Algoritmos Computacionales

Introducción a la programación

Contenido

I. Introducción a la programación

1.1 Definición de algoritmo

Un algoritmo es una secuencia de pasos lógicos necesarios para llevar a cabo una tarea especifica, como la solución de un problema. Los algoritmos son independientes tanto del lenguaje de programación en que se expresan como de la computadora que los ejecuta. En cada problema el algoritmo se puede expresar en un lenguaje diferente de programación y ejecutarse en una computadora distinta; sin embargo el algoritmo será siempre el mismo.

Por ejemplo en una analogía con la vida diaria, una receta de un plato de cocina se puede expresar en español, ingles o francés, pero cualquiera que sea el lenguaje, los pasos para la elaboración del plato se realizaran sin importar el cocinero.

Los pasos a seguir en la solución de una ecuación de segundo grado.

Los pasos matemáticos para la solución de un número factorial.

Las instrucciones para la liquidación de una nomina.

Las acciones que se deben seguir para la obtención de una estadística.

Para llegar a la realización de un programa es necesario el diseño previo de un algoritmo, de modo que sin algoritmo no puede existir un programa.

Características de los algoritmos

Las características fundamentales que debe cumplir todo algoritmo son:

- Un algoritmo debe ser preciso e indicar el orden de realización de cada paso.
- Un algoritmo debe estar definido. Si se sigue un algoritmo dos veces, se debe obtener el mismo resultado cada vez.
- Un algoritmo debe ser finito. Si se sigue un algoritmo, se debe terminar en algún momento; o sea debe de tener un número finito de pasos.

La definición de un algoritmo debe describir tres partes: Entrada, Proceso y Salida.

En el algoritmo citado anteriormente se tendrá:

Entrada ingredientes y utensilios empleados

Proceso elaboración de la receta de cocina

Salida terminación del plato (por ejemplo, cordero)

Un algoritmo exige que se tengan varias propiedades importantes:

Los pasos de un algoritmo deben ser simples y exentos de ambigüedades (diferentes significados), deben seguir un orden cuidadosamente prescrito, deben ser efectivos y deben de resolver el problema en un número finito de pasos.

El siguiente ejemplo muestra un algoritmo para cambiar un foco quemado.

Cambiar un foco quemado podría resumirse en dos pasos:

1. Quitar el foco quemado
2. Colocar un foco nuevo

Pero, si tuviera que entrenar un robot domestico para que efectúe esta tarea, tendrá que ser mas especifico y claro en los pasos a seguir, dar mas detalles (suponga que el foco se encuentra en el techo de una habitación):

1. Situar escalera bajo el foco quemado.
2. Elegir un foco de reemplazo (de la misma potencia que el anterior).
3. Subir por la escalera hasta alcanzar el foco.

4. Girar el foco contra las manecillas del reloj hasta que esté suelto.
5. Ubicar el foco nuevo en el mismo lugar que el anterior.
6. Enroscar en el sentido de las manecillas del reloj hasta que quede apretado.
7. Bajar de la escalera.

1.2 Lenguajes de programación (Lenguaje máquina, ensamblador y de alto nivel)

Al igual que los idiomas sirven de vehículo de comunicación entre seres humanos, existen lenguajes que realizan la comunicación entre ellos y las computadoras. Estos lenguajes permiten expresar las instrucciones que el programador desea que la computadora ejecute.

Los principales tipos de lenguajes utilizados en la actualidad son tres:

- Lenguaje maquina
- Lenguaje de bajo nivel (ensamblador)
- Lenguajes de alto nivel

Lenguajes máquina

Se llama lenguaje máquina a las instrucciones que se dan directamente a la computadora, utilizando una serie de dígitos binarios o bits, representados por los números 0 y 1 que especifican una operación. Aunque este lenguaje es el que entiende la computadora, es muy difícil de manejar en la comunicación humana. Las instrucciones en lenguaje maquina dependen del hardware de la computadora y, por lo tanto, diferirán de una computadora a otra.

Lenguajes de bajo nivel (ensamblador)

Los lenguajes de bajo nivel son más fáciles de utilizar que los lenguajes máquina, pero, al igual que ellos, dependen de la máquina en particular. El lenguaje de bajo nivel por excelencia es el ensamblador (assembler lenguaje). Las instrucciones en lenguaje ensamblador son conocidas como mnemotécnicos.

Por ejemplo, mnemotécnicos típicos de operaciones aritméticas son:

en ingles, ADD, SUB, DIV, etc.

en español, SUM,RES,DIV, etc.

Una instrucción típica de suma seria:

ADD M, N, P

Esta instrucción podría significar "sumar el número contenido en la posición de memoria M al número almacenado en la posición de memoria N y situar el resultado en la posición de memoria P". Evidentemente es mucho más sencillo recordar la instrucción anterior con un mnemotécnico que su equivalente en código máquina.

