Fundamentos de lógica de programación para topografía

Julián Garzón

Gonzalo Jiménez

John Jairo Duque

2018

Fundamentos de lógica de programación para topografía

Julián Garzón B, Gonzalo Jiménez C, John Jairo Duque A.

Facultad de ingeniería

Profesores Universidad del Quindío

Armenia, Quindío, Colombia

2018

Fundamentos de lógica de programación para topografía

Julián Garzón Barrero,
Tecnólogo en Topografía, Ingeniero de Sistemas, Especialista en Geomática, Especialista en SIG, Magister en Sistemas de Información Geográfica.

Gonzalo Jiménez Cleves,
Topógrafo, Especialista en Computación para la docencia, Magister en Ingeniería de Sistemas.

John Jairo Duque Arango,
Topógrafo, Arquitecto, Especialista en Gestión y Planeación del Desarrollo Urbano y Regional, Magister en Ingeniería.

ISBN: 978-958-8801-78-0

Reproducido y editado por: ELIZCOM SAS
www.elizcom.com
ventas@elizcom.com
Celular (57+) 311 334 9748
Fax: (56) (6) +7493244
Armenia, Quindío, Colombia-2018
Tiraje 200 ejemplares.

Dedicatoria

A nuestros estudiantes, que con sus respuestas nos dieron enormes sorpresas.

Julián Garzón Barrero
Gonzalo Jiménez Cleves
John Jairo Duque Arango

Diciembre 2018

juliangarzonb@uniquindio.edu.co

gjcleves@uniquindio.edu.co

jjduque@uniquindio.edu.co

Índice general

Presentación

Es comúnmente aceptado que la misión de aquellos que se forman en las diversas ramas de la ingeniería es detectar, modelar y solucionar problemas asociados a fenómenos locales. Para arribar a las soluciones óptimas es necesario disponer las herramientas suficientes para que los estudiantes desarrollen su pensamiento lógico, pues este tiene efectos de largo alcance y es sin duda es una de las bases que proporcionan madurez y agilidad para asimilar los conceptos de los entornos computacionales.

La primera parte de este libro trata la lógica como concepto fundamental para modelar situaciones problemáticas y diseñar su solución a través de diagramas de flujo y seudocódigo, estas técnicas son conducentes a la solución creativa de problemas en el ámbito topográfico. Los algoritmos de seudocódigo aquí desarrollados, serán estructurados bajo el programa PSeInt versión 20181009 desarrollado por Pablo Novara, de uso libre disponible en http://pseint.sourceforge.net/ y publicado bajo licencia GPL v2. Los diagramas de flujo se realizaron en el software de uso libre para diagramación online draw.io disponible en https://www.draw.io/

En el segundo capítulo se estudia la importancia de la naturaleza de los datos, dado que su tipo es el que define si se pueden o no aplicar operaciones matemáticas, relacionales o lógicas sin causar errores.

En el capítulo tres se tratan las tres estructuras básicas de control que emplea cualquier lenguaje de programación: secuencial, de decisión y cíclica. Estas estructuras determinan el orden de ejecución de las sentencias. Luego tenderemos un capítulo dedicado a estructuras vectoriales y matriciales, dado que es una forma fundamental de almacenamiento de datos topográficos, ampliamente demostrado a través del cálculo de área de predios y ajuste de poligonales.

El capítulo final está dedicado al trabajo de funciones que son unidades de código que permiten que un programa sea estructurado de forma ágil, y que se pueda llamar cuando sea necesario, en lugar de escribir el mismo código repetidas veces.

Por último, el objetivo de este libro es enseñar los fundamentos sobre la construcción de algoritmos y la programación lógica en entornos topográficos, utilizando un lenguaje claro, de tal forma que el estudiante pueda tener un entrenamiento posterior en algún lenguaje de programación de su gusto. El aprendizaje se fortalece con los ejercicios que se encuentran al final de cada capítulo y que darán al estudiante la posibilidad de hacer su entrenamiento y retroalimentación de los conceptos teóricos.

La lógica como aspecto fundamental de la programación

En mi niñez, mi familia y yo solíamos reunirnos para rezar las novenas de navidad en una casa diferente cada día, motivo por el cuál debíamos estar lo mejor presentados posible en cada encuentro. Cuando fue nuestro turno, unos instantes previos a la llegada de la familia, recibí un fortísimo llamado de atención por parte de mi tía, debido a que sin saberlo estaba usando mi zapato izquierdo en el pie derecho y viceversa, ¡qué pasa con tu lógica! Tu pie derecho no es igual que tu pie izquierdo — exclamó— A partir de ese momento empecé a construir el significado de la palabra Lógica: usar el zapato izquierdo en tu pie izquierdo, pero, a la edad de 4 años ¿cómo saber cuál de mis dos zapatos era el izquierdo? Pues bueno, a partir de varios días de observar mis zapatos, detecté que en su parte interna se formaba la curvatura de puente en lados distintos de cada uno de ellos. Asocié esa curvatura con la forma de mi pie, y así logré entender cuál zapato corresponde a cada pie.

Unos años después, por las mismas festividades, y con los zapatos correctamente puestos, pregunté a mi tía ¿qué opinas de mis zapatos? Y ella respondió: ¡al fin le puso lógica a la cosa! Levanté un poco mis pantalones y le dije: ahora, qué opinas de mis calcetines, — ¿he puesto el izquierdo en el pie que corresponde? — y de nuevo me atacó diciendo: ¡a usted le falta mucha lógica! y respondí preguntando ¿acaso una calceta no debería adaptarse al corte anatómico del pie? Inmediatamente fui castigado por discutir con mis mayores.

Luego de tomar varios cursos de lógica en mi pregrado de ingeniería de sistemas y de toparme con diversas definiciones —unas muy complejas y otras muy erradas— he llegado a la conclusión de que lógica es la manera más eficiente de resolver un

problema. Hoy varias décadas después del incidente de los zapatos y las calcetas, sigo creyendo que históricamente hemos tenido dos calcetines del mismo pie.

1.1 ¿Qué es un problema?

Cientos de personas despiertan cada día tratando de solucionar los problemas del mundo —*lo cual es asombroso*— pero, el mundo sigue lleno de problemas, algunos extremos como la inseguridad alimentaria o el cambio climático, otros molestos como el tráfico, o algunos menores como que tu cinturón no se adapta a los cambios de tu cintura.

Un problema se define como la brecha que existe entre el estado percibido de una situación específica y su estado ideal. Aunque muchos problemas tienen varias soluciones—*los medios para cerrar la brecha o corregir la desviación*— surgen dificultades donde, dichos medios no son obvios o no están disponibles de inmediato, por tanto, es necesario un modelo de solución o cambio de estado a través de la lógica.

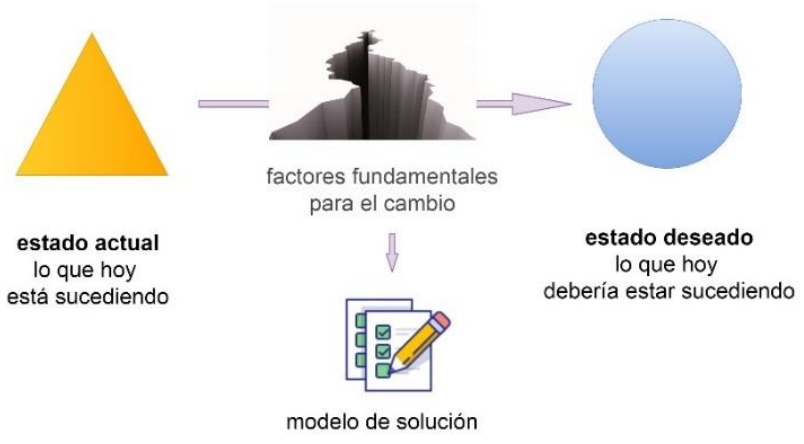


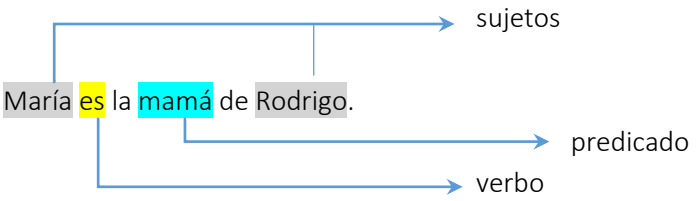
Figura 1. Situación problemática.

Si entendemos la lógica como la manera más eficiente de resolver un problema, será necesario asociarla a tres elementos fundamentales: sintaxis, semántica y reglas de inferencia. La sintaxis se refiere a la forma en que se juntan las palabras para formar oraciones. La semántica es el significado llevado por las expresiones, y las reglas de inferencia describen la forma correcta de derivar conclusiones.

María es la mamá de Rodrigo.
Gilma es la mamá de María.
Valentina es la mamá de Gilma.
Rodrigo es el papá de Amanda.

¿Quién es la abuela de Amanda?

Sintaxis:



Semántica:

Mamá: es una mujer que ha tenido al menos un hijo.
Papá: un hombre que ha tenido al menos un hijo.

reglas de inferencia:

1) Alguien es padre de otra persona, si ese alguien es o la mamá o el papá de esa otra persona.

2) Para que alguien sea la abuela de otra persona ese alguien deberá ser la mamá de alguien más que sea padre o madre de esa otra persona.

1.2 ¿Cómo solucionarlo?

Todos los seres humados tenemos problemas a diario, algunos muy cotidianos que solucionamos rápida e intuitivamente, o usamos alguna estrategia que funcionó en el pasado. Por ejemplo, si se quedó dormido en la mañana y llegará tarde al trabajo, puede llamar a su jefe y explicar su situación mientras se pone a punto en la mitad del tiempo habitual.

Los problemas se vuelven más complejos cuando no existe solución obvia y estrategias que has intentado antes, ahora no funcionan. Este tipo de problemas causan gran estrés y requieren estrategias innovadoras de solución.

Pues la forma de obvia de resolver un problema es buscando su solución, claro, en esta búsqueda es fundamental el planteamiento de un enfoque organizado que tenga:

Figura 2. Estructura de solución de problemas.

La capacidad de dar solución a problemas es fundamental en el ejercicio de cualquier profesión, especialmente aquellas asociadas a la ingeniería. Las soluciones propuestas serán eficaces en la medida en que el observador tenga gran capacidad de abstracción y pueda construir modelos con fundamentos lógicos (Serna & Polo, 2014).

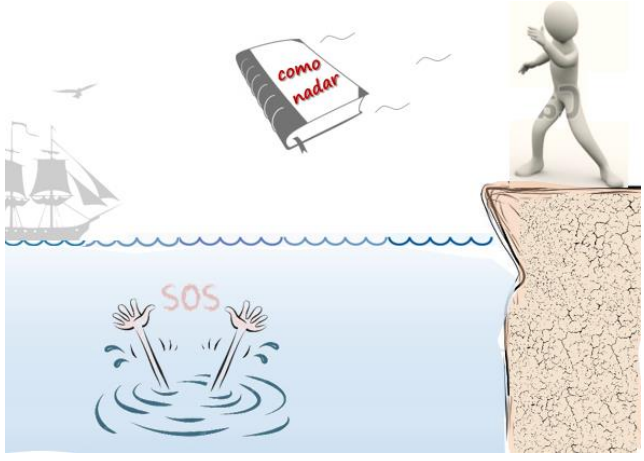


Figura 3. Modelo errado de solucionar un problema

¿Cuáles son las figuras que hacen falta en la siguiente serie?

Figura 4. Ejercicio de observación.

La serie tiene 9 figuras que se pueden reunir en tres grupos, cada grupo está conformado por tres elementos.
Las variables que intervienen son: forma, color y tamaño.
Desde la forma su relación es 2:1
Desde el color su relación es 1:2
Desde el tamaño su relación es 2:1
La relación entre los dos primeros grupos se presenta en la secuencia del color: una figura azul seguida de dos verdes, y en la secuencia del tamaño: dos figuras grandes seguidas de una pequeña. Ahora, analizando lo dicho, la serie se completará de la siguiente forma:

Figura 5. Solución ejercicio serial.

Ahora, queremos transcribir un cuento antiguo asociado a la solución creativa de problemas, proveniente de (Shah, 1967).

La fábula de los tres peces

"Había una vez tres peces que vivían en un charco. Uno de ellos era muy inteligente, el otro era un poco inteligente y el tercer pez era tonto. La vida transcurría para ellos como para los peces de cualquier lugar, hasta que un día llegó un hombre.

El hombre llevaba una red y el pez más inteligente lo vio a través del agua. Recurriendo a su experiencia, a los cuentos que había leído y a su habilidad, decidió ponerse en acción. "Hay pocos lugares para esconderse en este charco", pensó, "por lo tanto, fingiré estar muerto".

Reunió todas sus fuerzas y saltó fuera del charco, cayendo a los pies del pescador, quien quedó bastante sorprendido. Pero como el pez inteligente estaba conteniendo su respiración, el pescador pensó que estaba muerto y lo arrojó nuevamente al agua. Entonces, el pez se deslizó hacia una pequeña cavidad en la orilla.

El segundo pez, que era sólo un poco inteligente, no entendía del todo bien lo que estaba pasando. De modo que nadó hacia el pez más inteligente y le preguntó qué había hecho. El pez le respondió:

— Es simple. Fingí estar muerto. De este modo, el hombre me arrojó nuevamente al agua.
Entonces, el pez poco inteligente saltó inmediatamente fuera del agua, a los pies del pescador. "Qué extraño", pensó éste, "los peces están saltando a mi alrededor". Y como el pez poco inteligente había olvidado contener la respiración, el pescador se dio cuenta de que estaba vivo y lo puso en su bolsa.

Luego, se dio vuelta para observar atentamente dentro del agua y olvidó cerrar la solapa de la bolsa. El pez, entonces, aprovechó para liberarse y moviéndose a sacudidas logró saltar hasta el charco. Jadeante, nadó hacia la cavidad donde estaba escondido el primer pez y se quedó a su lado.

Mientras tanto el tercer pez, el tonto, no comprendió nada de lo que había pasado. Se acercó a los otros dos peces, que le contaron cada detalle, poniendo de relieve la importancia de no respirar para fingirse muerto.

— Muchísimas gracias — dijo el pez tonto —. Ahora entendí.

Diciendo esto, se arrojó fuera del agua y cayó junto al pescador.

Entonces, el pescador, que ya había perdido dos peces, puso al pez tonto en la bolsa sin molestarse en ver si respiraba o no. Pero esta vez cerró bien la solapa para que no se escapara. Luego tiró su red al charco sin ningún resultado, porque los otros peces estaban bien escondidos en la cavidad de la orilla.

Finalmente, el pescador se dio por vencido. Abrió su bolsa, comprobó que el pez tonto no respiraba y se lo llevó a su casa para el gato."

En la fábula se plantean diferentes modelos de solución a un mismo problema. El pez inteligente analiza la situación y con la información que posee rápidamente plantea y ejecuta su modelo de solución. Luego el pez poco inteligente, sin comprender completamente el problema usa el mismo modelo de solución de su amigo inteligente, y por poco le cuesta la vida. Por último, el pez tonto, llevó a cabo el mismo modelo de solución de forma minuciosa, su resultado fue un completo fracaso.

Este análisis sugiere que el modelo de solución a un problema deberá ser propio, los modelos reproducidos de otros producen impresiones, y deberán aplicarse rápidamente.

Ahora, haremos la conexión entre la lógica y uno de los problemas del quehacer diario de la topografía: el procedimiento de implantación espacial de proyectos, para ello daremos una revisión rápida al tema a través de los numerales 1.3, 1.3.1, 1.3.2 y 1.3.3.

1.3 El problema de localización en topografía

En el ámbito de la construcción el mercado cada día se hace más competitivo, cada vez aparecen más hardware y software disminuyendo el tiempo de ejecución y aumentando la calidad del producto final. Desde la óptica topográfica, la delicada práctica de las mediciones de obra exige de forma casi continua la resolución de problemas, que por su delicada naturaleza tendrán que ser abordados en tiempos cortos, de forma exitosa y bajo condiciones extremas.

El procedimiento topográfico de localización consiste en transferir y materializar sobre un marco de referencia —asociado a un sistema de referencia espacial—, de forma precisa, los puntos básicos que definen el proyecto mediante uso de instrumental topográfico.

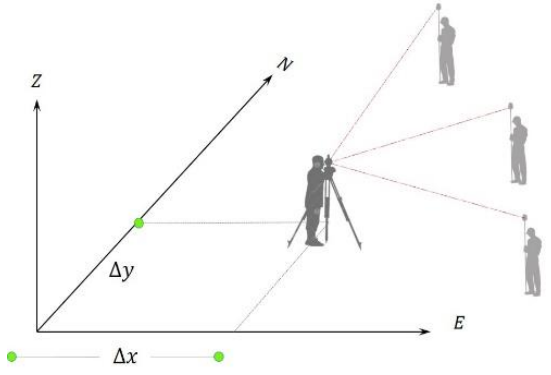


Figura 6. Sistema de localización polar.

Como es bien sabido, en el posicionamiento de un proyecto de ingeniería se aplican conceptos de precisión y eficacia, es decir, lograr el objeto específico con la calidad necesaria que garantice las tolerancias constructivas, usando la menor cantidad de

recursos en el menor tiempo posible. Para ello se deben cumplir las normas mínimas exigibles, teniendo conocimiento de qué y cómo se hará.

En nuestro país, las grandes obras de ingeniería no habrían podido ser construidas sin el soporte de la topografía y las mediciones especiales, por lo tanto, de la presencia activa de un topógrafo. No obstante, este espacio de acción tradicional de la topografía se ha ampliado considerablemente, cubriendo la exigencia de medianas y pequeñas obras de ingeniería.

Los datos para la localización provienen de información asociada con la geometría de la obra, que podrían ser de desarrollo lineal (carreteras, ferrocarriles, líneas de transmisión eléctrica, alcantarillados) o superficial (proyectos urbanísticos). La localización define el aspecto físico de la obra: su forma, tamaño, dimensión y posición.

La documentación del proyecto comprende planos de topografía base, memorias de estudios, y los planos propios del proyecto que definen su geometría y posición (arquitectónicos, estructurales, alcantarillado…). Esta información puede ser manejada desde un sistema CAD desde donde se toma la información necesaria y precisa para la localización de puntos, esta información se extrae directamente del plano trazado en el computador, el cual permite tomar medidas que son generadas a partir de un banco de datos alimentado con información procesada mediante un software adecuado. Es importante resaltar que el computador no puede reemplazar el criterio de la persona encargada de manejar la información, si se ingresan datos erróneos al sistema el resultado será desastroso, igual que si los datos se ingresan correctamente, pero, se procesan mal, el resultado será un proyecto fracasado.

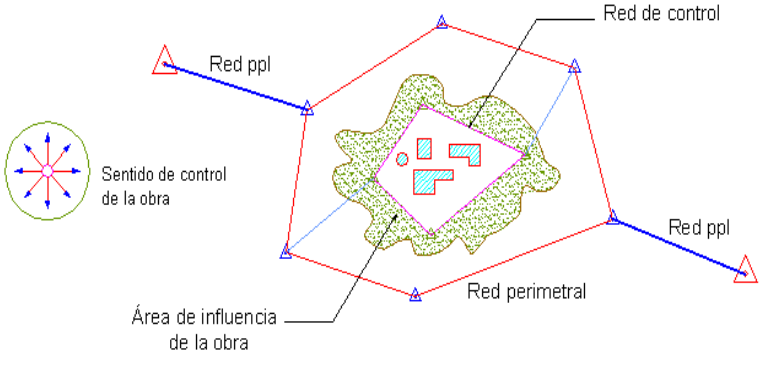


Figura 7. Esquema de red de control de la obra.

1.3.1 El sistema de apoyo geométrico

El sistema de apoyo geométrico (SAG) es la estructura que vincula el diseño con la realidad, debe ser cerrado gráfica y analíticamente mediante un polígono formado por líneas rectas sucesivas con puntos comunes llamados vértices, de los cuales se deberá conocer su incertidumbre en términos de la elipse de error, asegurando, que los vectores que dé él se deriven cumplan las tolerancias constructivas de los elementos a localizar. Para mayor profundidad del tema sugerimos leer (Garzón, Jiménez, & Cifuentes, 2016).

La precisión del SAG no depende solo de la precisión de las mediciones sino también de su configuración geométrica, para que una red sea fuerte y confiable los vértices deben encontrarse lo más uniforme posibles, la fuerza geométrica de las poligonales es especialmente baja cuando se cambia la dirección de un vector en ángulo de 30° con relación a la dirección principal del avance.

En el diseño de la red se debe tener en cuenta:

1. El desarrollo de la obra (superficial, lineal, o en altura).
2. La ubicación de los vértices.
3. Los instrumentos de medición y el método a usar.

Se debe plantear un diseño ágil para su utilización, que permita por su geometría la aplicación de distintos métodos de localización (polar, rectangular, libre estacionamiento…). La fuerza geométrica de una figura puede imaginarse como la estructura de un andamio y su fuerza relativa. Las figuras más fuertes son el triángulo equilátero y el cuadrilátero doble, pero debido a las dificultades que se presentan en la práctica estas generalmente no se acoplan a las obras de ingeniería, motivo por el cual se realizan distintos polígonos con diversas connotaciones desde el punto de vista de su rigidez, en un SAG, como con una estructura de andamio, entre más agudos sean sus ángulos interiores, más débil será la estructura. Entre mayor sea la rigidez que una red tiene, más seguro se puede estar que las mediciones realizadas son precisas. No sólo es importante la fuerza de las figuras geométricas individuales, sino también la manera en que estas se interconectan y relacionan entre sí en la red.

El SAG debe encerrar toda el área de trabajo, con el objeto de que cualquier punto que se localice desde un vértice resulte contenido dentro del polígono principal, y que no sea el resultado de extrapolación que ocurra por fuera del marco de referencia. El

mejor diseño será aquel que contenga la mayor cantidad de figuras geométricas cerradas, lo más homogéneas posibles garantizando rigidez al sistema.

1.3.2 El sistema de apoyo secundario

Se deriva del SAG, y deberá ser diseñado sobre el plano de localización teniendo en cuenta la posición de los elementos de detalle, deberán garantizar cierta permanencia en el tiempo de ejecución de la obra. A partir de su posición es que finalmente se fijan los ejes auxiliares y puntos de detalle que definirán la obra.

Los ejes de detalle se sitúan fuera del área de excavaciones y por lo general, están marcados con puntillas que definen zapatas, columnas y cimentaciones; desde las puntillas se tienden hilos que materializan las líneas que sirven de guías para que los obreros realicen su trabajo.

En función de las características del trabajo y de la importancia del punto su señalización varía, la materialización de puntos debe presentarse en un formato tal que todo el equipo humano que trabaje en el sitio pueda interpretar, respetar.

Generalmente se pueden definir tres tipos de puntos:

1. Puntos de localización primarios.
2. Puntos de localización secundarios.
3. Puntos de detalle.

Los puntos de localización primarios son los vértices del SAG que deben estar ligados a un sistema local de coordenadas para su orientación. Estos puntos deben permanecer durante toda la ejecución de la obra. Su materialización se recomienda con una placa de bronce empotrada en concreto, y deberán ser emplazados en sitios estratégicos de tal forma que no se vean afectados por el tráfico de la obra.

Los puntos de localización secundarios son aquellos se ubican cerca de los de detalle del proyecto, deberán ser localizados en un sitio estratégico donde no se vean afectados por algún tipo de desplazamiento (X, Y, Z) a causa de movimientos ocasionados por maquinaria pesada o por tráfico obligatorio de la obra, es totalmente entendible que todos los puntos no tengan permanencia absoluta dentro de la obra, pero éstos al menos deben ser garantizados durante la ejecución puntual de la obra, es decir, los puntos de detalle que dependan de él, deben estar completos antes de su desaparición.

Dado su carácter de temporalidad, para su materialización se aceptan estacones de madera de sección grande (aprox. 8 x 8 cm), sobre el cual se define el punto con una puntilla, dichos estacones deberán estar embebidos en una mezcla de concreto con esto se garantiza su vida útil al menos durante el periodo de construcción puntual de la obra.

