
	Manual de prácticas del Laboratorio de Fundamentos de programación	Código:	MADO-17
		Versión:	03
		Página	1/16
		Sección ISO	8.3
		Fecha de emisión	26 / agosto / 2021
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B	
La impresión de este documento es una copia no controlada			

Guía práctica de estudio 03: Solución de problemas y Algoritmos.



Elaborado por	Actualizado por:	Revisado por:
M.C. Edgar E. García Cano Ing. Jorge A. Solano Gálvez	M.C. Cintia Quezada Reyes Ing. Maricela Castañeda Perdomo	M.C. Laura Sandoval Montaño

	Manual de prácticas del Laboratorio de Fundamentos de programación	Código:	MADO-17
		Versión:	03
		Página	2/16
		Sección ISO	8.3
		Fecha de emisión	26 / agosto / 2021
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B	
La impresión de este documento es una copia no controlada			

Guía práctica de estudio 03: Solución de problemas y Algoritmos.

Objetivo:

El alumno elaborará algoritmos correctos y eficientes en la solución de problemas siguiendo las etapas de Análisis y Diseño pertenecientes al Ciclo de vida del software.

Actividades:

- A partir del enunciado de un problema, identificar el conjunto de entrada y el conjunto de salida.
- Elaborar un algoritmo que resuelva un problema determinado (dado por el profesor), identificando los módulos de entrada, de procesamiento y de salida.


Introducción

Un problema informático se puede definir como el conjunto de instancias al cual corresponde un conjunto de soluciones, junto con una relación que asocia para cada instancia del problema un subconjunto de soluciones (posiblemente vacío).

Para poder solucionar un problema nos apoyamos en la Ingeniería de Software que de acuerdo con la IEEE se define como "La aplicación de un enfoque sistemático, disciplinado y cuantificable hacia el desarrollo, operación y mantenimiento del software". Por lo que el uso y establecimiento de principios de ingeniería sólidos, son básicos para obtener un software que sea económicamente fiable y funcione eficientemente.

La Ingeniería de Software provee métodos que indican cómo generar software. Estos métodos abarcan una amplia gama de tareas:

- Planeación y estimación del proyecto.
- Análisis de requerimientos del sistema y software.
- Diseño de la estructura de datos, la arquitectura del programa y el procedimiento algorítmico.
- Codificación.
- Pruebas y mantenimiento (validación y verificación).

	Manual de prácticas del Laboratorio de Fundamentos de programación	Código:	MADO-17
		Versión:	03
		Página	3/16
		Sección ISO	8.3
		Fecha de emisión	26 / agosto / 2021
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B	
La impresión de este documento es una copia no controlada			

Ciclo de vida del software


La ISO (International Organization for Standardization) en su norma 12207 define al ciclo de vida de un software como:

“Un marco de referencia que contiene las actividades y las tareas involucradas en el desarrollo, la explotación y el mantenimiento de un producto de software, abarcando desde la definición hasta la finalización de su uso”

La Figura 1 muestra las actividades, también nombradas *etapas*, que se realizan en el ciclo de vida del software.



Figura 1: Ciclo de vida del software.

	Manual de prácticas del Laboratorio de Fundamentos de programación	Código:	MADO-17
		Versión:	03
		Página	4/16
		Sección ISO	8.3
		Fecha de emisión	26 / agosto / 2021
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B	
La impresión de este documento es una copia no controlada			

Solución de problemas

Dentro del ciclo de vida del software, en el análisis se busca comprender la necesidad, es decir, entender el problema.

El análisis es el proceso para averiguar qué es lo que requiere el usuario del sistema de software (análisis de requisitos). Esta etapa permite definir las necesidades de forma clara y concisa (especificación de requisitos).

Por lo tanto, la etapa del análisis consiste en conocer qué es lo que está solicitando el usuario. Para ello es importante identificar dos grandes conjuntos dentro del sistema: el conjunto de entrada y el conjunto de salida. En la Figura 2 se muestra de manera esquemática los componentes del sistema.

El **conjunto de entrada** está compuesto por todos aquellos datos que pueden alimentar al sistema.

El **conjunto de salida** está compuesto por todos los datos que el sistema regresará como resultado del proceso. Estos datos se obtienen a partir de los datos de entrada.

