TEMA 1: ESTRUCTURAS Y ELEMENTOS DEL LENGUAJE FORTRAN.

CARACTERES DEL LENGUAJE FORTRAN.

-El lenguaje Fortran tiene unos nÂ£meros y signos que utiliza y que funcionan como caracteres o letras, siendo los caracteres permitidos por este lenguaje los siguientes:

-Letras de la A a la Z (Tanto mayÂ£sculas como minÂ£sculas).

-NÂ£meros del 0 al 9.

-Caracteres de puntuaciÂ¢n ., ;.

-Caracteres matemÂ ticos +, *, /, -.

-Caracteres especiales $, , =, <, >, (), :, Â´, y el blanco.

-El lenguaje Fortran no distingue en la sintaxis las letras mayÂ£sculas y minÂ£sculas salvo en el caso de los literales, y los signos de la comparaciÂ¢n no los suele utilizar.

-Las nuevas versiones de Fortran usan todos los cÂ¢digos relacionales pero con la excepciÂ¢n de que forman parte de los literales.

-SemÂ ntica, es lo que define el contenido de ciertas palabras y un conjunto de palabras reservadas cada una con un cometido especial. En este lenguaje hay un conjunto pequeÂ¤o de palabras reservadas.

-No existen palabras reservadas como tal, sino palabras clave, sabiendo que cualquier palabra puede ser un identificador vÂ lido.

-Con objeto de dar nombre a las cosas que se manipulan (MÂ¢dulo, ficheros estructuras, constantes, etc.) se establecen unas normas que son:

-El carÂ cter de una palabra debe ser un carÂ cter alfabâ tico, que aparte de las 52 letras considera como caracteres alfabâ ticos los sÂ¡mbolos $ y .

-El nÂ£mero mÂ ximo de caracteres que se pueden utilizar son 31,

aunque el compilador sÂ¢lo reconoce los seis primeros.

-El nombre de los objetos debe tener un significado sobre el

objeto que se estÂ tratando (Que tenga sentido), se debe hacer

una normalizaciÂ¢n de los nombres.

FORMATO DEL LENGUAJE FORTRAN.

-El lenguaje Fortran no estÂ formateado. Las cinco primeras columnas son

columnas reservadas para las etiquetas (Un nÂ£mero desde el uno hasta el

9) y referenciarÂ una lÂ¡nea.

-La etiqueta comprenderÂ un nÂ£mero del 1 al 99999 entero y sin signo,

que deberÂ ser Â£nica y ocuparÂ memoria.

-No tiene por quâ estar ordenada pero no se puede poner una etiqueta

mÂ s de una vez en un programa (Recomendable colocar la etiqueta para

los formatos).

-La sexta columna irÂ en blanco y sÂ¢lo se utiliza para indicar que la

lÂ¡nea que se estÂ tratando es una lÂ¡nea de continuaciÂ¢n (Cualquier

carÂ cter es vÂ lido).

-Como mÂ ximo se pueden continuar 19 lÂ¡neas de programa y hay que hacer

una indicaciÂ¢n.

-Las columnas 7 a 72 sirven para colocar el cÂ¢digo fuente. Todas las

sentencias se clasificarÂ n en dos grupos:

-Ejecutables (Especifican la acciÂ¢n y generan una instrucciÂ¢n).

-No ejecutables (InformaciÂ¢n y naturaleza de los datos).

-Las columnas 73 a 80 sirven para realizar todo tipo de comentarios,

entre los cuales pueden figurar:

-Nombre del programador.

-VersiÂ¢n del programa.

-Fecha del programa.

-DescripciÂ¢n del programa.

-E/S del programa.

-Variables y quâ representan.

-La primera columna puede llevar un conjunto de caracteres con un

significado especial que serÂ n la c, C, * y el sÂ¡mbolo $.

-Los comentarios no pueden ir entre lÂ¡neas de continuaciÂ¢n pero pueden

estar en cualquier parte del programa excepto la anteriormente citada.

TIPOS DE MODULOS.

-Para referenciar un programa principal en Fortran se utilizarÂ el

siguiente formato, teniendo en cuenta que dentro del programa principal

irÂ n las acciones a cumplir:

- PROGRAM Identificador

acciÂ¢n1

acciÂ¢n2

.....

acciÂ¢nn

END

-El especificador o identificador del programa puede ser omitido y no

es necesario colocarlo dentro de un programa.

-FUNCTION Identificador (ParÂ metros)

acciÂ¢n1

acciÂ¢n2

.....

acciÂ¢nn

END

-Se utilizarÂ la estructura de FunciÂ¢n cuando se necesite devolver o no un valor, teniendo en cuenta que la lista de los parÂ metros pasados deben ser separados por comas.

-SUBROUTINE Identificador (ParÂ metros)

acciÂ¢n1

acciÂ¢n2

.....

acciÂ¢nn

END

-La Subrutina o Procedimiento se utilizarÂ en aquellos casos en los que se necesite establecer una relaciÂ¢n de mÂ s de un tipo de dato. La lista de parÂ metros pasados deben ser separados por comas.

-BLOCK DATA Identificador

acciÂ¢n1

acciÂ¢n2

.....

acciÂ¢nn

END

-Esta estructura representa un conjunto de instrucciones meramente descriptivas que serÂ n datos comunes o aparte de los Subprogramas y se pueden inicializar.

-Estos cuatro elementos se pueden colocar a partir del programa principal o se pueden ejecutar y compilar separadamente.

-El lenguaje no necesita que una variable estâ declarada anteriormente sino que tienen una declaraciÂ¢n implÂ¡cita (Lo distingue por la primera

letra).

TIPOS DE DATOS.

-Como norma general el lenguaje no utiliza tipos de datos.

DATOS DE TIPO ENTERO.

-Son todas las variables que comiencen por una letra que estâ en el intervalo I-N, serÂ n variables tomadas como enteras si no se declara explÂ¡citamente de otro tipo o tambiâ n se pueden declarar como uno de los siguientes:

-INTEGER *1 (Asigna al entero un byte de longitud).

-INTEGER *2 (Asigna al entero dos bytes de longitud).

-INTEGER *4 (Asigna al entero cuatro bytes de longitud).

-Si despuâ s de la palabra reservada INTEGER no se pone nada la longitud por defecto de la variable declarada serÂ n 4 bytes.

-El separador usado suele ser la coma y para definir varias variables de tipo entero se pueden usar los siguientes formatos:

-INTEGER *1 Cont, Cont1, Cont2 (Declara 3 variables enteras).

-Signo 2..36 0..9 Â¢ A..Z .

- NÂ£mero (Indica base 16 y es perfectamente vÂ lido).

-El segundo formato define un nÂ£mero en otra base que no sea base decimal, siendo el sÂ¡mbolo el que indica la base en la que se va ha representar el nÂ£mero.

DATOS DE TIPO REAL.

-Es el nÂ£mero manipulado por excelencia en Fortran. Hay dos operaciones de tipo general que son la coma fija y la coma flotante.

-Los formatos que utilizan los nÂ£meros reales serÂ n:

-Signo parte entera. parte fraccionaria (C. fija).

-Signo parte entera E signo parte fraccionaria (C. flotante)

-Para declarar una variable de tipo real se usarÂ uno de los siguientes formatos:

-REAL *4 (Asigna al real cuatro bytes de longitud).

-REAL *8 (Asigna al real ocho bytes de longitud).

-El error que se comete al usar cuatro bytes de longitud se producirÂ en las siete u ocho cifras despuâ s de la coma.

-Cuando se utiliza un formato de cuatro bytes estamos ante un nÂ£mero de simple precisiÂ¢n.

-El real declarado como ocho bytes abarca despuâ s de la coma 14 o 15 cifras significativas para la parte entera del nÂ£mero que es aproximadamente el doble del formato de cuatro bytes.

-Cuando se utiliza un formato de ocho bytes estamos ante un nÂ£mero de doble precisiÂ¢n.

DATOS DE TIPO COMPLEJO.

-Este tipo de datos consta de una parte real y de una parte que es imaginaria y en su representaciÂ¢n se asumirÂ la parte real y se quedarÂ con la parte imaginaria.

-Los formatos que se utilizan para los nÂ£meros complejos serÂ n:

-Complejo (Real, Imaginario).

-Para declarar una variable de tipo complejo se usarÂ n uno de los siguientes formatos:

-COMPLEX *4 (Asigna al complejo cuatro bytes de longitud).

-COMPLEX *8 (Asigna al complejo ocho bytes de longitud).

-Despuâ s de la palabra reservada COMPLEX puede aparecer un signo, y la parte real e imaginaria irÂ n entre comas).

-Al igual que en otros tipos de datos cuatro bytes serÂ n simple precisiÂ¢n y ocho bytes serÂ n doble precisiÂ¢n.

DATOS DE TIPO CARACTER.

-Estas variables se representan por la palabra reservada CHARACTER y el formato que utiliza Fortran para definir las variables de este tipo son:

-CHARACTER *NÂ£mero .

-En este formato, nÂ£mero nos indica la longitud en caracteres de la variable, siendo el tamaÂ¤o mÂ ximo que puede tomar un campo carÂ cter desde 256 hasta 32767 caracteres.

-Este formato define el tratamiento de cadenas.

DATOS DE TIPO LOGICO.

-Las variables de tipo lÂ¢gico se representan mediante la palabra reservada LOGICAL con las variables separadas por comas y con un formato como el siguiente:

-LOGICAL *2 (Asigna al campo dos bytes de longitud).

-LOGICAL *4 (ASigna al campo cuatro bytes de longitud).

-Los valores que puede tomar una variable de tipo lÂ¢gico son los tÂ¡picos de True y False.

DATOS DE TIPO REGISTRO.

-Para este tipo de variables tenemos que definir previamente una estructura y sus correspondientes campos que se van ha tomar, siendo dicha estructura del tipo:

-STRUCTURE Identificador

CHARACTER *25 Campo1

INTEGER Campo2

REAL Campo3

LOGICAL Campo4

.....

END STRUCTURE

-La implementaciÂ¢n del registro que podrÂ¡a usarse con la estructura representada podrÂ¡a ser la siguiente:

-RECORD Identificador1 Identificador2 (NÂ£mero).

-Donde identificador1 representa el nombre de la estructura creada, identificador2 serÂ el nombre del registro y NÂ£mero serÂ el nÂ£mero de elemento del registro.

-Si por ejemplo deseamos referenciar a un real del registro nÂ£mero 25 tendremos que utilizar la siguiente forma:

-Identificador2 (25).Campo3.

CONSTANTES.

-Para definir una constante en Fortran deberemos utilizar el siguiente formato:

-PARAMETER (Identificador=constante, ...).

-La lista de constantes irÂ separada por comas. Las constantes podrÂ n ser de tipo numâ rico o de tipo Hollerith (Cadena de caracteres con cualquier carÂ cter imprimible). Tambiâ n podrÂ n ser expresiones.

-Las constantes numâ ricas podrÂ n tener los siguientes formatos:

-Sin punto decimal ni coma.

-Precedida por un signo, en los enteros.

-Coma fija con punto decimal, en los reales.

-Coma flotante con exponente entero, en los reales.

-Simple precisiÂ¢n (n.m, n., .m, n.mEe, n.E+e, nE+e).

-Doble precisiÂ¢n (n.mD+e, n.De, mDe, nDe).

-Compleja como un par ordenado entre parâ ntesis.

-LÂ¢gicas .TRUE. y .FALSE..

-De caracteres.

-La constante Hollerith podrÂ tener el siguiente formato:

-NÂ£mero H carÂ cter.

-Una constante simbÂ¢lica definida en PARAMETER puede aparecer como una

expresiÂ¢n o como un valor de DATA.

VARIABLES.

-Para definir una variable el compilador conoce ya de antemano si la

variable va a ser de tipo real o de tipo entera, siendo por defecto de

tipo real.

-La estructura que deberÂ n seguir las declaraciones de las variables en

un programa Fortran serÂ la siguiente:

-PROGRAM Identificador

PARAMETER (Lista de constantes)

*VARIABLES

INTEGER Lista de variables

REAL Lista de variables

COMPLEX Lista de variables

CHARACTER Lista de variables

LOGICAL Lista de variables

RECORD Tipos

*INICIO

.....

-Un atributo es una palabra reservada que acompaÂ¤a a la variable y sirve

para que el programa realice un mejor aprovechamiento de memoria, mejor dimensionamiento de arrays dinÂ micos, etc.

-Si la variable comienza con una letra que se encuentre en el intervalo I-N se tratarÂ de una variable de tipo entera. AnÂ logamente si comienza con una letra del intervalo A-H, O-Z se tratarÂ de una variable real.

-Como norma general, una variable debe ser declarada antes de que pueda utilizarse. Para inicializar una lista de variables a determinados valores se usarÂ el siguiente formato:

-DATA Lista de variables, lista de constantes.

-Esta sentencia debe preceder a cualquier sentencia ejecutable. Una constante seguida de un asterisco indicarÂ repeticiÂ¢n.

EXPRESIONES.

-En las expresiones aritmâ ticas sabemos que si se realiza una operaciÂ¢n de un entero con un real, siempre se codificarÂ el entero como real y luego se realiza la operaciÂ¢n.

-Se convierte el tipo que contiene al otro, realizÂ ndose siempre con la mÂ xima precisiÂ¢n. Los operandos que se van a usar en las expresiones aritmâ ticas serÂ n los siguientes:

-** (Operador potencia).

-/ (Operador divisiÂ¢n real).

-* (Operador producto).

-- (Operador resta).

-+ (Operador suma).

-() (Parâ ntesis).

-Para el uso de operadores relacionales, Fortran no admite como unos operadores relacionales los signos siguientes:

-<=, >=, <, >, <>.

-Los operadores relacionales que estÂ n permitidos en este lenguaje serÂ n los siguientes:

-.LT. (Operador menor que).

-.LE. (Operador menor o igual que).

-.NE. (Operador distinto de).

-.GT. (Operador mayor que).

-.GE. (Operador mayor o igual que).

-.EQ. (Operador igual que).

-Una expresiÂ¢n relacional compara los valores de dos expresiones de tipo aritmâ tico o de caracteres y el resultado es de tipo lÂ¢gico. Los formatos que puede tomar una sentencia relacional serÂ n:

-Variable operador variable.

-Variable operador constante.

-Las variables no han de ser necesariamente variables, tambiâ n pueden ser expresiones de todo tipo.

-En cuanto a los operadores lÂ¢gicos que estÂ n permitidos por Fortran se encuentran los siguientes:

-.NOT. (Operador negaciÂ¢n).

-.AND. (Operador producto lÂ¢gico).

-.OR. (Operador suma lÂ¢gica).

-.XOR. (Operador suma exclusiva lÂ¢gica).

-.EQV. (Operador de equivalencia lÂ¢gica).

-.NEQV. (Operador de no equivalencia lÂ¢gica).

-El resultado de la evaluaciÂ¢n de una expresiÂ¢n de tipo lÂ¢gico serÂ un valor de tipo lÂ¢gico, pudiendo ser el formato de la expresiÂ¢n:

-Operando1 operador operando2.

-El orden de prioridad establecido para la creaciÂ¢n de variables en

lenguaje Fortran serÂ por tanto el siguiente:

-Expresiones aritmâ ticas.

-Expresiones de caracteres.

