2 ESTRUCTURAS DE CONTROL CONDICIONALES

- Hasta ahora, todas las sentencias que forman los programas se ejecutan. Sin embargo, hay ocasiones en que un determinado conjunto de sentencias se deben ejecutar sólo si una determinada condición es cierta y sino no.
- Los valores lógicos: constantes, variables y expresiones lógicas, permiten *controlar* la ejecución de las sentencias de un programa.
- Hay dos tipos de expresiones lógicas: las expresiones lógicas relacionales y las expresiones lógicas combinacionales.

2.1 Expresiones lógicas relacionales

- Las expresiones lógicas relacionales comparan los valores de dos expresiones aritméticas o dos expresiones de tipo carácter.
- La evaluación de una expresión lógica relacional produce un resultado de tipo lógico: .TRUE. o .FALSE.
- La sintaxis de una expresión lógica de tipo relacional es:

operando1 OPERADOR_LÓGICO_RELACIONAL operando2

  - *operando* es una expresión, variable o constante aritmética o de tipo carácter.
  - OPERADOR_LÓGICO_RELACIONAL puede ser:

OPERADOR LÓGICO RELACIONAL		SIGNIFICADO
F77	F90/95	
.EQ.	==	IGUAL
.NE.	/=	DISTINTO
.LT.	<	MENOR QUE
.LE.	<=	MENOR O IGUAL QUE
.GT.	>	MAYOR QUE
.GE.	>=	MAYOR O IGUAL QUE

Tabla 2.1: Operadores lógicos relacionales Fortran

- Los operadores lógicos relacionales de Fortran 77 han sobrevivido y funcionan en los compiladores actuales de Fortran 90/95. Por lo tanto, es interesante que el programador sepa reconocerlos en los programas, sin embargo, es preferible usar la forma de Fortran 90 en sus programas nuevos, que además, es mucho más intuitiva.

2.2 Ejemplos de expresiones lógicas relacionales

- Sea la sentencia de declaración de tipos:

INTEGER :: i=3,j=5

OPERACIÓN	RESULTADO
3<=i	.TRUE.
j**2-1>=0	.TRUE.
i==j	.FALSE.
i/=10	.TRUE.
‘ANA’<’PEPE’	.TRUE.

Tabla 2.2: Ejemplos de expresiones lógicas relacionales

- La última expresión lógica relacional de la Tabla 2.2 es cierta porque los caracteres se evalúan en orden alfabético. Más detalles en el capítulo 6.
- Son inválidas las siguientes expresiones lógicas relacionales:

<=5 Falta operando1

8.44/='XYZ' No se pueden comparar reales con caracteres.

i=3 No es una expresión lógica relacional, sino una sentencia de asignación! No confundir el operador lógico relacional de igualdad con el operador de asignación.

2.3 Expresiones lógicas combinacionales

- La evaluación de una expresión lógica combinacional produce un resultado de tipo lógico: .TRUE. o .FALSE.
- La sintaxis de una expresión lógica de tipo combinacional es:

operando1 OPERADOR_LÓGICO_COMBINACIONAL operando2

  - *operando* es una expresión relacional, variable lógica o constante lógica. *Operando1* no existe cuando el operador lógico usado es unario.
  - OPERADOR_LÓGICO_COMBINACIONAL puede ser:

OPERADOR	TIPO	SIGNIFICADO
.NOT.	UNARIO	ES LO OPUESTO A *OPERANDO2*
.AND.	BINARIO	ES .TRUE. SI Y SÓLO SI *OPERANDO1* Y *OPERANDO2* SON .TRUE.

.OR.	BINARIO	ES .TRUE. SI UNO DE LOS DOS OPERANDOS O LOS DOS ES .TRUE.
.EQV.	BINARIO	ES .TRUE. SI Y SÓLO SI LOS DOS OPERANDOS SON .TRUE. O LOS DOS OPERANDOS SON .FALSE.
.NEQV.	BINARIO	ES .TRUE. SI Y SÓLO SI LOS DOS OPERANDOS TIENEN VALORES DISTINTOS

Tabla 2.3: Operadores lógicos combinacionales Fortran 90/95

Sean *operando1* y *operando2* A y B, respectivamente, la tabla de verdad de los operadores lógicos combinacionales es:

A	B	.NOT.B	A.AND.B	A.OR.B	A.EQV.B	A.NEQV.B
.TRUE.	.TRUE.	.FALSE.	.TRUE.	.TRUE.	.TRUE.	.FALSE.
.TRUE.	.FALSE.	.TRUE.	.FALSE.	.TRUE.	.FALSE.	.TRUE.
.FALSE.	.TRUE.	.FALSE.	.FALSE.	.TRUE.	.FALSE.	.TRUE.
.FALSE.	.FALSE.	.TRUE.	.FALSE.	.FALSE.	.TRUE.	.FALSE.

