El apéndice A recoge las soluciones de algunos de los ejercicios propuestos en las prácticas, seleccionados para servir como referencia. El apéndice B incluye tres propuestas de Microproyectos, cuya realización por parte de los alumnos bajo la supervisión del instructor puede complementar el aprendizaje de esta asignatura, o formar parte de las actividades de otras asignaturas. En este sentido, la experiencia docente positiva de los autores de este libro tutelando microproyectos en otras asignaturas, ha motivado que se encuentre en fase de preparación una ampliación de propuestas. Por último, en el apéndice C se recoge una selección de libros y recursos de internet que pueden ser útiles para los estudiantes y los profesores del curso.

En las Prácticas se asume que el estudiante trabaja en Windows, utiliza como editor de textos el editor del MS-DOS y el compilador ELF90 (Essential Lahey Fortran 90). La versión de Windows utilizada, o la formación inicial de los alumnos, pueden hacer que sea más adecuado utilizar otro programa de edición de textos, o un entorno integrado de edición sensible al lenguaje Fortran (por ejemplo Lahey EDitor for Windows), sin que por ello se vea afectado el desarrollo de las prácticas. El compilador ELF90 implementa un subconjunto del lenguaje Fortran 90 que está diseñado para la docencia de este lenguaje (y sus aplicaciones), aunque eliminando algunas instrucciones marcadas como obsoletas en el estándar ISO que describe el lenguaje Fortran 90 y eliminando algunas opciones de sintaxis que son legales pero que son fuentes de errores de programación y que no contribuyen al aprendizaje de un estilo de programación estructurado. Sin embargo, los programas escritos en este subconjunto del lenguaje son compatibles con otros compiladores de Fortran, y, de hecho, buena parte de ellos se han compilado con los compiladores Compaq Visual Fortran 6 (Fortran 90), **G95** (Fortran 95) y **FTN95** personal edition (Fortran 95) en Windows.

En el texto se han utilizado las siguientes convenciones tipográficas:

Texto en la pantalla del ordenador: C:\EPAQ\VALVANERA> cinetical

Instrucciones de un programa: **PROGRAM** Prefacio

Líneas de comentario de un programa: ! Comentario en un programa

Enlace internos: [1.3](#)

Enlace externos: [Fifty Years of Fortran](#)



La señal de curva peligrosa se ha utilizado para indicar temas en los que los estudiantes suelen encontrar dificultades o que suelen dar lugar a malos entendidos.



Cuadros de texto en los que se advierte de *errores de sintaxis* (*¡tóxicos!*).



Cuadros de texto en los que se advierte de *errores de programación* (*¡inflamables!*).



Cuadros de texto en los que se advierte de *errores de ejecución* (*¡irritantes!*).

Queremos expresar nuestro agradecimiento al Dr. Miguel González Pérez del Departamento de Química Física de la Universidad de Barcelona por su ayuda y su apoyo en el desarrollo de este proyecto, a los doctores Judith Millán Moneo y Rodrigo Martínez Ruiz del área de Química Física de la Universidad de La Rioja, con los que hemos compartido la docencia de la asignatura *Estadística y Programación Aplicada a la Química*, y a los alumnos de la asignatura que han compartido sus dudas y comentarios con los autores.

Este libro ha sido escrito en LaTeX. Queremos extender nuestro agradecimiento al grupo de usuarios de TeX hispanohablantes (CervanTeX) por su ayuda para resolver alguno de los problemas que han surgido durante la edición de este libro.

Los autores,
Logroño, julio de 2006

Índice general

Índice general	v
I Programación básica	1
1. Elementos básicos del Fortran 90 (I)	3
1.1. MS-DOS: la instrucción PATH	4
1.2. Estructura y edición de un programa en Fortran 90	7
1.3. Compilación, ensamblaje y ejecución de un programa Fortran 90	11
1.4. Constantes y variables	12
1.5. Instrucciones de asignación y operaciones aritméticas	17
1.5.1. Operadores aritméticos	18
1.5.2. Operaciones con números enteros	19
1.5.3. Operaciones con números reales	19
1.5.4. Evaluación de expresiones aritméticas	20
1.5.5. Evaluación de expresiones con mezcla de tipos de constantes o variables	24
1.6. Constantes con nombre	28
1.7. Sentencias de asignación de caracteres	30
2. Elementos básicos del Fortran 90 (II)	33
2.1. Las funciones intrínsecas	34
2.2. Lectura y escritura de datos	35
2.2.1. Lectura de datos del teclado con formato libre	35
2.2.2. Escritura de datos en la pantalla con formato libre	37

3. Instrucciones de ejecución condicionada y transferencia de control	43
3.1. Constantes y variables lógicas	44
3.2. Sentencias de asignación lógicas	45
3.2.1. Expresiones de comparación	45
3.2.2. Operadores de lógica combinatoria	47
3.2.3. Prioridades en la evaluación de expresiones de comparación y lógicas	48
3.3. La instrucción IF (IF lógico)	49
3.4. Bloques IF	50
4. Instrucciones de control de flujo (I)	61
4.1. Ejercicios de repaso	62
4.2. Bucles DO WHILE - END DO	62
4.2.1. Estructuras anidadas	66
4.3. Bucles DO iterativos	67
4.3.1. Cálculos de tablas de valores	76
4.4. Lectura y escritura de datos con ficheros (sin formato)	80
5. Instrucciones de control de flujo (II)	87
5.1. Ejercicios de repaso	88
5.2. Bucles DO anidados	89
5.3. Transferencia de control en bucles DO: las instrucciones CYCLE y EXIT	93
5.3.1. La instrucción CYCLE	93
5.3.2. La instrucción EXIT	95
5.3.3. Bucles anidados y las sentencias CYCLE y EXIT	97
5.4. Bucles DO y bloques IF con nombre	103
6. Variables subindicadas (I)	107
6.1. Variables subindicadas: arreglos unidimensionales, vectores o listas	108
6.2. Cómo se declara una variable subindicada	109
6.3. Lectura y escritura de vectores	111
6.4. Cómo utilizar vectores en instrucciones en Fortran	118
6.5. Funciones intrínsecas de vectores	127
7. Aplicaciones	135
7.1. Cálculo de la media muestral, la varianza muestral y la varianza de la media muestral	136
7.2. Ajuste de mínimos cuadrados a una recta	137
7.2.1. Regresión lineal simple	137
7.2.2. Ajuste a exponenciales mediante regresión lineal simple	140

8. Variables subindicadas (II)	143
8.1. Variables subindicadas bidimensionales: matrices o tablas	144
8.2. Cómo se declara una variable subindicada	146
8.3. Lectura y escritura de matrices	147
8.4. Cómo utilizar matrices en instrucciones en Fortran	159
8.5. Aplicaciones	163
8.6. Funciones intrínsecas de matrices	166
II Funciones y subrutinas	177
9. Subprogramas: funciones	179
9.1. Subprogramas	180
9.2. Ejecución de un subprograma FUNCTION	181
9.3. Estructura de un subprograma FUNCTION	182
9.4. El bloque INTERFACE	187
9.5. Vectores y matrices como argumentos en subprogramas FUNCTION	194
10. Subprogramas: subrutinas	201
10.1. Estructura de un subprograma SUBROUTINE	202
10.2. Ejecución de un subprograma SUBROUTINE	205
10.3. El bloque INTERFACE	206
10.4. Vectores y matrices como argumentos en subprogramas SUBROUTINE	213
III Apéndices	219
A. Soluciones a ejercicios seleccionados	221
B. Microproyectos	247
B.1. Cinética Química: estudio de la cinética de reacciones químicas utilizando determinaciones gráficas	247
B.1.1. Ajuste de mínimos cuadrados a una recta con pesos	249
B.2. Espectrofotometría: determinación de la composición molar de una mezcla a partir de medidas espectrofotométricas	257
B.2.1. Solución de sistemas de ecuaciones lineales: el método de Gauss-Seidel	258
B.3. Descripción OM-CLOA de la molécula H ₂ ⁺	264
B.3.1. Integración numérica: el método del trapecio	265

Índice general

C. Libros y recursos en Internet	271
C.1. Libros	271
C.2. Recursos en Internet	273

Índice de ejemplos

1.1. Ejecución de una instrucción en MS-DOS	4
1.2. Contenido de la variable PATH del sistema	6
1.3. Estructura de un programa en Fortran 90	8
1.4. Tarea 1. Compilación, ensamblaje y ejecución de un programa escrito en Fortran 90	11
1.5. Evaluación de expresiones aritméticas	21
1.6. Evaluación de expresiones aritméticas (II)	21
1.7. Evaluación de expresiones aritméticas (III)	22
1.8. Tarea 2. Ejemplos de operaciones con números enteros	22
1.9. Tarea 3. Cálculo de la molaridad de una disolución	23
1.10. Tarea 4. Ejemplos de operaciones con expresiones mixtas	26
1.11. Tarea 5. Ejemplos de operaciones con expresiones mixtas (II)	26
1.12. Tarea 6. Cálculo de la presión de un gas ideal	29
1.13. Tarea 7. Uso de variables de tipo CHARACTER.	31
2.1. Tarea 8. Cálculo del pH de una disolución acuosa a partir de la concentración de H ⁺	39
2.2. Tarea 9. Cálculo de la molaridad de una disolución.	39
3.1. Tarea 10. Uso de los operadores de comparación.	46
3.2. Tarea 11. Ordenar tres números enteros de menor a mayor	54
3.3. Tarea 12. Corrosión electroquímica de un metal en ausencia de oxígeno en medio ácido	56
3.4. Tarea 13. Evaluación de una función $f(x, y)$ a partir de un par de valores x e y especificados por el usuario	57
4.1. Tarea 14. Ejemplo del funcionamiento de los bucles DO WHILE	64
4.2. Tarea 15. Ejemplo del funcionamiento de los bucles DO iterativos (I)	70
4.3. Tarea 16. Ejemplo del funcionamiento de los bucles DO iterativos (I)	71
4.4. Tarea 17. Ejemplo del funcionamiento de los bucles DO iterativos (III)	72
4.5. Tarea 18. Ejemplo del funcionamiento de los bucles DO iterativos (IV)	72
4.6. Tarea 19. Cálculo del factorial de un número	74

Índice de ejemplos

4.7. Tarea 20. Cálculo del sumatorio de una serie de datos	75
4.8. Tarea 21. Cálculo de una tabla de temperaturas conocidas la temperatura inicial, la temperatura final y el número total de valores de la tabla	77
4.9. Tarea 22. Cálculo de una tabla de temperaturas conocidas la temperatura inicial, la temperatura final y la diferencia entre dos valores consecutivos en la tabla	78
4.10. Tarea 23. Lectura de datos de un fichero y escritura de datos en un fichero.	83
5.1. Errores de sintaxis con bucles DO anidados (I)	90
5.2. Errores de sintaxis con bucles DO anidados (II)	91
5.3. Tarea 24. Bucles DO anidados	91
5.4. Tarea 26. Funcionamiento de la instrucción CYCLE con bucles DO iterativos	94
5.5. Tarea 27. Funcionamiento de la instrucción CYCLE con bucles DO WHILE	94
5.6. Tarea 29. Funcionamiento de la instrucción EXIT	96
5.7. Suma de una serie de datos	97
5.8. Tarea 30. Calculo del numero e	99
5.9. Tarea 31. Cálculo de la función de partición rotacional de una molécula diatómica	100
5.10. Bucles DO con nombre	105
6.1. Tarea 32. Lectura y escritura de elementos de un vector utilizando un bucle DO implícito	114
6.2. Tarea 33. Lectura y escritura de elementos de un vector utilizando el nombre del arreglo	116
6.3. Tarea 34. Lectura y escritura de elementos de un vector utilizando una tripleta	117
6.4. Errores de sintaxis: operaciones con vectores no compatibles	120
6.5. Tarea 35. Sentencias de asignación con vectores	121
6.6. Tarea 36. Operaciones con vectores	122
6.7. Tarea 37. Presión de vapor de un líquido puro	123
6.8. Tarea 38. La ley de Arrhenius	125
6.9. Tarea 39. Uso de la función intrínseca SUM	128
6.10. Tarea 40. Uso de la función intrínseca PRODUCT	131
8.1. Rango, forma y tamaño de un arreglo	146
8.2. Tarea 41. Escritura de una matriz con bucles DO iterativos	151
8.3. Tarea 42. Escritura de un subconjunto de elementos de una matriz con bucles DO iterativos	152
8.4. Tarea 43. Escritura de elementos de una matriz con bucles DO implícitos	153
8.5. Tarea 44. Lectura y escritura de matrices combinando bucles DO y bucles DO implícitos	155
8.6. Tarea 45. Lectura y escritura de matrices combinando bucles DO y bucles DO implícitos utilizando ficheros	156
8.7. Errores de sintaxis: operaciones con matrices no compatibles	162
8.8. Matriz identidad	162
8.9. Tarea 46. Uso de la función TRANSPOSE	166
8.10. Tarea 47. Uso de la función MATMUL	168

8.11. Tarea 48. Uso de la función SUM con matrices	173
9.1. Tarea 49. cálculo del factorial de un número utilizando una unidad FUNCTION	188
9.2. Tarea 50. Teoría del estado de transición	191
9.3. Tarea 51. Encontrar el mayor valor de una lista de diez valores utilizando vectores de tamaño fijo en una función	194
9.4. Tarea 52. Encontrar el mayor valor de una lista de diez valores utilizando vectores de tipo forma predefinida en una función	196
9.5. Tarea 53. Encontrar el mayor valor de una lista de diez valores utilizando vectores de tipo automático en una función	198
10.1. Tarea 54. Cálculo del área de un círculo utilizando una subrutina	208
10.2. Tarea 55. Uso de subrutinas para leer y escribir datos	209
10.3. Tarea 56. Cálculo de la constante cinética de una reacción química, k, utilizando la ecuación de Arrhenius	210
10.4. Tarea 57. Lectura y escritura de una matriz con subrutinas	213

Parte I

Programación básica

Práctica **1**

Elementos básicos del Fortran 90 (I)

CONTENIDOS

1.1. MS-DOS: la instrucción PATH	4
1.2. Estructura y edición de un programa en Fortran 90	7
1.3. Compilación, ensamblaje y ejecución de un programa Fortran 90	11
1.4. Constantes y variables	12
1.5. Instrucciones de asignación y operaciones aritméticas	17
1.5.1. Operadores aritméticos	18
1.5.2. Operaciones con números enteros	19
1.5.3. Operaciones con números reales	19
1.5.4. Evaluación de expresiones aritméticas	20
1.5.5. Evaluación de expresiones con mezcla de tipos de constantes o variables	24
1.6. Constantes con nombre	28
1.7. Sentencias de asignación de caracteres	30

Estas prácticas de programación están diseñadas asumiendo que los ejemplos y ejercicios que se proponen se realizarán abriendo sesiones o consolas de MS-DOS, en los sistemas operativos Windows 95/98/XP/2000. En esta consola, utilizando instrucciones básicas de MS-DOS, se pueden copiar, mover, editar o borrar ficheros y carpetas (directorios), o bien, se pueden compilar los programas escritos en Fortran para generar los correspondientes programas ejecutables, y ejecutarse éstos para comprobar su correcto funcionamiento.

Aunque antes de empezar estas prácticas es necesario conocer las instrucciones básicas de MS-DOS, en la primera sección de la Práctica 1 se revisa particularmente la instrucción PATH con el fin de repasar el mecanismo que utiliza el sistema operativo para identificar y ubicar una instrucción interna o un programa ejecutable. En las secciones posteriores, se estudiará la estructura general de un programa escrito en Fortran90, y se introducirán los procedimientos de edición, compilación y ejecución de un programa. Para finalizar la práctica, se aprenderá a escribir programas en los que se evalúan expresiones aritméticas simples y en los que se utiliza texto.

Con los programas de edición y compilación adecuados se pueden realizar estas prácticas en un entorno gráfico de Windows, o bien, en otros sistemas operativos (Linux, Mac-OS).

1.1. MS-DOS: la instrucción PATH

Cuando se introduce una secuencia de caracteres en la línea de comandos, por ejemplo, hola, el sistema operativo MS-DOS realiza el siguiente proceso:

- i Comprueba si la secuencia de caracteres corresponde a una orden interna del MS-DOS. Para ello, compara la cadena de caracteres HOLA con los nombres de las instrucciones internas.
- ii Si HOLA no coincide con una orden interna, entonces comprueba si la secuencia de caracteres corresponde a un fichero ejecutable (es decir, con las extensiones COM, EXE o BAT).

Para ello, realiza una búsqueda en el directorio activo que es la carpeta en la que se está trabajando. El objeto de la búsqueda son los archivos hola.com, hola.exe y hola.bat, siguiendo este orden.


En el caso de que existan dos ficheros ejecutables en dicha carpeta, por ejemplo hola.com y hola.exe, nunca se ejecutará el fichero hola.exe puesto que el proceso de búsqueda se habrá interrumpido antes al encontrar el fichero hola.com.

- iii Si encuentra el programa ejecutable correspondiente a HOLA, entonces lo ejecuta. Si, por el contrario, no ha encontrado el fichero ejecutable aparece en la pantalla el siguiente mensaje:

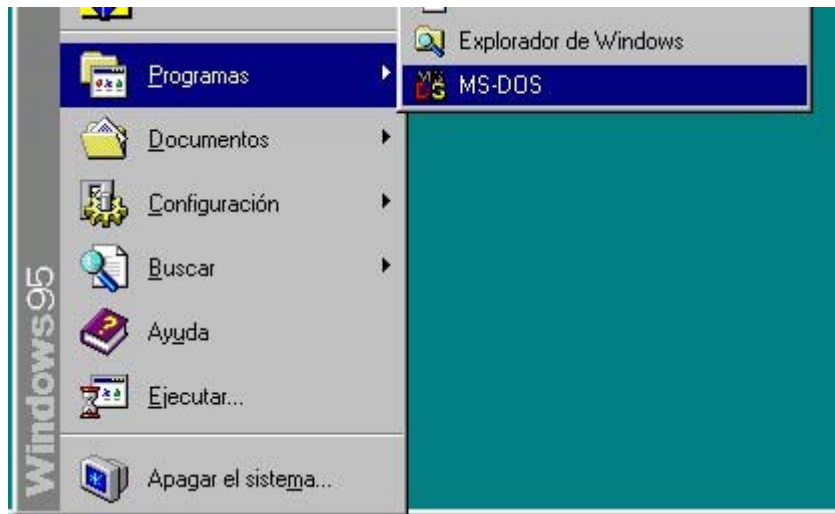
```
Comando o nombre de archivo incorrecto
```

Ejemplo 1.1 Ejecución de una instrucción en MS-DOS.

Realice la secuencia de instrucciones que se describe a continuación:


- a Pulse con el botón izquierdo del ratón el botón  en la barra tareas.

b Desplace el cursor hasta Programas y dentro de Programas seleccione MS-DOS:



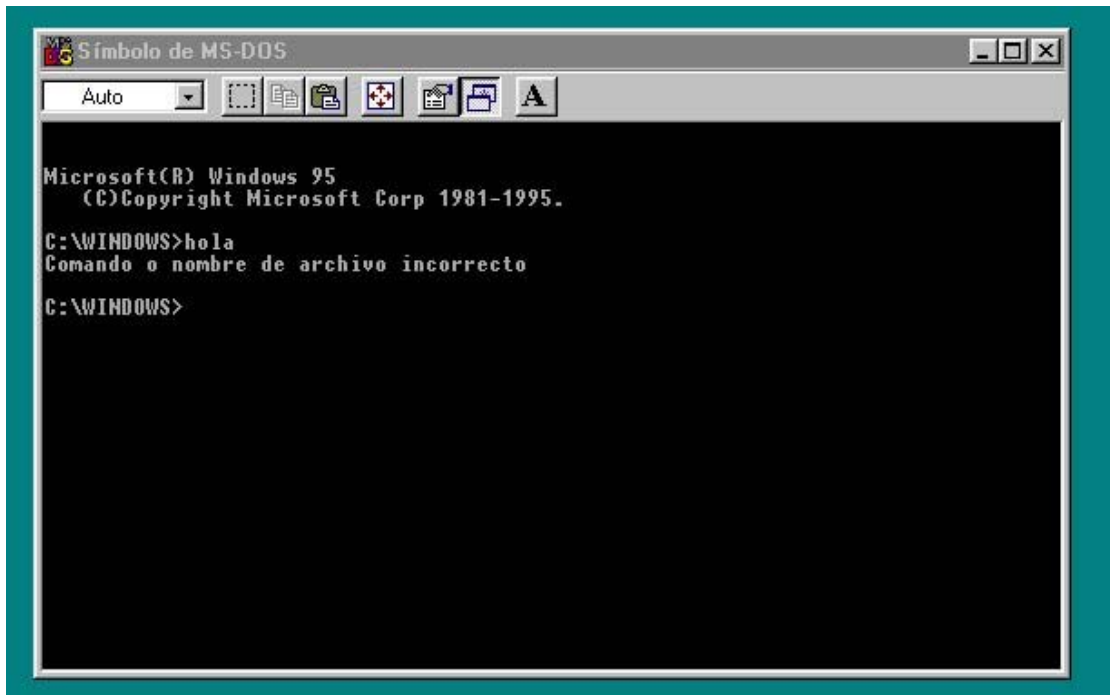
Una vez realizado esto, se abrirá una pantalla en la que se podrá trabajar con la interfaz de instrucciones del MS-DOS.



c Escriba la secuencia de caracteres `hola` y pulse la tecla 

`C:\WINDOWS>hola` 

Se obtendrá el siguiente resultado:



El MS-DOS puede buscar ficheros ejecutables en directorios distintos del **directorio activo o de trabajo** (C:\WINDOWS en el ejemplo anterior). Para ello, dispone de una variable de nombre **PATH** en la que el sistema operativo guarda una lista de directorios, en los que buscará los programas ejecutables en caso de no encontrarlos en el directorio activo. Debido a sus características, se dice que la variable **PATH** es una variable de entorno. Si se quiere ver la lista de directorios de la variable **PATH**, se puede utilizar la instrucción **PATH** (tienen el mismo nombre). Esta instrucción también sirve para modificar el contenido de la variable de entorno **PATH**.


Cuando el MS-DOS no encuentra un programa ejecutable en el directorio de trabajo, el sistema operativo busca el programa ejecutable en los directorios especificados en la variable **PATH**, en el orden en el que están especificados en ésta. En general, la variable **PATH** se inicializa al cargar el sistema operativo mediante las instrucciones del proceso por lotes que se guardan en el fichero AUTOEXEC.BAT.

Si el programa ejecutable se encuentra en un directorio, distinto del directorio activo, que no está incluido en la lista de la variable **PATH**, el sistema operativo no encontrará el programa y producirá el mensaje de Comando o nombre de archivo incorrecto.

Ejemplo 1.2 *Contenido de la variable **PATH** de nuestro sistema.*


Realice la secuencia de instrucciones que se describe a continuación:


- a Ejecute la instrucción **PATH** sin atributos.

```
C:\WINDOWS\ > PATH 
```

A partir de la respuesta del ordenador, haga una lista de los directorios guardados en la variable PATH.

b Cree un directorio de trabajo y haga que éste sea su directorio activo:

```
C:\WINDOWS> md C:\EPAQ 
```


```
C:\WINDOWS> cd C:\EPAQ 
```

c Cree un subdirectorio de trabajo personal.

```
C:\EPAQ> md Valvanera 
```

```
C:\EPAQ> cd Valvanera 
```

d Ejecute la instrucción PATH sin atributos.

```
C:\EPAQ\Valvanera> PATH 
```

Compruebe que el resultado es el mismo que en el apartado a.

1.2. Estructura y edición de un programa en Fortran 90

En el sistema operativo MS-DOS, los ficheros que contienen el código de un programa en Fortran 90 se especifican añadiendo al nombre del archivo la extensión `for` o `f90`. Así, el nombre prototipo de un fichero que contiene un programa en Fortran 90 es `nombre.for` o `nombre.f90`.

En Fortran, un programa consiste en un conjunto de instrucciones ejecutables y no ejecutables que tienen que seguir un orden específico. Las instrucciones se escriben de acuerdo con una sintaxis y una semántica inteligibles para el usuario y para el programa de traducción (el compilador de Fortran). Por otra parte, para escribir programas en Fortran 90 se pueden utilizar dos tipos de sintaxis: formato fijo y formato libre. En estas prácticas se utilizará la segunda opción. El programa se escribe utilizando un editor de texto, por lo que esta etapa se conoce como edición del programa.

Un programa escrito en Fortran 90 está compuesto por las instrucciones comprendidas entre la instrucción **PROGRAM** `nombre_programa`, que indica el comienzo de un programa, y la instrucción **END PROGRAM** `nombre_programa`, que es la última instrucción. Este programa se conoce como una unidad de tipo Programa Principal.


Dentro de esta unidad (Programa Principal), se pueden distinguir tres partes sucesivas:

- i. **Declaración.** Se encuentra al principio del programa y contiene un grupo de instrucciones no ejecutables que definen el nombre del programa y el número y tipo de las variables que este utiliza.
- ii. **Ejecución.** Contiene una o más instrucciones que describen las acciones que el programa ejecuta.
- iii. **Terminación.** Instrucción o instrucciones que interrumpen y paran la ejecución del programa e indican al compilador que el programa se ha completado.

El siguiente ejemplo muestra la estructura de un programa escrito en Fortran 90.

Ejemplo 1.3 Estructura de un programa en Fortran 90.

Para escribir un programa en Fortran 90 es necesario utilizar un editor de texto. Una posibilidad es utilizar el programa de edición del MS-DOS. Utilizando este editor de texto para escribir el programa *tarea1*, empiece por crear el fichero `tarea1.f90` (se podría llamar de otra manera):

```
C:\EPAQ\Valvanera> EDIT tarea1.f90 
```

A continuación, escriba el siguiente programa en Fortran 90:

```

----- Zona de declaración -----
PROGRAM Tarea1
!
! Este programa calcula la suma y el producto
! de tres numeros enteros
!
! Instrucciones de declaracion
!
IMPLICIT NONE
INTEGER :: i, j, k ! Variables que guardan datos
INTEGER :: suma, producto ! Variables que guardan resultados
!
----- Zona de ejecución -----
i = 24
j = 3
k = 7
suma = i+j+k
producto = i*j*k
!
WRITE(*,*) i, j, k, suma, producto
----- Zona de terminación -----
!
STOP
END PROGRAM Tarea1
-----

```

A continuación, se revisará con más detalle cada una de las partes del programa del ejemplo.

i. Zona de declaración

La zona de declaración consta de una serie de instrucciones no ejecutables que definen el nombre del programa, el nombre de las variables y de qué tipo son estas últimas.

La primera instrucción de esta sección es la instrucción **PROGRAM**:

```
PROGRAM Tarea1
```

Es una instrucción no ejecutable. Cuando el programa de traducción (compilador) se encuentra con ésta, entiende que empieza una unidad Programa Principal y que se llama Tarea1.

A continuación, en el programa del ejemplo, aparecen líneas de comentario que empiezan por el símbolo **!**. El compilador no traduce el texto escrito a la derecha de este símbolo. Estas líneas son opcionales, pero permiten documentar el programa y dejar espacios para leerlo con más claridad. Observe que si en una línea que contiene instrucciones aparece el signo **!**, el resto de la línea no se traduce.

Siguiendo con el programa del ejemplo, la siguiente instrucción:

IMPLICIT NONE

es una instrucción obligatoria y precede a las declaraciones de tipos de variables. Esta instrucción indica al compilador que todas las variables han de tener asignado un tipo de variable.

Finalmente, aparecen las líneas con las instrucciones de declaración de variables. Así, usando la instrucción no ejecutable **INTEGER**, se declaran las variables enteras *i*, *j*, *k*, *suma* y *producto*. Se discutirá esta instrucción más adelante.

ii. Zona de ejecución

Esta sección del programa incluye una o más instrucciones que describen las acciones que realizará el programa.

Las primeras instrucciones ejecutables del programa del ejemplo dan valores a las variables *i*, *j* y *k* utilizando instrucciones de asignación

```
i = 24
j = 3
k = 7
```

que se discutirán más adelante. A continuación, se calculan los valores de la suma y el producto de los números guardados en las variables *i*, *j* y *k*, que se almacenan en las variables *suma* y *producto*, respectivamente.

Finalmente, utilizando la instrucción **WRITE**, se escriben en la pantalla los valores almacenados en las variables *i*, *j*, *k*, *suma* y *producto*

```
WRITE(*,*) i, j, k, suma, producto
```

iii. Zona de terminación

Como se puede observar, esta sección incluye las instrucciones **STOP** y **END PROGRAM** nombre_programa. La instrucción **STOP** indica al procesador que termine la ejecución del programa. Por otro lado, la instrucción **END PROGRAM** nombre_programa indica al compilador que la unidad Programa Principal ha terminado.

Notas de estilo

Algunas convenciones que se utilizarán en los programas de estas prácticas no son parte de la sintaxis del lenguaje Fortran 90.

En la escritura de los programas se utilizará la convención de escribir las palabras clave del Fortran 90 como **PROGRAM**, **INTEGER**, **WRITE** o **STOP**, en mayúsculas. Y para los nombres de variables se utilizarán minúsculas o una combinación de mayúsculas y minúsculas. Estas normas se utilizarán para poder leer mejor los programas y aumentar el orden en la presentación. Sin embargo, a diferencia de otros lenguajes de programación, en Fortran no importa que el nombre de las palabras clave o de las variables se escriba indistintamente con mayúsculas o minúsculas en un programa.

Sintaxis de la instrucción **PROGRAM**

PROGRAM Nombre_de_programa

La primera instrucción de un programa es **PROGRAM** Nombre_de_programa. Es una instrucción no ejecutable que indica al compilador que empieza una unidad de programa llamada Nombre_de_programa. Es obligatorio darle nombre al programa y éste no puede coincidir con el nombre de una variable. Según el ejemplo anterior,

PROGRAM Tarea1

Sintaxis de la instrucción **END PROGRAM**

END PROGRAM Nombre_de_programa

Esta instrucción es la última instrucción del programa. Es una instrucción no ejecutable que indica al compilador donde termina el programa. Es obligatorio indicar el nombre del programa y que coincida con el nombre declarado en la instrucción **PROGRAM**. Según el ejemplo anterior,

END PROGRAM Tarea1

Sintaxis de la instrucción **STOP**


STOP

Esta instrucción indica al procesador que pare la ejecución del programa. El compilador ELF90 asume como obligatorio el uso de la instrucción **STOP** antes de finalizar el programa, es decir, inmediatamente antes de la instrucción **END PROGRAM** nombre_programa. Sin embargo, con otros compiladores la instrucción **STOP** no es de uso obligatorio.

1.3. Compilación, ensamblaje y ejecución de un programa Fortran 90


Para que un programa escrito en Fortran 90 funcione en un ordenador es necesario que sea traducido de forma que el procesador lo entienda. Este proceso se llama compilación y se realiza utilizando un programa compilador de Fortran 90 (en estas prácticas se utiliza el compilador ELF90).

Con el compilador ELF90, para realizar el proceso de compilación y ensamblaje de un programa, se utiliza la instrucción:


```
C:\EPAQ\Valvanera> elf90 nombre.f90 
```

El proceso de compilación comprende dos etapas. En primer lugar, se genera un fichero objeto (`nombre.obj`) que contiene el programa compilado. Después, realizando un proceso de ensamblaje (`LINK`), se obtiene el fichero ejecutable correspondiente (`nombre.exe`). En general, estos dos pasos se producen secuencialmente en respuesta a una sola instrucción del programa compilador. Los detalles del procedimiento dependen del compilador utilizado. Este proceso se puede entender mejor con el ejemplo 1.4 en el que se realizará la compilación y ensamblaje del programa del ejemplo 1.3.

Una vez que generado el fichero `nombre.exe`, el programa se ejecuta escribiendo su nombre

```
C:\EPAQ\Valvanera> nombre.exe 
```


aunque, en general, no se suele escribir la extensión del fichero,

```
C:\EPAQ\Valvanera> nombre 
```

Ejemplo 1.4 *Compilación, ensamblaje y ejecución de un programa escrito en Fortran 90.*

Realice la secuencia de instrucciones que se describe a continuación:

- i Compile y ensamble el programa `tarea1.f90`:

```
C:\EPAQ\Valvanera> elf90 tarea1.f90 
```

Si en el proceso de compilación y ensamblaje no se encuentran errores, la ejecución de esta instrucción produce los siguientes mensajes en la pantalla

```
C:\EPAQ\Valvanera>elf90 tarea1.f90
Essential Lahey Fortran 90 Compiler Release 4.00c S/N: E9014265
Copyright (C) 1994-2000 Lahey Computer Systems. All rights reserved.
Copyright (C) 1985-2000 Intel Corp. All rights reserved.
Registered to: QF
                UR

Options:
-nbind          -nc          -nlst          -out tarea1.exe
-pause         -stack 20000h -nsyn         -winconsole   -nwisk
-nxref
```

```

Compiling file tarea1.f90.
Compiling program unit TAREAL at line 1.
Encountered 0 errors, 0 warnings in file tarea1.f90.

386|LINK: 9.1fix1 -- Copyright (C) 1986-98 Phar Lap Software, Inc.

C:\EPAQ\Valvanera>



```

En este caso, la compilación del programa se completa generando un fichero llamado `tarea1.obj`. A continuación, se inicia el ensamblaje (LINK) del programa y, finalmente, se crea el fichero ejecutable `tarea1.exe`.

Si, por el contrario, apareciesen errores en la compilación del programa, no se generaría el fichero objeto (tipo OBJ), y sería necesario corregir el programa y volverlo a compilar.

- ii Compruebe la existencia de los ficheros `tarea1.obj` y `tarea.exe`, utilizando la instrucción `dir` del sistema operativo MS-DOS

```



C:\EPAQ\Valvanera> dir tarea1.obj 
C:\EPAQ\Valvanera> dir tarea1.exe 

```

- iii Ejecute el programa `tarea1.exe`

Utilice una de las instrucciones siguientes:

```

C:\EPAQ\Valvanera> tarea1.exe 
C:\EPAQ\Valvanera> tarea1 

```

Cuando se ejecute el programa, se obtendrá la siguiente respuesta en la pantalla del ordenador:

```

C:\EPAQ\Valvanera>tarea1

24          3          7          34          504

Program completed
Press Enter to Continue.

```

1.4. Constantes y variables

Existen cinco tipos de datos intrínsecos en Fortran : **INTEGER** (números enteros), **REAL** (números reales), **COMPLEX** (números complejos), **LOGICAL** (valores lógicos, verdadero o falso) y **CHARACTER** (caracteres, texto). Estos tipos de datos pueden ser utilizados como constantes o variables dentro de un programa.

Una variable en Fortran es un objeto, *data object*, que puede cambiar de valor durante la ejecución del programa. Dentro de un programa, cada variable se identifica con un nombre, y el nombre de una variable no puede coincidir con el nombre del programa. En general, las variables se declaran en las

primeras líneas del programa de modo que cada variable queda asociada con un nombre y un tipo de dato (sentencias de declaración de tipo). Cuando el compilador encuentra la declaración de una variable, asocia al nombre de la variable un espacio de memoria, al que accede el procesador cada vez que se utiliza dicha variable. Para modificar el valor de una variable se utiliza una instrucción de asignación cuya descripción se presenta en la próxima sección 1.5.

Una constante en Fortran es un dato que se fija inicialmente y su valor no cambia durante la ejecución del programa. El valor de la constante se guarda en una posición de memoria del ordenador a la que accede el procesador cada vez que el programa usa la constante. A veces, las constantes se declaran con nombre y tipo de dato (sección 1.6).

A continuación, se muestran algunos ejemplos de nombres de variables y de datos numéricos de tipo **INTEGER** y **REAL**.

Nombres de variables

En un programa en Fortran, el nombre de la unidad **PROGRAM** y el de sus variables deben ser únicos, es decir, éstos no pueden coincidir.

El nombre de una variable puede tener hasta 31 caracteres, puede contener letras y números y el símbolo `_`. Sin embargo, el primer carácter siempre tiene que ser una letra.

Ejemplos de nombres de variables correctos son:

tiempo
DeltaH
reactivo1
constante_equilibrio

Algunos ejemplos de nombres de variables incorrectos son:

un_nombre_de_variable_demasiado_largo este nombre tiene más de 31 caracteres
tiempo_1/2 contiene un carácter especial /
concentración no se pueden incluir acentos
aux % contiene el caracter especial %
potencial de onda media no se pueden dejar espacios en blanco

Constantes y variables enteras

El tipo de datos numérico **INTEGER** consiste en constantes y variables enteras, es decir, números sin parte decimal.

Las variables de tipo entero sirven para guardar números enteros. Las variables de tipo entero se declaran con la instrucción **INTEGER** según la forma:

INTEGER :: variable_1 ,variable_2, ...

En esta sentencia, los nombres `variable_1`, `variable_2`, ... que siguen a la palabra **INTEGER** ::, corresponden a los nombres de las variables que se están declarando en el programa como de tipo entero. Por ejemplo, en la sentencia

```
INTEGER :: n, l, m
```

se están declarando las variables de nombres `n`, `l` y `m` como variables de tipo entero en el programa.

Por otra parte, una constante entera en un programa es simplemente un número entero, es decir, un número que no tiene parte decimal. Si el número es positivo, el signo (+) es opcional. Además, una constante entera no puede incluir comas. A diferencia de las variables enteras, las constantes enteras no se declaran.

Ejemplos correctos de constantes enteras son:

```
0
-333
10000
+20
```

Ejemplos incorrectos son,

```
1,000,000  contiene comas
-100.     incluye un punto decimal
```

Constantes y variables reales

El tipo de datos numérico **REAL** consiste en constantes y variables reales, es decir, números con parte decimal.

Las variables de tipo real guardarán números reales. Las variables de tipo real se declaran con la instrucción **REAL** según la sentencia

```
REAL :: variable_1 ,variable_2, ...
```

Los nombres `variable_1`, `variable_2`, ... que siguen a la palabra **REAL** ::, corresponden a los nombres de variables del programa que se están declarando como de tipo real. Por ejemplo, la sentencia

```
REAL :: P, V, T
```

declara las variables del programa `P`, `V` y `T` como variables de tipo real.

Por otra parte, una constante real es simplemente un número que tiene parte decimal. La parte entera y la parte decimal del número se separan mediante un punto (.). Es decir, no puede incluir comas. Si el número es positivo, el signo (+) es opcional. A diferencia de las variables reales, las constantes reales no se declaran.

Ejemplos correctos de constantes reales son:

10.0
10.
-333.6

Ejemplos incorrectos son:

10,0 **contiene comas**
-333 **no tiene parte decimal**

Las constantes reales pueden expresarse en notación científica. En esta notación, junto con la mantisa, se incluye el exponente correspondiente a una potencia de 10, que se expresa con la letra E, seguida del signo + (opcional) ó -, y el valor del exponente. La mantisa del número debe incluir un punto decimal. Ejemplos de constantes reales en notación científica son:

0.001E0 $\equiv 0,001 \times 10^0 = 0,001$
1.0E-3 $\equiv 1,0 \times 10^{-3} = 0,001$
6.02E+23 $\equiv 6,02 \times 10^{23}$
0.12E1 $\equiv 0,12 \times 10^{+1} = 1,2$

Pregunta 1.1 *A continuación se presenta una lista de constantes numéricas.*

Indique si son correctas y a qué tipo de constante numérica (INTEGER o REAL) corresponden.

En caso de no ser correctas indique por qué.

10.0 100,0 123E-5 -32768 3.14159 17.877E+9 13.0^2

Pregunta 1.2 *A continuación se presentan cuatro pares de valores de constantes reales.*

Indique si los valores de cada par son equivalentes en el ordenador.

9.6450	9.65E+3
-12.71	-1.27E1
0.001	1.0E-4
3.14159E0	314.159E-3

Pregunta 1.3 *A continuación se presenta una lista de nombres de variables.*

Indique si los nombres son válidos. En caso de no serlo, justifique por qué.

longitud	radio	1radio
concentracion_inicial	rendimiento_reaccion%	_ok

Pregunta 1.4 Indique si las siguientes instrucciones son válidas. En caso de no serlo, justifique por qué

```
PROGRAM ejemplo_1
PROGRAM 2pH
INTEGER m,l,n
INTEGER :: 10
REAL 10.1E+2
REAL :: deltaH, deltaS, deltaG
REAL :: Keq
REAL :: P V T
REAL :: concentracion %peso
```

Sintaxis de la instrucción IMPLICIT NONE

IMPLICIT NONE

Esta instrucción indica al compilador que todas las variables del programa se van a declarar explícitamente.

Es la primera instrucción de las sentencias de declaración de tipo y, por lo tanto, precede a todas las instrucciones de declaración del programa.

Declaración de variables de tipo entero (INTEGER)

```
INTEGER :: variable_1 [,variable_2, ...]
```

La instrucción **INTEGER** es una instrucción de declaración de tipo. Declara que las variables que aparecen en la lista de variables `variable_1, [,variable_2, ...]` son de tipo entero. La lista debe contener al menos una variable.

Ejemplo,

```
INTEGER :: n, l, m
```

Declaración de variables de tipo real (REAL)

```
REAL :: variable_1 [,variable_2, ...]
```

La instrucción **REAL** es una instrucción de declaración de tipo. Declara que las variables que aparecen en la lista de variables `variable_1, [,variable_2, ...]` son de tipo real. La lista debe contener al menos una variable.

Ejemplo,

```
REAL :: pH, concentracion
```

1.5. Instrucciones de asignación y operaciones aritméticas

Los cálculos en Fortran se especifican mediante *instrucciones de asignación* que tienen la forma general

$$\text{nombre_de_variable} = \text{expresión aritmética/lógica/carácter}$$

Por ejemplo,

```
ass = 3*c + SQRT(4*delta +13.0)
inflamable = .TRUE.
compuesto = ' metano '
```

Cuando se ejecuta una instrucción de asignación, el procesador evalúa el valor de la expresión que se encuentra a la derecha del signo = (operador de asignación) y guarda el valor obtenido en la variable cuyo nombre aparece a la izquierda del signo =.

Note que = no significa *igual a* como en Álgebra, sino que significa *guardar el valor de la expresión en la posición de memoria asociada a nombre_de_variable*. Así, una orden como

$$i = i + 1$$

que no tiene sentido en el álgebra ordinaria, tiene sentido en Fortran. La instrucción indica: tome el valor guardado en la variable *i*, súmele 1, y guarde el resultado de nuevo en la variable *i*.

Si la expresión es una expresión aritmética, esta puede escribirse con una constante numérica, una variable o una combinación de constantes y variables con operadores aritméticos (+, -, *, /, **). Por ejemplo,

```
i = 1
a = 2*b + c/d
y = x**2 + 10.0/r
```

Sintaxis de la instrucción de asignación aritmética

```
Variable_1 = constante (integer o real)
Variable_1 = Variable_2 (integer o real)
Variable_1 = expresión aritmética
```

donde Variable_1 y Variable_2 pueden ser variables numéricas enteras o reales.
Ejemplos,

```
concentracion = 1.0E-3
resultado = suma
deltaG = deltaH - T*deltaS
```

1.5.1. Operadores aritméticos

En Matemáticas, se dispone de una serie de símbolos que indican determinadas operaciones. Así, + indica una suma, - una sustracción, etc. De modo semejante, el lenguaje Fortran utiliza también símbolos, tomados del conjunto de caracteres especiales, para indicar operaciones. En la tabla 1.1, se muestran los operadores aritméticos en Fortran 90, y algunos ejemplos de operaciones aritméticas en notación matemática junto con las expresiones equivalentes en Fortran 90.

Operación	Operador en Fortran 90	Ejemplo
Potenciación	**	$a^b \rightarrow a^{**}b$
División	/	$a/b \rightarrow a/b$
Multiplicación	*	$a \times b \rightarrow a*b$
Resta	-	$a-b \rightarrow a-b$
Suma	+	$a+b \rightarrow a+b$
Cambio de signo	-	$-a \rightarrow -a$

Tabla 1.1: Operadores aritméticos en Fortran 90.

Las expresiones aritméticas en Fortran se escriben como las expresiones algebraicas correspondientes, sin embargo, en la construcción de expresiones aritméticas deben observarse las siguientes reglas:

- i No pueden colocarse dos operadores consecutivos.

¿Cómo se escriben expresiones como x^{-2} ?

`x**-2` no es una expresión válida: hay dos operadores consecutivos

`x**(-2)` expresión válida: se utiliza el paréntesis para indicar el exponente y separar los operadores

- ii Los operadores de multiplicación, división y potenciación deben conectar dos constantes o variables.

`a*b` `b/2.0` `f**2`

- iii Los operadores + y - pueden aparecer acompañados de dos constantes o variables (cuando indican las operaciones de sumar o restar) o preceder a una constante o variable (cuando indican que el signo de un número se mantiene o cambia).

`a + b` `d - 2.0`
`-7` `+a`

- iv La multiplicación, a diferencia de las expresiones algebraicas, se indica de forma explícita. Así, la manera de expresar correctamente $a(b+c)$ es

`a*(b+c)`

Si se intenta compilar un programa que contiene la expresión aritmética:

$$a(b+c)$$

el compilador dará un mensaje de error indicando la existencia de un error de sintaxis (*syntax error*).

- v Los paréntesis pueden utilizarse para agrupar términos de una expresión cuando se desee. Cuando una expresión incluye términos entre paréntesis, éstos son los primeros que se evalúan (ver 1.5.4).

1.5.2. Operaciones con números enteros

Las expresiones aritméticas que trabajan con números enteros dan como resultado otro número entero. Esta afirmación que puede parecer irrelevante es muy importante en el caso de la división: el resultado entero de la división de dos números enteros se calcula a partir del resultado de la división de los dos números reales correspondientes, truncando la parte decimal. Por ejemplo,

$$\begin{aligned}6/2 &= 3 \\4/5 &= 0 \\-5/2 &= -2\end{aligned}$$



Errores de programación. Un error de programación habitual es no tener en cuenta las propiedades de la división de números enteros. Esto puede generar resultados inesperados. Por ejemplo,

$$\begin{aligned}5*4/5 &= 4 \\4/5*5 &= 0\end{aligned}$$

1.5.3. Operaciones con números reales

Si los operandos de una expresión aritmética son reales, entonces el resultado es un valor real.

$$\begin{aligned}6.0/2.0 &= 3.0 \\4.0/5.0 &= 0.8 \\-5.0/2.0 &= -2.5\end{aligned}$$

Una peculiaridad de los números reales es que tienen una precisión limitada. Debido a la longitud de palabra del ordenador (número de bits utilizados para aproximar un número real), algunos números sólo pueden representarse de forma aproximada. Un ejemplo, el número $1/3 = 0,333333\bar{3}$.

Considere las siguientes instrucciones:

$$\begin{aligned}y &= a/b \\z &= b*(a/b)\end{aligned}$$

Considere los casos (i) $a = 1.0$, $b = 3.0$, y (ii) $a = 1.0$ $b = 2.0$:

$$\begin{aligned}y &= 1.0/3.0 = 0.33333333 \\z &= 3.0*(1.0/3.0) = 0.99999999 \neq 1\end{aligned}$$

$$y = 1.0/2.0 = 0.50000000$$

$$z = 2.0*(1.0/2.0) = 1.00000000 = 1$$

Este ejemplo muestra cómo expresiones aritméticas matemáticamente equivalentes conducen a resultados distintos debido a la precisión limitada con la que el ordenador aproxima los números reales. En el caso (ii) se obtiene el resultado esperado, mientras que en (i) aparece una diferencia entre el valor esperado y el valor calculado, conocida como el error de redondeo del cálculo. Como es obvio, es conveniente minimizar estas diferencias ya que la acumulación de los errores de redondeo en la evaluación de expresiones aritméticas de un programa puede dar lugar a resultados erróneos. Existen distintas técnicas para minimizar este tipo de error pero están más allá de los objetivos de este curso.

La presencia del error de redondeo no sólo afecta a la evaluación de expresiones aritméticas en un programa sino que también condiciona el resultado de las operaciones de comparación de dos números reales en un programa. En cualquier caso, el error de redondeo es inherente a los cálculos computacionales y no es una limitación exclusiva del lenguaje Fortran, sino que está presente en los programas escritos en cualquier otro lenguaje de programación y en los programas comerciales de cálculo por ordenador como Excel, MatLab, Origin, etc.



Errores de programación. Debido a la precisión limitada con la que el ordenador aproxima los números reales, la evaluación de una expresión aritmética puede conducir a un resultado distinto del esperado. La diferencia entre ambos valores es el error de redondeo del cálculo. Esta diferencia puede ser pequeña pero la acumulación de los errores de redondeo en un programa puede generar resultados erróneos.

1.5.4. Evaluación de expresiones aritméticas

Cuando en una expresión aritmética aparece más de un operador, el procesador evalúa las operaciones asociadas de acuerdo con las siguientes reglas:

- i. Primero se evalúan las instrucciones entre paréntesis. Si hay varios paréntesis, primero se evaluarán los más internos.
- ii. Se evalúan las potencias.
- iii. Se evalúan las multiplicaciones y divisiones, con igual prioridad.
- iv. Se evalúan las sumas y restas, con igual prioridad.
- v. Si hay dos operaciones de igual prioridad, se evalúan en orden de precedencia, de izquierda a derecha.

Obsérvese el resultado de la evaluación de las siguientes expresiones:

$$4*3/2 \quad \longrightarrow \quad 12/2 \quad \longrightarrow \quad 6$$

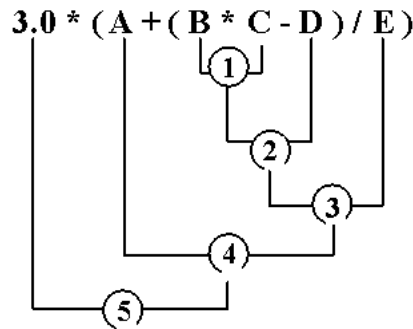
$$4*(3/2) \quad \longrightarrow \quad 4/1 \quad \longrightarrow \quad 4$$

$$3/2*4 \quad \longrightarrow \quad 1 * 4 \quad \longrightarrow \quad 4$$

Una expresión más complicada como:

$$3*(A+(B*C-D)/E)$$

se evalúa según el orden que se muestra en la siguiente figura.

**Ejemplo 1.5** Evaluación de expresiones aritméticas.

Las variables a, b, c, d, e, f, y g, se han inicializado con los valores:

$$a = 3.0 \quad b = 2. \quad c = 5. \quad d = 4. \quad e = 10. \quad f = 2.0 \quad g = 3$$

¿Qué valor se guardará en la variable resultado tras ejecutarse la siguiente instrucción de asignación?

$$\text{resultado} = a * b + c * d + e / f ** g$$

i. Expresión a evaluar.	$a * b + c * d + e / f ** g$
ii. Sustituir variables.	$= 3 * 2. + 5 * 4. + 10. / 2. ** 3$
iii. Evaluar $2. ** 3$	$= 3 * 2. + 5 * 4. + 10. / 8.$
iv. Evaluar multiplicaciones y divisiones de izquierda a derecha	$= 6. + 5 * 4. + 10. / 8.$ $= 6. + 20. + 10. / 8.$
v. Evaluar las sumas y restas de izquierda a derecha	$= 6. + 20. + 1.25$ $= 26. + 1.25$ $= 27.25$

Solución: resultado = 27.25

Ejemplo 1.6 Evaluación de expresiones aritméticas (II).

Considere las variables a, b, c, d, e, f, y g, del ejemplo anterior.

¿Qué valor se guardará en la variable resultado tras ejecutarse la siguiente instrucción de asignación?

$$\text{resultado} = a * (b + c) * d + (e / f) ** g$$

i. Expresión a evaluar.	$a * (b + c) * d + (e / f) ** g$
ii. Sustituir variables.	$= 3 * (2. + 5.) * 4. + (10. / 2.) ** 3$
iii. Evaluar términos entre paréntesis	$= 3 * (7.) * 4. + (10. / 2.) ** 3$ $= 3 * 7. * 4. + (5.) ** 3$
iv. Evaluar $5. ** 3$	$= 3 * 7. * 4. + 125.$

v. Evaluar multiplicaciones y divisiones	= 21.*4. + 125.
de izquierda a derecha	= 84. + 125.
vi. Evaluar la suma	= 209.

Solución: resultado = 209.

Ejemplo 1.7 Evaluación de expresiones aritméticas (III).

Considere las variables a, b, c, d, e, f, y g, del ejemplo anterior.

¿Qué valor se guardará en la variable resultado tras ejecutarse la siguiente instrucción de asignación?

$$\text{resultado} = a*(b + c)*(d + e)/f**g$$

i. Expresión a evaluar.	$a*(b + c)*(d + e)/f**g$
ii. Sustituir variables.	$= 3.*(2. + 5.)*(4. + 10.)/2.**3$
iii. Evaluar términos entre paréntesis	$= 3.*(7.)*(4. + 10.)/2.**3$
	$= 3.*7.*(14.)/2.**3$
iv. Evaluar 2.**3	$= 3.*7.* 14./8.$
v. Evaluar multiplicaciones y divisiones	$= 21.*14./8.$
de izquierda a derecha	$= 294./8.$
	$= 34.75$

Solución: resultado = 34.75

Pregunta 1.5 Evaluación de expresiones aritméticas

Las variables a, b, c se han inicializado con los valores:

$$a = 3 \quad b = 2 \quad c = 3$$


¿Qué valor se guardará en la variable resultado tras ejecutarse la siguiente instrucción de asignación?

- i resultado = a**(b**c)
- ii resultado = (a**b)**c
- iii resultado = a**b**c

Soluciones: 6561, 729, y 6561

Ejemplo 1.8 Ejemplos de operaciones con números enteros.

Con un editor de texto, edite el fichero tarea2.f90:

C:\EPAQ\Valvanera> EDIT tarea2.f90 

donde introducirá el siguiente programa en Fortran 90:

```

PROGRAM Tarea2
!
! Ejemplo de operaciones con numeros enteros y de las reglas de
! evaluacion de operadores
!
IMPLICIT NONE
INTEGER :: i1, i2, j1, j2
INTEGER :: k1, k2
!
i1 = 4*3/2
i2 = 4*(3/2)
WRITE(*,*) 'i1 =', i1, ' i2 =', i2
!
j1 = 3/2*4
j2 = 3/(2*4)
WRITE(*,*) 'j1 =', j1, ' j2 =', j2
!
k1 = 2**3-1
k2 = 2**(3-1)
WRITE(*,*) 'k1 =', k1, ' k2 =', k2
!
STOP
END PROGRAM Tarea2

```


Una vez editado el programa, compílolo y ejecútelo. Compruebe que los resultados impresos en la pantalla coinciden con los que esperaría obtener de la aplicación de las reglas de evaluación de expresiones aritméticas.

Ejemplo 1.9 *Cálculo de la molaridad de una disolución.*

Edite un programa que permita calcular la molaridad de una disolución de acuerdo con la expresión

$$\text{molaridad} = \frac{\text{numero de moles}}{\text{volumen}} = \frac{\frac{\text{gramos de sustancia}}{\text{pesomolecular}}}{\text{volumen}}$$

Con un editor de texto, edite el fichero tarea3.f90:

```
C:\EPAQ\Valvanera> EDIT tarea3.f90 
```

donde introducirá el siguiente programa en Fortran 90:

```

PROGRAM Calculo_Molaridad
!
! Programa para el calculo de la molaridad de una disolucion
! Ngramos ... numero de gramos de sustancia (15.0 gramos)
! GramosMol ... peso molecular de sustancia (36.5 gramos)
! Volumen ... volumen de disolucion (1.0 litros )

```

```

! Molaridad ... molaridad (moles/litro )
!
IMPLICIT NONE
REAL :: Ngramos, GramosMol, Volumen, Molaridad
!
Ngramos = 15.0
GramosMol = 36.5
Volumen = 1.0
Molaridad = (Ngramos/GramosMol )/Volumen
!
WRITE(*,*) "Molaridad = ", Molaridad," moles/litro "
!
STOP
END PROGRAM Calculo_Molaridad

```

Una vez editado el programa, compílelo y ejecútelo.

Compruebe los resultados impresos en la pantalla. Si el programa funciona correctamente, obtendrá un resultado de 0,41095890... moles/litro.

1.5.5. Evaluación de expresiones con mezcla de tipos de constantes o variables

La UAL¹ de un ordenador sólo trabaja con constantes y variables homogéneas, es decir, del mismo tipo. Por ejemplo, no puede multiplicar un número entero por un número real.

Por este motivo, cuando se mezclan en una expresión aritmética variables o constantes de tipo entero y real, antes de poder evaluar la expresión, el procesador convierte los datos enteros en reales. Después, utiliza la aritmética real para evaluar la operación. Así, por ejemplo,

Expresión tipo entero	$3/2 = 1$	(resultado tipo entero)
Expresión tipo real	$3.0/2.0 = 1.5$	(resultado tipo real)
Expresión tipo mixto	$3.0/2 = 1.5$	(resultado tipo real)

En expresiones más complejas, hay que tener en cuenta que la conversión sólo se produce cuando un entero y un real aparecen en la misma operación. Por ejemplo,

$$\begin{aligned}
 1 + 1/4 &= 1 \\
 1.0 + 1/4 &= 1.0 \\
 1.0 + 1/4.0 &= 1.25
 \end{aligned}$$


Errores de programación. Las expresiones de aritmética mixta son peligrosas ya que puede dar lugar a resultados inesperados. Evite utilizarlas en lo posible.

Un caso especial de expresiones mixtas son las potencias. Considere en primer lugar la expresión $y^{**}x$. Cuando el exponente x es real, la expresión se evalúa utilizando la equivalencia algebraica $y^x =$

¹UAL es la unidad aritmético-lógica: parte del procesador del ordenador responsable de realizar operaciones aritméticas y lógicas.

$e^{x \ln y}$. Sin embargo, si el valor del exponente es entero, este modo de operar es poco preciso. Como segundo ejemplo, considere que y es un número negativo, por ejemplo -2.0 . La operación $-2.0**2 = 4.0$, que es el valor esperado, sin embargo $-2.0**2.0$ producirá un error en la ejecución del programa al tratar de evaluar el neperiano de un número negativo.

Pregunta 1.6 Indique si las siguientes expresiones son válidas en Fortran 90. Indique su valor o, si es el caso, la razón por la que no son válidas.

- a. $37/3$ b. $37 + 17/3$ c. $28/3/4$ d. $(28/3)/4$
 e. $28/(3/4)$ f. $-3.**4/2$ g. $-3.**(4/2)$ h. $4.**-3$

Pregunta 1.7 Evalúe el resultado de las siguientes expresiones.

- a. $2+5*2-5$ b. $(2+5)*(2-5)$ c. $2 + (5*2) - 5$ d. $(2+5)*2-5$

Pregunta 1.8 ¿Qué valor almacena la variable resultado tras la ejecución de las instrucciones que se indican a continuación?

```
...
REAL :: a, b, c, resultado
...
a = 10.
b = 1.5
c = 5.0
resultado = a/b + b*c**2
...
```

Pregunta 1.9 ¿Qué valores almacenan las variables a y n tras la ejecución de las instrucciones que se indican a continuación?

```
...
REAL :: a
INTEGER :: n, i, j
...
i = 10
j = 3
n = i/j
a = i/j
...
```

Pregunta 1.10 ¿Qué valores almacenan las variables a y b tras la ejecución de las instrucciones que se indican a continuación?


```
...
REAL :: a, b
...
a = 2.5
b = 3.5
a = b
b = a
...
```

Pregunta 1.11 ¿Qué valores almacenan las variables *a* y *b* tras la ejecución de las instrucciones que se indican a continuación?

```
...
REAL :: a, b, t
...
a = 2.5
b = 3.5
t = a
a = b
b = t
...
```

Ejemplo 1.10 Ejemplos de operaciones con expresiones mixtas.

Con un editor de texto, edite el fichero tarea4.f90:

```
C:\EPAQ\Valvanera> EDIT tarea4.f90 
```

donde introducirá el siguiente programa en Fortran 90:


```
PROGRAM Tarea4
!
! Ejemplo de operaciones con numeros enteros y reales operadores
! aritmeticos
!
IMPLICIT NONE
REAL :: A,B
!
A = 1.0 + 1/2
B = 1 + 1.0/2
WRITE(*,*) ' A = ', A, ' B = ',B
!
STOP
END PROGRAM Tarea4
```

Una vez editado el programa, compílelo y ejecútelo.

