Capítulo 1

Entrada y Salida de datos de un programa

En este capítulo trataremos los parámetros de las sentencias que manejan el sistema de lectura y escritura de datos. Es decir, sus usos al momento del ingreso de datos por teclado, su lectura de una unidad de almacenamiento (disco rígido, SSD, Pen drive, etc) o la adquisición de estos de un dispositivo conectado a la computadora. Las sentencias para el egreso o salida de datos de la computadora tienen sentencias similares. Si bien para ambos casos, en un capítulo anterior habíamos visto sus formas más simples.

1.1. Archivos y sentencia OPEN()

Los archivos (files en inglés) son la unidad de almacenamiento en los discos y demás sistemas que guardan información en forma permanente. Los archivos se localizan en directorios cuya función es la organización de la información. Hay dos tipos de archivos en FORTRAN, los de acceso secuencial y los de acceso directo.

1.1.1. Archivos de acceso secuenciales

Como ya hemos visto en un capítulo anterior, estos son los archivos más comunes en todos los sistemas operativos. De hecho, un programa en Fortran es un archivo secuencial, lo mismo que su ejecutable una vez compilado. Cada Byte de información va uno detrás del otro sin distinción especial de posición o lugar. Son fáciles de manejar y de usar, pero tienen la desventaja que si se requiere información que está, por ejemplo, en el medio del archivo tengo que leerlo hasta ese lugar para encontrar los datos, con la consiguiente pérdida de tiempo. Además si se escribe un valor en mitad de archivo secuencial ya existente se borrarán todos los datos desde ese punto en adelante.

1.1.2. Archivos acceso directo

Estos archivo tienen renglones de tamaño fijo, este tamaño debe definirse al momento de su creación y es necesario conocerlo para poner leer el archivo[1]. En estos archivos se puede ir a leer o escribir datos en un renglón en particular sin tener que leer los anteriores. Además se puede

[1] Algunos sistemas operativos permiten la sentencia INQUIRE que pregunta al sistema operativo el tamaño de renglón de un archivo directo

escribir en cualquier renglón del archivo sin alterar o borrar los números posteriores a ese punto como sucede en los archivos secuenciales.

1.1.3. Sentencia OPEN()

Para leer o escribir en un archivo secuencial o directo se usa la sentencia **OPEN()**, que trabaja en conjunto con la sentencia **CLOSE()** que cierra los archivos. Open en inglés es abrir y en lenguaje de uso común se habla de "abrir" cierto archivo. Pero lo que realmente se está haciendo es indicar que cierto archivo es conectado a una Unidad Lógica especifica en el programa. Por lo tanto, cada vez que se lee o se escribe apuntando a esa Unidad Lógica, se lo está haciendo al archivo en particular que la sentencia OPEN() conectó.

OPEN(UNIT='número', file=*variable de caracteres*, ERR= *entero*, IOSTAT=*variable entera*, STATUS= parámetro, ACCESS= parámetro, RECL= *entero*)

Veamos que significan estos parámetros:

- Unidad Lógica (UNIT) es un número que se asigna al archivo que se va leer o escribir. En decir, este número identifica al archivo en cuestión y es utilizado por las sentencias que operan sobre los archivos. La Unidad Lógica se utiliza, por ejemplo, en el READ() o WRITE() para que puedan leer y escribir desde el archivo que oportunamente se identificó en una sentencia OPEN(). Este hecho adquiere una situación singular en los sistemas operativos UNIX (como el Linux) ya que todos los dispositivos de hardware se pueden ver como archivos de texto (en el directorio */dev*). El software corriendo también se puede ver como un archivo de texto (directorio */proc*) y por lo tanto pueden leerse abriéndolos como archivos con una sentencia OPEN(). Por lo tanto, un programa puede leer configuraciones y datos de la computadora donde está corriendo. Por lo tanto, se pueden acceder a todo un el conjunto de datos de la computadora como por ejemplo: la hora, fecha, la carga de trabajo de la CPU, incluso la temperatura de esta.

  Puede no ponerse la palabra UNIT y se escribe directamente el número (siempre y cuando sea el primero número que aparece luego del paréntesis que abre el OPEN()), de lo contrario es UNIT=número.

