UNIDAD 3: ESTRUCTURAS DE CONTROL DE FLUJO

TECNICAS DE PROGRAMACIÓN

La teoría de la programación considera las técnicas de:

programación modular y
programación estructurada.

El diseño de un programa establece la descomposición del problema en módulos – *programación modular* -, la programación de cada módulo mediante métodos estructurados – *programación estructurada* – y su posterior unión con procedimientos ascendentes o descendentes.

PROGRAMACIÓN MODULAR

El programa se divide en módulos (partes independientes), cada de una de las cuales ejecuta una única tarea y se codifican independiente de otros módulos.

Existe un programa principal que controla todo lo que sucede y va transfiriendo el control a los distintos submódulos o subprogramas.

Los módulos son independientes entre sí, ninguno tiene acceso a otro, excepto el programa principal.

La descomposición de un programa en módulos independientes se conoce como el método de divide y vencerás.

En el caso de las funciones de entrada, proceso y salida, es conveniente que cada una esté en un módulo separado.

PROGRAMACION ESTRUCTURADA

Se refiere a un conjunto de técnicas que aumentan considerablemente la productividad del programa.

Hace que los programas sean más fáciles de escribir, verificar, leer y mantener.

Conjunto de técnicas que incorporan:

Recursos abstractos
Diseño descendente (top-down)
Estructuras básicas

RECURSOS ABSTRACTOS

La programación estructurada se auxilia de recursos abstractos en lugar de los recursos concretos de que dispone como puede ser un determinado lenguaje de programación.

Descomponer un programa en términos de recursos abstractos consiste en descomponer una determinada acción compleja en función de un número de acciones más simples, capaces de ser ejecutadas y que constituirán las instrucciones.

DISEÑO DESCENDENTE (TOP-DOWN)

Es el proceso mediante el cual un problema se descompone en una serie de pasos sucesivos de refinamiento, obteniendo estructuras jerárquicas.

Nivel n: Vista exterior: ¿qué hace?

Nivel n+1: Vista interior: ¿cómo lo hace?

* ESTRUCTURAS BÁSICAS

Un programa puede ser escrito utilizando solamente tres tipos de estructuras de control:

- **Secuenciales**
- **Selectivas**
- **Repetitivas**

CARACTERISTICAS

Un programa cumple las siguientes características:

- Posee un solo punto de entrada y uno de salida para control del programa.
- Existen caminos desde la entrada hasta la salida que se pueden seguir y pasan por todas las partes del programa.
- Todas las instrucciones son ejecutables y no existen lazos o bucles infinitos.

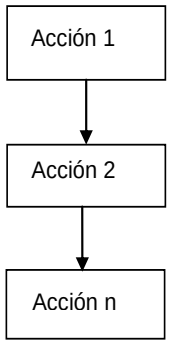
*** LA SECUENCIALIZACION**

Es una estructura que permite controlar la ejecución de un conjunto de acciones, en orden secuencial.

Estas acciones pueden ser operaciones primitivas elementales como: declaraciones de variables, leer datos, escribir o mostrar datos o calcular alguna expresión.

PSEUDOCODIGO

```
PROGRAMA nombre
INICIO
        Acción 1
        Acción 2
        ……
        Acción n
FINPROGRAMA
```



ELABORAR UN ALGORITMO

Siempre vamos a seguir estos pasos para elaborar un algoritmo :

1. Definir el problema
2. Analizar el problema
3. Diseñar el programa

EJEMPLO

1. Definir el problema
   Calcular el sueldo de un empleado.
2. Analizar el problema
   Entrada: nombre, horas trabajadas y valor por hora.
   Proceso: sueldo equivale a horas trabajadas por el valor de la hora
   Salida: nombre, sueldo
3. Diseñar el algoritmo:
   Diseñar la estructura de la solución, elaborando el algoritmo.

1. PROGRAMA calculaSueldo
2. VAR
3. empleado: CADENA
4. horasTrab: ENTERO
5. valorHora, sueldo: REAL
6. INICIO
7. // solicitar que ingresen los datos para leerlos
8. Escribir (“Ingrese nombre del empleado”)
9. Leer (empleado)
10. Escribir (“Ingrese número de horas trabajadas”)
11. Leer (horasTrab)
12. Escribir (“Ingrese el valor por hora”)
13. Leer (valorHora)
14. //calcular el sueldo
15. sueldo = horasTrab * valorHora
16. // mostramos el sueldo y empleado
17. Escribir(“El sueldo de”, empleado, “es de $”, sueldo)
18. FINPROGRAMA

* LA SELECCIÓN

Es una estructura que permite controlar la ejecución de acciones que requieren de condiciones para su realización.

