PRACTICA 0

Introducción a la programación Fortran en Linux.

Hay dos formas de escribir programas sin errores.
Sólo la tercera funciona.

Inicio de sesión. Como alumnos disponen ya de una *cuenta* en las computadoras del aula informatizada de la facultad. Esto es, disponen de un espacio propio en las computadoras para realizar su trabajo. Sin embargo, como todos los usuarios, antes de empezar una *sesión* de trabajo debemos identificarnos ante el sistema. Tal identificación consta de un *nombre de usuario*[1] y una contraseña[2]. Estos datos han sido proporcionados por el *administrador del sistema*[3].

De este modo, para iniciar una sesión, debemos ingresar el nombre de usuario y contraseña en el cuadro de diálogo que se nos presenta en el monitor de la computadora. Nótese que al escribir la contraseña, los caracteres no se muestran en pantalla con el fin de proteger la cuenta de ojos indiscretos. Si los datos son introducidos correctamente, el sistema iniciará la sesión, obteniendo un entorno de trabajo gráfico semejante al de otros sistemas operativos.

> **Mayúsculas o minúsculas.**
>
> Linux diferencia entre las mayúsculas y las minúsculas, lo que afecta a los nombres de usuarios, contraseñas, comandos y nombres de archivos y directorios. Así no es lo mismo `HoY` que `hoy` o incluso `HOY`.

Al iniciar una sesión, el sistema nos deposita en un espacio propio, único para cada usuario, denominado *directorio personal*[4]. Dentro de este directorio personal, y solo dentro del mismo, cada usuario tiene plenos derechos para realizar cambios o modificaciones, los cuales solo afectan a su cuenta y *no* a las de los demás usuarios.

Ejercicio 1. Iniciar una sesión de trabajo con su nombre de usuario y contraseña.

Cierre de sesión. Una vez que hayamos terminado de trabajar, antes de retirarnos *debemos* cerrar la sesión. Para ello utilizamos la opción *Quit* del menú principal, el cual se despliega al presionar con el *mouse* el primer botón localizado en la barra de tarea (ver figura 1).

[1] *Login*, en inglés.
[2] *Password*, en inglés.
[3] También conocido como *root* o superusuario.
[4] *Home directory*, en inglés.

> **Puntos a tener en cuenta.**
>
> ☞ Es importante no olvidarse cerrar la sesión al terminar de trabajar. De lo contrario la cuenta queda abierta y cualquiera podrá acceder a la misma.
>
> ☞ Nunca apagar ni reiniciar las computadoras del aula. Cualquier inconveniente consultar al administrador o los ayudantes del aula.

Ejercicio 2. Cerrar la sesión abierta en el ejercicio anterior (y volver a entrar si queremos seguir continuando con la práctica).

Información de la cátedra en la Web. La cátedra dispone de una página Web accesible en `http://anyp.fcaglp.unlp.edu.ar`. Podemos visitar la misma iniciando el navegador web localizado en la barra de tareas (ver figura 1). La página dispone de varias secciones, en particular una sección que permite descargar los enunciados de la práctica y otra que contiene la bibliografía recomendada para la materia.

Ejercicio 3. Navegar por las secciones de la página de la cátedra.

Interactuando con el sistema en la línea de comandos. Para nuestros propósitos de escribir (y ejecutar) programas que resuelvan problemas científicos, resulta fundamental que sepamos como interactuar con el sistema a través de la *línea de comandos*. Para ello necesitamos de una *terminal* que puede ser iniciada presionando el correspondiente ícono de la barra de tareas (ver figura 1). Al iniciarse la terminal se ejecuta un programa llamado *intérprete de comandos*[5], el cual muestra un indicador de línea (conocido como *prompt*) semejante a lo siguiente

```
$_
```

que nos "invita" a ingresar instrucciones, llamadas *comandos*, para que las ejecute el sistema. En general, un comando tiene la forma

```
$ comando -opciones argumentos
```

Después de escribir el nombre del comando (y posibles opciones y argumentos), éste es ejecutado apretando la tecla `Enter`. Por supuesto, para ello tenemos que conocer que comandos existen y para que sirven.