La unión del conjunto de entrada y el conjunto de salida forman lo que se conoce como el dominio del problema, es decir, los valores que el problema puede manejar.

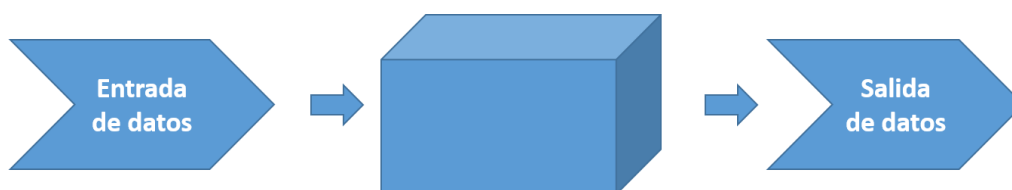



Figura 2. Sistema

La etapa de análisis es crucial para la creación de un software de calidad, ya que si no se entiende qué es lo que se desea realizar, no se puede generar una solución. Sin embargo, es común caer en ambigüedades debido al mal entendimiento de los requerimientos iniciales.

	Manual de prácticas del Laboratorio de Fundamentos de programación	Código:	MADO-17
		Versión:	03
		Página	5/16
		Sección ISO	8.3
		Fecha de emisión	26 / agosto / 2021
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B	
La impresión de este documento es una copia no controlada			

Ejemplo 1

PROBLEMA: Determinar si un número dado es positivo o negativo.

RESTRICCIONES: El número no puede ser cero.

DATOS DE ENTRADA: El conjunto de datos de entrada E está compuesto por el conjunto de los números reales, excepto el cero.

$$E \subset \mathbb{R}^1, \text{ donde} \\ \text{num} \in E \text{ de } (-\infty, \infty) - \{0\}$$

NOTA: \mathbb{R}^1 representa al conjunto de números reales de una dimensión.

DATOS DE SALIDA: El conjunto de salida S está compuesto por dos valores mutuamente excluyentes.

Un posible conjunto de salida son los valores enteros 0 o 1, donde 0 indica que el valor es positivo y 1 indica el valor es negativo.


$$\text{res} = 0, \text{ si num } (0, \infty), \text{ res} = 1, \text{ si num } (-\infty, 0)$$

Otro posible conjunto de datos de salida son los valores booleanos o lógicos *Verdadero* o *Falso*, donde *Verdadero* indica que el valor es positivo y *Falso* indica que el valor es negativo; o viceversa, *Verdadero* indica que el valor es negativo y *Falso* indica que el valor es positivo.

La Figura 3 muestra los componentes del sistema para el ejemplo 1.



Figura 3. Ejemplo 1

	Manual de prácticas del Laboratorio de Fundamentos de programación	Código:	MADO-17
		Versión:	03
		Página	6/16
		Sección ISO	8.3
		Fecha de emisión	26 / agosto / 2021
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B	
La impresión de este documento es una copia no controlada			

Ejemplo 2

PROBLEMA: Obtener el mayor de dos números diferentes dados.

RESTRICCIONES: Los números de entrada deben ser diferentes.

DATOS DE ENTRADA: El conjunto de entrada E está dividido en dos subconjuntos E y E' . El primer número (num1) puede adquirir cualquier valor del conjunto de los números reales ($E = (-\infty, \infty)$), sin embargo, el conjunto de entrada del segundo número (num2) es un subconjunto de E , es decir, E' está compuesto por el conjunto de los números reales excepto num1 ($E' = (-\infty, \infty) \neq \text{num1}$).

$E, E' \subset \mathbb{R}^1$, donde
 $\text{num1} \in E$ de $(-\infty, \infty)$,
 $\text{num2} \in E'$ de $(-\infty, \infty) - \{\text{num1}\}$

DATOS DE SALIDA: El conjunto de datos de salida S que puede tomar el resultado r está compuesto por el conjunto de los números reales.

$S \subset \mathbb{R}^1$, donde $r \in S$ de $(-\infty, \infty)$

La Figura 4 representa los components del sistema del ejemplo 2.