0110 1001 1010 1011

Un programa escrito en lenguaje ensamblador no puede ser ejecutado directamente por la computadora (en esto se diferencia esencialmente del lenguaje máquina) sino que requiere una fase de traducción al lenguaje máquina.

El programa original escrito en lenguaje ensamblador se denomina programa fuente y el programa traducido en lenguaje maquina se conoce como programa objeto, ya directamente entendible por la computadora.

El traductor de programas fuente a objeto es un programa llamado ensamblador (assembler), existente en casi todas las computadoras.

NOTA: No se debe confundir el programa ensamblador, encargado de efectuar la traducción del programa fuente escrito a lenguaje maquina, con el lenguaje ensamblador (assembly language), lenguaje de programación con una estructura y gramática definidas.

Los lenguajes ensambladores presentan la ventaja frente a los lenguajes maquina de su mayor facilidad de codificación y, en general, su velocidad de cálculo.

Los inconvenientes más notables de los lenguajes ensambladores son:

- Dependencia total de la maquina lo que impide la transportabilidad de los programas (posibilidad de ejecutar un programa en diferentes maquinas).
- La formación de los programadores es más compleja que la correspondiente a los programadores de alto nivel, ya que exige no sólo las técnicas de programación, sino también el conocimiento del interior de la máquina.
- Hoy día los lenguajes ensambladores tienen sus aplicaciones muy reducidas en la programación de aplicaciones y se centran en aplicaciones de tiempo real, control de procesos y de dispositivos electrónicos, etc.

Lenguajes de alto nivel

Los lenguajes de alto nivel son los mas utilizados por los programadores. Están diseñados para que las personas escriban y entiendan los programas de un modo mucho más fácil que los lenguajes máquina y ensambladores. Otra razón es que un programa escrito en un lenguaje de alto nivel es independiente de la máquina; esto es, las instrucciones del programa de la computadora no

dependen del diseño del hardware o de una computadora en particular. En consecuencia, los programas escritos en lenguajes de alto nivel son portables o transportables, lo que significa la posibilidad de poder ser ejecutados con poca o ninguna modificación en diferentes tipos de computadoras; al contrario que los programas en lenguaje máquina o ensamblador que sólo se pueden ejecutar en un determinado tipo de computadora.

Los lenguajes de alto nivel presentan las siguientes ventajas:

- El tiempo de formación de los programadores es relativamente corto comparado con otros lenguajes.
- La escritura de programas se basa en reglas sintácticas similares a los lenguajes humanos. Nombres de las instrucciones tales como READ, WRITE, PRINT, OPEN, etc. Las modificaciones y puestas a punto de los programas son más fáciles.
- Reducción del coste de los programas.
- Transportabilidad.

Los inconvenientes se concretan en:

- Incremento del tiempo de puesta a punto al necesitarse diferentes traducciones del programa fuente para conseguir el programa definitivo.
- No se aprovechan los recursos internos de la máquina que se explotan mucho mejor en lenguajes máquina y ensambladores.
- Aumento de la ocupación de memoria.
- El tiempo de ejecución de los programas es mucho mayor.

Al igual que pasa con los lenguajes ensambladores, los programas fuente tienen que ser traducidos por programas traductores, llamados compiladores e interpretes.

Los lenguajes de programación de alto nivel existentes en la actualidad son muy numerosos, aunque la práctica demuestra que su uso mayoritario se reduce a BASIC, COBOL, PASCAL, C, C++,... y en el campo de la primera enseñanza a LOGO, PILOT...

1.3 Traductores de lenguaje

Los traductores de lenguajes son programas que traducen a su vez los programas fuente escritos en lenguajes de alto nivel a código máquina.

Los traductores se dividen en:

- Compiladores
- Interpretes

Interpretes

Un interprete es un traductor que toma un programa fuente, lo traduce y a continuación lo ejecuta (dicho programa por medio de la computadora desarrolla una tarea especifica).

Un lenguaje que soporte un traductor de tipo intérprete se denomina lenguaje interpretado. BASIC es el modelo por excelencia interpretado.

Los programas fuente en BASIC se escriben con ayuda de un programa denominado editor que suele venir incorporado al programa intérprete.

Compiladores

Un compilador es un programa que traduce los programas fuente escritos en lenguajes de alto nivel a lenguaje máquina.

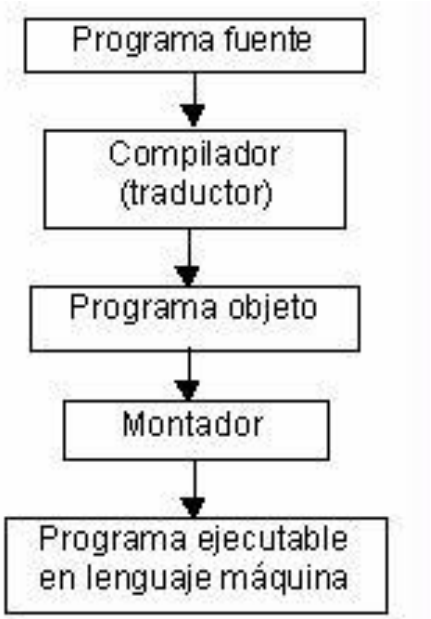
Los programas escritos en lenguajes de alto nivel (en el editor del lenguaje) se llaman programas fuente y el programa traducido programa objeto o código objeto. El compilador traduce (sentencia a sentencia) el programa fuente.