Tabla 2.4: Tabla de verdad de los operadores lógicos combinacionales

2.4 Precedencias lógicas-aritméticas

- Precedencia de los operadores lógicos combinacionales:

OPERADOR(ES)	PRECEDENCIA
( )	MAYOR
.NOT.	
.AND.	
.OR.	
.EQV. , .NEQV.	MENOR

Tabla 2.5: Orden de precedencia de operadores lógicos combinacionales Fortran

- Si una expresión lógica contiene dos o más operadores de la misma precedencia se siguen las siguientes reglas:
    - Cuando existen paréntesis anidados se evalúan desde el más interno hasta el más externo.

  - Si existen varios operadores .EQV. y/o .NEQV. se evalúan de izquierda a derecha.
- Precedencia de los operadores aritméticos, lógicos relacionales y lógicos combinacionales:

OPERACIÓN	PRIORIDAD
OPERADORES ARITMÉTICOS	
**	1
*, /	2
+ , -	3
OPERADORES LÓGICOS RELACIONALES	
> , >= , < <= , == ,.NE.	4
OPERADORES LÓGICOS COMBINACIONALES	
.NOT.	5
.AND.	6
.OR.	7
.EQV.,.NEQV.	8

Tabla 2.6: Orden de precedencia de operadores Fortran

2.5 Sentencia de asignación lógica

- Asigna un valor lógico (.TRUE. o .FALSE.) a una variable lógica.

variable_lógica = expresión_lógica

  - *variable_lógica* es el nombre de una variable, o elemento de matriz, declarada en el programa del tipo LOGICAL.
  - *expresión_lógica* es una expresión lógica Fortran válida.
- El funcionamiento es:
  - Se evalúa la expresión_lógica.
  - Se asigna el valor obtenido a la *variable_lógica*.
- Ej:

```
LOGICAL :: log1
INTEGER :: i = 10
log1 = (i==10)
```

2.6 Bloque IF

- Permite que un bloque de sentencias (puede ser sólo una) sea ejecutado si y sólo si el valor de una expresión lógica es cierta. Si la expresión lógica es falsa se salta ese bloque de sentencias y se ejecuta la primera sentencia ejecutable por debajo de END IF.

```
IF (expresión lógica) THEN
   bloque de sentencias
END IF
```

- ENDIF marca la terminación de la sentencia bloque IF.
- El *bloque de sentencias* suele dentarse por dos o más espacios para facilitar la lectura del bloque IF, aunque no es obligatorio hacerlo.
- La estructura del bloque IF puede ser más complicada. A veces, se quiere ejecutar un bloque de sentencias si una expresión lógica es cierta y diferentes bloques de sentencias si otras condiciones son ciertas. Por ejemplo:

```
IF (expresión lógica 1) THEN
   bloque de sentencias 1
ELSE IF (expresión lógica 2) THEN
   bloque de sentencias 2]
ELSE
   bloque de sentencias 3
END IF
```

- Si la *expresión lógica* 1 es cierta se ejecuta el *bloque de sentencias* 1 y se salta el resto de bloques hasta la primera sentencia ejecutable por debajo de END IF.
- Si la *expresión lógica 1* es falsa se salta el *bloque de sentencias* 1, se evalúa la *expresión lógica* 2 y si es cierta se ejecuta el *bloque de sentencias* 2 y se salta hasta la primera sentencia ejecutable por debajo de END IF.
- Si ambas expresiones lógicas son falsas, el programa ejecuta el *bloque de sentencias* 3.
- En general, un bloque IF puede contener opcionalmente cualquier número de subbloques ELSE IF (0, 1, 2, 3,...), pero sólo puede contener un subbloque ELSE.
- La sintaxis general de un bloque IF es:

```
IF (expresión lógica 1) THEN
   bloque de sentencias 1
[ELSE IF (expresión lógica 2) THEN
   bloque de sentencias 2]
```

```
...
[ELSE
  bloque de sentencias n]
END IF
```