Compruebe que los resultados impresos en la pantalla coinciden con los que esperaba obtener de la aplicación de las reglas de evaluación de expresiones aritméticas. ¿Puede explicar la diferencia entre los valores de *A* y *B*?

Ejemplo 1.11 Ejemplos de operaciones con expresiones mixtas (II).

Con un editor de texto, edite el fichero tarea5.f90:

```
C:\EPAQ\Valvanera> EDIT tarea5.f90 
```

donde introducirá el siguiente programa en Fortran 90:

```

PROGRAM Tarea5
!
! Ejemplo de operaciones con numeros enteros y reales
!
IMPLICIT NONE
INTEGER :: J,L,K
REAL :: A,B,X
!
J = 3
K = -2
A= 2.7
B= 3.5
!
X= A + J + K**2 + B
L= A + J + K**2 + B
WRITE(*,*) 'X =',X,' L =',L
!
X= 5*J/4*K
L= 5*J/4.0*K
WRITE(*,*) 'X =',X,' L =',L
!
X= K*3*A/2
L= K*3*A/2.0
WRITE(*,*) 'X =',X,' L =',L
!
STOP
END PROGRAM Tarea5

```

Una vez editado el programa, compílelo y ejecútelo.

Compruebe que los resultados impresos en la pantalla coinciden con los que esperaría obtener de la aplicación de las reglas de evaluación de expresiones aritméticas.

Ejercicio 1.1 *Escritura, compilación y ejecución de un programa sencillo en Fortran 90.*

Suponga que las variables reales X e Y toman los valores

$$\begin{aligned} X &= 3.1 \\ Y &= 4.6 \end{aligned}$$

y que las variables enteras L Y M toman los valores

$$\begin{aligned} L &= 2 \\ M &= -3 \end{aligned}$$

Escriba un programa que calcule y escriba en la pantalla los valores de A (real) y J (entero) que resultan de la evaluación de las siguientes proposiciones:

- a. $A = X - 2*Y + X/2**2$
 $J = X - 2*Y + X/2**2$
- b. $A = 8/M + 3*L$
 $J = 8.0/M - Y$
- c. $A = L/M + X/Y$
 $J = L/(M + X)*Y$
- d. $A = X + 2*Y$
 $A = 2*A + 4$
 $J = A$
- e. $J = L + 3 *M$
 $J = J*J + Y$
 $A = X + J$

Una vez editado el programa, compílo y ejecútelo. Compruebe que los resultados obtenidos concuerdan con los que esperaría obtener de la evaluación manual de estas expresiones, siguiendo las reglas de evaluación de expresiones aritméticas del Fortran 90.

1.6. Constantes con nombre

En ocasiones es conveniente asignar un nombre a una constante, ya sea porque se utiliza mucho en el programa o porque el nombre de la constante tiene un significado. Un ejemplo de este caso son constantes como los números π o e , o el número de Avogadro, N_{av} . Este tipo de constantes se declara introduciendo el atributo **PARAMETER** en una instrucción de declaración de tipo. Por ejemplo, para definir la constante con nombre pi que guardará el valor del número π , se escribe la instrucción

```
REAL, PARAMETER :: pi = 3.141493
```

El valor de la constante pi declarada no se puede cambiar dentro del programa. Si en una instrucción ejecutable posterior se escribe la instrucción de asignación

```
pi = 3.141492654
```

se detectará un error durante la compilación del programa y no se completará el proceso de traducción y ensamblaje.

Por ejemplo, considere la compilación del siguiente programa

```
PROGRAM Area0
IMPLICIT NONE
REAL :: Area, r
REAL, PARAMETER :: pi = 3.141493
!
r = 5.0
pi = 3.141492654
Area = pi *r*r
WRITE(*,*) 'Area del circulo ', Area
!
STOP
END PROGRAM Area0
```


Debido a que hay un error de sintaxis en la séptima línea del programa, el compilador da un mensaje de error:

```

Compiling file area.f90.
Compiling program unit AREA0 at line 2.
Line 7, file area.f90
pi = 3.141492654
  |
FATAL -- Illegal assignment to a PARAMETER variable (see "PARAMETER
Statement" in the Essential Lahey Fortran 90 Reference).

Encountered 1 error, 0 warnings in file area.f90.

```

y el proceso de compilación no se completa.

Pregunta 1.12 Indique si las siguientes de declaraciones de constantes con nombre son válidas y en caso contrario explique por qué no son válidas.

```

INTEGER, PARAMETER nmax = 10
REAL, PARAMETER :: R = 0.082
REAL, PARAMETER :: R = 0.082 Nav = 6.02E23
INTEGER, PARAMETER :: ndatos = 17.12

```

Sintaxis de la declaración de las constantes con nombre

TIPO, **PARAMETER** :: nombre1 = valor1[, nombre2 = valor2, ...]

donde TIPO es el tipo de constante (**REAL**, **INTEGER**, ...), nombre1 es el nombre asignado a la constante y valor1 el valor que se guarda en la constante.

Recuerde que el valor de las constantes declaradas con el atributo **PARAMETER** no puede modificarse durante la ejecución del programa.

Ejemplo de declaración de constantes con nombre,

```

INTEGER, PARAMETER :: muestras = 10, repeticiones = 5
REAL, PARAMETER :: Nav = 6.02E+23

```

Ejemplo 1.12 Cálculo de la presión de un gas ideal.

Escriba un pequeño programa que permita calcular la presión de un gas ideal utilizando la expresión:

$$P = \frac{nRT}{V}$$

Con un editor de texto, edite el fichero tarea6.f90:

```
C:\EPAQ\Valvanera> EDIT tarea6.f90 
```

e introduzca el siguiente programa en Fortran 90:

```
PROGRAM GasIdeal
!
! Programa para el calculo de la presion de un gas ideal
!
IMPLICIT NONE
REAL :: Presion, Volumen, NMoles, Temperatura
REAL, PARAMETER :: R = 0.082
!
NMoles = 1.0
Temperatura = 298.16
Volumen = 5.0
Presion = NMoles * Temperatura * R / Volumen
!
WRITE(*,*) "Presion = ",Presion," atmosferas"
!
STOP
END PROGRAM GasIdeal
```

Una vez editado el programa, compílelo y ejecútelo.

Compruebe que los resultados impresos en la pantalla coinciden con los que esperaba obtener.

1.7. Sentencias de asignación de caracteres

Declaración de variables de tipo texto o carácter

La sintaxis de estas instrucciones es

```
CHARACTER(LEN = numero entero) :: var1[,var2, ...]
```

donde **LEN** corresponde al mayor número de letras que puede almacenar esa variable. No tiene nada que ver con la longitud del nombre de la variable.

Ejemplos,

```
CHARACTER(LEN=16) :: archivo_datos, archivo_resultados
CHARACTER(LEN=20) :: compuesto
```

Las expresiones de asignación de variables de tipo CHARACTER


Las expresiones de asignación tienen la sintaxis:

```
variable1 = constante1
variable1 = variable2
variable1 = expresion tipo CHARACTER
```

donde variable1 y variable2 han sido declaradas anteriormente en el programa como variables de tipo **CHARACTER** y constante1 es una constante de tipo **CHARACTER**. Una constante de este tipo es cualquier texto contenido entre un par de (") o ('). El tipo de datos **CHARACTER** es útil para guardar letras o dígitos como texto.

Ejemplo 1.13 *Uso de variables de tipo CHARACTER.*

Con un editor de texto, edite el fichero tarea7.f90:

```
C:\EPAQ\Valvanera> EDIT tarea7.f90 
```

e introduzca el siguiente programa en Fortran 90:

```
PROGRAM aromaticos
!
! Ejemplo de utilizacion de las variables de tipo texto
!
IMPLICIT NONE
CHARACTER(LEN=20) :: texto_1, texto_2
CHARACTER(LEN=12) :: texto_3
CHARACTER(LEN=10) :: texto_4
!
texto_1 = "benceno "
texto_2 = "tolueno "
texto_3 = texto_1
texto_4 = texto_2
!
WRITE(*,*) texto_3, texto_4
!
STOP
END PROGRAM aromaticos
```

Una vez editado el programa, compílelo y ejecútelo.

Compruebe que los resultados impresos en la pantalla coinciden con los que esperaba obtener.

Práctica 2

Elementos básicos del Fortran 90 (II)

CONTENIDOS

2.1. Las funciones intrínsecas	34
2.2. Lectura y escritura de datos	35
2.2.1. Lectura de datos del teclado con formato libre	35
2.2.2. Escritura de datos en la pantalla con formato libre	37

2.1. Las funciones intrínsecas

En la práctica anterior se vio cómo se pueden escribir en Fortran 90 expresiones que incluyen la suma, la resta, la multiplicación o la división de dos valores utilizando los operadores aritméticos. Además, se vio cómo se pueden traducir expresiones más complicadas utilizando paréntesis y teniendo en cuenta el orden en el que se ejecutan las operaciones.

Sin embargo, hay muchas funciones de uso común en cálculo científico y técnico que no se pueden expresar como sumas, restas, multiplicaciones, divisiones o potenciaciones. Ejemplos de algunas de estas funciones son los logaritmos, las funciones exponenciales, las raíces cuadradas o las funciones trigonométricas. El lenguaje Fortran utiliza las funciones para poder hacer este tipo de cálculos. Las funciones más comunes se incluyen como parte del lenguaje y se denominan *funciones intrínsecas*. Otras funciones menos comunes se pueden implementar mediante las *funciones de usuario* que se describirán en la Práctica 9.

Una función en Fortran proporciona a partir de uno o más valores, conocidos como *argumentos* de la función, un resultado. El argumento puede ser una expresión aritmética, lógica o de tipo carácter. En la tabla 2.1 se recogen algunas funciones intrínsecas comunes.

Las funciones intrínsecas se utilizan como parte de una expresión aritmética. Por ejemplo, si se quiere calcular el seno de un ángulo theta, se escribiría

$$y = \text{SIN}(\text{theta})$$

donde theta es el argumento de la función intrínseca **SIN**.

Esta función es algo especial, como todas las funciones trigonométricas, ya que hay que tener cuidado de expresar el ángulo en radianes. Por ejemplo, sí se tiene en cuenta que 180° equivalen a π radianes, y si theta se expresa en grados, la forma correcta de calcular el seno del ángulo sería

$$y = \text{SIN}(\text{theta}*(3.141593/180.))$$

En este caso podría ser conveniente utilizar una constante con nombre para guardar el factor de conversión:

```
REAL,PARAMETER :: grado_a_radian = 3.141593/180.  
...  
y = SIN(theta*grado_a_radian)
```

Las funciones **REAL**, **INT**, y **NINT** pueden utilizarse para evitar los efectos de mezclar números de distintos tipos. La diferencia entre las funciones **INT** y **NINT** es que la primera transforma el real en entero mediante truncamiento, mientras que la segunda lo hace mediante redondeo. Así, la función **INT**(2.999995) produciría el valor 2 mientras que la función **NINT**(2.999995) daría el valor 3.

función y argumentos	función	tipo argumento	tipo resultado	observaciones
SQRT (x)	\sqrt{x}	R	R	$x \geq 0$
ABS (x)	$ x $	R,I	R,I	
SIN (x)	$\text{sen } x$	R	R	x en radianes
COS (x)	$\text{cos } x$	R	R	x en radianes
TAN (x)	$\text{tan } x$	R	R	x en radianes
EXP (x)	e^x	R	R	
LOG (x)	$\ln x$	R	R	$x > 0$
LOG10 (x)	$\log_{10} x$	R	R	$x > 0$
REAL (x)		I	R	convierte un entero en real
INT (x)		R	I	trunca un número real
NINT (x)		R	I	redondea un número real

Tabla 2.1: Algunas funciones intrínsecas comunes. R: real. I: entero.

2.2. Lectura y escritura de datos

2.2.1. Lectura de datos del teclado con formato libre



Para leer datos del teclado se utiliza la instrucción **READ** en cualquiera de la siguientes formas:

Lectura de datos del teclado

READ(*,*) lista de variables
READ(5,*) lista de variables
READ (unit=5,*) lista de variables
READ (unit=5,fmt=*) lista de variables

Ejemplos,

READ(*,*) temperatura
READ(*,*) muestra, concentracion

Cuando la ejecución del programa alcanza la instrucción de lectura, el cursor aparece parpadeando en la pantalla. Para que el procesador continúe la ejecución, los datos se han de teclear separados por un espacio o por comas. Cuando se completa la lista de datos se pulsa la tecla  ó  para que el procesador los lea y prosiga la ejecución del programa.


Por ejemplo, considere que un programa contiene la instrucción

READ(*,*) A, B, V, lper !A,B,V reales e lper entero

Cuando la ejecución del programa alcance esa línea, aparecerá en la pantalla del ordenador el prompt del sistema operativo

```
C:\EPAQ\Valvanera>_
```

y para que se complete la ejecución del programa, se tendrán que introducir los valores iniciales de las variables, por ejemplo

```
C:\EPAQ\Valvanera> 0.23 13.0 2.7E-8 23 
```

Tras ejecutar esta instrucción en el ordenador, se almacena el valor 0.23 en A, 13.0 en B, $2.7 \cdot 10^{-8}$ en V y 23 en IPER. Esta instrucción equivaldría a las instrucciones de asignación

```
A = 0.23
B = 13.0
C = 2.7E-8
IPER = 23
```

Sin embargo, a diferencia de estas instrucciones de asignaciones, la instrucción **READ**(*,*) nos permite utilizar distintos valores iniciales para las variables en cada ejecución del programa.



Errores de ejecución. Es importante tener en cuenta que los valores de las variables se asignan de acuerdo con el orden en el que aparecen los nombres de las variables en la instrucción **READ**. Si se equivoca al introducir los datos el programa le proporcionará respuestas erróneas.

Cada instrucción **READ**(*,*) inicia la lectura en una nueva línea de la pantalla. Si se teclean datos de más en una línea, éstos se desechan. Por ejemplo considere el programa

```
PROGRAM ejemplo_input
IMPLICIT NONE
INTEGER :: i, j, k, l
READ(*,*) i, j
READ(*,*) k, l
WRITE(*,*) i, j, k, l
STOP
END PROGRAM ejemplo_input
```

Si la entrada de datos es

```
1 2 3 4 
5 6 7 8 
```

Tras ejecutar las instrucciones **READ** del programa, en el ordenador se han almacenado los valores

variable	i	j	k	l
valor	1	2	5	6

2.2.2. Escritura de datos en la pantalla con formato libre

Para escribir datos en la pantalla se utiliza la instrucción **WRITE** en cualquiera de la siguientes formas:

Escritura de datos en pantalla

WRITE(*,*) lista de variables y/o constantes y/o expresiones
WRITE(6,*) lista de variables y/o constantes y/o expresiones
WRITE (**unit**=6,*) lista de variables y/o constantes y/o expresiones
WRITE (**unit**=6,**fmt**=*) lista de variables y/o constantes y/o expresiones

Ejemplos,

WRITE(*,*) "Temperatura ", T
WRITE(*,*) "Constante cinetica", a***EXP**(-B/T), "s-1"

Utilizando esta instrucción, también se pueden dejar líneas en blanco en la pantalla

Escritura de una línea en blanco

WRITE(*,*)
WRITE(6,*)
WRITE (**unit**=6,*)
WRITE (**unit**=6,**fmt**=*)

Por ejemplo, suponga que durante la ejecución de un programa las variables A, B, V, IPER tienen los valores

variable	A	B	V	IPER
valor	0.23	13.0	2.7E-8	23

Si posteriormente el programa contiene la instrucción

WRITE(**unit**=6,*) " Resultados ",A, B, V, Iper

entonces, cuando la ejecución del programa alcance esa línea, en la pantalla del ordenador aparecerá

```
C:\EPAQ\Valvanera> Resultados 0.23 13.0 2.7E-18 23
```

Pregunta 2.1 ¿Qué valores se imprimirán en la pantalla tras la ejecución del programa?.

Responda a la pregunta sin ejecutar el programa.

```
PROGRAM prueba1
IMPLICIT NONE
REAL :: a
INTEGER :: i

a = 0.05
i = NINT(2.*3.141593/a)
a = a*(5/3)
WRITE(*,*) i, a

STOP
END PROGRAM prueba1
```

Solución: 125 0.05

Pregunta 2.2 ¿Qué valores se imprimirán en la pantalla tras la ejecución del programa?.

Responda a la pregunta sin ejecutar el programa.

```
PROGRAM prueba2
IMPLICIT NONE
REAL :: a, b, c
INTEGER :: i, j, k

READ(*,*) i, j, a
READ(*,*) b, k
c = SIN((3.141593/180.)*a)
WRITE(*,*) i, j, k
WRITE(*,*) a, b, c

STOP
END PROGRAM prueba2
```

Utilice los datos de entrada:

```
1, 3
2., 45., 17.
30., 180., 6.
```

Solución:

```
1 3 180
2.00000 30.0000 0.348995E-01
```

Ejemplo 2.1 *Cálculo del pH de una disolución acuosa a partir de la concentración de H^+ .*


El pH se define como

$$\text{pH} = -\log_{10} |H^+|$$

El programa debe incluir los siguientes pasos:

- i Leer $|H^+|$
- ii Calcular pH
- iii Escribir el valor del pH

Con un editor de texto, edite el fichero tarea8.f90:

```
C:\EPAQ\Valvanera> EDIT tarea8.f90 
```

e introduzca el siguiente programa en Fortran 90:

```
PROGRAM peH
!
! Este programa calcula el pH de una disolucion
!
! H ... concentracion de H+ en moles.dm-3
! pH ... -log10(|H+|)
!
IMPLICIT NONE
REAL :: H,pH
!
WRITE(*,*) "Introduzca la concentracion de protones (mol/l)"
READ(*,*) H
pH = -log10(H)
WRITE(*,*) " La disolucion tiene un pH = ",pH
!
STOP
END PROGRAM peH
```

Una vez editado el programa, compílelo y ejecútelo.


Para comprobar el funcionamiento del programa, calcule el pH de disoluciones con concentración de H^+ 0.001, 10^{-7} y 10^{-9} mol.l⁻¹. Compruebe que los valores del pH impresos en la pantalla coinciden con los que ha calculado.

Ejemplo 2.2 *Cálculo de la molaridad de una disolución a partir de los gramos de soluto, los gramos-mol de soluto y el volumen de disolución en dm³.*

- i Leer gramos de soluto

- ii Leer gramos-mol de soluto
- iii Leer volumen de la disolución
- iv Calcular concentración molar
- v Escribir la concentración molar.

Con un editor de texto, edite el fichero tarea9.f90:

```
C:\EPAQ\Valvanera> EDIT tarea9.f90 
```

e introduzca el siguiente programa en Fortran 90:

```
PROGRAM Concentracion
!
! Programa para el calculo de la molaridad de una disolucion
!
IMPLICIT NONE
REAL :: NGramos, GramosMol,Volumen, Molaridad
!
WRITE(*,*) "Introduzca los gramos de soluto"
READ(*,*) NGramos
WRITE(*,*) "Introduzca los gramos por mol de soluto"
READ(*,*) GramosMol
WRITE(*,*) "Introduzca el volumen de la disolucion en litros "
READ(*,*) Volumen
!
Molaridad = (NGramos/GramosMol )/Volumen
!
WRITE(*,*) "Molaridad = ", Molaridad, "moles/ litro "
!
STOP
END PROGRAM Concentracion
```

Una vez editado el programa, compílelo y ejecútelo.

Para comprobar el funcionamiento del programa, calcule la molaridad del agua: 18 g.mol^{-1} , $1000 \text{ gramos.dm}^{-3}$. Compruebe si el valor de la molaridad impreso en la pantalla coincide con el que ha calculado.

Ejercicio 2.1 *Cálculo de la normalidad en valoraciones ácido-base.*

Escriba un algoritmo, y un programa que lo ejecute, para calcular la normalidad de valoraciones ácido-base.

Si tiene problemas para plantearlo, consulte el algoritmo descrito a continuación:

- i Definamos las variables reales:

NBase ... normalidad de la disolución de base (eq-gr/l)

Nacido ... normalidad de la disolución de ácido (eq-gr/l)

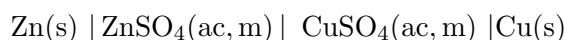
Vbase ... volumen empleado de la disolución de base (ml)

Vacido ... volumen de la disolución de ácido valorada (ml)

- ii Leer NBase, VAcido, VBase
- iii $N_{Acido} = V_{Base} * N_{Base} / V_{Acido}$
- iv Escribir "Normalidad de la base empleada ", NBase
- v Escribir "Volumen de la base empleada ", VBase
- vi Escribir "Volumen del ácido empleado ", VAcido
- vii Escribir "Normalidad del ácido ", Nacido

Ejercicio 2.2 *Escriba un algoritmo, y un programa que lo ejecute, para calcular el potencial estándar de una pila, conocidos los potenciales estándar de sus semielementos.*

Para comprobar el funcionamiento del programa, calcule el potencial estándar de la pila Daniels:



donde los potenciales estándar de los semielementos son $E_{\text{Cu}^{2+}|\text{Cu}} = 0,34 \text{ V}$ y $E_{\text{Zn}^{2+}|\text{Zn}} = -0,76 \text{ V}$

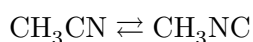
Ejercicio 2.3 *Escriba un programa que utilizando la ley de Arrhenius permita calcular la constante cinética k de una reacción química para distintas temperaturas.*

La ley de Arrhenius permite calcular la constante de velocidad de una reacción a distintas temperaturas y se expresa como:

$$k(T) = A \exp\left(-\frac{E_a}{RT}\right)$$

En esta expresión, A es el factor preexponencial, E_a la energía de activación en $\text{J}\cdot\text{mol}^{-1}$, T la temperatura en K y R es la constante de los gases ideales, $R = 8,31441 \text{ J}\cdot\text{K}^{-1}\cdot\text{mol}^{-1}$.

Para comprobar el funcionamiento del programa, calcule a 298 K la constante cinética de la reacción de isomerización



Esta reacción tiene una energía de activación de $E_a = 1,601 \text{ kJ}\cdot\text{mol}^{-1}$ y un factor preexponencial de $A = 3,98 \cdot 10^{-13} \text{ s}^{-1}$ a 298 K.

Práctica 3

Instrucciones de ejecución condicionada y transferencia de control

CONTENIDOS

3.1. Constantes y variables lógicas	44
3.2. Sentencias de asignación lógicas	45
3.2.1. Expresiones de comparación	45
3.2.2. Operadores de lógica combinatoria	47
3.2.3. Prioridades en la evaluación de expresiones de comparación y lógicas	48
3.3. La instrucción IF (IF lógico)	49
3.4. Bloques IF	50

En las prácticas anteriores, sólo se han escrito programas en los que las instrucciones se ejecutan secuencialmente. Con estas instrucciones se podían escribir programas simples en los que se leían datos, se procesaban y se obtenía una respuesta. Sin embargo, no se podía optar por ejecutar una parte del programa, ni repetir la ejecución de una serie de instrucciones.

En esta práctica se estudiarán las instrucciones que permiten la ejecución selectiva de partes de un programa. Estas instrucciones se denominan instrucciones de ejecución condicionada y de transferencia de control. Antes de discutir la sintaxis de estas instrucciones, se introducirán las expresiones que permiten codificar condiciones. Con este fin, se presentarán las variables de tipo lógico que sólo tienen dos valores posibles: verdadero o falso.

3.1. Constantes y variables lógicas

El tipo de dato lógico sólo puede tomar uno de dos valores: verdadero o falso.

Las constantes lógicas correspondientes son:

```
.TRUE.   (verdadero)
.FALSE.  (falso)
```

que incluyen los puntos a ambos lados del valor para diferenciarse del nombre de una variable.

Ejemplos de expresiones incorrectas de constantes lógicas serían:

```
TRUE     no incluye los puntos, es un nombre de variable
.FALSE   falta el punto después de la constante
```

Por otra parte, las variables de tipo lógico se declaran con la instrucción de declaración de tipo **LOGICAL**

```
LOGICAL :: variable_1 ,variable_2, ...
```

donde los nombres de variable `variable_1,variable_2, ...` que siguen a **LOGICAL**, corresponden a las variables que se declaran como de tipo lógico. Por ejemplo, la instrucción

```
LOGICAL :: corrosivo, acido
```

declara las variables `corrosivo, acido` como variables de tipo lógico. Estas variables sólo pueden tomar los valores `.TRUE.` o `.FALSE.`.

Declaración de variables de tipo lógico (LOGICAL)

```
LOGICAL :: variable_1 [,variable_2, ...]
```

La instrucción **LOGICAL** es una instrucción de declaración de tipo. Declara que las variables que aparecen en la lista de variables `variable_1, [,variable_2, ...]` son de tipo lógico. La lista debe contener al menos una variable.

Ejemplo,

```
LOGICAL :: test1, test2
```


3.2. Sentencias de asignación lógicas

Para dar valor a una variable lógica se puede emplear una función de asignación. En el caso de las variables de tipo lógico, estas expresiones pueden tomar la forma

```
variable_lógica = expresión de comparación
variable_lógica = expresión de relación
variable_lógica = constante lógica
```

3.2.1. Expresiones de comparación

Las expresiones de comparación comparan variables, expresiones o constantes del mismo tipo entre sí. La forma general de estas expresiones es

$$\text{expresion_1.operador.expresion_2}$$

donde *expresion_1* y *expresion_2* pueden ser expresiones aritméticas, variables o constantes numéricas o de tipo carácter, y *operador* es un operador de comparación. En la tabla 3.1 se incluyen los operadores de comparación del Fortran 90. También se incluye una columna (estilo antiguo) con la forma de los operadores en estándares anteriores del lenguaje Fortran.

Si el resultado de la comparación es cierto, la evaluación de la expresión dará el valor `.TRUE.`, y, si el resultado no es cierto, dará el valor `.FALSE.`. Algunos ejemplos de expresiones y resultados son

```
expresión   3 < 4   3 <= 4   3 == 4   3 > 4   3 >= 4   4 <= 4
resultado   .TRUE.   .TRUE.   .FALSE.  .FALSE.  .FALSE.  .TRUE.
```

definición	operador (F90)	operador (estilo antiguo)	expresión
¿son a y b iguales?	==	.EQ.	a == b
¿son a y b diferentes?	\=	.NE.	a \= b
¿es a mayor que b?	>	.GT.	a > b
¿es a mayor o igual que b?	>=	.GE.	a >= b
¿es a menor que b?	<	.LT.	a < b
¿es a menor o igual que b?	<=	.LE.	a <= b

Tabla 3.1: Operadores lógicos de comparación



Errores de sintaxis. Hay que tener cuidado para no confundir el operador lógico de igualdad (==) con el operador de asignación (=)

En la jerarquía de evaluación de expresiones, los operadores de comparación tienen menor prioridad que las operaciones aritméticas. Así, las expresiones

$$7 + 3 < 2 + 11$$

$$(7 + 3) < (2 + 11)$$

son equivalentes, y ambas producen el resultado `.TRUE.`

Es importante tener en cuenta que si se comparan un valor entero y uno real, el entero se transforma en real antes de realizar la comparación.

Pregunta 3.1 *¿Qué valores se imprimirán en la pantalla tras la ejecución del programa?*

Responda a la pregunta sin ejecutar el programa.

```
PROGRAM prueba3
IMPLICIT NONE
REAL :: a
INTEGER :: i
LOGICAL :: l

a = 0.05
i = NINT(2.*3.141593/a)
l = i > 100
WRITE(*,*) " i > a ", l

STOP
END PROGRAM prueba3
```

Solución: `i > a T` (T equivale a `.TRUE.` cuando se escribe en la pantalla. Si la respuesta fuera `.FALSE.` en la pantalla se escribiría F).

Ejemplo 3.1 *Uso de los operadores de comparación.*

Con un editor de texto, edite el fichero `tarea10.f90`:

```
C:\EPAQ\Valvanera> EDIT tarea10.f90 
```

e introduzca el siguiente programa en Fortran 90:

```
PROGRAM OperadoresComparacion
!
! Este programa muestra ejemplos de operadores de comparacion.
!
IMPLICIT NONE
INTEGER :: A,B
!
WRITE(*,*) "Introduzca dos numeros enteros"
READ(*,*) A,B
WRITE(*,*) " Es A mayor que B? ", A>B
```

```

WRITE(*,*) " Es A mayor o igual que B? ", A>=B
WRITE(*,*) " Es A igual que B? ", A==B
WRITE(*,*) " Es A distinto que B? ", A/=B
WRITE(*,*) " Es A menor que B? ", A<B
WRITE(*,*) " Es A menor o igual que B? ", A<=B
!
STOP
END PROGRAM OperadoresComparacion

```

Una vez editado el programa, compílelo y ejecútelo.

La ejecución del programa da como resultado en la pantalla:

```

C:\EPAQ\Valvanera>tarea10

Introduzca dos numeros enteros 6 7

Es A mayor que B? F
Es A mayor o igual que B? F
Es A igual que B? F
Es A distinto que B? T
Es A menor que B? T
Es A menor o igual que B? T

Program completed
Press Enter to Continue.

```

Compruebe si estos resultados se corresponden con los que esperaría obtener de la aplicación de las reglas de evaluación de expresiones lógicas.

3.2.2. Operadores de lógica combinatoria

Para poder hacer comparaciones más complejas que las estudiadas, Fortran dispone de otro conjunto de operadores lógicos. Estos operadores permiten realizar operaciones de lógica combinatoria. Las expresiones de comparación lógicas comparan variables, expresiones o constantes lógicas. El resultado de la comparación es una constante lógica (verdadero o falso).

La forma general de estas expresiones es

$$\text{expresion_1.operador.expresion_2}$$

donde `expresion_1` y `expresion_2` pueden ser expresiones, variables o constantes lógicas.

Un caso particular es cuando se utiliza el operador `.NOT`.

$$.NOT.\text{expresión_1}$$

porque este operador sólo se aplica a una expresión.

En la tabla 3.2 se presentan los operadores lógicos del Fortran 90.

operador	función	definición
.AND.	AND lógico	log1.AND.log2 .TRUE. si log1 y log2 son .TRUE.
.OR.	OR lógico	log1.OR.log2 .TRUE. si log1 y/o log2 es .TRUE.
.EQV.	equivalencia	log1.EQV.log2 .TRUE. si log1 y log2 son ambos .TRUE. o .FALSE.
.NEQV.	no equivalencia	log1.NEQV.log2 .TRUE. si uno es .TRUE. y el otro .FALSE.
.NOT.	NOT lógico	.NOT.log2 .TRUE. si log2 es .FALSE. y viceversa

Tabla 3.2: Operadores de lógica combinatoria

Los operadores lógicos no pueden tener operandos numéricos. Así, si un programa incluyera la instrucción

```
L = 3 .AND. 2
```

al compilarlo se encontraría un error de sintaxis y el compilador enviaría a la pantalla el siguiente mensaje de error

```
L = 3 .AND. 2
      |
      v
FATAL -- Shapes, types or type parameters for this operation are
incompatible.
```

3.2.3. Prioridades en la evaluación de expresiones de comparación y lógicas

Una expresión lógica puede constar de:

- Una constante o una variable lógica.
- Una llamada a una función lógica.
- Una expresión que emplee operadores de relación.
- Una expresión que emplee operadores lógicos.
- Una expresión que emplee operadores lógicos y de relación.

En la evaluación de estas expresiones se sigue una serie de prioridades:

- Se evalúan todas las operaciones aritméticas de acuerdo con su orden de preferencia.
- Se evalúan las llamadas a funciones lógicas.
- Se evalúan las expresiones con operadores de relación, trabajando de izquierda a derecha. Todos los operadores de relación tienen igual prioridad.
- Los operadores lógicos se evalúan de acuerdo con las prioridades:
.NOT. >.AND. >.OR. >.EQV. >.NEQV.

Como sucede con las expresiones aritméticas, cuando en una expresión lógica aparecen operadores de igual prioridad, las operaciones se ejecutan de izquierda a derecha, aunque las operaciones entre paréntesis siempre se evalúan en primer lugar.

Pregunta 3.2 Indique si las expresiones son válidas en Fortran 90. Indique su valor o la razón por la que no son válidas.

- | | | |
|--------------------------|------------------------|------------------|
| a. 5.5 >= 5 | b. 20 >20 | c. .NOT. 6>5 |
| d. 15 <= 'A' | e. .TRUE. >.FALSE. | f. 35/17. >35/17 |
| g. 7 <= 8 .EQV. 3/2 == 1 | h. 17.5 .AND. (3.3>2.) | |

Pregunta 3.3 Suponga que las variables reales *a*, *b* y *c* contienen los valores $-10.$, 0.1 y 2.1 , respectivamente, y que las variables lógicas *l1*, *l2* y *l3* contienen los valores `.TRUE.`, `.FALSE.` y `.FALSE.`, respectivamente.

Indique si las siguientes expresiones son válidas en Fortran 90. Indique su valor o la razón por la que no son válidas.

- | | |
|---------------------------------------|---------------------------------------|
| a. <code>a >b .OR. b >c</code> | b. <code>(.NOT. a) .OR. l1</code> |
| c. <code>l1 .OR. .NOT. l2</code> | d. <code>a <b .EQV. b <c</code> |
| e. <code>l1 .OR. l2 .EQV. l3</code> | f. <code>a .OR. b .AND. l1</code> |
| g. <code>l1 .OR. (l2 .AND. l3)</code> | h. <code>(l1 .OR. l2) .AND. l3</code> |

3.3. La instrucción IF (IF lógico)

Esta instrucción permite ejecutar una instrucción sólo cuando una condición se cumple. Su estructura algorítmica podría ser:

```

Si condición verdadera
  instrucción ejecutable
Fin Si

```

La sintaxis de esta instrucción es

```
IF (expresion_logica) instruccion
```

donde la instrucción ejecutable `instruccion` sólo se ejecuta cuando `expresion_logica` es cierta.

Por ejemplo,

```
IF (DeltaH > 0.0) WRITE(*,*) ' La reacción es endotérmica, Entalpia = ', DeltaH
```

Sintaxis de la instrucción IF lógico

```
IF (expresion_logica) instruccion
```

Ejecuta `instruccion` (instrucción ejecutable) sólo si `expresion_logica` es verdadera.

Ejemplo,

```
IF (pH > 7.0) WRITE(*,*) ' Solucion basica, pH = ', pH
```

3.4. Bloques IF

En su forma más sencilla, el BLOQUE IF permite ejecutar una serie de instrucciones si una condición se cumple. Su estructura algorítmica podría ser:

```

Si condición verdadera
    instrucción_1
    [instrucción_2]
    ...
Fin Si
  
```

La sintaxis del bloque IF es

```

IF (expresion_logica) THEN
    instruccion_1
    [instruccion_2]
    ...
END IF
  
```

donde las instrucciones ejecutables instruccion_1, ... sólo se ejecutarán siempre y cuando expresion_logica sea cierta. Si expresion_logica es falsa, el procesador ignorará las instrucciones y continuará en la línea siguiente a **END IF**.

Por ejemplo,

```

IF (DeltaG < 0.0) THEN
    WRITE(*,*) ' La reacción es espontanea, DeltaG = ', DeltaG
END IF
  
```

```

IF (DeltaH < 0.0) THEN
    WRITE(*,*) ' La reacción es exotérmica'
    WRITE(*,*) ' Entalpia = ', DeltaH
END IF
  
```

Sintaxis de los bloques IF (I)

```

IF (expresion_logica) THEN
    instruccion_1
    [instruccion_2]
    ...
END IF
  
```

Ejecuta las instrucciones comprendidas entre **IF** y **END IF** si expresion_logica es cierta. En cualquier caso, la ejecución del programa continúa con las instrucciones que siguen a **END IF**. Ejemplo,

```

IF (pH > 7) THEN
    WRITE(*,*) ' Solucion basica, pH', pH
END IF
  
```

Si se necesita ejecutar dos series de instrucciones alternativas, la estructura algorítmica sería

```

Si condición verdadera
    instrucción_1a
    [instrucción_2a]
    ...
Si condición falsa
    instrucción_1b
    [instrucción_2b]
    ...
Fin Si
  
```

La sintaxis de este bloque IF es

```

IF (expresion_logica) THEN
    instrucion_1a
    [instruccion_2a]
    ...
ELSE
    instrucion_1b
    [instruccion_2b]
    ...
END IF
  
```

Este bloque IF se suele aplicar, por ejemplo, cuando un valor negativo de una variable conduce a un *error de ejecución*. Introduciendo una condición para seleccionar valores válidos de la variable, se puede evitar la ejecución errónea. Esto se muestra en el siguiente fragmento de programa:

```

IF (x > 0.0) THEN
    y = log(x)
ELSE
    WRITE(*,*) "Valor erroneo de x: x<0 "
    STOP
END IF
  
```

Se pueden construir estructuras de mayor complejidad en casos en los que hay más de dos alternativas. Por ejemplo, si hay tres alternativas mutuamente excluyentes:

```

Si condición_1 verdadera
    instrucción_1a
    [instrucción_2a]
    ...
Si condición_1 falsa y condición_2 verdadera
    instrucción_1b
    [instrucción_2b]
    ...
Si condición_1 falsa y condición_2 falsa
    instrucción_1c
    [instrucción_2c]
    ...
Fin Si
  
```

Sintaxis de los bloques IF (II)

```

IF (expresion_logica) THEN
instruccion_1a
[instruccion_2a]
...
ELSE
instruccion_1b
[instruccion_2b]
...
END IF

```

Si la expresión lógica es cierta, se ejecutan las instrucciones comprendidas entre **IF** y **ELSE**, mientras que si es falsa se ejecutan las comprendidas entre **ELSE** y **END IF**.

Ejemplo,

```

IF (x > 0.0) THEN
y = log(x)
ELSE
WRITE(*,*) "Valor erroneo de x: x<0 "
STOP
END IF

```

La sintaxis de esta estructura en Fortran 90 es

```

IF (expresion_logica_1) THEN
instruccion_1a
...
ELSE IF (expresion_logica_2) THEN
instruccion_1b
...
ELSE
instruccion_1c
...
END IF

```

Por ejemplo, la estructura anterior es útil cuando el valor de una variable sirve para clasificar resultados en distintos grupos:

```

IF (DeltaH < 1.0E-10) THEN
WRITE(*,*) ' La reacción es exotérmica'
WRITE(*,*) ' Entalpia = ', DeltaH
ELSE IF (DeltaH > -1.0E-10) THEN
WRITE(*,*) ' La reacción es endotérmica'
WRITE(*,*) ' Entalpia = ', DeltaH
ELSE
WRITE(*,*) ' La reacción es termoneutra'
WRITE(*,*) ' Entalpia = ', DeltaH
END IF

```


Sintaxis de los bloques IF (III)

```

IF (expresion_logica_1) THEN
instruccion_1a
[instruccion_2a]
...
ELSE IF (expresion_logica_2) THEN
instruccion_1b
[instruccion_2b]
...
ELSE
instruccion_1c
[instruccion_2c]
...
END IF

```

Si la expresion_logica_1 es cierta, se ejecutan las instrucciones comprendidas entre **IF** y **ELSE IF**. Si la expresion_logica_1 es falsa, se evalúa la expresion_logica_2, y si ésta es cierta se ejecutan las instrucciones comprendidas entre **ELSE IF** y **ELSE**. Si no son ciertas ninguna de las expresiones lógicas, se ejecutan las instrucciones comprendidas entre **ELSE** y **END IF**.

Ejemplo,

```

IF (pH > 7.01) THEN
WRITE(*,*) "disolucion basica"
ELSE IF (pH < 6.99) THEN
WRITE(*,*) "disolucion acida"
ELSE
WRITE(*,*) "disolucion neutra "
END IF

```

Pregunta 3.4 *Las siguientes instrucciones pretenden alertar cuando los valores de temperatura de un reactor químico son anormales. ¿Es correcta la sintaxis del bloque IF?*

```

IF (temp < 36.5) THEN
WRITE(*,*) "Valores inferiores a los normales"
ELSE IF (temp >38.) THEN
WRITE(*,*) "Temperatura alta"
END IF

```

Pregunta 3.5 *Considere las siguientes instrucciones. Indique si son correctas.*

```

IF ( voltios < 205) THEN
WRITE(*,*) "ATENCION: Voltaje de linea bajo"
ELSE IF (voltios > 235) THEN

```

```
WRITE(*,*) "ATENCION: Voltaje de linea alto"  
ELSE  
WRITE(*,*) "Voltaje de linea dentro de la tolerancia"  
END IF
```

Ejercicio 3.1 Clasificación de muestras.

Suponga que quiere escribir un programa para clasificar una serie de muestras de acuerdo con el valor de una propiedad.

De acuerdo con dicho valor, las muestras se clasifican en los tipos siguientes:

$\text{valor} \geq 95$	A
$85 \leq \text{valor} < 95$	B
$75 \leq \text{valor} < 85$	C
$65 \leq \text{valor} < 75$	D
$\text{valor} \leq 65$	E

Escriba un bloque **IF** que clasifique las muestras de acuerdo con la tabla anterior.

Solución Una posible solución al problema sería

```
IF (valor >= 95) THEN  
  tipo = "A"  
ELSE IF (valor >= 85) THEN  
  tipo = "B"  
ELSE IF (valor >= 75) THEN  
  tipo = "C"  
ELSE IF (valor >= 65) THEN  
  tipo = "D"  
ELSE  
  tipo = "E"  
END IF
```

Ejemplo 3.2 Ordenar tres números enteros de menor a mayor.

El programa incluirá los siguientes pasos:

- i Leer tres números enteros i,j,k
- ii Ordenar los números
- iii Escribir el resultado

Este algoritmo se puede describir con mayor detalle:

- i Pedir al usuario tres valores enteros i,j,k
- ii Leer tres números enteros i,j,k

iii Ordenar los números

Si $i < j$ intercambiamos los valores de i y j

Si $i < k$ intercambiamos los valores de i y k

Si $k < j$ intercambiamos los valores de k y j

iv Escribir el resultado

Con un editor de texto, edite el fichero tarea11.f90:

```
C:\EPAQ\Valvanera> EDIT tarea11.f90 
```

donde introducirá el siguiente programa en Fortran 90:

```
PROGRAM orden3
!
! Este programa muestra ejemplos de operadores de comparacion.
! El programa lee tres numeros y los ordena de menor a mayor.
!
! Instrucciones de declaracion
!
IMPLICIT NONE
INTEGER :: x,y,z
INTEGER :: aux
!
WRITE(*,*) "Introduzca tres numeros"
READ(*,*) x,y,z
WRITE(*,*) x,y,z
! Comprobamos que no hay dos iguales
IF ((x==y).OR.(x==z).OR.(y==z)) STOP
IF (x>y) THEN
  aux = x
  x = y
  y = aux
END IF
IF (x>z) THEN
  aux = x
  x = z
  z = aux
END IF
IF (y>z) THEN
  aux = y
  y = z
  z = aux
END IF
WRITE(*,*) x,y,z
!
STOP
END PROGRAM orden3
```

Una vez editado el programa, compílelo y ejecútelo.

Utilice los valores 5, 4, 1 y 3, 39, 39 para comprobar el funcionamiento correcto del programa.

Ejemplo 3.3 *Corrosión electroquímica de un metal en ausencia de oxígeno en medio ácido.*

La oxidación de un metal en una disolución acuosa en medio ácido y en ausencia de oxígeno se produce cuando su potencial estándar de reducción es menor que el potencial del hidrógeno. El potencial estándar del hidrógeno varía con el pH de acuerdo con la expresión

$$E(\text{H}^+|\text{H}) = -0,059 \text{ pH V}$$

El programa incluirá los siguientes pasos:

- i Leer pH de la disolución
- ii Leer potencial de reducción del metal, E_M
- iii Calcular el potencial de reducción del H^+ , E_H
- iv Comparar E_M y E_H
- v Escribir el resultado

Con un editor de texto, edite el fichero tarea12.f90:

```
C:\EPAQ\Valvanera> EDIT tarea12.f90 
```

donde introducirá el siguiente programa en Fortran 90:

```
PROGRAM Corrosion
!
! Instrucciones de declaracion
!
IMPLICIT NONE
REAL :: pH, eh, eme
!
WRITE(*,*) "Introduzca el pH de la disolucion "
READ(*,*) pH
WRITE(*,*) "Introduzca el potencial estandar de reduccion del metal "
READ(*,*) eme
!
eh = -0.059*pH
WRITE(*,*) " Potencial estandar de reduccion del metal ", eme
WRITE(*,*) " Potencial de reduccion del hidrogeno ", eh
IF (eh > eme) THEN
  WRITE(*,*) " El metal se oxida a pH>= ", pH
ELSE
  WRITE(*,*) " El metal no se oxida a pH<= ", pH
END IF
!
STOP
END PROGRAM Corrosion
```

Una vez editado el programa, compílelo y ejecútelo.

Compruebe el funcionamiento del programa utilizando los potenciales estándar de reducción de los metales Li y Ag:

$$E(\text{Li}^+|\text{Li}) = -3,405 \text{ V}$$

$$E(\text{Ag}^+|\text{Ag}) = 0,071 \text{ V}$$

y los valores del pH: 1, 3, 5 y 7.

Por ejemplo, si se considera la oxidación de la plata a pH = 7, se obtendrá la siguiente respuesta en la pantalla del ordenador:

```
C:\EPAQ\Valvanera>tarea12

Introduzca el pH de la disolucion 7

Introduzca el potencial estandar de reduccion del metal 0.071

Potencial estandar de reduccion del metal      0.710000E-01
Potencial de reduccion del hidrogeno      -0.413000
El metal no se oxida a pH<=      7.00000

Program completed
Press Enter to Continue.
```

Ejemplo 3.4 Evaluación de una función $f(x, y)$ a partir de un par de valores x e y especificados por el usuario.

En este ejercicio utilice la función

$$f(x, y) = \begin{cases} x + y & x \geq 0 \ y \geq 0 \\ x^2 + y & x < 0 \ y \geq 0 \\ x + y^2 & x \geq 0 \ y < 0 \\ x^2 + y^2 & x < 0 \ y < 0 \end{cases}$$

La evaluación de la función depende del signo de las dos variables independientes x e y . Para decidir qué ecuación debe aplicar es necesario comprobar el signo de los valores de x e y .

El programa incluirá los siguientes pasos:

- i Leer los valores de x e y
- ii Calcular $f(x, y)$
- iii Escribir el resultado

Este algoritmo se puede describir con mayor detalle:

- i Pedir al usuario un par de valores de x e y
- ii Leer los valores de x e y
- iii Escribir los valores de entrada de x e y
- iv Comprobar el signo y calcular f(x,y)

```
Si  $x \geq 0$  e  $y \geq 0$  entonces  
f(x,y)  $\leftarrow x + y$   
Si no, y si  $x \geq 0$  e  $y < 0$  entonces  
f(x,y)  $\leftarrow x + y^{**2}$   
Si no y si  $x < 0$  e  $y \geq 0$  entonces  
f(x,y)  $\leftarrow x^{**2} + y$   
Si no  
f(x,y)  $\leftarrow x^{**2} + y^{**2}$ 
```

- v Escribir el resultado: f(x,y)

Con un editor de texto, edite el fichero tarea13.f90:

```
C:\EPAQ\Valvanera> EDIT tarea13.f90 
```

donde introducirá el siguiente programa en Fortran 90:

```
PROGRAM funxy  
!  
! Este programa evalua una funcion f(x,y) a partir de los valores de x e y  
! especificados por el usuario.  
! f(x,y) se define como:  
!  
!  
!           |  
!           | X + Y           X >= 0 and Y >= 0  
!           | X + Y**2       X >= 0 and Y < 0  
! f(X,Y) = | X**2 + Y       X < 0  and Y >= 0  
!           | X**2 + Y**2    X < 0  and Y < 0  
!           |  
!  
! Declaracion de variables  
!  
IMPLICIT NONE  
REAL :: x, y          ! variables independientes  
REAL :: fxy          ! resultado de evaluar la funcion  
!  
! Pide al usuario los valores de x e y  
!  
WRITE (*,*) 'Introduzca los valores de x e y: '  
READ (*,*) x, y
```

```

!
! Escribe los valores de x e y
!
WRITE (*,*) 'Los valores de x e y son: ', x, y
!
! Calculo de f(x,y) teniendo en cuenta los signos de x e y
!
IF ( ( x >= 0.) .AND. ( y >= 0.) ) THEN
  fxy = x + y
ELSE IF ( ( x >= 0.) .AND. ( y < 0.) ) THEN
  fxy = x + y**2
ELSE IF ( ( x < 0. ) .AND. ( y >= 0. ) ) THEN
  fxy = x**2 + y
ELSE
  fxy = x**2 + y**2
END IF
!
! Escribe el valor de la funcion
!
WRITE (*,*) 'El valor de f(x,y) es: ', fxy
!
STOP
END PROGRAM funxy

```

Una vez editado el programa, compílelo y ejecútelo.

Compruebe el funcionamiento del programa utilizando los pares de valores de x,y: (3,2), (3,-2), (-3,2) y (-3,-2). Los valores de la función f(x,y) correspondientes a dichos pares son:

$$f(3, 2) = 3 + 2 = 5$$

$$f(3, -2) = 3 + (-2)^2 = 7$$

$$f(-3, 2) = (-3)^2 + 2 = 11$$

$$f(-3, -2) = (-3)^2 + (-2)^2 = 13$$

¿Coinciden con los que ha obtenido en la pantalla?.

Práctica **4**

Instrucciones de control de flujo (I)

CONTENIDOS

4.1. Ejercicios de repaso	62
4.2. Bucles DO WHILE - END DO	62
4.2.1. Estructuras anidadas	66
4.3. Bucles DO iterativos	67
4.3.1. Cálculos de tablas de valores	76
4.4. Lectura y escritura de datos con ficheros (sin formato)	80

En las prácticas anteriores, se han escrito programas en los que instrucciones se ejecutan secuencialmente o bien de forma condicionada. En esta práctica se estudiarán las instrucciones que permiten la ejecución repetitiva de partes de un programa. Estas instrucciones se denominan instrucciones de control de flujo.

Por otra parte, también se introducirán las instrucciones que permiten leer datos de ficheros o escribir datos en ficheros del ordenador. Estas operaciones son muy útiles para el tratamiento de datos. Por ejemplo, la lectura nos permite analizar los datos que nos ha proporcionado un aparato del laboratorio (un espectrofotómetro, un conductímetro, etc) y la escritura nos permite guardar los resultados obtenidos en el ordenador. Después, estos resultados se podrían representar con un programa de gráficos científicos como Origin o Gnuplot.

4.1. Ejercicios de repaso

Ejercicio 4.1 *Escriba un programa que lea el pH de una disolución acuosa e imprima el valor de la concentración de protones, y que indique si ésta es ácida, básica o neutra.*

Considere como disoluciones neutras todas aquellas cuyo pH esté comprendido entre 6.99 y 7.01.

Ejercicio 4.2 *Escriba un programa que calcule las raíces de un polinomio de segundo grado (es decir los valores de x para los que $Ax^2 + Bx + C = 0$).*

En el programa contemple la posibilidad de que las raíces sean reales e iguales, reales y distintas, y complejas.

Compruebe el funcionamiento del programa calculando las raíces de los polinomios $x^2 + 1$, $x^2 - 1$ y $x^2 + 2x + 1$

4.2. Bucles DO WHILE - END DO

Esta construcción permite repetir la ejecución de un conjunto de instrucciones siempre y cuando una condición que se ha especificado sea cierta. Su algoritmo se puede expresar como

```
Mientras que condición verdadera
Instrucción 1
.....
Instrucción n
Fin del Mientras que
```

La sintaxis de esta construcción es:

```
DO WHILE (expresion logica)
instruccion_1
[instruccion_2]
...
END DO
```

donde las instrucciones ejecutables `instruccion_1`, `instruccion_2` ... comprendidas entre **DO WHILE** y **END DO** constituyen el cuerpo del bucle DO.

Cuando el procesador alcanza la instrucción **DO WHILE** (expresion logica) realiza las siguientes operaciones:

- i Evalúa la condición.
- ii.1 Si la condición es falsa, la ejecución del programa continúa en la sentencia siguiente a la instrucción **END DO**.
- ii.2 Si la condición es cierta, ejecuta las instrucciones comprendidas entre **DO WHILE** y **END DO**. Este grupo de instrucciones es el cuerpo del bucle DO.
 Cuando se alcanza la instrucción **END DO**, se evalúa de nuevo la condición. Si está es falsa, la ejecución de las instrucciones del bucle DO ha terminado y la ejecución del programa continúa en la sentencia siguiente a **END DO**. Si la condición es cierta, se repite de nuevo la ejecución de las sentencias comprendidas en el cuerpo del bucle.
 ii.2 se repite hasta que la condición sea falsa. Si la condición nunca es falsa, el programa se queda atrapado repitiendo el bucle.

Un ejemplo de esta estructura es:

```

r = 0.0
DO WHILE (r <= 5.0)
  Pr = 4.0*r*r*EXP(-2.0*r)
  r = r + 0.1
  WRITE(*,*) r, Pr
END DO

```

Sintaxis de los bucles DO WHILE

```

DO WHILE (expresion logica)
instruccion_1
[instruccion_2]
...
END DO

```

Si expresion logica es cierta, se ejecutan las instrucciones del programa comprendidas entre **DO WHILE** y **END DO**. Tras la ejecución de éstas se evalúa de nuevo expresion logica. Si es cierta de nuevo, se vuelve a repetir la ejecución de las instrucciones comprendidas en el bucle **DO**.

Si expresion logica es falsa, la ejecución del programa continúa con las instrucciones que siguen a **END DO**.

Ejemplo,

```

DO WHILE (l < 10)
  l = l+1
END DO

```

Pregunta 4.1 ¿Qué valores se imprimirán en la pantalla tras la ejecución de las siguientes instrucciones?.

```
J = 1
DO WHILE (J <= 100)
  WRITE(*,*) J
  J = J + 1
END DO
```

Pregunta 4.2 ¿Qué valor almacenará la variable Suma tras la ejecución de las siguientes instrucciones?.

```
x = 1.0
Suma = 0.0
DO WHILE (x >= 0.01)
  Suma = Suma + x
  x = x/10
END DO
```

Ejemplo 4.1 Ejemplo del funcionamiento de los bucles DO WHILE.

Con un editor de texto, edite el fichero tarea14.f90:

```
C:\EPAQ\Valvanera> EDIT tarea14.f90 
```

donde introducirá el siguiente programa en Fortran 90:

```
PROGRAM EJDOWH
!
! Instrucciones de declaracion de tipo
!
IMPLICIT NONE
INTEGER :: I
!
I = 0
DO WHILE (I <10)
  I = I + 1
  WRITE(6,*) "I =",I
END DO
WRITE(6,*) "He terminado la ejecucion del bucle Do While"
WRITE(6,*) "I =",I
!
STOP
END PROGRAM EJDOWH
```

Una vez editado el programa, compílelo y ejecútelo.

Cuando ejecute el programa, obtendrá la siguiente respuesta en la pantalla del ordenador:

```
C:\EPAQ\Valvanera>tarea14

I =          1
I =          2
I =          3
I =          4
I =          5
I =          6
I =          7
I =          8
I =          9
I =         10
He terminado la ejecucion del bucle Do While
I =         10

Program completed
Press Enter to Continue.
```

Compruebe que los valores impresos en la pantalla corresponden a los esperados de aplicar las reglas de ejecución de los bucles **DO WHILE**.

En un bucle **DO WHILE**, la condición que controla el número de veces que se repiten las instrucciones del cuerpo de bucle DO tiene que ser falsa en alguna iteración. En caso contrario, las instrucciones del bucle DO se repetirían infinitamente. Así, un programa como el siguiente

```
PROGRAM Infinito

IMPLICIT NONE
INTEGER :: i

i = 1
DO WHILE (i>0)
  i = i +1
  WRITE(*,*) "i = ", i
END DO

STOP
END PROGRAM Infinito
```

se ejecutaría indefinidamente ya que la condición que controla la ejecución del bucle se cumplirá siempre. Este tipo de comportamientos no es deseable y es un ejemplo de un *error de ejecución*, es decir, el programa no tiene errores de sintaxis pero los resultados no son los esperados. No obstante, el programa se parará cuando el valor de la variable *i* exceda el máximo valor admitido en la memoria del ordenador.



Errores de ejecución. En un bucle **DO WHILE** el valor de expresión lógica, que controla el número de veces que se repiten las instrucciones del cuerpo de bucle DO, tiene que ser falsa en alguna iteración. En caso contrario, las instrucciones del bucle DO se repetirán indefinidamente.

4.2.1. Estructuras anidadas

Un bucle DO WHILE puede definirse dentro de otros bucles DO WHILE , DO iterativos o bloques IF-END IF. Las instrucciones que lo componen tienen que estar comprendidas en el rango del bucle o bloque que lo contiene. Por ejemplo, este fragmento de programa es incorrecto

```

IF (J < K*L) THEN
  ...
  DO WHILE (I < 10)
    I = I + 1
  ...
  END IF
  ...
END DO

```

ya que el bloque IF debe contener al bucle DO WHILE. Si se compilara este programa, aparecería un error de sintaxis. Su codificación correcta es la siguiente:

```

IF (J < K*L) THEN
  ...
  DO WHILE (I < 10)
    I = I + 1
  ...
  END DO
END IF

```

Pregunta 4.3 *Identifique los errores de sintaxis y ejecución de los siguientes bucles DO WHILE.*

- a. **DO WHILE** (I <= 5)
- b. **DO WHILE** (I <= 5)
WRITE(*,*) I
END DO
- c. **DO WHILE** (x >= y)
IF (x > 0) **THEN**
 z = **SQRT**(x)
END DO
- d. **READ**(*,*) X
DO WHILE (x*x > 0.0)
 READ(*,*) x
 WRITE(*,*) x*y
END DO

Ejercicio 4.3 Serie de Fibonacci

Escriba un programa que calcule los primeros veinte términos de la serie de Fibonacci.

Los términos de la serie de Fibonacci

1, 1, 2, 3, 5, 8, 13, 21, ...

vienen dados por la expresión

$$x_1 = 1, x_2 = 1, x_3 = 2, \dots, x_j = x_{j-1} + x_{j-2}$$

Ejercicio 4.4 Serie de Fibonacci

Escriba un programa que calcule los términos de la serie de Fibonacci cuyo valor no exceda a 10000.

4.3. Bucles DO iterativos

Una de las utilidades más importantes de un ordenador es su capacidad de hacer cálculos a alta velocidad. Esto es especialmente importante si se han de ejecutar las mismas operaciones un gran número de veces. En Fortran esta tarea se puede realizar con los bucles DO WHILE ya descritos, en los que el número de iteraciones se determina utilizando una expresión lógica, o utilizando los bucles DO iterativos en los que se cuentan el número de iteraciones que han de realizarse.

Estos últimos responden a una estructura algorítmica que puede expresarse como

```
Desde que índice = n1 hasta que índice = n2 con un incremento n3 repite
  Instrucción 1
  Instrucción 2
  .....
  Instrucción k
Fin del desde que
```

La sintaxis de esta construcción es

```
DO indice = valor_inicial , valor_final , incremento
  instruccion_1
  [instruccion_2]
  ...
END DO
```

donde indice es el nombre de una variable de tipo **INTEGER**, y valor inicial , valor final e incremento son constantes, nombres de variables, o expresiones aritméticas enteras. Especificar incremento no es obligatorio ya que, si no se incluye, el valor 1 se asigna automáticamente.

Las instrucciones ejecutables instruccion_1, instruccion_2 ,... comprendidas entre **DO** y **END DO** constituyen el cuerpo del bucle DO en esta construcción.

Un ejemplo de esta estructura es

```

DO i = 1, 101, 1
  r = (i-1)*0.1
  Pr = 4.0*r*r*EXP(-2.0*r)
  WRITE(*,*) r, Pr
END DO

```

El bucle DO se ejecuta siguiendo la siguiente pauta:

- i Se evalúan los valores de `valor_inicial`, `valor_final` e incremento.
- ii Se guarda `valor_inicial` en la variable índice.
- iii Se calcula el número máximo de iteraciones NT utilizando la expresión siguiente:

$$NT = \text{INT}((\text{valor_final} - \text{valor_inicial}) / \text{incremento}) + 1$$

Si $NT < 0$, entonces se asigna $NT = 0$.

