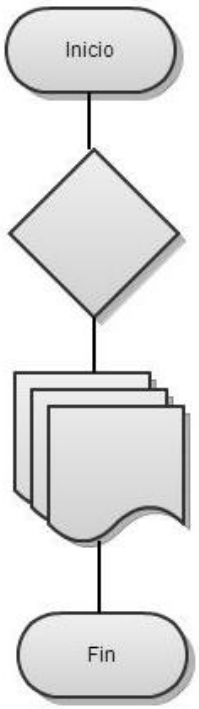
Universidad Tecnológica Nacional
Tecnicatura en Programacion

EJERCICIOS RESUELTOS:

DIAGRAMAS DE FLUJO



DOCENTE: ING. ALEXIS MASSÓN

amasson@tecnicatura.frc.utn.edu.ar

INTRODUCCIÓN

El presente documento pretende ser una guía práctica para la comprensión y ejercitación de diagramas de flujo, conteniendo una serie de conceptos básicos necesarios, estructuras de control, simbología y ejercicios prácticos resueltos.

CONTENIDOS

DEFINICIONES TEÓRICAS BÁSICAS

DIAGRAMA DE FLUJO

Un diagrama de flujo es una representación gráfica de un algoritmo.

ALGORITMO

Es un conjunto de instrucciones definidas, ordenadas y finitas (es decir, siempre tiene un fin) que permite mediante pasos sucesivos realizar una determinada actividad.

UTILIDAD DE UN DIAGRAMA DE FLUJO

Un diagrama de flujo es útil para comunicar un determinado algoritmo.

En la práctica generalmente es un recurso empleado por un analista funcional en la especificación de una determinada actividad, programa o proceso que se considera lo suficientemente complejo como para no poder ser expresado claramente por escrito con vocabulario natural. Por tanto, un diagrama de flujo se considera una opción mucho más precisa que suple esta falencia asociada a la ambigüedad del lenguaje natural.

LENGUAJE NATURAL

El lenguaje natural es lenguaje hablado o escrito por humanos para propósitos generales de comunicación. Este lenguaje tiene el problema que en ciertas circunstancias el mensaje que se quiere comunicar puede ser interpretado de distintas maneras de acuerdo al contexto y al receptor del mismo, por lo tanto se dice que puede ser impreciso y ambiguo.

ESTRUCTURA DE CONTROL

Una estructura de control es un mecanismo que permite alterar el flujo de ejecución de un programa.

Existen tres tipo de estructuras de control: secuencial, alternativa o de selección y iterativas o repetitivas.

DIAGRAMAS DE FLUJO

INICIO Y FIN

Un diagrama de flujo siempre tiene un único punto de inicio y un único punto de fin

SIMBOLOGÍA

Es **absolutamente necesario**, para poder entender y resolver ejercicios de diagrama de flujo, el conocimiento de la simbología específica a utilizar. A continuación se detallan la simbología básica y más utilizada en la construcción de diagramas de flujo (hay símbolos que no están contemplados en esta guía):

Terminal. Demarca el inicio o el fin del diagrama de flujo.

Proceso. Utilizado para la declaración de variables o ejecución de operaciones matemáticas y de asignación.

Entrada. Ingreso de datos, se utiliza para representar las entradas del algoritmo o valores ingresados por un usuario que está utilizando el programa.

Decisión. Bloque que permite alterar el flujo del algoritmo eligiendo entre varios caminos o escenarios alternativos. Decisión simple, doble y múltiple.

Impresora. Indica la salida o impresión de un resultado, puede interpretarse como que se está mostrando un resultado del algoritmo al usuario que está empleado el programa.

ESTRUCTURAS DE CONTROL

ESTRUCTURA DE CONTROL SECUENCIAL

Es la más simple de las estructuras de control; se caracteriza porque una acción o sentencia se ejecuta detrás de la otra. Por ello, el flujo del algoritmo coincide con el orden físico en el que se han puesto las sentencias del algoritmo.

ESTRUCTURA DE CONTROL DE SELECCIÓN O ALTERNATIVA

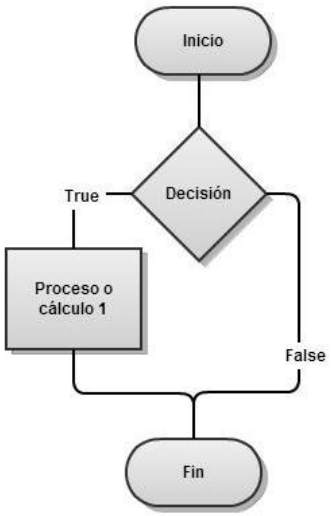
Como su mismo nombre lo indica, este tipo de estructuras realizan una selección de las acciones o sentencias a ejecutar, es decir, dependiendo de si se cumple o no una determinada condición, se ejecutan o no, un determinado grupo de sentencias.

Es importante mencionar que la condición que rige el control del flujo puede ser tan elaborada como la naturaleza del problema lo requiera, pero se deberá tener la garantía de que el valor final de la condición podrá ser evaluado como un valor booleano, esto es, como verdadero o falso, ya que de lo contrario la condición tendrá ambigüedad y esto viola una de las características más importantes de un algoritmo.

Las estructuras de selección pueden ser simples, dobles o múltiples.

