INSTITUTO SUPERIOR TECNOLÓGICO
VICENTE LEÓN

Guía general de estudio de la asignatura

FUNDAMENTOS DE PROGRAMACIÓN

Alex Fernando Aldaz Corrales

INSTITUTO SUPERIOR TECNOLÓGICO
VICENTE LEÓN

RIMANA EDITORIAL

Carrera de Desarrollo de Software
Fundamentos de Programación
DS03-1B186
Primer nivel

www.istvicenteleon.edu.ec

INSTITUTO SUPERIOR TECNOLÓGICO
VICENTE LEÓN
Belisario Quevedo #501 y Gral. Maldonado / Latacunga – Cotopaxi
Campus Matriz

FUNDAMENTOS DE PROGRAMACIÓN

Autor: Alex Fernando Aldaz Corrales

MSc. Ángel Velásquez Cajas Editor

Directorio editorial institucional

Mg. Omar Sánchez Andrade Rector
Mg. Fabricio Quimba Herrera Vicerrector
Mg. Milton Hidalgo Achig Coordinador de la Unidad de Investigación

Diseño y diagramación

Mg. Alex Zapata Álvarez
Mtr. Leonardo López Lidioma

Revisión técnica de pares académicos

– Ing. Segundo Humberto Corrales Beltrán
Universidad Técnica de Cotopaxi
segundo.corrales@utc.edu.ec
– Ing. Víctor Hugo Medina Matute
Universidad Técnica de Cotopaxi
victor.medina@utc.edu.ec

ISBN: 978-9942-7211-6-7
Primera edición
Diciembre 2023

RIMANA EDITORIAL

Índice

DESARROLLO GUÍA DE ESTUDIO

1. Datos informativos

Alex Fernando Aldaz Corrales, Ingeniero en sistemas computacionales, con un Diplomado Superior en gestión para el aprendizaje universitario y Maestría en Informática Empresarial, ha sido catedrático de la Universidad de las Américas y catedrático del Instituto Superior Vicente León, además laboro en el área de TIC's en empresas públicas y privadas.

2. Presentación de la Asignatura

En la asignatura de fundamentos de la programación, es indispensable las bases técnicas a través de lenguajes de alto nivel de estructura, ya que requiere un proceso de transcripción de códigos en distintas áreas del conocimiento.

Es necesario precisar que los pseudocódigos ayudan a exponer los pasos que va a realizar un programa.

3. Competencias Específicas de la Carrera

– Documenta los requerimientos del sistema con base en un estudio de requerimientos técnicos, operativos y económicos.

– Modela el análisis del sistema según la estructura del diseño lógico planteado.

– Modela información para el diseño eficaz de una base de datos.

– Desarrolla sitios web responsivos mediante el uso de frameworks web.

– Desarrolla aplicaciones móviles con conexión a servicios en la nube, aplicando metodologías acordes a las normativas y estándares vigentes.

– Aplica criterios y estándares vigentes para la gestión e implementación de redes de computación, optimizando operatividad de los recursos de hardware y software.

– Aplica las diferentes normativas y estándares en seguridad informática, así como la legislación relacionada.

4. Introducción de los Temas

Esta guía presenta el siguiente tema de estudio.

La unidad I trata aspectos relevantes referentes a la computadora, sus componentes, ciclos de vida de software, así como la lógica de creación de algoritmos, pseucódigos y diagramas de flujo, que permite a los alumnos el análisis de la lógica de solución a problemas, antes de la codificación a través de las diferentes sentencias (estructuras) de control.

En la presente guía el alumno aprenderá a razonar adecuadamente ante los problemas presentados.

5. Objetivos de Aprendizaje

Al concluir el estudio de esta guía, el alumno será capaz de:

– Identificar las partes del computador, dispositivos de entrada y salida. (hardware – software)

– Conocer el ciclo de vida de desarrollo de software.

– Entender la lógica de programación a través de algoritmos y pseudocódigos.

– Conocer y entender las diferentes estructuras de control.

– Aplicar adecuadamente las diferentes estructuras de control en los problemas planteados.

6. Competencia de Unidad

Diseña e implementa programas a través de software libre y comercial, mediante el uso de tecnología actualizada.

7. Unidad y Subunidades

1. Fundamentos de programación
1.1. El computador y sus partes
1.2. Elementos de un computador
1.2.1. Dispositivos de entrada

8. Resultados de Aprendizaje

– Analiza los problemas planteados y los soluciona a través de pseudocódigos y diagramas de flujos.

– Compara los diagramas de flujos con la resolución de problemas planteados.

– Realiza sentencias e instrucciones de programación usando un lenguaje estructurado.

– Realiza pruebas de la funcionalidad de los algoritmos desarrollados.

– Describe como se aplican las sentencias e instrucciones de programación en la resolución de problemas planteados.

9. Estrategias Metodológicas

Estrategias Metodológicas	Finalidad	Técnicas
Experiencia Concreta	Explora los saberes empíricos con los que llegan sus participantes, a través de lluvias de ideas, preguntas – respuestas, conversatorios.	Investigaciones, observación directa, proyecciones, viaje imaginario sustentado en la práctica real docente, experimentación.
Reflexión	Desde un contexto comunicativo contextualizado a su realidad, plantea el tema utilizando lecturas informativas, gráficos, con el fin de inducir a los alumnos a conectar sus conocimientos previos con la nueva información que se les proporciona.	Lluvia de ideas, diálogos, foros.

Conceptualización	La intervención del docente debe estar dirigida a actividades como la presentación de la nueva información (contenidos curriculares)	Cuadros comparativos, resúmenes, esquemas sintéticos, ilustraciones, análisis, síntesis, procedimientos, etc.
Aplicación	La acumulación del aprendizaje debe reflejar la adquisición de los nuevos contenidos, conectados con los saberes y experiencias anteriores.	Cuadros comparativos, resolución de problemas planteados, elaboración de informes, producción de textos, construcción y solución de cuestionarios, etc.
Los Recursos Didácticos		
Materiales Convencionales	– Libros, documentos, etc. – Tableros didácticos: pizarra y franelógrafo.	
Materiales Audiovisuales	– Fotografías fijas. – Materiales audiovisuales (vídeo): audio y video visuales y vídeos.	
Nuevas Tecnologías	– Programas informáticos (CD u on-line). – Educativos: actividades de aprendizaje, presentaciones, multimedia, animaciones y simulaciones interactivas. – Servicios telemáticos: páginas web, weblogs, aulas virtuales, webquest, unidades didácticas. – Vídeo interactivo.	

10. Criterios de Evaluación

Fases	Instrumentos	Primer Parcial %(puntos)	Segundo Parcial %(puntos)	Promedio %(puntos)
Fase 1: Trabajos Prácticos	Trabajos Individual	2	2	2
	Trabajo de clase o colaborativo	2	2	2
	Exposiciones	2	2	2
Fase 2: Lecciones	Escritas	2	2	2
Fase 3: Evaluación	Cuestionario	2	2	2
	Total:	10	10	10

Horas Docencia= 36
Horas de Trabajo Autónomo= 33
Horas de prácticas experimentales= 31

11. Desarrollo de las Subunidades

Fundamentos de Programación

De acuerdo con la naturaleza del funcionamiento de las computadoras, estas siempre ejecutan órdenes en un formato que les resulta entendible; dichas órdenes se agrupan en programas, conocidos como software.

1.1 El Computador y sus Partes

Según (Ospina, 2006) menciona "Una computadora u ordenador es un dispositivo electrónico desarrollado para ejecutar un conjunto de instrucciones, facilitar el manejo de la información y procesar datos a grandes velocidades".

1.2 Elementos de un Computador

1.2.1 Dispositivos de Entrada

Para Ospina, 2006 generalmente estos componentes:

Se encuentran colocados fuera de la torre del computador y permiten ingresar datos para ser procesados por la CPU. Entre estos se encuentran: el mouse, teclado, lápiz óptico, tablas digitalizadoras, control de juegos o joystick, pantallas sensibles al tacto que permiten el usuario señalar acciones como emplean algunos cajeros electrónicos, cámaras, scanner, micrófonos, entre otros. (pág. 19)

Figura 1

Dispositivos de entrada

Nota. Varios de los dispositivos de entrada que integran el computador. Tomado de curiosodatos.com [figura], ¿Cuántos dispositivos de entrada (ridum.umanizales.edu.co) hay?, 2023, https://curiosodatos.com/cuantos-dispositivos-de-entrada-hay/

1.2.2. Dispositivos de Salida

Según (Ospina, 2006) menciona que estos permiten la visualización de la información en una amplia gama de formatos o modos. Entre estos se encuentran: las impresoras, las pantallas o monitores, parlantes, entre otros. Con ellos se busca la representación de la información de una manera más amena y entendible. (pág. 20)

Figura 2

Dispositivos de salida

Nota. Algunos de los dispositivos de salida. Tomado de tecnologia-informatica.com [figura], Periféricos de Salida, 2023, https://www.tecnologia-informatica.com/perifericos-de-salida/

1.2.3 Dispositivos de Almacenamiento

"A este grupo pertenecen todas las unidades de lectura / escritura de datos. Existen unidades internas y externas a la torre del computador, creadas para dar solución a las diferentes necesidades del usuario, aplicando diferentes tecnologías de almacenamiento" (Ospina, 2006, pág. 20).

Las unidades CD-RW y DVD son tanto internas como externas, las cuales utilizan discos compactos para almacenar datos utilizando tecnologías ópticas para el almacenamiento de datos.

