Manual de prácticas del laboratorio de Programación Básica

Elaborado por:	Revisado por:	Autorizado por:	Vigente desde:
Ing. Jorge Luis López García **Ing. Mayelly Reynoso Andrade**	**M. C. Laura Sandoval Montaño**	**Dra. Rocío Alejandra Aldeco Pérez** **Ing. Luis Sergio Valencia Castro**	**10-agosto-23**

Índice de prácticas

No	Nombre	Página
1	Utilerías de software y en Internet	3
2	Análisis de algoritmos	17
3	Diseño de algoritmos con instrucciones en secuencia	26
4	Diseño de algoritmos con instrucciones de selección y repetición	42
5	Diseño de algoritmos avanzados	58
6	Entornos y fundamentos de programación en lenguaje FORTRAN	80
7	Programas estructurados con instrucciones de selección	92
8	Programas estructurados con instrucciones de repetición	100
9	Arreglos unidimensionales numéricos y cadenas	109
10	Arreglos bidimensionales parte 1	117
11	Arreglos bidimensionales parte 2	126
12	Funciones y subrutinas	132

Guía práctica de estudio 01: Utilerías de software y en Internet

Elaborado por:	Revisado por:	Autorizado por:
Ing. Jorge Luis López García **Ing. Mayelly Reynoso Andrade**	**M. C. Laura Sandoval Montaño**	**Dra. Rocío Alejandra Aldeco Pérez** **Ing. Luis Sergio Valencia Castro**

Guía práctica de estudio 01: Utilerías de software y en Internet

Objetivo:

El alumnado conocerá software, servicios y herramientas libres que se pueden utilizar de manera local y en Internet para el manejo de trabajos durante su desarrollo académico.

Actividades:

1. Revisar entorno del sistema operativo en el laboratorio asignado (Windows, Fedora o MacOS).

2. Conocer la terminal del sistema operativo.

3. Identificar las herramientas de software instaladas y servicios en Internet a usar durante el semestre como editores de texto plano tanto en terminal (vi) como gráficos, herramientas de diseño de algoritmos, IDE de programación, editores de documentos de texto en línea y almacenamiento en la nube.

Introducción:

Sistemas Operativos

Un sistema operativo es tal vez la parte más importante del software del sistema y es el que controla y gestiona los recursos de la computadora. En la práctica es la colección de programas de computadora que controla la interacción del usuario y el hardware de la computadora. Es el administrador principal de la computadora, este software es el responsable de dirigir todas las operaciones de la computadora y gestionar todos sus recursos.

El sistema operativo se encarga de realizar importantes y diferentes tareas como transmitir información entre los programas de aplicación, controlar el funcionamiento de los dispositivos periféricos (impresoras, teclados, etc.), evitar problemas de seguridad en ciertos programas, entre otros.

Clasificación de los sistemas operativos

Los sistemas operativos se clasifican de la siguiente manera:

- Monotarea: solo puede ejecutar una tarea o programa a la vez. Se trata de los sistemas operativos más antiguos.
- Monousuario: es el sistema operativo que solo puede responder a un usuario a la vez.
- Multitarea: son aquellos que permiten que varios programas se ejecuten en el mismo momento en uno o más ordenadores.
- Multiprocesador: hace posible que un mismo programa sea ejecutado en más de un ordenador o distribuir los programas en ejecución en varios procesadores.
- Multiusuario: permite que dos o más usuarios puedan acceder a los servicios y procesamientos de un sistema operativo al mismo tiempo.
- Tiempo Real: son aquellos que responde a un estímulo externo dentro de un tiempo especificado.

Ejemplos de algunos de ellos:

Sistema operativo Windows.

Figura 1*. Escritorio de Windows 11.*

Sistema operativo Linux.

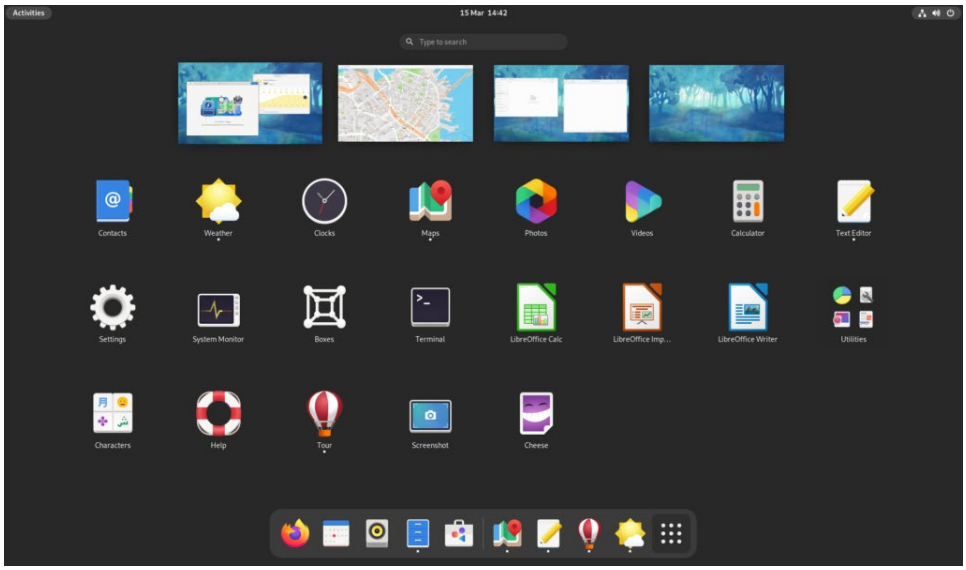


Figura 2. *Escritorio de Fedora 34.*

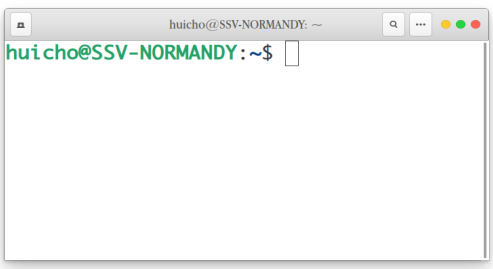
Figura 3. *Escritorio de Ubuntu 22.04.*

Sistema operativo MacOs.

Figura 4. *Escritorio de macOS Ventura 13.*

Terminal o consola

La terminal o consola es una forma extensiva de llamar a la interfaz de usuario de línea de comandos.



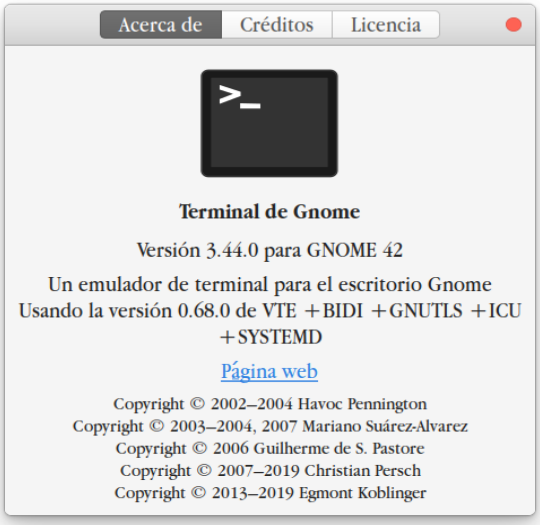


Figura 5. *Interfaz gráfica y ventana "Acerca de" para Terminal de Gnome.*

Editores de texto plano

Vi

Editor de texto plano multipropósito que no está basado en ventanas, por tanto su uso se limita a terminales de sistemas operativos basados en UNIX. Contiene una amplia lista de comandos lo que puede ser complicado para usuarios primerizos pero es muy útil cuando el sistema no cuenta con una interfaz gráfica y se desea editar archivos simples o de configuración.

Figura 6. *Interfaz de edición de vi.*

Gedit

Editor de texto plano de propósito general para documentos simples pero también enfocado a desarrollo de software.

Soporta codificación UTF-8 y resaltado de sintaxis de lenguajes como Python, Shell, C, C++, HTML, CSS, JavaScript, entre otros.

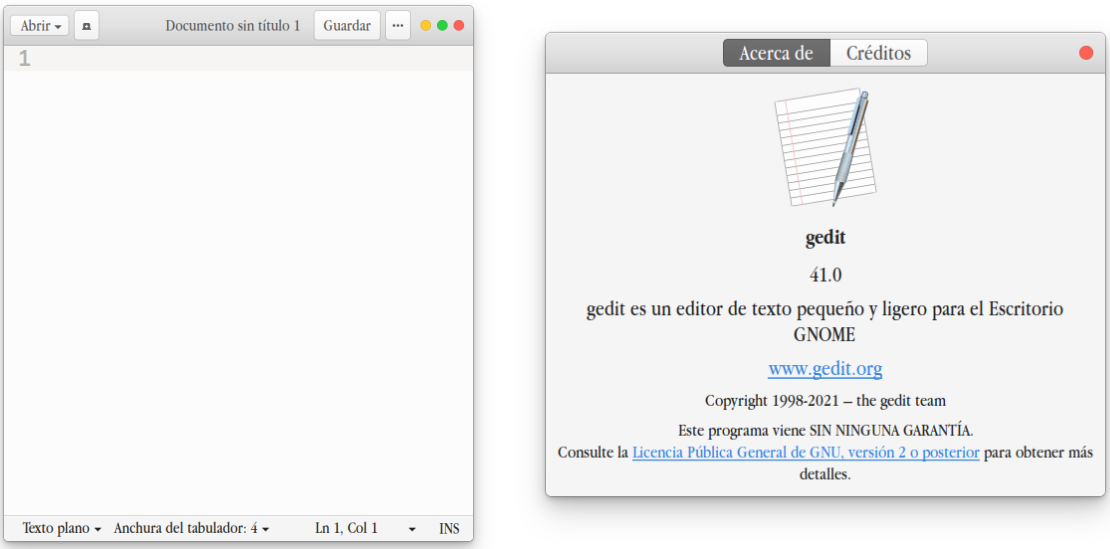


Figura 7. *Interfaz gráfica y ventana "Acerca de" para gedit.*

Herramientas de diseño de algoritmos

Dia

Es un programa de código abierto para dibujar diagramas y mapas conceptuales. Entre sus opciones puede generar diagramas de flujo, UML, entidad relación, entre otros.

Es compatible con los sistemas Windows, Linux y macOS. Además permite exportar a distintos formatos gráficos.

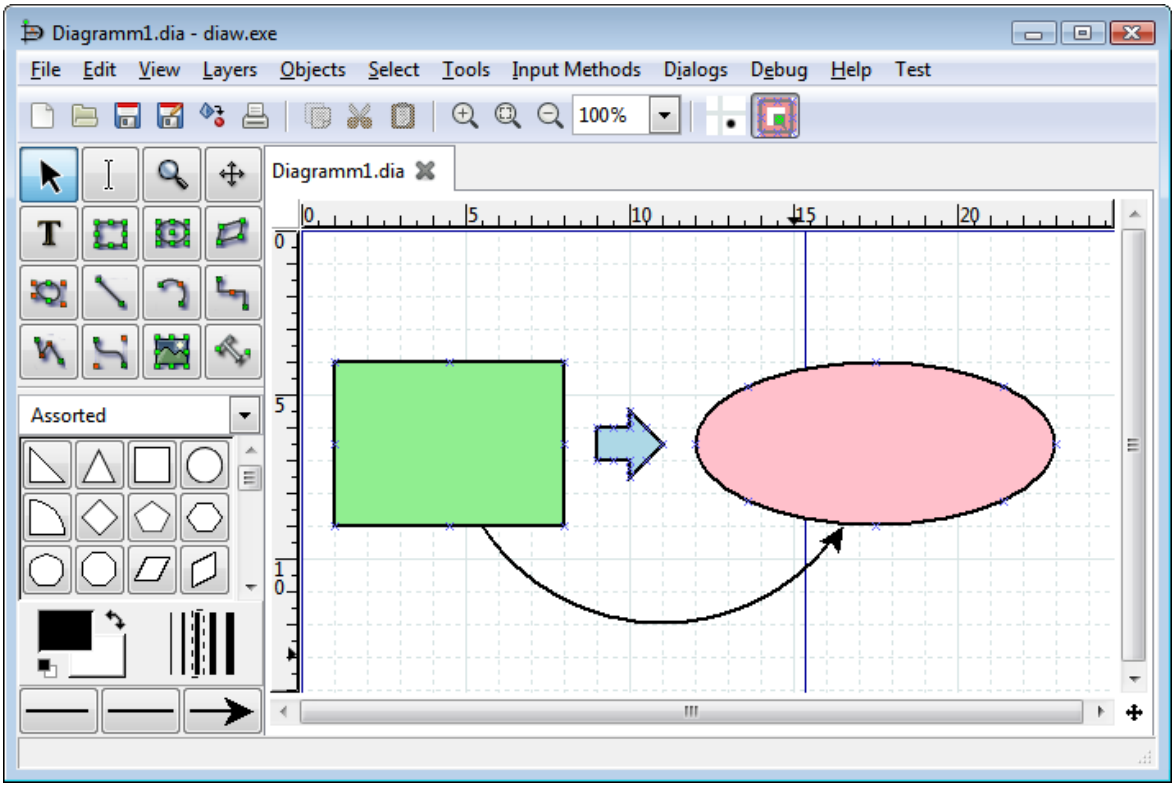


Figura 8. *Interfaz gráfica de Dia.*

PSeInt

Es una herramienta de software muy útil en la enseñanza de algoritmos y programación, cuenta con un editor simple de pseudocódigo en español y un editor de diagramas de flujo, puede interpretar el pseudocódigo para su ejecución con lo cual el alumnado podrá comprobar si su algoritmo es correcto y cumple lo solicitado.

Es libre, multiplataforma (Windows, Linux y macOS) y permite convertir el algoritmo a diferentes lenguajes de programación como C, C++, C#, Java y Python.

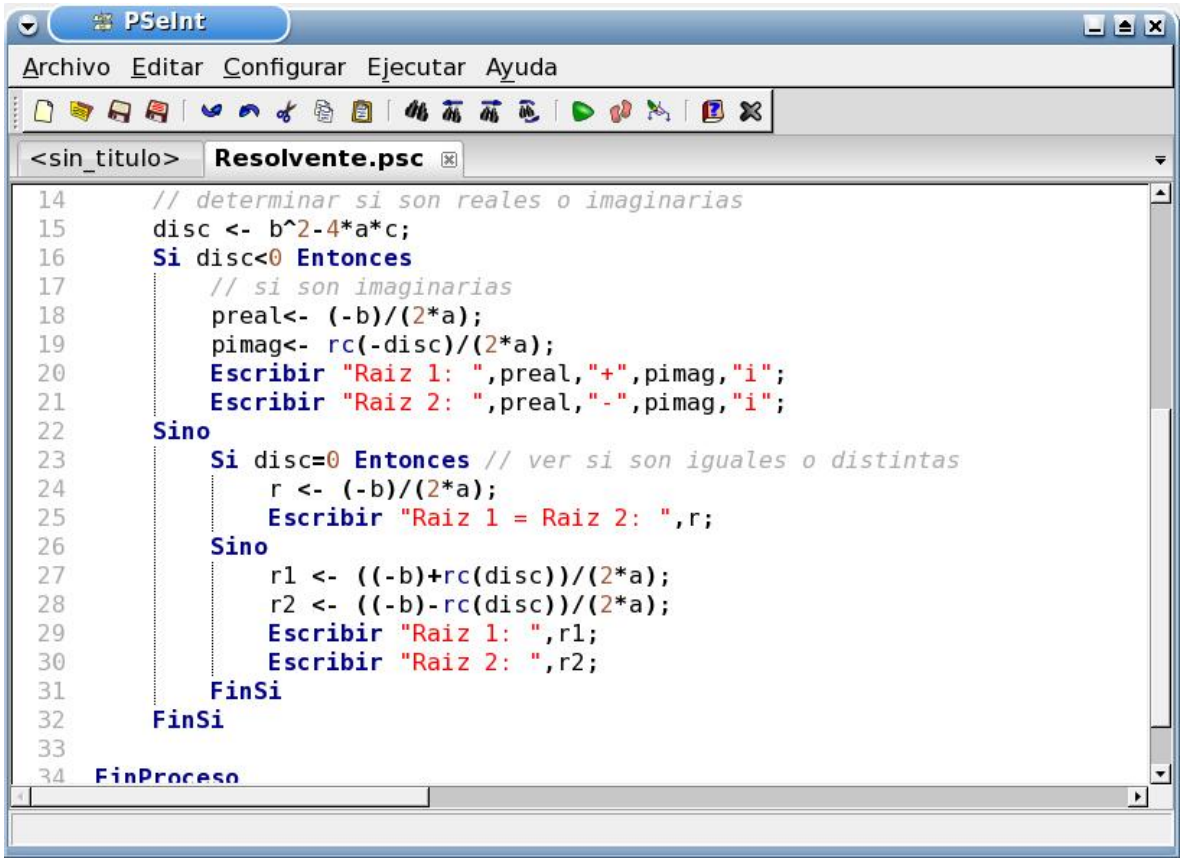


Figura 9. *Interfaz gráfica de PSeInt.*

IDE para programación

Geany

Entorno de desarrollo integrado ligero con soporte para más de 50 lenguajes como C, C++, C#, Java, Python, FORTRAN, entre otros. Entre sus características está el resaltado de sintaxis, autocompletado, gestión de proyectos sencilla, compilado y ejecución de código. Es software libre y se ejecuta en Windows, Linux y macOS.

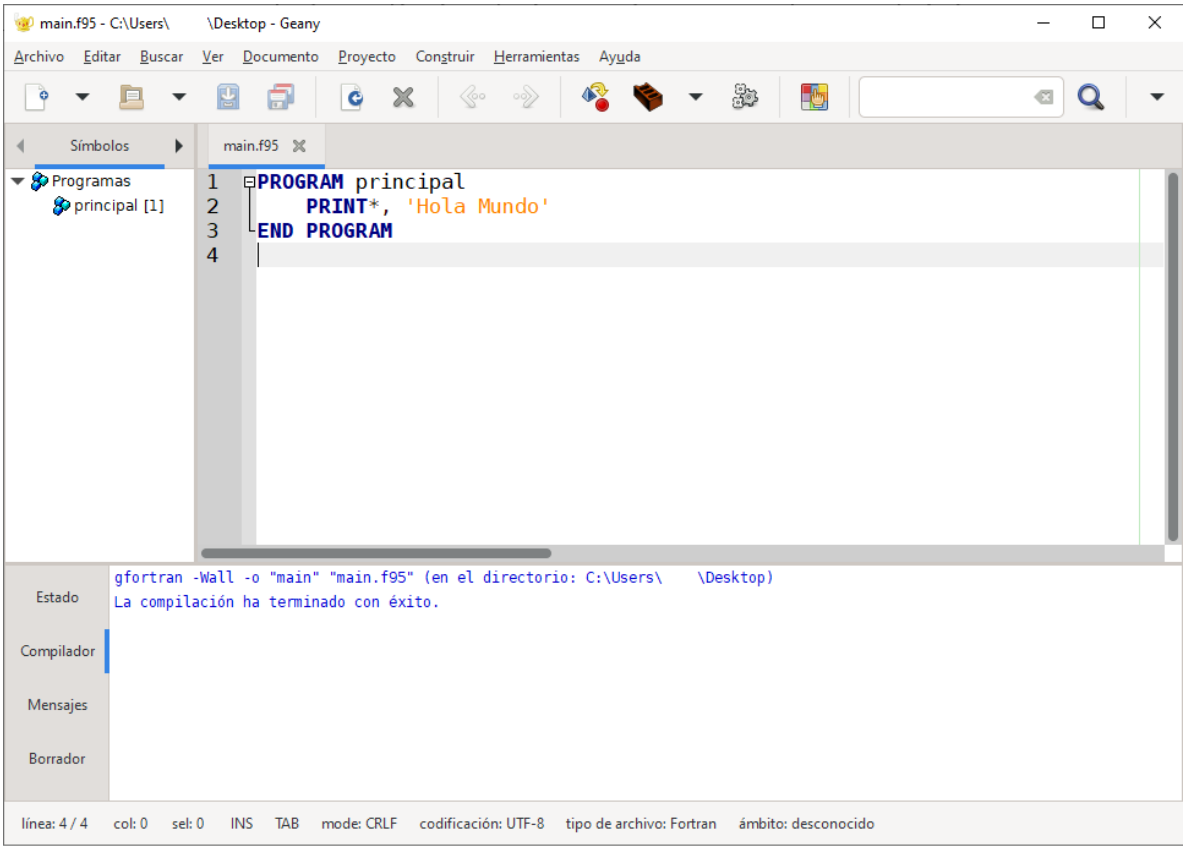


***Figura 10**. Interfaz gráfica de Geany.*

OnlineGDB

Es una herramienta de compilación y depuración en línea para diversos lenguajes de programación entre los que destacan C, C++, Java, FORTRAN. El usuario puede editar, compilar y depurar su código.

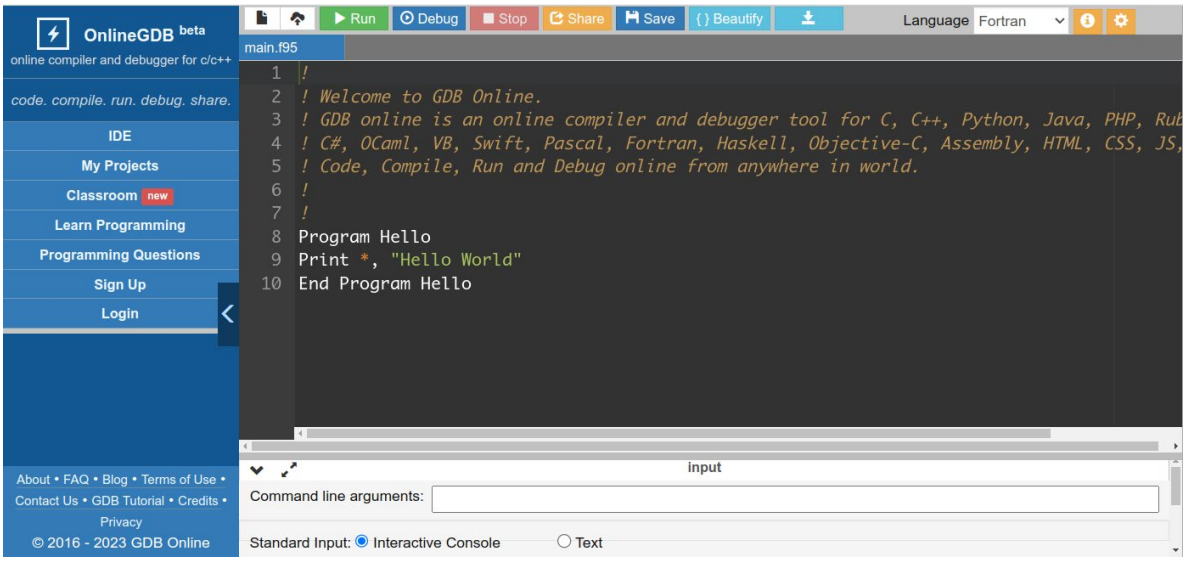


Figura 11. *Interfaz Web de OnlineGDB*

Aplicaciones y servicios en la nube

Google Docs

Permite crear y colaborar en documentos en línea en tiempo real y desde cualquier dispositivo. Además permite la edición de archivos de Microsoft Word y la exportación a PDF.

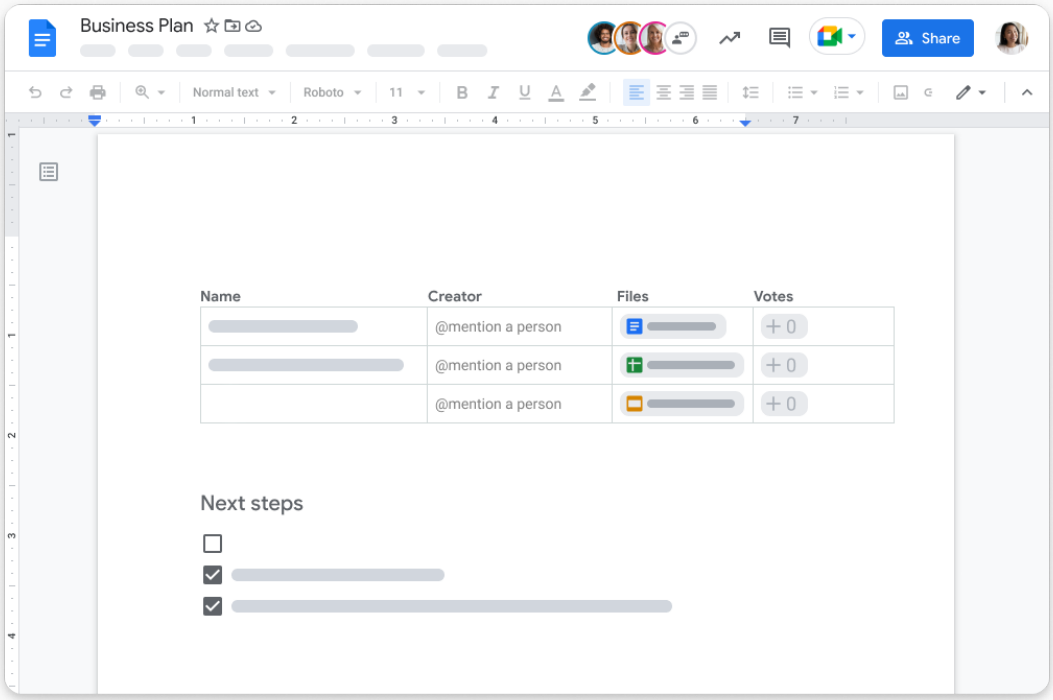


***Figura 12**. Google Docs.*

Google Drive

Permite guardar y compartir tanto archivos como carpetas completas subidas desde dispositivo móvil, tableta o computadora personal. Se integra con las aplicaciones nativas de la nube Documentos, Hojas de cálculo y Presentaciones para permitir la colaboración en tiempo real.