-Expresiones relacionales.

-Expresiones lÂ¢gicas.

-Para la declaraciÂ¢n de variables se utiliza la palabra reservada

IMPLICIT, que toma una variable cualquiera a no ser que se indique lo

contrario. El formato que utiliza la instrucciÂ¢n es el siguiente:

-IMPLICIT REAL (A-Z).

-El formato anterior indica que todas las variables que empiecen desde

la a hasta la z sean tratadas como reales de forma implÂ¡cita. Una

variable suele ser tratada siempre como un real.

FINAL DE UN PROGRAMA.

-Hay dos formas de terminar la ejecuciÂ¢n de un programa, lÂ¢gicamente o

fÂ¡sicamente.

-Para terminar la ejecuciÂ¢n de un programa lÂ¢gicamente se utiliza el

siguiente formato:

-STOP NÂ£mero.

-NÂ£mero es la constante de caracteres o constante entera sin signo de

cinco dÂ¡gitos, o puede ser un mensaje de error encerrado entre dos

comillas.

-Para terminar la ejecuciÂ¢n de un programa fÂ¡sicamente se utiliza el

siguiente formato:

-END.

ENTRADA Y SALIDA DE DATOS.

-Para la introducciÂ¢n de datos se utiliza la palabra reservada READ con

el formato siguiente:

-READ (NÂ£mero, Etiqueta).

-Donde nÂ£mero es el nÂ£mero que indica la unidad de entrada, colocando el

valor * si no se quiere poner nada. Etiqueta indica la etiqueta que va

ha definir un formato, si no se quiere poner nada se usa el valor *.

-Para la lectura de elementos con READ las variables se separan por un

blanco o por comas, si se ha de introducir una frase o algo de tipo

carÂ cter, irÂ encerrado entre comillas.

-Si no hay suficientes datos para esta instrucciÂ¢n se produce un error

y el programa aborta.

-Para la salida de datos se utilizan dos mâ todos igualmente vÂ lidos, que

son dos palabras reservadas y que tienen el siguiente formato:

-PRINT *, Lista de variables (Separadas por comas).

-WRITE (*,*), Lista de variables (Separadas por comas).

-La lista de variables pueden ser tanto variables, como constantes como

expresiones de todo tipo. El cursor avanza a la siguiente lÂ¡nea. Este

tipo de sentencias se suelen incluir antes de una sentencia READ.

TEMA 2: FUNCIONES INTRINSECAS.

CONCEPTO DE FUNCION.

-Son aquellas funciones incorporadas al compilador, que son accesibles

desde un programa. Pueden ser llamadas en un programa dando el

identificador de la FunciÂ¢n seguido por sus argumentos entre

parâ ntesis.

-Estas funciones pueden tener uno o varios argumentos y se caracterizan

porque:

-El nombre y sus valores de entrada son uno mismo.

-Nunca puede ser usada en el lado izquierdo de la asignaciÂ¢n.

-El nombre determina el tipo de salida de la FunciÂ¢n.

-Los argumentos son del mismo tipo que la FunciÂ¢n.

-Los argumentos pueden ser expresiones de todo tipo.

-El formato de la FunciÂ¢n serÂ el siguiente:

-Nombre (Identificador1, identificador2, ..., identificadorn).

FUNCIONES ARITMETICAS.

-Son aquellas que realizan algÂ£n tipo de operaciÂ¢n matemÂ tica.

-Calcula el valor absoluto de un argumento real y devuelve un resultado

real y positivo:

-ABS (ExpresiÂ¢n numâ rica).

-Convierte un real a un entero truncÂ ndolo o eliminando la parte decimal

devuelve un entero:

-Var=INT (ExpresiÂ¢n numâ rica).

-INT (10**VarN*VarX)/10**VarN (N son los decimales a truncar).

-INT (10**VarN*VarX+0.5)/10**VarN (N son los decimales deseados).

-NINT (ExpresiÂ¢n numâ rica) (Realiza un redondeo).

-Convierte un nÂ£mero entero a un real:

-FLOAT (ExpresiÂ¢n numâ rica).

-Eleva un nÂ£mero real a la enâ sima potencia donde e es la base del

logaritmo natural, devuelve un real:

-EXP (NÂ£mero).

-Var=EXP (NÂ£mero).

-DEXP (NÂ£mero) (Donde nÂ£mero es de doble precisiÂ¢n).

-CEXP (NÂ£mero) (Donde nÂ£mero es un nÂ£mero complejo).

-Calcula el logaritmo natural de base e y es la inversa de la FunciÂ¢n anterior, devuelve un real:

-Var=LOG (ExpresiÂ¢n numâ rica).

-Calcula el logaritmo decimal o de base diez del argumento indicado, devuelve un real:

-Var=LOG10 (ExpresiÂ¢n numâ rica).

-Calcula la raÂ¡z cuadrada del argumento indicado, devuelve un real:

-Var=SQRT (ExpresiÂ¢n numâ rica).

-Devuelve el mayor valor de una serie de argumentos indicados:

-MAX0 (Lista de identificadores) (Devuelve un entero).

-AMAX1 (Lista de identificadores) (Devuelve un real).

-DMAX1 (Lista de identificadores) (Devuelve un doble precisiÂ¢n).

-Devuelve el menor valor de una serie de argumentos indicados:

-MIN0 (Lista de identificadores) (Devuelve un entero).

-AMIN1 (Lista de identificadores) (Devuelve un real).

-DMIN1 (Lista de identificadores) (Devuelve un doble precisiÂ¢n).

-Calcula el mÂ¢dulo o resto de dos nÂ£meros, devuelve un entero:

-MOD (NÂ£mero1, nÂ£mero2).

-Transfiere el signo del segundo identificador al primero:

-SIGN (ExpresiÂ¢n1, expresiÂ¢n2).

-Convierte un nÂ£mero de doble precisiÂ¢n a un nÂ£mero real:

-SNGL (NÂ£mero).

FUNCIONES TRIGONOMETRICAS.

-Son aquellas que realizan alguna operaciÂ¢n con las razones de tipo trigonomâ trico.

-Calcula el seno de una expresiÂ¢n dada, devuelve un real y debe estar

entre 1 y -1:

-Var=SIN (ExpresiÂ¢n numâ rica) (Devuelve un real).

-DSIN (ExpresiÂ¢n numâ rica) (Devuelve un doble precisiÂ¢n).

-CSIN (ExpresiÂ¢n numâ rica) (Devuelve un complejo).

-Calcula el coseno de una expresiÂ¢n dada, devuelve un real y debe estar

entre 1 y -1:

-Var=COS (ExpresiÂ¢n numâ rica) (Devuelve un real).

-DCOS (ExpresiÂ¢n numâ rica) (Devuelve un doble precisiÂ¢n).

-CCOS (ExpresiÂ¢n numâ rica) (Devuelve un complejo).

-Calcula la tangente de una expresiÂ¢n dada, devuelve un real:

-Var=TAN (ExpresiÂ¢n numâ rica) (Devuelve un real).

-DTAN (ExpresiÂ¢n numâ rica) (Devuelve un doble precisiÂ¢n).

-Calcula el arcoseno de una expresiÂ¢n dada, devuelve un real:

-Var=ASIN (ExpresiÂ¢n numâ rica).

-Calcula el arcocoseno de una expresiÂ¢n dada, devuelve un real:

-Var=ACOS (ExpresiÂ¢n numâ rica).

-Calcula el arcotangente de una expresiÂ¢n dada, devuelve un real:

-Var=ATAN (ExpresiÂ¢n numâ rica).

-Calcula el seno hiperbÂ¢lico de una expresiÂ¢n dada, devuelve un real:

-Var=SINH (ExpresiÂ¢n numâ rica).

-Calcula el coseno hiperbÂ¢lico de una expresiÂ¢n dada, devuelve un real:

-Var=COSH (ExpresiÂ¢n numâ rica).

-Calcula la tangente hiperbÂ¢lico de una expresiÂ¢n dada, devuelve un

real:

-Var=TANH (ExpresiÂ¢n numâ rica).

FUNCIONES DE NUMEROS COMPLEJOS.

-Realiza diferentes funciones con los nÂ£meros complejos.

-Toma la parte real de un argumento complejo:

-REAL (ExpresiÂ¢n compleja).

-Toma la parte imaginaria de un argumento complejo:

-AIMAG (ExpresiÂ¢n compleja).

-Calcula el conjugado de un argumento complejo:

-CONJ (ExpresiÂ¢n compleja).

-Representa un argumento complejo tomado de nÂ£meros reales:

-COMPLEX (NÂ£mero1, nÂ£mero2).

TEMA 3: ESTRUCTURAS DE CONTROL SELECTIVAS Y REPETITIVAS.

ESTRUCTURAS DE BIFURCACION INCONDICIONAL.

-Este tipo de estructuras son unas herramientas que sirven para confeccionar las estructuras selectivas. No son recomendables y sÂ¢lo se mantienen por compatibilidad con otras versiones.

-La siguiente estructura es un IF aritmâ tico, en el que se realiza la elecciÂ¢n de tres posibles etiquetas. Se evalÂ£a la condiciÂ¢n y en FunciÂ¢n de esa condiciÂ¢n se tomarÂ n los siguientes valores:

-CondiciÂ¢n <0 toma Etiqueta1.

-CondiciÂ¢n =0 toma Etiqueta2.

-CondiciÂ¢n >0 toma Etiqueta3.

-En este formato se pueden colocar dos etiquetas iguales, siendo el formato de la instrucciÂ¢n:

-IF (ExpresiÂ¢n aritmâ tica) Etiqueta1, Etiqueta2, Etiqueta3.

-La siguiente instrucciÂ¢n se usarÂ en el caso de que haya sentencias repetitivas que no estâ n implementadas en el compilador, siendo su

misiÂ¢n la de transferir el control a la sentencia especicada en la

etiqueta:

-GOTO Etiqueta.

-La siguiente instrucciÂ¢n transfiere el control a la etiqueta enâ sima

de la lista segÂ£n un valor entero. Es el GOTO calculado.

-En esta sentencia se evalÂ£a la expresiÂ¢n entera entre uno y el nÂ£mero

de etiquetas, y se bifurcarÂ a la etiqueta indicada para ejecutar un

conjunto de sentencias segÂ£n la expresiÂ¢n entera. Su formato es:

-GOTO (Etiqueta1, etiqueta2, ..., etiquetan) ExpresiÂ¢n entera.

-La instrucciÂ¢n GOTO asignado tiene como formato el siguiente:

-ASSIGN Constante entera TO Variable entera.

-GOTO Variable entera (Etiqueta1, etiqueta2, ..., etiquetan).

-Esta instrucciÂ¢n produce una estructura multibifurcaciÂ¢n. La variable

debe tener una de las etiquetas de GOTO y su funcionamiento es igual

que la anterior.

-Para llevar la variable a la etiqueta se usa la opciÂ¢n de ASSIGN. Si

la variable no estÂ asignada el GOTO no es vÂ lido.

ESTRUCTURAS ALTERNATIVAS.

-Dentro de las estructuras alternativas hay tres tipos.

-La estructura alternativa simple, realiza un conjunto de acciones si la

condiciÂ¢n es verdadera y sigue el flujo de control secuencial si la

condiciÂ¢n es falsa.

-En las estructuras alternativas todas las condiciones se colocan

siempre entre parâ ntesis. El formato de la alternativa simple es:

-IF (ExpresiÂ¢n lÂ¢gica) Sentencia.

-La expresiÂ¢n lÂ¢gica estÂ formada por una expresiÂ¢n con relacionales y

la sentencia a continuaciÂ¢n no pueden ser las palabras reservadas DO,

ELSEIF, ELSE, ENDIF, END o IF.

-La estructura alternativa doble, selecciona una de dos opciones. Si la

condiciÂ¢n es verdadera se ejecutan unas acciones y si es falsa otras

acciones. El formato de esta estructura serÂ¡a:

-IF (CondiciÂ¢n) THEN

acciÂ¢n1

acciÂ¢n2

.....

acciÂ¢nn

ENDIF

-Esta Â£ltima estructura se puede anidar con varias sentencias IF, y su

estructura quedarÂ¡a:

-IF (CondiciÂ¢n) THEN

acciÂ¢n1

acciÂ¢n2

.....

acciÂ¢nn

ELSE

acciÂ¢nn+1

acciÂ¢nn+2

.....

acciÂ¢nn+n

.....

ENDIF

-Si se quieren anidar varios niveles de sentencias IF-ELSE usaremos la

siguiente estructura:

-IF (CondiciÂ¢n) THEN

acciÂ¢n1

acciÂ¢n2

.....

acciÂ¢nn

ELSEIF (CondiciÂ¢n2) THEN

acciÂ¢nn+1

acciÂ¢nn+2

.....

acciÂ¢nn+n

.....

ELSE

acciÂ¢nm+1

acciÂ¢nm+2

.....

acciÂ¢nm+n

ENDIF

-La estructura alternativa mÂ£ltiple se utiliza cuando la condiciÂ¢n tiene

mÂ s de dos valores con selecciÂ¢n mÂ£ltiple. El valor de su expresiÂ¢n a

de ser un entero, carÂ cter o lÂ¢gico.

-Si el carÂ cter utilizado es numâ rico no irÂ entre comillas, mientras

que si es de tipo carÂ cter irÂ entre comillas. Esta Â£ltima medida exige

que el cÂ¢digo ASCII del primer carÂ cter sea menor que el segundo.

-El formato de la estructura serÂ el siguiente:

-SELECT CASE (ExpresiÂ¢n)

CASE (Caso1)

acciÂ¢n1

acciÂ¢n2

.....

acciÂ¢nn

CASE (Caso2)

acciÂ¢nn+1

acciÂ¢nn+2

.....

acciÂ¢nn+n

.....

CASE DEFAULT

acciÂ¢nm+1

acciÂ¢nm+2

.....

acciÂ¢nm+n

END SELECT

CONCEPTO DE BUCLE.

-Consiste en una estructura de control que gobierna los procesos de tipo repetitivo dentro de un programa. Repite una secuencia de acciones mientras o hasta una condiciÂ¢n sea verdadera o falsa.

-IteraciÂ¢n de un bucle es la repeticiÂ¢n de sentencias interiores que hay dentro de un bucle.

ESTRUCTURA REPETITIVA. SENTENCIA "DO".

-Esta estructura contiene dos formatos. El primer formato se utiliza cuando se conoce el nÂ£mero de iteraciones y su formato es:

-DO Etiqueta VarA=Inicio, Final, Incremento .

-Las variables se separan por comas y el incremento puede ser positivo o negativo y es opcional. Normalmente inicio debe ser menor que la variable final.

-El otro formato es lo mismo que una estructura de tipo repetir. Se usa cuando se desea repetir una condiciÂ¢n un nÂ£mero de veces siendo una por defecto. Su formato es:

-DO Etiqueta VarA=Inicio, Final, Incremento

acciÂ¢n1

acciÂ¢n2

.....

acciÂ¢nn

Etiqueta CONTINUE

ENDDO

-Las variables pueden ser enteras, constantes o expresiones, y es mÂ s rÂ pida y potente con variables y constantes con lo que se evitan los errores de redondeo.