ESTRUCTURA DE SELECCIÓN SIMPLE

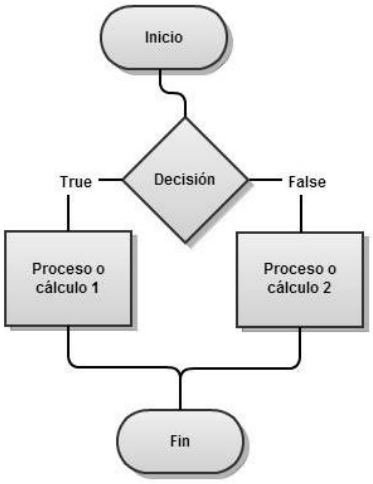
En este tipo de estructura de control la decisión es ejecutar o no un cierto bloque de proceso o cálculo. Si el resultado de la condición es TRUE, entonces el flujo del programa continua para realizar el proceso o cálculo, si es FALSE elije el camino alternativo que saltea esta operación, hasta finalmente volver al curso normal, en este caso, llegando al fin del algoritmo.



En C# equivale a la estructura de control "if".

ESTRUCTURA DE SELECCIÓN DOBLE

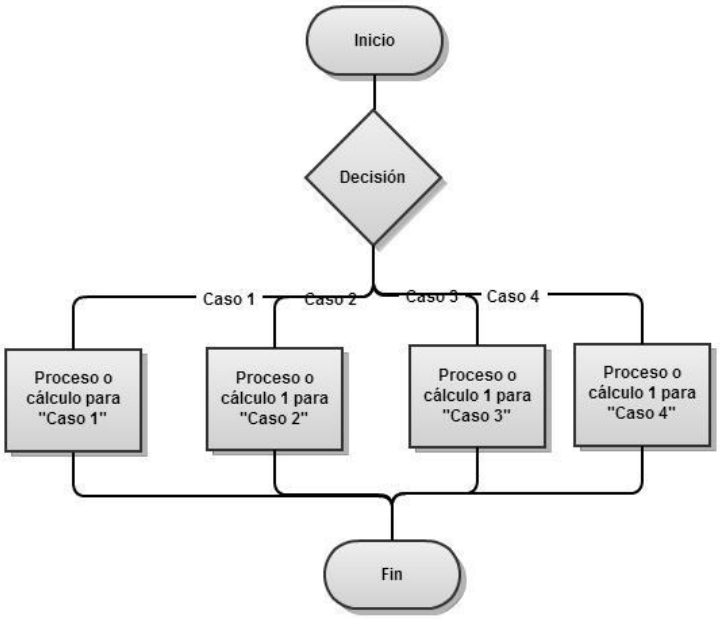
La estructura de selección doble tiene la capacidad de elegir entre dos alternativas de procesamiento o cálculo. De acuerdo al valor resultante de la evaluación de la decisión TRUE/FALSE elije un camino u otro.



En C# equivale a la estructura de control "if-else".

ESTRUCTURA DE SELECCIÓN MÚLTIPLE

La estructura de control de selección múltiple nos permite manipular el flujo del algoritmo para que siga por uno de varios caminos posibles de acuerdo al valor de una determinada variable (un valor puntual, no una expresión, notar la diferencia con los anteriores, donde se evaluaba una expresión booleana).



En C# equivale con la estructura de control "switch-case"

ESTRUCTURA DE CONTROL ITERATIVA O REPETITIVA

Las estructuras de control iterativas o repetitivas se caracterizan por ser bucles o ciclos que se ejecutan una determinada cantidad de veces, realizando alguna acción en cada iteración o repetición del bucle/ciclo. Es común el uso de variables especiales: contadores, acumuladores y banderas.

CICLO/BUCLE

Un ciclo es cada vez que se repite una operación o conjunto de operaciones definida dentro de la estructura iterativa en cuestión. Cada estructura iterativa por lo general pretende tener 1 o más ciclos o bucles.

CONTADOR

Un contador es una variable especial de tipo número entero ("int" en C#) que se incrementa en una unidad cada ciclo o bucle. Sirve para contar las iteraciones o bucles que han transcurrido y realizar alguna determinada acción en base a esa cantidad.

ACUMULADOR

Un acumulador es una variable especial numérica ("int" si es entera, "float" o "double" si es decimal) que tiene como objetivo sumar y acumular la suma acumulada, la cual se incrementa o decrementa en cada ciclo del iterador.

BANDERA

Una bandera es una variable especial de tipo boolean (true/false) que permite modificar las acciones a realizar dentro del iterador. Por ejemplo, puede utilizarse para realizar una acción concreta dentro del iterador una única vez (generalmente alguna tarea de inicialización); de tal forma que las iteraciones sucesivas que ocurran omitan esta acción, consultando el valor de la bandera.

TIPOS DE ESTRUCTURAS DE CONTROL

Las estructuras de control se pueden clasificar en:

- **Controladas por contador**. Se emplea una variable contadora para repetir un número fijo de iteraciones. Contempla a la estructura "for".
- **Controlada por condición.** Se emplea una condición de corte, es decir, la continuidad o no del bucle depende del resultado de la evaluación de una expresión booleana, que decide si se continúa o se sale del iterador en cuestión. Contempla a la estructura "while" y "do-while".

CUÁNDO USAR ESTRUCTURAS ITERATIVAS O REPETITIVAS

- Recorrer un arreglo o vector o procesar una cadena de caracteres
- Realizar operaciones matemáticas como promedio, integración numérica, etc.
- Ordenar y buscar datos
- Cualquier situación que requiere ejecutar una acción más de una vez

ESTRUCTURA "FOR"

La estructura "For" se caracteriza por ser controlada por contador, es decir que se ejecuta un número fijo de veces.

