INTRODUCCIÓN

La programación de computadores es considerada una disciplina científica, que es parte de las Ciencias de la computación.

Su origen está asociado con la aparición de las primeras máquinas que tenían la capacidad de ejecutar instrucciones y con capacidades para realizar cálculos numéricos y operaciones lógicas, para lo cual éstas deben ser descritas en términos de operaciones sencillas, lo que en palabras simples es la programación de computadores, y que por tanto nace de la mano con los computadores, y es consustancial al uso de los mismos, en el sentido de que para sacar provecho de los computadores, es necesaria la existencia variados software, es decir de programa que han sido diseñados y escritos para tratar con problemas generales o específicos con el apoyo de estas máquinas.

En estos términos, la programación de computadores debemos entenderla como el acto de escribir conjuntos de instrucciones, cálculos numéricos y operaciones lógicas, para automatizar la solución de un problema, utilizando algún lenguaje de programación, como Pascal, C, Java, VBasic, etc.

Así por ejemplo, podríamos usar alguno de estos lenguajes para automatizar la solución del problema de determinar los valores de las variables x e y en cualquier sistema de ecuaciones de primer grado con dos incógnitas.

Sin embargo, la escritura de un programa en un lenguaje como los que señalamos constituye el paso más fácil de la programación, ya que cuando se llega a este punto, se debería haber hecho el paso previo de establecer las secuencias de instrucciones, acciones, cálculos numéricos y operaciones lógicas, que ha de considerar el programa que se va a escribir.

Si pensamos en el ejemplo anterior, previo a la escritura del programa en algún lenguaje, se debe establecer el orden de las acciones, los cálculos, etc. que son necesarias de realizar para determinar los valores de las variables x e y para cualquier sistema de ecuaciones de primer grado con dos incógnitas.

Por tanto, la premisa básica y fundamental en la programación de computadores es que si se quiere indicar al computador mediante un lenguaje lo que queremos que haga, primero debemos disponer de una descripción clara, detallada y precisa de lo que queremos pedirle que realice. La descripción clara, detallada y precisa, se expresa mediante un algoritmo.

El diseño de algoritmos es entonces el aspecto más crucial e importante de la programación de computadores. Cuando un programa computacional funciona de manera incorrecta o está incompleto, casi con toda seguridad, se debe a errores o incompletitud en el diseño del algoritmo.

Ahora bien, aprender a diseñar algoritmos es una tarea de alguna forma compleja, en el sentido de que, para tener éxito en ellas, requiere que el aprendiz desarrolle lo que se denomina un aprendizaje activo, lo que significa que no sólo se han de estudiar y comprender los fundamentos del diseño de algoritmos, sino que además se debe realizar una práctica activa de aplicación de estos fundamentos, lo que se refleja en una frase típica a la que hacemos mención todos quienes llevamos varios años enseñando programación de computadores, "el camino correcto para aprender a diseñar algoritmos, es diseñándolos". Esto significa que la práctica directa en el diseño de algoritmos es una cuestión esencial para lograr el aprendizaje. Dicho de otra forma, "nadie aprenderá a diseñar algoritmos sólo mirando como otros los resuelven o copiando algoritmos ya resueltos".

El propósito del presente texto de estudio es aportar al aprendizaje y la compresión de los fundamentos del diseño de algoritmos, y está pensado para estudiantes que cursan una primera o quizás, única asignatura se Fundamentos de programación de computadores, o de Introducción a la programación de computadores, o Programación de computadores, que son los nombres alternativos que suelen darse a estas asignaturas, dentro del currículum de muchas carreras de formación profesional. Es importante también precisar que dado que este texto es de carácter básico en estas materias, no está concebido para estudiantes de algunas de las carreras de Ingeniería relacionadas directamente con Ciencias de la Computación, ya que para ese tipo de carreras, la programación de computadores, constituye un cuerpo de conocimientos de carácter nuclear, que abarca prácticamente todo el currículum de formación profesional, contexto en el cual, un primer curso

de programación, es bastante más complejos que para el caso de las carreras que no son específicamente del área de las Ciencia de la computación.
Para estos efectos, este texto de estudio se ha organizado de la siguiente manera:
En el capítulo 1 se da respuesta a la clásica pregunta ¿Qué es un algoritmo?, resaltando la importancia que el propio concepto de Algoritmo tiene para la programación de computadores, entendiendo esta como la herramienta para modelar soluciones a problemas de programación.
En el capítulo 2 se abordan los aspectos metodológicos de la resolución de problemas de programación, es decir, los pasos que se deben realizar para tratar con el diseño de soluciones a problemas de programación, enfatizando los propósitos o intencionalidades de cada paso.
En el capítulo 3 se presentan los detalles de los lenguajes para la representación de algoritmos, particularmente, de los Diagramas de Flujo (DF) y de los Pseudocódigos, los dos lenguajes que serán usado para los ejemplos que se muestran en el capítulo 5 de problemas resueltos de diseño de algoritmos.
En el capítulo 4 se presentan los detalles de la herramienta para la escritura y edición de algoritmos PSeInt, la cual es una herramienta Free para estos propósitos.
Hemos considerado la inclusión de esta herramienta, ya que consideramos conforme a nuestra experiencia docente dictando la asignatura de Fundamentos de programación, que su uso representa una ayuda importante, no solo para escribir los algoritmos, sino que también, para que al usar las opciones de ejecución de algoritmos que la herramienta posee, los estudiantes tenga la posibilidad concreta de ver y observar los detalles de cómo un cierto algoritmo es ejecutado, lo cual para nosotros, es una contribución importante para el aprendizaje de las capacidades para el diseño de algoritmos.
Se presenta en primer término una introducción a la herramienta, luego los detalles para descargar e instalar la herramienta, y a continuación los detalles para entender el entorno de trabajo y el modo de uso de los comandos básicos para escribir instrucciones y estructuras de control
En el penúltimo capítulo (capitulo 5) presentamos varios ejemplos de problemas resueltos de diseño de algoritmos, los cuales hemos organizado en función de la complejidad, en problemas que usan estructuras secuenciales, luego ejemplos que usan estructuras selectivas tanto simples como dobles y finalmente, estructuras repetitivas del tipo Mientras, Repetir y Para.
Finalmente, el capítulo 6 presentamos un conjunto variado de problemas propuestos organizados en dos apartados, los que requieren del uso de estructuras Selectivas y los que requieren de estructuras Repetitivas. La idea con estos problemas propuestos es que el estudiante disponga de un repertorio de ejercicios para resolver, para potenciar su aprendizaje, en el bien entendido de lo que ya expresamos en párrafos anteriores, en el sentido de que la única forma efectiva de aprender resolver problemas de diseño algoritmos, es resolviéndolos y no mirando como otros los hacen, o aprendiendo de memoria los algoritmos ya resueltos.

Textos Extraidos del Apunte Completo

01 CDoc Cuaderno-Docente_FP ok iresm completo y con psint.

1. ¿QUÉ ES UN ALGORITMO?

Un algoritmo es la descripción clara, detallada y precisa del conjunto de instrucciones que al ser ejecutadas conducen a obtener la solución de un problema.

Así por ejemplo la expresión matemática $x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$ es un algoritmo, ya que al realizar las operaciones que en ella se indican, para un cierto conjunto de coeficientes a, b y c de una ecuación de segundo grado que está escrita en la forma general $ax^2 + bx + c = 0$, se podrá resolver el problema de encontrar dos soluciones de la ecuación.

Desde la perspectiva computacional, un algoritmo representa un modelo de solución para un determinado tipo de problemas, y su formulación debe cumplir con el requisito de que éste debe ser independiente del lenguaje de programación que se utilice para escribirlo como un programa computacional.

Así, el diseño, construcción, y escritura de un algoritmo es el proceso más relevante de la programación de computadores. La calidad de la solución de un problema en programación, está fuertemente determinada por la calidad del algoritmo. Una solución defectuosa o incompleta para un problema, generalmente es producto de un algoritmo mal diseñado.

La construcción de un algoritmo es parte de un proceso ingenieril propio de las ciencias de la computación, que se denomina Metodología de resolución de problemas de programación.

Para cumplir con sus propósitos un algoritmo debe poseer las siguientes características fundamentales:

- ✓ **Debe ser preciso** e indicar el orden de realización de cada paso.
- ✓ **Debe describir acciones elementales**, si una acción no puede ejecutarse de forma simple, debe ser descompuesta.
- ✓ **Debe ser finito**, el algoritmo debe terminar en algún momento; o sea, debe tener un número finito de pasos.
- ✓ **Debe expresarse en un lenguaje estandarizado**, dos o más personas que entienden el lenguaje estandarizado, deben interpretar el algoritmo de la misma forma.
- ✓ **El resultado que produce al ser ejecutado debe ser previsible**. Si se ejecuta el algoritmo dos o más veces, se debe obtener el mismo resultado cada vez.

Cualquier algoritmo debe definir tres elementos**:**

1. **Entrada,** información o datos que se requieren para ejecutar el **Proceso**
2. **Proceso**, el conjunto de acciones que deben ejecutarse para convertir la información o datos de entrada en la **Salida**, y
3. **Salida**, lo que resultan de la ejecución del **Proceso**

En la siguiente figura se muestran esquemáticamente estos elementos:

Figura 1: Elementos que se definen en un algoritmo

2. METODOLOGÍA DE RESOLUCIÓN DE PROBLEMAS DE PROGRAMACIÓN

La metodología consta de un conjunto de fases que se deben realizar al momento de solucionar un problema mediante el computador.

Estas fases son:

1.- Análisis del problema.

2.- Diseño del algoritmo.

3.- Codificación en un Lenguaje de Programación.

4.- Compilación y ejecución.

5.- Verificación y depuración.

6.- Documentación.

Teniendo en cuenta los propósitos de este apunte, trataremos en profundidad sólo las primeras dos fases, Análisis del problema y Diseño del algoritmo.

2.1.- Análisis del problema

Esta fase se desarrolla con el propósito de comprender el problema y saber exactamente qué es lo que se desea obtener como resultado.

Dado un problema, el desarrollo de esta fase se puede orientar dando respuesta a las siguientes preguntas:

Pregunta 1: ¿Cuál es el problema que se ha de resolver?

Pregunta 2: ¿Cuál o cuáles son los resultados que debe producir la solución?

Pregunta 3: ¿De qué información se dispone para resolver el problema?

Pregunta 4: ¿Cuál o cuáles son las restricciones o condiciones que debe satisfacer la solución?

Es importante precisar que, aunque la Pregunta 1 pudiera considerarse innecesaria, pueden ocurrir situaciones en que se esté intentando resolver un problema que esta fuera del alcance de los conocimientos que el resolutor posee, en cuyo caso, será imposible tener éxito en la tarea de diseñar una solución.

Para ejemplificar, el desarrollo de esta fase, supongamos que se quiere resolver el problema de determinar el volumen de un cilindro cualquiera, a partir del radio de la base y la altura de este. Las respuestas a las preguntas anteriores podrían ser:

Respuesta a la pregunta 1: Se trata con el problema de calcular el volumen de un cilindro cualquiera.

Respuesta a la pregunta 2: El resultado que debe producir la solución es el valor del volumen de un cilindro en particular.

Respuesta a la pregunta 3: En el enunciado del problema no se expresa información, pero como se quiere calcular el volumen de un cilindro cualquiera, se requiere conocer el radio o el diámetro de la base del cilindro, y la altura de este.

Respuesta a la pregunta 4: Dadas las características del problema, los valores de radio o diámetro de la base y el valor de la altura que se usen para calcular el volumen deben ser positivos.

2.2.- Diseño del algoritmo.

Clarificado lo anterior, corresponde ahora diseñar el algoritmo de solución, en otras palabras, se debe pensar en el proceso y las acciones que este debe contener, para conseguir la solución al problema.

Una primera aproximación a la secuencia de acciones que deberían realizarse para resolver el problema podría ser:

i. **Leer radio**
ii. **Leer altura**
iii. **Verificar que los valores de radio y altura sean positivos**
iv. **Calcular volumen**
v. **Escribir el resultado**

A partir de esta secuencia de acciones, el paso siguiente es construir el algoritmo, para lo cual se debe usar un lenguaje específico y apropiado para ello.

Para este propósito, existen varios lenguajes algorítmicos, en nuestro caso haremos referencias a dos de ellos que son, los Diagramas de flujo DF y los pseudocódigos.

En las páginas siguientes abordamos los detalles de estos dos lenguajes para la construcción de algoritmos, sin embargo, aunque más adelante trataremos con el uso de estos lenguajes, cuando se resuelven problemas concretos, nos permitimos en este punto, anticipar las construcciones algorítmicas en DF y pseudocódigo para el problema que hemos venido tratando, las que mostramos en las imágenes siguientes.

DF

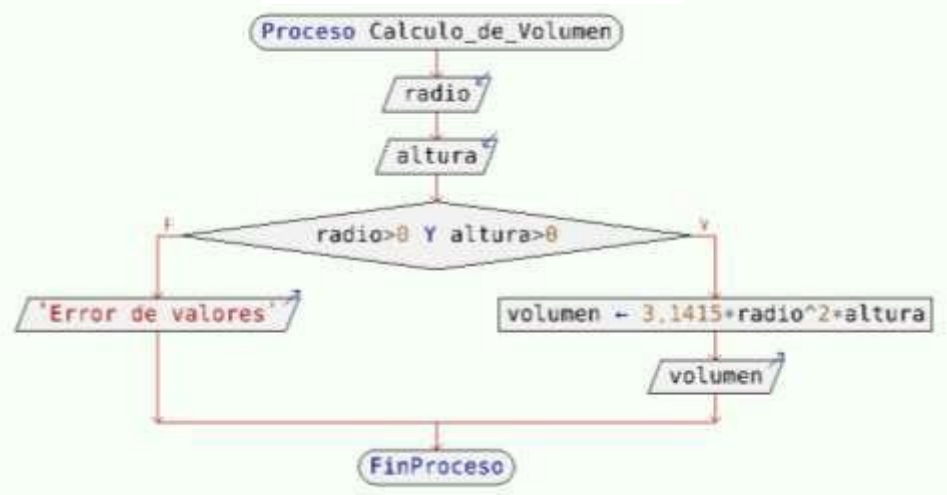


Figura 2: DF (Diagrama de Flujo) para el problema del cálculo del volumen de un cilindro

SEUDOGÓDIGO

```
Proceso Calculo_de_Volumen
    Leer radio;
    Leer altura;
    Si  radio>0 y altura>0 Entonces
        volumen<-3.1415*radio^2*altura;
        Escribir volumen;
    SiNo
        Escribir "Error de valores";
    Fin Si
FinProceso
```

Figura 3: Pseudocódigo para el problema del cálculo del volumen de un cilindro

3. LENGUAJES PARA LA REPRESENTACIÓN DE ALGORITMOS

La representación de algoritmos se realiza usando lenguajes específicos para tal propósito, como los que se muestran en las figuras 2 y 3. Los lenguajes que usaremos en este apunte son los **Diagramas de Flujo** y los **Pseudocódigos**.