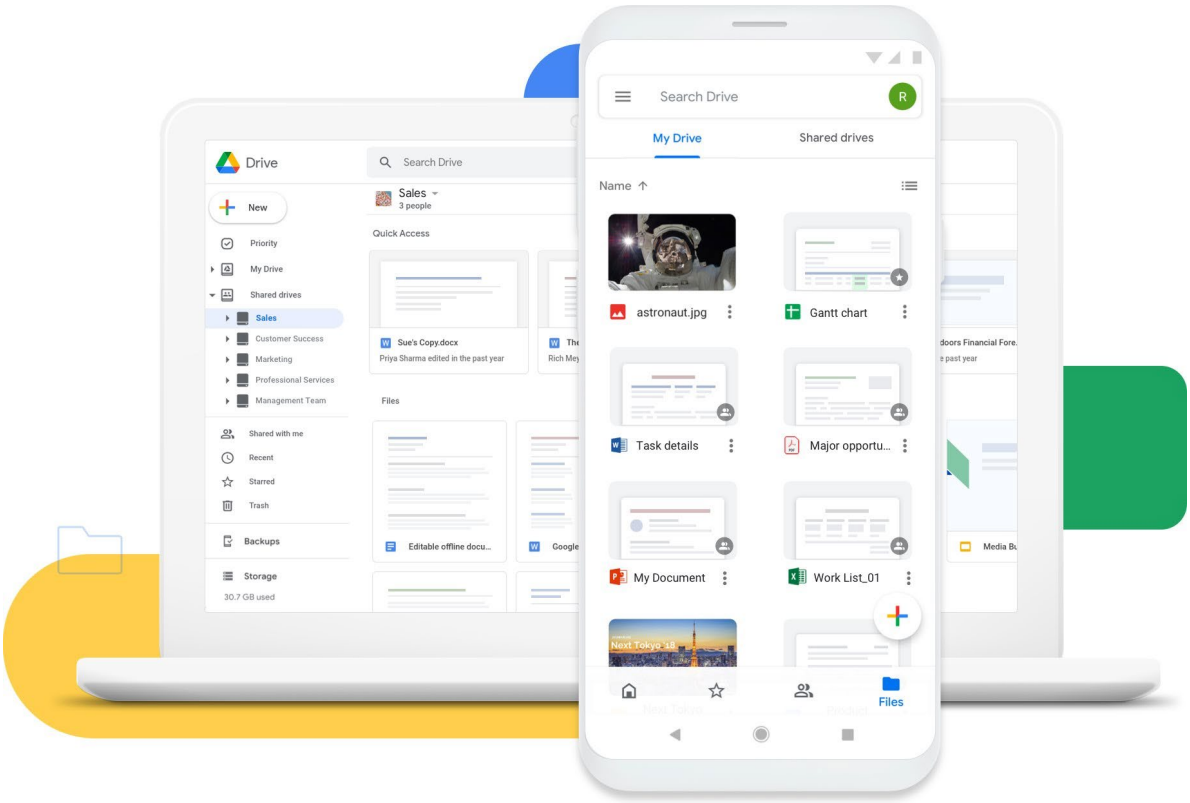


Figura 13. *Google Drive.*

Referencias

- Day, A. (2021, marzo 16). Fedora Workstation 34 feature focus: Updated Activities Overview. Fedora Magazine; Fedora Project. https://fedoramagazine.org/fedora-34-feature-focus-updated-activities-overview/

- Descubre el escritorio de la Mac. (s/f). Apple Support. Recuperado el 10 de agosto de 2023, de https://support.apple.com/es-mx/guide/mac-help/mh40612/mac

- Google Docs: Online document editor. (s/f). Google.com. Recuperado el 10 de agosto de 2023, de https://www.google.com/docs/about/?utm_source=gaboutpage&utm_medium=docslink&utm_campaign=gabout

- Home. (s/f). Geany.org. Recuperado el 10 de agosto de 2023, de https://www.geany.org/

- Macke, S. (s/f). Dia draws your structured diagrams: Free Windows, Mac OS X and Linux version of the popular open source program. Dia-installer.de. Recuperado el 10 de agosto de 2023, de http://dia-installer.de/index.html.en

- No title. (s/f). Oracle.com. Recuperado el 10 de agosto de 2023, de https://docs.oracle.com/cd/E19620-01/805-7644/6j76klopa/index.html

- Personal cloud storage & file sharing platform - Google. (s/f). Google.com. Recuperado el 10 de agosto de 2023, de https://www.google.com/drive/

- Presentamos Windows 11. (2021, junio 24). News Center Latinoamérica. https://news.microsoft.com/es-xl/features/presentamos-windows-11/

- PSeInt. (s/f). Sourceforge.net. Recuperado el 10 de agosto de 2023, de https://pseint.sourceforge.net/index.php?page=imagenes.php&sel=0

- Smith, O. (2022, abril 22). Ubuntu 22.04 LTS – what's new for the world's most popular Linux desktop? Ubuntu. https://ubuntu.com/blog/ubuntu-22-04-lts-whats-new-linux-desktop

- Qué es un Sistema Operativo. (2014, mayo 12). En: Significados.com. Disponible en: https://www.significados.com/sistema-operativo/ Consultado: 10 de agosto de 2023

Guía práctica de estudio 02: Análisis de algoritmos

Elaborado por:	Revisado por:	Autorizado por:
Ing. Jorge Luis López García Ing. Mayelly Reynoso Andrade	M. C. Laura Sandoval Montaño	Dra. Rocío Alejandra Aldeco Pérez Ing. Luis Sergio Valencia Castro

Guía práctica de estudio 02: Análisis de algoritmos

Objetivo:

El alumnado aprenderá a identificar las etapas de análisis de un problema desde su planteamiento, datos de entrada a procesar y salidas esperadas para su posterior diseño y solución.

Actividades:

1. Definir un problema, identificar datos de entrada y salida sin realizar la etapa de proceso para un ejemplo proporcionado por el profesor.

2. Identificar datos de entrada y salida para problemas simples propuestos de física, matemáticas y áreas de ingeniería sin realizar la etapa de proceso.

Introducción:

Algoritmo:

- Método para resolver un problema mediante una serie de pasos ordenados, numerados, secuenciales y finitos.

Tipos:

- Gráficos: Diagrama de Flujo.
- No Gráficos: Pseudocódigo.
- Computacionales: Se pueden programar por contar con operaciones.
- No computacionales: No se pueden programar por ser recetas o instructivos.

Etapas de un algoritmo:

- Análisis del problema.
  - Análisis de requerimientos.
  - Restricciones.
  - Identificar datos de entrada y salida.
- Diseño o construcción del algoritmo.
  - Descripción detallada de los pasos para resolver el problema en forma de lenguaje natural, pseudocódigo o diagrama de flujo.
- Verificación del algoritmo.
  - Prueba de escritorio.

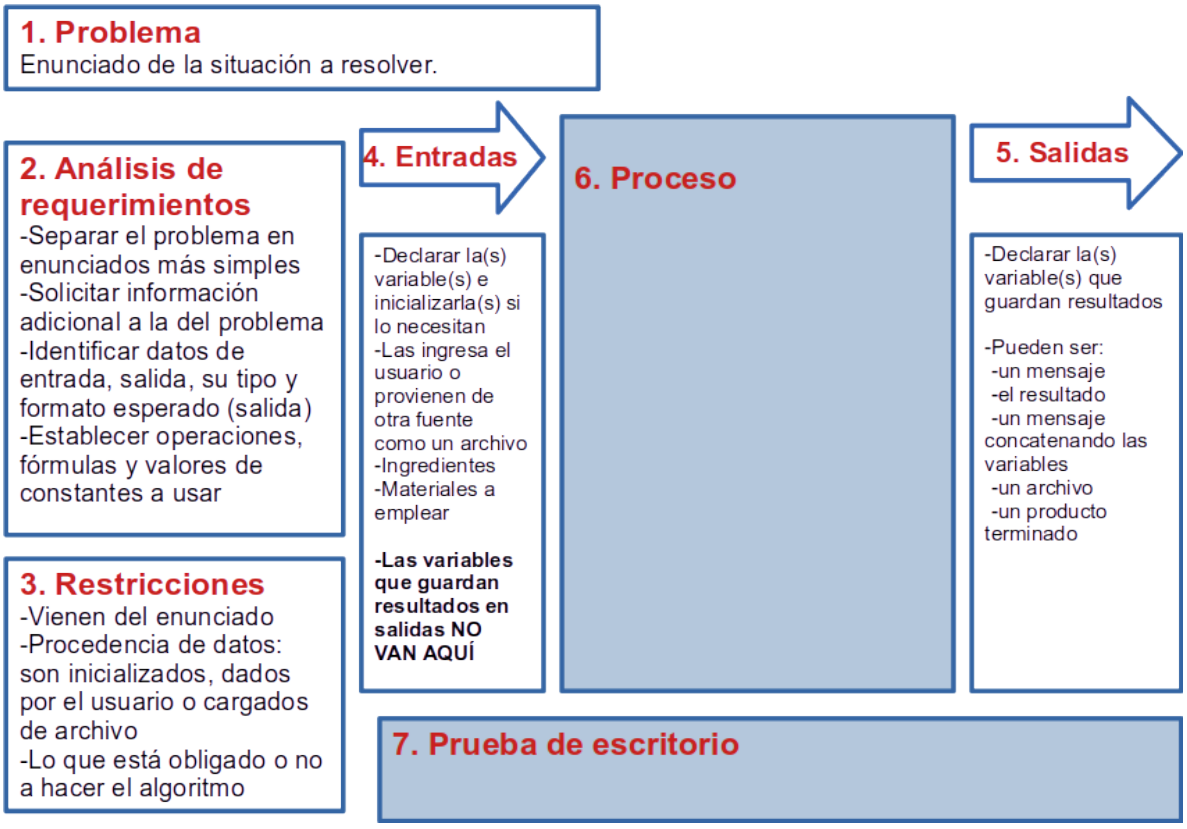


Figura 1. *Etapas de un algoritmo.*

Otra forma más simple de realizar el diseño del algoritmo consiste únicamente en identificar datos de entrada y de salida:

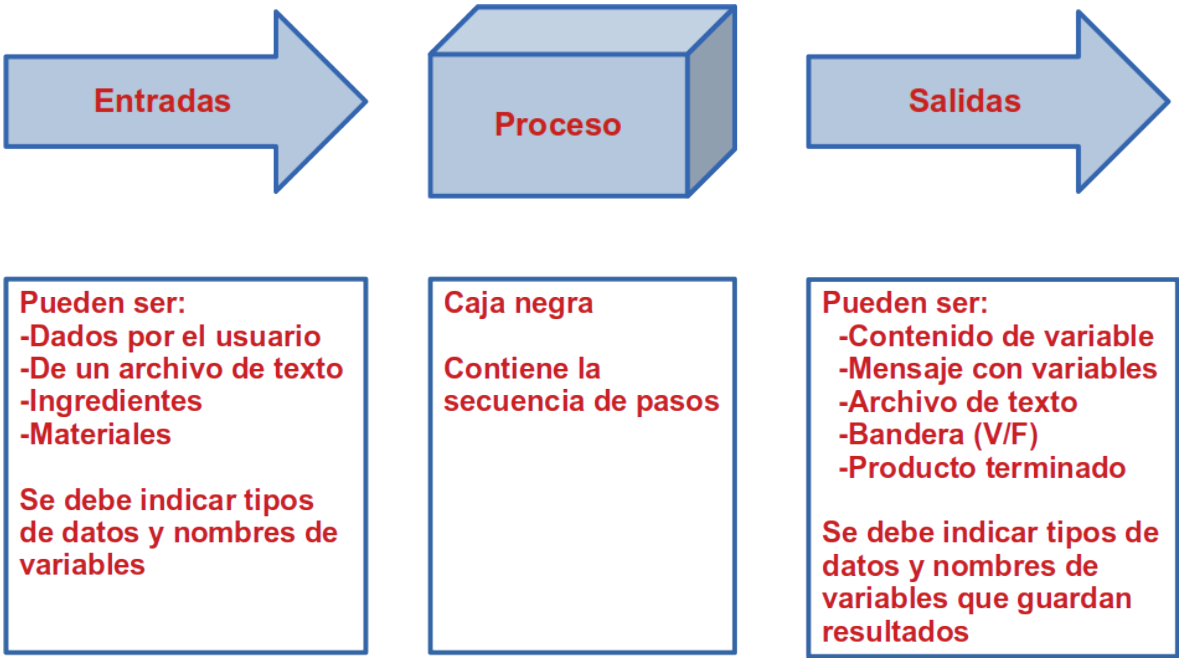


Figura 2. *Datos de entrada y salida.*

Una vez identificados los datos de entrada y salida se procede a declarar las variables necesarias para cada uno. La acción de declarar una variable involucra asignarle un nombre y un tipo de dato que puede ser tomado de la siguiente tabla:

Tipo	Variante	Valores que toma
Números	`Entero`	Número sin punto decimal
	`Real`	Número CON punto decimal
Caracteres	`Carácter`	Una letra minúscula, mayúscula, símbolo, dígito
	`Cadena de caracteres`	Secuencia de letras minúsculas, mayúsculas, símbolos, dígitos
Lógico		Verdadero o Falso

Tabla 1. *Tipos de datos.*

Las variables permiten guardar información de acuerdo al tipo elegido y se les puede asignar un dato inicial al declararse, un dato dado por el usuario, copiar el contenido de otra variable o asignar el resultado de alguna operación.

Para dar el nombre a una variable el alumnado puede adoptar las siguientes recomendaciones:

- No emplear solo una letra dado que es fácil olvidar la información que guardará.
- Emplear una palabra relacionada a la información.
- Si se desea emplear más de una palabra recurrir a alguna notación que las junte usando mayúsculas, guiones bajo o guiones medio dado que no puede haber espacios en nombres de variables.
- Además de los espacios también se debe omitir el uso de símbolos especiales.

Ejemplo 1

Realiza la etapa de análisis de algoritmo para el siguiente problema:

1. Problema

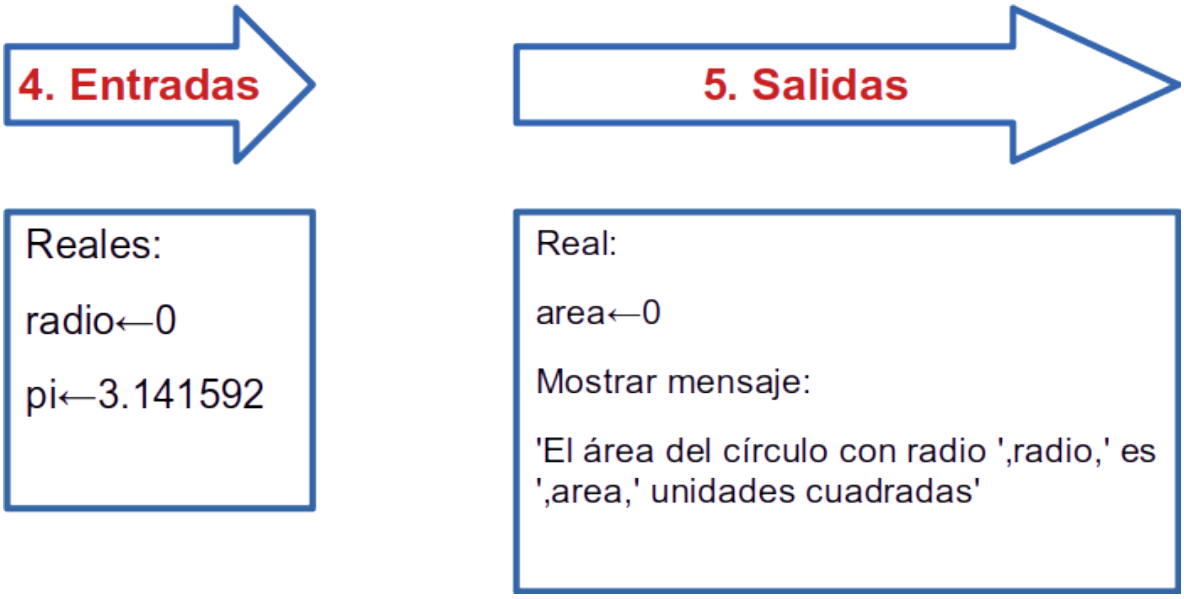
Calcular el área de un círculo con radio dado por el usuario

2. Análisis de requerimientos

-Calcular área
-Es un círculo
-La entrada es real positivo
-Es dada por el usuario
-La salida es un real positivo y un mensaje
-Las unidades de salida son las mismas que las de entrada
-Usar area←pi*radio^2 o area←pi*radio*radio
-Usar pi←3.141592

3. Restricciones

-Es área
-Es círculo
-El usuario interactúa



En el mensaje de la salida se observa la concatenación, operación que ayuda a unir una cadena con el contenido de una variable. Iniciando el mensaje se encuentra la cadena delimitada con comillas simples, a continuación el operador de concatenación "," y posteriormente la variable de la cual se desea conocer su contenido ya sea proporcionado originalmente por el usuario o resultante de alguna operación. El mensaje en una salida puede estar conformado por un conjunto de concatenaciones de mensajes y variables recordando que entre cada uno no debe faltar el operador ",".

Además de los algoritmos computacionales también puede presentarse la necesidad de resolver problemas que no implican operaciones matemáticas ni uso de variables como es el caso de instructivos, manuales o recetas, estos son los algoritmos no computacionales o cualitativos.

Ejemplo 2

Realiza la etapa de análisis de algoritmo para el siguiente problema:

1. Problema

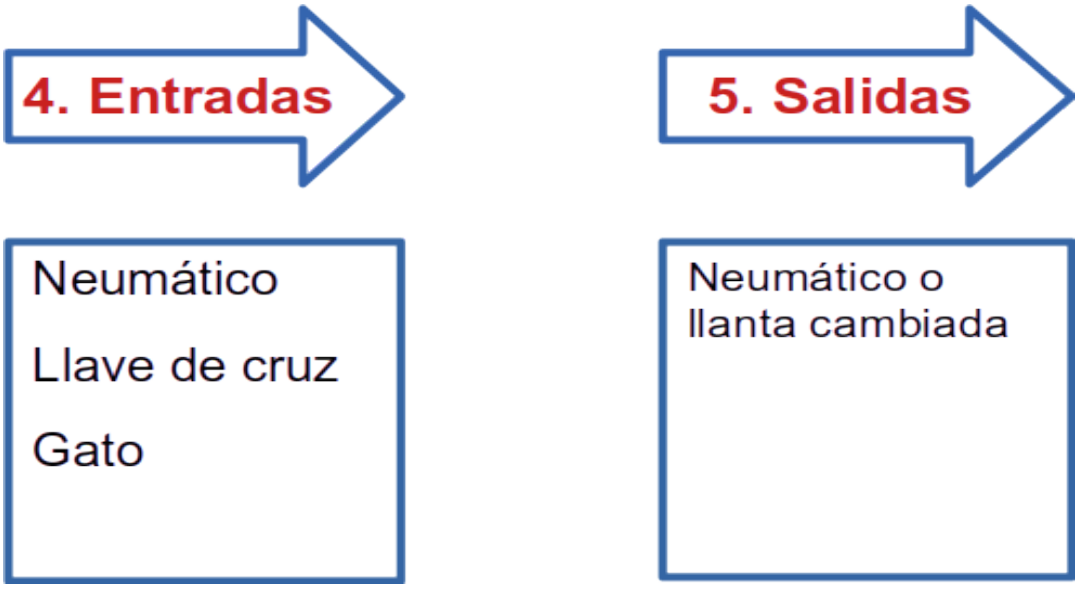
Cambiar un neumático de un automóvil

2. Análisis de requerimientos

-Cambio de llanta
-Las entradas son las herramientas necesarias
-No hay operaciones
-No hay variables
-La salida es la acción deseada terminada

3. Restricciones

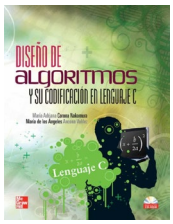
-Contar con la llanta de refacción
-Tener la herramienta



Bibliografía

Cairó, O. (2005). Metodología de la programación: Algoritmos, diagramas de flujo y programas. (3a. ed.). Alfaomega.

Corona, M. A. y Ancona, M. A. (2011). Diseño de algoritmos y su codificación en lenguaje C. Mc Graw Hill.



Guía práctica de estudio 03: Diseño de algoritmos con instrucciones en secuencia

Elaborado por:	Revisado por:	Autorizado por:
Ing. Jorge Luis López García Ing. Mayelly Reynoso Andrade	M. C. Laura Sandoval Montaño	Dra. Rocío Alejandra Aldeco Pérez Ing. Luis Sergio Valencia Castro

Guía práctica de estudio 03: Diseño de algoritmos con instrucciones en secuencia

Objetivo:

El alumnado diseñará algoritmos gráficos y no gráficos a partir de un análisis previo para resolver problemas que involucren instrucciones de tipo secuencia como parte del proceso.

Actividades:

1. Definir un problema, identificar datos de entrada y salida, diseñar un proceso en pseudocódigo y en diagrama de flujo para resolver un ejemplo proporcionado por el profesor.

2. A partir del problema y la etapa de análisis de algoritmos desarrollados previamente para ejercicios simples de física, matemáticas y áreas de ingeniería, realizar la etapa de diseño de algoritmos en forma de diagrama de flujo y pseudocódigo involucrando acciones como la declaración de variables, lectura de datos, operaciones e imprimir datos.

Introducción:

Como se mencionó anteriormente las etapas para resolver un problema mediante un algoritmo, pueden resumirse en su análisis, diseño y verificación. En la práctica anterior se abordó como parte del análisis el planteamiento del problema, análisis de requerimientos, restricciones, identificar datos de entrada y datos de salida.

Corresponde a esta práctica pasar a las etapas de diseño y verificación del algoritmo.

Las características de todas las etapas pueden verse en la siguiente figura:

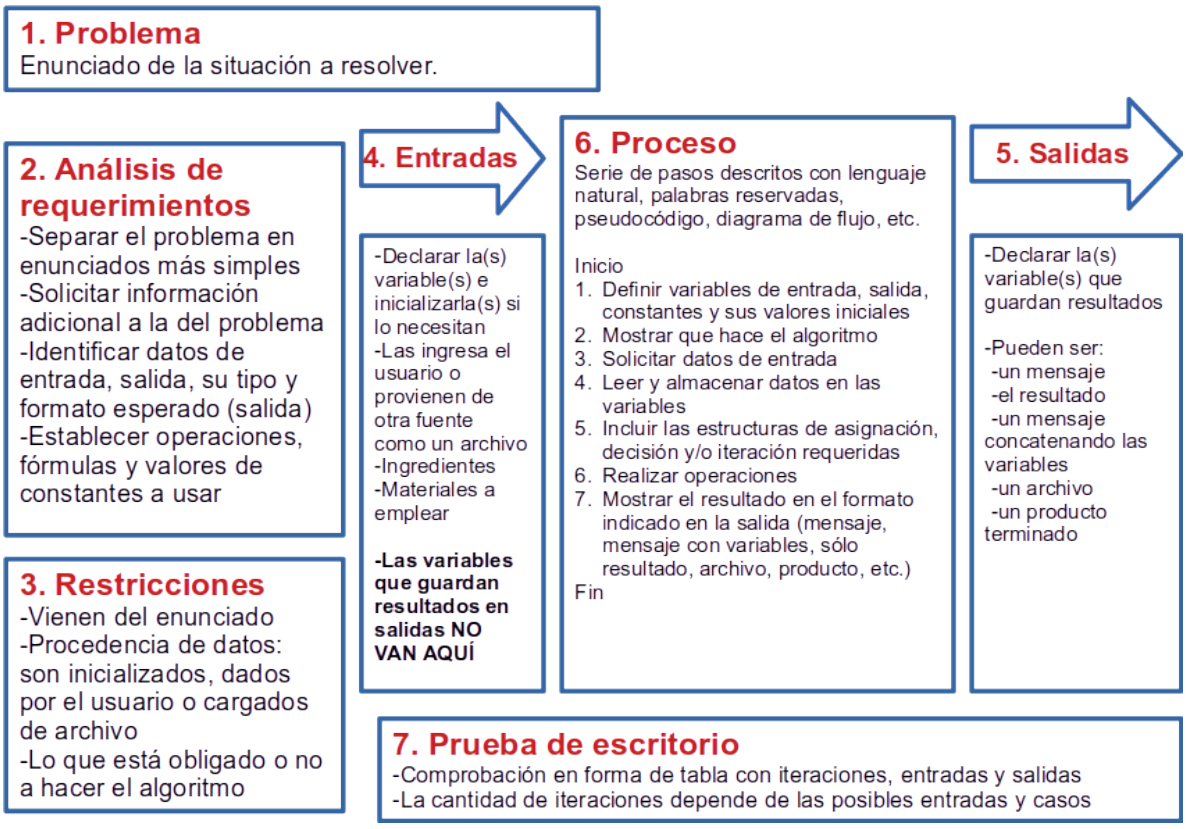


Figura 1. *Etapas de un algoritmo.*

Para elaborar el proceso propio de la etapa de diseño se puede recurrir como se muestra en la imagen a palabras reservadas que representen alguna acción importante a lo que se conoce como pseudocódigo o a símbolos específicos que corresponden también a alguna de esas acciones colocados en secuencia formando un diagrama de flujo.

