

I. INTRODUCCIÓN A LA PROGRAMACIÓN

1.1 Definición

1.1.1. Algoritmo

1.1.2. Programas

1.2. Etapas para la Resolución de Problemas por computadora

1.2.1. Definición del problema

1.2.2. Análisis y Diseño del Problema

1.2.3. Programación

1.2.3.1. Algoritmo

1.2.3.2. Prueba de escritorio

1.2.3.3. Codificación

1.2.3.4. Compilación/Ejecución

1.2.3.4.1. Tipos de Errores

1.2.4. Documentación

I. INTRODUCCION A LA PROGRAMACION

Encontrar solución a un problema y convertirlo en un programa de computación es una actividad compleja relacionada con una terminología precisa y particular.

Por ello, antes de iniciar el aprendizaje de la programación, es necesario conocer los conceptos implicados en la resolución de un problema mediante una computadora.

A continuación se presentan un conjunto de conceptos involucrados en el proceso de resolución de problemas a través de un computador.

1.1 Definiciones

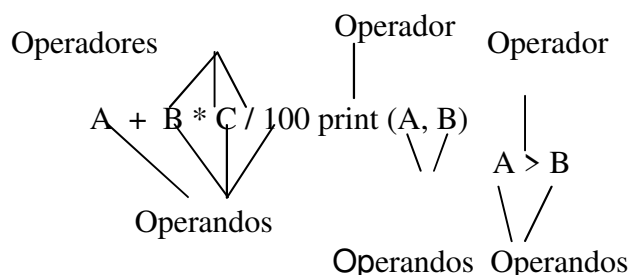
1.1.1 Programa

Es una secuencia lógica de instrucciones escritas en un determinado lenguaje de programación, que establece las operaciones que van a ser realizadas por la computadora.

En esta definición encontramos dos elementos nuevos.

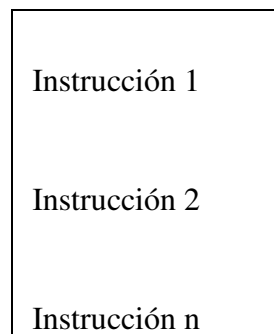
1. **Instrucción:** Orden que se le dá al computador. Está compuesta de dos partes: operando y el operador. El operador indica el tipo de operación que se va a realizar sobre los datos y el operando es el conjunto de valores con los cuales va a trabajar el operador.

Ejemplos:

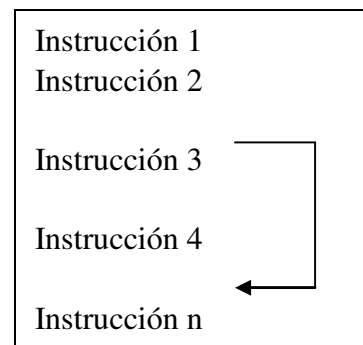


Secuencia lógica: se refiere al orden de ejecución de cada instrucción en el programa.

2.



(a) Secuencia lógica lineal



(b) Secuencia lógica no lineal

Fig. #1.1. Secuencia lógica de un programa
Luis Joyanes Aguilar(1998), Fundamentos de Programación

Secuencia lógica lineal : las instrucciones se ejecutan en el orden en que aparecen, sin bifurcaciones, decisiones ni instrucciones repetitivas. Ver fig. # 1.1. (a)

Secuencia lógica no lineal: es cuando se interrumpe la secuencia mediante instrucciones de bifurcación. Una bifurcación son los saltos que se dan dentro del programa y puede ser hacia delante o hacia atrás. Ver fig. # 1.1. (b)

a. Lenguaje de Programación

Es una notación que maneja símbolos y reglas que permiten escribir programas. Todo lenguaje de programación está compuesto por su sintaxis (reglas) y su semántica (significado de sus símbolos y palabras utilizadas) y es a través de los lenguajes de programación que se logra una comunicación con el computador.

b. Tipos de Lenguaje.

1. Lenguaje Absoluto o de máquina.

Es el lenguaje nativo de una CPU. Son aquellos cuyas instrucciones son directamente entendibles por la computadora. Sus instrucciones se expresan en términos de la unidad de memoria más pequeña, el bit (código binario 1 ó bien 0). En esencia, una secuencia de bits que especifican la operación y las celdas de memoria implicadas en dicha operación.