- La expresión lógica de una cláusula es evaluada sólo si las expresiones lógicas de todas las cláusulas precedentes han sido falsas. Cuando la prueba de una expresión lógica es cierta se ejecuta el bloque de sentencias correspondiente y se salta a la primera sentencia ejecutable por debajo de END IF.
- Cuando el bloque IF no incluye la cláusula ELSE, puede ocurrir que ninguna de las expresiones lógicas sean ciertas y que, por lo tanto, no se ejecute ninguno de los bloques de sentencias dados.

2.7 Bloque IF con nombre

- Es posible asignar un nombre a un bloque IF.
- La sintaxis general de un bloque IF con nombre es:

```
[nombre:] IF (expresión lógica 1) THEN
  bloque de sentencias 1
[ELSE IF (expresión lógica 2) THEN [nombre]
  bloque de sentencias 2]
...
[ELSE [nombre]
  bloque de sentencias n]
END IF [nombre]
```

- *nombre* es cualquier identificador válido.
- Es recomendable usar nombres en los bloque IF largos y complicados. Por un lado, el programador estructura mejor los programas y, por otro, el compilador encuentra errores en su código de forma más precisa.
- Además, los bloques IF pueden estar *anidados*. Dos bloques IF se dice que están anidados cuando uno de ellos se encuentra dentro de otro. En este caso, cada uno de los bloques IF requerirá su propia sentencia ENDIF.

```
[nombre_externo:] IF (expresión lógica 1) THEN
  bloque de sentencias 1
  [nombre_interno:]IF (expresión lógica 2) THEN
  bloque de sentencias 2
  END IF [nombre_interno]
END IF [nombre_externo]
```

- La expresión lógica 2 sólo se evalúa cuando la expresión lógica 1 ha sido cierta.

2.8 Ejemplos de bloques IF

```
IF (num == 0) THEN
  cont0=cont0+1
  ELSE IF (num>0) THEN
  contp=contp+1
ELSE
  contn=cotn+1
END IF


IF (x<0) THEN
  y=SQRT(ABS(x))
  z=x+1-y
ELSE
  y=SQRT(x)
  z=x+1-y
END IF
```

2.9 IF lógico

- La sentencia IF lógica es el caso particular más simple del bloque IF. La sentencia IF lógica evalúa una expresión lógica y ejecuta una sentencia si dicha expresión es cierta.

IF (expresión lógica) sentencia

- *sentencia* es cualquier sentencia ejecutable excepto DO, END, bloque IF u otra sentencia IF lógica.
- Si el valor de la expresión lógica es .TRUE., se ejecuta la sentencia. Si es .FALSE. se salta esa sentencia y se pasa el control a la sentencia siguiente.
- Ej:

```
res=0
IF (a==b) res=a+b
res=res+10
```

2.10 Bloque SELECT CASE

- El bloque SELECT CASE aparece en Fortran 90/95 como otra forma de controlar la ejecución de determinados bloques de sentencias junto con el bloque IF.
- Su sintaxis general es:

```
[nombre:] SELECT CASE (expresión caso)
CASE (selector de caso 1) [nombre]
  bloque de sentencias 1
[CASE (selector de caso 2) [nombre]
  bloque de sentencias 2]
…
[CASE DEFAULT [nombre]
  bloque de sentencias n]
END SELECT [nombre]
```

- El bloque SELECT CASE ejecuta un bloque determinado de sentencias cuando el valor de la *expresión caso* coincide o pertenece al rango dado de su correspondiente *selector de caso*.
- Opcionalmente puede existir un CASE DEFAULT en un bloque SELECT CASE. El bloque de sentencias de este *caso por defecto* se ejecuta cuando el valor de la *expresión caso* no coincide con ningún *selector de caso*.
- *expresión caso* es una expresión entera, lógica o de caracteres.
- *Selector de caso* es una lista de uno o más valores posibles del mismo tipo que la *expresión caso*. Cada selector de caso debe ser mutuamente excluyente. Los valores pueden escribirse como:
  - valor
  - valormin: valormax
  - : valormax
  - valormin:
  - o una combinación de las formas anteriores separadas por comas.
- Es recomendable poner nombre a un bloque SELECT CASE largo y complicado. Por un lado, el programador estructura mejor los programas y, por otro, el compilador encuentra errores en su código de forma más precisa.