Tiene cuatro componentes esenciales:

- Valor de inicialización. Define el valor de inicio del contador que determinará la cantidad de ciclos que se van a ejecutar.
- Condición booleana de corte o paro. Expresión booleana que será evaluada al principio de cada ciclo para determinar si se debe continuar o si se escapa de la estructura iterativa.
- Incremento. Define cuántas unidades se suman al contador tras cada iteración, generalmente se trata de un incremento unitario, expresado como "i = i + 1" o en forma abreviada como "i++" (que significa que la variable "i" se incrementa en una unidad).
- Expresión a repetir. Es el corazón de la estructura, define cuál es la operación u operaciones a repetir en cada ciclo del iterador.

En C# se la sintaxis es la siguiente:

```
for(inicialización; condición; incremento)
{
        expresiones a repetir;
}
```

Ejemplo concreto: Mostrar por consola números de 0 a 9.

```
for(int i = 0; i< 10; i++){
        Console.WriteLine(i);
}
```

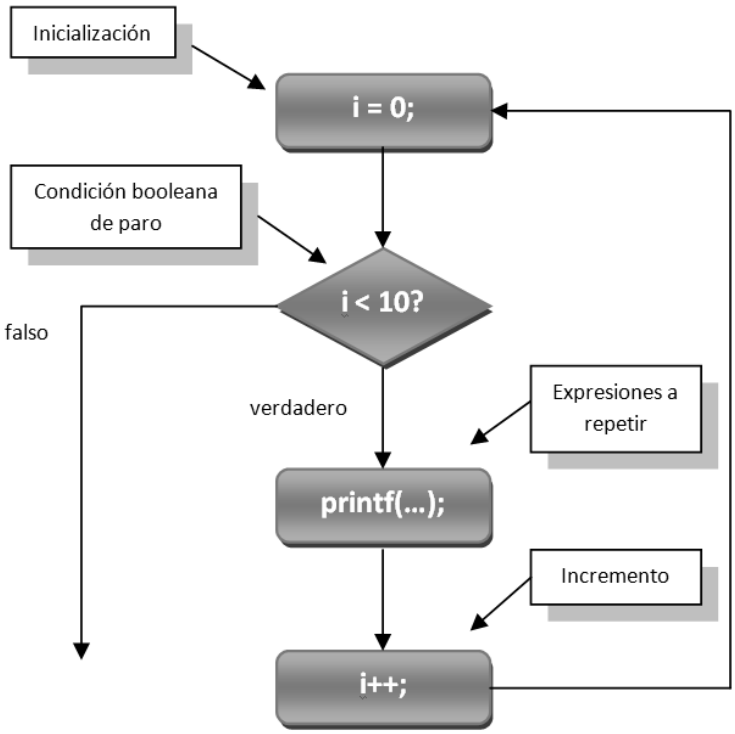


Figura: Diagrama de flujo del ciclo *for*

ESTRUCTURA "WHILE"

La estructura "While" es una estructura de control iterativa controlada por condición, es decir, que finaliza cuando una determinada expresión a evaluar devuelve FALSE.

Para definir una estructura "While" es necesario contar con:

- Condición booleana de corte o paro. Expresión que es evaluada y en caso de resultar TRUE continúa la ejecución del próximo ciclo del iterador, y en caso de ser FALSE finaliza y escapa del iterador.
- Expresión a repetir. Es el corazón de la estructura, define cuál es la operación u operaciones a repetir en cada ciclo del iterador.

En C# la sintaxis es la siguiente:

```
while(condicion){
        expresión a repetir;
}
```

Ejemplo concreto: Mostrar por consola números de 0 a 9.

```
int contador = 0;
while(contador < 10){
        Console.WriteLine(contador);
        contador = contador +1;
}
```

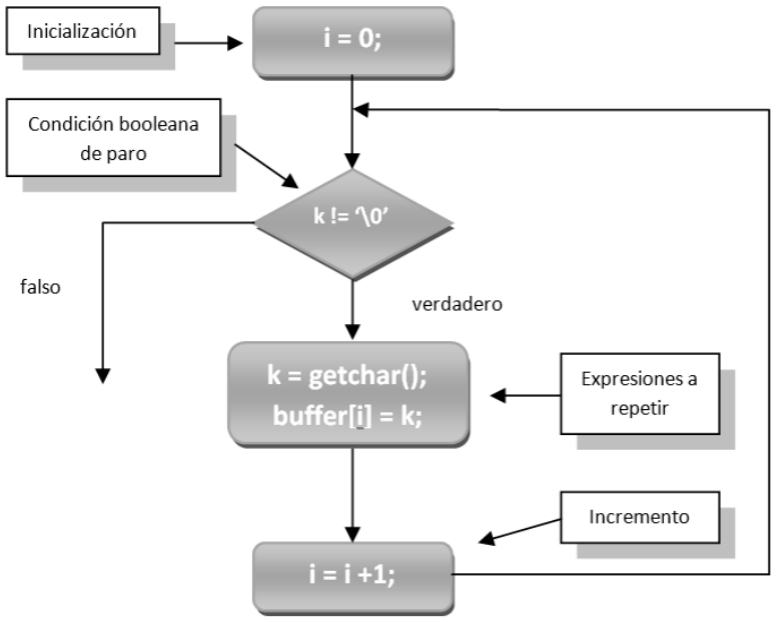


Figura: Diagrama de flujo del ciclo *while*

ESTRUCTURA "DO-WHILE"

La estructura "Do-While" también es una estructura de control iterativa controlada por condición, sin embargo se diferencia de la estructura While por el hecho que siempre se garantiza que la acción a repetir se ejecute al menos una vez. En la estructura While, de acuerdo a la condición de corte podría ocurrir que la acción a repetir no se ejecute ni una sola vez.