Ejemplo :

Código de operación	operando
0101 1100	1100 0010 1100 0100
+	2 4

Es un lenguaje orientado a la máquina por lo tanto presenta las siguientes ventajas :

1. No necesita traducción,
2. Se aprovecha toda la capacidad del computador (uso de la memoria),
3. El tiempo de acceso es más rápido

Desventajas:

1. Difícil de escribir y entender,
2. Su elaboración toma mucho tiempo,
3. Por ser un lenguaje de unos y ceros, se pueden cometer errores.

2. Lenguaje Simbólico.

Son aquellos, en los cuales las instrucciones o sentencias son escritas con palabras similares a los lenguajes humanos. Están compuestos de símbolos , letras y números.

Ejemplo: `If (numero > 0) then printf "El numero es positivo"`

Es un lenguaje orientado al programador y presenta las siguientes ventajas:

1. Fácil de escribir y entender,
2. Disminuye la probabilidad de cometer errores,
3. El tiempo para hacerlo es menor ,
4. Son independientes de la máquina (transportabilidad)

Desventajas:

1. Si necesita traducción,
2. Su tiempo de ejecución es mayor.

c. Tipos de Programa

Programa fuente: es un programa escrito en un lenguaje de programación específico, preparado por el programador y que será suministrado a la máquina.

Ejemplos: Programas escritos en : C, C++, JAVA, PASCAL, Visual Basic.

Programa objeto: es un programa compuesto de unos y ceros, producto de la compilación de un programa Fuente. Es el programa que entiende la máquina.

ch. Procesadores de Lenguajes.

Son programas que traducen a los programas fuentes escritos en lenguajes de programación de alto nivel a código de máquinas. Los procesadores de lenguaje se dividen en:

1. Compilador:

Es un programa, suministrado por el fabricante del lenguaje, con el propósito de traducir el programa fuente a programa objeto. El compilador realiza las siguientes funciones:

- a. Traduce las instrucciones del programa fuente,
- b. Asigna áreas de memoria y dirección,
- c. Suministra constantes y otros datos,
- d. Produce un diagnóstico de errores,
- e. Produce el programa objeto.

Compilación: Es el proceso de traducción por la computadora de las instrucciones escritas por el programador (programa fuente) a instrucciones escritas en el lenguaje propio de la máquina (programa objeto), fig #.1.2. Como consecuencia de este proceso existen muchos compiladores, por ejemplo: compiladores para lenguajes tales como C++, C, Visual Basic, Pascal, etc.



fig #. 1.2. Proceso de Compilación

Todo programa fuente debe ser traducido a programa objeto para que pueda ser ejecutado por el computador.

2. Intérprete

Es un programa que va leyendo poco a poco el código que se ha escrito, programa fuente, y va traduciéndolo y ejecutándolo según se traduce. Aquí no hay una traducción completa ya que no genera un programa directamente ejecutable. Este permite al programador ejecutar interactivamente las diferentes secciones del programa, en vez de esperar a que todo el programa sea traducido y probado, como se hace con un compilador. Los lenguajes interpretados son menos eficientes que los compilados.

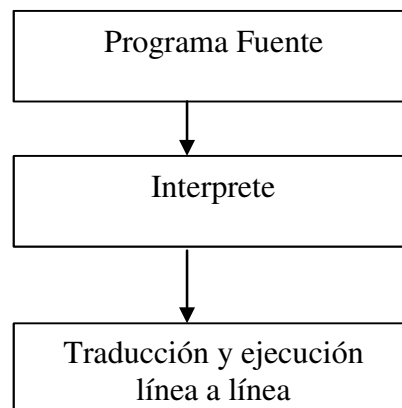


Fig #.1. 3 Intérprete

Luis

Joyanes Aguilar, Fundamentos de Programación

Ejemplos de Intérpretes: Lisp, Smalltalk , Prolog.