1.3 Partes del Computador

1.3.1 Mouse o Ratón

Para Ospina, 2006 el mouse es:

Un dispositivo apuntador que permite a un usuario de un equipo de cómputo navegar sobre la interfaz gráfica, facilitando el acceso a acciones dentro del ordenador. Algunos mouses emplean aun la tecnología mecánica de rodillos y bolilla para ubicar el cursor del ratón dentro de una interfaz gráfica. (pág. 22)

Figura 3

Mouse

Nota. Dispositivo de entrada y salida. Tomado de static.educalingo.com [figura], Qué significa mouse en español, 2018, https://static.educalingo.com/img/es/800/raton-informatica.

1.3.2 El Teclado

Para (Ospina, 2006) el teclado es el dispositivo básico de entrada de acciones hacia un ordenador presionando una serie de teclas o mediante una combinación de las mismas. Consta con teclas de función numeradas como F1 al F12, un teclado numérico, un teclado alfanumérico, teclas de desplazamiento del cursor e indicadores de paneles activos. (pág. 21)

Figura 4

Teclado

Nota. Teclado del computador. Tomado de co.pinterest.com [figura], El teclado, 2020, https://i.pinimg.com/564x/9c/65/19/9c6519b30035f5e5d62f5ddfa463d319

1.3.3 Monitor

Según (Ospina, 2006, pág. 22) menciona que es el dispositivo principal de salida donde se visualizan las imágenes generadas por el ordenador, conectado mediante un adaptador de video, el cual responde por una gran parte en la calidad de la imagen generada y otra proporcionada por el monitor.

Figura 5

Monitor

Nota. Monitor del computador. Tomado de 247tecno.com [figura], Tipos de monitores para PC, 2020, https://247tecno.com/wp-content/uploads/2018/06/tactil.

Las tecnologías de monitores que se utilizan actualmente son los monitores LED los cuales se componen de luces led capaces de emitir luz. Este tipo de monitores funcionan por medio de módulos monocromáticos capaces de representar visualmente las imágenes generadas en el computador.

1.3.4 Torre del Computador

"La mayoría de usuarios la llaman CPU. Dicho término no está bien empleado ya que CPU corresponde a la Unidad Central de Proceso de la computadora, y la torre es implemente el soporte para una serie de componentes albergados dentro" (Ospina, 2006).

Explorando con detenimiento los elementos contenidos en la torre se encuentran:

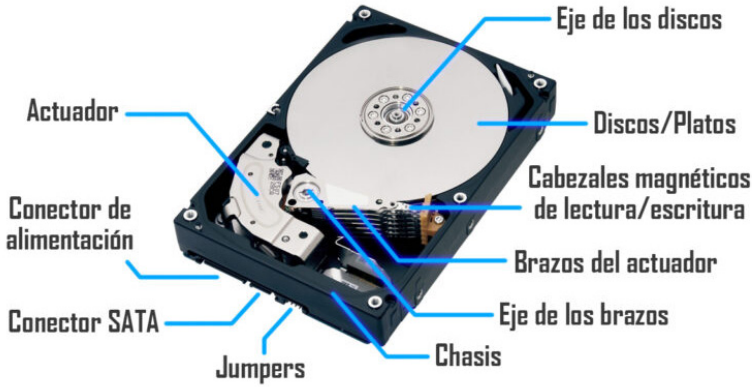
1.3.5 Disco Duro

Es la principal unidad de almacenamiento de los datos de un computador. Su forma interna corresponde a un apilamiento de discos o laminas rígidas de forma circular, albergando los datos mediante procesos magnéticos. Estas láminas giran alrededor de 3600 revoluciones por minuto. Cuentan con cabezas de lectura escritura muy similar a las agujas de los equipos de sonido para el manejo de acetatos, donde este cada uno se posiciona sobre un área del disco para guardar o leer datos.

(Ospina, 2006)

Figura 6

Disco Duro



Nota. Partes de un disco duro. Tomado de tecnotrono.com [figura], Disco duro o HDD, 2020, https://tecnotrono.com/wp-content/uploads/2020/09/Partes-de-los-HDD.jpg.

1.3.6 CPU

La Unidad Central de Proceso (Central Processing Unit, CPU, en inglés), dirige y controla el proceso de información ejecutado por la computadora. La UCP procesa o manipula la información almacenada en memoria; puede

recuperar información desde memoria. Además, puede almacenar los resultados de estos procesos en memoria para su uso posterior. (Aguilar, 2006, pág. 44)

Figura 7

Unidad central de proceso

Nota. La Unidad central de proceso (CPU) integrada en la tarjeta madre. Tomado de www.misaelaleman.com [figura], Unidad Central de Proceso, 2014, https://lh6.googleusercontent.com/-4qx8_zrg0zQ/Us7mkjn9ZbI/AAAAAAAACvw/xqObDZooeks/s405/microprocesador-unidad-central-de-proceso.jpg

La unidad central de proceso contiene la unidad aritmética lógica (ALU) que realiza todo tipo de operaciones lógicas y una serie de registro de almacenamiento temporal para ejecutar operaciones.

1.3.7 Memoria RAM

Según (Ospina, 2006) RAM (memoria de acceso aleatorio) es la memoria de trabajo del ordenador, y se emplea para labores de procesos que se efectúan con el procesador y otros periféricos, guardando datos mientras se estén usando, cuando se finaliza cualquier tarea los espacios de memoria son usados por otra tarea de procesamiento. (pág. 24)

Figura 8

Memoria RAM

Nota. Memoria interna del computador. Tomado de www.accesoriosparacomputadores.co [figura], La Memoria RAM, 2022, https://www.accesoriosparacomputadores.co/blog/wp-content/uploads/2022/02/memoria-ram-800x445.jpg

1.3.8 Tarjeta Madre, Mainboard o Placa Base

Esta unidad electrónico contiene el procesador, la memoria RAM, los buses de datos y otros elementos, es por eso que se denomina comúnmente tarjeta madre, es una gran tarjeta que interconecta todos los elementos contenidos dentro de una torre de computador. (Ospina, 2006, pág. 24)

Figura 9

Tarjeta Madre

Nota. Tarjeta madre. Tomado de informaticaavanzada2013.blogspot.com [figura], La Tarjeta Madre, 2014, https://www.accesoriosparacomputadores.co/blog/wp-content/uploads/2022/02/memoria-ram-800x445.jpg

1.3.9 Buses de Datos

"Son los encargados de conectar cada dispositivo dentro de la torre del computador con la tarjeta madre. Las labores de envió y recibo de datos a través de los buses o correas de datos, son administradas por el microprocesador" (Eckel, 2011, pág. 24).

Figura 10

Buses de datos

Nota. Buses de datos para conectar periféricos. Tomado de conceptoabc.com [figura], Bus de datos, 2021, https://conceptoabc.com/wp-content/uploads/2021/05/tecno-bus-SATA.jpg.

1.3.10 Unidades de CD-ROM

Es un dispositivo de lectura en un disco compacto empleando tecnología de láser óptico.

1.3.11 Unidades de CD-RW

"Dispositivo de lectura / escritura en un disco compacto con una capacidad de almacenamiento de alrededor de 650 megas bytes". (Ospina, 2006)

Figura 11
Unidades de CD-RW

Nota. Unidades ópticas. Tomado de compusoftwareusco.webnode.com.co [figura], Unidades de CD-RW, 2021, https://e97817d9aa.cbaul cdnwnd.com/3f8dadb8554ee-ce00e2ee2119d295a03/200000073-b5665b6624/unidades.jpg.

1.4 Software

Según (Ospina, 2006) menciona que el software es la parte lógica de un sistema de cómputo, no es tangible, el software abarca todo tipo de aplicaciones (programas) que sirven para controlar, manipular, capturar datos, que luego son expuestos como información entendible para el usuario. (pág. 18)

Para conocer de una manera diferente todos los tipos de programas que tiene en su ordenador, se desarrollará etapa por etapa el simple proceso desde encender su ordenador y dejarlo listo para ejecutar cualquier aplicación.

Presionando el botón de encendido de su sistema de cómputo suceden las siguientes operaciones: existe un programa residente en un chip que se encarga de verificar y chequear la configuración "hardware" de su máquina, por esto, cuando enciende su máquina se muestra en el monitor ciertos textos que muestran información de su computadora. Este chip se conoce como la BIOS. (Ospina, 2006, pág. 26)

Una vez finalizada la tarea de chequeo se inicia la ejecución del sistema operativo.

Este programa es el encargado de administrar todos los recursos de su máquina. El sistema operativo más popular mundialmente es Windows,

desarrollado por la empresa Microsoft Corporation. Pero existen otras alternativas como está el sistema operativo Linux apoyado por la organización de software libre Gnu2, en cada una de sus diferentes distribuciones y para los usuarios de equipos Mac esta MacOs2. La labor de cualquier sistema operativo en este proceso de arranque es preparar y verificar el estado del hardware ya combinado con algunas aplicaciones para el usuario pueda empezar a interactuar y sacar el máximo provecho. (Ospina, 2006, pág. 26)

En resumen, para utilizar de todo el hardware de un computador es necesario el uso de programas que permitan interactuar directamente de una manera discreta.

1.5 Ciclo de Vida de un Software

"El ciclo de vida es el conjunto de fases por las que pasa el sistema que se está desarrollando desde que nace la idea inicial hasta que el software es retirado o remplazado (muere). También se denomina a veces paradigma" (INTECO, 2009, pág. 23).

Entre las funciones que debe tener un ciclo de vida se pueden destacar:

– Determinar el orden de las fases del proceso de software.

– Establecer los criterios de transición para pasar de una fase a la siguiente.

– Definir las entradas y salidas de cada fase.

– Describir los estados por los que pasa el producto.

– Describir las actividades a realizar para transformar el producto.

