Introducción al Lenguaje de Programación Fortran

Fortran es el lenguaje de programación más viejo. Ayudaba a realizar cálculos numéricos e intentaba resolver problemas.

* 1957 - Fortran 1
* 1960-1965 - Fortran IV

Usado para calcular las trayectorias de proyectiles durantes las guerras

También se contratan mujeres para realizar los cálculos y desarrollando el software

Comandos

* s enlistar contenido

ls -a muestra archivos ocultos

ls -l muestra lista completa de comandos

* d[rwxr -xr -x] permiso de quien puede leer el archivo

d - archivos

rwxr - dueño

-xr - grupo

-x - los demás

* mk dir generar carpetas

mk = make

* cd Fortran

cambiar directorios, desde la carpeta en la que te encuentras a una carpeta pasada o nueva

* man man enseña el manual
* PRINT – muestra el mensaje en la pantalla
* READ – lee el mensaje que el usuario escribió en el programa.

Comunicación

La comunicación humano-máquina es muy importante pero difícil en cierta manera.

Lenguaje de la máquina

1. **Lenguaje de bajo nivel**

* Binario
* Solo se cuenta con operaciones aritméticas básicas
* Ej. lenguaje ensamblador

1. **Lenguaje de alto nivel (Fortran, C, C++, Java, Python)**

* más amigable con el ser humano ya que la comunicación se hace con el idioma inglés.
* frases, sintaxis

Lenguajes de alto nivel

* **Por interpretación**
  + uso de intérpretes
  + cada instrucción es interpretada de forma inmediata
  + Se detectan errores en la instrucción de forma rápida y sencilla
  + los procesos/cálculos pueden ser lentos
* **Por compilacion** 
  + se usa un compilador
  + todas las instrucciones se proporcionan juntas, usando archivos que las contienen
  + se diseñan los algoritmos rápidos para calcular
    - compilador
    - Front-end: verifica las instrucciones
    - Back-end: produce un archivo ejecutable
  + C, C++, FORTRAN, MAPLE

Fortran

compilador: gfortran

compilador del proyecto GNU

Primeros Pasos

* Terminal
  + introducir instrucciones
  + desde ella llamamos al compilador
  + las instrucciones se pueden escribir en
    - gedit
    - emacs

**COMENTARIOS:** ayudan a recordar y simplificar un proceso desarrollado, si no hay comentarios uno batallara en entender/recordar lo que estaba haciendo y puede que tenga que volver a hacerlo.

Toda unidad de Fortran se construye en una unidad de programación

1. Program nombre\_programa

Debe de ser solo una palabra, si son mas no debe de haber espacios. Además, debe de iniciar con una letra

1. instrucciones

En el editor de textos

1. llamar al compilador desde la terminal

gfortran nombre\_archivo

si no aparece un mensaje entonces todo está bien

1. producción del archivo ejecutable

a.out

1. ejecución del programa en terminal

./a.out

1. gfortran archivo -o (nombre que se le quiera poner al ejecutable)

ejecutar el archivo con un nombre más fácil

Tipos intrínsecos

En fortran es necesario trabajar con variables y constantes

* variables: se almacena la memoria para almacenar info. durante la ejecución del programa. Valores introducidos por el usuario
* constantes: se reserva la memoria y se le asigna un valor antes de la ejecución del programa (el valor de puede modificarse después de haber sido declarada).

Al declarar una variable o constante ésta debe de tener

* Puras letras y números
* Puede tener \_
* No debe de tener espacios, (no sin comillas)
* Siempre debe de empezar con letras, no números.
* Deberá ser compuesta mayormente por letras, no números
* No deben de terminar con punto

Para reservar memoria debemos de declarar nuestras variables y constantes.

Nos ayudan a manejar variables y constantes de diferente naturaleza.

* **CHARACTER** – manejo de variables en forma de letras o palabras.

Si la variable está entre comillas entonces es de tipo CHARACTER

CHARACTER(len=10) la cantidad de letras en la variable (contando el espacio)

* **INTEGER** – permite manejar cantidades numéricas sin punto decimal.
* **REAL** – manejo de cantidades numéricas con punto decimal

Problemas!!!