2.11 Ejemplos de bloque SELECT CASE

- Determinar si un número entero entre 1 y 10 es par o impar y visualizar un mensaje adecuado en consecuencia.

```
INTEGER :: valor
parimpar: SELECT CASE (valor)
CASE (1, 3, 5, 7, 9)
  WRITE (*,*) 'el valor es impar'
CASE (2, 4, 6, 8, 10)
  WRITE (*,*) 'el valor es par'
CASE (11:)
  WRITE (*,*) 'el valor es demasiado grande'
CASE DEFAULT
  WRITE (*,*) 'el valor es negativo o cero'
END SELECT parimpar
```

- Visualizar un mensaje de acuerdo con el valor de la temperatura dada.

```
INTEGER :: temp
friocalor: SELECT CASE (temp)
CASE (:-1)
  WRITE (*,*) 'por debajo de 0 Celsius'
CASE (0)
  WRITE (*,*) 'Está helando'
CASE (1:10)
  WRITE (*,*) 'Hace frío'
CASE (11:20)
  WRITE (*,*) 'templado'
CASE (21:)
  WRITE (*,*) 'Hace calor'
END SELECT friocalor
```

EJERCICIOS RESUELTOS

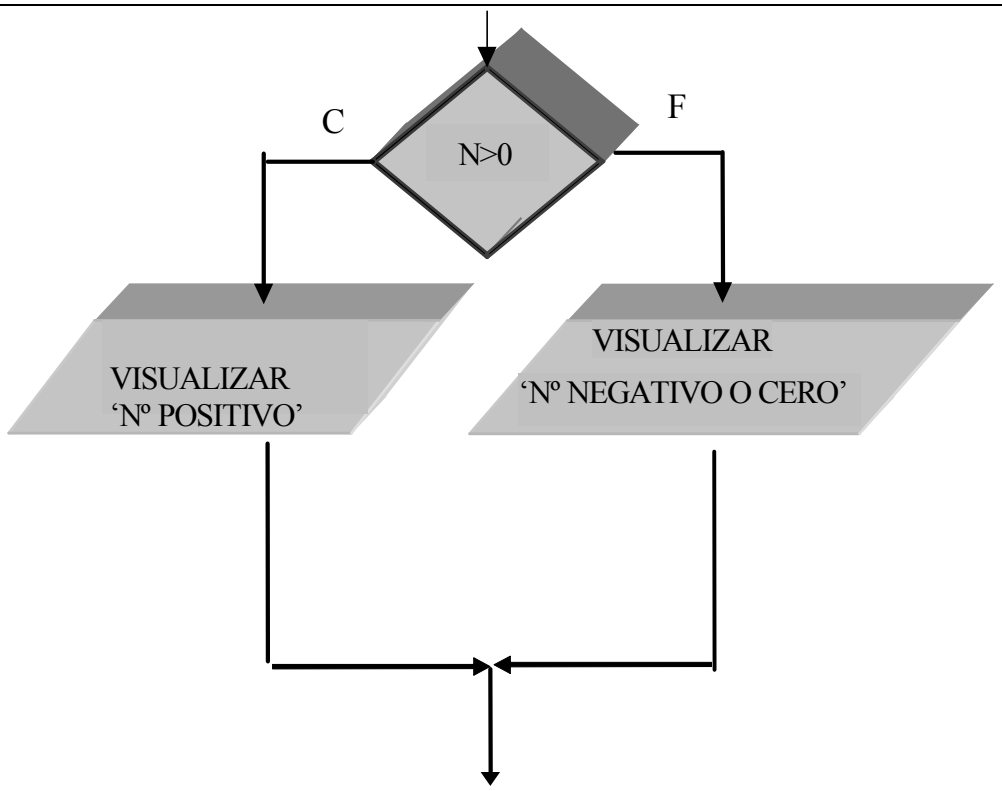
Objetivos:

Aprender a ejecutar unas instrucciones u otras dependiendo de que se cumplan o no una serie de condiciones. Es decir, manejar los bloques IF simples y con anidamiento y los bloques SELECT CASE.