La siguiente tabla contiene ciertas palabras reservadas elegidas para escribir pseudocódigo y su símbolo equivalente para diagrama de flujo:

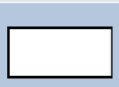
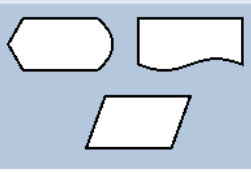
Acción de pseudocódigo	Símbolo de diagrama de flujo	Descripción
`INICIO / ALGORITMO` `FIN / FIN ALGORITMO`		Inicio y fin del algoritmo
`ASIGNAR / DECLARAR / DEFINIR`		Dar nombre, tipo y dato inicial a una variable
`CALCULAR / REALIZAR OPERACIÓN`		Asignar el resultado de una operación o el contenido de una variable a otra
`LEER Y GUARDAR EN / LEER Y ALMACENAR EN`		Recibir valor dado por el usuario y guardar en variable
`IMPRIMIR / ESCRIBIR / MOSTRAR MENSAJE`		Mandar mensajes y contenido de variables al usuario

***Tabla 1**. Equivalencia de acciones en pseudocódigo y diagrama de flujo.*

Ejemplo 1

Realiza las etapas de análisis y diseño de algoritmo tanto en pseudocódigo como en diagrama de flujo para el siguiente problema:

1. Problema

Mostrar al usuario el mensaje "Hola Mundo"

2. Análisis de requerimientos

-No hay entradas
-No hay variables
-No hay operaciones
-La salida es un mensaje

3. Restricciones

-El mensaje debe ser "Hola Mundo"
-El usuario no interactúa

4. Entradas

5. Salidas

Mostrar mensaje:

'Hola Mundo'

```
6. PROCESO

INICIO

1.ESCRIBIR 'Hola Mundo'

FIN
```

```
1  Algoritmo holaMundo
2      Escribir 'Hola Mundo'
3  FinAlgoritmo
```

El pseudocódigo puede escribirse tan simple en una hoja de papel usando las palabras reservadas elegidas o con la ayuda de herramientas de software como PSeInt.

Figura 2. *Diagrama de flujo para "Hola Mundo" en Dia*

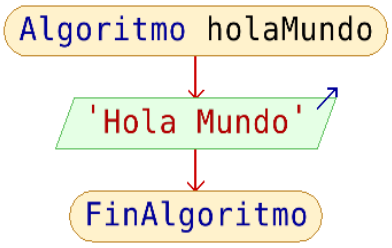


Figura 3. *Diagrama de flujo para "Hola Mundo" en PseInt*

Al igual que el pseudocódigo, el diagrama de flujo puede dibujarse en una hoja de papel empleando los símbolos apropiados o con la ayuda de herramientas de software como Dia, PSeInt u otras herramientas en línea.

Un proceso no es tan sencillo como solo mandar un mensaje al usuario, para resolver un problema computacional generalmente se requieren operadores matemáticos como parte de los pasos del diseño representados con símbolos, palabras reservadas o funciones.

Operación	Símbolo o función	Ejemplo de uso
ASIGNACIÓN	<- ←	a<- 1 resultado<- a+b
SUMA	+	a+b
RESTA	-	a-b
MULTIPLICACIÓN	*	a*b No usar: 2a ni 2(a)
DIVISIÓN	/	a/b
MÓDULO	MOD	a MOD b Residuo de a/b
RAÍZ CUADRADA	RC( ) RAIZ( ) SQRT( )	RC(a) RAIZ(a) SQRT(a)
POTENCIA	^	x^2
TRUNCAR	TRUNC( )	TRUNC(a) 5.9 da 5, 9.4 da 9

Tabla 2. *Operadores y funciones matemáticas.*

Operación	Símbolo o función	Ejemplo de uso
VALOR ABSOLUTO	**ABS( )**	`ABS(x)`
EXPONENCIAL	**EXP( )**	`EXP(x)`
LOGARITMO NATURAL	**LN( )**	`LN(x)`
NÚMERO ALEATORIO	**RANDOM( )**	`RANDOM(x) rango entre 0 y x-1`
	AZAR( )	`AZAR(x) rango entre 0 y x-1`
	ALEATORIO( )	`ALEATORIO(a,b) rango entre a y b`
REDONDEAR	**ROUND( )**	`ROUND(x) 3.3 da 3, 5.5 da 6`
	REDON( )	`REDON(x)`

Tabla 3. *Operadores y funciones matemáticas.*

Operación	Símbolo o función	Ejemplo de uso
OPERADOR LÓGICO Y	**AND**	`a AND b`
	Y	`a Y b`
OPERADOR LÓGICO O	**OR**	`a OR b`
	O	`a O b`
OPERADOR LÓGICO NEGACIÓN	**NOT**	`NOT a`
	NO	`NO a`

Tabla 4. *Operadores lógicos.*

Ejemplo 2

Realiza las etapas de análisis, diseño y verificación de un algoritmo para el siguiente problema:

1. Problema

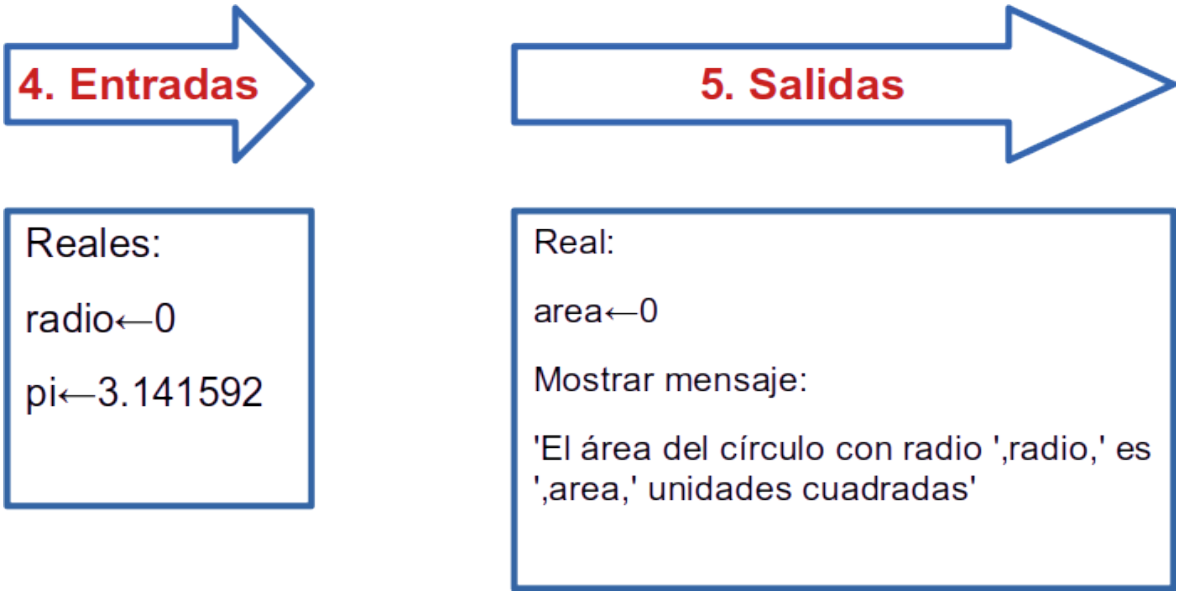
Calcular el área de un círculo con radio dado por el usuario

2. Análisis de requerimientos

-Calcular área
-Es un círculo
-La entrada es real positivo
-Es dada por el usuario
-La salida es un real positivo y un mensaje
-Las unidades de salida son las mismas que las de entrada
-Usar area←pi*radio^2 o area←pi*radio*radio
-Usar pi←3.141592

3. Restricciones

-Es área
-Es círculo
-El usuario interactúa



Proceso en pseudocódigo con palabras reservadas:

```
ETAPA 6. PROCESO

INICIO

1.DEFINIR radio<-0, area<-0, pi<-3.141592 como Reales

2.ESCRIBIR 'Este algoritmo calcula el área de un círculo con radio
  dado por el usuario'
3.ESCRIBIR 'Ingrese el valor del radio positivo: '
4.LEER y ALMACENAR EN radio
5.REALIZAR LA OPERACIÓN radio<-ABS(radio)
6.REALIZAR LA OPERACIÓN area<-pi*radio^2 o area<-pi*radio*radio
7.ESCRIBIR 'El área del círculo con radio ',radio,' es ',area,'
  unidades cuadradas'

FIN
```

Proceso en pseudocódigo con la herramienta PSeInt:

```
1  Algoritmo areaCirculo
2      Definir radio,area Como Real
3      radio← 0
4      area← 0
5
6      Escribir 'Este algoritmo calcula el área de un círculo con radio dado por el usuario'
7      Escribir ' '
8      Escribir 'Ingrese el valor del radio positivo: ' Sin Saltar
9      Leer radio
10     radio← ABS(radio)
11     area← pi*radio↑2
12     Escribir 'El área del círculo con radio ',radio,' es ',area,' unidades cuadradas'
13 FinAlgoritmo
```

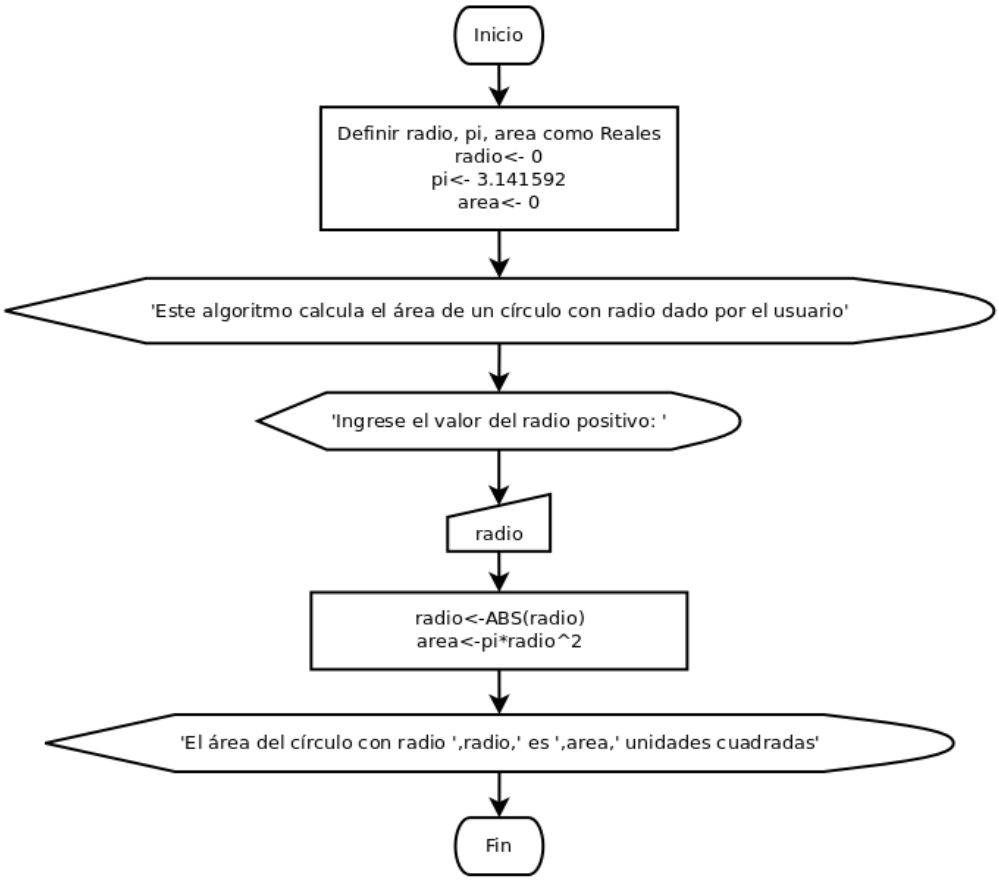


Figura 4. *Diagrama de flujo para el área del círculo en Dia.*

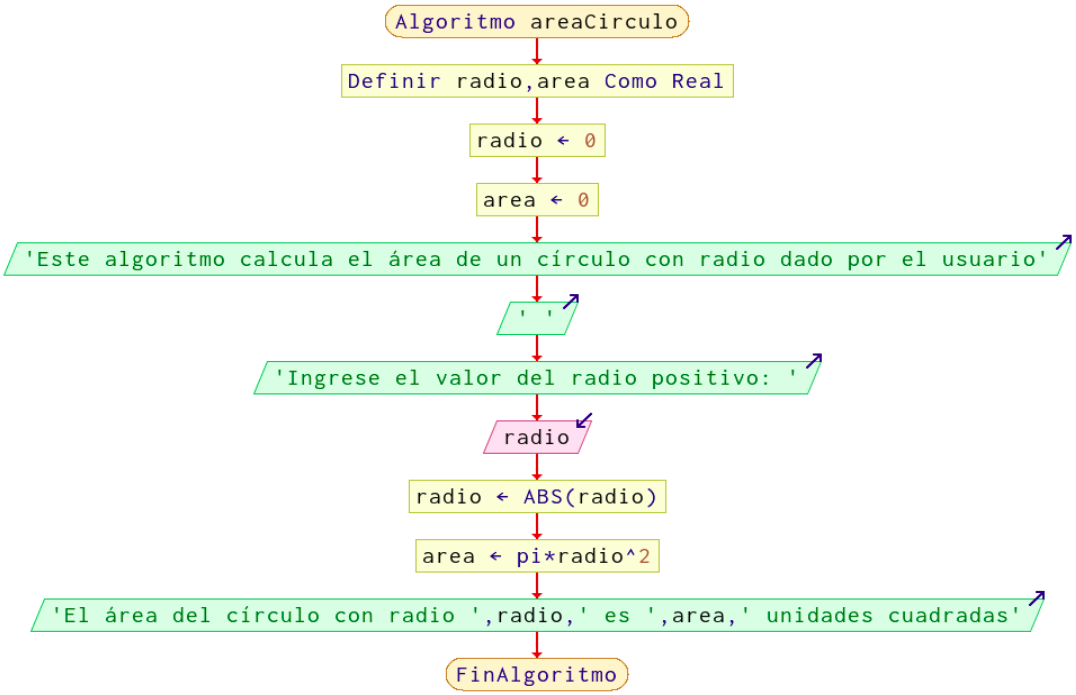


Figura 5. *Diagrama de flujo para el área del círculo en PSeInt.*

7. Prueba de escritorio

Iteración	Entrada(s)	Salidas(s)
1	radio= 1	area= 3.141592 El área del círculo con radio 1 es 3.141592 unidades cuadradas
2	radio= -1	area= 3.141592 El área del círculo con radio 1 es 3.141592 unidades cuadradas
3	radio= 0	area= 0 El área del círculo con radio 0 es 0 unidades cuadradas

Reglas y recomendaciones para escribir pseudocódigos:

- Los pasos siempre son secuenciales, de arriba a abajo.
- Comenzar con alguna de las palabras reservadas elegidas en el caso de un algoritmo computacional.
- Las palabras reservadas preferentemente distinguirse en mayúsculas.
- En los mensajes de salida concatenar los datos ingresados por el usuario para que confirme su veracidad y las variables con resultados.
- El pseudocódigo debe ser genérico, sin relación a algún lenguaje de programación.

Reglas y recomendaciones para diseñar diagramas de flujo:

- La secuencia del diagrama debe ser de arriba hacia abajo y de izquierda a derecha.
- Las flechas que unen los símbolos deben ser rectas, verticales u horizontales, no en diagonal.
- Sólo una flecha llega a cada símbolo, en caso de retornos por ciclos la flecha de regreso debe conectarse con la flecha que llegará al símbolo.

Ejemplo 3

Realiza las etapas de análisis, diseño y verificación de un algoritmo para el siguiente problema no computacional:

1. Problema

Cambiar un neumático de un automóvil

2. Análisis de requerimientos

-Cambio de llanta
-Las entradas son las herramientas necesarias
-No hay operaciones
-No hay variables
-La salida es la acción deseada terminada

3. Restricciones

-Contar con la llanta de refacción
-Tener la herramienta

4. Entradas

Neumático

Llave de cruz

Gato

5. Salidas

Neumático o llanta cambiada

```
ETAPA 6. PROCESO

INICIO

1.Bajar la herramienta y el neumático de repuesto del automóvil
2.Aflojar con la llave de cruz los birlos del neumático averiado
3.Acomodar el gato en el punto adecuado de soporte del automóvil
4.Levantar el automóvil con el gato
5.Quitar los birlos del neumático desinflado
6.Quitar el neumático desinflado
7.Colocar el neumático de repuesto
8.Fijar los birlos del neumático de repuesto
9.Bajar el automóvil
10.Apretar de forma definitiva los birlos del neumático de repuesto

FIN
```

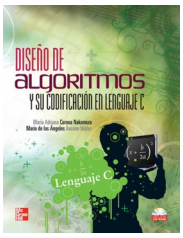
Se observa que la serie de pasos no cuenta con ninguna de las acciones o palabras reservadas de pseudocódigo para el algoritmo computacional, en su lugar las acciones se describen con verbos en infinitivo.

Para la etapa 7 no se cuenta con una tabla de iteraciones.

Bibliografía

Cairó, O. (2005). Metodología de la programación: Algoritmos, diagramas de flujo y programas. (3a. ed.). Alfaomega.

Corona, M. A. y Ancona, M. A. (2011). Diseño de algoritmos y su codificación en lenguaje C. Mc Graw Hill.



Guía práctica de estudio 04: Diseño de algoritmos con instrucciones de selección y repetición

Elaborado por:	Revisado por:	Autorizado por:
Ing. Jorge Luis López García Ing. Mayelly Reynoso Andrade	M. C. Laura Sandoval Montaño	Dra. Rocío Alejandra Aldeco Pérez Ing. Luis Sergio Valencia Castro

Guía práctica de estudio 04: Diseño de algoritmos con instrucciones de selección y repetición

Objetivo:

El alumnado diseñará algoritmos gráficos y no gráficos a partir de un análisis previo para resolver problemas que involucren instrucciones simples de selección y repetición como parte del proceso.

Actividades:

1. Definir un problema, identificar datos de entrada y salida, diseñar un proceso en pseudocódigo y en diagrama de flujo para un ejemplo proporcionado por el profesor.

2. Realizar la etapa de diseño de algoritmos para ejercicios simples de física, matemáticas y áreas de ingeniería en forma de diagrama de flujo y pseudocódigo involucrando acciones como la declaración de variables, leer datos, operaciones, imprimir datos, instrucciones de selección Si, Si y Caso contrario, así como la instrucción de repetición Para.

Introducción:

Estructuras de selección.

Al realizar la etapa de diseño de un algoritmo, ya sea en pseudocódigo o diagrama de flujo, es muy probable que se presente la necesidad de validar un dato proporcionado por el usuario con el fin de evitar que provoque alguna falla durante la ejecución, como puede ser comprobar que sea positivo, mayor que otro, se encuentre dentro de un rango, entre otras situaciones.

Para tales tareas son muy útiles las estructuras de selección Si, Si y caso contrario, así como su combinación al anidarse como resultado de una anterior con lo que el sentido del flujo del diagrama cambia o el bloque a realizar depende de si las condiciones evaluadas se cumplieran o no.

Las siguientes tablas contienen las palabras reservadas elegidas para escribir pseudocódigo y su símbolo equivalente para diagrama de flujo para algunas variantes de las estructuras de selección.

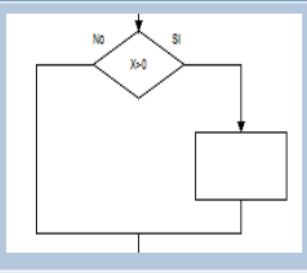
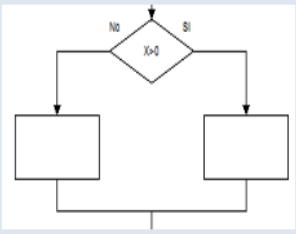
Acción de pseudocódigo	Símbolo de diagrama de flujo	Descripción
SI x>0 ENTONCES acciones FIN DEL SI		Sólo valida el caso donde la condición se cumpla
SI x>0 ENTONCES acciones CASO CONTRARIO acciones FIN DEL SI		Valida los casos donde la condición se cumpla y también que no se cumpla

Tabla 1. *Equivalencia de acciones en pseudocódigo y diagrama de flujo para selección simple y doble selección*

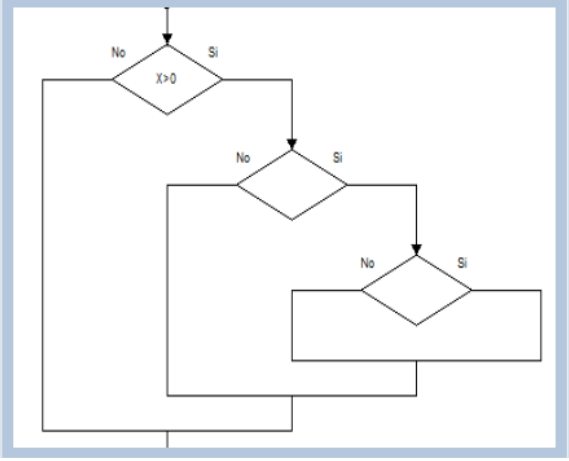
Acción de pseudocódigo	Símbolo de diagrama de flujo	Descripción
SI x>0 ENTONCES SI ENTONCES SI ENTONCES FIN DEL SI FIN DEL SI FIN DEL SI		Pueden anidarse decisiones del lado del SÍ

Tabla 3. *Equivalencia de acciones en pseudocódigo y diagrama de flujo para decisiones anidadas*

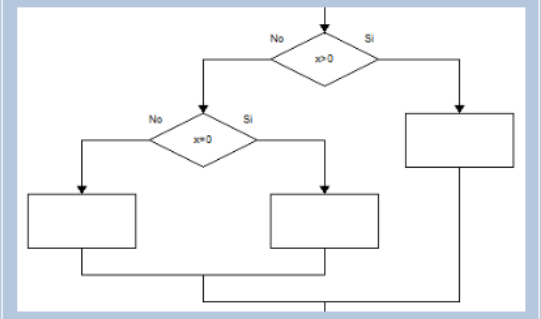
Acción de pseudocódigo	Símbolo de diagrama de flujo	Descripción
SI x>0 ENTONCES acciones CASO CONTRARIO SI x=0 ENTONCES acciones CASO CONTRARIO acciones FIN DEL SI FIN DEL SI		Pueden anidarse decisiones tanto del lado del SÍ como del lado del NO

Tabla 2. *Equivalencia de acciones en pseudocódigo y diagrama de flujo para decisiones anidadas*

Operadores de comparación

Permiten establecer la relación entre una variable y otra o entre una variable y un número. Dicha relación puede ser de igualdad, diferencia, mayor o menor. Al comprobar la veracidad de la relación o condición o por el contrario su falsedad se decide cuál símbolo o instrucción será la siguiente a realizar.

Símbolo	Operador
>	MAYOR QUE
<	MENOR QUE
>=	MAYOR O IGUAL QUE
<=	MENOR O IGUAL QUE
= ==	IGUAL E IDÉNTICO
!= <>	DIFERENTE

Tabla 4. *Operadores de comparación*

Operadores lógicos

Dada una condición a evaluar en una estructura de selección puede ayudar a optimizar el algoritmo el agregar más relaciones entre variables dentro de la misma condición.