Lenguajes compiladores típicos son: PASCAL, COBOL, C..

Fases de la compilación

La compilación es el proceso de la traducción de programas fuente a programas objeto.

El programa objeto obtenido de la compilación no ha sido traducido normalmente a código máquina sino a ensamblador. Para conseguir el programa máquina real se debe utilizar un programa llamado montador o enlazador (linker). El proceso de montaje conduce a un programa en lenguaje máquina directamente ejecutable:

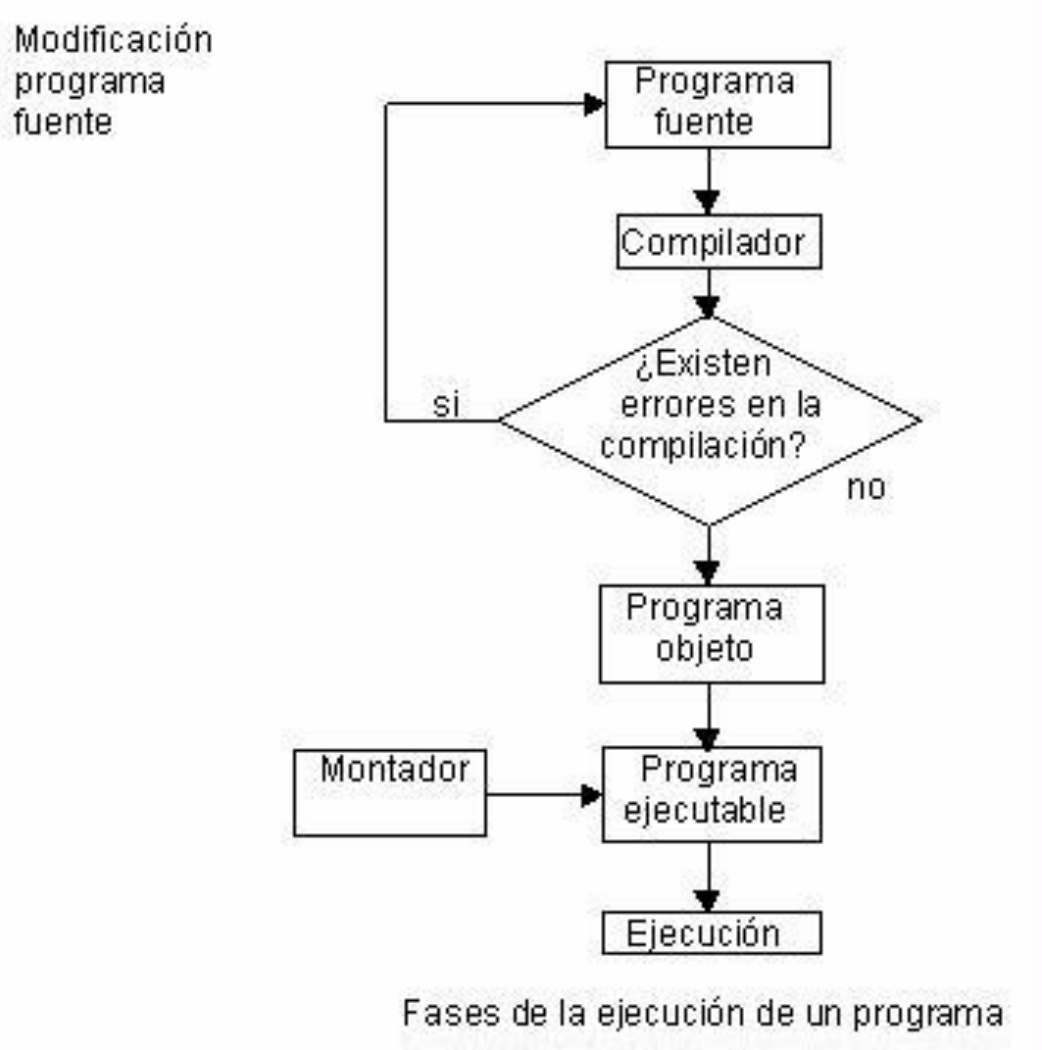


Fases de la compilación

Por ejemplo:

El proceso de ejecución de un Programa en C++ tiene los siguientes pasos:

1. Escritura del programa fuente con un editor (programa que permite a una computadora actuar de modo similar a una máquina de escribir electrónica) y guardarlo en un dispositivo de almacenamiento (un disco).
2. Introducir el programa fuente en memoria.
3. Compilar el programa con el compilador C++.
4. Verificar y corregir errores de compilación (listado de errores).
5. Obtención del programa objeto.
6. El montador obtiene el programa ejecutable.
7. Se ejecuta el programa y si no existen errores, se tendrá la salida del mismo.