Al igual que la estructura While, sus componentes principales son la condición booleana de corte o paro y la expresión a repetir.

En C# la sintaxis es:

```
do {
        expresión a repetir;
} while (condición)
```

Ejemplo concreto: Mostrar por consola números de 0 a 9.

```
int contador = 0;
do {
        Console.WriteLine(contador);
        contador = contador +1;
} while(contador < 10)
```

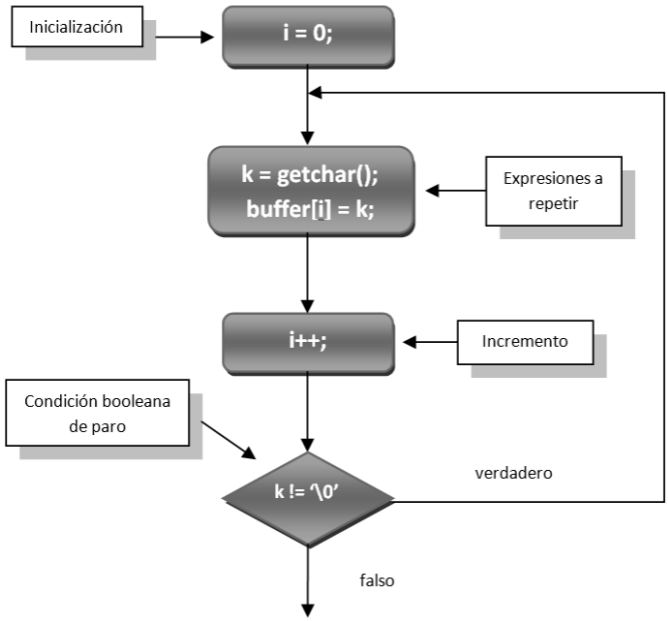


Figura: Diagrama de flujo del ciclo *do-while*

RESOLUCIÓN DE EJERCICIOS DE DIAGRAMA DE FLUJO

Primera etapa. Análisis.

Al momento de resolver un diagrama de flujo es muy importante primero que nada entender cuál es el **objetivo del algoritmo**. En base a ese objetivo, es fácilmente distinguible cuál será la salida esperada, y dada esa salida determinar qué entradas me permitirán calcular dicha salida, cumplimentando el objetivo del algoritmo.

Por tanto, decimos que la primera etapa para resolver un ejercicio de diagrama de flujo es:

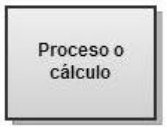
1. Entender el objetivo del algoritmo.
2. Determinar la salida esperada.
3. Determinar las entradas necesarias para lograr la salida esperada.

Segunda etapa. Construcción.

Todo diagrama de flujo siempre tiene un punto de inicio y un punto de fin. Entonces el primer paso en la construcción es diagrama el punto de inicio.

Por lo general, inmediatamente a continuación del punto de inicio se deja previsto un bloque para declaración de variables. Este bloque es muy útil al momento que se quiere programar el algoritmo en un lenguaje de programación real.

Las entradas esperadas son las que determinan los datos a ingresar, por lo tanto, generalmente también se incluye un bloque de ingreso de datos.



A continuación es común el uso de estructuras de control secuenciales, selectivas o iterativas de acuerdo al algoritmo en particular que se esté modelando.

Finalmente, todo diagrama de flujo finaliza con un bloque de fin.

Tercera etapa. Verificación. Prueba de escritorio.

Una vez construido el diagrama de flujo es necesario asegurarse del correcto funcionamiento del algoritmo, esto significa que dado el universo de posibles entradas siempre el algoritmo logre alcanzar la salida esperada. Para ello se deben elegir conjuntos de entradas, ejecutar paso a paso el algoritmo y chequear que la salida real corresponda con la salida esperada.

Si la salida real difiere de la esperada se deberá analizar la causa de la diferencia y tomar acciones correctivas para corregir el problema.

EJERCICIOS RESUELTOS

METODOLOGÍA DE TRABAJO

Para explicar los ejercicios de diagrama de flujo que se presentarán a continuación se presentará un enunciado expresado en lenguaje natural (tal cual se le facilitará al alumno en el momento de evaluar esta unidad). A continuación, todo lo que procede en dicho ejercicio es parte de la estrategia de resolución descripta en el apartado anterior.

EJERCICIOS RESUELTOS DE ESTRUCTURAS SECUENCIALES

EJERCICIO 1

Enunciado: Realizar el diagrama de flujo un algoritmo que permita sumar dos números.