Símbolo			Operador
Y	AND	&	Y
O	OR	|	O
NO	NOT	~	NEGACIÓN

Tabla 4. *Operadores lógicos*

Ejemplo 1

Realiza las etapas de análisis, diseño y verificación de un algoritmo para el siguiente problema:

1. Problema

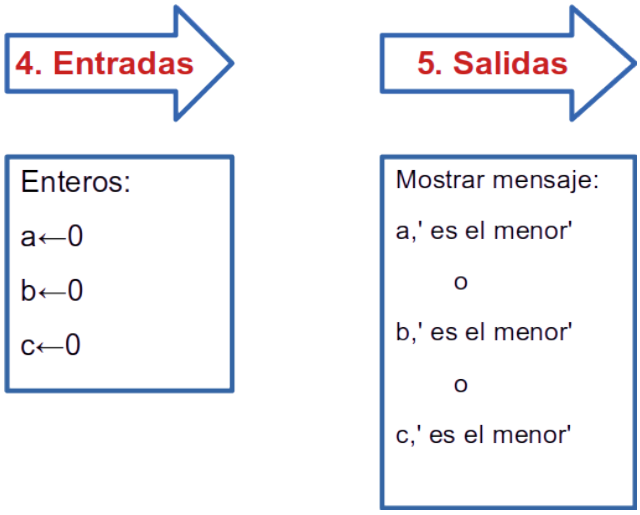
Leer 3 números y mostrar el menor

2. Análisis de requerimientos

-Mostrar el menor de 3
-Las entradas son enteros
-Son dadas por el usuario
-Usar operadores de comparación y opcionalmente lógicos
-No hay variables para guardar salidas
-La salida es un mensaje con la variable de entrada con contenido menor

3. Restricciones

-Son 3 números
-Muestra el menor



Proceso en pseudocódigo con palabras reservadas:

```
ETAPA 6. PROCESO

INICIO

1. DEFINIR a<-0, b<-0, c<-0 como Enteros
2. ESCRIBIR 'Este algoritmo lee 3 números y dice cuál es el menor'
3. ESCRIBIR 'Ingresa los tres números separados por comas: '
4. LEER y ALMACENAR EN a,b,c
5. SI a<b ENTONCES
    5.1 SI a<c ENTONCES
        5.1.1 ESCRIBIR a,' es el menor'
    5.2 CASO CONTRARIO
        5.2.1 ESCRIBIR c,' es el menor'
    5.3 FIN DEL SI
6. CASO CONTRARIO
    6.1 SI b<c ENTONCES
        6.1.1 ESCRIBIR b,' es el menor'
    6.2 CASO CONTRARIO
        6.2.1 ESCRIBIR c,' es el menor'
    6.3 FIN DEL SI
7. FIN DEL SI

FIN
```

Proceso en pseudocódigo con la herramienta PSeInt:

```
1  Algoritmo menorDe3
2      Definir a,b,c Como Entero
3      a←0
4      b←0
5      c←0
6      Escribir 'Este algoritmo lee 3 números y dice cuál es el menor'
7      Escribir 'Ingresa los tres números separados por ENTER: '
8      Leer a,b,c
9
10     Si a<b Entonces
11         Si a<c Entonces
12             Escribir a,' es el menor'
13         Sino
14             Escribir c,' es el menor'
15         Fin Si
16     Sino
17         Si b<c Entonces
18             Escribir b,' es el menor'
19         Sino
20             Escribir c,' es el menor'
21         Fin Si
22     Fin Si
23 FinAlgoritmo
```

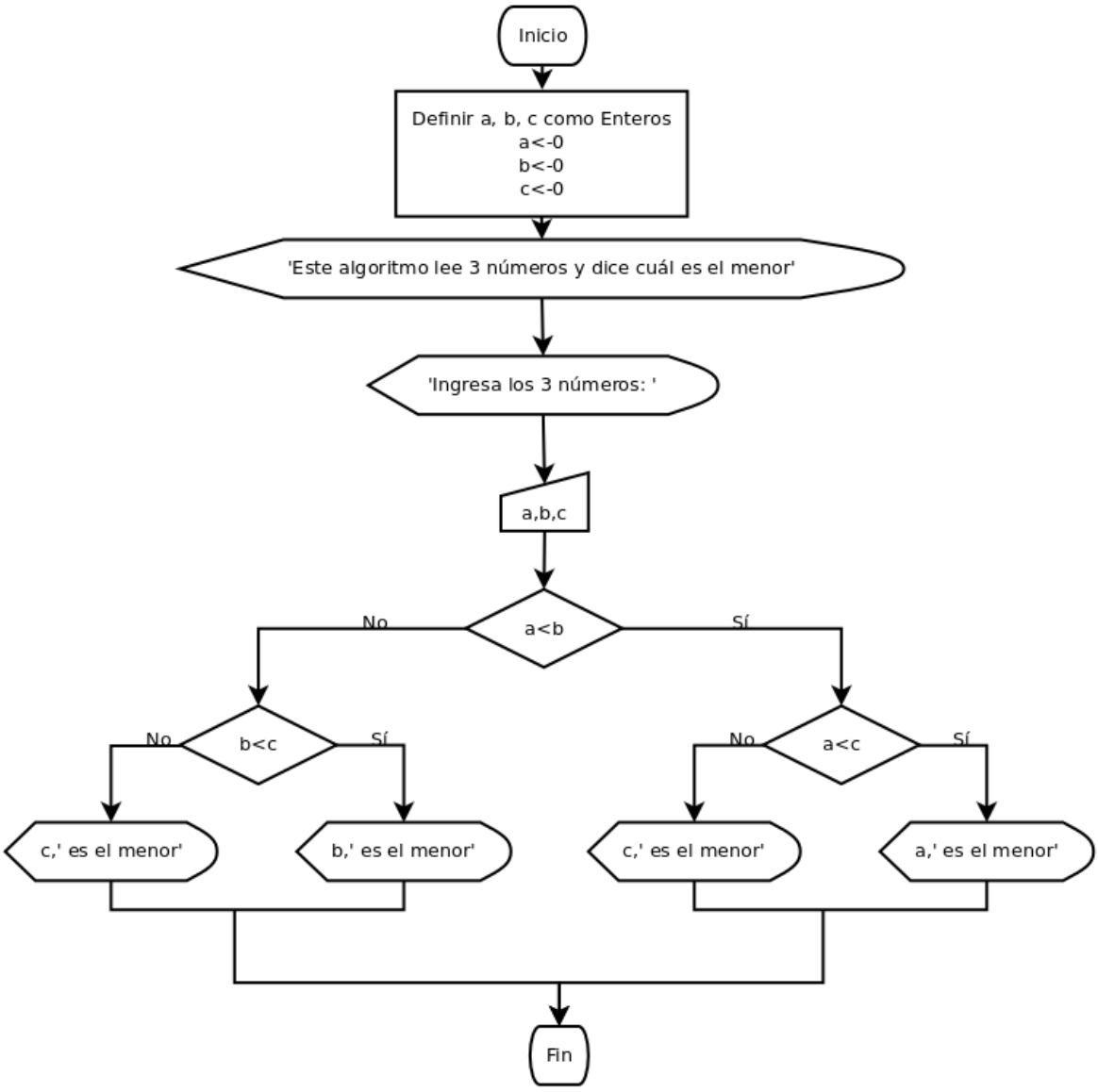


Figura 1. *Diagrama de flujo para el menor de 3 números en Dia.*

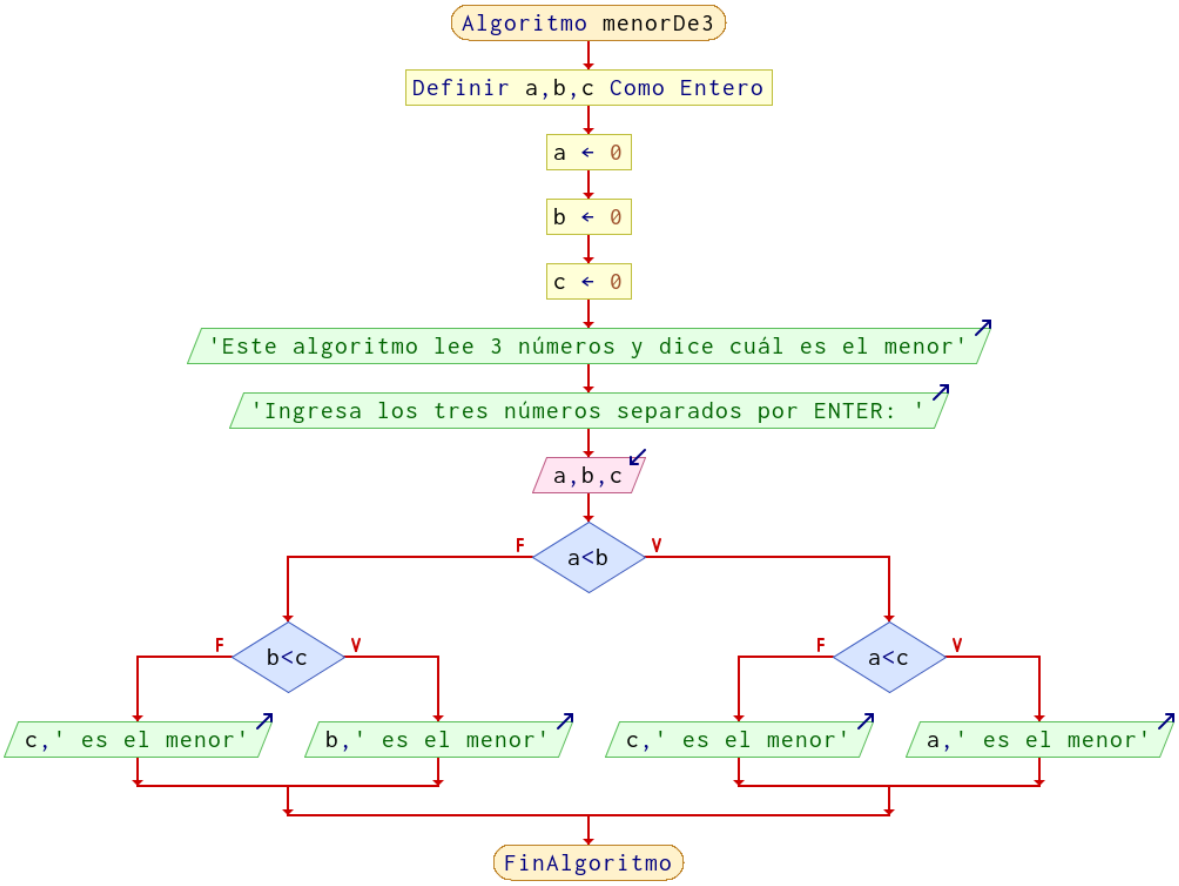


Figura 2. *Diagrama de flujo para el menor de 3 números en PSeInt.*

7. Prueba de escritorio

Iteración	Entrada(s)	Salidas(s)
1	a= 1, b= 2, c= 3	1 es el menor
2	a= 3, b= 2, c= 1	1 es el menor
3	a= 1, b= 3, c= 2	1 es el menor

Estructuras de repetición

Los ciclos o estructuras de repetición permiten repetir una tarea o conjunto de instrucciones de acuerdo a cierto número de veces o mientras que una condición se cumpla.

La siguiente tabla contiene las palabras reservadas elegidas para escribir pseudocódigo y su símbolo equivalente para diagrama de flujo para la estructura de repetición más simple.

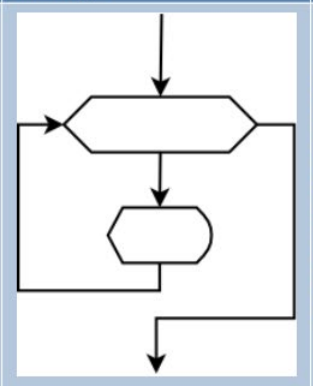
Acción de pseudocódigo	Símbolo de diagrama de flujo	Descripción
DEFINIR conde<-0 como Entero PARA conde<-1 HASTA conde<=10 INCREMENTO<-1 acciones FIN PARA		Se usa cuando los 3 elementos son conocidos y fijos todo el ciclo

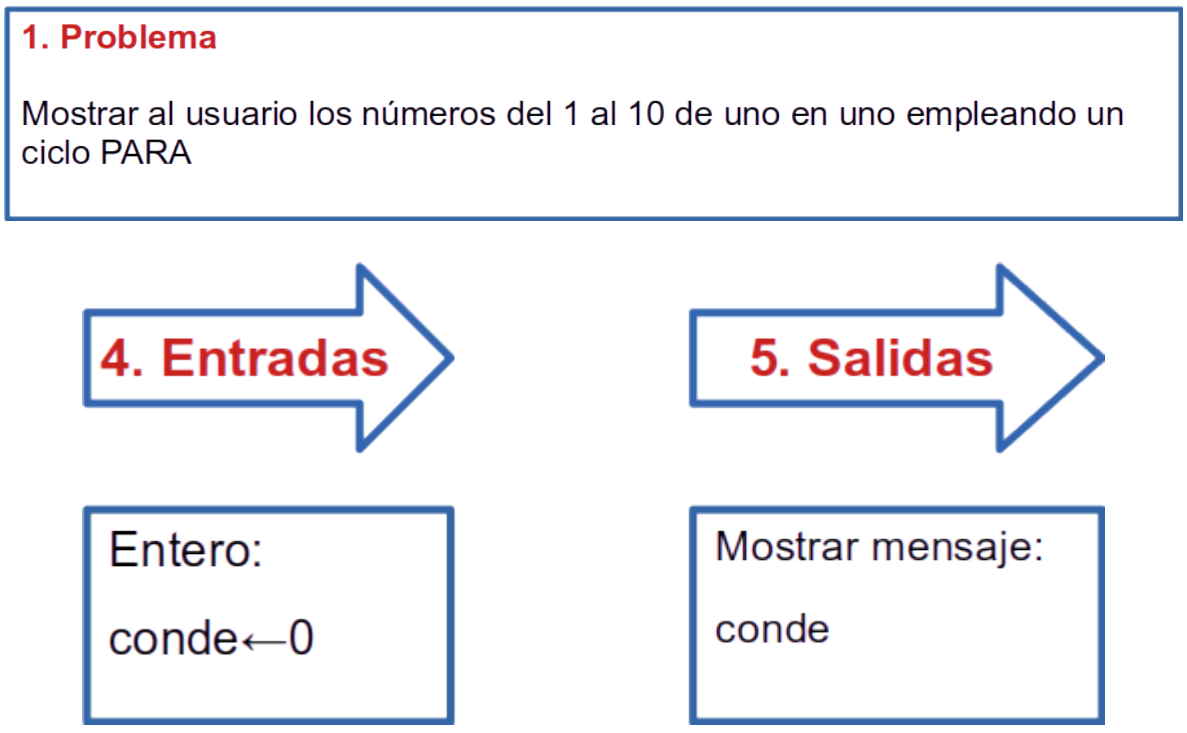
Tabla 5. *Equivalencia de acciones en pseudocódigo y diagrama de flujo para la estructura PARA*

Variable auxiliar contador

Variable cuyo incremento o decremento constante controla la cantidad de veces que se repite un proceso o cálculo. Debe ser declarada como un tipo de dato numérico entero y de preferencia con algún nombre relacionado aunque es común emplear las letras i, j, k, l, m, n por simplicidad.

Ejemplo 2

Realiza las etapas de análisis (sólo problema, entradas y salidas) y diseño de un algoritmo para el siguiente problema:



Proceso en pseudocódigo con palabras reservadas:

```
ETAPA 6. PROCESO

INICIO

1. DEFINIR conde<-0 como Entero

2. ESCRIBIR 'Este algoritmo muestra del 1 al 10 usando un ciclo Para'
3. PARA conde<-1 HASTA conde<=10 INCREMENTO<-1
    3.1 ESCRIBIR conde
4. FIN PARA

FIN
```

Proceso en pseudocódigo con la herramienta PSeInt:

```
1  Algoritmo para1A10
2      Definir conde Como Entero
3      conde ← 0
4      Escribir 'Este algoritmo muestra del 1 al 10 usando un ciclo Para'
5      Para conde←1 Hasta 10 Hacer
6          Escribir conde
7      FinPara
8  FinAlgoritmo
```

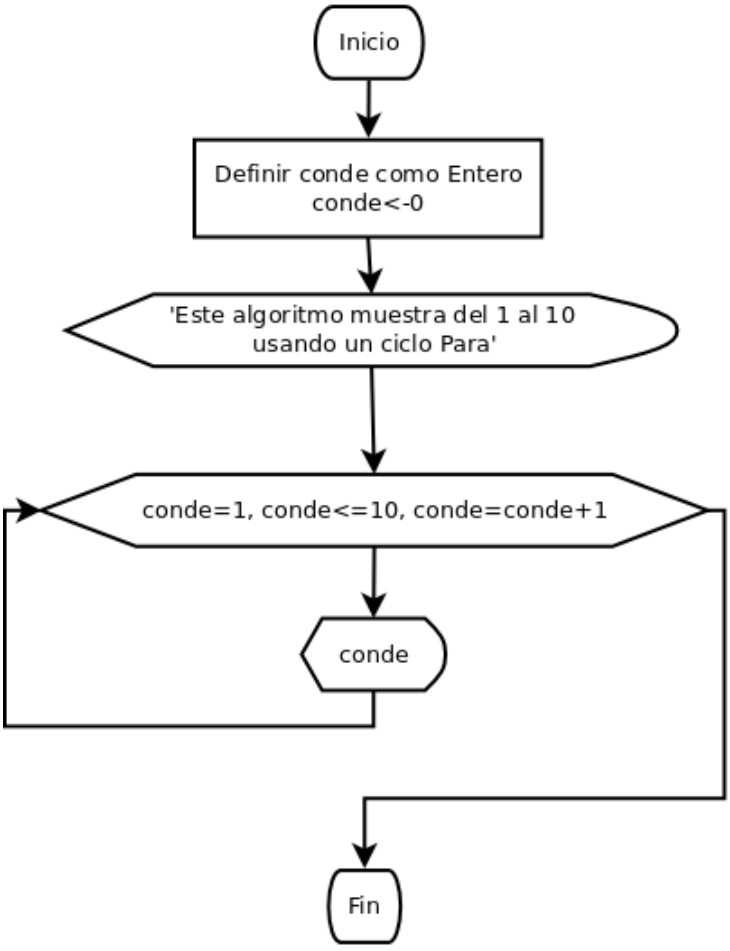


***Figura 3**. Diagrama de flujo para mostrar del 1 al 10 en Dia*

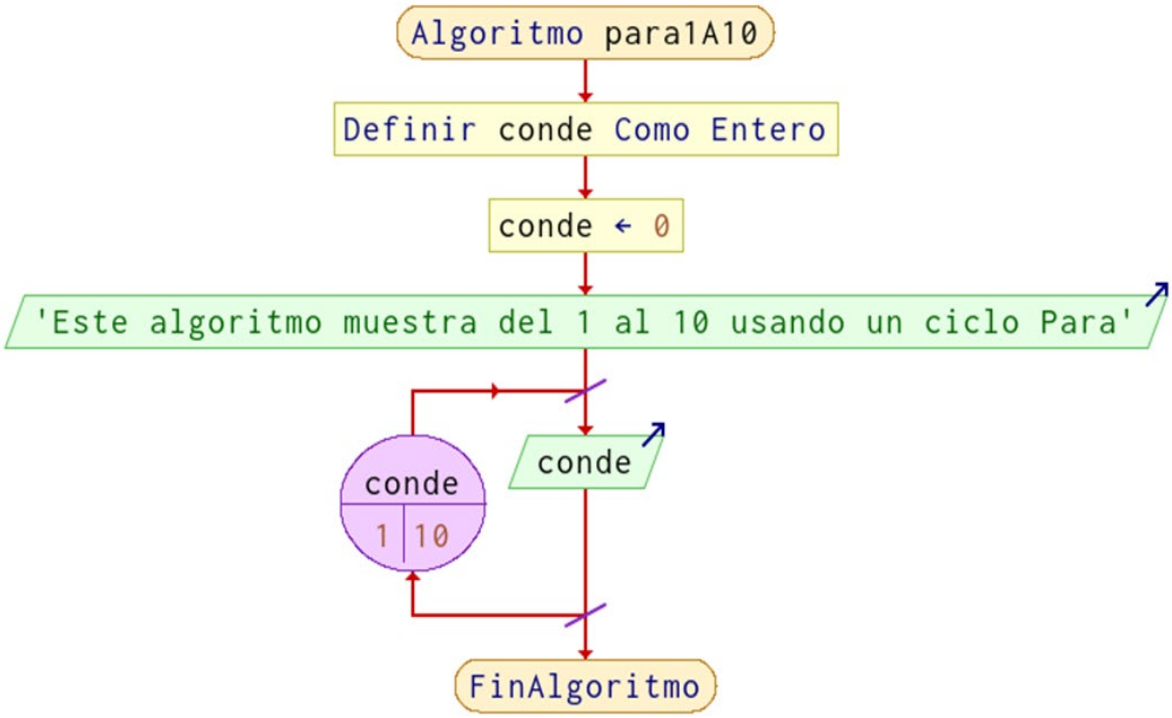


Figura 4. *Diagrama de flujo para mostrar del 1 al 10 en PSeInt*

Bibliografía

Cairó, O. (2005). Metodología de la programación: Algoritmos, diagramas de flujo y programas. (3a. ed.). Alfaomega.

Corona, M. A. y Ancona, M. A. (2011). Diseño de algoritmos y su codificación en lenguaje C. Mc Graw Hill.

Guía práctica de estudio 05: Diseño de algoritmos avanzados

Elaborado por:	Revisado por:	Autorizado por:
Ing. Jorge Luis López García Ing. Mayelly Reynoso Andrade	M. C. Laura Sandoval Montaño	Dra. Rocío Alejandra Aldeco Pérez Ing. Luis Sergio Valencia Castro

Guía práctica de estudio 05: Diseño de algoritmos avanzados

Objetivo:

El alumnado diseñará algoritmos gráficos y no gráficos a partir de un análisis previo para resolver problemas que involucren instrucciones que busquen la mejora y optimización del proceso.

Actividades:

1. Definir un problema, identificar datos de entrada y salida, diseñar un proceso en pseudocódigo y en diagrama de flujo para un ejemplo proporcionado por el profesor.

2. Realizar la etapa de diseño de algoritmos para ejercicios simples de física, matemáticas y áreas de ingeniería en forma de diagrama de flujo y pseudocódigo involucrando instrucciones en secuencia, decisión y repetición optimizando el algoritmo con la implementación de opciones, así como menús que se repiten, validaciones de datos.

Introducción:

En la práctica anterior se abordaron las instrucciones de selección simple, doble selección y decisiones anidadas que permiten validar condiciones y de acuerdo a su veracidad tomar un sentido u otro en el algoritmo.

A continuación, se muestra la estructura de selección múltiple que brinda al usuario la oportunidad de elegir una operación o solución para una problemática específica mostrada como una opción en un menú.

Acción de pseudocódigo	Símbolo de diagrama de flujo	Descripción

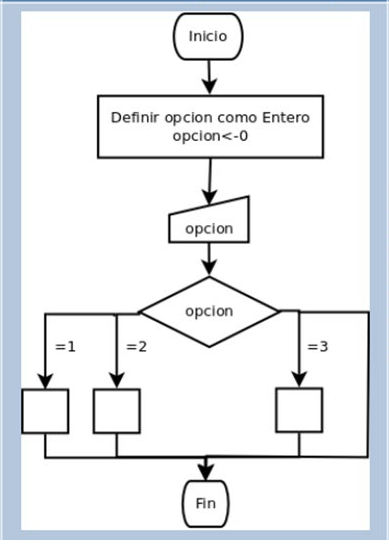
`SEGÚN_SEA(opcion)`
`Inicio`
`  Caso 1:`
`  acciones`
`  salir`
`  Caso 2:`
`  acciones`
`  salir`
`  Caso 3:`
`  acciones`
`  salir`
`  Caso contrario:`
`  acciones`
`  salir`
`Fin`
		De acuerdo a la opción ingresada el flujo del algoritmo se dirigirá a la condición que se haya cumplido.

Tabla 1. *Equivalencia de acciones en pseudocódigo y diagrama de flujo para selección múltiple.*

Ejemplo 1

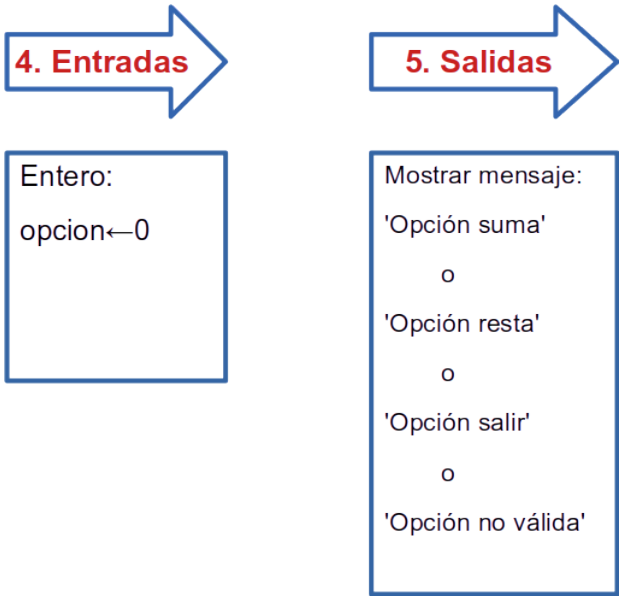
Realiza la etapa de análisis (problema, entradas y salidas) y diseño de un algoritmo para el siguiente problema:

1. Problema

Implementar un menú con la instrucción de selección múltiple (SEGÚN_SEA o SEGÚN) que muestre y lea del usuario las opciones:

1. Suma
2. Resta
3. Salir

Al ingresar a la opción elegida el algoritmo únicamente mostrará un mensaje indicando la opción con el fin de comprobar que el menú se construyera correctamente.