- iv El valor calculado del número máximo de iteraciones NT se guarda en un contador de iteraciones.
- v El valor del contador de iteraciones se compara con cero.
 - v.2 Si el contador es cero, continúa la ejecución del programa en la instrucción que sigue a **END DO** (paso ix de este esquema).
El contador será inicialmente cero si: `valor_inicial > valor_final` con un incremento > 0 ó `valor_inicial < valor_final` con un incremento < 0 .
 - v.2 Si el valor de contador es mayor que cero, el programa continúa en el paso vi.
- vi Se ejecutan las instrucciones comprendidas entre **DO** y **END DO**.
- vii Se actualiza el valor de la variable índice (`índice = índice + incremento`).
- viii Se resta 1 al contador de iteraciones del bucle DO. Vuelve al paso v.
- ix La ejecución del bucle DO ha terminado. El procesador continúa la ejecución en la primera sentencia que sigue a **END DO**.

A continuación, se muestran algunos ejemplos para aclarar el funcionamiento de los bucles DO iterativos. Considere el siguiente fragmento de programa

```

DO i = 1, 10
  instruccion_1
  ...
  instruccion_n
END DO

```


Dado que no se especifica el incremento, se asigna el valor +1 automáticamente. En este caso, el contador del bucle DO almacenará el valor inicial de 10, y por lo tanto, las instrucciones del cuerpo del bucle se ejecutarán 10 veces. En la primera iteración, i es igual a 1, en la segunda toma el valor 2, y así sucesivamente. En la última iteración, el valor de i es 10 y, tras terminar la ejecución, i vale 11.

Como segundo ejemplo considere

```
DO i = 1, 10, 2
  instruccion_1
  ...
  instruccion_n
END DO
```

El valor de incremento es +2. Las instrucciones del cuerpo del bucle se evalúan 5 veces. En la primera iteración i es igual a 1, en la segunda toma el valor 3, y así sucesivamente. En la última iteración el valor de i es 9 y, tras terminar la ejecución, i vale 11.

En el siguiente fragmento:

```
DO i = 1, 10, -1
  instruccion_1
  ...
  instruccion_n
END DO
```

las instrucciones del bucle no se ejecutarán ya que como $1 < 10$ y el valor de incremento es -1 . En este caso, el contador del bucle DO será cero, e i tomará el valor 1.

Finalmente, considere un último ejemplo:

```
DO i = 3, -3, -2
  instruccion_1
  ...
  instruccion_n
END DO
```

El valor de incremento es -2 . Las instrucciones del cuerpo del bucle se evaluarán 4 veces. En la primera iteración, i es igual a 3; en la segunda, i toma el valor 1; en la tercera, i toma el valor -1 y en la última iteración, el valor de i es -3 . Tras completarse la ejecución del bucle, i vale -5 .

Pregunta 4.4 ¿Cuántas veces se repetirán las instrucciones del cuerpo del bucle DO iterativo?

- (a) DO $i = 7, 10$ (b) DO $j = 1, 10, -1$
 (c) DO $i = 1, 10, 10$ (d) DO $j = -2, 10, 2$
 (e) DO $i = -2, -10, -1$ (f) DO $j = -10, -7, -3$

Pregunta 4.5 Indique el valor de la variable *iter* después de la ejecución del bucle DO.

```
iter = 0
DO indice = 1, 10, 1
  iter = iter + 1
END DO
```

Pregunta 4.6 Indique el valor de la variable *iter* después de la ejecución del bucle DO.

```
iter = 0
DO indice = 1, 10, 1
  iter = iter + indice
END DO
```

Sintaxis de los bucles DO iterativos

```
DO indice = valor inicial , valor final , incremento
instruccion_1
[instruccion_2]
...
END DO
```

donde *indice* es el nombre de una variable de tipo **INTEGER**, y *valor inicial* , *valor final* e *incremento* son constantes, nombres de variables, o expresiones aritméticas enteras. *incremento* no es obligatorio. Si no se especifica, se asigna automáticamente el valor 1. El valor de la variable *indice* no se puede modificar dentro del cuerpo del bucle DO iterativo.

Ejemplo,

```
DO indice = 1,3,1
i = i + 1
END DO
```

Ejemplo 4.2 Funcionamiento de los bucles DO iterativos (I).

Con un editor de texto, edite el fichero *tarea15.f90*:

```
C:\EPAQ\Valvanera> EDIT tarea15.f90 
```

donde introducirá el siguiente programa en Fortran 90:

```
PROGRAM Ejdo1
!
! Instrucciones de declaracion de tipo
!
IMPLICIT NONE
INTEGER :: i, n, iteraciones
!
! Programa para mostrar el funcionamiento de los ciclos DO.
!
n = 2
!
iteraciones = 0
WRITE(*,*) "Ejecucion del bucle DO – 1"
```

```

DO i = 1, 3 * n - 3
  iteraciones = iteraciones + 1
  WRITE(*,*) " Numero de iteracion: ", iteraciones, " i = ", i
END DO
WRITE(*,*) "He terminado la ejecucion del bucle DO - 1"
WRITE(*,*) " Numero total de iteraciones: ", iteraciones
WRITE(*,*) " i = ", i

STOP
END PROGRAM Ejdo1

```

Una vez editado el programa, compílelo y ejecútelo. Compare los valores obtenidos con los que esperaba obtener a partir de la aplicación de las reglas de ejecución de los bucles DO iterativos.

Ejemplo 4.3 Funcionamiento de los bucles DO iterativos (II).

Con un editor de texto, edite el fichero tarea16.f90:

```
C:\EPAQ\Valvanera> EDIT tarea16.f90 
```

donde introducirá el siguiente programa en Fortran 90:

```

PROGRAM Ejdo2
!
!  Instrucciones de declaracion de tipo
!
  IMPLICIT NONE
  INTEGER :: i, n, iteraciones
!
!  Programa para mostrar el funcionamiento de los ciclos DO.
!
  n = 2
!
  iteraciones = 0
  WRITE(*,*) "Ejecucion del bucle DO - 2"
  DO i = n, 2*n, 2
    iteraciones = iteraciones + 1
    WRITE(*,*) " Numero de iteracion: ", iteraciones, " i = ", i
  END DO
  WRITE(*,*) "He terminado la ejecucion del bucle DO - 2"
  WRITE(*,*) " Numero total de iteraciones: ", iteraciones
  WRITE(*,*) " i = ", i

STOP
END PROGRAM Ejdo2

```

Una vez editado el programa, compílelo y ejecútelo. Compare los valores obtenidos con los que esperaba obtener a partir de la aplicación de las reglas de ejecución de los bucles DO iterativos.

Ejemplo 4.4 *Funcionamiento de los bucles DO iterativos (III).*

Con un editor de texto, edite el fichero tarea17.f90:

```
C:\EPAQ\Valvanera> EDIT tarea17.f90 
```

donde introducirá el siguiente programa en Fortran 90:

```
PROGRAM Ejdo3
!
! Instrucciones de declaracion de tipo
!
IMPLICIT NONE
INTEGER :: i, n, iteraciones
!
! Programa para mostrar el funcionamiento de los ciclos DO.
!
n = 2
!
iteraciones = 0
WRITE(*,*) "Ejecucion del bucle DO – 3"
DO i = 3*n, 4 * n, -1 ! El incremento es -1
    iteraciones = iteraciones +1
    WRITE(*,*) " Numero de iteracion: ", iteraciones, " i = ", i
END DO
WRITE(*,*) "He terminado la ejecucion del bucle DO – 3"
WRITE(*,*) " Numero total de iteraciones: ", iteraciones
WRITE(*,*) " i = ", i

STOP
END PROGRAM Ejdo3
```

Una vez editado el programa, compílelo y ejecútelo. Compare los valores obtenidos con los que esperaba obtener a partir de la aplicación de las reglas de ejecución de los bucles DO iterativos.

Ejemplo 4.5 *Funcionamiento de los bucles DO iterativos (IV).*

Con un editor de texto, edite el fichero tarea18.f90:

```
C:\EPAQ\Valvanera> EDIT tarea18.f90 
```

donde introducirá el siguiente programa en Fortran 90:

```
PROGRAM Ejdo4
!
! Instrucciones de declaracion de tipo
!
IMPLICIT NONE
INTEGER :: i, n, iteraciones
!
```

```

! Programa para mostrar el funcionamiento de los ciclos DO.
!
n = 2
!
iteraciones = 0
WRITE(*,*) "Ejecucion del bucle DO - 4"
DO i = n, n, -1 ! El incremento es -1
  iteraciones = iteraciones + 1
  WRITE(*,*) " Numero de iteracion: ", iteraciones, " i = ", i
END DO
WRITE(*,*) "He terminado la ejecucion del bucle DO - 4"
WRITE(*,*) " Numero total de iteraciones: ", iteraciones
WRITE(*,*) " i = ", i

STOP
END PROGRAM Ejdo4

```

Una vez editado el programa, compílo y ejecútelo. Compare los valores obtenidos con los que esperaba obtener a partir de la aplicación de las reglas de ejecución de los bucles DO iterativos.

Es importante tener en cuenta que la variable índice no se puede modificar dentro del cuerpo del bucle DO iterativo. Así, un programa como el siguiente

```

PROGRAM error1

IMPLICIT NONE
INTEGER :: i
DO i = 1, 10, 1
  i = i + 1
  WRITE(*,*) " i = ", i
END DO

STOP
END PROGRAM error1

```

generará un error de sintaxis en la compilación del programa, y aparecerá un mensaje de error:

```

Line 6, file error1.f90
      i = i + 1
      |
FATAL -- DO variable must not become undefined or redefined
        while DO construct is active

```



Errores de sintaxis. El valor de la variable índice no se puede modificar dentro del cuerpo del bucle DO iterativo.

Ejemplo 4.6 *Cálculo del factorial de un número.*

El factorial de un número se define como

$$N! = \begin{cases} 1 & \text{si } N = 0, \\ N * (N - 1) * (N - 2) * \dots * 3 * 2 * 1 & \text{si } N > 0 \end{cases}$$

El programa incluirá los siguientes pasos:

- i Leer el número n
- ii Calcular el factorial $n!$
- iii Escribir el resultado

Este algoritmo se puede detallar en fragmentos menores:

- i Pedir al usuario un numero entero positivo
- ii Leer n
- iii Comprobar si $n > 0$
- iv Escribir n
- v Asignar 1 a la variable factorial , factorial $\leftarrow 1$
- vi Repetir desde $i = n$ hasta 2 con un incremento -1
factorial \leftarrow factorial * i
- vii Escribir el resultado: factorial

Con un editor de texto, edite el fichero tarea19.f90:

```
C:\EPAQ\Valvanera> EDIT tarea19.f90 
```

donde introducirá el siguiente programa en Fortran 90:

```
PROGRAM factorialn
!
! Este programa calcula el factorial de un entero positivo
! que se define como:
!
!      _
!      | 1                si n = 0
!      | n*n(-1)*(n-2)*...*2*1 si n > 0
!      | _
!
! Declaracion de variables
!
```

```

IMPLICIT NONE
INTEGER :: i, n, factorial
!
! Pide al usuario un numero entero positivo n
!
WRITE (*,*) 'Introduzca un entero positivo: '
READ (*,*) n
IF (n < 0) THEN
  WRITE(* ,*) 'n debe ser un entero positivo'
  STOP
END IF
!
! Calculo de n!
!
  factorial = 1
  DO i = n, 2, -1
    factorial = factorial * i
  END DO
!
! Escribe el valor de n!
!
  WRITE (*,*) 'El factorial de ', n, ' es ', factorial
!
STOP
END PROGRAM factorialn

```

Una vez editado el programa, compílelo y ejecútelo. Compruebe los resultados obtenidos.

Ejemplo 4.7 *Cálculo del sumatorio de una serie de datos.*

Escriba un programa que lea n datos y calcule su suma,

$$\text{sumatorio} = \sum_{i=1}^n x_i = x_1 + x_2 + \dots + x_{n-1} + x_n$$

El programa incluirá los siguientes pasos:

- i Leer el numero de datos n
- ii Leer los datos
- iii Calcular el sumatorio
- iv Escribir el resultado

El cálculo del sumatorio puede escribirse utilizando un bucle DO iterativo

```

sumatorio = 0
DO i = 1, n, 1
  READ(* ,*) x
  sumatorio = sumatorio + x
END DO

```

Con un editor de texto, edite el fichero tarea20.f90:

```
C:\EPAQ\Valvanera> EDIT tarea20.f90 
```

donde introducirá el siguiente programa en Fortran 90:

```
PROGRAM suma_n
!
! Este programa calcula la suma de n datos:
!
!      sumatorio = \sum_{i=1}^n x_i = x_1 + x_2 + ... + x_n
!
! Declaracion de variables
!
IMPLICIT NONE
INTEGER :: i, n
REAL :: x, sumatorio
!
!
WRITE (*,*) 'Introduzca el numero de datos '
READ (*,*) n
IF (n < 0) THEN
    WRITE(*,*) 'n debe ser un entero positivo'
    STOP
END IF
!
! Calculo de sumatorio
!
WRITE (*,*) 'Introduzca los datos (uno por linea) '
sumatorio = 0
DO i = 1, n, 1
    READ(*,*) x
    sumatorio = sumatorio + x
END DO
!
! Escribe el valor de sumatorio
!
WRITE (*,*) 'La suma de los ',n, ' datos es ', sumatorio
!
STOP
END PROGRAM suma_n
```

Compile y ejecute el programa. Compruebe si los resultados son los esperados.

4.3.1. Cálculos de tablas de valores

Una de las utilidades del bucle DO iterativo es que permite repetir un cálculo con distintos valores de una variable, incrementándose el valor de ésta en cada repetición. En los siguientes ejemplos se mostrarán dos casos prácticos muy frecuentes.

Caso I. Se conoce valor inicial, valor final y número total de valores de la tabla

Ejemplo 4.8 *Escriba un programa que calcule y escriba los valores de la temperatura, conocidas la temperatura inicial (Ti), la temperatura final (Tf) y el número total de valores de la tabla (nt).*

El programa incluirá los siguientes pasos:

- i Leer el Ti, Tf, nt
- ii Calcular la temperatura T
- iii Escribir el resultado

Como la variable T es de tipo REAL no se puede utilizar como índice del bucle DO iterativo. Por lo tanto, es necesario calcular cuánto varía T en cada iteración. Este incremento de T se llamará IncT. Para calcular el incremento se utilizará la fórmula

$$\text{incremento} = \frac{\text{valor final} - \text{valor inicial}}{\text{Numero datos} - 1,0}$$

Por ejemplo, si se quiere calcular una tabla de 10 valores de una variable en el intervalo 1,0 a 10,0, el incremento de la variable en cada iteración será

$$\text{incremento} = \frac{10,0 - 1,0}{10,0 - 1,0} = \frac{9,0}{9,0} = 1,0$$

La expresión en Fortran del cálculo del incremento IncT se escribe

$$\text{IncT} = (\text{Tf} - \text{Ti}) / \text{REAL}(\text{nt} - 1)$$

donde se introduce la función intrínseca **REAL** (que transforma un número entero en un número real) para evitar los efectos de mezclar valores de tipo INTEGER y tipo REAL.

Un programa que realiza la tarea propuesta es

```

PROGRAM temperatura0
!
!
! Este programa calcula una tabla de temperaturas conocidas
! la temperatura inicial (Ti), la temperatura final (Tf) y
! el numero de datos de la tabla (nt)
!
! t ... temperatura
! IncT ... incremento de la temperatura despues de cada iteracion
!
!
IMPLICIT NONE
INTEGER :: i, nt
REAL :: T, Ti, Tf, IncT
!
WRITE(* ,*) " Introduzca la temperatura inicial y final "
```

```

READ(*,*) Ti, Tf
WRITE(*,*) " Introduzca el numero de datos de la tabla "
READ(*,*) nt
!
IncT = (Tf - Ti)/REAL(nt - 1)
WRITE(*,*) "IncT = ",IncT

DO i = 1, nt, 1
    T = Ti + (i-1)*IncT
    WRITE(*,*) "I = ", i, " T = ", T
END DO

STOP
END PROGRAM temperatura0

```

Para comprobar el funcionamiento del programa, edite el fichero tarea21.f90:

```
C:\EPAQ\Valvanera> EDIT tarea21.f90 
```

donde introducirá el anterior programa en Fortran 90.

Compile y ejecute el programa. ¿Qué resultado se obtiene si se utilizan los valores de temperatura (K) $T_i = 200$, $T_f = 300$ y $nt = 10$?.

Caso II. Se conoce valor inicial, valor final y diferencia entre valores consecutivos de la tabla

Ejemplo 4.9 *Escriba un programa que calcule los valores de la temperatura, conocidas la temperatura inicial (T_i), la temperatura final (T_f) y la diferencia entre dos valores consecutivos en la tabla ($IncT$).*

El programa incluirá los siguientes pasos:

- i Leer T_i , T_f , $IncT$
- ii Calcular la temperatura
- iii Escribir el resultado

Como la variable T es de tipo **REAL** no se puede utilizar como índice del bucle **DO** iterativo. Por lo tanto, es necesario calcular el número de iteraciones (nt) necesarias para completar la tabla de temperaturas. Este número se calcula utilizando la fórmula

$$\text{Numero datos} = \frac{\text{valor final} - \text{valor inicial}}{\text{incremento}} + 1$$

Por ejemplo, si se quiere calcular una tabla de valores de una variable en el intervalo 1,0 a 10,0, con un incremento de 1,0, el número de datos será

$$\text{Numero datos} = \frac{10,0 - 1,0}{1,0} + 1 = 10$$

La expresión en Fortran del cálculo del número de datos nt se escribe

$$nt = \text{INT}((Tf - Ti) / \text{IncT}) + 1$$

donde se introduce la función intrínseca **INT** (que transforma un número real en un número entero) para evitar los efectos de mezclar valores de tipo INTEGER y tipo REAL.

Un programa que realiza la tarea propuesta es

```

PROGRAM temperatura1
!
!
! Este programa calcula una tabla de temperaturas conocidas
! la temperatura inicial (Ti), la temperatura final (Tf) y
! la diferencia entre dos valores consecutivos de la tabla (IncT)
!
! T ... temperatura
! nt ... numero de datos de la tabla
!
!
! IMPLICIT NONE
! INTEGER :: i, nt
! REAL :: T, Ti, Tf, IncT
!
! WRITE(*,*) " Introduzca la temperatura inicial y final "
! READ(*,*) Ti, Tf
! WRITE(*,*) " Introduzca el incremento de temperatura "
! READ(*,*) IncT
!
! nt = INT((Tf-Ti)/IncT) + 1
! WRITE(*,*) "nt = ", nt
!
! DO i = 1, nt, 1
!   T = Ti + (i-1)*IncT
!   WRITE(*,*) "I = ", i, " T = ", T
! END DO
!
! STOP
! END PROGRAM temperatura1

```

Para comprobar el funcionamiento del programa, edite el fichero tarea22.f90:

```
C:\EPAQ\Valvanera> EDIT tarea22.f90 
```

donde introducirá el anterior programa en Fortran 90.

Compile y ejecute el programa. ¿Cuántos datos tiene la tabla si utiliza los valores de temperatura (K) $T_i = 200.0$, $T_f = 300.0$ e $\text{IncT} = 10.0$?

Ejercicio 4.5 Estudio de una cinética de primer orden.

Escriba un programa que proporcione las fracciones de concentración del reactivo A, $|A|/|A|_0$, frente al tiempo para un reactivo que sigue una cinética de primer orden. El programa debe pedir al usuario el valor de la constante cinética k con unidades de s^{-1} , el valor final del tiempo t_{final} en s, y el número de valores calculados (nt).

En una reacción que sigue una cinética de primer orden con una estequiometría



la concentración del reactivo A varía de acuerdo con la expresión:

$$|A| = |A|_0 e^{-kt}$$

donde $|A|_0$ es la concentración inicial del reactivo A a $t = 0$, k es la constante cinética con unidades de tiempo^{-1} y t es el tiempo transcurrido desde el inicio de la reacción ($t > 0$).

Para comprobar el funcionamiento del programa utilice los valores $k = 10 \text{ s}^{-1}$, $t_{\text{final}} = 30 \text{ s}$ y $nt = 20$.

4.4. Lectura y escritura de datos con ficheros (sin formato)

Según lo aprendido en las prácticas anteriores, para proporcionar datos a un programa se emplea la instrucción **READ**(*,*) y se teclean los datos en el teclado, y para conocer los resultados de la ejecución de un programa se emplea la instrucción **WRITE**(*,*) y se obtienen los datos en la pantalla. En esta sección de la práctica, utilizando las instrucciones **READ** y **WRITE**, se introducirá la lectura de datos desde ficheros y la escritura de datos en ficheros.

Para leer datos de un fichero se utiliza la instrucción:

READ(u,f) lista de variables

mientras que para escribir datos en un fichero se utiliza la instrucción:

WRITE(u,f) lista de variables y/o constantes

En estas instrucciones, las letras u y f corresponden a la unidad lógica y al formato, respectivamente.

- u hace referencia a la unidad lógica. Es un número entero que identifica el fichero o periférico (teclado, pantalla) desde donde se reciben los datos de entrada (**READ**) o hacia donde se envían los resultados de salida (**WRITE**). Cuando los datos se introducen con el teclado, $u = 5$ y se puede sustituir por *. Análogamente, si se quieren escribir los resultados en la pantalla, $u = 6$ y se puede sustituir por *
- f hace referencia al formato o plantilla de lectura/escritura. Puede ser una plantilla de formato o la etiqueta de una instrucción **FORMAT**. En estas prácticas se utilizará siempre el *formato libre*, para el que la plantilla de formato es *.

Ejemplos,

READ(10,*) x,y

WRITE(12,*) "tiempo(s) ",t," concentración", a0***EXP**(-k*t)

Con excepción de los casos $u=5$ (teclado) y $u=6$ (pantalla), que están preasignados, antes de poder utilizar una unidad lógica hay que realizar una asignación de la unidad mediante la instrucción **OPEN**. La sintaxis de esta instrucción es:

OPEN (UNIT = xx, FILE = "filename")

- **UNIT** = xx, donde xx es un número entero arbitrario (distinto de 5 y 6) con el que se hará referencia en el programa a una determinada unidad lógica. Este número sustituirá a u en la instrucción **READ(u,*)** o **WRITE(u,*)**.
- **FILE** = "filename". En este caso, "filename" contiene el nombre del fichero en el que se van a leer o escribir los datos. La instrucción **OPEN** asocia la unidad u con el fichero.

Ejemplos del uso de esta instrucción son:

OPEN(UNIT=10, FILE="espectro.dat")
OPEN(UNIT=12, FILE="resultados.dat")

Para desconectar (disociar) una unidad lógica se emplea la instrucción **CLOSE** que tiene la sintaxis

CLOSE (UNIT = xx)

Ejemplos,

CLOSE(UNIT = 10)
CLOSE(UNIT = 12)

La instrucción **CLOSE** debe emplearse:

- Al final de la ejecución de un programa (puesto que el fichero no va a emplearse de nuevo).
- Cuando el fichero va a ser empleado de nuevo pero cambiando algunos de los parámetros de la instrucción **OPEN**.
- Cuando la unidad quiere emplearse para otro fichero.
- Cuando se desea asociar un fichero a una unidad diferente.

Una vez que se ha desconectado una unidad lógica, no se puede volver a utilizar dentro del programa si no se define de nuevo con una instrucción **OPEN**.

Sintaxis de las instrucciones READ y WRITE

READ(u,f) lista de variables

WRITE(u,f) lista de variables y/o constantes

- u hace referencia a la unidad lógica. Es un número entero que identifica el fichero o periférico (teclado, pantalla) desde donde se reciben los datos de entrada (**READ**) o hacia donde se envían los resultados de salida (**WRITE**). Cuando los datos se introducen con el teclado, u = 5 y se puede sustituir por *. Análogamente, si se quieren escribir los resultados en la pantalla, u = 6 y se puede sustituir por *
- f hace referencia al formato o plantilla de lectura/escritura. Puede ser una plantilla de formato o la etiqueta de una instrucción **FORMAT**. En estas prácticas se utilizará siempre el *formato libre*, para el que la plantilla de formato es *.

Sintaxis de la instrucción OPEN

OPEN (**UNIT** = xx, **FILE** = "filename")

- **UNIT** = xx, donde xx es un número entero arbitrario (distinto de 5 y 6) con el que se hará referencia en el programa a una determinada unidad lógica. Este número sustituirá a u en la instrucción **READ**(u,*) o **WRITE**(u,*).
- **FILE** = 'filename'. En este caso, 'filename' contiene el nombre del fichero en el que se van a leer o escribir los datos. La instrucción **OPEN** asocia la unidad u con el fichero.

Ejemplo,

OPEN(**UNIT**=12, **FILE**="parametros.dat")

Sintaxis de la instrucción CLOSE

CLOSE (**UNIT** = xx)

- **UNIT** = xx, donde xx es un número entero con el que se hace referencia a la unidad lógica que se quiere disociar.


Ejemplo,

CLOSE(**UNIT**=12)

Ejemplo 4.10 *Lectura de datos de un fichero y escritura de datos en un fichero.*

Este programa muestra el funcionamiento de las instrucciones **OPEN** y **CLOSE** y cómo se pueden utilizar para leer datos escritos en un fichero (INP.DAT) y para escribir datos en otro fichero (OUT.DAT).

1. Edite el fichero INP.DAT.

```
C:\EPAQ\Valvanera> EDIT inp.dat 
```

Escriba en este fichero los siguientes valores


```
1.0E+23 4
```

2. Escriba el siguiente programa en Fortran 90 en el fichero tarea23.f90.

PROGRAM OPCLOS

```
!
! Se muestra el funcionamiento de las instrucciones OPEN y CLOSE
! para leer datos de un fichero y escribir datos en un fichero.
!
! Instrucciones de declaracion de tipo
!
IMPLICIT NONE
INTEGER :: IDAT
REAL :: RDAT
!
OPEN(4, FILE = "inp.dat")
OPEN(7, FILE = "out.dat")
!
READ(4,*) RDAT, IDAT
WRITE(7,*) 'Los datos son'
WRITE(7,*) RDAT, IDAT
WRITE(7,*) 'He terminado'
!
CLOSE(4)
CLOSE(7)
!
STOP
END PROGRAM OPCLOS
```

3. Compile y ejecute el programa. Compruebe el contenido del fichero de salida OUT.DAT. Esto se puede hacer utilizando la instrucción del MS-DOS TYPE:

```
C:\EPAQ\Valvanera> TYPE out.dat 
```

La ejecución de esta instrucción produce la siguiente respuesta en la pantalla del ordenador:

```
C:\EPAQ\Valvanera>type out.dat
Los datos son
    0.100000E+24          4
He terminado

C:\EPAQ\Valvanera>_
```

El proceso de lectura

Por defecto, en Fortran 90, los datos de un fichero se tratan como si estuvieran escritos en una cinta magnética (fichero de acceso secuencial): cada línea es tratada como un registro dentro de la cinta magnética.

Cuando se utiliza formato libre en la lectura de datos, al ejecutar la instrucción **READ**(u,*) el procesador realiza las siguientes operaciones:

- i Abre el fichero asociado a la unidad u correspondiente.
- ii Si es la primera vez que en el programa se leen datos del fichero, la lectura se inicia en la primera línea del fichero. Si no es la primera vez, la lectura de datos comienza en la línea siguiente a la que se utilizó en la última instrucción de lectura.
- iii Se leen los datos de la línea del fichero. Si hay menos datos que número de variables en la instrucción de lectura, la lectura continúa en la línea siguiente del fichero. Si, por el contrario, en la línea del fichero se encuentran más datos que los necesarios, el procesador ignora los datos sobrantes.
- iv Cierra el fichero.

Ejercicio 4.6 *La ecuación de Clausius-Clapeyron.*

Escriba un programa que sirva para evaluar los valores de P_v en un intervalo de temperaturas T_i a T_f conocidas P_0 , T_0 y ΔH_v y el incremento de temperaturas, ΔT .

Utilice ficheros para introducir los parámetros del programa y recoger los resultados de la ejecución.

Utilizando el programa que ha escrito, calcule la presión de vapor del agua en el intervalo 21° C a 30° C, con un incremento de temperatura de 0.5° C. $R = 8.314 \text{ J mol}^{-1} \text{ K}^{-1}$.

La ecuación de Clausius-Clapeyron predice que la dependencia de la presión de vapor con la temperatura está dada por

$$\ln \frac{P_1}{P_0} = -\frac{\Delta H_v}{R} \left(\frac{1}{T_1} - \frac{1}{T_0} \right)$$

La entalpía de vaporización del agua es prácticamente constante en el intervalo 20 °C a 30 °C ($\Delta H_v = 43.9 \text{ kJ.mol}^{-1}$). A 20 °C (T_0), $P = 17.535 \text{ Torr}$ (P_0).

Ejercicio 4.7 *La ley de Arrhenius.*

Escriba un programa que calcule, utilizando la ley de Arrhenius, la constante cinética k en un intervalo de temperaturas de T_i a T_f , conocidos el factor preexponencial A , la energía de activación E_a , y el incremento de temperaturas ΔT .

Utilice ficheros para introducir los parámetros del programa y recoger los resultados de la ejecución.

La ley de Arrhenius permite calcular la constante de velocidad de una reacción a distintas temperaturas,

$$k = A \exp \left(-\frac{E_a}{R T} \right)$$

Utilizando el programa, calcule k en el intervalo de 200 a 300 K con un incremento de 10 K para la reacción de isomerización isocianuro \rightleftharpoons acetonitrilo.

Esta reacción tiene una energía de activación, $E_a = 1,601 \text{ kJ}\cdot\text{mol}^{-1}$ y un factor preexponencial $A = 3,98 \cdot 10^{-13} \text{ s}^{-1}$.

Ejercicio 4.8 *Cálculo de la media y desviación típica muestrales.*

Escriba un programa que lea n datos de un fichero y calcule su media y su varianza muestral.

Utilice ficheros para introducir los parámetros del programa y recoger los resultados de la ejecución.

La media muestral \bar{x} viene dada por

$$\bar{x} = \frac{\sum_{i=1}^n x_i}{n}$$

La desviación típica muestral puede calcularse utilizando la expresión

$$s(x) = \sqrt{\frac{n \sum_{i=1}^n x_i^2 - (\sum_{i=1}^n x_i)^2}{n(n-1)}}$$

En una serie de medidas del índice de refracción del acetato de etilo puro se obtuvieron los resultados

1.3546	1.3535	1.3547	1.3538	1.3539
1.3540	1.3542	1.3543	1.3536	1.3542

Utilizando el programa calcule su media y varianza muestral.

Práctica 5

Instrucciones de control de flujo (II)

CONTENIDOS

5.1. Ejercicios de repaso	88
5.2. Bucles DO anidados	89
5.3. Transferencia de control en bucles DO: las instrucciones CYCLE y EXIT	93
5.3.1. La instrucción CYCLE	93
5.3.2. La instrucción EXIT	95
5.3.3. Bucles anidados y las sentencias CYCLE y EXIT	97
5.4. Bucles DO y bloques IF con nombre	103

5.1. Ejercicios de repaso

Ejercicio 5.1 Presión de vapor de mezclas binarias. Ley de Raoult.

Escriba un programa que calcule la presión de vapor de una mezcla binaria utilizando la ley de Raoult. El programa debe leer las presiones de vapor de las sustancias puras P_1^0 y P_2^0 , y el número de valores que desee que tenga la tabla. Como resultado de la ejecución del programa se espera obtener una tabla con las presiones parciales P_1 y P_2 de cada componente, la presión de vapor total P_T y la composición en fracción molar X_1 en el intervalo $X_1 = 0 - 1$.

Para una disolución ideal de dos componentes, se sabe que las presiones parciales de cada componente, P_1 y P_2 , están dadas por la ley de Raoult

$$P_1 = P_1^0 X_1$$

$$P_2 = P_2^0 X_2$$

donde P_i^0 es la presión de vapor del componente i puro y X_i es la fracción molar del componente i en la disolución.

La presión total se calcula como la suma de las presiones parciales:

$$P_t = P_1 + P_2$$

Compruebe el funcionamiento del programa calculando las presiones parciales de mezclas de benceno y tolueno a 20°C. $P^0(\text{benceno}, 20^\circ\text{C}) = 74,7$ Torr y $P^0(\text{tolueno}, 20^\circ\text{C}) = 22,3$ Torr

Ejercicio 5.2 Descenso crioscópico.

Escriba un programa que calcule el descenso crioscópico de un disolvente en un intervalo de concentraciones de soluto.

El descenso crioscópico, ΔT_f , viene dado por la expresión

$$\Delta T_f = K_f \cdot m$$

donde K_f es la constante crioscópica ($\text{K mol}^{-1} \text{kg}$) y m la concentración molal (mol kg^{-1}).

Compruebe el funcionamiento del programa utilizando disoluciones de naftaleno de concentraciones comprendidas entre $1,0 \cdot 10^{-3}$ y $0,1$ m en ácido acético, benceno o agua.

soluto	disolvente	$K_f / \text{K} \cdot \text{mol}^{-1} \cdot \text{kg}$
naftaleno	ácido acético	3,90
naftaleno	benceno	5,12
naftaleno	agua	1,86

5.2. Bucles DO anidados

Es posible escribir un bucle DO (bucle DO interno) dentro de otro bloque DO (bucle DO externo). En este caso decimos que los bucles DO están anidados. Las reglas aplicables a estas estructuras son semejantes a las de los bucles DO simples. Sin embargo, hay que tener en cuenta que las variables índice deben ser independientes. Esto supone que:

- i El nombre de la variable índice del bucle interno, índice2, no puede coincidir con el nombre de la variable índice del bucle externo, índice1.
- ii Un bucle DO interno no puede solaparse con uno externo.

El siguiente fragmento de programa muestra una estructura correcta

```

DO indice1 = Valor inicial1 , Valor final1 , Incremento1
...
...
DO indice2 = Valor inicial2 , Valor final2 , Incremento2
***
END DO
...
END DO

```

mientras que la estructura siguiente

```

DO indice1 = valor_inicial1 , valor_final1 , incremento1
***
***
DO indice1 =valor_inicial2 , valor_final2 , incremento2
***
END DO
***
END DO

```

es incorrecta, pues utiliza en el bucle DO interno la misma variable índice que en el bucle DO externo.

Ejemplos de bucles correctos son,

```

DO i = 1, 3, 1
DO j = 1, 6, 2
WRITE(*,*) i, j
END DO
END DO

```

```

DO i = 1, 3, 1
DO j = i, 2*i, 1
WRITE(*,*) i, j
END DO
END DO

```

Sin embargo, la estructura

```
DO i = 1, 3, 1
  DO i = 1, 6, 2
    WRITE(*,*) i, j
  END DO
END DO
```

es incorrecta porque utiliza el mismo índice para los bucles DO interno y externo, y el bucle

```
DO i = 1, 3, 1
  DO j = i, 2*i, 1
    i = i + 1
    WRITE(*,*) i, j
  END DO
END DO
```

es incorrecto pues intenta modificar el valor del índice del bucle DO externo en el bucle DO interno.



Errores de sintaxis. Los nombres de las variables índice de los bucles DO anidados deben ser distintos, no pudiéndose modificar el valor de las variables índice de un bucle externo en un bucle interno.

Ejemplo 5.1 Errores de sintaxis con bucles DO anidados (I).

En el ejemplo siguiente se muestran dos bucles DO anidados que tienen la misma variable índice

```
PROGRAM doanidados1
IMPLICIT NONE
INTEGER :: i
DO i = 1, 3, 1
  DO i = 1, 6, 2
    WRITE(*,*) i
  END DO
END DO
STOP
END PROGRAM doanidados1
```

La compilación de este programa produce el siguiente mensaje de error:

```
Line 5, file doanidados1.f90
      DO i = 1, 6, 2
          |
FATAL -- DO variable must not become undefined or redefined while DO
construct is active (see "DO Construct" in the Essential Lahey Fortran 90
Reference).

Encountered 1 errors, 0 warning in file doanidados1.f90.
```

Ejemplo 5.2 *Errores de sintaxis con bucles DO anidados (II).*

En el ejemplo siguiente se muestran dos bucles DO anidados en el que, en el bucle DO interno, se modifica el valor de la variable índice del bucle DO externo

```
PROGRAM doanidados1
IMPLICIT NONE
INTEGER :: i, j
DO i = 1, 3, 1
  DO j = 1, 6, 2
    i = i + j
    WRITE(*,*) i,j
  END DO
END DO
STOP
END PROGRAM doanidados1
```

La compilación de este programa produce el mensaje de error:

```
Line 6, file doanidados2.f90
      i = i + j
      |
FATAL -- DO variable must not become undefined or redefined while DO
construct is active (see "DO Construct" in the Essential Lahey Fortran 90
Reference).

Encountered 1 error, 0 warnings in file doanidados2.f90.
```

Ejemplo 5.3 *Bucles DO anidados.*

En el ejemplo siguiente se muestran dos bucles DO anidados. El programa proporciona el valor de las variables índice de los bucles y su producto para cada iteración.

Con un editor de texto, edite el fichero tarea24.f90:

```
C:\EPAQ\Valvanera> EDIT tarea24.f90 
```

donde introducirá el siguiente programa en Fortran 90

```
PROGRAM doanidados1
IMPLICIT NONE
INTEGER :: i, j, pdto
DO i = 1, 3, 1
  DO j = 1, 3, 1
    pdto = i*j
    WRITE(*,*) 'i= ', i, ' j= ', j, ' i*j = ', pdto
  END DO
END DO
STOP
END PROGRAM doanidados1
```

Compile y ejecute el programa.

En este ejemplo, el bucle DO externo asigna el valor 1 a la variable índice *i* y, a continuación, comienza la ejecución del bucle interno. El bucle DO interno se ejecuta asignando los valores 1, 2, 3 a la variable índice *j*. Cuando se completa la ejecución del bucle interno, se produce una nueva iteración del bucle externo. Estas instrucciones se repiten hasta que se completa la ejecución del bucle DO externo.

Compruebe que la ejecución del programa produce los siguientes resultados:

```
i= 1   j= 1   i*j = 1
i= 1   j= 2   i*j = 2
i= 1   j= 3   i*j = 3
i= 2   j= 1   i*j = 2
i= 2   j= 2   i*j = 4
i= 2   j= 3   i*j = 6
i= 3   j= 1   i*j = 3
i= 3   j= 2   i*j = 6
i= 3   j= 3   i*j = 9
```

Cuando el compilador de Fortran encuentra una instrucción **END DO**, la asigna al bucle DO más interno que en ese momento esté sin cerrar. Esto puede dar lugar a un error si accidentalmente se omite una instrucción **END DO** en el programa. Por ejemplo, si en el programa anterior se omite la instrucción **END DO** del bucle más interno,

```
PROGRAM doanidados1
IMPLICIT NONE
INTEGER :: i, j, pdto
DO i= 1, 3, 1
  DO j= 1, 3, 1
    pdto = i*j
    WRITE(*,*) 'i= ', i, ' j= ', j, ' i*j = ', pdto
  END DO
STOP
END PROGRAM doanidados1
```

la compilación del programa devolverá la respuesta:

```
Line 10, file tarea24.f90
      END PROGRAM doanidados1
      |
FATAL -- End of program unit reached before construct is closed.

Encountered 1 error, 0 warnings in file doanidados1.f90.
```

Con este error, el compilador indica que hay un problema en la sintaxis del programa: hay un bucle DO que no está cerrado. Sin embargo, el compilador no puede localizar dónde se encuentra este bucle DO sin cerrar porque no detecta el error hasta que se alcanza la instrucción **END PROGRAM**. En un programa pequeño es fácil encontrar el error, sin embargo, en un programa complejo o de gran tamaño es difícil detectar estos errores. Este problema puede evitarse utilizando los bucles con nombre que se discutirán en la sección 5.4.



5.3. Transferencia de control en bucles DO: las instrucciones CYCLE y EXIT

Existen dos comandos adicionales, **CYCLE** y **EXIT**, que permiten controlar el funcionamiento de los bucles DO WHILE y DO iterativos.

5.3.1. La instrucción CYCLE

La instrucción **CYCLE** es útil cuando se quiere parar la ejecución de una iteración sin interrumpir la ejecución del bucle DO. Cuando se ejecuta la instrucción **CYCLE**, la ejecución de las instrucciones del cuerpo del bucle se interrumpe, el control se transfiere al inicio del bucle DO y se inicia la ejecución de una nueva iteración. La sintaxis de la instrucción es

CYCLE

Para mostrar el funcionamiento de la instrucción, considere el siguiente programa

```
PROGRAM test_cycle
IMPLICIT NONE
INTEGER :: i
DO i = 1, 5, 1
  IF (i == 3) CYCLE
  WRITE(*,*) i
END DO
WRITE(*,*) ' Fin del bucle DO '
STOP
END PROGRAM test_cycle
```

Cuando el programa se ejecuta, se obtiene en la pantalla el siguiente resultado

```
C:\EPAQ\Valvanera>tarea25
1
2
4
5
Fin del bucle DO

Program completed
Press Enter to Continue.
```

Note que cuando la variable índice, *i*, toma el valor 3, no se ejecuta la instrucción **WRITE(*,*) i**, ya que el control se transfiere al inicio del bucle y comienza una nueva iteración de éste.

Sintaxis de la instrucción CYCLE**CYCLE**

En el momento en el que se alcanza esta instrucción, la ejecución de las instrucciones del cuerpo del bucle se interrumpe y se inicia la ejecución de una nueva iteración del bucle DO.

Ejemplo,

```
DO i = 1, 5, 1
  IF (i == 3) CYCLE
  WRITE(*,*) i
END DO
```

Ejemplo 5.4 *Funcionamiento de la instrucción CYCLE con bucles DO iterativos.*

Con un editor de texto, edite el fichero tarea26.f90:

```
C:\EPAQ\Valvanera> EDIT tarea26.f90 
```

donde introducirá el siguiente programa en Fortran 90

```
PROGRAM EjCycle1

! Este programa muestra el funcionamiento de la instrucción CYCLE

IMPLICIT NONE
INTEGER :: indice

DO indice = 1, 10, 1
  WRITE(*,*) "Dentro del DO antes del IF :: indice = ", indice
  IF (indice == 6) CYCLE
  WRITE(*,*) "Dentro del DO despues del IF :: indice = ", indice
END DO
WRITE(*,*) " Fuera del DO :: indice = ", indice
WRITE(*,*)

STOP
END PROGRAM EjCycle1
```

Compile y ejecute el programa. Compruebe si la ejecución del programa produce el resultado esperado.

Ejemplo 5.5 *Funcionamiento de la instrucción CYCLE con bucles DO WHILE.*

Con un editor de texto, edite el fichero tarea27.f90:

```
C:\EPAQ\Valvanera> EDIT tarea27.f90 
```

donde introducirá el siguiente programa en Fortran 90

```
PROGRAM EjCycle2
```

! Este programa es otro ejemplo del funcionamiento de la instrucción CYCLE

```
IMPLICIT NONE
```

```
INTEGER :: indice
```

```
indice = 1
```

```
DO WHILE (indice < 10)
```

```
  WRITE(*,*) "Dentro del DO antes del IF :: indice = ", indice
```

```
  indice = indice + 1
```

```
  IF (indice == 6) CYCLE
```

```
  WRITE(*,*) "Dentro del DO :: despues del IF indice = ", indice
```

```
END DO
```

```
WRITE(*,*)
```

```
WRITE(*,*) "Fuera del DO :: indice = ", indice
```

```
STOP
```

```
END PROGRAM EjCycle2
```

Compile y ejecute el programa. Compruebe si la ejecución del programa le devuelve el resultado esperado.

5.3.2. La instrucción EXIT

Cuando una instrucción **EXIT** se ejecuta en el cuerpo de un bucle **DO**, la ejecución del bucle se interrumpe y continúa en la primera instrucción ejecutable que sigue a **END DO**. La sintaxis de la instrucción es:

EXIT

Para mostrar el funcionamiento de la instrucción, considere el siguiente programa

```
PROGRAM test_exit
IMPLICIT NONE
INTEGER :: i
DO i = 1, 5, 1
  IF (i == 3) EXIT
  WRITE(*,*) i
END DO
WRITE(*,*) ' Fin del bucle DO '
STOP
END PROGRAM test_exit
```

Cuando se ejecuta el programa, se obtiene el siguiente resultado en la pantalla

```
C:\EPAQ\Valvanera>tarea28

      1
      2
Fin del bucle DO

Program completed
Press Enter to Continue.
```

Note que el comando **EXIT** se ejecuta cuando la variable *i* toma el valor 3, y que después el control se transfiere a la primera instrucción ejecutable que sigue a **END DO**.

Sintaxis de la instrucción EXIT

EXIT

En el momento en el que se alcanza esta instrucción, la ejecución de las instrucciones del cuerpo del bucle se interrumpe y el programa continúa su ejecución en la primera instrucción que sigue a **END DO**.

Ejemplo,

```
DO i = 1, 5, 1
  IF (i == 3) EXIT
  WRITE(*,*) i
END DO
```

Ejemplo 5.6 Funcionamiento de la instrucción EXIT.

Con un editor de texto, edite el fichero tarea29.f90:

```
C:\EPAQ\Valvanera> EDIT tarea29.f90 
```

donde introducirá el siguiente programa en Fortran 90

```
PROGRAM EjExit

IMPLICIT NONE
INTEGER :: indice

DO indice = 1, 10, 1
  WRITE(*,*) 'Dentro del DO :: indice = ', indice
  IF (indice == 6) EXIT
END DO
WRITE(*,*) 'Fuera del DO :: indice = ', indice

STOP
END PROGRAM EjExit
```

Compile y ejecute el programa. Compruebe que la ejecución del programa produce los resultados esperados.

Ejemplo 5.7 Suponga que quiere escribir un programa que utiliza bucles DO para calcular la suma:

$$1 + 2 + 3 + 5 + 6 + 7 + 8 + 9 + 10$$

Un programa que realizaría esa tarea es:

```
PROGRAM EjCycle3

IMPLICIT NONE
INTEGER :: I,ISUM

ISUM = 0
DO I = 1, 10, 1
    IF (I == 4) CYCLE
    ISUM = ISUM + I
END DO
WRITE(*,*) 'ISUM =', ISUM

STOP
END PROGRAM EjCycle3
```

Observe que cuando se realiza un sumatorio es importante dar el valor inicial de 0 a la variable donde se acumulará la suma de los datos. En caso de suprimir la instrucción ISUM = 0, el programa puede dar respuestas numéricas erróneas.

5.3.3. Bucles anidados y las sentencias CYCLE y EXIT

Cuando las instrucciones **CYCLE** y **EXIT** aparecen en bucles anidados, estas se aplican sobre el bucle más interno en el que están insertadas. Si los bucles tienen nombre, las instrucciones también pueden aplicarse a bucles más externos al que están insertadas, haciendo referencia al nombre de bucle como se verá en la sección 5.4.



Para mostrar el comportamiento de estas instrucciones en bucles anidados, considere el programa siguiente

```
PROGRAM test_exit2
IMPLICIT NONE
INTEGER :: i,j
DO i = 1, 3, 1
    DO j = 1, 2, 1
        IF (i == 2) EXIT
        WRITE(*,*) 'Bucle interno ', i, j
    END DO
END DO
WRITE(*,*) 'Fin del bucle DO '
STOP
END PROGRAM test_exit2
```

La ejecución del programa produce el resultado

```
Bucle interno      1      1
Bucle interno      1      2
Bucle interno      3      1
Bucle interno      3      2
Fin del bucle DO
```

Si se cambiase la posición de la instrucción **IF** (`i == 2`) **EXIT**, de modo que el programa se transformara en

```
PROGRAM test_exit3
IMPLICIT NONE
INTEGER :: i,j
DO i = 1, 3, 1
  IF (i == 2) EXIT
  DO j = 1, 2, 1
    WRITE(*,*) ' Bucle interno ', i, j
  END DO
END DO
WRITE(*,*) ' Fin del bucle DO '
STOP
END PROGRAM test_exit3
```

la ejecución del programa produciría,

```
Bucle interno      1      1
Bucle interno      1      2
Fin del bucle DO
```

En este caso, la instrucción **EXIT** interrumpe la ejecución del bucle **DO** externo, mientras que en el primer programa la instrucción **EXIT** interrumpía la ejecución del bucle **DO** interno.

Pregunta 5.1 Examine el siguiente fragmento de programa. Indique si es correcto e indique qué salida esperarías observar si se ejecutase esta parte del programa.

```
DO i = 1, 3, 1
  DO j = 1, 2, 1
    WRITE(*,*) ' Bucle interno ', i, j
  END DO
IF (i == 2) EXIT
END DO
```

Pregunta 5.2 Examine el siguiente fragmento de programa. Indique si es correcto e indique qué salida esperarías observar si se ejecutase esta parte del programa.

```
DO i = 1, 3, 1
  DO j = 1, 2, 1
    IF (i == 2) EXIT
    WRITE(*,*) ' Bucle interno ', i, j
  END DO
END DO
```

Pregunta 5.3 Examine el siguiente fragmento de programa. Indique si es correcto e indique qué salida esperarías observar si se ejecutase esta parte del programa.

```

DO i = 1, 3, 1
  DO j = 1, 2, 1
    WRITE(*,*) ' Bucle interno ', i, j
  IF (i == 2) EXIT
  END DO

```

Ejemplo 5.8 Cálculo del número e.

Escriba un programa que calcule el valor del número e.

El valor $e = 2,718282$ se puede aproximar mediante una serie infinita

$$e = \sum_{n=0}^{\infty} \left(\frac{1}{n!} \right) \approx \frac{1}{0!} + \frac{1}{1!} + \frac{1}{2!} + \frac{1}{3!} + \dots$$

Con un editor de texto, edite el fichero tarea30.f90:

```
C:\EPAQ\Valvanera> EDIT tarea30.f90 
```

donde introducirá el siguiente programa en Fortran 90

```

PROGRAM Numero_e
IMPLICIT NONE
INTEGER :: i, j
REAL :: e, fact, inc
!
WRITE(*,*) "Introduce el valor del error maximo "
READ(*,*) inc
e = 0.0
DO i = 0, 1000, 1
  ! Calculo del factorial de i
  fact = 1.0
  DO j = 2, i, 1
    fact = fact*j
  END DO
  ! Calculo del número e
  e = e + 1.0/fact
  IF (inc > 1.0/fact) EXIT
END DO
WRITE(*,*) " Terminos en la serie = ", i
WRITE(*,*) " e(apro.) = ", e
WRITE(*,*) " e(exac.) = ", EXP(1.0)
!
STOP
END PROGRAM Numero_e

```

Compile y ejecute el programa. Para comprobar que el programa funciona correctamente utilice un valor del error máximo de 0.001.

La ejecución del programa con estos datos de entrada da como resultado en la pantalla:

```
C:\EPAQ\Valvanera>tarea30

Introduce el valor del error maximo  0.001

Terminos en la serie =                7
e(apro.) =                2.71825
e(exac.) =                2.71828

Program completed
Press Enter to Continue.
```

Ejemplo 5.9 Cálculo de la función de partición rotacional de una molécula diatómica.

Escriba un programa que calcule la función de partición rotacional de una molécula diatómica conocida su temperatura rotacional, θ_r .

El valor de la función de partición rotacional de una molécula diatómica puede escribirse como

$$q_r = \sum_{J=0}^{\infty} (2J + 1) e^{-\frac{J(J+1)\theta_r}{T}}$$

donde J es el número cuántico rotacional y T la temperatura en Kelvin. La temperatura rotacional, θ_r , es una constante característica de cada molécula. Nótese que este cálculo es un sumatorio de una serie infinita.

Sin embargo, cuando θ_r es pequeña, q_r puede aproximarse como

$$q_r \approx \frac{T}{\theta_r}$$

Un programa que permite estimar el valor de la función de partición rotacional utilizando la definición de Termodinámica Estadística, o bien su aproximación, sería el siguiente

```
PROGRAM particion_rotacional
!
! Este programa calcula la funcion de particion rotacional de
! una molecula diatómica
!
! J          ... numero cuantico rotacional
! q_r       ... funcion de particion rotacional
! theta_r   ... temperatura rotacional
! T         ... temperatura (K)
! inc       ... valor de cada termino de la serie en el calculo de q_r
!           inc = (2.0*J+1.0)*EXP(-J*(J+1)*theta_r/T)
!
IMPLICIT NONE
INTEGER :: J
REAL :: q_r, theta_r, T
REAL :: inc
! Pide datos al usuario
```



```

WRITE(*,*) "Introduce el valor de la temperatura rotacional de la molecula "
READ(*,*) theta_r
WRITE(*,*) "Introduce el valor de la temperatura (K) "
READ(*,*) T
! Calculo de q_r
q_r = 0.0
DO J = 0, 1000,1
  inc = (2.0*J+1.0)*EXP(-J*(J+1)*theta_r/T)
  q_r = q_r + inc
  IF (inc < 1.0E-6) EXIT
END DO
! Resultados
WRITE(*,*) " Terminos en la serie = ", J
WRITE(*,*) " qr(aprox.) = ", T/theta_r
WRITE(*,*) " qr(serie.) = ", q_r
!
STOP
END PROGRAM particion_rotacional

```

Con un editor de texto, edite el fichero tarea31.f90:

```
C:\EPAQ\Valvanera> EDIT tarea31.f90 
```

donde introducirá el programa anterior en Fortran 90.

Compile y ejecute el programa. Para comprobar el buen funcionamiento del programa calcule los valores de q_r para las moléculas de la tabla a 300 y 500 K.

Molécula	N ₂	O ₂	CO	HI	HCl
θ_r/K	2,86	2,07	2,77	15,2	9,0

La ejecución de programa para la molécula de nitrógeno a 300 K da como resultado en la pantalla:

```

C:\EPAQ\Valvanera>tarea31

Introduce el valor de la temperatura rotacional de la molecula 2.86

Introduce el valor de la temperatura (K) 300

Terminos en la serie =                44
qr(aprox.) =          104.895
qr(serie.) =          105.229

Program completed
Press Enter to Continue.

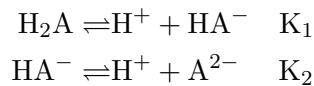
```

Nota: se ha utilizado un valor fijo de $1E-6$ para el error máximo aproximado.

Ejercicio 5.3 *Equilibrio de un ácido diprótico.*

Escriba un programa que lea los valores de pK_1 y pK_2 ($pK = -\log K$) de un ácido diprótico y calcule f en el intervalo de $pH = 0-14$, donde f es la fracción. f_{H_2A} , f_{HA^-} y $f_{A^{2-}}$.

Para los equilibrios de un ácido diprótico H_2A y sus bases conjugadas HA^- y A^{2-}



se sabe que las fracciones de las especies H_2A , HA^- y A^{2-} (f_{H_2A} , f_{HA^-} y $f_{A^{2-}}$ expresadas en función de la concentración de H^+), están dadas por

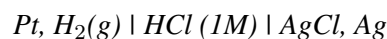
$$\begin{aligned} f_{H_2A} &= \frac{|H^+|^2}{D} \\ f_{HA^-} &= K_1 \cdot \frac{|H^+|}{D} \\ f_{A^{2-}} &= \frac{K_1 \cdot K_2}{D} \end{aligned}$$

donde

$$D = |H^+|^2 + K_1 \cdot |H^+| + K_1 \cdot K_2$$

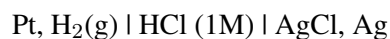
Ejercicio 5.4 *Dependencia de la FEM de una pila con la temperatura.*

Escriba un programa que calcule E^0 de la pila



en un intervalo de temperaturas de T_i a T_f , dado el incremento de temperaturas ΔT .

La FEM de la pila electroquímica



varía con la temperatura de acuerdo con la ecuación

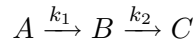
$$E^0 = 0,2239 - 6,455 \cdot 10^{-3}(T - 25) - 3,248 \cdot 10^{-6}(T - 25)^2 + 9,948 \cdot 10^{-9}(T - 25)^3$$

donde T es la temperatura en grados centígrados.

Compruebe el programa calculando E^0 en el intervalo de temperaturas $0-40^\circ C$ con un incremento de temperatura $\Delta T = 5^\circ C$.

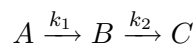
Ejercicio 5.5 *Cinética de reacciones consecutivas.*

Escriba un programa que calcule las concentraciones de A, B y C en función del tiempo para una reacción consecutiva que sigue un mecanismo



El programa debe leer como datos de entrada las constantes cinéticas k_1 , k_2 , la concentración inicial de la especie A $|A|_0$, el tiempo t_f y el incremento de tiempo Δt .

Un mecanismo común en reacciones complejas es el de las reacciones consecutivas. El caso más sencillo de este mecanismo es



En este mecanismo, las concentraciones de A, B y C varían en el tiempo de acuerdo con

$$[A] = [A]_0 \cdot e^{-k_1 \cdot t}$$

$$[B] = \left\{ \frac{k_1}{k_1 - k_2} \right\} \cdot \left\{ e^{-k_1 \cdot t} - e^{-k_2 \cdot t} \right\} [A]_0$$

$$[C] = [A]_0 - [A] - [B]$$

Utilice los valores $k_1 = 0,1 \text{ s}^{-1}$ y $k_2 = 0,2 \text{ s}^{-1}$, $|A|_0 = 5 \text{ M}$, $t_f = 5 \text{ s}$ y $\Delta t = 0,1 \text{ s}$. para comprobar el funcionamiento del programa.

Ejercicio 5.6 *Números primos.*

Escriba un programa que encuentre todos los números primos de tres dígitos.

Recuerde que un número P es primo sólo si es divisible por sí mismo y por uno. A la hora de escribir el programa es útil tener en cuenta que si un número P es primo entonces no hay otro número A, que cumpla que $1 < A < P$, tal que el resto de la división, $P - A * (P/A)$, sea cero.

En Fortran hay una función intrínseca que calcula el resto de una división de enteros:

$$\text{MOD}(A,P) = A - P * \text{INT}(A/P)$$

5.4. Bucles DO y bloques IF con nombre

En Fortran 90 puede añadirse una etiqueta a los nombres de los bucles DO y a los bloques IF. Esto es útil cuando se combina con las instrucciones EXIT y CYCLE en bucles DO anidados.

Se puede encontrar más información en la bibliografía.



Sintaxis de los bucles DO con nombre

```
Nombre : DO indice = valor inicial , valor final , incremento
instruccion_1
instruccion_2
...
instruccion_n
END DO Nombre
```

```
Nombre : DO WHILE (expresion logica)
instruccion_1
instruccion_2
...
instruccion_n
END DO Nombre
```

Nombre es la etiqueta del bucle DO. Si en un programa se utilizan más de un bucle DO con Nombre, cada bucle ha de tener su propio Nombre.

Ejemplo,

```
ej1 : DO i = 1, 5, 1
      WRITE(*,*) i
      END DO ej1
```

Sintaxis de la instrucción EXIT con nombre

```
EXIT Nombre
```

En el momento en el que se alcanza esta instrucción se interrumpe la ejecución del bucle Nombre DO/DO WHILE y continúa la ejecución en la línea siguiente a END DO Nombre.

Ejemplo,

```
ej1 : DO i = 1, 5, 1
      IF (i == 2) EXIT ej1
      WRITE(*,*) i
      END DO ej1
```

Sintaxis de la instrucción CYCLE con nombre**CYCLE** Nombre

En el momento en el que se alcanza esta instrucción, se interrumpe la ejecución de las instrucciones del cuerpo del bucle DO y se inicia una nueva iteración del bucle.

Ejemplo,

```
ej1 : DO i = 1, 5, 1
      IF ( i == 2) CYCLE ej1
      WRITE(*,*) i
      END DO ej1
```

Ejemplo 5.10 Bucles DO con nombre.