– Definir un esquema que sirve como base para planificar, organizar, coordinar, desarrollar.

Un ciclo de vida para un proyecto se compone de fases sucesivas compuestas por tareas que se pueden planificar. Según el modelo de ciclo de vida, la sucesión de fases puede ampliarse con bucles de realimentación, de manera que lo que conceptualmente se considera una misma fase se pueda ejecutar más de una vez a lo largo de un proyecto, recibiendo en cada pasada de ejecución aportaciones a los resultados intermedios que se van produciendo (realimentación).

Fases: Las fases son conjunto de actividades relacionadas con un objetivo en el desarrollo del proyecto. Se realiza agrupando tareas (actividades elementales) que pueden compartir un tramo determinado del tiempo de vida de un proyecto. La agrupación temporal de tareas impone requisitos temporales correspondientes a la asignación de recursos (humanos, financieros o materiales). (Díaz y otros, 2013, pág. 25)

Entregables: Son los productos intermedios que generan las fases. Pueden ser materiales o inmateriales (documentos, software). Los entregables permiten evaluar la marcha del proyecto mediante comprobaciones de su adecuación o no a los requisitos funcionales y de condiciones de realización previamente establecidos.

Las actividades del ciclo de vida del desarrollo del software son:

– Especificación: lo que el sistema debería hacer y sus restricciones de desarrollo.

– Desarrollo: producción del sistema software.

– Validación: comprobar que el sistema es lo que el cliente quiere.

– Evolución: cambiar el software en respuesta a las demandas de cambio.

Figura 12

Desarrollo a medida

Nota. Software en respuesta a las demandas de cambio. Tomado de plantl.mineco.gob.es [figura], Desarrollo a medida, 2021, http://ww25.agile-spain.com/?subid1=20230728-0254-2855-9fed-1f987376d050.

1.6 Modelos de Ciclo de Vida del Software

Según (Vacas, 2009) menciona que:

La ingeniería del software se vale de una serie de modelos que establecen y muestran las distintas etapas y estados por los que pasa un producto software, desde su concepción inicial, pasando por su desarrollo, puesta en marcha y posterior mantenimiento, hasta la retirada del producto. A estos modelos se les denomina "Modelos de ciclo de vida del software". El primer modelo concebido fue el de Royce, más comúnmente conocido como "Cascada" o "Lineal Secuencial". Este modelo establece que las diversas actividades que se van realizando al desarrollar un producto software, se suceden de forma lineal. (pág. 325)

Los modelos de ciclo de vida del software describen las fases del ciclo de software y el orden en que se ejecutan las fases.

Un modelo de ciclo de vida de software es una vista de las actividades que ocurren durante el desarrollo de software, intenta determinar el orden de las etapas involucradas y los criterios de transición asociados entre estas etapas.

Un modelo de ciclo de vida del software:

– Describe las fases principales de desarrollo de software.
– Define las fases primarias esperadas de ser ejecutadas durante esas fases.
– Ayuda a administrar el progreso del desarrollo.
– Provee un espacio de trabajo para la definición de un proceso detallado de desarrollo de software.

Según (Vacas, 2009), las principales diferencias entre distintos modelos de ciclo de vida están en:

– El alcance del ciclo dependiendo de hasta dónde llegue el proyecto correspondiente. Un proyecto puede comprender un simple estudio de

viabilidad del desarrollo de un producto, o su desarrollo completo o en el extremo, toda la historia del producto con su desarrollo, fabricación y modificaciones posteriores hasta su retirada del mercado.

– Las características (contenidos) de las fases en que dividen el ciclo. Esto puede depender del propio tema al que se refiere el proyecto, o de la organización.

– La estructura y la sucesión de las etapas, si hay realimentación entre ellas, y si tenemos libertad de repetirlas (iterar).

1.6.1 Modelo en Cascada

"Es una guía metodológica que ordena las etapas del ciclo de vida del software, de forma que el inicio de cada etapa debe esperar a la finalización de la inmediatamente anterior" (INTECO, 2009).

El modelo en cascada es un proceso de desarrollo secuencial, en el que el desarrollo se ve fluyendo hacia abajo (como una cascada) sobre las fases que componen el ciclo de vida.

La primera descripción formal del modelo en cascada se cree que fue en un artículo publicado en 1970 por Winston W. Royce, aunque Royce no usó el término cascada en este artículo. Royce estaba presentando este modelo como un ejemplo de modelo que no funcionaba, defectuoso.

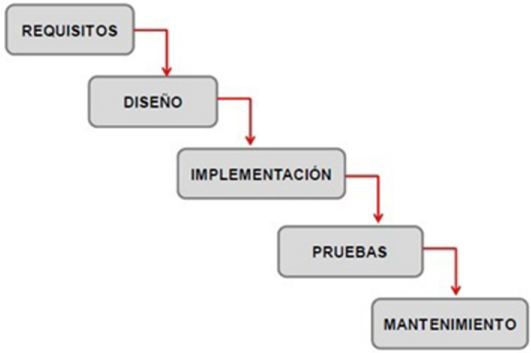
En el modelo original de Royce, existían las siguientes fases:

1. Especificación de requisitos
2. Diseño
3. Construcción (Implementación o codificación)
4. Integración
5. Pruebas
6. Instalación
7. Mantenimiento

Para utilizar el modelo en cascada, se avanza de una fase a la siguiente en una forma secuencial.

Figura 13

Modelo de ciclo de vida en cascada



Nota. Modelo en cascada del desarrollo de software. Tomado de www.researchgate.net [figura], Modelo de ciclo de vida en cascada. 2016, https://www.researchgate.net/profile/Arturo-Cordova Rangel/publication/310512230/figure/fig11/AS:430326523666434@1479609246539/Figura-1-Modelo-de-ciclo-de-vida-en-cascada.png.

Si bien ha sido ampliamente criticado desde el ámbito académico y la industria, sigue siendo el paradigma más seguido a día de hoy.

1.6.2 Modelo en V

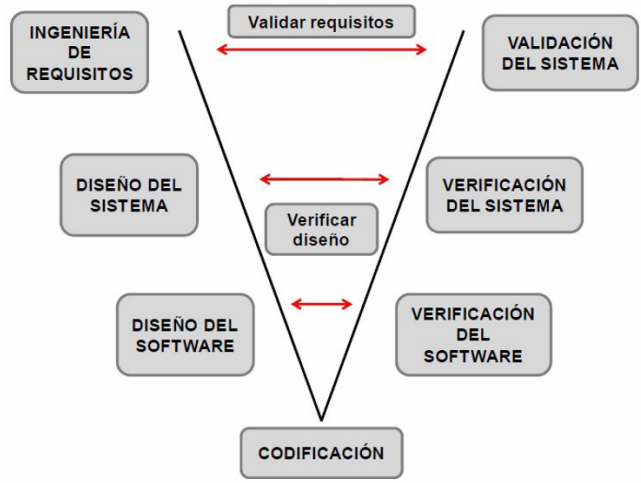
Según (INTECO, 2009) menciona:

El modelo en v se desarrolló para terminar con algunos de los problemas que se vieron utilizando el enfoque de cascada tradicional. Los defectos estaban siendo encontrados demasiado tarde en el ciclo de vida, ya que las pruebas no se introducían hasta el final del proyecto. El modelo en v dice que las pruebas necesitan empezarse lo más pronto posible en el ciclo de vida. También muestra que las pruebas no son sólo una actividad basada en la ejecución. Hay una variedad de actividades que se han de realizar antes del fin de la fase de codificación. Estas actividades deberían ser llevadas a cabo en paralelo con las actividades de desarrollo, y los técnicos de pruebas necesitan trabajar con los desarrolladores y analistas de negocio de tal forma que puedan realizar estas actividades y tareas y producir una serie de entregables de pruebas. (pág. 27)

El modelo en V es un proceso que incorpora la secuencia de pasos en el desarrollo del ciclo de vida de un proyecto. Indica las actividades y resultados que se realizan durante el desarrollo del producto. La parte izquierda de la V indica la descomposición de los requisitos y la creación de las especificaciones del sistema. El lado derecho de la V indica la integración de partes y su verificación. V significa "Validación y Verificación".

Figura 14

Modelo de ciclo de vida en V



Nota. El modelo en V es un proceso que indica la secuencia de pasos en el desarrollo del ciclo de vida de un software. Tomado de www.efectodigital.online [figura], Modelo V, 2018,https://static.wixstatic.com/media/fd2cc9_122b8f0171ed42a9b980021be2a-1decf~mv2.png/v1/fill/w_740,h_567,al_c,lg_1,q_90,enc_auto/fd2cc9_122b8f0171e-d42a9b980021be2a1decf~mv2.png.

(Vacas, 2009) Menciona que realmente las etapas individuales del proceso pueden ser similares a las del modelo en cascada. Sin embargo, hay una gran diferencia. En vez de ir para abajo de una forma lineal las fases del proceso vuelven hacia arriba tras la fase de codificación, formando una v. La razón de esto es que para cada una de las fases de diseño se ha encontrado que hay un homólogo en las fases de pruebas que se correlacionan. (pág. 324)

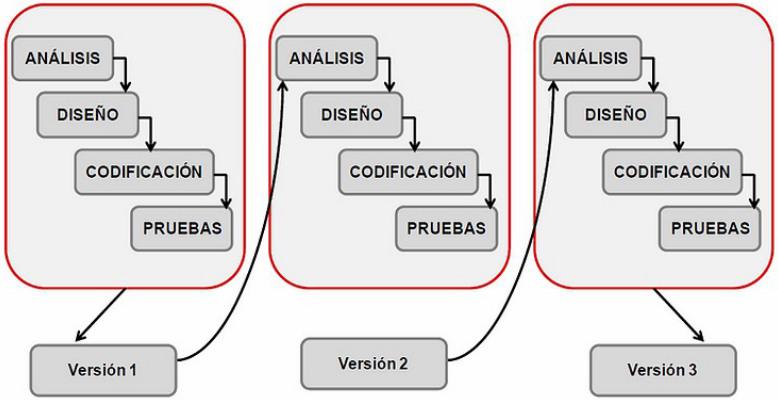
1.6.3. Modelo Iterativo

Es un modelo procedente del ciclo de vida en cascada. "Modelo el cual busca reducir el riesgo que surge entre las necesidades del usuario y el producto final durante la etapa de recogida de requisitos" (Díaz y otros, 2013).