-Como mÂ¡nimo se ejecuta una vez. Ejecuta un conjunto de instrucciones hasta que se encuentra a la etiqueta, y si no se usa la etiqueta se deberÂ cerrar el bucle con ENDDO.

-Por defecto el incremento del bucle es uno. Si se usa la etiqueta habrÂ que tener en cuenta una serie de restricciones:

-No se puede usar un GOTO incondicional o asignado.

-No se puede usar ELSE, IF, SELECT CASE, ENDIF.

-No puede seguirle un ENDSELECT, EXIT, RETURN o STOP.

-Para evitar estas restricciones bastarÂ con colocar una sentencia en

blanco. Si se quiere usar una variable Â¡ndice despuâ s de salir la

variable Â¡ndice la variable tendrÂ el valor final mÂ s el incremento.

ESTRUCTURA REPETITIVA. SENTENCIA "WHILE".

-Utiliza una condiciÂ¢n que puede ser una expresiÂ¢n lÂ¢gica o relacional.

Esta condiciÂ¢n se evalÂ£a antes y despuâ s de cada ejecuciÂ¢n del bucle.

Si la condiciÂ¢n es falsa no se ejecuta nunca.

-Los bucles deben terminar siempre y pueden ser controlados por:

-Contador, necesita una variable dentro del bucle que se debe

inicializar, comprobar e incrementar.

-Centinela, es un valor especial para el final de una lista y es

un valor que jamÂ s se procesa, pero debe ser del mismo tipo que

los datos.

-Interruptor, es una variable lÂ¢gica.

-El formato de la instrucciÂ¢n serÂ la siguiente:

-DO Etiqueta WHILE (ExpresiÂ¢n lÂ¢gica)

acciÂ¢n1

acciÂ¢n2

.....

acciÂ¢nn

Etiqueta CONTINUE

ENDDO

-Todas las restricciones anteriormente indicadas para la otra estructura

repetitiva sirven para DO WHILE.

-Si por cualquier circunstancia el compilador no reconoce este tipo de

estructuras se podrÂ implementar de la siguiente:

-Etiqueta IF (CondiciÂ¢n) THEN

acciÂ¢n1

acciÂ¢n2

.....

acciÂ¢nn

GOTO Etiqueta

ENDIF

-Hay que tener en cuenta que GOTO sÂ¢lo se usarÂ en el caso de este tipo de circunstancias y sÂ¢lamente para ellas.

-Cuando se usen etiquetas, una sÂ¢la etiqueta valdrÂ para cerrar varios bucles que estâ n anidados. Pero si se usan ENDDO hay que colocar tantos como DO hayan.

TEMA 4: ENTRADAS Y SALIDAS CON FORMATOS.

INSTRUCCION CON FORMATO DE ENTRADA. SENTENCIA "READ".

-Este tipo de instrucciones no es muy potente porque sÂ¢lo estÂ n pensadas para introducir y sacar datos. La unidad estÂ ndar en el teclado estÂ denominado como un asterisco (*).

-La lectura mediante la instrucciÂ¢n READ se realiza con la siguiente sintaxis:

-READ (Lista de descriptores), Lista de variables.

-La Lista de variables siempre irÂ separada por comas.

-Con esta sintaxis la lectura se realiza del dispositivo predefinido que puede ser uno de los siguientes:

-Constante entera (Que referencia otra sentencia).

-Variable carÂ cter.

-Array de caracteres.

-ExpresiÂ¢n carÂ cter (Entre parâ ntesis y apÂ¢strofes).

-Variable entera (Con una etiqueta FORMAT).

-Dentro de la lista de descriptores podemos encontrarnos las siguientes opciones:

-Unidad, que sirve para que a cada dispositivo se le asigne una unidad o un entero (Sin signo) y la palabra UNIT es opcional. Si no se coloca UNIT debe ir en primer lugar, si se pone la palabra reservada UNIT podrÂ ir en otro lugar. Su sintaxis es:

- UNIT= Entero.

-Formato, en el que podrÂ n colocarse los siguientes parÂ metros:

-Etiqueta, asignada a la sentencia especial que serÂ FORMAT.

-Variable, entera a la que previamente se le halla asignado una Etiqueta vÂ lida.

-ExpresiÂ¢n, de tipo cadena que contenga los cÂ¢digos de los formateos entre parâ ntesis que estarÂ n cerrados entre unas comillas.

-Variable o array, de caracteres a los cuales se le hallan asignado los cÂ¢digos de formato.

-Asterisco (*), que indicarÂ una entrada sin formato.

sabiendo que el formato estarÂ definido por la sintaxis:

- FMT= Formato.

-NÂ£mero de registro, que sÂ¢lo se usarÂ en el tratamiento de los archivos de acceso directo, donde se le indica que lea el nÂ£mero relativo al registro siendo su estructura o sintaxis:

-REC=NÂ£mero registro.

-CÂ¢digo chequeador, de la operaciÂ¢n al cual deposita el resultado de analizar la operaciÂ¢n siendo su sintaxis:

-IOSTAT=Variable entera.

-Control del fin de archivo, que se especifica con la sintaxis:

-END=Etiqueta.

-Transferencia de control, que se produce cuando hay un error y

que contiene la siguiente sintaxis:

-ERR=Etiqueta.

-Si se admiten UNIT y FMT, sus correspondientes valores deberÂ n estar

en el primer y segundo lugar de la instrucciÂ¢n READ respectivamente.

-Los dos primeros formatos se han de usar obligatoriamente, mientras que

los otros parÂ metros sÂ¢lo se usarÂ n para ficheros externos. Un ejemplo

de uso de esta sentencia serÂ¡a:

-READ (UNIT=3, FMT=20, REC=10, IOSTAT=Cod, END=100, ERR=Error).

-Si la lectura es desde teclado el fin de fichero debe ser tecleado por

el usuario con la combinaciÂ¢n CTRL+Z.

INSTRUCCION CON FORMATO DE SALIDA. SENTENCIA "WRITE".

-Para este tipo de instrucciones nos encontramos dos formatos que estÂ n

diferenciados:

-PRINT (Lista de descriptores), Lista de salida.

-WRITE (Lista de descriptores), Lista de salida.

-El primer formato siempre se utilizarÂ cuando el dispositivo de salida

estâ predefinido por el sistema, mientras que el segundo formato se

usarÂ para una salida a una unidad especÂ¡fica.

-La lista de descriptores que se podrÂ usar serÂ n los mismos que en la

instrucciÂ¢n READ. UNIT y FMT podrÂ n no figurar si se pone primero el

nÂ£mero de la unidad y luego el formato.

INSTRUCCION DE FORMATOS. SENTENCIA "FORMAT".

-Esta instrucciÂ¢n facilita la informaciÂ¢n necesaria para que se haga un reconocimiento de la representaciÂ¢n que los datos van a tomar en la memoria principal. La sintaxis de FORMAT serÂ la siguiente:

-Etiqueta FORMAT (Lista de cÂ¢digo de formato).

-Etiqueta serÂ el nÂ£mero de etiqueta (Obligatorio), aunque se podrÂ referenciar tambiâ n con una variable entera a la cual se ha asignado una etiqueta con ASSIGN.

-La lista de cÂ¢digos puede estar vacÂ¡a si la lista de sentencias de E/S estÂ vacÂ¡a con un salto de lÂ¡nea.

-Esta sentencia no es ejecutable, puede escribirse en cualquier parte y conviene agruparlas todas al principio o al final del programa.

-La Lista de cÂ¢digo de formato podrÂ contener uno o varios de los siguientes tipos:

-CÂ¢digos numâ ricos o de datos (Repetibles) entre los cuales tenemos:

-CÂ¢digo I.

-CÂ¢digo F.

-CÂ¢digo E.

-CÂ¢digo D.

-CÂ¢digo G.

-CÂ¢digo P.

-CÂ¢digo L.

-CÂ¢digo A.

-CÂ¢digos de posicionamiento (No repetibles) entre los cuales tenemos:

-CÂ¢digo X.

-CÂ¢digos T, Tl, Tr.

-CÂ¢digo /.

-CÂ¢digos especiales (No repetibles) entre los cuales tenemos:

-CÂ¢digos S, Sp, Ss.

-CÂ¢digos Bn, Bz.

-CÂ¢digo H.

-CÂ¢digo :.

-CÂ¢digo .

-CÂ¢digos de control de carro.

CODIGO NUMERICO "I".

-En los cÂ¢digos de datos o numâ ricos siempre hay que diferenciar tres

partes que son fundamentales:

-Tipo de dato a representar.

-TamaÂ¤o para la representaciÂ¢n.

-Puntos y tamaÂ¤os de la parte fraccionaria.

-Entre los cÂ¢digos numâ ricos existen una serie de caracterÂ¡sticas:

-Los blancos de relleno son ignorados.

-Si el signo es negativo se genera el signo -.

-Tiene prioridad el punto decimal de entrada ante la posiciÂ¢n

decimal especificada.

-Los campos se ajustan a la derecha.

-Si se produce error de salida salen asteriscos.

-Para la entrada de este cÂ¢digo los datos deben ser de tipo entero,

siendo I el identificador de cÂ¢digo y w el ancho del campo y la

sintaxis:

-Iw.

-Transfiere datos enteros desde el soporte externo a variables, tantos caracteres como valor tenga w contando dÂ¡gitos, blancos y signos y asocia variables de izquierda a derecha con las normas siguientes:

-Blancos entre dÂ¡gitos o al final se interpretan como ceros.

-No usan cÂ¢digo de formato cuando la lectura es por teclado.

-Con Bn los blancos se ignoran y con Bz se toman como ceros.

-Para la salida los datos van desde la memoria interna al soporte externo con un valor w de ancho ajustÂ ndose a la derecha y si es negativo se coloca un signo delante del dÂ¡gito mÂ s significativo.

-Si el nÂ£mero de caracteres es mayor que el ancho de salida el campo de salida se llena de asteriscos.

-Si se utiliza un coeficiente de repeticiÂ¢n n deberÂ ser una constante entera sin signo mayor que cero con la sintaxis:

-nIw.

-Si se quiere indicar el nÂ£mero mÂ¡nimo de dÂ¡gitos que aparecerÂ n en la salida se usarÂ m, una constante entera sin signo con la sintaxis:

-nIw.m.

-Se debe contemplar la opciÂ¢n de que el nÂ£mero sea negativo con lo que se deberÂ aumentar el formato en una unidad.

-Si en la salida se colocan menos dÂ¡gitos que en el formato indicado se suprimirÂ n todos los ceros que no sean significativos.

CODIGO NUMERICO "F".

-Este cÂ¢digo transfiere datos de tipo real desde un dispositivo externo hasta la memoria o a la inversa, siendo su sintaxis la siguiente:

-Fw.d.

-En este cÂ¢digo, F serÂ el carÂ cter del cÂ¢digo, w serÂ la longitud total

del campo incluyendo los blancos, signos, puntos y dÂ¡gitos, y d serÂ

el nÂ£mero de decimales de la parte real.

-Si no se introduce un punto decimal se sitÂ£an los decimales de forma

automÂ tica. El cÂ¢digo F permite ser un cÂ¢digo repetible con lo que su

sintaxis serÂ la siguiente:

-nFw.d.

-Se pueden introducir nÂ£meros reales en notaciÂ¢n exponencial, siendo el

exponente que sigue al nÂ£mero real de dos formas:

-Constante entera con signo + o -.

-CarÂ cter E o D seguido de un signo y una constante entera.

-Los caracteres de las variables reales se ajustan a la derecha del

campo de anchura. El punto decimal se genera con d posiciones a la

derecha.

-En la salida no se producen ceros a la izquierda a no ser que la

mantisa sea menor que uno, con lo que habrÂ un cero a la izquierda del

punto decimal.

-Si la parte fraccionaria tiene mÂ s anchura que d habrÂ un redondeo del

valor antes de ajustarse y si el dato no cabe en w se producirÂ una

salida de w asteriscos.

-Entre los ejemplos que se pueden citar para este formato estarÂ n:

CODIGO NUMERICO "E".

-Este formato lee o escribe datos reales en simple o doble precisiÂ¢n en

notaciÂ¢n exponencial. E es el cÂ¢digo del exponente, w el ancho del

campo y d los dÂ¡gitos decimales con una sintaxis del tipo:

-Ew.d.

-La opciÂ¢n w contarÂ la mantisa, el dÂ¡gito que precede al punto decimal

el punto decimal de la mantisa y el exponente que tendrÂ tres dÂ¡gitos

de forma que la representaciÂ¢n serÂ del tipo:

-(-)0.E(-)nn.

-Si el punto no figura en la entrada, la parte decimal serÂ n los d

dÂ¡gitos mÂ s a la derecha de la mantisa. Si aparece el punto decimal

en la entrada no se tendrÂ en cuenta la especificaciÂ¢n d en el formato.

-El signo del exponente se omite si es positivo y E o D pueden omitirse

si el exponente tiene signo. Es recomendable especificar el punto

decimal en la entrada.

-Para la salida han de reservarse una posiciÂ¢n para el punto decimal y

el dÂ¡gito cero que precede al punto decimal si hay posiciÂ¢n para â l. Si

la mantisa es negativa otra posiciÂ¢n y el exponente cuatro posiciones.

-Se ejecuta la anchura del campo a la derecha y si w es menor que las

posiciones para la salida se muestran asteriscos.

-En FunciÂ¢n del desplazamiento del punto decimal se calcula el exponente

y la mantisa se puede redondear, pudiâ ndose omitir el cero antecedente

al punto decimal si no hay posiciÂ¢n.

-El cÂ¢digo E puede ser repetible con la siguiente sintaxis:

-nEw.d.

-Otro formato en el que una constante entera e nos indica el nÂ£mero de

dÂ¡gitos del exponente serÂ¡a:

-Ew.dEe.

-Entre los ejemplos que se pueden citar para este formato estarÂ n:

CODIGO NUMERICO "D".

-Describe nÂ£meros reales en simple o doble precisiÂ¢n en forma

exponencial con el mismo efecto que el cÂ¢digo E siendo su sintaxis:

-Dw.d.

-Para la entrada de datos este cÂ¢digo es tratado de igual manera que el cÂ¢digo E. Normalmente se utiliza este cÂ¢digo para enfatizar que el nÂ£mero que se trata es de doble precisiÂ¢n.

-El exponente para la salida cambia la letra E por la letra D y tiene la siguiente sintaxis:

-D(+/-)nn.

-Este cÂ¢digo numâ rico es repetible siendo su sintaxis:

-nDw.d.

CODIGO NUMERICO "G".

-Se utiliza este cÂ¢digo para entrada y salida de datos de simple y doble precisiÂ¢n siendo sus sintaxis mÂ s generales las siguientes:

-Gw.d.

-Gw.dEe.

-La entrada de datos reales tiene el mismo significado que los cÂ¢digos F, E y D.

-En la salida actÂ£a como formato F cuando el valor de salida estÂ en el rango entre 0.1 y 10**d, mientras que actÂ£a como formato E cuando el nÂ£mero es mÂ s pequeÂ¤o de 0.1 o mayor que 10**d.

-Si actÂ£a como formato F entonces los Â£ltimos cuatro caracteres serÂ n blancos y el valor se imprimirÂ en un ancho de campo de w-4 caracteres.

-Si se usan formatos a la entrada y se coloca un punto decimal cuando el formato es distinto, habrÂ disyunciÂ¢n entre el formato del teclado y el establecido en el programa.