3.1.- Diagrama de Flujo

Un **Diagrama de Flujo** es un lenguaje para la representación gráfica de un proceso, en el cual cada paso o acción de este, es representado mediante un símbolo particular, y además utiliza líneas con flechas para conectar los símbolos gráficos y expresar la secuencia y orden en que los pasos o acciones del proceso se irán ejecutando.

Así con estos recursos visuales el **Diagrama de Flujo** muestra una descripción gráfica de las actividades implicadas en un proceso, evidenciando la relación secuencial ente ellas, facilitando la rápida comprensión de cada actividad y su relación con las demás, y el flujo de la información, así como la forma en que se van tratando los datos de **entrada** hasta producir los datos de **salida**.

Dado su carácter gráfico, un **Diagrama de flujo** facilita la visualización de los detalles de las acciones que se ejecutan dentro de un proceso.

3.2.- Pseudocódigo

El **Pseudocódigo** ("casi código") es un lenguaje para la representación de algoritmos que utiliza palabras claves muy parecidas a las instrucciones o sentencias de un lenguaje de programación de computadores, por tanto, a diferencia de un **Diagrama de Flujo**, no utiliza símbolos gráficos conectados mediante flechas.

Por ser un lenguaje para la representación de algoritmos que se asemeja a los lenguajes de programación, genera la facilidad de que es una forma de representación que facilita la conversión del algoritmo a un lenguaje de programación de computadores.

Lógicamente, existe una equivalencia entre el lenguaje de representación de algoritmos basado en **Diagrama de flujo** y el basado en **Pseudocódigo**.

A continuación, se presentan los símbolos gráficos de **Diagramas de flujo** y su equivalencia en el lenguaje de **Pseudocódigos** que utilizaremos en este apunte.

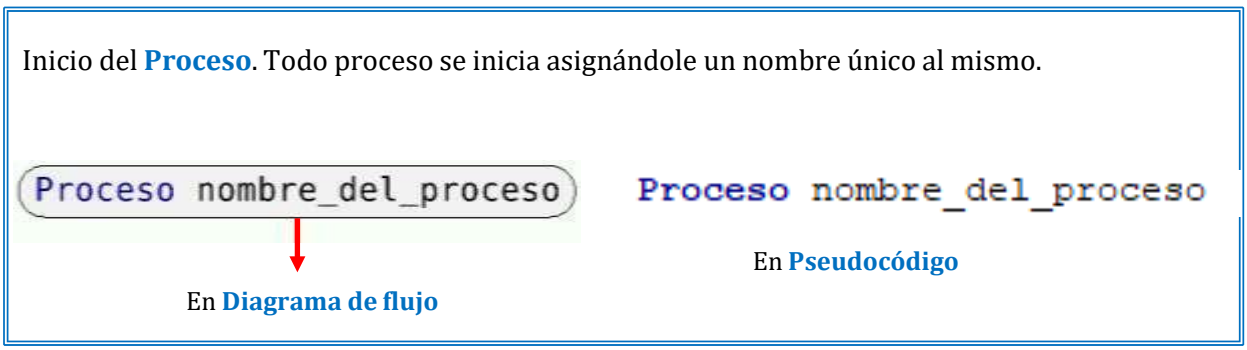
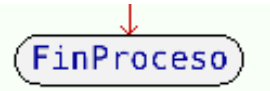


Figura 4: Representación de Inicio de Proceso

Fin del **Proceso**.



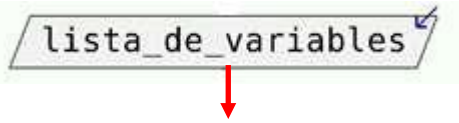
En **Diagrama de flujo**

```
FinProceso
```

En **Pseudocódigo**

Figura 5: Representación de Fin de Proceso

Leer datos. Esta es la acción para ingresar datos al proceso. Se pueden **Leer** varios datos identificándolos con nombres de variables único y separándolos por comas (,)



En **Diagrama de flujo**

```
Leer lista_de_variables;
```

En **Pseudocódigo**

Figura 6: Acción de Leer Datos

Acción de **Escribir**. Esta puede usarse para escribir valores de variables, resultados de expresiones matemáticas y contenidos textuales. Cada elemento a escribir debe separase por una coma (,)

Los contenidos textuales se declaran entre apostrofes en los **Diagramas de flujo** y entre comillas en los **Pseudocódigos**

En **Diagrama de flujo**

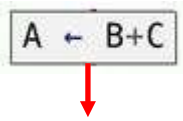
```
Escribir lista_de_variables;

Escribir "textos";
```

En **Pseudocódigo**

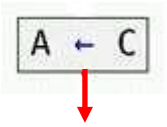
Figura 7: Acción de Escribir

Acción de **Asignar** a una variable el resultado de evaluar una expresión. La variable a la que se asigna está a la izquierda, y la expresión está a la derecha del signo de asignación.

Esta acción puede usarse también para asignar una variable a otra.




```
A<-B+C;
```



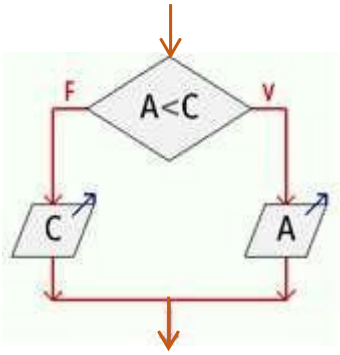

```
A<-C;
```

En **Diagrama de flujo**

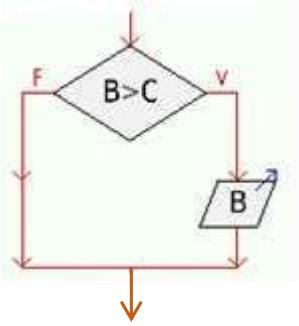
En **Pseudocódigo**

Figura 8: Acción de Asignar

Estructura para el control condicional del flujo del **Proceso**, Si la comparación lógica es **verdadera**, el proceso continúa con las acciones de la variante de la derecha (**V**), por el contrario si la comparación lógica es **falsa**, el proceso continua con las acciones de la variante de la izquierda (**F**). También es posible una estructura de control de flujo del **Proceso** conocida como condicional **simple**, ya que en ella sólo se indican las acciones que se ejecutaran si la comparación lógica es **verdadera**.




```
Si A<C Entonces
    Escribir A;
SiNo
    Escribir C;
Fin Si
```




```
Si B>C Entonces
    Escribir B;
Fin Si
```

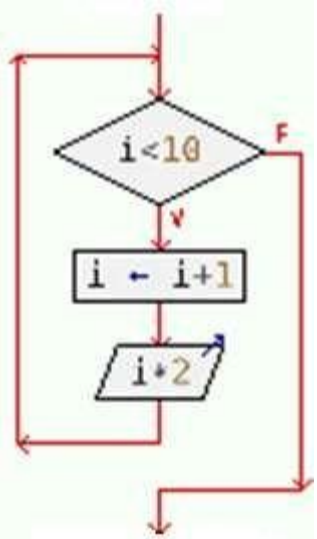
En **Diagrama de flujo**

En **Pseudocódigo**

Figura 9: Estructura para el control condicional del flujo del proceso

Estructura de control de flujo del **Proceso** para hacer que se ejecuten acciones si una cierta condición es **verdadera** (**V**). Si la condición es **falsa** (**F**), las acciones dejan de ejecutarse.

Esta estructura de control se conoce como estructura **Mientras** ya que las acciones se ejecutarán **Mientras** la condición sea verdadera (**V**), en caso contrario, dejarán de ejecutarse.



En **Diagrama de flujo**

```
Mientras i<10 Hacer
    i<-i+1;
    Escribir i*2;
Fin Mientras
```

En **Pseudocódigo**

Figura 10: Estructura de control Mientras.

4. HERRAMIENTA PARA LA ESCRITURA Y EDICIÓN DE ALGORITMOS EN PSEUDOCÓDIGO Y EN DIAGRAMAS DE FLUJO PSeInt

4.1.- Introducción

El programa conocido como **PSeInt**, debe su nombre a la contracción ***PSeudocode Interpreter***, (Intérprete de Pseudocódigo), y en su primera versión, fue el resultado del trabajo de proyecto final de la asignatura *Programación 1,* del estudiante Pablo Novara de la carrera *Ingeniería en Informática* de la Facultad de Ingeniería y Ciencias Hídricas de la Universidad Nacional del Litoral.

Según su autor, el principal propósito de la aplicación es ayudar al aprendizaje de diseño de algoritmos. Con los apoyos que brinda la herramienta, así, el aprendiz podrá poner sus esfuerzos intelectuales principalmente en los aspectos lógicos de la solución, en las acciones, la secuenciación de ellas como procesos, y en los aspectos referidos a la organización de estas como estructura de programación. En su versión actual, es un software de licencia GLP (General Public License)

Dentro de sus principales características podemos destacar:

- Es un software libre de licencia GPL y gratuito,
- Mutiplataforma, existen versiones para instalación en sistemas operativos Microsoft Windows de 32 y 64 bits, Mac OS i686 y Mac PowerPC, GNU/Linux de 32 y 64 bits
- Permite la definición de perfiles para adaptar los lenguajes de pseudocódigos y de diagrama de flujos diferentes estilos y sintaxis.
- La escritura de los algoritmos se puede hacer directamente como sentencias en pseudocódigo, o utilizando elementos de la ventana de comandos que representan las sentencias usando símbolos del lenguaje de diagramas de flujo.
- Para la escritura de los pseudocódigos se dispone de facilidades de: autocompletado; ayudas emergentes; plantillas de comandos; coloreado de sintaxis; resaltado de bloques lógicos; indentado inteligente; y listado de funciones, operadores y variables de uso común en programación.
- Permite generar y editar el diagrama de flujo a partir del pseudocódigo, con la opción de grabarlo como archivo de imagen.
- Permite la edición simultánea de múltiples algoritmos.
- Puede interpretar (ejecutar) los algoritmos escritos, contando con facilidades para: modificar el algoritmo y ver los cambios en la ejecución inmediatamente (sin reingresar los datos); modificar uno o más datos seleccionados de una ejecución ya finalizada para observar cómo varían los resultados; deshacer una ejecución para reiniciarla o repetirla desde un punto arbitrario; ejecutar el algoritmo paso a paso controlando la velocidad e inspeccionando variables y expresiones; confeccionar automáticamente una tabla de prueba de escritorio; y un modo especial en el que describe las acciones realizadas en cada paso.
- Determina y marca claramente los errores señalando: errores de sintaxis en tiempo real (mientras escribe); errores en tiempo de ejecución; y ofrece descripciones detalladas de cada error, con sus causas y soluciones más frecuente.

- Permite convertir el algoritmo de pseudocódigo a código en variados lenguajes de programación tales como: C, C++, C#, Java, JavaScript, MatLab, Pascal, PHP, Python 2, Python 3, QBasic Visual Basic

4.2.- Instalación

El software puede ser descargado desde http://pseint.sourceforge.net/

Luego de bajar y ejecutar el instalador, se abrirá la siguiente ventana:

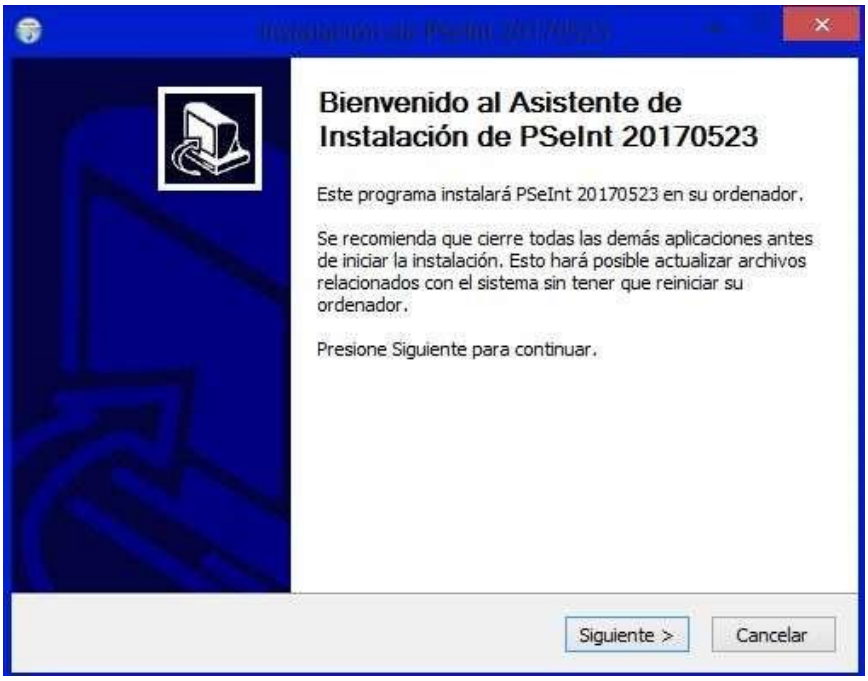


Figura 11: Ventana del instalador de PSeInt

Luego, el asistente del instalador le guiará hasta completar la instalación del programa.