En el ejemplo siguiente se muestran dos bucles DO anidados con nombre y cómo se utilizan las instrucciones **CYCLE** y **EXIT** para controlar la ejecución de los bucles.

```
PROGRAM EjDONombre

IMPLICIT NONE
INTEGER :: i, j

externo : DO i = 1, 10, 1
  WRITE(*,*) "bucle DO Externo"
  WRITE(*,*) 'i = ', i
  WRITE(*,*) "bucle DO Interno"
  interno : DO j = 1, 10, 1
    WRITE(*,*) "i =", i, " j =", j
    IF ( i == 3) CYCLE externo
    WRITE(*,*) "Despues de CYCLE Externo :Indices =", i, j
    IF ( i == 4) EXIT externo
    WRITE(*,*) "Despues de EXIT Externo :Indices =", i, j
    IF ( i == 2) EXIT interno
    WRITE(*,*) "Despues de Exit Interno :Indices =", i, j
  END DO Interno
  WRITE(*,*) "Despues de END DO Interno :Indices =", i, j
  WRITE(*,*) "i = ", i, " j = ", j
END DO Externo
WRITE(*,*)
WRITE(*,*) " Fuera bucles DO i = ", i, " j = ", j

STOP
END PROGRAM EjDONombre
```

La ejecución del programa produce en la pantalla los resultados,

```
Despues de Exit Interno :Indices =          1          10
Despues de END DO Interno :Indices =        1          11
i =          1 j =          11
bucle DO Externo
i =          2
bucle DO Interno
i =          2 j =          1
Despues de CYCLE Externo :Indices =          2          1
Despues de EXIT Externo :Indices =          2          1
Despues de END DO Interno :Indices =        2          1
i =          2 j =          1
bucle DO Externo
i =          3
bucle DO Interno
i =          3 j =          1
bucle DO Externo
i =          4
bucle DO Interno
i =          4 j =          1
Despues de CYCLE Externo :Indices =          4          1

Fuera bucles DO i =          4 j =          1

Program completed
Press Enter to Continue.
```

Práctica 6

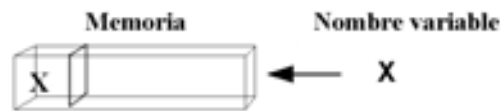
Variables subindicadas (I)

CONTENIDOS

6.1. Variables subindicadas: arreglos unidimensionales, vectores o listas	108
6.2. Cómo se declara una variable subindicada	109
6.3. Lectura y escritura de vectores	111
6.4. Cómo utilizar vectores en instrucciones en Fortran	118
6.5. Funciones intrínsecas de vectores	127

6.1. Variables subindicadas: arreglos unidimensionales, vectores o listas

Las variables que se han utilizado hasta ahora se llaman *variables no subindicadas* o *variables escalares*. Una variable escalar está asociada con una celda de memoria y en cada celda de la memoria se almacena un dato (un número entero, un número real, caracteres, etc). Podemos representar una variable, X, como una caja

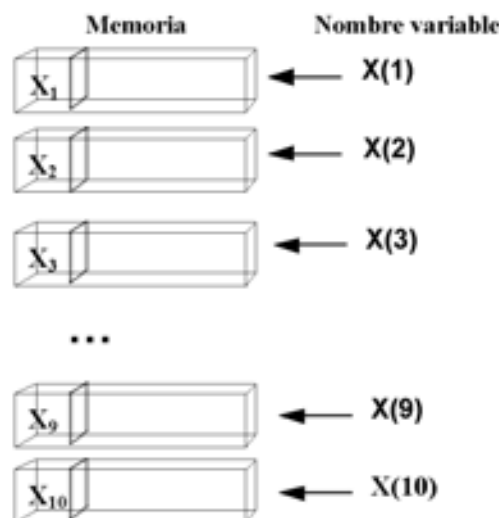


Suponga que se han hecho una serie de n medidas de una propiedad x y se quiere calcular su media. Esta operación se expresaría:

$$\bar{x} = \frac{1}{n} \sum_{i=1}^n x_i = \frac{1}{n} \{x_1 + x_2 + \dots + x_n\}$$

donde x_1 es el resultado de la primera medida (el primer valor de la serie), x_2 es el resultado de la segunda medida (el segundo valor de la serie), ..., y x_n es el resultado de la n -sima medida (el n -simo valor de la serie). Como se puede observar todas las medidas se representan con el mismo nombre, pero se distinguen con un número que indica el lugar que ocupan en la serie. En Fortran, cuando trabajamos con una serie de valores se utiliza un tipo de variable conocida como *arreglo* o *variable subindicada*. Estas variables están asociadas a un conjunto de celdas de memoria que tienen el mismo nombre pero que se distinguen por el valor de un índice. Cuando la variable subindicada tiene sólo un índice, se llama arreglo lineal, arreglo unidimensional, vector o lista. Cuando tiene más de un índice es un arreglo multidimensional o matriz.

Un vector con n elementos es un arreglo de n variables que tienen el mismo nombre y en el que cada elemento se distingue por el valor de su índice. Suponga que la variable X corresponde a un vector de diez elementos. La variable X se puede representar como un conjunto de diez cajones (celdas de memoria), donde se hace referencia a cada cajón mediante el nombre de la variable y un número entre paréntesis (el índice).



Esta herramienta es particularmente útil cuando se desea aplicar un algoritmo de manera iterativa a un conjunto de datos. Por ejemplo, considere que necesita calcular la raíz cuadrada de 100 valores reales. Suponga que dispone de una lista de 100 valores. Si se introduce un vector $a(i)$, el cálculo puede realizarse con el código

```
DO i = 1, 100, 1
  a(i) = SQRT(a(i))
END DO
```

Si el cálculo tuviera que hacerse sin usar un arreglo, se necesitarían 100 variables distintas y 100 instrucciones de asignación. Es evidente que el código que incluye un arreglo es más sencillo y claro.

Para describir un arreglo se tiene que especificar¹:

1. El tipo de datos que almacena.
2. El nombre del arreglo.
2. El número de subíndices del arreglo: uno (vector), dos (matriz), etc. Esta propiedad se denomina rango (*rank*) del arreglo.
3. Los valores posibles de los subíndices del arreglo que determinan la extensión (*extent*) del arreglo. El rango y la extensión definen la forma (*shape*) del arreglo. El número máximo de valores o tamaño (*size*) del arreglo viene determinado por la forma y la extensión de este.

6.2. Cómo se declara una variable subindicada

Antes de utilizar una variable subindicada es necesario declarar su tipo y el número de elementos que contiene. Para esto se utiliza una instrucción de declaración de tipo que indica al compilador el tipo de datos a almacenar en la variable y que reserve la memoria necesaria para almacenarlos. Por ejemplo, si se desea declarar el vector Intensidad de 20 elementos, se declararía

```
REAL, DIMENSION(1:20) :: Intensidad
```

En el ejemplo anterior, se declara una lista de 20 elementos: Intensidad(1), Intensidad(2), ..., Intensidad(20). El atributo **DIMENSION** en una declaración de tipo indica que esas variables corresponden a un arreglo. Con la expresión (1:20) se indica al compilador que en este arreglo los elementos se identifican con un único índice que puede variar desde 1 a 20 y que la lista es de veinte elementos.

La declaración del ejemplo anterior también puede escribirse como

```
REAL, DIMENSION(20) :: Intensidad
```

ya que cuando no se indica el valor inicial del índice del arreglo el compilador le asigna automáticamente 1.

¹Las definiciones de rango, extensión, forma o tamaño son particularmente importantes cuando se trabaja con matrices o tablas en la próxima práctica.

Otros ejemplos de declaración de variables son

```
REAL, DIMENSION(5:9) :: a  
INTEGER, DIMENSION(0:99) :: r
```

Con la primera de estas instrucciones, se define una variable real a que puede guardar cinco valores en los elementos a(5), a(6), a(7), a(8) y a(9). Con la segunda, se define una variable entera r que puede almacenar hasta cien valores en r(0), r(1), ..., r(99).

A veces, se recomienda utilizar constantes con nombre de tipo entero para determinar el tamaño de un arreglo. Es decir, en la instrucción de declaración del arreglo los valores de los índices se sustituyen por constantes con nombre. De esta manera, basta con cambiar el valor de la constante para que se modifique la dimensión del arreglo. Por ejemplo,

```
INTEGER, PARAMETER :: nmin=1, nmax = 100  
REAL, DIMENSION(nmin:nmax) :: P,Q,R
```

Pregunta 6.1 *Indique si las siguientes declaraciones de arreglos son válidas. En caso contrario indique por qué no son válidas*

- a. **INTEGER, DIMENSION(1:15) :: atomo**
- b. **CHARACTER, DIMENSION(1:-10) :: atomo**
- c. **REAL, DIMENSION(-10:10) :: linea**
- d. **REAL, DIMENSION(2.5:4.4) :: error**
- e. **INTEGER, PARAMETER :: nmin = 0, nmax= 5**
REAL, DIMENSION(nmax:nmin) :: linea

Pregunta 6.2 *¿Qué tamaño (size) tienen los vectores declarados con las siguientes instrucciones?*

- a. **INTEGER, DIMENSION(1:15) :: experimento**
- b. **REAL, DIMENSION(0:10) :: fase**
- c. **CHARACTER, DIMENSION(-1:1) :: tipo**
- d. **REAL, DIMENSION(-3:0) :: error**

Sintaxis de declaración de variables subindicadas: vectores

TIPO, **DIMENSION**([Ninicial]:Nfinal) :: Variable1 [, Variable2, ...]

Para declarar una variable subindicada hay que indicar junto al tipo de variable el atributo **DIMENSION**. A continuación del atributo **DIMENSION**, el número de elementos del arreglo se indica con dos números entre paréntesis separados por dos puntos : . Ninicial es el número con el que se hace referencia al valor del índice del primer elemento del arreglo mientras que Nfinal es el valor del índice para el último elemento del arreglo. Estos valores son arbitrarios, pueden ser negativos o cero, pero siempre $N_{final} > N_{inicial}$. El vector tiene $N_{final} - N_{inicial} + 1$ elementos. Si se suprime el índice Ninicial, el compilador le asigna automáticamente el valor 1.

Ejemplos,

```
REAL, DIMENSION(20) :: Intensidad
REAL, DIMENSION(1:100) :: concentracion, pH
INTEGER, DIMENSION(0:5) :: prueba
CHARACTER(LEN=3), DIMENSION(-1:1) :: transicion
```

6.3. Lectura y escritura de vectores

Existen distintos métodos para leer (asignar valores con instrucciones de lectura) datos de variables que corresponden a un vector.

i. Lectura y escritura de elementos aislados de un vector

Suponga que en un programa ha definido la variable escalar x y el vector t. La instrucción

```
READ(*,*) x, t(3)
```

sirve para leer el valor de x y el valor del elemento t(3) del vector t. Una instrucción semejante sirve para escribir en la pantalla esos valores

```
WRITE(*,*) x, t(3)
```

Si en lugar de leer elementos aislados del vector se quieren leer (o imprimir) todos sus elementos, con este método se tendrían que indicar explícitamente todos los elementos del arreglo. Por ejemplo, si quiere leer valores para el vector ab que consta de 10 elementos, y su índice varía de 1 a 10, puede leerlos utilizando la instrucción:

```
READ(*,*) ab(1), ab(2), ab(3), ab(4), ab(5), ab(6), ab(7), ab(8), ab(9), ab(10)
```

ii. Lectura y escritura de elementos de un vector utilizando un bucle DO

Con este método se pueden leer todos o parte de los elementos de un vector. Se basa en relacionar el índice del bucle DO con el índice del arreglo. Si el arreglo *ab* consta de *n* elementos con valores de su índice de 1 a *nmax*, una manera sencilla de leerlos sería

```
DO i = 1, nmax, 1
  READ(*,*) ab(i)
END DO
```

Si sólo se quieren leer los valores correspondientes a los elementos del vector con valores del índice entre *j_inicial* y *j_final*, se puede escribir

```
DO i = j_inicial, j_final, 1
  READ(5,*) ab(i)
END DO
```

Recuerde que en cada ejecución del bucle DO la instrucción **READ** de los ejemplos anteriores sólo lee una línea. Es decir, si los datos se leen desde el teclado, deben proporcionarse de uno en uno.

Estas instrucciones también se pueden utilizar para escribir. Por ejemplo, el fragmento de programa

```
DO j = 1, n, 1
  WRITE(6,*) ab(j)
END DO
```

escribiría *n* líneas, y en cada una de ellas aparecería el valor de un elemento del arreglo *ab*.

iii. Lectura y escritura de elementos de un vector utilizando un bucle DO implícito

El funcionamiento de los bucles DO implícitos es muy similar al de los bucles DO cuando se utilizan para leer o escribir datos de elementos de un arreglo.

Por ejemplo, la instrucción

```
READ(*,*) (a(i), i = 1,5,1)
```

es equivalente a la instrucción

```
READ(*,*) a(1), a(2), a(3), a(4), a(5)
```

La forma general de estos bucles DO implícitos es

```
READ(unit,f) ( lista de variables, indice = valor inicial, valor final, incremento)
WRITE(unit,f) ( lista de variables, indice= valor inicial, valor final, incremento)
```

A diferencia de los bucles DO, la instrucción **READ** en un bucle DO implícito sólo se ejecuta una vez, es decir, intenta leer todos los datos en una sola línea de pantalla o fichero. Si no los encuentra, continúa en la línea siguiente. Análogamente, la instrucción **WRITE(u,*)** en un bucle DO implícito intenta escribir todos los datos en una línea de la pantalla o fichero, y salta a la línea siguiente si no hay suficiente espacio.

Sintaxis de la lectura y escritura con bloques DO implícitos

READ(unit,f) (lista de variables, índice = valor inicial , valor final , incremento)
WRITE(unit,f) (lista de variables, índice= valor inicial , valor final , incremento)

Por ejemplo,

```
READ(*,*) (A(I), I= 1, N)
WRITE(*,*) (A(I), I= 1, N)
```

iv. Lectura y escritura de elementos de un vector utilizando sólo el nombre del arreglo

Cuando se desean leer o escribir todos los elementos de un vector, basta con escribir el nombre del arreglo.

Por ejemplo,

```
READ(*,*) a
WRITE(*,*) a
```

v. Lectura y escritura de un subconjunto de elementos de un vector

En Fortran 90 también es posible hacer referencia a un subconjunto de los elementos de un arreglo – *array subset* –. Los elementos que componen el subconjunto se identifican sustituyendo el índice por una tripleta de valores como se muestra en la instrucción siguiente:

```
READ(*,*) a(1:10:2)
```

que indica que los datos leídos se almacenarán en el subconjunto de elementos del vector **a** cuyos índices varían entre 1 y 10, pero con un incremento de 2. Es decir, lee los valores de los elementos **a(1)**, **a(3)**, **a(5)**, **a(7)** y **a(9)**.

La estructura de una tripleta es:

```
( valor_inicial : valor_final : incremento)
```

donde **valor_inicial** es el valor del índice del primer elemento del subconjunto, **valor_final** es el valor del índice del último elemento del subconjunto e **incremento** es la diferencia entre los valores de los índices de dos elementos consecutivos.

Si en una tripleta se suprime incremento, el compilador asigna automáticamente el valor 1. Por ejemplo,

```
READ(*,*) a(1:10)
```

Otras posibilidades de la sintaxis de las tripletas se comentarán en ejemplos posteriores. En cualquier caso, se puede encontrar una descripción detallada de éstas en la bibliografía.

Pregunta 6.3 Examine el siguiente fragmento de programa.

Indique si el código es correcto. ¿Qué elementos de las listas espera que se impriman?

```
INTEGER, PARAMETER :: n = 10 ! valor maximo del indice
INTEGER :: i                ! indice bucle
REAL, DIMENSION(n) :: numero ! numeros
REAL, DIMENSION(n) :: raiz2 ! raiz cuadrada de numero
REAL, DIMENSION(n) :: raiz3 ! raiz cubica de numero
...
WRITE (*,*) (numero(i), raiz2(i), raiz3(i), i = 1, n, 1)
```

Pregunta 6.4 Examine el siguiente fragmento de programa.

Indique si el código tiene errores de sintaxis o puede dar lugar a un error de ejecución. En su caso, corrija los errores para que el programa sea válido.

```
INTEGER, PARAMETER :: n = 10 ! valor maximo del indice
INTEGER :: i                ! indice bucle
REAL, DIMENSION(n) :: x
...
READ (*,*) (x(i), raiz2(i), raiz3(i), i = 1, n, 1)
```

Ejemplo 6.1 Lectura y escritura de elementos de un vector utilizando un bucle DO implícito.

Este ejemplo muestra la utilización de los bucles DO y bucles DO implícitos para leer y escribir datos. El programa también muestra las diferencias entre ambos métodos de lectura y escritura de datos.

Con un editor de texto, edite el fichero tarea32.f90:

```
C:\EPAQ\Valvanera> EDIT tarea32.f90 
```

donde introducirá el siguiente programa en Fortran 90

```
PROGRAM ejdoimplicito
!
!
! Mostrar el uso de los bucles DO y bucles DO implícitos
! para leer y escribir los elementos de un vector.
```

```

!

IMPLICIT NONE
INTEGER :: i, n
INTEGER, DIMENSION (1:5) :: a
INTEGER, DIMENSION (-1:3) :: b

n = 5

WRITE(* ,*) " Utiliza bucles DO para leer y escribir datos "
WRITE(* ,*) " Introduce cinco pares de valores enteros separados por un espacio"
WRITE(* ,*) " Despues de escribir cada par de valores pulse ENTER"
WRITE(* ,*)

DO i=1,n
  READ(* ,*) a(i), b(i-2)
END DO
DO i=1,n
  WRITE(* ,*) a(i), b(i-2)
END DO

WRITE(* ,*) " Utiliza bucles DO implicitos para leer y escribir datos "
WRITE(* ,*) " Introduce cinco pares de valores enteros separados por un espacio"
WRITE(* ,*) " Escriba todos los valores en la misma linea"
WRITE(* ,*) " Despues de escribir cada par de valores pulse ENTER"
WRITE(* ,*)

READ(* ,*) (a(i),b(i-2),i=1,n,1)
WRITE(* ,*)(a(i),b(i-2),i=1,n,1)

STOP
END PROGRAM ejdoimplicito

```

Compile y ejecute el programa.

La ejecución del programa dará como resultado en la pantalla:

```

C:\EPAQ\Valvanera>tarea32

Utiliza bucles DO para leer y escribir datos
Introduce cinco pares de valores enteros separados por un espacio
Despues de escribir cada par de valores pulse ENTER
1 2
3 4
5 6
7 8
9 0

          1          2
          3          4
          5          6
          7          8
          9          0

Utiliza bucles DO implicitos para leer y escribir datos

```

Introduce cinco pares de valores enteros separados por un espacio
Escriba todos los valores en la misma línea.

Después de escribir cada par de valores pulse ENTER

1 2 3 4 5 6 7 8 9 0

1	2	3	4	5	6
7	8	9	0		

Program completed
Press Enter to Continue.

Ejemplo 6.2 Lectura y escritura de elementos de un vector utilizando el nombre del arreglo.

Este ejemplo muestra la utilización del nombre del arreglo para leer y escribir los valores guardados en los vectores a y b sin necesidad de utilizar bucles DO.

Note que en la ejecución del programa primero se leen los valores del arreglo a hasta que éste se llena y a continuación se leen los valores que se almacenarán en el arreglo b.

Con un editor de texto, edite el fichero tarea33.f90:

```
C:\EPAQ\Valvanera> EDIT tarea33.f90 
```

donde introducirá el siguiente programa en Fortran 90:

```
PROGRAM esnombrearreglo
!
! Mostrar la lectura y escritura de los elementos de un arreglo
! utilizando el nombre del arreglo.
!
IMPLICIT NONE
INTEGER, DIMENSION(1:5) :: a
INTEGER, DIMENSION(-1:3) :: b

WRITE(*,*) " Introduce cinco pares de valores enteros separados por un espacio"
WRITE(*,*) " Escriba todos los valores en la misma línea"
WRITE(*,*)

READ(*,*) a,b

WRITE(*,*) " Escribe los valores guardados en el arreglo A"
WRITE(*,*) a
WRITE(*,*) " Escribe los valores guardados en el arreglo B"
WRITE(*,*) b

STOP
END PROGRAM esnombrearreglo
```

Compile y ejecute el programa.

La ejecución del programa dará como resultado en la pantalla:

```
C:\EPAQ\Valvanera>tarea33

Introduce cinco pares de valores enteros separados por un espacio
Escriba todos los valores en la misma linea
 1 2 3 4 5 6 7 8 9 0

Escribe los valores guardados en el arreglo A
          1          2          3          4          5
Escribe los valores guardados en el arreglo B
          6          7          8          9          0

Program completed
Press Enter to Continue.
```

Ejemplo 6.3 *Lectura y escritura de un subconjunto de los elementos de un vector utilizando una tripleta.*

Este ejemplo muestra la utilización de una tripleta para leer valores de un subconjunto de elementos del vector a.

Note que en la ejecución del programa primero se asigna el valor 0 a todos los elementos del arreglo a. Posteriormente, en la lectura sólo se asignan valores a los elementos a(1), a(3) y a(5).

Con un editor de texto, edite el fichero tarea34.f90:

```
C:\EPAQ\Valvanera> EDIT tarea34.f90 
```

donde introducirá el siguiente programa en Fortran 90:

```
PROGRAM ejsubsetarreglo
!
!
! Mostrar el uso de una tripleta para la lectura y escritura
! de un subconjunto de los elementos de un arreglo
!
IMPLICIT NONE
INTEGER, DIMENSION(1:5) :: a

a = 0

WRITE(*,*) " Introduce cinco valores enteros separados por un espacio"
WRITE(*,*) " Escriba todos los valores en la misma linea"
WRITE(*,*)

READ(*,*) a(1:5:2)

WRITE(*,*) " Escribe los valores guardados en el arreglo A"
WRITE(*,*) a

STOP
END PROGRAM ejsubsetarreglo
```

Compile y ejecute el programa.

La ejecución del programa dará como resultado en la pantalla:

```
C:\EPAQ\Valvanera>tarea34

Introduce cinco valores enteros separados por un espacio
Escriba todos los valores en la misma linea
      1  2  3  4  5

Escribe los valores guardados en el arreglo A
      1          0          2          0          3

Program completed
Press Enter to Continue.
```

6.4. Cómo utilizar vectores en instrucciones en Fortran

i. Asignación de valores a elementos aislados de un vector

Cada uno de los elementos de un arreglo se comporta como cualquier otra variable. Por tanto, puede incluirse en una instrucción en lugar de cualquier otra variable escalar del mismo tipo. Los elementos de un arreglo pueden utilizarse en expresiones aritméticas y lógicas, y pueden almacenar el resultado de evaluar una expresión. Por ejemplo,

$$b(4) = 3 * R - \text{EXP}(x(2))$$

ii. Asignación de valores a varios elementos de un vector utilizando bucles DO

Cuando se quiere asignar valores a una serie de elementos de un vector se utiliza un bucle DO. Por ejemplo, considere un fragmento de programa en el que se calculan valores de constantes cinéticas en un intervalo de temperaturas utilizando la ley de Arrhenius,

```
DO i = 1, nt, 1
  T(i) = T0 + (i-1)*incT
  k(i) = A*EXP(-Ea/(R*T(i)))
END DO
```

En estos casos puede ser conveniente utilizar una tripleta en lugar del bucle DO. Por ejemplo, un código equivalente al anterior sería

```
DO i = 1, nt, 1
  T(i) = T0 + (i-1)*incT
END DO
k(1:nt:1) = A*EXP(-Ea/(R*T(1:nt:1)))
```

iii. Operaciones con todos los elementos de un vector

Fortran 90 permite hacer operaciones con todos los elementos de un vector como si fueran variables escalares. Considere los vectores x e y declarados

```
REAL, DIMENSION(1:10) :: x, y
```

La instrucción

```
x = 1.0
```

equivale a utilizar el siguiente bucle DO

```
DO i = 1, 10, 1
  x(i) = 1.0
END DO
```

Muchas de las funciones intrínsecas del Fortran que se utilizan con valores escalares aceptan arreglos como argumentos y producen otro arreglo como resultado. El arreglo resultante se obtiene evaluando la expresión y aplicando la función elemento a elemento ². Funciones de este tipo son **ABS**, **SIN**, **COS**, **EXP**, **LOG**. Así, la instrucción

```
x = a*EXP(-x)
```

donde a es una variable escalar equivale a

```
DO i = 1, 10, 1
  x(i) = a*EXP(-x(i))
END DO
```

Las expresiones con vectores pueden incluir varios vectores. Para que estas operaciones sean válidas, los arreglos que aparecen en la expresión deben tener la misma forma (*shape*). Esto supone que: (1) deben tener el mismo rango y (2) deben tener el mismo número de elementos en cada dimensión (*extensión*). Cuando dos arreglos cumplen esta condición se dice que son *conformables* (compatibles) y pueden utilizarse en expresiones aritméticas que implican a todos los elementos de ambos vectores.

Los vectores x e y que se declararon arriba, son compatibles. Así, las siguientes expresiones (y sus equivalentes) son correctas:

- $x = y$
que equivale a

```
DO i = 1, 10, 1
  x(i) = y(i)
END DO
```

²Estas funciones intrínsecas, que se evalúan elemento a elemento cuando el argumento es un arreglo, se conocen como *funciones intrínsecas elementales*

- $y = x + y$
representa la suma de dos vectores y equivale a

```
DO i = 1, 10, 1
  y(i) = x(i) + y(i)
END DO
```

- $y = x * y$
equivale a

```
DO i = 1, 10, 1
  y(i) = x(i) * y(i)
END DO
```

NOTA: No debe confundirse esta expresión con el producto escalar

- $y = x * \text{EXP}(y)$
equivale a

```
DO i = 1, 10, 1
  y(i) = x(i) * EXP(y(i))
END DO
```

Si intentamos operar con vectores que no son compatibles, el compilador producirá un error de compilación.

Ejemplo 6.4 Errores de sintaxis: operaciones con vectores no compatibles.

Considere el siguiente programa.

```
PROGRAM noequivalentes
IMPLICIT NONE
REAL, DIMENSION(1:10) :: x
REAL, DIMENSION(1:20) :: z

z = 1
x = z

WRITE(*,*) x

STOP
END PROGRAM noequivalentes
```

Los arreglos x y z tienen distinta extensión. En el programa, se intenta realizar una operación con ambos vectores pero, dado que los vectores tienen distinta forma, el código no es aceptable. Al intentar compilar el programa, el compilador envía el siguiente mensaje de error

```
Compiling file noconform.f90.
Compiling program unit NOEQUIVALENTES at line 2.
Line 7, file noconform.f90
```

```

          x = z
          |
FATAL -- Array shapes do not conform.

Encountered 1 error, 0 warnings in file noconform.f90.

```

e interrumpe el proceso de compilación y ensamblaje del programa.



Errores de sintaxis. Cuando se realizan operaciones que implican a todos los elementos de dos o más arreglos, estos deben tener la misma forma. En el caso de vectores (arreglo de rango 1), esto supone que ambos vectores deben tener el mismo número de elementos, no importa que los valores de los índices no coincidan.

Ejemplo 6.5 Sentencias de asignación con vectores.

Este programa muestra la asignación de valores a los elementos de un vector y la utilización de un vector para calcular un sumatorio

Con un editor de texto, edite el fichero tarea35.f90:

```
C:\EPAQ\Valvanera> EDIT tarea35.f90 
```

donde introducirá el siguiente programa en Fortran 90:

```

PROGRAM vector1
!
!
! Este programa muestra algunas operaciones con vectores
! (1) Asignacion de valores a los elementos de un vector
! (2) Utilizacion de un vector para calcular un sumatorio
!
!
IMPLICIT NONE
INTEGER :: i
REAL :: sumatorio
REAL, DIMENSION(1:10) :: x

x = 1.0
sumatorio = 0.0
DO i = 1, 10, 1
    sumatorio = sumatorio + x(i)
END DO
WRITE(*,*) " Sumatorio – bucle DO 1 :: ", sumatorio

DO i = 1, 10, 1
    x(i) = 1.0
END DO
sumatorio = 0.0
DO i = 1, 10, 1

```

```

        sumatorio = sumatorio + x(i)
END DO
WRITE(*,*) " Sumatorio – bucle DO 2 :: ", sumatorio

STOP
END PROGRAM vector1

```

Compile y ejecute el programa.

La ejecución del programa dará como resultado en la pantalla:

```

C:\EPAQ\Valvanera>tarea35

Sumatorio - bucle DO 1 ::      10.0000
Sumatorio - bucle DO 2 ::      10.0000

Program completed
Press Enter to Continue.

```

Ejemplo 6.6 Operaciones con vectores.

Este programa muestra algunas operaciones con vectores.

Con un editor de texto, edite el fichero tarea36.f90:

```
C:\EPAQ\Valvanera> EDIT tarea36.f90 
```

donde introducirá el siguiente programa en Fortran 90:

```

PROGRAM vector3
!
! Este programa muestra algunas operaciones con vectores
!
IMPLICIT NONE
INTEGER :: i
REAL, DIMENSION(1:4) :: x,y,z

y = 1
DO i = 1,4,1
    x(i) = REAL(i)
END DO

WRITE(*,*) "X = ",x
WRITE(*,*) "Y = ",y

z = x + y ! Suma de dos vectores
WRITE(*,*) "Z = X + Y", z

z = x * y ! Producto de los elementos de dos vectores

```

```
WRITE(*,*) "Z=X*Y ", z
```

```
z = EXP(X) ! Exponencial de los elementos de un vector
```

```
WRITE(*,*) "z = EXP(X) ",z
```

```
STOP
```

```
END PROGRAM vector3
```

Compile y ejecute el programa.

La ejecución del programa dará como resultado en la pantalla:

```
C:\EPAQ\Valvanera>tarea36
```

```
X =      1.00000      2.00000      3.00000      4.00000
Y =      1.00000      1.00000      1.00000      1.00000
Z = X + Y      2.00000      3.00000      4.00000      5.00000
Z=X*Y      1.00000      2.00000      3.00000      4.00000
z = EXP(X)      2.71828      7.38906      20.0855      54.5981
```

```
Program completed
```

```
Press Enter to Continue.
```

Ejemplo 6.7 Presión de vapor de un líquido puro.

Escriba un programa que sirva para evaluar los valores de la presión de vapor de un líquido puro P_v en un intervalo de temperaturas desde T_i a T_f , conocidas P_0 , T_0 y $\Delta_v H$ y el incremento de temperatura, ΔT .

Para comprobar el funcionamiento del programa calcule la presión de vapor del agua en el intervalo 21 - 30 °C con un incremento de temperatura de 1,5 °C. $R = 8.314 \text{ J}\cdot\text{mol}^{-1}\cdot\text{K}^{-1}$. La entalpía de vaporización del agua es prácticamente constante en el intervalo 20 - 30 °C ($\Delta_v H = 43,9 \text{ kJ}\cdot\text{mol}^{-1}$). La presión de vapor del agua a 20 °C es 17,535 Torr.

La ecuación de Clausius-Clapeyron predice que la dependencia de la presión de vapor de un líquido puro con la temperatura viene dada por

$$\ln \frac{P_1}{P_0} = -\frac{\Delta H_v(T)}{R} \left(\frac{1}{T_1} - \frac{1}{T_0} \right)$$

que también se puede escribir como

$$P_1 = P_0 \cdot \exp \left[-\frac{\Delta H_v(T)}{R} \left(\frac{T_0 - T_1}{T_0 T_1} \right) \right]$$

Un programa que realiza esta tarea es

```
PROGRAM clausius
```

```
!
```

```
! Este programa calcula la presion de vapor de una sustancia  
! utilizando la ecuación de Clausius_Clapeyron
```

```

!
! nt      ... numero de datos de la tabla
! Ti      ... temperatura inicial del intervalo
! Tf      ... temperatura final del intervalo
! incT    ... incremento del temperatura
! P       ... presion de vapor
! T       ... temperatura
! DeltaHv ... entalpia de vaporizacion
! P0      ... presion de vapor a la temperatura T0
! T0      ... temperatura de referencia
!
IMPLICIT NONE
INTEGER :: i, nt
REAL,PARAMETER :: R = 8.314
REAL :: T0,P0, DeltaHv, Q
REAL :: Ti, Tf, incT
REAL, DIMENSION(1:100) :: T, P

WRITE(*,*) "Introduce en Kelvin Ti, Tf, IncT "
READ(*,*) Ti, Tf, incT
WRITE(*,*) "Introduce DeltaHV en J/mol, P0 en Torr, T0 en K "
READ(*,*) DeltaHv, P0, T0

nt = INT((Tf -Ti)/incT) + 1
IF (nt >100) THEN
  WRITE(*,*) "nt deber ser <= 100)"
  STOP
END IF

WRITE(*,*) "Entalpia de vaporizacion ", DeltaHv
WRITE(*,*) "P0 ", P0
WRITE(*,*) "T0 ", T0
WRITE(*,*)

Q = DeltaHv/R
DO i = 1,nt,1
  T(i) = Ti + (i-1)*incT ! Calculo el valor de Ti
  P(i) = P0*EXP(-Q*(T0-T(i))/(T0*T(i))) ! Calculo el valor de Pi
END DO
WRITE(*,*)
WRITE(*,*) "Temperatura /K P / Torr "
DO i = 1,nt,1
  WRITE(*,*) T(i) ,P(i)
END DO

STOP
END PROGRAM clausius

```

Con un editor de texto, edite el fichero tarea37.f90:

```
C:\EPAQ\Valvanera> EDIT tarea37.f90 
```

donde introducirá el anterior programa en Fortran 90.

Compile y ejecute el programa.

La ejecución del programa dará como resultado en la pantalla:

```
C:\EPAQ\Valvanera>tarea37

Introduce en Kelvin Ti, Tf, IncT 293.0 303.0 1.0

Introduce DeltaHV en J/mol, P0 en Torr, T0 en K 43900 17.535 293.0

Entalpia de vaporizacion      43900.0
P0      17.5350
T0      293.000

Temperatura /K      P / Torr
293.000      17.5350
294.000      18.6435
295.000      19.8138
296.000      21.0489
297.000      22.3519
298.000      23.7260
299.000      25.1745
300.000      26.7009
301.000      28.3088
302.000      30.0018
303.000      31.7840

Program completed
Press Enter to Continue.
```

Ejemplo 6.8 *La ley de Arrhenius.*

Escriba un programa que calcule utilizando la ley de Arrhenius la constante cinética k en un intervalo de temperaturas desde T_i a T_f , conocidos el factor preexponencial A , la energía de activación E_a y el número de valores a calcular nt .

El programa debe utilizar arreglos para almacenar los valores de la temperatura y la constante cinética correspondiente.

Para comprobar su funcionamiento calcule los valores de la constante cinética de descomposición de acetaldehído en el intervalo 700 - 1000 K, sabiendo que $A = 5,3 \cdot 10^{11} \text{ s}^{-1}$ y $E_a = 184 \text{ kJ}\cdot\text{mol}^{-1}$.

Con un editor de texto, edite el fichero tarea38.f90:

```
C:\EPAQ\Valvanera> EDIT tarea38.f90 
```

donde introducirá el siguiente programa en Fortran 90:

PROGRAM Arrhenius2

```
!
! Este programa calcula los valores de k para una reaccion
! en un intervalo de temperaturas utilizando la ecuacion de
! Arrhenius
```

$$k = A * \exp(-Ea / (R * T))$$

```
!
! nt      ... numero de datos de la tabla
! Ti      ... temperatura inicial del intervalo
! Tf      ... temperatura final del intervalo
! incT    ... incremento del temperatura
! k       ... constante cinetica
! A       ... factor preexponencial
! Ea      ... energia de activacion
! R       ... constante de los gases ideales
!
```

IMPLICIT NONE

INTEGER :: i, nt

REAL, PARAMETER :: R = 8.314

REAL :: A, Ea

REAL :: Ti, Tf, incT

REAL, DIMENSION(1:100) :: T, K

WRITE(*,*) "Introduce Tinicial , Tfinal y nt (nt <100) "

READ(*,*) Ti, Tf, nt

WRITE(*,*) "Introduce A y Ea "

READ(*,*) A, Ea

IF (NT >100) **THEN**

WRITE(*,*) "nt deber ser <= 100)"

STOP

END IF

WRITE(*,*) "Factor Prexponencial ",A

WRITE(*,*) "Energia de activacion ",Ea

incT = (Tf-Ti)/**REAL**(nt-1)

DO i = 1, nt, 1

 T(i) = Ti + (i-1)*incT

 k(i) = A***EXP**(-Ea/(R*T(i)))

END DO

WRITE(*,*) "Temperatura k "

DO i = 1, nt, 1

WRITE(*,*) T(i), k(i)

END DO

STOP

END PROGRAM Arrhenius2

Compile y ejecute el programa.

La ejecución del programa con los datos de prueba produce el resultado siguiente en la pantalla:

```
C:\EPAQ\Valvanera>tarea38
Introduce Tinicial, Tfinal y NT (NT <100) 700.0, 1000.0, 5

Introduce A y Ea 5.3E11, 184.0E3

Factor Prexponencial      0.530000E+12
Energia de activacion      184000.
Temperatura                k
  700.000                  0.985216E-02
  775.000                  0.210046
  850.000                  2.60978
  925.000                  21.5498
 1000.00                   129.645

Program completed
Press Enter to Continue.
```

Ejercicio 6.1 Niveles de energía de una partícula en una caja unidimensional.

Escriba un programa que calcule la energía de los n primeros estados de un electrón confinado en una caja de potencial unidimensional.

De acuerdo con la mecánica cuántica los niveles de energía de un electrón confinado en una caja de potencial unidimensional de longitud l vienen dados por

$$E_n = \frac{h^2 n^2}{8 m_e l^2}$$

donde n es un número cuántico con valores $n = 1, 2, \dots$, $h = 6,626 \cdot 10^{-34}$ Js, $m_e = 9,109 \cdot 10^{-31}$ kg, y l es la longitud de la caja en m.

Para comprobar el funcionamiento del programa utilice los valores $n = 3$ y $l = 10,0 \text{ \AA} = 1,0010^{-9}$ m.

Solución: Utilizando estos valores se obtiene: $E_1 = 6,02 \cdot 10^{-20}$ J, $E_2 = 24,1 \cdot 10^{-20}$ J, $E_3 = 54,2 \cdot 10^{-20}$ J,

6.5. Funciones intrínsecas de vectores

Existen funciones intrínsecas en Fortran 90 que sirven para realizar operaciones con vectores o matrices. En este apartado se introducirán las funciones que permiten calcular sumatorios y productorios de elementos de un vector.

Cálculo de un sumatorio

Suponga la siguiente operación

$$\text{sumatorio} = \sum_{i=1}^{NDatos} x_i$$

y que los datos están almacenados en el vector x . Se puede calcular el valor del sumatorio de los datos del vector utilizando los siguientes métodos:

i. Utilizando un bucle DO:

```
sumatorio = 0.0
DO i = 1,ndatos,1
    sumatorio = sumatorio + x(i)
END DO
```

ii. Utilizando la función intrínseca **SUM**.

```
Sumatorio = SUM(x(1:ndatos))
```

NOTA: Si el tamaño del vector x es superior al número de elementos utilizados en el programa ($ndatos$), la instrucción **SUM**(x) producirá resultados inesperados si no se ha asignado previamente el valor 0 a los elementos restantes del vector.



Sintaxis de la función intrínseca SUM

SUM(array[, **mask**])

La función intrínseca **SUM** suma los elementos del arreglo. Si incluimos la palabra clave **MASK**, que es opcional, sólo se suman aquellos elementos del vector para los que la condición indicada en **MASK** es verdad. La función también puede utilizar subconjuntos de un arreglo si estos se identifican con una tripleta.

Considere el arreglo x declarado mediante la instrucción,

```
REAL, DIMENSION(1:10) :: x
```

entonces

SUM(x) equivale a $\sum_{i=1}^{10} x_i$, suma todos los elementos del arreglo.

SUM($x(n:m)$) equivale a $\sum_{i=n}^m x_i$

SUM(x , **MASK** = $x < 5$) equivale a $\sum_{i=1}^{10} x_i$, con la condición de que $x_i < 5$

SUM($x(n:m)$, **MASK** = $x < 5$) equivale a $\sum_{i=n}^m x_i$, con la condición de que $x_i < 5$

SUM($x(n:m)$, $x < 5$) equivale a **SUM**($x(n:m)$, **MASK** = $x < 5$)

Ejemplo 6.9 Uso de la función intrínseca SUM.

El siguiente programa muestra cómo se utiliza la función intrínseca **SUM** con vectores para calcular distintos sumatorios.

Con un editor de texto, edite el fichero tarea39.f90:

```
C:\EPAQ\Valvanera> EDIT tarea39.f90 
```

donde introducirá el siguiente programa en Fortran 90:

```
PROGRAM Sumas

IMPLICIT NONE
INTEGER :: i
REAL :: sumatorio
REAL, DIMENSION(1:10) :: x

DO i = 1, 10, 1
  x(i) = REAL(i)
END DO

sumatorio = 0.0
DO i = 1, 10, 1
  sumatorio = sumatorio + x(i)
END DO
WRITE(*,*) " Sumatorio – bucle DO :: ", sumatorio

sumatorio = 0.0
sumatorio = SUM(x)
WRITE(*,*) " Sumatorio – SUM(X) :: ", sumatorio

sumatorio = 0.0
sumatorio = SUM(x(1:10))
WRITE(*,*) " Sumatorio – SUM(X(1:10)) :: ", sumatorio

sumatorio = 0.0
sumatorio = SUM(x(3:10))
WRITE(*,*) " Sumatorio – SUM(X(3:10)) :: ", sumatorio

sumatorio = 0.0
sumatorio = SUM(x, x>2)
WRITE(*,*) " Sumatorio – SUM(X, MASK = X>2) :: ", sumatorio

sumatorio = 0.0
sumatorio = SUM(x(1:10), x>2)
WRITE(*,*) " Sumatorio – SUM(X(1:10),MASK= X>2) :: ", sumatorio

STOP
END PROGRAM Sumas
```

Compile y ejecute el programa.

La ejecución del programa produce el siguiente resultado en la pantalla:

```
C:\EPAQ\Valvanera>tarea39

Sumatorio – bucle DO ::      55.0000
Sumatorio – SUM(X)  ::      55.0000
```

```

Sumatorio - SUM(X(1:10)) ::      55.0000
Sumatorio - SUM(X(3:10)) ::      52.0000
Sumatorio - SUM(X, MASK = X>2) ::      52.0000
Sumatorio - SUM(X(1:10), MASK= X>2) ::      52.0000

Program completed
Press Enter to Continue.

```

Ejercicio 6.2 Vectores ortogonales.

Escriba un programa que lea las coordenadas (x, y, z) de dos vectores y compruebe que los vectores son ortogonales.

Se dice que dos vectores \mathbf{a} y \mathbf{b} son ortogonales si

$$\mathbf{a} \cdot \mathbf{b} = \sum a_i \cdot b_i = 0$$

Como los errores numéricos pueden hacer que para dos vectores ortogonales el sumatorio difiera ligeramente de cero, considere que el producto escalar es cero si el valor del sumatorio es menor en valor absoluto que un número muy pequeño, ε . En el programa utilice `epsilon = 1.0E-12` con este fin.

Para comprobar el funcionamiento del programa utilice los vectores ortogonales $\mathbf{a} = (1, 0, 0)$ y $\mathbf{b} = (0, 1, 0)$. Utilice también como ejemplo los vectores no ortogonales $\mathbf{a} = (1, 1, 1)$ y $\mathbf{b} = (1, 5, 5)$.

Cálculo de un productorio.

Suponga que se necesita hacer la siguiente operación

$$\text{productorio} = \prod_{i=1}^{\text{NDatos}} x_i$$

Si los datos están almacenados en el vector x , el valor de productorio puede calcularse con los siguientes métodos:

i. Utilizando un bucle DO:

```

productorio = 1.0
DO i = 1, ndatos, 1
  productorio = productorio * x(i)
END DO

```

ii. Utilizando la función intrínseca **PRODUCT**.

```
productorio = PRODUCT(x(1:ndatos))
```

NOTA: Si el tamaño del vector x es superior al número de elementos utilizados en el programa (`ndatos`), la instrucción **PRODUCT**(x) producirá resultados inesperados si no se ha asignado previamente el valor 1.0 a los elementos restantes del vector.

Sintaxis de la función intrínseca PRODUCT**PRODUCT**(array[, **mask**])

La función intrínseca **PRODUCT** multiplica los elementos de un vector. Si se incluye la palabra clave **MASK**, que es opcional, sólo se multiplican aquellos elementos del vector para los que la condición indicada en **MASK** es verdad. La función también puede utilizar subconjuntos de un arreglo si estos se identifican con una tripleta.

Considere el arreglo **x** declarado mediante la instrucción,

REAL, DIMENSION(1:10) :: x

entonces

PRODUCT(**x**) equivale a $\prod_{i=1}^{10} x_i$, producto todos los elementos del arreglo.

PRODUCT(**x**(**n:m**)) equivale a $\prod_{i=n}^m x_i$

PRODUCT(**x**, **MASK = x < 5**) equivale a $\prod_{i=1}^{10} x_i$, con la condición de que $x_i < 5$

PRODUCT(**x**(**n:m**), **MASK = x < 5**) equivale a $\prod_{i=n}^m x_i$, con la condición de que $x_i < 5$

PRODUCT(**x**(**n:m**), **x < 5**) equivale a **PRODUCT**(**x**(**n:m**), **MASK = x < 5**)

Ejemplo 6.10 *Uso de la función intrínseca PRODUCT.*

El siguiente programa muestra cómo se utiliza la función **PRODUCT** con vectores para calcular distintos productorios.

Con un editor de texto, edite el fichero tarea40.f90:

```
C:\EPAQ\Valvanera> EDIT tarea40.f90 
```

donde introducirá el siguiente programa en Fortran 90:

```
PROGRAM productorios

IMPLICIT NONE
INTEGER :: i
INTEGER :: producto
INTEGER, DIMENSION(1:10) :: x

DO i = 1, 10, 1
  x(i) = REAL(i)
END DO
```

```

producto = 1.0
DO i = 1, 10, 1
  producto = producto * x(i)
END DO
WRITE(*,*) " producto – bucle DO :: ", producto

producto = PRODUCT(x)
WRITE(*,*) " producto – PRODUCT(x) :: ", producto

producto = PRODUCT(x(1:10))
WRITE(*,*) " producto – PRODUCT(x(1:10)) :: ", producto

producto = PRODUCT(x(3:10))
WRITE(*,*) " producto – PRODUCT(x(3:10)) :: ", producto

producto = PRODUCT(x, MASK = x>2)
WRITE(*,*) " producto – PRODUCT(x, MASK = x>2) :: ", producto

producto = PRODUCT(x(1:10), x>2)
WRITE(*,*) " producto – PRODUCT(x(1:10),MASK= x>2) :: ", producto

STOP
END PROGRAM productorios

```

Compile y ejecute el programa.

La ejecución del programa produce el siguiente resultado en la pantalla:

```

C:\EPAQ\Valvanera> tarea40

producto - bucle DO ::          3628800
producto - PRODUCT(x) ::        3628800
producto - PRODUCT(x(1:10)) ::   3628800
producto - PRODUCT(x(3:10)) ::   1814400
producto - PRODUCT(x, MASK = x>2) :: 1814400
producto - PRODUCT(x(1:10),MASK= x>2) :: 1814400

Program completed
Press Enter to Continue.

```

Ejercicio 6.3 Operaciones con vectores.

Considere dos vectores **a** y **b** de componentes

$$\mathbf{a} = (a_1, a_2, \dots, a_n)$$

$$\mathbf{b} = (b_1, b_2, \dots, b_n)$$

Escriba un programa que:

i Lea la dimensión de los vectores $n \leq 5$ y sus componentes.

ii Calcule la suma de los vectores.

$$\mathbf{c} = \mathbf{a} + \mathbf{b} = (a_1 + b_1, a_2 + b_2, \dots, a_n + b_n)$$

iii Calcule su producto escalar

$$d = \mathbf{a} \cdot \mathbf{b} = (a_1 * b_1 + a_2 * b_2 + \dots + a_n * b_n)$$

iii Calcule su norma

$$|\mathbf{a}| = \sqrt{\mathbf{a} \cdot \mathbf{a}} = \sqrt{a_1 * a_1 + a_2 * a_2 + \dots + a_n * a_n}$$

$$|\mathbf{b}| = \sqrt{\mathbf{b} \cdot \mathbf{b}} = \sqrt{b_1 * b_1 + b_2 * b_2 + \dots + b_n * b_n}$$

Compruebe el funcionamiento del programa con los vectores $\mathbf{a} = (1, 0, 1)$ y $\mathbf{b} = (0, 1, 0)$.

Ejercicio 6.4 Cálculo de la entropía de una mezcla ideal.

Escriba un programa que calcule la variación de entropía de mezcla dado el número de moles de cada especie.

La entropía de mezcla viene dada por

$$\Delta_m S = -R \sum_i X_i \ln X_i$$

donde X_i es la fracción molar de la especie i en la mezcla final.

Calcule la variación de entropía para una mezcla de los gases A (5 moles), B (2 moles), C (0.7 moles) y D (5 moles).

Ejercicio 6.5 Geometrías moleculares.

Escriba un programa que lea las coordenadas cartesianas (x, y, z) en el sistema de centro de masas de los núcleos de una molécula triatómica ABC, calcule las distancias internucleares r_{AB} , r_{BC} y r_{AC} , y calcule el ángulo $\theta = \widehat{ABC}$.

Recuerde

$$\mathbf{r}_{ij} = (x_i - x_j, y_i - y_j, z_i - z_j)$$

$$\mathbf{r}_{ij} \cdot \mathbf{r}_{jk} = r_{ij} \cdot r_{jk} \cdot \cos \theta$$

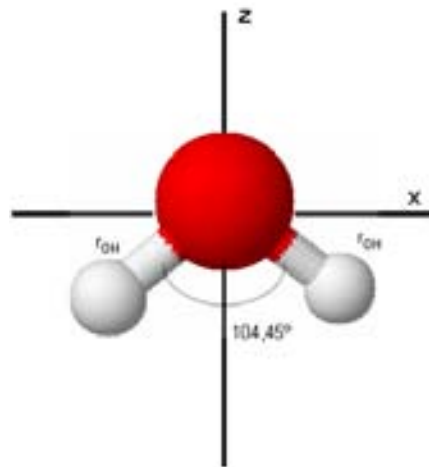
Para comprobar el funcionamiento del programa utilice las coordenadas cartesianas de la molécula H_2O (HOH) :

H1: (0.07575, 0, -0.05214)

O: (0, 0, 0.00657)

H2: (-0.07575, 0, -0.05214)

que corresponden a la distancia internuclear $r_{\text{OH}} = 0,09584 \text{ nm}$ y $\theta = 104,45^\circ$.



Otras funciones intrínsecas de vectores

Otras funciones intrínsecas para vectores comunes son:

DOT_PRODUCT(vector1, vector2) calcula el producto escalar de dos vectores, vector1 y vector2
MAXVAL(vector1[,MASK]) devuelve el mayor valor del arreglo vector1
MINVAL(vector1[,MASK]) devuelve el menor valor del arreglo vector1

Práctica 7

Aplicaciones

CONTENIDOS

7.1. Cálculo de la media muestral, la varianza muestral y la varianza de la media muestral . . .	136
7.2. Ajuste de mínimos cuadrados a una recta	137
7.2.1. Regresión lineal simple	137
7.2.2. Ajuste a exponenciales mediante regresión lineal simple	140

7.1. Cálculo de la media muestral, la varianza muestral y la varianza de la media muestral

La media muestral de N medidas de una magnitud X se define como

$$\hat{\mu}_x = \bar{x} = \frac{\sum_{i=1}^N x_i}{N} \quad (7.1)$$

La varianza muestral de los datos viene dada por

$$\hat{\sigma}(x) = s^2(x) = \frac{1}{N-1} \sum_{j=1}^N (x_j - \bar{x})^2 \quad (7.2)$$

mientras que la varianza muestral de las medias muestrales de experimentos con N medidas viene dada por

$$\hat{\sigma}(\bar{x}) = s^2(\bar{x}) = \frac{1}{N(N-1)} \sum_{j=1}^N (x_j - \bar{x})^2 = \frac{s^2(x)}{N} \quad (7.3)$$

Ejercicio 7.1 Cálculo de parámetros muestrales.

Escriba un programa que lea un número arbitrario de datos x_i y calcule los parámetros muestrales media \bar{x} , varianza $s^2(x)$ y varianza de la media $s^2(\bar{x})$, y escriba los resultados en un fichero.

En el programa considere que la lista de datos más grande con la que puede trabajar tiene 100 elementos.

Las tareas que realizará el programa pueden resumirse en

- i Abrir los ficheros `datos.dat` y `media.dat`
- ii Leer del fichero `datos.dat` el número de valores de la lista, `ndatos`
- iii Leer del fichero `datos.dat` los valores $x(1)$ a $x(ndatos)$
- iv Calcular \bar{x} , $s^2(x)$ y $s^2(\bar{x})$
- v Escribir los resultados del cálculo en el fichero `media.dat`

Aplicación: cálculo de la concentración de una disolución

Utilizando el programa que acaba de escribir, calcule \bar{x} , $s^2(x)$ y $s^2(\bar{x})$ para la siguiente serie de medidas de la concentración de una disolución de ácido sulfúrico:

$c(\text{H}_2\text{SO}_4)/\text{mol}\cdot\text{dm}^{-3}$: 0,4311 0,4315 0,4310 0,4313 0,4312 0,4311

7.2. Ajuste de mínimos cuadrados a una recta

7.2.1. Regresión lineal simple

Suponga que tiene un conjunto de n pares de datos (x_i, y_i) , con $i = 1, \dots, n$ y que existe una relación lineal entre ellos

$$y_i = A \cdot x_i + B + \varepsilon_i \quad (7.4)$$

donde A es la pendiente de la recta, B la ordenada en el origen y ε_i es el error aleatorio en cada una de las medidas y_i obtenido cuando la variable controlada X toma el valor x_i .

El método de mínimos cuadrados proporciona un procedimiento para calcular A y B . Como los valores y_i tienen un error aleatorio ε_i , los valores de la pendiente y la ordenada en el origen que se calculan a partir de los datos experimentales, puede que no coincidan con los valores reales A y B . Por eso, los valores calculados con el método de mínimos cuadrados de la pendiente y la ordenada en el origen se nombran como a y b , respectivamente.

En el método de mínimos cuadrados, a y b se calculan como los valores que minimizan la suma de los cuadrados de los residuos:

$$\begin{aligned} R &= \sum_{i=1}^N w_i (y_i - (a \cdot x_i + b))^2 \\ &= \sum_{i=1}^N w_i (y_i - y_{\text{calculado}})^2 \end{aligned} \quad (7.5)$$

donde w_i es un coeficiente que da el peso del punto (x_i, y_i) en el ajuste.

En una regresión lineal simple, $w_i = 1,0$, y R se calcula a partir de la expresión

$$\begin{aligned} R &= \sum_{i=1}^N (y_i - (a \cdot x_i + b))^2 \\ &= \sum_{i=1}^N (y_i - y_{\text{calculado}})^2 \end{aligned} \quad (7.6)$$

Para obtener las expresiones que nos permiten calcular a y b , aplicaremos a R la condición de mínimo

$$\begin{aligned} \frac{\partial R}{\partial a} &= 0 \\ \frac{\partial R}{\partial b} &= 0 \end{aligned} \quad (7.7)$$

De esta forma, se obtiene un sistema de dos ecuaciones con dos incógnitas (a y b) cuya solución es

$$\text{pendiente } a = \frac{N \sum_{i=1}^n (x_i y_i) - \sum_{i=1}^n x_i \sum_{i=1}^n y_i}{\Delta} \quad (7.8)$$

$$o.o \ b = \frac{\sum_{i=1}^n x_i^2 \sum_{i=1}^n y_i - \sum_{i=1}^n x_i \sum_{i=1}^n (x_i y_i)}{\Delta} \quad (7.9)$$

donde

$$\Delta = N \sum_{i=1}^n x_i^2 - \left(\sum_{i=1}^n x_i \right)^2 \quad (7.10)$$

En este procedimiento de ajuste, la varianza s^2 viene dada por

$$s^2 = \frac{R}{N-2} = \frac{\sum_{i=1}^N (y_i - (a \cdot x_i + b))^2}{N-2} \quad (7.11)$$

Por otra parte, la varianza de la pendiente, $s^2(a)$, viene dada por

$$s^2(a) = \frac{s^2 \cdot N}{\Delta} \quad (7.12)$$

y la varianza de la ordenada en el origen, $s^2(b)$, viene dada por

$$s^2(b) = \frac{s^2 \cdot \sum_{i=1}^N x_i^2}{\Delta} \quad (7.13)$$

El coeficiente de correlación lineal, $r(x, y)$, que es una estima del coeficiente de correlación lineal $\rho(x, y)$, es:

$$r = \frac{N \cdot \sum_{i=1}^N x_i y_i - \sum_{i=1}^N x_i \cdot \sum_{i=1}^N y_i}{\left[\left\{ N \cdot \sum_{i=1}^N x_i^2 - \left(\sum_{i=1}^N x_i \right)^2 \right\} \cdot \left\{ N \cdot \sum_{i=1}^N y_i^2 - \left(\sum_{i=1}^N y_i \right)^2 \right\} \right]^{1/2}} \quad (7.14)$$

Si los datos se ajustan a una recta, este valor debe ser próximo a $-1,00$, si la pendiente es negativa, y $1,00$, si la pendiente es positiva.

Ejercicio 7.2 Ajuste de datos a una recta.

Escriba un programa que lea un número arbitrario de pares de datos x_i, y_i , y calcule su recta de ajuste, el coeficiente de correlación lineal r , la varianza s^2 y las varianzas de a , $s^2(a)$, y b , $s^2(b)$.

Considere que el vector más grande con el que puede trabajar en el programa tiene 100 elementos.

Las tareas que realizará el programa pueden resumirse en

i Abrir los ficheros datos.dat, resul.dat, y tabla.dat

- ii Leer del fichero `datos.dat` el número de valores guardado en ese fichero, `ndatos`
- iii Leer del fichero `datos.dat` los valores $x(i)$ e $y(i)$ con $i = 1, 2, \dots, \text{ndatos}$
- iv Calcular $a, b, s^2, s^2(a), s^2(b)$ y r
- v Escribir $a, b, s^2, s^2(a), s^2(b)$ y r en el fichero `resul.dat`
- vi Escribir $x(i), y(i), a \cdot x(i) + b, y(i) - (a \cdot x(i) + b)$ en el fichero `tabla.dat`

Como este programa es largo y complejo se recomienda que se divida el proceso de escritura y testeo del programa en varias etapas, de complejidad creciente, creando tres versiones del programa:

versión 1 En la versión 1 del programa, se realizan las siguientes tareas:

- Leer de `datos.dat` : `np` y la tabla de valores de x_i e y_i
- Escribir en `resul.dat` el valor de `np`
- Escribir en `tabla.dat` los valores de x_i e y_i

versión 2 En la versión 2 del programa, se realizan las siguientes tareas:

- Leer de `datos.dat` : `np` y la tabla de valores de x_i e y_i
- Escribir en `resul.dat`, `np`
- Calcular a y b
- Escribir en `resul.dat` los valores de a y b
- Escribir en `tabla.dat` los valores de $x_i, y_i, y_{\text{calculado}}, y_i - y_{\text{calculado}}$

versión 3 En la versión 3 del programa, se realizan las siguientes tareas:

- Leer de `datos.dat` : `np` y la tabla de valores de x_i e y_i
- Escribir en `resul.dat`, `np`
- Calcular a y b
- Escribir en `resul.dat` los valores de a y b
- Escribir en `tabla.dat` los valores de $x_i, y_i, y_{\text{calculado}}, y_i - y_{\text{calculado}}$
- Calcular $s^2, s^2(a), s^2(b), r$
- Escribir en `resul.dat` los valores de $s^2, s^2(a), s^2(b), r$

Aplicación. Cálculo de una recta de calibrado

Utilizando el programa que acaba de escribir, calcule los parámetros de la recta de calibrado que ajusta el conjunto de medidas de la tabla.

x	4,0	8,0	12,5	16,0	20,0	25,0	31,0	35,0	40,0	40,0
y	3,7	7,8	12,1	15,6	19,8	24,5	31,1	35,5	39,4	39,5

Para que pueda comprobar que el programa funciona correctamente se detallan a continuación los resultados intermedios del cálculo de los sumatorios que aparecen en las ecuaciones de mínimos cuadrados:

$$\sum_{i=1}^n x_i = 231,5 \quad \sum_{i=1}^n y_i = 229,0 \quad \sum_{i=1}^n x_i y_i = 6849,15$$

$$\sum_{i=1}^n x_i^2 = 6903,25 \quad \sum_{i=1}^n y_i^2 = 6796,66$$

$$\Delta = 10 \times (6903,25) - 231,5^2 = 15440,25$$

y a, b, varianzas y coeficiente de correlación lineal:

$$a = \frac{10 \cdot (6849,15) - 231,5 \cdot 229,0}{15440,25} = 1,00245$$

$$b = \frac{6903,25 \cdot 229,0 - 231,5 \cdot 6849,15}{15440,25} = -0,306605$$

$$s^2 = 0,121971$$

$$s^2(a) = \frac{0,121974 \cdot 10}{15440,25} = 7,899742E - 5 \rightarrow s(a) = 8,888049E - 3$$

$$s^2(b) = \frac{0,121974 \cdot 6903,25}{15440,25} = 0,054533 \rightarrow s(b) = 0,2335$$

$$r = \frac{10 \cdot 6849,15 - 231,5 \cdot 229,0}{[\{10 \cdot 6903,25 - 231,5^2\} \cdot \{10 \cdot 6796,66 - 229,0^2\}]^{1/2}} = 0,99969$$

7.2.2. Ajuste a exponenciales mediante regresión lineal simple

Es muy frecuente que la ecuación que describe el comportamiento de los datos experimentales no sea una línea recta. Un tipo de ajuste que se realiza con mucha frecuencia, es un ajuste no lineal a una función exponencial

$$y_i = b' \exp [a \cdot x_i]$$

Sin embargo, otra forma de obtener los valores de los coeficientes a y b' de la función anterior, es ajustar los logaritmos neperianos de la variable dependiente y_i a una recta según la relación

$$y'_i = \ln(y_i) = ax_i + \ln b' = ax_i + b$$

donde $b = \ln b'$.