Fases de la ejecución de un programa

1.4 Definición de programa

Un programa de computadora es un conjunto de instrucciones (ordenes dadas a la máquina) que producirán la ejecución de una determinada tarea. En esencia, **un programa es un medio para conseguir un fin**. El fin será normalmente definido como la información necesaria para solucionar un problema.

El proceso de programación es, por consiguiente, un proceso de solución de problemas (como ya se vio anteriormente) y el desarrollo de un programa requiere las siguientes fases:

1. Definición y análisis del problema.
2. Diseño de algoritmos.

   - diagrama de flujo;

   - pseudocódigo.

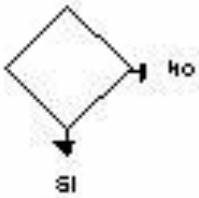
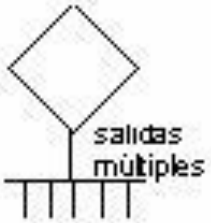
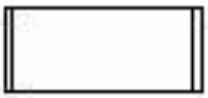
3. Codificación del programa.
4. Depuración y verificación del programa.
5. Documentación.
6. Mantenimiento.

En este curso el objetivo fundamental son las fases 1 y 2.

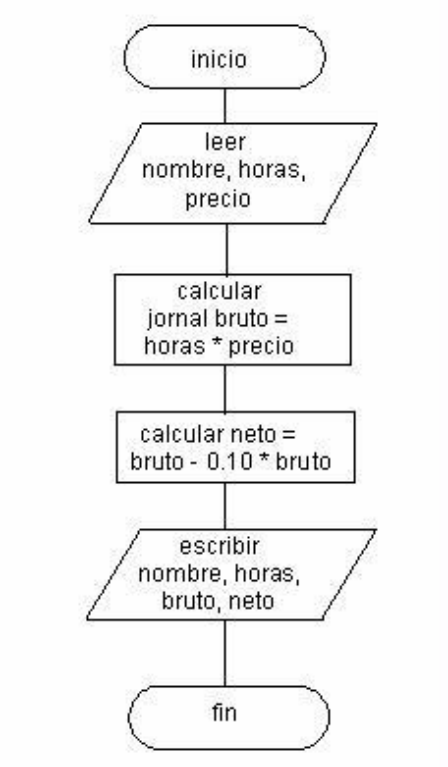
1.5 Diagrama de flujo (*Representación gráfica*)

Un diagrama de flujo (flowchart) es una de las técnicas de representación de algoritmos más antigua y a la vez más utilizada, aunque su empleo ha disminuido considerablemente, sobre todo desde la aparición de lenguajes de programación estructurados. Un diagrama de flujo es un diagrama que utiliza los símbolos (cajas) y que tiene los pasos del algoritmo escritos en esas cajas unidas por flechas, denominadas líneas de flujo, que indican la secuencia en que se deben ejecutar.

Los símbolos estándar normalizados por ANSI (abreviatura de American National Standards Institute) son muy variados, aquí se presentan algunos:

Símbolos principales	Función
	Terminal (Representa el "inicio" y el "fin" de un programa. Puede representar también una interrupción que sea necesario realizar en un programa).
	Entrada/Salida (Cualquier tipo de introducción de datos en la memoria desde los periféricos, "entrada", o registro de la información procesada en un periférico, "salida").
	Proceso (Cualquier tipo de operación que pueda originar cambio de valor, formato o posicion de la informacion almacenada en memoria, operaciones aritméticas, de transferencia, etc.).
	Decisión (Indica operaciones lógicas o de comparación entre datos (normalmente dos) y en funcion del resultado de la misma determina cuál de los distintos caminos alternativos del programa se debe seguir, generalmente tiene dos salidas(respuestas Si o No), pero puede tener tres o más segun los casos).
	Decisión múltiple (En función del resultado de la comparación se seguirá uno de los diferentes caminos de acuerdo con dicho resultado).
	Conector (Sirve para enlazar dos partes cualesquiera de un ordinograma a traves de un conector en la salida y otro conector en la entrada. Se refiere a la conexión en la misma página del diagrama).
	Indicador de direccion o linea de flujo (Indica el sentido de ejecución de las operaciones).
	Línea conectora (Sirve de unión entre dos símbolos).
	Conector (Conexion entre dos puntos del organigrama situado en páginas diferentes).
	Llamada a subrutina o a un proceso predeterminado (Una subrutina es un modulo independiente del programa principal, recibe una entrada procedente de dicho programa, realiza una tarea determinada y al terminar regresa al programa principal).

Ejemplo básico de un diagrama de flujo:



El diagrama citado representa la resolución de un programa que deduce el salario neto de un trabajador a partir de la lectura del nombre, horas trabajadas, precio de la hora, y sabiendo que los impuestos aplicados son el 10 % sobre el salario bruto.