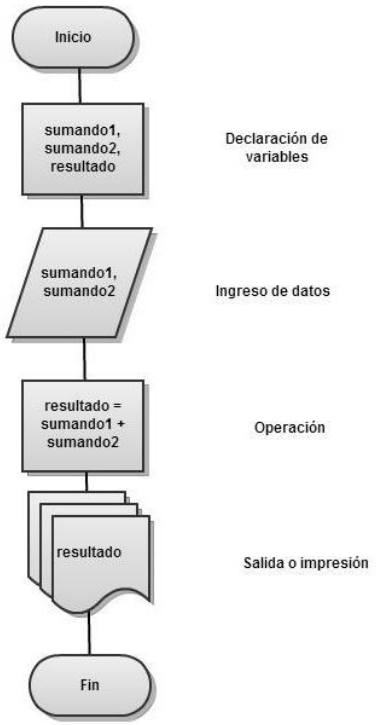
Resolución

Objetivo del algoritmo: Sumar dos números

Entradas: Números a sumar (sumando 1 y sumando 2)

Salida esperada: Valor resultante de aplicación la operación suma a las entradas del algoritmo

Diagrama de flujo:



Explicación de la construcción del diagrama

1. Declaración del bloque de inicio
2. Dejar previsto el bloque de declaración de variables (generalmente lo dejamos en blanco hasta terminar el diagrama de flujo, ya que ahí vamos a declarar todas las variables que necesitemos en el algoritmo, y puede ser que al principio no tengamos totalmente claro qué variables vamos a utilizar, algunas van a ir apareciendo de acuerdo a las necesidades puntuales de cálculo u operación que demande el algoritmo en cuestión).
3. Definir el ingreso de datos. Esto es muy simple si ya se ha analizado el problema y se conocen las entradas necesarias.
4. Realizar operaciones necesarias. En este paso se pueden emplear estructuras de control de selección o de iteración, sin embargo este caso es trivial, así que solo emplea una estructura de control secuencial.
5. Mostrar el resultado de la operación. En este caso, el total de la suma efectuada.
6. Completar el bloque de declaración de variables previsto en el punto 2
7. Finalizar el algoritmo incluyendo un nodo de "Fin".

Prueba de escritorio:

Para realizar la prueba de escritorio es bueno construir una tabla que contenga tantas columnas como variables se hayan definido en el bloque de declaración de variables del el algoritmo (en este caso: sumando 1, sumando2 y resultado), seguidas de la salida esperada y una columna que indique si el resultado fue satisfactorio (Ok) o erróneo (Error).

Sumando 1	**Sumando 2**	**Resultado**	**Salida esperada**	**Verificado**
3	5	8	8	Ok
9	17	26	26	Ok
55	12	67	67	Ok

Para cada entrada del algoritmo se debe elegir un valor en forma aleatoria o bajo algún criterio de elección, y a partir de dichas entradas proceder a interpretar el algoritmo desde el bloque de inicio hasta el bloque de fin, completando durante este proceso las columnas correspondientes a cada variables del problema.

En pocas palabras se trataría de una "ejecución del algoritmo en papel".

En caso de encontrarse diferencias entre la salida real del algoritmo, producto de su ejecución, y la salida esperada que debería haber surgido, se debe re-analizar la solución propuesta en el diagrama de flujo, detectarse la falencia y proceder a corregir el problema. Tras esto, se volverá a ejecutar la prueba de escritorio hasta que el resultado sea satisfactorio para todos los escenarios o combinación de entradas planteadas.

EJERCICIO 2

Enunciado: Dado el valor de la longitud de la hipotenusa y el cateto menor de un triángulo rectángulo, calcular su área

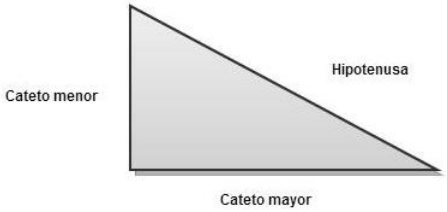
Resolución

Objetivo del algoritmo: Calcular el área de un triángulo rectángulo

Entradas: Hipotenusa, cateto menor

Salida esperada: Área del triángulo

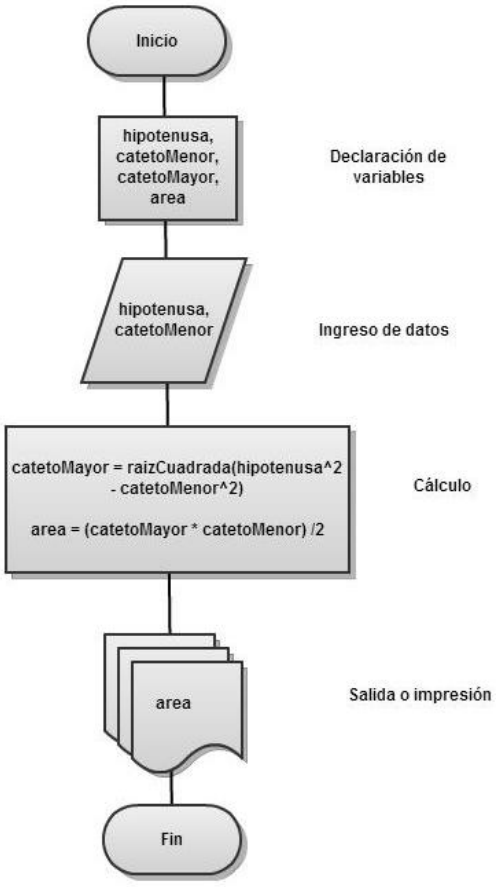
Esquema conceptual necesario:



En ocasiones es bueno valerse de algún diagrama, anotación o gráfico que nos ayude a razonar y hacer inferencias para resolver el problema. En este caso es útil ayudarse con un gráfico del triángulo rectángulo y la relación de Pitágoras para triángulos rectángulos.