2.1 Lenguajes Interpretados.

Existen lenguajes que utilizan ambos conceptos, en particular el lenguaje JAVA aplica una aproximación intermedia entre éstas dos. Existe una “compilación” inicial donde el compilador de Java traduce el código fuente a bytecode, el cual es una representación de programa a bajo nivel, pero que hasta el momento no es entendido por ningún microprocesador. El bytecode

no es el programa objeto de ninguna CPU; éste sería el código intermedio que será posteriormente interpretado(leído y ejecutado) por la máquina .

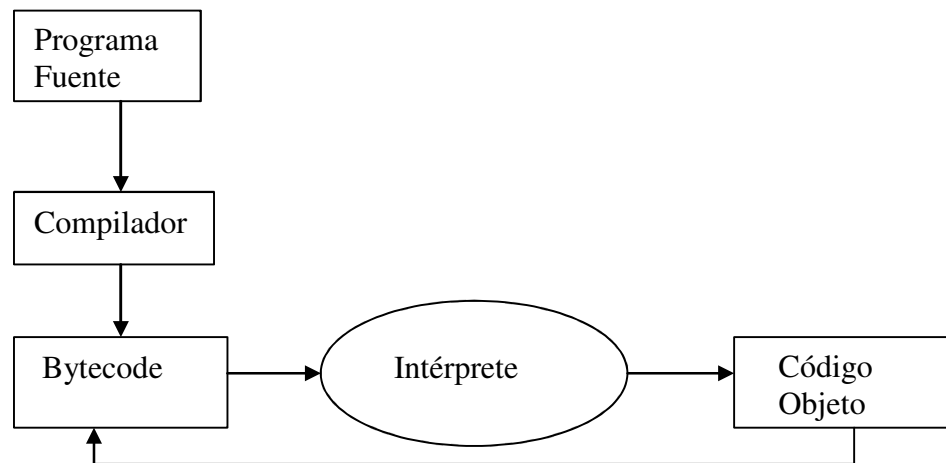


Fig # 1.4 Proceso de compilación e interpretación en Java

1.1.2 Qué es un algoritmo ?

Se denomina **algoritmo** a un **grupo finito de instrucciones organizadas de manera lógica y ordenada** que permite solucionar un determinado **problema**. Se trata de una serie de instrucciones o reglas establecidas que, por medio de una sucesión de pasos, permiten arribar a un resultado o solución.

Los algoritmos suelen estar asociados a muchos ámbitos, como en la matemática, vida diaria, informática, etc.

Ejemplos en el ámbito matemático:

- Averiguar el cociente entre un par de números.
- El máximo común divisor entre dos cifras.

Ejemplos en la vida diaria:

- Al cocinar, preparar un pastel de manzana.
- Un manual de instrucciones para el funcionamiento de un electrodoméstico.
- Un manual de instrucciones para armar un librero.
- Una serie de órdenes del jefe a un empleado para desarrollar una cierta tarea.

Ejemplos en la informática:

- La aplicación que permite escuchar la radio en el celular.
- La aplicación que permite usar la calculadora en el celular o la pc.
- La aplicación que permite chatear.

Esta amplitud de significados permite apreciar que no existe una definición formal y **única** de algoritmo. El término suele ser señalado como el **número fijo de pasos necesarios para transformar información de entrada (un problema) en una salida (su solución)**. Asimismo, tampoco podemos pasar por alto que los algoritmos se pueden expresar a través de:

- Lenguajes de programación: Visual Basic, Java, C, C++ Python.
- Pseudocódigo: Código que tiene sus propias reglas y sintaxis escritas en un lenguaje natural. Ejemplo: Leer (usuario, contraseña).
- Lenguaje natural: Español, inglés, etc.
- Diagramas de flujo: Conjunto de instrucciones representada gráficamente.

Cuando se plantean programas de computadoras, se utilizan los algoritmos para esquematizar los **pasos** de solución usando un lenguaje de especificaciones de algoritmos llamados pseudocódigo, **de algoritmos** que requieren menos precisión que un lenguaje de programación formal. De hecho, la computadora no puede ejecutar el pseudocódigo de un algoritmo: sólo **ayuda al programador a determinar cómo escribir la solución planteada en un lenguaje de programación**.