-Siempre se da prioridad al formato introducido por teclado.

-Entre los ejemplos que se pueden citar para este formato estarÂ n:

CODIGO DE DATOS "P".

-Es un factor de escala y sirve para ver los valores exponenciales con

un entero delante del punto decimal, aplicable sÂ¢lamente a los cÂ¢digos

F, E, D y G, afectando a la entrada de datos y a su salida.

-El valor n serÂ un entero que puede llevar signo y su sintaxis serÂ :

-nP.

-Al aplicar el cÂ¢digo P en la entrada sÂ¢lo es aplicable si el valor

externo carece de exponente con lo que se expresarÂ mediante la

relaciÂ¢n:

-Valor interno=(Valor externo)/10n.

-El factor de escala afecta a los cÂ¢digos de formato que aparecen a

continuaciÂ¢n hasta encontrar otro factor de escala.

-Cuando se usa para el cÂ¢digo F en la salida se produce desplazamiento

del punto en el valor verdadero siendo su relaciÂ¢n:

-Valor externo=Valor interno*10n.

-Si se usa en concexiÂ¢n con D o E la salida no cambia pero se desplaza

el punto de la mantisa al hacer el producto por 10**n, y el exponente

se decrementa en n, al igual que con la forma mantisa-exponente.

-Con el formato G no se usa el factor de escala porque da problemas.

-Si tenemos las siguientes instrucciones:

-100 FORMAT (2PE7.2, F6.3, -1P8.4)

READ (5, 100) VarA, VarB, VarC

WRITE (*, '(1X, 3F12.5)') VarA, VARB, VarC

y le introducimos la siguiente entrada:

-b17.E0157.132-453261.7.

tendremos la siguiente tabla de referencia:

CODIGO DE DATOS "L".

-Este cÂ¢digo se usa para la ediciÂ¢n de datos lÂ¢gicos siendo w el ancho del campo y siendo su sintaxis la siguiente:

-Lw.

-Para la entrada busca en el campo de forma que si el primer carÂ cter es T (True) o F (False) los demÂ s caracteres del campo son ignorados.

-Para la salida se produce una conversiÂ¢n de TRUE a FALSE o T a F ajustÂ ndose a la derecha del campo.

-El cÂ¢digo L es un cÂ¢digo de tipo repetible siendo su sintaxis:

-nLw.

CODIGO DE DATOS "A".

-Este cÂ¢digo transfiere datos de tipo carÂ cter de la memoria al soporte externo y a la inversa, permitiendo el manejo, entrada y salida de cualquier variable de tipo carÂ cter, siendo su sintaxis:

-Aw.

-TambiÂ n tiene formato repetible siendo su sintaxis:

-nAw.

-En las entradas se debe colocar apÂ¢strofes que delimiten la cadena si es con lista directa, pero con el cÂ¢digo A puede evitarse y tomarÂ la longitud de la variable asociada teniendo en cuenta:

-Si w=n todo el dato es asignado a la variable carÂ cter.

-Si w>n los Â£ltimos n caracteres se almacenan en la variable.

-Si w<n los caracteres se ajustan a la izquierda rellenÂ ndose los n-w caracteres de la derecha a blancos.

-En las salidas existen otras consideraciones:

-Si w=n tendrÂ de longitud n.

-Si w>n los n caracteres se ajustan a la derecha y los w-n

primeros caracteres se rellenan a blancos.

-Si w<n los w primeros caracteres de la cadena salen a la salida

perdiâ ndose los restantes.

CODIGO DE POSICIONAMIENTO "X".

-Los cÂ¢digos de posicionamiento determinan la posiciÂ¢n dentro de una

lÂ¡nea. La posiciÂ¢n puede ser relativa a la actual del cursor o una

posiciÂ¢n absoluta.

-El cÂ¢digo X salta n caracteres tanto para la salida como para la

entrada en el medio externo siendo su sintaxis:

-nX.

-La opciÂ¢n n indica los caracteres a saltar a partir de la posiciÂ¢n

actual del cursor.

-Para la entrada salta n posiciones hacia adelante desde la posiciÂ¢n

del cursor. Para la salida en la lÂ¡nea o registro de salida genera n

blancos y mueve el cursor n posiciones a la derecha.

-Este cÂ¢digo no es repetible.

CODIGOS DE POSICIONAMIENTO "T", "Tl" Y "Tr".

-Estos cÂ¢digos producen una tabulaciÂ¢n en el registro y T especifica

una posiciÂ¢n absoluta dentro del registro de entrada o salida siendo

su sintaxis:

-Tc.

-La opciÂ¢n c indica la columna dentro del registro desde donde se va a

posicionar el cursor y donde comienzan las transferencias de datos.

-El tabulador Tl mueve la posiciÂ¢n del cursor hacia la izquierda y Tr

mueve la posiciÂ¢n del cursor hacia la derecha con las siguientes

sintaxis:

-Tls.

-Trs.

-La opciÂ¢n s indica el nÂ£mero de posiciones que se han de desplazar el

cursor desde la posiciÂ¢n actual.

-Si tenemos las siguientes instrucciones:

-50 FORMAT (2A, I2, T1, A, T18, A, I2)

CHARACTER *6 Nombre1, Nombre2, Apellido1*8, Apellido2*8

INTEGER Edad1, Edad

READ (1, 50) Nombre1, Apellido1, Edad1, Nombre2, Apellido2, Edad

y le introducimos la siguiente entrada:

-Josâ   -Martinez8Torralva25.

obtendremos la siguiente lista de variables en la salida:

-Nombre1 - Josâ   -.

-Apellido1 - MartÂ¡nez.

-Edad1 - 0.

-De modo anÂ logo, si tenemos las siguientes instrucciones:

-100 FORMAT (T8, I5, Tl2, I4, Tl6, A6)

CHARACTER *8 Cadena

INTEGER VarA, VarB

READ (*, 100) VarA, VarB, Cadena

y le introducimos la siguiente entrada:

-Visita12345678.

obtendremos la siguiente lista de variables en la salida:

-M - 23456.

-N - 5678.

-Cadena - 345678.

-Estos cÂ¢digos no son repetibles.

CODIGO DE POSICIONAMIENTO "/".

-Este es un cÂ¢digo de posicionamiento vertical que da por terminado un

registro o deja registros o lÂ¡neas vacÂ¡as y su sintaxis es:

-/.

-Puede separase por comas o no y sitÂ£a el puntero en el primer carÂ cter

de un nuevo registro.

-En la salida causa n-1 registros o lÂ¡neas vacÂ¡as y no es un cÂ¢digo de

tipo repetible.

CODIGOS ESPECIALES "S", "Sp" Y "Ss".

-Estos cÂ¢digos sÂ¢lo son vÂ lidos para formatos de salida, para nÂ£meros

y controlan la salida del signo + en los nÂ£meros positivos de forma:

-S no imprime el signo +.

-Sp imprime el signo +.

-Ss no imprime el signo +.

-El formato Sp sÂ¢lo valdrÂ hasta que sea el final de los especificadores

de formato o hasta un cÂ¢digo S o Ss.

-Estos cÂ¢digos no son repetibles.

CODIGOS ESPECIALES "Bn" Y "Bz".

-Estos cÂ¢digos dirigen la interpretaciÂ¢n de los caracteres en blanco

en los formatos numâ ricos de forma que:

-Bn ignora los blancos.

-Bz toma los blancos como ceros.

-Si un campo de entrada estÂ en blanco se considera como cero.

CODIGO ESPECIAL "HOLLERITH".

-Este cÂ¢digo sÂ¢lo puede formar parte de un formato de salida y no es vÂ lido para entradas, siendo su sintaxis:

-wHc.

-Este cÂ¢digo indica que habrÂ una salida de w caracteres que son los que van a aparecer en c. Los apÂ¢strofes se consideran como cualquier otro carÂ cter.

-El nÂ£mero de caracteres de la constante carÂ cter determina el ancho del campo de salida.

CODIGO ESPECIAL ":".

-Este cÂ¢digo provoca la terminaciÂ¢n de la operaciÂ¢n de salida si no hay mÂ s elementos en la lista de la sentencia de salida.

-Si el cÂ¢digo es encontrado durante una entrada de datos o si quedan elementos en la lista de salida es ignorado.

CODIGO ESPECIAL " ".

-Este cÂ¢digo facilita formatear la salida de datos en la pantalla. Este cÂ¢digo se aplica en formatos de sentencias de salida.

-Causa que el procesador suprima en la salida de acceso secuencial la separaciÂ¢n de registros actual y el siguiente registro.

CODIGOS ESPECIALES DE CONTROL DE CARRO.

-Exigen que se escriban los cÂ¢digos como primer carÂ cter dentro de la sentencia WRITE que emplea la impresora como salida siendo sus cÂ¢digos los siguientes:

-Blanco avanza una lÂ¡nea.

-0 avanza dos lÂ¡neas.

-1 sitÂ£a en la primera lÂ¡nea de la siguiente pÂ gina.

-+ imprime sobre la misma lÂ¡nea y no avanza.

-Si el primer carÂ cter no es un signo de estos se toma como blanco. En la mayorÂ¡a de las sentencias FORMAT tiene especificado en la columna 1 o 1X.

REGISTROS MULTIPLES.

-Si el nÂ£mero de elementos de la lista de entrada y salida es menor que el nÂ£mero de cÂ¢digos de datos, los cÂ¢digos sobrantes se ignoran.

-Si el nÂ£mero de elementos es mayor que el nÂ£mero de cÂ¢digos de formato se van asociando los cÂ¢digos de datos con los datos de izquierda a derecha.

-Cuando se alcanza el parâ ntesis de cierre de FORMAT se empieza un nuevo registro repitiâ ndose los cÂ¢digos a partir del parâ ntesis de apertura precedente.

TEMA 5: SUBPROGRAMAS Y FUNCIONES EN FORTRAN.

UNIDADES DE PROGRAMA.

-Ante la necesidad de una organizaciÂ¢n jerÂ rquica Fortran permite dividir el programa en mÂ¢dulos llamados unidades de programa. Hay dos clases de unidades de programa:

-Programa principal.

-Subprograma.

-Cada programa tiene un sÂ¢lo programa principal que puede contener cero o mÂ s Subprogramas que pueden ser:

-Funciones.

-Subrutinas.

-Las Funciones pueden ser de varios tipos:

-Externas, es un mÂ¢dulo independiente.

-IntrÂ¡nsecas, es un mÂ¢dulo independiente.

-UnilÂ¡neas, son locales al mÂ¢dulo o unidad de programa donde estÂ n definidas.

-Las Subrutinas son Subprogramas que pueden ser usados para devolver un conjunto de cero a n datos y suelen ser de propÂ¢sito general.

-Los mÂ¢dulos pueden estar uno a continuaciÂ¢n del otro y no estarÂ n separados a nivel lÂ¢gico aunque sÂ¡ lo estarÂ n a nivel fÂ¡sico.

-La FunciÂ¢n se usa cuando se necesita devolver un sÂ¢lo valor, mientras que las Subrutinas se utilizan para devolver mÂ s valores o en su defecto ninguno.

FUNCIONES SENTENCIA (UNILINEA).

-Es un procedimiento especificado en una sentencia simple, con forma similar a una sentencia de asignaciÂ¢n aritmâ tica, lÂ¢gica o carÂ cter. Este tipo de Funciones representan una fÂ¢rmula.

-Se escribe en la misma unidad de programa que va a ser usada, son locales a la unidad de programa en la que estÂ definida.

-Es una sentencia de tipo no ejecutable y ha de ser escrita antes de ser invocada, su sintaxis suele ser:

- Tipo Identificador (Lista de parÂ metros actuales)=ExpresiÂ¢n.

-Si no se colocan parÂ metros, habrÂ que poner parâ ntesis igualmente.

-El tipo de dato del identificador de la FunciÂ¢n sentencia y los argumentos ficticios estÂ n determinados por el tipo implÂ¡cito del identificador.

-La invocaciÂ¢n se realiza escribiendo su identificador y entre parâ ntesis los argumentos verdaderos o actuales que sustituyen a los argumentos que son ficticios.

-Los argumentos actuales pueden ser expresiones y han de corresponderse

en nÂ£mero, orden y tipo con los argumentos ficticios. La llamada a la FunciÂ¢n debe formar parte de una sentencia ejecutable.

-Se puede invocar el paso de varios parÂ metros separados por una coma o no ser invocados.

FUNCIONES INTRINSECAS.

-Son las Funciones propias del lenguaje Fortran, que ya fueron descritas en el tema 2.

-Estas Funciones se usan con el identificador de la FunciÂ¢n seguido de los valores de los parÂ metros.

-HabrÂ que tener en cuenta que los argumentos sean del mismo tipo y que correspondan dentro del rango.

FUNCIONES EXTERNAS.

-Este tipo de Funciones devuelve un valor a travâ s del identificador de la FunciÂ¢n aunque puede devolver otros valores. Su sintaxis es:

- Tipo FUNCTION Identificador (Lista de parÂ metros formales)

acciÂ¢n1

acciÂ¢n2

.....

acciÂ¢nn

Identificador=ExpresiÂ¢n

.....

RETURN

END

-El tipo es opcional y especifica el tipo de datos del valor que se devuelve. Si no se especifica se siguen las normas del tipo implÂ¡cito de los identificadores.

-Esta sentencia debe ser la sentencia inicial de una Subrutina FUNCTION.

El tipo de la FunciÂ¢n estÂ determinado por la especificaciÂ¢n tipo en la

cabecera.

-Puede tener cualquier sentencia excepto las definiciones de otras

Subrutinas. El valor asignado debe ser del mismo tipo que el de la

FunciÂ¢n.

-El fÂ¡n lÂ¢gico de una FunciÂ¢n es RETURN que puede aparecer mÂ s veces.

La Â£ltima sentencia de cÂ¢digo fuente de definiciÂ¢n de una FunciÂ¢n es

END que serÂ el fÂ¡n fÂ¡sico.

-Desde una FunciÂ¢n se puede invocar otra Subrutina pero no a la misma

FunciÂ¢n y no se permite la recursividad.

LLAMADA A FUNCIONES EXTERNAS.

-La llamada a una FunciÂ¢n ha de formar parte de una sentencia siendo la

sintaxis de la llamada la siguiente:

-Identificador (Lista de parÂ metros actuales).

-La referencia a la FunciÂ¢n se puede realizar desde cualquier otra

unidad de programa. La lista de parÂ metros deberÂ estar separada por

comas.

-Los parÂ metros se pueden pasar por valor colocando la palabra reservada

VALUE delante de las variables. No se debe modificar los valores de las

variables de la lista. Es mejor duplicarlos o protegerlos.

-Los parÂ metros actuales deben coincidir con los argumentos ficticios

con las siguientes especificaciones:

-Constantes, variables o expresiones, excepto la concatenaciÂ¢n

de operandos.

-Nombre de array.

-Funciones intrÂ¡nsecas o externas.

-Subrutinas.

PASO DE ARGUMENTOS A LAS SUBRUTINAS.

-Las llamadas por valor realizan unas copias del argumento verdadero

con lo que â ste no cambia durante la ejecuciÂ¢n de la Subrutina.