Ahora bien, la información que nos interesa guardar en nuestras cuentas estará organizada en *archivos* y *directorios*. Un *archivo* es un conjunto de información al que se le ha dado un nombre. Esta información

[5] *Shell*, en inglés

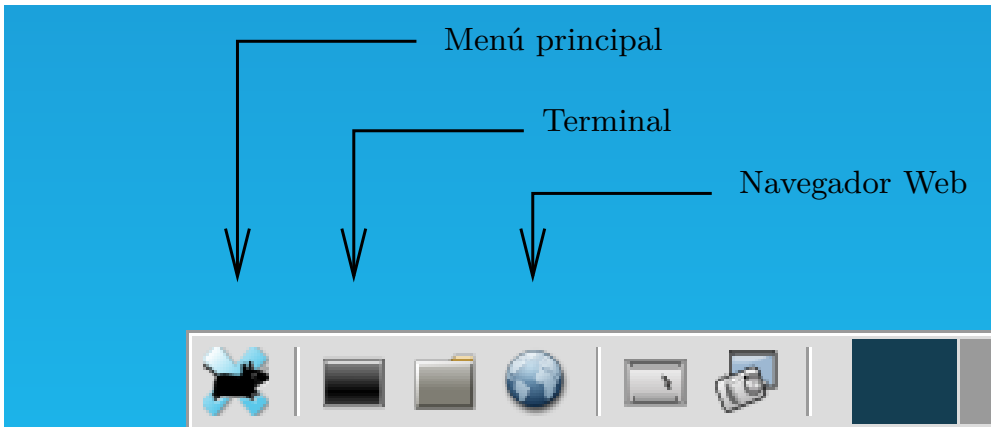

Figure 1. Elementos en la barra de tarea.

puede ser de distintos tipos, como ser archivos de texto, archivos gráficos, programas ejecutables, etc. Por su parte, los archivos se guardan en *directorios*, los cuales además pueden contener no sólo archivos, sino otros directorios. Los comandos que describiremos a continuación están, pues, orientados a tratar con archivos y directorios. Por supuesto, existen muchos más comandos de los que vamos a ver aquí, pero éstos nos alcanzarán para comenzar a trabajar.

> **Estructura de árbol.**
>
> La disposición jerárquica de archivos y directorios suele representarse en forma de árbol. Para un usuario, el directorio principal constituye su raíz y de él cuelgan todos los demás directorios y archivos que conforman dicho árbol, en el que los directorios serían las ramas y los archivos las hojas.

Dónde estamos. El comando `pwd` informa el directorio en el que nos encontramos. Como al ingresar al sistema nos encontramos en el directorio personal, si lo ejecutamos en ese instante podremos saber cuál es tal directorio.

Ejercicio 4. Ejecutar el comando `pwd`. ¿Cuál es su directorio personal?

Qué hay en un directorio. El comando `ls` lista los archivos y subdirectorios que hay en un directorio. Si se ejecuta sin argumentos, el comando genera un listado del contenido del directorio actual de trabajo.

Ejercicio 5. Listar los contenidos del directorio principal. (*Nota*: puesto que las cuentas están recién creadas contendrán pocos archivos).

Ejercicio 6. Descargar en su directorio personal el archivo correspondiente a la PRÁCTICA 0 desde la página Web de la cátedra. Controlar que el archivo ha sido descargado ejecutando el comando `ls`.

Crear y eliminar directorios. El comando `mkdir` nos permite crear directorios, mientras que el comando `rmdir` los borra (siempre y cuando estén vacíos). Ciertamente, cuando la cuenta comienza a ser utilizadas es mejor tenerla organizada. Para ello crearemos directorios y subdirectorios donde alojar las prácticas y nuestro trabajo.

Ejercicio 7. Crear el directorio `anyp` bajo el directorio principal. Controlar con el comando `ls` que tal directorio fue efectivamente creado.