A veces surge la necesidad de usar condiciones y según el resultado de eso seguir un camino u otro.

Estas estructuras evalúan una condición y en función del resultado de la misma se realiza una acción u otra.

TIPOS DE ESTRUCTURAS SELECTIVAS

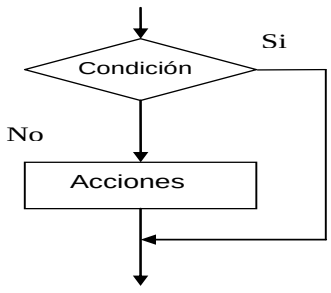
Las estructuras selectivas pueden ser:

- **Simples**
- **Dobles**
- **Múltiples**

* SELECCIÓN SIMPLE

Ejecuta una determinada acción cuando se cumple una condición. Si la condición es verdadera se ejecuta la acción, si es falsa no se ejecuta ninguna acción.

SI condición ENTONCES
Acciones
FINSI



EJEMPLO
Elaborar un algoritmo para calcular y mostrar el sueldo de un empleado. Si el empleado ha trabajado más de 40 horas le agregamos a su sueldo un 5% de incentivo.

1. Definir el problema
   - Calcular el sueldo de un empleado
2. Analizar el problema
   - Entrada: nombre, horas trabajadas y valor por hora.
   - Proceso: sueldo = horas trabajadas * valor por hora
   - Si trabajó más de 40 horas :
     sueldo = sueldo + (sueldo * 0.05)
   - Salida: nombre, sueldo
3. Diseñar el algoritmo:

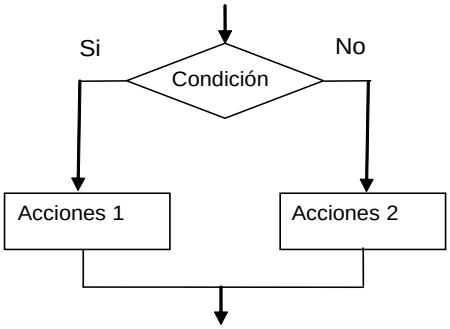
PROGRAMA calculaSueldo2

1. VAR
2. empleado: CADENA
3. horasTrab: ENTERO
4. valorHora, sueldo: REAL
5. INICIO
6. // solicitar que ingresen los datos para leerlos
7. Escribir ("Ingrese nombre del empleado")
8. Leer (empleado)
9. Escribir ("Ingrese número de horas trabajadas")
10. Leer (horasTrab)
11. Escribir ("Ingrese el valor por hora")
12. Leer (valorHora)
13. //calcular el sueldo
14. sueldo = horasTrab * valorHora
15. SI horasTrab > 40 ENTONCES
    sueldo = sueldo + (sueldo * 0,05)
15. FINSI
16. // mostramos el sueldo y empleado
17. Escribir("El sueldo de", empleado, "es de $", sueldo)
18. FINPROGRAMA

* **SELECCIÓN DOBLE**

Esta estructura permite elegir entre dos opciones en función del cumplimiento de una determinada condición. Se ejecuta una u otra, pero nunca las dos a la vez.

SI condición ENTONCES
    Acciones 1
SINO
    Acciones 2
FINSI



EJEMPLO
Elaborar un algoritmo para calcular y mostrar el sueldo de un empleado. Si el empleado ha trabajado más de 40 horas, a las horas que exceden la pagamos al doble de su valor.