-Las llamadas por referencia no realizan esa copia y el argumento

verdadero puede cambiar durante la ejecuciÂ¢n de la Subrutina.

-Para saber cuando se deben pasar valores por valor o referencia nos

fijaremos en la siguiente tabla:

-Por referencia si es una variable, array, elemento de array o

caracteres.

-Por valor si es una constante o una expresiÂ¢n.

SUBRUTINAS.

-Las Subrutinas empiezan con la sentencia SUBROUTINE y puede tener

cualquier sentencia excepto las usadas para definir otros mÂ¢dulos o

unidades de programa.

-Puede devolver cero, uno o mÂ s valores siendo la transmisiÂ¢n por

parÂ metros de cabecera. La ejecuciÂ¢n termina con un fÂ¡n lÂ¢gico o RETURN

siendo la Â£ltima sentencia el fÂ¡n fÂ¡sico o END.

-Si sÂ¢lo hay un RETURN se puede omitir porque END funcionarÂ como si

fuera RETURN. La sintaxis de las Subrutinas es:

-SUBROUTINE Identificador (Lista de parÂ metros formales)

acciÂ¢n1

acciÂ¢n2

.....

acciÂ¢nn

RETURN

.....

END

-El identificador del procedimiento ha de ser Â£nico, el primer carÂ cter debe ser una letra.

-En la Subrutina no se asocia un valor al identificador de la misma, devuelve los datos de salida modificando sus argumentos si son:

-Nombres de variable.

-Nombres de array.

-Subrutinas ficticias.

-Asteriscos.

-El tipo de los argumentos se especifica explÂ¡cita o implÂ¡citamente. El argumento array se define con un tamaÂ¤o fijo, anidado o ajustable.

-El argumento asterisco se usa para la salida mÂ£ltiple de una Subrutina no permitiendo la autollamada o la recursiÂ¢n.

LLAMADA A SUBRUTINAS.

-Una Subrutina puede ser invocada desde otra Subrutina o unidad de programa principal con la siguiente sintaxis:

-CALL Identificador (Lista de parÂ metros actuales).

-Los argumentos deben coincidir en orden y tipo con los argumentos ficticios de la Subrutina referenciada. La ejecuciÂ¢n de CALL causa que el control de la ejecuciÂ¢n pase a la Subrutina referenciada.

-Los argumentos de CALL pueden ser:

-Expresiones.

-Arrays.

-Subrutinas.

-El argumento correspondiente al asterisco ha de ser del tipo *N siendo

N una etiqueta de una sentencia ejecutable. Los valores se transmiten

por referencia.

-Un ejemplo de la utilizaciÂ¢n de Subrutinas podrÂ¡a ser:

-CHARACTER *20 Cab

REAL Gwr

PARAMETER (Cab='EspecificaciÂ¢n de la gravedad')

Gwr=9.82337

WRITE (*, *) Cab, Gwr

CALL Lista (Cab, Gwr)

WRITE (*, *) Cab, Gwr

SUBROUTINE Lista (TÂ¡tulo, Datos)

CHARACTER *20 TÂ¡tulo

REAL Datos

TÂ¡tulo='Densidad Kg/cm'

Datos=0.57975

WRITE (*, *) TÂ¡tulo, Datos

RETURN

END

-El resultado que proporcionarÂ el anterior trozo de cÂ¢digo serÂ un

resultado imprevisible, puesto que no se puede modificar una constante

a una variable.

-La relaciÂ¢n general para el paso de parÂ metros serÂ la siguiente:

ParÂ metros actuales ParÂ metros formales

Variables Nombre variable

Elementos de array Nombre variable

Estructuras Nombre variable

Expresiones Nombre variable

Arrays Array

*Etiqueta Etiqueta

Subrutinas Nombre Â£nico

Funciones Nombre Â£nico

SENTENCIAS "EXTERNAL" E "INTRINSIC".

-Estas sentencias se usan para cuando el argumento es el identificador

simbÂ¢lico de un Subprograma o FunciÂ¢n.

-La sentencia EXTERNAL declara que un identificador es un nombre de una

FunciÂ¢n externa o de una Subrutina. No es una sentencia ejecutable y

debe aparecer antes del cÂ¢digo de sentencias ejecutables.

-La sintaxis de EXTERNAL es:

-EXTERNAL Identificador1, Identificador2, ..., Identificadorn.

-La sentencia INTRINSIC declara que un identificador es nombre de una

FunciÂ¢n intrÂ¡nseca, apareciendo el identificador en la sintaxis:

-INTRINSIC FunciÂ¢n1, FunciÂ¢n2, ..., FunciÂ¢nn.

-Las Funciones intrÂ¡nsecas que no pueden ser argumentos actuales de las

Subrutinas son los siguientes:

-De conversiÂ¢n de tipos, INT, IFIX, IDINT, FLOAT y CHAR.

-LexicogrÂ ficas, LGE, LGT, LLE y LLT.

-De mÂ ximos y mÂ¡nimos.

ENTRADA MULTIPLE A UNA SUBRUTINA. SENTENCIA "ENTRY".

-Sirve para que el comienzo de una Subrutina sea en una sentencia

especÂ¡fica contenida en la Subrutina. Puede estar en cualquier punto

del programa excepto dentro de un bloque IF o DO.

-Esta sentencia es de tipo no ejecutable siendo su sintaxis:

-ENTRY Identificador (Lista de parÂ metros actuales).

-Los identificadores simbÂ¢licos de varios puntos de entrada tienen que ser diferentes entre sÂ¡ y del nombre del procedimiento.

-Un punto ENTRY se referencia desde otro mÂ¢dulo. El proceso prosigue hasta encontrar un RETURN.

SALIDA MULTIPLE DE UNA SUBRUTINA. SENTENCIA "RETURN N".

-Esta sentencia causa que el control de la ejecuciÂ¢n retorne a la unidad de programa desde donde se invocÂ¢. La sintaxis es la siguiente:

-RETURN n.

-La especificaciÂ¢n n se usarÂ en una Subrutina, para devolver el control a una sentencia especÂ¡fica, sin ser la siguiente a la llamada.

-El valor de n ha de estar comprendido entre uno y el nÂ£mero de asteriscos de la cabecera de la Subrutina.

TEMA 6: VECTORES Y MATRICES (ARRAYS).

ARRAYS UNIDIMENSIONALES.

-Consiste en una lista de un nÂ£mero finito de datos del mismo tipo que se referencian por un identificador comÂ£n y un nÂ£mero de orden que son consecutivos.

-Las variables que representan los arrays se denominan variables de subÂ¡ndice. El tamaÂ¤o de un vector es el nÂ£mero de elementos que componen el vector. Una variable de subÂ¡ndice tiene el formato:

-Variable (SubÂ¡ndice).

-La variable puede ser un array de los siguientes tipos:

-Numâ rico.

-Cadena.

-LÂ¢gico.

-Complejo.

-El subÂ¡ndice puede ser:

-Constante numâ rica.

-Variable.

-ExpresiÂ¢n matemÂ tica.

-Para saber una determinada posiciÂ¢n de un elemento se deben cumplir las

siguientes caracterÂ¡sticas:

-Todos los elementos del array son del mismo tipo.

-El vector tiene un nombre Â£nico y los elementos estÂ n ordenados

por el subÂ¡ndice.

DECLARACION DE UN ARRAY. SENTENCIA "DIMENSION".

-Hay dos formas de definir un vector o matriz, utilizando la forma

comÂ£n a todas las versiones cuyo formato serÂ :

-DIMENSION Identificador1 (MÂ¡nimo:MÂ ximo), ..., Identificadorn

(MÂ¡nimo:MÂ ximo).

-El nÂ£mero de elementos que obtendremos vendrÂ dado por la fÂ¢rmula

MÂ ximo-MÂ¡nimo+1, sabiendo que mÂ ximo deberÂ ser mayor o igual que el

valor de mÂ¡nimo.

-El identificador es una variable con las mismas reglas. Opcionalmente

se puede colocar despuâ s del Identificador un alias o ALLOCATE que se

usarÂ cuando estemos en tiempo de ejecuciÂ¢n. Como ejemplos:

-ALLOCATE Array (Valor).

-DIMENSION Array (11:25).

-DIMENSION (14).

-Array (:).

-Lo que indica la opciÂ¢n ALLOCATE es que el valor de sus subÂ¡ndices se va a dimensionar durante la fase de ejecuciÂ¢n.

-El subÂ¡ndice izquierdo puede tomar los valores cero, negativo o positivo al igual que el subÂ¡ndice derecho, pero â ste Â£ltimo debe ser igual o mayor que el subÂ¡ndice izquierdo.

-No se puede modificar la dimensiÂ¢n una vez definida, y es necesario colocar corchetes para definir un array en tiempo de ejecuciÂ¢n.

-En el ejemplo siguiente se declara un array de una dimensiÂ¢n que se dimensionarÂ posteriormente:

-DIMENSION Array ALLOCATE (:).

-Para desasignar el array y liberar la memoria se utilizarÂ la opciÂ¢n DEALLOCATE como sigue:

-DEALLOCATE Array (Valor).

-Las dimensiones se separan por comas y se pueden definir como mÂ ximo siete dimensiones, aunque pueden ser mÂ s dependiendo de la memoria del ordenador.

-Si el lÂ¡mite es uno se puede omitir el valor de mÂ¡nimo, quedando la sintaxis siguiente:

-DIMENSION Identificador1 (MÂ ximo), ..., Identificadorn (MÂ ximo).

-La segunda forma de definir vectores es con la especificaciÂ¢n de tipos de sintaxis:

-Tipo Identificador (MÂ¡nimo:MÂ ximo).

-Tipo es cualquiera de los tipos definidos en Fortran. Mediante esta forma se asocia el tipo de dato al identificador y definirlo como un array con tantos elementos como haya.

-Los lÂ¡mites inferiores y superiores son expresiones enteras siempre

con constantes nunca variables.

-Otros atributos que podemos colocar despuâ s del identificador son:

-REFERENCE.

-C.

-PASCAL.

-VALUE.

-Se usarÂ n estas opciones cuando se va a pasar el array por parÂ metro a otros lenguajes, por lo que habrÂ que normalizar el lenguaje.

-Los arrays se almacenan en la memoria de una forma distinta que en otros lenguajes, ya que se almacenan columnas a columnas unas a continuaciÂ¢n de otras.

-Se almacenan en orden creciente de sus subÂ¡ndices a partir de una posiciÂ¢n determinada. Para sacar los elementos por filas se usarÂ el DO implÂ¡cito.

OPERACIONES CON ARRAYS Y ELEMENTOS DE UN ARRAY.

-Hay dos tipos de operaciones bÂ sicas, con los elementos que serÂ n las mismas que permiten los tipos de datos o con estructuras como el recorrido de un array.

-Para hacer el recorrido de un array se usarÂ una estructura del tipo FOR como:

-DO Etiqueta VarA=Izquierdo, Derecho

acciÂ¢n1

acciÂ¢n2

.....

acciÂ¢nn

Etiqueta CONTINUE

ENDDO

-Se puede leer o escribir un array completo con sÂ¢lo poner el

Identificador del array en la sentencia de E/S.

-Se puede inicializar un array completo con sÂ¢lo poner su identificador

y los valores que se inicializan en la sentencia DATA.

-Consiste en obtener datos de un dispositivo externo y almacenarlos en

un vector o escribir en un dispositivo los datos de un array.

-La E/S puede realizarse con el identificador del array y por tanto

se procesa en total o elemento a elemento con un DO implÂ¡cito o

explÂ¡cito.

ENTRADA Y SALIDA DE ARRAYS CON DO "IMPLICITO".

-Se utiliza en vez de la estructura de tipo FOR. Esta estructura estÂ

ya definida y hace sencilla la manipulaciÂ¢n de los arrays en la lectura

y escritura.

-Con este tipo de DO se ejecuta una sÂ¢la vez la sentencia READ de modo

que se lee un sÂ¢lo registro fÂ¡sico. Su sintaxis es:

-READ (*, *) (Identificador (VarA), VarA=Inicio, Final,

Incremento).

-Cuando el incremento es uno se puede omitir. Tambiâ n puede emplearse

para acceder a los elementos de un array de mÂ s de una dimensiÂ¢n para

leer o escribir e inicializar.

-El DO implÂ¡cito se puede anidar con tantos niveles como sea necesario

o tantos niveles como dimensiones tenga el array.

-Un ejemplo del uso del DO implÂ¡cito serÂ¡a:

-READ (*, *) (Array (VarA), VarA=1, 20).

-El ejemplo anterior lee veinte variables estableciâ ndose un bucle. Otro

ejemplo:

-READ (*, *) (Array (VarA), VarA=Izdo, Dcho, Paso).

-En el ejemplo anterior Paso indica que la variable toma el valor de

izdo y sus sucesivos valores repitiendo el proceso hasta que dcho

tiene mayor valor que izdo.

-Pero si paso es negativo entonces dcho ha de ser menor forzosamente. El

siguiente ejemplo tendrÂ¡a como representaciÂ¢n con DO la siguiente:

-READ (*, 100) (Array (VarA), VarB=1)

READ (*, *) ((Array (VarA, VarB), VarB=1, VarC), VarA=1, VarD)

-DO VarA=1, VarD

DO VarB=1, VarC

READ (*, *) (Array (VarA, VarB))

ENDDO

ENDDO

PASO DE ARRAYS POR PARAMETRO.

-El paso de un array se realiza por referencia y cuando un array es

pasado, en realidad se pasa la direcciÂ¢n en memoria del primer elemento

ahorrÂ ndose memoria y espacio.

-Un ejemplo de ello es el siguiente:

-REAL Array (100, 200), VarA, VarB

CALL Lista (Array, VarA, VarB)

SUBROUTINE Lista (Arrayauxiliar, VarC, VarD)

-En la definiciÂ¢n de los argumentos ficticios para los arrays en las

Subrutinas o Funciones, no es necesario que sean iguales los lÂ¡mites

superior e inferior de cada dimensiÂ¢n con los lÂ¡mites del argumento

actual del array transmitido.

-Siempre se exige que una variable sea dimensionada por lo que dentro de la Subrutina se colocarÂ la siguiente declaraciÂ¢n:

-REAL Arrayauxiliar (100, 200).

-En esta Subrutina no se crea la variable arrayauxiliar sino que se define dicha variable. El array en esa declaraciÂ¢n se puede ajustar siempre que la dimensiÂ¢n sea menor que la declarada anteriormente.

-En todo caso el tamaÂ¤o del argumento ficticio para el array no puede ser mayor que el del argumento actual.

-Se puede poner un asterisco que es la opciÂ¢n por defecto y que indica que toma el valor de la dimensiÂ¢n iniciada en el programa principal.

ARRAYS DE TAMAÃ O AJUSTABLE Y TAMAÃ O ASUMIDO.

-El argumento ficticio array en la FunciÂ¢n MÂ ximo su tamaÂ¤o se ajusta a N elementos que es un dato transmitido, y por tanto la definiciÂ¢n de arrays ajustables.

-La definiciÂ¢n de un argumento ficticio array de tamaÂ¤o ajustable en una FunciÂ¢n o en una Subrutina es la Â£nica situaciÂ¢n en la que una definiciÂ¢n de array puede incluir una variable en la especificaciÂ¢n del rango de cada dimensiÂ¢n.