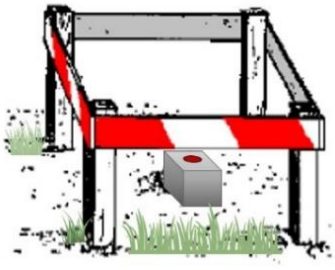


Figura 8. Materialización de puntos secundarios

Todos estos puntos deben de reunir una serie de condiciones:

1. Estabilidad dimensional: que no varíe de forma y tamaño.
2. Estabilidad del material: debe estar construido con materiales resistentes a los agentes externos, tanto atmosféricos, como personas, animales o máquinas.
3. Estabilidad espacial: no variar de situación o posición absoluta en el espacio.
4. Con visibilidad, sobre la zona donde se encuentren los puntos a replantear y el resto de bases de replanteo.
5. Fácilmente localizable: de forma que pueda ser encontrado con rapidez.
6. Materializar de forma adecuada, fina, precisa e inequívoca.
7. Fácilmente estacionable.
8. Fácilmente observable.

Los puntos de detalle son finalmente los que definen el proyecto, los que marcan las características del trabajo como: pilas, zapatas, columnas, ejes viales, paramentos, inicio de brechas y todos aquellos puntos que definen tridimensionalmente el proyecto. Se suele utilizar para su señalización estacas de madera y puentes de referencia—*debido a su bajo costo y simplicidad de implantación en el terreno*— que mediante hilos tendidos adecuadamente materializan los ejes de construcción.

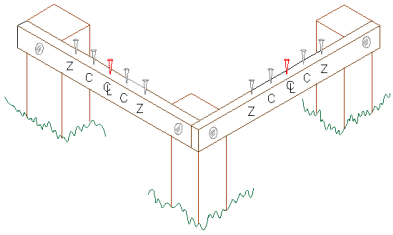


Figura 9. Puntos de detalle

1.3.3 El plano y la cartera de localización

Una vez definidos los ejes principales, el método o métodos de localización, las coordenadas de los puntos de intersección de los ejes, todo bajo el mismo sistema de referencia, se procede a realizar los trabajos de campo necesarios para la localización de puntos.

El archivo donde se registran los datos de localización constituye un documento tan importante como el plano de localización, puesto que los dos conforman el engranaje final para la ubicación de los puntos que definirán el proyecto, para ésta tarea no sirve llevar a campo solo el plano, ni solo la cartera de localización, puesto que uno es el complemento del otro. Este documento debe aparecer con total claridad para el entendimiento del topógrafo, y no deben omitirse datos. Debe indicarse cuál será el punto de estación, cuál el de orientación, así como las coordenadas de los puntos, cada uno de los ángulos y distancias horizontales a medir desde el punto de estación al punto de detalle.

```
Proyecto:                    Localización manzanas La Nueva Granada
Topógrafo:                   Pedrito Pérez
Fecha inicio:                agosto 25 de 2018
Fecha finalización:          agosto 25 de 2018
Instrumento:                 Leica TPS-802  emc:2"; 3mm+2ppp
Puntos a localizar:          150
Temp. ambiente:              ±25°C
```

Cartera de localización

```
Pto estación:    1000      N       981266.850          Delta 1
                           E      1152306.420          1205.00

Pto orientación: 2000      N       981269.650          Delta 2
                           E      1152315.520          1204.00

Pt    Azimut          Distancia        Coordenadas      Desc/elev
================================================================
```

1	300-00-00	13.00m	N	981273.350	Ref-1
			E	1152295.162	1205.91
2	300-00-00	19.00m	N	981276.350	Ref-2
			E	1152289.966	1206.34
3	53-32-25	68.19m	N	981307.372	eje camino
			E	1152361.263	1203.99
4	48-44-10	65.74m	N	981310.207	eje camino
			E	1152355.835	1204.04

Pt	**Azimut**	**Distancia**	**Coordenadas**		**Desc/elev**
5	43-59-30	62.50m	N	981311.815	eje camino
			E	1152349.830	1204.03
6	35-34-10	55.98m	N	981312.385	eje camino
			E	1152338.983	1204.08
7	32-27-00	52.96m	N	981311.541	eje camino
			E	1152334.836	1203.99
8	29-30-10	49.21m	N	981309.679	eje camino
			E	1152330.654	1204.03

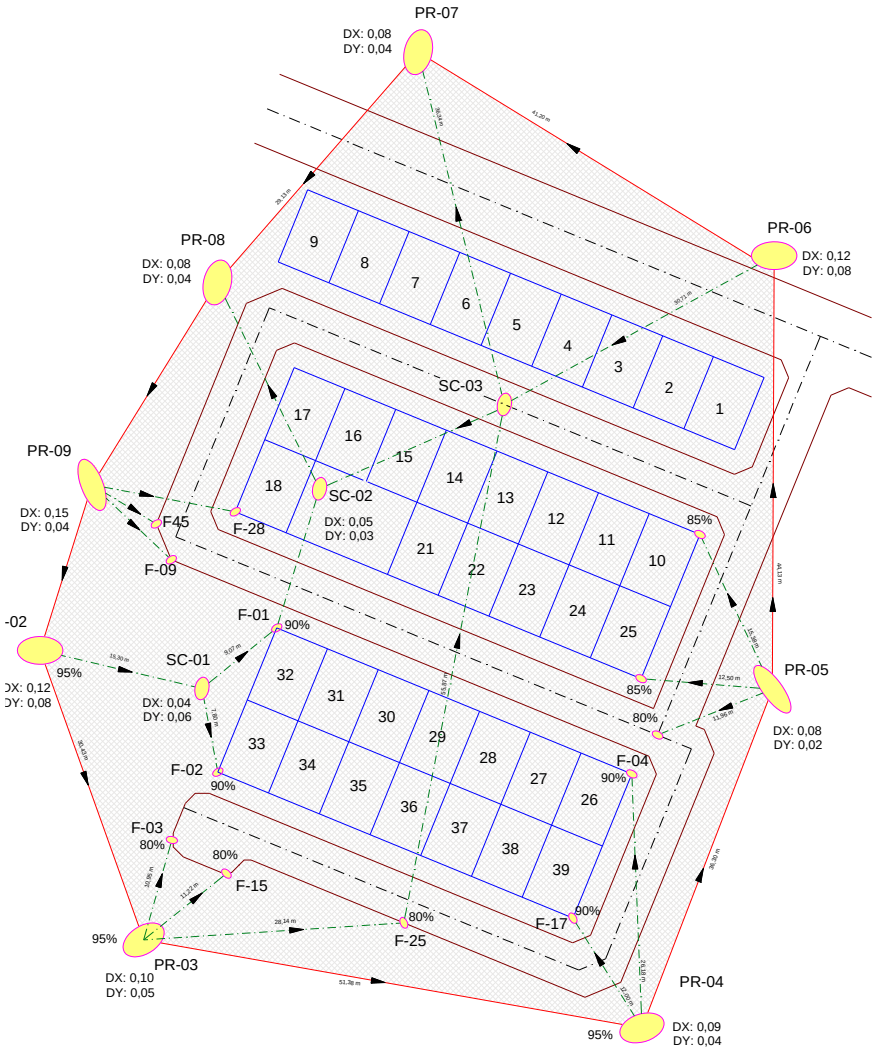


Figura 10. Plano de localización

1.3.4 Argumentos, premisas y conclusiones

Cuando el razonamiento se expresa con palabras, recibe el nombre de "argumento". El argumento es la parte de la lógica que demuestra con fundamentos que lo que se expresa es cierto. Las premisas de un argumento son proposiciones previas a la conclusión.

Argumento:

1. Todos los seres humanos tienen pelo.
2. Carlos Humberto es un ser humano.

3. Por lo tanto, Carlos Humberto tiene pelo.

En este argumento, las proposiciones 1 y 2 son las premisas, y la proposición 3 es la conclusión.

"Todos los ángulos internos de un polígono suman (n-2)*180°" aparece como premisa en el siguiente argumento:

1. Todos los ángulos internos de un polígono suman (n-2)*180°
2. Las poligonales topográficas se deben ajustar
3. Por lo tanto, en todas las poligonales topográficas ajustadas la sumatoria de sus ángulos interiores es igual a (n-2)*180°

Y como conclusión en el siguiente argumento:

Todas las poligonales topográficas cerradas se deben ajustar

1. En un mismo vértice la suma de los ángulos interno y externo es igual a 360°
2. La sumatoria teórica de ángulos externos de una poligonal cerrada es igual a (n+2)*180°
3. Por tanto, todos los ángulos internos de un polígono suman (n-2)*180°

Expresiones como "por tanto", "por lo cual", "de ello se deduce", sirven para introducir la conclusión de un argumento, en tanto que "pues" y "porque" se usan para introducir las premisas.

Enunciado Simple: Es el que no contiene otro enunciado como parte componente.

"Los azimutes no pueden ser mayores a 360°"

Enunciado compuesto: Es el que se compone de varios enunciados.

"Los azimutes no pueden ser mayores a 360° y las deflexiones no pueden ser mayores que 180°".

Cuando los enunciados se unen por la conjunción Y, se denominan enunciados conyuntos. Cuando se unen por el conector O, se denominan enunciados disyuntos. De aquí surgen las tablas de verdad. (Tema que será tratado en *operadores lógicos*).

1.4 Algoritmos

Actualmente, el manejo de información es caracterizado por el volumen de datos y su velocidad de cambio, esto hace que en todas las profesiones—e*specialmente en las ingenierías*—se tenga la necesidad de interactuar con hardware, software, y lenguajes de programación para desarrollar aplicaciones útiles que permitan solucionar problemas acudiendo a la lógica.

Un algoritmo es un método para resolver un problema, presentado como una secuencia ordenada de instrucciones precisas, lógicas y finitas que conllevan a su solución. En otras palabras, los algoritmos son mapas de ruta para llevar a cabo una tarea dada y bien definida. Entonces, un código que calcula los términos de la secuencia de Fibonacci es una implementación de un algoritmo en particular. Incluso una función simple para sumar dos números es un algoritmo en cierto sentido, aunque sea simple.

Algunos algoritmos, como los que calculan las secuencias de Fibonacci, son intuitivos y pueden estar inmersos de manera innata en nuestras habilidades de pensamiento lógico y resolución de problemas. Sin embargo, existen algoritmos más complejos y que la mayoría de nosotros usamos aun cuando no se ven, pero que están en todas partes, por ejemplo, al revisar el correo electrónico desde tu móvil, o escuchar música en la computadora reproducida desde YouTube, o la ruta que sigue el sistema de navegación de tu carro, son ejemplos claros de su inmersión en nuestras vidas diarias.

1.5 Técnicas de representación de algoritmos

La creación de un algoritmo es la parte más creativa de la programación, no importa su complejidad, siempre podrá adaptarse el modelo que proviene de un popular refrán: '*Divide y vencerás'*, donde, se resuelve un problema complejo en partes más simples, y luego, integrarlas a todo el conjunto para obtener la solución. Existen tres técnicas para representar algoritmos: descripción narrada, diagrama de flujo y seudocódigo. No importa cuál se utilice, todos los algoritmos siguen ciertos principios de diseño.

1.5.1 Descripción narrada

Se caracteriza por seguir un proceso de ejecución común y lógico, describiendo textualmente cada una de las actividades a realizar a través de pasos.

Veamos un ejemplo clásico de topografía para la delicada tarea de la medición de un ángulo por reiteraciones:

Nombre del algoritmo:	Reiteraciones
Descripción:	Protocolo para la medición de un ángulo por reiteración (entre dos señales) tres veces, (tres posiciones)
Autor:	JGB - GJC - GGG
Fecha:	diciembre 2018
Pasos:	

Paso 1 - Visar la señal de la izquierda y colocar en el limbo 00° 00' + una lectura en segundos superior a cero.

Paso 2 - Barrer el ángulo en sentido horario, puntear la segunda señal y registrar en cartera el valor leído en el círculo horizontal.

Paso 3 - Transitar, volver a puntear la misma señal, registrar en cartera el valor que se lee en el círculo (debe diferir en 180° con el valor leído en el paso 2).

Paso 4 - Barrer el ángulo en sentido contra-horario hasta la primera señal, puntearla y registrar la lectura del círculo. {Hasta este punto el ángulo se ha medido dos veces, una en directa y sentido horario, otra en inversa y sentido contra-horario.}

Paso 5 - Estando el instrumento punteando la señal de la izquierda en posición inversa, colocar en el círculo el valor correspondiente al 2do. origen (en inversa) de manera que al transitar el instrumento quede en posición directa, y en el círculo pueda leerse directamente el origen correspondiente.

Paso 6 - Barrer el ángulo en sentido horario hasta la segunda señal, puntearla y registrar la lectura correspondiente.

Paso 7 - Transitar, volver a visar la segunda señal, registrar la lectura.

Paso 8 - Barrer el ángulo en sentido contra-horario hasta la primera señal, puntearla y registrar la lectura correspondiente. {En este punto se han hecho dos posiciones del ángulo, es decir se ha medido dos veces en sentido horario y dos en sentido contra-horario, tanto en directa como en inversa}.

{Para iniciar la tercera posición:}

Paso 9 – Igual al paso uno, pero, teniendo en el círculo horizontal como origen el valor de 120°, puede revisar los diferentes orígenes referidos en la Tabla 1, y continuar con el procedimiento mencionado. {El procedimiento descrito se repite hasta completar el número de posiciones requerido (mínimo dos), debe notarse que 1a., 3a., 5a., etc. empiezan en directa y 2a., 4a. etc. empiezan en inversa}.

Tabla 1
Orígenes para las reiteraciones.

Tel.	D	I	D	I	D	I
2	00°	270° (90°)				
3	00°	240° (60°)	120°			
4	00°	225° (45°)	90°	315° (135°)		
5	00°	216° (36°)	72°	288°(108°)	144°	
6	00°	210° (30°)	60°	270° (90°)	120°	330°(150°)

Ahora, construyamos otro algoritmo, usando esta misma técnica. El algoritmo deberá describir el proceso para materializar un mojón en concreto, con placa metálica que tenga una coordenada Y fija, sobre un eje perpendicular al eje de referencia. Se deberá usar estación total para las mediciones necesarias.

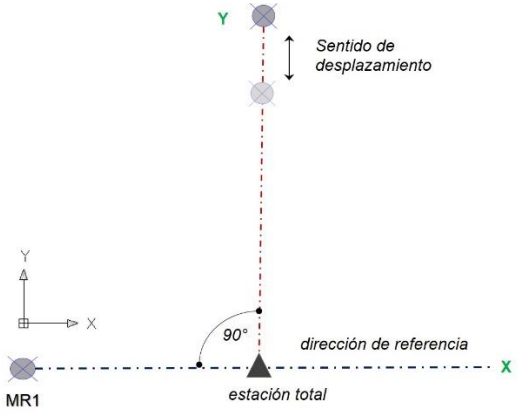


Figura 11. Referenciación

Nombre del algoritmo:	Referencia topográfica
Descripción:	Protocolo para la medición y materialización sobre mojón con placa metálica.
Autor:	JGB - GJC - JJDA
Fecha:	febrero 2018
Pasos:	

Paso 1 – Fijar la dirección angular en la estación total y medir la distancia fija sobre su alineamiento.

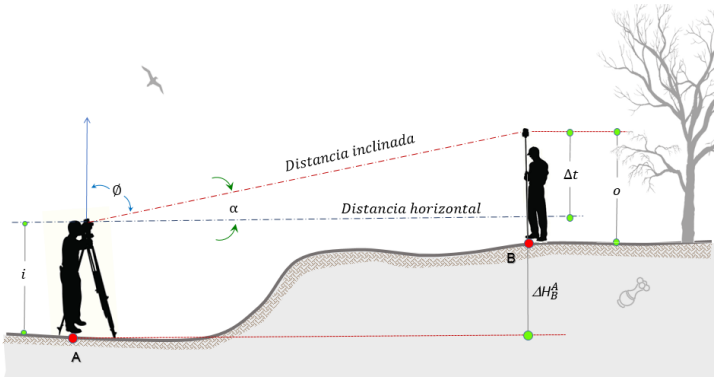


Figura 12. medición de distancia fija.

Paso 2 – clavar una estaca de madera en el lugar definido en el paso 1.

Paso 3 – Los ayudantes de campo deberán tender un hilo que permita alinear la estaca con la dirección fija de la estación total (desviación ±2 cm) para realizar la excavación del mojón.

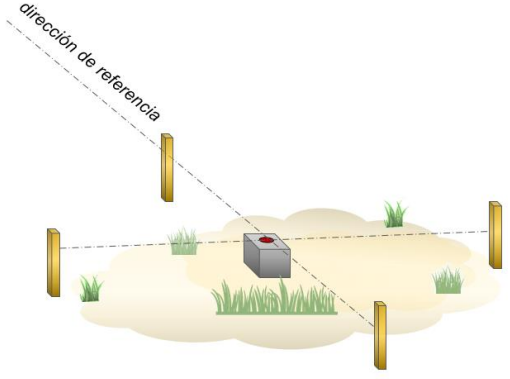


Figura 13. alineamiento auxiliar.

Paso 4 – Los ayudantes de campo deberán preparar la mezcla, fijarán la formaleta y harán el vaciado del mojón donde estaba la estaca, sin perder las estacas con las que se tienden los hilos del Paso 3.

Paso 5 – Cuando el concreto haya fraguado un poco, en el centro del mojón se incrustará una placa metálica de 10 x 10 cm con ganchos soldados para asegurar su estabilidad.

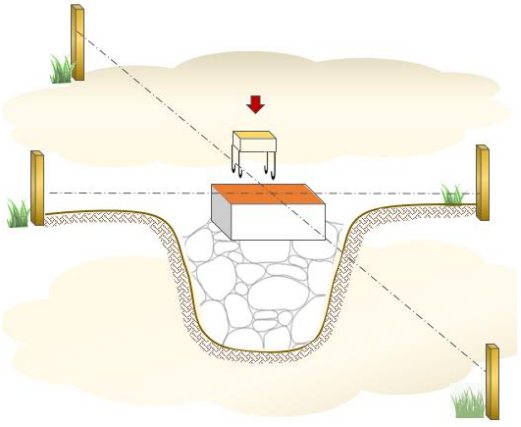


Figura 14. Colocación de la placa.

Paso 6 – Cuando el concreto haya fraguado totalmente, se replanteará angularmente sobre la placa metálica la dirección angular (90°) en las dos posiciones del círculo (D e I).

Paso 7 – Marcar el punto con lápiz en la dirección promedio de las II posiciones del círculo.

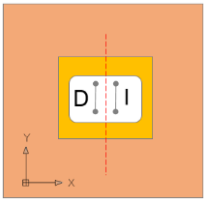


Figura 15. Marcas de replanteo de las posiciones directa e inversa sobre la placa.

Paso 8 – Replantear la distancia con prisma puesto sobre trípode, buscando la distancia que se necesita desde la estación topográfica para definir la coordenada Y del mojón.

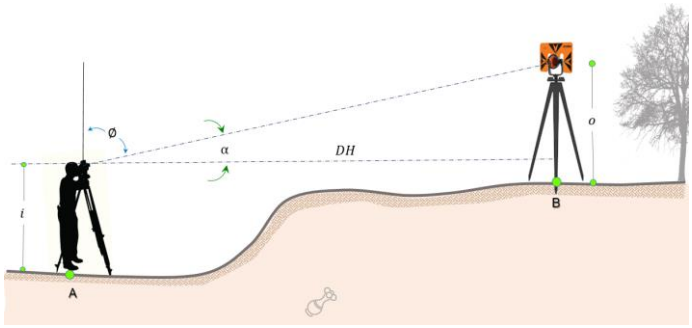
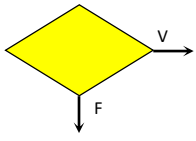
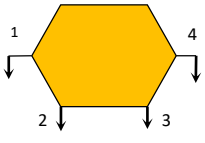


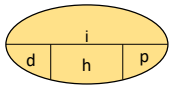
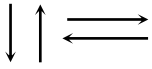
Figura 16. Replanteo de distancia

1.5.2 Diagramas de flujo (DF)

Son gráficos estandarizados de la secuencia lógica de los datos que determinan el proceso de los algoritmos, usan un esquema intuitivo para indicar las operaciones en cajas conectadas por líneas y flechas que gráficamente indican el control de flujo de datos. La dirección principal del flujo se acepta de arriba abajo y de izquierda a derecha. Si el diagrama está completo y correcto, su paso a un lenguaje de programación es relativamente simple y directo.

Tabla 2
Simbología básica de diagramación.

símbolo	Nombre	Función
	inicio/final	indica el comienzo o punto final del diagrama
	entrada	Se usa para ingresar datos. Expresa lectura.
	proceso	Transferencia de datos u operaciones aritméticas.
	decisión	Verifica el resultado de una condición, dependiendo de su resultado se sigue por uno de los dos caminos alternativos(V ó F)
	decisión múltiple	Es un selector, dependiendo de su valor, se sigue por uno de sus caminos.

símbolo	Nombre	Función
	ciclo desde	Es una estructura de repetición que ejecuta sentencias un número definido de veces. d:desde, h:hasta, p: paso
	flujos	Es el sentido que toman las instrucciones dentro del diagrama.
	salida	Representa información de resultados que el usuario puede leer.
	conector interno	Es un enlace entre dos partes del diagrama en la misma página.
	Conector externo	Es un enlace entre dos partes del diagrama en hojas diferentes.

Los DF deben cumplir los siguientes principios de diseño:

1) Tener inicio y final.
2) Cada símbolo se asocia con una función específica.
3) Las flechas de flujo deben ser rectas, verticales y horizontales.
4) Todas las líneas deberán estar conectadas a algún símbolo, y no se deberán cruzar.
5) Los diagramas se leen de arriba hacia abajo, y de izquierda a derecha.
6) Las operaciones se ejecutan hasta que un símbolo terminal muestre el final de la ejecución del proceso.

Siguiendo los principios de diseño, vamos a construir un DF para calcular las proyecciones meridianas y paralelas, teniendo como datos de entrada los datos de campo: azimut y distancia horizontal.