1. Definir el problema
   Calcular el sueldo de un empleado
2. Analizar el problema
   Entrada: Nombre, horas trabajadas y valor por hora.
   Proceso:
   Si trabajó menos o igual que 40 hs.:
   sueldo = horas trabajadas * valor por hora
   Si trabajó más de 40 hs.:
   sueldo = (40 * valor por hora) + ((horas trabajadas – 40) * (valor por hora * 2))
   Salida: nombre, sueldo
3. Diseñar el algoritmo

PROGRAMA calculaSueldo3

1. VAR
2. empleado: CADENA
3. horasTrab: ENTERO
4. valorHora, sueldo: REAL
5. INICIO
6. // solicitar que ingresen los datos para leerlos
7. Escribir (“Ingrese nombre del empleado”)
8. Leer (empleado)
9. Escribir (“Ingrese número de horas trabajadas”)
10. Leer (horasTrab)
11. Escribir (“Ingrese el valor por hora”)
12. Leer (valorHora)
13. //calcular el sueldo
14. sueldo = horasTrab * valorHora
15. SI horasTrab <= 40 ENTONCES
    sueldo = horasTrab * valorHora
15. SINO
    sueldo=(40 * valorHora) + ((horasTrab – 40) * (valorHora * 2))
15. FINSI
16. // mostramos el sueldo y empleado
17. Escribir (“El sueldo de”, empleado, “es de $”, sueldo)
18. FINPROGRAMA

* **SELECCIÓN MULTIPLE**

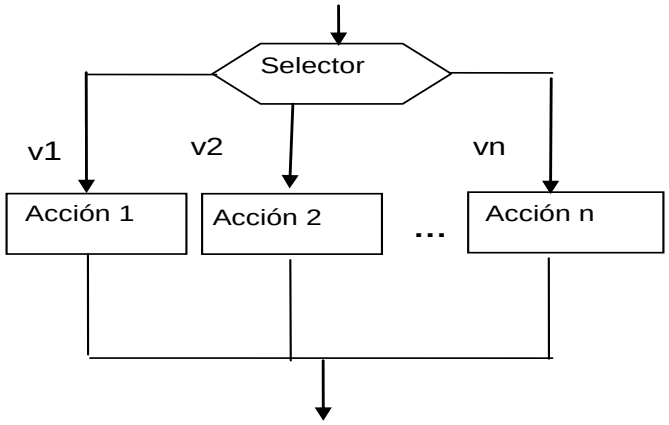
Es cuando existen más de dos alternativas posibles, es decir que en base a una condición se pueden seguir más de 2 caminos.

En el caso de selección múltiple se evalúa una expresión que podrá tomar n valores distintos, según el resultado podrá seguir una de las n acciones posibles**.**

*** ALTERNATIVAS ANIDADAS**

El caso de selección anidada o alternativas anidadas se presenta cuando tenemos una selección simple y dentro anidada otra, y ésta a otra y así sucesivamente. Es decir, que luego de tomar una decisión y señalar el correspondiente es necesario tomar otra decisión y así sucesivamente.

SEGUN CASO selector HACER
v1: acción 1
v2: acción 2
….
DE OTRO MODO
acción n
FINSEGUN



EJEMPLO

Elaboremos un algoritmo que lea el número del día de la semana (1 a 7) e imprima el nombre del día (lunes, martes,…, domingo).

1. Definir el problema
   Escribir el nombre del día de la semana dado su número de orden
2. Analizar el problema
   Entrada: variable que indica número del día de la semana.
   Proceso: Según número ingresado se dará el nombre del día:
   Si 1: Lunes, 2: Martes, 3: Miércoles, 4: Jueves, 5: Viernes, 6: Sábado, 7: Domingo.
   Salida: nombre del día de la semana
3. Diseñar el algoritmo