1.6 Pseudocódigo

El pseudocódigo es un lenguaje de especificación de algoritmos.

En sí es una mezcla de lenguaje de programación y de lenguaje natural. La idea del pseudocódigo consiste en aprovechar la flexibilidad y poder expresivo del lenguaje natural por un lado, y las reglas de composición de los lenguajes de programación de alto nivel por el otro.

El pseudocódigo utiliza para representar las acciones sucesivas palabras reservadas en inglés - similares a sus homónimas en los lenguajes de programación -, tales como **start, end, stop, if-then-else, while** etc. La escritura del pseudocódigo exige normalmente la indentación (sangría en el margen izquierdo) de diferentes líneas. La representación en pseudocódigo del diagrama de flujo del ejemplo anterior sería:

```
Start

        {cálculo de impuesto y salario}

        read nombre, hora, precio_hora

        salario_bruto ← horas * precio_hora

        tasa ← 0.1 *salario_bruto

        salario_neto ← salario_bruto - tasa

        write nombre, salario_bruto, tasa, salario_neto

end
```

El algoritmo comienza con la palabra **start** y finaliza con la palabra **end**, en inglés (en español, **inicio** y **fin**). Entre estas palabras, sólo se escribe una instrucción o acción por línea.

La línea encerrada entre llaves { ... } se denomina comentario. Es una información al lector del programa y no realiza ninguna instrucción ejecutable; sólo tiene efecto de documentación interna del programa. Algunos autores suelen utilizar corchetes en lugar de llaves [...].

1.7 Lenguaje algorítmico

El lenguaje algorítmico extrae las mejores características de los dos enfoques anteriores y los combina en un lenguaje especial para expresar algoritmos. Del pseudocódigo se tomó la facilidad de descripción de la prosa, al que se agrego lo conciso del diagrama de flujo.

La traducción del lenguaje algorítmico a un lenguaje de programación debe resultar sencilla, sin importar cual se utilice. La decisión final sobre el lenguaje de programación a emplear depende de muchos factores, como la naturaleza de la aplicación particular y las características del lenguaje.

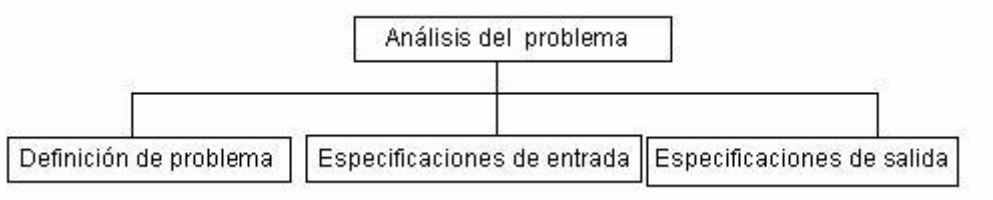
1.8 Metodología de solución

La principal razón para que las personas aprendan a programar en general y los lenguajes de programación en particular es utilizar la computadora como una herramienta para la resolución de problemas ayudado por una computadora. La resolución de un problema consta de ocho etapas:

1. Definición y delimitación del problema a solucionar (enunciado del problema)
2. Pseudocódigo o diagrama de flujo (algoritmo)
3. Prueba de escritorio
4. Codificación
5. Digitación
6. Compilación o interpretación del programa
7. Ejecución del Programa
8. Evaluación de los resultados

1. - Definición y delimitación del problema a solucionar

El problema debe estar bien definido si se desea llegar a una solución satisfactoria para poder definir con precisión el problema se requiere que las especificaciones de entrada y salida sean descritas con detalle. Una buena definición del problema, junto con una descripción detallada de las especificaciones de entrada y salida, son los requisitos más importantes para llegar a una solución eficaz.



El análisis del problema exige una lectura previa del problema a fin de obtener una idea general de lo que se solicita. La segunda lectura deberá servir para responder a las preguntas:

- ¿Qué información debe proporcionar la resolución del problema?
- ¿Qué datos se necesitan para resolver el problema?

La respuesta a la primera pregunta indicará los resultados deseados o las salidas del problema. La respuesta a la segunda indicará qué datos se proporcionan o las entradas del problema.

2. - Pseudocódigo o diagrama de flujo (algoritmo)

Una computadora no tiene la capacidad para solucionar problemas más que cuando se le proporcionan los sucesivos pasos a realizar. Estos pasos sucesivos que indican las instrucciones a ejecutar por la máquina constituyen, como ya conocemos, el algoritmo.

En esta etapa es donde se determinan los pasos o instrucciones que deben llevarse a cabo y el orden lógico de su ejecución para dar una eficiente solución al problema.

La información proporcionada al algoritmo constituye su entrada y la información producida por el algoritmo constituye su salida.

3. - Prueba de escritorio (prueba de un algoritmo)

Para comprobar que un algoritmo realiza la tarea para la cual fue diseñado, debe ejecutarse a mano. Para esto deben utilizarse datos representativos y anotarse los valores que toman las variables en cada paso. Esto se conoce como **corrida de escritorio**.