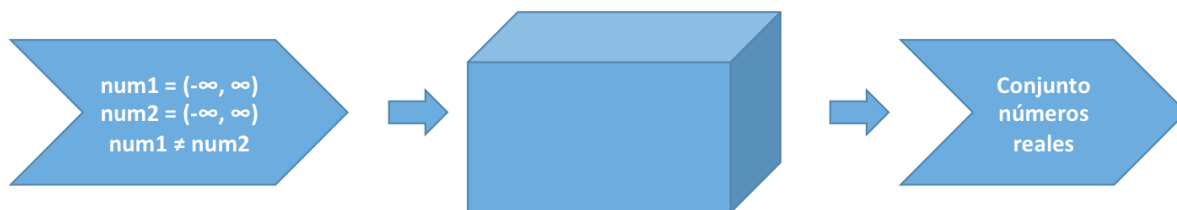



Figura 4. Ejemplo 2

	Manual de prácticas del Laboratorio de Fundamentos de programación	Código:	MADO-17
		Versión:	03
		Página	7/16
		Sección ISO	8.3
		Fecha de emisión	26 / agosto / 2021
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B	
La impresión de este documento es una copia no controlada			

Ejemplo 3

PROBLEMA: Obtener el factorial de un número dado. El factorial de un número está dado por el producto de ese número por cada uno de los números anteriores hasta llegar a 1. El factorial de 0 (0!) es 1:

$$n! = n * (n-1)!$$

RESTRICCIONES: El número de entrada debe ser entero positivo o cero. No puede ser negativo.

DATOS DE ENTRADA: El conjunto de entrada E está dado por el conjunto de los números naturales o por el cero.

$$E \subset \mathbb{N}^1, \text{ donde} \\ \text{num} \in E \text{ de } [1, \infty) \cup \{0\}$$


DATOS DE SALIDA: El conjunto de salida S está conformado por el conjunto de los números naturales.

$$S \subset \mathbb{N}^1; \text{ donde} \\ \text{res} \in S \text{ de } [1, \infty)$$

Los componentes del sistema de este ejemplo se muestran en la Figura 5.



Figura 5. Ejemplo 3

	Manual de prácticas del Laboratorio de Fundamentos de programación	Código:	MADO-17
		Versión:	03
		Página	8/16
		Sección ISO	8.3
		Fecha de emisión	26 / agosto / 2021
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B	
La impresión de este documento es una copia no controlada			

Algoritmos

Una vez realizado el análisis, es decir, ya que se entendió qué es lo que está solicitando el usuario y ya identificado el conjunto de entrada y el conjunto de salida, se puede proceder al diseño de la solución, esto es, a la generación del algoritmo.

Dentro del ciclo de vida del software, la creación de un algoritmo se encuentra en la etapa de diseño. Ver figura 1.

Durante el diseño se busca proponer una o varias alternativas viables para dar solución al problema y con base en esto tomar la mejor decisión para iniciar la construcción.

Un problema matemático es computable si éste puede ser resuelto, en principio, por un dispositivo computacional.


La teoría de la computabilidad es la parte de la computación que estudia los problemas de decisión que pueden ser resueltos con un algoritmo.

Un algoritmo se define como un conjunto de reglas, expresadas en un lenguaje específico, para realizar alguna tarea en general, es decir, un conjunto de pasos, procedimientos o acciones que permiten alcanzar un resultado o resolver un problema. Estas reglas o pasos pueden ser aplicados un número ilimitado de veces sobre una situación particular.

Un algoritmo es la parte más importante y durable de las ciencias de la computación debido a que éste puede ser creado de manera independiente tanto del lenguaje como de las características físicas del equipo que lo va a ejecutar.

Las principales características con las que debe cumplir un algoritmo son:

- Preciso: Debe indicar el orden de realización de paso y no puede tener ambigüedad.
- Definido: Si se sigue dos veces o más se obtiene el mismo resultado.
- Finito: Tiene fin, es decir tiene un número determinado de pasos.
- Correcto: Cumplir con el objetivo.
- Debe tener al menos una salida y ésta debe de ser perceptible
- Debe ser sencillo y legible
- Eficiente: Realizarlo en el menor tiempo posible
- Eficaz: Que produzca el efecto esperado

	Manual de prácticas del Laboratorio de Fundamentos de programación	Código:	MADO-17
		Versión:	03
		Página	9/16
		Sección ISO	8.3
		Fecha de emisión	26 / agosto / 2021
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B	
La impresión de este documento es una copia no controlada			

Por tanto, un buen algoritmo debe ser correcto (cumplir con el objetivo) y eficiente (realizarlo en el menor tiempo posible), además de ser entendible para cualquier persona.