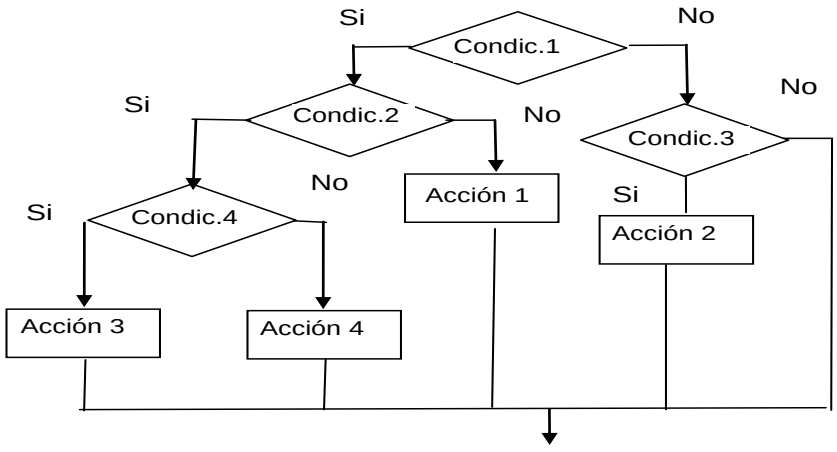
PROGRAMA nombreDia

1. VAR
2. numDia: ENTERO

3. INICIO
4. // solicitar que ingrese el número del día
5. Escribir (“Ingrese número del día de la semana”)
6. Leer (numDia)
7. //determinar el nombre del día
8. SEGUN CASO numDia HACER
9. 1: Escribir (“Lunes”)
10. 2: Escribir (“Martes”)
11. 3: Escribir (“Miércoles”)
12. 4: Escribir (“Jueves”)
13. 5: Escribir (“Viernes”)
14. 6: Escribir (“Sábado”)
15. 7: Escribir (“Domingo”)
16. DE OTRO MODO
17. Escribir (“No está en el rango de 1 a 7”)
18. FINSEGUN
19. FINPROGRAMA

*** SELECCIÓN ANIDADA**

```
SI condición 1 ENTONCES
        SI condición 2 ENTONCES
                SI condición 4 ENTONCES
                    Acción3
                SINO
                    Acción 4
                FINSI
        SINO
                Acción 1
        FINSI
SINO
        SI condición 3 ENTONCES
                Acción 2
        FINSI
FINSI
```



Elaboremos un algoritmo que lea el monto total de una compra y según el mismo informe el porcentaje de descuento que le corresponde.

1. Definir el problema
   Establecer el porcentaje de descuento de una compra según el monto total.
2. Analizar el problema
   Entrada: variable que indica el monto de la compra.
   Proceso: Establecer en función del monto el descuento a realizar.
   - Si monto <= 200: 0%
   - Si monto > 201 y < 300: 10%
   - Si monto >= 301 y < 400: 20%
   - Si monto >= 401: 25%

   Salida: Porcentaje de descuento
3. Diseñar el algoritmo

PROGRAMA Descuento

```
1. VAR
2. montoCompra, descuento: ENTERO
3. INICIO
4. //solicitar que ingrese el monto de la compra para leer
5. Escribir ("Ingrese monto de la compra")
6. Leer (montoCompra)
7. //determinar el porcentaje de descuento

8. SI montoCompra > 200 ENTONCES
9.                 SI montoCompra > 300  ENTONCES
10.                         SI montoCompra > 400 ENTONCES
11.                                 descuento  =  25
12.                         SINO
13.                                 descuento  =  20
14.                         FINSI
15.                  SINO
16.                         descuento  = 10
17.                 FINSI
18.         SINO
19.                 descuento  =  0
20. FINSI
21. //Mostramos el porcentaje de descuento
22. Escribir (" Le corresponde un descuento del", descuento," %")
23. FINPROGRAMA
```

* ESTRUCTURAS REPETITIVAS: CICLOS O BUCLES

Hemos visto las estructuras de control Secuenciales y las Selectivas.
Pero hay problemas que necesitan que algunas operaciones se ejecuten un número determinado de veces. A esto se le llama estructuras repetitivas.
Las estructuras que repiten una secuencia de instrucciones un número determinado de veces se denominan bucles o ciclos, y se llama iteración al hecho de repetir la ejecución de una secuencia de acciones.

Las estructuras repetitivas son:

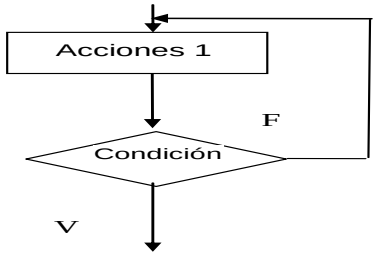
- **Repetir**

- **Mientras**
- **Variar**
- **Iterar**

*** REPETIR – HASTA (CICLO 1-X)**

Ejecuta las acciones 1 tantas veces como sea necesario hasta que se cumpla la condición, es decir que la salida del bucle es cuando la condición es Verdadera.
Se dice ciclo 1-x, porque se sabe que al menos se ejecuta 1 vez, pero no sabemos cuántas veces.
Se llama "cola inteligente" porque la condición está al final.