- FILE='nombre del archivo', indica el nombre del archivo al que ahora le asigno el número de UNIT. Puede ponerse una variable de caracteres que contenga el nombre del archivo. Pueden también incluirse los directorios con el fin de navegar por la computadora desde el root (/) donde comienza el sistema de directorios (que es una manera absoluta de navegar los directorios), o puede también navegarse en forma relativa a directorio donde se corre el programa, es decir puede poner ../../dir1/dir2[2] (es decir, subo 2 directorios y luego bajo al dir1 y dentro de este al dir2, ya que "../" significa subir un directorio).

- ERR='número', indica el número de etiqueta de sentencia al que se salta si se produce algún error. Si se produce el error y el ERR no se programa (ya que es opcional, no obligatorio) el programa finaliza indicando que hubo error. En cambio, si la orden está, el programa continua en la sentencia a la que se etiquetó a saltar. Esta orden permite programar contingencias en casos de errores. Es muy útil para programas que corren largos períodos de tiempo sin control humano. Como ya indicamos ERR es opcional y puede no estar escrito en la sentencia.

- IOSTAT='variable entera' indica una variable entera a la cual se le carga un número que es una referencia del error que apareció al intentar abrir un archivo. Este número figura en los

[2] Microsoft, creadora del Windows, suele usar el símbolo \ para indicar los directorios

manuales del compilador indicando la naturaleza el error. Si este valor es cero no hubo error alguno. Por ejemplo, no se puede abrir el archivo o el disco está lleno, etc. Es un complemento para funcionar en conjunto con el ERR, para que el propio programa resuelva la situación a través de alguna sección del código, programada para tal efecto.

Por ejemplo, si tengo un error, salto a una sección del programa que identifica el número en el IOSTAT que indica que el disco rígido está lleno y puedo hacer que mi programa borre archivos que ya no son necesarios. Entonces se lo hace volver a la operación OPEN() original (la que dio el error) pero ahora el programa puedo abrir el archivo sin error ya que el problema está resuelto. IOSTAT es opcional.

- STATUS= es un parámetro, este puede ser: OLD, NEW, UNKNOWN o SCRATCH. Este parámetro sirve de control, si se indica OLD significa que el archivo tiene que estar en el directorio, si no se lo encuentra se declara error. En cambio si se indica NEW el archivo no debe estar en el directorio y el OPEN() lo creará en ese momento, si en cambio el archivo ya existía se declarará error. SCRATCH indica que el archivo es un borrador y será borrado cuando el programa finalice, o se ejecute la sentencia CLOSE() que cierre este OPEN(). UNKNOWN es el default, es decir, si no se escribe la orden STATUS, se considerará UNKNOWN. Este indica que da lo mismo que el archivo exista o no. Si no existe será creado cuando se quiera escribir, y si existe será leído o sobrescrito según lo que indique en el código. STATUS es opcional.

- ACCESS se indica un parámetro que es 'SEQUENTIAL' si el archivo es secuencial (que es el default, o sea que no es necesario ponerlo explícitamente), o si es de acceso directo cuyo parámetro es 'DIRECT'. ACCESS es opcional para archivos secuenciales y obligatorio para archivos de acceso directo.

- RECL sólo se puede usar en combinación con ACCESS='DIRECT' e indica el tamaño de renglón para archivos de acceso directo. No se usa en caso de archivo secuencial.

1.1.4. Sentencia CLOSE()

Para indicar que un archivo dejará de usarse, en el uso común del lenguaje "cerrarlo" , se utiliza la sentencia CLOSE() con parámetros similares a las de OPEN().

CLOSE(UNIT='número', STATUS=parámetro, ERR= *entero*, IOSTAT=*variable entera*)

Esta sentencia le da la orden al Sistema Operativo de que cierre la conexión al archivo y que por consiguiente si estoy escribiendo en el disco que guarde los datos (o si ya guardó parcialmente, que termine de hacerlo). Si no está la orden el CLOSE() se hace automáticamente al finalizar la ejecución del programa. En muchos sistemas, si por ejemplo, se corta la electricidad antes de que esta orden se ejecute (el archivo continua abierto) todos los datos ya escritos al disco rígido se perderán. También es importante notar que en los OS modernos muchas veces para evitar que se sature la conexión al disco, los datos pasan por una cache en la memoria RAM donde esperan su turno para ser guardados. Esta sentencia fuerza a que estos datos se salven el disco.
Los parámetros de UNIT, ERR e IOSTAT son iguales a los descriptos para la orden OPEN() y actúan de la misma manera. Mientras que los parámetros de STATUS pueden ser 'KEEP' (conservar) que es el default, y 'DELETE' que borra el archivo al cerrarlo. ERR, IOSTAT y STATUS son opcionales.