Consiste en la iteración de varios ciclos de vida en cascada. Al final de cada iteración se le entrega al cliente una versión mejorada del producto. El cliente es quien, después de cada iteración, evalúa el producto y lo corrige o propone mejoras. Estas iteraciones se producirán hasta obtener un producto que satisfaga las necesidades del cliente.

Figura 15

Modelo de ciclo de vida iterativo



Nota. Modelo que radica en la iteración de varios ciclos de vida en cascada. Tomado de www.efectodigital.online [figura], Modelo V, 2018, https://static.wixstatic.com/media/fd2cc9_410fd994bd3d4262a5dfd5f6c174191f~mv2.png/v1/fill/w_740,h_380,al_c,q_85,usm_0.66_1.00_0.01,enc_auto/fd2cc9_410fd994bd3d4262a5dfd5f6c-174191f~mv2.png

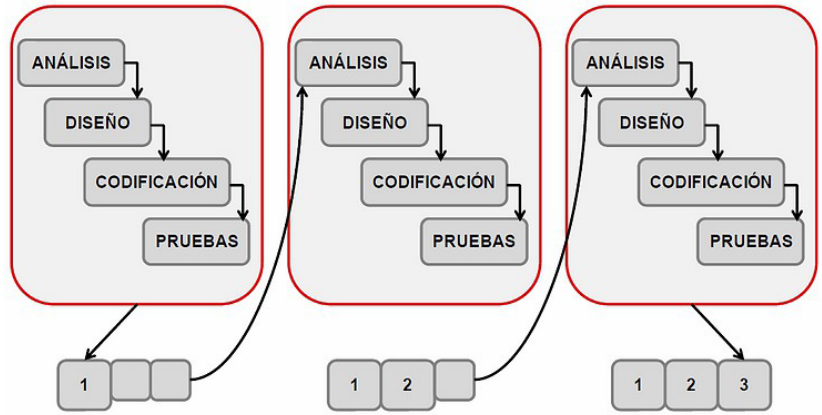
Este modelo se suele utilizar en proyectos en los que los requisitos no están claros por parte del usuario, por lo que se hace necesaria la creación de distintos prototipos para presentarlos y conseguir la conformidad del cliente.

1.6.4 Modelo de Desarrollo Incremental

El modelo incremental combina elementos del modelo en cascada con la filosofía participativa de construcción de prototipos. Se basa en la filosofía de construir incrementando las funcionalidades del programa. Este modelo aplica secuencias lineales de forma escalonada mientras progresa el tiempo en el calendario. Cada secuencia lineal produce un incremento del software. (INTECO, 2009, pág. 28)

Figura 16

Modelo de ciclo de vida incremental



Nota. Modelo de desarrollo incremental. Tomado de www.efectodigital.online [figura], Modelo de desarrollo incremental, 2018, https://static.wixstatic.com/media/fd2cc9_2f5b70ca261a483eb32ae647c83bf5b5~mv2.png/v1/fill/w_740,h_382,al_c,q_85,usm_0.66_1.00_0.01,enc_auto/fd2cc9_2f5b70ca261a483eb32ae647c83bf5b5~mv2.

Cuando se utiliza un modelo incremental, el primer incremento es un producto esencial, sólo con los requisitos básicos. Este modelo se concentra en la entrega de un producto operativo con cada incremento. Los primeros incrementos son versiones incompletas del producto final, pero proporcionan al usuario la funcionalidad que precisa y también una plataforma para la evaluación.

1.6.5 Modelo en Espiral

(Díaz y otros, 2013) mencionan que el desarrollo del modelo en espiral, es un modelo de ciclo de vida desarrollado por Barry Boehm en 1985, utilizado de forma generalizada en la ingeniería del software. Las actividades de este modelo se realizan en una espiral, cada bucle representa un conjunto de actividades. Las actividades no están fijadas a priori, sino que las siguientes se eligen en función del análisis de riesgos, comenzando por el bucle anterior. (pág. 240)

Al ser un modelo de ciclo de vida orientado a la gestión de riesgos se dice que uno de los aspectos fundamentales de su éxito radica en que el equipo que lo aplique tenga la necesaria experiencia y habilidad para detectar y catalogar correctamente riesgos.

Tareas: Para cada ciclo habrá cuatro actividades:

1. Determinar o fijar objetivos:
– Fijar también los productos definidos a obtener: requerimientos, especificación, manual de usuario.
– Fijar las restricciones.
– Identificar riesgos del proyecto y estrategias alternativas para evitarlos.
– Hay una cosa que solo se hace una vez: planificación inicial o previa.

2. Análisis del riesgo:
– Estudiar todos los riesgos potenciales y se seleccionan una o varias alternativas propuestas para reducir o eliminar los riesgos.

3. Desarrollar, verificar y validar (probar):
– Tareas de la actividad propia y de prueba.
– Identificación de resolución de riesgos.
– Una vez obtenido el resultado de la evaluación de riesgos, se elige un modelo para el desarrollo, que puede ser cualquiera de los otros existentes, como formal, evolutivo, cascada, etc. Así, por ejemplo, si los riesgos de la interfaz de usuario son dominantes, un modelo de desarrollo apropiado podría ser la construcción de prototipos evolutivos.

4. Planificar:

– Revisar todo lo que se ha llevado a cabo, evaluándolo y decidiendo si se continúa con las fases siguientes y planificando la próxima actividad.

El proceso empieza en la posición central. Desde allí se mueve en el sentido de las agujas del reloj.

Figura 17

Ciclo de vida en espiral

Nota. En la figura cada bucle representa un conjunto de actividades Tomado de www.efectodigital.online [figura], Modelo en espiral, 2018, https://static.wixstatic.com/media/fd2cc9_ed1284d21a354363a2eff370c4823a2c~mv2.png/v1/fill/w_740,h_461,al_c,q_85,usm_0.66_1.00_0.01,enc_auto/fd2cc9_ed1284d21a354363a2eff370c4823a2c~mv2.png

1.6.6 Modelo de Prototipos

Según (Díaz y otros, 2013) mencionan que:

El paradigma de construcción de prototipos inicia con la recolección de requisitos. El desarrollador y el cliente encuentran y definen los objetivos globales para el software, identifican los requisitos conocidos y las áreas del esquema en donde es obligatoria más definición. Entonces aparece un diseño

rápido. El diseño rápido se centra en una representación de esos aspectos del software que serán visibles para el usuario/cliente. El diseño rápido lleva a la construcción de un prototipo. El prototipo lo evalúa el cliente/usuario y se utiliza para refinar los requisitos del software a desarrollar. La iteración ocurre cuando el prototipo se pone a punto para satisfacer las necesidades del cliente, permitiendo al mismo tiempo que el desarrollador comprenda mejor lo que se necesita hacer. (pág. 55)

Comparativa de los modelos de ciclo de vida

A continuación, se van a mostrar las ventajas e inconvenientes más significativas de cada uno de los modelos de ciclo de vida:

Tabla 1

Descripción de áreas de proceso de gestión de procesos

	Ventajas	Inconvenientes
Modelo en cascada	– Modelo en el que está todo bien organizado. – No se mezclan las fases. – Simple y fácil de llevar a la práctica. – Fácil de gestionar.	– Rara vez los proyectos siguen una secuencia lineal. – Difícil establecer todos los requisitos al principio. – Visibilidad del producto cuando está terminado.
Modelo en V	– Simple y fácil de llevar a la práctica. – En cada una de las fases hay entregables específicos. – Desarrollo de planes de prueba en etapas tempranas del ciclo de vida. – Suele funcionar bien para proyectos pequeños donde los requisitos son entendidos fácilmente.	– Tiene poca flexibilidad y ajustar el alcance es difícil y caro. – El modelo no proporciona caminos claros para problemas encontrados durante las fases de pruebas.

Modelo incremental	– Se genera software operativo de forma rápida y en etapas tempranas del ciclo de vida del software. – Modelo más flexible, por lo que se reduce el coste en cambios de alcance y requisitos. – Es más fácil probar y depurar en una iteración más pequeña. – Es más fácil gestionar riesgos. – Cada iteración es un hito gestionado fácilmente.	– Se requiere mucha experiencia para definir los incrementos y distribuir en ellos las tareas de forma proporcionada. – Cada fase de una iteración es rígida y no se superpone con otras. – Todos los requisitos han de definirse al inicio.
Modelo iterativo	– No hace falta que los requisitos estén totalmente definidos desde el principio. – Desarrollo en pequeños ciclos. – Es más fácil gestionar riesgos. – Cada iteración es un hito gestionado fácilmente.	– Que los requisitos no estén definidos desde el principio también puede verse como un inconveniente ya que pueden surgir problemas con la arquitectura.
Modelo de prototipos	– Visibilidad del producto desde el inicio del ciclo de vida con el primer prototipo. – Permite introducir cambios en las iteraciones siguientes del ciclo. – Permite la realimentación continua del cliente.	– Puede ser un desarrollo lento.