1. **REPETIR**
2. **Acciones 1**
3. **HASTA QUE condición**



Algoritmo para realizar la división entera de dos números enteros mediante restas sucesivas

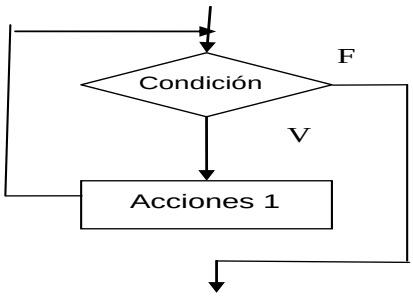
```
PROGRAMA divisionEntera
VAR
dividendo, divisor, cociente, resto : ENTERO
INICIO
cociente = 0  // inicializo variables
Escribir ("introduzca el dividendo y divisor")
Leer (dividendo)
REPETIR
    Leer (divisor)      // ingreso datos
HASTA QUE divisor > 0
resto = dividendo
REPETIR
resto = resto - divisor
cociente = cociente + 1
HASTA QUE resto < divisor
Escribir ("cociente: ",cociente," y resto: ", resto)
FINPROGRAMA
```

*** MIENTRAS (CICLO 0..X)**

Esta estructura lo primero que hace es analizar la condición, si la condición se cumple (es verdadera) se ejecutan las acciones 1, si la condición es Falsa sale del bucle.
Se llama "cabeza inteligente" porque la condición está al principio.

Se dice ciclo 0-x, porque no se sabe cuántas veces se van a ejecutar las acciones 1 y puede ser que no se ejecuten nunca si la condición es falsa la primera vez.

MIENTRAS condición HACER
    Acciones 1
FIN MIENTRAS



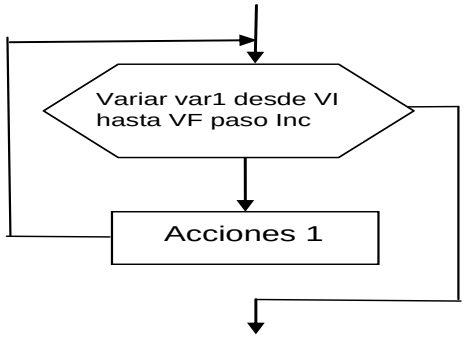
EJEMPLO Algoritmo para realizar la división entera de dos números enteros mediante restas sucesivas

```
PROGRAMA divisionEntera
VAR
    dividendo, divisor, cociente, resto: ENTERO
INICIO
cociente = 0
Escribir ("introduzca el dividendo y divisor")
Leer (dividendo)
REPETIR
    Leer(divisor)
HASTA QUE divisor > 0
resto = dividendo
MIENTRAS resto > = divisor HACER
    resto = resto - divisor
    cociente = cociente + 1
FIN_MIENTRAS
Escribir ("cociente: ",cociente," y resto: ", resto)
FINPROGRAMA
```

*** VARIAR (CICLO EXACTO)**

VARIAR var1 DESDE VI HASTA VF PASO Inc
    Acciones 1
FINVARIAR
Donde:
Var1 : variable de control (ENTERO)
VI: variable inicial (ENTERO)
VF: variable final (ENTERO)
Inc: variable, constante o expresión aritmética (ENTERO). Si no se incluye se asume 1.

Nunca deben modificarse entre las acciones 1 las variables de control, inicial, final e incremento.



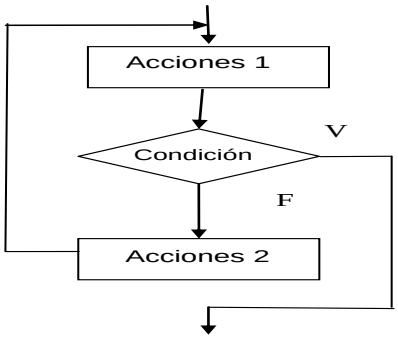
EJEMPLO: Algoritmo que dado un valor n entero mayor que 0, realice la suma de los n primeros números enteros.

```
PROGRAMA sumaEnteros
Var
num, suma, contador : ENTERO
INICIO
Escribir ("introduzca el número n:")
Leer (num)
suma = 0
VARIAR contador DESDE 1 HASTA num
     suma = suma + contador
FINVARIAR
Escribir ("la suma resultante es:",suma)
FINPROGRAMA
```

*** ITERAR**

Es una combinación de repetir y mientras.
Se ejecutan las acciones 1 y luego se evalúa la condición, si la condición es Falsa ejecuta las acciones 2. Luego va a ejecutar las acciones 1 y vuelve a evaluar la condición y así sucesivamente.
La salida del bucle es cuando la condición es Verdadera.