Proceso en pseudocódigo con palabras reservadas:

```
ETAPA 6. PROCESO

INICIO

1. DEFINIR opcion<-0 como Entero
2. ESCRIBIR 'Este algoritmo implementa menú con selección múltiple'
3. ESCRIBIR '1.Suma 2.Resta 3.Salir. Ingresa tu opción: '
4. LEER y ALMACENAR EN opcion
5. SEGÚN_SEA(opcion)
    Inicio
        5.1 Caso 1:
            5.1.1 ESCRIBIR 'Opción suma'
            5.1.2 Salir
        5.2 Caso 2:
            5.2.1 ESCRIBIR 'Opción resta'
            5.2.2 Salir
        5.3 Caso 3:
            5.3.1 ESCRIBIR 'Opción salir'
            5.3.2 Salir
        5.4 Caso contrario
            5.4.1 ESCRIBIR 'Opción no válida'
            5.4.2 Salir
    Fin

FIN
```

Proceso en pseudocódigo con la herramienta PSeInt:

```
1  Algoritmo menu
2      Definir opcio Como Entero
3      opcio ← 0
4      Escribir 'Este algoritmo implementa un menú con selección múltiple'
5      Escribir '1.Suma 2.Resta 3.Salir. Ingresa tu opción: '
6      Leer opcio
7      Segun opcio  Hacer
8          1:
9              Escribir 'Opción suma '
10         2:
11             Escribir 'Opción resta'
12         3:
13             Escribir 'Opción salir'
14         De Otro Modo:
15             Escribir 'Opción no válida'
16     FinSegun
17 FinAlgoritmo
```

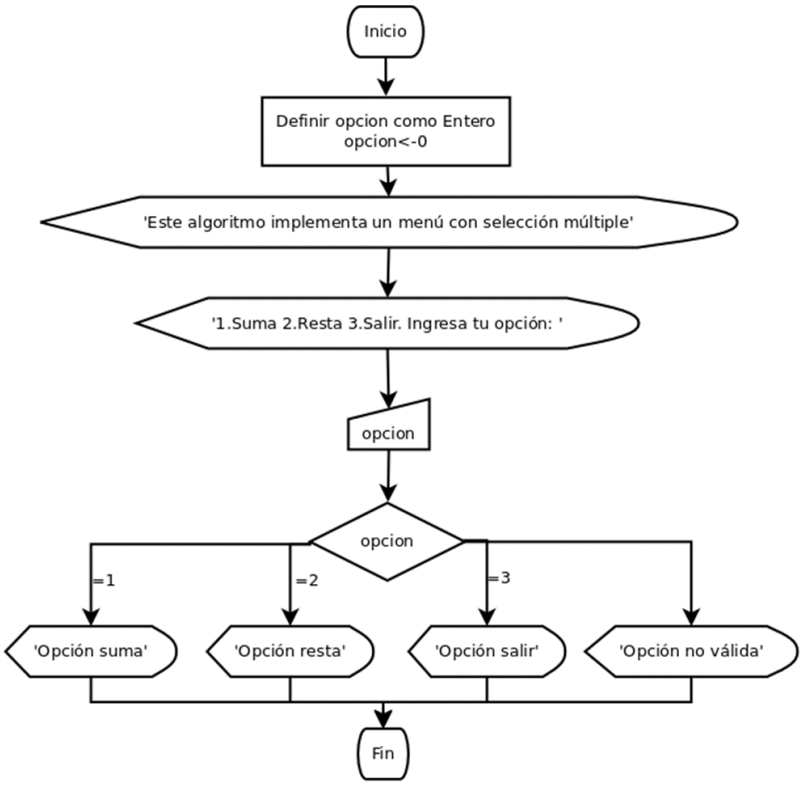


Figura 1 *Diagrama de flujo para mostrar menú de 3 opciones en Dia.*

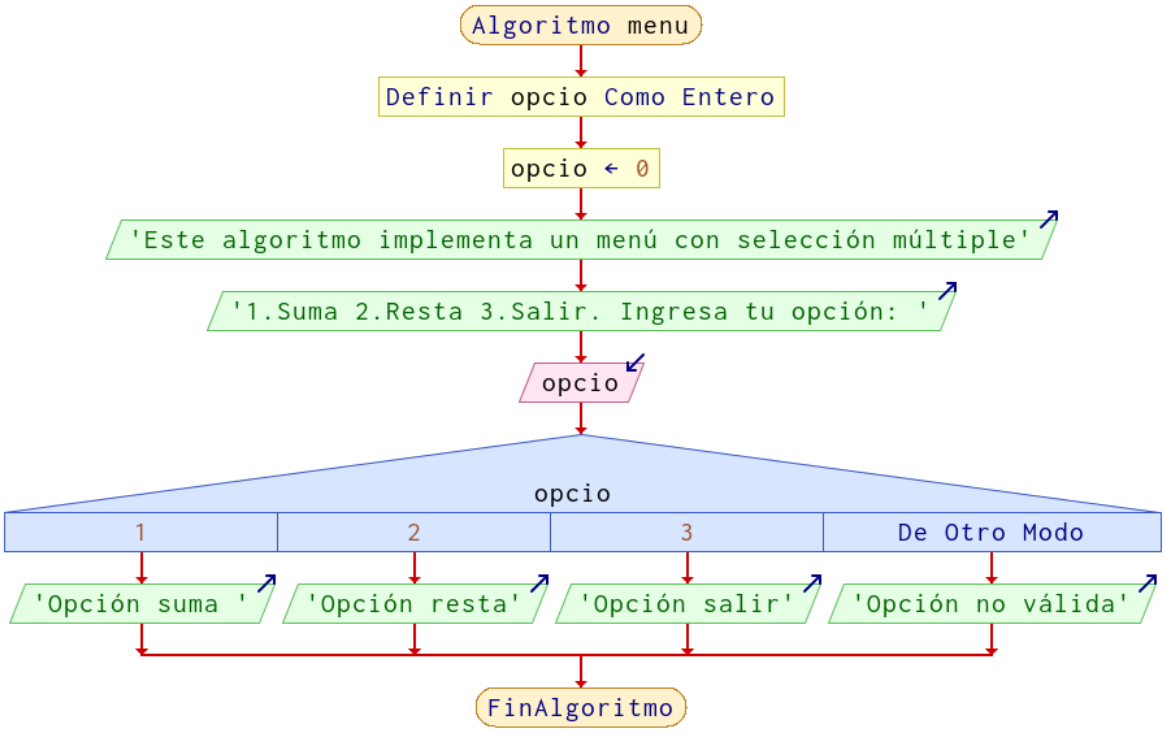


Figura 2. *Diagrama de flujo para mostrar menú de 3 opciones en PSeInt*

Como se planteó en la práctica anterior la estructura de repetición más simple es el ciclo PARA, por contar con el inicio, la condición y el incremento o decremento sobre la misma instrucción. Además del anterior existen dos estructuras más que se observan en las siguientes tablas:

Acción de pseudocódigo	Símbolo de diagrama de flujo	Descripción

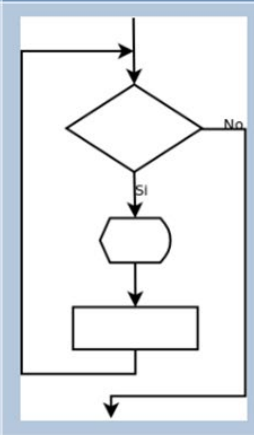
`DEFINIR conde<-1 como Entero`
 	
`MIENTRAS QUE conde<=10`
`  acciones`
`FIN MIENTRAS QUE`
		Se emplea cuando alguno de los elementos cambiará durante el ciclo como la condición sin una cantidad fija de repeticiones

Tabla 2. *Equivalencia de acciones en pseudocódigo y diagrama de flujo para la estructura MIENTRAS QUE.*

Acción de pseudocódigo	Símbolo de diagrama de flujo	Descripción

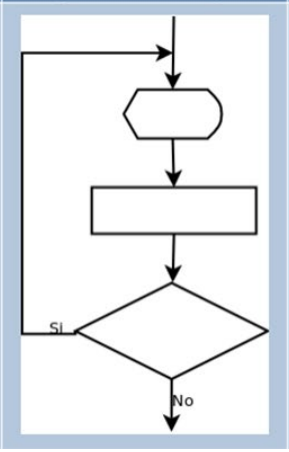
`DEFINIR conde<-1 como Entero`
 	
`HACER`
`  acciones`
`FIN HACER`
 	
`MIENTRAS QUE conde<=10`
		Siempre realiza por lo menos una vez el bloque de instrucciones agrupados en HACER se cumpla o no la condición que se evalúa en MIENTRAS

Tabla 3*. Equivalencia de acciones en pseudocódigo y diagrama de flujo para la estructura HACER-MIENTRAS QUE.*

Ejemplo 2

Realiza las etapas de análisis (problema, entradas y salidas) y diseño de un algoritmo para el siguiente problema:

1. Problema

Mostrar al usuario los números del 1 al 10 de uno en uno empleando un ciclo MIENTRAS QUE

4. Entradas

Entero:

conde←0

5. Salidas

Mostrar mensaje:

conde

Proceso en pseudocódigo con palabras reservadas:

```
ETAPA 6. PROCESO

INICIO

1. DEFINIR conde<-1 como Entero

2. ESCRIBIR 'Este algoritmo muestra del 1 al 10 usando un ciclo Mientras que'
3. MIENTRAS QUE conde<=10
    3.1 ESCRIBIR conde
    3.2 CALCULAR conde<- conde+1
4. FIN MIENTRAS QUE

FIN
```

Proceso en pseudocódigo con la herramienta PSeInt:

```
1  Algoritmo mientras1A10
2      Definir conde Como Entero
3      conde ← 1
4      Escribir 'Este algoritmo muestra del 1 al 10 usando un ciclo Mientras que'
5      Mientras conde≤10 Hacer
6          Escribir conde
7          conde ← conde+1
8      FinMientras
9  FinAlgoritmo
```

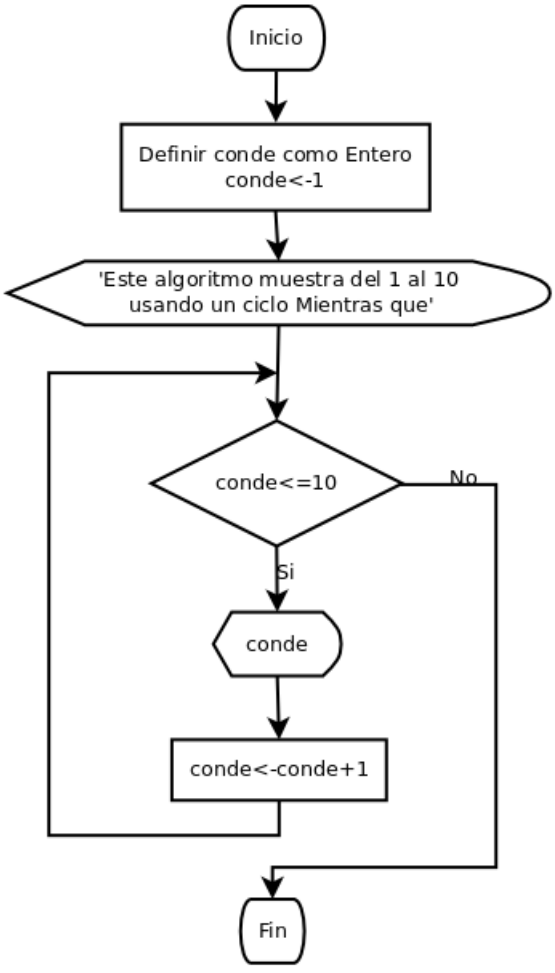


***Figura 3**. Diagrama de flujo para mostrar del 1 al 10 en Dia.*

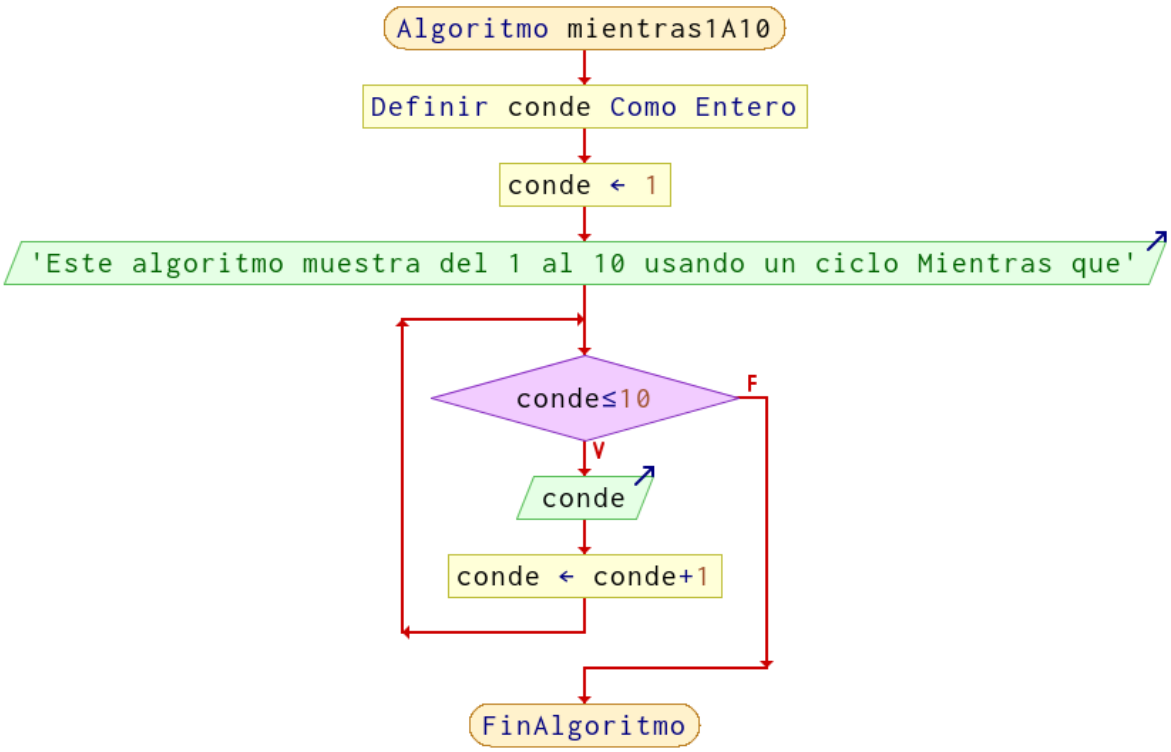


Figura 4. *Diagrama de flujo para mostrar del 1 al 10 en PSeInt.*

Ejemplo 3

Realiza la etapa de análisis (problema, entradas y salidas) y diseño de un algoritmo para el siguiente problema:

1. Problema

Mostrar al usuario los números del 1 al 10 de uno en uno empleando un ciclo HACER – MIENTRAS QUE

4. Entradas

Entero:

conde←0

5. Salidas

Mostrar mensaje:

conde

Proceso en pseudocódigo con palabras reservadas:

```
ETAPA 6. PROCESO

INICIO

1. DEFINIR conde<-1 como Entero

2. ESCRIBIR 'Este algoritmo muestra del 1 al 10 usando un ciclo Hacer-Mientras que'
3. HACER
    3.1 ESCRIBIR conde
    3.2 CALCULAR conde<- conde+1
4. FIN HACER
5. MIENTRAS QUE conde<=10

FIN
```

Proceso en pseudocódigo con la herramienta PSeInt:

```
1  Algoritmo hacerMientras1A10
2      Definir conde Como Entero
3      conde ← 1
4      Escribir 'Este algoritmo muestra del 1 al 10 usando un ciclo Hacer-Mientras que'
5      Repetir
6          Escribir conde
7          conde ← conde+1
8      Mientras Que conde≤10
9  FinAlgoritmo
```

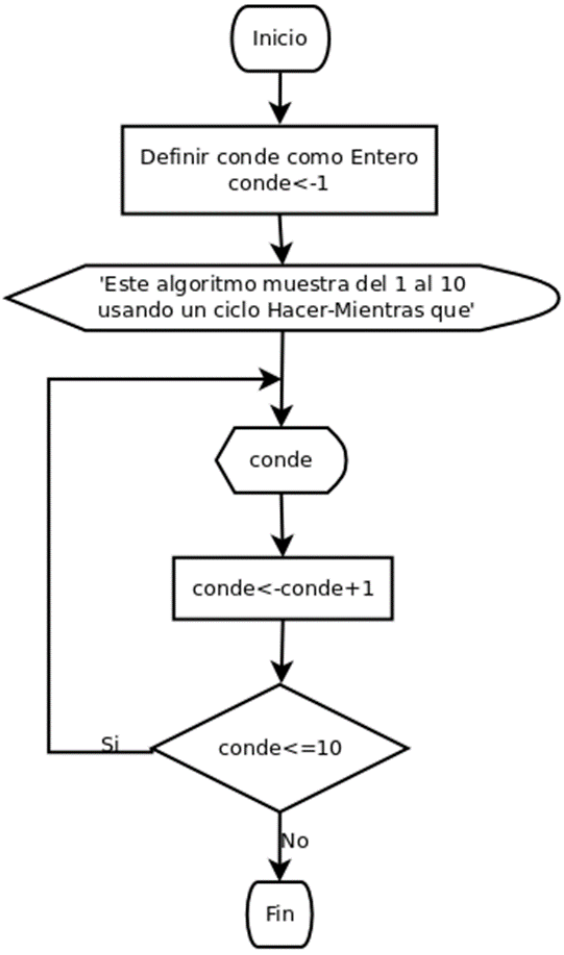


Figura 5. *Diagrama de flujo para mostrar del 1 al 10 con ciclo Hacer – Mientras Que en Dia*

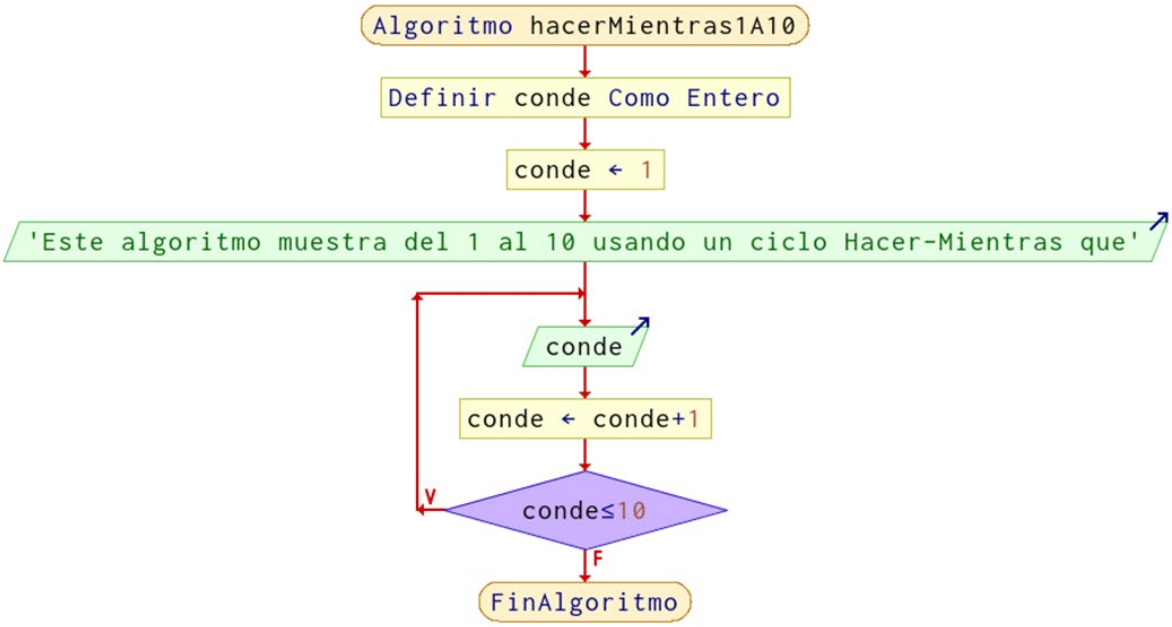


Figura 6. *Diagrama de flujo para mostrar del 1 al 10 con ciclo Hacer – Mientras Que en PSeInt*

Ejemplo 4

Realiza la etapa de diseño de un algoritmo para un menú de 3 opciones y que además pregunte al usuario si desea realizar otra operación.

Proceso en pseudocódigo con palabras reservadas:

```
ETAPA 6. PROCESO

INICIO

1. DEFINIR opcion<-0 como Entero
2. DEFINIR repite<-'' como Carácter
3. ESCRIBIR 'Este algoritmo implementa menú con selección múltiple'
4. ESCRIBIR '1.Suma 2.Resta 3.Salir. Ingresa tu opción: '
5. LEER y ALMACENAR EN opcion
6. HACER
     6.1 SEGÚN_SEA(opcion)
          Inicio
               6.1.1 Caso 1:
                    6.1.1.1 ESCRIBIR 'Opción suma'
                    6.1.1.2 Salir
               6.1.2 Caso 2:
                    6.1.2.1 ESCRIBIR 'Opción resta'
                    6.1.2.2 Salir
               6.1.3 Caso 3:
                    6.1.3.1 ESCRIBIR 'Opción salir'
                    6.1.3.2 Salir
               6.1.4 Caso contrario
                    6.1.4.1 ESCRIBIR 'Opción no válida'
                    6.1.4.2 Salir
          Fin
     6.2 ESCRIBIR 'Desea realizar otra operación (s/n): '
     6.3 LEER y ALMACENAR EN repite
7. FIN HACER
8. MIENTRAS QUE repite='s' O repite='S'

FIN
```

Proceso en pseudocódigo con la herramienta PSeInt:

```
Algoritmo menuNumRepite
    Definir opcio Como Entero
    Definir repite Como Caracter
    opcio← 0
    repite← ''

    Repetir
        Escribir 'Este algoritmo implementa un menú con selección múltiple'
        Escribir '1.Suma 2.Resta 3.Salir. Ingresa tu opción: '
        Leer opcio
        Segun opcio  Hacer
            1:
                Escribir 'Opción suma '
            2:
                Escribir 'Opción resta'
            3:
                Escribir 'Opción salir'
            De Otro Modo:
                Escribir 'Opción no válida'
        FinSegun
        Escribir 'Desea realizar otra operación (s/n): '
        Leer repite
    Mientras Que(repite='s' O repite='S')

FinAlgoritmo
```

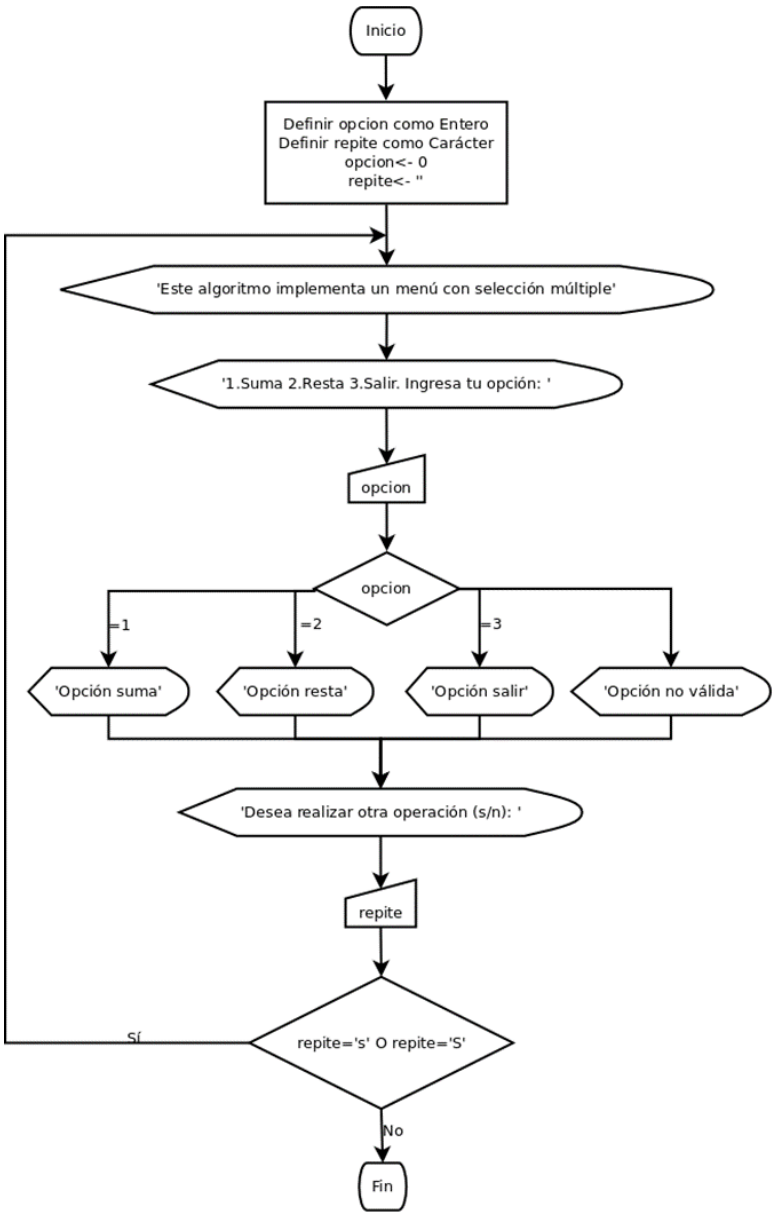


Figura 7. *Diagrama de flujo con menú que se repite en Dia*

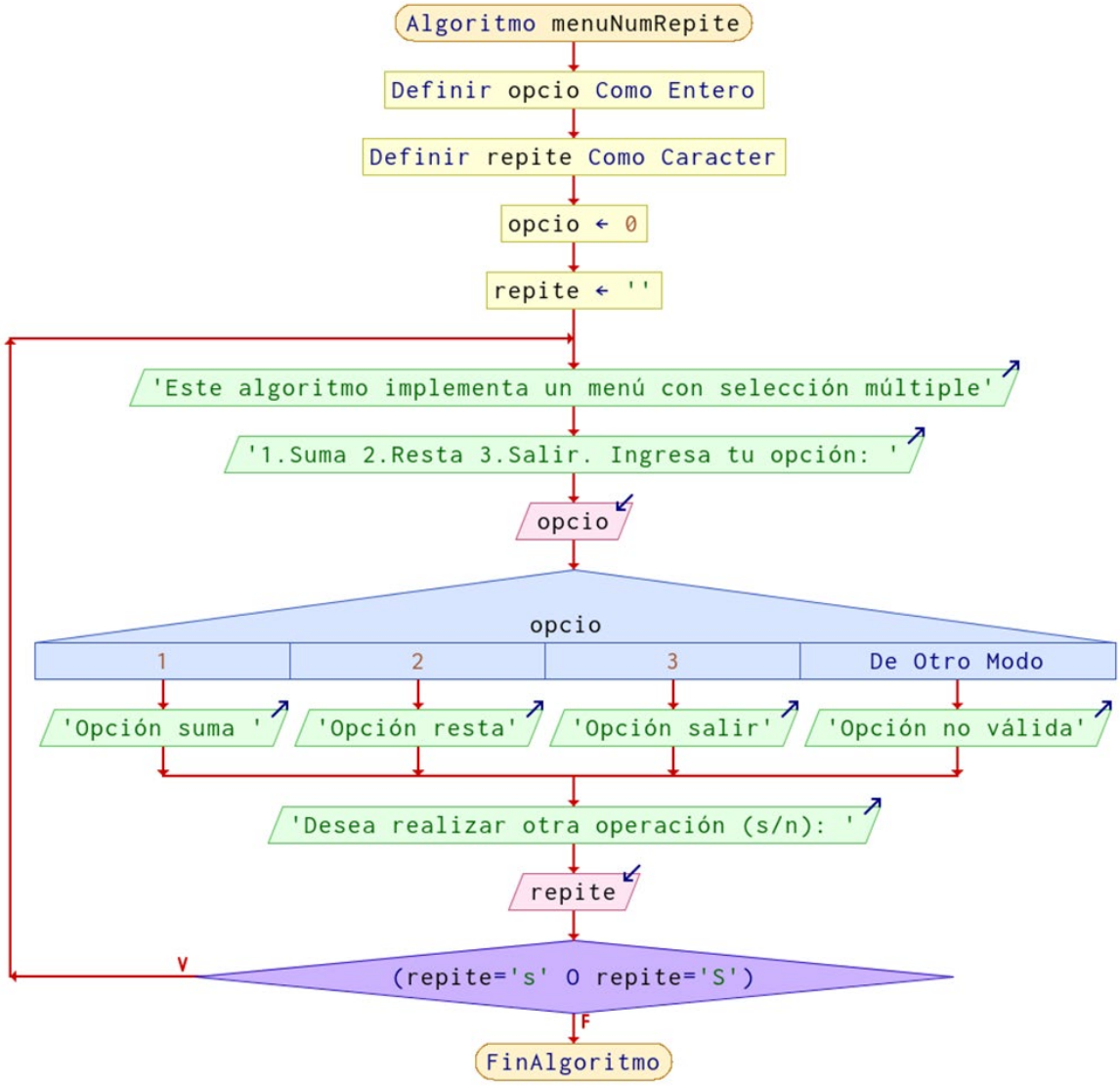


Figura 8. *Diagrama de flujo con menú que se repite en PSeInt*

Continuando con los usos de las instrucciones de repetición, además de ayudar a repetir un menú también son útiles para validar datos ingresados por el usuario para que cumplan cierta condición,

con la ventaja sobre las instrucciones de selección de que el ciclo no permitirá el avance del algoritmo hasta que se ingrese el dato con el formato solicitado.