Modelo en espiral	– Reduce riesgos del proyecto. – Incorpora objetivos de calidad. – Integra el desarrollo con el mantenimiento. – No es rígido ni estático. – Se produce software en etapas tempranas del ciclo de vida.	– Modelo que genera mucho trabajo adicional. – Exige un alto nivel de experiencia y cierta habilidad en los analistas de riesgos. – Modelo costoso.

Nota. Ventajas e inconvenientes de los modelos.

1.7 Lógica de Programación

Más que acumular conocimientos, el alumno debe aprender a resolver problemas, debe desarrollar habilidades para establecer un marco de trabajo que le sea útil en la resolución de los mismos. Este marco comprende la capacidad de entender plenamente el enunciado, saber cómo organizarse para buscar información e interpretar los resultados.

1.7.1 Algoritmos

"Es un método para resolver un problema, conformado por una secuencia finita de pasos ordenados de manera lógica que se deben seguir para resolver un problema" (Figueroa Piscoya y otros, 2021).

Un algoritmo cumple con las siguientes características:

– Preciso: Indicar el orden de cada paso de manera clara y sin ambigüedades.

– Definido: Se debe obtener el mismo resultado cada vez que se ejecute el algoritmo.

– Finito: Debe terminar en algún momento.

Según (Mathieu, 2014) indica que:

Un algoritmo constituye una lista bien definida, ordenada y finita de operaciones, que permite encontrar la solución a un problema determinado.

Dado un estado inicial y una entrada, es a través de pasos sucesivos y bien definidos que se llega a un estado final, en el que se obtiene una solución (si hay varias) o la solución (si es única). (pág. 13)

Ejemplo:

Problema: Gestionar la lista de compras que una empresa realiza durante un mes.

Un algoritmo puede ser expresado en:

– Lenguaje natural (a veces, este no resulta muy claro, pero es muy útil para problemas simples)
– Pseudocódigo
– Diagramas de flujo
– Programas

El uso de algún elemento de la lista anterior para la expresión de un algoritmo, se hace según el nivel de descripción de dicho algoritmo. Es evidente que el lenguaje natural es de mayor utilidad para transmitir las ideas del algoritmo. Al contrario, un programa es difícil de entender por simple lectura, aun por una persona que conoce el lenguaje del programa, e imposible para aquellas que no lo conocen.

Los algoritmos se pueden presentar no solamente ante problemas matemáticos complejos, sino también con problemas cotidianos, por ejemplo, suponga que deseamos ir a ver la nueva película de Marvel, entonces nos planteamos la siguiente pregunta:

¿Qué hacer para ver la nueva película de superhéroes de Marvel?
La respuesta es muy sencilla y puede ser descrita de la siguiente forma:

– Ir al cine.
– Comprar una entrada (o ticket).
– Ver la película.
– Regresar a casa.

Esto representa claramente un algoritmo que consta de cuatro acciones básicas, cada una de las cuales debe ser ejecutada antes de realizar la siguiente.

La forma en que se respondió a la pregunta anterior se llama Algoritmo en Lenguaje Natural y como su nombre lo indica se puede redactar en el idioma que más domine el estudiante (en este caso español). Pero además le agregaremos una numeración correlativa iniciando en el paso cero (0. Inicio) y terminando siempre con el paso n (n. Fin), donde n corresponde al último orden de la numeración. De esta manera el algoritmo anterior queda de la siguiente manera:

0. Inicio
1. Ir al cine.
2. Comprar una entrada (o ticket).
3. Ver la película.
4. Regresar a casa.
5. Fin

Pero esta no es la única forma de responder a la pregunta, es decir, pueden existir diferentes formas de resolver el mismo problema, es allí donde el ingenio y la experiencia del estudiante toman gran importancia. Recuerda que mientras más detallado y preciso sea tu algoritmo, será mucho mejor la solución.

De esta manera, en el mismo ejemplo de ir al cine podrían haber surgido diferentes escenarios, como que todas las salas donde transmiten esa película ya están llenas lo cual requerirá tomar otras acciones, o al comprar una entrada, verificar si hay cola, sino optar por una compra en línea por medio de la página web del cine o por su aplicación móvil, etc.

1.7.2 Etapas de Desarrollo de un Algoritmo

(Mathieu, 2014) señala que las etapas de desarrollo de un algoritmo, con base en la lógica, son las siguientes:

1. Definición: En esta etapa se especifica el propósito del algoritmo y se ofrece una definición clara del problema por resolver. Además, aquí también se establece lo que se pretende lograr con su solución.

2. Análisis: En este punto se analiza el problema y sus características, y se determinan las entradas y salidas del problema. De igual modo, también se realiza una investigación sobre si ya se conoce alguna o varias soluciones de este. En el caso de que ya se conozcan varias soluciones, entonces se determina cuál es la más conveniente para el problema que estamos tratando. Si no se conoce ninguna, o no nos satisfacen las soluciones existentes, se propone una nueva.

3. Diseño: Aquí es donde se plasma la solución del problema. Con ese fin, se emplea una herramienta de diseño, que consiste en el diagrama de flujo y el pseudocódigo.

4. Implementación: En este último paso es donde se realiza o se ve concretado el programa y, por ende, se hacen varias pruebas.

En cada una de las etapas especificadas antes, se utiliza un tipo de descripción conveniente e inteligible para cada uno de los participantes en el proceso de concepción y realización del algoritmo. Hoy en día, existe una rama de las ciencias de la computación que se ocupa del manejo de los proyectos de desarrollo de programas: la ingeniería de software

1.7.3 Tipos de Datos

(Ospina, 2006) indica que el objetivo principal de todo programa para computador es procesar información, la cual es clasificada según su característica o naturaleza, como: datos personales, monetarios, financieros, estadísticos, entre otros. Los tipos de datos proporcionan una variada gama de modelos de datos, más adecuados para albergar esa información dependiendo de su singularidad. Observe la siguiente analogía, no es correcto parquear un camión de carga, en donde se puede parquearun automóvil pequeño como un compacto. De esa misma manera; no es correcto alojar el nombre de una persona en un tipo de dato, en donde se pueden almacenar datos numéricos. (pág. 66)

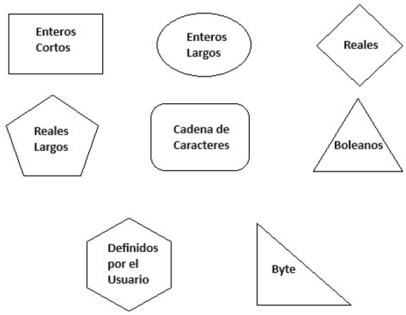
Los tipos de datos según (Patiño, 2017) indica que se crearon para representar la información que se encuentra en la naturaleza, como: datos de temperatura, estatura de una persona, edad, altura en una manejable por los

sistemas de cómputo. Un programa de computador representa o simula un comportamiento humano, social, matemático.

Definiendo lo anterior en otros términos se tiene: cuando se maneja y procesa información lo más adecuado es que dicha información no pierda sus características y su forma. Los sistemas implementados en computadoras deben permitir dicha conservación de características y para ello ofrecen modelos de datos. Entonces, los modelos de datos de forma homóloga a una plantilla de figuras geométricas, se construyeron con la finalidad de proporcionar varias formas de las cuáles el usuario podrá disponer cada vez que las necesite y adaptarlas a la naturaleza de su uso.

Figura 18

Tipos de datos



Nota. Plantilla de modelos o tipos de datos.

Datos Tipo Numérico

Como su nombre lo indica, son tipos de datos en los cuáles se puede almacenar y representar números y sobre ellos se pueden efectuar operaciones o cálculos matemáticos. Los datos de tipo numérico se clasifican en dos categorías distintas: los numéricos enteros y los numéricos reales.

Tipo Numérico Entero

Este tipo de dato permite acumular cantidades enteras o cantidades sin valores decimales, y pueden ser positivos o negativos, excepto los datos

de tipo Byte. Existe además, una subclasificación de este tipo de dato en Byte, enteros cortos y enteros largos. Lo anterior con el fin de proporcionar un tipo de dato más al acomodo de las necesidades en el manejo de cantidades enteras.

Byte

Este tipo de dato puede manejar cantidades en el rango de valores: 0 a 255. Por lo tanto, una cantidad de tipo Byte no puede ser menor de 0, y mayor que 255.

Ejemplos de este tipo de dato son: 245 13 50

Enteros Cortos

Este tipo de dato puede manejar cantidades en el rango de valores: -32768 a 32767. Por lo tanto, una cantidad entera corta no puede ser menor de –32768, y mayor que 32767.

Ejemplos de este tipo de dato son: 245 -8 25000 7894 -30000

Enteros Largos

Este tipo de dato puede manejar cantidades en el rango de valores: -2147483648 a 2147483647. Por lo tanto, una cantidad entera larga no puede ser menor de – 2147483648, y mayor que 2147483647.

Ejemplos de este tipo de dato son: 40000 -8 -2000000 5000000 -45792

Tipo Numérico Real

Este tipo de dato permite acumular cantidades reales o cantidades con valores decimales, y pueden ser positivos o negativos. Existe además, una subclasificación de este tipo de dato en reales cortos y reales largos.

Reales Cortos

Este tipo de dato puede manejar cantidades en el rango de valores: -3.40E+38 a 3.40E+38. Por lo tanto, una cantidad real corta no puede ser menor de –3.40E+38, y mayor que 3.40E+38.

Ejemplos de este tipo de dato son: 245.68 -8.23 25000.1245 7.894 -30000.478

Reales Largos

Este tipo de dato puede manejar cantidades en el rango de valores: -1.79D+308 a 1.79D+308. Por lo tanto, una cantidad real larga no puede ser menor de –1.79D+308, y mayor que 1.79D+308.