Cambio de directorio. El comando `cd` nos permite movernos de un directorio a otro. Así que si queremos ir del directorio principal al directorio `anyp` creado en el ejercicio anterior, ejecutamos el comando como sigue:

```
$ cd anyp
```

Ejercicio 8. Crear el subdirectorio `practica-00` dentro del directorio `anyp`. Moverse a dicho directorio y comprobar con el comando `pwd` que nos encontramos en tal directorio.

El ejercicio anterior genera, como es mencionado más arriba, para el conjunto de directorios creados, una estructura de árbol cuya jerarquía se ilustra en la figura 2. El comando `cd` nos permite navegar por este árbol como sigue. Si escribimos únicamente el comando `cd` y presionamos `Enter`, nos moveremos inmediatamenta a nuestro directorio principal independientemente de donde hayamos estado (lo cual podemos comprobar fácilmente ejecutando inmediatamente el comando `pwd`):

```
$ cd
$ pwd
```

Para movernos desde el directorio principal al directorio `practica-00` que se encuentra bajo el directorio `anyp` ejecutamos, en el ejercicio anterior, la secuencia de comandos

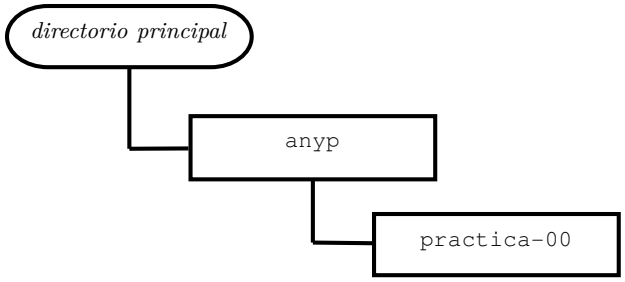

Figure 2. Estructura jerárquica de directorios

```
$ cd anyp
$ cd practica-00
```

Sin embargo, podemos proceder de otro modo. A saber, ejecutando, desde el directorio principal, el comando:

```
$ cd anyp/practica-00
```

Ejercicio 9. Comprobar que los dos procedimientos anteriores conducen a posicionarnos en el mismo directorio.

Si, ahora, queremos ir al directorio inmediatamente superior que contiene al directorio de trabajo actual, podemos ejecutar el comando

```
$ cd ..
```

Ejercicio 10. ¿A que directorio se llega si ejecutamos nuevamente el comando `cd ..`?

> ✏ **¿Qué significan ., .. y /?**
>
> El carácter . representa al directorio actual, mientras que el carácter .. representa el directorio inmediatamente superior o directorio *padre*. Por otra parte, el carácter / es el separador de directorios.

Copiar, mover y borrar archivos. Los comandos `cp`, `mv` y `rm` nos permiten trabajar con archivos y estructuras de directorios ya que se pueden aplicar tanto a archivos, directorios o ramas de la estructura. Pero, por el momento, a nosotros nos interesa llevar el archivo `anyp-00.pdf` (correspondiente al enunciado de esta práctica) al subdirectorio `anyp/practica-00` del directorio principal. Podemos hacer ésto copiando el archivo a dicho subdirectorio con el comando `cp` como sigue:

```
$ cp anyp-00.pdf anyp/practica-00
```

Ejercicio 11. Comprobar que el archivo fue copiado efectivamente en dicho directorio.

Ahora tenemos dos archivos, uno bajo el directorio principal y otro en un subdirectorio. Podemos borrar el archivo que se encuentra en el directorio principal ejecutando el comando `rm` como sigue:

```
$ rm anyp-00.pdf
```

Ejercicio 12. Verificar que el archivo fue efectivamente eliminado.

De manera alternativa, para llevar nuestro archivo al subdirectorio correspondiente, podríamos haber utilizado el comando `mv` para mover el mismo:

```
$ mv anyp-00.pdf anyp/practica-00
```

> ✏ **Potencia de los comandos.**
>
> En Linux, los comandos son *obedientes* y *silenciosos*, esto es, realizan su tarea tal como se escribe en la línea de comandos y, si es exitosa, no informan el resultado. Debido a ésto un comando como `rm` es, a la vez, muy potente y peligroso. ¡Archivos borrados son irrecuperables!