Ejemplo 5

1. Problema

Validar que un dato proporcionado por el usuario cumpla con cierto formato, por ejemplo cuando desea ingresa el valor del radio de un círculo y éste debe ser positivo para evitar fallas en la lógica del algoritmo y en la impresión de su valor. El algoritmo no avanzará hasta que se proporcione un valor que cumpla con la condición sin importar la cantidad de veces que se haga incorrectamente.

4. Entradas

Real:

radio←0

5. Salidas

Mostrar mensaje:

'El valor del radio es ',radio

Proceso en pseudocódigo con palabras reservadas:

```
ETAPA 6. PROCESO

INICIO

1. DEFINIR radio<-0 como Real

2. ESCRIBIR 'Este algoritmo valida que el radio dado por el usuario sea
   positivo, de lo contrario repite la petición'
3. MIENTRAS QUE radio<=0
    3.1 ESCRIBIR 'Ingresa el valor del radio positivo: '
    3.2 LEER radio
4. FIN MIENTRAS QUE
5. ESCRIBIR 'El valor del radio es ',radio

FIN
```

Proceso en pseudocódigo con la herramienta PSeInt:

```
1   Algoritmo validaMientras
2       Definir radio Como Real
3       radio ← 0
4
5       Escribir 'Este algoritmo valida que el radio dado por el' Sin Saltar
6       Escribir ' usuario sea positivo, de lo contrario repite la petición'
7
8       Mientras radio≤0 Hacer
9           Escribir 'Ingresa el valor del radio positivo: ' Sin Saltar
10          Leer radio
11      FinMientras
12      Escribir 'El valor del radio es ',radio
13  FinAlgoritmo
```

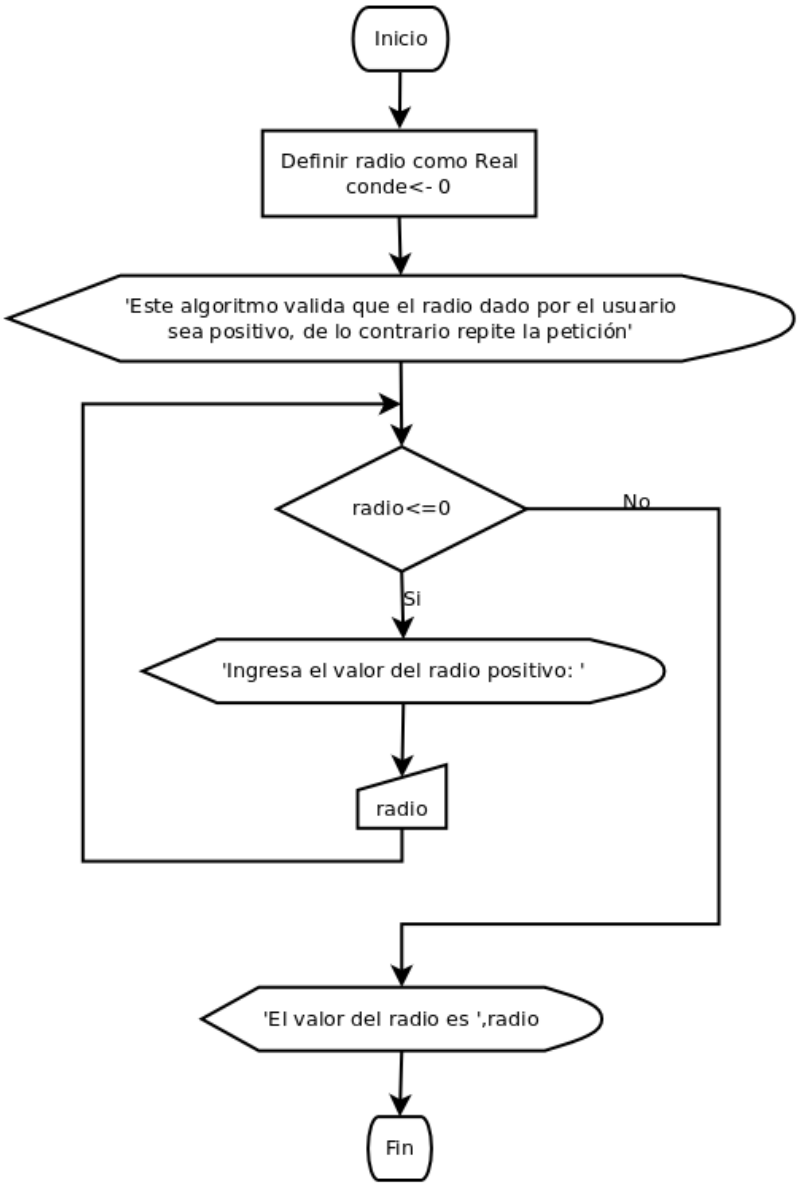


Figura 9. *Diagrama de flujo para validar el radio en Dia*

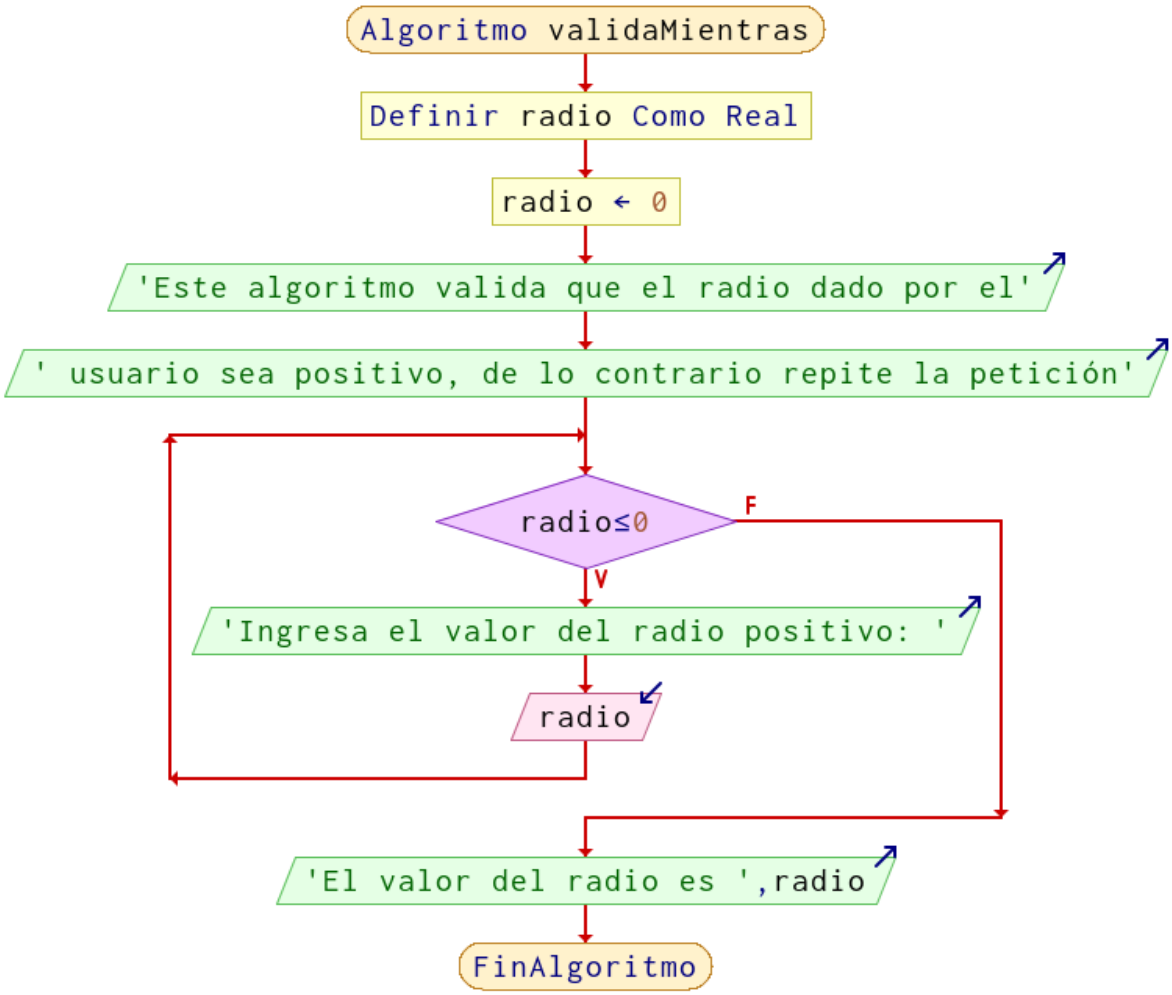
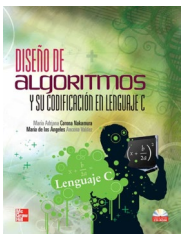


Figura 10. *Diagrama de flujo para validar el radio en PSeInt*

Bibliografía

Cairó, O. (2005). Metodología de la programación: Algoritmos, diagramas de flujo y programas. (3a. ed.). Alfaomega.

Corona, M. A. y Ancona, M. A. (2011). Diseño de algoritmos y su codificación en lenguaje C. Mc Graw Hill.



Guía práctica de estudio 06: Entornos y fundamentos de programación en lenguaje FORTRAN

Elaborado por:	Revisado por:	Autorizado por:
Ing. Jorge Luis López García Ing. Mayelly Reynoso Andrade	M. C. Laura Sandoval Montaño	Dra. Rocío Alejandra Aldeco Pérez Ing. Luis Sergio Valencia Castro

Guía práctica de estudio 06: Entornos y fundamentos de programación en lenguaje FORTRAN

Objetivo:

El alumnado elaborará programas en lenguaje FORTRAN desde diferentes entornos y herramientas que incluyan la declaración de variables, lectura de datos, operaciones e impresión de expresiones.

Actividades:

1. Realizar un algoritmo en pseudocódigo o diagrama de flujo que inicialice o lea del teclado diferentes tipos de datos y los muestre en la salida estándar.

2. Codificar en lenguaje FORTRAN un programa que sea equivalente al algoritmo del punto anterior en un editor de texto plano.

3. Abrir una terminal del Sistema Operativo empleado, desde ahí compilar y ejecutar el programa.

4. Abrir el código fuente en un IDE instalado localmente en el equipo, modificarlo, compilarlo y ejecutarlo.

5. Opcionalmente codificar su programa en un compilador en línea y realizar el mismo proceso.

Introducción:

FORmula TRANslator o traductor de fórmulas fue el primer lenguaje de alto nivel desarrollado para escribir programas de cómputo científico, es más rápido que otros lenguajes al no tener que incluir o importar bibliotecas para operaciones matemáticas.

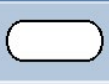
Con el avance de las versiones tiene 2 formatos:

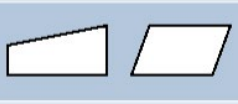
Formato Fijo

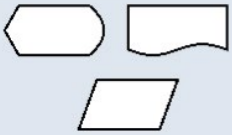
- Aplicable para los programas escritos en versiones anteriores a la 90.
- Guardados con extensión .f y .for.
- Máximo 72 caracteres por renglón incluyendo espacios.
- La línea que comience con C se considera comentario.
- Máximo una instrucción por renglón.
- La instrucción comienza en la séptima columna y las primeras 6 deben ser espacios.
- Las primeras 5 columnas están destinadas para etiquetas numéricas enteras positivas de máximo 5 dígitos.

Formato Flexible

- Aplicable para programas escritos en versiones a partir de la 90.
- Están guardados con la extensión .f90 o .f95.
- Máximo 132 caracteres por renglón incluyendo espacios.
- ! comienza un comentario de un renglón o del resto de la línea.
- ; separa 2 instrucciones en la misma línea.
- En caso de falta de espacio para terminar una instrucción en una línea se puede usar “&” al final de la primera y al inicio de la siguiente.

Acción de pseudocódigo	Símbolo de diagrama de flujo	Lenguaje FORTRAN
INICIO / ALGORITMO		`PROGRAM nombre`
ASIGNAR / DECLARAR / DEFINIR		`INTEGER::x` `REAL::y`

LEER Y GUARDAR EN / LEER Y ALMACENAR EN		`READ*, x`
		`READ(*,*) y`
		`READ*, x,y`

IMPRIMIR / ESCRIBIR / MOSTRAR MENSAJE		`PRINT*, 'Hola Mundo'`
		`WRITE(*,*) 'Hola Mundo'`
		`PRINT*, 'La suma es: ',x`

CALCULAR / REALIZAR OPERACIÓN		`z= x + y` `suma= suma + conde`
FIN / FIN ALGORITMO		`END PROGRAM [nombre]`

Tabla 1. *Equivalencia de acciones en pseudocódigo, diagrama de flujo y lenguaje FORTRAN.*

Operación	Símbolo en FORTRAN	Ejemplo de uso
SUMA	+	`a+b`
RESTA	-	`a-b`
MULTIPLICACIÓN	*	`a*b  No usar: 2a  ni  2(a)`
DIVISIÓN	/	`a/b`
POTENCIA	**	`x**2  eleva x al cuadrado`

Tabla 2. *Operadores y funciones matemáticas en lenguaje FORTRAN.*

Operación	Función en FORTRAN	Operando	Salida
VALOR ABSOLUTO	**ABS(x)**	Entero, real o complejo	El mismo
RAÍZ CUADRADA	**SQRT(x)**	Real o complejo	El mismo
MÓDULO	**MOD(x,y)**	Entero o real	El mismo
PARTE ENTERA DE UNA REAL	**INT(x)**	Real	Entero
CONVERSIÓN A REAL	**REAL(x)**	Entero	Real
REDONDEO AL ENTERO MÁS PRÓXIMO	**NINT(x)**	Real	Entero

Tabla 3. *Operadores y funciones matemáticas en lenguaje FORTRAN.*

Operación	Función en FORTRAN	Operando	Salida
SENO	**SIN(x)**	Real o complejo	El mismo
COSENO	**COS(x)**	Real o complejo	El mismo
TANGENTE	**TAN(x)**	Real o complejo	El mismo
EXPONENCIAL	**EXP(x)**	Real o complejo	El mismo
LOGARITMO NATURAL	**LOG(x)**	Real o complejo	El mismo

Tabla 4. *Operadores y funciones matemáticas en lenguaje FORTRAN.*

Operación	Función en FORTRAN	Operando	Salida
PSEUDO ALEATORIO DE 0 A 1	`RANDOM_NUMBER(x)`	`Real`	`El mismo`
MÁXIMO DE x1, x2, … xn	`MAX(x1,…,xn)`	`Entero o Real`	`El mismo`
MÍNIMO DE x1, x2, … xn	`MIN(x1,…,xn)`	`Entero o Real`	`El mismo`

Tabla 5. *Operadores y funciones matemáticas en lenguaje FORTRAN.*

Tipo	Bytes	Rango/uso

`INTEGER variable`
`INTEGER::variable`
`INTEGER(KIND=4)::var`
	4	**-2,147,483,648 <= X <= 2,147,483,647**
		Números enteros grandes y control de ciclos

`REAL variable`
`REAL*4 variable`
`REAL::variable`
`REAL*4::variable`
`REAL(KIND=4)::var`
	4	**1.18e-38 <= |X| <= 3.40e38**
		Precisión científica (6-7 dígitos significativos)

`DOUBLE PRECISION var`
`REAL*8 variable`
`REAL*8::variable`
`REAL(KIND=8)::var`
	8	**2.23e-308 <= |X| <= 1.79e308**
		Precisión científica (15-16 dígitos significativos)

Tabla 6. *Rangos de tipos de datos.*

Tipo	Bytes	Rango/uso

`COMPLEX variable`
`COMPLEX*8 variable`
`COMPLEX(KIND=8)::var`
	8	Complejo de simple precisión
		El primer elemento es la parte real y después la parte imaginaria
		Cada parte ocupa 4 bytes

`CHARACTER variable` `CHARACTER*1 variable`	1	Si no se especifica la longitud corresponde a sólo un carácter. Almacena ASCII

`CHARACTER*n variable`
`CHARACTER(LEN=n)::var`
`CHARACTER(n)::var`
	1xn	El tamaño es de 1 byte por cada carácter que forme la cadena. Almacena ASCII

`LOGICAL var`	4	`.TRUE.` o `.FALSE.`

Tabla 7. *Rangos de tipos de datos.*

Licencia Pública General de GNU

El software contenido en esta práctica es libre bajo la GNU GPL con lo cual el alumnado está en libertad de usar, estudiar, compartir y modificar el software mientras se mantenga la licencia.

```
!Descripción general del programa
!Copyright 2023 Jorge Luis López
!
!This program is free software: you can redistribute it and/or modify it under the terms of the GNU
! General Public License as published by the Free Software Foundation, either version 3 of the License, or
! (at your option) any later version.
!
!This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without
! even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See
! the GNU General Public License for more details.
!
!You should have received a copy of the GNU General Public License along with this program. If not, see
! <https://www.gnu.org/licenses/>.
```

Ejemplo 1 – holaMundo.f95

```
PROGRAM holaMundo

    !no declare en automático i-n como enteros y el resto como reales
    IMPLICIT NONE

    PRINT*, 'Hola mundo 1'      !impresión de mensaje primera forma
    WRITE(*,*) 'Hola mundo 2'   !impresión de mensaje segunda forma

    READ*   !pone en espera la ejecución
END PROGRAM holaMundo
```

El código del programa debe colocarse entre los equivalentes de inicio y fin de algoritmos con lo cual se logra que el programa sea ejecutable y sea el primer bloque de código que busque. Estas palabras reservadas pueden escribirse de cualquiera de las siguientes 3 formas:

```
PROGRAM nombre → END
PROGRAM nombre → END PROGRAM
PROGRAM nombre → END PROGRAM nombre
```

Además de las equivalencias entre algoritmo y lenguaje FORTRAN puede ser necesario la inclusión de líneas de código adicionales como:

• IMPLICIT NONE después de PROGRAM evita que declare en automático como enteras las variables de la "i" a la "n" y el resto como reales.

• Para mantener estática o en espera la ejecución del programa al finalizar y así evitar el cierre de la ventana se emplea la instrucción de lectura (READ*) pero sin una variable de almacenamiento o la instrucción de pausa (PAUSE)

Compilación desde una terminal

La edición, compilación y ejecución puede realizarse directamente desde la terminal usando los editores vi o gedit si se está en un entorno basado en UNIX como Linux.

Escribir código en gedit con extensión **.f95**

```
]$ gedit holaMundo.f95
```

Compilar con gfortran y nombra automáticamente **a.out** el ejecutable

```
]$ gfortran holaMundo.f95
```

Compilar con gfortran y el modificador -o para asignar nombre al ejecutable que puede o no llevar extensión

```
]$ gfortran holaMundo.f95 -o holaMundo.exe
```

Ejecutar por nombre asignado

```
]$ ./holaMundo.exe
```

Si el compilador se instala correctamente en un Sistema Operativo Windows también es posible compilar y ejecutar desde su consola, pero la edición se haría con su propio editor bloc de notas u otro enfocado a programación.

Recomendaciones para nombres de identificadores:

- Su nombre identifica una posición de memoria donde se guardará el dato.
- Iniciar con letra de alfabeto inglés y después puede contener guión bajo (_).
- No debe contener caracteres especiales como $, #, @
- Después de la primera letra puede contener más letras, números o (_).
- No tener espacios como "mi variable".
- NO distingue mayúsculas de minúsculas.
- No usar palabras reservadas del lenguaje como INTEGER, DO, etc.

Antes de realizar la codificación de cualquier proyecto siempre es necesario realizar el análisis y diseño, éste último puede ser como ya se ha visto en pseudocódigo o diagrama de flujo.

Posteriormente encontrar la instrucción equivalente en el lenguaje de programación a cada símbolo del diagrama de flujo o acción del pseudocódigo además de agregar las instrucciones propias que requiera el lenguaje.

Para el siguiente ejemplo se retomará el diseño de un algoritmo desarrollado en prácticas previas para obtener su equivalente en lenguaje FORTRAN y construir su programa.

Proceso en pseudocódigo con palabras reservadas:

```
ETAPA 6. PROCESO

INICIO

1.DEFINIR radio<-0, area<-0, pi<-3.141592 como Reales

2.ESCRIBIR 'Este algoritmo calcula el área de un círculo con radio
  dado por el usuario'
3.ESCRIBIR 'Ingrese el valor del radio positivo: '
4.LEER y ALMACENAR EN radio
5.REALIZAR LA OPERACIÓN radio<-ABS(radio)
6.REALIZAR LA OPERACIÓN area<-pi*radio^2 o area<-pi*radio*radio
7.ESCRIBIR 'El área del círculo con radio ',radio,' es ',area,'
  unidades cuadradas'

FIN
```

Ejemplo 2 – areaCirculo.f95

```
PROGRAM areaCirculo

    !no declare en automático i-n como enteros y el resto reales
    IMPLICIT NONE

    REAL radio, area, pi
    radio= 0.
    area= 0.
    pi= 3.141592

    PRINT*, 'Este programa calcula el área de un '//&
        &'círculo con radio dado por el usuario'
    PRINT*, ''
    PRINT*, 'Ingrese el valor del radio positivo: '
    READ*, radio
    radio= ABS(radio)
    area= pi*radio**2
    PRINT*, 'El área del círculo con radio '&
        &,radio,' es ',area,' unidades cuadradas'
    READ*   !pone en espera la ejecución
END PROGRAM areaCirculo
```

La edición de código puede realizarse en editores de texto plano y de forma independiente compilarse y ejecutarse desde una terminal o consola, o en IDEs propios para programación.

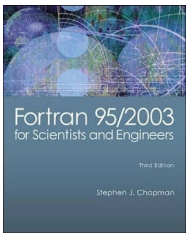
Entre los editores se tiene vi en modo consola en un sistema operativo basado en UNIX, gedit en Linux o bloc de notas en Windows.

Si se desea trabajar desde un entorno de desarrollo integrado los siguientes soportan el lenguaje FORTRAN: Geany, CodeBlocks, Plato o Eclipse.

Finalmente, si la intención es trabajar con una herramienta en línea la opción puede ser OnlineGDB.

Bibliografía:

Chapman, S. J. (2008). Fortran 95/2003 for Scientists and Engineers. (3a ed.). McGraw-Hill.



Guía práctica de estudio 07: Programas estructurados con instrucciones de selección

Elaborado por:	Revisado por:	Autorizado por:
Ing. Jorge Luis López García Ing. Mayelly Reynoso Andrade	M. C. Laura Sandoval Montaño	Dra. Rocío Alejandra Aldeco Pérez Ing. Luis Sergio Valencia Castro

Guía práctica de estudio 07: Programas estructurados con instrucciones de selección

Objetivo:

El alumnado elaborará programas en lenguaje FORTRAN que incluyan las instrucciones IF, IF-ELSE, SELECT CASE para la resolución de problemas básicos.

Actividades:

1. Codificar en lenguaje FORTRAN programas que hagan uso de las instrucciones de selección IF, IF-ELSE y SELECT CASE.

2. Cada programa empleará los diversos operadores relacionales y lógicos.

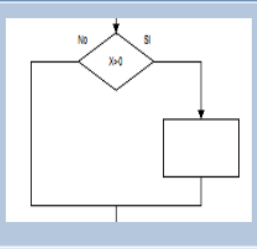
Introducción:

Las instrucciones de selección evalúan una condición y dependiendo si ésta se cumple o no el flujo del programa cambia.

La condición se coloca entre paréntesis con una variable contra otra o una variable contra un número unidos en ambos casos por un operador de comparación.

Las siguientes tablas contienen las palabras reservadas elegidas para escribir pseudocódigo, su símbolo equivalente para diagrama de flujo y la instrucción equivalente en lenguaje FORTRAN para algunas variantes de las estructuras de selección.