4.3.- Entorno de PSeInt.

En la ventana de la aplicación se pueden identificar los siguientes elementos:

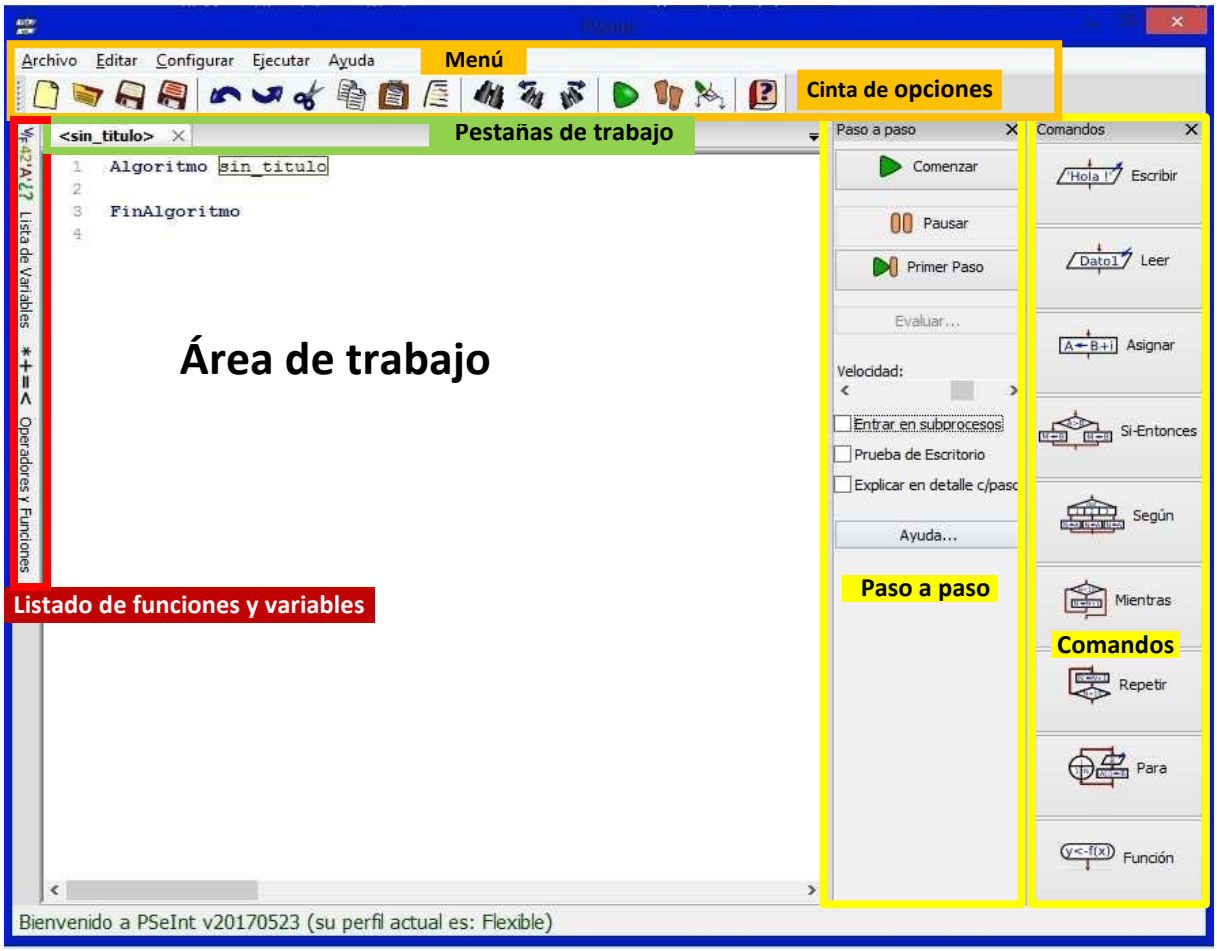


Figura 12: Elementos del Entorno de Trabajo

Menú Contiene las opciones de **Menú** para acceder a las siguientes acciones:

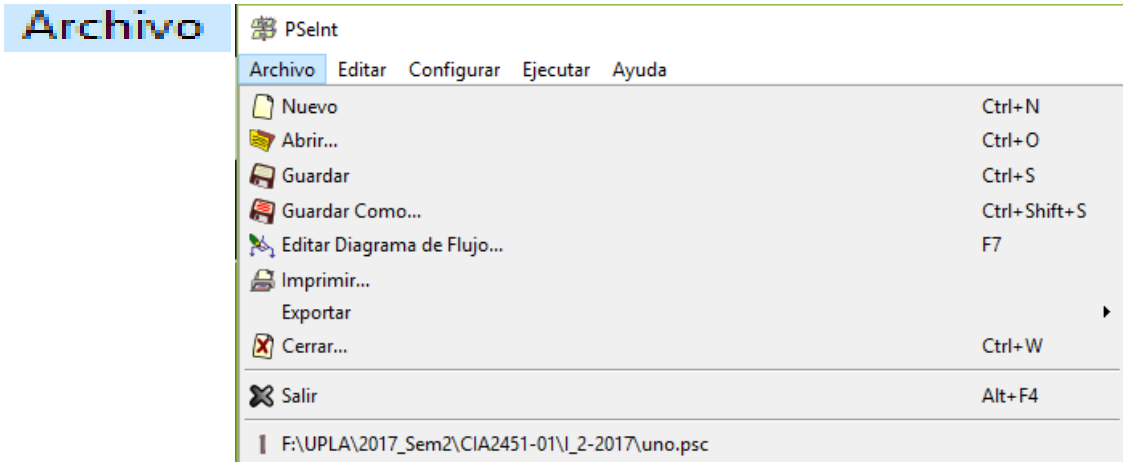


Figura 13: Acciones de la opción Archivo del Menú

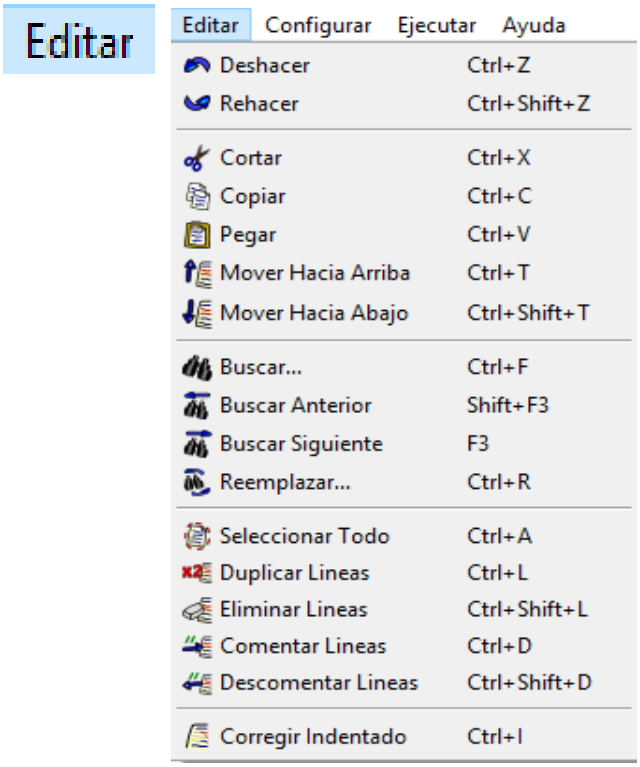


Figura 14: Acciones de la opción Archivo del Menú

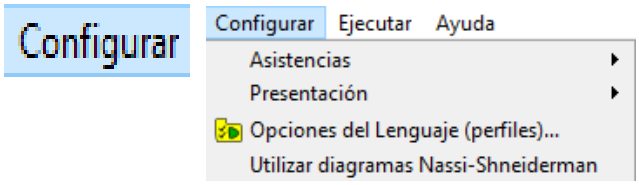


Figura 15: Acciones de la opción Configurar del Menú

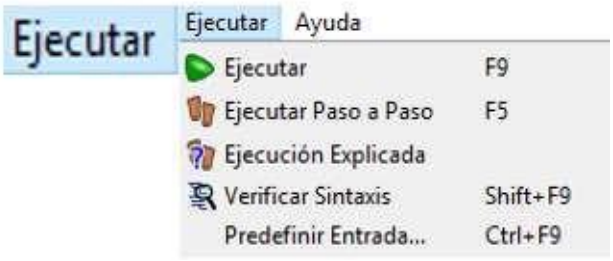


Figura 16: Acciones de la opción Ejecutar del Menú

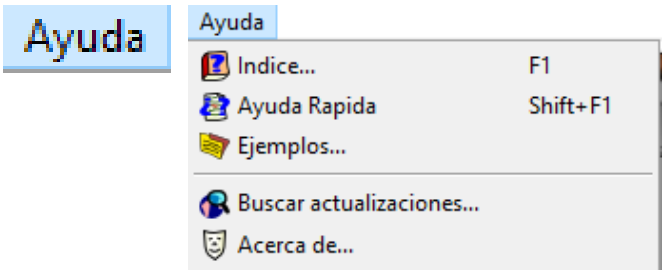


Figura 17: Acciones de la opción Ayuda del Menú

En tanto desde la **Cinta de opciones** se tiene acceso de forma directa a las siguientes acciones:

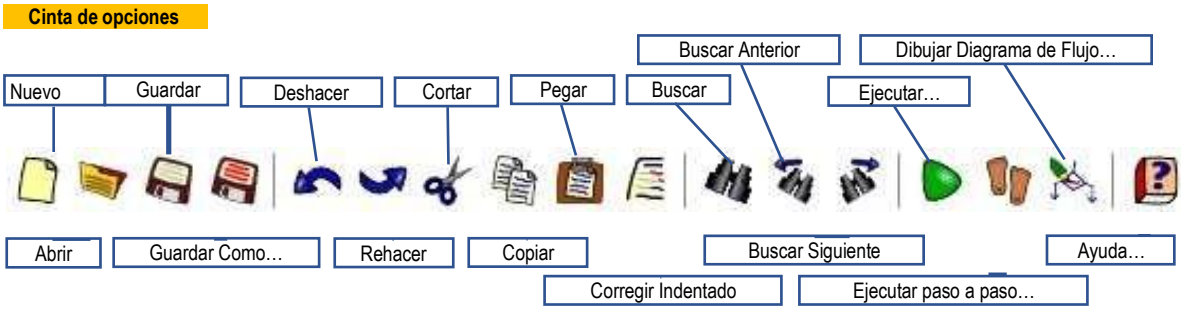


Figura 18: Acciones de la Cinta de opciones

Área de trabajo

Corresponde a la zona dónde se escribe el **Pseudocódigo** del algoritmo. El **Pseudocódigo** se puede escribir directamente, o escogiendo el comando correspondiente desde la ventana de comandos.

A la izquierda de cada línea de pseudocódigo se muestra un número entero positivo que es el identificador de la línea, cuya referencia se utiliza cuando se indican los errores.

Pestañas de trabajo Cada pseudocódigo nuevo o abierto se muestra en una pestaña diferente. Si se asigna un nombre al algoritmo, este se mostrará en la pestaña correspondiente, en caso contrario se muestra con el nombre <sin_titulo>

Listado de funciones y variables

Área de trabajo Corresponde a la zona vertical que se muestra en el borde izquierdo del

Esta zona está compuesta por dos pestañas que mostramos en posición horizontal en las siguientes imágenes:

Al activar la pestaña superior **Lista de Variables**, se abre una ventana que muestra un listado de las variables, según el orden en que aparecen en el **Pseudocódigo**.

Figura 19: Pestaña de Lista de Variables

Al activar la pestaña de **Operadores y Funciones**, se abre una ventana que muestra la lista ordenada por categorías y dentro de cada categoría, , los operadores y funciones que se pueden utilizar como parte del pseudocódigo.

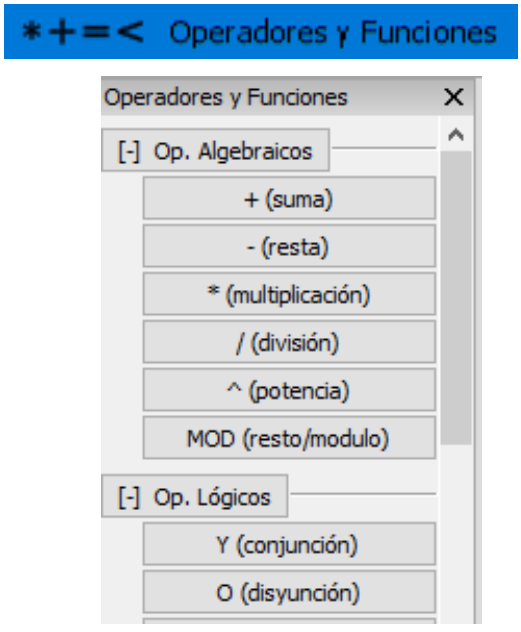


Figura 20: Pestaña y ventana de Operadores y Funciones

Comandos En esta ventana están disponibles como botones, los distintos comandos de sentencias y estructuras de control, expresados en lenguaje de **Diagrama de flujo (DF)**.

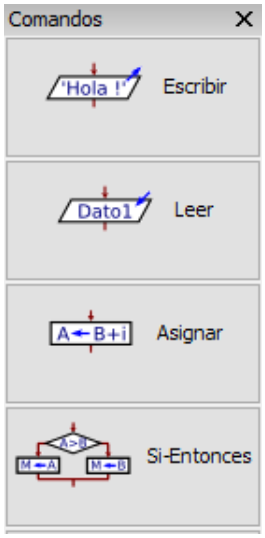


Figura 21: Ventana de Comandos

Al hacer doble clic sobre uno de los botones de la ventana de **Comandos**, se agrega en lenguaje de **Pseudocódigo**, la sentencia o estructura de control correspondiente, en la línea de edición que se encuentre activa en el **Área de trabajo**

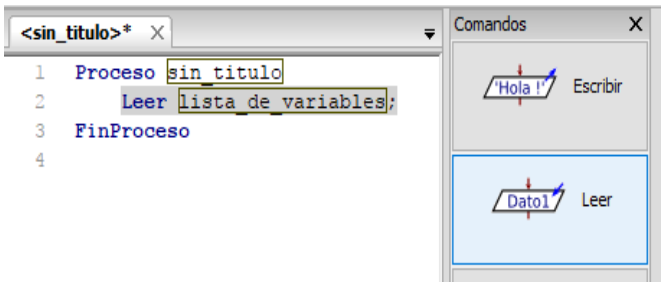


Figura 22: Uso de Comandos

Si la ventana de **Comandos** no está visible, puede hacer clic sobre la pestaña superior que se encuentra a la derecha del **Área de trabajo**

Figura 23: Activación de la Ventana de Comandos

Paso a paso

En esta venta se tiene acceso a las opciones para el control de la ejecución **Paso a paso**.