4. - Codificación

El programa que implementa el algoritmo debe ser escrito en un lenguaje de programación y siguiendo las reglas gramaticales o sintaxis del mismo. La fase de conversión del algoritmo en un lenguaje de programación se denomina codificación, ya que el algoritmo escrito en un lenguaje especifico de programación (lenguaje de alto nivel) se denomina código.

5. - Digitación

Tras la codificación del programa las instrucciones se convierten a un medio legible para la computadora; a igual procedimiento se someten los datos (en disquetes, cassettes, cintas, etc.), utilizando dispositivos como digitadoras o consolas.

6. - Compilación o interpretación del programa

En esta etapa la computadora chequea si todas las instrucciones están escritas correctamente desde el punto de vista de la sintaxis y gramática de cada lenguaje y las transcribe, dentro de la memoria, del lenguaje de alto nivel al lenguaje máquina para obtener el llamado programa objeto.

7. - Ejecución del Programa

El programa objeto es ejecutado por la computadora para llegar a los resultados esperados, utilizando los dispositivos, unidades y memoria necesaria, según cada caso o programa.

8. - Evaluación de los resultados

Obtenidos los resultados se les evalúa para verificar que sean correctos. En caso contrario, se revisa en las etapas anteriores para detectar la falla o error, entrar a corregirla y reiniciar desde este punto los pasos para resolver de nuevo y en forma correcta el problema.

1.9 Prueba de un algoritmo (Ejemplos)

Ejemplo No. 1

Diseñe un algoritmo que dadas 4 calificaciones (Cal1, Cal2, Cal3, Cal4), calcule la calificación promedio y escriba el resultado final junto con un mensaje explicativo.