1.1.3 Estilos de escritura o convención de nombres

En **Programación** existen diferentes estilos de escrituras. Para esto se utilizan un conjunto de reglas para la elección de la secuencia de caracteres que se utilice para **identificadores** que denoten variables, **tipos**, **funciones** y otras entidades en el **código fuente** y la documentación.

Algunas de las razones para utilizar una convención de nombres son:

- reducir el esfuerzo necesario para leer y entender el código fuente;
- mejorar la apariencia del código fuente (por ejemplo, al no permitir nombres excesivamente largos o abreviaturas poco claras).
- Estandarizar entre empresas, programadores y lenguajes, sin embargo, esto es un tema polémico, porque cada grupo piensa que la suya es la mejor.

Entre ellas tenemos las siguientes convenciones **de nombres**:

- **Estilo Pascal (PascalCase)**
La primera letra del identificador y la primera letra de las siguientes palabras concatenadas están en mayúsculas, por ejemplo: ExamenEstudiante
- **Estilo camelCase**
La primera letra del identificador está en minúscula y la primera letra de las

siguientes palabras concatenadas en mayúscula, por ejemplo:
calcular_Salario

- **Estilo Mayúsculas (ALL_CAPS)**
Todas las letras del identificador se encuentran en mayúsculas, por ejemplo: **PI**
- **Estilo minúsculas (small_caps)**
Todas las letras del identificador se encuentran en minúsculas, por ejemplo: **salario**

1.2 Conceptos de paradigmas de programación

Un paradigma de programación es un modelo básico de **diseño** y desarrollo de programas, que permite producir programas con un conjunto de normas.

1.2.1 Programación estructurada

Es un paradigma de programación orientado a mejorar la claridad, calidad y tiempo de desarrollo de un programa de computadora, utilizando el concepto modular (divide y vencerás) a través de **funciones**, estructuras básicas de la programación: **secuencia** (ejecución de las instrucciones de **arriba** (top) hacia **abajo** (down)), **alternativa** (if y switch) y de **repetición** (bucles **for**, **while**, **repeat**), lo cual eliminó el uso de la instrucción **go to** que podría conducir a "código espagueti", que es mucho más difícil de seguir y de mantener, y era la causa de muchos errores de programación. Surgió en la década de 1960.

1.2.2 Programación orientada a objetos

La programación orientada a objetos se puede definir como una técnica o estilo de programación que utiliza objetos como bloques esenciales de construcción a partir de clases definidas por el programador. Los elementos básicos de la POO son: objetos, mensajes, métodos y clases. Considero las estructuras básicas de la programación estructurada (secuencia, alternativa y repetición).

La potencia real de los objetos está en sus propiedades: abstracción, herencia y polimorfismo. Por lo que se logra mayor seguridad en los datos y código. Y además permite la reutilización de código. Surgió en la década de 1970.

1.2.3 Programación orientada a eventos

La programación dirigida por eventos es un paradigma de programación en el que tanto la estructura como la ejecución de los programas van

determinados por los sucesos que ocurran en el sistema, definidos por el usuario o que ellos mismos provoquen. Para entender la programación dirigida por eventos, podemos oponerla a estructurada porque es el programador el que define cuál va a ser el flujo del programa, en la programación dirigida por eventos será el propio usuario o lo que sea que esté accionando el programa el que dirija el flujo del programa. Promovió aplicaciones con entornos gráficos. También consideró las estructuras básicas de la programación.

Surgió en las décadas de los 70 y 80.

1.3. Etapas para solución de un problema por computadora

El proceso de solución de un problema con una computadora conduce a la escritura de un programa y a su ejecución en la misma. Aunque el proceso de diseñar programas es esencialmente un proceso creativo, se pueden considerar una serie de etapas o pasos comunes, que generalmente deben seguir todos los programadores.

Las etapas para la solución de un problema por computadora son:

1.3.1 Definición del problema.

Describe en forma narrativa o esquemática de modo claro y concreto en un lenguaje corriente el problema que ha de resolverse. Presenta la procedencia y el aspecto de los datos a procesar y describe los resultados y cómo hay que presentarlos.