-Para definir un argumento ficticio array como asumido se especifica el lÂ¡mite superior de la Â£ltima dimensiÂ¢n del array con un asterisco. El nÂ£mero de elementos del array ficticio es el mismo que el array pasado por parÂ metro.

-SÂ¢lo se puede especificar el lÂ¡mite superior de la Â£ltima dimensiÂ¢n con un asterisco, aunque para los caracteres tambiâ n es vÂ lido.

-Para las Funciones carÂ cter externas usaremos la siguiente sintaxis:

-CHARACTER * (*) FUNCTION Identificador (Lista de parÂ metros).

TEMA 7: SENTENCIAS ESPECIALES DE FORTRAN.

SENTENCIA "EQUIVALENCE".

-Esta sentencia declara que dos o mÂ s identificadores son equivalentes

y al menos dos. Hace que compartan la misma posiciÂ¢n de almacenamiento

que puede ser referenciado de mÂ s de una forma.

-La sintaxis de esta orden es la siguiente:

-EQUIVALENCE (Lista de Identificadores).

-En una misma sentencia EQUIVALENCE pueden haber mÂ s de una lista de

identificadores, cada una de las cuales se refiere a la misma posiciÂ¢n

de almacenamiento:

-EQUIVALENCE (Lista de Identificadores),

+ (Lista de Identificadores).

-Las variables de distinto tipo pueden hacerse equivalentes, de forma

que habrÂ almacenamiento compartido pero no una posible conversiÂ¢n de

tipos.

-La sentencia de almacenamiento de los identificadores de la lista

comienza con la primera unidad de almacenamiento de las entidades de

la lista.

-Se pueden hacer equivalentes los arrays y los elementos de los arrays.

Los Â¡ndices de las variables array escritos en una lista de la

sentencia deben ser constantes o expresiones formadas por constantes.

-En una sentencia EQUIVALENCE no puede provocarse que una misma variable

ocupe dos posiciones de memoria distintas.

-Esta sentencia puede utilizarse entre variables o arrays de tipo

carÂ cter aunque sus longitudes no sean las mismas.

-Las variables carÂ cter equivalentes tendrÂ n el mismo primer carÂ cter

ubicado en la misma posiciÂ¢n de memoria.

-En Fortran 77 no estÂ permitido hacer equivalentes variables de tipo carÂ cter con variables numâ ricas, y los nombres de FunciÂ¢n no pueden hacerse equivalentes con otra entidad.

-Esta sentencia se suele utilizar cuando la memoria de que se dispone es muy pequeÂ¤a, pero produce que la lÂ¢gica del programa pueda acceder a la variable correcta.

SENTENCIA "COMMON".

-Una forma de comunicarse entre el programa o Unidad de programa que llama a una Subrutina o FunciÂ¢n, y la Subrutina o FunciÂ¢n llamada, es a travâ s de la lista de argumentos.

-Con estos argumentos se referencian zonas de memoria comunes desde distintas Unidades de programa, la Unidad de programa que llama y la Unidad llamada.

-Con esta sentencia se pueden definir zonas comunes de memoria entre diversas Unidades de que forman parte un programa, entre la Unidad principal y una o varias Subrutinas o Funciones.

-Es una sentencia no ejecutable que debe aparecer en la Unidad de programa que llama y en el Subprograma llamado antes de todas las sentencias ejecutables.

-En esta sentencia se listan los nombres de las variables y los nombres de los arrays con su dimensiÂ¢n y es una forma alternativa de comunicar datos a la lista de argumentos de los Subprogramas.

-Si un array va ha estar en una zona comÂ£n se puede definir su lista de identificadores en la lista COMMON o definir el array y ponerlo en la lista de la sentencia COMMON.

-Las variables y los arrays son asignados a un almacenamiento comÂ£n en el orden en que aparecen en la sentencia COMMON, siendo la sintaxis de esta:

-COMMON Identificador1, Identificador2, ..., Identificadorn.

-La sintaxis anterior pertenece a una sentencia COMMON sin nombre o en blanco, pero existe una sentencia COMMON etiquetada o con nombre.

-El orden de los identificadores especificados en una sentencia COMMON determina la equivalencia de identificadores simbÂ¢licos entre varias Unidades de un programa (El bloque comÂ£n de memoria es lo global).

-La lista de identificadores deben ser de igual tipo en todas las sentencias COMMON estableciâ ndose las equivalencias por el orden en que estÂ n en la lista.

-Un identificador de una lista COMMON no puede figurar como argumento en una Subrutina o FunciÂ¢n ni como parÂ metro en las llamadas (Puede haber un solapamiento de memoria).

-El bloque comÂ£n de memoria es Â£nico para todo el programa. La memoria comÂ£n se establece de forma contigua.

-En un bloque comÂ£n de memoria no puede haber variables carÂ cter y no carÂ cter mezcladas, siendo todo de variables carÂ cter o variables no carÂ cter.

-Las variables o arrays que figuran en la lista de COMMON sin nombre no pueden ser inicializadas con DATA, sÂ¢lo pueden inicializarse con sentencias de asignaciÂ¢n.

SENTENCIA "COMMON" CON NOMBRE. USO CONJUNTO DE "COMMON" Y "EQUIVALENCE".

-Esta sentencia etiquetada permite definir varias zonas comunes de memoria, cada una con su nombre o etiqueta.

-Se forman de igual manera que las anteriores pero se escribe su nombre

o etiqueta con dos barras (Slash) antes de la lista de variables.

-El nombre de COMMON debe ser un identificador vÂ lido y pueden definirse

dos o mÂ s zonas comunes. Es conveniente usar tantas sentencias COMMON

como zonas comunes haya.

-Se pueden mezclar en una zona comÂ£n variables numâ ricas con variables

o arrays lÂ¢gicos.

-En el siguiente ejemplo tendremos sÂ¢lamente dos bloques COMMON uno

con nombre y otro sin nombre:

-COMMON /Nombre-1/ VarA, VarB, VarC

COMMON VarH, VarI, VarJ

COMMON /Nombre-1/ VarK, VarL

COMMON VarM, VarN

-Una comparaciÂ¢n entre un trozo de programa con COMMON y otro sin â l

podrÂ¡a ser el siguiente:

-REAL VarD, VarE, VarF, VarG, VarZ

READ *, VarD, VarE, VarF, VarG

VarZ=2

VarT=FunciÂ¢n (VarD, VarE, VarF, VarG)

STOP

END

FUNCTION FunciÂ¢n (VarA, VarB, VarC, VarX)

VarI=VarA*VarX**2+VarB*VarX+VarC

RETURN

END

-REAL VarD, VarE, VarF, VarG, VarZ

```
COMMON /Coeficientes/ VarD, VarE, VarF
VarZ=2
VarT=FunciÂ¢n (VarZ)
STOP
END
FUNCTION FunciÂ¢n (VarX)
COMMON /Coeficientes/ VarA, VarB, VarC
VarF=VarA*VarX**2+VarB*VarX+VarC
RETURN
END
```

-Si en una sentencia COMMON se precede a los identificadores de dos barras o slash indicarÂ que son zonas comunes sin nombre. Es indiferente el orden de definiciÂ¢n de las zonas comunes.

-Los arrays o variables de COMMON con etiqueta pueden ser inicializados con la sentencia DATA, pero no las variables de un COMMON sin nombre.

-Un COMMON con etiqueta tiene que tener el mismo tamaÂ¤o en todas las Unidades de programa.

-Las variables o arrays pueden aparecer en ambas sentencias COMMON y EQUIVALENCE siempre que no causen conflicto en el orden en el que se almacenan.

-Dos variables que estâ n en una zona comÂ£n no pueden ser equivalentes entre sÂ¡.

SENTENCIA "SAVE".

-Esta sentencia se utiliza para almacenar estÂ ticamente los valores o datos de una invocaciÂ¢n a otra.

-SAVE declara que las variables locales y los arrays sean retenidos

despuâ s de ejecutar RETURN o la siguiente llamada al Subprograma.

-Las variables o arrays locales contendrÂ n el Â£ltimo valor adquirido

en la ejecuciÂ¢n anterior al Subprograma. Su sintaxis es:

-SAVE (Lista de identificadores).

-La lista de identificadores, que podrÂ n ser variables, arrays o bloques

COMMON, retendrÂ el Â£ltimo valor adquirido antes de la ejecuciÂ¢n de

RETURN.

-Las excepciones en las que las variables no quedan indefinidas al salir

de un Subprograma son:

-Sentencias SAVE del Subprograma.

-Bloques COMMON en blanco o sin nombre.

-Bloques COMMON etiquetados y definidos en la Unidad de programa

principal y uno o mÂ s Subprogramas.

-Las variables en bloques COMMON etiquetados en Subprogramas quedan

indefinidos sÂ¢lo cuando hay una salida desde un Subprograma pero no se

pueden retener con SAVE y el nombre del bloque COMMON.

-En una lista de SAVE no pueden aparecer parÂ metros de Subrutinas o

Funciones, ni nombres de FunciÂ¢n y Subrutinas ni variables o arrays

de bloques COMMON.

-Puede haber mÂ s de un SAVE o escribir la lista en un sÂ¢lo SAVE. Si hay

mÂ s de un SAVE no podrÂ n repetirse nombres de variables o arrays.

INICIALIZACION DE VARIABLES. SENTENCIA "DATA".

-Esta sentencia permite inicializar variables con la siguiente sintaxis:

-DATA Lista1 /Constantes/, ..., Listan /Constantes/.

-La lista contendrÂ los nombres de las variables o arrays a inicializar

separados por comas e inicializa las constantes que vendrÂ n separadas

por comas.

-Puede especificarse mÂ s de una lista y sus constantes o agrupar todas las variables en una sÂ¢la lista. La inicializaciÂ¢n se realiza en el orden en el que aparecen, de izquierda a derecha.

-Si hay constantes consecutivas iguales se podrÂ poner:

-NÂ£mero*Constante.

-En el formato anterior NÂ£mero es el nÂ£mero de repeticiones que se especificarÂ de dicha forma.

-Las reglas mÂ s importantes de las sentencias DATA son las siguientes:

-El nÂ£mero de constantes ha de ser igual al nÂ£mero de elementos de la lista, variables o arrays.

-No pueden aparecer argumentos ficticios de Subprogramas, nombres de FunciÂ¢n y elementos de bloque COMMON en blanco.

-SÂ¢lo pueden estar los bloques COMMON etiquetados en los bloques DATA.

-El tipo de variable o array a inicializar debe corresponderse con el tipo de la constante. Lo mismo para las variables de tipo carÂ cter (Si hay exceso se ignora y si hay defecto se completa con blancos a la derecha).

-Esta sentencia no es ejecutable y puede aparecer despuâ s de las especificaciones de datos pero es mejor colocarlas antes de las sentencias ejecutables.

-Las sentencias DATA pueden utilizar el DO implÂ¡cito para inicializar el array a los datos que se quieran o sÂ¢lo definirlo. Un ejemplo de ello serÂ¡a:

-INTEGER VarA, Orden, Alfa, Lista (100)

REAL Coeficiente (4), Epsilon (2), Pi (5), VarX (5, 5)

CHARACTER *15 Ayuda

DATA VarA /0/, Orden /3/

DATA Coeficiente /1.0, 2*3.0, 1.0/, Epsilon (1) /0.0001/

DATA ((VarX (VarI, VarJ), VarI=1, VarJ), VarJ=1.5) /15*1.0/

DATA Lista /100*0/

DATA Ayuda /'Ayuda'/

-El siguiente formato puede ir en cualquier parte del programa y tiene

como misiÂ¢n hacer una llamada al camino o ruta especificado:

-$INCLUDE 'Path Nombre.For'.

SENTENCIA "PARAMETER".

-Esta sentencia identifica constantes mediante nombres identificadores

o simbÂ¢licos para que despuâ s se pueda hacer referencia a la constante

por el identificador. La sintaxis de la orden es:

-PARAMETER (Identificador=ExpresiÂ¢n constante).

-Dentro de los parâ ntesis se pueden especificar tantos parÂ metros como

se quiera separÂ ndolos por comas.

-La expresiÂ¢n debe coincidir en su tipo con el identificador y el valor

que va ha representar el identificador al evaluarse la expresiÂ¢n. Debe

ajustarse a las reglas establecidas para las sentencias de asignaciÂ¢n.

-El tipo de dato del identificador puede definirse de forma implÂ¡cita

o explÂ¡cita, describiendo la sentencia de definiciÂ¢n antes.

-La expresiÂ¢n constante puede hacer referencia a otro parÂ metro pero

ha de estar definida antes en otro PARAMETER o en la misma sentencia

PARAMETER.

-Un identificador de constante no puede cambiar despuâ s el valor que se

le ha impuesto.

-El Â mbito de los parÂ metros es la Unidad de programa en que estÂ n definidos y una vez que el parÂ metro es definido puede ser referenciado en los sitios en que pueden referenciarse las constantes excepto:

-No se pueden usar los parÂ metros en una especificaciÂ¢n de un formato.

-Un parÂ metro no puede usarse como parte de otra constante.

-Esta sentencia no es ejecutable y puede aparecer despuâ s de cualquier sentencia de especificaciÂ¢n de tipo o antes de que se haga uso del parÂ metro.

-Cuando un valor aparece varias veces en una Unidad de programa se debe asociarle un nombre simbÂ¢lico y usar dicho nombre para despuâ s hacer la referencia a dicho nombre.

INICIALIZACION DE UN COMMON CON NOMBRE. SUBPROGRAMA "BLOCK DATA".

-Este Subprograma asigna valores iniciales a variables y a arrays de un COMMON etiquetado, puesto que los COMMON en blanco se inicializan en las sentencias DATA.

-La sentencia BLOCK DATA que puede tener un identificador termina con la sentencia END y las sentencias que se pueden especificar son todas las no ejecutables para la inicializaciÂ¢n de la lista de COMMON etiquetados y que son:

-IMPLICIT.

-PARAMETER.

-DIMENSION.

-SAVE.

-COMMON.

-EQUIVALENCE.

-DATA.

-La sintaxis de este Subprograma serÂ :

-BLOCK DATA Identificador

sentencia1

sentencia2

.....

sentencian

END

-Si el identificador se coloca es considerado como un identificador

global y no puede coincidir con el nombre de una FunciÂ¢n o el de una

Subrutina.

-El COMMON etiquetado al inicializarlo hay que especificarlo en el

Subprograma BLOCK DATA de forma completa aunque haya variables no

inicializables.

-En un programa ejecutable pueden haber mÂ s de un BLOCK DATA pero sÂ¢lo

uno puede ser sin nombre y todos los demÂ s nombres distintos. Un COMMON

sÂ¢lo puede estar en un sÂ¢lo BLOCK DATA.

-Un ejemplo del uso de este Subprograma serÂ¡a:

-BLOCK DATA Nombre

COMPLEX VarA, VarB

LOGICAL VarC, VarD

INTEGER VarI, VarJ, VarK, Lista

REAL VarX

COMMON /Bloque1/ VarX (10), VarI, VarJ, VarA

COMMON /Bloque2/ Lista (6), VarC, VarD, VarB

DATA VarX /10*0.0/, VarI, VarJ /1, 0/, VarC /False/

DATA VarA, VarB, /2*(0, 1)/

END

-Normalmente este tipo de Subprogramas se suelen colocar cuando acaba el programa principal.