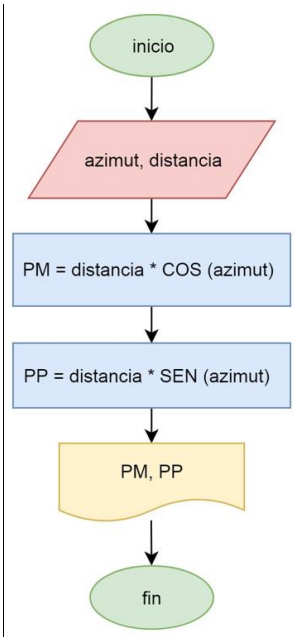


Figura 17. DF proyecciones

Si deseamos construir un DF para calcular la distancia horizontal, a partir de las coordenadas planas cartesianas de los puntos involucrados, deberíamos modelarlo de la siguiente forma:

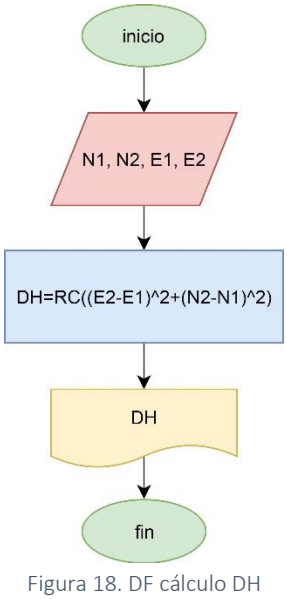


Figura 18. DF cálculo DH

Ahora, queremos desarrollar un algoritmo que permita leer un azimut, y determinar si dicho valor coincide con la dirección Norte, de no ser así, se deberá indicar si su magnitud se encuentra en el rango permitido de sus fluctuaciones: 0°< azimut <360°. Se puede plantear de la siguiente manera:

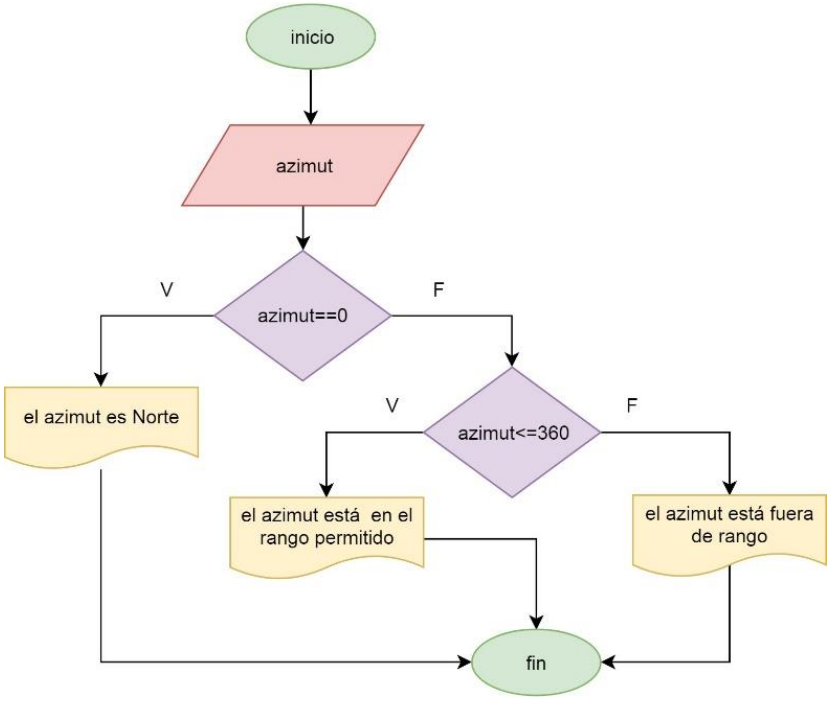
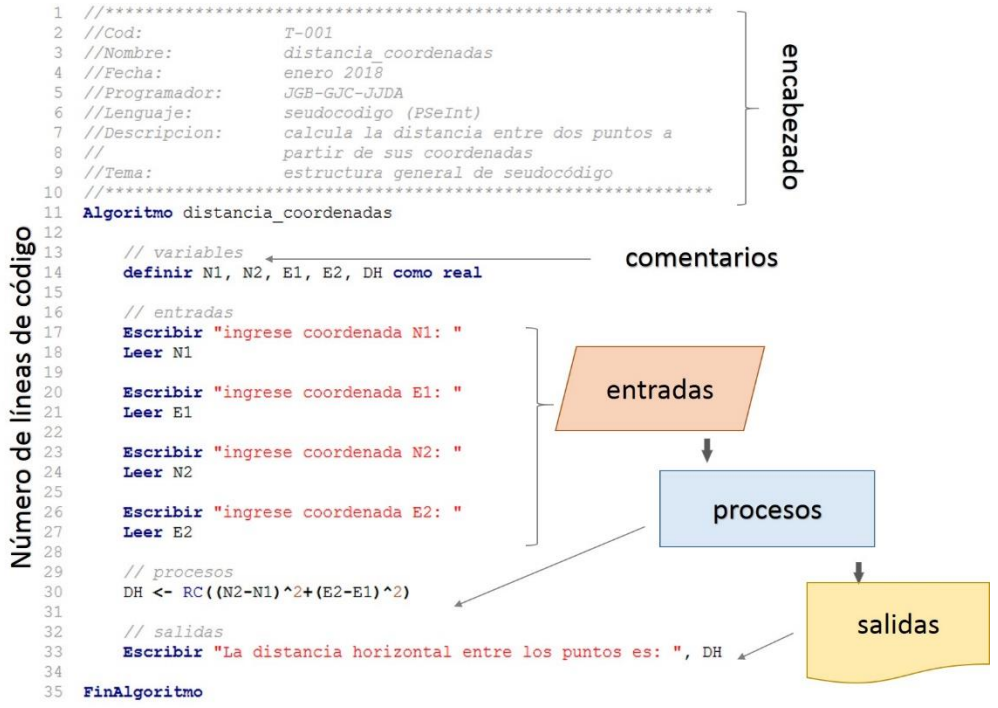


Figura 19. DF validación azimut.

1.5.2 Seudocódigo

Es un modelo informal y textual de alto nivel que se usa para construir un algoritmo, utiliza estructuras de un lenguaje de programación a través de palabras reservadas de su lenguaje informal, usadas de manera detallada y legible. El objetivo del seudocódigo es comprender las estructuras de programación sin necesidad de centrarse en la sintaxis de un lenguaje específico. Su resultado es un producto lógico que define las estructuras y operaciones que se aplicarán a los datos.

Veamos un ejemplo del algoritmo de la Figura 20 escrito en seudocódigo:




```
1  //*************************************************************
2  //Cod:              T-001
3  //Nombre:           distancia_coordenadas
4  //Fecha:            enero 2018
5  //Programador:      JGB-GJC-JJDA
6  //Lenguaje:         seudocodigo (PSeInt)
7  //Descripcion:      calcula la distancia entre dos puntos a
8  //                  partir de sus coordenadas
9  //Tema:             estructura general de seudocódigo
10 //*************************************************************
11 Algoritmo distancia_coordenadas
12
13     // variables
14     definir N1, N2, E1, E2, DH como real
15
16     // entradas
17     Escribir "ingrese coordenada N1: "
18     Leer N1
19
20     Escribir "ingrese coordenada E1: "
21     Leer E1
22
23     Escribir "ingrese coordenada N2: "
24     Leer N2
25
26     Escribir "ingrese coordenada E2: "
27     Leer E2
28
29     // procesos
30     DH <- RC((N2-N1)^2+(E2-E1)^2)
31
32     // salidas
33     Escribir "La distancia horizontal entre los puntos es: ", DH
34
35 FinAlgoritmo
```

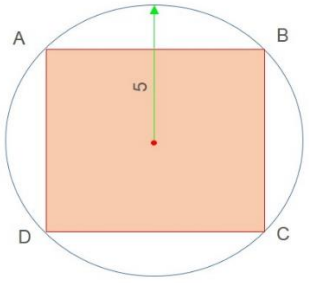
Como ya habrás podido notar, un programa de computador, tiene tres partes fundamentales para su funcionamiento: **entradas**, que son todos los datos necesarios para la implementación de los **procesos**, que son las operaciones para transformar los datos, y **salidas** que son el resultado de los datos procesados, ofrecido por el sistema. Todo esto encadenado mediante procedimientos lógicos almacenados en un único archivo.

Se considera una buena práctica de programación el uso de cometarios para documentar el programa, una serie de estos comentarios se suelen colocar en el encabezado como un bloque descriptivo, usando doble barra inclinada // por cada línea que se desee comentar, allí se indican detalles asociados con la fecha de creación, programador, nombre del archivo, el lenguaje en que se ha desarrollado y una breve descripción de su objetivo. Se sugiere que estos comentarios se encuentren también a lo largo del código fuente de manera legible y precisa.

Una palabra reservada es una unidad funcional más pequeña de una declaración del programa y que tiene un significado especial predefinido. Estas palabras no pueden ser usadas como funciones o variables dentro del código fuente.
Todos los lenguajes tienen su propio conjunto reservado de estas palabras, que aparecen en color azul: **`Algoritmo`**, **`Escribir`**, **`Leer`**, **`Mientas`**, **`Hacer`**, **`FinMientas`**…

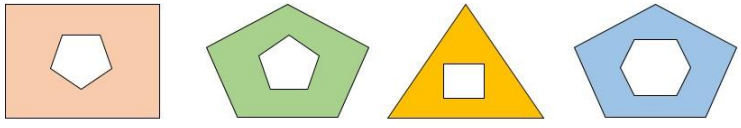
1.6 Ejercicios propuestos

1) Si el segmento AB=8, y el radio de la circunferencia es 5. ¿Cuál es el área en unidades cuadradas del polígono circunscrito ABCD?



(a) 64 (b) 56 (c) 48 (d) 40

2) ¿Cuál de las siguientes figuras no guarda relación con las otras?



(1) (2) (3) (4)

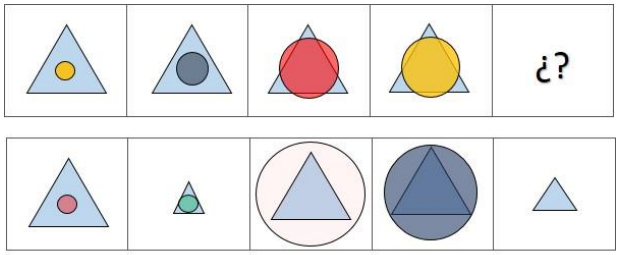
3) Utilizando la técnica de descripción narrada, escriba un algoritmo que permita poner un teodolito en estación.

4) Dibuje un diagrama de flujo que permita calcular la distancia horizontal teniendo como datos de entrada HS, HM, lectura del círculo vertical, y K.

5) Desarrolle un algoritmo en seudocódigo que permita calcular las coordenadas de un punto teniendo como datos de entrada la distancia inclinada y la lectura de los círculos horizontal y vertical.

6) Dado un ángulo en grados, minutos y segundos, dibuje un diagrama de flujo que permita convertir dicho ángulo a radianes.

7) ¿Cuál es la siguiente figura en la serie?



(a) (b) (c) (d) (e)

8) Desarrolle un algoritmo en seudocódigo que calcule el punto de cero, teniendo como datos de entrada las alturas de corte, lleno, y la distancia entre dichas alturas.

9) Elabore un algoritmo en seudocódigo que calcule el área de un triángulo teniendo como datos de entrada sus tres lados.

10) Haga un programa en seudocódigo que calcule la cartera de localización con azimutes y las distancias horizontales a partir de la siguiente información:

```
Pto estación:     1000      N       981266.850          Delta 1
                            E      1152306.420          1205.00

Pto orientación:  2000      N       981269.650          Delta 2
                            E      1152315.520          1204.00
```

Pto	N	E
1	981273.350	1152295.162
2	981276.350	1152289.966
3	981307.372	1152361.263
4	981273.350	1152289.966

Trabajo con datos

Piensa que estás de visita en el médico porque has tenido quebrantos de salud. El médico indagará a cerca de los síntomas que has presentado, luego te hará una auscultación básica, y te preguntará si padeces de alguna alergia. Finalmente, te formulará unos medicamentos o algunos exámenes especializados, según corresponda.

En el mundo de la programación, y haciendo una analogía con el párrafo anterior, considera tu cuerpo como los datos, la enfermedad como el tipo de dato, los medicamentos como operadores, y por supuesto, tú serías el programador, es decir, el médico. Ahora, a un nivel muy básico, necesitas saber con qué tipo de datos estás trabajando. Sin eso, no sería fácil y, en algunos casos, casi imposible abordar la solución.

Ahora, si sustituyes el concepto de médico por el de compilador, la importancia del tipo de datos es aún mayor, ya que el compilador debe detectar si el programa desarrollado es compatible con los datos o no. Para ser más precisos, el compilador verificará si las operaciones que se intentan realizar cumplen con la estructura de los datos.

2.1 ¿Qué son los datos?

Un dato topográfico es el resultado de las observaciones básicas que procesadas de forma adecuada representan las formas y dimensiones de cualquier objeto. Bajo un concepto más generalizado, un dato es información no elaborada, que carece de sentido en sí misma, un dato no dice nada del porqué de las cosas, y por si mismo tiene poca importancia. En el ámbito computacional los datos son representaciones que se traducen a una forma eficiente para su procesamiento. En relación a las computadoras, estos son convertidos formatos digitales binarios y almacenados en archivos.

2.1.1 ¿Cómo se almacenan?

Las computadoras representan datos incluidos en imágenes satelitales, fotografías aéreas, tecnología LiDAR, como valores binarios de un par de números: 0 y 1. La mínima unidad de dato se llama bit y representa un solo valor. Luego aparecen los bytes que agrupan 8 bits, usualmente el almacenamiento y la memoria se miden en megabytes y gigabytes.

Tabla 3.
Unidades de información computacional

unidad	abreviación	valor
bit		1 bit
byte	B	8 bits
kilobyte	KB	1024 bytes
megabyte	MB	1024 kilobytes
gigabyte	GB	1024 megabytes
terabyte	TB	1024 gigabytes
petabyte	PB	1024 terabytes
exabyte	EB	1024 Petabytes
zettabyte	ZB	1024 exabytes
yottabyte	YB	1024 zettabytes
brontobyte	BB	1024 yottabytes

La codificación internacional de caracteres ASCII definió que 1 byte es igual a 8 bits, las 256 combinaciones que permiten 8 bits desde 00000000 hasta 11111111 son suficientes para representar los caracteres de un idioma, números decimales, símbolos matemáticos y caracteres especiales.

2.1.2 Tipos de dato

A diferencia del ser humano, la computadora no detecta diferencia entre 'ABCD' y '0123'. En programación los datos se clasifican en función de la variable que puedan contener. El tipo determina el valor que puede tener un objeto y las operaciones que sobre él se pueden realizar. De acuerdo con el tipo se reserva un determinado espacio de memoria.

Cuando los programadores crean aplicaciones informáticas, los tipos de datos deben ser referenciados y utilizados correctamente, para garantizar el resultado adecuado y un programa sin errores. Los tipos de datos que trabajaremos en este libro son los más usuales en entornos de lógica de programación: numéricos (**reales** o **enteros**), alfanuméricos (**cadenas** y **caracteres**) y **lógicos**.

Tabla 4.
Rangos y tamaños de acuerdo al tamaño de dato.

tipo	rango	tamaño
real	3,4*10^-38 ... 3,4*10^38	4 bytes
entero	-32.768 ... 32.767	4 bytes
caracter	0...127 (ASCII)	1 byte por carácter
cadena	-128 ... 127	1 byte
lógico	0 - 1	2 bytes

Los datos numéricos incluyen números con signo y sin signo, están creados para las operaciones aritméticas y se dividen en **reales** y **enteros**. Los **reales** tienen fracción decimal y los **enteros** no. Un número **entero** es 89007517 y 10,257 es un número **real**.

Los datos alfanuméricos abarcan todas letras y números de un idioma específico, en nuestro caso los caracteres de la A a la Z y los números arábigos del 0 al 9. Para propósitos informáticos se incluyen letras mayúsculas y minúsculas, signos de puntuación y símbolos (*, &, %, @, $ y otros). Los datos alfanuméricos se dividen en **caracteres** y **cadenas**.

Un **carácter** es la unidad de visualización equivalente a una letra, número o símbolo, el sistema universal para caracteres está definido por el código ASCII, se definen en comillas sencillas. Por ejemplo: `'J'`, `'@'`, `'$'`, `'8'`. Cuando un número es utilizado como **carácter** no sirve para ejecutar operaciones aritméticas.

La definición de una **cadena** es una secuencia de **caracteres**, estas se escriben entre comillas dobles. Las **cadenas** son muy útiles cuando se comunica información al usuario del programa. Por ejemplo: `"ingrese el valor del azimut: "`.

Los datos **lógicos** contienen valores de uno de dos estados como **verdadero**/**falso**, 1/0, si/no. Se usan para representar alternativas a determinadas condiciones, por ejemplo, en la transferencia de datos de una estación total al computador, el estado inicial instrumento deberá ser ¿encendido o apagado?

2.1.3 Identificadores, variables y constantes

Todos los datos que procesa el computador, deben ser almacenados en celdas de memoria para su utilización, estas celdas tienen un nombre que permite su identificación. Cada lenguaje de programación tiene su propio conjunto de reglas para crear identificadores, pero, en general se mantienen los siguientes:

El primer carácter deberá ser una letra minúscula, los demás podrán ser letras, dígitos o el símbolo _, se recomienda que el nombre sea asociado con la naturaleza del elemento nombrado, por ejemplo, el identificador para almacenar un ángulo horario podría ser **angulo_horario**

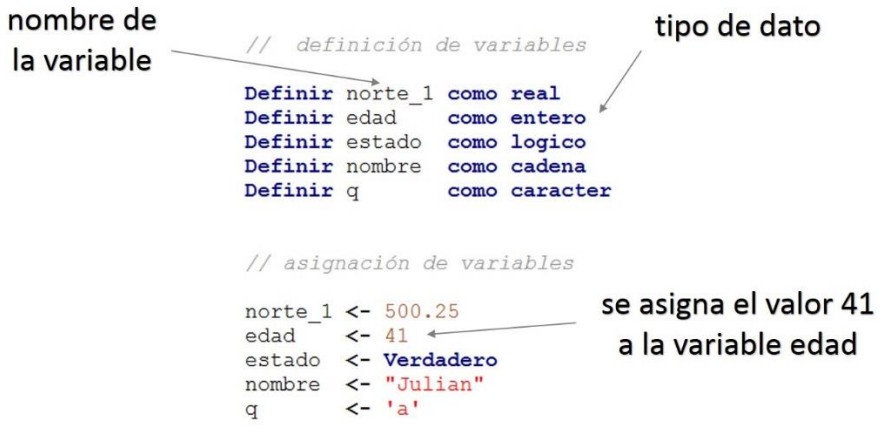
Las variables son objetos cuyo contenido puede cambiar en el tiempo, poseen una ubicación en la memoria RAM y se puede usar una y otra vez con distintos valores. Se pueden escribir instrucciones de programa una vez y usarlas en cientos de cálculos por separado. Por el contrario, las constantes son valores que no cambian en el tiempo de ejecución del programa, por ejemplo, el número e de Euler: 2,71828 o la velocidad de la luz en el vacío: 299'792.458 m/s.

La definición o declaración de variables se hará teniendo en cuenta el nombre de la variable, y su tipo de dato. La asignación —*que es el valor que van a contener*— tanto de variables como de constantes tiene la siguiente estructura:

variable <- expresión

De lado izquierdo se escribirá el nombre de la variable y de lado derecho el valor con el cual será cargada. El símbolo de asignación es una flecha con punta hacia la izquierda construida con los símbolos **<-**

Por ejemplo:



2.1.4 Operadores

Son símbolos que le indican al compilador que realice manipulaciones aritméticas, relacionales o lógicas específicas con los datos.

Tabla 5.
Operadores aritméticos

símbolo	operación	uso	explicación
^	potencia	x^y	la base x se eleva a la potencia y
*	multiplicación	x*y	multiplica los valores a cada lado del operados
/	división	x/y	divide el operando de la izquierda entre el de la derecha
+	suma	x+y	adiciona los valores a cada lado del operador
-	resta	x-y	resta el operando de la derecha del de la izquierda
% ó mod	módulo	x%y	divide el operando de la izquierda por el de la derecha y retorna el residuo

Una propiedad fundamental de cualquier lenguaje de programación es la de considerar alternativas, para proceder de una determinada forma si se cumplen o no las condiciones fijadas. Los operadores relacionales se utilizan cuando es necesaria una comparación entre operandos, luego de evaluar la relación retornan un valor **`Verdadero`** o **`Falso`**, según sea el caso.

Tabla 6.
Operadores relacionales

símbolo	operación	uso	explicación
==	Igual	x==y	comprueba si los valores de dos operandos son iguales o no, si es así, la condición es **verdadera**.
<>	diferente	x<>y	comprueba si los valores de dos operandos son iguales o no, si no lo son, entonces la condición se vuelve **verdadera**.
>	mayor que	x>y	comprueba si el valor del operando izquierdo es mayor que el derecho, si es así, entonces la condición se vuelve **verdadera**
<	menor que	x<y	comprueba si el valor del operando izquierdo es menor que el valor del derecho, si es así, la condición se vuelve **verdadera**.
>=	mayor o igual	x>=y	comprueba si el valor del operando izquierdo es mayor o igual que el valor del operando derecho, si es así, entonces la condición se vuelve **verdadera**.
<=	menor o igual	x<=y	comprueba si el valor del operando de la izquierda es menor o igual que el valor del operando de la derecha, si es así, la condición es **verdadera**.

Los operadores lógicos, son binarios, permiten combinar los resultados de los operadores relacionales comprobando que se cumplan las condiciones necesarias. Los operadores son de conjunción (&), disyunción (|), y negación (~).

Tabla 7.
Operadores lógicos

símbolo	operación	uso	explicación
& ó Y	conjunción	x&y	retorna **verdadero** si tanto x como y son **verdaderos**, de lo contrario, retorna **falso**.
| ó O	disyunción	x|y	retorna **verdadero** si al menos una de las dos condiciones (x ó y) es **verdadera**, de lo contrario, retorna **falso**
~ ó NO	negación	x~y	retorna **verdadero** si x es **falso**, o retorna **falso** si x es **verdadero**

Los operadores lógicos utilizan las tablas de la verdad para demostrar la función lógica en todas sus combinaciones, esta se compone por una o más variables (generalmente dos) y los operadores lógicos con los que se quiere combinar.

Tabla 8.
Tabla de verdad de los operadores lógicos

x	y	~x	~y	x|y	x&y
verdadero	verdadero	falso	falso	verdadero	verdadero
verdadero	falso	falso	verdadero	verdadero	falso
falso	verdadero	verdadero	falso	verdadero	falso
falso	falso	verdadero	verdadero	falso	falso

2.1.5 Jerarquía de los operadores

En los entornos de programación cuando una expresión involucra múltiples operaciones, instruye al compilador acerca del orden en que se va a ejecutar dicha expresión.

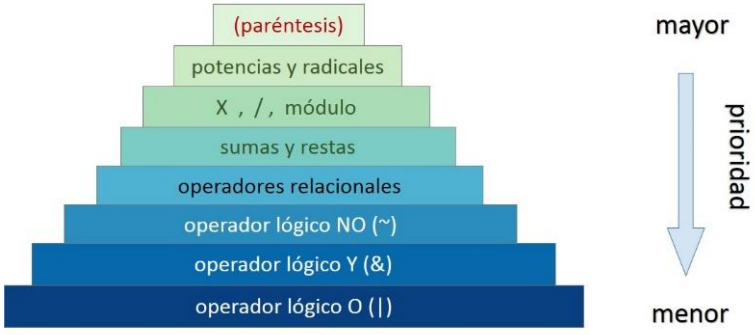
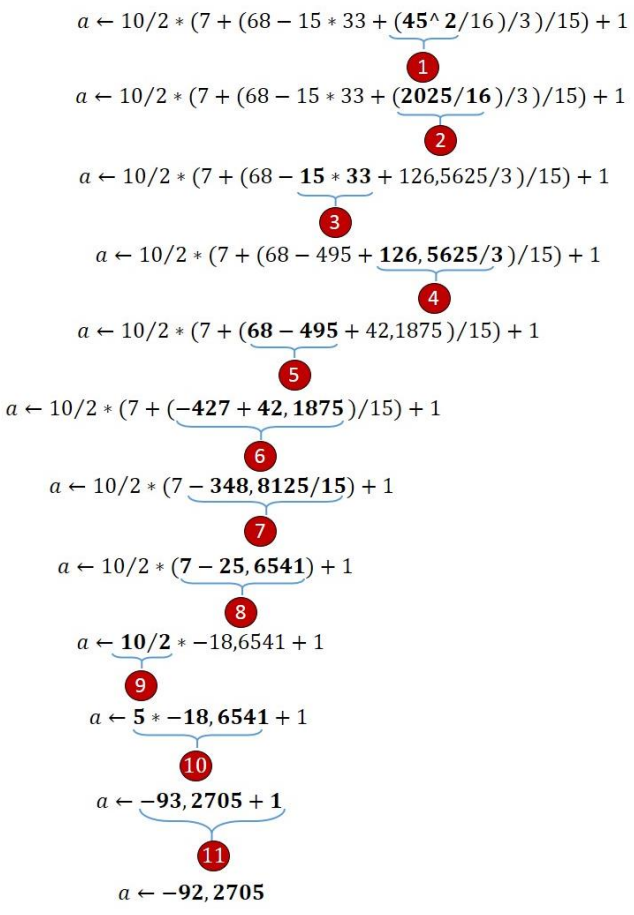


Figura 21. Precedencia de operadores

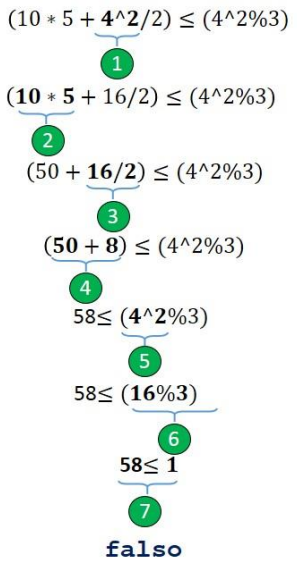
Al valorar expresiones con cualquier tipo de operador, se debe priorizar su superioridad, hay que indicar que el operador paréntesis () es de tipo asociativo y siempre tiene la mayor jerarquía en cualquier entorno de programación. Cuando una expresión —*aritmética, relacional o lógica*— contiene sub-expresiones entre paréntesis, estas se evalúan primero respetando la jerarquía de sus operadores para cada sub-expresión. En el caso en que las sub-expresiones se encuentren anidadas por paréntesis se evalúan primero aquellas de último nivel de anidamiento, es decir, aquellas que estén más internas. Cuando se presentan dos operadores con el mismo nivel jerárquico se prioriza por el que se encuentre más a la izquierda.