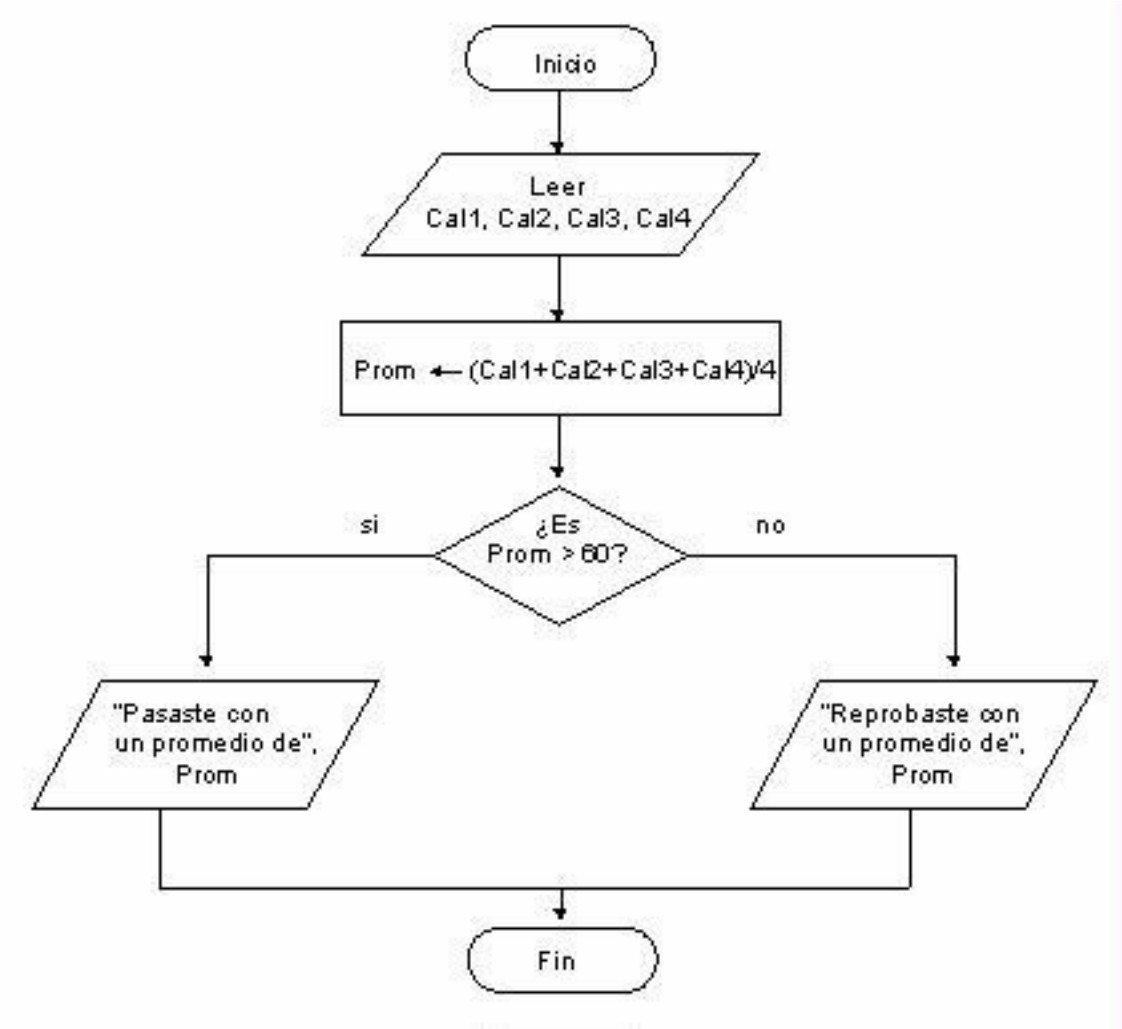
Análisis del problema

Entrada: Leer las calificaciones individuales

Proceso: Calcular la calificación promedio

Salida: Escribir resultado con mensaje explicativo

Diagrama de flujo



Pseudocódigo

Inicio

{Calculo del promedio de calificaciones}

Leer Cal1, Cal2, Cal3, Cal4

Prom ← (Cal1+Cal2+Cal3+Cal4)/4

Escribir ("Resultado final es", Prom)

Fin

Ejemplo No. 2

En cierto curso de computación, la calificación final del estudiante se determina a partir de su rendimiento en tres aspectos del trabajo anual. Existe una calificación de la mitad del curso, que cuenta un 30% del total; el trabajo de laboratorio que lleva una calificación, cuenta un 20% del total; y el examen final, cuenta el 50% restante.

Diseñe un algoritmo que, dadas las calificaciones individuales, calcule la calificación final, escriba la salida final proporcionando el nombre del estudiante, sus calificaciones individuales y la calificación final. La variable se llama Final.

Análisis del problema

Calcular la calificación final a partir de las calificaciones parciales

Entrada: Lectura del nombre del estudiante, calificación del laboratorio, calificación de mitad del curso, calificación del examen final.

Salida: Nombre del estudiante, las tres calificaciones obtenidas y el promedio final, con comentarios correspondientes.

Solución del problema

1. Lectura del nombre del alumno y de sus calificaciones en cada una de las partes
2. Calculo e impresión de la calificación final junto con el nombre del alumno

Variables:

Nombre (para nombre del alumno)

Cal_Lab (para la calificación del laboratorio)

Cal_mitcurso (para la calificación de mitad del curso)

Cal_ExaFin (para la calificación del examen final)

Final (para la calificación final)

1. - Obtener los datos de entrada

Leer (Cal_Lab, Cal_mitcurso, Cal_ExaFin)

2. - Calcular la calificación final

Final ← 0.20*Cal_Lab+0.30*Cal_mitcurso+0.50*Cal_ExaFin

3. - Imprimir los resultados

4. - Termino

Algoritmo

Inicio

Leer (Cal_Lab, Cal_mitcurso, Cal_ExaFin)

Final ← 0.20*Cal_Lab+0.30*Cal_mitcurso+0.50*Cal_ExaFin

Escribir ("Nombre del alumno", Nombre)

Escribir ("Calificación de laboratorio", Cal_Lab)

Escribir ("Calificación de mitad del curso", Cal_mitcurso)

Escribir ("Calificación del examen final", Cal_ExaFin)

Escribir ("Calificación final", Final)

Fin

Del ejemplo anterior suponga que se dan los siguientes valores:

Alberto Rodríguez, 72, 68, 65

Solución

0.20*72 = 14.4

0.30*68 = 20.4

0.50*65 = 32.5

suma = 67.3

Resultado

Nombre del alumno Alberto Rodríguez

Calificación de laboratorio 72

Calificación de mitad del curso 68

Calificación del examen final 65

Calificación final 67.3

Andrés Miramontes, 75, 75, 75

Solución

0.20*75 = 15.0

0.30*75 = 22.5

0.50*75 = 37.5

suma = 75.0

Resultado

Nombre del alumno Andrés Miramontes

Calificación de laboratorio 75

Calificación de mitad del curso 75

Calificación del examen final 75

Calificación final 75.0

Ejemplo No. 3

Dados tres números, determine si la suma de cualquier pareja de ellos es igual al tercer número. Si se cumple esta condición escribir "Iguales", y en caso contrario, escribir "Distintas". (Diagrama de flujo y pseudocódigo)

Solución

Suponga que los números son:

3 9 6

la respuesta es "Iguales", ya que 3+6=9.

Pero si los números fueran:

2 3 4

El resultado será "Distintas".

Algoritmo de resolución:

1. - Leer los tres valores, A, B y C.