Ejemplo 1.1:

Definición o dominio del problema:

Leer dos lados de un triángulo y calcular e imprimir la hipotenusa del triángulo.

La fórmula para calcular la hipotenusa es $h = \sqrt{a^2 + b^2}$

1.3.2. Análisis y Diseño del problema.

El análisis y diseño se centra en la investigación del problema, buscando identificar y describir los elementos en el dominio del mismo. A partir de estos elementos, que surgen como consecuencia de los requisitos del problema, se definen sus datos, relaciones y procesos.

Basándonos en la definición del problema se debe identificar la procedencia de los datos (**Entrada**), los pasos y operaciones más importantes del proceso y el orden en que hay que realizarlos (**Proceso**) y los resultados (**Salida**) que serán requeridas por el **ANÁLISIS**.

Esto se puede obtener respondiendo a las siguientes preguntas y en el siguiente orden :

1. Que datos de entrada me proporciona la definición del problema? -estos pueden ser **datos variables (valores a leer)** o **constantes** (valores fijos).
2. Qué pide la definición como salida? -indicará que datos o información deseamos obtener para imprimirla o desplegarla.
3. Qué procesos debo realizar para obtener la salida? -indicará los cálculos, operaciones o decisiones que debemos realizar.

Es muy importante decir que el análisis lo harán con las preguntas en el orden indicado e identificando lo solicitado correspondientemente en el enunciado o definición del problema, sombreándola de los colores según cada pregunta de la siguiente forma:

Definición o dominio del problema:

↓ 1. Leer dos lados de un triángulo y calcular e imprimir la hipotenusa del triángulo.
 La fórmula para calcular la hipotenusa es $h = \sqrt{a^2 + b^2}$
 ↓ 3.

Una vez realizamos el análisis del problema entonces podemos definir los elementos del **diseño**, llenado el cuadro siguiente

Resultados extraídos de las preguntas realizadas durante el análisis

Leer dos lados de un triángulo
La fórmula para calcular la hipotenusa es $h = \sqrt{a^2 + b^2}$
imprimir la hipotenusa del triángulo

1.3.3. Programación:

DISEÑO

Entrada	a, b
Proceso	calcular h
Salida	h

Aunque suene contradictorio, programar es mucho más que escribir un programa.

No es suficiente escribir código en un lenguaje para resolver un problema y que éste funcione correctamente. El programa resultante debe ser también claro, eficiente y fácil de modificar y usar. Esto implica una disciplina de programación y una metodología, que imponga un estilo de desarrollo que garantice la calidad del producto.

Es la etapa lógica para la resolución del problema, mediante métodos efectivos, escribiendo el código del programa en un Lenguaje de Programación específico.

Para plasmar la solución final de ésta etapa, es necesario iniciar, seleccionando una técnica que permita tener una visión más clara y

detallada de los pasos lógicos a realizar (algoritmos, diagramas de flujo, diagramas de acción , otros).

Posteriormente realizamos la prueba de escritorio, luego pasamos a la codificación, compilación , ejecución y prueba del programa.

1.3.3.1 Algoritmo

Desarrollar software sin un buen algoritmo es como construir una casa sin tener unos buenos planos. Los resultados pueden ser terribles.

Podemos definir un algoritmo como una técnica de solución de problemas, en la cual se escribe una serie de instrucciones paso a paso y que produce resultados a un problema determinado. Los algoritmos se utilizan para esquematizar los pasos de solución usando un lenguaje de especificaciones de algoritmos llamado **seudocódigo**, que requieren menos precisión que un lenguaje de programación formal.

Un buen algoritmo debe ser independiente pero fácilmente traducible a cualquier lenguaje formal de programación. Por lo tanto todo algoritmo debe poseer las siguientes características:

- a. Debe estar compuesto de acciones bien definidas, (ausencia de ambigüedad)
- b. Constar de una secuencia lógica de operaciones,
- c. Debe ser finito.

Continuando con el Ejemplo 1.1. Veamos el siguiente **algoritmo** escrito **lenguaje natural**:

Paso 1: Solicitar al usuario los dos lados del triángulo (a y b).