Veamos el procedimiento jerárquico paso a paso, usando operadores aritméticos, para ello se desea asignar a la variable a, el resultado de las siguientes operaciones:

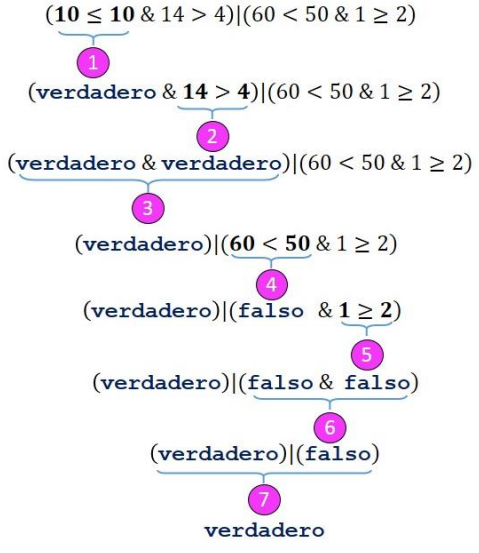
$$a \leftarrow 10/2 * (7 + (68 - 15 * 33 + (45\text{^}2/16)/3)/15) + 1$$



Ahora, desarrollemos un caso de operadores relacionales asociado con artiméticos a través de la siguiente función: $(10 * 5 + 4\text{^}2/2) \leq (4\text{^}2\%3)$



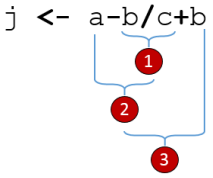
Finalmente, y teniendo como referencia la Tabla 9 veamos el procedimiento para hallar el valor de la siguiente operación lógica $(10 \leq 10 \& 14 > 4)|(60 < 50 \& 1 \geq 2)$



2.1.6 Equivalencia de expresiones algebraicas

La representación convencional de expresiones algebraicas no es directamente transmisible a un entorno de programación, estas deberán ser escritas en una sola línea, teniendo en cuenta la jerarquía de los operadores, el operador asociativo (), y

las palabras claves de cada lenguaje, por ejemplo la expresión algebraica $a = \sqrt{b}$ se escribirá así: `a <- RAIZ(b)`. Otras expresiones como $j = a - \frac{b}{c} + b$, tendrán su equivalente lineal así: `j <- a-b/c+b`. Nótese el orden de precedencia en función de los operadores.



Para expresar una relación que soluciona una ecuación de segundo grado $ax^2 + bx + c$, es fundamental utilizar la función cuadrática.

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

Si el tipo de dato es real, la ecuación podrá tener dos soluciones si el discriminante (d) —*lo que se encuentra al interior de la raíz cuadrada*—es mayor que cero, o una solución si d es igual a cero, o cero soluciones si d es menor que cero.
Su equivalencia algorítmica será la siguiente:

```
x1 <- (-b+RAIZ(b^2-4*a*c))/(2*a)
x2 <- (-b-RAIZ(b^2-4*a*c))/(2*a)
```

2.1.7 Prueba de escritorio

Es una simulación del comportamiento del algoritmo diseñado, permite ver lo que haría la computadora ejecutando cada línea de programación. Este trabajo se realiza en base a una tabla cuyos encabezados son las variables que se usan en el algoritmo, debajo de ellas se van escribiendo los valores que toman las variables paso a paso y siguiendo el flujo indicado por el algoritmo, hasta llegar al final. Es fundamental para detectar errores u omisiones, o para mejorar el algoritmo.

```
1  //*************************************************************
2  //Cod:              T-002
3  //Nombre:           proyecciones
4  //Fecha:            enero 2018
5  //Programador:      JGB-GJC-JJDA
6  //Lenguaje:         seudocodigo (PSeInt)
7  //Descripcion:      calula proyecciones meridiana (PM)
8  //                  y paralela (PP) de un vector, que tiene
9  //                  como entradas el azimut (GGMMSS) y la
10 //                  distancia horizontal
11 //Tema:             prueba de escritorio
12 //*************************************************************
13 Algoritmo proyecciones
14
15     //variables
16     definir dh, azimut, azimut_rad, pm, pp, g, m, s Como Real
17
18     //entradas
19     Escribir "Ingrese la distancias horizontal: "
20     Leer dh
21
22     Escribir " Ingrese grados: "
23     Leer g
24
25     Escribir " Ingrese minutos: "
26     Leer m
27
28     Escribir " Ingrese segundos: "
29     Leer s
30
31     //procesos
32     azimut <- g + m/60 + s/3600
33     azimut_rad <- azimut*PI/180
34
35     pm <- dh*cos(azimut_rad)
36     pp <- dh*sen(azimut_rad)
37
38     //salidas
39     Escribir "la proyeccion meridiana es: ", pm
40     Escribir "la proyeccion paralela  es: ", pp
41
42 FinAlgoritmo
```

Utilizando el algoritmo anterior, supongamos, que se quieren calcular las proyecciones meridiana y paralela a una distancia horizontal de 100 m y un azimut de 45°20’10”. Veamos la ejecución a través de sus líneas de programación:

Tabla 9.
Prueba de escritorio algoritmo proyecciones

línea	dh	g	m	s	azimut	azimut_rad	pm	pp
20	100							
23		45						

línea	dh	g	m	s	azimut	azimut_rad	pm	pp
26			20					
29				10				
32					45.3361111111			
33						0.7912644089		
35							70.294657623	
36								71.124265265

Los entornos de programación tienen sistema decimal, pero en términos trigonométricos, estos deberán convertirse a sexagesimal para poder trabajar con magnitudes angulares. Por ese motivo fue necesario hacer la asignación de la línea 32: `azimut <- g + m/60 + s/3600` ahora, para utilizar las funciones trigonométricas seno y coseno es necesario convertir la magnitud angular sexagesimal en radianes: `azimut_rad <- azimut*PI/180`.

Siguiendo la ejecución del programa línea a línea se obtendrán los resultados expresados en la Tabla 9. Es importante confrontar estos resultados con los obtenidos en una calculadora para detectar posibles inconsistencias y corregirlas en la línea que corresponda.

2.2 Ejercicios propuestos

1) ¿Cuántos byts hay en 1,8 terabytes? Si una imagen Landsat5 tiene un tamaño de 357 MB, ¿ a cuántos KB equivale?

2) Realice las siguientes asignaciones teniendo en cuenta la jerarquía de los operadores:

```
j <- -8*(5-(-2))-25/(6+(-15))-11*(9+(-7))+36/((-2)+(-10))
```

```
g <- (2*3-2)^2-4*8/(2^3/4)
```

```
b <- 2+2^2*2+2%2
```

3) determine la salida de las siguientes operaciones lógicas, sabiendo que,

```
x <- 10; b <- 22; z <- 4
```

```
((x=b)|(x<z))&((x<z)|(x>=b))
```

```
~(x=z)&(z>b)
```

```
((x>=b)|(x<=z))&((x>=z)&(c>=b))
```

4) ¿cuál es la salida de la siguiente expresión relacional?

Si `a <- 4` evalúe la expresión para obtener el valor de `b`

```
b <- (a-2)%2>=a+10/2-4*a
```

5) Encuentre el resultado del operador relacional luego de procesar la siguiente información:

```
a <- 2 ; b <- -2 ; c <- 4

(a*a+5+b^21/2)<=(b^(12%2)-4+c%7)
```

6) encuentre la equivalencia algorítmica de las siguientes expresiones algebraicas:

$a)\ binomio = a^2 + b^2 + ab + ab$

$b)\ sol = \frac{3w^2+x^2}{y^4+z^4}$

$c)\ agua = \sqrt[3]{\frac{3w^2+x^2}{y^4+z^4}} + \sqrt{a^2+b^2} - \frac{ab^{j^{g^b}}}{\frac{\frac{a}{b}}{y^4}} - 1$

$d)\ mat = \frac{(a-b)\cdot(b-c)}{b-c} \cdot \frac{(a+b)\cdot(b+c)}{b+c} \cdot \frac{1}{a^2-b^2}$

7) La siguiente expresión algorítmica ¿a cuál expresión algebráica corresponde?

```
sol <- w+x/y+z
```

$a)\ sol = \frac{w+x}{y+z}$

$b)\ sol = w + \frac{x}{y} + z$

$c)\ sol = \frac{w+x}{y} + z$

$d)\ sol = w + \frac{x}{y+z}$

8) realice una prueba de escritorio al siguiente algoritmo de asignaciones e indique los valores finales de cada una de las variables definidas.

```
Algoritmo asignaciones

    definir TOPO            Como Entero
    definir I, J, DIR, INV  Como Real
    definir CAR             Como Caracter
    definir BAND            Como Logico


    I <- 0
    I <- I+1
    TOPO <- 0
    J <- trunc (5^2/3)
    CAR <- 'a'
    TOPO <- trunc(J/I)
    DIR <- TOPO/3
    BAND <- (8>5) y (15<2^3)
    INV <- TOPO*5/J^2
    I <- I*3
    DIR <- DIR/5
    BAND <- BAND o (I=J)
    I <- DIR
    CAR <- 'J'

FinAlgoritmo
```

9) detecte los errores en el siguiente algoritmo, e indique cómo solucionarlos.

```
Algoritmo proyecciones

    //variables
    definir dh, azimut_rad, g, m, s Como Real
    definir azimut como cadena
    definir pm, pp como caracter
    //entradas
    Escribir "Ingrese la distancias horizontal: "
    Leer dh

    Escribir " Ingrese grados: "
    Leer g

    Escribir " Ingrese minutos: "
    Leer m

    Escribir " Ingrese segundos: "
    Leer s

    //procesos
+   azimut <- g + m/60 + s/3600
+   azimut_rad <- azimut*PI/180

+   pm <- dh*cos(azimut_rad)
+   pp <- dh*sen(azimut_rad)

    //salidas
    Escribir "la proyeccion meridiana es: ", pm
    Escribir "la proyeccion paralela  es: ", pp

FinAlgoritmo
```

Estructuras algorítmicas de control

Es usual en programación hacer la analogía de una receta de cocina con la definición de algoritmo, la lista de ingredientes es perfectamente comparable a las variables, las ollas y recipientes serán los tipos de dato. Ahora deberemos seguir las instrucciones para obtener nuestro platillo. En términos de programación, esas instrucciones se pueden presentar en alguno de estos tres modelos, o sus posibles combinaciones: secuencial, condicional, e iterativo. Por ejemplo: secuencial: primero mezcle los ingredientes líquidos, y luego agregue los secos; condicional: Si los tomates están frescos entonces cocínelos a fuego lento, sino, fríalos en mantequilla. Iterativo: bata las claras de huevo hasta que formen punto de nieve. Ahora verás en este capítulo lo fácil y ventajoso que es trabajar con estas estas estructuras.

3.1 Estructura secuencial

Bajo este modelo se ejecutan en orden lineal una serie de acciones o tareas. Una secuencia puede contener cualquier cantidad de tareas, una vez que se inicia la ejecución con la tarea A, se debe continuar con la B, y luego la C, hasta que la secuencia termine, no existe posibilidad de saltar alguna acción. Estos programas son fáciles de escribir, pero carecen de flexibilidad.

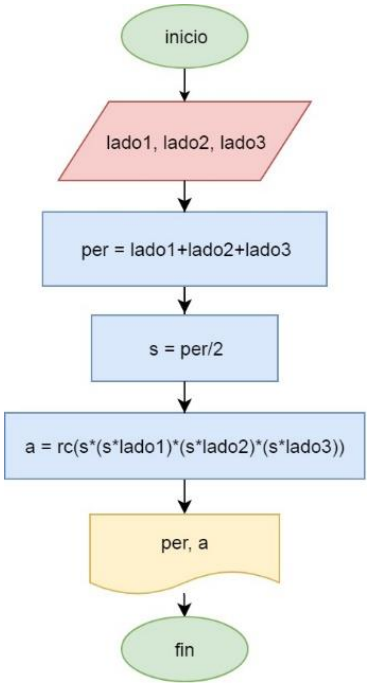


Figura 22. Triángulo Herón

```
//*****************************************************************
//Cod:              T-003
//Nombre:           area_triangulo_escaleno
//Fecha:            enero 2018
//Programador:      JGB-GJC-JJDA
//Lenguaje:         seudocodigo (PSeInt)
//Descripcion:      calcula el área de un triángulo escaleno,
//                  teniendo como datos de entrada sus lados.
//Tema:             programación secuencial
//*****************************************************************
Algoritmo area_traingulo_escaleno

    // variables
    Definir lado1, lado2, lado3, per, s, a Como Real

    // entradas
    Escribir "ingrese el lado 1: "
    Leer lado1
```

```
20      Escribir "ingrese el lado 2: "
21      Leer lado2
22
23      Escribir "ingrese el lado 3: "
24      Leer lado3
25
26      // procesos
27      per <- lado1 + lado2 + lado3  //calculo perimetro
28      s <- per / 2 // semiperimetro
29      a <- rc(s*(s-lado1)*(s-lado2)*(s-lado3))  // teorema Heron
30
31      // salidas
32      Escribir "el perímetro es: ", per, " unidades lineales"
33      Escribir " el área es: ", a, " unidades cuadradas"
34
35  FinAlgoritmo
```

En el ejemplo de la Figura 21. Precedencia de operadores, se plantea el modelo de solución para el cálculo del área de un triángulo escaleno (aquél cuyos lados son distintos), para ello se piden como entradas los tres lados del triángulo, luego su primera tarea de procesamiento es el cálculo del perímetro, su segunda tarea el calcular el semi-perímetro, su tercera tarea es el cálculo del área.

3.2 Estructuras condicionales

Cada día al despertar tomamos decisiones, para ello, debemos aplicar condiciones. Si quiero tomar mi desayuno antes o después de la ducha, exige que tome una decisión: ¿antes o después? Si quiero llegar a mi trabajo caminando, en transporte público o en mi carro, exige otra nueva decisión. Las decisiones son fundamentales en nuestra cotidianidad, y también en programación. Habrá muchas situaciones en las que haya definirse de entre dos o más opciones, y se tendrá que seleccionar una alternativa de acuerdo con las condiciones establecidas. Ahora bien, ¿cómo escribir en líneas de código fuente para simular estas situaciones? Pues, existen tres modelos condicionales para ello, **`Si Entonces`**, **`Si Entonces SiNo`**, **`Segun`**.

3.2.1 Condicional simple

Se identifica porque está compuesta solo de una condición, si la condición es **`Verdadera`** entonces ejecuta la acción (o acciones si son varias), en caso de que la condición sea **`Falsa`** entonces, no hará nada.

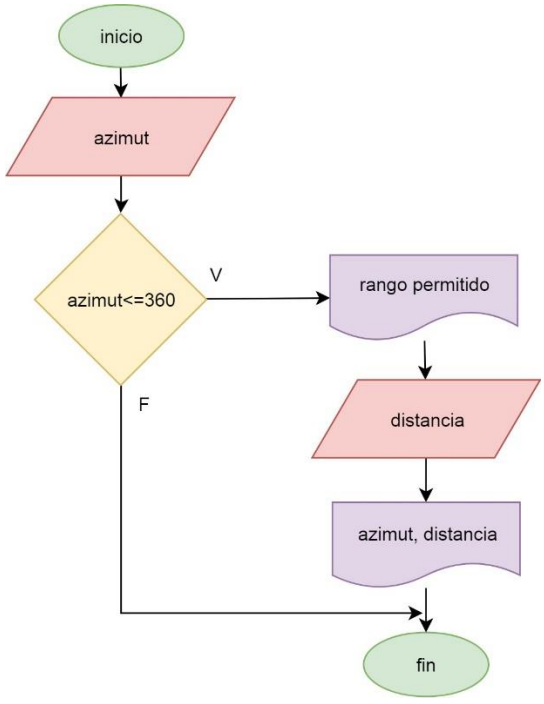


Figura 23. Condición simple para evaluar la pertinencia de un azimut.

Ahora, revisa cuidadosamente la manera de escribir este modelo condicional en seudocódigo:

```
//*********************************************************************
//Cod:              T-004
//Nombre:           decision_azimut
//Fecha:            enero 2018
//Programador:      JGB-GJC-JJDA
//Lenguaje:         seudocodigo (PSeInt)
//Descripcion:      evalua la condición de aceptación de rango, si
//                  si cumple, indica que es una magnitud permitida.
//Tema:             condicional simple
//*********************************************************************
Algoritmo decision_azimut

    Definir azimut, distancia Como Reales

    Escribir "ingrese la magnitud del azimut : "
    Leer azimut

    Si (azimut<=360) Entonces

        Escribir "rango permitido"
        Escribir "Ingrese la distancia: "
        Leer      distancia
        Escribir "El azimut ingresado fue: ", azimut
        Escribir "La distancia ingresada fue: ", distancia

    Fin Si

FinAlgoritmo
```

3.2.2 Condicional doble

Esta estructura evalúa la expresión de condición, el flujo de datos dependerá de este resultado, es decir, si la evaluación es **`Verdadera`** se ejecutarán las acciones que de ello dependan y en caso contrario, se ejecutarán las acciones que tengan el flujo por la salida **`Falsa`**. En el siguiente ejemplo se usará la estructura condicional doble para resolver si un número leído por un algoritmo es par o impar.

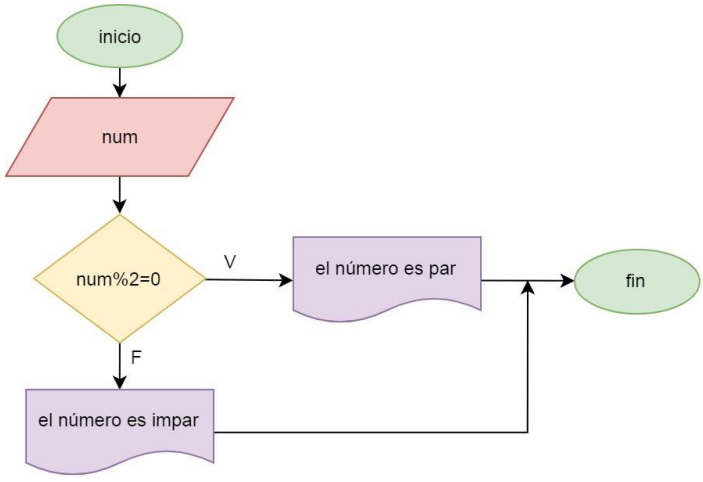
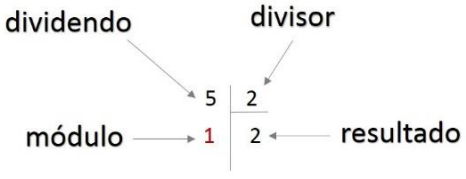


Figura 24. Condición doble para definición de par o impar

Para ello usaremos la función **`%`** (módulo). Como se mostró en el capítulo anterior, esta función **`%`**, retorna el residuo de la división, por lo tanto, si, `num <- 5`, entonces,



A diferencia del caso anterior, cuando el resultado de la condición evaluada es **`Falso`**, existe una acción a ejecutar, en este caso, imprimir un mensaje indicando que el número ingresado es impar. Ahora, revisa la estructura del mismo algoritmo, pero, escrito en seudocódigo:

```
//************************************************************
//Cod:              T-005
//Nombre:           par_impar
//Fecha:            enero 2018
//Programador:      JGB-GJC-JJDA
//Lenguaje:         seudocodigo (PSeInt)
//Descripcion:      determina si un número es par o impar
//                  usando la función módulo (%)
//Tema:             condicional doble
//************************************************************
Algoritmo par_impar

    Definir num Como entero

    Escribir "ingrese un número : "
    Leer num

    Si (num%2=0) Entonces
        Escribir "el número es par"
    SiNo
        Escribir "el número es impar"
    Fin Si

FinAlgoritmo
```

Nótese que, en la estructura condicional, aparece la palabra **SiNo** dando la posibilidad a ejecutar una tarea específica en caso de que el resultado de la condición sea **Falso**.

3.2.3 Condicional múltiple

No todas las decisiones en programación se solucionan con **Si** o **No**. Las excepciones ocurren todo el tiempo. Los lenguajes de programación proporcionan formas para hacer frente a dichas excepciones, lo que le permite elaborar código fuente que se ejecuta en base a múltiples posibilidades. La instrucción **Segun** es una decisión de múltiples vías que comprueba si una expresión coincide con un valor entero, que da la entrada a una serie de acciones específicas. Veamos:

```
//************************************************************
//Cod:              T-006
//Nombre:           rumbo
//Fecha:            enero 2018
//Programador:      JGB-GJC-JJDA
//Lenguaje:         seudocodigo (PSeInt)
//Descripcion:      calcula el azimut de una línea teniendo
//                  como entrada su rumbo.
//Tema:             condicional múltiple
//************************************************************
```

```
11  Algoritmo rumbos
12
13      Definir opc, rumbo Como enteros
14
15      Escribir "opción 1. Nor-Este"
16      Escribir "opción 2. Sur-Este"
17      Escribir "opción 3. Sur-Oeste"
18      Escribir "opción 4. Sur-Oeste"
19      Escribir "Elija una opción (1-4): "
20      Leer opc
21
22      Segun opc Hacer
23          1:
24              Escribir "ingrese el valor del rumbo NE: "
25              Leer rumbo
26              azimut <- rumbo
27              Escribir " el azimut es: ", azimut
28
29          2:
30              Escribir "ingrese el valor del rumbo SE: "
31              Leer rumbo
32              azimut <- 180-rumbo
33              Escribir " el azimut es: ", azimut
34
35          3:
36              Escribir "ingrese el valor del rumbo SW: "
37              Leer rumbo
38              azimut <- 180+rumbo
39              Escribir " el azimut es: ", azimut
40
41          4:  Escribir "ingrese el valor del rumbo NW: "
42              Leer rumbo
43              azimut <- 360-rumbo
44              Escribir " el azimut es: ", azimut
45
46
47          De Otro Modo:
48              Escribir "opción errada..."
49      Fin Segun
50
51  FinAlgoritmo
```

Como habrás notado, la primera parte del algoritmo produce un menú de opciones con cuatro posibilidades, deberás elegir una, para convertir un rumbo en azimut, con la selección de un número entero (1, 2, 3 o 4) el algoritmo ejecutará una secuencia de acciones diferente de las otras. Como bien lo sabes, el cálculo del azimut es diferente cada vez que se cambia de cuadrante. ¿pero, qué sucede cuándo el rumbo coincide con los ejes cardinales? Bueno, para ello te invitamos a que revises el siguiente tipo de estructura condicional.