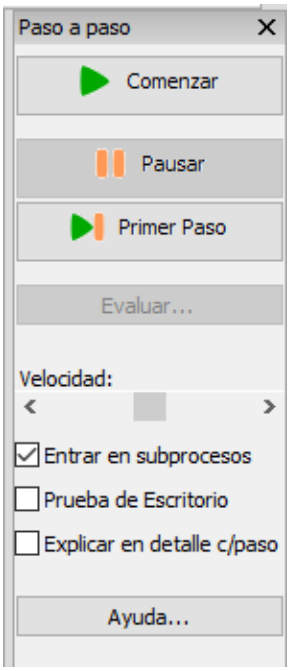


Figura 24: Ventana Paso a paso

Si la ventana **Paso a paso** no está visible, puede hacer clic sobre la pestaña superior que se encuentra a la derecha del **Área de trabajo**

Figura 25: Activación de la ventana Paso a paso

4.4.- Modo de uso de los Comandos básicos para escribir instrucciones y estructuras de control.

Teniendo en consideración que el propósito fundamental de este manual, es ser un apoyo para el aprendizaje del diseño de algoritmos de los estudiantes de un primer curso de programación, sumado a ello la creencia de que el uso de esta aplicación les será de utilidad para el aprendizaje. A continuación, se muestran los aspectos generales de la escritura y significancia de los comandos de instrucciones y estructuras de control.

Figura 26: Comando Escribir

Este comando refiere a la sentencia para escribir. La acción de escribir puede hacerse respecto de un contenido textual, o una variable, o una expresión, o combinaciones de estos elementos.

Al hacer doble clic sobre este comando, en el área de trabajo se mostrará el equivalente en pseudocódigo de esta sentencia, la que corresponde a:

```
Escribir lista_de_expresiones;
```

donde `lista_de_expresiones;` puede ser alguna de las siguientes:

- Un contenido textual, para lo cual el contenido debe ponerse entre comilla quedando como:

```
Escribir "Contenido Textual";
```

- El valor de una variable y/o expresión como: `Escribir a,a+b;`

  En este ejemplo, se quiere escribir el valor de una cierta variable a y el resultado de sumar el valor de la misma variable a más otra cierta variable b. Notar que cada elemento a escribir se separa por una coma (,)

- En el siguiente ejemplo se quiere escribir un contenido textual y luego el valor de una cierta variable a

```
Escribir "El resultado es",a;
```

Figura 27: Comando Leer

Este comando refiere a la sentencia para leer. Las acciones de lectura se realizar para ingresar valores de variables (datos) durante la ejecución del algoritmo.

Al hacer doble clic sobre este comando, en el área de trabajo se mostrará el equivalente en pseudocódigo de esta sentencia, la que corresponde a:

```
Leer lista_de_variables;
```

`lista_de_variables` puede contener una o más variables separadas por coma (,)

```
Leer a,b;
```

Figura 28: Comando Asignar

Mediante este comando se declara la acción de Asignar a una variable (la que se encuentra al lado izquierdo de ←) el contenido de otra variable o el resultado de una expresión (que se encuentra a lado derecho de ←)

Al hacer doble clic sobre este comando, en el área de trabajo se mostrará el equivalente en pseudocódigo de esta sentencia, la que corresponde a:

```
variable<-expresion;
```

Ejemplos de la forma que puede esta sentencia, son: `p<-b;` `c<-a+b;`

Figura 29: Comando Si-Entonces

Este comando permite incorporar en el algoritmo a una estructura de control mediante la cual, se define que sentencia o conjunto de sentencias se ejecutarán si una cierta expresión_lógica es VERDADERA y que sentencias o conjunto de ellas se ejecutaran si la misma expresión_lógica es FALSA.

Al hacer doble clic sobre este comando, en el área de trabajo se mostrará el equivalente en pseudocódigo de esta estructura de control, la que corresponde a:

```
Si expresion_logica Entonces
    acciones_por_verdadero
SiNo
    acciones_por_falso
Fin Si
```

Figura 30: Estructura Si-Entonces

Un ejemplo de uso de esta estructura podría ser:

```
Si a>=b Entonces
    Escribir "a es mayor o igual que b";
SiNo
    Escribir "b es mayor que a";
Fin Si
```

Figura 31: Ejemplo de Si-Entonces

De ser necesario, esta estructura también puede ser usada en su versión simplificada como:

```
Si a>b Entonces
    Escribir "a es mayor que b";
Fin Si
```

Figura 32: Ejemplo de Si-Entonces sin sino

para lo cual se deben eliminar las líneas correspondientes al `SiNo` y las siguientes, hasta la anterior al `Fin Si`

En `expresion logica` la pueden usarse operadores lógicos tales como & ó Y (Conjunción), | ó O (Disyunción), ~ ó NO (Negación).

Ejemplo:

```
Si a>b O a=b Entonces
    Escribir "a es mayor o igual que b";
SiNo
    Escribir "b es mayor que a";
Fin Si
```

Figura 33: Ejemplo de uso de operadores lógicos en Si-Entonces

Figura 34: Comando Mientras

Este comando incorpora en el algoritmo, la estructura de control Mientras para repetir una sentencia o conjuntos de sentencias.

Al hacer doble clic sobre el comando, a partir de la línea correspondiente del pseudocódigo se agregará la estructura:

```
Mientras expresion_logica Hacer
    secuencia_de_acciones
Fin Mientras
```

Figura 35: Estructura Mientras

Al igual que en el caso de la estructura de control Si-Entonces anterior, la `expresion logica` puede contener una comparación simple de la forma a<7, o usar los operadores lógicos & ó Y, | ó O, ~ ó NO.

Ejemplo:

```
Mientras i<5 O i=5 Hacer
    i<-i+1;
    Escribir "Sigo sumando 1 a i";
Fin Mientras
```

Figura 36: Ejemplo uso de operadores lógicos en Mientras

Figura 37: Comando Repetir

Este comando incorpora en el algoritmo, la estructura de control Repetir la que corresponde a otra estructura para repetir una sentencia o conjuntos de sentencias.

Al hacer doble clic sobre el comando, a partir de la línea correspondiente del pseudocódigo se agregará la estructura:

```
Repetir
    secuencia_de_acciones
Hasta Que expresion_logica
```

Figura 38: Estructura Repetir

Al igual que en los casos `expresion logica` anteriores, la puede utilizar una expresión lógica simple u operadores lógicos

Figura 39: Comando Para

Este es el comando para la tercera forma de incorpora una estructura para repetir una sentencia o conjuntos de sentencias, dentro de un algoritmo

Al hacer doble clic sobre el comando, a partir de la línea correspondiente del pseudocódigo se agregará la estructura:

```
Hasta Que expresion_logica
Para variable_numerica<-valor_inicial Hasta valor_final Con Paso paso Hacer
    secuencia_de_acciones
Fin Para
```

Figura 40: Estructura Para

5. EJEMPLOS DE PROBLEMAS RESUELTOS DE DISEÑO DE ALGORITMOS

5.1.- Algoritmos de estructura Secuencial

Un **algoritmo de estructura secuencial** tiene la característica de que, al ser ejecutado, siempre ejecutará las acciones en el mismo orden (secuencia).

Ejemplo 1:

Diseñe un algoritmo representándolo en DF y en pseudocódigo que calcule y muestre la suma de dos números cualquiera.

Análisis problema

- Datos de entrada: Por tratarse de números cualquiera, se deben ingresar los dos números.
- Datos de salida: El resultado de la suma de los dos números.
- Restricciones: No existen restricciones ya que pueden ser números cualesquiera.
- Proceso: Calcular la suma de dos números ingresados.

Diseño de la solución

- Ingreso de dos números.
- Suma de los dos números.
- Mostrar el resultado de la suma.

Representación en Diagrama de Flujo (DF)

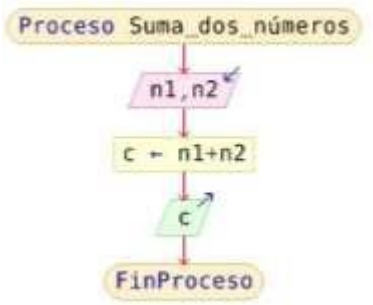


Figura 41: Diagrama de flujo (DF) Suma de dos números cualquiera

Representación en Pseudocódigo

```
1  Proceso Suma_dos_números
2      Leer n1,n2;
3      c<-n1+n2;
4      Escribir c;
5  FinProceso
```

Figura 42: Pseudocódigo Suma de dos números cualquiera

Ejemplo 2:

Diseñe un algoritmo representándolo en DF y pseudocódigo para calcular y mostrar el valor la función $f(x) = 3x + 2$ para un valor cualquiera

Análisis problema

- Datos de entrada: Un valor cualquiera para x
- Datos de salida: El resultado del valor de la función $f(x)$ para el x ingresado.
- Restricciones: No existen restricciones.
- Proceso: Calcular el valor de la función $f(x)$para el valor de x ingresado.

Diseño de la solución

- Ingreso de un valor.
- Calcular el valor de la función.
- Mostrar el valor calculado para la función.

Representación en DF

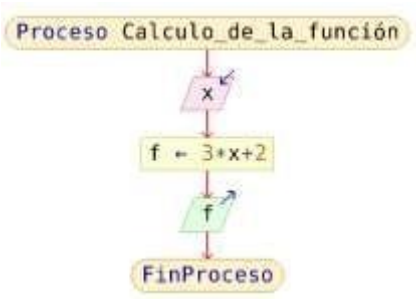


Figura 43: DF cálculo de la función f(x)

Representación en Pseudocódigo

```
1  Proceso Calculo_de_la_función
2      Leer x;
3      f<-3*x+2;
4      Escribir f;
5  FinProceso
```

Figura 44: Pseudocódigo cálculo de la función f(x)

5.2.- Algoritmos de estructura Selectiva

Este tipo de estructuras algorítmicas son utilizadas cuando se requiere que el proceso o parte de él, no tiene una única secuencia. El proceso que no es secuencial o la parte del proceso que no lo es, es controlada por una condición lógica.

Por tanto, se evalúan una condición y en función del resultado de esta se realiza una opción u otra. Las condiciones se especifican usando expresiones lógicas.

Existen dos casos generales de este tipo de estructuras selectivas, las de tipo **selectivas simples** y las del tipo **selectivas dobles**.

5.2.1.- Algoritmos de estructura Selectiva Simple

Las estructuras **Selectivas Simples** están compuesta por una condición lógica, la que al ser verdadera, continuará con la ejecución de una o más acciones, si por el contrario, la condición es falsa no hará nada.

Ejemplo 3:

Diseñe un algoritmo representándolo en DF y Pseudocódigo para mostrar un mensaje si un cierto valor es positivo.

Análisis problema

- Datos de entrada: Un valor cualquiera
- Datos de salida: Mostrar un mensaje si el valor es positivo.
- Proceso: Determinar si el valor es positivo.

Diseño de la solución

- Ingreso de un valor.
- Determinar si el valor es positivo.
- Mostrar el mensaje si es positivo.

Representación en DF

Figura 45: DF; Verifica si un valor es positivo

Comentario: Observe en el DF (figura 37) que mediante una condición lógica se evalúa si el valor leído es positivo. Si la condición lógica resulta ser verdadera (V) el proceso continúa por la variante de la derecha, y luego el proceso completo finaliza, por el contrario, si la condición lógica resulta ser falsa (F) el proceso continúa por la variante de la izquierda, con lo cual el mismo finaliza.

Representación en Pseudocódigo

```
1  Proceso Verifica_si_es_positivo
2      Leer n;
3      Si n>0 Entonces
4          Escribir "El valor es positivo";
5          Fin Si
6  FinProceso
```

Figura 46: Pseudocódigo; Verifica si un valor es positivo

Comentario: Observe en el Pseudocódigo (figura 38) que mediante una condición lógica (línea 3), se evalúa si el valor leído es positivo. Si la condición lógica resulta ser verdadera el proceso continúa en la línea 4, luego la condición termina en la línea 5, y finaliza el proceso en la línea 6. Por el contrario, si la condición lógica resulta ser falsa el proceso finaliza (línea 6).

Ejemplo 4:

Diseñe un algoritmo representándolo en DF y Pseudocódigo para calcular y mostrar la suma de dos números cualquiera, solo si estos son distintos.

Análisis problema

- Datos de entrada: Dos números cualquiera.
- Datos de salida: Mostrar el resultado de la suma de los dos números.
- Proceso: Determinar si los números son distintos, y sumarlos si es el caso.

Diseño de la solución

- Ingreso de dos números cualquiera.
- Determinar si los números son distintos, y sumarlos si es el caso.
- Mostrar el resultado de la suma de los dos números.