Ejercicio 7.3 Ajuste de datos a una recta (II).

Escriba un programa que lea un número arbitrario de pares de datos x_i , y_i , y calcule su recta de ajuste, el coeficiente de correlación lineal r , la varianza s^2 y las varianzas de a , $s^2(a)$, y b , $s^2(b)$.

El programa debe considerar la posibilidad de que los datos se ajusten a una expresión lineal

$$y_i = ax_i + b$$

o exponencial

$$y_i = b' \exp [a \cdot x_i]$$

En este segundo caso, antes de realizar el ajuste, el programa debe realizar la transformación

$$y'_i = \ln(y_i) = ax_i + \ln b' = ax_i + b$$

Considere que el vector más grande con el que puede trabajar en el programa tiene 100 elementos.

Observe que este programa es una versión más complicada del programa del ejercicio 7.2. Los pasos que se espera que tenga el programa pueden resumirse en

- i Abrir los ficheros `datos.dat`, `resul.dat` y `tabla.dat`
- ii Leer del fichero `datos.dat` el número de valores guardado en ese fichero, np
- iii Leer del fichero `datos.dat` el tipo de ajuste (i_w). Si $i_w = 0$ se tiene un ajuste lineal, mientras que si $i_w = 1$ se tiene un ajuste a una exponencial.
- iv Leer del fichero `datos.dat` los valores de los valores x_i e y_i con $i = 1, 2, \dots, np$.
- v Si $i_w = 1$, calcular $y(i) = \text{LOG}(y(i))$
- vi Calcular $a, b, s^2, s^2(a), s^2(b)$ y r
- vii Escribir en el fichero `resul.dat` $a, b, s^2, s^2(a), s^2(b)$ y r
- viii Si $i_w = 1$, escribir en el fichero `resul.dat` $b' = e^b$
- ix Escribir en el fichero `tabla.dat` $x(i), y(i), a \cdot x(i) + b, y(i) - (a \cdot x(i) + b)$
- x Escribir en `tabla.dat` los valores de $x_i, y_i, y_{\text{calculado}}, y_i - y_{\text{calculado}}$

Aplicación: Cálculo de la entalpía de vaporización del agua pura

Para el equilibrio de vaporización del agua pura se obtuvieron los siguientes valores de P y T.

T / °C	78,4	81,5	84,1	87,0	89,4	92,1	94,0	96,2	98,1
P / mmHg	328,3	374,3	418,3	454,3	514,3	570,3	628,3	660,3	726,3

Ajuste los datos a la ecuación de Clausius-Clapeyron integrada (ver ejemplo 6.7) y obtenga la entalpía molar de vaporización del agua.

Práctica 8

Variables subindicadas (II)

CONTENIDOS

8.1. Variables subindicadas bidimensionales: matrices o tablas	144
8.2. Cómo se declara una variable subindicada	146
8.3. Lectura y escritura de matrices	147
8.4. Cómo utilizar matrices en instrucciones en Fortran	159
8.5. Aplicaciones	163
8.6. Funciones intrínsecas de matrices	166

8.1. Variables subindicadas bidimensionales: matrices o tablas

En la práctica 6 se introdujo el uso de arreglos unidimensionales que permiten manipular una lista de valores en Fortran 90. En los arreglos unidimensionales, que también se denominan *vectores* o *listas*, cada elemento del arreglo se identifica por un índice. Por ejemplo, en un arreglo real x con diez elementos designados $x(1)$, $x(2)$, ..., $x(10)$, el elemento $x(5)$ corresponde al quinto elemento de la lista.

En Fortran 90 es posible definir arreglos en los que sus elementos se identifican con dos o más índices. En esta práctica se introducirá el uso de los arreglos con dos índices que se denominan *arreglos bidimensionales*, *tablas* o *matrices*¹.

Los arreglos bidimensionales son útiles para representar datos que se pueden presentar en tablas,

x / y	y_1	y_2	y_3	y_4	
x_1	$a(1,1)$	$a(1,2)$	$a(1,3)$	$a(1,4)$	← fila 1
x_2	$a(2,1)$	$a(2,2)$	$a(2,3)$	$a(2,4)$	← fila 2
x_3	$a(3,1)$	$a(3,2)$	$a(3,3)$	$a(3,4)$	← fila 3

↑ ↑ ↑ ↑
 columna 1 columna 2 columna 3 columna 4

o para representar matrices

a_{11}	a_{12}	a_{13}	← fila 1
a_{21}	a_{22}	a_{23}	← fila 2
a_{31}	a_{32}	a_{33}	← fila 3
a_{41}	a_{42}	a_{43}	← fila 4

↑ ↑ ↑
 columna 1 columna 2 columna 3

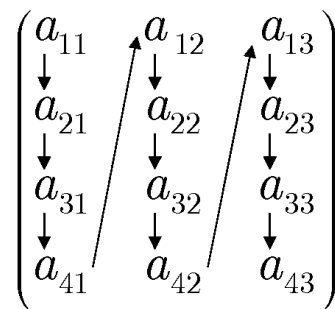
Cada elemento de la tabla o matriz se identifica con dos índices que hacen referencia a la fila (primer subíndice) y la columna (segundo subíndice). Por ejemplo en la matrix anterior, a_{32} corresponde al elemento que se encuentra en la tercera fila y segunda columna.

Considere un arreglo bidimensional A de 4 filas y 3 columnas (4x3). Este arreglo tiene $4 \times 3 = 12$ elementos, y cada elemento se identifica con los valores de los índices de la fila y la columna que ocupa en la matriz/tabla correspondiente al arreglo. Los elementos del arreglo se nombrarán

¹Desde el punto de vista matemático, se puede definir una matriz como una tabla de números ordenados, que se distinguen por la fila y la columna en la que se sitúan.

a(1,1) a(1,2) a(1,3) a(1,4)
 a(2,1) a(2,2) a(2,3) a(2,4)
 a(3,2) a(3,2) a(3,3) a(3,4)
 a(4,4) a(4,2) a(4,3) a(4,4)

En general, los elementos de una variable bidimensional se denotan como una tabla o una matriz. Sin embargo, el ordenador no almacena los datos ni como tabla ni como matriz; el ordenador almacena los datos como una cadena de celdas de memoria. En el ejemplo anterior, la matriz A se almacena como una cadena de 12 celdas de memoria, ordenadas según se lee la matriz por columnas,



Por lo tanto, en este conjunto de celdas de memoria, primero varía el primer índice y después varía el segundo.

memoria	matriz
A(1,1)	$\begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \\ a_{41} & a_{42} & a_{43} \end{pmatrix}$
A(2,1)	
A(3,1)	
...	
A(3,3)	
A(4,3)	

Para definir un arreglo bidimensional es necesario conocer:

- i El tipo de datos.
- ii El nombre del arreglo.
- iii El número de subíndices del arreglo: dos.

- iv Los valores posibles de los subíndices del arreglo. Los posibles valores de los subíndices determinan la extensión (*extent*) de cada dimensión del arreglo. El rango y la extensión definen la forma (*shape*) del arreglo. El número máximo de valores o tamaño (*size*) de la lista viene determinado por la forma y la extensión de este.

El número de subíndices del arreglo es una propiedad que se denomina como el rango (*rank*) del arreglo. La posibles valores de cada índice determinan la extensión (*extent*) de cada dimensión de un arreglo. El rango y la extensión definen la forma (*shape*) del arreglo. En un arreglo bidimensional, la forma se representa con un vector cuyo primer elemento es el número de filas y el segundo el número de columnas (n,m). Esta propiedad se utiliza internamente para determinar si las operaciones que implican a dos o más arreglos son posibles. Por ejemplo, en la suma de vectores, ambos tienen que tener la misma extensión o en la multiplicación de matrices donde el número de columnas de la primera debe coincidir con el número de filas de la segunda. Finalmente, el tamaño del arreglo corresponde al número de elementos que lo componen, que en un arreglo bidimensional es $N \times M$.

8.2. Cómo se declara una variable subindicada

Para definir un arreglo necesitamos conocer su *rango* - el número de índices-. En el caso de un vector el rango es 1 mientras que en una matriz el rango es 2. También hay que determinar la extensión de cada índice, es decir, el número de filas y columnas de la matriz. En una matriz el primer índice corresponde a la fila, mientras que el segundo corresponde a la columna. Para indicar las dimensiones de una matriz una notación útil es $N \times M$ donde N es el número de filas y M el número de columnas. Conviene recordar que el número de elementos que puede almacenar una matriz, el tamaño (*size*), es $N \times M$.

Considere la instrucción de declaración de la matriz `matriz_z` de $N = 3$ filas y $M = 4$ columnas,

```
REAL, DIMENSION(1:3,1:4) :: matriz_z
```

Como se puede observar, además de declarar el tipo de datos que almacenará, con el atributo **DIMENSION** se especifican los valores iniciales y finales, separados por `:`, de los índices de la fila y de la columna. La matriz declarada se llama `matriz_z` y tiene $3 \times 4 = 12$ elementos.

Si el valor inicial de los índices es 1, se pueden omitir en la declaración del arreglo. Así, `matriz_z` podría declararse

```
REAL, DIMENSION(3,4) :: matriz_z
```

Ejemplo 8.1 *Indique el rango, forma y tamaño del arreglo.*

```
INTEGER, DIMENSION(0:5,1:3) :: prueba
```

El arreglo `prueba` está declarado como un arreglo bidimensional y por lo tanto su rango es 2. La primera dimensión tiene de extensión 6 mientras que la segunda es 4, de modo que su forma es (6,4) y su tamaño $6 \times 4 = 24$.

Pregunta 8.1 Indique el rango, forma y tamaño de los arreglos:

- a. **REAL, DIMENSION**(-10:10,0:4) :: datos
- b. **REAL, DIMENSION**(1:3,1:3) :: a
- c. **CHARACTER, DIMENSION**(1:10,-1:1) :: tipo
- d. **REAL, DIMENSION**(1,2) :: error

Sintaxis de declaración de variables subíndicadas: matrices o tablas

TIPO, **DIMENSION**([Ninicial]:Nfinal,[Minicial]:Mfinal) :: Variable1 [, Variable2, ...]

Para declarar una matriz, hay que indicar junto al tipo de variable el atributo **DIMENSION**. A continuación, entre paréntesis y separadas por una coma, se introducen el número de filas y el número de columnas.

El número de filas se indica con dos números Ninicial y Nfinal separados por dos puntos ":" que corresponden al primer valor y al último valor del índice de las filas. Analogamente, se define el número de columnas con los valores Minicial y Mfinal del índice de las columnas, separados por ":".

Los valores de los índices son arbitrarios, pero siempre Nfinal > Ninicial y Mfinal > Minicial. Si se suprime Ninicial o Minicial, el compilador asignará automáticamente el valor 1.

Ejemplos,

```
REAL, DIMENSION(1:100, 1:100) :: muestra
INTEGER, DIMENSION(0:5,1:3) :: prueba
CHARACTER(LEN=3), DIMENSION(1:100,-1:1) :: transicion
```

8.3. Lectura y escritura de matrices

En Fortran 90, se dispone de distintos métodos para introducir valores en variables bidimensionales (asignar valores con instrucciones de lectura) y escribir los valores guardados en una matriz.

i. Lectura y escritura de elementos aislados de una matriz

Suponga que en un programa se han definido la variable escalar x, el vector t(1:3) y la matriz a(1:3,1:3).

La instrucción

```
READ(*,*) x, t(3), a(2,1)
```

lee el valor de x, el del elemento t(3) del vector t y el elemento a(2,1) de la matriz a. Una instrucción semejante se utiliza para escribir los valores guardados en x, t(3) y a(2,1)

```
WRITE(*,*) x, t(3), a(2,1)
```

Con este método, si se quisieran leer (o escribir) todos o una parte (por ejemplo, una fila o una columna) de los elementos de una matriz se tendrían que indicar explícitamente todos los elementos de la matriz. Por ejemplo, si se quieren introducir valores para la matriz $a(1:3,1:3)$, que consta de 9 elementos, se puede utilizar la instrucción:

```
READ(*,*) a(1,1), a(1,2), a(1,3), a(2,1), a(2,2), a(2,3), a(3,3)
```

Evidentemente este método no es muy conveniente para trabajar con matrices.

ii. Lectura y escritura de elementos de un matriz utilizando bucles DO

Si una matriz a consta de $N \times M$ elementos, con índices $n = 1 - n_{\max}$ y $m = 1 - m_{\max}$, se pueden introducir los valores de la matriz utilizando el siguiente fragmento de programa

```
DO n = 1, nmax, 1
  DO m = 1, mmax, 1
    READ(*,*) a(n,m)
  END DO
END DO
```

Con un código similar, se pueden leer valores para los elementos de una parte de la matriz cuyos índices n y m varían entre $(n1 - n2)$ y $(m1 - m2)$

```
DO n = n1, n2, 1
  DO m = m1, m2, 1
    READ(*,*) a(n,m)
  END DO
END DO
```

Utilizando este bloque, en cada ejecución del bucle DO interno se lee un valor y, por lo tanto, se tienen que introducir los valores de la matriz de uno en uno en el teclado. Si los datos están en un fichero, cada dato debe ocupar una línea.

Estas instrucciones también se pueden utilizar para escribir. Continuando con los ejemplos anteriores, para escribir los valores almacenados en la matriz $a(1:3,1:3)$ se puede escribir

```
DO n = 1, nmax, 1
  DO m = 1, mmax, 1
    WRITE(*,*) a(n,m)
  END DO
END DO
```

Para escribir una parte de esta matriz,

```
DO n = n1, n2, 1
  DO m = m1, m2, 1
    WRITE(*,*) a(n,m)
  END DO
END DO
```

En estos casos, se escribiría una línea en cada ejecución del bucle y contendría el valor almacenado en el elemento correspondiente del arreglo a .

iii. Lectura y escritura de elementos de una matriz utilizando bucles DO implícitos

Al igual que se pueden leer o escribir los elementos de un vector utilizando un bucle DO implícito, estos bucles también se pueden utilizar para leer o escribir los elementos de una matriz.

Los bucles DO implícitos también se pueden anidar unos dentro de otros. Así, los elementos de la matriz $a(1:5,1:5)$ se pueden imprimir utilizando la instrucción,

```
WRITE (*,*) ((a(i,j), j= 1, 5, 1), i=1,5,1)
```

En este caso, los valores de todos los elementos de la matriz se escribirán una fila detrás de otra en una línea de impresión. Si no caben todos, continuará la impresión en la siguiente línea. Si $a(i,j)$ se sustituye por $a(j,i)$ en la instrucción, los valores de los elementos de la matriz se escribirán empezando por la primera columna y terminando por la última.

Si se ejecuta la instrucción

```
WRITE(* ,*) (a(i,i), i= 1, 5, 1)
```

sólo se escriben los valores almacenados en la diagonal principal, y si se ejecuta la instrucción

```
WRITE (*,*) (a(1,j), j= 1, 5, 1)
```

se escribirán los valores almacenados en la primera fila.

Si la instrucción anterior se introduce en un bucle DO iterativo en el que se varía el índice de las filas

```
DO i = 1, 5, 1
    WRITE (*,*) (a(i,j), j= 1, 5, 1)
END DO
```

se imprimirá una fila de la matriz por cada iteración del bucle DO y los datos formarán una tabla.

Sintaxis de la lectura y escritura de matrices con bucles DO implícitos anidados

```
READ(unit,f) ((variables, indice1 = min1, max1, inc1), indice2 = min2, max2, inc2)
WRITE(unit,f) ((variables, indice1 = min1, max1, inc1), indice2 = min2, max2, inc2)
```

A diferencia de las operaciones de lectura con bucles DO iterativos anidados, en un bucle DO implícito se intentan leer todos los datos en una línea de pantalla o de fichero, y si no se encuentran suficientes datos, se continúa en la línea siguiente. Cuando se utilizan bucles DO iterativos anidados se realiza una lectura en cada ejecución del bucle interno. En escritura sin formato, con un bucle DO implícito anidado, se intentan escribir todos los datos en una línea de pantalla o de fichero y si no caben la impresión continúa en la línea siguiente. Cuando se utilizan bucles DO iterativos anidados, en cada ejecución del bucle interno se imprime una línea nueva.

Por ejemplo,

```
READ(* ,*) ((a(i,j), j= 1, n, 1), i = 1, m, 1)
WRITE(* ,*) ((a(i,j), j= 1, n, 1), i = 1, m, 1)
```



iv. Escritura o lectura de todos los elementos de una matriz sin utilizar bucles DO

En Fortran 90, basta con escribir el nombre de la matriz en las instrucciones **READ** o **WRITE** para que se puedan leer o escribir todos los elementos de la matriz. Por ejemplo,

```
READ(*,*) a
WRITE(*,*) a
```

Sin embargo, a la hora de utilizar esta instrucción es importante tener en cuenta que el procesador almacena en la memoria los datos de la matriz ordenados por columnas (ver página 145) y, por lo tanto, los datos se leerán o escribirán de acuerdo con este orden.

v. Lectura y escritura de elementos de un subconjunto de un arreglo utilizando tripletas

En la página 113 de la práctica 6 se introdujeron los subconjuntos de un arreglo y cómo se podían utilizar en las operaciones de lectura y escritura de vectores.

La definición de un subconjunto de un arreglo bidimensional es similar a la de un vector pero en este caso se necesitan dos tripletas, una para cada dimensión del arreglo para definir los elementos seleccionados.

Para mostrar el uso de las tripletas se pueden revisar algunos ejemplos prototípicos. En todos los ejemplos se utilizará la matriz $a(1:3,1:3)$.

La instrucción

```
WRITE(*,*) a(1:3,1:3)
```

equivale a **WRITE(*,*) a**, y en la pantalla se obtendría

```
a(1,1) a(2,1) a(3,1) a(1,2) a(2,2) a(3,2) a(1,3) a(2,3) a(3,3)
```

Si se quisieran imprimir los elementos de la segunda columna de la matriz, se podría escribir la instrucción

```
WRITE(*,*) a(1:3:1,2)
```

que es equivalente a

```
WRITE(*,*) a(:,2)
```

donde se omiten el primero y el último valores del primer índice (por ser sus valores mínimo y máximo) y se omite :1 por ser 1 el incremento en la tripleta. Ejecutando esta instrucción se obtendría en la pantalla:

```
a(1,2) a(2,2) a(3,2)
```

Esta instrucción también se podría haber escrito utilizando el bucle DO implícito

WRITE(*,*) (a(i,2), i = 1, 3, 1).

De forma análoga, para imprimir los elementos de la segunda fila se puede utilizarse la instrucción

WRITE(*,*) a(2,1:3:1)

o

WRITE(*,*) a(2,:)

que imprimirá en la pantalla

a (2 , 1) a (2 , 2) a (2 , 3)

Esta instrucción equivale a utilizar el bucle DO implícito **WRITE**(*,*) (a(2,i), i = 1, 3, 1).



Errores de programación. El uso de tripletas permite simplificar la codificación de algunas instrucciones, sin embargo, se deben utilizar con mucho cuidado ya que es muy fácil realizar una selección equivocada de los elementos y por lo tanto obtener resultados erróneos.

Ejemplo 8.2 Escritura de una matriz con bucles DO iterativos.

Este ejemplo muestra la utilización de los bucles DO anidados para escribir todos los valores almacenados en una matriz.

Con un editor de texto, edite el fichero tarea41.f90:

C:\EPAQ\Valvanera> EDIT tarea41.f90 

donde introducirá el siguiente programa en Fortran 90:

```
PROGRAM EjMat1
!
! Este programa muestra como pueden utilizarse los bucles DO
! para escribir los elementos de una matriz
!
IMPLICIT NONE
INTEGER, PARAMETER :: Nmax =2, Mmax =2
INTEGER :: N, M
REAL , DIMENSION (1:NMax,1:MMax) :: A

A(1,1) = 1.0
A(1,2) = 2.0
A(2,1) = 3.0
A(2,2) = 4.0
```

```

DO N=1,Nmax
  DO M=1,MMax
    WRITE(*,*) "Fila ",N, " Columna ",M, " Elemento ",A(N,M)
  END DO
END DO

STOP
END PROGRAM EjMat1

```

Compile y ejecute el programa.

La ejecución del programa dará como resultado en la pantalla:

```

C:\EPAQ\Valvanera> tarea41

Fila          1 Columna          1 Elemento          1.00000
Fila          1 Columna          2 Elemento          2.00000
Fila          2 Columna          1 Elemento          3.00000
Fila          2 Columna          2 Elemento          4.00000

Program completed
Press Enter to Continue.

```

Como se puede observar, el programa imprime una línea con cada elemento de la matriz.

Ejemplo 8.3 *Escritura de un subconjunto de elementos de una matriz con bucles DO iterativos.*

Este ejemplo muestra la utilización de los bucles DO para escribir los valores almacenados en una fila o una columna de una matriz.

Con un editor de texto, edite el fichero tarea42.f90:

```

C:\EPAQ\Valvanera> EDIT tarea42.f90 

```

donde introducirá el siguiente programa en Fortran 90:

```

PROGRAM EjDOMat2
IMPLICIT NONE
INTEGER, PARAMETER :: nmax =2, mmax =2
INTEGER :: n, m
REAL , DIMENSION (1:nmax,1:mmax) :: a
!
! Este programa muestra como pueden utilizarse lo bucles DO
! pra escribir elementos de una matriz
!
a(1,1) = 1.0
a(1,2) = 2.0
a(2,1) = 3.0

```

```
a(2,2) = 4.0
```

```
! Escribe los valores almacenados en la primera columna
```

```
m = 1
```

```
WRITE(*,*) " Escribe los valores de la columna ", m
```

```
DO n=1, nmax, 1
```

```
  WRITE(*,*) "Fila ", n, " Columna ", m, " Elemento ", a(n,m)
```

```
END DO
```

```
! Escribe los valores almacenados en la primera fila
```

```
n = 1
```

```
WRITE(*,*) " Escribe los valores de la fila ", n
```

```
DO m=1, mmax
```

```
  WRITE(*,*) "Fila ", n, " Columna ", m, " Elemento ", a(n,m)
```

```
END DO
```

```
STOP
```

```
END PROGRAM EjDOMat2
```

Compile y ejecute el programa.

La ejecución del programa dará como resultado en la pantalla:

```
C:\EPAQ\Valvanera> tarea42

  Escribe los valores de la columna          1
Fila          1 Columna          1 Elemento    1.00000
Fila          1 Columna          1 Elemento    3.00000
  Escribe los valores de la fila           1
Fila          1 Columna          1 Elemento    1.00000
Fila          1 Columna          2 Elemento    2.00000

Program completed
Press Enter to Continue.
```

Ejemplo 8.4 Escritura de elementos de una matriz con bucles DO implícitos.

Este ejemplo muestra la utilización de los bucles DO implícitos para escribir los valores almacenados en una matriz.

Con un editor de texto, edite el fichero tarea43.f90:

```
C:\EPAQ\Valvanera> EDIT tarea43.f90 
```

donde introducirá el siguiente programa en Fortran 90:

```
PROGRAM EjMat3
IMPLICIT NONE
INTEGER, PARAMETER :: nmax =3, mmax =3
```

```

INTEGER :: n, m
REAL , DIMENSION (1:nmax,1:mmax) :: a

! Este programa muestra como pueden utilizarse lo bucles DO implícitos
! para escribir los elementos de una matriz

a(1,1) = 1.0
a(1,2) = 2.0
a(1,3) = 3.0
a(2,1) = 4.0
a(2,2) = 5.0
a(2,3) = 6.0
a(3,1) = 7.0
a(3,2) = 8.0
a(3,3) = 9.0

! Escribe los valores almacenados en la matriz con bucles DO implícitos anidados
WRITE(*,*) " Valores almacenados en la matriz A"
WRITE(*,*)
WRITE(*,*) ((a(n,m), n=1,nmax),m=1,mmax)
WRITE(*,*)
WRITE(*,*)

! Escribe los valores almacenados en la matriz por filas combinando
! un bucle DO y un bucle DO implícito
WRITE(*,*) " Matriz A"
WRITE(*,*)
DO n=1,nmax
  WRITE(*,*) (a(n,m), m=1,mmax)
END DO

STOP
END PROGRAM EjMat3

```

Compile y ejecute el programa.

La ejecución del programa dará como resultado en la pantalla:

```

C:\EPAQ\Valvanera> tarea43

Valores almacenados en la matriz A

      1.00000      4.00000      7.00000      2.00000      5.00000
      8.00000      3.00000      6.00000      9.00000

Matriz A

      1.00000      2.00000      3.00000
      4.00000      5.00000      6.00000
      7.00000      8.00000      9.00000

Program completed
Press Enter to Continue.

```

Ejemplo 8.5 *Lectura y escritura de matrices combinando bucles DO y bucles DO implícitos.*

El programa muestra cómo utilizar una combinación de bucles DO y bucles DO implícitos para leer y escribir los datos de una matriz.

Con un editor de texto, edite el fichero tarea44.f90:

```
C:\EPAQ\Valvanera> EDIT tarea44.f90 
```

donde introducirá el siguiente programa en Fortran 90:

```
PROGRAM EjMat4
IMPLICIT NONE
INTEGER, PARAMETER :: nmax =3, mmax =3
INTEGER :: n, m
REAL , DIMENSION (1:nmax,1:mmax) :: a

! Programa para mostrar el funcionamiento de los ciclos DO
! para leer y escribir los valores guardados en una matriz
!
! Lee los valores a guardar de la matriz por filas .

WRITE(*,*) " Introduce los valores de la matriz 3x3 por filas "
WRITE(*,*) " Despues de introducir cada fila pulse Intro "
WRITE(*,*)
DO n=1,nmax
  READ(*,*) (a(n,m), m=1,mmax, 1)
END DO

! Escribe los valores almacenados en la matriz por filas

WRITE(*,*)
WRITE(*,*) " Escribe la matriz "
WRITE(*,*)
DO n=1,Nmax
  WRITE(*,*) (A(n,m), m=1,mmax)
END DO

STOP
END PROGRAM EjMat4
```

Compile y ejecute el programa.

Utilice como matriz para probar el programa la matriz:

$$\begin{pmatrix} 1,0 & 2,0 & 3,0 \\ 4,0 & 5,0 & 6,0 \\ 7,0 & 8,0 & 9,0 \end{pmatrix}$$

La ejecución del programa dará como resultado en la pantalla:

```
C:\EPAQ\Valvanera> tarea44

Introduce los valores a almacenar en la matriz de 3x3
Despues de introducir cada fila pulse Intro
  1.0 2.0 3.0
  4.0 5.0 6.0
  7.0 8.0 9.0

Escribe la matriz

  1.00000      2.00000      3.00000
  4.00000      5.00000      6.00000
  7.00000      8.00000      9.00000

Program completed
Press Enter to Continue.
```

Ejemplo 8.6 *Lectura y escritura de matrices combinando bucles DO y bucles DO implícitos utilizando ficheros.*

Este ejemplo muestra la utilización de los bucles DO implícitos para leer y escribir valores de una matriz. En este caso, se utiliza un fichero para leer los datos y se escriben los datos en la pantalla.

Con un editor de texto, edite el fichero tarea45.f90:

```
C:\EPAQ\Valvanera> EDIT tarea45.f90 
```

donde introducirá el siguiente programa en Fortran 90:

```
PROGRAM EjMat5
IMPLICIT NONE
INTEGER, PARAMETER :: nmax = 100, mmax = 100
INTEGER :: n, m
INTEGER :: i, j
REAL, DIMENSION (1:nmax,1:mmax) :: a

! Programa para mostrar el funcionamiento de los ciclos DO
! para leer y escribir los valores guardados en una matriz
! Lee los valores de la matriz por filas .

! Abre el fichero donde se almacenan los datos
OPEN (11, FILE="matriz.dat")

READ(11,*) n, m
WRITE(*,*)
WRITE(*,*) "La matriz tiene ", n, " filas y ", m, " columnas"
WRITE(*,*)
```



```

! Lee los valores de la matriz por filas .
  DO i=1, n, 1
    READ(11,*) (a(i,j), j=1, m, 1)
  END DO

! Escribe los valores almacenados en la matriz por filas

  WRITE(*,*)
  WRITE(*,*) " Escribe la matriz  "
  WRITE(*,*)
  DO i= 1, n, 1
    WRITE(*,*) (a(i,j), j=1, m, 1)
  END DO

  CLOSE(11)
  STOP
  END PROGRAM EjMat5

```

Compile y ejecute el programa.

Como en el caso anterior, considere la matriz:

$$\begin{pmatrix} 1,0 & 2,0 & 3,0 \\ 4,0 & 5,0 & 6,0 \\ 7,0 & 8,0 & 9,0 \end{pmatrix}$$

Edite el archivo `matriz.dat`, introduciendo los datos

```

3 3
1.0 2.0 3.0
4.0 5.0 6.0
7.0 8.0 9.0

```

La ejecución del programa dará como resultado en la pantalla:

```

C:\EPAQ\Valvanera> tarea45
La matriz tiene          3 filas y          3 columnas

Escribe la matriz

      1.00000      2.00000      3.00000
      4.00000      5.00000      6.00000
      7.00000      8.00000      9.00000

Program completed
Press Enter to Continue.

```

Pregunta 8.2 Suponga que el fichero `datos.dat` asignado a la unidad 2 contiene los siguientes datos:

11.2	16.5	31.3	3.14	16.0	12.0
1.1	9.0	17.1	11.0	15.0	-1.3
10.0	11.0	12.0	13.0	14.0	5.0
15.1	16.7	18.9	21.1	24.0	-22.2

¿Qué valores se almacenarán en la matriz x después de ejecutar las siguientes instrucciones?. ¿Qué valor guarda el elemento $x(2,4)$?

```
REAL, DIMENSION(1:3,1:5) :: x
...
READ(2,*) x
```

Pregunta 8.3 Considere el fichero `datos.dat` asignado a la unidad 2 de la pregunta anterior.

¿Qué valores se almacenarán en la matriz x después de ejecutar las siguientes instrucciones?. ¿Qué valor guarda el elemento $x(2,4)$?

```
REAL, DIMENSION(0:2,2:6) :: x
...
READ(2,*) x
```

Pregunta 8.4 Considere el fichero `datos.dat` de la pregunta 8.2 asignado a la unidad 2.

¿Qué valores se almacenarán en la matriz x después de ejecutar las siguientes instrucciones?. ¿Qué valor guarda el elemento $x(2,4)$?

```
REAL, DIMENSION(1:3,1:5) :: x
...
READ(2,*) ((x(i,j), j= 1,5,1), i = 1, 3, 1)
```

Pregunta 8.5 Considere el fichero `datos.dat` de la pregunta 8.2 asignado a la unidad 2.

¿Qué valores se almacenarán en la matriz x después de ejecutar las siguientes instrucciones?. ¿Qué valor guarda el elemento $x(2,4)$?

```
REAL, DIMENSION(1:3,1:5) :: x
...
DO i = 1, 3, 1
  READ(2,*) (x(i,j), j= 1, 5, 1)
END DO
```

Ejercicio 8.1 Modifique el programa del ejemplo 8.6 para que imprima los valores de la primera fila en una línea y de la primera columna en forma de columna.

Ejercicio 8.2 Modifique el programa del ejemplo 8.6 para que imprima los elementos de la diagonal principal de la matriz en una fila.

Ejercicio 8.3 Modifique el programa del ejemplo 8.6 para que en la lectura y escritura de datos, en lugar de utilizar un bucle `DO` implícito se utilice una tripleta.

8.4. Cómo utilizar matrices en instrucciones en Fortran

i. Asignación de valores a elementos aislados de una matriz

Cada uno de los elementos de un arreglo se comporta como cualquier otra variable escalar. Por tanto, el arreglo se puede incluir en lugar de cualquier otra variable del mismo tipo. Los elementos de un arreglo se pueden utilizar en expresiones aritméticas y lógicas, y el resultado se puede asignar a un elemento de otro arreglo.

Así, para asignar valores a elementos aislados de una matriz basta con especificar los valores de los índices del arreglo correspondiente,

$$\begin{aligned} a(1,5) &= 3 \\ b(4,2) &= 3 * R - \text{EXP}(x(2)) \end{aligned}$$

ii. Asignación de valores a un subconjunto de elementos de una matriz

La asignación de valores a un subconjunto de elementos de una matriz es similar a la asignación de valores a un subconjunto de elementos de un vector. Sin embargo, en una matriz se pueden variar dos índices, mientras que en un vector sólo se varía uno, y esto hace que el número de posibilidades que se tienen para una matriz sea más amplio.

Utilizando un bucle DO

El bucle DO se puede utilizar para variar el índice de la fila (o la columna), como en el ejemplo

```
REAL, DIMENSION(1:2,1:100) :: conc
...
DO j = 1, 100, 1
  t = REAL(i-1)*inct
  conc(1,j) = c0*EXP(-k*t) ! conc. reactivo
  conc(2,j) = c0 - conc(1,j) ! conc. producto
END DO
```

Utilizando bucles DO anidados

Si el código se escribe utilizando bucles DO anidados, el índice de uno de los bucles se utilizará para cambiar el índice de las columnas de la matriz y el índice del segundo bucle se utilizará para cambiar el índice de las filas. Por ejemplo,

```
DO n = 1, nt, 1
  DO m = 1, mt, 1
    T(n) = T0 + (n-1)*incT
    k(n,m) = A*exp(-Ea/(R*T(n)))
  END DO
END DO
```

Este ejemplo puede simplificarse utilizando tripletas

```
DO n = 1, nt, 1
  T(n) = T0 + (n-1)*incT
  k(n,1:mt:1) = A*exp(-Ea/(R*T(n)))
END DO
```

iii. Operaciones con todos los elementos de una matriz

En Fortran 90, utilizando sólo los nombres de las matrices, se pueden hacer operaciones con sus elementos. Cuando se opera con varias matrices, éstas deben tener la misma forma, es decir, el mismo número de filas y de columnas.

Suponga que se han declarado las matrices x, y, z con la instrucción

```
REAL, DIMENSION(1:5,1:5) :: x, y, z
```

Si se escribe la instrucción

```
x = 1.0
```

se asigna el valor inicial 1.0 a todos los elementos de la matriz x. Esta instrucción equivale a utilizar los bucles

```
DO n = 1, 10, 1
  DO m = 1, 10, 1
    x(n,m) = 1.0
  END DO
END DO
```

Muchas de las funciones intrínsecas de Fortran que se utilizan con valores escalares aceptan arreglos como argumentos y producen otro arreglo como resultado. El arreglo resultante se obtiene evaluando la expresión y aplicando la función elemento a elemento ². Funciones de este tipo son **ABS**, **SIN**, **COS**, **EXP**, **LOG**. Así, la instrucción

```
x = a*LOG(x)
```

donde a es una variable escalar, equivale a

```
DO n = 1, 5, 1
  DO m = 1, 5, 1
    x(n,m) = a*LOG(x(n,m))
  END DO
END DO
```

²Estas funciones intrínsecas, que se evalúan elemento a elemento cuando el argumento es un arreglo, se conocen como *funciones intrínsecas elementales*

Las expresiones con arreglos pueden incluir varias matrices. Para que estas operaciones sean válidas, los arreglos que aparecen en la expresión deben tener la misma forma (*shape*). Esto supone que: (1) deben tener el mismo rango y (2) deben tener el mismo número de elementos en cada dimensión (*extensión*). Cuando dos arreglos cumplen esta condición se dice que son *conformables* (compatibles) y pueden utilizarse en expresiones aritméticas que implican a todos los elementos de ambas matrices.

Así, las siguientes expresiones (y sus equivalentes), que operan con las matrices *x* e *y* declaradas anteriormente (compatibles), son válidas:

- $x = y$
equivale a

```
DO n = 1, 5, 1
  DO m = 1, 5, 1
    x(n,m) = y(n,m)
  END DO
END DO
```

- $z = x + y$
equivale a

```
DO n = 1, 5, 1
  DO m = 1, 5, 1
    z(n,m) = x(n,m) + y(n,m)
  END DO
END DO
```

- $z = x * y$
equivale a

```
DO n = 1, 5, 1
  DO m = 1, 5, 1
    z(n,m) = x(n,m) * y(n,m)
  END DO
END DO
```

NOTA: no debe confundirse con el producto de matrices.

- $z = x * \text{EXP}(y)$
equivale a

```
DO n = 1, 5, 1
  DO m = 1, 5, 1
    z(n,m) = x(n,m)*EXP(y(n,m))
  END DO
END DO
```

Si intentamos operar con matrices que no son compatibles, el compilador producirá un error de compilación.

Ejemplo 8.7 Errores de sintaxis: operaciones con matrices no compatibles.

Considere el siguiente programa

```
PROGRAM noequivalentes2
IMPLICIT NONE
REAL, DIMENSION(1:10,1:10) :: x
REAL, DIMENSION(1:10,1:20) :: z

z = 1.0
x = z

WRITE(*,*) x

STOP
END PROGRAM noequivalentes2
```

Los arreglos x y z tienen distinta extensión en una de sus dimensiones. En el programa se intenta realizar una operación con ambas matrices pero, debido a que las matrices tienen distinta forma, el código no es aceptable. En consecuencia, al compilar el programa, el compilador envía el siguiente mensaje de error

```
Compiling file noconform2.f90.
Compiling program unit NOEQUIVALENTES2 at line 2.
Line 7, file noconform2.f90
    x = z
    |
FATAL -- Array shapes do not conform.

Encountered 1 error, 0 warnings in file noconform2.f90.
```

e interrumpe el proceso de compilación y ensamblaje del programa.



Errores de sintaxis. Cuando se realizan operaciones que implican a todos los elementos de dos o más arreglos, estos deben tener la misma forma. En el caso de matrices (arreglo de rango 2) esto supone que éstas deben tener el mismo número de filas y columnas, no importa que los valores de los índices no coincidan.

Ejemplo 8.8 Matriz identidad.

Escriba un fragmento de programa que calcule la matriz identidad de dimensiones $N \times N$

La matriz identidad es una matriz cuadrada de orden N con todos sus elementos cero excepto aquellos situados en la diagonal principal que valen 1

$$\mathbf{I} = \begin{pmatrix} 1 & 0 & \cdots & 0 \\ 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & \cdots & 1 \end{pmatrix}$$

Teniendo en cuenta la definición de matriz identidad, un código para obtener esta matriz es

```

...
DO j = 1, n, 1
  DO k = 1, n, 1
    IF (j == k) THEN
      I(j,k) = 1.0
    ELSE
      I(j,k) = 0.0
    END IF
  END DO
END DO
...

```

Este código se puede simplificar introduciendo el valor 0 en todos los elementos de la matriz I:

```

...
I = 0.0
DO j = 1, n, 1
  I(j,j) = 1.0
END DO
...

```

o en un subconjunto de ellos:

```

...
I(1:n,1:n) = 0.0
DO j = 1, n, 1
  I(j,j) = 1.0
END DO
...

```

8.5. Aplicaciones

Ejercicio 8.4 *Traza de una matriz.*

Escriba un programa que lea una matriz cuadrada desde un fichero, calcule la traza de esta matriz y escriba en un fichero de resultados la matriz original y su traza.

En el programa, considere que la mayor matriz con la que se puede trabajar tiene 25x25 elementos.

La traza de una matriz cuadrada se define como la suma de los elementos de su diagonal principal. Sea la matriz \mathbf{A} de orden $n \times n$, su traza viene dada por

$$\chi(\mathbf{A}) = \sum_{i=1}^n a_{ii}$$

Las tareas que realizará el programa pueden resumirse en

- i Abrir los ficheros `matriz.dat` y `out.dat`
- ii Leer el número de filas n y el número de columnas m de la matriz del fichero `matriz.dat`
- iii Realizar las siguientes comprobaciones:
 - La matriz es cuadrada: $n == m$. En caso contrario, enviar un mensaje de error explicativo.
 - El tamaño de la matriz: $n \leq 25$ y $m \leq 25$. En caso contrario, enviar un mensaje de error explicativo.
- iv Leer la matriz \mathbf{A} escrita por filas del fichero `matriz.dat`
- v Escribir por filas la matriz \mathbf{A} en el fichero `out.dat`
- vi Calcular la traza de la matriz, $\chi(\mathbf{A})$
- vii Escribir $\chi(\mathbf{A})$ en el fichero `out.dat`

Si utiliza para comprobar el funcionamiento del programa la matriz:

$$\mathbf{A} = \begin{pmatrix} 1,0 & 2,0 & 3,0 & 4,0 \\ 5,0 & 6,0 & 7,0 & 8,0 \\ 9,0 & 10,0 & 11,0 & 12,0 \\ 13,0 & 14,0 & 15,0 & 16,0 \end{pmatrix}$$

obtendrá una traza de

$$\chi(\mathbf{A}) = 1,0 + 6,0 + 11,0 + 16,0 = 34,0$$

Ejercicio 8.5 *Matriz traspuesta.*

Escriba un programa que lea una matriz desde un fichero, calcule la traspuesta de esta matriz y escriba en el fichero de resultados la matriz original y su traspuesta.

En el programa, considere que la mayor matriz con la que se puede trabajar tiene 25x25 elementos.

La matriz traspuesta de la matriz \mathbf{A} de orden $n \times m$ elementos, \mathbf{A}^T , es una matriz de $m \times n$ elementos que vienen dados por

$$a_{ij}^T = a_{ji}$$

Las tareas que realizará el programa pueden resumirse en

- i Abrir los ficheros `matriz.dat` y `out.dat`
- ii Leer el número de filas n y el número de columnas m de la matriz del fichero `matriz.dat`
- iii Realizar las siguientes comprobaciones:

- La matriz es cuadrada: $n == m$. En caso contrario, enviar un mensaje de error explicativo.
 - El tamaño de la matriz: $n \leq 25$ y $m \leq 25$. En caso contrario, enviar un mensaje de error explicativo.
- iv Leer la matriz \mathbf{A} escrita por filas del fichero `matriz.dat`
- v Escribir por filas la matriz \mathbf{A} en el fichero `out.dat`
- vi Calcular la matriz traspuesta, \mathbf{A}^T
- vii Escribir la matriz \mathbf{A}^T en el fichero `out.dat`

Si utiliza para comprobar el funcionamiento del programa la matriz:

$$\mathbf{A} = \begin{pmatrix} 2,0 & 3,0 & 4,0 \\ 3,0 & 4,0 & 5,0 \end{pmatrix}$$

obtendrá la siguiente matriz traspuesta

$$\mathbf{A}^T = \begin{pmatrix} 2,0 & 3,0 \\ 3,0 & 4,0 \\ 4,0 & 5,0 \end{pmatrix}$$

Ejercicio 8.6 *Multiplicación de matrices.*

Escriba un programa que lea dos matrices, compruebe si son conformables, realice su multiplicación y escriba en un fichero de resultados las matrices iniciales y la matriz producto.

En el programa considere que la mayor matriz con la que puede trabajar tiene 25x25 elementos.

Dos matrices \mathbf{A} ($n_1 \times m_1$) y \mathbf{B} ($n_2 \times m_2$) se pueden multiplicar en el orden $\mathbf{A} \times \mathbf{B}$ si son congruentes, es decir, si el número de columnas de la primera, m_1 , coincide con el número de filas de la segunda, n_2 .

La matriz producto $\mathbf{C} = \mathbf{A} \times \mathbf{B}$ será de orden $n_1 \times m_2$ y sus elementos vendrán dados por

$$c_{ij} = \sum_{k=1}^{m_1} a_{ik} \cdot b_{kj}$$

Las tareas que realizará el programa pueden resumirse en

- i Abrir los ficheros `matrices.dat` y `out.dat`
- ii Leer del fichero `matrices.dat`:
 - el número de filas n_1 y el número de columnas m_1 de la matriz \mathbf{A}
 - el número de filas n_2 y el número de columnas m_2 de la matriz \mathbf{B}
- iii Realizar las siguientes comprobaciones:
 - La matrices \mathbf{A} y \mathbf{B} son congruentes: $m_1 == n_2$. En caso contrario, enviar un mensaje de error explicativo.

- Tamaño de la matriz **A**: $n1 \leq 25$ y $m1 \leq 25$. En caso contrario, enviar un mensaje de error explicativo.
 - Tamaño de la matriz **B**: $n2 \leq 25$ y $m2 \leq 25$. En caso contrario, enviar un mensaje de error explicativo.
- iv Leer escritas por filas las matrices **A** y **B** del fichero `matrices.dat`.
- v Escribir por filas las matrices **A** y **B** en el fichero `out.dat`.
- vi Calcular la matriz producto **C**
- vii Escribir por filas la matriz **C** en el fichero `out.dat`.

Si utiliza para comprobar el funcionamiento del programa las matrices:

$$\mathbf{A} = \begin{pmatrix} 2,0 & 3,0 & 4,0 \\ 3,0 & 4,0 & 5,0 \end{pmatrix} \quad \mathbf{B} = \begin{pmatrix} 2,0 & 3,0 \\ 3,0 & 4,0 \\ 4,0 & 5,0 \end{pmatrix}$$

obtendrá la siguiente matriz producto

$$\mathbf{C} = \mathbf{A} \times \mathbf{B} = \begin{pmatrix} 29,0 & 38,0 \\ 38,0 & 50,0 \end{pmatrix}$$

8.6. Funciones intrínsecas de matrices

La función TRANSPOSE

La función intrínseca **TRANSPOSE(B)** obtiene la matriz traspuesta de la matriz **B** sobre la que se aplica la función:

$$\mathbf{B}^T = \mathbf{TRANSPOSE}(\mathbf{B})$$

Si las dimensiones de las matrices **B** o **B^T** en la declaración de las variables son mayores que las utilizadas en la ejecución del programa, se puede utilizar la función **TRANSPOSE** sobre un subconjunto de elementos del arreglo

$$\mathbf{B}^T(1:m, 1:n) = \mathbf{TRANSPOSE}(\mathbf{B}(1:n, 1:m))$$

Ejemplo 8.9 *Uso de la función TRANSPOSE.*

El siguiente programa muestra el uso de la función intrínseca **TRANSPOSE**.

Con un editor de texto, edite el fichero `tarea46.f90`:

```
C:\EPAQ\Valvanera> EDIT tarea46.f90 
```

donde introducirá el siguiente programa en Fortran 90:

```

PROGRAM EjMat5
IMPLICIT NONE
INTEGER, PARAMETER :: nmax = 100, mmax = 100
INTEGER :: n, m
INTEGER :: i, j
REAL ,DIMENSION (1:nmax,1:mmax) :: b, bT

!
! Lee una matriz de dimensiones n x m y obtiene su traspuesta

! Abre el fichero donde se almacenan los datos

OPEN (11, FILE="matriz.dat")

READ(11,*) n, m
WRITE(*,*)
WRITE(*,*) "La matriz tiene ", n, " filas y ", m," columnas"
WRITE(*,*)

! Lee los valores de la matriz por filas .
DO i=1, n, 1
READ(11,*) (b(i,j), j=1, m, 1)
END DO

! Escribe los valores almacenados en la matriz por filas

WRITE(*,*)
WRITE(*,*) " Matriz inicial "
WRITE(*,*)
DO i= 1, n, 1
WRITE(*,*) (b(i,j), j=1, m, 1)
END DO

!
! Calcula y escribe la matriz traspuesta
!

bt(1:m,1:n) = TRANSPOSE(b(1:n,1:m))
WRITE(*,*)
WRITE(*,*) " Matriz traspuesta "
WRITE(*,*)
DO i= 1, m, 1
WRITE(*,*) (bt(i,j), j=1, n, 1)
END DO

CLOSE(11)
STOP
END PROGRAM EjMat5

```

Compile y ejecute el programa.

Utilice como matriz para probar el programa la matriz:

$$\mathbf{B} = \begin{pmatrix} 1,0 & 2,0 & 3,0 \\ 4,0 & 5,0 & 6,0 \\ 7,0 & 8,0 & 9,0 \end{pmatrix}$$

La ejecución del programa dará como resultado en la pantalla:

```
C:\EPAQ\Valvanera> tarea46
La matriz tiene          3 filas y          3 columnas

Matriz inicial
  1.00000    2.00000    3.00000
  4.00000    5.00000    6.00000
  7.00000    8.00000    9.00000

Matriz traspuesta
  1.00000    4.00000    7.00000
  2.00000    5.00000    8.00000
  3.00000    6.00000    9.00000

Program completed
Press Enter to Continue.
```

La función MATMUL

La función intrínseca **MATMUL**(A,B) obtiene la matriz producto de las matrices **A** y **B**

$$\mathbf{C} = \mathbf{MATMUL}(\mathbf{A},\mathbf{B})$$

donde $\mathbf{C} = \mathbf{A} \times \mathbf{B}$

Si en la declaración de las variables **A**, **B** o **C** las dimensiones son mayores que las utilizadas en la ejecución del programa, se puede utilizar la función **MATMUL** sobre subconjuntos de los arreglos

$$\mathbf{C}(1:n1,1:m2) = \mathbf{MATMUL}(\mathbf{A}(1:n1,1:m1),\mathbf{B}(1:n2,1:m2))$$

Recuerde que para que ambas matrices se puedan multiplicar el número de columnas de la matriz **A**, $m1$ debe coincidir con el número de filas de la matriz **B**, $n2$ y el resultado es una matriz de dimensiones $n1 \times m2$.

Ejemplo 8.10 Uso de la función MATMUL.

El siguiente programa muestra el uso de la función intrínseca **MATMUL**.

Con un editor de texto, edite el fichero tarea47.f90:

```
C:\EPAQ\Valvanera> EDIT tarea47.f90 
```

donde introducirá el siguiente programa en Fortran 90:

```

PROGRAM EjMatmul
IMPLICIT NONE
INTEGER, PARAMETER :: nmax =3, mmax =3
INTEGER :: n1, m1, n2, m2
REAL ,DIMENSION (1:nmax,1:mmax) :: a, b, c
INTEGER :: i, j

! Lee la matriz A de dimensiones n1 x m1 y
! B de dimensiones n2 x m2 y obtiene la matriz producto
! C = A x B

OPEN (11, FILE="matrices.dat")

! Lee la matriz A

READ(11,*) n1, m1
WRITE(*,*)
WRITE(*,*) "La matriz A tiene ", n1, " filas y ", m1," columnas"
WRITE(*,*)

DO i=1, n1, 1
  READ(11,*) (a(i,j), j=1, m1, 1)
END DO
WRITE(*,*) " Matriz A"
WRITE(*,*)
DO i= 1, n1, 1
  WRITE(*,*) (a(i,j), j=1, m1, 1)
END DO

! Lee la matriz B

READ(11,*) n2, m2
WRITE(*,*)
WRITE(*,*) "La matriz B tiene ", n2, " filas y ", m2," columnas"
WRITE(*,*)

DO i=1, n2, 1
  READ(11,*) (b(i,j), j=1, m2, 1)
END DO
WRITE(*,*) " Matriz B"
WRITE(*,*)
DO i= 1, n2, 1
  WRITE(*,*) (b(i,j), j=1, m2, 1)
END DO

! Calcula C = A x B, y escribe el resultado

c(1:n1,1:m2) = MATMUL(a(1:n1,1:m1),b(1:n2,1:m2))
WRITE(*,*)
WRITE(*,*) " Matriz producto C = A x B"
WRITE(*,*)
DO i= 1, n1, 1

```

```

WRITE(*,*) (c(i,j), j=1, m2, 1)
END DO

CLOSE(11)
STOP
END PROGRAM EjMatmul

```

Compile y ejecute el programa.

Para comprobar el funcionamiento del programa utilice las siguientes matrices :

$$\mathbf{A} = \begin{pmatrix} 2,0 & 3,0 & 4,0 \\ 3,0 & 4,0 & 5,0 \end{pmatrix} \quad \mathbf{B} = \begin{pmatrix} 2,0 & 3,0 \\ 3,0 & 4,0 \\ 4,0 & 5,0 \end{pmatrix}$$

cuyo producto es

$$\mathbf{C} = \mathbf{A} \times \mathbf{B} = \begin{pmatrix} 29,0 & 38,0 \\ 38,0 & 50,0 \end{pmatrix}$$

La ejecución del programa dará como resultado en la pantalla:

```

C:\EPAQ\Valvanera> tarea47

La matriz A tiene          2 filas y          3 columnas

Matriz A

    2.00000    3.00000    4.00000
    3.00000    4.00000    5.00000

La matriz B tiene          3 filas y          2 columnas

Matriz B

    2.00000    3.00000
    3.00000    4.00000
    4.00000    5.00000

Matriz producto C = A x B

    29.0000    38.0000
    38.0000    50.0000

Program completed
Press Enter to Continue.

```

La función intrínseca SUM

La función intrínseca **SUM** también puede utilizarse con arreglos bidimensionales (y multidimensionales). Sin embargo, el uso de esta función con estos arreglos es más complejo y más expuesto a errores que cuando se usa con vectores, puesto que deben utilizarse dos o más índices para identificar los elementos a sumar. No se discutirá exhaustivamente como utilizar esta función con matrices pero para mostrar su uso, se expondrán algunas operaciones típicas en las que se utiliza la función **SUM**.



1. Sumar todos los elementos de una matriz

Suponga que se quiere hacer la siguiente operación,

$$Sumatorio = \sum_{i=1}^{nmax} \sum_{j=1}^{mmax} x_{ij}$$

en la que los elementos de una matriz se almacenan en un arreglo x de dimensiones $nmax \times mmax$.

La operación puede realizarse:

i Utilizando bucles DO anidados.

```
sumatorio = 0.0
DO i= 1, nmax, 1
  DO j= 1, mmax, 1
    sumatorio = sumatorio + x(i, j)
  END DO
END DO
```

ii Utilizando la función intrínseca **SUM**.

```
sumatorio = SUM(x)
```

iii Utilizando la función intrínseca **SUM** y un subconjunto del arreglo.

```
sumatorio = SUM(x(1:nmax, 1:mmax))
```

2. Sumar una sección de los elementos de una matriz

Suponga que se quiere hacer la siguiente operación,

$$Sumatorio = \sum_{i=n1}^{n2} \sum_{m1}^{m2} x_{ij}$$

en la que los elementos de la tabla se almacenan en un arreglo x de dimensiones $nmax \times mmax$.

La operación puede realizarse:

i Utilizando bucles DO anidados

```
sumatorio = 0.0
DO i= n1, n2, 1
  DO j= m1, m2, 1
    sumatorio = sumatorio + x(i, j)
  END DO
END DO
```

- ii Utilizando la función intrínseca **SUM** y un subconjunto del arreglo.
 sumatorio = **SUM**(x(n1:n2, m1:m2))

3. Sumar los elementos de una fila

Suponga que se quiere sumar los elementos de la fila n de la matriz según

$$Sumatorio = \sum_{j=1}^{mmax} x_{nj}$$

donde los elementos de la matriz se almacenan en un arreglo x de dimensiones nmax × mmax.

La operación puede realizarse:

- i Utilizando un bucle DO
- ```
sumatorio = 0.0
DO j= 1, mmax, 1
 sumatorio = sumatorio + x(n,j)
END DO
```
- ii Utilizando la función intrínseca **SUM** y un subconjunto del arreglo.  
 sumatorio = **SUM**(x(n, 1:mmax))
- que también puede expresarse como  
 sumatorio = **SUM**(x(n, :))

### 4. Sumar los elementos de una columna

Suponga que se quieren sumar los elementos de la columna m de una matriz,

$$Sumatorio = \sum_{i=1}^{nmax} x_{im}$$

donde los elementos de la tabla se almacenan en un arreglo x de dimensiones nmax × mmax.

La operación puede realizarse:

- i Utilizando un bucle DO
- ```
sumatorio = 0.0
DO i= 1, nmax, 1
  sumatorio = sumatorio + x(i,m)
END DO
```
- ii Utilizando la función intrínseca **SUM** y un subconjunto del arreglo.
 sumatorio = **SUM**(x(1:nmax,m))
- que también puede expresarse como
 sumatorio = **SUM**(x(:,m))

Ejemplo 8.11 *La función intrínseca SUM con matrices.*

El siguiente programa muestra cómo funciona la función SUM con matrices, y cómo puede utilizarse para calcular distintos sumatorios.

Con un editor de texto, edite el fichero tarea48.f90:

```
C:\EPAQ\Valvanera> EDIT tarea48.f90 
```

donde introducirá el siguiente programa en Fortran 90:

```
PROGRAM EjMatSum

!
IMPLICIT NONE
INTEGER :: n,m
REAL :: Sumatorio
REAL, DIMENSION(1:2,1:2) :: x

! La matriz es
!   1 1
!   1 1
x = 1.0
WRITE(*,*) " La matriz es "
DO n = 1,2,1
  WRITE(*,*) (x(n,m),m=1,2,1)
END DO

!
! Suma de todos los elementos de la matriz
!
Sumatorio = 0.0
DO n = 1,2,1
  DO m = 1,2,1
    Sumatorio = Sumatorio + x(n,m)
  END DO
END DO
WRITE(*,*) " Suma de todos los elementos de la matriz "
WRITE(*,*) " Bucles DO :: ", Sumatorio
Sumatorio = SUM(x)
WRITE(*,*) " SUM(X) :: ", Sumatorio

!
! Suma de los elementos de la fila 1
!
Sumatorio = 0.0
n = 1
DO m = 1,2,1
  Sumatorio = Sumatorio + x(n,m)
END DO

WRITE(*,*) " Suma de todos los elementos de primera fila "
WRITE(*,*) " Bucles DO :: ", Sumatorio
```

```

Sumatorio = SUM(x(1,1:2))
WRITE(*,*) " Sumatorio – SUM(X(1,1:2)) :: ", Sumatorio
Sumatorio = SUM(x(1,:))
WRITE(*,*) " Sumatorio – SUM(X(1,:)) :: ", Sumatorio

!
! Suma de los elementos de la columna 1
!
Sumatorio = 0.0
m = 1
DO n = 1,2,1
    Sumatorio = Sumatorio + x(n,m)
END DO
WRITE(*,*) " Suma de todos los elementos de primera fila "
WRITE(*,*) " Bucles DO :: ", Sumatorio
Sumatorio = SUM(x(1:2,1))
WRITE(*,*) " Sumatorio – SUM(X(1:2,1)) :: ", Sumatorio
Sumatorio = SUM(x(:,1))
WRITE(*,*) " Sumatorio – SUM(X(:,1)) :: ", Sumatorio

!
! Suma de los elementos de la matriz cuyos valores son
! mayores que 2.
! Esta suma tiene que salir cero porque todos los elementos
! valen 1.0.
!
Sumatorio = SUM(x, MASK = x>2)
WRITE(*,*) " Sumatorio – SUM(x, MASK = X>2) :: ", Sumatorio

STOP
END PROGRAM EjMatSum

```

Compile y ejecute el programa.