3.2.4 Condicional anidado

En el desarrollo de la solución de problemas complejos, casi siempre luego de tomar una decisión y marcar el camino a seguir, es necesario tomar otra decisión que señala un nuevo flujo luego de evaluar la condición, y nuevamente se deberá tomar otra decisión. Este proceso puede repetirse numerosas veces; en este caso, se están aplicando estructuras anidadas o también llamadas en cascada. Estas estructuras no son más que la combinación de las estructuras ya vistas. Es ideal cuando se quiere calcular el azimut y distancia de una línea a partir de las coordenadas que definen sus puntos extremos. Analiza la lógica del siguiente diagrama de flujo:

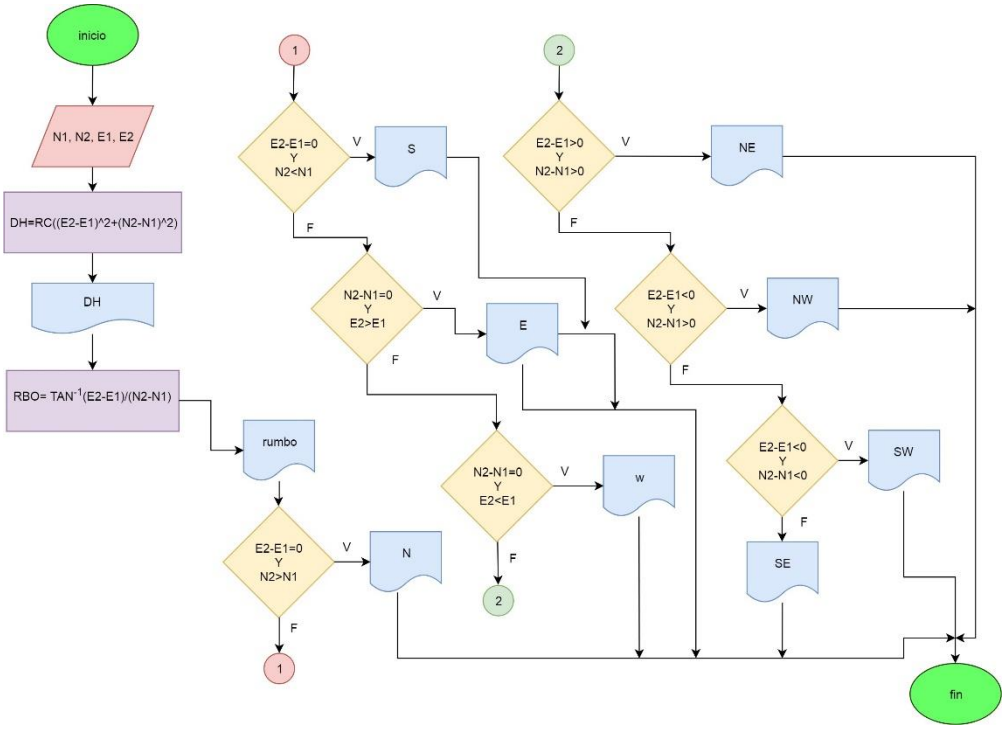


Figura 25. Cálculo de cuadrantes topográficos

Utilizando el planteamiento de la Figura 25. Cálculo de cuadrantes topográficos desarrollemos el algoritmo en seudocódigo, pero, teniendo en cuenta que el usuario desea ver el resultado en grados sexagesimales, es decir, grados, minutos y segundos.

```
//*****************************************************************************************
//Cod:              T-007
//Nombre:           coordenadas_a_rumbo
//Fecha:            enero 2018
//Programador:      JGB-GJC-JJDA
//Lenguaje:         Seudocodigo (SPSeInt)
//Descripcion:      calcula el rumbo y la distancia de una línea
//                  a partir de las coordenadas de los extremos
//Tema:             desición anidada
//*****************************************************************************************
Algoritmo rumbos

    definir N1, N2, E1, E2, DH, rumbo, auxiliar, grados, minutos, seg como reales

    Escribir "ingrese N1: "
    Leer N1
    Escribir "ingrese E1: "
    Leer E1
    Escribir "ingrese N2: "
    Leer N2
    Escribir "ingrese E2: "
    Leer E2

    //cáloulo distancia
    DH <- rc((N2-N1)^2 +(E2-E1)^2)

    //cáloulo del rumbo
    rumbo <- atan((E2-E1)/(N2-N1))
    auxiliar <- abs((rumbo)*(180/PI)) // rumbo en radianes almacenado en la variable auxiliar

    Si (E2-E1=0 Y N2>N1) Entonces
        Escribir "el rumbo es NORTE"
        Escribir "la distancia horizontal es: ", DH
    SiNo
        Si (E2-E1=0 Y N2<N1) Entonces
            Escribir "el rumbo es SUR"
            Escribir "la distancia horizontal es: ", DH
        SiNo
            Si (N2-N1=0 Y E2>E1) Entonces
                Escribir "el rumbo es ESTE"
                Escribir "la distancia horizontal es: ", DH
            SiNo
                Si (N2-N1=0 Y E2<E1) Entonces
                    Escribir "el rumbo es OESTE"
                    Escribir "la distancia horizontal es: ", DH
                Fin Si
            Fin Si
        Fin Si
    Fin Si
```

```
50
51      //numerador de la ecuación
52      Si (N2-N1>0) Entonces
53      |   Escribir "el rumbo es: "
54      |   Escribir "NORTE"
55      SiNo
56      |   Escribir "el rumbo es: "
57      |   Escribir "SUR"
58      Fin Si
59
60      // Proceso que convierte la magnitud angular de radianes a grados sexagesimales
61      grados  <- TRUNC(auxiliar)
62      minutos <- TRUNC((auxiliar-grados)*60)
63      seg     <- (((auxiliar-grados)*60)-(TRUNC((auxiliar-grados)*60)))*60
64
65      //Estructura de desición que permite hacer el redondeo de los segundos
66      Si (seg-TRUNC(seg)>=0.50) Entonces
67      |   seg <- TRUNC(seg)+1
68      SiNo
69      |   seg <- TRUNC(seg)
70      Fin Si
71      Escribir grados,"°", minutos,"´", seg, "´´"
72
73      //denominador de la ecuación
74      Si (E2-E1>0) Entonces
75      |   Escribir "ESTE"
76      SiNo
77      |   Escribir "OESTE"                          .
78      Fin Si
79      Escribir "la distancia horizontal es: ", DH
80
81  FinAlgoritmo
```

Ahora, revisemos un nuevo ejemplo de anidamiento de decisiones, para hacer la conversión de un azimut en rumbo:

```
1  //*************************************************************************************************
2  //Cod:              T-008
3  //Nombre:           azimut_a_rumbo
4  //Fecha:            enero 2018
5  //Programador:      JGB-GJC-JJDA
6  //Lenguaje:         Seudocodigo (SPSeInt)
7  //Descripcion:      convierte un azimut en rumbo
8  //Tema:             desición anidada
9  //*************************************************************************************************
10 Algoritmo azimut_rumbo
11
12     Definir gg, mm, ss como enteros
13     Definir angulo, rbo2, gg2, mm2, ss2, rbo3, gg3, mm3, ss3, rbo4, gg4, mm4, ss4 como reales
14
15     Escribir "Ingrese grados: "
16     Leer gg
17
18     Escribir "Ingrese minutos: "
19     Leer mm
20
21     Escribir "Ingrese segundos: "
22     Leer ss
23
24     angulo<-gg+(mm/60)+(ss/3600)
25
```

```
    Si (angulo=0 O angulo=360) Entonces
        Escribir "El rumbo es NORTE"

    SiNo
        Si(angulo>0 Y angulo<90) Entonces
            Escribir "El rumbo es: ", gg "°" , mm "´" ,ss "´´" " " "Noreste"
        SiNo
            Si(angulo=90) Entonces
                Escribir "El rumbo es ESTE"
            SiNo
                Si (angulo>90 Y angulo<180) Entonces
                    rbo2 <- (180 - angulo)
                    gg2  <- TRUNC (rbo2)
                    mm2  <- TRUNC ((rbo2-gg2)*60)
                    ss2  <- (((rbo2-gg2)*60)-(TRUNC((rbo2-gg2)*60)))*60
                    Escribir "El rumbo es: ", gg2 "°" , mm2 "´" , ss2 "´´" " " "Sureste"
                SiNo
                    Si (angulo=180) Entonces
                        Escribir "El rumbo es SUR"
                    SiNo
                        Si (angulo>180 Y angulo<270) Entonces
                            rbo3 <- (angulo-180)
                            gg3  <- TRUNC (rbo3)
                            mm3  <- TRUNC((rbo3-gg3)*60)
                            ss3  <- (((rbo3-gg3)*60)-(TRUNC((rbo3-gg3)*60)))*60
                            Escribir "El rumbo es: ", gg3 "°", mm3 "´", ss3 "´´" " " "Suroeste"
                        SiNo
                            Si (angulo=270) Entonces
                                Escribir "El rumbo es OESTE"
                            SiNo
                                Si (angulo>270 Y angulo<360) Entonces
                                    rbo4 <- (360-angulo)
                                    gg4  <- TRUNC (rbo4)
                                    mm4  <- TRUNC ((rbo4-gra4)*60)
                                    ss4  <- (((rbo4-gg4)*60)-(TRUNC((rbo4-gg4)*60)))*60
                                    Escribir "El rumbo es: ", gg4 "°", mm4 "´", ss4 "´´" " " "Noroeste"
                                SiNo
                                    Escribir "El azimut se encuentra fuera de rango..."
                                Fin Si
                            Fin Si
                        Fin Si
                    Fin Si
                Fin Si
            Fin Si
        Fin Si
    Fin Si

FinAlgoritmo
```

3.3 Estructuras cíclicas

Los ciclos de programación tienen que ver con hacer lo mismo una y otra vez, esto es lo que se llama iteración en un lenguaje de programación. Estas estructuras también llamadas bucles, son una secuencia de instrucciones que se repite continuamente hasta que se alcanza una cierta condición. Existen tres modelos básicos para ello: **Mientras**, **Repetir** y **Para**. Todos ejecutan acciones repetitivas, pero tienen variaciones diferentes de acuerdo a su uso, si se logran dominar dichas variaciones, y reconocer cuándo son necesarias, entonces, la programación será mucho más fácil.

3.3.1 Ciclo Mientras

Se evalúa una condición booleana, si resulta verdadera, se ejecutan una serie de acciones y el flujo regresa para evaluar nuevamente la prueba de expresión. Esto se

repite cada vez hasta que la condición se evalúe como **Falso**, en cuyo caso, el bucle termina.

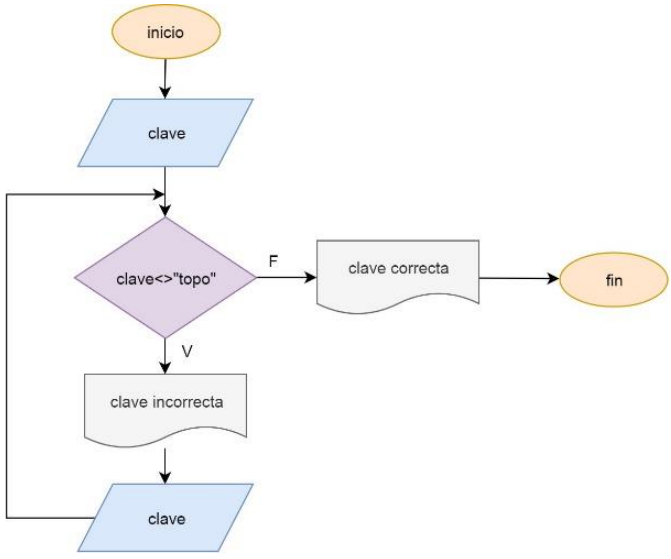


Figura 26. Acceso a través de clave

Ahora, revisa la manera de escribir el algoritmo de la clave con el ciclo **Mientras**, en seudocódigo:

```
1  //*********************************************************************
2  //Cod:              T-009
3  //Nombre:           password
4  //Fecha:            enero 2018
5  //Programador:      JGB-GJC-JJDA
6  //Lenguaje:         seudocodigo (PSeInt)
7  //Descripción:      solicita clave de ingreso, si no es correcta,
8  //                  preguntará de nuevo hasta que sea correcta
9  //Tema:             ciclo mientras
10 //*********************************************************************
11 Algoritmo password
12
13     Definir clave como cadena
14
15     Escribir "ingrese su clave de acceso"
16     Leer clave
17
18     Mientras (clave<>"topo") Hacer
19         Escribir "no es la clave correcta!"
20         Escribir "ingrese nuevamente la clave.·.·"
21         Leer     clave
22     Fin Mientras
23
24     Escribir "has ingresado correctamente"
25
26 FinAlgoritmo
```

Si revisas el código fuente que se encuentra con fondo amarillo, las acciones se ejecutan siempre que la clave sea diferente de topo. Si la condición se cumple se escribirán dos mensajes: `"no es la clave correcta!"` e `"ingrese nuevamente la clave…"`. En el momento que se ingrese la clave de manera correcta, es decir, cuando `clave="topo"` el ciclo termina y se ejecutan las acciones por el flujo **`Falso`**.

3.3.2 Ciclo Repetir

Esta estructura se usa para iterar sobre un bloque de código repetidas veces, la verificación de la condición se hace para salir del bucle, dentro de las acciones a repetir se hace uso de un contador que permite iterar el ciclo para cumplir con la condición y poder salir de él.

En el caso de una nivelación diferencial simple que tiene vistas intermedias, este ciclo funciona perfectamente, pues, con una altura instrumental fija se pude calcular la elevación de 'n' posiciones de la mira en vistas intermedias

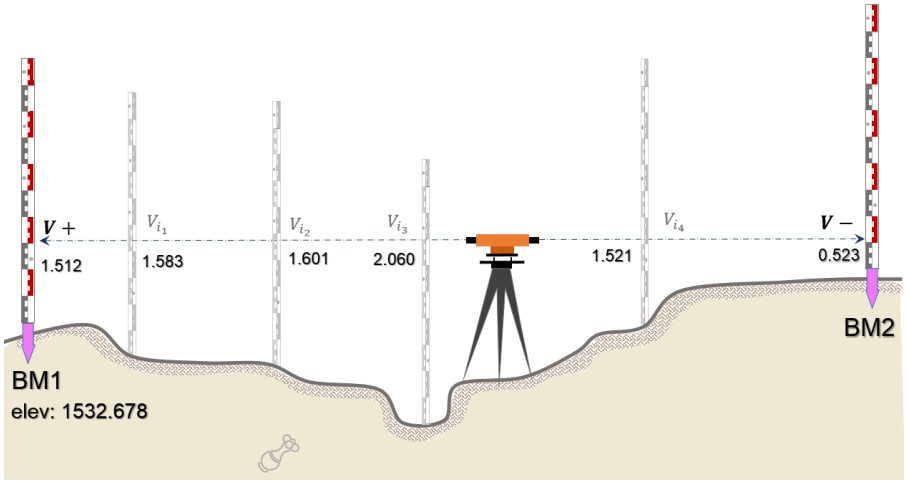


Figura 27. Nivelación diferencial con vistas intermedias

```
//*********************************************************************
//Cod:              T-010
//Nombre:           nivelacion
//Fecha:            enero 2018
//Programador:      JGB-GJC-JJDA
//Lenguaje:         seudocodigo (PSeInt)
//Descripción:      calcula desniveles intermedios entre dos BMs
//Tema:             ciclo repetir
//*********************************************************************
Algoritmo desniveles

    Definir elev_inicial, vmas, altura_instrumental como Reales
    Definir num, contador, vi, elev, vmenos, elev_final como Reales

    Escribir "ingrese la elevación del BM1: "
    Leer elev_inicial

    Escribir "ingrese V+ al BM1: "
    leer vmas

    altura_instrumental <- elev_inicial+vmas

    Escribir "ingrese número de vistas intermedias: "
    Leer num
        contador <- 0
        Repetir
            Escribir "ingrese lectura de la vista intermedia: "
            Leer vi
            elev <- altura_instrumental-vi
            Escribir "la elevación es: ", elev
            contador <- contador+1
        Hasta Que (contador=num)
    Escribir "ingrese V- al BM2: "
    Leer vmenos
    elev_final  <- altura_instrumental-vmenos
    Escribir "la elevación del BM2 es: ", elev_final

FinAlgoritmo
```

Si revisas, la estructura **Repetir** en el algoritmo de nivelación, podrás notar que la condición se evalúa al final del ciclo **Hasta Que** (`contador=num`), es decir, que la iteración se producirá por lo menos una vez, además, deberás notar que la variable `contador` se incrementa en una unidad cada vez que hay una iteración, esto con el fin de poder igualar la cantidad de iteraciones con la variable `num` que calculará tantas elevaciones como vistas intermedias hayan. Al cumplirse la condición el ciclo terminará y se cerrará la nivelación con la vista menos del BM2.Hagamos una prueba de escritorio del algoritmo con los datos de nivelación de la Figura 27. Nivelación diferencial con vistas intermedias

Tabla 10.
Prueba de escritorio algoritmo nivelación

línea	elev_ inicial	v mas	altura_ inst	num	contador	vi	elev	v menos	elev_ final
16	1532.678								
19		1.512							
21			1534.190						
24				4					
25					~~0~~				
28						~~1.583~~			
29							~~1532.607~~		
30					~~1~~				
28						~~1.601~~			
29							~~1532.589~~		
31					~~2~~				
28						~~2.060~~			
29							~~1532.130~~		
31					~~3~~				
28						1.521			
29							1532.669		
31					4				
34								0.523	
35									1533.667

Como habrás podido notar las dos estructuras de repetición que hemos estudiado tienen ciertas diferencias, en la Tabla 11. las encontrará resumidas.

Tabla 11.
Resumen estructuras mientras/repetir

Mientras	Repetir
La condición se prueba antes de que se ejecute el cuerpo de operaciones.	La condición se prueba después de que se ejecuta el cuerpo de operaciones.
Es posible que el cuerpo de operaciones nunca se ejecute.	Las acciones del ciclo siempre se ejecután al menos una vez.
El ciclo se termina cuando la evaluación de la condición es **`Falsa`**.	El ciclo se termina cuando la evaluación de la condición es **`Verdadera`**.

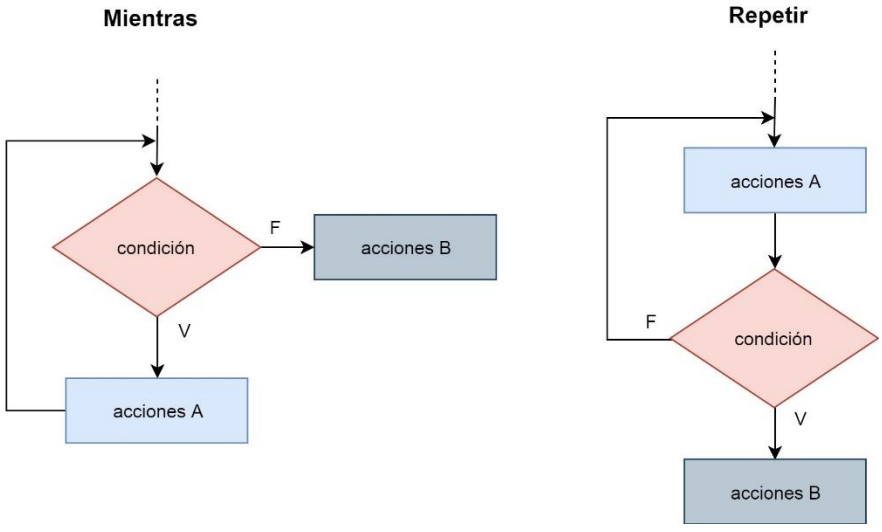


Figura 28. Diagramas de flujo ciclos Mientras / Repetir

3.3.3 Ciclo Para

Esta estructura se usa para repetir un bloque específico de código un número conocido de veces. Se deberá inicializar una variable que funcionará como contador. Luego se evalúa una expresión de condición antes de cada iteración, y finalmente, se define el incremento que se debe aplicar a la variable que actúa como contador.

```
1  //************************************************************************
2  //Cod:              T-011
3  //Nombre:           poligonal
4  //Fecha:            febrero 2018
5  //Programador:      JGB-GJC-JJDA
6  //Lenguaje:         seudocodigo (PSeInt)
7  //Descripción:      calcula parámetros de una poligonal cerrada
8  //Tema:             ciclo Para
9  //************************************************************************
10
11 Algoritmo poligonal_cerrada_gp
12
13     Definir n, norte, este, i, azimut, dh, pm, pp como Reales
14     Definir delta_pm, delta_pp, perimetro, error_cierre, gp como Reales
15
16     Escribir"ingrese el # de estaciones: "
17     Leer n
18
19     Escribir "ingrese coordenada NORTE DELTA 1: "
20     Leer norte
21
22     Escribir "ingrese coordenada ESTE DELTA 1: "
23     Leer este
24
```

```
25      // acumuladores del ciclo
26      delta_pm  <- 0
27      delta_pp  <- 0
28      perimetro <- 0
29
30      Para i<-1 Hasta n Con Paso 1 Hacer
31
32          Escribir "ingrese el azimut: "
33          Leer azimut
34
35          azimut <- azimut*PI/180
36
37          Escribir "ingrese la distancia: "
38          Leer dh
39
40          pm <- dh*cos(azimut)
41          pp <- dh*sen(azimut)
42
43          delta_pm  <- delta_pm+pm
44          delta_pp  <- delta_pp+pp
45          perimetro <- perimetro+dh
46
47      Fin Para
48
49      error_cierre <- rc((delta_pm^2)+(delta_pp^2))
50      gp           <- perimetro/error_cierre
51
52      Escribir "delta PM: ", delta_pm
53      Escribir "delta PP: ", delta_pp
54      Escribir "perimetro: ", perimetro
55      Escribir "error de cierre: ",error_cierre
56      Escribir "GP 1/: ", grado_precision
57
58  FinAlgoritmo
```

Si revisas el anterior algoritmo, podrás detectar que se utiliza el ciclo **Para** en la ejecución repetida (**n** veces) de pedir al usuario el azimut y la distancia horizontal de un vector a fin de calcular sus proyecciones. Un concepto muy usado en programación es de los acumuladores, que son variables que almacenan resultados generados al interior del ciclo, aquí hemos usado tres acumuladores para totalizar las sumatorias de las proyecciones y del perímetro. Estos acumuladores se inicializan cero antes de la ejecución del ciclo, para tener control de su punto de inicio. También notarás que el cálculo del error de cierre y grado de precisión se hacen por fuera del ciclo, esto debido que son valores únicos para el conjunto de observaciones.

Ahora, vamos a usar el ciclo **Para** en el cálculo de un desnivel entre dos BMs a partir de una nivelación diferencial compuesta, revisa la Figura 29. Nivelación diferencial compuestapara que recuerdes que, en este tipo de nivelaciones, la altura instrumental va cambiando cada vez que se hace un par de lecturas (una atrás y otra adelante). En la Figura 29. Nivelación diferencial compuesta solo hay dos puntos de

cambio, pero, debes tener en cuenta que el algoritmo tendrá que ser diseñado pensando que el usuario defina la cantidad cambios.