1.2. Lectura y escritura en archivos - Sentencias READ() y WRITE()

1.2.1. Lectura y escritura de archivos secuenciales

Para la lectura de datos se utiliza la sentencia READ() (leer en inglés) o WRITE() (escribir en inglés). Estas órdenes en sus versiones más simple, necesitan muy pocos parámetros. Hay que determinar de dónde se lee o escribe, cómo se lo hace y que se lee o escribe. La sentencia en su forma mínima podría escribirse así como ya hemos visto:

READ(*,*) X,Y,Z
WRITE(*,*) X,Y,Z

Pero hay muchos más parámetros que son de utilidad, en su versión más extensa se puede escribir como:

READ(UNIT= *Variable entera*, FMT=*Formato*, ERR= *Variable entera*, END=*Variable entera*, IOSTAT=*Variable entera*, REC=*Variable entera*) Lista de variables

WRITE(UNIT= *Variable entera*, FMT=*Formato*, ERR= *Variable entera*, END=*Variable entera*, IOSTAT=*Variable entera*, REC=*Variable entera*) Lista de variables

Veamos que ordenan todos estos parámetros:

- Unidad Lógica (UNIT) es el lugar asignado para leer o escribir un número. El UNIT es asignado por una sentencia OPEN() a un archivo en particular o por hardware/sofware a un dispositvo Puede ponerse el número sin la palabra UNIT y si hay un '*' se está indicando que se lee del teclado o se escribe a la pantalla del que está corriendo el programa.
- Formato, FMT= es el número de la etiqueta de la sentencia FORMAT que tiene la información de los formatos de esta entrada de datos. La palabra FMT= puede no estar, si este sigue después del Unidad Lógica. Si se pone un '*' significa que se cede la soberanía de la decisión al Fortran, tanto para leer como para escribir.
- END indica el número de etiqueta de sentencia a la que hay que saltar (como un GOTO) en caso de que se llegó al final del archivo que estoy leyendo[3]. END es opcional.
- ERR, al igual que en la sentencia OPEN() este número es una etiqueta de sentencia al que se salta si se produce algún tipo de error. Si esta sentencia no se encuentra el programa finaliza indicando que hubo un error, si la sentencia está, el programa no termina y continua en la sentencia que tiene la etiqueta indicada. ERR es opcional.
- IOSTAT, indica un número que identifica al error. Este se puede ir a buscar ya que está en los manuales del compilador, y obtener una descripción del problema. IOSTAT es opcional.
- REC, es el número de renglón al cual voy a escribir o leer en el caso de archivos de acceso directo y es obligatorio para leste tipo de archivos. No tiene uso en el caso de archivos secuenciales.

La lista de variables se crea nombrando las variables separadas por comas. Puede hacerse lo que se llama DO implícitos, por ejemplo:

[3] El sistema se encuentra con el caracter ASCII EOF (end of file) que sirve para indicar que el archivo llegó a su fin

READ(*,*) (A(i), i=1,100)
o
READ(*,*) ((B(i,j), i=1,100),j=1,100)

Veamos un ejemplo interesante del uso del END para leer un archivo secuencial en el cual no conozco cuantos renglones de datos posee. Esta es una situación muy común en la vida real, sobre todo cuando los datos viene de aparatos de medida automáticos que miden diferente cantidad de datos dependiendo de condiciones externas como la meteorología:

```
      ⋮
      OPEN(23, FILE='datos.dat')
      I=1
10     READ(23,END=99) A(I),B(I)
         I = I + 1
         GOTO 10
99     N= I - 1
      ⋮
```