Representación en DF

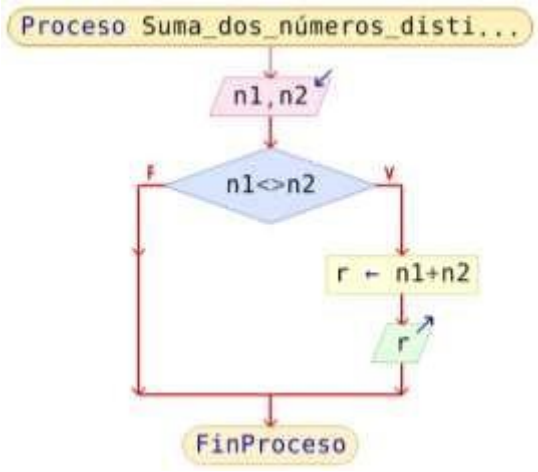


Figura 47: DF; Suma de dos números distintos

Comentario: Observe en el DF (figura 47) que mediante una condición lógica se evalúa si los números ingresados son distintos. Si la condición lógica resulta ser verdadera (V) el proceso continúa por la variante de la derecha, con lo cual se suman los dos números y se muestra el resultado, luego el proceso completo finaliza. Por el contrario, si la condición lógica resulta ser falsa (F) el proceso continúa por la variante de la izquierda, y finaliza.

Representación en Pseudocódigo

```
1  Proceso Suma_dos_números_distintos
2      Leer n1,n2;
3      Si n1<>n2 Entonces
4          r<-n1+n2;
5          Escribir r;
6      Fin Si
7  FinProceso
```

Figura 48: Pseudocódigo; Suma de dos números distintos

Comentario: Observe en el Pseudocódigo (figura 48) que mediante la condición lógica (línea 3), se evalúa si los números ingresados son distintos. Si la condición lógica resulta ser verdadera el proceso continúa en las líneas 4 y 5, con lo cual se suman los dos números y se muestra el resultado respectivamente, para luego el proceso completo finalizar. Por el contrario, si la condición lógica resulta ser falsa el proceso continúa en la línea 7 finalizando.

Para verificar si los números son distintos (línea 3), se usó el operador lógico <>, pero también es posible usar el operador lógico != que es equivalente.

5.2.2.- Algoritmos de estructura Selectiva Doble

Las estructuras **Selectivas Dobles** están compuesta por una condición lógica, la que sí es verdadera ejecutara una acción o acciones, pero si es falsa, ejecutará una acción o acciones diferentes.

Ejemplo 5:

Diseñe un algoritmo representándolo en DF y Pseudocódigo para mostrar el mensaje APROBADO si la nota final de un estudiante es mayor o igual que 4,0, y el mensaje REPROBADO en caso contrario.

Análisis problema

- Datos de entrada: La nota final de un estudiante
- Datos de salida: El mensaje **APROBADO** o **REPROBADO** según corresponda.
- Proceso: Determinar si la nota final ingresada es igual o mayor que 4,0 en cuyo caso se muestra el mensaje **APROBADO**, en caso contrario, se muestra el mensaje **REPROBADO**.

Diseño de la solución

- Ingreso de una nota.
- Determinar si el valor es mayor o igual que 4,0, de ser así se muestra el mensaje **APROBADO**, en caso contrario de muestra el mensaje **REPROBADO**

Representación en DF

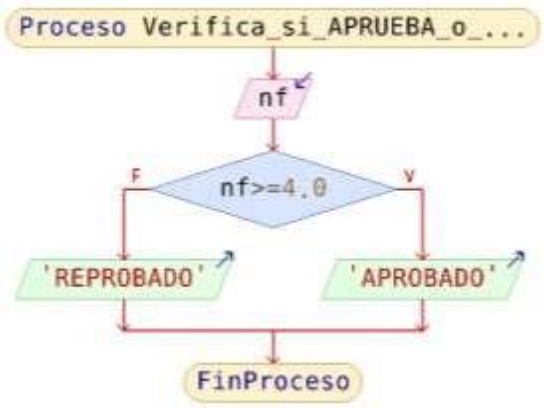


Figura 49: DF; Verifica si Aprueba o Reprueba

Comentario: Observe en el DF (figura 49) que mediante la condición lógica se evalúa si el valor de nf leído es mayor o igual que 4,0. Si la condición lógica resulta ser verdadera (V) el proceso continúa por la variante de la derecha, escribiendo el mensaje APROBADO, luego el proceso completo finaliza. Por el contrario, si la condición lógica resulta ser falsa (F) el proceso continúa por la variante de la izquierda, escribiendo el mensaje REPROBADO, y luego el proceso completo finaliza. En consecuencia, existen acciones distintas para cada resultado de la condición lógica.

Representación en Pseudocódigo

```
1  Proceso Verifica_si_APRUEBA_o_REPRUEBA
2      Leer nf;
3      Si nf>=4.0 Entonces
4          Escribir "APROBADO";
5      SiNo
6          Escribir "REPROBADO";
7      Fin Si
8  FinProceso
```

Figura 50: Pseudocódigo; Verifica si Aprueba o Reprueba

Comentario: Observe en el Pseudocódigo de la figura 50 que si el valor de nf es mayor o igual que 4,0 (línea 3), se realiza la acción de escribir APROBADO (línea 4), luego de lo cual el proceso finaliza (línea 8). Por el contrario, si la condición lógica resulta ser falsa el proceso continuará en el SiNo (línea 5) por lo que se escribirá REPROBADO (línea 6), y luego el proceso finaliza (línea 8).

Ejemplo 6:

Este ejemplo es una variante del anterior (Ejemplo 5) dónde se quiere resolver el mismo caso, pero considerando que el valor de nf debe cumplir con el requisito de ser una nota que se encuentre en el rango 1,0 a 7,0. Esta exigencia constituye una restricción que debe cumplir la solución.

Análisis problema

- Datos de entrada: La nota final de un estudiante
- Datos de salida: El mensaje **APROBADO** o **REPROBADO** según corresponda.
- Proceso: Determinar si la nota final ingresada es igual o mayor que 4,0 en cuyo caso se muestra el mensaje **APROBADO**, en caso contrario, se muestra el mensaje **REPROBADO**.
- Restricción: La nota final ingresada debe estar en el rango 1,0 a 7,0, de no ser así el algoritmo mostrará un mensaje apropiado, y no mostrará el mensaje **APROBADO** ni **REPROBADO.**

Diseño de la solución

- Ingreso de una nota.
- Verificar si la nota ingresada está en el rango 1,0 a 7,0 de ser así continúa con el paso siguiente, en caso contrario, muestra un mensaje apropiado y finaliza.
- Determinar si el valor es mayor o igual que 4,0, de ser así se muestra el mensaje **APROBADO**, en caso contrario de muestra el mensaje **REPROBADO**

Representación en DF

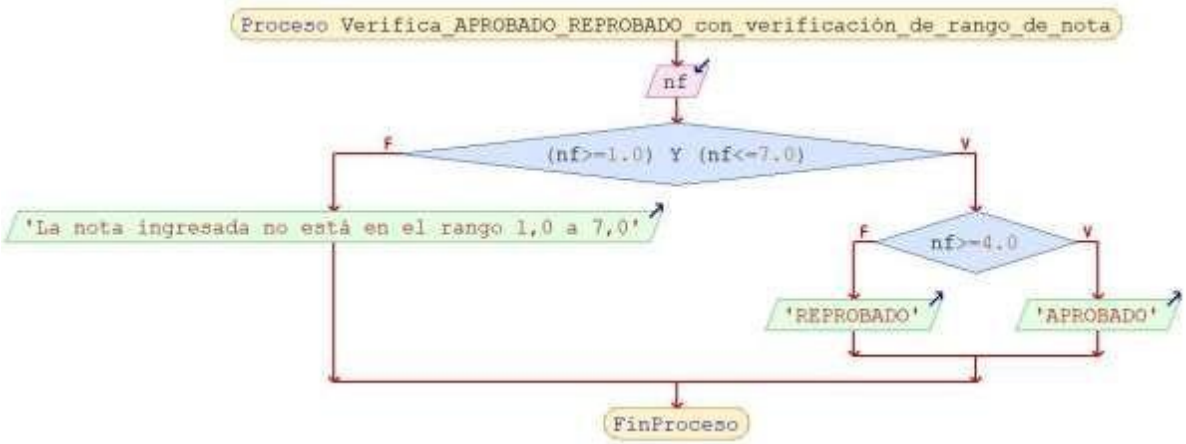


Figura 51: DF; Aprobado Reprobado con verificación de nota

Comentario: Observe en el DF anterior que en la primera condición cuyo propósito es verificar si el nf leído está en el rango 1.0 a 7.0, se está usando el conector lógico Y para asegurar que, si la condición lógica es verdadera, es porque se han cumplido las dos condiciones. Por cuestiones de más claridad en la sintaxis se aconseja que cada condición lógica se escriba entre paréntesis ().

Representación en Pseudocódigo

```
1   Proceso Verifica_APROBADO_REPROBADO_con_verificación_de_rango_de_nota
2       Leer nf;
3       Si (nf>=1.0) Y (nf<=7.0) Entonces
4           Si nf>=4.0 Entonces
5               Escribir 'APROBADO';
6           SiNo
7               Escribir 'REPROBADO';
8           FinSi
9       SiNo
10          Escribir 'La nota ingresada no está en el rango 1,0 a 7,0';
11      FinSi
12  FinProceso
```

Figura 52: Pseudocódigo; Aprobado Reprobado con verificación

Comentario: En la línea 3 del Pseudocódigo de la figura 52 se realiza la verificación para saber si la nota nf está en el rango 1,0 a 7,0. Si esto es cierto, se evalúa si la misma es mayor o igual que 4,0 en cuyo caso se escribe 'APROBADO', luego de lo cual el proceso continuaría en la línea 12 finalizando. Por el contrario, si nf no cumple esta última condición anterior, se escribe 'REPROBADO' luego de lo cual el proceso continuaría también en la línea 12 finalizando.

Por otra parte, si la condición lógica de la línea 3 resulta ser falsa, el proceso continuará en la línea 9 para escribir el mensaje 'La nota ingresada no está en el rango 1,0 a 7,0' luego de lo cual el proceso finalizará en la línea 12.

Ejemplo 7:

Diseñe un algoritmo representándolo en DF y Pseudocódigo que determine y muestre las raíces que son solución de una ecuación de segundo grado escrita de la forma $ax^2 + bx + c = 0$. El algoritmo debe ser capaz de tratar con ecuaciones de soluciones imaginarias.

Análisis problema

- Datos de entrada: Los coeficientes a, b y c
- Datos de salida: Las soluciones de la ecuación indicando si son imaginarias, si es el caso. Eventualmente la salida podría ser el mensaje EL COEFICIENTE a no puede ser cero, para el caso que corresponda.
- Proceso: Verificar que el coeficiente a sea distinto de cero, de ser así, determinar el argumento de la raíz y verificar que es cero o positivo, en cuyo caso calcular y mostrar las soluciones de la ecuación. Si el argumento de la raíz es negativo, obtener su valor absoluto y calcular y mostrar las soluciones de la ecuación indicando que las soluciones son imaginarias.
- Si el coeficiente $a$es cero mostrar el mensaje correspondiente.
- Restricción: El coeficiente a no puede ser cero para el caso de una ecuación de este tipo.

Diseño de la solución

- Ingreso de los coeficientes.
- Verificar si el coeficiente a es distinto de cero, de ser así, determinar el argumento de la raíz y verificar que es cero o positivo, en cuyo caso calcular y mostrar las soluciones de la ecuación, para luego finalizar. Si el argumento de la raíz es negativo, obtener su valor absoluto y calcular y mostrar las soluciones de la ecuación indicando que las soluciones son imaginarias, y finalizar
- Si el coeficiente $a$es cero mostrar el mensaje correspondiente y finalizar.

Representación en DF

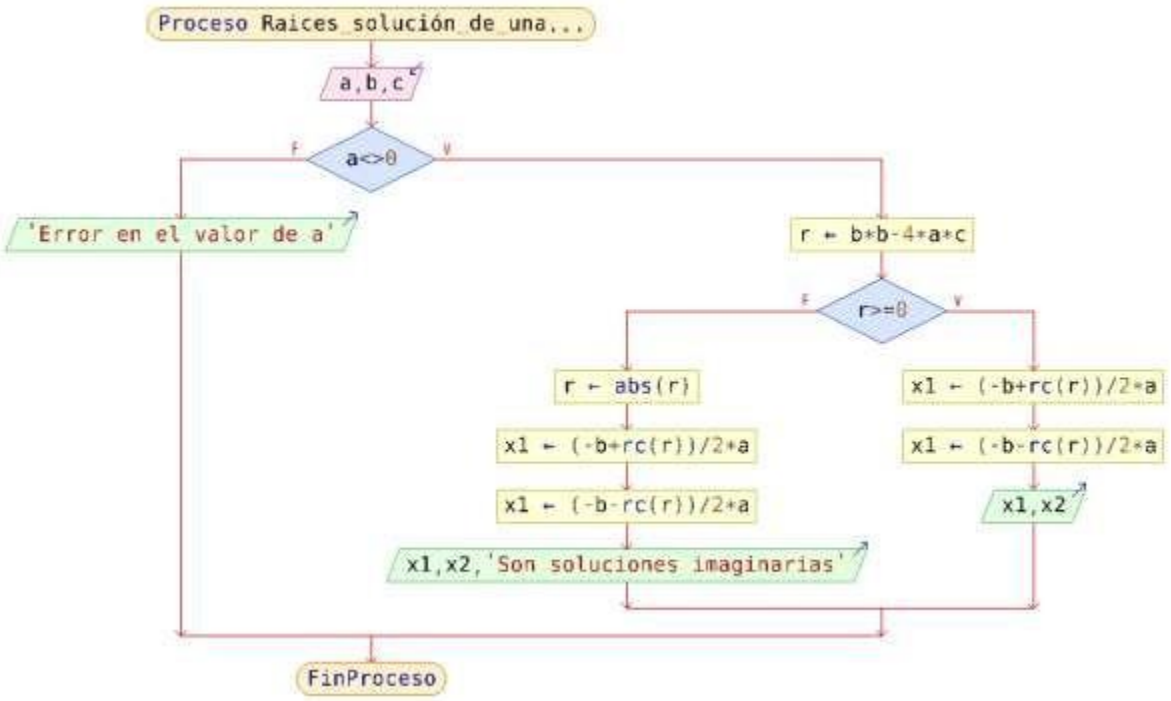


Figura 53: DF; Solución para las raíces de una ecuación de 2º grado

Comentario: Después de leer los coeficientes a, b, c se verifica que a sea distinto de cero. De ser así, el proceso continúa por la variante de la derecha y se determina el valor del argumento de la raíz de la fórmula para resolver este tipo de ecuaciones. A continuación de esto, se verifica que el valor del argumento de la raíz sea positivo o cero, en cuyo caso, el proceso sigue por la variante de la derecha (V) con la determinación de las dos soluciones $x1, x2$ para luego escribir los valores de las soluciones y finalizar el proceso. Se usa la función $rc()$ para obtener la raíz cuadrada de r

Por el contrario, si se verifica que el valor del argumento de la raíz no es positivo, el proceso continúa por la izquierda (F), convirtiendo r a su valor absoluto mediante la aplicación de la función $abs()$ y determinando los valores de las soluciones con el valor de r convertido a positivo, y luego escribiendo sus valores junto con el mensaje 'Son soluciones imaginarias'

Por último, si la condición para verificación del valor a resulta ser falsa (F), el proceso continuará por la izquierda, con lo cual se escribirá el mensaje 'Error en el valor de a' y el proceso finalizará.