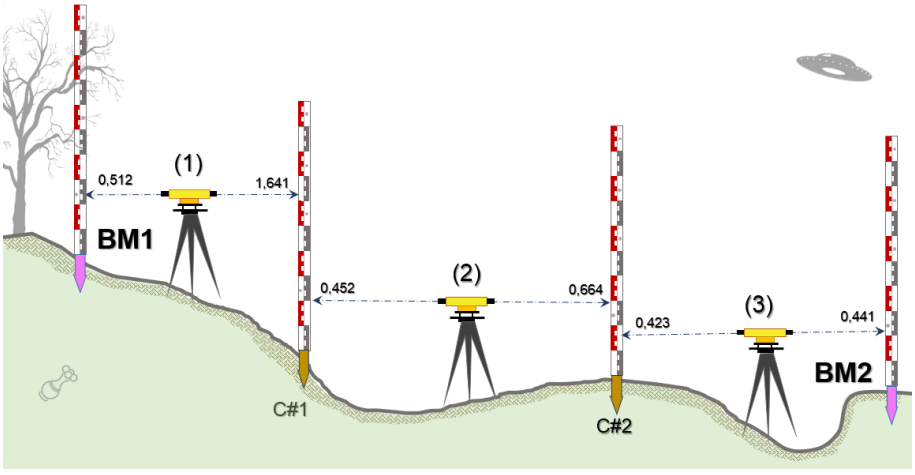


Figura 29. Nivelación diferencial compuesta

```
1  //*************************************************************************
2  //Cod:              T-012
3  //Nombre:           desniveles
4  //Fecha:            febrero 2018
5  //Programador:      JGB-GJC-JJDA
6  //Lenguaje:         seudocodigo (PSeInt)
7  //Descripción:      calcula desnivel entre BMs
8  //Tema:             ciclo Para
9  //*************************************************************************
```

```
10  Algoritmo desniveles
11
12      Definir n, vista_inicial, acu_v_mas, acu_v_menos, i, v_menos como Reales
13      Definir v_mas, vista_final, sum_v_mas, sum_v_menos, desnivel como Reales
14
15      Escribir "ingrese # cambios"
16      leer n
17
18      Escribir "ingrese V+ al BM#1"
19      leer vista_inicial
20
21      // acumuladores de vistas
22      acu_v_mas   <- 0
23      acu_v_menos <- 0
24
25      Para i=1 Hasta n Con Paso 1 Hacer
26
27          Escribir "ingrese V- al cambio ",i
28          leer v_menos
29          acu_v_menos <- acu_v_menos+v_menos
30
31          Escribir "ingrese V+ al cambio ",i
32          leer v_mas
33          acu_v_mas <- acu_v_mas+v_mas
34
35      Fin Para
36
37      Escribir "ingrese V- al BM#2"
38      leer vista_final
39
40      sum_v_mas   <- acu_v_mas+vista_inicial
41      sum_v_menos <- acu_v_menos+vista_final
42      desnivel    <- sum_v_mas-sum_v_menos
43      Escribir    " el desnivel entre los BMs es:", desnivel
44
45  FinAlgoritmo
```

Siguiendo el flujo de datos del anterior algoritmo, y las mediciones de campo de la Figura 29. Nivelación diferencial compuesta, en una prueba de escritorio obtendremos lo siguiente:

Tabla 12.
Prueba de escritorio nivelación diferencial compuesta

línea	n	vista_ inicial	acu_v mas	acu_v menos	i	v_ menos	v_ mas	vista_ final	sum_v_ mas	sum_v menos	desnivel
16	2										
19		0.512									
22			~~0~~								
23				~~0~~							
25					~~1~~						
28						~~1.641~~					
29				~~1.641~~							
32							~~0.452~~				
33			~~0.452~~								
25					~~2~~						
28						0.664					
29				2.305							
32							0.423				

línea	n	vista_ inicial	acu_v mas	acu_v menos	i	v_ menos	v_ mas	vista_ final	sum_v_ mas	sum_v menos	desnivel
33			0.875								
25					3						
38								0.441			
40									1.387		
41										2.746	
42											-1.359

3.3.4 Estructuras repetitivas anidadas

Un bucle anidado es un bucle dentro de otro. Cuando se tienen ciclos anidados, cada iteración del bucle "externo" contiene varias iteraciones del bucle "interno". Esto se repite hasta que el bucle externo termine. Por supuesto, una ruptura dentro del bucle interno o externo interrumpirá este proceso.

Si quisiéramos realizar la impresión del patrón de asteriscos de la Figura 29. Nivelación diferencial compuesta, debemos revisar el número total de líneas que presentan asteriscos, además es fundamental observar que la línea 1 tiene 1 asterisco, la línea 2 tiene 2 asteriscos, …, la línea 10 tiene 10 asteriscos. Por lo tanto, se detecta una estructura cíclica así.

```
*
**
***
****
*****
******
*******
********
*********
**********
```

Figura 30. Patrón de impresión de asteriscos

¿Cómo podríamos realizar la impresión de este patrón a través de código fuente de programación? Es necesario abordar el problema desde la ejecución de ciclos anidados. Se necesita de un ciclo externo que permita identificar la línea en la cual se imprimirán los asteriscos, y otro ciclo interno que controle la cantidad de asteriscos. Cuando se active el ciclo externo `Para i<-1 Hasta 10 Con Paso 1 Hacer` se iniciará en la línea 1 que está señalada por el auxiliar `i`, luego, el flujo ingresa al ciclo interno `Para j<-1 Hasta i Con Paso 1 Hacer` donde se hará la impresión del asterisco `i` veces, al cumplirse esto se hará el cambio de línea a través de la impresión de un

espacio vacío, y se incrementará el auxiliar `i` del ciclo externo, ingresando nuevamente al ciclo interno e imprimiendo dos asteriscos esta vez, y así hasta completar la línea 10. Analiza el párrafo anterior con el siguiente DF:

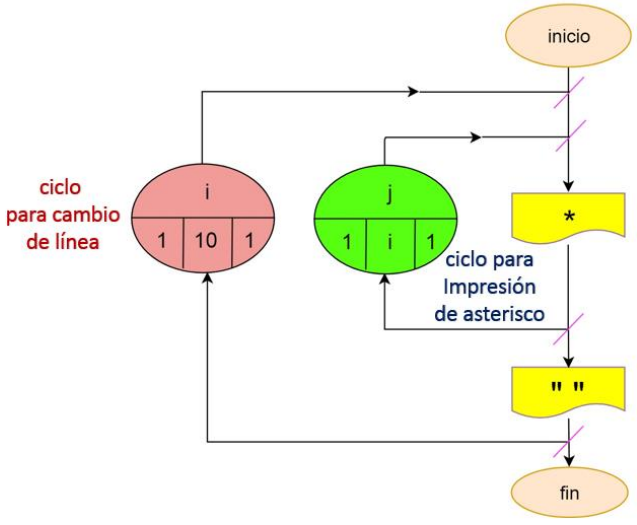


Figura 31. Ciclo anidado para impresión de un patrón de asteriscos

Ahora, revisa la forma de escribir el algoritmo en seudocódigo y realiza la prueba de escritorio para que compruebes el resultado.

```
1  //*****************************************************
2  //Cod:              T-013
3  //Nombre:           asteriscos
4  //Fecha:            febrero 2018
5  //Programador:      JGB-GJC-JJDA
6  //Lenguaje:         seudocodigo (PSeInt)
7  //Descripción:      imprime * de forma secuencial
8  //Tema:             ciclos anidados
9  //*****************************************************
10 Algoritmo asteriscos
11
12     Definir i, j como Enteros
13
14     Para i<-1 Hasta 10 Con Paso 1 Hacer
15
16         Para j<-1 Hasta i Con Paso 1 Hacer
17             Escribir "*" Sin Saltar
18         Fin Para
19
20         Escribir ""
21
22     Fin Para
23
24 FinAlgoritmo
```

ciclo interno

ciclo externo

3.3.5 Combinación de estructuras de control

Ya hemos trabajado de manera independiente las estructuras para controlar el flujo de datos en un algoritmo: secuenciales, condicionales y cíclicas con todas sus variaciones, estas estructuras pueden cambiar el control de flujo en un algoritmo, ahora, podemos combinarlas para solucionar tareas complejas. En topografía, es usual hacer poligonales como soporte geométrico para derivar radiaciones de cada vértice que permitan detallar la ubicación espacial de distintos elementos en un levantamiento.

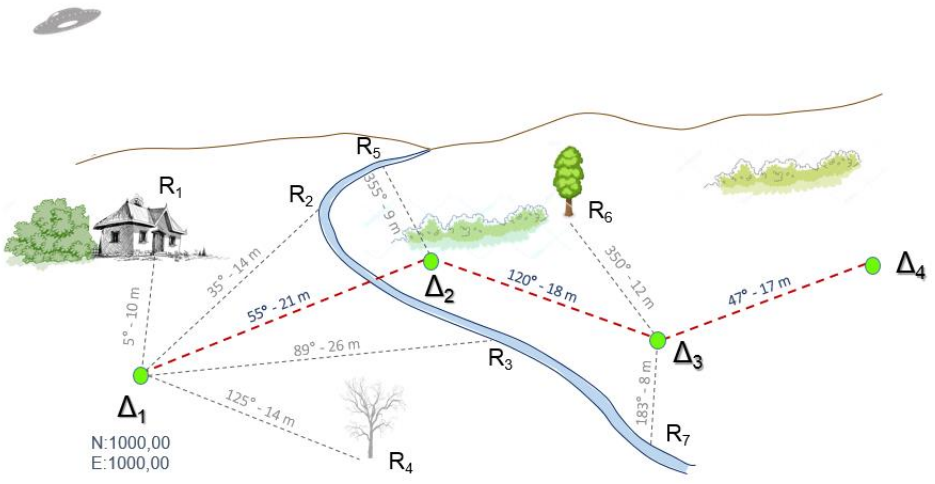


Figura 32. Poligonal abierta con radiaciones

Desarrollemos entonces, un algoritmo en seudocódigo que permita calcular la posición de las estaciones topográficas en una poligonal abierta, y las radiaciones que de cada una dependan, para ello, tomemos como referencia las mediciones de campo de la Figura 32. Poligonal abierta con radiaciones De cualquier forma, el diseño del algoritmo deberá contemplar que es el usuario quien define el número de estaciones y radiaciones a calcular.

```
//*****************************************************************************
//Cod:              T-014
//Nombre:           poligonal abierta
//Fecha:            marzo 2018
//Programador:      JGB-GJC-JJDA
//Lenguaje:         seudocodigo (PSeInt)
//Descripción:      Calcula las coordenadas de los vértices y de las
//                  radiaciones de una poligonal abierta
//Tema:             estructuras de control
//*****************************************************************************
Algoritmo poligonal_abierta

    Definir n, radiaciones, i, j, k, aux_radiaciones como Enteros
    Definir norte, este, azimut, distancia, pm, pp como Reales
    Definir  nueva_norte, nueva_este, norte_rad, este_rad como Reales

    Escribir"ingrese el # de estaciones: "
    Leer n

    Escribir "ingrese coordenada NORTE DELTA 1: "
    Leer norte

    Escribir "ingrese coordenada ESTE DELTA 1: "
    Leer este

    nueva_norte <- norte
    nueva_este  <- este

     Para i<-1 Hasta n-1 Con Paso 1 Hacer

        Escribir "ingrese # de radiaciones que dependen de delta ",i
        Leer radiaciones

        Si (radiaciones>0) Entonces

            Para j<-1 Hasta radiaciones Con Paso 1 Hacer
                aux_radiaciones <- aux_radiaciones+1
                Escribir "DELTA ",i " - ", "R",aux_radiaciones

                Escribir "ingrese el azimut: "
                Leer azimut
                azimut <- azimut*PI/180

                Escribir "ingrese la distancia: "
                Leer distancia

                pm <- distancia*cos(azimut)
                pp <- distancia*sen(azimut)

                norte_rad <- nueva_norte+pm
                este_rad  <- nueva_este+pp

```

```
                Escribir "PM radiacion: ", pm
                Escribir "PP radiacion: ", pp
                Escribir "norte radiacion: ", norte_rad
                Escribir "este radiacion: ", este_rad
            Fin Para
        Fin Si
        Escribir "ingrese el azimut DELTA ", i, " DELTA ",i+1
        Leer azimut
        azimut <- azimut*PI/180

        Escribir "ingrese la distancia: "
        Leer distancia

        pm <- distancia*cos(azimut)
        pp <- distancia*sen(azimut)

        Si (i=1) Entonces
            nueva_norte <- norte+pm
            nueva_este <- este+pp
        SiNo
            nueva_norte <- nueva_norte+pm
            nueva_este <- nueva_este+pp
        Fin Si

        Escribir "PM estación: ", pm
        Escribir "PP estación: ", pp
        Escribir "norte estación: ", nueva_norte
        Escribir "este estación: ", nueva_este
    Fin Para

    Si (i=n) Entonces
        Escribir "ingrese # de radiaciones que dependen de delta ",i
        Leer radiaciones

        Para K<-1 Hasta radiaciones Con Paso 1 Hacer
            Escribir "DELTA ", i, "-", "RAD ", k
            Leer azimut
            azimut <- azimut*PI/180

            Escribir "ingrese la distancia: "
            Leer distancia

            pm <- distancia*cos(azimut)
            pp <- distancia*sen(azimut)

            norte_rad <- nueva_norte+pm
            este_rad  <- nueva_este+pp

            Escribir "PM radiación: ", pm
            Escribir "PP radiación: ", pp
            Escribir "norte radiación: ", norte_rad
            Escribir "este radiación: ", este_rad
        Fin Para
    Fin Si

FinAlgoritmo
```

3.4 Ejercicios propuestos

1) Diseñe un algoritmo que permita hacer la conversión de azimut a rumbo.

2) Escriba un programa en seudocódigo que lea tres números enteros diferentes y muestre como resultado los números ordenados de menor a mayor.

3) Elabore un algoritmo que calcule la elevación de un punto mediante nivelación taquimétrica. Que tenga como datos de entrada: altura instrumental, HS, HM, HI, y lectura del ángulo cenital. Se debe tener en cuenta que: si al momento de registrar el ángulo cenital no se pudo proyectar la altura instrumental con el hilo medio [por un obstáculo en la visual], se debe aplicar la corrección al ángulo α =[(T-O) cos α']/D* sen 1"; donde D es la distancia inclinada, T es la altura del instrumento, y O es la altura del punto visado.

4) Dibuje un diagrama de flujo que permita calcular las coordenadas de N número de puntos en un levantamiento radial, teniendo como datos de entrada azimut y distancia horizontal.

5) Realice un algoritmo en seudocódigo que permita hacer conversión de ángulos horarios en deflexiones.

6) Escriba un programa en seudocódigo que obtenga e imprima y muestre la sumatoria de los términos de la siguiente serie: 2, 5, 7, 10, 12, 15, 17, ...,1800

7) modifique el algoritmo T-006 para cuando el rumbo toma los sentidos N, S, E, W.

8) Elabore un programa en seudocódigo que calcule la sumatoria de ángulos internos y externos de una poligonal.

9) Escriba un programa en seudocódigo de forma tal que, dado como datos de entrada 20 magnitudes angulares con formato GG.MMSS, obtenga el promedio de todos aquellos que sean ≤ 180°30'. El formato GG.MMSS tiene la siguiente equivalencia: 45°20'10"= 45.2010

10) Escriba un diagrama de flujo que lea un número entero N y calcule el resultado de la siguiente serie:

$$1 - \frac{1}{2} + \frac{1}{3} - \frac{1}{4} + \cdots \pm \frac{1}{N}$$

11) Escriba un algoritmo en seudocódigo que permita hacer la multiplicación de dos números enteros mediante sumas sucesivas. Ejemplo: 5x4 = 5+5+5+5 = 20

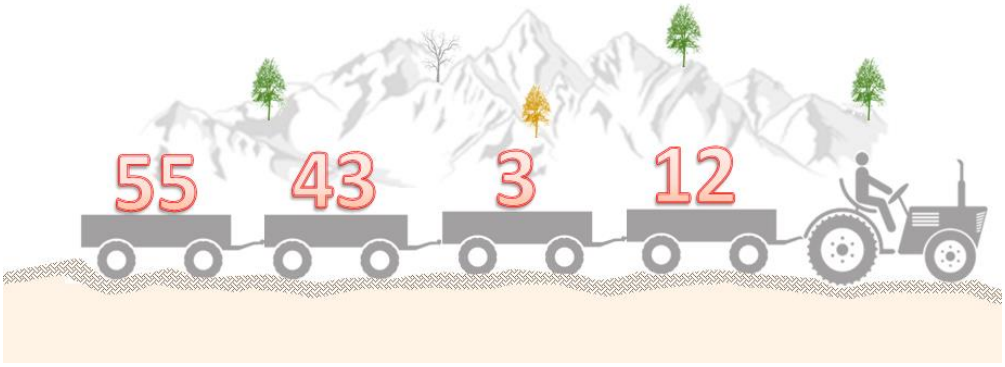
12) Diseñe un algoritmo en seudocódigo que permita calcular el número factorial de número entero, exceptuando el cero. Ejemplo: 5! = 1 x 2 x 3 x 4 x 5 = 120

13) Desarrolle un algoritmo que detecte si un número entero cualquiera es primo o no.

14) Haga un algoritmo que identifique los números pares que existan entre 1 y 'n'. El usuario deberá definir 'n'.

15) Modifique el algoritmo de la nivelación diferencial simple con vistas intermedias, de tal manera que se logre con el ciclo **Mientas**.

16) Suponga que se quieren vender 1000 boletas para la rifa de un reloj, la numeración de manera secuencial inicia en 000 y termina en 999. La persona que participa en la rifa deberá pagar un valor en pesos igual al número de su boleto. Por ejemplo, si el participante sacó la boleta 350 deberá pagar $350. En caso de que se venda la totalidad de las boletas, ¿cuánto dinero se recauda?



Estructuras de datos

Vectores & matrices

Un programador se preocupa por desarrollar e implementar algoritmos para múltiples tareas, por ejemplo, en el cálculo del área de un predio, por el método de coordenadas, usualmente se aplica a sus valores el procedimiento de multiplicaciones cruzadas, pero, ¿qué sucede si uno de estos valores se ingresa erróneamente? no habrá más remedio de entrar de nuevo todo el listado de coordenadas —*que por lo general nunca es corto*— así mismo sucede con los datos de campo de cualquier poligonal. En la medida en que las tareas ganan en complejidad el algoritmo se facilitará a través de estructuras de datos organizados, estos datos podrán almacenarse a través de diferentes estructuras para acceder a ellos, y manejarlos de una manera más ágil, ¡exploremos algunas de sus capacidades!

4.1 Arreglos

Son una colección de elementos finitos del mismo tipo de dato, cada uno de los cuales se indexa por un número que indica su posición, a esta colección de elementos se hace referencia con un nombre común. Todos los elementos de un arreglo se almacenan en posiciones consecutivas de la memoria RAM. Los arreglos pueden ser de una, dos o más dimensiones, aquellos de una dimensión se llaman vectores y los de dos dimensiones matrices.

Como ya lo sabemos, todas las variables deben ser declaradas antes de ser usadas en el programa, así mismo, cualquier arreglo (vector o matriz) tendrá que ser declarado antes de ser utilizado, en la declaración se debe especificar su tamaño. El tamaño declarado informa al compilador para asignar y reservar las ubicaciones de memoria especificadas. Empecemos entonces por los vectores.

4.1.1 Vectores

Como ya se dijo, un vector es una colección de variables del mismo tipo de dato, los elementos individuales que se hallan en su interior son variables indexadas, el número entre corchetes **[]** se llama índice y su longitud corresponde al número de variables indexadas que tenga. Para declarar un arreglo se usa la palabra reservada **Dimension**, la longitud del vector es dada entre corchetes así: **A[4]**

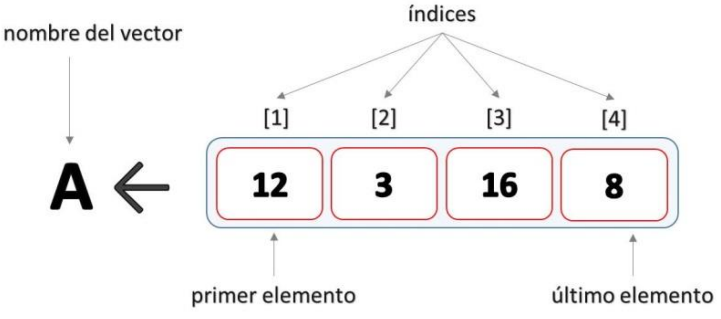


Figura 33. Estructura de un vector

```
Algoritmo vector

    Definir A como Entero
    Dimension A[4]

    A[1] <- 12
    A[2] <- 3
    A[3] <- 16
    A[4] <- 8

FinAlgoritmo
```

Las operaciones básicas asociadas a los arreglos son lectura, búsqueda, modificación y eliminación de datos, estas operaciones se aplican tanto a vectores ordenados como desordenados. Un vector ordenado es aquel que cumple la condición A[1] ≤A[2] ≤A[3] ≤…≤A[N]. Uno desordenado no exige dicha condición. En este libro trataremos solo los desordenados, pues la naturaleza y variabilidad de los datos topográficos que se almacenan en estas estructuras son coordenadas o mediciones de campo donde no se cumple que el primer elemento sea menor o igual que el segundo, y así de manera sucesiva.

4.1.2 Lectura

Consiste en hacer la asignación a cada uno de los elementos del vector, como esta es una tarea repetitiva que se da en función del número de elementos, lo recomendado es hacerlo con una estructura de repetición.

```
1  //*************************************************************************
2  //Cod:          T-015
3  //Nombre:       lectura
4  //Fecha:        marzo 2018
5  //Programador:  JGB-GJC-JJDA
6  //Lenguaje:     seudocódigo (PseInt)
7  //Descripción:  Asigna e imprime los elementos de un vector
8  //Tema:         vectores
9  //*************************************************************************
10
11 Algoritmo vector_lectura
12
13     Definir A,i,n,elemento,x Como Enteros
14     Definir bandera Como Logico
15
16     Escribir "ingrese la cantidad de elementos del vector: "
17     Leer n
18
19     Dimension A[n]
20
21     //*****lectura del vector*********************************************//
22
23     Para i<-1 Hasta n Con Paso 1 Hacer
24         Escribir "ingrese el elemento ", i
25         Leer    elemento
26         A[i] <- elemento
27     Fin Para
28
29     //*****impresión del vector*******************************************//
30
31     Para i<-1 Hasta n Con Paso 1 Hacer
32         Escribir A[i] " " sin saltar
33     Fin Para
34
35 FinAlgoritmo
```

4.1.3 Búsqueda

Se trata de hallar un elemento específico dentro del vector e informar de la posición que ocupa en él. En la Figura 34. Operación de búsqueda en un vector desordenado se tiene un vector con 6 elementos ingresados a la estructura de manera desordenada, el elemento señalado con la letra **x** corresponde al buscado dentro del vector, por lo tanto, el algoritmo deberá señalar que el elemento se ubica en la posición número cuatro.

Figura 34. Operación de búsqueda en un vector desordenado

```
//************************************************************************
//Cod:          T-016
//Nombre:       búsqueda
//Fecha:        abril 2018
//Programador:  JGB-GJC-JJDA
//Lenguaje:     seudocódigo (PseInt)
//Descripción:  Busca un elemento en un vector e identifica su posición
//Tema:         vectores
//************************************************************************

Algoritmo vector_busqueda

    Definir A,i,n,elemento,x Como Enteros
    Definir bandera Como Logico

    Escribir "ingrese la cantidad de elementos del vector: "
    Leer n

    Dimension A[n]

    //*****lectura del vector*************************************************//

    Para i<-1 Hasta n Con Paso 1 Hacer
        Escribir "ingrese el elemento ", i
        Leer    elemento
        A[i] <- elemento
    Fin Para

    //*****impresión del vector***********************************************//

    Para i<-1 Hasta n Con Paso 1 Hacer
        Escribir A[i] " " sin saltar
    Fin Para

    Escribir " "
    Escribir "ingrese el elemento que desea buscar: "
    Leer x

    i <- 1
    bandera <- Falso
```

```
42      //*****búsqueda del elemento*****************************************//
43
44      Mientras (i<=n Y bandera=Falso) Hacer
45          Si (A[i]=x) Entonces
46              bandera <- Verdadero
47              Escribir "El elemento se encuentra en la posición: ", i
48          SiNo
49              i <- i+1
50          Fin Si
51      Fin Mientras
52
53      Si (bandera=Falso) Entonces
54          Escribir "El elemento no se encuentra en el vector..."
55      Fin Si
56
57  FinAlgoritmo
```

4.1.4 Modificación

El algoritmo de modificación se apoya en el de búsqueda, puesto que para modificar un elemento dentro del vector primero habrá que ubicarlo, una vez se conozca su posición bastará con reemplazarlo. En la Figura 35. Operación de modificación en un vector desordenado la variable **x** almacena el elemento que se quiere buscar en un vector, y la variable **nuevo** almacena el elemento por el cual será reemplazado. Ahora revisa las siguientes líneas de código fuente para comprender el funcionamiento del algoritmo.

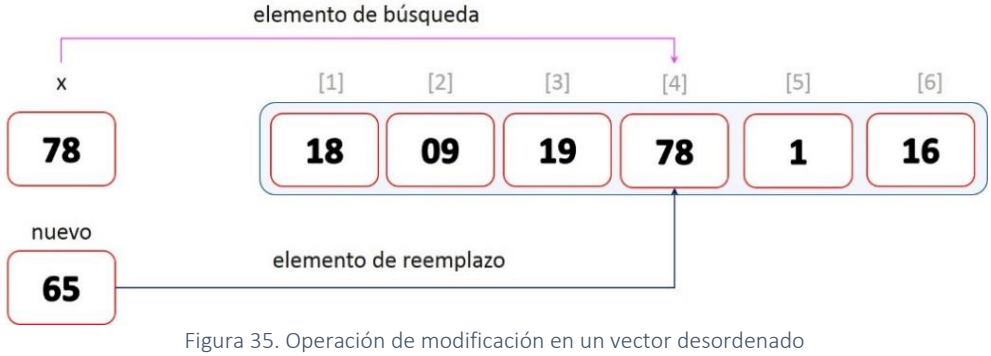


Figura 35. Operación de modificación en un vector desordenado

```
42      //*****búsqueda y reemplazo del elemento*****************************//
43
44      Mientras (i<=n Y bandera=Falso) Hacer
45          Si (A[i]=x) Entonces
46              bandera = Verdadero
47              Escribir "El elemento se encuentra en la posición: ", i
48              Escribir "ingrese el nuevo elemento: "
49              Leer nuevo
50              A[i] <- nuevo
51          SiNo
52              i <- i+1
53          Fin Si
54      Fin Mientras
```

4.1.5 Eliminación

Se refiere al borrado de un elemento existente en el vector, posterior a ello se deberán reorganizar todos sus elementos. En la Figura 36. Operación de eliminación en un vector desordenado la variable **x** almacena el elemento de búsqueda en el vector para su eliminación, una vez hallado los elementos que se encuentran a su derecha deberán desplazarse una posición a la izquierda para formar el nuevo vector.