Ejemplos de este tipo de dato son: 40000.12455665 -8.2654878787 50000.14898211542

Datos Tipo Lógico

Este tipo de dato puede albergar uno de dos valores, Verdadero o Falso. Se emplea a menudo en condiciones, banderas, entre otros usos más comunes.

Ejemplos de este tipo de dato: Verdadero, Falso.

Datos de Tipo Cadena de Caracteres

Se conocen como cadena de caracteres aquellos elementos que albergan uno o más símbolos alfabéticos, numéricos y especiales de uso regular, y se encuentran contenidos entre comillas simples (´´. A continuación, se mostrarán ejemplos de cada tipo:

Símbolos alfabéticos: el alfabeto (a,b,...y,z) (A,B,...,Y,Z).
Símbolos numéricos: los números (0,1,2,...,8,9).
Símbolos especiales: (¡, @, #, $,..., &,...)

Una cadena de caracteres puede estar conformada por solo símbolos alfabéticos, o numéricos o especiales.

Tipos de Datos Definidos por el Usuario

A partir de los anteriores tipos de datos preestablecidos (enteros cortos, enteros largos, reales cortos, entre otros), se pueden definir otros tantos

mediante una serie de combinaciones. Esta característica brinda una amplia gama de posibilidades para resolver un algoritmo. Estos tipos de datos podrían clasificarse en:

– Arreglos (Vectores, Matrices.)
– Estructuras de datos.
– Punteros.

1.7.4 Variables y Constantes

Variable

Según (Ospina, 2006) señala que "La variable es un espacio reservado en memoria que recibe un nombre representativo, donde se almacena un dato de cualquier tipo mencionado anteriormente. Las variables poseen la característica de cambiar de contenido sobre la ejecución de un algoritmo" (pág. 71).

Además, una variable solo puede almacenar información de un tipo de dato, en la cual se almacenan datos numéricos, o cadenas de caracteres, o datos de tipo lógico. Nunca una misma variable puede almacenar datos de diferentes tipos en un propio algoritmo.

Reglas para Nombrar Variables

El nombre de una variable debe comenzar con una letra, seguida de otras letras, números o caracteres. Como ejemplo:

A, valor, num1, valor_num1, Nombre, ciudad, potencia.

El nombre de las variables nunca debe usar caracteres especiales como:

¡, !, |, ", @, ·, #, $, %, /, (,), =, ', ¿, ?, +, -, *, {, }, [,], ^, entre otros.

Cuando se nombre una variable, nunca emplee tildes, ni espacios, ni signos de puntuación. Por ejemplo:

Usos incorrectos

Código Nombre Empleado Cantidad.

Usos correctos

Codigo Nombre Empleado Cantidad

En las variables no se puede tener nombres empleados por palabras reservadas.

Con lo anterior se concluye que las variables son espacios reservados en memoria, definidos sobre un modelo de dato que indica que tipo de valor puede ser almacenado en cada una de ellas. También que cada una de ellas recibe un nombre simbólico con el cuál pueda ser usado.

El formato para la declaración de variables es el siguiente:

Declaracion Nombre_Variable

Por ejemplo:

Declaracion Numero

Otra forma de declarar variables es también:

Declaración Nombre, Salario

Donde se indica que Nombre y Salario son dos variables independientes una de la otra, pero declaradas sobre la misma instrucción de Declaración.

1.7.5 Clasificación de las Variables según su Función

Existen ciertos tipos de variables que cumplen una función específica dentro del proceso y se utiliza teniendo en cuenta ciertas reglas que se deben respetar.

Según (Patiño, 2017) los tipos de esta clase de variables más comunes son:

Contador

Es una variable a la que se suma constantes y se utiliza para realizar un conteo. Siempre debe ser inicializado (generalmente el valor inicial del contador es cero). También se puede utilizar para realizar una cuenta regresiva, restándole la constante al contador.

Sintaxis: Contador = Contador + Constante

Auxiliar

Es una variable que se utiliza para almacenar el contenido de otra variable. El auxiliar debe ser del mismo tipo que la otra variable.

Se utiliza para:

– Conservar el valor que tiene una variable en determinado momento, debido a que un proceso modificara ese valor.
– Intercambiar el contenido de dos variables
Sintaxis: Auxiliar = Variable

Acumulador

Es una variable a la que se le suma otra variable y se utiliza para realizar una sucesión de sumas. Al igual que el contador, el acumulador siempre debe ser inicializado.

Sintaxis: Acumulador = Acumulador + Variable

Banderas

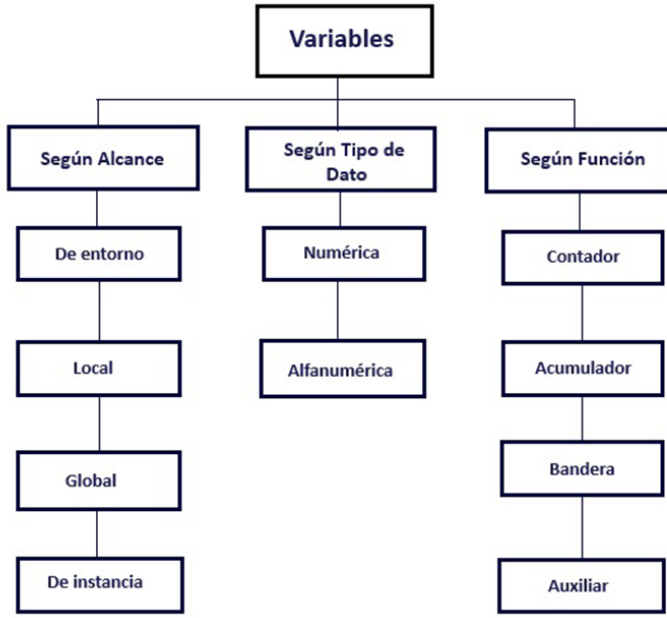
Es una variable booleana (es decir que solo admite dos estados posibles: 0 o 1), Si la bandera es igual a 0, significa que está inactiva o apagado, Si la bandera es igual a 1, significa que está activa, encendida o levantada.

Los usos más comunes de la bandera son:

– Saber si el programa ha pasado o no por un punto determinado.
– Terminar un ciclo cuando se produce determinada condición que modifica el estado de la bandera
– Ejecutar procesos diferentes según el estado de la bandera.
– Sintaxis: Bandera = Valor Booleano

Figura 19

Clasificación de las variables



Nota. Clasificación de las variables según su función.

Constantes

Como su nombre lo indica son datos que siempre van a tener el mismo valor, y por ninguna razón este valor puede ser modificado. También comparten el mismo concepto de reglas empleado en la declaración de variables. Las constantes se clasifican en: (Patiño, 2017)

Constantes literales

Son constantes que expresan el sentido directo de un valor de Cualquier tipo o de una cadena de caracteres. Por ejemplo: "la programación es fácil" 4895.6 "Carlos".

Constantes Declaradas

Son constantes que deben ser declaras antes de ser usadas. En la declaración debe usarse un nombre representativo que no se repita, seguidamente del valor que se desea asignar. Cabe resaltar que para poder declararse una constante declarada debe usarse una palabra reservada Constante. El formato de declaración es el siguiente:

Constante Nombre: valor o cadena de caracteres

Por ejemplo:
Constante Pi = 3.1416
Constante Nombre = "Carlos Andrés"
Constante Salario = 150000.

1.7.6 Operadores

"Los operadores son los componentes con los cuáles, las variables pueden realizar operaciones o cálculos" (Ospina, 2006, pág. 91).

Con el siguiente ejemplo se demostrará la importancia de los operadores.

Observe: 12 18
Simplemente esa línea representa dos números 12 y 18.
Ahora explique la siguiente línea: 12 + 18.
El operador + (mas) indica que se debe realizar una suma entre 12 y 18.

Explicación: para poderse interpretar esa expresión matemática, se requería de un operador que indicará que operación debía efectuarse. De igual

forma, las computadoras necesitan conocer los componentes y el operador para efectuar la correspondiente expresión.

Operador de Asignación (=)

Como su nombre lo indica asigna un valor a una variable.
Por ejemplo: Número1 = 12
Cuando se expresa Numero1 = 12 se dice que la variable llamada Numero1 se le asigna el valor de 12 o toma el valor de 12.

Operador de Asociación (())

La acción que cumplen los paréntesis es indicar que tipo de operación se debe realizar primero.
Por ejemplo:
5 * 2 + 10 = 20.
5 * (2 + 10) = 60.

En el caso anterior, primero se realice la suma y luego el valor total de la suma se multiplica por 5. Cuando se conozcan todos los operadores se explicará las reglas de prioridad.

Operadores Matemáticos

En esta sección se conocerán los operadores matemáticos más empleados, su símbolo de operador, su funcionamiento y ejemplos explicativos para una mayor compresión y entendimiento.

Suma (+)

Este operador ejecuta la suma correspondiente entre dos términos o variables. Por ejemplo:

Con constantes

12 + 18 = 30.

Con variables

A = 8 la variable A se le asigna el valor de 8
B = 3 la variable B se le asigna el valor de 3
C = A + B la variable C se le asigna el valor de la suma de A + B, que es 11.

Resta (-)

Este operador ejecuta la resta correspondiente entre dos términos o variables.

Por ejemplo:

Con constantes
18 - 12 = 6.

Con variables
A = 8 la variable A se le asigna el valor de 8
B = 3 la variable B se le asigna el valor de 3
C = A - B la variable C se le asigna el valor de la resta de A - B, que es 5.