Representación en Pseudocódigo

```
1   Proceso Raices_solución_de_una_ecuac_grado2
2       Leer a,b,c;
3       Si a<>0 Entonces
4           r <- b*b-4*a*c;
5           Si r>=0 Entonces
6               x1 <- (-b+rc(r))/2*a;
7               x1 <- (-b-rc(r))/2*a;
8               Escribir x1,x2;
9           SiNo
10              r <- abs(r);
11              x1 <- (-b+rc(r))/2*a;
12              x1 <- (-b-rc(r))/2*a;
13              Escribir x1,x2,'Son soluciones imaginarias';
14          FinSi
15      SiNo
16          Escribir 'Error en el valor de a';
17      FinSi
18  FinProceso
```

Figura 54: Pseudocódigo; Raíces Solución de una ecuación de 2º grado

Comentario: Como se observa en la figura anterior (figura 54), en la línea 3 se verifica que a sea distinto de cero. De ser así, el proceso continúa en la línea 4 con la determinación del valor del argumento de la raíz. A continuación, en la línea 5 se verifica que el valor del argumento de la raíz sea positivo o cero, en cuyo caso, el proceso sigue en la línea 6 con la determinación de las dos soluciones $x1, x2$ (líneas 6 y 7) para luego escribir los valores de las soluciones (línea 8) y finalizar el proceso en la línea 18.

Si se verifica que el valor del argumento de la raíz no es positivo, el proceso continúa, convirtiendo r a su valor absoluto en la línea 10, determinado las soluciones y escribiéndolas junto al mensaje correspondiente para este caso.

Por último, si la condición de la línea 3 es falsa, el proceso continuará en la línea 16 escribiendo el mensaje y finalizando.

5.3.- Estructuras Repetitivas

Las estructuras repetitivas son aquellas que permiten la ejecución de un conjunto de acciones o instrucciones varias veces dentro de un algoritmo. El número de veces que dichas acciones o instrucciones se llevan a cabo, se pueden especificar a través de una condición lógica que, al ser evaluada, establece si se continúa con la repetición o la misma finaliza.

Las estructuras repetitivas están compuestas por tres partes:

- La condición lógica,
- Las acciones o instrucciones que se repiten, y
- La salida de la repetición

Existen tres tipos de estructuras repetitivas:

- Mientras
- Repetir hasta
- Para

5.3.1.- Mientras.

Permite ejecutar, repetidamente, (cero o más veces) un bloque de acciones o instrucciones, mientras que, una determinada condición sea verdadera.

El caso más simple de este tipo de repeticiones es cuando la propia repetición está controlada por un contador, como es el caso del siguiente ejemplo:

Ejemplo 8:

Escriba un algoritmo en DF y Pseudocódigo que muestre las raíces cuadradas de todos los números entre 11 y 30.

Análisis problema

- Datos de entrada: No se requieren datos de entrada ya que se sabe que los números corresponden a los que están entre 11 y 30

  Datos de salida: Las raíces cuadradas de todos los números entre 11 y 30.
- Proceso: Inicializar un contador en 11 el cual debe llegar hasta 30, con incrementos de 1, y para cada número mostrar la raíz cuadrada.
- Restricción: No existen restricciones.

Diseño de la solución

- Inicializar un contador en 11 el cual debe llegar hasta 30, con incrementos de 1, y para cada número mostrar la raíz cuadrada. Luego de mostrar todas las raíces, finalizar.

Representación en DF

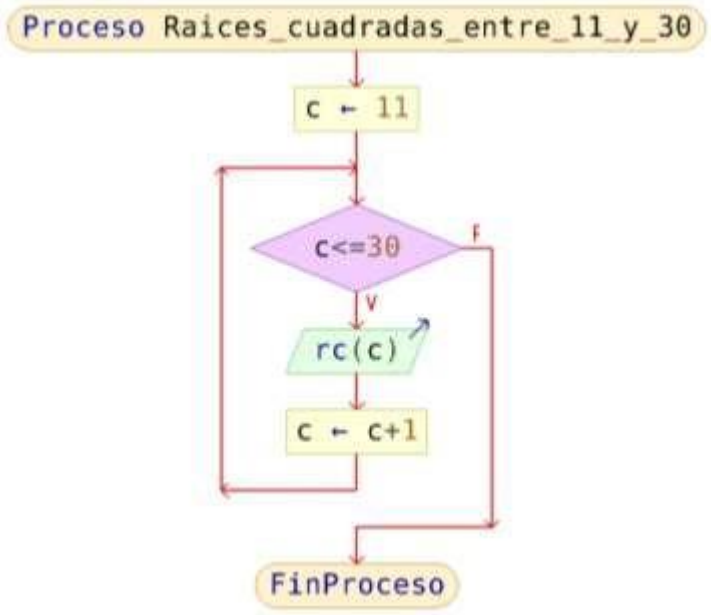


Figura 55: DF; Raíces cuadradas entre 11 y 30

Comentario: Como se observa en la figura anterior (figura 55), después de iniciado el proceso, se asigna a c el valor 11. Esta acción se denomina inicializador del contador. Esta acción es necesaria para asegurar que el contador c parte en el valor que el problema requiere.

c es el contador ya que mediante él se controla el número de repeticiones que se necesitan realizar.

Después de escribir el valor de la raíz cuadrada de c es necesario asignar 1 más al valor de c, ya que con ello se asegura que en algún momento el valor de c pasará de 30, con lo cual las repeticiones terminaran.

Representación en Pseudocódigo

```
1  Proceso Raices_cuadradas_entre_11_y_30
2      c<-11;
3      Mientras c<=30 Hacer
4          Escribir rc(c);
5          c<-c+1;
6      Fin Mientras
7  FinProceso
```

Figura 56: Pseudocódigo; Raíces cuadradas entre 11 y 30

Comentario: En la línea 2 se inicializa el contador. En la línea 3 se maneja la condición lógica para controlar la repetición, la cual terminará cuando el contador c deje de ser menor o igual que 30.

En la línea 5 se incrementa en 1 el contador c para asegurar que la repetición termine, luego de lo cual terminará el proceso en la línea 7.

Ejemplo 9:

Diseñe un algoritmo representándolo en DF y Pseudocódigo que muestre todos los divisores de un número entero positivo cualquiera.

Análisis problema

- Datos de entrada: Un número entero positivo

  Datos de salida: Todos los divisores del número.
- Proceso: Leer un número, verificar si es entero y positivo y, determinar y mostrar todos sus divisores.
- Restricción: El algoritmo sólo debe tratar con números enteros y positivos

Diseño de la solución

- Leer un número, verificar que sea entero y positivo, de ser así, determinar y mostrar todos sus divisores.

Representación en DF

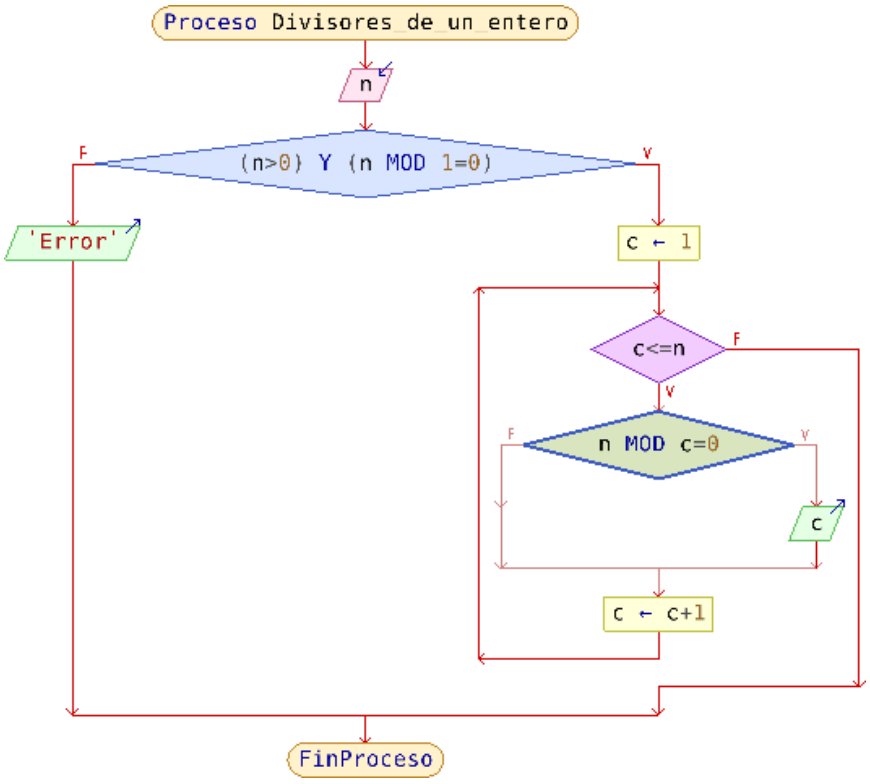


Figura 57: DF; Divisores de un entero

Comentario: Como se observa en la figura anterior (figura 57), el proceso comienza con el ingreso de un número, luego de lo cual se verifica que sea positivo ($n > 0$) y además que sea entero, para lo cual se utiliza el operador MOD que lo que hace, es devolver el resto de la división entre dos números. Así si n es entero, al dividirlo por 1, deberá tener resto 0.

Si las dos condiciones lógicas anteriores son verdaderas, el proceso continuará por la rama de la derecha (V), con lo cual se asigna a c (contador) el valor 1 y se entra a la condición de la repetición mientras.

Como se observa en la condición lógica del mientras, la repetición continuará mientras c sea menor o igual que n.

En la repetición se usa también el operador MOD pero ahora con el propósito de verificar si el resto de la división entre n y c es cero, ya que de ser así podemos asegurar que c es divisor de n. Dentro de la repetición se debe considerar el incremento del valor del contador c para que la repetición trabaje adecuadamente. Cuando la repetición mientras finalice, se finalizará también el proceso.

Si la verificación para determinar si el número es estero y positivo falla (F), se escribirá el mensaje 'Error' y el proceso terminará.

Representación en Pseudocódigo

```
1   Proceso Divisores_de_un_entero
2       Leer n;
3       Si (n>0) Y (n MOD 1=0) Entonces
4           c <- 1;
5           Mientras c<=n Hacer
6               Si n MOD c=0 Entonces
7                   Escribir c;
8               FinSi
9               c <- c+1;
10          FinMientras
11      SiNo
12          Escribir 'Error';
13      FinSi
14  FinProceso
```

Figura 58: Pseudocódigo; Divisores de un entero

Comentario: En la línea 4 se inicializa el contador. En la línea 5 se maneja la condición lógica para controlar la repetición, la cual terminará cuando el contador c deje de ser menor o igual que n.

En la línea 6 se tiene la condición lógica para determinar si c es divisor de n, lo cual se verifica si el resto de la división entre n y c es cero. Si la condición es verdadera se escribe el valor de c en la línea 7.

El incremento del contador c se ha puesto fuera de la condición lógica, ya que esta acción debe realizarse independientemente de si la condición es verdadera o falsa.

En la línea 12 está el escribir que deberá ejecutarse si la condición lógica que verifica si n es positivo y entero, es falsa.

Ejemplo 10:

Diseñe un algoritmo representándolo en DF y Pseudocódigo que determine y muestre suma de todos los divisores de un número entero positivo cualquiera.

Análisis problema

- Datos de entrada: Un número entero positivo

  Datos de salida: La suma de todos los divisores del número.
- Proceso: Leer un número, verificar si es entero y positivo y, determinar sus divisores y sumarlos, y mostrar la suma de todos sus divisores.
- Restricción: El algoritmo sólo debe tratar con números enteros y positivos

Diseño de la solución

- Leer un número, verificar que sea entero y positivo, de ser así, determinar sus divisores y sumarlos, luego de ello mostrar la suma de todos sus divisores.