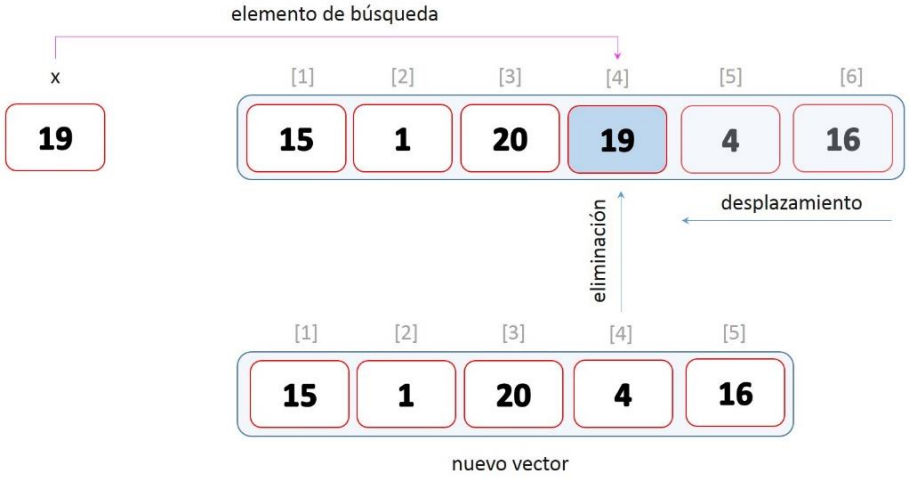


Figura 36. Operación de eliminación en un vector desordenado

Revisa el algoritmo de eliminación adaptado de (Cairó Battistutti, 2005) donde podrás ver que una estructura de repetición anidada y una estructura de decisión solucionan el problema de manera eficiente.

```
//************************************************************************
Algoritmo vector_eliminacion

    definir n, A, elemento, i, x, k como entero
    definir bandera como logico

    Escribir "Entre el número de elementos del vector: "
    leer n

    //*****lectura del vector*********************************************//

    Dimension A[n]  // se declara el # de elementos que tiene el vector

    Para i=1 Hasta n Con Paso 1 Hacer
        Escribir "ingrese el elemento ", i, ":"
        Leer elemento
        A[i] <- elemento
    Fin Para

    //*****impresión del vector*******************************************//

    Escribir("El vector es :")

    Para i<-1 Hasta n Con Paso 1 Hacer
        Escribir A[i] " " sin saltar
    Fin Para

```

```
36      //*****eliminación del elemento dentro del vector*******************//
37
38      Escribir "ingese el elemento que se va a eliminar: "
39      Leer x
40
41      i <- 1
42      bandera <- Falso
43
44      Mientras (i<=n Y bandera=Falso) Hacer
45          Si (A[i] = x) Entonces
46              bandera <- Verdadero
47              n <- n-1
48              k <- i
49
50              Mientras (k<=n) Hacer
51                  A[k] <- A[k+1]
52                  k <- k+1
53              Fin Mientras
54
55          SiNo
56              i <- i+1
57          Fin Si
58      Fin Mientras
59
60      Si (bandera=Falso) Entonces
61          Escribir "el elemento ", x, " no está en el vector"
62      Fin Si
63
64      //*****impresión del nuevo vector***********************************//
65
66      Escribir("El nuevo vector es :")
67
68      Para i<-1 Hasta n Con Paso 1 Hacer
69          Escribir A[i] " " sin saltar
70      Fin Para
71
72  FinAlgoritmo
```

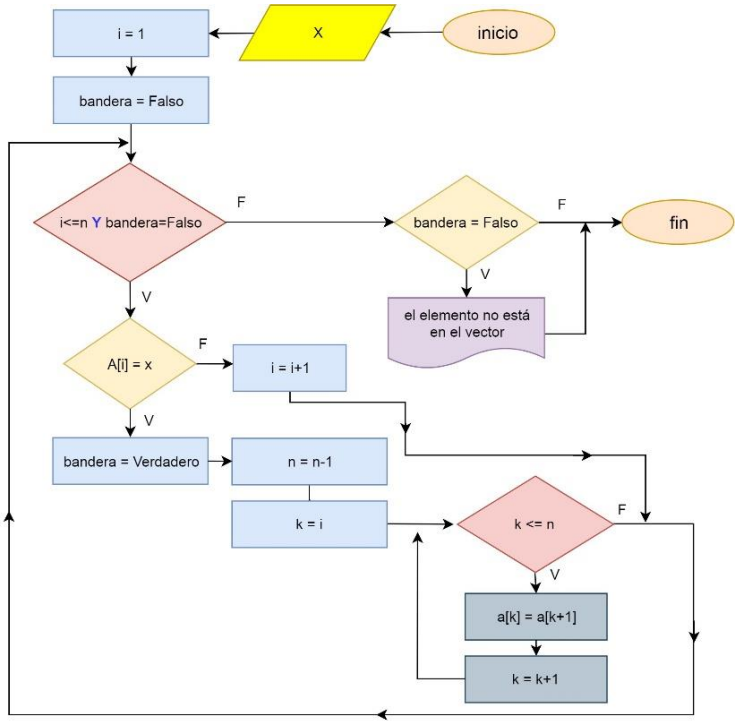


Figura 37. Algoritmo de eliminación

4.1.6 Aplicación topográfica de cálculo de áreas

Este método analítico se basa en puntos de coordenadas planas conocidas producidas a partir de mediciones de campo (ángulos y distancias). Para determinar el área de un predio se aplica la fórmula de Gauss:

$$A = \frac{1}{2}\left|\sum_{i=1}^{n-1} x_i \cdot y_{i+1} + x_n \cdot y_i - \sum_{i=1}^{n-1} x_{i+1} \cdot y_i - x_1 \cdot y_n\right| \quad (1)$$

$$A = \frac{1}{2}|x_1.y_2 + x_2.y_3 + \cdots x_{n-1}.y_n + x_n.y_n - x_2.y_1 - x_3.y_2 - \cdots - x_n.y_{n-1} - x_1.y_n| \quad (2)$$

Donde, A es el área del polígono, n es el número de lados, $(x_i.y_i)i = 1,2,\ldots,n$ son los vértices del polígono.

Luego de los respectivos procedimientos de campo, en la medición de dos predios referidos en la Figura 38. Plano de levantamiento planimétrico se obtuvo el listado de coordenadas de la Tabla 13.

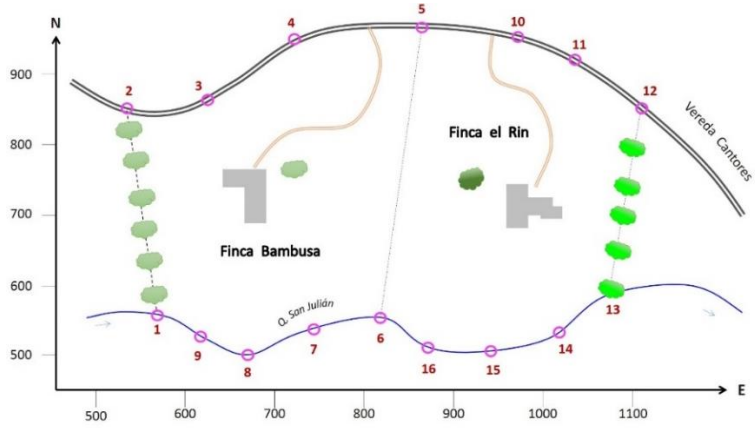
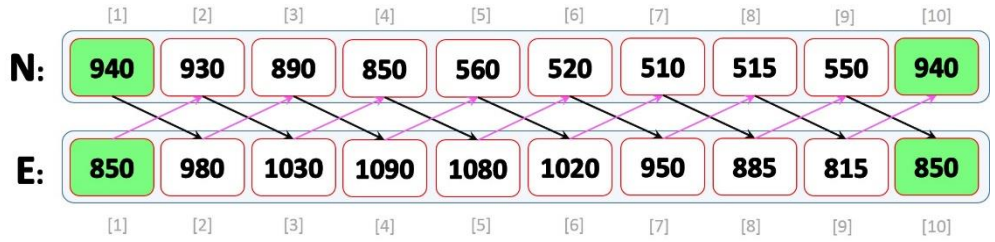


Figura 38. Plano de levantamiento planimétrico

Tabla 13.
Listado de coordenadas para cálculo de área

Punto	Norte	Este		Punto	Norte	Este
1	560	580		9	520	615
2	850	560		10	930	980
3	860	640		11	890	1030
4	930	720		12	850	1090
5	940	850		13	560	1080
6	550	815		14	520	1020
7	540	730		15	510	950
8	500	690		16	515	885

Usando el modelo de ordenamiento vectorial de las ecuaciones 1 y 2, para calcular el área de la Finca El Rin en la Figura 38. Plano de levantamiento planimétrico se obtiene:



$$A = \frac{1}{2}|(940 * 980) + (930 * 1030) + (890 * 1090) + (850 * 1080) + (560 * 1020) + (520 * 950) + (510 * 885) + (515 * 815) + (550 * 850) - (850 * 930) - (980 * 890) - (1030 * 850) - (1090 * 560) - (1080 * 520) - (1020 * 510) - (950 * 515) - (885 * 550) - (815 * 940)|$$

$$A = \frac{1}{2}|6'170.975 - 5'972.500|$$

$$A = 99.237{,}5\ m^2$$

Usando vectores se puede solucionar el problema a través del siguiente código fuente:

```
//*****************************************************************************
//Cod:          T-019
//Nombre:       area_coordenadas
//Fecha:        mayo 2018
//Programador:  JGB-GJC-JJDA
//Lenguaje:     seudocódigo (PseInt)
//Descripción:  Calcular el area de un predio por el método de Gauss
//Tema:         vectores
//*****************************************************************************

Algoritmo area_coordenadas

    Escribir "ingrese el número de puntos que definen el límite del predio: "
    Leer n

    Dimension norte[100]
    Dimension  este[100]

    //*****lectura del vectores Norte y Este*********************************//

    Para i<-1 Hasta n Con Paso 1 Hacer

        Escribir "NORTE ",i, ":"
        Leer coord_norte
        norte[i] <- coord_norte

        Escribir "ESTE ",i, ":"
        Leer coord_este
        este[i] <- coord_este

        j <- i
    Fin Para
```

```
34      Escribir "ingrese la NORTE del punto 1: "
35      Leer coord_norte
36      norte[j+1] <- coord_norte
37
38      Escribir "ingrese la ESTE del punto 1: "
39      Leer coord_este
40      este[j+1] <- coord_este
41
42      //*****impresión de los vectores*************************************//
43
44      Para i<-1 Hasta n+1 Con Paso 1 Hacer
45
46          Escribir i, "***", norte[i], "***", este[i]
47
48      Fin Para
49
50      //*****cálculo de productos*****************************************//
51
52      acumulador_descendente <- 0
53      acumulador_ascendente  <- 0
54
55      Para i<-1 Hasta n+1 Con Paso 1 Hacer
56          producto_descendente  <- (norte[i]*este[i+1])
57          acumulador_descendente <- acumulador_descendente+producto_descendente
58
59          producto_ascendente   <- (este[i]*norte[i+1])
60          acumulador_ascendente <- acumulador_ascendente+producto_ascendente
61      Fin Para
62
63      Imprimir "descendente", acumulador_descendente
64      Imprimir "ascendente", acumulador_ascendente
65
66      area <- abs(acumulador_descendente-acumulador_ascendente)/2
67
68      Escribir "el area del predio es: ", area, "m²"
69
70  FinAlgoritmo
```

4.1.7 Matrices

Son arreglos bidimensionales estructurados en forma de tabla, cada elemento se ubica a través de un par de números en función de su posición fila y columna. A través de los índices se tiene acceso a sus los elementos en forma individual. Como sucede con los vectores, las matrices admiten operaciones básicas de lectura, búsqueda, modificación y eliminación.

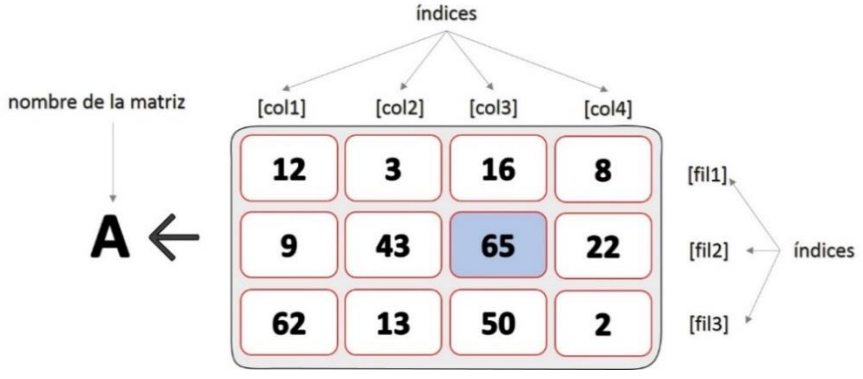


Figura 39. Estructura de una matriz

Este tipo de estructuras usan la misma palabra reservada del sistema que los vectores para su declaración, así, la matriz de la Figura 39. Estructura de una matriz se declarará de la siguiente forma: `Dimension A[3,4]`, los números entre corchetes corresponden a la cantidad de filas y columnas. El elemento resaltado en color azul se ubica en la posición: `A[2,3].`

4.1.8 Lectura

Cuando se introdujo la lectura en vectores se usó un ciclo para asignar valores a cada uno de los componentes. Lo mismo sucede con los arreglos bidimensionales, sin embargo, como sus elementos deben referenciarse con dos índices, se usan dos ciclos para la asignación de sus elementos de forma secuencial.

Si revisas el DF de la Figura 40. Lectura de una matriz, el ciclo para mover las posiciones de las columnas se halla dentro del ciclo que mueve las posiciones de las filas.

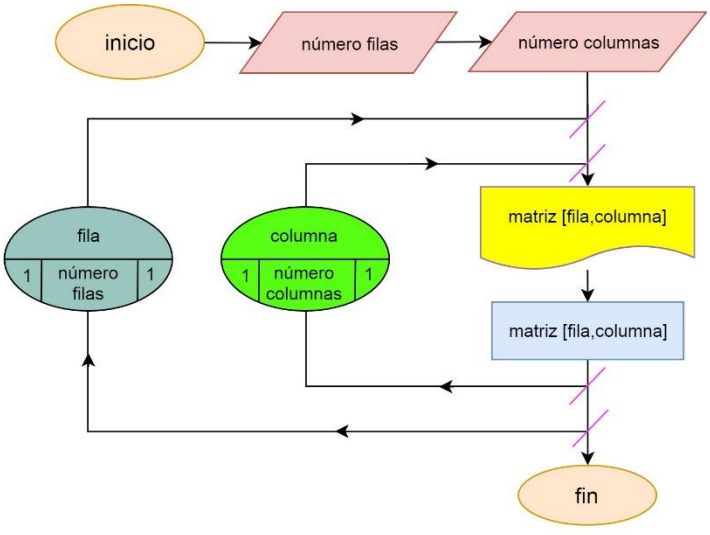


Figura 40. Lectura de una matriz

Ahora, analiza la implementación del algoritmo en seudocódigo, y podrás percibir que también hemos puesto otro ciclo anidado para la impresión de los elementos de la matriz. Si te das cuenta, el algoritmo es muy parecido al de lectura e impresión de vectores, la diferencia radica en el anidamiento del ciclo para el recorrido de los elementos a través de las columnas. Por este motivo no desarrollaremos los algoritmos de búsqueda, modificación y eliminación.

```
//*****************************************************************************
//Cod:          T-020
//Nombre:       lectura_impresión
//Fecha:        enero 2019
//Programador:  JGB-GJC-JJDA
//Lenguaje:     seudocódigo (PseInt)
//Descripción:  asigna e imprime los elementos a una matriz
//Tema:         matrices
//*****************************************************************************

Algoritmo matriz_lectura

    Definir matriz, num_filas, num_columnas, fila, columna como Entero

    Escribir "ingrese la cantidad de filas de la matriz: "
    Leer num_filas

    Escribir "ingrese la cantidad de columnas de la matriz: "
    Leer num_columnas

    Dimension matriz[num_filas,num_columnas]

    //*****lectura de la matriz*************************************************//

    Para fila <- 1 Hasta num_filas Con Paso 1 Hacer
        Para columna <- 1 Hasta num_columnas Con Paso 1 Hacer
            Escribir sin saltar "matriz [", fila, ",", columna, "]"
            leer matriz[fila,columna]
        Fin Para
    Fin Para

    //*****impresión de la matriz***********************************************//

    Escribir "Contenido de la matriz"
    Para fila <-1 Hasta num_filas Con Paso 1 Hacer
        Para columna <- 1 Hasta num_columnas Con Paso 1 Hacer
            Escribir sin saltar  matriz[fila, columna], '  '
        Fin Para
        Escribir ''
    Fin Para

FinAlgoritmo
```

Ahora, para desarrollar habilidades en este tema tan interesante, intentemos un algoritmo para transponer una matriz. Para ello se deberá intercambiar la posición de las filas con la de las columnas.

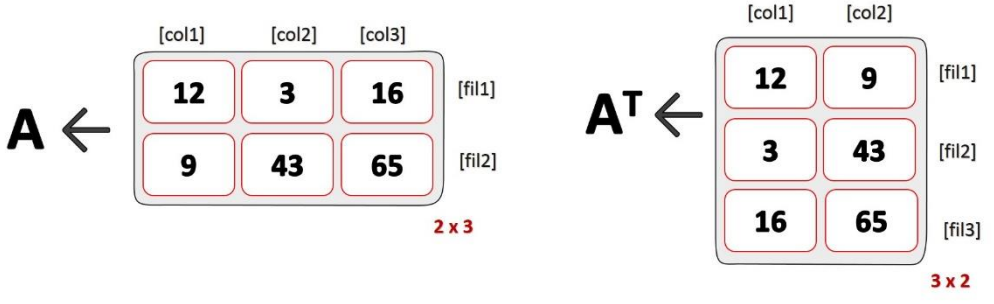


Figura 41. Matriz transpuesta

Puedes utilizar el algoritmo de lectura e impresión, y complementarlo con el siguiente código fuente. Observa que bastará con declarar la matriz transpuesta y crear un nuevo ciclo anidado donde:

```
transpuesta[columna,fila] <- matriz[fila,columna]
```

```
43      //*****transposición de la matriz*************************
44
45      Para fila <-1 Hasta num_filas Con Paso 1 Hacer
46          Para columna <- 1 Hasta num_columnas Con Paso 1 Hacer
47              transpuesta[columna,fila] <- matriz[fila,columna]
48          Fin Para
49      Fin Para
50
```

Ahora, si quieres imprimir el resultado de la transposición, analiza el siguiente extracto de código:

```
53      Escribir "Matriz transpuesta"
54      Para fila <-1 Hasta num_columnas Con Paso 1 Hacer
55          Para columna <- 1 Hasta num_filas Con Paso 1 Hacer
56              Escribir sin saltar  transpuesta[fila, columna], '  '
57          Fin Para
58          Escribir ''
59      Fin Para
```

4.1.9 Ajuste de poligonales – método de la brújula

Este modelo de ajuste desarrollado por Nathaniel Bowditch se soporta en que el error de cierre del polígono se produce a partir de desviaciones homogéneas derivadas de las mediciones de ángulos y distancias, por eso, se distribuyen proporcionalmente a cada lado del polígono. El error de cierre se elimina haciendo las correcciones para cada longitud, lo cual se obtiene dividiendo la sumatoria de las proyecciones entre el perímetro del polígono, su modelo matemático es el siguiente:

$$\delta PM = \sum_{i=1}^{n} PM \quad ; \quad \delta PP = \sum_{i=1}^{n} PP \tag{3}$$

$$CPM = -\left(\frac{\delta PM}{\sum L_i}\right) \cdot L_i \quad ; \quad CPP = -\left(\frac{\delta PP}{\sum L_i}\right) \cdot L_i \tag{4}$$

$$PMC = PM + CPM \quad ; \quad PPC = PP + CPP \tag{5}$$

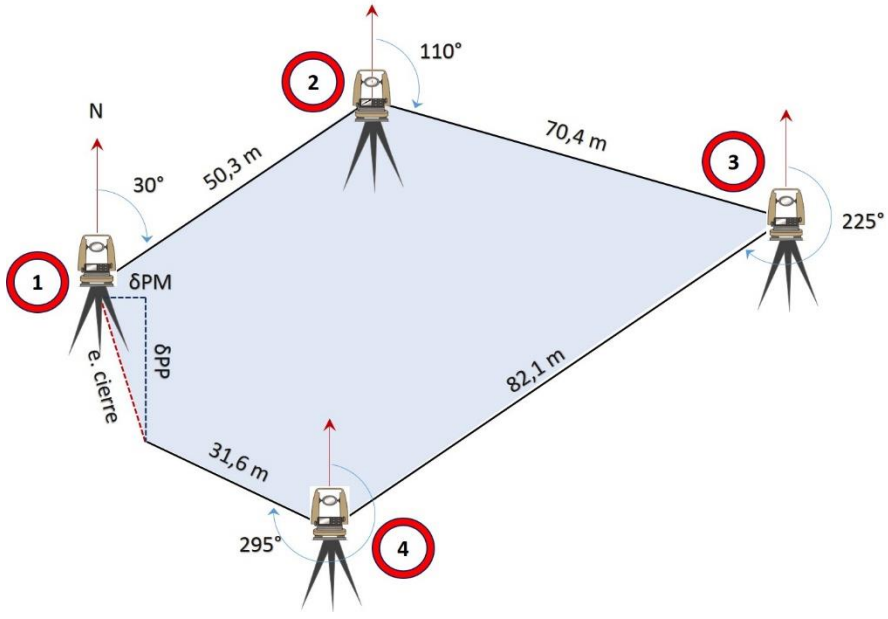


Figura 42. Mediciones de campo poligonal cerrada

Supongamos que en el levantamiento de un predio se realizó la poligonal de la Figura 42. Mediciones de campo poligonal cerrada, integrado los datos de campo con las ecuaciones 3, 4, y 5 se obtiene:

Tabla 14.
Parámetros de ajuste

línea	Az.	DH	PM	PP	CPM	CPP	PMC	PPC	N	E
									500,000	500,000
1-2	30°	50,3	43,561	25,150	5,411	-0,990	48,972	24,160	548,972	524,160
2-3	110°	70,4	-24,078	66,154	7,573	-1,385	-16,505	64,769	532,467	588,930
3-4	225°	82,1	-58,053	-58,053	8,832	-1,615	-49,221	-59,669	483,246	529,261
4-1	295°	31,6	13,354	-28,639	3,399	-0,622	16,754	-29,261	500,000	500,000
		∑Li: 234,4	δPM: -25,216	δPP: 4,612						

Ahora, para automatizar el procedimiento a través de un algoritmo y poder cambiar la cantidad de vértices, se puede usar una estructura matricial así:

```
//*************************************************************************
//Cod:          T-022
//Nombre:       ajuste_brujula
//Fecha:        mayo 2018
//Programador:  JGB-GJC-JJDA
//Lenguaje:     seudocódigo (PseInt)
//Descripción:  ajusta poligonal cerraada por método de la brújula
//Tema:         matrices
//*************************************************************************

Algoritmo ajuste_brujula

    definir n, f, c, i como enteros
    definir matriz, norte, este, angulo, angulo_radianes como reales
    definir perimetro, delta_pm, delta_pp como reales


    Escribir sin saltar  "ingrese el # de vertices: "
    leer n

    Dimension matriz[n,10]

    Escribir sin saltar "ingrese coordenada norte de delta 1: "
    leer norte

    Escribir sin saltar "ingrese coordenada este de delta 1: "
    leer este

    f <- 1
    c <- 1

//*****lectura de datos de campo y cálculo de proyecciones descorregidas***

    Para i<-1 Hasta n Con Paso 1 Hacer
        Si (i<=n-1) Entonces
            Escribir sin saltar "ingrse azimut de la linea ", i, "-",i+1
        SiNo
            Escribir sin saltar "ingrese el azimut de la linea ", i "-",1
        Fin Si
        leer angulo
        angulo_radianes <- (angulo*PI)/180
        matriz[f,c] <- angulo_radianes


        Si (i<=n-1) Entonces
            Escribir sin saltar "ingrese distancia ", i, "-", i+1
        SiNo
            Escribir sin saltar "ingrese distancia ", i, "-",1
        Fin Si
        leer distancia

```

```
        matriz[f,c+1] <- distancia
        perimetro <- perimetro+distancia

        pm <- distancia*cos(angulo_radianes)
        matriz[f,c+2] <- pm
        delta_pm <- delta_pm + pm

        pp <- distancia*sen(angulo_radianes)
        matriz[f,c+3] <- pp
        delta_pp <- delta_pp + pp

        f <- f+1

    Fin Para

//*****cálculo de correcciones y proyecciones corregidas******************

    c <- 5

    Para f<-1 Hasta n Con Paso 1 Hacer

        matriz[f,c]   <- -(delta_pm/perimetro) * matriz[f,2] // CPM
        matriz[f,c+1] <- -(delta_pp/perimetro) * matriz[f,2] // CPP

        matriz[f,c+2] <- matriz[f,3] + matriz[f,5] //PMC
        matriz[f,c+3] <- matriz[f,4] + matriz[f,6] //PPC

    Fin Para


//*****cálculo de coordenadas corregidas*********************

    matriz[1,9]  <- norte + matriz[1,7]  //norte vertice 2
    matriz[1,10] <- este  + matriz[1,8]  //este vertice 2

    Para i<-2 Hasta n Con Paso 1 Hacer
        matriz[i,9]  <- matriz[i-1,9]   + matriz[i,7]
        matriz[i,10] <-  matriz[i-1,10] + matriz[i,8]
    Fin Para

//*****impresión de la matriz final*************************

    Para f <- 1 Hasta 4 Con Paso 1 Hacer
        Para c <- 1 Hasta 10 Con Paso 1 Hacer
            Escribir sin saltar matriz[f,c], ' '
        Fin Para
        Escribir ' '
    Fin Para

FinAlgoritmo
```

4.2 Ejercicios propuestos

1) Elabore un algoritmo que lea una vector de 20 elementos, calcule e imprima el promedio de los elementos de las posiciones pares.