Las actividades a realizar en la elaboración de un algoritmo para obtener una solución a un problema de forma correcta y eficiente se muestran en la Figura 6.

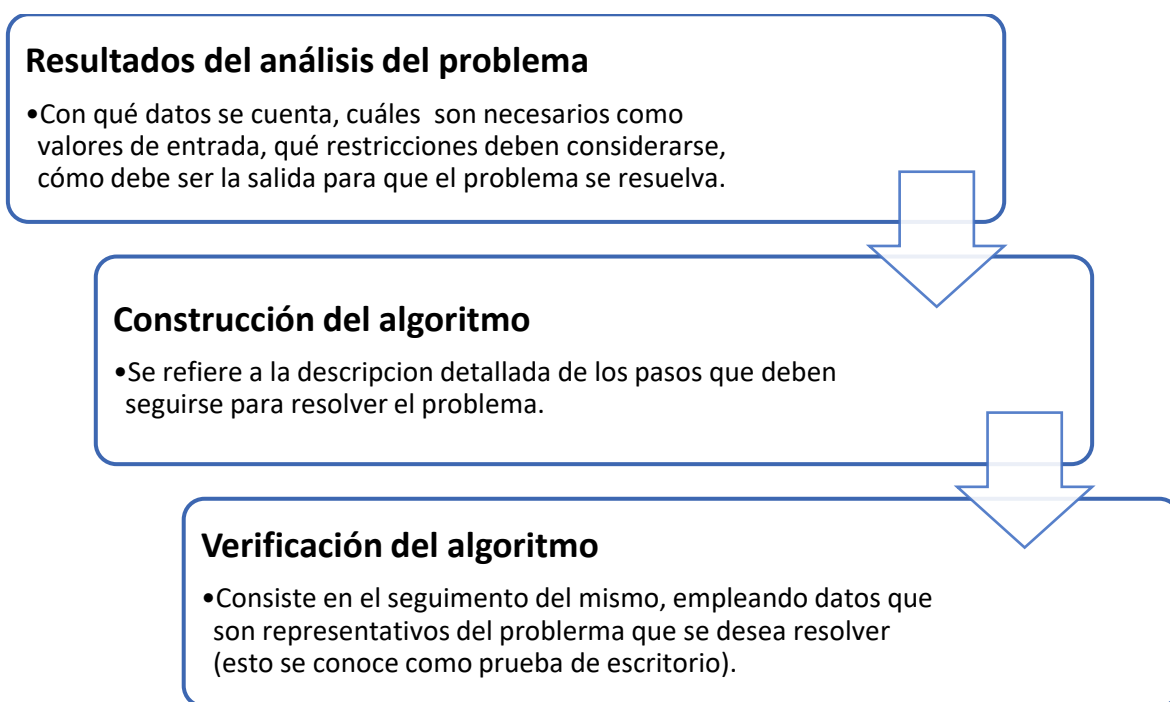



Figura 6. Elaboración de un algoritmo

	Manual de prácticas del Laboratorio de Fundamentos de programación	Código:	MADO-17
		Versión:	03
		Página	10/16
		Sección ISO	8.3
		Fecha de emisión	26 / agosto / 2021
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B	
La impresión de este documento es una copia no controlada			

Un algoritmo consta de 3 módulos básicos (Figura 7):