Representación en DF

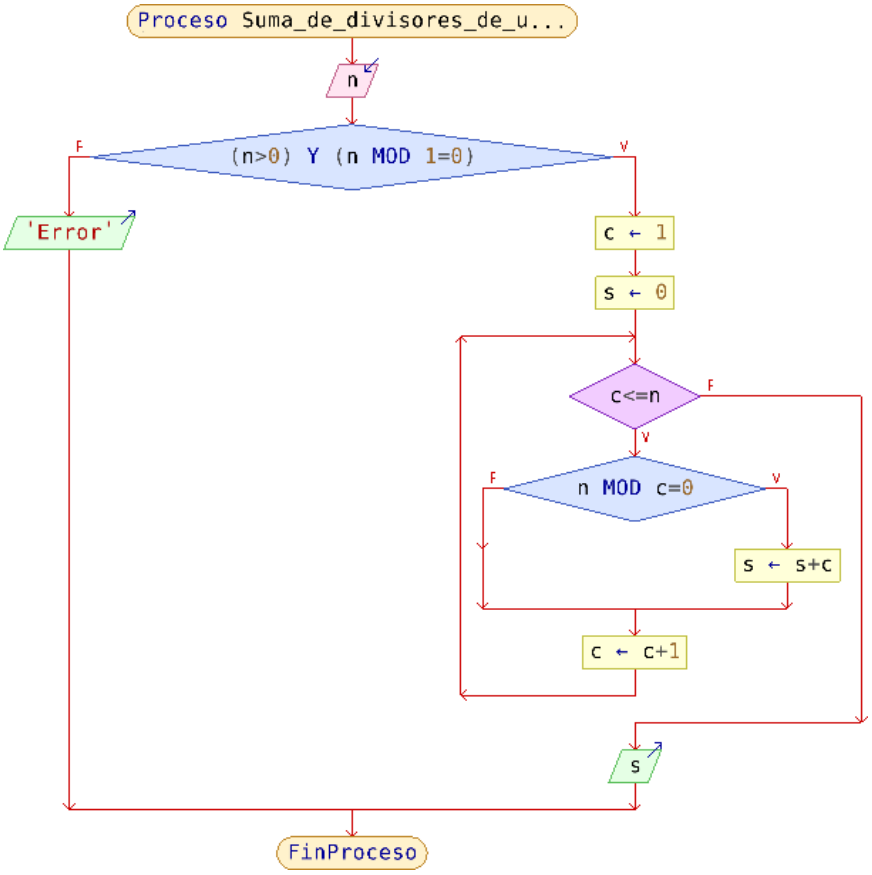


Figura 59: DF Suma de los divisores de un entero

Comentario: En esencia este problema es muy similar al anterior. La diferencia está que en lugar se mostrar los divisores del número, estos deben ir sumándose para mostrar la suma una vez finalizada la repetición.

Para sumar cada uno de los divisores del número, se ha usado un acumulador de suma que en este caso es s. Se denomina acumulador de suma ya que, como se observa en el DF, en esta variable se va sumando cada c que se verifica que es divisor de n.

Cuando se usan acumuladores estos deben inicializarse para asegurar que acumulan sólo los valores que interesan, y esta inicialización debe estar fuera de la repetición, lo que en este caso que se trata de un acumulador de suma, la variable debe inicializarse a cero.

Representación en Pseudocódigo

```
1   Proceso Suma_de_divisores_de_un_entero
2       Leer n;
3       Si (n>0) Y (n MOD 1=0) Entonces
4           c <- 1;
5           s<-0;
6           Mientras c<=n Hacer
7               Si n MOD c=0 Entonces
8                   s<-s+c;
9               FinSi
10              c <- c+1;
11          FinMientras
12          Escribir s;
13      SiNo
14          Escribir 'Error';
15      FinSi
16  FinProceso
```

Figura 60: Pseudocódigo; Suma de los divisores de un entero

5.3.2.- Repetir.

Este tipo de estructura algorítmica es muy similar a la estructura repetitiva **Mientras**, siendo su principal diferencia (como se mostrará en los siguientes ejemplos), el hecho de que la **condición lógica** que determina si se continúa con la repetición, se encuentra al final de la propia estructura repetitiva.

Utilizando los fragmentos de DF que se muestran en la siguiente figura explicamos esta diferencia.

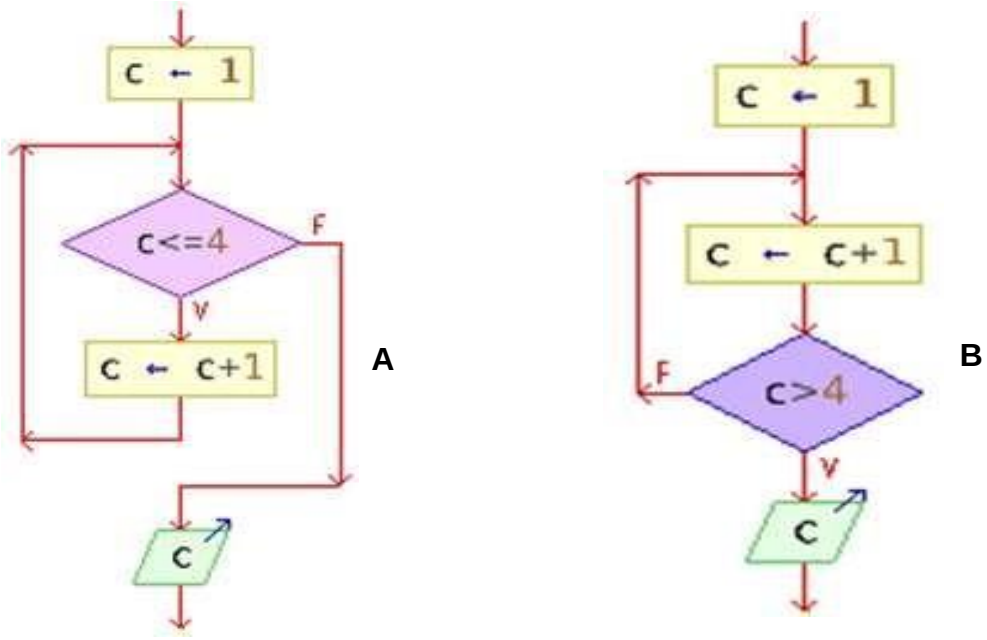


Figura 61: DF (1); Diferencia Mientras (A), Repetir (B)

Antes que nada, partamos por reconocer que tanto el fragmento **A** como el **B**, al ser ejecutados, escribirán el mismo valor de ***c*** una vez que terminen sus repeticiones, y el cual corresponde a **5**

Como se observa en el fragmento **B** la **condición lógica** para continuar o terminar las repeticiones se ubica al final de la propia repetición, con lo cual se requiere escribirla de manera diferente que para el caso **A**.

```
c <- 1;
Mientras c<=4 Hacer
    c <- c+1;
FinMientras
Escribir c;
```

A

```
c <- 1;
Repetir
    c <- c+1;
Hasta Que c>4
Escribir c;
```

B

Figura 62: Pseudocódigo (1); Diferencia Mientras (A), Repetir (B)

En la Figura 62 se presentan las equivalencias en Pseudocódigo de los fragmentos **A** y **B** mostrados en la Figura 53

Como se observa, para el caso de la estructura Repetir se utilizan la secuencia de palabras claves **Repetir** y **Hasta Que** y en esta última se incorpora la **condición lógica**

Si bien en muchos algoritmos se pueden usar indistintamente las estructuras repetitivas **Mientras** o **Repetir**, es importante tener en claro que el hecho de que la **condición lógica** para controlar la repetición se encuentra al final de una estructura **Repetir**, esto puede ocasionar repeticiones no deseadas para algunos casos.

Vemos el siguiente ejemplo para ilustrar lo que estamos señalando.

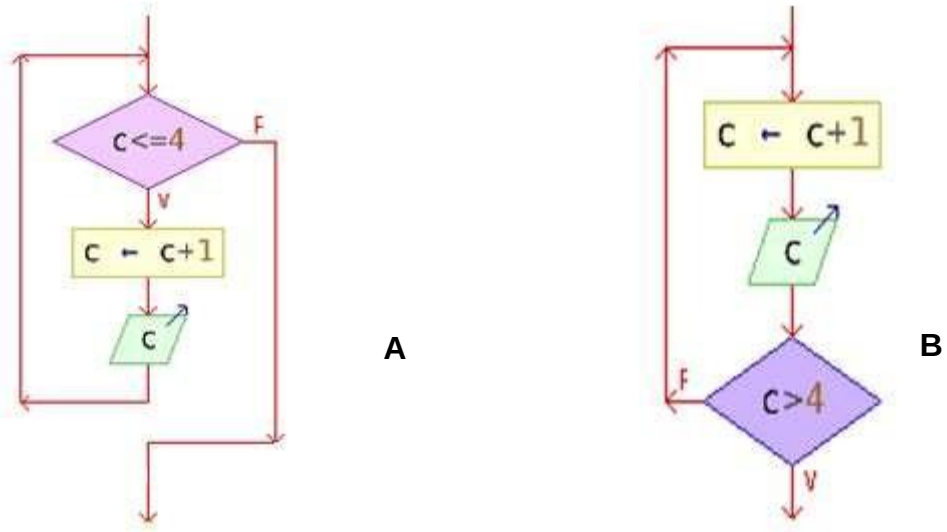


Figura 63: DF; Repeticiones Mientras (A) y Repetir (B) no equivalentes

Teniendo en cuenta los DF de la Figura anterior (Figura 63) y suponiendo que ***c*** antes de la repetición **Mientras** en (**A**) tienen cualquier valor superior a **4** dicha repetición nunca se ejecutará, sin embargo, si es el mismo caso para el **Repetir** en (**B**), ésta se ejecutará una vez.

En la siguiente figura se muestran los **Pseudocódigos** correspondientes a los DF anteriores.

```
Mientras c<=4 Hacer
    c <- c+1;
    Escribir c;
FinMientras
```

A

```
Repetir
    c <- c+1;
    Escribir c;
Hasta Que c>4
```

B

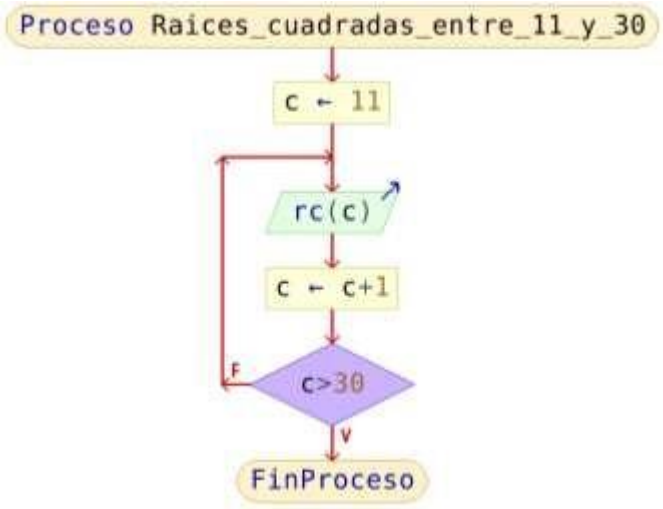
Figura 64: Pseudocódigo; Repeticiones Mientras (A) y Repetir (B) no equivalentes

Para ejemplificar la estructura repetitiva **Repetir** usaremos los mismos ejemplos que mostrados para la estructura **Mientras**.

Ejemplo 11:

Diseñe un algoritmo representándolo en DF y Pseudocódigo que muestre las raíces cuadradas de todos los números entre 11 y 30.

Representación Algorítmica



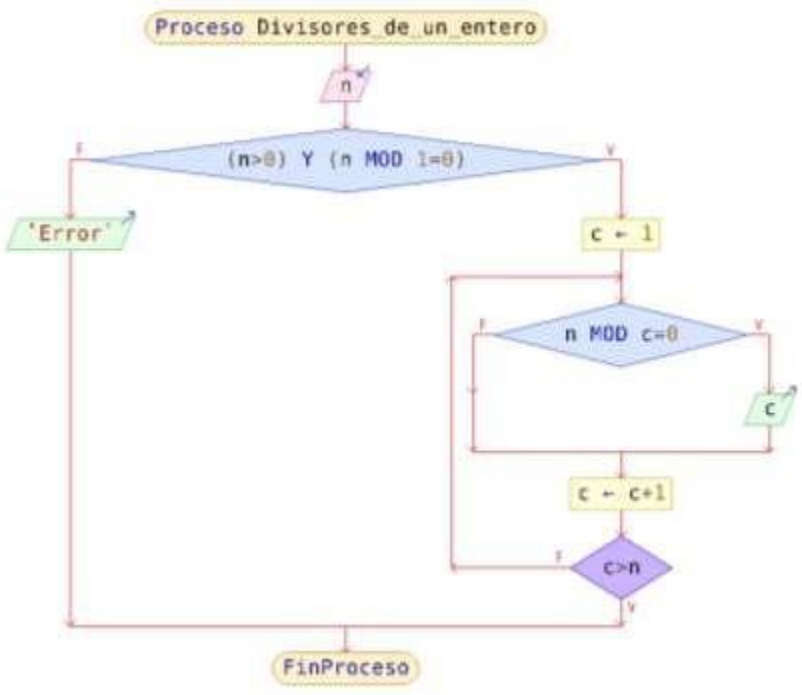

```
1  Proceso Raices_cuadradas_entre_11_y_30
2      c<-11;
3      Repetir
4          Escribir rc(c);
5          c<-c+1;
6      Hasta Que c>30
7  FinProceso
```

Figura 65: DF y Pseudocódigo; Raíces cuadradas entre 11 y 30 usando Repetir

Ejemplo 12:

Diseñe un algoritmo representándolo en DF y Pseudocódigo que muestre todos los divisores de un número entero positivo cualquiera.