~ **Overflow** – número máximo de valores representables

~ **Underflow** – número mínimo de valores representables

~ **Error de redondeo** – hay un número entre dos variables representadas y la máquina usa el valor anterior o superior.

¡Para evitar esto se incrementan los bits disponibles!

\*32 bits (precisión sencilla)

\*64 bits (precisión doble)

\*128 bits (precisión cuádruple)

\*REAL(8) :: etiqueta

\*REAL(4) :: etiqueta

\*REAL(16) :: etiqueta

* SELECTED\_REAL\_KIND

**Características**

~precisión – cantidad de dígitos significativos de la variable

~rango – intervalo de números que representan a una variable

* **COMPLEX** – manejo de cantidades complejas (reales e imaginarias)

**Funciones Intrínsecas**

**~REAL(C1):** regresa la parte real del complejo C1

PRINT\*, “la parte real de C1 es”, REAL(C1)

**~INT(C1):** regresa la parte real en tipo INTEGER.

PRINT\*, “la parte real de C1 es”, INT(C1)

**~AIMAG(C1):** regresa la parte imaginaria del complejo

PRINT\*, “la parte imaginaria de C1 es”, AIMAG(C1)

**~CABS(C1):** regresa la magnitud C1=

PRINT\*, “la magnitud de C1 es”, CABS(C1)

**~CONJG(C1):** muestra el conjugado del complejo

PRINT\*, “el conjugado es”, CONJG(C1)

**~CMPLX(C1):** almacena dos reales (a1, a2) en el complejo C1.

PRINT\*, “la parte real de C1”

READ\*, C1

PRINT\*, “la parte imaginaria de C1”

READ\*, C1

* **LOGICAL** – variables que solo admiten dos respuestas posibles: verdadero o falso.

~Permiten controlar ciertas secciones del código.

~Expresiones lógicas: expresiones que solo tienen respuestas verdaderas o falsas.

**Sintaxis de IF**

IF (expresión lógica) THEN

secuencia de instrucciones

END IF

Solo en el caso de que la expresión lógica arroje una respuesta verdadera se ejecuta la secuencia de instrucciones.

Ejemplo:

x=4

If (x>5) Then

Print\*, “Hola”

End If

No se mostrará el mensaje porque la expresión es falsa.

Para construir expresiones lógicas contamos con 2 tipos de operadores

1. Operadores de relación: determinan la relación entre dos tipos de cantidades
   1. = = igualdad
   2. > mayor que
   3. < menor que
   4. >= mayor o igual
   5. <= menor o igual
   6. /= diferente a

EL RESULTADO A ESTOS OPERADORES DEBE DE SER VERDADERO O FALSO

1. Operadores de combinación: estas permiten combinar dos o más expresiones lógicas.
   1. .AND. operador y
   2. .OR. operador o
   3. .EQV. equivalencia
   4. .NEQV. no equivalencia
   5. .NOT. negación

Ejemplo:

If (l1 .OR. l2) Then

Print\*, “mensaje”

End If

Para determinar el resultado del uso de un operador de combinación siempre es posible recurrir a una tabla de verdad.

| L1 | L2 | L1 .AND. L2 | L1 .OR. L2 | L1 .EQV. L2 | L1 .NEQV. L2 |
| --- | --- | --- | --- | --- | --- |
| T | T | T | T | T | F |
| T | F | F | T | F | T |
| F | T | F | T | F | T |
| F | F | F | F | T | F |

Se salta el mensaje

*Jerarquía de operadores*

1. Todas las operaciones aritméticas
2. operadores de relación
3. .NOT.
4. .AND.
5. .OR.
6. .EQV. + .NEQV. (de izquierda a derecha)

El uso de paréntesis puede modificar el orden

PARCIAL 2

Estructuras de Selección (IF)

Construcción de varias rutas/ramas para la ejecución de un programa

*Diagrama de flujo de un IF simple*



La única manera de que tengamos acceso al bloque de instrucciones es si el resultado de la expresión lógica es TRUE. Si el resultado es FALSE se ignora el bloque de instrucciones.