Todos los comandos de Linux disponen de documentación en línea, conocida como *páginas de manual*. Para visualizar la página de manual de un comando utilizamos el comando `man` seguido por el nombre del comando del que deseamos obtener ayuda. Para navegar en la página, moviéndonos hacia adelante y hacia atrás de la misma, utilizamos la `barra espaciadora` y la letra `b`, respectivamente, o bien las teclas del cursor. Para salir de la ayuda, simplemente apretamos la letra `q`.

Ejercicio 13. Consulte la página de manual del comando `man`.

Visualizando e imprimiendo los enunciados de la prácticas. Los enunciados de la prácticas de la cátedra están disponibles electrónicamente en la página de la cátedra en un formato conocido como PDF[6]. Este formato electrónico resulta adecuado para la distribución de documentación que va a ser visualizada en la pantalla de la computadora y/o ser impresa, como es nuestro caso. En Linux, podemos visualizar por pantalla los archivos PDF con el programa `xpdf`.

Asumiendo que estamos en el directorio que contiene al archivo PDF correspondiente a la PRÁCTICA 0, la ejecución del siguiente comando en una terminal despliega una nueva ventana mostrando el contenido del mismo:

[6] Abreviatura de *portable document format*, en inglés.

```
$ xpdf anyp-00.pdf &
```

> **Por qué hay un & en la línea de comandos.**
>
> El carácter & hace que un programa se ejecute en segundo plano, permitiendo que la línea de comandos continúe esperando nuevas órdenes. Si este carácter no se introduce, entonces la terminal quedará bloqueada hasta que cerremos el programa.

El programa `xpdf` permite también imprimir el archivo PDF.

Ejercicio 14. Localizar la opción de impresión en el menú del `xpdf` e imprimir la práctica.

Primeros pasos en programación. La resolución de un problema científico con una computadora, tarde o temprano, conduce a la escritura de un *programa* que implemente la solución del problema. Un programa es un conjunto de instrucciones, ejecutables sobre una computadora, que permite cumplir una función específica. Ahora bien, ¡la creación del programa no comienza directamente en la computadora! El proceso comienza en papel diseñando un *algoritmo* para resolver el problema. Un algoritmo es un conjunto de pasos (o instrucciones) *precisos*, *definidos* y *finitos* que a partir de ciertos datos conducen al resultado del problema.

> **Características de un algoritmo.**
>
> - *preciso*: el orden de realización de cada paso está especificado,
> - *definido*: cada paso está especificado sin ambigüedad,
> - *finito*: el resultado se obtiene en un número finito de pasos.
> - *entrada/salida*: dispone de cero o más datos de entrada y devuelve uno o más resultados.

Para describir un algoritmo utilizaremos un *pseudocódigo*. Un pseudocódigo es un lenguaje de especificación de algoritmos donde las instrucciones a seguir se especifican de forma similar a como las describiríamos con nuestras palabras.

Consideremos, como ejemplo, el diseño de un algoritmo para calcular el área de un círculo. Nuestro primer intento, bruto pero honesto, es:

```
Calcular el área de un círculo.
```

Sin embargo, este procedimiento no es un algoritmo, por cuanto no se especifica, como dato de entrada, cuál es el círculo a considerar ni tampoco cual es el resultado. Un mejor procedimiento sería:

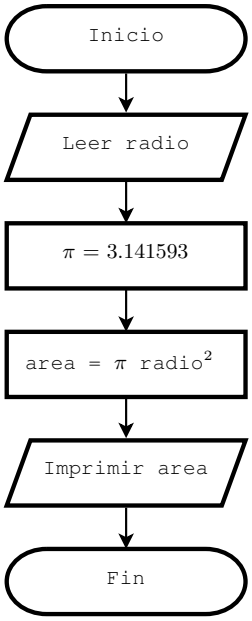

Figure 3. Diagrama de flujo para el algoritmo que calcula el área de un círculo.

```
Leer el radio del círculo.
Calcular el área del círculo.
Imprimir el área.
```

Sin embargo, éste procedimiento aún no es un algoritmo por cuanto la segunda instrucción no especifica cómo se calcula el área de un círculo de radio dado. Explicitando la fórmula matemática tenemos finalmente un algoritmo apropiado:

```
Leer el radio del círculo.
Tomar π = 3.141593.
Calcular área = π × radio².
Imprimir el área.
```

Una manera complementaria de describir un algoritmo es realizar una representación gráfica del mismo conocida como *diagrama de flujo*. El correspondiente diagrama de flujo para el algoritmo anterior se ilustra en la figura 3.