La ejecución del programa dará como resultado en la pantalla:

```

C:\EPAQ\Valvanera> tarea48

La matriz es
    1.00000    1.00000
    1.00000    1.00000
Suma de todos los elementos de la matriz
Bucles DO ::      4.00000
SUM(X) ::        4.00000
Suma de todos los elementos de primera fila
Bucles DO ::      2.00000
Sumatorio – SUM(X(1,1:2)) ::      2.00000
Sumatorio – SUM(X(1,:)) ::      2.00000
Suma de todos los elementos de primera fila
Bucles DO ::      2.00000
Sumatorio – SUM(X(1:2,1)) ::      2.00000
Sumatorio – SUM(X(:,1)) ::      2.00000
Sumatorio – SUM(X, MASK = X>2) ::      0.00000

```

```
Program completed  
Press Enter to Continue.
```

La función intrínseca **PRODUCT**

Las mismas consideraciones que se han hecho respecto de la función intrínseca **SUM** pueden hacerse respecto de la función intrínseca **PRODUCT**.



Parte II

Funciones y subrutinas

Práctica **9**

Subprogramas: funciones

CONTENIDOS

9.1. Subprogramas	180
9.2. Ejecución de un subprograma FUNCTION	181
9.3. Estructura de un subprograma FUNCTION	182
9.4. El bloque INTERFACE	187
9.5. Vectores y matrices como argumentos en subprogramas FUNCTION	194

9.1. Subprogramas

A menudo se repiten las mismas operaciones en distintas partes de un programa (por ejemplo, hacer un sumatorio). Cada vez que se tiene que repetir una de estas operaciones, se tienen que repetir en el programa todas las instrucciones que la describen. Por otra parte, hay algoritmos que son útiles en distintos programas (por ejemplo, la resolución de un sistema de ecuaciones). Si se quiere utilizar un algoritmo en varios programas, tienen que escribirse en cada programa todas las instrucciones asociadas al algoritmo. Además, conforme el tamaño de los programas aumenta, comprobar que el programa funciona bien es más complicado ya que es más difícil localizar dónde están los errores. Comprobar el funcionamiento de un programa es más sencillo si el programa se puede dividir en partes más pequeñas. En cada una de estas partes se ejecuta una porción del algoritmo y puede comprobarse fácilmente si el programa tiene errores y, en caso de que los tenga, dónde están y corregirlos.

En Fortran, las posibles divisiones de un programa se denominan **unidades de programa** (del inglés *program units*). Una unidad está constituida por un conjunto de instrucciones y se puede compilar de modo independiente. Sin embargo, no todas las unidades se pueden ejecutar independientemente. Las unidades **PROGRAM** de los capítulos anteriores se pueden ejecutar independientemente. Otros tipos de unidades sólo se ejecutan a petición de otras unidades principales. Ejemplos de éstas últimas son las unidades **FUNCTION** y **SUBROUTINE**.

En esta práctica se introduce el uso de las unidades **FUNCTION** mientras que en la práctica siguiente se introducirán las unidades **SUBROUTINE**. La estructura de las unidades **PROGRAM**, **FUNCTION** y **SUBROUTINE** es muy semejante.

Sintaxis de la unidad PROGRAM

```
PROGRAM Nombre_programa
IMPLICIT NONE
.....: instrucciones de declaración de tipo
.....: instrucciones ejecutables
STOP
END PROGRAM Nombre_programa
```

Sintaxis de la unidad FUNCTION

```
FUNCTION Nombre_funcion (lista de argumentos) RESULT (variable_resultado)
IMPLICIT NONE
.....: instrucciones de declaración de tipo
.....: instrucciones ejecutables
RETURN
END FUNCTION Nombre_funcion
```

La variable `variable_resultado` sirve para devolver el resultado de ejecutar la función.

Sintaxis de la unidad SUBROUTINE

```

SUBROUTINE Nombre_subrutina (lista de argumentos)
IMPLICIT NONE
    .....: instrucciones de declaración de tipo
    .....: instrucciones ejecutables
RETURN
END SUBROUTINE Nombre_subrutina

```

9.2. Ejecución de un subprograma FUNCTION

Existen varios tipos de subprogramas función en Fortran 90. En este curso se describen dos tipos:

Subprogramas de funciones intrínsecas. Esta incorporadas como parte del lenguaje Fortran 90. Este tipo de funciones se introdujo en la práctica 2. Ejemplos: **SQRT**(), **SIN**(), **SUM**().

Subprogramas de funciones externas. Son funciones que escribe el programador.

Para comprender mejor la utilidad de las funciones, se pueden analizar las acciones del procesador cuando se ejecuta una sentencia que utiliza una función intrínseca. Por ejemplo, suponga que se tiene que evaluar la siguiente sentencia:

$$X = \mathbf{SQRT}(A*A + B*B)$$

que contiene una función intrínseca.

Cuando el procesador ejecuta esa instrucción:

- i. Calcula el valor de $A*A + B*B$ y utiliza este valor como argumento del subprograma función **SQRT**.
- ii. Inicia la ejecución de la función **SQRT** enviándole el valor del argumento. En este paso se dice que se invoca o *llama* a la función (*call*).
- iii. El subprograma función **SQRT** calcula la raíz cuadrada del argumento ($A*A + B*B$) y asigna el valor resultante a la variable de resultados **SQRT** (en este caso tiene el mismo nombre que la función)
- iv. La expresión aritmética **SQRT**($A*A + B*B$) se sustituye por el valor de la variable **SQRT**
- v. Este valor se asigna a la variable X.

Un subprograma **FUNCTION** se ejecuta de la misma manera que una función intrínseca. Es decir, en aquellas expresiones aritméticas en las que se desee evaluar el valor de la función se escribe:

$$\text{Nombre_funcion} (\text{arg}_1, \text{arg}_2, \dots, \text{arg}_n)$$

Como en las funciones intrínsecas, los argumentos pueden ser, por ejemplo, constantes, variables simples o expresiones aritméticas. Para que el procedimiento funcione correctamente, los argumentos deben corresponder uno a uno con el orden y el tipo de dato de los argumentos que se especifican en la definición del subprograma **FUNCTION** correspondiente.

Por ejemplo, suponga que ha creado una función Mayor que encuentra el mayor número de tres números reales, y que se utiliza en la sentencia

$$\text{RES} = \text{Mayor}(\text{A}, \text{B}, \text{C})$$

donde A, B y C son los argumentos de la función Mayor. Cuando el procesador ejecuta esta instrucción invoca a la función Mayor enviándole los valores de A, B y C, y, tras ejecutarse la función, el valor resultante se asigna a RES.

Los argumentos de la función pueden ser más complicados, por ejemplo,

$$\text{RES} = \text{Mayor}(\text{S}(\text{I}, \text{J}), 60.0, \text{SQRT}(\text{F}))$$

Un subprograma **FUNCTION** que hace esta tarea es el siguiente:

```

!
FUNCTION Mayor(A,B,C) RESULT (D)
!
! Declaraciones de tipo
!
IMPLICIT NONE
REAL :: D ! Variable resultado
REAL, INTENT(IN) :: A, B, C ! Argumentos de la funcion
REAL :: Dummy ! Variable local

!
IF (A <= B) THEN
  D=B
ELSE IF (A <= C) THEN
  D= C
ELSE
  D = A
END IF

!
RETURN
END FUNCTION Mayor

```

9.3. Estructura de un subprograma **FUNCTION**

Como existen varias formas de la estructura de un subprograma **FUNCTION** se considerará la estructura más general.

Estructura de un subprograma FUNCTION

```

FUNCTION Nombre_funcion (lista de argumentos) RESULT (variable_resultado)
IMPLICIT NONE
      ::::::::::::::: instrucciones de declaración de tipo de los argumentos
      ::::::::::::::: instrucciones de declaración de tipo de variable_resultado
      ::::::::::::::: instrucciones de declaración de tipo de las variables locales
      ::::::::::::::: instrucciones ejecutables
RETURN
END FUNCTION Nombre_funcion

```

La variable variable_resultado sirve para devolver el resultado de ejecutar la función.

Para mostrar la estructura de un subprograma **FUNCTION** considere la siguiente tarea. Suponga que quiere calcular mediante una función externa la constante cinética de una reacción, k , utilizando la ecuación de Arrhenius:

$$k = A \cdot e^{-E_a/(RT)}$$

El cálculo de la constante en el programa puede realizarse con la instrucción:

$$k = \text{Arrhenius}(A, E_a, T)$$

donde Arrhenius(A,Ea,T) es la función externa y A, Ea y T los argumentos factor preexponencial, energía de activación y temperatura, todos variables de tipo real.

Un subprograma **FUNCTION** que realizaría esta tarea sería

```

!
! Funcion para el calculo de k utilizando la ecuacion de Arrhenius
!
FUNCTION Arrhenius(A,Ea,T) RESULT(kT)
!
! A ... factor preexponencial, unidades constante cinetica
! Ea ... energia de activacion, J.mol-1
! R ... constante gases ideales, J.mol-1.K-1
! T ... temperatura en Kelvin
! kT ... valor de la constante cinetica
!
! Declaraciones de tipo
!
IMPLICIT NONE
REAL :: kT ! Variable resultado
REAL, INTENT(IN) :: A,Ea,T ! Argumentos de la funcion
REAL, PARAMETER :: R = 8.3145 ! Variables y constantes propias de la funcion
!
! Instrucciones de la funcion
!
      kT = A*EXP(-Ea/(R*T))

```

```

!
!
RETURN
END FUNCTION Arrhenius

```

A continuación, se analizan las partes del subprograma.

i. La primera línea del subprograma FUNCTION

La primera instrucción del subprograma que define la función tiene la forma

FUNCTION	Nombre_función	(<i>lista de argumentos</i>)	RESULT	(<i>variable resultado</i>)
FUNCTION	↓	↓	RESULT	↓
FUNCTION	Arrhenius	(A,Ea,T)	RESULT	(kT)

donde después de **FUNCTION** aparece el nombre de la función, que ha de ser el mismo que se utiliza para invocar la función en el programa principal, acompañado de la lista de argumentos entre paréntesis. En el ejemplo, A ,Ea y T son los *argumentos o variables formales* de la función Arrhenius. A continuación, después de la palabra **RESULT** aparece el nombre de la *variable resultado*, kT, en la que se almacenará el valor resultante de la ejecución de la función.

ii. Instrucciones de declaración de tipo de los argumentos de una función y de la variable resultado.

La lista de argumentos que aparecen en la primera instrucción del subprograma **FUNCTION** son las variables que sirven para compartir o transferir valores entre la unidad principal (desde donde se llama a la función) y la unidad **FUNCTION**. La variable_resultado sirve para devolver a la unidad principal el resultado de evaluar la función.

Los argumentos de una unidad **FUNCTION** (o **SUBROUTINE**) pueden clasificarse en:

Argumentos de entrada. Sirven para enviar desde la unidad principal los valores iniciales de algunas variables del subprograma. Cuando se declaran en el subprograma, se especifican con el atributo **INTENT(IN)** y han de ser del mismo tipo que los argumentos de la función en la llamada del programa principal.

Cuando se utilizan nombres de variables como argumentos de entrada al llamar a la función, éstos no tienen porqué coincidir con los nombres de los argumentos del subprograma.

Argumentos de salida. Variables que se utilizan para almacenar resultados producidos en la ejecución de la función, y transferirlos a la unidad principal. Cuando se declaran en el subprograma, se especifican con el atributo **INTENT(OUT)** y han de ser del mismo tipo que los argumentos de la función en la llamada del programa principal.

La variable_resultado se distingue de estos argumentos de salida en que, cuando se ejecuta la función, el valor de la variable_resultado sustituye a la llamada a la función en el programa principal.

Argumentos mixtos. Variables que pueden utilizarse como argumentos de entrada y salida. Cuando se declaran en el subprograma, se especifican con el atributo **INTENT(IN OUT)** y han de ser del mismo tipo que los argumentos de la función en la llamada del programa principal.

Sintaxis de la declaración de tipo de los argumentos en unidades FUNCTION**Argumentos de entrada.**

TIPO, **INTENT(IN)** :: arg1,arg2,...

Argumentos de salida.

TIPO, **INTENT(OUT)** :: arg1,arg2,...

Argumentos de entrada y salida.

TIPO, **INTENT(IN OUT)** :: arg1,arg2,...

donde TIPO corresponde al tipo de variable (**REAL, INTEGER, LOGICAL,...**) e **INTENT** especifica si el argumento es de entrada (**IN**), salida (**OUT**) o mixto (**IN OUT**).

En general, la utilización de argumentos de salida y entrada-salida en unidades **FUNCTION** es complicada y no se recomienda en este curso. Si se desea intercambiar datos a través de los argumentos del subprograma se recomienda utilizar un subprograma **SUBROUTINE**.

La variable `_resultado` de una unidad **FUNCTION** se declara como cualquier variable en las unidades **PROGRAM**.

En la función Arrhenius del ejemplo, los argumentos A, Ea y T, están declarados como argumentos de entrada con la instrucción

REAL, INTENT(IN) :: A,Ea,T

la variable resultado es kT, es de tipo **REAL** y se declara en la sentencia

REAL :: kT

iii. Instrucciones de declaración de las variables locales

En Fortran, las variables de cada unidad (**PROGRAM, SUBROUTINE, FUNCTION**) son por defecto **variables locales**. Esto quiere decir que las distintas unidades no comparten ni los nombres de las variables, ni los valores que se guardan en éstas. En el caso de las unidades **FUNCTION** los argumentos y la variable `_resultado` no son variables locales.

Las variables locales de las unidades **FUNCTION**, que no aparecen en la primera instrucción del subprograma, se declaran como las variables de una unidad **PROGRAM**. En el caso de la declaración de vectores y matrices tienen algunas posibilidades adicionales que se estudiarán más adelante.

En la función Arrhenius, se ha declarado localmente una constante con nombre en la sentencia

REAL, PARAMETER :: R = 8.3145

iv. Obtención del valor de `variable_resultado`

Para que la unidad **FUNCTION** opere correctamente debe contener al menos una instrucción de asignación para que la `variable_resultado` reciba un valor. Una forma genérica de esta instrucción sería

$$\text{variable_resultado} = \text{exp. aritmética} / \text{exp. lógica} / \text{exp. carácter}$$

donde la expresión debe ser del mismo tipo (**REAL**, **INTEGER**, ...) que la `variable resultado`.

En el caso del ejemplo, la instrucción de asignación es

$$kT = A * \text{EXP}(-Ea/(R*T))$$

En esta instrucción, se utilizan los valores de los argumentos de entrada A, Ea y T, y el valor de la constante R para evaluar la expresión de la ecuación de Arrhenius cuyo valor se asignará a la variable kT.

v. La instrucción **RETURN**

Con la instrucción **RETURN**, se indica al procesador que termine la ejecución de la función y que continúe la ejecución en la unidad principal, en la instrucción que ha invocado la función.

Cuando se llega a la instrucción **RETURN**, el procesador sustituye el valor de `variable_resultado` en la expresión en la que se invocó la función. En el ejemplo, el valor de kT se guardará en la variable k del programa principal

$$k = \text{Arrhenius}(A, Ea, T)$$

Sintaxis de la instrucción **RETURN** en unidades **FUNCTION**

RETURN

Indica al procesador el fin de la ejecución del subprograma **FUNCTION** Nombre_funcion. Debe preceder a **END FUNCTION** Nombre_funcion.

vi. La última instrucción de un subprograma **FUNCTION**

La última instrucción de un subprograma función debe ser **END FUNCTION** Nombre_función. Indica al compilador dónde terminan las instrucciones de la unidad **FUNCTION**.

9.4. El bloque INTERFACE

Cuando se va a invocar una función externa desde una unidad principal, el compilador requiere que en esta unidad se declare que Nombre_funcion hace referencia a una función externa, sus argumentos y de qué tipo son, y cuál es la variable resultado y su tipo. Esta declaración debe hacerse explícitamente, y para ello se utilizará una *interfaz explícita*. La interfaz se escribe como un bloque **INTERFACE**. Si en la unidad se utilizan varios subprogramas, el bloque **INTERFACE** puede contener las interfaces de varias unidades **FUNCTION** y **SUBROUTINE**. Este bloque se escribe a continuación de la declaración de las variables de la unidad principal que utiliza la función.

El bloque **INTERFACE** tiene una utilidad adicional: permite que el compilador compruebe si la llamada a la función es correcta (coinciden el número de argumentos y los tipos de las variables o expresiones utilizadas en los argumentos).

El siguiente programa muestra la definición de una interfaz explícita:

```

PROGRAM Area
! Este programa calcula el area de un circulo
! a partir de su radio.
! Utiliza el subprograma FUNCTION Area_Circulo (r)
!
IMPLICIT NONE
REAL :: radio
INTERFACE
  FUNCTION Area_Circulo(r) RESULT(A)
    IMPLICIT NONE
    REAL, INTENT(IN) :: r
    REAL :: A
  END FUNCTION Area_Circulo
END INTERFACE
!
write(* ,*) "Radio del circulo  "
read(* ,*) radio
! Escriba el radio del circulo
write(* ,*) " El area del un circulo con radio", radio," es", Area_Circulo(radio)
!
STOP
END PROGRAM Area

!
! Funcion que calcula el area de un circulo a partir de
! su radio
!
FUNCTION Area_Circulo(r) RESULT(A)
IMPLICIT NONE
REAL, INTENT(IN) :: r
REAL :: A
REAL, PARAMETER :: Pi = 3.1415927

A = Pi * r * r

RETURN
END FUNCTION Area_Circulo

```

Sintaxis del bloque INTERFACE con unidades FUNCTION**INTERFACE****FUNCTION** Nombre_funcion1 (lista_de_argumentos) **RESULT** (variable_resultado1)**IMPLICIT NONE**

TIPO :: variable_resultado1

TIPO, **INTENT(IN)** :: arg11, arg12 ! declaracion de tipo de los argumentosTIPO, **INTENT(IN)** :: arg13, arg14 !de la funcion**END FUNCTION** Nombre_funcion1.....
.....**FUNCTION** Nombre_funcion4 (lista_de_argumentos) **RESULT** (variable_resultado4)**IMPLICIT NONE**

TIPO :: variable_resultado4

TIPO, **INTENT(IN)** :: arg41, arg42 ! declaracion de tipo de los argumentosTIPO, **INTENT(IN)** :: arg43, arg44 !de la funcion4**END FUNCTION** Nombre_funcion4**END INTERFACE****Ejemplo 9.1** *Cálculo del factorial de un número.**Escriba un programa para calcular el factorial de un número entero haciendo uso de una unidad **FUNCTION**.*La función $k!$ (factorial de k) de un entero positivo se define como

$$k! = \begin{cases} 1 \times 2 \times \dots \times (k-1) \times k & \text{si } k \geq 0 \\ 1 & \text{si } k = 0 \end{cases}$$

Para realizar esta operación puede utilizarse el siguiente algoritmo:

Factorial :

Lee k

Si $k \geq 0$ Calcula $FACTK \leftarrow FACT(K)$ Escribe $FACTK$

si no

Escribe 'k debe ser mayor o igual que cero'

Fin de Factorial

Funcion Fact (I):

FACT = 1

Desde J = 2 hasta I

FACT = J* FACT

Fin de función FACT

Con un editor de texto, edite el fichero tarea49.f90:

C:\EPAQ\Valvanera> EDIT tarea49.f90 

donde introducirá el siguiente programa en Fortran 90:

```

PROGRAM FACTORIAL
!
! Programa que calcula el factorial de un numero.
!
IMPLICIT NONE
INTEGER :: I
INTEGER :: FACTI
!
! Procedimientos
!
INTERFACE
FUNCTION FACT(K) RESULT(Factorial)
IMPLICIT NONE
INTEGER :: Factorial
INTEGER, INTENT(IN) :: K
END FUNCTION FACT
END INTERFACE

WRITE(*,*) 'Este programa calcula el factorial de un numero entero'
WRITE(*,*) 'Introduce un numero entero '
READ(*,*) I
IF ( I < 0) THEN
WRITE(*,*) ' I DEBE SER MAYOR O IGUAL QUE CERO '
ELSE
FACTI = FACT (I)
WRITE(*,*) 'El factorial de ', I, ' es ', FACTI
END IF

STOP
END PROGRAM FACTORIAL
!
! Funcion para el calculo del factorial
!
FUNCTION FACT(K) RESULT (Factorial)
!
IMPLICIT NONE
INTEGER :: Factorial
INTEGER, INTENT(IN) :: K
INTEGER :: J
!
Factorial = 1
DO J= 2, K,1
Factorial = Factorial *J
END DO

RETURN
END FUNCTION FACT

```

Compile y ejecute el programa. Si utiliza el programa para calcular 5! obtendrá:

```
C:\EPAQ\Valvanera>tarea49
```

```
Este programa calcula el factorial de un numero entero
Introduce un numero entero 5
```

```
El factorial de          5 es          120
```

```
Program completed
Press Enter to Continue.
```

Como se comentó al principio de la práctica, las unidades **PROGRAM**, **FUNCTION** (y **SUBROUTINE**) se pueden compilar de modo independiente. Aunque sólo las unidades **PROGRAM** pueden ejecutarse independientemente. De acuerdo con esto, la tarea de este ejercicio se podría haber realizado escribiendo dos ficheros, uno para el programa principal (tarea49a.f90) y el otro para el subprograma función (tarea49b.f90).


Con un editor de texto, edite los ficheros tarea49a.f90 y tarea49b.f90:

```
C:\EPAQ\Valvanera> EDIT tarea49a.f90 
```

```
C:\EPAQ\Valvanera> EDIT tarea49b.f90 
```

En el primer fichero introducirá las instrucciones de la unidad **PROGRAM** FACTORIAL, mientras que en el segundo copiará las instrucciones de la unidad **FUNCTION** FACT.

La mejor manera de compilar por separado los ficheros, y además ensamblarlos para obtener un programa ejecutable final, es utilizar la siguiente instrucción

```
C:\EPAQ\Valvanera> elf90 tarea49a.f90 tarea49b.f90 
```

en la que se escriben de forma sucesiva los nombres de los ficheros que contienen las unidades de programa.

La ejecución de esta instrucción produce los siguientes mensajes en la pantalla

```
C:\EPAQ\Valvanera>elf90 tarea49a.f90 tarea49b.f90
Essential Lahey Fortran 90 Compiler Release 4.00c S/N: E9014261
Copyright (C) 1994-2000 Lahey Computer Systems. All rights reserved.
Copyright (C) 1985-2000 Intel Corp. All rights reserved.
Registered to: QF
                UR
Options:
-nbind          -nc          -nlst          -out tarea49a.exe
-pause          -stack 20000h -nsyn          -winconsole   -nwisk
-nxref

Compiling file tarea49a.f90.
Compiling program unit FACTORIAL at line 2.
Encountered 0 errors, 0 warnings in file tarea49a.f90.

Compiling file tarea49b.f90.
Compiling program unit FACT at line 5.
Encountered 0 errors, 0 warnings in file tarea49b.f90.

386|LINK: 9.1fix1 -- Copyright (C) 1986-98 Phar Lap Software, Inc.
```

Tras los mensajes con los datos del compilador y las opciones de compilación y ensamblado, el mensaje le informa de la compilación sucesiva de las unidades de programa FACTORIAL (tarea49a.f90) y FACT (tarea49b.f90). Tras comprobar que la compilación no genera errores, ensambla el programa ejecutable. El programa ejecutable llevará el nombre del primer fichero escrito en la instrucción. Observe que en la octava línea indica la opción de compilación `-out tarea49a.exe`, que significa que el compilador ha asignado automáticamente el nombre del programa ejecutable como `tarea49a.exe`. Utilice este programa para calcular $5!$.

Puede asignar otro nombre al programa ejecutable añadiendo a la instrucción de compilación `-out nombre.exe`. Por ejemplo,

```
elf90 tarea49a.f90 tarea49b.f90 -out factorial.exe.
```

Ejemplo 9.2 Teoría del estado de transición.

Escriba un programa que utilizando una función calcule la constante cinética k de una reacción química utilizando la formulación termodinámica de la teoría del estado de transición.

La teoría del estado de transición (TST) predice que la constante cinética k de una reacción química puede evaluarse utilizando la ecuación

$$k = \frac{k_B \cdot T}{h} \cdot \exp\left(\frac{\Delta S^\ddagger}{R}\right) \cdot \exp\left(-\frac{\Delta H^\ddagger}{RT}\right)$$

donde k_B es la constante de Boltzmann, h es la constante de Planck, R la constante de los gases ideales, T la temperatura en Kelvin, ΔS^\ddagger la entropía del estado de transición en $\text{J}\cdot\text{mol}^{-1}\cdot\text{K}^{-1}$, y ΔH^\ddagger la entalpía del estado de transición en $\text{J}\cdot\text{mol}^{-1}$.

Con un editor de texto, edite el fichero `tarea50.f90`:

```
C:\EPAQ\Valvanera> EDIT tarea50.f90 
```

donde introducirá el siguiente programa en Fortran 90:

```
PROGRAM TST
!
! Programa que calcula la constante cinetica utilizando la
! formulacion termodinamica de la teoria del estado de transicion.
!
IMPLICIT NONE
REAL :: T,DeltaSET, DeltaHET, KT
!
! Procedimientos
!
INTERFACE
FUNCTION KTST(T,DeltaS,DeltaH) RESULT(KT)
IMPLICIT NONE
REAL :: KT
REAL, INTENT(IN) :: T,DeltaS,DeltaH
END FUNCTION KTST
```

```

END INTERFACE

OPEN(7,FILE='resul.dat')
WRITE(*,*) 'Este programa calcula k utilizando la TST'
WRITE(*,*) 'Introduce la temperatura '
READ(*,*) T
WRITE(*,*) 'Introduce la entropia del estado de transicion J/(mol.K) '
READ(*,*) DeltaSET
WRITE(*,*) 'Introduce la entalpia del estado de transicion J/mol '
READ(*,*) DeltaHET
WRITE(7,*) 'Temperatura ', T, ' K'
WRITE(7,*) 'Entropia del estado de transicion ', DeltaSET, ' J/(mol.K)'
WRITE(7,*) 'Entalpia del estado de transicion ', DeltaHET, ' J/ mol '

!
! Calcula k
!

KT = KTST(T,DeltaSET,DeltaHET)
WRITE(7,*) ' k = ', kT, ' molec-1.m3.s-1'

CLOSE(7)
STOP
END PROGRAM TST

!
! Funcion para el calculo de la contante cinetica
!

FUNCTION KTST(T,DeltaS,DeltaH) RESULT(KT)
IMPLICIT NONE
REAL :: KT
REAL, INTENT(IN) :: T,DeltaS,DeltaH
REAL, PARAMETER :: R = 8.3145,KB = 1.38066E-23, H = 6.62608E-34

KT = (KB*T/H)*EXP(DeltaS/R)*EXP(-DeltaH/(R*T))

RETURN
END FUNCTION KTST

```

Compile y ejecute el programa utilizando los siguientes valores:

$$T = 373.16 \text{ K}$$

$$\Delta S^\ddagger = -50,42 \text{ J.mol}^{-1}.\text{K}^{-1}$$

$$\Delta H^\ddagger = 90660,0 \text{ J.K}^{-1}$$

Tras ejecutar el programa el contenido del fichero `resul.dat` es:

```

Temperatura      373.160      K
Entropia del estado de transicion      -50.4200      J/(mol.K)
Entalpia del estado de transicion      90660.0      J/ mol
k =      0.368900E-02 molec-1.m3.s-1

```

El valor obtenido es muy próximo al experimental $k_{\text{exp}} = 0,340 \cdot 10^{-2} \text{ molec}^{-1}.\text{m}^3.\text{s}^{-1}$ y la diferencia con el valor teórico está dentro de la incertidumbre del experimento.

Ejercicio 9.1 Estructura vibracional de una molécula diatómica.

Escriba un programa que calcule utilizando una unidad FUNCTION la energía vibracional de una molécula diatómica en la aproximación del oscilador armónico.

Para describir la vibración de una molécula diatómica se puede utilizar la aproximación del oscilador armónico:

$$V(r) = \frac{1}{2} \cdot k \cdot (r - r_e)^2$$

donde $V(r)$ es la energía potencial, r la distancia internuclear, k la constante de fuerza del oscilador y r_e la distancia internuclear de equilibrio.

Utilizando este potencial, la solución de la ecuación de Schrödinger proporciona las funciones propias y los valores propios correspondientes a la vibración de una molécula diatómica. La energía de los distintos niveles de vibración viene dada por

$$E_n = \left(n + \frac{1}{2} \right) \cdot h \cdot c \cdot \bar{\nu}_e \quad \text{con } n = 0, 1, 2, \dots$$

donde n es el número cuántico vibracional y $\bar{\nu}_e$ la frecuencia fundamental de vibración de la molécula expresada en cm^{-1} .

Para comprobar el funcionamiento correcto del programa utilice los valores de las frecuencias fundamentales de vibración de las moléculas de N_2 y HCl :

$$\text{N}_2, \bar{\nu}_e = 2359.61 \text{ cm}^{-1} \quad \text{con } n = 0, 1, 2, 3$$

$$\text{HCl}, \bar{\nu}_e = 2989.74 \text{ cm}^{-1} \quad \text{con } n = 0, 1, 2, 3$$

$$h = 6,62608 \cdot 10^{-34} \text{ J s}, c = 2,99792458 \cdot 10^{10} \text{ cm s}^{-1}.$$

Ejercicio 9.2 Estructura vibracional de una molécula diatómica (II).

Escriba un programa que calcule utilizando una unidad FUNCTION la energía vibracional de una molécula diatómica en la aproximación de un oscilador anarmónico.

El potencial que mejor describe la vibración de las moléculas diatómicas es el potencial de Morse:

$$V(r) = D_e \{1 - \exp[-\beta (r - r_e)]\}^2$$

donde $V(r)$ es la energía potencial, r la distancia internuclear, D_e la energía de disociación incluyendo la energía de punto cero de la molécula y β un parámetro relacionado con la constante de fuerza k del oscilador.

Utilizando este potencial, la solución de la ecuación de Schrödinger proporciona de forma aproximada las funciones de onda y energías para los niveles vibracionales de una molécula diatómica. La energía de los distintos niveles de vibración en este modelo viene dada por

$$E_n = h \cdot c \cdot \bar{\nu}_e \cdot \left[\left(n + \frac{1}{2} \right) - x_e \left(n + \frac{1}{2} \right)^2 \right] \quad \text{con } n = 0, 1, 2, \dots$$

donde n es el número cuántico vibracional y $\bar{\nu}_e$ la frecuencia de vibración de la molécula expresada en cm^{-1} y x_e el coeficiente de anarmonicidad.

Para comprobar el funcionamiento correcto del programa utilice los valores de las frecuencias fundamentales de vibración y constantes de anarmonicidad de las moléculas de N_2 y HCl :

$$\text{N}_2, \bar{\nu}_e = 2359,61 \text{ cm}^{-1}, x_e = 0,00613 \quad \text{con } n = 0,1,2,3$$

$$\text{HCl}, \bar{\nu}_e = 2989,74 \text{ cm}^{-1}, x_e = 0,01741 \quad \text{con } n = 0,1,2,3$$

$$h = 6,62608 \cdot 10^{-34} \text{ J s}, c = 2,99792458 \cdot 10^{10} \text{ cm s}^{-1}.$$

9.5. Vectores y matrices como argumentos en subprogramas FUNCTION

En los subprogramas FUNCTION también pueden utilizarse vectores o matrices como argumentos. Existen varias maneras de declarar estos argumentos.

Vectores o matrices de tamaño fijo

En este caso los vectores o matrices utilizados para ejecutar la función tienen que tener las mismas dimensiones que los vectores o matrices declarados en la función. Se declaran de igual manera que en las unidades **PROGRAM**.

Este método tiene la limitación de que se tiene que determinar *a priori* la dimensión del vector declarado en la función.

Ejemplo 9.3 Encontrar el mayor valor de una lista de diez valores utilizando vectores de tamaño fijo en una función.

El siguiente ejemplo muestra el uso de vectores con subprogramas **FUNCTION**. La función Mayor10 encuentra el mayor valor de una lista de diez valores.

Con un editor de texto, edite el fichero tarea51.f90:

```
C:\EPAQ\Valvanera> EDIT tarea51.f90 
```

donde introducirá el siguiente programa en Fortran 90:

```
PROGRAM EJDIM1
!
! Programa para encontrar el numero
! mayor de una lista de 10 numeros.
!
IMPLICIT NONE
INTEGER :: MAYOR
INTEGER, DIMENSION(1:10) ::X
INTERFACE
FUNCTION Mayor10(Y) RESULT (Z)
IMPLICIT NONE
INTEGER, DIMENSION(1:10),INTENT(IN) ::Y
```

```

    INTEGER ::Z
    END FUNCTION Mayor10
END INTERFACE
    READ(*,*)X
    MAYOR = Mayor10(X)
    WRITE(6,*)'El mayor de los diez numeros es', MAYOR
STOP
END PROGRAM EJDIM1
!
! Funcion que calcula el mayor de 10 numeros
!
FUNCTION Mayor10(Y) RESULT(Z)
IMPLICIT NONE
INTEGER, DIMENSION(1:10),INTENT(IN) :: Y
INTEGER :: Z
INTEGER :: K
    Z = Y(1)
    DO K = 2,10,1
        IF (Z < Y(K)) Z = Y(K)
    END DO
RETURN
END FUNCTION Mayor10

```

Compile y ejecute el programa.

La ejecución del programa da como resultado en la pantalla:

```

C:\EPAQ\Valvanera>tarea51
1 3 4 56 78 13 33 12 4 51

El mayor de los diez numeros es          78

Program completed
Press Enter to Continue.

```



Errores de programación. Cuando se utilizan vectores o matrices de tamaño fijo, las dimensiones del vector o la matriz en el subprograma **FUNCTION** y en la unidad que la ejecuta deben coincidir. De no hacerlo, la ejecución del programa puede dar resultados impredecibles.

Vectores o matrices de tamaño variable

En general, cuando se trabaja con matrices o vectores en una función es preferible dejar indeterminada la dimensión de los argumentos. La dimensiones de éstos se fijarán cuando el programa se ejecute.

Pueden distinguirse dos tipos de arreglos con estas características:

- Arreglos de forma predefinida (*assumed shape arrays*).
- Arreglos automáticos (*automatic arrays*).

i. Arreglos de forma predefinida

Con esta denominación se hace referencia a argumentos de un subprograma **FUNCTION** o **SUBROUTINE** de los que sólo se sabe que son vectores o matrices pero sus dimensiones no aparecen determinadas. Este tipo de arreglos permite utilizar vectores y matrices sin necesidad de transmitir las dimensiones del vector o la matriz como argumento. De acuerdo con las dimensiones de los argumentos de entrada que le llegan a la función, esta decide las dimensiones de los vectores o matrices que utiliza cuando se ejecuta.

Sintaxis de declaración de vectores o matrices del tipo *forma predefinida*

Para declarar una variable subíndicada del tipo *assumed shape* hay que indicar el tipo de variable (**INTEGER**, **REAL**, ...), el atributo **DIMENSION** y a continuación un paréntesis que incluya tantos **:** como dimensiones tenga el arreglo.

No es necesario declarar las dimensiones del vector o la matriz, ya que el procedimiento **FUNCTION** o **SUBROUTINE** las determinará cada vez que se ejecute el subprograma.

En el caso de un vector (una sola dimensión):

```
TIPO, DIMENSION (:), INTENT(IN) :: Variable1 [, Variable2 ,...]  
TIPO, DIMENSION (:), INTENT(OUT) :: Variable1 [, Variable2, ...]  
TIPO, DIMENSION (:), INTENT(IN OUT) :: Variable1 [, Variable2, ...]
```

En el caso de una matriz (dos dimensiones):

```
TIPO, DIMENSION (:,:), INTENT(IN) :: Variable1 [, Variable2 ,...]  
TIPO, DIMENSION (:,:), INTENT(OUT) :: Variable1 [, Variable2, ...]  
TIPO, DIMENSION (:,:), INTENT(IN OUT) :: Variable1 [, Variable2, ...]
```

Ejemplo 9.4 Encontrar el mayor valor de una lista de diez valores utilizando vectores de tipo forma predefinida en una función.

Este ejemplo es otra versión del programa del ejercicio 9.3 donde se utilizan vectores del tipo forma predefinida en lugar de vectores de tamaño fijo.

Con un editor de texto, edite el fichero tarea52.f90:

```
C:\EPAQ\Valvanera> EDIT tarea52.f90 
```

donde introducirá el siguiente programa en Fortran 90:


```

PROGRAM EJDIM2
!
! Programa para encontrar el numero mayor de una lista de 10 numeros.
!
IMPLICIT NONE
INTEGER :: MAYOR
INTEGER, DIMENSION(1:10) :: X
INTERFACE
  FUNCTION MayorN(Y,N) RESULT(Z)
    IMPLICIT NONE
    INTEGER, DIMENSION(:),INTENT(IN) :: Y
    INTEGER,INTENT(IN) :: N
    INTEGER :: Z
  END FUNCTION MayorN
END INTERFACE

READ(*,*) X
MAYOR = MayorN(X(1:10),10)
WRITE(6,*) 'El mayor de los diez numeros es', MAYOR

STOP
END PROGRAM EJDIM2

!
! Funcion que encuentra el mayor valor de una lista de N numeros
!
FUNCTION MayorN(Y,N) RESULT(Z)
IMPLICIT NONE
INTEGER, DIMENSION(:),INTENT(IN) :: Y
INTEGER,INTENT(IN) :: N
INTEGER :: Z
INTEGER :: K

Z = Y(1)
DO K = 2,N,1
  IF (Z < Y(K)) Z = Y(K)
END DO
RETURN
END FUNCTION MayorN

```

Compile y ejecute el programa.

La ejecución del programa da como resultado en la pantalla:

```

C:\EPAQ\Valvanera>tarea52
1 3 4 56 78 13 33 12 4 51

El mayor de los diez numeros es          78

Program completed
Press Enter to Continue.

```

Ejercicio 9.3 *Escriba un programa que utilizando un subprograma función, y un vector como argumento, calcule la media de una serie de valores.*

Los datos de entrada han de leerse desde un fichero DATOS.DAT y el resultado ha de escribirse en un fichero MEDIA.DAT.

Arreglos automáticos

Con esta denominación se hace referencia a un tipo de argumentos de un subprograma **FUNCTION** o **SUBROUTINE** que almacenan un vector o una matriz, que se dimensionan durante la ejecución del subprograma, pero que, a diferencia de los arreglos de forma predefinida, las dimensiones del arreglo también son argumentos de entrada del subprograma.

Sintaxis de declaración de vectores o matrices del tipo automático

Para declarar una variable subindicada del tipo *automatico* hay que indicar el tipo de variable que es (**REAL**, **INTEGER**, **LOGICAL**, **COMPLEX**, ...), el atributo **DIMENSION**, y entre paréntesis los nombres de los argumentos de entrada del subprograma que son las dimensiones de los arreglos. Las dimensiones de éstos se determinan cuando se ejecuta el subprograma.

En el caso de un vector (una sola dimensión):

```
TIPO, DIMENSION (Nmin:Nmax), INTENT(IN) :: Variable1 [, Variable2,...]
TIPO, DIMENSION (Nmin:Nmax), INTENT(OUT) :: Variable1 [, Variable2,...]
TIPO, DIMENSION (Nmin:Nmax), INTENT(IN OUT) :: Variable1 [, Variable2,...]
```

donde Nmin y Nmax son los nombres de argumentos de entrada del subprograma que determinan el tamaño del vector.

En el caso de una matriz (dos dimensiones):

```
TIPO, DIMENSION (Nmin:Nmax, Mmin:Mmax), INTENT(IN) :: Variable1 [, Variable2,...]
TIPO, DIMENSION (Nmin:Nmax, Mmin:Mmax), INTENT(OUT) :: Variable1 [, Variable2,...]
TIPO, DIMENSION (Nmin:Nmax, Mmin:Mmax), INTENT(IN OUT) :: Variable1 [, Variable2,...]
```

donde Nmin, Nmax, Mmin y Mmax son los nombres de argumentos de entrada de la función que determinan el número de filas y columnas de la matriz.

Ejemplo 9.5 *Encontrar el mayor valor de una lista de diez valores utilizando vectores de tipo automático en una función.*

Este ejemplo es otra versión del programa del ejercicio 9.3 donde se utilizan vectores del tipo automático en lugar de vectores de tamaño fijo en el subprograma **FUNCTION**.

Con un editor de texto, edite el fichero tarea53.f90:

```
C:\EPAQ\Valvanera> EDIT tarea53.f90 
```

donde introducirá el siguiente programa en Fortran 90:

```

PROGRAM EJDIM3
!
! Programa para encontrar el numero mayor de una lista de 10 numeros.
!
IMPLICIT NONE
INTEGER :: Ndatos, Mayor
INTEGER, DIMENSION(1:100) :: X
INTERFACE
    FUNCTION MayorN3(Y,Nmin,Nmax) RESULT (Z)
    IMPLICIT NONE
    INTEGER,INTENT(IN) :: Nmin,Nmax
    INTEGER, DIMENSION(Nmin:Nmax),INTENT(IN) :: Y
    INTEGER :: Z
END FUNCTION MayorN3
END INTERFACE

WRITE(*,*) 'Escriba el numero total de datos'
READ(*,*) NDatos
IF ((NDatos < 2).OR.(NDatos>100)) STOP !Comprueba que la lista tiene menos de 100 valores
READ(*,*) X(1:NDatos)

MAYOR = MayorN3(X(1:NDatos),1,NDatos)
WRITE(*,*) 'El mayor de los numeros es ', MAYOR

STOP
END PROGRAM EJDIM3

!
! Funcion que encuentra el mayor valor de una lista de N numeros
!
FUNCTION MayorN3(Y,Nmin,Nmax) RESULT (Z)
IMPLICIT NONE
INTEGER,INTENT(IN) :: Nmin,Nmax
INTEGER, DIMENSION(Nmin:Nmax),INTENT(IN) :: Y
INTEGER :: Z
INTEGER :: K

Z = Y(Nmin)
DO K = Nmin+1,Nmax,1
    IF (Z < Y(K)) Z = Y(K)
END DO

RETURN
END FUNCTION MayorN3

```

Compile y ejecute el programa.

Vectores y matrices como variables locales en unidades FUNCTION

En el caso de las variables locales o propias de la unidad pueden definirse vectores y matrices con un tamaño fijo, como en las unidades **PROGRAM**. Sin embargo, en las unidades **FUNCTION** y **SUBROUTINE** pueden utilizarse vectores o matrices cuya dimensión se determina al comienzo de la ejecución de la función.

Declaración de un vector o una matriz local

Para declarar una variable subindicada en una unidad **FUNCTION** con dimensiones fijas hay que indicar el tipo de variable —**REAL, INTEGER, ...**—, el atributo **DIMENSION** y las dimensiones del vector o matriz que vienen dadas por números enteros Nmin, Nmax, Mmin, Mmax. En el caso de un vector (una sola dimensión):

```
TIPO, DIMENSION (Nmin:Nmax) :: Variable1 [, Variable2,...]
```

```
TIPO, DIMENSION (Nmin:Nmax) :: Variable1 [, Variable2,...]
```

```
TIPO, DIMENSION (Nmin:Nmax) :: Variable1 [, Variable2,...]
```

En el caso de una matriz (dos dimensiones):

```
TIPO, DIMENSION (Nmin:Nmax, Mmin:Mmax) :: Variable1 [, Variable2,...]
```

```
TIPO, DIMENSION (Nmin:Nmax, Mmin:Mmax) :: Variable1 [, Variable2,...]
```

```
TIPO, DIMENSION (Nmin:Nmax, Mmin:Mmax) :: Variable1 [, Variable2,...]
```

Recuerde, las variables locales no tienen el atributo **INTENT** porque no son argumentos de la función.

Para declarar una variable subindicada en una unidad **FUNCTION** con dimensiones variables hay que indicar el tipo de variable —**REAL, INTEGER, ...**—, el atributo **DIMENSION** y las dimensiones del vector o matriz Nmin, Nmax, Mmin, Mmax, que son argumentos de entrada de la función. Por lo tanto, las dimensiones del vector o matriz local se determinan cuando se ejecuta la función.

Otras observaciones importantes

A la hora de trabajar con unidades **FUNCTION** o **SUBROUTINE** hay que tener en cuenta que cuando el programa abandona un procedimiento libera todas las direcciones de memoria a las que se hacen referencia en el subprograma. Es decir, cada vez que se ejecuta el subprograma lo hace como si antes nunca se hubiera ejecutado.

Práctica 10

Subprogramas: subrutinas

CONTENIDOS

10.1. Estructura de un subprograma SUBROUTINE	202
10.2. Ejecución de un subprograma SUBROUTINE	205
10.3. El bloque INTERFACE	206
10.4. Vectores y matrices como argumentos en subprogramas SUBROUTINE	213

10.1. Estructura de un subprograma SUBROUTINE

En general, las mismas reglas y restricciones que se aplican a los subprogramas **FUNCTION** (funciones) son aplicables a los subprogramas **SUBROUTINE** (subrutinas). Sin embargo, a diferencia de los subprogramas función, las subrutinas sólo intercambian valores a través de los argumentos de la subrutina. Como se verá, la ejecución de las unidades **SUBROUTINE** también difiere de la de los subprogramas **FUNCTION**.

Estructura de un subprograma SUBROUTINE

```

SUBROUTINE Nombre_subrutina (lista_de_argumentos)
IMPLICIT NONE
.....      ! instrucciones de declaracion de tipo de los argumentos
.....      !
.....      ! instrucciones de declaracion de tipo de las variables locales
.....      !
.....      ! instrucciones ejecutables
.....      !
RETURN
END SUBROUTINE Nombre_subrutina

```

Para mostrar la estructura de una subrutina considere la siguiente tarea que corresponde al cálculo de la constante cinética de una reacción química, k , utilizando la ecuación de Arrhenius

$$k = A \cdot e^{-E_a/(RT)}$$

```

!
!   Subrutina para el calculo de k utilizando la ecuacion de Arrhenius
!
SUBROUTINE Arrhenius(A,Ea,T,kT)
!
!   A ... factor preexponencial, unidades constante cinetica
!   Ea ... energia de activacion, J.mol-1
!   R ... constante gases ideales, J.mol-1.K-1
!   T ... temperatura en Kelvin
!   kT ... valor de la constante cinetica
!
IMPLICIT NONE
! Declaracion de tipo de los argumentos
REAL, INTENT(IN) :: A,Ea,T
REAL, INTENT(OUT) :: kT
! Declaracion de tipo de las variables y constantes propias de la subrutina
REAL, PARAMETER :: R = 8.3145
!
!   kT = A*EXP(-Ea/(R*T))
!

```

```

RETURN
END SUBROUTINE Arrhenius

```

Como puede observarse, a diferencia de las unidades **FUNCTION**, en las unidades **SUBROUTINE** no aparece una variable `_resultado`.

i. La primera línea del subprograma SUBROUTINE

La primera instrucción del subprograma que define la subrutina tiene la forma

```

SUBROUTINE Nombre_subrutina (lista de argumentos)
                ↓                ↓
SUBROUTINE Arrhenius (A,Ea,T,kT)

```

donde después de **SUBROUTINE** aparece el nombre de la subrutina, acompañado de la lista de argumentos entre paréntesis. En el ejemplo, (A,Ea,T,kT), son los *argumentos* o *variables formales* de la subrutina Arrhenius. En una subrutina los resultados de la ejecución se envían a la unidad principal que la ejecuta utilizando un argumento de salida o mixto.

ii. Instrucciones de declaración de tipo de los argumentos de una subrutina.

La lista de argumentos que aparecen en la primera instrucción del subprograma **SUBROUTINE** son las variables que sirven para compartir o transferir valores entre la unidad principal (desde donde se invoca a la subrutina) y la unidad **SUBROUTINE**.

Los argumentos de una unidad **SUBROUTINE** (o **FUNCTION**) pueden clasificarse en:

Argumentos de entrada. Sirven para enviar desde la unidad principal los valores iniciales de algunas variables del subprograma. Cuando se declaran en el subprograma, se especifican con el atributo **INTENT(IN)** y han de ser del mismo tipo que los argumentos de la subrutina cuando se invoca desde el programa principal.

Cuando se utilicen nombres de variables como argumentos de entrada al llamar a la subrutina, éstos no tienen porqué coincidir con los nombres de los argumentos del subprograma.

Argumentos de salida. Variables que se utilizan para almacenar resultados producidos en la ejecución de la subrutina, y transferirlos a la unidad principal. Cuando se declaran en el subprograma, se especifican con el atributo **INTENT(OUT)** y han de ser del mismo tipo que los argumentos de la subrutina en la llamada del programa principal. Una subrutina habitualmente se utiliza para obtener varios resultados simultáneamente.

Argumentos mixtos. Variables que pueden utilizarse como argumentos de entrada y salida. Cuando se declaran en el subprograma, se especifican con el atributo **INTENT(IN OUT)** y han de ser del mismo tipo que los argumentos de la subrutina en la llamada del programa principal. Este tipo de argumentos no suele utilizarse en las unidades **FUNCTION**.

Sintaxis de la declaración de tipo de los argumentos en unidades SUBROUTINE**Argumentos de entrada.**

TIPO, **INTENT(IN)** :: arg1,arg2,...

Argumentos de salida.

TIPO, **INTENT(OUT)** :: arg1,arg2,...

Argumentos de entrada y salida.

TIPO, **INTENT(IN OUT)** :: arg1,arg2,...

donde TIPO corresponde al tipo de variable (**REAL, INTEGER, LOGICAL,...**) e **INTENT** especifica si el argumento es de entrada (**IN**), salida (**OUT**) o mixto (**IN OUT**).

En la subrutina Arrhenius del ejemplo, los argumentos A, Ea y T son argumentos de entrada de tipo real y están declarados con la instrucción

REAL, INTENT(IN) :: A, Ea, T

En cambio, la variable kT es un argumento de salida de tipo real y se declara en la sentencia

REAL, INTENT(OUT) :: kT

para indicar que a través de este argumento se devuelve un valor al programa principal.

iii. Instrucciones de declaración de las variables locales

Como ya se comentó en la práctica anterior, las variables de cada unidad (**PROGRAM, SUBROUTINE, FUNCTION**) son por defecto **variables locales**. Esto quiere decir que las distintas unidades no comparten ni los nombres de las variables, ni los valores que se guardan en éstas. En el caso de las unidades **SUBROUTINE**, los argumentos no son variables locales.

Las variables locales de las unidades **SUBROUTINE**, que no aparecen en la primera instrucción del subprograma, se declaran como las variables de una unidad **PROGRAM**. En el caso de la declaración de vectores y matrices tienen algunas posibilidades adicionales que se estudiarán más adelante.

En la subrutina Arrhenius se ha declarado localmente una constante con nombre en la sentencia

REAL, PARAMETER :: R = 8.3145

Las variables locales de las unidades **FUNCTION** y **SUBROUTINE** se declaran como en las unidades **PROGRAM**. Los vectores y las matrices tienen algunas posibilidades adicionales que se estudiarán más adelante.

iv. Instrucciones ejecutables

A continuación de las declaraciones de tipo, se escriben las instrucciones ejecutables de la subrutina. En éstas se asignarán valores a los argumentos de salida y mixtos del subprograma.

En el ejemplo, se utiliza la instrucción de asignación

$$kT = A * \text{EXP}(-Ea/(R * T))$$

En esta instrucción, de forma semejante a la función descrita en la sección 9.3, se utilizan los valores de los argumentos de entrada A, Ea y T, y el valor de la constante local R para evaluar la expresión de la ecuación de Arrhenius cuyo valor se asignará a la variable kT. Sin embargo, a diferencia de la función, esta variable es un argumento de salida de la subrutina Arrhenius.

v. La instrucción RETURN

Con la instrucción **RETURN** se indica al procesador que termine la ejecución de la subrutina y que continúe la ejecución en la unidad principal. Cuando se llega a la instrucción **RETURN**, el procesador actualiza el valor de las variables de la unidad principal correspondientes a los argumentos de salida y mixtos de la subrutina, y continúa la ejecución en la instrucción siguiente del programa.

Sintaxis de la instrucción RETURN en unidades SUBROUTINE

RETURN

Indica al procesador el fin de la ejecución del subprograma **SUBROUTINE** Nombre_subrutina. Debe preceder a la última instrucción de la subrutina **END SUBROUTINE** Nombre_subrutina.

vi. La última instrucción de un subprograma SUBROUTINE

La última instrucción de un subprograma subrutina debe ser **END SUBROUTINE** Nombre_subrutina. Indica al compilador dónde terminan las instrucciones de la unidad **SUBROUTINE**.

10.2. Ejecución de un subprograma SUBROUTINE

Un subprograma **SUBROUTINE** se ejecuta utilizando la siguiente instrucción en el programa principal:

CALL Nombre_subrutina (arg1, arg2, ..., argn)

Cuando el procesador llega a esta instrucción, comienza la ejecución de la subrutina Nombre_subrutina. Este nombre debe coincidir con el nombre del subprograma correspondiente y los argumentos deben ser compatibles y estar ordenados de acuerdo con la primera instrucción del subprograma.

Ejecución de una subrutina (llamada a una subrutina)

CALL Nombre_subrutina(arg1, arg2, ..., argn)

Los argumentos de entrada pueden ser constantes, variables o expresiones aritméticas. Sin embargo, los argumentos de salida o mixtos deben ser obligatoriamente nombres de variables. Los argumentos pueden corresponder a nombres de funciones intrínsecas u otros subprogramas pero no se discutirá su uso en este curso.

Para el ejemplo utilizado en esta sección, escribiríamos la siguiente instrucción en el programa principal

CALL Arrhenius(A, E, T, k)

donde los nombres de variable A, E, T y k, son del mismo tipo y corresponden a los argumentos A, Ea, T y kT de la subrutina.

10.3. El bloque INTERFACE

Al igual que las funciones, la unidad que llama a la subrutina no sabe *a priori* cuántos argumentos tiene, ni de qué tipo son éstos. Estos deben indicarse explícitamente, y para ello se utilizará una *interfaz explícita*. Esto se hace utilizando un bloque **INTERFACE** a continuación de la declaración de las variables de la unidad principal. Si en la unidad se utilizan varios subprogramas el bloque **INTERFACE** puede contener las interfaces de varias unidades **SUBROUTINE** o **FUNCTION**.

El bloque **INTERFACE** tiene una utilidad adicional, permite que el compilador compruebe si la llamada a la subrutina es correcta (coinciden el número de argumentos y los tipos de las variables o expresiones utilizadas en los argumentos).

La estructura general de un programa que utiliza una subrutina puede esquematizarse como:

```

PROGRAM Nombre_programa
IMPLICIT NONE
..... !
..... ! instrucciones de declaracion de tipo
..... !
INTERFACE !
..... ! interfaces de funciones
..... !
..... ! interfaces de subrutinas
..... !
END INTERFACE !
..... ! instrucciones ejecutables
CALL Nombre_subrutina(arg1, arg2,...)
..... ! instrucciones ejecutables
STOP
END PROGRAM Nombre_programa

SUBROUTINE Nombre_subrutina (arg1, arg2, ...)
IMPLICIT NONE
..... ! instrucciones de declaracion de tipo de los argumentos
..... ! instrucciones de declaracion de tipo de las variables locales
..... ! instrucciones ejecutables
RETURN
END SUBROUTINE Nombre_subrutina

```

Sintaxis del bloque INTERFACE con unidades SUBROUTINE

```

INTERFACE
SUBROUTINE Nombre_subrutina1 (lista_de_argumentos)
IMPLICIT NONE
TIPO, INTENT(IN) :: arg11, arg12 ! declaracion de tipo de los argumentos
TIPO, INTENT(OUT) :: arg13, arg14 !de la subrutina
TIPO, INTENT(IN OUT) :: arg15, arg16
END SUBROUTINE Nombre_subrutina1
.....
SUBROUTINE Nombre_subrutina4 (lista_de_argumentos)
IMPLICIT NONE
TIPO, INTENT(IN) :: arg41, arg42 ! declaracion de tipo de los argumentos
TIPO, INTENT(OUT) :: arg43, arg44 !de la subrutina
TIPO, INTENT(IN OUT) :: arg45, arg46
END SUBROUTINE Nombre_subrutina4
END INTERFACE

```

Si la unidad principal también utiliza funciones, sus interfaces deben incluirse dentro del bloque INTERFACE. No importa el orden en el que se especifiquen las interfaces de subrutinas y funciones.

Ejemplo 10.1 *Cálculo del área de un círculo utilizando una subrutina.*

Este ejemplo muestra el uso de una subrutina. El programa calcula el área de un círculo y utiliza una subrutina para realizar el cálculo.

Con un editor de texto, edite el fichero tarea54.f90:

```
C:\EPAQ\Valvanera> EDIT tarea54.f90 
```

donde introducirá el siguiente programa en Fortran 90:

```
PROGRAM Circulo
!-----
! Este programa calcula el area de un circulo
! a partir de su radio
! SUBROUTINE Area_Circulo (r,A)
!-----

IMPLICIT NONE
REAL :: radio, area
INTERFACE
  SUBROUTINE Area_Circulo(r,a)
  IMPLICIT NONE
  REAL, INTENT(IN) :: r
  REAL, INTENT(OUT) :: a
  END SUBROUTINE Area_Circulo
END INTERFACE
!
write(*,*) "Radio del circulo (m) "
read(*,*) radio
CALL Area_Circulo(radio,area)
! Escriba el radio del circulo
write(*,*) " El area del un circulo con radio", radio,"(m) es", area, "m2"
!
STOP
END PROGRAM Circulo

!-----
! Subrutina que calcula el area de un circulo a partir de
! su radio
!-----

SUBROUTINE Area_Circulo(r,a)
IMPLICIT NONE
REAL, INTENT(IN) :: r
REAL, INTENT(OUT) :: a
REAL, PARAMETER :: Pi = 3.1415927

A = Pi * r * r
```

```

RETURN
END SUBROUTINE Area_Circulo

```

Compile y ejecute el programa.

Ejemplo 10.2 *Uso de subrutinas para leer y escribir datos.*

Este ejemplo demuestra el uso de argumentos de entrada, salida y mixtos en un programa que utiliza varias subrutinas.

Con un editor de texto, edite el fichero tarea55.f90:

```
C:\EPAQ\Valvanera> EDIT tarea55.f90 
```

donde introducirá el siguiente programa en Fortran 90:

```

PROGRAM Ej10a
!
IMPLICIT NONE
REAL :: X
!
INTERFACE
  SUBROUTINE LEER1(A)
    IMPLICIT NONE
    REAL, INTENT(OUT) :: A
  END SUBROUTINE LEER1
  SUBROUTINE Mult2(A)
    IMPLICIT NONE
    REAL, INTENT(IN OUT) :: A
  END SUBROUTINE Mult2
  SUBROUTINE Escribe1(A)
    IMPLICIT NONE
    REAL, INTENT(IN) :: A
  END SUBROUTINE Escribe1
END INTERFACE

CALL Leer1(X)
WRITE(*,*) "Unidad Program – Tras ejecutar Leer1", X
CALL Mult2(X)
WRITE(*,*) "Unidad Program – Tras ejecutar Mult2", X
CALL Escribe1(X)

STOP
END PROGRAM Ej10a
!
! Subrutina para leer un valor real y devolverlo al programa principal
!
SUBROUTINE LEER1(A)
IMPLICIT NONE
REAL, INTENT(OUT) :: A

```

```

WRITE(*,*) " Unidad Leer1 – Introduzca un numero real"
READ(*,*) A

RETURN
END SUBROUTINE LEER1
!
! Subrutina que multiplica un numero real por 2 y devuelve su valor
!
SUBROUTINE Mult2(A)
IMPLICIT NONE
REAL, INTENT(IN OUT) :: A

A =2*A

RETURN
END SUBROUTINE Mult2
!
! Subrutina que escribe en la pantalla un numero real
!
SUBROUTINE Escribe1(A)
IMPLICIT NONE
REAL, INTENT(IN) :: A

WRITE(*,*) "Unidad Escribe1 – El numero real vale"
WRITE(*,*) A

RETURN
END SUBROUTINE Escribe1

```

Compile y ejecute el programa.