Antes de abordar los primeros programas se muestran los organigramas que representan los algoritmos de los problemas planteados.

Los problemas planteados usan expresiones lógicas en las que intervienen operadores lógicos relacionales y/o combinacionales.

1. Pedir un número real por teclado y escribir si es positivo o no.

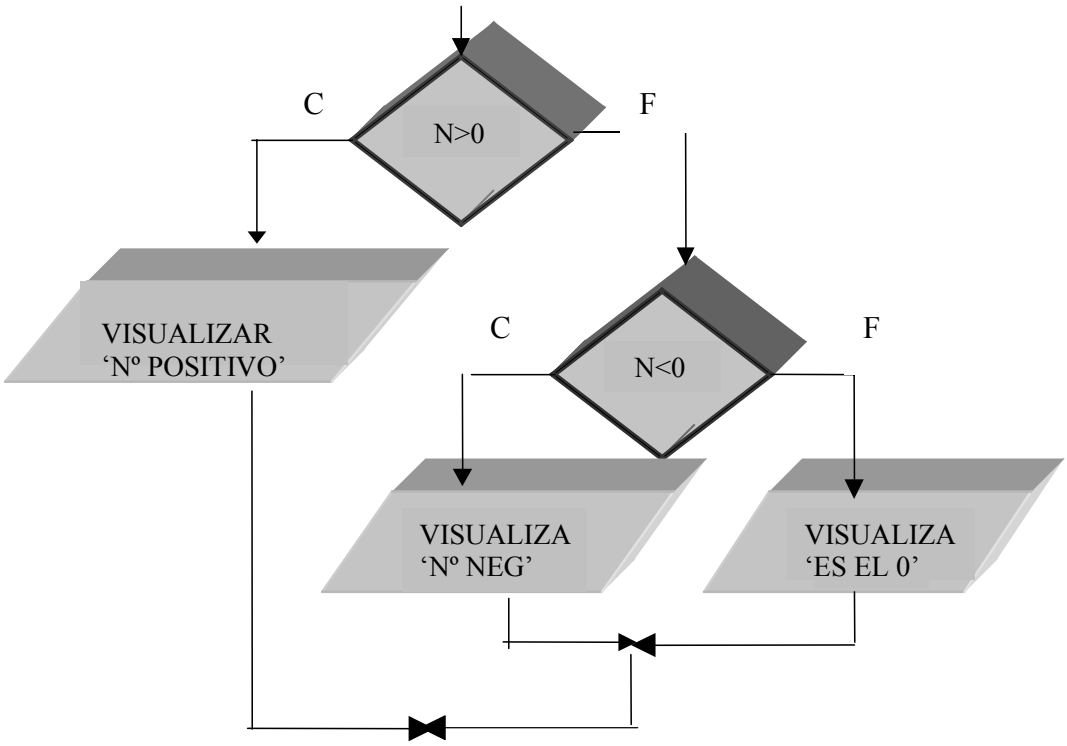



```
PROGRAM cap2_1

REAL :: num
WRITE(*,*)'DAME UN NUMERO'
READ (*,*) num
IF (num > 0) THEN
  WRITE(*,*)'ES POSITIVO'
ELSE
  WRITE(*,*)'ES NEGATIVO O CERO'
END IF
END PROGRAM cap2_1
```

– Este ejercicio requiere un bloque IF simple.

2. Pedir un número real por teclado y escribir si es positivo, negativo o cero.




```
PROGRAM cap2_2

REAL :: num
WRITE (*,*) 'DAME UN NUMERO'
READ (*,*) num
IF (num > 0) THEN
  WRITE (*,*) 'ES POSITIVO'
ELSE IF (num < 0) THEN
   WRITE (*,*) 'ES NEGATIVO'
ELSE
   WRITE (*,*) 'ES EL 0'
END IF
END PROGRAM cap2_2
```