Una vez que disponemos del algoritmo apropiado, su implementación en la computadora requiere de un *lenguaje de programación*. Un lenguaje de programación es un lenguaje utilizado para escribir programas de computadora. Como todo lenguaje, cada lenguaje de programación tiene una sintaxis y gramática particular que debemos aprender para poder utilizarlo. Por otra parte, si bien existen muchos lenguajes de programación para escoger, un lenguaje adecuado para problemas científicos es el denominado lenguaje Fortran[7]. El proceso de implementar un algoritmo en un lenguaje de programación es llamado *codificación* y su resultado *código* o *programa fuente*. Siguiendo con nuestro ejemplo, la codi-

[7] El nombre es un acrónimo en inglés de *formula translation*.

ficación del algoritmo en el lenguaje Fortran conduce al siguiente programa:

Código 1. Cálculo del área de un círculo

```
PROGRAM calcular_area
! Cálculo del área de un círculo
! de radio dado

! Declaración de variables
IMPLICIT NONE
REAL :: radio  ! radio del círculo
REAL :: area   ! área del  círculo
REAL, PARAMETER :: PI = 3.141593

! Entrada de datos
WRITE(*,*) 'Ingrese radio del círculo'
READ(*,*) radio

! Calcular area
area = PI*radio**2

! Imprimir resultado
WRITE(*,*) 'Area = ', area

END PROGRAM calcular_area
```

Aún cuando no conozcamos todavía la sintaxis y gramática del Fortran, podemos reconocer en el código anterior ciertas características básicas que se corresponden con el algoritmo original. En particular, notemos que los nombres de las variables involucradas son similares a la nomenclatura utilizada en nuestro algoritmo, que los datos de entrada y salida estan presentes (junto a un mecanismo para introducirlos y mostrarlos) y que el cálculo del área se expresa en una notación similar (aunque no exactamente igual) a la notación matemática usual. Además hemos puesto una cantidad de comentarios que permiten comprender lo que el código realiza.

Un algoritmo es independiente del lenguaje de programación.

Si bien estamos utilizando Fortran como lenguaje de programación debemos enfatizar que un algoritmo es independiente del lenguaje de programación que se utiliza para su codificación. De este modo un mismo algoritmo puede ser implementado en diversos lenguajes.

Disponemos ya de un código para nuestro problema. Ahora bien, ¿cómo lo llevamos a la computadora para obtener un programa que se pueda ejecutar? Este proceso involucra dos etapas: la *edición* y la *compilación*. Comencemos, pues, con la edición. Nuestro código fuente debe ser almacenado en un archivo de *texto plano* en la computadora. Para ello debemos utilizar un *editor de texto*, el cual es un programa que permite ingresar texto por el teclado para luego almacenarlo en un archivo. El archivo resultante se conoce como *archivo fuente* y para un código escrito en Fortran puede tener el nombre que querramos pero debe terminar con la *extensión* `.f90`. Si bien en Linux existen varios editores de texto disponibles, nosotros utilizaremos el editor `emacs`. Así para ingresar el código en un archivo que llamaremos `area.f90` ejecutamos en la línea de comandos de la terminal el comando:

```
$ emacs area.f90 &
```

El editor `emacs` es un editor de texto de propósito general pero que adapta sus posibilidades al contenido del archivo que queremos guardar. En particular permite que la programación en Fortran resulte muy cómoda al resaltar con distintos colores las diversas instrucciones que posee el lenguaje y facilitar, además, la generación de código sangrado apropiadamente para mayor legibilidad.

