**Estructuras de Selección (IF)**

Construcción de rutas para la ejecución de un programa (ramas), en donde la única manera de tener acceso al bloque de instrucciones es si la expresión lógica resulta ser TRUE.

A veces es necesario construir más de dos ramas de IFs.

También vamos a usar IFs dentro de otro IF = **IF anidado**.

Ej **if1:** IF (exp. log) THEN

bloque de instrucciones

**if2:** IF (exp. log) THEN

bloque de instrucciones

END IF if2

END IF if1

En ocasiones será necesario decidir de entre un conjunto de opciones

IF ELSE IF ELSE

Las opciones deben de ser mutuamente excluyentes.

IF (exp. log1) THEN

Bloque de instrucciones 1

ELSE IF (exp log2) THEN

Bloque de instrucciones 2

ELSE IF (exp log3) THEN

Bloque de instrucciones 3

ELSE = para instrucciones que no están relacionadas con las expresiones lógicas.

Bloque de instrucciones

END IF

**EXIT y STOP después de IF (exp log) nos ayudan a controlar loops de IFs**

**Programación Modular**

Un programa principal dirige a varios subprogramas, y estos serán “expertos” en realizar alguna operación.

Ventajas

* Reciclado de programas - los subprogramas ya están tan especializados que ya no necesitan modificarse más. Podemos usar un programa ya hecho y convertirlo en subprograma
* Fácil mantenimiento de código

**Function**

Permite construir funciones definidas por el usuario.

Recibe argumentos pero solo regresa una cantidad.

EJEMPLO FUNCTION area\_circulo**(Xo)**

**“Dummy Variables”: variables que se declaran en el programa principal y que se usan en los subprogramas.**

**Argumentos no deben de llamarse igual que en el principal (de preferencia), la unica condicion es que deben de tener el mismo tipo que la variable recibida.**

IMPLICIT NONE

REAL :: Xo

REAL :: area\_circulo

REAL, PARAMETER :: Pi = 3.1159

area\_circulo = Pi\*Xo\*\*2

END FUNCTION area\_circulo

**Subroutine**

Se construye el código necesario para el funcionamiento del programa (operaciones, cálculos, etc)

Recibe argumentos y puede producir varias cantidades como salida

EJEMPLO SUBROUTINE nombre**(arg1, arg2, arg3, arg4)**

IMPLICIT NONE

REAL, INTENT(IN) :: arg1

REAL, INTENT(OUT) :: arg2

REAL, INTENT(INOUT) :: arg3, arg4

**dummy variables que entran y salen de la subroutine**

INTEGER :: indice (variables locales)

**variables usadas en la subroutine**

END SUBROUTINE nombre

(no se repiten los argumentos en END SUBROUTINE)

Para llamar a una subroutine desde otro módulo

CALL nombre(todos los argumentos)

Para compilar

gfortran -c sub.f90 **para ver si el programa tiene errores antes de compilar**

gfortran sub.f90 -o ejec **para cambiar el nombre del ejecutable ./ejec**

Para el ejecutable

gfortran principal.f90 sub.f90 -o ejecutable

**Atributos de Dummy Variables**

* INTENT(IN) - se calculó fuera de la subrutina y la recibe como input sin posible modificación
* INTENT(OUT) - se calculo dentro de la subrutina y se envía como output a otro módulo
* INTENT(INOUT) - calculadas fuera de la subrutina, son recibidas como input con posible modificación y son enviadas como output a otro módulo.

**SELECT CASE**

funciona de manera similar al IF-ELSE IF-ELSE

Le presenta varias opciones a elegir al usuario.

case1: SELECT CASE(opcion)

**Es una variable declarada como INTEGER o CHARACTER que almacena una opción**

CASE (opcion 1)

Bloque de instrucciones

CASE (opcion 2)

Bloque de instrucciones

CASE (opcion 3)

Bloque de instrucciones

**Opciones deben de ser mutuamente excluyentes**

CASE DEFAULT **por si al usuario introduce una opcion invalida**

Instrucciones

END SELECT case1

**Estructuras de Repetición (LOOPS)**

Repeticiones de secciones del código, aproximando el número de soluciones (puede ser una cantidad enorme de repeticiones)

Maneras de construir un LOOP

* DO WHILE: no se sabe la cantidad de veces que se repetirá el código.

DO

IF (exp. log) EXIT **corta el loop cuando la expresión lógica sea V, si es F seguirá hasta que se cumpla**

instrucciones

END DO

* DO ITERATIVO: se conoce de entrada la cantidad de veces que se repetirá el código.