3. Resolver una ecuación de 2º grado. Suponer que es efectivamente de 2º grado (A≠0).

```
PROGRAM cap2_3
IMPLICIT NONE
REAL :: a,b,c,d,r1,r2,pr,pi
WRITE (*,*) 'DAME A, B, C, CON A DISTINTO DE CERO'
READ (*,*) a,b,c
d=b**2-4*a*c
discriminante: IF (d == 0) THEN
  r1=-b/(2*a)
  r2=r1
  WRITE (*,*)'SOLUC. REALES DOBLES, R1=R2=',r1,r2
ELSE IF (d > 0) THEN
  r1=(-b+SQRT(d))/(2*a)
  r2=(-b-SQRT(d))/(2*a)
  WRITE (*,*)'LAS RAICES DEL POLINOMIO SON:'
  WRITE (*,*)'R1=',r1,' R2=',r2
ELSE
  pr=-b/(2*a)
  pi=SQRT(ABS(d))/(2*a)
  WRITE (*,*)'SOLUCIONES COMPLEJAS'
  WRITE (*,*)'PARTE REAL:',pr
  WRITE (*,*)'PARTE IMAGINARIA:',pi
END IF discriminante
END PROGRAM cap2_3
```

– ABS(argumento) es una función intrínseca que devuelve el valor absoluto del argumento escrito entre paréntesis. Su tipo puede ser entero o real.

– Repite el ejercicio cambiando el orden de evaluación de las expresiones lógicas relacionales.

4. Resolver una ecuación de 2º grado con A, B y C cualquier valor.

```
PROGRAM cap2_4
IMPLICIT NONE
REAL :: a,b,c,d,r1,r2,pr,pi,r
```

```
WRITE (*,*) 'INTRODUCE A,B,C, CON A, B, C CUALQUIER VALOR'
READ (*,*) a,b,c
d=b**2-4*a*c

ec2: IF (a /= 0) THEN

 discriminante: IF (d == 0) THEN
  r1=-b/(2*a)
  r2=r1
  WRITE (*,*) 'SOLUC. REALES DOBLES, R1=R2=',r1,r2
 ELSE IF(d > 0) THEN discriminante
    r1=(-b+SQRT(d))/(2*a)
    r2=(-b-SQRT(d))/(2*a)
    WRITE (*,*) 'LAS RAICES DEL POLINOMIO SON:'
    WRITE (*,*) 'R1=',r1,'R2=',r2
 ELSE discriminante
    pr=-b/(2*a)
    pi=SQRT(ABS(d))/(2*a)
    WRITE (*,*) 'SOLUCIONES COMPLEJAS'
    WRITE (*,*) 'PARTE REAL:',pr
    WRITE (*,*) 'PARTE IMAGINARIA:',pi
 END IF discriminante

ELSE IF (b /= 0) THEN ec2
  r=-c/b
  WRITE (*,*) 'SOLUCION UNICA, R=',r
ELSE IF (c /= 0) THEN ec2
  WRITE (*,*) 'ECUACION IMPOSIBLE'
  ELSE
  WRITE (*,*) 'ECUACION INDETERMINADA'
END IF ec2
END PROGRAM cap2_4
```

5. Dada una calificación numérica obtener la correspondiente alfabética según la siguiente clasificación:

0≤ Nota < 5	Suspenso
5≤ Nota < 7	Aprobado

7≤ Nota < 9	Notable
9≤ Nota < 10	Sobresaliente
Nota = 10	Matricula de Honor

```
PROGRAM cap2_5

REAL :: nota
WRITE (*,*) 'DAME UNA NOTA'
READ (*,*) nota
IF (nota < 0 .OR. nota > 10) THEN
  WRITE (*,*) 'NOTA INCORRECTA'
ELSE IF (nota < 5) THEN
  WRITE (*,*) 'SUSPENSO'
ELSE IF (nota < 7) THEN
  WRITE (*,*) 'APROBADO'
ELSE IF (nota < 9) THEN
  WRITE (*,*) 'NOTABLE'
ELSE IF (nota < 10) THEN
  WRITE (*,*) 'SOBRESALIENTEE'
ELSE
  WRITE (*,*) 'MATRICULA DE HONOR'
END IF
END PROGRAM cap2_5
```