$Hipotenusa^2 = Cateto\ mayor^2 + Cateto\ menor^2$

Diagrama de flujo:



Explicación de la construcción del diagrama:

1. Declaración del bloque de inicio
2. Definir bloque de declaración de variables , completar después.
3. Ingresar datos de entrada necesarios, en este caso la hipotenusa y el valor del catetoMenor del triángulo rectángulo
4. Para calcular el área de un triángulo con la fórmula "(base * altura)/2" vemos que conocemos la altura (el cateto menor), pero no la base (cateto mayor). Sin embargo, ayudándonos de la hipotenusa podemos calcular el valor de la base (cateto mayor) mediante la relación de Pitágoras.
   De ella deducimos que: $Cateto\ mayor^2 = Hipotenusa^2 - Cateto\ menor^2$
   Por lo tanto:
   $Cateto\ mayor = Raíz\ cuadrada\ de(Hipotenusa^2 - Cateto\ menor^2)$
   Luego, aplicando la fórmula para el cálculo del área llegamos al resultado:
   Área = (Base * altura)/2 = (Cateto mayor * Cateto menor) / 2
5. Mostrar el resultado de la operación. En este caso, el área del triángulo.
6. Completar el bloque de declaración de variables previsto en el punto 2.
7. Finalizar el algoritmo incluyendo un nodo de "Fin".

Prueba de escritorio:

Dado las variables hipotenusa , cateto menor, cateto mayor y área, se define la siguiente tabla para realizar una prueba de escritorio.

Hipotenusa	Cateto menor	Cateto mayor	Área	Salida esperada	Verificado
21,54	8	20	80	80	Ok
31,06	17	26	221	221	Ok
68,06	12	67	402	402	Ok

EJERCICIOS RESUELTOS DE ESTRUCTURAS SELECTIVAS O ALTERNATIVAS

EJERCICIO 3

Enunciado: Dados dos números distintos (no pueden ser iguales), determinar cuál de ellos es el mayor

Resolución

Objetivo del algoritmo: Determinar cuál es el número mayor

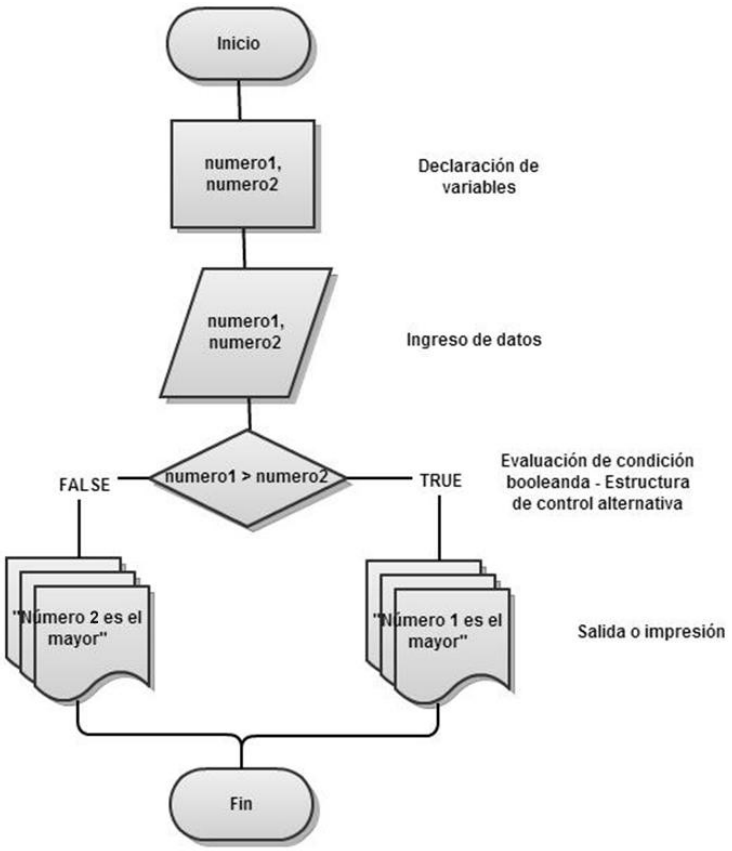
Entradas: Número 1, número 2

Salida esperada: Número mayor

Diagrama de flujo:

Explicación de la construcción del diagrama

1. Definir bloque de inicio
2. Prever bloque de declaración de variables
3. Ingresar valores de entrada, número 1 y número 2
4. Definir estructura de control alternativa, en este caso una expresión booleana para determinar si el número 1 es mayor al número 2.
    a. En caso de que la expresión sea TRUE, imprimir una cadena de caracteres con el siguiente mensaje "Número 1 es el mayor" (notar que al ser una cadena de caracteres va expresada entre comillas dobles)
    b. En caso de que la expresión sea FALSE, imprimir una cadena de caracteres con el siguiente mensaje "Número 2 es el mayor".



5. Completar el bloque de declaración de variables con todas las variables que hayan sido utilizadas
6. Declara bloque de fin.

Prueba de escritorio:

Dado las variables número 1 y número 2 se define la siguiente tabla para realizar una prueba de escritorio.