Paso 2: Calcular la hipotenusa, según $H = \sqrt{a^2 + b^2}$

PASO 3: Desplegar en pantalla el valor de la hipotenusa (H)

Los algoritmos son técnicas que se utilizan en nuestra vida diaria, ya sean escritos o verbales en los cuales no reflexionamos, sólo las ejecutamos automáticamente; de los cuales podemos mencionar algunos ejemplos como : preparar una gelatina, lavarse el cabello, cambiar una llanta , salir del salón de clase , quitarse las medias, ir al cine.

Ejemplo #1.

Qué hacer para ver la película “El Guasón”?

Algoritmo:

1. Ir al cine
2. Comprar una entrada
3. Sentarse en la butaca
4. Ver la película
5. Salir del cine
6. Regresar a casa

Como podemos observar este algoritmo consta de 6 pasos genéricos, que pueden ser más detallados y específicos para lograr ir al cine a ver una película.

1.3.3.2. Prueba de Escritorio

Es un método que consiste en examinar exhaustivamente que la solución produzca los resultados deseados, detectando, localizando y eliminando errores.

Debe considerar varias posibilidades de valores de entrada y escribir cualquier resultado generado durante la revisión del algoritmo. En el conjunto de datos de entrada, trate de incluir valores no muy comunes, para determinar qué hará el algoritmo.

Por ejemplo, suponga que un algoritmo requiere que el usuario ingrese un valor para buscar la Raíz Cuadrada. De seguro, el usuario no debería introducir un valor negativo, porque la raíz cuadrada de un número negativo es imaginaria. Sin embargo, que hará el algoritmo si el usuario lo hace?. Otra posibilidad de ingreso que siempre debe tomarse en cuenta es un ingreso de cero (0) en el denominador de una división.

Continuando con el Ejemplo 1.1. Presentamos la prueba de escritorio para el algoritmo

1. Establezcamos los valores de prueba para a y para b: (3, 2), (-5, 1), (0, 5)
2. Hagamos un cuadro que cuente con los datos de entrada y el resultado del cálculo. Empecemos con el primer juego de datos para a=3 y b=2

Pasos	Variables en memoria			Pantalla
	a	b	h	
1	3	2		Entrar el valor de a: 3.0 Entrar el valor de b: 2.0
2			$\sqrt{3^2 + 2^2}$ 3.60	
3			3.60	Hipotenusa es: 3.60

Continuamos con el segundo juego de datos para a= -5 y b=1

Pasos	Variables en memoria			Pantalla
	a	b	h	
1	-5	1		Entrar el valor de a: -5.0 Entrar el valor de b: 1
2			$\sqrt{-5^2 + 1^2}$ 5.09	
3				Hipotenusa es: 5.09

* Pruebe usted, con el último juego de datos

1.3.3.3 Codificación

La codificación implica la convertir del algoritmo resultante a un lenguaje de programación. Al resultado se le denomina programa fuente.

Tomando como base el ejemplo1. 1:

```
// nombre del archivo que contiene este programa :  
progh.c  
#include <math.h>  
#include <iostream.h>  
#include <conio.h>  
void main( )  
{   float a, b;  
    float h;  
    clrscr( );  
    //Paso 1  
    cout << "Entrar el valor de a : ";  
    cin >> a;  
    cout << "Entrar el valor de b : ";  
    cin >> b;  
    //Paso 2  
    h = sqrt(pow( a,2) +pow( b,2));  
    //Paso 3  
    cout<<"Hipotenusa es = "<< h;  
    getch();  
}
```

1.3.3.4 Compilación/Ejecución

❖ Compilación.

Una vez que el algoritmo ha sido convertido a un programa fuente, es preciso introducirlo a la máquina. Una vez adentro, el programa debe ser compilado o traducido al programa objeto.

Si tras la compilación se presentan errores (de sintaxis) en el programa fuente, éstos se deben corregir y proceder a compilar otra vez el programa.