6. Realizar una operación entre cuatro posibles según se desee. El programa pide dos números y una operación visualizando el resultado de la misma.

```
PROGRAM cap2_6

REAL :: a,b,c
CHARACTER (LEN=1) :: oper
WRITE (*,*) 'DAME 2 NUMEROS '
READ (*,*)  a, b
WRITE (*,*) 'DAME UNA OPERACION (+, -, *, /)'
READ (*,*) oper
IF (oper == '+') THEN
```

```
  c=a+b
  WRITE (*,*) 'C=',c
ELSE IF (oper == '-') THEN
  c=a-b
  WRITE (*,*) 'C=',c
ELSE IF (oper == '*') THEN
  c=a*b
  WRITE (*,*) 'C=',c
ELSE IF (oper == '/' .AND. b /= 0) THEN
  c=a/b
  WRITE (*,*) 'C=',c
ELSE IF (oper == '/' .AND. b == 0) THEN
  WRITE (*,*) 'NO SE PUEDE DIVIDIR POR CERO'
ELSE
  WRITE (*,*) oper,' NO ES UNA OPERACION VALIDA'
END IF
END PROGRAM cap2_6
```

– En este ejercicio, OPER es el nombre que damos a la variable donde almacenamos el símbolo aritmético de la operación. La última expresión lógica evaluada requiere del operador lógico combinacional Y para evitar que el programa pueda dividir por cero y provoque un error en tiempo de ejecución.

7. Comprobar si un número leído por teclado pertenece al intervalo cerrado [0-10] o no. Escribir un mensaje por pantalla que informe de ello.

```
PROGRAM cap2_7

REAL :: num
WRITE (*,*) 'DAME UN NUMERO'
READ (*,*) num
IF (num >= 0 .AND. num <= 10) THEN
  WRITE (*,*) num,' PERTENECE AL INTERVALO [0-10]'
ELSE
  WRITE (*,*) num,' NO PERTENECE AL INTERVALO [0-10]'
END IF
END PROGRAM cap2_7
```

8. Presentar un menú en pantalla con diferentes opciones.

```
PROGRAM cap2_8

INTEGER :: num
WRITE (*,*) 'DAME UNA OPCION'
WRITE (*,*) '1 PARA EJECUTAR BLOQUE 1'
WRITE (*,*) '2 PARA EJECUTAR BLOQUE 2'
WRITE (*,*) '3 PARA SALIR'
READ (*,*) num
SELECT CASE (num)
CASE (1)
WRITE (*,*) 'SE HACE BLOQUE 1'
CASE (2)
WRITE (*,*) 'SE HACE BLOQUE 2'
CASE (3)
WRITE (*,*) 'ADIOS'
CASE DEFAULT
WRITE (*,*) 'OPCION INCORRECTA'
END SELECT
END PROGRAM cap2_8
```

– Cuando el valor de la variable entera num coincide con algún valor de CASE, el control del programa pasa a ejecutar el bloque de sentencias correspondiente. Si el valor de num no coincide con ningún valor de CASE, se ejecuta el CASE DEFAULT visualizando el mensaje “OPCION INCORRECTA”.

– Repite el ejercicio usando la estructura condicional IF y compara con la estructura SELECT CASE.

EJERCICIOS PROPUESTOS

1) Programa que acepte el número del año, y muestre "PRESENTE" si el número es el año actual, "PASADO" si es menor o "FUTURO" si es mayor.

2) Programa que pida un número natural y muestre si es par o impar.

3) Programa que pida una temperatura y su escala (Celsius/Fahrenheit) y muestre su valor en la otra escala (0 C=32 F y 100 C=212 F). Ecuación de conversión: $C= \frac{5}{9}(F-32)$

4) Programa que acepte el número de un año e indique si es bisiesto o no. Un año es bisiesto si es múltiplo de cuatro, pero no de cien. Excepción a la regla anterior son los múltiplos de cuatrocientos que siempre son bisiestos. Son bisiestos: 1600,1996, 2000, 2004. No son bisiestos: 1700, 1800, 1900, 1998, 2002.

5) Programa que pida la longitud de los lados de un triángulo, compruebe si los datos son correctos, muestre si es equilátero, isósceles o escaleno, y el valor de sus ángulos en grados. A saber: los lados de un triángulo son correctos si cada uno de ellos es menor que la suma de los otros dos. Un triángulo es equilátero si sus tres lados son iguales, isósceles si dos lados son iguales y escaleno si sus 3 lados son distintos. Teorema del coseno $a^2=b^2+c^2-2bc\cos(b,c)$.

6) Programa que pida coordenadas cartesianas de un punto e indique en qué cuadrante se encuentra dicho punto, en qué eje o si se trata del origen de coordenadas.