Número 1	**Número 2**	**Salida real**	**Salida esperada**	**Verificado**
8	20	"El número 2 es el mayor"	"El número 2 es el mayor"	Ok
15	6	"El número 1 es el mayor"	"El número 1 es el mayor"	Ok

EJERCICIO 4

Enunciado: Dados dos números (PUEDEN ser iguales), determinar cuál de ellos es el mayor

Resolución

Objetivo del algoritmo: Determinar cuál es el número mayor o en su defecto determinar si son iguales

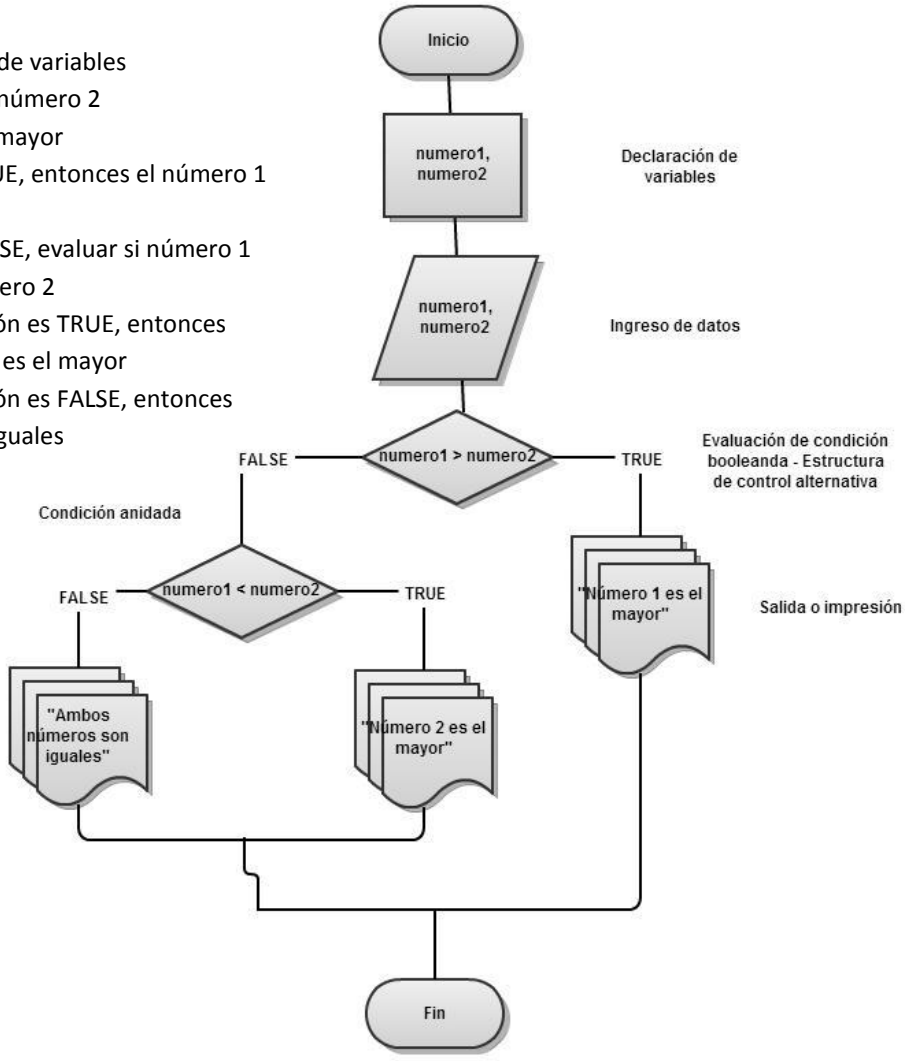
Entradas: Número 1, número 2

Salida esperada: Número mayor o igualdad

Diagrama de flujo:

Explicación de la construcción del diagrama

1. Bloque de inicio
2. Prever bloque de declaración de variables
3. Ingreso de datos, número 1 y número 2
4. Determinar si número 1 es el mayor
    a. Si la expresión es TRUE, entonces el número 1 es el mayor
    b. Si la expresión es FALSE, evaluar si número 1 es menor que el número 2
        i. Si la expresión es TRUE, entonces el número 2 es el mayor
        ii. Si la expresión es FALSE, entonces ambos son iguales
5. Completar bloque de declaración de variables
6. Bloque de fin



Prueba de escritorio:

Dado las variables número 1 y número 2 se define la siguiente tabla para realizar una prueba de escritorio.

Número 1	Número 2	Salida real	Salida esperada	Verificado
8	20	"El número 2 es el mayor"	"El número 2 es el mayor"	Ok
9	9	"Ambos números son iguales"	"Ambos números son iguales"	Ok
80	21	"El número 1 es el mayor"	"El número 1 es el mayor"	Ok

En este caso se optó por no considerar en la prueba de escritorio una columna para cada evaluación booleana, pero si fuera necesario podrían incluirse cada una de estas expresiones como una columna adicional, de tal forma que se garantice una trazabilidad desde las entradas al resultado obtenido, lo que posteriormente permite, en caso de un error, la determinación de la causa del mismo y la posible corrección o solución.

EJERCICIOS RESUELTOS DE ESTRUCTURAS ITERATIVAS O REPETITIVAS

EJERCICIO 5

Enunciado: Se solicita realizar un diagrama de flujo de un algoritmo que permita sumar todos los números naturales del intervalo cerrado [0;99].