Ejercicio 15. Utilizar el editor `emacs` para almacenar el código de nuestro ejemplo en el archivo fuente `area.f90` bajo el subdirectorio `anyp/practica-00` del directorio principal.

Nuestro programa fuente, almacenado ahora en el archivo fuente `area.f90`, *no es todavía un programa que la computadora pueda entender directamente*. Esto se debe a que Fortran es uno más de los denominados *lenguajes de alto nivel*, los cuales están diseñados para que los programadores (es decir, nosotros) puedan escribir instrucciones con palabras similares a los lenguajes humanos (en general, como vemos en nuestro código, en idioma inglés)[8]. En contraste, una computadora no puede entender directamente tales lenguajes, pues las computadoras solo entienden *lenguajes de máquina* donde las instrucciones se expresan en términos de los dígitos binarios 0 y 1. De este modo, nuestro programa fuente, escrito en un lenguaje de alto nivel, debe ser traducido a instrucciones de bajo nivel para que la computadora pueda ejecutarlo. Esto se realiza con ayuda de un programa especial conocido como *compilador*. Así pues, el compilador toma el código fuente del programa y origina un *programa ejecutable* que la computadora puede entender directamente. Este proceso es denominado *compilación*. En Linux el compilador de Fortran es llamado `gfortran`. Así, para compilar el archivo fuente `area.f90` y generar un programa ejecutable, que llamaremos `area`, escribimos en la línea de comandos:

```
$ gfortran -Wall -o area area.f90
```

Debería ser claro que la opción `-o` permite dar el nombre para el programa ejecutable resultante de la compilación. Por otra parte la opción `-Wall` le dice

[8] De hecho, Fortran es el abuelo de todos los lenguajes de alto nivel, pues fue el primero ellos.

al compilador que nos advierta de posibles errores (no fatales) durante el proceso de compilación[9].

Ejercicio 16. Compile el programa fuente. Verifique que se genera, efectivamente, el programa ejecutable.

Si no se producen errores, habremos creado nuestro primer programa. El programa se puede ejecutar desde la línea de comandos con sólo teclar su nombre:

```
$ ./area
```

> ✏ **¿Qué significa ./?**
>
> Puesto que el carácter . representa al directorio actual y / al separador de directorios, ./ es el directorio actual de trabajo.

Ejercicio 17. Verificar que el programa da resultados correctos. En particular verificar que para un radio igual a la unidad el área que se obtiene es π y que el área se cuadruplica cuando el radio es igual a 2.

Ejercicio 18. Modificar el programa para que calcule el área $A = \pi ab$ de una elipse de semiejes a y b.

Cuando las cosas fallan. Tres tipos de errores se pueden presentar: *errores de compilación*, *errores de ejecución* y *errores lógicos*. Los errores de compilación se producen normalmente por un uso incorrecto de las reglas del lenguaje de programación (típicamente *errores de sintaxis*). Debido a ellos el compilador no puede generar el programa ejecutable. Por otra parte, los errores de ejecución se producen por instrucciones que la computadora puede comprender pero no ejecutar. En tal caso se detiene abruptamente la ejecución del programa y se imprime un mensaje de error. Finalmente, los errores lógicos se deben a un error en la lógica del programa. Debido a estos errores, aún cuando el compilador nos da un programa ejecutable, el programa dará resultados incorrectos. Todos estos tipos de errores obligan a revisar el código, originando un ciclo de desarrollo del programa que consta de compilación, revisión y nueva compilación hasta que resulten subsanados todos los errores.

Ejercicio 19. Procediendo con la compilación y ejecución, identificar que tipo de errores se producen en las siguientes situaciones con nuestro código.

a) La sentencia
`WRITE(*,*) 'Ingrese el radio del círculo'`
del código es cambiada por
`WROTE(*,*) 'Ingrese radio del círculo'`.

b) Durante la ejecución se ingresa una letra cuando se pide el radio del círculo.

c) El valor de la constante `PI` es asignada a 2.7182818 en el código.

[9] *Wall* significa, en inglés, *warnings all*.