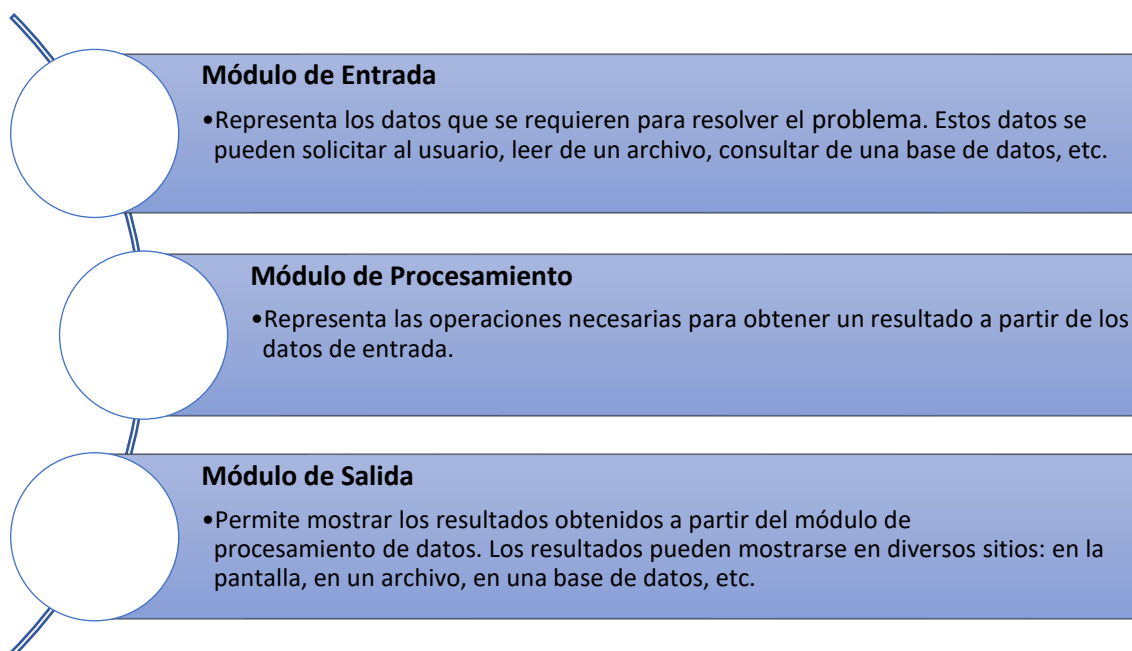


Figura 7. Módulos básicos del algoritmo

Variables

Un algoritmo requiere del uso de *variables* porque guardan el valor (numérico o no numérico) de los datos de entrada; también suelen ser utilizadas para almacenar datos generados en el proceso y datos de salida. Es a través del valor de dichas variables que el algoritmo puede fluir en la secuencia de pasos a seguir.

Ejemplos de algoritmos

Ejemplo 1


PROBLEMA: Determinar si un número dado es positivo o negativo.

RESTRICCIONES: El número no puede ser cero.

DATOS DE ENTRADA: Número real.

DATOS DE SALIDA: La indicación de si el número es positivo o negativo

DOMINIO: Todos los números reales.

	Manual de prácticas del Laboratorio de Fundamentos de programación	Código:	MADO-17
		Versión:	03
		Página	11/16
		Sección ISO	8.3
		Fecha de emisión	26 / agosto / 2021
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B	
La impresión de este documento es una copia no controlada			

SOLUCIÓN:

1. Solicitar un número real y almacenarlo en una variable
2. Si el número ingresado es cero, se regresa al punto 1.
3. Si el número ingresado es diferente de cero, se validan las siguientes condiciones:
 - 3.1 Si el número ingresado es mayor a 0 se puede afirmar que el número es positivo.
 - 3.2 Si el número ingresado es menor a 0 se puede afirmar que el número es negativo.

Prueba de escritorio

El diseño de la solución de un problema implica la creación del algoritmo y la validación de este. La validación se suele realizar mediante una *prueba de escritorio*.


Una prueba de escritorio es una matriz formada por los valores que van adquiriendo cada una de las variables del algoritmo en cada iteración. Una iteración es el número de veces que se ejecuta una parte del algoritmo y permite ver los valores que van adquiriendo las variables en cada repetición.

Para el ejemplo en cuestión, la prueba de escritorio quedaría de la siguiente manera (considerando a X como la variable que almacena el número solicitado):

Iteración	X	Salida
1	5	El número es positivo

Iteración	X	Salida
1	-29	El número es negativo

Iteración	X	Salida
1	0	-
2	0	-
3	0	-
4	100	El número es positivo

	Manual de prácticas del Laboratorio de Fundamentos de programación	Código:	MADO-17
		Versión:	03
		Página	12/16
		Sección ISO	8.3
		Fecha de emisión	26 / agosto / 2021
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B	
La impresión de este documento es una copia no controlada			

Ejemplo 2

PROBLEMA: Obtener el mayor de dos números dados.