Representación Algorítmica



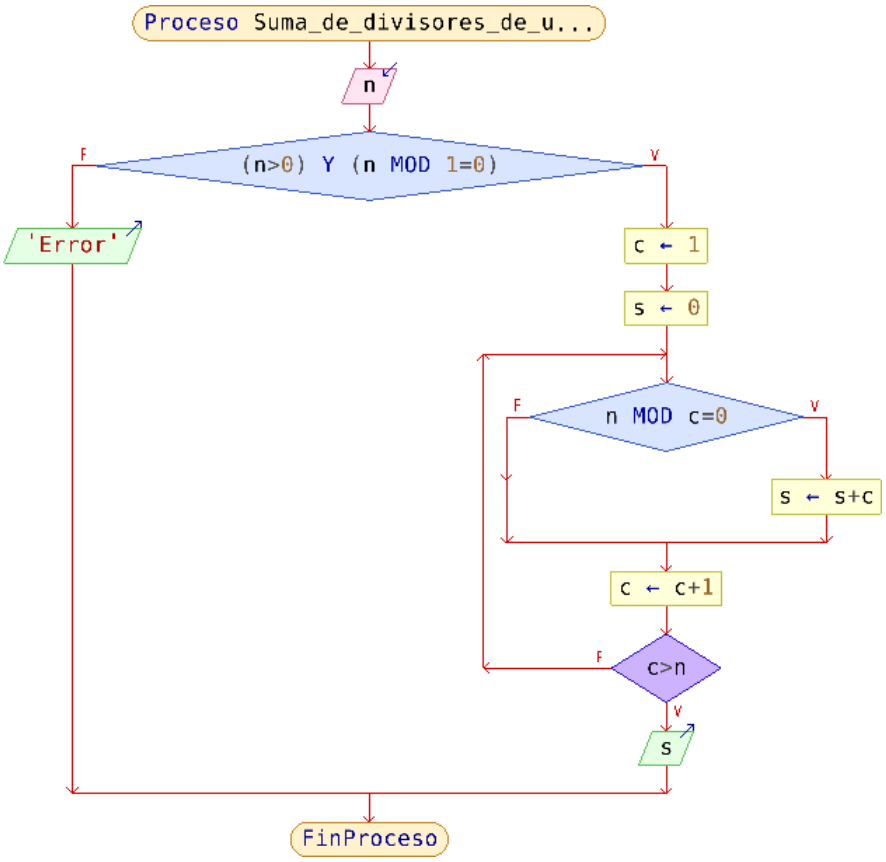

```
1   Proceso Divisores_de_un_entero
2       Leer n;
3       Si (n>0) Y (n MOD 1=0) Entonces
4           c <- 1;
5           Repetir
6               Si n MOD c=0 Entonces
7                   Escribir c;
8               FinSi
9               c <- c+1;
10          Hasta Que c>n
11      SiNo
12          Escribir 'Error';
13      FinSi
14  FinProceso
```

Figura 66: DF y Pseudocódigo; Divisores de un entero usando Repetir

Ejemplo 13:

Diseñe un algoritmo representándolo en DF y Pseudocódigo que determine y muestre suma de todos los divisores de un número entero positivo cualquiera.

Representación Algorítmica




```
1   Proceso Suma_de_divisores_de_un_entero
2       Leer n;
3       Si (n>0) Y (n MOD 1=0) Entonces
4           c <- 1;
5           s<-0;
6           Repetir
7               Si n MOD c=0 Entonces
8                   s<-s+c;
9               FinSi
10              c <- c+1;
11          Hasta Que c>n
12          Escribir s;
13      SiNo
14          Escribir 'Error';
15      FinSi
16  FinProceso
```

Figura 67: DF y Pseudocódigo; Suma de los divisores de un entero usando Repetir

5.3.3.- Para.

Este tipo de estructura algorítmica es la tercera forma de estructura repetitiva, siendo su principal característica, ser una estructura más compacta en lo referido al control de las repeticiones, respecto de las otras dos.

Para explicar esto partiremos mostrado primero su Pseudocódigo:

```
Para variable_numerica<-valor_inicial Hasta valor_final Con Paso paso Hacer
    secuencia_de_acciones
Fin Para
```

Figura 68: Pseudocódigo; Elementos de la estructura de repetición Para

De donde:

`variable_numerica` Nombre de una variable numérica que se utilizará para controlar el número de repeticiones.

`valor_inicial` Valor inicial que tomará la variable numérica al comenzar las repeticiones.

`valor_final` Valor final que tendrá la variable numérica para finalizar las repeticiones.

`paso` Valor del incremento o decremento que tendrá la variable numérica en cada repetición.

En DF esta estructura tiene la siguiente forma y elementos.

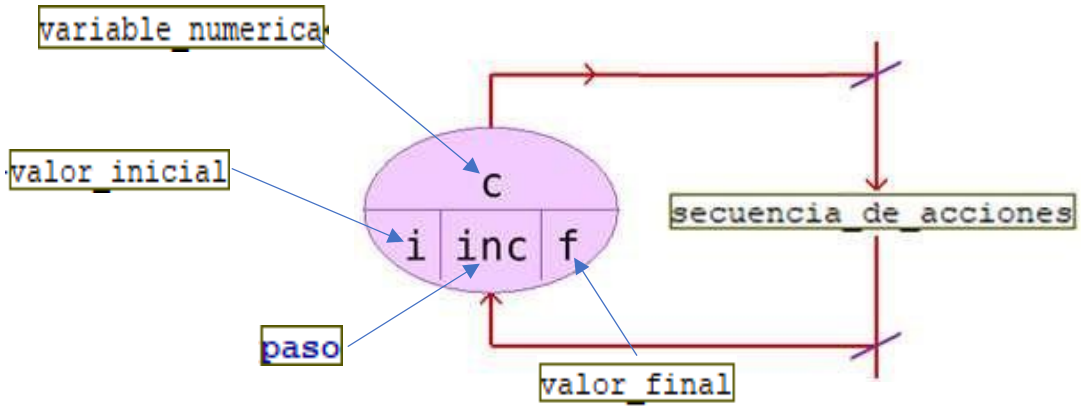


Figura 69: DF; Forma y elementos de la Repetición Para

El siguiente es un ejemplo sencillo donde se ilustra el funcionamiento de la estructura **Para** y la relación entre los elementos del **DF** y el **Pseudocódigo**.

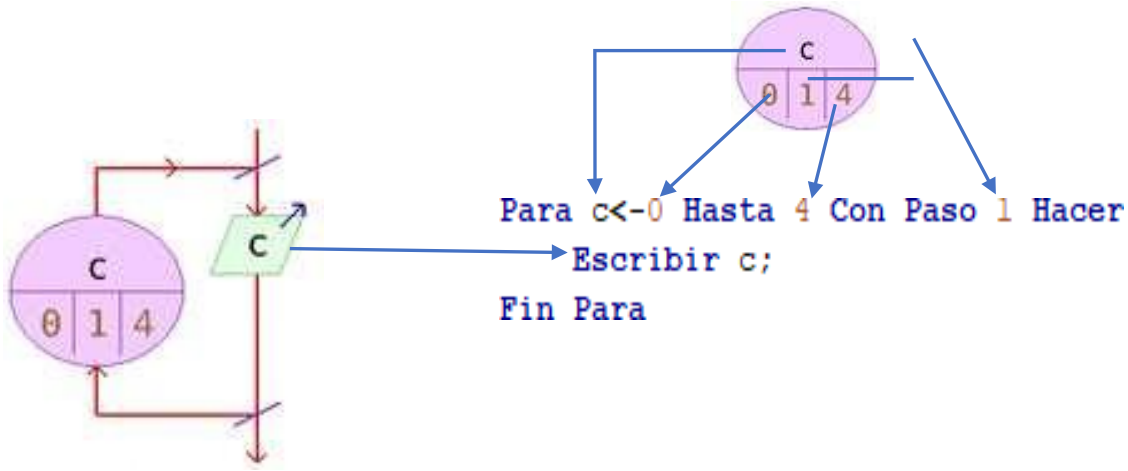


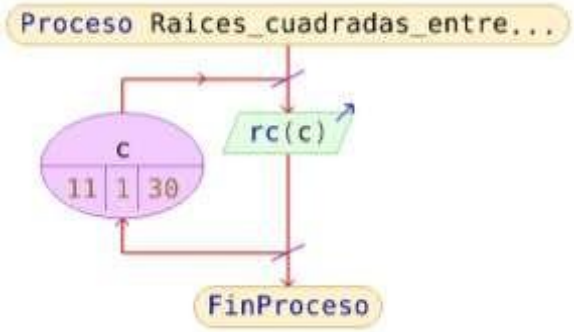
Figura 70: Funcionamiento y Elementos de la estructura repetitiva Para

Para ejemplificar la estructura repetitiva **Para** al igual que en los casos anteriores, usaremos los mismos ejemplos que se han mostrado hasta ahora.

Ejemplo 14:

Diseñe un algoritmo representándolo en DF y Pseudocódigo que muestre las raíces cuadradas de todos los números entre 11 y 30.

Representación Algorítmica




```
1  Proceso Raices_cuadradas_entre_11_y_30
2      Para c<-11 Hasta 30 Con Paso 1 Hacer
3          Escribir rc(c);
4      Fin Para
5  FinProceso
```

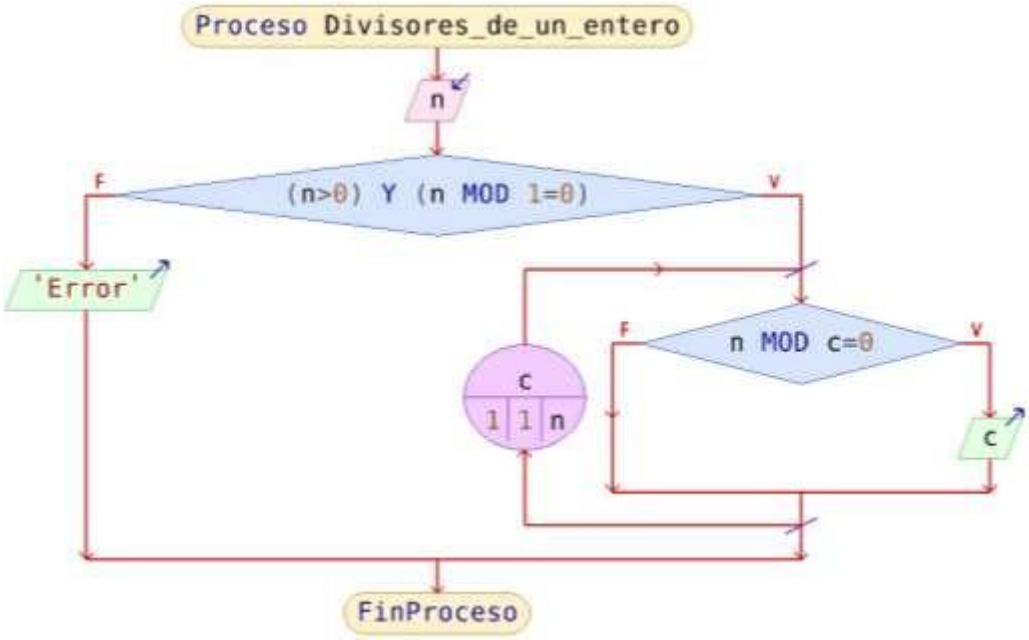
Figura 71: DF y Pseudocódigo; Raíces cuadradas entre 11 y 30 usando Para

Comentario: Como se observa en la figura anterior (Figura 71), la repetición Para posee una estructura más compacta que Mientras y Repetir, ya que por una parte no requiere de la inicialización de la variable que controla el número de repeticiones antes de la propia estructura repetitiva, y además no necesita una condición lógica para establecer en cuando se terminan las repeticiones. Sin embargo, es importante aclarar que, aunque la condición lógica no está de manera explícita, si lo está de forma implícita en la palabra clave Hasta contenida en la cabecera de la estructura.

Ejemplo 15:

Diseñe un algoritmo representándolo en DF y Pseudocódigo que muestre todos los divisores de un número entero positivo cualquiera.

Representación Algorítmica



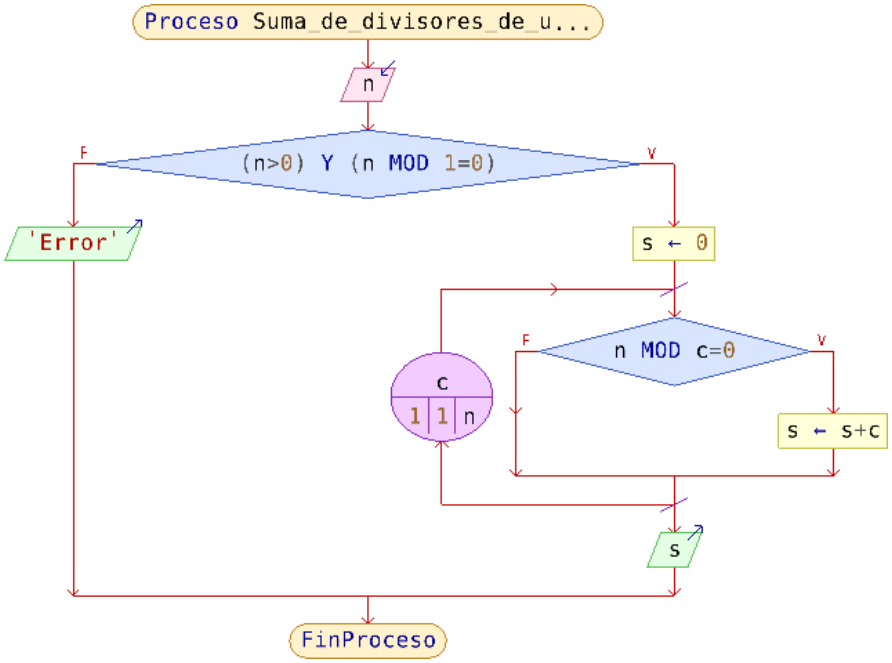

```
1   Proceso Divisores_de_un_entero
2       Leer n;
3       Si (n>0) Y (n MOD 1=0) Entonces
4           Para c<-1 Hasta n Con Paso 1 Hacer
5               Si n MOD c=0 Entonces
6                   Escribir c;
7               FinSi
8           Fin Para
9       SiNo
10          Escribir 'Error';
11      FinSi
12  FinProceso
```

Figura 72: DF y Pseudocódigo; Divisores de un entero usando Para

Ejemplo 16:

Diseñe un algoritmo representándolo en DF y Pseudocódigo que determine y muestre suma de todos los divisores de un número entero positivo cualquiera.

Representación Algorítmica




```
1   Proceso Suma_de_divisores_de_un_entero
2       Leer n;
3       Si (n>0) Y (n MOD 1=0) Entonces
4           s <- 0;
5           Para c<-1 Hasta n Con Paso 1 Hacer
6               Si n MOD c=0 Entonces
7                   s <- s+c;
8               FinSi
9           Fin Para
10          Escribir s;
11      SiNo
12          Escribir 'Error';
13      FinSi
14  FinProceso
```

Figura 73: DF y Pseudocódigo; Suma de los divisores de un entero usando Para

.