Habrá situaciones en donde sea necesario construir más de dos ramas.

IF (exp. log) THEN

bloque de instrucciones 1

END IF

IF (exp. log) THEN

bloque de instrucciones 2

END IF

Es posible encontrarnos con situaciones en las que debamos usar un IF dentro de otro IF (a esto se le llama IFs anidados)

IF (expl log. 1) THEN

bloque de instrucciones 1

IF (exp. log. 2) THEN

bloque de instrucciones 2

END IF

END IF

En ocasiones será necesario decidir de entre un conjunto de opciones

IF ELSE IF ELSE

Las opciones deben de ser mutuamente excluyentes.

EJEMPLO: Aritmética de complejos

PRINT,\* ¿Que operación desea realizar?

PRINT\*, “1. Suma”

PRINT\*, “2. Resta”

PRINT\*, “3. Multiplicación”

PRINT\*, “4. División”

READ\*, opción

Una variable tipo Integer previamente declarada

IF (opcion == 1) THEN

C=C1+C2

ELSE IF (opcion == 2) THEN

C=C1-C2

ELSE IF (opcion == 3) THEN

C=C1\*C2

ELSE IF (opcion == 4) THEN

C=C1/C2

ELSE

PRINT\*, “Opción inválida”

STOP

END IF

* C, C1, C2 son variables tipo complex declaradas anteriormente

Programación Modular



Se ha encargado de realizar todas las instrucciones que se le han dado

* mensajes en pantalla
* cálculos, operaciones etc
* unidad de programación: PROGRAM



La función principal del programa maestro será la de dirigir a varios subprogramas

Cada subprograma será “experto” en realizar una operación.

*Ventajas*

* Reciclado de código: los subprogramas ya están tan especializados que ya no necesitan modificarse más.
* Fácil mantenimiento del código

**FUNCTION**

Nueva unidad de programación.

*Estructura*

FUNCTION nombre(argumentos)

información recibida de un programa maestro

IMPLICIT NONE

sección de declaración de variables

bloque de instrucciones

END FUNCTION

*EJEMPLO*

* funcion.f90

FUNCTION area\_circulo (xo)

“Dummy Variables”

* + Los argumentos no necesitan llamarse igual que en el principal, se recomienda que no tengan el mismo nombre
  + El único requisito es que sean del mismo tipo de la cantidad (variable que reciben)

IMPLICIT NONE

REAL, INTENT(IN) :: xo

REAL :: area\_circulo

REAL, PARAMETER :: Pi = 3.14159

area\_circulo = Pi\*xo\*\*2

END FUNCTION area\_circulo

*Atributos de las Variables Dummy*

* INTENT(IN) - la variable se obtiene de otro subprograma y se transfiere al actual sin posibilidad de modificación
* INTENT(OUT) - la variable se calcula en el subprograma actual y se manda como salida hacia otros
* INTENT(INOUT) - la variable se obtiene en otro subprograma, se transfiere al actual con posibilidad de modificación y enviarlo como salida a otros

*...Más sobre unidades de programación…*

1. PROGRAM - cuando el programa es completamente modular, su función es dirigir a los demás subprogramas
2. FUNCTION - permite construir funciones definidas por el usuario
3. SUBROUTINE - se construye el código necesario para el funcionamiento del programa (operaciones, cálculos, etc)

**DIFERENCIAS**

| **FUNCTION** | **SUBROUTINE** |
| --- | --- |
| REcibe argumentos pero solo regresa una cantidad | Recibe argumentos y puede producir varias cantidades como salida |

Estructuras de Selección: SELECT CASE

Funcionan de manera muy similar al IF-ESLE IF-ELSE. Nos ayuda a elegir entre varias opciones

nombre: SELECT CASE (selector)

variable declarada INTEGER o CHARACTER, se almacena una opción

CASE (op1, op2, op3)

bloque de instrucciones

CASE (op4)

instrucciones

CASE DEFAULT

instrucciones

* Para todo lo demás

END SELECT nombre

* Las opciones son mutuamente excluyentes

Para instalar paquetes en Debian

sudo apt-get install [PAQUETE]