Multiplicación (*)

Este operador ejecuta la multiplicación correspondiente entre dos términos o variables.

Por ejemplo:

Con constantes
12 * 5 = 60.

Con variables
A = 8 la variable A se le asigna el valor de 8
B = 3 la variable B se le asigna el valor de 3
C = A * B la variable C se le asigna el valor de la multiplicación de A * B, que es 24.

División (/)

Este operador ejecuta la división correspondiente entre dos términos o variables.

Por ejemplo:
Con constantes
12 / 2 = 6.

Con variables
A = 27 la variable A se le asigna el valor de 27
B = 3 la variable B se le asigna el valor de 3
C = A / B la variable C se le asigna el valor de la división de A / B, que es 9.

División Entera (\)

Este operador ejecuta la división entera correspondiente entre dos términos o variables.

Por ejemplo:
Con constantes
20 \ 3 = 6.

Con variables
A = 27 la variable A se le asigna el valor de 27
B = 3 la variable B se le asigna el valor de 3
C = A \ B la variable C se le asigna el valor de la división de A \ B, que es 9.
Módulo (MOD)

Este operador ejecuta la división correspondiente entre dos términos o variables, obteniendo el residuo.

Por ejemplo:
Con constantes
12 MOD 2 = 0.

Con variables

A = 8 la variable A se le asigna el valor de 8

B = 2 la variable B se le asigna el valor de 2

C = A MOD B la variable C se le asigna el valor del Residuo entre A MOD B, que es 0.

Exponencial (^)

Este operador ejecuta el exponente correspondiente entre dos términos o variables.

Por ejemplo:

Con constantes

2 ^ 3 = 8.

Con variables

A = 8 la variable A se le asigna el valor de 8, la base.

B = 2 la variable B se le asigna el valor de 2, el exponente.

C = A ^ B la variable C se le asigna el valor del exponente de A ^ B, que es 64.

Operadores de Cadenas de Caracteres

Concatenar (&)

El operador & concatena o une dos o más cadenas de caracteres.

Por ejemplo:

Con constantes

"la programación " & "es fácil" = "la programación es fácil"

Con variables

A = "Carlos" la variable A se le asigna la cadena "Carlos"

B = "Andrés" la variable B se le asigna la cadena "Andrés"

C = A & B la variable C se le asigna el valor de concatenar A & B, que es "CarlosAndres".

Reglas de Prioridad

Las expresiones matemáticas que contienen dos o más operadores, necesitan una serie de reglas que determinen el orden al ejecutarse sus operaciones, las cuáles son las siguientes:

– Las operaciones que se encuentran encerradas entre paréntesis se ejecutan primero.

Por ejemplo:

5 * (2 + 10) 12 + (8 + 3) + (5 - 3) = 25.

– El orden de prioridad de los operadores aritmético, obedece a la siguiente jerarquía que se muestra a continuación:

Figura 20

Jerarquía de operadores aritméticos

Operador	Nombre	Jerarquía al efectuar una operación
^	Exponencial	1
*	Multiplicación	2
/	División.	3
Div	División entera	4
Mod.	Residuo	5
+	Suma	6
-	Resta	7

Nota. Operadores aritméticos.

1.7.7 Pseudocódigo

El Pseudocódigo representa un algoritmo, pero empleando palabras simbólicas en vez de gráficos. Para complementar esta explicación, se cita a (Patiño, 2017) el cual menciona que:

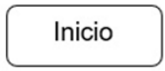
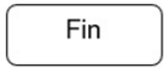
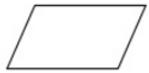
Es un lenguaje para la especificación de algoritmos previo a la codificación en términos de programación, muy fácil de entender y manipular.

Se puede considerar como un medio para representar estructuras de control que surgió como un lenguaje muy similar al de los humanos que puede ser convertido rápidamente a lenguaje de programación. (pág.40)

En la siguiente figura se muestra las palabras simbólicas, de forma similar y en el mismo orden, de los símbolos gráficos:

Figura 21

Palabra en Pseudocódigo

Símbolo en Diagrama de Flujo	Palabra en Pseudocódigo	Comentario
	Inicio	Representa el comienzo de un programa
	Fin	Representa el final de un programa.
	Leer	Captura de datos introducidos por teclado.
	Escribir	Los mensajes o resultados de procesos sobre la información son mostrados por pantalla.
	Proceso	Anteriormente se realizaba una asignación u operación en el símbolo del proceso, en Pseudocódigo la operación o asignación se efectuará de manera igual, se omite por completo el símbolo.
		En DF la flecha indica el flujo u orden con el cuál se realizan las operaciones, en Pseudocódigo se reemplazará por la justificación de cada una de las instrucciones.

Nota. Palabras simbólicas Pseudocódigo.

Su estructura cumple con las partes del algoritmo Inicio-Proceso-Fin, el proceso se establece como acciones que describen los pasos que se deben realizar, se ejecutan en el orden de cada línea de arriba hacia abajo.

Así:

Inicio
Acción 1
Acción 2
...
Acción n
Fin

Ejemplo:

Hacer un Pseudocódigo que lea dos números (N1 y N2) y determine cuál es el mayor.

1. **Inicio:**
2. Declaración N1 y N2
3. Leer N1 y N2
4. **Si** N1 = N2 **entonces**
5. Escribir: "N1 es igual a N2"
6. **Si** N1>N2 **entonces**
7. Escribir: "N1 es mayor que N2"
8. **Sino entonces**
9. Escribir: "N2 es mayor que N1"
10. **Fin_si**
11. **Fin_si**
12. **Fin**

Ejemplo:

Se solicita realizar un Pseudocódigo para calcular la suma, resta, multiplicación y división de cantidades dadas (C1 y C2).

1. **Inicio**
2. Declaración C1 y C2
3. **Leer** C1 y C2
4. **Si** C2=0 **entonces**

5. **Calcular**
6. Suma= C1+C2
7. Resta= C1-C2
8. Multiplicación= C1*C2
9. **Escribir**: Suma, Resta, Multiplicación, "No es posible la división por cero".
10. **Sino entonces**
11. Calcular
12. Suma= C1+C2
13. Resta= C1-C2
14. Multiplicación= C1*C2
15. División= C1/C2
16. Escribir: Suma, Resta, Multiplicación, División.
17. **Fin_si**
18. **Fin**

1.7.8 Diagrama de Flujo de Datos

El diagrama de Flujo es una representación gráfica de un algoritmo, que utiliza una serie de símbolos, para indicar un proceso o instrucción sobre el cuál se ejecuta una acción. (Ospina, 2006) menciona que:

Un diagrama de flujo se comporta de manera similar a la representación de un organigrama dentro de una compañía. En el cual se muestra el rango de mando u operación, se señala además, el flujo o conducto regular para la toma de decisiones dentro de la misma. Además, cuando se realiza la tarea del análisis y diseño de cualquier proceso, unas de las herramientas de modelado emplean características similares al DF, como son: El diagrama de flujo de datos, diagrama entidad – relación, diagramas de transición de estados, para el análisis estructurado moderno.(pág. 83)

Características del Diagrama de Flujos de Datos

Simplemente con observarlo se puede comprender, los diagramas de flujo generalmente ocupan una página, logrando captar visualmente toda la atención por parte del que lo observa.

La siguiente figura muestra los símbolos empleados por un diagrama de flujo.

Figura 22 Diagramas de Flujo

Diagramas de Flujo

Símbolo	Nombre	Descripción
	Terminal	Representa el comienzo: "inicio"; y el final: "fin" de un programa. Puede representar también una parada o interrupción programada 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 posición de la información almacenada en memoria, operación aritméticas, de transferencia, etc.
	Indicador de dirección o línea de flujo	Indica el sentido de ejecución de las operaciones.
	Decisión	Indica operaciones lógicas o de comparación entre datos – normalmente dos – y en función del resultado de la misma determina cuál de los distintos caminos alternativos del programa que se debe seguir, normalmente tiene 2 salidas: SI, NO.
	Conector	sirve para enlazar dos partes cualesquiera de un ordinograma a través de un conector en la salida y otro conector en la entrada. Se refiere a la conexión en la misma página del diagrama.
	Conector	Conexión entre dos puntos del ordinograma situado en páginas diferentes.
	Llamada a subrutina	Una subrutina es un módulo independiente del programa principal, que recibe una entrada procedente de dicho programa, realiza una tarea determinada y al terminar regresa al programa principal.

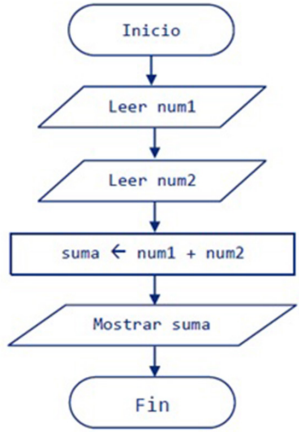
Nota. Símbolos utilizados en Diagramas de Flujo.

Ejemplo:

Realizar un algoritmo para operar la suma de dos números enteros.

Figura 23

Diagrama de Flujo



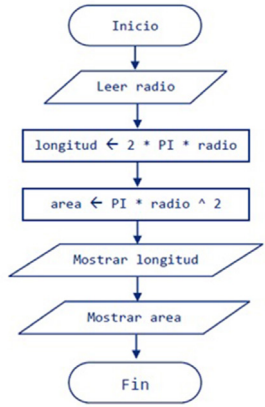
Nota. Diagrama de Flujo suma de dos números enteros.

Ejemplo:

Calcular y visualizar la longitud de la circunferencia y el área de un círculo de radio dado.