RESTRICCIONES: Los números de entrada deben ser diferentes.

DATOS DE ENTRADA: Dos números reales.

DATOS DE SALIDA: La escritura del número más grande.

DOMINIO: Todos los números reales.


SOLUCIÓN:

1. Solicitar un primer número real y almacenarlo en una variable.
2. Solicitar un segundo número real y almacenarlo en otra variable.
3. Si el segundo número real es igual al primer número real, se regresa al punto 2.
4. Si el segundo número real es diferente al primer número real, se validan las siguientes condiciones:
 - 4.1 Si se cumple con la condición de que el primer número es mayor al segundo número, entonces se puede afirmar que el primer número es el mayor de los números.
 - 4.2 Si se cumple con la condición de que el segundo número es mayor al primer número, entonces se puede afirmar que el segundo número es el mayor de los números.

Prueba de escritorio. La variable X almacena el primer número solicitado, y la variable Y almacena el segundo:

Iteración	X	Y	Salida
1	5	6	El segundo número es el mayor de los números

Iteración	X	Y	Salida
1	-99	-222.2	El primer número es el mayor de los números

	Manual de prácticas del Laboratorio de Fundamentos de programación	Código:	MADO-17
		Versión:	03
		Página	13/16
		Sección ISO	8.3
		Fecha de emisión	26 / agosto / 2021
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B	
La impresión de este documento es una copia no controlada			

Iteración	X	Y	Salida
1	15	15	-
2	15	15	-
3	15	15	-
4	15	10	El primer número es el mayor de los números

Ejemplo 3

PROBLEMA: Obtener el factorial de un número dado.

RESTRICCIONES: El número de entrada debe ser entero y no puede ser negativo.

Nota: El factorial de un número está dado por el producto de ese número por cada uno de los números anteriores hasta llegar a 1. La factorial de 0 (0!) es 1.


DATOS DE ENTRADA: Número entero.

DATOS DE SALIDA: El factorial del número.

DOMINIO: Todos los números naturales y el cero.

SOLUCIÓN:

1. Solicitar un número entero y almacenarlo en una variable.
2. Si el número entero es menor a cero regresar al punto 1.
3. Si el número entero es mayor o igual a cero se crea una variable entera *contador* que inicie en 2 y una variable entera *factorial* que inicie en 1.
4. Si la variable *contador* es menor o igual al número entero de entrada se realiza lo siguiente:
 - 4.1 Se multiplica el valor de la variable *contador* con el valor de la variable *factorial*. El resultado se almacena en la variable *factorial*.
 - 4.2 Se incrementa en uno el valor de la variable *contador*.
 - 4.3 Regresar al punto 4.
5. Si la variable *contador* no es menor o igual al número entero de entrada se muestra el resultado almacenado en la variable *factorial*.

	Manual de prácticas del Laboratorio de Fundamentos de programación	Código:	MADO-17
		Versión:	03
		Página	14/16
		Sección ISO	8.3
		Fecha de emisión	26 / agosto / 2021
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B	
La impresión de este documento es una copia no controlada			


Prueba de escritorio. (La variable X almacena el número entero del que se calculará el factorial)

Iteración	X	factorial	contador	Salida
1	0	1	2	El factorial es 1

Iteración	X	factorial	contador	Salida
1	-2	1	2	-
2	-67	1	2	-
3	5	1	2	-
4	5	2	3	-
5	5	6	4	-
6	5	24	5	-
7	5	120	6	El factorial es 120

Iteración	X	factorial	contador	Salida
1	7	1	2	-
2	7	2	3	-
3	7	6	4	-
4	7	24	5	-
5	7	120	6	-
6	7	720	7	-
7	7	5040	8	El factorial es 5040

Si bien se ha ejemplificado la construcción de algoritmos que resuelven problemas numéricos, la construcción de algoritmos no se limita a resolver sólo a este tipo de problemas. A continuación, se presentan ejercicios que ponen a prueba al ejecutor el buen seguimiento del algoritmo para obtener un resultado correcto.