TIPO DE DATO COMPLEJO.

-Este tipo de datos se representa por un par ordenado de nÂ£meros reales de doble precisiÂ¢n, enteros o una combinaciÂ¢n de ellos encerrados entre parâ ntesis y separados por comas.

-Para definir un identificador de tipo complejo contamos con COMPLEX, y pueden haber variables, arrays complejos y Funciones complejas.

-La memoria que ocupa una variable compleja es el doble de una variable real. Se puede asignar una constante compleja, otra variable compleja o una expresiÂ¢n compleja inicializÂ ndose con DATA.

-Cuando a la variable compleja se le quiere asignar un nÂ£mero complejo que tiene la parte real, la parte imaginaria o ambas debe usarse la FunciÂ¢n intrÂ¡nseca COMPLEX.

-Los nÂ£meros complejos pueden sumarse, restarse, multiplicarse, elevarse a una potencia y dividirse. No puede usarse en la expresiÂ¢n aritmâ tica de la sentencia IF aritmâ tico, y no puede usarse como subÂ¡ndice de un array.

-Otras Funciones que tienen los nÂ£meros complejos son:

-AIMAG (ExpresiÂ¢n numâ rica) (Parte imaginaria como nÂ£mero real).

-CONJ (ExpresiÂ¢n numâ rica) (Devuelve el complejo conjugado).

-Otras Funciones internas que tienen un nombre especÂ¡fico para el argumento complejo y el valor que devuelven es tambiâ n complejo son:

-CSQRT (ExpresiÂ¢n numâ rica) (RaÂ¡z cuadrada de un complejo).

-CABS (ExpresiÂ¢n numâ rica) (MÂ¢dulo del complejo).

-CEXP (ExpresiÂ¢n numâ rica) (FunciÂ¢n exponencial de un complejo).

-CLOG (ExpresiÂ¢n numâ rica) (Logaritmo natural de un complejo).

-CSIN (ExpresiÂ¢n numâ rica) (Seno de un complejo).

-CCOS (ExpresiÂ¢n numâ rica) (Coseno de un complejo).

SENTENCIA "PAUSE".

-Esta sentencia hace una parada temporal en la ejecuciÂ¢n de un programa

para detener la salida hasta que el usuario haya podido leer toda la

informaciÂ¢n. Su sintaxis es:

-PAUSE.

-PAUSE 'Cadena de caracteres'.

-PAUSE NÂ£mero (Constante de hasta cinco dÂ¡gitos).

-Al producirse la parada se visualiza un mensaje propio regido por el

nÂ£mero indicado o la cadena si ha sido especificada.

ASIGNACION DE ETIQUETAS A VARIABLES ENTERAS. SENTENCIA "ASSIGN TO".

-Esta sentencia permite asignar un nÂ£mero de etiquetas por una constante

entera a una variable entera siendo su sintaxis:

-ASSIGN Etiqueta TO Variable.

-Etiqueta es la etiqueta de una sentencia ejecutable o de una sentencia

FORMAT, siendo variable el identificador de una variable entera.

-Despuâ s de la ejecuciÂ¢n de ASSIGN el valor de la variable no puede ser

considerada como un dato entero.

-Si la etiqueta asignada a la variable es la de una sentencia FORMAT, la

variable puede ser usada como un identificador de formato.

-Si la etiqueta asignada es ejecutable la variable puede ser usada en

un GOTO asignado como:

-GOTO Variable.

-GOTO Variable (Etiqueta1, Etiqueta2, ..., Etiquetan).

-Cuando se ejecuta una sentencia GOTO asignada el control del programa es transferido a la sentencia con la etiqueta del Â£ltimo valor asignado a la variable con ASSIGN.

TEMA 8: TRATAMIENTO DE CADENAS EN FORTRAN.

CADENAS DE CARACTERES.

-Una cadena es un conjunto de caracteres encerrados entre apÂ¢strofes. Si se quiere representar un apÂ¢strofe dentro de una cadena se deberÂ representar por dos apÂ¢strofes consecutivos.

-Una cadena se declara con la siguiente sintaxis:

-CHARACTER *NÂ£mero Lista de variables.

-AquÂ¡ nÂ£mero representa el nÂ£mero de caracteres de las variables de una cadena. Un array que contiene caracteres se define con una sentencia CHARACTER y declarada de dos modos distintos.

LAS CADENAS COMO ARGUMENTO DE SUBPROGRAMAS.

-Un Subprograma puede especificar una cadena de caracteres sin darle una longitud especÂ¡fica y equivale a la longitud de una array con una variable entera.

-Se puede definir en un Subprograma un array de n variables cada una con su cadena de caracteres sin especificar la longitud de cadena en la sentencia CHARACTER.

-La longitud de la cadena es siempre positiva nunca igual a cero, y dicha longitud no se puede alterar aunque sÂ¡ asignar cadenas cuya longitud es diferente.

-Una cadena con longitud mÂ s corta que la de la variable, rellena a

blancos por la derecha y si es mÂ s larga la trunca.

ASIGNACION DE VALORES A LAS CADENAS.

-Se realiza con la sentencia de asignaciÂ¢n y una constante de caracteres

usÂ ndose una variable cadena para inicializar otra variable de cadena.

-Si los caracteres asignados no coinciden con la longitud se rellenan

a blancos y si es mayor que la longitud se trunca por la derecha.

COMPARACION DE CADENAS.

-Esta comparaciÂ¢n se realiza carÂ cter a carÂ cter de izquierda a derecha

con las siguientes reglas:

-Si las cadenas tienen igual longitud y los caracteres son los

mismos, las cadenas son iguales.

-Si una cadena es mÂ s corta que la otra se aÂ¤aden blancos a la

derecha de la otra cadena, de modo que pueda proceder a la

evaluaciÂ¢n como si las cadenas fueran iguales.

-Las reglas de ordenaciÂ¢n tÂ¡picas son las siguientes:

-Las letras mayÂ£sculas estÂ n ordenadas de A a Z.

-Los dÂ¡gitos ordenados de 0 a 9.

-El carÂ cter blanco es menor que cualquier letra o nÂ£mero.

SUBCADENAS.

-Es cualquier cadena que representa un subconjunto de la cadena original

y mantiene el orden original. Para especificar una subcadena de una

variable de carÂ cter o un elemento de un array de carÂ cter se usa:

-Nombrecadena (ExpresiÂ¢n1 : ExpresiÂ¢n2).

-ExpresiÂ¢n1 es la posiciÂ¢n en nombrecadena del primer carÂ cter de la

subcadena y expresiÂ¢n2 es la posiciÂ¢n en nombrecadena del Â£ltimo

carÂ cter de la subcadena.

-ExpresiÂ¢n1 y expresiÂ¢n2 deben ser del tipo entero y cumplir:

-1<=ExpresiÂ¢n1<=ExpresiÂ¢n2<=Longitud de la cadena.

-Si se omite la expresiÂ¢n1 se toma por defecto uno. Si se omite la expresiÂ¢n2 se toma el valor de la longitud de la cadena original, siendo la subcadena:

-ExpresiÂ¢n2-ExpresiÂ¢n1+1.

CONCATENACION DE CADENAS.

-Consiste en combinar dos o mÂ s cadenas de caracteres en una Â£nica cadena, siendo el operador que realiza la concatencaciÂ¢n o uniÂ¢n de cadenas el siguiente:

-//.

FUNCION LONGITUD. SENTENCIAS "LEN" Y "GETLEN".

-LEN determina la longitud de la cadena de caracteres argumento siendo su sintaxis:

-LEN (Cadena de caracteres).

-Si la cadena de caracteres es una constante de carÂ cter su longitud es el nÂ£mero de caracteres. Si es una variable de cadena o elemento de array la longitud es la definida en la declaraciÂ¢n.

-Si cadena es una subcadena con el formato (ExpresiÂ¢n1:ExpresiÂ¢n2) su longitud es la siguiente:

-ExpresiÂ¢n2-ExpresiÂ¢n1+1.

-GETLEN calcula la longitud de una cadena de caracteres excluyendo a los caracteres en blanco siendo su formato:

-GETLEN (Cadena de caracteres).

FUNCIONES DE TRATAMIENTO DE CARACTERES. SENTENCIAS "CHAR" E "ICHAR".

-CHAR determina el carÂ cter de la cadena que ocupa la posiciÂ¢n relativa en la secuencia de caracteres ASCII siendo su sintaxis:

-CHAR (PosiciÂ¢n).

-El valor de posiciÂ¢n debe estar entre 0 y 255 caracteres de la cadena.

-ICHAR es la FunciÂ¢n inversa de CHAR. El argumento es un carÂ cter y la FunciÂ¢n devuelve un entero que es la posiciÂ¢n del carÂ cter en la secuencia ordenada de caracteres ASCII con el formato:

-ICHAR (CarÂ cter).

FUNCION DE BUSQUEDA. SENTENCIA "INDEX".

-Esta FunciÂ¢n localiza una subcadena dentro de otra. Devuelve un valor entero que indica la posiciÂ¢n inicial de la cadena de caracteres destino dentro de la cadena original siendo su sintaxis:

-INDEX (Cadena fuente, Cadena destino).

-Si la cadena destino no existe el formato devuelve el valor cero.

OTRAS FUNCIONES.

-La FunciÂ¢n LEN_TRIM devuelve la longitud de la cadena dada sin los espacios en blanco siendo su sintaxis:

-LEN_TRIM (Cadena de caracteres).

-La FunciÂ¢n SCAN busca una subcadena en una cadena dada y muestra la primera posiciÂ¢n en la que coinciden ambas cadenas, buscando carÂ cter a carÂ cter, siendo su sintaxis:

-SCAN (Cadena1, Cadena2).

-La FunciÂ¢n VERIFY devuelve un entero y verifica que una cadena estÂ incluÂ¡da en otra, devolviendo la posiciÂ¢n del carÂ cter que sea distinto de los demÂ s y siendo su sintaxis:

-VERIFY (Cadena1, Cadena2).

-Otras Funciones que devuelven un valor lÂ¢gico y que sirven para la comparaciÂ¢n son:

-LGE (Cadena1, Cadena2), verifica si cadena1 es mayor o igual que cadena2.

-LGT (Cadena1, Cadena2), verifica si cadena1 es mayor que cadena2.

-LLE (Cadena1, Cadena2), verifica si cadena1 es menor o igual que cadena2.

-LLT (Cadena1, Cadena2), verifica si cadena1 es menor que cadena2.

-En estas cuatro Â£ltimas Funciones el argumento debe ser siempre un carÂ cter.

TEMA 9: FICHEROS.

INTRODUCCION.

-

TEMA 9: FICHEROS.

ESTRUCTURA DE UN FICHERO.

-Un fichero es una colecciÃ³n de datos organizados de alguna manera y almacenados generalmente en disco o cinta.

-Un fichero estÃ¡ formado por registros y estos constan de campos que pueden ser numÃ©ricos o de caracteres.

-Los ficheros formateados pueden ser editados, imprimirse o visualizarse mientras que los ficheros no formateados no pueden hacer esas acciones. Aunque la lectura y la escritura son mÃ¡s rÃ¡pidas y ocupan poca memoria.

ORGANIZACION DE FICHEROS.

-Se consideran dos tipos de acceso a los registros de un fichero que son los siguientes:

-Acceso Secuencial.

-Acceso Directo.

-El acceso Secuencial implica el acceso a un fichero segÃºn el orden de almacenamiento de sus registros, uno a uno.

-El acceso Directo implica situarse en un registro determinado sin que ello implique la consulta de los registros precedentes.

-La OrganizaciÃ³n de un fichero define la forma en que los registros se disponen sobre el soporte de almacenamiento:

-OrganizaciÃ³n Secuencial.

-OrganizaciÃ³n Directa.

-OrganizaciÃ³n Indexada.

-Un fichero con organizaciÃ³n Secuencial es una sucesiÃ³n de registros almacenados consecutivamente sobre un soporte externo de tal modo que para acceder al registro n, necesariamente hay que pasar por los n-1 registros precedentes.

-Un fichero con organizaciÃ³n Directa exige soporte direccionable. Cada posiciÃ³n se localiza por su direcciÃ³n absoluta, que en el caso del disco suele venir definida por nÃºmero de pista y de sector.

-El lenguaje Fortran no es muy fuerte es la manipulaciÃ³n de archivos aunque tiene capacidad para manipularlos.

-Los archivos suelen ser pequeÃ±os y la informaciÃ³n a tratar suele estar en sentencias DATA.

APERTURA DE UN FICHERO. SENTENCIA "OPEN".

-Esta sentencia es la encargada de la apertura de un fichero, conectando un fichero a un nÃºmero de Unidad, de forma que para referirse despuÃ©s al fichero se harÃ¡ con el nÃºmero de Unidad establecido en la apertura.

-Cuando se quiere crear un fichero en un programa se utilizarÃ¡ la

sentencia OPEN para que quede en una Unidad.

-La sentencia OPEN tambiÃ©n es usada para declarar las propiedades del

fichero, si es Secuencial o Directo, si es Formateado o no Formateado

y otras propiedades.

-La sintaxis de la sentencia serÃ¡ la siguiente:

-OPEN (UNIT= N, FILE='Nombre' , ACCESS=Tipo , FORM=Formato ,

STATUS=Estado , IOSTAT=N , ERR=Etiqueta , BLANK=Tipo ,

RECL=N , BLOCKSIZE=N , MODE=Tipo).

-La sentencia OPEN es la primera sentencia que debe aparecer al utilizar

un fichero.

-La opciÃ³n UNIT siempre debe figurar en la sentencia OPEN y se podrÃ¡

omitir, en cuyo caso N deberÃ¡ figurar como primer parÃ¡metro.

-Esta opciÃ³n se encarga de asignar un canal de comunicaciÃ³n para

ejecutar el archivo.

-La opciÃ³n FILE indica el nombre del fichero que va a estar conectado a

la unidad N. Si no se coloca el nombre del fichero se podrÃ¡ pasar por

parÃ¡metro el nombre de dicho fichero.

-La opciÃ³n ACCESS, donde Tipo es una expresiÃ³n de tipo carÃ¡cter e indica

el tipo de fichero que se va a utilizar, siendo por defecto Secuencial

y puediendo tomar los valores:

-SEQUENTIAL.

-DIRECT.

-La opciÃ³n FORM, donde Formato es una expresiÃ³n de tipo carÃ¡cter en que

el valor por defecto es Secuencial, asigna un formato al fichero,

siendo los valores que puede tomar:

-FORMATTED.

-UNFORMATTED.

-BINARY.

-Si la opciÃ³n es omitida OPEN asumirÃ¡ FORMATTED si el fichero es de tipo Secuencial y UNFORMATTED si es un fichero Directo. BINARY se utiliza en Fortran 90 y representa a los ficheros binarios.

-La opciÃ³n STATUS, donde Estado es una expresiÃ³n de tipo carÃ¡cter y se usa para saber si el fichero ya existe o es nuevo y por tanto va a ser creado. Sus opciones son las siguientes:

-OLD (Si el fichero existe, sino produce error).

-NEW (Crea el nuevo fichero y FILE no debe existir en la Unidad).

-SCRATCH (Si crea un fichero temporal y no debe darse nombre al fichero).