Acción de pseudocódigo	Símbolo de diagrama de flujo	Lenguaje FORTRAN	Descripción

SI x>0 ENTONCES
  acciones
FIN DEL SI
		IF(x>0)sentencia
		
		IF(x>0) THEN
		  sentencias
		END IF
			Sólo valida el caso donde la condición se cumpla realizando una instrucción o varias

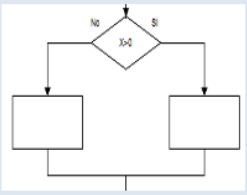
SI x>0 ENTONCES
  acciones
CASO CONTRARIO
  acciones
FIN DEL SI
		IF(x>0) THEN
		  sentencias
		ELSE
		  sentencias
		END IF
			Valida los casos donde la condición se cumpla y también que no se cumpla

***Tabla 1**. Equivalencia de acciones en pseudocódigo, diagrama de flujo y lenguaje FORTRAN para selección simple y doble selección.*

Acción de pseudocódigo	Símbolo de diagrama de flujo	Lenguaje FORTRAN	Descripción

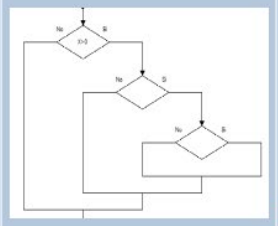
SI x>0 ENTONCES
  SI … ENTONCES
  SI … ENTONCES
 	
  FIN DEL SI
  FIN DEL SI
FIN DEL SI
		IF(x>0) THEN
		  IF(x>1) THEN
		  IF(x>2) THEN
		  sentencias
		  END IF
		  END IF
		END IF
			Pueden anidarse decisiones del lado verdadero pero cada IF abierto deberá cerrarse

***Tabla 2.** Equivalencia de acciones en pseudocódigo, diagrama de flujo y lenguaje FORTRAN para decisiones anidadas.*

Acción de pseudocódigo	Símbolo de diagrama de flujo	Lenguaje FORTRAN	Descripción

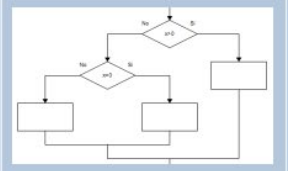
SI x>0 ENTONCES
  acciones
CASO CONTRARIO
  SI x=0 ENTONCES
  acciones
  CASO CONTRARIO
  acciones
  FIN DEL SI
FIN DEL SI
		IF(x>0) THEN
		  sentencias
		ELSE
		  IF(x==0) THEN
		  sentencias
		  ELSE
		  sentencias
		  END IF
		END IF
			Pueden anidarse decisiones del lado verdadero y del caso contrario de una condición pero cada IF abierto deberá cerrarse

Tabla 3. *Equivalencia de acciones en pseudocódigo, diagrama de flujo y lenguaje FORTRAN para decisiones anidadas*

Operador	Símbolo f90 / 95	Símbolo f77
MAYOR QUE	>	.GT.
MENOR QUE	<	.LT.
MAYOR O IGUAL QUE	>=	.GE.
MENOR O IGUAL QUE	<=	.LE.
IGUAL	==	.EQ.
DIFERENTE	/=	.NE.

Tabla 4*. Operadores de comparación.*

Operador	Símbolo
Y	.AND.
O	.OR.
NEGACIÓN	.NOT.

***Tabla 5**. Operadores lógicos.*

Licencia Pública General de GNU

El software contenido en esta práctica es libre bajo la GNU GPL con lo cual el alumnado está en libertad de usar, estudiar, compartir y modificar el software mientras se mantenga la licencia.

```
!Descripción general del programa
!Copyright 2023 Jorge Luis López
!
!This program is free software: you can redistribute it and/or modify it under the terms of the GNU
! General Public License as published by the Free Software Foundation, either version 3 of the License, or
! (at your option) any later version.
!
!This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without
! even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See
! the GNU General Public License for more details.
!
!You should have received a copy of the GNU General Public License along with this program. If not, see
! <https://www.gnu.org/licenses/>.
```

El siguiente ejemplo muestra el uso de decisiones anidadas tanto del lado verdadero como del falso de la condición para mostrar el menor de 3 números dados por el usuario.

Ejemplo 1 – menor3Numeros.f95

```
PROGRAM menor3Numeros

    !no declare en automático i-n como enteros y el resto como reales
    IMPLICIT NONE

    REAL::a, b, c   !declaración de variables real

    PRINT*, 'Este programa lee 3 números y '//&
        &'dice cual es el menor'
    PRINT*, ''      !imprime un salto de línea
    PRINT*, 'Ingresa los 3 números separados por coma: '
    READ*, a,b,c
    PRINT*, ''      !imprime un salto de línea

    IF(a<b) THEN
        IF(a<c) THEN
            PRINT*, a,' es el menor'
        ELSE
            PRINT*, c,' es el menor'
        END IF
    ELSE
        IF(b<c) THEN
            PRINT*, b,' es el menor'
        ELSE
            PRINT*, c,' es el menor'
        END IF
    END IF

    READ*   !hace una pausa
END PROGRAM menor3Numeros
```

Acción de pseudocódigo	Símbolo de diagrama de flujo	Lenguaje FORTRAN

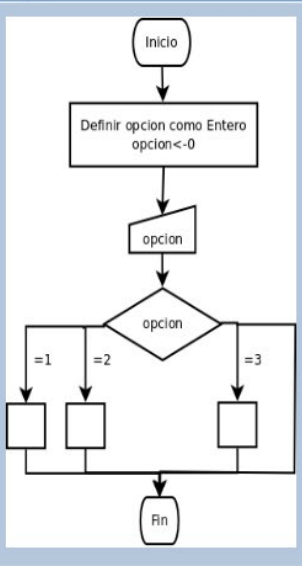
SEGÚN_SEA(opcion)
Inicio
  Caso 1:
  acciones
  salir
  Caso 2:
  acciones
  salir
  Caso 3:
  acciones
  salir
  Caso contrario:
  acciones
  salir
Fin
		SELECT CASE(opcion)
		  CASE (1)
		  sentencias
		  CASE (2)
		  sentencias
		  CASE (3)
		  sentencias
		  CASE DEFAULT
		  sentencias
		END SELECT

Tabla 6. *Equivalencia de acciones en pseudocódigo, diagrama de flujo y lenguaje FORTRAN para selección múltiple.*

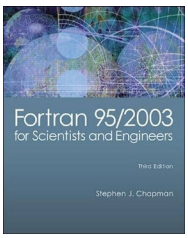
El siguiente programa muestra el cascarón de un menú de 3 opciones cuyo objetivo es ilustrar el funcionamiento de la estructura SELECT CASE. El programa puede adaptarse a la cantidad de opciones que se requieran y en el lugar de la impresión de cada caso agregar el código correspondiente a lectura de datos, operaciones e impresión de resultados.

Ejemplo 2 – menuSelect.f95

```
PROGRAM menuSelect

    !no declare en automático i-n como enteros y el resto como reales
    IMPLICIT NONE

    INTEGER opcion

    PRINT*, 'Este programa implementa un menú'//&
        &' con selección múltiple: '
    PRINT*, '1. Suma'
    PRINT*, '2. Resta'
    PRINT*, '3. Salir'
    PRINT*, 'Elige una opción: '
    READ*, opcion

    SELECT CASE(opcion)
        CASE(1)
            PRINT*, 'Opción suma'
        CASE(2)
            PRINT*, 'Opción resta'
        CASE(3)
            PRINT*, 'Opción salir'
            READ*   !pone en espera la ejecución
            STOP    !termina la ejecución justo aquí
        CASE DEFAULT
            PRINT*, 'Opción no válida'
    END SELECT

    READ*   !pone en espera la ejecución
END PROGRAM menuSelect
```

Bibliografía:

Chapman, S. J. (2008). Fortran 95/2003 for Scientists and Engineers. (3a ed.). McGraw-Hill.



Guía práctica de estudio 08: Programas estructurados con instrucciones de repetición

Elaborado por:	Revisado por:	Autorizado por:
Ing. Jorge Luis López García Ing. Mayelly Reynoso Andrade	M. C. Laura Sandoval Montaño	Dra. Rocío Alejandra Aldeco Pérez Ing. Luis Sergio Valencia Castro

Guía práctica de estudio 08: Programas estructurados con instrucciones de repetición

Objetivo:

El alumnado elaborará programas en lenguaje FORTRAN que incluyan las instrucciones DO, DO WHILE para la resolución de problemas básicos.

Actividades:

1. Codificar en lenguaje FORTRAN programas que hagan uso de las instrucciones de repetición DO indexado, DO WHILE y DO-EXIT mostrando sus diferencias.

2. Combinar con instrucciones en secuencia y de selección para lograr programas más complejos y optimizados que empleen validaciones de datos y menús que se repiten.

Introducción:

Los ciclos o instrucciones de repetición permiten repetir una tarea determinada cierto número de veces o mientras una condición que es validada se cumpla.

Además en las siguientes prácticas ayudarán en el recorrido de arreglos unidimensionales y bidimensionales.

Acción de pseudocódigo	Lenguaje FORTRAN	Descripción
DEFINIR conde<-0 como Entero PARA conde<-1 HASTA conde<=10 INCREMENTO<-1 acciones FIN PARA	INTEGER::conde DO conde=1,10,1 sentencias END DO	Se usa cuando los 3 elementos son conocidos y fijos todo el ciclo
DEFINIR conde<-1 como Entero MIENTRAS QUE conde<=10 acciones FIN MIENTRAS QUE	INTEGER::conde conde=1 DO WHILE(conde<=10) sentencias !conde= conde+1 END DO	Se emplea cuando alguno de los elementos cambiará durante el ciclo como la condición sin una cantidad fija de repeticiones

Tabla 1. *Equivalencia de acciones en pseudocódigo y lenguaje FORTRAN para instrucciones de repetición.*

Acción de pseudocódigo	Lenguaje FORTRAN	Descripción

DEFINIR conde<-1 como Entero
 	
HACER
acciones
FIN HACER
 	
MIENTRAS QUE conde<=10
	INTEGER::conde
	conde=1
	
	DO
	sentencias
	!conde= conde+1
	IF(conde>10) EXIT
	END DO
		Siempre realiza por lo menos una vez el bloque de instrucciones agrupados en DO se cumpla o no la condición que se evalúa en el IF. EXIT termina el ciclo al ser cierta la condición

Tabla 2. *Equivalencia de acciones en pseudocódigo y lenguaje FORTRAN para instrucciones de repetición.*

Licencia Pública General de GNU

El software contenido en esta práctica es libre bajo la GNU GPL con lo cual el alumnado está en libertad de usar, estudiar, compartir y modificar el software mientras se mantenga la licencia.

```
!Descripción general del programa
!Copyright 2023 Jorge Luis López
!
!This program is free software: you can redistribute it and/or modify it under the terms of the GNU
! General Public License as published by the Free Software Foundation, either version 3 of the License, or
! (at your option) any later version.
!
!This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without
! even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See
! the GNU General Public License for more details.
!
!You should have received a copy of the GNU General Public License along with this program. If not, see
! <https://www.gnu.org/licenses/>.
```

El siguiente código muestra el funcionamiento de un ciclo DO indexado imprimiendo del 1 al 10 con incremento constante de 1 aprovechando el valor de la variable auxiliar contador que se incrementa en cada repetición.

Ejemplo 1 – doIndexado.f95

```
PROGRAM doIndexado

    !no declare i-n como enteros
    IMPLICIT NONE

    INTEGER::conde  !contador propio declarado

    DO conde=1,10,1
        PRINT*, conde   !imprime valor de contador
    END DO

    READ*   !pone en espera la ejecución
END PROGRAM doIndexado
```

El siguiente código muestra el funcionamiento de un ciclo DO WHILE imprimiendo del 1 al 10 con incremento constante de 1 aprovechando el valor de la variable auxiliar contador que se incrementa en cada repetición.

Ejemplo 2 – do_While.f95

```
PROGRAM do_While

    !no declare i-n como enteros
    IMPLICIT NONE

    INTEGER::conde  !contador propio declarado
    conde=1

    DO WHILE(conde<=10)
        PRINT*, conde   !imprime valor de contador
        conde= conde+1
    END DO

    READ*   !pone en espera la ejecución
END PROGRAM do_While
```

El siguiente código muestra el funcionamiento de un ciclo DO EXIT imprimiendo del 1 al 10 con incremento constante de 1 aprovechando el valor de la variable auxiliar contador que se incrementa en cada repetición.

Ejemplo 3 – doExit.f95

```
PROGRAM doExit

    !no declare i-n como enteros
    IMPLICIT NONE

    INTEGER::conde  !contador propio declarado
    conde=1

    DO
        PRINT*, conde    !imprime valor de contador
        conde= conde+1
        IF(conde>10) EXIT
    END DO

    READ*    !pone en espera la ejecución
END PROGRAM doExit
```

Un ciclo DO WHILE también puede emplearse para validar datos proporcionados por el usuario para que cumpla cierto formato y así evitar que el programa arroje resultados incorrectos o termine abruptamente su ejecución.

La ventaja de usar un ciclo contra instrucciones de selección simples es que el bloque se repetirá las veces necesarias hasta que el usuario ingrese el dato en el formato solicitado.

Ejemplo 4 – validaDoWhile.f95

```
PROGRAM validaDoWhile

    !no declare i-n como enteros
    IMPLICIT NONE

    REAL::radio
    radio=0.

    DO WHILE (radio<=0)
        PRINT*, 'Ingresa el valor del radio positivo: '
        READ*, radio
    END DO

    PRINT*, 'El valor del radio es ',radio

    READ*,  !pone en espera la ejecución
END PROGRAM validaDoWhile
```

Los ciclos DO WHILE o DO EXIT son requeridos cuando se desea que un menú de opciones ya sea con decisiones anidadas o con la estructura SELECT CASE se repita antes de concluir la ejecución con el fin de elegir otra opción después de terminar el bloque de la elegida previamente.

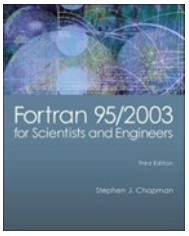
Es muy común en este tipo de menú que se pregunte al usuario si desea repetir presionando opciones con variables de tipo entero como 1 y 2 o con variables de tipo carácter con las opciones 's' o 'n'.

Ejemplo 5 – menuSelectRepite.fp5

```
PROGRAM menuSelectRepite

    !no declare i-n como enteros
    IMPLICIT NONE

    INTEGER opcion
    CHARACTER repite
    repite='s'  !inicializar para que entre al ciclo la primera vez

    !todo lo que quieran repetir va dentro del bloque DO
    DO WHILE(repite=='s' .OR. repite=='S')
        PRINT*, ''
        PRINT*, 'Este programa implementa un menú'//&
            &' con selección múltiple: '
        PRINT*, '1. Suma'
        PRINT*, '2. Resta'
        PRINT*, '3. Salir'
        PRINT*, 'Elige una opción: '
        READ*, opcion

        SELECT CASE(opcion)
            CASE(1)
                PRINT*, 'Opción suma'
            CASE(2)
                PRINT*, 'Opción resta'
            CASE(3)
                PRINT*, 'Opción salir'
                READ*   !pone en espera la ejecución
                STOP    !termina la ejecución justo aquí
            CASE DEFAULT
                PRINT*, 'Opción no válida'
        END SELECT

        PRINT*, ''
        PRINT*, 'Desea realizar otra operación (s/n): '
        READ*, repite
    END DO  !cierra bloque DO

    READ*   !pone en espera la ejecución
END PROGRAM menuSelectRepite
```

Bibliografía:

Chapman, S. J. (2008). Fortran 95/2003 for Scientists and Engineers. (3a ed.). McGraw-Hill.



Guía práctica de estudio 09: Arreglos unidimensionales numéricos y cadenas

Elaborado por:	Revisado por:	Autorizado por:
Ing. Jorge Luis López García Ing. Mayelly Reynoso Andrade	M. C. Laura Sandoval Montaño	Dra. Rocío Alejandra Aldeco Pérez Ing. Luis Sergio Valencia Castro

Guía práctica de estudio 09: Arreglos unidimensionales numéricos y cadenas

Objetivo:

El alumnado elaborará programas en lenguaje FORTRAN para resolver problemas que requieran agrupar y almacenar una determinada cantidad de elementos del mismo tipo de dato numérico o carácter de forma consecutiva para su posterior procesamiento.

Actividades:

1. Codificar en lenguaje FORTRAN programas que declaren arreglos de una dimensión mostrando su recorrido mediante índices y ciclos.

2. Mostrar cómo leer, agrupar y operar datos almacenados en más de un arreglo unidimensional.

Introducción:

Un arreglo estático unidimensional ya sea numérico o de caracteres da la oportunidad al usuario de almacenar una mayor cantidad de datos y manejarlos con simplicidad al contrario de usar la misma cantidad de variables que de datos, lo cual complicaría al programador y limitaría al usuario.

El arreglo se integra por una secuencia de espacios de memoria del mismo tipo acorde a la cantidad de elementos solicitados en la declaración. Cada elemento tiene el mismo nombre de la variable de arreglo, pero se identifica por un índice de posición.

Los índices de posición van de 1 hasta la cantidad de elementos reservados.

El arreglo se lee, opera e imprime un elemento a la vez, por lo cual facilita su recorrido el uso de instrucciones de repetición, siendo el más simple el DO INDEXADO.

Declaración de un arreglo:

```
INTEGER::array(3)          !3 espacios
INTEGER::arreglo(10)       !10 espacios
```

Declaración de 3 arreglos de la misma dimensión:

```
INTEGER,DIMENSION(3)::a, b, c    !3 espacios
```

Impresión de un cuadrito en específico del arreglo:

```
PRINT*, 'Elemento 1: ',arreglo(1)
```

Lectura de un cuadrito en específico del arreglo:

```
READ*, arreglo(1)
```

En lenguaje FORTRAN un arreglo se declara con el tipo, nombre de la variable, entre paréntesis la cantidad de elementos a reservar y si se pretende inicializarlo eso se anotará en la siguiente línea o líneas.

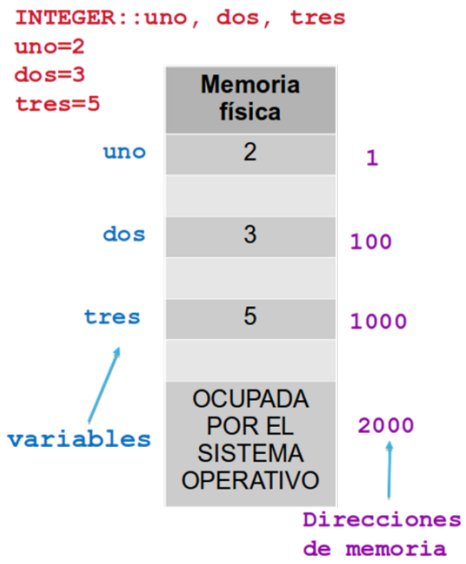


Figura 1. *Diagrama de memoria para variables independientes.*

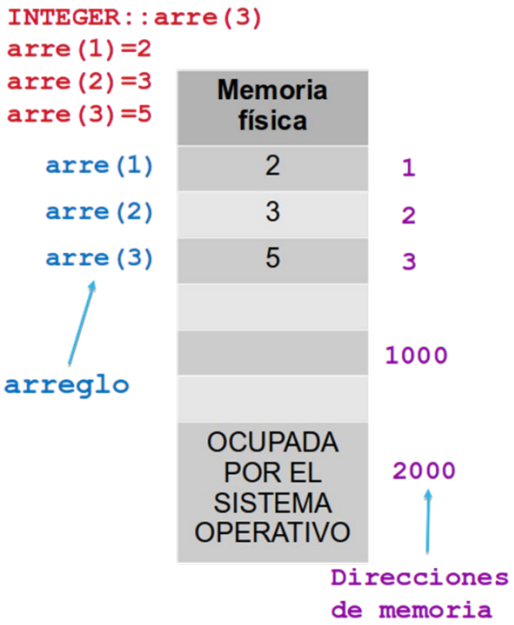


Figura 2. *Diagrama de memoria para arreglo unidimensional.*

Licencia Pública General de GNU

El software contenido en esta práctica es libre bajo la GNU GPL con lo cual el alumnado está en libertad de usar, estudiar, compartir y modificar el software mientras se mantenga la licencia.

```
!Descripción general del programa
!Copyright 2023 Jorge Luis López
!
!This program is free software: you can redistribute it and/or modify it under the terms of the GNU
! General Public License as published by the Free Software Foundation, either version 3 of the License, or
! (at your option) any later version.
!
!This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without
! even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See
! the GNU General Public License for more details.
!
!You should have received a copy of the GNU General Public License along with this program. If not, see
! <https://www.gnu.org/licenses/>.
```

El siguiente ejemplo muestra desde como declarar e inicializar el arreglo hasta las opciones para imprimirlo como usando su nombre, individualmente y con un ciclo DO INDEXADO.

Ejemplo 1 – arregloUni.f95

```
PROGRAM arregloUni

    !no va IMPLICIT NONE para usar i como INTEGER
    INTEGER::arreglo(3)

    arreglo=(/ 3, 33, 333 /)

    arreglo(1)= 3
    arreglo(2)= 33
    arreglo(3)= 333

    PRINT*, 'Impresión usando nombre de variable: '
    PRINT*, arreglo

    PRINT*, ''
    PRINT*, 'Impresión cuadrito por cuadrito: '
    PRINT*, 'Elemento 1: ',arreglo(1)
    PRINT*, 'Elemento 2: ',arreglo(2)
    PRINT*, 'Elemento 3: ',arreglo(3)

    PRINT*, ''
    PRINT*, 'Impresión con ciclo do indexado: '

    !DO i=1,3,1
    DO i=1,3
        PRINT*, 'Elemento ',i,': ',arreglo(i)
    END DO

    READ*   !pone en espera la ejecución
END PROGRAM arregloUni
```

Es común operar un arreglo contra un número, una variable o para el siguiente ejemplo contra otro arreglo y guardar el resultado en un tercer arreglo.

Ejemplo 2 – sumaVectores.f95

```
PROGRAM sumaVectores

    !no va IMPLICIT NONE para usar i como INTEGER
    !INTEGER,DIMENSION(3)::a, b, c
    INTEGER::a(3), b(3), c(3)

    PRINT*, 'Programa que suma 2 vectores de 3 elementos'
    PRINT*, ''

    PRINT*, 'Primer vector a: '
    DO i=1,3    !ciclo para leer a
        PRINT*, 'Ingrese el valor de a(',i,'): '
        READ*, a(i)
    END DO

    PRINT*, ''
    PRINT*, 'Segundo vector b: '
    DO i=1,3    !ciclo para leer b
        PRINT*, 'Ingrese el valor de b(',i,'): '
        READ*, b(i)
    END DO

    DO i=1,3    !ciclo para sumar a y b
        c(i)= a(i) + b(i)
    END DO

    PRINT*, '   a(',a(1),', ',a(2),', ',a(3),')'
    PRINT*, ' + b(',b(1),', ',b(2),', ',b(3),')'
    PRINT*, ' = c(',c(1),', ',c(2),', ',c(3),')'

    READ*   !pone en espera la ejecución
END PROGRAM sumaVectores
```

Al procesar datos proporcionados por el usuario el programa deber ser capaz de soportar la cantidad de elementos que éste requiera y no obligarlo a un arreglo fijo desde un inicio con el riesgo de ser una cantidad menor o mayor.

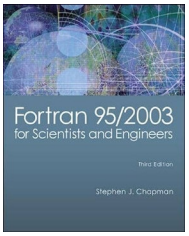
Se puede declarar un arreglo de gran espacio y después solicitar al usuario ingrese la cantidad de elementos a procesar siempre y cuando se encuentre dentro del rango del arreglo reservado.

Ejemplo 3 – arregloUniUsuario.f95

```
PROGRAM arregloUniUsuario

    !no va IMPLICIT NONE para usar i como INTEGER
    INTEGER,PARAMETER::MAXIMO=100
    INTEGER::cuantos
    REAL::arreglo(MAXIMO)

    PRINT*, 'Programa que lee e imprime arreglo de n elementos'
    PRINT*, ''
    PRINT*, 'Ingresa la cantidad de elementos: '
    READ*, cuantos

    PRINT*, 'Arreglo a: '
    DO i=1,cuantos    !ciclo para leer arreglo
        PRINT*, 'Ingrese elemento(',i,'): '
        READ*, arreglo(i)
    END DO

    PRINT*, ''
    PRINT*, 'Arreglo a: '
    DO i=1,cuantos    !ciclo para imprimir arreglo
        PRINT*, 'Elemento(',i,'): ',arreglo(i)
    END DO

    READ*   !pone en espera la ejecución
END PROGRAM arregloUniUsuario
```

Bibliografía:

Chapman, S. J. (2008). Fortran 95/2003 for Scientists and Engineers. (3a ed.). McGraw-Hill.



Guía práctica de estudio 10: Arreglos bidimensionales parte 1

Elaborado por:	Revisado por:	Autorizado por:
Ing. Jorge Luis López García Ing. Mayelly Reynoso Andrade	M. C. Laura Sandoval Montaño	Dra. Rocío Alejandra Aldeco Pérez Ing. Luis Sergio Valencia Castro

Guía práctica de estudio 10: Arreglos bidimensionales parte 1

Objetivo:

El alumnado elaborará programas en lenguaje FORTRAN para resolver problemas que requieran agrupar y almacenar una determinada cantidad de elementos del mismo tipo de dato en arreglos de dos dimensiones para su posterior procesamiento.

Actividades:

1. Codificar en lenguaje FORTRAN programas que declaren un arreglo de dos dimensiones mostrando su recorrido mediante índices y ciclos.

2. Mostrar cómo leer y agrupar datos en un arreglo, así como operarlos con variables simples.

Introducción:

Un arreglo bidimensional al igual que el unidimensional se integra por espacios consecutivos de memoria, pero en esta ocasión la dimensión depende de la cantidad de renglones y columnas solicitados.