ITERAR
Acciones 1
SALIR SI condición
Acciones 2
FINITERAR



EJEMPLO: Algoritmo para Ingresar n valores enteros y obtener la suma. El ingreso de datos finaliza cuando se ingresa un cero.

```
PROGRAMA sumaValores
VAR
num, suma: ENTERO
INICIO
suma = 0
ITERAR
    Escribir ("introduzca el número n:")
    Leer (num)
  SALIR SI  num = 0
    suma = suma + num   // calculo suma
FINITERAR
Escribir ("la suma resultante es:",suma)
FINPROGRAMA
```

*** CUADRO COMPARATIVO**

	Repetir	**Mientras**	**Iterar**	**Variar**
Ciclo	**1 – x**	**0 – x**	**Acciones 1: 1-x Acciones 2: 0-x**	**exacto**
Condición de salida	**Verdadera**	**Falsa**	**Verdadera**	**No tiene condición**
Posición de la condición	**Final “cola inteligente”**	**Principio “cabeza inteligente”**	**Medio**	**No tiene condición**

* **<u>PUESTA A PUNTO DE PROGRAMAS</u>**

Consiste en localizar, verificar y corregir los errores de programación.
El objetivo es prevenir tantos errores como sea posible a la hora de ejecutar un programa.
Una prueba o ensayo es satisfactoria cuando se detectan errores.

La puesta a punto de un programa consta de las siguientes fases:

1. Detección de errores
2. Depuración de errores
    a. Localización
    b. Eliminación
3. Prueba del programa

* ERRORES TIPICOS

Errores típicos:

De sintaxis: se originan en la fase de compilación/interpretación del programa y se deben a causas propias de la sintaxis del lenguaje, como escrituras incorrectas, omisión de signo, etc.

De lógica: pueden detener la ejecución del programa o producir resultados erróneos. Ej. Si en una división n1/n2 no se valida que **n2 sea distinto de cero al ejecutar esta instrucción se iría fuera de programa.**

* PRUEBA DE UN PROGRAMA

Prueba de programa: realizar pruebas con conjuntos de datos de muestra, cuya solución sea conocida y correcta.

En una tabla con todas las variables del programa ir siguiendo instrucción por instrucción y ver los resultados obtenidos. Se plantean juegos de datos y el resultado previsto.

*VALIDACION DE DATOS Y TRAZA

Validación de datos: que todos los datos de entrada sean correctos y estén dentro del rango válido.

Traza: indica la secuencia de ejecución de las instrucciones dado un conjunto de valores.

* PARA PREPARAR UNA PRUEBA

1. Determinar los caminos del programa que deben ser controlados en cada estructura.
2. Se definen los recorridos del programa.
3. Se deducen a partir del paso 1 los datos que deben introducirse en la entrada.
4. Se deducen los datos que deben obtenerse a la salida del programa.

EJEMPLO

Calcular la suma y el producto de dos números enteros

Ingresar nro1 y nro2
Calcular nro1 + nro2 y nro1*nro2
Mostrar el resultado de la suma y el producto

```
1 PROGRAMA sumaprod
2 VAR nro1, nro2, suma, producto: ENTERO
3 INICIO
4 // ingresar datos
5 Escribir (“Ingrese primer número”)
6 Leer (nro1)
7 Escribir (“Ingrese segundo número”)
8 Leer (nro2)
```

9 //calcular la suma y el producto
10 suma = nro1 + nro2
11 producto = nro1 * nro2
12 //mostrar resultados
13 Escribir ("Suma:", suma, "Producto:", producto)
14 FINPROGRAMA

Determinar datos de prueba

	nro1	nro2	suma	producto
1ra. Prueba	23	56	79	1288
2da. Prueba	-89	90	1	-8010
3ra. Prueba	24	0	24	0
4ta. Prueba	-7	-67	-74	469