-UNKNOWN (Si no se sabe si existe o no el fichero siendo la opciÃ³n por defecto).

-Un fichero SCRATCH desaparece cuando se cierra la Unidad o cuando termina la ejecuciÃ³n.

-La opciÃ³n IOSTAT, donde N es un identificador de una variable entera. Almacena un cÃ³digo numÃ©rico de cÃ³mo se ejecuta la sentencia OPEN. Si OPEN se ejecuta sin error se almacena un cero en N.

-La opciÃ³n ERR, donde Etiqueta es la sentencia donde se bifurca incondicionalmente si el fichero no puede ser abierto al producirse un error.

-La opciÃ³n BLANK, donde Tipo es una expresiÃ³n de tipo carÃ¡cter que puede tomar los siguientes valores:

-NULL (Los blancos son ignorados y es opciÃ³n por defecto).

-ZERO (Los blancos son interpretados como ceros).

-Esta opciÃ³n se usa para especificar la interpretaciÃ³n de los espacios

en blanco en los datos de entrada.

-La opciÃ³n RECL, donde N es una expresiÃ³n entera, indica la longitud

en caracteres de los registros en un fichero de acceso Directo. SÃ³lo

debe aparecer en los ficheros Directos.

-La opciÃ³n BLOCKSIZE, donde N es un nÃºmero entero, asigna un tamaÃ±o al

buffer de lectura intermedio.

-La opciÃ³n MODE, donde Tipo es una cadena de caracteres, es el modo de

apertura del fichero. La opciÃ³n por defecto es la de lectura escritura

siendo las opciones:

-READ.

-WRITE.

-READWRITE.

-Todas las opciones son opcionales excepto UNIT que es obligatorio, y si

el fichero es de acceso Directo es obligatorio especificar RECL.

ESCRITURA DE UN FICHERO SECUENCIAL.

-Consiste en transferir informaciÃ³n desde la memoria principal al

soporte externo donde estÃ¡ el fichero.

-Para escribir en un fichero Secuencial se usa WRITE especificando la

Unidad a la que estÃ¡ conectada el fichero indicando que el fichero es

Secuencial.

-La sentencia de escritura tiene la siguiente sintaxis:

-WRITE (UNIT= N , FORM=Formato , ERR=Etiqueta , IOSTAT=N ,

ENDFILE=Unidad).

-La Ãºnica opciÃ³n obligatoria serÃ–a UNIT, todas las demÃ¡s han quedado

explicadas.

-Al crear un fichero los registros se escriben uno a continuaciÃ³n del otro, escribiendo un registro especial EOF al final de los datos escritos.

-Normalmente con CLOSE el registro EOF es automÃ¡ticamente escrito al cerrar el fichero, aunque se puede usar la siguiente sentencia:

-ENDFILE N.

-ENDFILE (UNIT= N , ERR=Etiqueta , IOSTAT=N).

-Donde N es el nÃºmero de la Unidad a la que se conectÃ³ el fichero Secuencial. La sintaxis mÃ¡s empleada es la primera. Todas las demÃ¡s opciones ya han quedado explicadas.

-Si se ejecuta ENDFILE, el nombre de la Unidad a la que estÃ¡ conectado un fichero nuevo, sin datos, crearÃ¡ un fichero vacÃ–o.

SENTENCIA DE CIERRE DE UN FICHERO "CLOSE".

-Esta sentencia rompe la conexiÃ³n de un fichero con la Unidad a la que se conectÃ³ un OPEN siendo su sintaxis:

-CLOSE (UNIT= N , STATUS=Tipo , IOSTAT=N , ERR=Etiqueta).

-La opciÃ³n UNIT, siendo N un entero, es el nÃºmero de la Unidad donde se encuentra el fichero que se va a desconectar y puede omitirse, en cuyo caso, serÃ¡ el primer parÃ¡metro de CLOSE.

-La opciÃ³n STATUS, siendo Tipo una expresiÃ³n tipo carÃ¡cter indica el estado del fichero en ese momento, puediendo tener los valores:

-KEEP (El fichero se guarda despuÃ©s de ser cerrado).

-DELETE (El fichero es borrado perdiÃ©ndose despuÃ©s del cierre).

-Por defecto toma el valor KEEP, excepto para los ficheros que son de tipo SCRATCH.

-Los posibles valores que se pueden sacar de las opciones FORM y ACCESS

son los siguientes:

-Registros Secuencial-Formateados.

-Registros Secuencial-No formateados.

-Registros Directo-Formateados.

-Registros Directo-No formateados.

-Registros Binario-Formateado.

-Registros Binario-No formateado.

-Los registros de tipo Secuencial-Formateados se almacenan en cÃ³digo

ASCII y permiten verse con cualquier editor de texto y manipularlos.

-Si tenemos las siguientes instrucciones:

-VarI=4

OPEN (33, FILE='FSEQ')

WRITE (33, '(A, I3)') 'RECORD', I/3

WRITE (33, 'C3')

WRITE (33, '(11H El Reg-N3)')

CLOSE (33)

-La salida que darÃ¡ este ejemplo serÃ¡n tres segmentos de longitud

variable cuya estructura de registros serÃ¡ la siguiente:

-Se puede observar como despuÃ©s de la primera marca de fichero OA, el

segundo registro no aparece por ser de 0 bytes, mientras que el primero

tendrÃ¡ 9 bytes y el tercero tendrÃ¡ 11 bytes.

-Los registros de tipo Secuencial-No formateados tienen longitud

variable y no estÃ¡n en cÃ³digo ASCII, sino otro cÃ³digo que lo introduce

el programa y lo organiza en bloques fÃ–sicos de 128 bytes como mÃ¡ximo.

-Si tenemos las siguientes instrucciones:

-CHARACTER Array (3)

INTEGER *4 Datos (35)

DATA Datos /35*-1/, Array / 'X', 'Y', 'Z'/

OPEN (33, FILE='UFSEQ', FORM='UNFORMATTED')

WRITE (33) Datos

WRITE (33) Array

CLOSE (33)

-La salida que darÃ¡ este ejemplo serÃ¡ la siguiente:

-Como el bloque tiene un tamaÃ±o mÃ¡ximo de 128 bytes por registro, al

sacar todos los elementos del Array Datos (Que necesitarÃ¡ 140 bytes)

se particiona en dos registros uno de 128 bytes y otro de 12 bytes.

-Los registros de tipo Directo-Formateados tienen la misma longitud,

que se deberÃ¡ especificar en la longitud del registro.

-A cada registro siempre se le aÃ±ade el carÃ¡cter de control de carro

(OD) y el carÃ¡cter de avance de lÃ–nea (OA). Si tenemos las siguientes

instrucciones:

-OPEN (33, FILE='FDIR', FORM='FORMATTED', ACCESS='DIRECT',

RECL=10)

WRITE (33, '(A)', REC=1) 'Registro 1'

WRITE (33, '(I5)', REC=3) 30303

CLOSE (33)

-La salida que darÃ¡ este ejemplo serÃ¡ la siguiente:

-Se puede ver que del registro primero se pasa al tercero. En los

ficheros Directos, se reserva espacio para los registros.

-De esa forma si luego viene un registro segundo se insertarÃ–a entre los

registros primero y tercero.

-Los registros de tipo Directo-No formateados es la opciÃ³n por defecto,

en la que la longitud del registro es la misma para todos pero el

compilador no introduce los caracteres OD y OA.

LECTURA DE UN FICHERO SECUENCIAL.

-Transfiere la informaciÃ³n contenida en los registros del fichero a la

memoria del ordenador representada por las variables que aparecen en

la lista de la sentencia de lectura.

-La sentencia READ tiene la sintaxis para ficheros Secuenciales

siguiente:

-READ (UNIT= N, FORM=Formato , END=Etiqueta , ERR=Etiqueta ,

IOSTAT=N , REC=N) Lista de variables.

-La opciÃ³n que siempre debe figurar es el nÃºmero de Unidad perteneciente

al de apertura del fichero Secuencial.

-La opciÃ³n END, donde Etiqueta es un nÃºmero entero indica que al

detectar el carÃ¡cter de fin de fichero transfiera el control de

ejecuciÃ³n del programa a la etiqueta indicada.

-Todas las demÃ¡s opciones ya se han explicado.

-Para leer un fichero la primera opciÃ³n a realizar es abrirlo en la

opciÃ³n STATUS con el valor OLD.

POSICIONAMIENTO EN UN FICHERO SECUENCIAL.

-La lectura y escritura de registros se realiza uno a uno y en serie.

Hay dos sentencias para regresar un registro (BACKSPACE) y para

posicionarse al principio del fichero (REWIND).

-La sintaxis de BACKSPACE es la siguiente:

-BACKSPACE N.

-BACKSPACE (UNIT= N, IOSTAT=N , ERR=Etiqueta).

-N es el nÃºmero de la Unidad a la cual estÃ¡ conectado el fichero. Si no

hay registro anterior porque el fichero se acaba de abrir no tiene

efecto.

-Todas las demÃ¡s opciones ya se han explicado.

-La sentencia REWIND tiene la siguiente sintaxis:

-REWIND N.

-REWIND (UNIT= N, IOSTAT=N , ERR=Etiqueta).

-El efecto es rebobinar el fichero al comienzo, apuntando al primer

registro. Todas las demÃ¡s opciones ya se han explicado.

CREACION DE UN FICHERO DE ACCESO DIRECTO.

-Cada registro de un fichero de acceso Directo es identificado

Ãºnicamente por su posiciÃ³n lÃ³gica en el fichero o nÃºmero de registro

que es un entero de 1 a n.

-El acceso a los registros se hace siempre a partir del nÃºmero de

registro con el que fue creado. En un fichero de acceso Directo no

tiene sentido el registro EOF.

-Por ello ENDFILE no se debe ejecutar sobre estos ficheros, porque no

tiene efecto.

-En un fichero de acceso Directo se reserva espacio de almacenamiento

para todos los posibles registros.

-En los ficheros de acceso Directo los datos de entrada o de salida

deben ir siempre con formato. Este tipo de ficheros es creado al

abrirlo, especificando acceso directo y dÃ¡ndole una longitud.

-Los ficheros no formateados guardan la informaciÃ³n en cÃ³digo binario,

siendo el acceso a los registros mÃ¡s rÃ¡pidos porque se elimina el

tiempo de conversiÃ³n a binario.

-Los registros pueden ser escritos o leÃ–dos en cualquier orden. Siempre

hay que indicar en las sentencias READ y WRITE un parÃ¡metro REC que indicarÃ¡ el nÃºmero de registro a acceder.

-El proceso para leer un registro es similar al de escribir. Hay que especificar el nÃºmero de registro que se quiere leer.

SENTENCIA "INQUIRE".

-Con esta sentencia durante la ejecuciÃ³n de un programa puede obtenerse informaciÃ³n sobre las caracterÃ−sticas de una Unidad o de un fichero.

-Puede ejecutarse antes de que un fichero haya sido abierto, conectado a una Unidad.

-La informaciÃ³n que puede requerirse de un fichero o de una Unidad es muy variada. La sintaxis para esta sentencia es la siguiente:

-INQUIRE (UNIT=N, Lista de especificadores).

-Donde N es un nÃºmero de unidad sobre el que se va a preguntar siendo la lista de especificadores opcionales los siguientes:

-ACCESS=Var*10 ('SEQUENTIAL' o 'DIRECT').

-BLANK=Var*4 ('NULL' o 'ZERO').

-DIRECT=Var*7 ('YES', 'NO' o 'UNKNOWN').

-EXIST=Var ('.TRUE.' o '.FALSE.').

-FORM=Var ('UNFORMATTED' o 'FORMATTED).

-FORMATTED=Var ('YES', 'NO' o 'UNKNOWN').

-NAME=Var*3 ('.TRUE.', '.FALSE.' o 'Nombre').

-NEXTREC=N (Registro despuÃ©s del Ãºltimo accedido).

-NUMBER=N (NÃºmero de Unidad igual a UNIT).

-OPENED=Var ('.TRUE.' o '.FALSE.').

-RECL=N (Longitud del registro).

-SEQUENTIAL=Var ('YES', 'NO' o 'UNKNOWN').

-UNFORMATTED=Var ('YES', 'NO' o 'UNKNOWN').

-IOSTAT=N (CÃ³digo de estado).

-ERR=Etiqueta (Bifurcaciones de error).

-BINARY=Var*10 ('YES', 'NO' o 'UNKNOWN').

-BLOKSIZE=N (TamaÃ±o del buffer).

-Si se produce error en INQUIRE todas las variables que figuran en la sentencia queda un valor indefinido excepto la variable entera de STATUS y las variables de los especificadores pueden ser elementos de un Array.

-Otra posibilidad de INQUIRE es preguntar por el fichero con la siguiente sintaxis:

-INQUIRE (FILE='Nombre', Lista de especificadores).

-La lista de especificadores es la misma que la anterior. Si el nombre del fichero es un identificador vÃ¡lido para el sistema y si el fichero existe se aÃ±adirÃ¡n los siguientes:

-DIRECT, FORMATTED, NAME, SEQUENTIAL y UNFORMATTED.

-Si el fichero estÃ¡ abierto puede aplicarse:

-ACCESS, BLANK, FORM, NEXTREC, NUMBER, BELL, IOSTAT y ERR.

-La ventaja es que provee de informaciÃ³n muy valiosa para abrir ficheros o evitar errores que abortan la ejecuciÃ³n de un programa.

FICHEROS INTERNOS.

-Cuando el programa ejecuta READ o WRITE con un fichero o dispositivo externo se realizan las dos operaciones conjuntamente.

-En los ficheros internos la transferencia de informaciÃ³n se produce entre dos Ã¡reas de memoria interna. Un fichero interno es un Ã¡rea de almacenamiento interno.

-Las entradas o salidas con ficheros internos deben ser siempre

formateadas con los cÃ³digos de formato deseados por el programador.

-Con los ficheros internos hay que especificar siempre los cÃ³digos de

formato en las sentencias WRITE y READ.

-Para acceder a mÃ¡s de un registro en un fichero interno hay que

ejecutar una sÃ³la vez la sentencia READ o WRITE.

-Las opciones que no se permiten usar son las siguientes:

-OPEN, CLOSE, INQUIRE, REWIND, BACKSPACE y ENDFILE.

FICHEROS BINARIOS.

-Es un fichero de tipo Secuencial aunque tambiÃ©n podemos tener ficheros

Directos, pero de esta forma permite recibir o escribir mÃ¡s de un

registro a la vez.

-No se separan los registros y se lee de la misma manera pero en el

Directo hay que poner en la opciÃ³n de formato lo siguiente:

-FORM='BINARY'.

-Si tenemos las siguientes instrucciones:

-INTEGER *1 VarA (4)

CHARACTER VarB (3)

CHARACTER *4 VarC

DATA VarA /4*7/

DATA VarC /'Esto'/, VarB /'A', 'B', 'C'/

OPEN (33, FILE='FBIN', FORM='BINARY')

WRITE (33) VarB, VarC

WRITE (33) 'Que', 'Quieres'

WRITE (33) VarA

CLOSE (33)

-La salida que producirÃ¡ en el registro serÃ¡ la siguiente:

-Notar que la separaciÃ³n es imaginaria. Siempre en un fichero Directo

se han de inicializar las variables normalmente.