2) Construya un algoritmo en seudocódigo que permita leer dos vectores A y B, ambos de igual longitud y un tercer vector C que calcule e imprima la suma de los vectores anterior, sabiendo que C[i] = A[i]+B[i]

3) Desarrolle un algoritmo que permita leer un vector de N componentes, y ordenarlo de manera ascendente, imprimir el vector antes y después de ordenarlo.

4) Realice un algoritmo que permita ingresar un número 'n' por teclado, luego, el algoritmo deberá cargar un vector con los 'n' primeros términos de la serie Fibonacci. Se deberá declarar un vector de gran tamaño, por ejemplo 100 elementos.

5) Modifique el algoritmo T-019 area_coordenadas de tal manera que, luego de imprimir los valores de las coordenadas, el usuario pueda modificar o eliminar algún valor del listado.

6) Escriba el código fuente para luego de leer un vector de 'n' posiciones, se pueda insertar un elemento más en la posición que desee el usuario.

7) Desarrolle un algoritmo que permita hacer la compensación a una red de nivelación, el usuario deberá definir la cantidad de BMs que conforman su circuito.

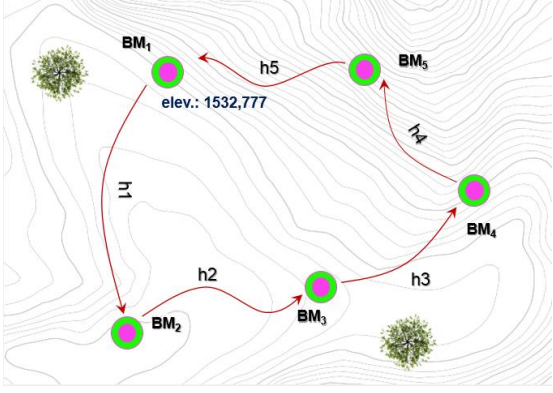


Figura 43. Red de nivelación

Teniendo en cuenta la distribución de los BMs de la Figura 43. Red de nivelación y los datos de campo de la Tabla 15. se deberán aplicar las ecuaciones 6, 7, y 8 tomadas de (Jiménez Cleves, Garzón Barrero, & Londoño Pinilla, 2017), para hallar las elevaciones compensadas.

Tabla 15.
Registros de nivelación de campo

itinerario	desnivel	distancia (km)	corrección	desnivel corregido	elevación compensada
BM_1-BM_2	-1,377	3,8			
BM_2-BM_3	2,433	4,3			
BM_3-BM_4	-1,622	3,5			
BM_4-BM_5	0,033	3,3			
BM_5-BM_1	0,524	3,6			

Error de cierre:

$$e_c = \sum_{j=1}^{n} \cdot H_j' \tag{6}$$

Correcciones:

$$C_j = -\frac{D_j}{\sum_{j=1}^{n} \cdot D_j} \cdot e_c \tag{7}$$

Desniveles corregidos:

$$H_j = H_j' + C_j \tag{8}$$

8) Elabore un algoritmo que cargue una matriz de 20x20 con números aleatorios e imprima por pantalla aquellos números que se repitan.

9) Desarrolle un algoritmo para hacer el ajuste de una poligonal cerrada por el método del tránsito.

10) Escriba un algoritmo que permita hacer la multiplicación entre dos matrices. Recuerda que el número de columnas de A debe ser igual al número de filas de B.

Creación de

funciones & procedimientos

Las funciones son módulos de código autónomos, comprensibles y manejables que ejecutan una tarea específica. Usualmente toman datos los procesan y retornan un resultado, una vez que se escribe una función esta puede ser llamada dentro del programa principal una y otra vez. Este modelo de programación tiene grandes ventajas: (i) permiten concebir el programa como un conjunto de sub-programas, (ii) permiten reutilizar el código fuente en lugar de re-escribirlo. Y (iii) permiten probar pequeñas partes del programa de forma aislada del resto.

5.1 ¿Qué diferencia existe entre funciones y procedimientos?

Las funciones y los procedimientos son muy cercanos: se puede decir que los procedimientos son un caso especial de las funciones, así como un cuadrado es un tipo especial de rectángulo. En general, las funciones retornan un valor, mientras que los procedimientos no lo hacen. Al devolver un valor, queremos decir que, la función produce algún resultado, que se pasa de nuevo cuando se hace el llamado de la función.

5.2 Procedimientos

De acuerdo con los profesores (Herrera Morales, Gutierrez Posada, & Pulgarín Giraldo, 2017) "un procedimiento es un conjunto finito de instrucciones con un único propósito bien definido y que puede ser invocado por medio de un nombre que lo identifica de manera única", esto permite hacer más cómodos nuestros códigos y reutilizarlos, especialmente cuando son muy extensos.

Revisa el siguiente código fuente, hemos usado la palabra reservada **Subproceso** para la creación del procedimiento, su nombre es **traer_mensaje**. En la línea 14 se encuentra la instrucción que ejecutará el procedimiento cuando sea llamado, se escribirá el mensaje "si lo puedes soñar, lo puedes hacer". Si revisas el algoritmo principal te darás cuenta que en las líneas 24, 25, 26, y 27 se hace el llamado al procedimiento **traer_mensaje**

```
10  //**************PROCEDIMIENTO************************
11
12  Subproceso traer_mensaje
13
14      Escribir"si lo puedes soñar, lo puedes hacer"
15
16  FinSubProceso
17
18  //***************************************************
19
20  //*************ALGORITMO PRINCIPAL*******************
21
22  Algoritmo frase_motivacional
23
24      traer_mensaje
25      traer_mensaje
26      traer_mensaje
27      traer_mensaje
28
29  FinAlgoritmo
30  //***************************************************
```

Una vez se ejecuta el programa obtendremos lo siguiente:

```
*** Ejecución Iniciada. ***
si lo puedes soñar, lo puedes hacer
si lo puedes soñar, lo puedes hacer
si lo puedes soñar, lo puedes hacer
si lo puedes soñar, lo puedes hacer
*** Ejecución Finalizada. ***
```

5.3 Funciones

Una función es un fragmento de programa que realiza una tarea definida. Por ejemplo, se puede escribir una función que calcule el promedio de tres números. Una vez escrita, esta función se puede utilizar muchas veces sin tener que volver a escribirla una y otra vez. Las funciones proporcionan mejor modularidad para su aplicación y un alto grado de reutilización de código fuente. Existen dos formas de hacer el paso de parámetros a una función: por valor y por referencia

5.3.1 Funciones con parámetros por valor

Cuando una función pasa sus parámetros por valor, los cambios que se den en la variable que recibe el parámetro no influencian el programa principal cuando se hace el llamado de la función.

En la línea 11 el nombre de la función es **definitiva** que espera tres entradas **nota1**, **nota2**, **nota3**. En programación estas entradas se llaman parámetros y representan los tres valores enviados cuando se utiliza la función.

```
 9  //*********************************************************
10
11  Funcion resultado <- definitiva ( nota1,nota2,nota3 )
12
13      Definir resultado como real
14
15      resultado <- (nota1+nota2+nota3)/3
16
17  Fin Funcion
18  //*********************************************************
```

La función tiene su propia variable privada llamada **resultado** que se calcula a partir de los parámetros, luego la función 'retorna' el resultado. El acto de usar la función,

es lo que se conoce como 'llamar a la función', en el siguiente programa hay 6 llamados a la función entre las líneas 23 a 28.

```
19 //**********************ALGORITMO PRINCIPAL**********************************
20
21 Algoritmo notas
22
23     Escribir "La nota definitiva de Adolfo es: ", definitiva(3.0, 2.8, 3.7)
24     Escribir "La nota definitiva de Amalia es: ", definitiva(3.5, 2.2, 3.2)
25     Escribir "La nota definitiva de Arturo es: ", definitiva(2.5, 3.2, 4.7)
26     Escribir "La nota definitiva de Marina es: ", definitiva(2.2, 1.3, 1.8)
27     Escribir "La nota definitiva de Hector es: ", definitiva(3.5, 2.2, 3.4)
28     Escribir "La nota definitiva de Brayan es: ", definitiva(4.1, 1.8, 2.9)
29
30 FinAlgoritmo
31
32 //*****************************************************************************
```

Al ejecutar el algoritmo se obtiene:

```
*** Ejecución Iniciada. ***
La nota definitiva de Adolfo es: 2.8333333333
La nota definitiva de Amalia es: 2.9666666667
La nota definitiva de Arturo es: 3.4666666667
La nota definitiva de Marina es: 1.7666666667
La nota definitiva de Héctor es: 3.0333333333
La nota definitiva de Brayan es: 2.9333333333
*** Ejecución Finalizada. ***
```

Como puedes ver, las funciones en la programación son muy similares a las de matemáticas, y sirven para "empaquetar" algunos cálculos a fin de poderlas separar y usar una y otra vez. Mira como trabajamos con la siguiente función:

$$f_{(y)} = 3x + 2 \tag{9}$$

Para evaluar $f_{(y)}$ se le debe dar un valor a x

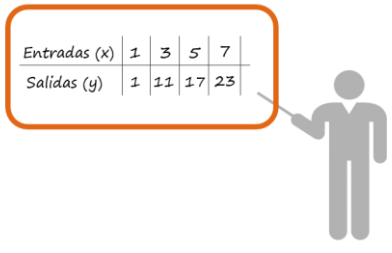


Figura 44. salidas de la función de sustitución

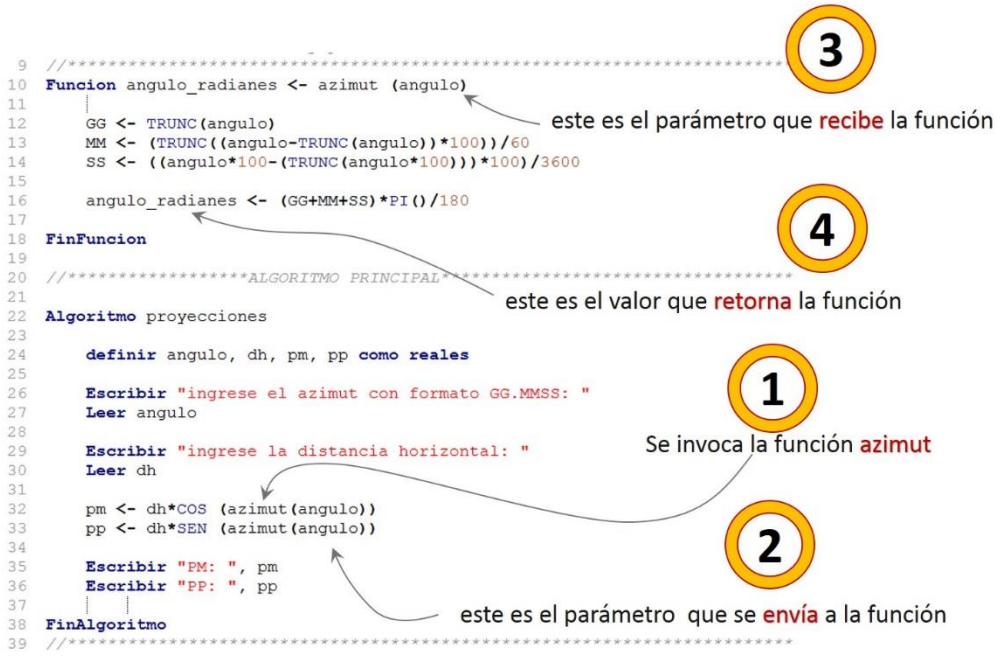
```
 9 //*****************************************************
10 Funcion resultado <- f (x)
11
12     Definir resultado como real
13
14     resultado <- 3*x+2
15
16 FinFuncion
```

```
18  //******************ALGORITMO PRINCIPAL**********************
19
20      Algoritmo funcion_matematica
21
22          Escribir f(1)
23          Escribir f(3)
24          Escribir f(5)
25          Escribir f(7)
26
27  FinAlgoritmo
28  //***************************************************************
```

Al correr el algoritmo tenemos:

```
*** Ejecución Iniciada. ***
5
11
17
23
*** Ejecución Finalizada. ***
```

Ahora, escribamos un programa en seudocódigo usando una función que calcule proyecciones meridianas y paralelas a partir de azimut y distancia horizontal, el azimut deberá ser ingresado con el formato GG.MMSS, de tal forma que si el azimut es 15°20'10" deberá ser ingresado 15.2010




```
9   //***************************************************************************
10  Funcion angulo_radianes <- azimut (angulo)
11
12      GG <- TRUNC(angulo)
13      MM <- (TRUNC((angulo-TRUNC(angulo))*100))/60
14      SS <- ((angulo*100-(TRUNC(angulo*100)))*100)/3600
15
16      angulo_radianes <- (GG+MM+SS)*PI()/180
17
18  FinFuncion
19
20  //******************ALGORITMO PRINCIPAL**********************************
21
22  Algoritmo proyecciones
23
24      definir angulo, dh, pm, pp como reales
25
26      Escribir "ingrese el azimut con formato GG.MMSS: "
27      Leer angulo
28
29      Escribir "ingrese la distancia horizontal: "
30      Leer dh
31
32      pm <- dh*COS (azimut(angulo))
33      pp <- dh*SEN (azimut(angulo))
34
35      Escribir "PM: ", pm
36      Escribir "PP: ", pp
37
38  FinAlgoritmo
39  //***************************************************************************
```

5.3.2 Funciones con parámetros por referencia

Al pasar un parámetro por referencia se apunta a la dirección de memoria de la variable original. La función opera sobre el argumento, por tanto, si el parámetro sufre algún cambio, el valor original pasado por referencia también cambiará.

Cuando un parámetro se pasa por **referencia**, tanto el que invoca como el invocado usan la misma variable para el parámetro. Si el invocado modifica la variable del parámetro el efecto es visible al invocador de la variable.

Cuando un parámetro se pasa por **valor**, el invocador y el invocado tienen dos variables independientes con el mismo valor, si el invocado modifica la variable del parámetro, el efecto no es visible al invocador.

Actualmente, el paso de parámetros por referencia se considera una práctica poco recomendable en programación, debido a que los lenguajes modernos hacen el paso de parámetros exclusivamente por valor.

La siguiente función, recibe el parámetro **distancia** por valor, y el parámetro **to** por referencia, nótese que al cambiar el valor de la variable **to** dentro de la función es también cambiado su valor en el programa principal.

```
10  //**************FUNCIÓN*****************************************************
11
12  Funcion correccion_temperatura (distancia por valor, to por referencia)
13
14      to <- 20
15      ct <- 0.000012*distancia*(t-to)
16      Escribir "corrección: ", ct
17
18  FinFuncion
19
20  //**************************************************************************
21
22  //*************ALGORITMO PRINCIPAL*****************************************
23
24  Algoritmo error_sistematico
25
26      Escribir "ingrese la temperatura ambiente al momento de la medición: "
27      Leer t
28
29      Escribir "ingrese la distancia medida: "
30      Leer distancia
31
32      to <- 25
33      correccion_temperatura(distancia,to)
34      Escribir "este valor cambió Por Referencia: to: ", to
35
36  FinAlgoritmo
37  //**************************************************************************
```

5.4 Ejercicios propuestos

Todos los algoritmos propuestos deberán ser construidos usando procedimientos y funciones, pues es la naturaleza de este capítulo.

1) Construya un algoritmo que lea un número entero entre 0 y 50 e identifique si se trata o no de un número primo.

2) Reescriba el algoritmo de la página 56 en términos de funciones y procedimientos.

3) Diseñe un algoritmo que retorne el área de un círculo teniendo como entrada su radio.

4) Escriba un algoritmo que permita calcular el factorial de un número entero positivo.

5) Elabore un algoritmo que lea una matriz de 3 filas x 3 columnas y calcule su matriz transpuesta.

6) Construya un algoritmo que permita leer un vector de 'n' elementos y calcule su media.

7) Reestructure el programa de ajuste por el método de la brújula de la página 90 a través de funciones y procedimientos.

8) Realice un algoritmo que permita hacer conversión de ángulos horarios en deflexiones.

9) Elabore un programa en seudocódigo que calcule la sumatoria de ángulos internos y externos de una poligonal.

10) Escriba un algoritmo que permita calcular el área de un predio a partir de las coordenadas de sus límites.

Glosario

A

archivo
también llamado fichero informático, es un objeto de computadora que almacena datos o comandos bajo una estructura propia de lectura e interpretación de un programa., 14

asignación
quiere decir que se asocia simbólicamente un valor específico a un nombre. Por ejemplo, A <- 30, quiere decir que a la letra A se le asigna el valor 30., 46

B

binario
es un sistema numérico que solo utiliza dos dígitos 0 y 1. Sobre este sistema se construye la tecnología digital. Un bit (abreviatura de dígito binario) es la mínima unidad de dato en una computadora, cada bit tiene un único valor de 0 o 1. Los programas que se ejecutan en la computadora se identifican como archivos binarios., 48

C

código fuente
"Es un programa escrito en un lenguaje de alto nivel que se convierte en código de objeto o código de máquina por un compilador antes de su ejecución" (BusinessDictionary, 2018), 39

compilador
"es un programa que procesa sentencias escritas en un lenguaje de programación particular y las convierte en lenguaje de máquina o "código" que utiliza el procesador de una computadora" (techtarget, 2018), 43

constante
es un valor que no puede ser cambiado durante la ejecución del algoritmo., 13

D

dato
es la unidad mínima de representación de una variable que posee un atributo, los datos procesados de manera correcta se convierten en información., 43

E

expresión
"Es una combinación de constantes, variables o funciones, interpretada de acuerdo a las normas de precedencia y asociación para un lenguaje de programación en particular" (Wikipedia, 2018), 49

H

hardware
se refiere a los elementos físicos que conforman una computadora y todo aquello que le sea físicamente tangible y conectable., 21

I

inicializar una variable
es asignar el valor que tomará la variable en su primera ejecución., 69

L

lenguaje de programación
"Es un lenguaje artificial utilizado para escribir instrucciones que pueden traducirse al lenguaje de máquina y luego ejecutarse por una computadora" (American Heritage®, 2016), 13

M

memoria RAM
es un tipo de memoria de la computadora que almacena de forma temporal los datos que necesitan los programas para funcionar correctamente., 46

P

palabra reservada
: “Es un término o frase apropiada para un uso específico, que no podrá ser usada para la declaración de variables. Por ejemplo, la palabra reservada Escribir es una función en varios lenguajes de programación para mostrar un texto en pantalla” (Computer Hope, 2018), 39

S

software
: es el componente no visible del sistema informático, con él se puede interactuar con los componentes físicos de la computadora. El software permite comunicarse con teléfonos inteligentes, computadores, reproductores de música y otros dispositivos., 5

V

variable
: es un nombre simbólico para designar información, se llama así debido a que la información representada puede cambiar debido al flujo de datos o las operaciones que sobre ella se asignen., 48

Bibliografía

American Heritage®. (2016). *Dictionary of the English Language.* Houghton Mifflin Harcourt Publishing Company.

BusinessDictionary. (15 de Diciembre de 2018). Obtenido de BusinessDictionary.com: http://www.businessdictionary.com/definition/source-code.html

Cairó Battistutti, O. (2005). *Metodología de la programación* (Tercera ed.). México: Alfaomega.

Computer Hope. (08 de ocutbre de 2018). *Computer Hope*. Obtenido de https://www.computerhope.com/jargon/r/reseword.htm

Garzón, J., Jiménez, G., & Cifuentes, X. (2016). Poligonación topográfica de alta precisión en el campus de la Universidad del Quindío. *Entre Ciencia e Ingeniería*, 50-60.

Herrera Morales, J. O., Gutierrez Posada, J. E., & Pulgarín Giraldo, R. (2017). *Introducción a la lógica de programación.* Armenia: elizcom s.a.s.

Hinojosa Gutiérrez, Á. P. (2016). *Python, paso a paso.* Bogotá: Ediciones de la U.

Jiménez Cleves, G., Garzón Barrero, J., & Londoño Pinilla, D. F. (2017). *Introducción a la altimetría.* Armenia: elizcom sas.

Joyanes Aguilar, L. (2007). *Fundmentos de programación: algoritmos, estructuras de datos y objetos.* McGraw-Hill interamericana.

Juganaru Mathieu, M. (2014). *Introducción a la programación.* Grupo editorial patria.

Serna, E., & Polo, J. (2014). Lógica y abstracción en la formación de ingenieros: una relación necesaria. *Ingeniería Investigación y Tecnología*, 299-310.

Shah, I. (Ed.). (1967). *Tales of the Dervishes.* Barcelona, Buenos Aires, México: Paidós ibérica.

techtarget. (14 de ocubre de 2018). *https://whatis.techtarget.com/*. Obtenido de https://whatis.techtarget.com/definition/compiler

Trejos Buriticá, O. I. (2017). *Lógica de programación.* Ediciones de la U.

Villalobos, J. A., & Casallas G., R. (2006). *Fundamentos de programación.* Pearson Educación de México S.A. de C.V.

Wikipedia. (15 de Diciembre de 2018). *Expresión (informática)*. Obtenido de Wikipedia, la enciclopedia libre: https://es.wikipedia.org/wiki/Expresi%C3%B3n_(inform%C3%A1tica)

Este libro se terminó de imprimir
En los talleres de Elizcom SAS
En el mes de diciembre 2018

ISBN: 978-958-8801-78-0