Todos los espacios tienen el mismo nombre, pero cada espacio se identifica ahora por 2 índices de posición, uno indica el renglón y el otro la columna.

Los índices van de 1 hasta el valor de la cantidad reservada para renglones y columnas respectivamente.

Se declaran indicando su tipo de dato, nombre y tamaño que se crea necesitar de renglones y columnas entre paréntesis separados por coma.

El arreglo no se opera al mismo tiempo, debe hacerse un cuadrito a la vez, aunque sea bidimensional.

Se emplean 2 ciclos anidados para recorrer cada elemento del arreglo.

Declaración de un arreglo:

```
INTEGER::m(3,3)        !3 renglones por 3 columnas
INTEGER::mat(10,10)    !10 reng por 10 col
```

Declaración de 3 arreglos de la misma dimensión:

```
INTEGER,DIMENSION(3,3)::a, b, c   !3x3 espacios
```

Impresión de un cuadrito en específico del arreglo:

```
PRINT*, 'Elemento 1,1: ',arregloBidi(1,1)
```

Lectura de un cuadrito en específico del arreglo:

```
READ*, arregloBidi(1,1)
```

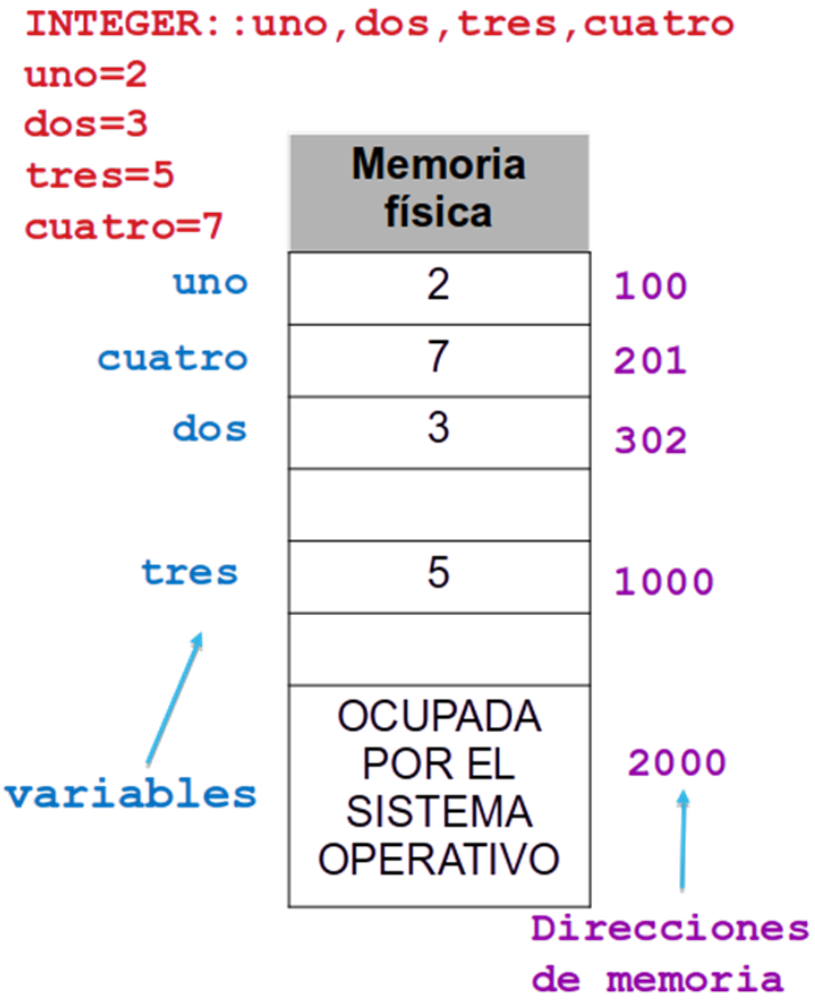


Figura 1. *Diagrama de memoria para variables independientes.*

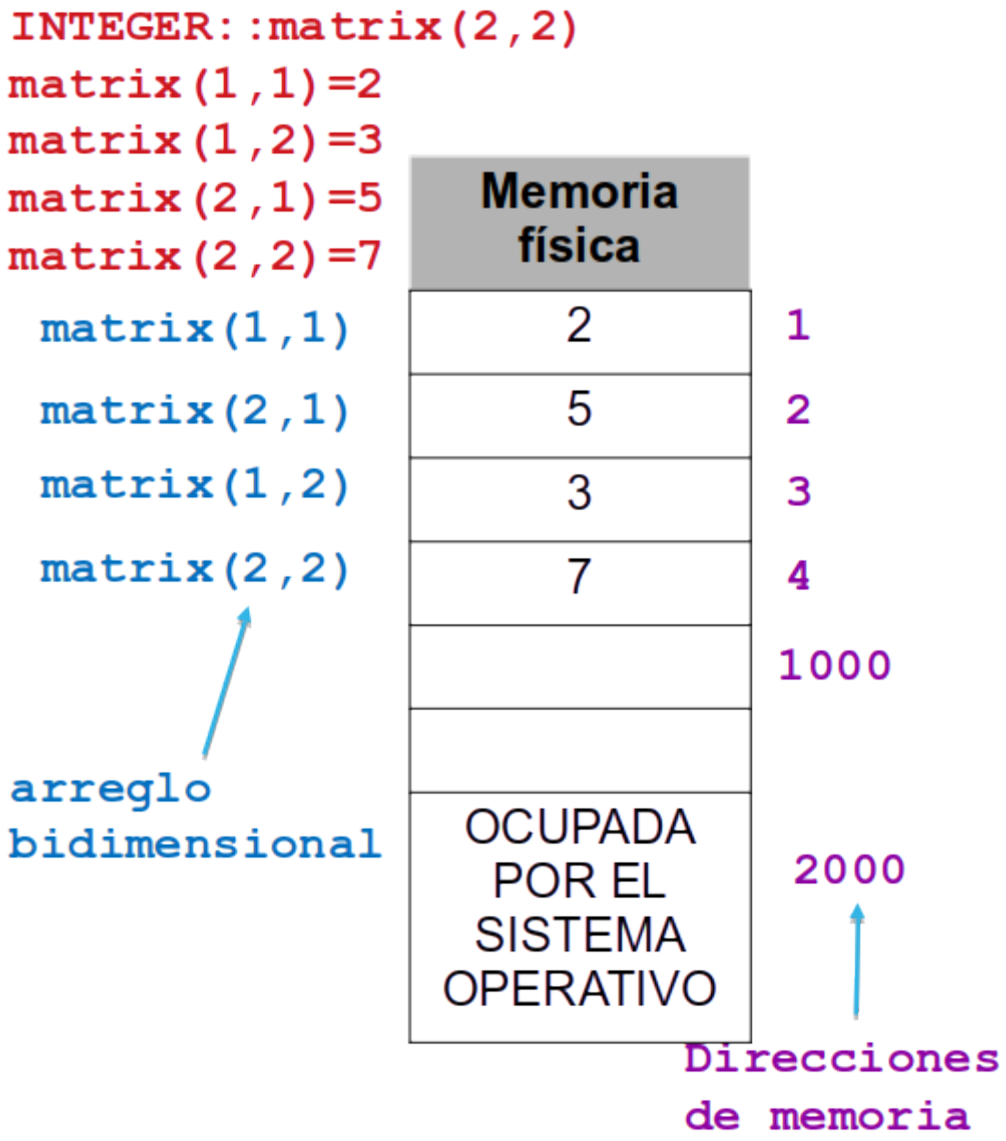


Figura 2. *Diagrama de memoria para arreglo bidimensional.*

Licencia Pública General de GNU

El software contenido en esta práctica es libre bajo la GNU GPL con lo cual el alumnado está en libertad de usar, estudiar, compartir y modificar el software mientras se mantenga la licencia.

```
!Descripción general del programa
!Copyright 2023 Jorge Luis López
!
!This program is free software: you can redistribute it and/or modify it under the terms of the GNU
! General Public License as published by the Free Software Foundation, either version 3 of the License, or
! (at your option) any later version.
!
!This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without
! even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See
! the GNU General Public License for more details.
!
!You should have received a copy of the GNU General Public License along with this program. If not, see
! <https://www.gnu.org/licenses/>.
```

El siguiente ejemplo muestra desde como declarar e inicializar el arreglo hasta las opciones para imprimirlo como usando su nombre, individualmente y con dos ciclos DO INDEXADO.

Ejemplo 1 – arregloBidimensional.f95

```
PROGRAM arregloBidimensional

    !no va IMPLICIT NONE para usar i como INTEGER
    INTEGER::arregloBidi(2,2)

    arregloBidi(1,1)= 1
    arregloBidi(1,2)= 3
    arregloBidi(2,1)= 5
    arregloBidi(2,2)= 7

    PRINT*, 'Impresión por nombre y orden en memoria: '
    PRINT*, arregloBidi

    PRINT*, ''
    PRINT*, 'Impresión cuadrito por cuadrito: '
    PRINT*, 'Elemento 1,1: ',arregloBidi(1,1)
    PRINT*, 'Elemento 1,2: ',arregloBidi(1,2)
    PRINT*, 'Elemento 2,1: ',arregloBidi(2,1)
    PRINT*, 'Elemento 2,2: ',arregloBidi(2,2)

    PRINT*, ''
    PRINT*, 'Impresión con ciclos do indexado: '

    !DO i=1,2,1
    DO i=1,2        !ciclo para renglones
        DO j=1,2    !ciclo para columnas
            PRINT*, 'Elemento(',i,', ',j,'): ',arregloBidi(i,j)
        END DO
    END DO

    READ*   !pone en espera la ejecución
END PROGRAM arregloBidimensional
```

Además de inicializar los valores del arreglo con el mismo valor, individualmente o dados por el usuario, también pueden provenir de la subrutina RANDOM_NUMBER() que genera números pseudoaleatorios en un rango de 0 a 1.

Ejemplo 2 – arregloBidiAleatorio.f95

```
PROGRAM arregloBidiAleatorio

    !no va IMPLICIT NONE para usar i como INTEGER
    REAL::m(3,3)

    PRINT*, 'Este programa genera números'//&
        &' aleatorios para arreglo bidimensional'

    DO i=1,3    !ciclo para renglones
        DO j=1,3 !ciclo para columnas
            call RANDOM_NUMBER(m(i,j)) !genera de 0 a 1
        END DO
    END DO

    PRINT*, ''
    PRINT*, 'Matriz m:'
    DO i=1,3   !ciclo para imprimir arreglo
        PRINT*, (m(i,j), j=1,3)
    END DO

    READ*   !pone en espera la ejecución
END PROGRAM arregloBidiAleatorio
```

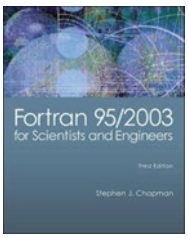
Para imprimir en formato de matriz se recurre al DO implícito dentro de la impresión debajo del DO INDEXADO que recorre los renglones. Esta forma de impresión se observa tanto en el ejemplo anterior como en el siguiente.

Ejemplo 3 – arregloMatriz.f95

```
PROGRAM arregloMatriz

    !no va IMPLICIT NONE para usar i como INTEGER
    REAL::m(2,2)

    PRINT*, 'Este programa lee e imprime arreglo bidimensional '
    PRINT*, ''

    PRINT*, 'Ingrese elementos por renglón separados por enter'
    DO i=1,2
        DO j=1,2
            PRINT*, 'Ingrese elemento(',i,', ',j,'): '
            READ*, m(i,j)
        END DO
    END DO

    PRINT*, ''
    PRINT*, 'Impresión con ciclo implícito: '
    DO i=1,2
        PRINT*, (m(i,j), j=1,2)
    END DO

    READ*   !pone en espera la ejecución
END PROGRAM arregloMatriz
```

Bibliografía:

Chapman, S. J. (2008). Fortran 95/2003 for Scientists and Engineers. (3a ed.). McGraw-Hill.



Guía práctica de estudio 11: Arreglos bidimensionales parte 2

Elaborado por:	Revisado por:	Autorizado por:
Ing. Jorge Luis López García Ing. Mayelly Reynoso Andrade	M. C. Laura Sandoval Montaño	Dra. Rocío Alejandra Aldeco Pérez Ing. Luis Sergio Valencia Castro

Guía práctica de estudio 11: Arreglos bidimensionales parte 2

Objetivo:

El alumnado elaborará programas en lenguaje FORTRAN para resolver problemas que requieran agrupar y almacenar una determinada cantidad de elementos del mismo tipo de dato en arreglos de dos dimensiones para su posterior procesamiento.

Actividades:

1. Codificar en lenguaje FORTRAN programas que declaren más de un arreglo de dos dimensiones mostrando su recorrido mediante índices y ciclos.

2. Mostrar cómo leer, agrupar y operar datos almacenados en más de un arreglo bidimensional.

Introducción:

De igual forma que con los arreglos unidimensionales, es común operar un arreglo contra un número, una variable o para el siguiente ejemplo contra otro arreglo y guardar el resultado en un tercer arreglo.

Como se mencionó en la práctica anterior, un arreglo bidimensional al igual que el unidimensional se integra por espacios consecutivos de memoria, pero en esta ocasión la dimensión depende de la cantidad de renglones y columnas solicitados.

Ejemplo 1 – sumaMatrices.f95

```
PROGRAM sumaMatrices

    !no va IMPLICIT NONE para usar i como INTEGER
    !INTEGER,DIMENSION(2,2)::a, b, c
    INTEGER::a(2,2), b(2,2), c(2,2)

    PRINT*, 'Programa que suma 2 matrices de 2x2'
    PRINT*, ''

    PRINT*, 'Primer matriz a: '
    DO i=1,2
        DO j=1,2
            PRINT*, 'Ingrese elemento(',i,', ',j,'): '
            READ*, a(i,j)
        END DO
    END DO

    PRINT*, ''
    PRINT*, 'Segunda matriz b: '
    DO i=1,2
        DO j=1,2
            PRINT*, 'Ingrese elemento(',i,', ',j,'): '
            READ*, b(i,j)
        END DO
    END DO
```

```
    DO i=1,2
        DO j=1,2
            c(i,j)= a(i,j) + b(i,j)
        END DO
    END DO

    PRINT*, ''
    PRINT*, 'Primer matriz a: '
    DO i=1,2
        PRINT*, (a(i,j), j=1,2)
    END DO

    PRINT*, ''
    PRINT*, 'Segunda matriz b: '
    DO i=1,2
        PRINT*, (b(i,j), j=1,2)
    END DO

    PRINT*, ''
    PRINT*, 'La matriz c con la suma es: '
    DO i=1,2
        PRINT*, (c(i,j), j=1,2)
    END DO

    READ*   !pone en espera la ejecución
END PROGRAM sumaMatrices
```

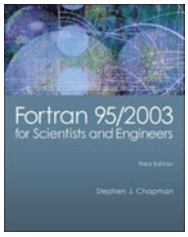
Se puede declarar un arreglo de gran espacio y después solicitar al usuario ingrese la cantidad de renglones y columnas a procesar siempre y cuando se encuentre dentro del rango del arreglo reservado.

Ejemplo 2 – arregloBidiUsuario.f95

```
PROGRAM arregloBidiUsuario

    !no va IMPLICIT NONE para usar i, j como INTEGER
    INTEGER,PARAMETER::MAXIMO=100
    INTEGER::renglones,columnas
    REAL::m(MAXIMO,MAXIMO)

    PRINT*, 'Programa que lee e imprime arreglo bidi de mxn elementos'
    PRINT*, ''
    PRINT*, 'Ingresa la cantidad de renglones: '
    READ*, renglones
    PRINT*, 'Ingresa la cantidad de columnas: '
    READ*, columnas

    PRINT*, 'Matriz m: '
    DO i=1,renglones    !ciclo para renglones
        DO j=1,columnas !ciclo para columnas
            PRINT*, 'Ingrese elemento(',i,', ',j,'): '
            READ*, m(i,j)
        END DO
    END DO

    PRINT*, ''
    PRINT*, 'Matriz m: '
    DO i=1,renglones
        PRINT*, (m(i,j), j=1,columnas)
    END DO

    READ*   !pone en espera la ejecución
END PROGRAM arregloBidiUsuario
```

Bibliografía:

Chapman, S. J. (2008). Fortran 95/2003 for Scientists and Engineers. (3a ed.). McGraw-Hill.



Guía práctica de estudio 12: Funciones y subrutinas

Elaborado por:	Revisado por:	Autorizado por:
Ing. Jorge Luis López García Ing. Mayelly Reynoso Andrade	M. C. Laura Sandoval Montaño	Dra. Rocío Alejandra Aldeco Pérez Ing. Luis Sergio Valencia Castro

Guía práctica de estudio 12: Funciones y subrutinas

Objetivo:

El alumnado elaborará programas en lenguaje FORTRAN optimizados en funciones y subrutinas que permitan resolver problemas dividiéndolos en bloques más simples o especializados.

Actividades:

1. Identificar los tipos de subprogramas de FORTRAN.

2. Definir e implementar funciones y subrutinas con paso de argumentos desde el programa principal.

Introducción:

FORTRAN cuenta con 2 tipos de subprogramas:

- Funciones:
  - Devuelven un valor al final.
  - Puede ser llamada en una expresión u operación para emplear el resultado que regresa.

- Subrutinas:
  - No devuelven valor.
  - Los resultados pueden transmitirse a través de sus argumentos.

Características de los subprogramas

- Su código es independiente del programa principal y de otros subprogramas.
- Las variables declaradas dentro del subprograma son locales y no son visibles desde otro, aunque lleven el mismo nombre.
- Los datos necesarios se reciben como argumentos en el paréntesis.
- Existen funciones y subrutinas intrínsecas de FORTRAN, es decir, están definidas como parte del lenguaje.
- Pueden ser creadas funciones y subrutinas propias.
- Las funciones se llaman en una expresión por su nombre y lista de argumentos dentro del paréntesis y se asignan a una variable del mismo tipo que el retorno de la función.
- Las subrutinas se llaman usando la sentencia CALL por su nombre y lista de argumentos dentro del paréntesis que pueden cambiar su valor original si se declaran como INOUT.
- Las subrutinas se emplean para leer, imprimir o modificar arreglos unidimensionales y bidimensionales.

Argumentos de entrada, salida o ambos

- INTENT(IN)
  - Argumento solo de entrada.
  - No puede ser modificado dentro del bloque.
- INTENT(OUT)
  - Argumento de salida.
  - Guarda resultado para devolver al programa principal.

- INTENT(INOUT)
  - Argumento de entrada y salida.
  - Pasa valor y devuelve resultado.

Las siguientes tablas contienen ejemplos de bloques de código de subrutinas y funciones, así como sus llamadas desde el programa principal (lado derecho) incluyendo la declaración de variables necesarias para pasar como argumentos o recibir el valor devuelto por la función.

Es importante mencionar que para el caso de las funciones también debe declararse en el programa principal una variable con el mismo nombre de la función y el mismo tipo de retorno.

Función o Subrutina	Llamada

`SUBROUTINE saludo()`
`  PRINT*, 'Hola desde subrutina'`
`END SUBROUTINE saludo`
	`!NO recibe argumentos`
	`!NO devuelve`
	`CALL saludo()`

`SUBROUTINE saludo(nom)`
`  !len=* Toma tamaño de cadena origen`
`  CHARACTER(len=*), INTENT(IN)::nom`
`  PRINT*, 'Hola ',nom,' desde subrutina'`
`END SUBROUTINE saludo`
	`!RECIBE 1 cadena`
	`!NO devuelve`
	`CHARACTER*30::nombre`
	`CALL saludo(nombre)`

***Tabla 1**. Ejemplos de subrutinas con sus respectivas llamadas en el programa principal.*

Función o Subrutina	Llamada

`SUBROUTINE suma(a, b)`
`  INTEGER, INTENT(IN)::a,b`
`  INTEGER::c`
`  c= a+b`
`  PRINT*, a,' + ',b,' es: ',c`
`END SUBROUTINE suma`
	`!RECIBE 2 INTEGER`
	`!NO devuelve`
	`INTEGER::uno, dos`
	`CALL suma(uno, dos)`

`INTEGER FUNCTION suma2(a, b)`
`  INTEGER, INTENT(IN)::a,b`
`  INTEGER::c`
`  c= a+b`
`  suma2= c`
`END FUNCTION suma2`
	`!RECIBE 2 INTEGER`
	`!SÍ devuelve 1 INTEGER`
	`INTEGER::uno, dos, res`
	`INTEGER::suma2`
	`res= suma2(uno,dos)`

Tabla 2. *Ejemplos de subrutinas y funciones con sus respectivas llamadas en el programa principal.*

Función o Subrutina	Llamada

`INTEGER FUNCTION resta(a, b)`
`  INTEGER, INTENT(IN)::a,b`
`  resta= a-b`
`END FUNCTION resta`
	`!RECIBE 2 INTEGER`
	`!SÍ devuelve 1 INTEGER`
	`INTEGER::uno, dos, r`
	`INTEGER::resta`
	`r= resta(uno, dos)`

`REAL FUNCTION divi(a, b)`
`  INTEGER, INTENT(IN)::a,b`
`  divi= REAL(a)/b`
`END FUNCTION divi`
	`!RECIBE 2 INTEGER`
	`!SÍ devuelve 1 REAL`
	`INTEGER::uno, dos`
	`REAL::d`
	`REAL::divi`
	`d= divi(uno, dos)`

`INTEGER FUNCTION triple(a)`
`  INTEGER, INTENT(IN)::a`
`  triple= a*3`
`END FUNCTION triple`
	`!RECIBE 1 INTEGER`
	`!SÍ devuelve 1 INTEGER`
	`INTEGER::uno, t`
	`INTEGER::triple`
	`t= triple(uno)`

Tabla 3. *Ejemplos de subrutinas y funciones con sus respectivas llamadas en el programa principal.*

Licencia Pública General de GNU

El software contenido en esta práctica es libre bajo la GNU GPL con lo cual el alumnado está en libertad de usar, estudiar, compartir y modificar el software mientras se mantenga la licencia.

```
!Descripción general del programa
!Copyright 2023 Jorge Luis López
!
!This program is free software: you can redistribute it and/or modify it under the terms of the GNU
! General Public License as published by the Free Software Foundation, either version 3 of the License, or
! (at your option) any later version.
!
!This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without
! even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See
! the GNU General Public License for more details.
!
!You should have received a copy of the GNU General Public License along with this program. If not, see
! <https://www.gnu.org/licenses/>.
```

El siguiente programa muestra la diferencia entre subrutina y función.

Ejemplo 1 – subrutinaFuncion.f95

```
PROGRAM subFuncionArit

   IMPLICIT NONE

    INTEGER::uno, dos, res;
    INTEGER::suma2  !tipo de dato que devuelve la función
    uno= 1
    dos= 2

    PRINT*, 'Este programa implementa subrutina y función'

    CALL saludo()           !llamada a subrutina

    CALL suma(uno, dos)     !llamada a subrutina

    res= suma2(uno, dos)    !llamada a función
    PRINT*, uno,' + ',dos,' es: ',res

    READ*   !pone en espera la ejecución
END PROGRAM subFuncionArit
```

```
SUBROUTINE saludo()
  PRINT*, 'Hola desde subrutina'
END SUBROUTINE saludo

SUBROUTINE suma(a, b)
    INTEGER, INTENT(IN)::a,b    !argumentos de entrada
    INTEGER::c                  !variable local
    c= a + b
    PRINT*, 'La suma de ',a,' + ',b,' es: ',c
END SUBROUTINE suma

INTEGER FUNCTION suma2(a, b)    !devuelve un entero
    INTEGER, INTENT(IN)::a,b    !argumentos de entrada
    INTEGER::c                  !variable local
    c= a + b
    suma2= c    !se asigna resultado a nombre de función
END FUNCTION suma2
```

Arreglos y subrutinas

El siguiente programa muestra la utilidad de las subrutinas para leer, imprimir y operar arreglos, en este caso un arreglo unidimensional, pero aplica también para arreglos bidimensionales.

Ejemplo 2 – subrutinaArregloUni.f95

```
PROGRAM subrutinaArregloUni

    INTEGER,PARAMETER::MAXIMO=100
    INTEGER::cuantos
    REAL::arreglo(MAXIMO)

    PRINT*, 'Programa que lee e imprime arreglo de n elementos'
    PRINT*, ''
    PRINT*, 'Ingresa la cantidad de elementos: '
    READ*, cuantos

    PRINT*, ''
    PRINT*, 'Arreglo a: '
    CALL leerArreglo(arreglo, cuantos)

    PRINT*, ''
    PRINT*, 'Arreglo a: '
    CALL imprimirArreglo(arreglo, cuantos)

    READ*   !pone en espera la ejecución
END PROGRAM subrutinaArregloUni
```

```
SUBROUTINE leerArreglo(a, cantidad)

    INTEGER, INTENT(IN)::cantidad
    REAL, INTENT(INOUT)::a(cantidad)

    DO i=1,cantidad    !ciclo para leer arreglo
        PRINT*, 'Ingrese elemento(',i,'): '
        READ*, a(i)
    END DO

END SUBROUTINE leerArreglo

SUBROUTINE imprimirArreglo(a, cantidad)

    INTEGER, INTENT(IN)::cantidad
    REAL, INTENT(INOUT)::a(cantidad)

    DO i=1,cantidad    !ciclo para imprimir arreglo
        PRINT*, 'Elemento(',i,'): ',a(i)
    END DO

END SUBROUTINE imprimirArreglo
```

Bibliografía:

Chapman, S. J. (2008). Fortran 95/2003 for Scientists and Engineers. (3a ed.). McGraw-Hill.