Figura 1.2 Proceso de Compilación

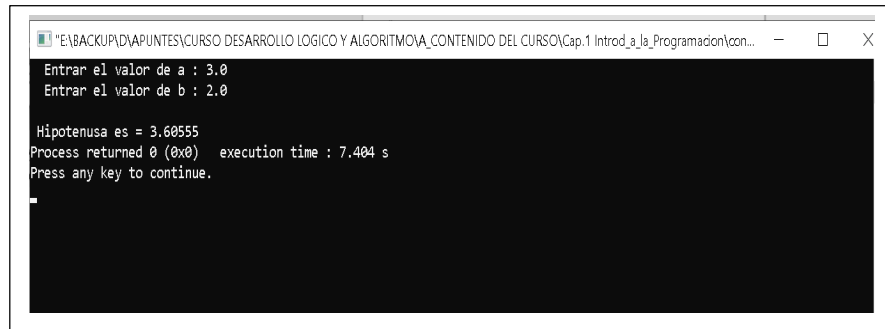
Ejecución.

La ejecución de un programa consiste en que el computador procese cada una de las instrucciones del programa. Al ejecutar el programa debe hacerse con una amplia variedad de datos de

entrada, llamados datos de prueba, que determinarán si el programa tiene errores.

Para realizar la verificación se debe desarrollar una amplia gama de datos de prueba:

- valores normales de entrada,
- valores extremos de entrada que comprueben los límites del programa y
- valores que comprueben aspectos especiales del programa.



```
E:\BACKUP\DIAPUNTOS\CURSO DESARROLLO LOGICO Y ALGORITMO\A CONTENIDO DEL CURSO\Cap.1 Introd_a_la Programacion\con...
Entrar el valor de a : 3.0
Entrar el valor de b : 2.0

Hipotenusa es = 3.60555
Process returned 0 (0x0) execution time : 7.404 s
Press any key to continue.
```

Figura 1.3 Compilación y Ejecución

1.3.3.5 Tipos de Errores.

Durante el desarrollo de un programa es necesario poner especial cuidado para evitar que el producto obtenido presente errores que lo hagan inservible. Los errores se clasifican de la siguiente manera:

- Errores de Compilación: Los errores en tiempo de compilación, o errores sintácticos, se derivan del incumplimiento de las reglas sintácticas del lenguaje, como, por ejemplo, una palabra reservada mal escrita y/o una instrucción incompleta. Si existe un error de sintaxis, la computadora no puede comprender la instrucción y no puede generarse el programa objeto. El compilador despliega una lista de todos los errores encontrados.
- Errores de Ejecución: Estos errores se producen por instrucciones que la computadora puede comprender pero no ejecutar; por ejemplo: la división entre cero o el cálculo de raíces cuadradas de número negativos. En estos casos se detiene la ejecución del programa y se imprime un mensaje de error. Son más difíciles de detectar y corregir que los errores sintácticos, ya que ocurren o no dependiendo de los datos de entrada que se utilicen.
- Errores de Lógica: Consisten en resultados incorrectos obtenidos por el programa. Son los más difíciles de detectar, ya que el programa puede funcionar y no producir errores de compilación ni de ejecución, y sólo puede detectarse el error comparando los resultados del programa con los obtenidos por un método conocido como prueba de escritorio (por ejemplo, operaciones aritméticas hechas a mano). En este caso se debe volver a la etapa de diseño del algoritmo, modificar el algoritmo, cambiar el programa fuente y compilar y ejecutar una vez más.

1. Documentación.

La documentación de un programa consta de las descripciones de los pasos a seguir en el proceso de resolución de un problema. La importancia de la documentación debe ser destacada por su decisiva influencia en el producto final. Los programas deficientemente documentados son difíciles de leer, más difíciles de depurar y casi imposibles de mantener y modificar.

La documentación de un programa se hace a través de las líneas de comentarios, y se incluyen tantas como sean necesarias para aclarar o explicar el significado de las líneas de código que no son obvias, especialmente en lo que respecta a:

- los identificadores y estructuras de datos declaradas,
- las estructuras de control,
- los métodos y sus parámetros.

Bibliografía

Extracto del Capítulo I del libro

“Introducción a la Programación Orientada a Objetos.”

Primera Edición. Imprenta Universitaria. U.T.P.

O. Barraza, F. Krol, L. Meléndez y M. Velásquez.

Modificado por la Prof.F. de Krol