	Manual de prácticas del Laboratorio de Fundamentos de programación	Código:	MADO-17
		Versión:	03
		Página	15/16
		Sección ISO	8.3
		Fecha de emisión	26 / agosto / 2021
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B	
La impresión de este documento es una copia no controlada			

Ejercicio 1

PROBLEMA: Seguir el algoritmo para obtener una figura

ENTRADA: Hoja tamaño carta en limpio, regla y lápiz.

SALIDA: Figura correcta.

Algoritmo

1. Dibuja una V invertida. Empieza desde el lado izquierdo, sube, y baja hacia el lado derecho, no levantes el lápiz.
2. Ahora dibuja una línea en ángulo ascendente hacia la izquierda. Debe cruzar la primera línea más o menos a $1/3$ de la altura. Todavía no levantes el lápiz del papel.
3. Ahora, dibuja una línea horizontal hacia la derecha. Debe cruzar la V invertida más o menos a $2/3$ de la altura total. Sigue sin levantar el lápiz.
4. Dibuja una línea en un ángulo descendente hasta el punto de inicio. Las líneas deben unirse.
5. Ahora ya puedes levantar el lápiz del papel. Has terminado la estrella de 5 puntas.

Ejercicio 2


PROBLEMA: Seguir el algoritmo para obtener una figura

ENTRADA: Hoja tamaño carta en limpio, regla y lápiz.

SALIDA: Figura correcta.

Algoritmo

1. Empieza dibujando un círculo con un compás. Coloca un lápiz en el compás. Coloca la punta del compás en el centro de una hoja de papel.
2. Ahora gira el compás, mientras mantienes la punta apoyada en el papel. El lápiz dibujará un círculo perfecto alrededor de la punta del compás.
3. Marca un punto en la parte superior del círculo con el lápiz. Ahora, coloca la punta del compás en la marca. No cambies el radio del compás con que hiciste el círculo.

	Manual de prácticas del Laboratorio de Fundamentos de programación	Código:	MADO-17
		Versión:	03
		Página	16/16
		Sección ISO	8.3
		Fecha de emisión	26 / agosto / 2021
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B	
La impresión de este documento es una copia no controlada			

4. Gira el compás para hacer una marca en el propio círculo hacia la izquierda. Haz una marca también en el lado derecho.
5. Ahora, coloca la punta del compás en uno de los puntos. Recuerda no cambiar el radio del compás. Haz otra marca en el círculo.
6. Continúa moviendo la punta del compás a las otras marcas, y continúa hasta que tengas 6 marcas a la misma distancia unas de otras. Ahora, ya puedes dejar tu compás a un lado.
7. Usa una regla para crear un triángulo que empiece en la marca superior del círculo. Coloca el lápiz en la marca superior. Ahora dibuja una línea hasta la segunda marca por la izquierda. Dibuja otra línea, ahora hacia la derecha, saltándote la marca de la parte más baja. Complementa el triángulo con una línea hacia la marca superior. Así completarás el triángulo.
8. Crea un segundo triángulo empezando en la marca en la base del círculo. Coloca el lápiz en la marca inferior. Ahora conéctala con la segunda marca hacia la izquierda. Dibuja una línea recta hacia la derecha, saltándote el punto superior. Completa el segundo triángulo dibujando una línea hasta la marca en la parte inferior.
9. Borra el círculo. Has terminado de dibujar tu estrella de 6 puntos.

Referencias

- Raghu Singh (1995). International Standard ISO/IEC 12207 Software Life Cycle Processes. Agosto 23 de 1996, de ISO/IEC. Consulta: Junio de 2015. Disponible en: <http://www.abelia.com/docs/12207cpt.pdf>
- Carlos Guadalupe (2013). Aseguramiento de la calidad del software (SQA). [Figura 1]. Consulta: Junio de 2015. Disponible en: <https://www.mindmeister.com/es/273953719/aseguramiento-de-la-calidad-delsoftware-sqa>
- Andrea S. (2014). Ingeniería de Software. [Figura 2]. Consulta: Junio de 2015. Disponible en: <http://ing-software-verano2014.blogspot.mx>
- Michael Littman. (2012). Intro to Algorithms: Social Network Analysis. Consulta Junio de 2015, de Udacity. Disponible en: <https://www.udacity.com/course/viewer#!/c-cs215/l-48747095/m-48691609>