La ejecución del programa produce en la pantalla:

```

C:\EPAQ\Valvanera>tarea55

Unidad Leer1 - Introduzca un numero real 24

Unidad Program - Tras ejecutar Leer1      24.0000
Unidad Program - Tras ejecutar Mult2     48.0000
Unidad Escribe1 - El numero real vale    48.0000

Program completed
Press Enter to Continue.

```

Ejemplo 10.3 *Cálculo de la constante cinética de una reacción química, k , utilizando la ecuación de Arrhenius.*

La ecuación de Arrhenius se expresa como

$$k = A \cdot e^{-E_a/(RT)}$$

La constante cinética k se calculará utilizando una función, y la lectura y escritura de datos se realizará utilizando subrutinas.

Con un editor de texto, edite el fichero tarea56.f90:

```
C:\EPAQ\Valvanera> EDIT tarea56.f90 
```

donde introducirá el siguiente programa en Fortran 90:

```
PROGRAM Ej10b
!
! Programa que calcula el valor de la constante cinética k
! utilizando la ecuacion de Arrhenius
!
IMPLICIT NONE
REAL :: A,E,T
REAL :: k
!
! Procedimientos
!
INTERFACE
  SUBROUTINE LeerDatos(A,E,T)
    IMPLICIT NONE
    REAL, INTENT(OUT) :: A,E,T
  END SUBROUTINE LeerDatos
  FUNCTION Arrhenius(A,Ea,T) RESULT(kT)
    IMPLICIT NONE
    REAL :: kT
    REAL, INTENT(IN) :: A,Ea,T
  END FUNCTION Arrhenius
  SUBROUTINE Escribek(T,k)
    IMPLICIT NONE
    REAL, INTENT(IN) :: T,k
  END SUBROUTINE Escribek
END INTERFACE

CALL LeerDatos(A,E,T)
k = Arrhenius(A,E,T)
CALL Escribek(T,k)

STOP
END PROGRAM Ej10b
!
! Subrutina para leer los parametros de la ecuacion de Arrhenius
!
SUBROUTINE LeerDatos(A,E,T)
IMPLICIT NONE
REAL, INTENT(OUT) :: A,E,T

WRITE(*,*) " Factor prexponencial "
READ(*,*) A
WRITE(*,*) " Energia de activacion "
READ(*,*) E
WRITE(*,*) " Temperatura"
```

```

READ(*,*) T

RETURN
END SUBROUTINE LeerDatos
!
! Subrutina que escribe en la pantalla el valor de k
!
SUBROUTINE Escribek(T,k)
IMPLICIT NONE
REAL, INTENT(IN) :: T,k

WRITE(*,*) "Temperatura ", T, "k = ", k

RETURN
END SUBROUTINE Escribek
!
! Funcion para el calculo de k utilizando la ecuacion de Arrhenius
!
FUNCTION Arrhenius(A,Ea,T) RESULT(kT)
!
! A ... factor preexponencial, unidades constante cinetica
! Ea ... energia de activacion, J.mol-1
! R ... constante gases ideales, J.mol-1.K-1
! T ... temperatura en Kelvin
! kT ... valor de la constante cinetica
!
IMPLICIT NONE
REAL :: kT ! Declaracion tipo variable resultado
REAL, INTENT(IN) :: A,Ea,T ! Declaracion tipo argumentos
REAL, PARAMETER :: R = 8.3145 ! ctes y variables locales
!
kT = A*EXP(-Ea/(R*T))
!
RETURN
END FUNCTION Arrhenius

```

Compile y ejecute el programa.

Para comprobar el funcionamiento del programa utilice los valores: $A = 2,0510^{13} \text{ s}^{-1}$, $E_a = 102,544 \text{ kJ mol}^{-1}$, y $T = 300 \text{ K}$.

En la ejecución se obtiene:

```

C:\EPAQ\Valvanera>tarea56

Factor preexponencial 2.5E13

Energia de activacion 102.544E3

Temperatura 300

Temperatura      300.000      k =      0.349842E-04

```



```
Program completed
Press Enter to Continue.
```

10.4. Vectores y matrices como argumentos en subprogramas SUBROUTINE

En los subprogramas **SUBROUTINE** también pueden utilizarse vectores o matrices como argumentos. Las consideraciones hechas para las unidades **FUNCTION** también son válidas para las unidades **SUBROUTINE**.

Los argumentos pueden declararse como:

- Arreglos de tamaño fijo. Este método tiene la limitación de que se tienen que determinar *a priori* la dimensión del vector o matriz declarado en la subrutina.
- Arreglos de tamaño variable. Se pueden distinguir dos tipos de arreglos con estas características:
 - i. Arreglos de forma predefinida *assumed shape arrays*. En este tipo de arreglos sólo ha de indicarse que el argumento sirve para guardar una matriz o un vector pero no se especifican sus dimensiones. De acuerdo con los arreglos que llegan a la subrutina como argumentos, ésta decide las dimensiones de los vectores o matrices que utiliza cuando se ejecuta. Su sintaxis se explica en la práctica anterior (ver página 196).
 - ii. Arreglos automáticos *automatic arrays*. En este tipo de arreglos, el vector o la matriz se dimensionan al ejecutar la subrutina, pero que, a diferencia de los arreglos de forma predefinida, las dimensiones del arreglo también son argumentos de entrada del subprograma. Su sintaxis se describe en la practica anterior (ver página 198).

Ejemplo 10.4 Lectura y escritura de una matriz con subrutinas.

En este ejemplo, se propone un programa que utiliza una subrutina para leer una matriz desde un fichero y otra subrutina para imprimirla en otro fichero.

Además, en el ejemplo se mostrará el procedimiento para compilar y ensamblar un programa en el que las subrutinas y el programa principal están en ficheros diferentes. Esto es muy útil cuando la subrutina o función que se quiere utilizar ha sido escrita por otro compañero, copiado de un CD o bajado de un sitio web.

Con un editor de texto, edite en el fichero tarea57.f90:

```
C:\EPAQ\Valvanera> EDIT tarea57.f90 
```

donde introducirá el siguiente programa en Fortran 90, que es la unidad PROGRAM (programa principal):

```

PROGRAM ej_submat
!
! Programa que lee una matriz N filas( N <= 20)
! y M columnas (N<= 20)
! El programa utiliza una matriz para leer los datos
! y otra para escribirlos .
!
IMPLICIT NONE
INTEGER, PARAMETER :: nfilas = 20, ncols = 20
INTEGER :: n,m
REAL, DIMENSION(nfilas,ncols) :: a
INTERFACE
SUBROUTINE Leer_Matriz(a, nmax, mmax, n,m)
  IMPLICIT NONE
  INTEGER, INTENT(IN) :: nmax, mmax
  INTEGER, INTENT(OUT) :: n, m
  REAL, DIMENSION(:,:),INTENT(OUT):: a
END SUBROUTINE Leer_Matriz

SUBROUTINE Imprime_Matriz (b,n,m)
  IMPLICIT NONE
  INTEGER, INTENT(IN) :: n, m
  REAL, DIMENSION(:,:),INTENT(IN) :: b
END SUBROUTINE Imprime_Matriz
END INTERFACE
!
! Lee la matriz
!
CALL Leer_Matriz(a, nfilas, ncols, n,m)
!
! Escribe la matriz
!
CALL Imprime_Matriz(a,n,m)
!
STOP
END PROGRAM ej_submat

```

Con un editor de texto, edite en el fichero leermatriz.f90:

```
C:\EPAQ\Valvanera> EDIT leermatriz.f90 
```

donde introducirá el siguiente programa en Fortran 90, que es la unidad SUBROUTINE Leer_Matriz:

```

SUBROUTINE Leer_Matriz(a, nmax, mmax, n,m)
IMPLICIT NONE
INTEGER, INTENT(IN) :: nmax, mmax
INTEGER, INTENT(OUT) :: n, m
REAL, DIMENSION(:,:),INTENT(OUT):: a
!
INTEGER :: i, j

OPEN(4, FILE = 'matriz.dat')

```

```

READ(4,*) n,m
WRITE(*,*) ' La matriz tiene '
WRITE(*,*) ' Filas ',n,' Columnas ',m
!
! Comprueba que la matriz no excede los limites.
!
IF ((n > nmax).OR.(m>mmax)) THEN
  WRITE(*,*) ' N debe ser menor que ', nmax
  WRITE(*,*) ' M debe ser menor que ', mmax
  STOP
END IF
!
! Lee la matriz por filas
!
READ(4,*) ((a(i,j), j=1,m),i=1,n)

CLOSE(4)

RETURN
END SUBROUTINE Leer_Matriz

```

y, finalmente, edite en el fichero imprimematriz.f90:

```
C:\EPAQ\Valvanera> EDIT imprimematriz.f90 
```

donde introducirá el siguiente programa en Fortran 90, que es la unidad SUBROUTINE Imprime_Matriz:

```

SUBROUTINE Imprime_Matriz (b,n,m)

IMPLICIT NONE
INTEGER, INTENT(IN) :: n, m
REAL, DIMENSION(:,:),INTENT(IN) :: b
INTEGER :: i,j

OPEN(11, FILE = 'matriz.out')

DO i= 1,n
  WRITE (11,*) (b(i,j), j=1,m)
END DO

CLOSE(11)
RETURN
END SUBROUTINE Imprime_Matriz

```

Como las distintas unidades se encuentran en distintos ficheros, el proceso de compilación y ensamblaje será análogo al utilizado en la segunda parte del ejemplo 9.1 en la página 190:

```
C:\EPAQ\Valvanera>ELF90 tarea57.f90 leermatriz.f90 imprimematriz.f90 + 
```

Si la compilación se produce sin errores, observará en la pantalla una serie de mensajes como la que sigue:

```
C:\EPAQ\Valvanera>ELF90 tarea57.f90 leermatriz.f90 imprimematriz.f90

Essential Lahey Fortran 90 Compiler Release 4.00c S/N: E9014262
...
...
Compiling file tarea57.f90.
Compiling program unit EJ_SUBMAT at line 1.
Encountered 0 errors, 0 warnings in file ejsubmat.f90.

Compiling file leermatriz.f90.
Compiling program unit LEER_MATRIZ at line 1.
Encountered 0 errors, 0 warnings in file leermatriz.f90.

Compiling file imprimematriz.f90.
Compiling program unit IMPRIME_MATRIZ at line 1.
Encountered 0 errors, 0 warnings in file imprimematriz.f90.

386|LINK: 9.1fix1 -- Copyright (C) 1986-98 Phar Lap Software, Inc.
```

Para comprobar el programa, introduzca en un fichero `matriz.dat` los siguientes datos

```
4 4
1. 2. 3. 4.
5. 6. 7. 8.
9. 10. 11. 12.
13. 14. 15. 16.
```

Después de la ejecución del programa, edite o visualice los resultados del fichero `matriz.out`.

```
C:\EPAQ\Valvanera>edit matriz.out + 
```

```
C:\EPAQ\Valvanera>type matriz.out + 
```

Ejercicio 10.1 *Traza de una matriz.*

Escriba un programa que lea una matriz cuadrada desde un fichero (utilizando la subrutina `Leer_Matriz` del ejemplo 10.4), calcule la traza de esta matriz utilizando una función y escriba en el fichero de resultados la matriz original y su traza.

En el programa considere que la matriz mayor con la que puede trabajar tiene 25x25 elementos.

La traza de una matriz cuadrada se define como la suma de los elementos de la diagonal principal. Sea la matriz \mathbf{A} de orden $N \times N$, su traza viene dada por

$$\chi(\mathbf{A}) = \sum_{i=1}^n a_{ii}$$

Utilice para comprobar el funcionamiento del programa la matriz:

$$\mathbf{A} = \begin{pmatrix} 1,0 & 2,0 & 3,0 & 4,0 \\ 5,0 & 6,0 & 7,0 & 8,0 \\ 9,0 & 10,0 & 11,0 & 12,0 \\ 13,0 & 14,0 & 15,0 & 16,0 \end{pmatrix}$$

cuya traza es

$$\chi(\mathbf{A}) = 1,0 + 6,0 + 11,0 + 16,0 = 34,0$$

Las tareas que realizará el programa pueden resumirse en

- i Leer una matriz de $N \times M$ elementos (utilice la subrutina `Leer_Matriz`)
- ii Comprobar que la matriz es cuadrada
 - Si la matriz no es cuadrada, enviar un mensaje de error.
 - Si la matriz es cuadrada, calcula la traza de la matriz utilizando una función. Imprime la matriz y su traza.

Ejercicio 10.2 Obtención de la matriz traspuesta.

Escriba un programa que lea una matriz desde un fichero (utilizando la subrutina `Leer_Matriz` del ejemplo 10.4), calcule la traspuesta de esta matriz (utilice una función o subrutina) y escriba en un fichero de resultados (utilizando la subrutina `Imprime_Matriz` del ejemplo 10.4) la matriz original y su traspuesta.

En el programa considere que la mayor matriz con la que se puede trabajar tiene 25x25 elementos.

La matriz traspuesta de la matriz \mathbf{A} de orden $N \times M$ elementos, \mathbf{A}^T es una matriz de $M \times N$ elementos que vienen dados por

$$a_{ij}^T = a_{ji}$$

Utilice para comprobar el funcionamiento del programa la matriz:

$$\mathbf{A} = \begin{pmatrix} 2,0 & 3,0 & 4,0 \\ 3,0 & 4,0 & 5,0 \end{pmatrix}$$

cuya traspuesta viene dada por

$$\mathbf{A}^T = \begin{pmatrix} 2,0 & 3,0 \\ 3,0 & 4,0 \\ 4,0 & 5,0 \end{pmatrix}$$

Ejercicio 10.3 Cálculo de parámetros muestrales.

Escriba un programa que lea un número arbitrario de datos x_i de un fichero y calcule las variables muestrales: media, \bar{x} , varianza, $s^2(x)$ y varianza de la media, $s^2(\bar{x})$.

En el programa considere que la lista más grande con la que se puede trabajar tiene 100 elementos.

Parte III

Apéndices

Soluciones a ejercicios seleccionados

Práctica 4

4.2

PROGRAM raices

```
!
!
! Este programa calcula las raices de una polinomio
! de segundo grado:  $A * X * X + B * X + C$ 
!
!
! a      ... coeficiente del polinomio de segundo grado
! b      ... coeficiente del polinomio de primer grado
! c      ... coeficiente del polinomio de orden cero
! delta  ... discriminante de la ecuacion
! absdelta ...
! x1     ... raiz real 1 del polinomio
! x2     ... raiz real 2 del polinomio
! preal  ... parte real para raices complejas
! pimag  ... parte imaginaria para raices complejas
!
!
! IMPLICIT NONE
! REAL :: a,b,c
! REAL :: delta
! REAL :: x1, x2
! REAL :: preal, pimag
!
! Pide al usuario los coeficientes de la ecuacion
!
! WRITE (*,*) 'Este programa calcula las raices de un polinomio de '
! WRITE (*,*) 'segundo grado de la forma  $A * X**2 + B * X + C = 0.$  '
! WRITE (*,*) 'Introduzca los coeficientes a, b, c: '
! READ (*,*) a, b, c
! WRITE (*,*) 'Los coeficientes A, B, and C son: ', a, b, c
```

! Comprueba que el polinomio es de segundo grado

IF (A /= 0) **THEN**

! Calcula el discriminante

delta = b**2 - 4. * a * c

! Calculo de las raices teniendo en cuenta el valor
! del discriminante

IF (delta > 0.) **THEN** ! dos raices reales

x1 = (-b + **SQRT**(delta))/(2.0*a)

x2 = (-b - **SQRT**(delta))/(2.0*a)

WRITE (*,*) 'Esta ecuación tiene dos raices reales :'

WRITE (*,*) 'X1 = ', x1

WRITE (*,*) 'X2 = ', x2

ELSE IF (delta == 0.) **THEN** ! dos raices reales dobles

WRITE(*,*) 'Raices reales dobles'

WRITE(*,*) 'X1 = X2 =', x1

ELSE ! dos raices complejas

preal = -b/(2. * a)

pimag = **SQRT**(**ABS**(delta))/(2. * a)

WRITE (*,*) 'Esta ecuacion tiene raices complejas: '

WRITE (*,*) 'X1 = ', preal, ' + ', pimag, 'i '

WRITE (*,*) 'X2 = ', preal, ' - ', pimag, 'i '

END IF

ELSE

WRITE(*,*) ' La ecuacion no es de segundo grado'

END IF

STOP

END PROGRAM Raices

4.4

PROGRAM Fibo2

!

! Programa para calcular los terminos de

! la serie de Fibonacci que no exceden 10000

!

IMPLICIT NONE

INTEGER :: i,x1,x2,x3

x1 = 1

x2 = 1

WRITE (*,*) 'i = 1 x = ', x1

WRITE (*,*) 'i = 2 x = ', x2

i = 2

DO WHILE (x3 <10000)

 i = i + 1

 x3 = x1 + x2

 x1 = x2

Apéndice A

```
x2 = x3
IF (x3<10000) WRITE(*,*) 'i = ',i,' x = ',x3
END DO

STOP
END PROGRAM Fibo2
```

4.5

```
PROGRAM cinetica1
!
!
! Este programa calcula la fraccion de concentraciones
! en una reaccion quimica que sigue una cinetica
! de primer orden
!
! n          ... numero de datos
! t          ... tiempo
! ti         ... valor inicial del tiempo
! tf         ... valor final del tiempo
! inct       ... incremento del tiempo
! k          ... constante cinetica
! concentracion ... concentracion relativa del reactivo
! concentracion0 ... concentracion inicial , ti = 0
!
IMPLICIT NONE
INTEGER :: i, n
REAL :: t, ti, tf, inct
REAL :: k
REAL :: concentracion, concentracion0

! Pide al usuario datos

WRITE (*,*) 'Este programa calcula la concentracion relativa '
WRITE (*,*) 'de un reactivo en una cinetica de primer orden '
WRITE (*,*)
WRITE (*,*) 'Introduzca la constante cinetica, k (s-1) '
READ (*,*) k
WRITE (*,*) 'Introduzca el tiempo final del intervalo, t (s) '
READ (*,*) tf
WRITE (*,*) 'Introduzca el numero de datos a calcular, n '
READ (*,*) n

! Calcula inct

ti = 0
inct = (tf - ti)/REAL(n - 1)

! Calcula la tabla tiempo - concentracion

concentracion0 = 1
DO i= 1, n, 1
    t = ti + (i-1)*inct
```

```

    concentracion = concentracion0 * EXP(-k*t)
    WRITE(*,*) t,concentracion
  END DO

  STOP
  END PROGRAM cinetica1

```

Ejemplo de la ejecución del programa

```

C:\EPAQ\Valvanera> cinetica1

Este programa calcula la concentracion relativa
de un reactivo en una cinetica de primer orden

Introduzca la constante cinetica, k (s-1) 0.0693

Introduzca el tiempo final del intervalo, t (s) 20

Introduzca el numero de datos a calcular, n11

    0.00000      1.00000
    2.00000      0.870576
    4.00000      0.757903
    6.00000      0.659812
    8.00000      0.574417
   10.00000      0.500074
   12.00000      0.435352
   14.00000      0.379007
   16.00000      0.329955
   18.00000      0.287251
   20.00000      0.250074

Program completed
Press Enter to Continue.

```

4.6

PROGRAM Pvpap

```

! Calcula la presion de vapor de una sustancia mediante la ecuacion de
! Clausius–Clapeyron

! R    ... constante de los gases ideales 8.314 J/(K.mol)
! DVAP ... entalpia de vaporizacion (J/mol)
! P0   ... presion de vapor de referencia (Torr)
! P1   ... presion de vapor a estimar (Torr)
! T0   ... temperatura de referencia (K)
! T1   ... temperatura a la que se realiza el calculo (K).
! Ti   ... temperatura a la que se inicia el calculo (K).
! Tf   ... temperatura a la que se finaliza el calculo (K).
! nt   ... numero de valores a calcular.
!

```

```

IMPLICIT NONE

```

Apéndice A

```
INTEGER :: i, NT
REAL :: DVap, P0, P1, T0, T1, Ti, Tf, IncT
REAL :: EX
REAL, PARAMETER :: R=8.314

OPEN(4, FILE="inp.dat")
OPEN(7, FILE="out.dat")

READ(4,*) DVap
READ(4,*) T0, Ti, Tf, NT
READ(4,*) P0

IncT = (Tf - Ti)/REAL(NT - 1)
DO i = 1,NT,1
    T1 = Ti + (i-1)*IncT
    Ex = -DVap/R*(1.0/T1 - 1.0/T0)
    P1 = P0 * EXP(Ex)
    WRITE(7,*) T1-273.15,' grados C',P1,' Torr'
END DO

CLOSE(4)
CLOSE(7)

STOP
END PROGRAM Pvap
```

Datos entrada (inp.dat):

```
43.9E3
293.15    294.15    303.15    9
17.535
```

Datos salida (out.dat)

```
21.0000    grados C    18.6423    Torr
22.1250    grados C    19.9619    Torr
23.2500    grados C    21.3638    Torr
24.3750    grados C    22.8525    Torr
25.5000    grados C    24.4325    Torr
26.6250    grados C    26.1086    Torr
27.7500    grados C    27.8859    Torr
28.8750    grados C    29.7695    Torr
30.0000    grados C    31.7650    Torr
```

Práctica 5

5.1

```

PROGRAM Raoult
!
!
! Este programa calcula la presión total y las presiones
! parciales de una mezcla binaria ideal utilizando la ley
! de Raoult
!
! n      ... número de datos
! p01    ... presión de vapor del componente puro 1
! p02    ... presión de vapor del componente puro 2
! p1     ... presión parcial del componente 1
! p2     ... presión parcial del componente 2
! pt     ... presión de vapor de la mezcla
! x1     ... fracción molar del componente 1
!
IMPLICIT NONE
INTEGER :: i, n
REAL :: p01, p02, p1, p2, pt
REAL :: x1, incx
REAL, PARAMETER :: x1max = 1.0

! Pide al usuario datos

WRITE (*,*) 'Este programa calcula la presión de vapor de una mezcla binaria '
WRITE (*,*) 'utilizando la ley de Raoult '
WRITE (*,*)
WRITE (*,*) 'Introduzca la presión de vapor del componente 1 puro, P01 (Torr) '
READ (*,*) p01
WRITE (*,*) 'Introduzca la presión de vapor del componente 2 puro, P02 (Torr) '
READ (*,*) p02
WRITE (*,*) 'Introduzca el número de datos a calcular, n '
READ (*,*) n

! Calcula incx
!  $X1(\min) = 0.0$   $X1(\max) = 1.0$ 

incx = (1.0 - 0.0)/REAL(n - 1)

! Calcula la tabla fracción molar – presión vapor

DO i = 1, n, 1
  x1 = x1max - (i-1)*incx
  p1 = p01*x1
  p2 = p02*(1.0-x1)
  pt = p1 + p2
  WRITE(* ,*) x1, p1, 1-x1, p2, pt
END DO

STOP
END PROGRAM Raoult

```

Apéndice A

Ejemplo de la ejecución del programa

```
C:\EPAQ\Valvanera>raoult
```

```
Este programa calcula la presion de vapor de una mezcla binaria
utilizando la ley de Raoult
```

```
Introduzca la presion de vapor del componente 1 puro, P01 (Torr) 22.3
```

```
Introduzca la presion de vapor del componente 2 puro, P02 (Torr) 74.7
```

```
Introduzca el numero de datos a calcular, n 5
```

1.00000	22.3000	0.00000	0.00000	22.3000
0.750000	16.7250	0.250000	18.6750	35.4000
0.500000	11.1500	0.500000	37.3500	48.5000
0.250000	5.57500	0.750000	56.0250	61.6000
0.000000	0.00000	1.00000	74.7000	74.7000

```
Program completed
Press Enter to Continue.
```

5.6

PROGRAM primos

```
!
! Calcula los numeros primos comprendidos entre 1 y 999
!
! i      ... numero a testar
! j      ... posible divisor de i
! resto  ... resto de la division
! resto = i - j*(i/j) resto == 0 si i divisible por j
!
```

IMPLICIT NONE

```
INTEGER :: i, j, resto
```

```
DO i = 1, 999, 1
```

```
  resto = 1
```

```
  DO j = 2, i/2, 1
```

```
    resto = MOD(i,j)
```

```
    IF (resto == 0) EXIT
```

```
  END DO
```

```
  IF (resto /= 0) WRITE(*,*) i
```

```
END DO
```

```
STOP
```

```
END PROGRAM primos
```

Práctica 6

6.1

```

PROGRAM particula_caja
!
! Este programa calcula energia de un electron confinado
! en una caja de potencial unidimensional
! nmax ... numero de niveles a calcular
! n ... numero cuantico n
! h ... constante de Planck (J.s)
! me ... masa del electron (kg)
! l ... longitud de la caja (m)
! En ... energia del nivel n
!
!
IMPLICIT NONE
INTEGER, PARAMETER :: imax = 100
REAL,PARAMETER :: h = 6.626E-34
REAL,PARAMETER :: me = 9.109E-31
INTEGER :: n,nmax
REAL :: l
REAL, DIMENSION(1:imax) :: E
WRITE(*,*) "Introduce la longitud de la caja (m) "
READ(*,*) l
WRITE(*,*) "Introduce el numero de niveles a calcular "
READ(*,*) nmax
!
! El calculo de la energia puede simplificarse si se define una
! constante con nombre
! const = 6.0139E-38
! que es el valor del cociente de constantes (h*h)/(8.0*me), y
! E(n) = const*(n*n)/(l*l)
!
!
DO n = 1,nmax,1
    E(n) = (h*h)/(8.0*me)*(n*n)/(l*l)
    WRITE(*,*) "n = ",n," E_n = ",E(n)," J"
END DO

STOP
END PROGRAM particula_caja

```

Ejemplo de la ejecución del programa

Para comprobar el funcionamiento del programa se utilizaron los valores $n = 4$ y $l = 10,0 \text{ \AA} = 1,0010^{-9} \text{ m}$ obteniéndose el resultado,

```

C:\EPAQ\Valvanera>caja

Introduce la longitud de la caja (m) 1.0e-9

Introduce el numero de niveles a calcular 4

n =          1 E_n =    0.602479E-19 J
n =          2 E_n =    0.240992E-18 J

```


Apéndice A

```
n =          3 E_n =    0.542231E-18 J
n =          4 E_n =    0.963967E-18 J
```

```
Program completed
Press Enter to Continue.
```

6.5

PROGRAM triatomica

```
!
!   Calcular las coordenadas internas de una molecula triatomica
!   a partir de las coordenadas cartesianas (x, y, z) de los nucleos.
!
!   atomo1 ... coordenadas del atomo i
!   rij     ... vector atomo1 - atomoj
!   dij     ... distancia interatómica entre atomos i-j
!   theta   ... angulo A-B-C
!
!   Variables auxiliares
!
!   i       ... indice de los bucles DO
!   dij2    ... cuadrado de la distancia internuclear dij
!   costheta ... coseno angulo A-B-C
!
IMPLICIT NONE
INTEGER :: i
REAL, DIMENSION(1:3) :: atomo1, atomo2, atomo3
REAL, DIMENSION(1:3) :: r12, r23, r13
REAL :: d12, d23, d13
REAL :: d122, d232, d132
REAL :: theta, costheta
!
WRITE(*,*) " Introduzca las coordenadas del atomo 1 "
READ(*,*) (atomo1(i), i=1,3,1)
WRITE(*,*) " Introduzca las coordenadas del atomo 2 "
READ(*,*) (atomo2(i), i=1,3,1)
WRITE(*,*) " Introduzca las coordenadas del atomo 3 "
READ(*,*) (atomo3(i), i=1,3,1)
!
r12 = atomo1 - atomo2
r23 = atomo3 - atomo2
r13 = atomo1 - atomo3
!
!   Calcula y escribe las distancias interatomicas
!
d122 = 0.0
d232 = 0.0
d132 = 0.0
DO i= 1,3,1
    d122 = d122 + r12(i)*r12(i)
    d232 = d232 + r23(i)*r23(i)
    d132 = d132 + r13(i)*r13(i)
END DO
d12 = SQRT(d122)
```

```

d23 = SQRT(d232)
d13 = SQRT(d132)
cosTheta = 0.0
DO i= 1,3,1
  cosTheta = costheta + r12(i)*r23(i)/(d12*d23)
END DO
theta = ACOS(costheta) *180.0/ACOS(-1.0)

```

```

WRITE(*,*) " Distancias internucleares y angulo"
WRITE(*,*) " d12 = ", d12," nm"
WRITE(*,*) " d23 = ", d23," nm"
WRITE(*,*) " d13 = ", d13," nm"
WRITE(*,*) " theta = ", theta

```

! El calculo tambien puede realizarse utilizando la funcion intrínseca SUM

```

d122 = SUM(r12*r12)
d232 = SUM(r23*r23)
d132 = SUM(r13*r13)
d12 = SQRT(d122)
d23 = SQRT(d232)
d13 = SQRT(d132)
cosTheta = SUM(r12*r23)/(d12*d23)
theta = ACOS(costheta) *180.0/ACOS(-1.0)

```

```

WRITE(*,*)
WRITE(*,*) " Distancias internucleares y angulo – utilizando SUM"
WRITE(*,*) " d12 = ", d12," nm"
WRITE(*,*) " d23 = ", d23," nm"
WRITE(*,*) " d13 = ", d13," nm"
WRITE(*,*) " theta = ", theta

```

! El calculo tambien puede realizarse utilizando la funcion intrínseca SUM pero no variables auxiliares

```

d12 = SQRT(SUM(r12*r12))
d23 = SQRT(SUM(r23*r23))
d13 = SQRT(SUM(r13*r13))
cosTheta = SUM(r12*r23)/(d12*d23)
theta = ACOS(costheta) *180.0/ACOS(-1.0)

```

```

WRITE(*,*)
WRITE(*,*) " Distancias internucleares y angulo – utilizando SUM sin variables auxiliares"
WRITE(*,*) " d12 = ", d12," nm"
WRITE(*,*) " d23 = ", d23," nm"
WRITE(*,*) " d13 = ", d13," nm"
WRITE(*,*) " theta = ", theta

```

! El calculo tambien puede realizarse utilizando la funcion intrínseca DOT_PRODUCT (producto escalar)

```

d12 = SQRT(DOT_PRODUCT(r12,r12))
d23 = SQRT(DOT_PRODUCT(r23,r23))
d13 = SQRT(DOT_PRODUCT(r13,r13))

```

Apéndice A

```
cosTheta = DOT_PRODUCT(r12,r23)/(d12*d23)
theta = ACOS(costheta) *180.0/ACOS(-1.0)
```

```
WRITE(*,*)
WRITE(*,*) " Distancias internucleares y angulo – utilizando DOT_PRODUCT"
WRITE(*,*) " d12 = ", d12," nm"
WRITE(*,*) " d23 = ", d23," nm"
WRITE(*,*) " d13 = ", d13," nm"
WRITE(*,*) " theta = ", theta

STOP
END PROGRAM triatomica
```

Ejemplo de la ejecución del programa

La ejecución del programa con los datos de prueba produce el resultado (molécula de H₂O):

```
C:\EPAQ\Valvanera> triatomica

Introduzca las coordenadas del atomo 1 0.07575 0.0 -0.05214

Introduzca las coordenadas del atomo 2 0.0 0.0 0.00657

Introduzca las coordenadas del atomo 3 -0.07575 0.0 -0.05214

Distancias internucleares y angulo
d12 = 0.958380E-01 nm
d23 = 0.958380E-01 nm
d13 = 0.151500 nm
theta = 104.445

Distancias internucleares y angulo - utilizando SUM
d12 = 0.958380E-01 nm
d23 = 0.958380E-01 nm
d13 = 0.151500 nm
theta = 104.445

Distancias internucleares y angulo - utilizando SUM sin variables auxiliares
d12 = 0.958380E-01 nm
d23 = 0.958380E-01 nm
d13 = 0.151500 nm
theta = 104.445

Distancias internucleares y angulo - utilizando DOT_PRODUCT
d12 = 0.958380E-01 nm
d23 = 0.958380E-01 nm
d13 = 0.151500 nm
theta = 104.445

Program completed
Press Enter to Continue.
```

Práctica 7

7.2

```

PROGRAM ajuste_recta
!
! Version 3
!
! Este programa calcula los parametros del ajuste a una recta mediante
! el metodo de minimos cuadrados.
!
! nmax ... numero maximo de datos (PARAMETER)
! np ... numero de datos
! x ... variable x
! y ... variable y
! a ... pendiente
! b ... ordenada en el origen
! ycalc ...  $a \cdot x_i + b$ 
! residuo ...  $y - ycalc$ 
! sumx ... sum x
! sumy ... sum y
! sumxy ... sum  $x \cdot y$ 
! sumx2 ... sum  $x^2$ 
! sumy2 ... sum  $y^2$ 
! delta ...  $n \cdot \text{sumx}^2 - \text{sumx}^2$ 
! s2 ... varianza del ajuste
! s2a ... varianza de a
! s2b ... varianza de b
! r ... coeficiente de correlación lineal
!
! Variables auxiliares :
! i ... indice bucles DO
!
IMPLICIT NONE
INTEGER, PARAMETER :: nmax = 100
INTEGER :: i, np
REAL, DIMENSION(1:nmax) :: x, y
REAL, DIMENSION(1:nmax) :: ycalc, residuo
REAL :: a, b
REAL :: sumx, sumy, sumxy, sumx2, sumy2, delta
REAL :: s2, s2a, s2b, r
!
OPEN(4, FILE='datos.dat')
OPEN(7, FILE='resul.dat')
OPEN(8, FILE='tabla.dat')
!
! Lee el numero de datos (n) y comprueba que el numero de datos no es muy pequenyo (n <= 2)
! ni muy grande (n>nmax)
!
READ(4,*) np
IF (np < 0) THEN
  WRITE(*,*) 'Np debe ser un entero positivo'
  STOP

```

```

ELSE IF (np < 2) THEN
    WRITE(*,*) 'Para calcular una recta al menos se necesitan 2 valores'
    STOP
ELSE IF (np > nmax) THEN
    WRITE(*,*) 'Np debe ser menor o igual que', nmax
    STOP
ELSE
    WRITE(*,*) 'El fichero datos.dat contiene ', np, ' datos'
END IF
WRITE(7,*)
WRITE(7,*) 'El fichero datos.dat contiene np ', np, ' puntos'
!
! Lee los datos
!
DO i = 1, np, 1
    READ(4,*) x(i), y(i)
END DO
!
! Calcula a y b
!
sumx = SUM(x(1:np))
sumy = SUM(y(1:np))
sumxy = SUM(x(1:np)*y(1:np))
sumx2 = SUM(x(1:np)**2)
delta = np*sumx2 - sumx*sumx

a = (np*sumxy - sumx*sumy)/delta
b = (sumx2*sumy - sumx*sumxy)/delta

WRITE(*,*)
WRITE(7,*)
WRITE(*,*) 'sumx =', sumx, ' sumy =', sumy
WRITE(7,*) 'sumx =', sumx, ' sumy =', sumy
WRITE(*,*) 'sumxy =', sumxy, ' sumx2 =', sumx2
WRITE(7,*) 'sumxy =', sumxy, ' sumx2 =', sumx2
WRITE(*,*) 'delta =', delta
WRITE(7,*) 'delta =', delta
WRITE(*,*) 'pendiente, a =', a, ' ordenada origen, b =', b
WRITE(7,*) 'pendiente, a =', a, ' ordenada origen, b =', b
!
ycalc(1:np) = a*x(1:np) +b
residuo(1:np) = y(1:np) - ycalc(1:np)
DO i = 1, np, 1
    WRITE(8,*) x(i), y(i), ycalc(i), residuo(i)
END DO
!
! Calcula s, s(a) y s(b) y r
!
s2 = SUM(residuo(1:np)**2)/REAL(np -2)
s2a = s2*np/delta
s2b = s2*sumx2/delta
sumy2 = SUM(y(1:np)**2)

```

```

r = (np*sumxy - sumx*sumy)/SQRT((np*sumx2 - sumx**2)*(np*sumy2 - sumy**2))
WRITE(*,*)
WRITE(7,*)
WRITE(*,*) 's2 =', s2, ' s2(a) =', s2a, ' s2(b) =', s2b
WRITE(7,*) 's2 =', s2, ' s2(a) =', s2a, ' s2(b) =', s2b
WRITE(*,*) 's =', SQRT(s2), ' s(a) =', SQRT(s2a), ' s(b) =', SQRT(s2b)
WRITE(7,*) 's =', SQRT(s2), ' s(a) =', SQRT(s2a), ' s(b) =', SQRT(s2b)
WRITE(*,*) 'r =', r
WRITE(7,*) 'r =', r
!
STOP
END PROGRAM ajuste_recta

```

Ejemplo de la ejecución del programa.**Datos entrada (datos.dat):**

```

10
4.0 3.7
8.0 7.8
12.5 12.1
16.0 15.6
20.0 19.8
25.0 24.5
31.0 31.1
35.0 35.5
40.0 39.4
40.0 39.5

```

Datos salida (resul.dat):

```

El fichero datos.dat contiene np          10 puntos

sumx =    231.500      sumy =    229.000
sumxy =    6849.15    sumx2 =    6903.25
delta =    15440.3
pendiente, a =    1.00244      ordenada origen, b =    -0.306598

s2 =    0.121971      s2(a) =    0.789954E-04  s2(b) =    0.545325E-01
s =    0.349243      s(a) =    0.888794E-02  s(b) =    0.233522
r =    0.999686

```

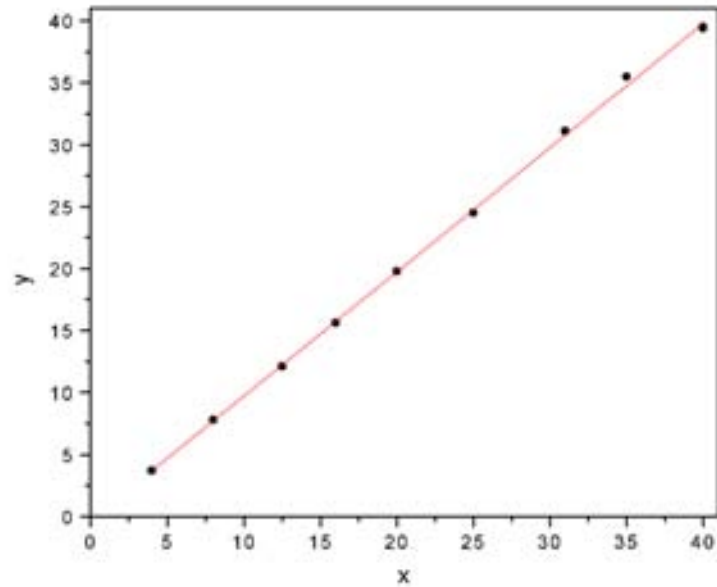
Datos salida (tabla.dat):

4.00000	3.70000	3.70318	-0.318122E-02
8.00000	7.80000	7.71296	0.870395E-01
12.5000	12.1000	12.2240	-0.123962
16.0000	15.6000	15.7325	-0.132520
20.0000	19.8000	19.7423	0.576992E-01
25.0000	24.5000	24.7545	-0.254524
31.0000	31.1000	30.7692	0.330809
35.0000	35.5000	34.7790	0.721027
40.0000	39.4000	39.7912	-0.391193
40.0000	39.5000	39.7912	-0.291195

Apéndice A

Puede comprobar que los datos del fichero `result.dat` coinciden con los obtenidos con la calculadora.

La representación de los datos de (x_i, y_i) (puntos) y $(x_i, y_{calculado})$ (línea) con Origin resulta



lo que muestra la bondad del ajuste realizado. Si se ajusta la recta con Origin se obtienen los mismos resultados que con el programa,

```
Fit and Residual data have been saved into TABLA worksheet.  
[05/09/02 16:29 "/Graph1" (2453253)]  
Linear Regression for TABLA_B:  
Y = A + B * X
```

```
Parameter Value Error
```

```
-----  
A -0.3066 0.23352
```

```
B 1.00244 0.00889  
-----
```

```
R SD N P
```

```
-----  
0.99969 0.34924 10 <0.0001  
-----
```

```
Fit and Residual data have been saved into TABLA worksheet.
```

Práctica 8

8.4

```

PROGRAM traza_matriz
IMPLICIT NONE
INTEGER, PARAMETER :: nmax =25, mmax =25
INTEGER :: n, m
REAL ,DIMENSION (1:nmax,1:mmax) :: a
REAL :: traza
INTEGER :: i, j
!
!
! Lee una matriz cuadrada A de dimensiones N x N y calcula su traza
!
!   nmax ... dimension maxima de las matrices (PARAMETER)
!   n     ... numero de filas
!   m     ... numero de columnas
!   a     ... matriz A
!   traza ... traza de A
!
! Variables auxiliares :
!   i, j   ... indices bucles DO
!
!
OPEN (4, FILE="matriz.dat")
OPEN (7, FILE="out.dat")

! Lee la matriz A

READ(4,*) n, m
IF (n /= m) THEN
  WRITE(*,*) ' La matriz no es cuadrada '
  STOP
ELSE IF (n > 25) THEN
  WRITE(*,*) ' La dimensión de la matriz excede 25 x 25 '
END IF
DO i=1, n, 1
  READ(4,*) (a(i,j), j=1, n, 1)
END DO
WRITE(7,*) " Matriz A"
WRITE(7,*)
DO i= 1, n, 1
  WRITE(7,*) (a(i,j), j=1, n, 1)
END DO
! Calcula la traza de A

traza = 0.0
DO i = 1, n, 1
  traza = traza + a(i, i)
END DO
WRITE(7,*)
WRITE(7,*) " La traza de la matriz es", traza

```


Apéndice A

```
CLOSE(4)
CLOSE(7)
STOP
END PROGRAM traza_matriz
```

Ejemplo de la ejecución del programa.

Datos entrada (matriz.dat):

```
4 4
1.0 2.0 3.0 4.0
5.0 6.0 7.0 8.0
9.0 10.0 11.0 12.0
13.0 14.0 15.0 16.0
```

Datos salida (out.dat):

```
Matriz  A
1.00000      2.00000      3.00000      4.00000
5.00000      6.00000      7.00000      8.00000
9.00000      10.0000     11.0000     12.0000
13.0000      14.0000     15.0000     16.0000

La traza de la matriz es      34.0000
```

8.6

```
PROGRAM EjMatmul2
IMPLICIT NONE
INTEGER, PARAMETER :: nmax =25, mmax =25
INTEGER :: n1, m1, n2, m2
REAL ,DIMENSION (1:nmax,1:mmax) :: a, b, c
INTEGER :: i, j, k

!
!
! Lee las matriz A de dimensiones N1 x M1 y
! B de dimensiones N2 x M2 y calcula la matriz producto
! C = A x B
!
!
! nmax    ... dimension maxima de las matrices (PARAMETER)
! n1 x m1 ... dimensiones de la matriz A
! n2 x m2 ... dimensiones de la matriz B
! a       ... matriz A
! b       ... matriz B
! c       ... matriz C = A x B
!
! Variables auxiliares :
! i,j,k   ... indices bucles DO
!
```

```

OPEN (4, FILE="matrices.dat")
OPEN (7, FILE="out.dat")

```

! Lee la matriz A

```

READ(4,*) n1, m1
WRITE(*,*)
WRITE(*,*) "La matriz A tiene ", n1, " filas y ", m1," columnas"
WRITE(*,*)

DO i=1, n1, 1
  READ(4,*) (a(i,j), j=1, m1, 1)
END DO
WRITE(7,*) " Matriz A"
WRITE(7,*)
DO i= 1, n1, 1
  WRITE(7,*) (a(i,j), j=1, m1, 1)
END DO

```

! Lee la matriz B

```

READ(4,*) n2, m2
WRITE(*,*) "La matriz B tiene ", n2, " filas y ", m2," columnas"
WRITE(*,*)

DO i=1, n2, 1
  READ(4,*) (b(i,j), j=1, m2, 1)
END DO
WRITE(7,*)
WRITE(7,*) " Matriz B"
WRITE(7,*)
DO i= 1, n2, 1
  WRITE(7,*) (b(i,j), j=1, m2, 1)
END DO

```

! Comprueba que las matrices son congruentes y sus dimensiones menores que 25 x 25

```

IF (n2 /= m1) THEN
  WRITE(7,*) ' Las matrices A y B no son congruentes '
  STOP
ELSE IF ((n1 > 25).or.(m1 > 25)) THEN
  WRITE(7,*) ' La dimensión de la matriz A excede 25 x 25 '
  STOP
ELSE IF ((n2 > 25).or.(m2 > 25)) THEN
  WRITE(7,*) ' La dimensión de la matriz B excede 25 x 25 '
  STOP
END IF

```

! Calcula $C = A \times B$, y escribe el resultado

```

c(1:n1,1:m2) = 0.0
DO i=1, n1, 1
  DO j = 1, m2, 1

```

Apéndice A

```
      DO k = 1, m1, 1
        c(i,j) = c(i,j) + a(i,k)*b(k,j)
      END DO
    END DO
  END DO
  WRITE(7,*)
  WRITE(7,*) " Matriz producto C = A x B"
  WRITE(7,*)
  DO i= 1, n1, 1
    WRITE(7,*) (c(i,j), j=1, m2, 1)
  END DO
```

!
! El calculo tambien puede realizarse utilizando la funcion intrínseca SUM
!

```
c(1:n1,1:m2) = 0.0
DO i=1, n1, 1
  DO j = 1, m2, 1
    c(i,j) = SUM(a(i,1:m1)*b(1:m1,j))
  END DO
END DO
WRITE(7,*)
WRITE(7,*) " Matriz producto C = A x B – utilizando SUM"
WRITE(7,*)
DO i= 1, n1, 1
  WRITE(7,*) (c(i,j), j=1, m2, 1)
END DO
```

!
! El calculo tambien puede realizarse utilizando la funcion intrínseca MATMUL
!

```
c(1:n1,1:m2) = MATMUL(a(1:n1,1:m1),b(1:n2,1:m2))
WRITE(7,*)
WRITE(7,*) " Matriz producto C = A x B – utilizando MATMUL"
WRITE(7,*)
DO i= 1, n1, 1
  WRITE(7,*) (c(i,j), j=1, m2, 1)
END DO

CLOSE(11)
STOP
END PROGRAM EjMatmul2
```

Ejemplo de la ejecución del programa.**Datos entrada (matrices.dat):**

```
2 3
2.0 3.0 4.0
3.0 4.0 5.0
3 2
2.0 3.0
3.0 4.0
4.0 5.0
```

Datos salida (out.dat)

Matriz A

```
2.00000      3.00000      4.00000
3.00000      4.00000      5.00000
```

Matriz B

```
2.00000      3.00000
3.00000      4.00000
4.00000      5.00000
```

Matriz producto C = A x B

```
29.0000      38.0000
38.0000      50.0000
```

Matriz producto C = A x B - utilizando SUM

```
29.0000      38.0000
38.0000      50.0000
```

Matriz producto C = A x B - utilizando MATMUL

```
29.0000      38.0000
38.0000      50.0000
```

Práctica 9

9.1

```

PROGRAM Vibracion1
!
! Este programa calcula la energia vibracional de una molecula diatomica
! utilizando la aproximacion del oscilador armonico.
! El programa utiliza la funcion externa OsciladorArmonico.
!
! nmax ... numero de niveles a calcular (0–nmax)
! n     ... numero cuantico n
! ve    ... frecuencia fundamental de vibracion (cm–1)
! En    ... energia del nivel vibracional n (J)
!
IMPLICIT NONE
INTEGER :: n,nmax
REAL :: ve, En
!
! Interfaz de la funcion
!
INTERFACE
FUNCTION OsciladorArmonico(ve,n) RESULT(E)
  IMPLICIT NONE
  REAL :: E
  REAL, INTENT(IN) :: ve
  INTEGER, INTENT(IN) :: n
END FUNCTION OsciladorArmonico
END INTERFACE
!
WRITE(*,*) ' Este programa calcula la energia vibracional '
WRITE(*,*) ' de una molecula diatomica en la aproximacion del oscilador armonico '
WRITE(*,*)
WRITE(*,*) ' Introduzca la frecuencia fundamental de vibracion ve (cm–1) '
READ(*,*) ve
WRITE(*,*) ' Introduzca el numero cuantico del mayor nivel a calcular (0,1,2,...), nmax '
READ(*,*) nmax
!
! Calcula En
!
DO n= 0,nmax,1
  En = OsciladorArmonico(ve,n)
  WRITE(*,*) ' n = ',n,' En = ',En,' J'
END DO

STOP
END PROGRAM Vibracion1
!
! Funcion para el calculo de la energia vibracional
! de una molecula diatomica utilizando la aproximacion
! del oscilador armonico.
!
! n     ... numero cuantico n
! h     ... constante de Planck (J.s)      (PARAMETER)

```

```

! c      ... velocidad de la luz (cm.s-1) (PARAMETER)
! ve     ... frecuencia fundamental de vibracion (cm-1)
! E      ... energia del nivel vibracional n (J)
!
FUNCTION OsciladorArmonico(ve,n) RESULT(E)
IMPLICIT NONE
REAL :: E
REAL, INTENT(IN) :: ve
INTEGER, INTENT(IN) :: n
REAL,PARAMETER :: h = 6.62608E-34
REAL,PARAMETER :: c = 2.99792458E10
!
E = h*c*ve*(n + 0.5)
!
RETURN
END FUNCTION OsciladorArmonico

```

Ejemplo de la ejecución del programa.

```
C:\EPAQ\Valvanera>vibracion1
```

```
Este programa calcula la energia vibracional
de una molecula diatomica en la aproximacion del oscilador armonico
```

```
Introduzca la frecuencia fundamental de vibracion ve (cm-1) 2359.61
```

```
Introduzca el numero cuantico del mayor nivel a calcular (0,1,2,...), nmax 3
```

```

n =          0 En =      0.234362E-19 J
n =          1 En =      0.703087E-19 J
n =          2 En =      0.117181E-18 J
n =          3 En =      0.164054E-18 J

```

```
Program completed
Press Enter to Continue.
```

Práctica 10

10.2

```

PROGRAM Matriz_Traspuesta
!
! Programa que lee una matriz N filas( N <= 20)
! y M columnas (N<= 20)
! El programa utiliza una subrutina para leer los datos
! y otra para escribirlos .
!
! Lee las matriz A de dimensiones N x M y calcula su traspuesta
!
!   nfilas  ... numero maximo de fila (PARAMETER)
!   ncols   ... numero maximo de columnas (PARAMETER)
!   n x m   ... dimensiones de la matriz A
!   a       ... matriz A
!   at      ... matriz traspuesta de A
!
IMPLICIT NONE
INTEGER, PARAMETER :: nfilas = 20, ncols = 20
INTEGER :: n,m
REAL, DIMENSION(NFilas,NCols) :: a, at
!
! Interfaces de las subrutinas
!
INTERFACE
SUBROUTINE Leer_Matriz(a, nmax, mmax, n,m)
  IMPLICIT NONE
  INTEGER, INTENT(IN) :: nmax, mmax
  INTEGER, INTENT(OUT) :: n, m
  REAL, DIMENSION(:,:),INTENT(OUT):: a
END SUBROUTINE Leer_Matriz
SUBROUTINE Imprime_Matriz (b,n,m)
  IMPLICIT NONE
  INTEGER, INTENT(IN) :: n, m
  REAL, DIMENSION(:,:),INTENT(IN) :: b
END SUBROUTINE Imprime_Matriz
SUBROUTINE Traspuesta (a, at, n, m)
  IMPLICIT NONE
  INTEGER, INTENT(IN) :: n, m
  REAL, DIMENSION(:,:),INTENT(IN):: a
  REAL, DIMENSION(:,:),INTENT(OUT):: at
END SUBROUTINE Traspuesta
END INTERFACE
!
! Lee la matriz y la imprime
!
CALL Leer_Matriz(a, nfilas, ncols, n,m)
WRITE(*,*)
WRITE(*,*) ' Matriz original '
WRITE(*,*)
  CALL Imprime_Matriz(a,n,m)

```

```

!
! Calcula la matriz traspuesta y la imprime
!
  CALL Traspuesta(a, at, n, m)
  WRITE(*,*)
  WRITE(*,*) ' Matriz traspuesta '
  WRITE(*,*)
  CALL Imprime_Matriz(at, n,m)
!
  STOP
  END PROGRAM Matriz_Traspuesta
!
! Subrutina que lee una matriz por filas
!
  SUBROUTINE Leer_Matriz(a, nmax, mmax, n,m)
  IMPLICIT NONE
  INTEGER, INTENT(IN) :: nmax, mmax
  INTEGER, INTENT(OUT) :: n, m
  REAL, DIMENSION(:,:),INTENT(OUT):: a
!
  INTEGER :: i, j

  OPEN(4, FILE = 'matriz.dat')

  READ(4,*) n,m
  WRITE(*,*) ' La matriz tiene '
  WRITE(*,*) ' Filas ', n, ' Columnas ',m
!
! Comprueba que la matriz no excede los limites.
!
  IF ((n > nmax).OR.(m>mmax)) THEN
    WRITE(*,*) ' N debe ser menor que ', nmax
    WRITE(*,*) ' M debe ser menor que ', mmax
    STOP
  END IF
!
! Lee la matriz por filas
!
  READ(4,*) ((a(i,j), j=1,m),i=1,n)

  CLOSE(4)

  RETURN
  END SUBROUTINE Leer_Matriz
!
! Subrutina que imprime una matriz por filas
!
  SUBROUTINE Imprime_Matriz (b,n,m)

  IMPLICIT NONE
  INTEGER, INTENT(IN) :: n, m
  REAL, DIMENSION(:,:),INTENT(IN) :: b
  INTEGER :: i,j

```


Apéndice A

```
OPEN(11, FILE='matriz.out')

DO i= 1,n
  WRITE (*,*) (b(i,j), j=1,m)
END DO

CLOSE(11)

RETURN
END SUBROUTINE Imprime_Matriz
!
! Subrutina que calcula la matriz traspuesta
!
SUBROUTINE Traspuesta (a,at, n,m)
IMPLICIT NONE
INTEGER, INTENT(IN) :: n, m
REAL, DIMENSION(:,:),INTENT(IN):: a
REAL, DIMENSION(:,:),INTENT(OUT):: at
INTEGER :: i,j

DO i = 1, n, 1
  DO j = 1, m, 1
    at(j,i)= a(i,j)
  END DO
END DO

RETURN
END SUBROUTINE Traspuesta
```

Ejemplo de la ejecución del programa

Datos entrada (matriz.dat):

```
4 4
1.0 2.0 3.0 4.0
5.0 6.0 7.0 8.0
9.0 10.0 11.0 12.0
13.0 14.0 15.0 16.0
```

La ejecución del programa dará como resultado en la pantalla:

```
C:\EPAQ\Valvanera>traspuesta

La matriz tiene
Filas          4 Columnas          4

Matriz original

1.00000          2.00000          3.00000          4.00000
5.00000          6.00000          7.00000          8.00000
9.00000          10.0000         11.0000         12.0000
13.0000          14.0000         15.0000         16.0000
```

Matriz traspuesta

1.00000	5.00000	9.00000	13.0000
2.00000	6.00000	10.0000	14.0000
3.00000	7.00000	11.0000	15.0000
4.00000	8.00000	12.0000	16.0000

Program completed
Press Enter to Continue.

Microproyectos

B.1. Cinética Química: estudio de la cinética de reacciones químicas utilizando determinaciones gráficas

En este microproyecto se utilizarán las ecuaciones integradas de velocidad de reacciones químicas con el fin de obtener, a partir de las correspondientes representaciones gráficas, la constante cinética del proceso y, posteriormente, las dependencia de las concentraciones de los reactivos y productos con el tiempo.

- (a) Considere los siguientes datos experimentales para la descomposición catalítica de amoníaco sobre wolframio a 1100K:

t / s	0	100	200	300	400	500	1000
P / Torr	200	186	173	162	152	141	88

- (i) Determine el orden de reacción (dos, uno o cero) y la constante cinética correspondiente.
 (ii) Estime la presión parcial de los productos H_2 y N_2 , y la presión total, con el tiempo.
- (b) A veces, una reacción química no tiene un orden global de reacción e incluso puede no tener órdenes parciales definidos respecto de cada componente. Por ejemplo, en la descomposición catalítica de amoníaco sobre Platino a 1273 K, debido a la fuerte adsorción del producto sobre la superficie, la ecuación cinética del proceso viene dada por:

$$\frac{dP_{NH_3}}{dt} = -k \frac{P_{NH_3}}{P_{H_2}}$$

Sin embargo, obteniendo su ecuación cinética integrada se podría determinar, mediante una representación gráfica adecuada, su constante cinética.

Utilizando los siguientes datos experimentales para la reacción citada:

t / s	0	30	60	100	160	200	250
P / Torr	100	88	84	80	77	74	72

- (i) Determine la constante cinética k .
 (ii) Estime los valores de la presión parcial de los productos H_2 y N_2 , y la presión total, con el tiempo.

Reacciones químicas: ecuaciones de velocidad y ecuaciones de velocidad integradas

La velocidad de una reacción química v se define como:

$$v = \frac{1}{\nu_i} \frac{d|I|}{dt} \quad (\text{B.1})$$

donde $|I|$ es la concentración del reactivo o producto I de la reacción y ν_i es su correspondiente coeficiente estequiométrico, que será negativo para los reactivos y positivo para los productos.

La velocidad de una reacción química depende de la composición y la temperatura del sistema químico reaccionante. Esta dependencia se expresa mediante una ley de velocidad, cuya forma general es:

$$v = f(|A|, |B|, \dots) \quad (\text{B.2})$$

donde la velocidad de la reacción química se expresa como función de las concentraciones de todas las especies químicas presentes en el sistema. La ecuación cinética (ley de velocidad) de una reacción química se determina experimentalmente, y en general no se puede inferir a partir de la ecuación química de la reacción.

Muchas reacciones químicas obedecen a leyes de velocidad de la forma:

$$v = k |A|^a |B|^b \dots \quad (\text{B.3})$$

donde el coeficiente k se conoce como la constante cinética de la reacción química, que es independiente de las concentraciones de los componentes de la reacción pero que depende apreciablemente de la temperatura. Las potencias a, b, \dots , a las que se elevan las concentraciones de las especies químicas A, B, \dots (ya sean reactivos o productos) en esta ecuación cinética se denominan órdenes parciales de la reacción respecto a las especies químicas correspondientes. Y se conoce como orden global de la reacción a la suma de los órdenes parciales $a + b + \dots$

Por ejemplo, una reacción química cuya ecuación cinética es:

$$v = k |A| |B|^2$$

es de primer orden respecto de A , de segundo orden respecto de B y de orden global tres, o de tercer orden. Si su ecuación cinética fuera:

$$v = k |A|^3$$

sería de tercer orden respecto de A y de tercer orden global. Y si su ecuación de velocidad fuera:

$$v = k$$

la reacción sería de orden cero. Esta ecuación cinética sólo se encuentra en reacciones heterogéneas.

Las leyes de velocidad son ecuaciones diferenciales. Por lo tanto, pueden integrarse si se necesita conocer las concentraciones de las especies con el tiempo.

Las ecuaciones cinéticas más complejas pueden ser integradas numéricamente. Sin embargo, existen algunos casos de ecuaciones cinéticas más sencillas en los que se pueden obtener las soluciones analíticas. A continuación se muestran algunos ejemplos:

Orden 1. $A \rightarrow X + \dots$

$$\frac{d|A|}{dt} = -k|A| \quad |A| = |A|_0 e^{-kt} \quad (\text{B.4})$$

Orden 2. $A + A \rightarrow X + \dots$

$$\frac{d|A|}{dt} = -k|A|^2 \quad |A| = \frac{|A|_0}{1 + kt|A|_0} \quad (\text{B.5})$$

Orden 2. $A + B \rightarrow X + \dots$

$$\frac{d|A|}{dt} = -k|A||B| \quad kt = \frac{1}{|A|_0 - |B|_0} \left\{ \ln \left(\frac{|A|_0}{|A|} \right) - \ln \left(\frac{|B|_0}{|B|} \right) \right\} \quad (\text{B.6})$$

En todos los casos, reorganizando las ecuaciones cinéticas integradas, se puede realizar una representación gráfica de los datos, esperando un comportamiento lineal. La aplicación del método de mínimos cuadrados sobre estos datos puede proporcionar los coeficientes de la regresión lineal, a partir de los cuales se puede determinar la constante cinética del proceso químico estudiado.