Este programa lee con un **READ()** (en el cual está indicado el **END** que apunta a la sentencia 99) valores de dos vectores A y B. En cada lectura de los elementos de los vectores, el programa incrementa el valor I y vuelve a leer los próximos datos, ya que la sentencia **GOTO 10** cierra el ciclo. Este ciclo queda trabado leyendo ambos vectores hasta que se acaban los datos. En el fin del archivo este ciclo se rompe, ya que el **READ()** salta a la sentencia 99, tal como lo indica el END.
Para saber cuántos valores se leyeron simplemente se le resta 1 a la variable I y se lo asigna a N. De esta manera, quedan leídos los datos del archivo en las variables A y B, y además en N queda guardado el número de valores leídos del archivo. Este N lo puedo usar en sentencias **DO**s posteriores para procesar ambos vectores.

1.2.2. Archivos con datos binarios

En realidad la versión con menos parámetros posibles de escribir para leer o escribir y que funcione sería así:

READ(UNIT= *número entero)* Lista de variables
WRITE(UNIT= *número entero)* Lista de variables

En esta versión del **READ()** o **WRITE()**, no indico el formato que voy a usar y entonces se leen o se escriben los datos en binario, tal como se encuentran en la memoria RAM de la computadora: Con la ventaja de que ocupan mucho menos espacio, porque un real*4 en binario, ocupa 4 bytes. En cambio si uso un formato y la escribo en base 10 (formato humano) ocuparía 1 byte por dígito. Por otro lado, la escritura o lectura es mucho más rápida porque no hay que traducir de binario a decimal o a la inversa los datos que escribo o leo al disco. Suelen ser muy útiles para transportar grandes cantidades de datos ya que ocupan menos espacio y se pueden leer más rápido.
Como contra partida, debo leer los datos con una computadora similar a la que usé para escribirlos ya que computadoras de distintos tipos pueden no utilizar las mismas normas de definición de datos binarios, es decir misma cantidad de bits en la mantisa y el exponente del número, o distintos orden de los Byte para los números enteros.

1.2.3. Lectura y escritura de archivos de acceso directo

Para el caso de los archivos de acceso directo defino en la sentencia **OPEN()** la variable RECL= número que define el tamaño de renglón, en el caso de los **READ()** o **WRITE()** debo indicar que renglón leo y escribo. Para ello se utiliza la variable REC que indica el número de renglón. Este tipo de archivos se utiliza como base para construir el formato FITS (Flexible Image Transport System) que se utiliza como estándar para transferir y guardar datos astronómicos. Este estándar utiliza renglones de 2880 Bytes de tamaño. Ejemplo:

OPEN(22, FILE='3C299.fits',ACCESS='DIRECT',RECL=2880)
READ(22, REC=128) X,Y

En estas líneas, se leen las variables X,Y del renglón 128 sin tocar ninguno de los renglones anteriores o posteriores del archivo. De la misma manera se podría poner:

WRITE(22, REC=99) X,Y

Donde ahora se escribiría en el renglón 99 sin modificar ninguno de los otros renglones del archivo.

1.2.4. Comandos asociados al manejo de archivos

Hay varios comandos que sirven para manejar situaciones de la lectura de archivos secuenciales, veremos dos de los más útiles:

Sentencia REWIND

Esta orden permite volver a empezar a leer un archivo sin tener que cerrarlo y volverlo a abrir. Es básicamente la orden de rebobinar, que normalmente es virtual, salvo que se le aplique a un grabador/lector de cinta magnética en la cual físicamente se rebobinará la cinta. El comando es:

REWIND(UNIT= *número entero*, ERR= *entero*, IOSTAT=*Variable entera*)

Donde UNIT, ERR y IOSTAT funcionan como ya hemos visto. UNIT, la unidad lógica identifica el archivo a rebobinar. ERR y IOSTAT son opcionales y pueden no estar.

Sentencia BACKSPACE

Esta sentencia permite volver un renglón para atrás para volver a leer variables que ya fueron leídas.

BACKSPACE(UNIT= *número entero*, ERR= *entero*, IOSTAT=*Variable entera*)

Donde UNIT, ERR y IOSTAT cumplen con las mismas funciones, que detallamos anteriormente. ERR y IOSTAT son opcionales.