La traza sería siempre la misma: 1,2,3,4,5,6,7,8,9,10,11,12,13 y 14

PRUEBAS DE PROGRAMAS

En la tabla se reflejan los estados de las variables a lo largo del programa.
Las variables deben estar:
declaradas: qué tipo de dato son
definidas: que tengan valor

1ra. prueba	Instruc-ción	nro1	nro2	suma	producto	Pantalla
	2	ENTERO (Vacío)	ENTERO (Vacío)	ENTERO (Vacío)	ENTERO (Vacío)	
	5					Ingrese primer número
	6	23				
	7					Ingrese segundo número
	8		56			
	10			79		
	11				1288	
Resulta-dos previstos	13					Suma: 79 Producto: 1288

2 da. prueba	Instruc-ción	nro1	nro2	suma	producto	Pantalla
	2	ENTERO (Vacío)	ENTERO (Vacío)	ENTERO (Vacío)	ENTERO (Vacío)	

	5					Ingrese primer número
	6	-89				
	7					Ingrese segundo número
	8		90			
	10			1		
	11				-8010	
Resulta-dos previstos	13					Suma: 1 Producto: -8010

3 ra. Prueba	Instruc-ción	nro1	nro2	suma	producto	Pantalla
	2	ENTERO (Vacío)	ENTERO (Vacío)	ENTERO (Vacío)	ENTERO (Vacío)	
	5					Ingrese primer número
	6	24				
	7					Ingrese segundo número
	8		0			
	10			24		
	11				0	
Resultados previstos	13					Suma: 24 Producto: 0

4 ta. Prueba	Instruc-ción	nro1	nro2	suma	producto	Pantalla
	2	ENTERO (Vacío)	ENTERO (Vacío)	ENTERO (Vacío)	ENTERO (Vacío)	
	5					Ingrese primer número
	6	-7				
	7					Ingrese segundo número

	8		**-67**			
	10			**-74**		
	11				**469**	
Resultados previstos	**13**					**Suma: -74 Producto: 469**

COMPLEJIDAD COMPUTACIONAL

La Complejidad Computacional estudia los recursos requeridos durante el cálculo para resolver un problema.

Así, un cálculo difícil requerirá más recursos que uno de menor dificultad.

Los recursos estudiados son:

Tiempo: número de pasos de ejecución de un algoritmo.

Espacio: cantidad de memoria utilizada.

CLASIFICACION DE PROBLEMAS

El estudio de los procesos computacionales, conduce a una clasificación en dos grandes clases:

los problemas con solución y

los problemas sin solución.

Los problemas solucionables, requieren gran cantidad de recursos.

ORDEN DE COMPLEJIDAD**El algoritmo *más* eficiente es:**

- **el más rápido (lo más común)**
- **el que ocupa menos memoria (algunas veces)**
- **la eficiencia se expresa como función del tamaño del problema**

Los problemas que tienen una solución con orden de complejidad lineal son los que se resuelven en un tiempo que se relaciona linealmente con su tamaño.

¿COMO SE MIDE?

Saber con exactitud el tiempo que va a tardar un programa en dar una salida se vuelve una tarea muy difícil hoy en día.

En vez de calcular el tiempo exacto que va a tardar nuestro algoritmo se calcula la cantidad de operaciones en función del tamaño de la entrada.

Tamaño de la entrada: es el tiempo de

1 operación.

EJEMPLO CLASICO

El problema de la ordenación de números, es uno de los más común.

Por eso existen bastantes algoritmos para resolver este problema.

¿Cuál es la diferencia entre ellos? Aparte del nombre, La complejidad .

ANALISIS DE UN ALGORITMO

Normalmente se habla de dos tipos:

Análisis del peor caso

Análisis del caso promedio

Análisis del peor caso: Es calcular su función de crecimiento dándole la peor de todas las entradas posibles.

Análisis del caso promedio: Es el promedio de cuánto tardaría nuestro algoritmo de cada una de las entradas posibles.

COSTO COMPUTACIONAL

¿Cuál es mejor? El del peor caso es el más fácil de razonar.

Para analizar el costo computacional de un algoritmo, se siguen tres pasos:

1. Suponer el peor de los casos (en el que se ejecutan más líneas y el más grande).
2. Asignar un costo a cada operación o línea de código.
3. Ver cuantas veces va a ser ejecutada cada línea de código.