Orden 1.

$$\ln \left(\frac{|A|}{|A|_0} \right) = -kt \quad (\text{B.7})$$

Orden 2.

$$\frac{1}{|A|} = \frac{1}{|A|_0} - kt \quad (\text{B.8})$$

Orden 2.

$$\ln \left(\frac{|B|/|B|_0}{|A|/|A|_0} \right) = (|B|_0 - |A|_0)kt \quad (\text{B.9})$$

B.1.1. Ajuste de mínimos cuadrados a una recta con pesos

Suponga que dispone de un conjunto de n pares de datos (x_i, y_i) , con $i = 1, \dots, n$, y que entre ellos existe una relación lineal

$$y_i = A \cdot x_i + B + \varepsilon_i \quad (\text{B.10})$$

donde A es la pendiente de la recta, B la ordenada en el origen y ε_i es el error en cada una de las medidas y_i obtenidas cuando la variable controlada X toma el valor x_i . Se supone que los valores de x_i no tienen asociado un error.

En el método de mínimos cuadrados, a y b se calculan como los valores que minimizan la suma

$$\begin{aligned}
 R &= \sum_{i=1}^N w_i (y_i - (a \cdot x_i + b))^2 \\
 &= \sum_{i=1}^N w_i (y_i - y_{\text{calculado}})^2
 \end{aligned}
 \tag{B.11}$$

donde w_i es un coeficiente que da el peso del punto (x_i, y_i) en el ajuste, y R es la suma de los cuadrados de los residuos de la recta ajustada.

En una regresión lineal simple los pesos de todos los pares de datos son iguales, $w_i = 1$. En el caso de una regresión lineal ponderada el peso de cada punto, w_i , viene dado por

$$w_i = \frac{1}{\sigma^2(y_i)}
 \tag{B.12}$$

Como experimentalmente sólo es posible obtener $s^2(y_i)$, este coeficiente se calcula como

$$w_i = \frac{1}{s^2(y_i)}
 \tag{B.13}$$

Para obtener las expresiones que permiten calcular a y b , se aplica a R la condición de mínimo

$$\begin{aligned}
 \frac{\partial R}{\partial a} &= 0 \\
 \frac{\partial R}{\partial b} &= 0
 \end{aligned}
 \tag{B.14}$$

Al aplicar esta condición, se obtiene un sistema de dos ecuaciones con dos incógnitas (a y b) cuya solución es

$$a = \frac{\sum_{i=1}^n w_i \cdot \sum_{i=1}^n w_i (x_i y_i) - \sum_{i=1}^n w_i x_i \sum_{i=1}^n w_i y_i}{\Delta}
 \tag{B.15}$$

$$b = \frac{\sum_{i=1}^n w_i x_i^2 \cdot \sum_{i=1}^n w_i y_i - \sum_{i=1}^n w_i x_i \sum_{i=1}^n w_i (x_i y_i)}{\Delta}
 \tag{B.16}$$

donde

$$\Delta = \sum_{i=1}^n w_i \cdot \sum_{i=1}^n w_i x_i^2 - \left(\sum_{i=1}^n w_i x_i \right)^2
 \tag{B.17}$$

La varianza del ajuste, s^2 , viene dada por

$$s^2 = \frac{R}{N-2} = \frac{\sum_{i=1}^N w_i \cdot (y_i - (a \cdot x_i + b))^2}{N-2} \quad (\text{B.18})$$

La varianza de la pendiente, $s^2(a)$, viene dada por

$$s^2(a) = \frac{s^2 \cdot \sum_{i=1}^n w_i}{\Delta} \quad (\text{B.19})$$

y la varianza de la ordenada en el origen, $s^2(b)$, por

$$s^2(b) = \frac{s^2 \cdot \sum_{i=1}^N w_i x_i^2}{\Delta} \quad (\text{B.20})$$

El coeficiente de correlación lineal, $r(x, y)$, que es una estima del coeficiente de correlación lineal $\rho(x, y)$, es

$$r = \frac{N \cdot \sum_{i=1}^N w_i x_i y_i - \sum_{i=1}^N w_i x_i \cdot \sum_{i=1}^N w_i y_i}{\left[\left\{ N \cdot \sum_{i=1}^N w_i x_i^2 - \left(\sum_{i=1}^N w_i x_i \right)^2 \right\} \cdot \left\{ N \cdot \sum_{i=1}^N w_i y_i^2 - \left(\sum_{i=1}^N w_i y_i \right)^2 \right\} \right]^{1/2}} \quad (\text{B.21})$$

Si los datos se ajustan a una recta, este valor debe ser próximo a -1,00, si la pendiente es negativa, y 1,00, si la pendiente es positiva.

Un tipo de ajuste que se realiza con mucha frecuencia es aquel en el que se ajustan datos a una exponencial

$$y_i = b' \cdot \exp [ax_i] \quad (\text{B.22})$$

y para obtener los valores de los coeficientes a y b , se ajusta a una recta el logaritmo de la variable dependiente, y_i

$$y'_i = \ln(y_i) = ax_i + \ln b' = ax_i + b \quad (\text{B.23})$$

donde $b = \ln b'$.

En este tipo de ajustes, w_i viene dado por

$$w_i = \frac{y_i^2}{s^2(y_i)} \quad (\text{B.24})$$

Como normalmente no se conoce $s^2(y_i)$, es frecuente suponer que todas las medidas tienen la misma incertidumbre, ($s^2(y_i) \approx \text{cte.}$), y por tanto

$$w_i \propto y_i^2 \quad (\text{B.25})$$

Ejemplo B.1 *Ajuste de datos a una recta.*

Escriba un programa que lea un número arbitrario de pares de datos x_i, y_i , y calcule la recta de ajuste, el coeficiente de correlación lineal del ajuste, r , la varianza del ajuste, s^2 , y las varianzas de a , $s^2(a)$, y b , $s^2(b)$.

El programa debe permitir realizar una regresión lineal simple o ponderada.

Además, el programa debe considerar la posibilidad de que los datos se ajusten a una expresión lineal

$$y_i = ax_i + b$$

o exponencial

$$y_i = b' \exp[a \cdot x_i]$$

En este caso, el programa debe realizar la transformación

$$y'_i = \ln(y_i) = ax_i + \ln b' = ax_i + b$$

antes de realizar el ajuste.

En el programa considere que el vector más grande con el que puede trabajar tiene 100 elementos.

Descripción del algoritmo

El programa incluirá los siguientes pasos:

- i abrir los ficheros `input.dat`, `resul.dat`, `tabla.dat`
- ii leer del fichero `input.dat` el número de valores guardado en ese fichero, `np`
- iii leer del fichero `input.dat` el tipo de ajuste (IW) :
 - IW = 0: regresión lineal simple. Lee de `input.dat` x_i, y_i
 - IW = 1: regresión lineal ponderada. Lee de `input.dat` $x_i, y_i, s(y_i)$
 - IW = 2: regresión lineal simple de una exponencial. Lee de `input.dat` x_i, y'_i
 - IW = 3: regresión lineal simple de una exponencial. Lee de `input.dat` x_i, y'_i
- iv calcular w_i
 - IW = 0: $w_i = 1$
 - IW = 1: $w_i = 1/s^2(y_i)$
 - IW = 2: $w_i = 1$
 - IW = 3: $w_i = y_i^2$
- v si IW = 2 ó IW = 3, $y(i) = \text{LOG}(y(i))$
- vi calcular $a, b, s^2, s^2(a), s^2(b)$ y r
- vii escribir $a, b, s^2, s^2(a), s^2(b)$ y r en el fichero `resul.dat`
- viii escribir $x_i, y_i, a * x(i) + b, y_i - (a * x_i + b)$ en el fichero `tabla.dat`

Traducción del algoritmo a un programa Fortran

Como puede observar, este programa esta basado en el descrito en la página 232 para el ajuste lineal simple. El programa se ha modificado incluyendo las instrucciones necesarias para poder hacer un ajuste lineal ponderado

```

PROGRAM ajuste_recta
!
! Este programa calcula los parametros del ajuste a una recta mediante
! el metodo de minimos cuadrados simple o ponderado.
!
! nmax ... numero maximo de datos (PARAMETER)
! np ... numero de datos
! iw ... tipo de ajuste
! x ... variable x
! y ... variable y
! sy ... desviacion tipica y
! w ... peso de y
! a ... pendiente
! b ... ordenada en el origen
! ycalc ...  $a \cdot x_i + b$ 
! residuo ...  $y - y_{calc}$ 
! sumx ... sum  $x_i$ 
! sumy ... sum  $y_i$ 
! sumxy ... sum  $x_i \cdot y_i$ 
! sumx2 ... sum  $x_i^2$ 
! sumy2 ... sum  $y_i^2$ 
! delta ...  $n \cdot \text{sum}x2 - \text{sum}x^2$ 
! s2 ... varianza del ajuste
! s2a ... varianza de a
! s2b ... varianza de b
! r ... coeficiente de correlación lineal
!
! Variables auxiliares :
! i ... indice bucles DO
!
IMPLICIT NONE
INTEGER, PARAMETER :: nmax = 100
INTEGER :: i, np
INTEGER :: iw
REAL, DIMENSION(1:nmax) :: x, y, sy
REAL, DIMENSION(1:nmax) :: w, ycalc, residuo
REAL :: a, b
REAL :: sumx, sumy, sumxy, sumx2, sumy2, delta
REAL :: sumw
REAL :: s2, s2a, s2b, r
!
OPEN(4, FILE='datos.dat')
OPEN(7, FILE='resul.dat')
OPEN(8, FILE='tabla.dat')
!
! Lee el numero de datos (n) y comprueba que el numero de datos
! no es muy pequeño ( $n \leq 2$ ), ni muy grande ( $n > n_{max}$ )
!

```

```

READ(4,*) np
IF (np < 0) THEN
  WRITE(*,*) 'Np debe ser un entero positivo'
  STOP
ELSE IF (np < 2) THEN
  WRITE(*,*) 'Para calcular una recta al menos se necesitan 2 valores'
  STOP
ELSE IF (np > nmax) THEN
  WRITE(*,*) 'Np debe ser menor o igual que', nmax
  STOP
ELSE
  WRITE(7,*) 'El fichero datos.dat contiene ', np, ' datos'
END IF

```

! Lee el tipo de ajuste (IW)

```

READ(4,*) iw
IF (iw == 0) THEN
  WRITE(7,*) 'regresion lineal simple. w(i) = 1.0 '
ELSE IF (iw == 1) THEN
  WRITE(7,*) 'regresion lineal ponderarada. w(i) = 1.0/sy(i)**2 '
ELSE IF (iw == 2) THEN
  WRITE(7,*) 'regresion lineal simple. w(i) = 1.0'
ELSE IF (iw == 3) THEN
  WRITE(7,*) 'regresion lineal ponderada. w(i) = y(i)**2 '
ELSE
  WRITE(*,*) 'Valor erroneo de iw. iw = ', iw
  STOP
END IF

```

! Lee los datos

```

IF (iw == 1) THEN
  DO i = 1, np, 1
    READ(4,*) x(i), y(i), sy(i)
  END DO
ELSE
  DO i = 1, np, 1
    READ(4,*) x(i), y(i)
  END DO
END IF
WRITE(*,*) iw, i

```

! Calcula los pesos, w_i

```

w = 1.0
IF (iw == 1) THEN
  w(1:np) = 1.0/sy(1:np)**2
ELSE IF (iw == 3) THEN
  w(1:np) = y(1:np)**2
END IF
IF (iw > 1) y(1:np) = LOG(y(1:np))

```

! Calcula a y b

```

sumw = SUM(w(1:np))
sumx = SUM(w(1:np)*x(1:np))
sumy = SUM(w(1:np)*y(1:np))
sumxy = SUM(w(1:np)*x(1:np)*y(1:np))
sumx2 = SUM(w(1:np)*x(1:np)**2)
delta = sumw*sumx2 - sumx*sumx

```

$$a = (\text{sumw} * \text{sumxy} - \text{sumx} * \text{sumy}) / \text{delta}$$

$$b = (\text{sumx}^2 * \text{sumy} - \text{sumx} * \text{sumxy}) / \text{delta}$$

```

WRITE(7,*)
WRITE(7,*) ' sumw =', sumw
WRITE(7,*) ' sumx =', sumx, ' sumy =', sumy
WRITE(7,*) ' sumxy =', sumxy, ' sumx2 =', sumx2
WRITE(7,*) ' delta =', delta
WRITE(7,*) ' pendiente, a =', a
WRITE(7,*) ' ordenada origen, b = ', b

```

```

!
ycalc(1:np) = a*x(1:np) +b
residuo(1:np) = y(1:np) - ycalc(1:np)
DO i = 1, np, 1
  WRITE(8,*) x(i), y(i), ycalc(i), residuo(i)
END DO

```

! Calcula s, s(a) y s(b) y r

```

s2 = SUM(w(1:np)*residuo(1:np)**2)/REAL(np -2)
s2a = s2*sumw/delta
s2b = s2*sumx2/delta
sumy2 = SUM(w(1:np)*y(1:np)**2)
r = (np*sumxy - sumx*sumy)/SQRT((np*sumx2 - sumx**2)*(np*sumy2 - sumy**2))

```

```

WRITE(7,*)
WRITE(7,*) ' s2 =', s2, ' s2(a) =', s2a, ' s2(b) =', s2b
WRITE(7,*) ' s =', SQRT(s2), ' s(a) =', SQRT(s2a), ' s(b) =', SQRT(s2b)
WRITE(7,*) ' r =', r

```

```

!
STOP
END PROGRAM ajuste_recta

```

Aplicación. Cálculo de la constante de velocidad de una reacción de primer orden utilizando una técnica de regresión lineal ponderada

En un recipiente, en el que se había hecho vacío, se introdujo N_2O_5 a una temperatura de 44.9°C . El resultado de medidas de la presión de N_2O_5 en el recipiente con el tiempo se indica en la tabla siguiente:

tiempo/ minutos	10	20	30	40	50	60	70	80	90
$P(\text{N}_2\text{O}_5) / \text{Torr}$	248	185	142	106	79.0	56.5	43.4	32.8	25.3

Asumiendo que la reacción sigue una cinética de primer orden, calcule la constante de velocidad, k , para la descomposición de N_2O_5 y estime la presión de éste que se introdujo inicialmente en el recipiente.

Dado que la cinética es de primer orden,

$$\frac{d|\text{N}_2\text{O}_5|}{dt} = -k|\text{N}_2\text{O}_5|$$

Suponiendo que el gas es ideal,

$$[N_2O_5] = P(N_2O_5) / (RT)$$

de modo que

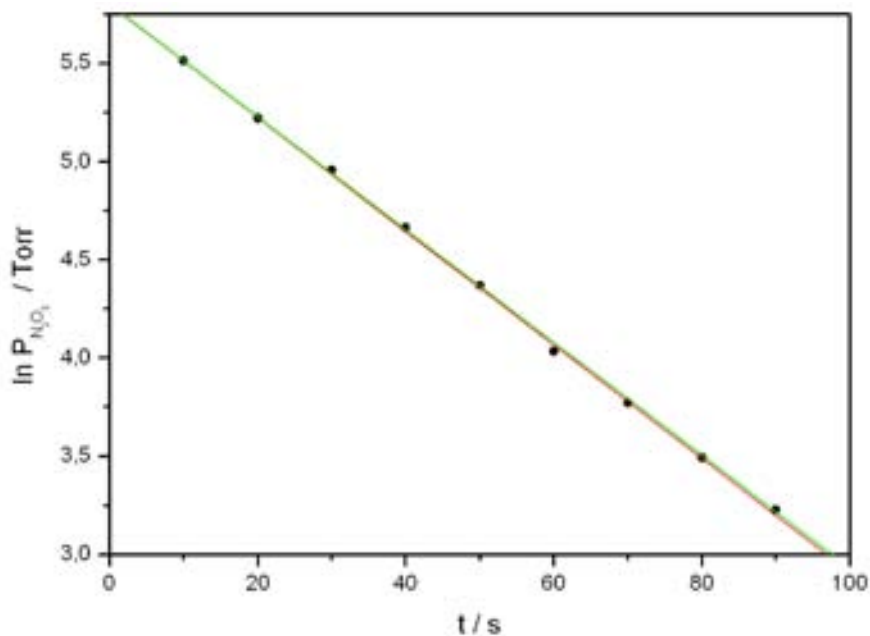
$$\frac{dP(N_2O_5)}{dt} = -kP(N_2O_5) \rightarrow \ln P(N_2O_5) = \ln P_0(N_2O_5) - kt$$

Así, los coeficientes a y b de la regresión lineal $y = ax + b$ proporcionan una estima de $k = -a$ y $\ln P_0(N_2O_5) = b$.

La siguiente tabla recoge el resultado de los distintos parámetros que resultan de emplear una regresión lineal simple, $w_i = 1,0$, y ponderada, $w_i = y_i^2$, en el ajuste de los datos experimentales.

ajuste	k / min^{-1}	$b \pm s(b)$	$P(N_2O_5) \pm s(P) / \text{Torr}$	r
$w_i = 1$	$(2,89 \pm 0,26)10^{-2}$	$5,80 \pm 0,015$	$330,3 \pm 4,9$	-0.9997
$w_i = y_i^2$	$(2,87 \pm 0,25)10^{-2}$	$5,802 \pm 0,007$	$331,0 \pm 2,3$	-1.0002

La representación de los datos experimentales (puntos) frente a las datos del ajuste simple (línea roja) y ponderado (línea verde) con Origin



muestra que en este caso no se aprecian diferencias significativas entre el ajuste lineal simple y ponderado. Sin embargo, este no es siempre el caso.

B.2. Espectrofotometría: determinación de la composición molar de una mezcla a partir de medidas espectrofotométricas

Una aplicación del método de la espectrofotometría en análisis químico es la determinación de la composición molar de mezclas de varios componentes, asumiendo que en las condiciones experimentales utilizadas se cumple para todas las especies la ley de Lambert-Beer. En estos trabajos sucede que finalmente el problema que se plantea es resolver un sistema de ecuaciones lineales. En este microproyecto, se propone el análisis de los siguientes datos de espectrofotometría de una mezcla de cinco componentes químicos en disolución:

λ	ε_1	ε_2	ε_3	ε_4	ε_5	A_{obs}
λ_1	100	10	2	1	0.5	0.1135
λ_2	12	120	10	5	0.9	0.2218
λ_3	30	30	90	10	2	0.2700
λ_4	1	4	18	140	24	0.2992
λ_5	2	4	8	16	120	0.1350

(a) Determinar las concentraciones molares de los componentes mezcla

Ley de Lambert-Beer

En espectroscopia de absorción, la relación entre la intensidad de luz transmitida I y la intensidad de luz incidente I_0 , a una frecuencia espectral ν determinada, se denomina Transmitancia, T , de la muestra a dicha frecuencia:

$$T = \frac{I}{I_0} \quad (\text{B.26})$$

Empíricamente se observa que la intensidad transmitida I depende del camino óptico l y de la concentración molar c de la especie que absorbe de acuerdo con la ley de Lambert-Beer:

$$I = I_0 10^{-\varepsilon c l} \quad (\text{B.27})$$

La cantidad ε es el coeficiente de absorción decimal molar (antiguamente, y ampliamente utilizado ahora, coeficiente de extinción molar). Este coeficiente depende de la naturaleza de la especie que absorbe y de la frecuencia de la radiación incidente, y toma sus más altos valores donde se produce la absorción más intensa. Si se introduce el concepto de Absorbancia (A) de la muestra a una frecuencia espectral dada según

$$A = \log \frac{I_0}{I} = -\log T \quad (\text{B.28})$$

entonces la ley de Lambert-beer se escribe:

$$A = \varepsilon \cdot c \cdot l \quad (\text{B.29})$$

donde el producto $\varepsilon \cdot c \cdot l$ se conoce como densidad óptica de la muestra.

Si una disolución contiene i componentes, y asumimos que se cumple la ley de Lambert-Beer para todos ellos a una frecuencia espectral determinada, entonces la absorbancia de la disolución a dicha frecuencia vendrá dada por

$$A = \sum_i \varepsilon_i \cdot c_i \cdot l \quad (\text{B.30})$$

Si se dispone de medidas de Absorbancia de la disolución multicomponente a varias frecuencias espectrales, y asumimos que en todas ellas se cumple la ley de Lambert-Beer para todos los componentes de la mezcla, entonces podríamos generalizar de la forma

$$A_{obs,j} = \sum_i \varepsilon_{ji} \cdot c_i \cdot l \quad (\text{B.31})$$

Como en general el camino óptico es 1cm, conociendo las Absorbancias de la mezcla a distintas frecuencias espectrales y los correspondientes coeficientes de absorción decimal molar de los componentes, la determinación de las concentraciones molares se realizará resolviendo el sistema de ecuaciones:

$$\begin{aligned} \varepsilon_{11} \cdot c_1 + \varepsilon_{12} \cdot c_2 + \dots + \varepsilon_{1m} \cdot c_m &= A_{obs,1} \\ \varepsilon_{21} \cdot c_1 + \varepsilon_{22} \cdot c_2 + \dots + \varepsilon_{2m} \cdot c_m &= A_{obs,2} \\ \vdots & \\ \varepsilon_{n1} \cdot c_1 + \varepsilon_{n2} \cdot c_2 + \dots + \varepsilon_{nm} \cdot c_m &= A_{obs,m} \end{aligned} \quad (\text{B.32})$$

B.2.1. Solución de sistemas de ecuaciones lineales: el método de Gauss-Seidel

En muchas ocasiones nos encontramos con problemas en los que hay que resolver un sistema de ecuaciones lineales

$$\begin{aligned} a_{11}x_1 + a_{12}x_2 + \dots + a_{1m}x_m &= y_1 \\ a_{21}x_1 + a_{22}x_2 + \dots + a_{2m}x_m &= y_2 \\ \vdots & \\ a_{n1}x_1 + a_{n2}x_2 + \dots + a_{nm}x_m &= y_m \end{aligned} \quad (\text{B.33})$$

Utilizando notación matricial el sistema de ecuaciones puede escribirse como:

$$\mathbf{AX} = \mathbf{Y} \quad (\text{B.34})$$

donde \mathbf{A} es la matriz de coeficientes con n filas por m columnas, cuyos términos son los coeficientes a_{ij} con $i = 1, \dots, n$ y $j = 1, \dots, m$, \mathbf{X} es el vector de soluciones con los términos x_i con $i = 1, \dots, n$ e \mathbf{Y} es el vector de constantes con términos y_j con $j = 1, \dots, m$.

El sistema de ecuaciones se podría resolver por el método de Cramer, pero para sistemas de más de tres ecuaciones hay métodos más eficientes. A continuación se describe el método de Gauss-Seidel. Puede encontrar más información en los textos de Johnson [Johnson80] o Jurs [Jurs96].

Este método sirve para resolver un sistema de ecuaciones mediante un método iterativo. En este, se eligen unos valores iniciales de x_i, x_i^0 , que se proponen como solución del sistema de ecuaciones. A partir de estos valores se obtienen nuevos valores de las posibles soluciones $x_i^{(1)}$. El procedimiento se

repita hasta que no se observa una variación apreciable en los valores de las soluciones propuestas en iteraciones sucesivas, $x_i^{(m)} \simeq x_i^{(m-1)}$.

A continuación se describe el algoritmo de Gauss-Seidel. Considere un sistema con 3 incógnitas:

$$\begin{aligned} a_{11}x_1 + a_{12}x_2 + a_{13}x_3 &= y_1 \\ a_{21}x_1 + a_{22}x_2 + a_{23}x_3 &= y_2 \\ a_{31}x_1 + a_{32}x_2 + a_{33}x_3 &= y_3 \end{aligned} \quad (\text{B.35})$$

y ensaye la solución $x_1^{(0)} = x_2^{(0)} = x_3^{(0)} = 0$.

En la primera iteración, se obtiene la primera aproximación a la solución $x_1^{(1)}, x_2^{(1)}, x_3^{(1)}$ que viene dada por:

$$\begin{aligned} x_1^{(1)} &= \frac{y_1 - 0 - 0}{a_{11}} \\ x_2^{(1)} &= \frac{y_2 - 0 - 0}{a_{22}} \\ x_3^{(1)} &= \frac{y_3 - 0 - 0}{a_{33}} \end{aligned} \quad (\text{B.36})$$

En la segunda iteración, se obtiene una nueva aproximación a la solución que viene dada por:

$$\begin{aligned} x_1^{(2)} &= \frac{y_1 - a_{12}x_2^{(1)} - a_{13}x_3^{(1)}}{a_{11}} \\ x_2^{(2)} &= \frac{-a_{21}x_1^{(2)} + y_2 - a_{23}x_3^{(1)}}{a_{22}} \\ x_3^{(2)} &= \frac{-a_{31}x_1^{(2)} - a_{32}x_2^{(2)} + y_3}{a_{33}} \end{aligned} \quad (\text{B.37})$$

En la m-sima iteración se obtendrá

$$\begin{aligned} x_1^{(m)} &= \frac{y_1 - a_{12}x_2^{(m-1)} - a_{13}x_3^{(m-1)}}{a_{11}} \\ x_2^{(m)} &= \frac{-a_{21}x_1^{(m)} + y_2 - a_{23}x_3^{(m-1)}}{a_{22}} \\ x_3^{(m)} &= \frac{-a_{31}x_1^{(m)} - a_{32}x_2^{(m)} + y_3}{a_{33}} \end{aligned} \quad (\text{B.38})$$

La solución del sistema se obtiene tras q iteraciones cuando $x_1^{(q-1)} \approx x_1^{(q)}, x_2^{(q-1)} \approx x_2^{(q)}$ y $x_3^{(q-1)} \approx x_3^{(q)}$.

Estas expresiones se pueden generalizar para un sistema con n incógnitas. Tras m iteraciones,

$$x_k^{(m+1)} = \frac{-\sum_{h=1}^{k-1} a_{kh}x_h^{(m+1)} + y_k - \sum_{l=k+1}^n a_{kl}x_l^{(m)}}{a_{kk}} \quad (\text{B.39})$$

con $a_{kk} \neq 0$.

Para utilizar el algoritmo resuelva el siguiente ejemplo sencillo. Considere el sistema de ecuaciones,

$$\begin{aligned} 30x_1 + x_2 - x_3 &= 29 \\ x_1 + 20x_2 - x_3 &= 38 \\ x_1 + x_2 + 10x_3 &= 33 \end{aligned}$$

Si ensaya la solución $x_1^{(0)} = x_2^{(0)} = x_3^{(0)} = 0$, obtendrá

$$\begin{aligned} x_1^{(1)} &= \frac{29 - 1 \cdot 0 + 1 \cdot 0}{30} = 0,967 \\ x_2^{(1)} &= \frac{38 - 1 \cdot 0,967 + 1 \cdot 0}{20} = 1,85 \\ x_3^{(1)} &= \frac{33 - 1 \cdot 0,967 - 1 \cdot 1,85}{33} = 3,02 \end{aligned}$$

En la segunda iteración se obtiene,

$$\begin{aligned} x_1^{(2)} &= \frac{29 - 1 \cdot 1,85 + 1 \cdot 3,02}{30} = 1,01 \\ x_2^{(2)} &= \frac{38 - 1 \cdot 1,01 + 1 \cdot 3,02}{20} = 2,02 \\ x_3^{(2)} &= \frac{33 - 1 \cdot 1,01 - 1 \cdot 2,02}{33} = 3,00 \end{aligned}$$

En la tercera iteración,

$$\begin{aligned} x_1^{(3)} &= \frac{29 - 1 \cdot 2,02 + 1 \cdot 3,00}{30} = 0,999 \\ x_2^{(3)} &= \frac{38 - 1 \cdot 0,999 + 1 \cdot 3,00}{20} = 2,00 \\ x_3^{(3)} &= \frac{33 - 1 \cdot 0,99 - 1 \cdot 2,00}{33} = 3,00 \end{aligned}$$

y el método converge a la solución $x_1 = 1,00$, $x_2 = 2,00$, y $x_3 = 3,00$.

Ejemplo B.2 El método de Gauss-Seidel.

Copie el siguiente programa y compruebe su funcionamiento con el sistema de ecuaciones que ha resuelto manualmente.

Note que se utiliza como criterio de convergencia el valor de la variable tol que se calcula como

$$tol = \sum_{i=1}^n \frac{|x_i^{(iter)} - x_i^{(iter-1)}|}{x_i^{(iter)}}$$

donde $iter$ es un contador de iteraciones del algoritmo.


```

PROGRAM GaussSeidel
!
! Este programa resuelve un sistema lineal de N ecuaciones y N
! incógnitas con el método de Gauss–Seidel.
!
!           Y = A*X
!
! El programa se basa en el programa GAUSEI de
! K.F. Johnson, 'Numerical Methods in Chemistry', Marcel Dekker (1980)
!
! Ver también,
! P.C. Jurs, 'Computer Software Applications in Chemistry', 2nd Ed.,
! Wiley (1996)
!
! Nmax ... dimensiones máximas del sistema (PARAMETER)
! n     ... número de ecuaciones del sistema
! a     ... matriz de coeficientes
! y     ... vector de constantes
! x0    ... vector de soluciones de prueba
! x     ... vector de soluciones
! tol   ... tolerancia (criterio de convergencia)
!       tol = 1.0E-6
!
! Variables auxiliares
! iter  ... índice de las iteraciones del algoritmo realizadas
! i, j  ... índices de bucles DO
!
! IMPLICIT NONE
! INTEGER, PARAMETER :: NMax = 10
! INTEGER :: n
! REAL, DIMENSION(1:Nmax,1:Nmax) :: a
! REAL, DIMENSION(1:NMax) :: y, x0, x
! REAL :: tol
!
! INTEGER :: i, j, iter
!
! OPEN (UNIT=4, FILE='ecuacion.dat')
!
! READ(4,*) n
! IF ((n <= 0).or.(n > 10)) STOP
!
! Lee la matriz de coeficientes
!
! READ(4,*) ((a(i,j), j=1,n), i=1,n)
! WRITE(*,*) 'Matriz de coeficientes A'
! DO i=1,n,1
!   WRITE(*,*) (a(i,j), j=1,N)
! END DO
! WRITE(*,*)
!
! Lee el vector Y
!
! READ(4,*) (y(i), i=1,n,1)

```

```
WRITE(*,*) ' Vector de constantes Y'
WRITE(*,*)(Y(i),i=1,N,1)
```

! Lee el vector X, la estima de las soluciones

```
READ(4,*) (x(i),i=1,n,1)
WRITE(*,*) ' Vector de soluciones iniciales x'
WRITE(*,*)(x(i),i=1,n,1)
WRITE(*,*)
WRITE(*,*) ' Iteracion , Tolerancia, Soluciones '
```

! Comienza la solucion del sistema de ecuaciones.

! tol = SUM(ABS(x-X0)/x) < 1.0E-6)

```
tol = 0.0
DO iter =1, 1000, 1
  DO i=1, n, 1          ! Calculo de las soluciones
    x0(i) = x(i)
    x(i) = y(i)
    DO j= 1,n,1
      IF (j == i) CYCLE
      x(i) = x(i) - a(i,j)*x(j)
    END DO
    x(i) = x(i)/a(i,i)
    tol = tol + ABS(x0(i)-x(i))/x(i) ! Calculo parametro convergencia
  END DO
  write(*,FMT='(I2,2X,11(G11.5:))' iter , tol , (x(i) , i=1,n)
  IF (tol <= 1.0E-6) EXIT
  tol = 0.0
END DO
WRITE(*,*)
WRITE(*,*) ' Soluciones '
WRITE(*,*) ' Total de iteraciones ', iter
WRITE(*,*) ' Tolerancia ', tol
WRITE(*,*) ' Soluciones ', (x(i) , i=1,n)

STOP
END PROGRAM GaussSeidel
```

En el programa, el código correspondiente al cálculo de las soluciones se puede simplificar ligeramente utilizando la función intrínseca **SUM**,

```
tol = 0.0
DO iter =1, 1000, 1
  x0(1:n) = x(1:n)
  DO i=1, n, 1          ! Calculo de las soluciones
    x(i) = (-SUM(a(i,1:i-1)*x(1:i-1)) + y(i) - SUM(a(i,i+1:n)*x(i+1:n)))/a(i,i)
  END DO
  tol = SUM(ABS(x0(1:n)-x(1:n))/x(1:n)) ! Calculo parametro convergencia
  write(*,FMT='(I2,2X,11(G11.5:))' iter , tol , (x(i) , i=1,n)
  IF (tol <= 1.0E-6) EXIT
END DO
```

Utilizando este programa se obtienen los resultados,

```
C:\EPAQ\Valvanera>GaussSeidel

Matriz de coeficientes A
 30.0000      1.00000      -1.00000
  1.00000     20.0000     -1.00000
  1.00000      1.00000      10.0000

Vector de constantes Y
 29.0000      38.0000      33.0000
Vector de soluciones iniciales x
 0.00000      0.00000      0.00000

Iteracion, Tolerancia, Soluciones
1  3.0000      0.96667      1.8517      3.0182
2  0.11939     1.0056       2.0006     2.9994
3  0.61302E-020.99996      2.0000      3.0000
4  0.59982E-04 1.0000       2.0000      3.0000
5  0.15100E-05 1.0000       2.0000      3.0000
6  0.0000      1.0000       2.0000      3.0000

Soluciones
Total de iteraciones          6
Tolerancia      0.00000
Soluciones      1.00000      2.00000      3.00000

Program completed
Press Enter to Continue.
```

Modifique el programa *GaussSeidel* de modo que para la lectura y escritura de datos, y la aplicación del método de Gauss-Seidel, se utilicen subrutinas o funciones.

B.3. Descripción OM-CLOA de la molécula H_2^+

En este microproyecto, y con el fin de introducir la aplicación de métodos numéricos en el estudio teórico del enlace químico, se evaluarán numéricamente las integrales para obtener los valores de S, J y K del tratamiento OM-CLOA de H_2^+ .

(a) Evalúe numéricamente las integrales de solapamiento S, Coulomb J y de resonancia K, y la energía total del orbital molecular $E(R)$ en el intervalo de distancias internucleares $a_0 < R < 4a_0$. Compare los resultados obtenidos mediante la integración numérica con los obtenidos con las ecuaciones analíticas.

(b) Utilice los resultados de las integraciones numéricas para dibujar un gráfico de la energía total $E(R)$ y determine la energía molecular en la distancia internuclear de equilibrio y la energía de disociación espectroscópica (D_e).

Esta propuesta esta basada en el microproyecto 2.9 del texto de P. W. Atkins, *Physical Chemistry 6th ed.*, Oxford University Press, Oxford, 2000.

OM-CLOA de la molécula H_2^+

El orbital molecular combinación lineal de orbitales atómicos (OM-CLOA) construido a partir de las funciones de onda hidrogenoides 1s, centradas en los núcleos separados a una distancia R, y teniendo simetría g, no describe exactamente el estado fundamental del ión molécula de H_2^+ . Sin embargo, proporciona información acerca de las características funcionales de las funciones de onda, enlace, y los métodos numéricos necesarios para la resolución de problemas en Química Cuántica.

Considere el Hamiltoniano para un electrón en la molécula H_2^+ :

$$\hat{H} = -\frac{\hbar^2}{2m_e} \nabla_1^2 + V \quad (\text{B.40})$$

$$\hat{V} = -\frac{e^2}{4\pi\epsilon_0} \left(\frac{1}{r_{A1}} + \frac{1}{r_{B1}} - \frac{1}{R} \right) \quad (\text{B.41})$$

y el orbital molecular CLOA

$$\psi_{\pm} = N(A \pm B) \quad (\text{B.42})$$

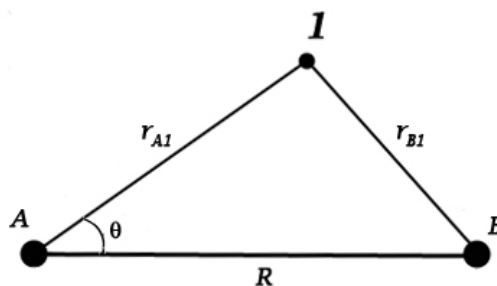


Figura B.1: Coordenadas de la molécula H_2^+ : R distancia internuclear, r_{A1} y r_{B1} distancias del electrón a los núcleos A y B, respectivamente. Note que r_{A1} y r_{B1} no son independientes:

$$r_{B1} = (r_{A1}^2 + R^2 - 2r_{A1} \cos \theta)^{\frac{1}{2}}.$$

donde A y B son los orbitales atómicos $1s$ centrados en los núcleos A y B , respectivamente,

$$A = \psi_{H1sA} = \frac{1}{(\pi a_0^3)^{\frac{1}{2}}} e^{-r_A/a_0} \quad (\text{B.43a})$$

$$B = \psi_{H1sB} = \frac{1}{(\pi a_0^3)^{\frac{1}{2}}} e^{-r_B/a_0} \quad (\text{B.43b})$$

y N es el factor de normalización

$$N = \frac{1}{(2(1+S))^{\frac{1}{2}}} \quad (\text{B.44})$$

siendo S la integral de solapamiento.

Para calcular la energía del orbital molecular enlazante, se puede calcular el valor esperado del hamiltoniano. En este caso, el cálculo es bastante simple ya que no incluye términos de repulsión interelectrónica como sucede con los sistemas polielectrónicos. El valor esperado del hamiltoniano de la ecuación (B.40) es

$$E_{\pm} = E_{H1s} + \frac{e^2}{4\pi\epsilon_0 R} - \frac{J \pm K}{1 \pm S} \quad (\text{B.45})$$

Las integrales que aparecen en esta expresión son

$$S = \int AB d\tau = \left\{ 1 + \frac{R}{a_0} + \frac{1}{3} \left(\frac{R}{a_0} \right)^2 \right\} \quad (\text{B.46a})$$

$$J = \frac{e^2}{4\pi\epsilon_0} \int \frac{A^2}{r_b} d\tau = \frac{e^2}{4\pi\epsilon_0 R} \left\{ 1 - \left(1 + \frac{R}{a_0} \right) e^{-2R/a_0} \right\} \quad (\text{B.46b})$$

$$K = \frac{e^2}{4\pi\epsilon_0} \int \frac{AB}{r_A} d\tau = \frac{e^2}{4\pi\epsilon_0 a_0} \left\{ 1 + \left(1 + \frac{R}{a_0} \right) e^{-R/a_0} \right\} \quad (\text{B.46c})$$

La integral S es la integral de solapamiento, la integral J se conoce como integral de Coulomb y es una medida de la interacción entre un núcleo y la densidad electrónica centrada en el otro núcleo, y la integral K se llama integral de resonancia, y es una medida de la interacción entre un núcleo y la densidad electrónica en exceso que se produce en la región internuclear debido al solapamiento orbital. Todas las integrales son positivas y decaen exponencialmente a cero a grandes distancias internucleares. Las integrales de las ecuaciones (B.46) se pueden evaluar analíticamente, para dar los resultados recogidos anteriormente.

B.3.1. Integración numérica: el método del trapecio

Un método numérico sencillo para calcular integrales definidas de funciones es el método del trapecio, cuyo algoritmo puede programarse fácilmente en Fortran 90.

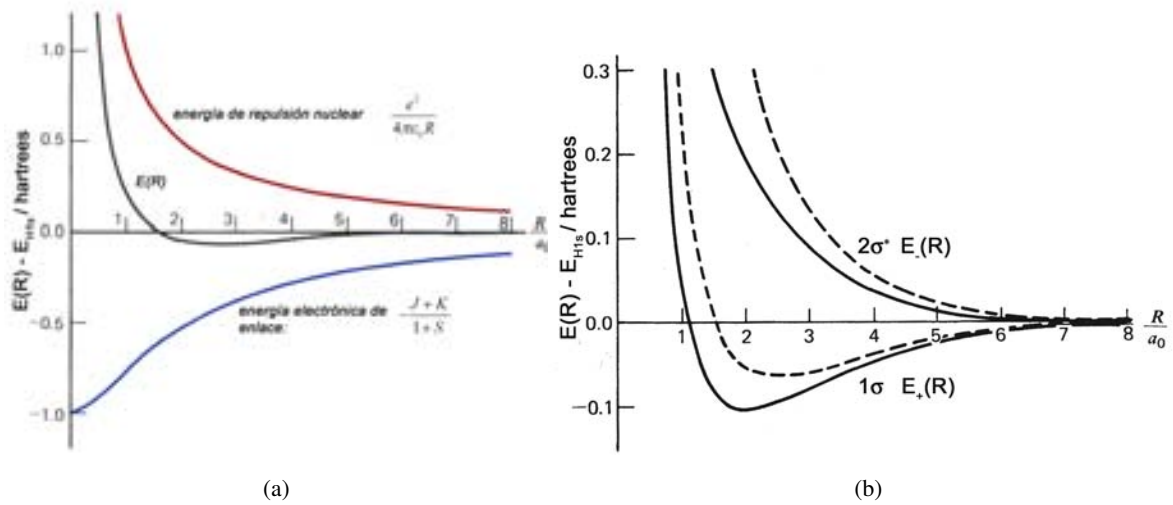


Figura B.2: (a) Energía de la molécula de H_2^+ en la aproximación LCAO en su estado fundamental ($E_+(R)$). (b) Comparación de las curvas de energía exactas (líneas continuas) y las obtenidas con el método LCAO (línea discontinua) para los dos primeros estados de la molécula de H_2^+ .

Suponga que quiere calcular la integral

$$I = \int_a^b F(x) dx \tag{B.47}$$

La regla del trapecio dice que el valor de la integral se puede aproximar con la fórmula

$$I \approx \frac{h}{2} [F(a) + F(b)] + h \sum_{i=1}^{n-1} F(a + ih) \tag{B.48}$$

donde

$$h = \frac{b - a}{n} \tag{B.49}$$

El valor de la integral aproximada (B.48) se acerca más al valor real de la integral cuanto menor es el valor de h .

Para mostrar cómo funciona el método, se escribirá un programa que calcule una integral de la que se conoce su valor. Por ejemplo,

$$I = \int_0^1 x \ln(1 + x) dx = \frac{1}{4}$$

Si inicialmente se toma el valor $h = b - a = 1,0$ y se utiliza la expresión (B.48)

$$I \approx \frac{1,0}{2} [1,0 * \ln(1 + 1,0) + 0,0 * \ln(1 + 0,0)] = 0,3466$$

En la siguiente iteración se divide h por la mitad, de modo que $n = (b - a)/h = (1,0 - 0,0)/0,5 = 2$.

$$\begin{aligned} I &\approx \frac{h}{2} [F(1,0) + F(0,0)] + h \sum_{i=1}^{n-1} F(0,0 + i * 0,5) \\ &= 0,25 [1,0 * \ln(2,0)] + 0,5 [0,5 * \ln(1,5)] \\ &= 0,2747 \end{aligned}$$

Si se divide de nuevo h por la mitad, de modo que $n = (b - a)/h = (1,0 - 0,0)/0,25 = 4$, el valor aproximado de la integral es:

$$\begin{aligned} I &\approx \frac{h}{2} [F(1,0) + F(0,0)] + h \sum_{i=1}^3 F(0,0 + i * 0,25) \\ &= 0,125 [1,0 * \ln(2,0)] + 0,25 [0,25 * \ln(1,25) + 0,5 * \ln(1,5) + 0,75 * \ln(1,75)] \\ &= 0,2562 \end{aligned}$$

Como puede observar cuando se reduce el valor de h , el valor aproximado se acerca más al valor real (que en este caso es conocido).

Una forma de decidir cuándo la integral aproximada ha convergido al valor real de la integral es utilizar el error relativo. En este caso el error relativo de la integral se define como,

$$\left| \frac{I(h/2) - I(h)}{I(h/2)} \right|$$

donde $I(h)$ es el valor de la integral aproximada obtenido con un paso de integración h e $I(h/2)$ es el valor de la integral aproximada obtenido con un paso de integración $h/2$.

Si tol es el valor mayor del error relativo que se admite en el cálculo de la integral, se considerará que el cálculo de la integral ha convergido cuando,

$$\left| \frac{I(h/2) - I(h)}{I(h/2)} \right| < tol$$

donde $tol < 1$.

Ejemplo B.3 *Escriba un programa que calcule la integral*

$$I = \int_0^1 x \ln(1 + x) dx$$

utilizando el método del trapecio. El programa debe leer los límites de integración a y b y el valor del error máximo relativo tol .

Descripción del algoritmo

El programa incluirá los siguientes pasos:

- i Leer los límite de integración a y b , y el error relativo máximo, tol
- ii Repetir el cálculo de la integral hasta que el error relativo $err_rel < tol$
- iii Escribir el resultado de la integración

Este algoritmo se puede describir con mayor detalle:

Leer a y b

Leer tol

$n \leftarrow 1; h \leftarrow (b - a)/n;$

$fa \leftarrow F(a); fb \leftarrow F(b); inth \leftarrow (fa + fb) * h/2,0$

Escribir "h =", h, "inth = ", inth

Para $i = 1$ hasta $i = 10$ con un incremento 1

$h \leftarrow h/2,0; n \leftarrow (b - a)/h$

$inth2 \leftarrow (fa + fb) * h/2,0$

Para $j = 1$ hasta $n - 1$ con un incremento 1

$inth2 \leftarrow inth2 + h * F(a + j * h)$

$err_rel \leftarrow ABS(inth - inth2)/inth2$

Escribir "h =", h, "int(h) = ", inth2

Si $err_rel \leq 0$ EXIT

$inth \leftarrow inth2$

Si $err_rel \leq 0$

Escribir ' El calculo ha convergido '

Escribir ' h = ', h, ' Integral = ', inth2

ELSE

Escribir ' El calculo no ha convergido '

Escribir ' Error relativo ', err_rel

Escribir ' Error relativo maximo ', tol

Escribir ' h = ', h, ' Integral = ', inth2

Traducción del algoritmo a un programa Fortran

```

PROGRAM trapecio
!
! Calcula la integral definida de x*ln(1+x) con los límites a y b
! mediante el metodo del trapecio
!
! a ... limite de integracion inferior
! b ... limite de integracion superior
! fa ... F(a) = a*log(1.0 + a)
! fb ... F(b) = b*log(1.0 + b)
! h ... paso de integracion
! inth ... Int(h) ! inth2 ... Int(h/2)
! errel ... error relativo
! tol ... error relativo maximo (<1.0)
!
IMPLICIT NONE
INTEGER :: i, j, n
REAL :: a, b, h
REAL :: inth, inth2, tol, errel
REAL :: fa, fb, x
!
WRITE(*,*) "Introduce el limite de integracion inferior a "
READ(*,*) a
WRITE(*,*) "Introduce el limite de integracion superior b "
READ(*,*) b
WRITE(*,*) "Introduce el valor del error relativo maximo (TOL<1.0) "
READ(*,*) tol
!
! Calcula la integral con el paso de integración h = b-a
!
fa = a*LOG(1.0 + a)
fb = b*LOG(1.0 + b)
h = b-a
inth = (fa+fb)*h/2.0
WRITE(*,*) "h =", h, " Area: ", inth
!
! Calcula la aproximación a la integral dividiendo h por 2 en cada iteración
!
DO i = 1,10,1
h = h/2.0
n = (b-a)/h
inth2 = (fa + fb)*h/2.0
DO j=1,n-1,1
x = a+j*h
inth2 = inth2 + h*x*LOG(1.0+x)
END DO
errel = ABS(inth2-inth)/inth2 ! calcula el error relativo
IF (errel <= tol) EXIT ! comprueba si la integral ha convergido
WRITE(*,*) "h =", h, " Area: ", inth2
inth = inth2
END DO

```

```

WRITE(*,*)
IF (errel <= tol) THEN
  WRITE(*,*) " El calculo ha convergido "
  WRITE(*,*) " h =", h," Area: ",inth2
ELSE
  WRITE(*,*) " El calculo no ha convergido "
  WRITE(*,*) " El error relativo es ", errel
  WRITE(*,*) " El error relativo maximo es ", tol
  WRITE(*,*) " hmin =", h," Area: ",inth2
END IF

STOP
END PROGRAM Trapecio

```

Con un editor de texto, edite el fichero trapecio.f90:

```
C:\EPAQ\Valvanera> EDIT trapecio.f90 
```

donde introducirá el anterior programa en Fortran 90.

Compile y ejecute el programa.

La ejecución del programa da como resultado en la pantalla:

```

C:\EPAQ\VALVANERA>trapecio

Introduce el limite de integracion inferior a 0.0

Introduce el limite de integracion superior b 1.0

Introduce el valor del error relativo maximo (TOL<1.0) 0.0001

h =      1.00000      Area:      0.346574
h =      0.500000     Area:      0.274653
h =      0.250000     Area:      0.256201
h =      0.125000     Area:      0.251553
h =      0.625000E-01 Area:      0.250388
h =      0.312500E-01 Area:      0.250097
h =      0.156250E-01 Area:      0.250024

El calculo ha convergido
h =      0.781250E-02 Area:      0.250006

Program completed
Press Enter to Continue.

```

Libros y recursos en Internet

C.1. Libros

Manuales de Fortran 90 en castellano

- [GMerayo03] F. GARCÍA MERAYO, . *El lenguaje de programación Fortran 90*. Paraninfo, Madrid, 2003.
- [GMerayo05] F. GARCÍA MERAYO, V. MARTÍN AYUSO, S. BOCETA MARTÍNEZ, y E. SALETE CASINO. *Problemas resueltos de programación en Fortran 95*. Thomson, Madrid, 2005.
- [Ventura00] S. VENTURA SOTO, J. L. CRUZ SOTO, y C. ROMERO MORALES. *Curso básico de Fortran 90*. Universidad de Córdoba, Córdoba, 2000.

Manuales de Fortran 90 en inglés

- [Brooks97] DAVID R. BROOKS, . *Problem Solving with Fortran 90 For Scientists and Engineers*. Springer Verlag, New York, 1997.
- [Smith95] IAN M. SMITH, . *Programming in Fortran 90. A First Course for Engineers and Scientist*. Wiley, Chichester, 1998.
- [Nyhoff97] LARRY NYHOFF, y SANFORD LEESTMA. *FORTTRAN 90 for Engineers and Scientists*. Prentice Hall, New York, 1997.
- [Metcalf04] MICHAEL METCALF, JOHN REID, y MALCOM COHEN. *Fortran 95/2003*. Oxford University Press, Oxford, 2004.
- [Chapman98] STEPHEN J. CHAPMAN, . *Fortran 90/95 for Scientist and Engineers*. WCB/Mc Graw-Hill, Boston, Massachusetts, 1998.
El libro tiene una página web <http://www.mhhe.com/engcs/general/chapman/> en la que están disponibles las soluciones a los ejercicios y problemas propuestos, material complementario y una página de recursos en Internet para usuarios de Fortran.

- [Brainerd96] WALTER S. BRAINERD, CHARLES H. GOLDBERG, y JEANNE C. ADAMS. *Programmer's Guide to Fortran 90. 3rd ed.* Springer Verlag, New York, 1997.
- [Mayo96] WILLIAM E. MAYO, y MARTIN CWIAKALA. *Programming with Fortran 90.* Schaum's Outline Series. Mc Graw-Hill, New York, 1996.

Química Computacional

- [Hinchliffe03] ALAN HINCHLIFFE, . *Molecular Modelling for Beginners.* Wiley, Chichester, 2003.
- [Rappe97] ANTHONY K. RAPPÉ, y CARLA J. CASEWIT. *Molecular Mechanics across Chemistry.* University Science Books, Sausalito, 1997.
- [Rogers03] DONALD W. ROGERS, . *Computational Chemistry using the PC.* Wiley, Hoboken, 2003. El libro incluye acceso a material complementario a través de la página web de [Wiley](#).
- [Grant95] GUY H. GRANT, y W. GRAHAM RICHARDS. *Computational Chemistry*, tomo 29. Oxford University Press, Oxford, 1995.
- [Gast93] J. GASTEIGER, y T. ENGEL (eds.) *Cheminformatics. A textbook.* Wiley-VCH, Weinheim, 2003.
- [Andres00] JUAN ANDRÉS BORT, y JUAN BERTRÁN RUSCA EDS. *Química Teórica y Computacional.* Universidad Jaime I, Castellón, 2000.
- [Paniagua97] JUAN CARLOS PANIAGUA VALLE, . *Introducció a la programació per a químics. Text-Guia.* Universitat de Barcelona, Barcelona, 1997.
- [Johnson80] KEITH F. JOHNSON, . *Numerical Methods in Chemistry.* Marcel Dekker, 1980.
- [Field05] MARTIN J. FIELD, . *A Practical Introduction to the Simulation of Molecular Systems.* Cambridge University Press, Cambridge, 2005. El libro incluye una librería de programas en Fortran 90 para la realización de distintos tipos de simulaciones http://www.psc.edu/biomed/dynamo/www_v2/Main.html.
- [Jurs96] PETER C. JURTS, . *Computer Software Applications in Chemistry, Second edition.* Wiley, New York, 1996.

Métodos numéricos y análisis de datos

- [Bev03] P. R. BEVINGTON, y D. K. ROBINSON. *Data Reduction and Error Analysis for the Physical Sciences, Third edition.* Mc Graw Hill, New York, 2003.
- [Brandt99] SIEGMUND BRANDT, . *Data analysis : statistical and computational methods for scientists and engineers, Third edition.* Springer-Verlag, New York, 1999. Incluye CD-ROM con los programas en Fortran descritos en el texto.
- [Press93b] WILLIAM H. PRESS, BRIAN P. FLANNERY, SAUL A. TEUKOLSKY, y WILLIAM T. VETTERLING. *Numerical Recipes in FORTRAN. The Art of Scientific Computing. Example Book.* Cambridge University Press, Cambridge, 1993.

- [Press93] —. *Numerical Recipes in FORTRAN. The Art of Scientific Computing. Second edition*, tomo 1 of Fortran Numerical Recipes in Fortran. Cambridge University Press, Cambridge, 1993. Dispone de una página web <http://www.nr.com/> donde se puede acceder a los programas de los libros de la serie *Numerical Recipes* de los autores.
- [Press96] —. *Numerical Recipes in Fortran 90. The Art of Parallel Scientific Computing*, tomo 2 of Fortran Numerical Recipes in Fortran. Cambridge University Press, Cambridge, 1996.

C.2. Recursos en Internet

En esta sección se recoge una breve¹ selección de enlaces a páginas web cuyos contenidos pueden ser de interés para los estudiantes o los instructores del curso.

Grupos de noticias y listas de distribución

comp.lang.fortran

Foro de noticias de usuarios de Fortran. Existe una lista de preguntas frecuentes(FAQ) que se envían al foro que puede consultarse en ***Fortran FAQ***.

comp-fortran-90

Lista de correo sobre programación en Fortran 90/95/2003. En la página se pueden consultar los archivos de mensajes enviados y puede solicitarse la inscripción en la lista de correo.

Documentos con información sobre compiladores, tutoriales, depósitos de programas en Fortran.

The Fortran Company.

Aunque es una página comercial, ofrece bastante material de uso libre. Un documento de esta página, habitualmente recomendado en los grupos de noticias es ***Fortran 90/95/HPF Information File*** de Michael Metcalf, una lista muy detallada y ordenada de compiladores, libros, cursos y otros materiales disponibles en la red.

F90 FAN's : Fortran 90 Frequently Asked about News.

Otra lista muy detallada y recomendada en los grupos de noticias mantenida por Michael Olanon.

Clive Page's list of Fortran Resources.

Otro documento con información para programadores en Fortran. Incluido entre las recomendaciones de Michael Olanon.

DISLIN.

Un ejemplo de librerías² gráficas en Fortran. Permiten crear interfaces gráficas, dibujar de gráficos y superficies sin necesidad de conocer otros lenguajes de programación.

European Standards For Writing and Documenting Exchangeable Fortran 90 Code

Recomendaciones para la escritura de programas en Fortran 90 elaboradas por distintas instituciones de meteorología europeas.

¹Una búsqueda en Google con la palabra *Fortran* en julio de 2006 proporcionó 32,8 millones de referencias

²Una librería es fundamentalmente una colección de funciones y subrutinas

Programas de Química

Gaussian 03, MOPAC 2002, Hyperchem, Spartan, ...

Venus, CHARMM, AMBER, Tinker, ...

Minnesota Computational Chemistry Group

Quantum Chemistry Program Exchange

Computational Chemistry List

The Chemistry section of The WWW Virtual Library

Linux4Chemistry

Chemistry and Biology applications on Linux

Índice alfabético

Arreglo

- bidimensional, 144
- declaración, 109
- extent, 109, 146
- funciones y subrutinas
 - arreglos automáticos, 198
 - arreglos de forma predefinida, 196
 - assumed shape arrays, 196, 213
 - automatic arrays, 198
 - automatic arrays, 213
 - tamaño fijo, 213
- lectura y escritura, 111–118, 147–151
- rank, 109, 146
- shape, 109, 146
- size, 109, 146
- subconjuntos de un arreglo, 113, 150
- subrutinas, 213–216
- unidimensional, 108

CALL, 206

CHARACTER, 31

CLOSE, 82, 83

Constantes y variables

- constantes con nombre, 28
- tipo entero, 13
- tipo lógico, 44
- tipo real, 14
- tipo texto, 30
- tipos, 12

CYCLE, 93

- bucles DO anidados, 97
- bucles DO iterativos, 94
- bucles DO WHILE, 94

DIMENSION, 111, 146

Directorio activo, 6

Directorio de trabajo, 6

DO

- bucles DO con nombre, 103
- CYCLE con nombre, 105
- EXIT con nombre, 104
- Bloques DO iterativos, 80
- bucles DO anidados, 89
- Bucles DO iterativos, 67
- Bucles DO WHILE, 62–67
- errores de sintaxis, 90, 91
- implicito, 112, 149

EDIT, 8

ELF90, 11

EXIT, 95

- bucles DO anidados, 97

Funciones intrínsecas, 34

- ABS, 34
- COS, 34
- de vectores, 127
- elementales, 119
- EXP, 34
- LOG, 34
- MATMUL, 168
- PRODUCT, 131, 175
- SIN, 34
- SQRT, 34
- SUM, 127, 171, 173
- TRANSPPOSE, 166

FUNCTION, 180

IF

- bloques IF con nombre, 103
- Bloques IF, 50–59
- ELSE, 52

-
- ELSE IF ... THEN, 53
 - IF ... THEN, 52
 - IF lógico, 49
 - IMPLICIT NONE, 16
 - INTEGER, 16
 - INTENT, 185, 188, 204, 207
 - INTERFACE, 187, 206

 - Lectura y escritura de datos, 35
 - escritura en ficheros, 80
 - escritura en pantalla, 37
 - formato libre, 35
 - lectura del ficheros, 80
 - lectura del teclado, 35
 - LOGICAL, 44

 - OPEN, 82, 83
 - Operaciones aritméticas
 - aritmética entera, 19
 - aritmética mixta, 24
 - aritmética real, 19
 - instrucciones de asignación, 17
 - jerarquía de operaciones, 20
 - operadores, 18
 - sentencias de asignación, 17
 - +, 18
 - , 18
 - *, 18
 - /, 18
 - ** , 18
 - Operaciones lógicas
 - expresiones de comparación, 45
 - expresiones de relación, 47
 - operadores lógicos de comparación, 45
 - operadores lógicos de relación, 47
 - sentencias de asignación, 45
 - ==, 45
 - <=, 45
 - <, 45
 - >=, 45
 - >, 45
 - .AND., 47
 - .OR., 47
 - .NOT., 47
 - .EQV., 47
 - .NEQV., 47

 - PARAMETER, 28
 - PATH, 6

 - PROGRAM, 10
 - READ, 35, 82
 - REAL, 16

 - STOP, 10
 - SUBROUTINE, 202

 - Variables subíndicas
 - listas, 108
 - matrices, 144
 - tablas, 144
 - vectores, 108

 - WRITE, 37, 82

Material didáctico
Química v. 1



UNIVERSIDAD
DE LA RIOJA