Resolución

Objetivo del algoritmo: Calcular suma de números naturales en intervalo [0;99]

Entradas: No aplica

Salida esperada: Suma calculada

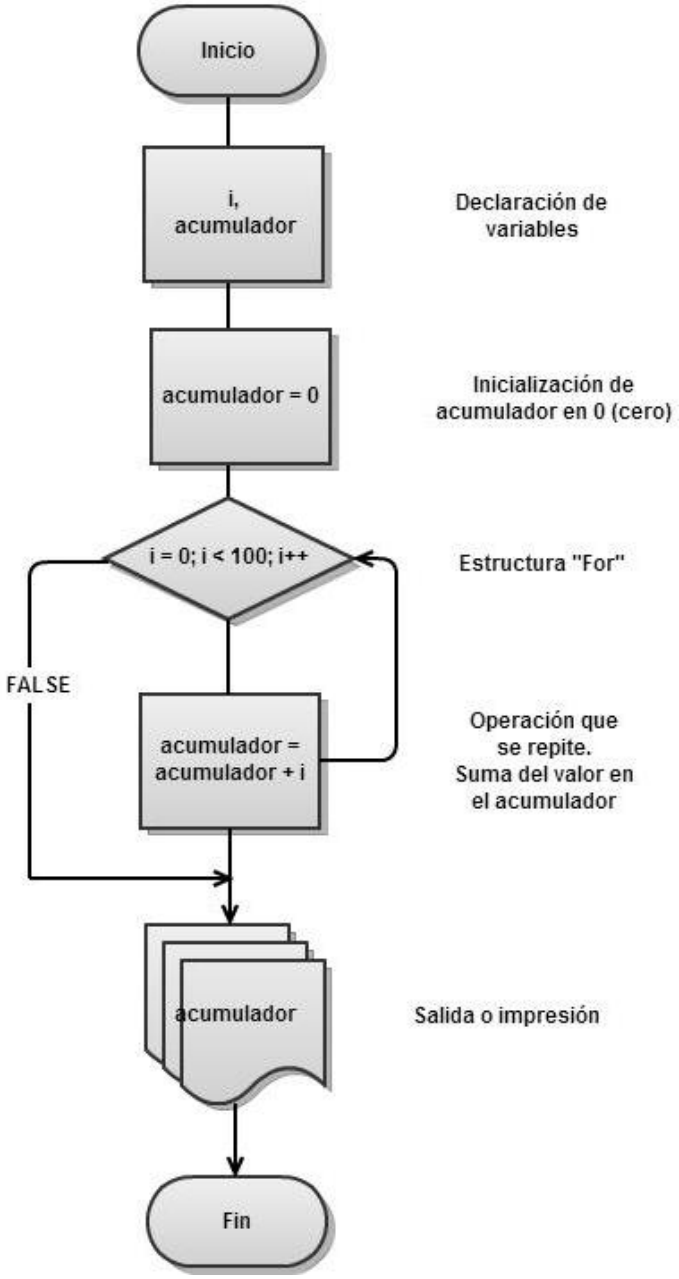
Diagrama de flujo:

Explicación de la construcción del diagrama

1. Bloque de inicio
2. Prever bloque de declaración de variables
3. Inicializar acumulador. En este caso, el resultado de la operación es una suma acumulada de valores entre 0 y 99, por lo tanto necesito de una variables especial de acumulación que será utilizada en cada bucle o ciclo del iterador.
4. Definición de estructura "for".
   Se especifica el valor de inicialización, en este caso 0 (cero), ya que la suma acumulada es entre 0 y 99.

   La condición de corte o paro es: i<100, es decir, el iterador va a ejecutarse hasta que i sea igual o mayor a 100 (ose que el último número que se va a acumular es el 99).

   Incremento: i++, significa que se incrementará la variable "i" una unidad por ciclo o bucle.
5. Cuando se deja de cumplir la condición i<100, se escapa del iterador "for" y se procede a imprimir el resultado "acumulador"
6. Completar el bloque de declaración de variables
7. Bloque de fin



Prueba de escritorio:

Las pruebas de escritorio en algoritmos que tienen alguna estructura de control iterativa comienzan a complejizarse, y ya no se puede expresar toda la operatoria en una sola fila, sino que se van a requerir múltiples filas de la tabla siguiente para realizar un seguimiento del algoritmo en ejecución.

i	acumulador	Salida real	Salida esperada	Verificado
0	0	-	-	-
1	1	-	-	-
2	2	-	-	-
3	4	-	-	-
4	7	-	-	-
5	11	-	-	-
6	16	-	-	-
...	...	...	...	...
97	4753	-	-	-
98	4851	-	-	-
99	4950	4950	4950	Ok

NOTA ACLARATORIA

Con el objetivo de mantener enfocado al alumno en las estructuras de control más sencillas (secuencial y selectiva), a fin de permitirle adoptar los nuevos conceptos en forma incremental, se propone profundizar sobre las estructuras de control iterativas/repetitivas en una etapa más avanzada del curso, por ello no se incluyen ejercicios resueltos de estructuras "While" o "Do-while".

CONCLUSIÓN

El estudio de diagramas de flujo es un buen punto de inicio para lograr un cambio en la estructura mental del alumno y acercarlo a un enfoque lógico, necesario para la resolución de problemas de programación; objetivo fundamental de este primer tramo de la carrera Tecnicatura en Programación.

Esperamos con esto preparar a alumno para adentrarse en el campo de la programación y el desarrollo de software.