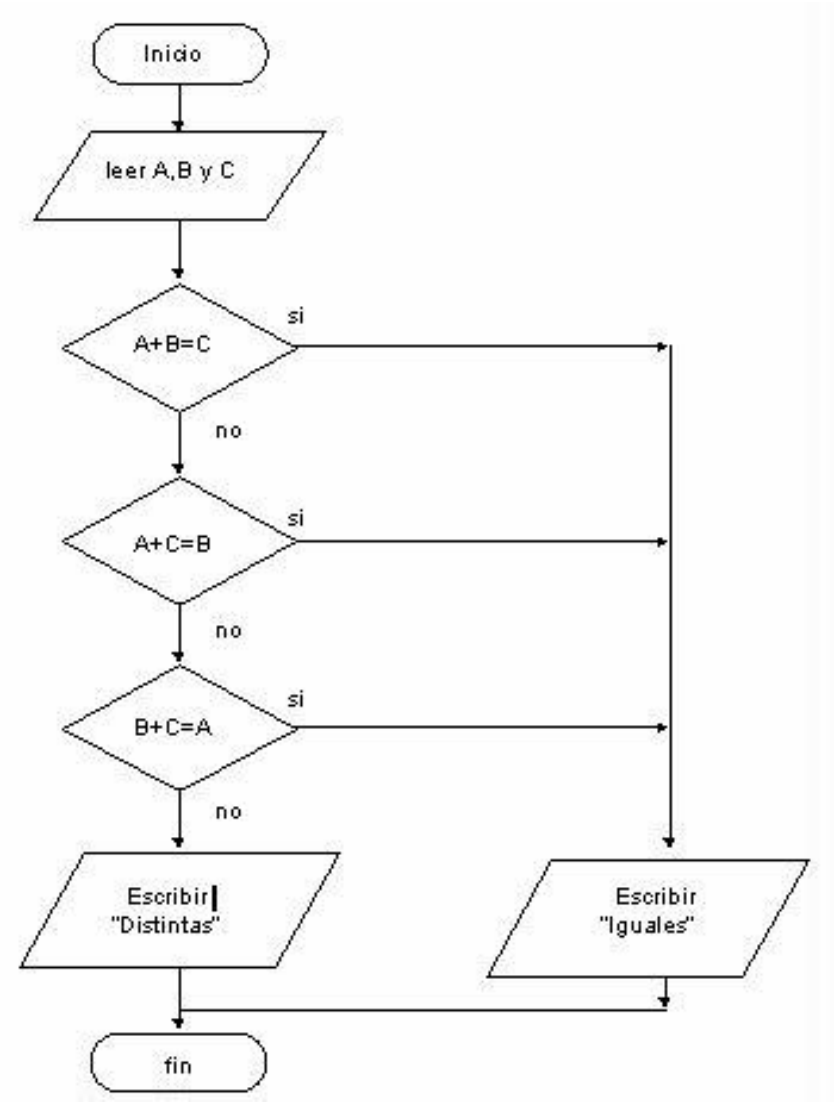
2. - Si A+B=C escribir "Iguales" y parar.

3. - Si A+C=B escribir "Iguales" y parar.

4. - Si B+C=A escribir "Iguales" y parar.

5. - Escribir "Distintas" y parar.

Diagrama de flujo:



Ejemplo No. 4

Escribir un algoritmo para calcular el área de un triángulo dada la base y la altura.

Solución:

Análisis

La fórmula geométrica del área o superficie de un triángulo es:

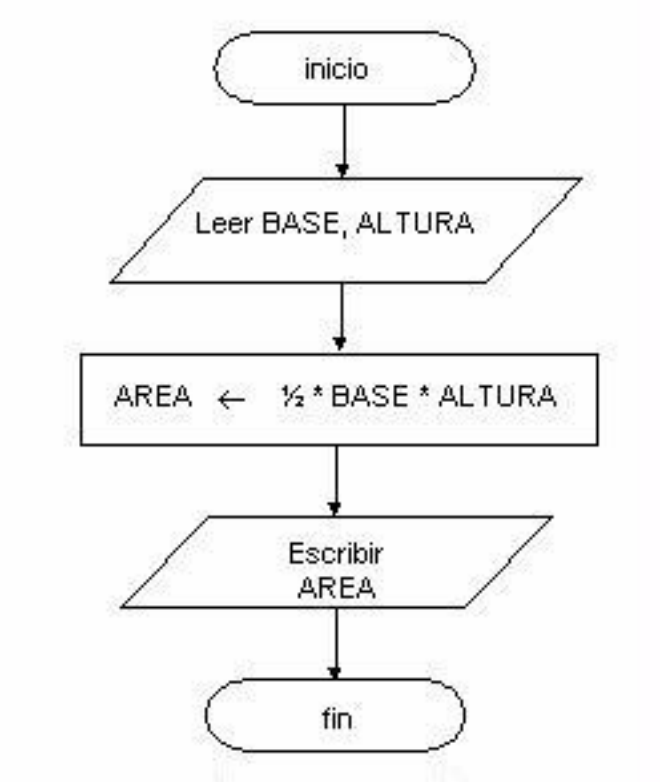
A = (1/2)B*H B = base H = altura

variables: Base, Altura

Suponga que B = 4.5 H = 7.2

A = (1/2) 4.5 * 7.2 = 16.2

Diagrama de flujo



Tomado de http://correo.uan.edu.mx/~iavalos/introprog.htm#Ejemplos