Figura 24

Diagrama de Flujo



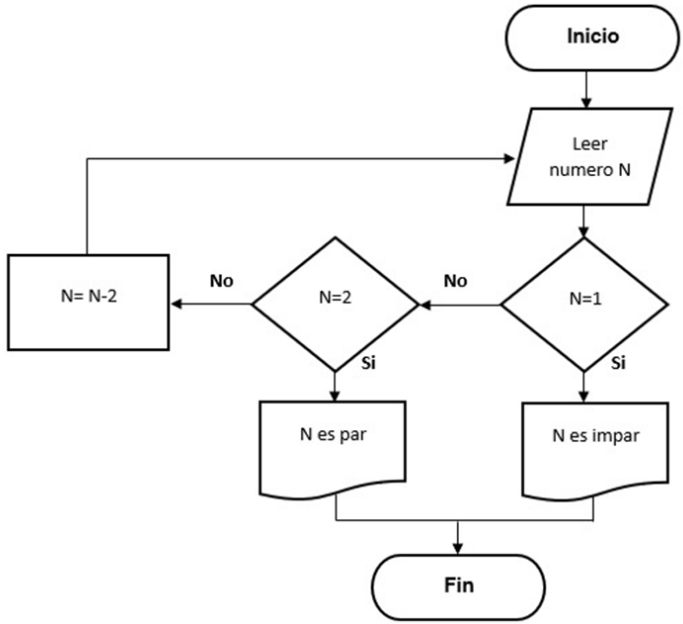
Nota. Diagrama de flujo, longitud de la circunferencia y el área de un círculo de radio

Ejemplo:

Dibujar un diagrama de flujo para determinar si un número N es par o impar.

Figura 25

Diagrama de flujo



Nota. Diagrama de flujo.

1.8 Estructuras de Control

(Figueroa Piscoya y otros, 2021) Mencionan que a finales de los años 1960 surgió una nueva forma de programar que no solamente daba lugar a programas fiables y eficientes, sino que además estaban escritos de manera que facilitaba su comprensión posterior. Estas Estructuras de Control son:

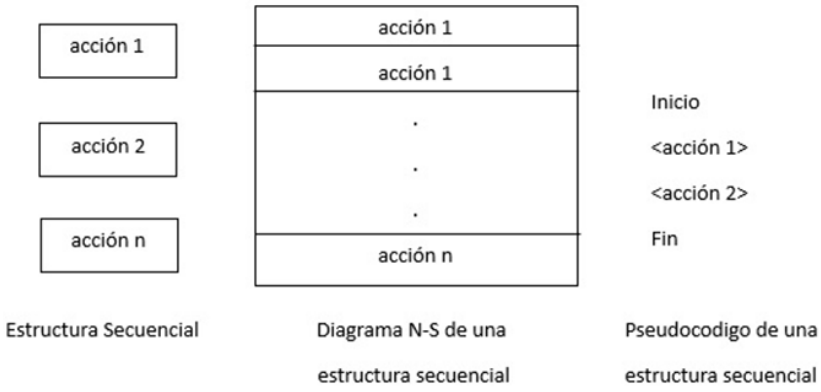
– Estructura de Control Secuencial.
– Estructura de Control Selectiva o Condicional.
– Estructura de Control Repetitiva o Iterativa.

1.8.1 Estructura Secuencial

Son aquellas instrucciones que se ejecutan una tras otra, en secuencia, es decir que una instrucción no se ejecuta hasta que finaliza la anterior.

Figura 26 Estructura secuencial

Estructura secuencial



Nota. Diagramas de la estructura secuencial

Ejemplo:

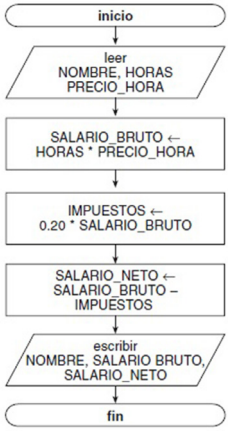
Se solicita calcular el salario neto de un trabajador en función del número de horas trabajadas, precio de la hora de trabajo y considerando unos descuentos fijos, el sueldo bruto en concepto de impuestos (20 por 100).

Pseudocódigo

1. Inicio
2. Leer (nombre, horas, precio_hora)
3. Salario_bruto ← horas * precio_hora
4. Impuestos ← 0.20 * salario_bruto
5. Salario_neto ← salario_bruto - impuestos
6. Escribir (nombre, salario_bruto, salario_neto)
7. Fin

Figura 27

Diagrama de flujo



Nota. Diagrama de flujo salario neto.

1.8.2 Estructuras Selectivas

Según (Ospina, 2006) menciona, la especificación formal de algoritmos tiene realmente utilidad cuando el algoritmo requiere una descripción más complicada que una lista sencilla de instrucciones. Este es el caso cuando existen un número de posibles alternativas resultantes de la evaluación de una determinada condición. Las estructuras selectivas se utilizan para tomar decisiones lógicas; de ahí que se suelan denominar también estructuras de decisión o alternativas. (pág.128)

En las estructuras selectivas menciona (Ospina, 2006) que se evalúa una condición y en función del resultado de esta se realiza una opción u otra. Las condiciones se especifican usando expresiones lógicas. La representación de una estructura selectiva se hace con palabras en pseudocódigo (if, then, else o bien en español si, entonces, si_no), con una figura geométrica en forma de rombo o bien con un triángulo en el interior de una caja rectangular. Las estructuras selectivas o alternativas pueden ser:

– simples,
– dobles,
– múltiples.

"La estructura simple es si (if) en formato C. La estructura selectiva doble es igual que la estructura simple si a la cual se le añade la cláusula si-no (else). La estructura selectiva múltiple es según_sea (switch en lenguaje C)" (Ospina, 2006).

Alternativa Simple (SI-ENTONCES / IF-THEN)

La estructura simple si-entonces ejecuta una determinada acción cuando se cumple una determinada condición. La selección si-entonces evalúa la condición y

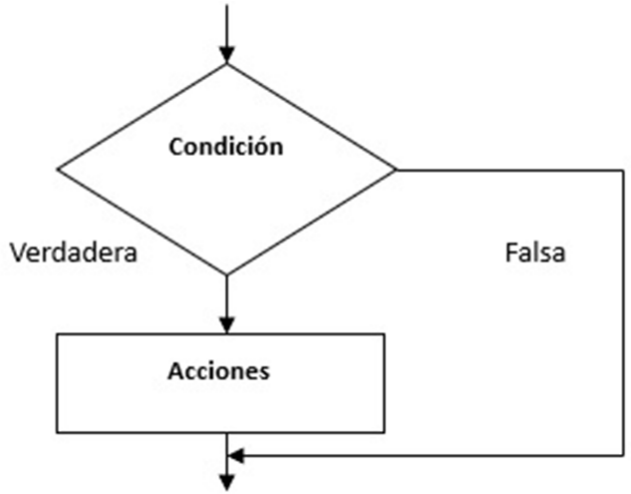
– Si la condición es verdadera, entonces ejecuta la acción S1 (o acciones caso de ser S1 una acción compuesta y constar de varias acciones),

– Si la condición es falsa, entonces no hace nada.

Las representaciones gráficas de la estructura condicional simple se muestran a continuación:

Figura 28

Estructuras alternativas simples



Nota. Estructuras alternativas simples: Tomado de Fundamentos de Programación, Algoritmos, estructura de datos y objetos [figura], Estructuras alternativas simples, Luis Aguilar, 2008.

Pseudocódigo en español

```
Si      <condición> entonces
        <acción SI>
fin_si
```

Pseudocódigo en ingles

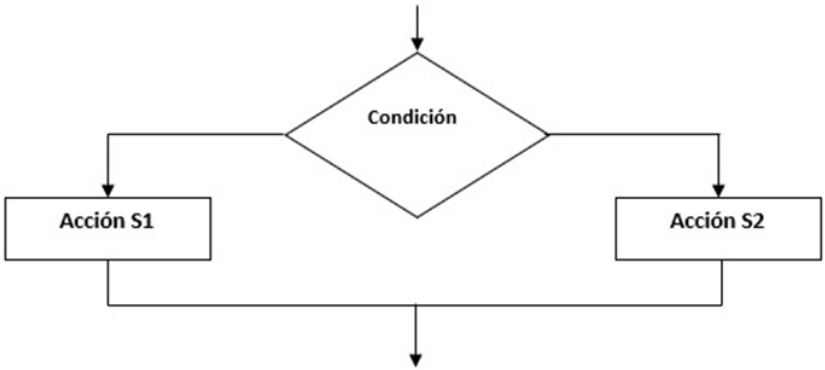
```
If      <condición> then
        <acción SI>
endif
```

Alternativa Doble (si-entonces-sino / if-then-else)

Esta estructura permite elegir entre dos opciones o alternativas posibles, en función del cumplimiento o no de una determinada condición. Si la condición es verdadera, se ejecuta la acción S1 y, si es falsa, se ejecuta la acción S2.

Figura 29

Estructura alternativa doble



Nota. Estructura alternativa doble. Tomado de Fundamentos de Programación, Algoritmos, estructura de datos y objetos [figura], Estructuras alternativas simples, Luis Aguilar, 2008.

Pseudocódigo en español

```
Si<condición> entonces
  <acción SI>
Si_no
  <acción S2>
fin_si
```

Pseudocódigo en ingles

```
If <condición> then
  <acción SI>
else
  <acción S2>
endif
```

Obsérvese que en el pseudocódigo las acciones que dependen de entonces y si_no están indentadas en relación con las palabras si y fin_si; este procedimiento aumenta la legibilidad de la estructura y es el medio más idóneo para representar algoritmos.

Ejemplo:

Resolución de una ecuación de primer grado.

Si la ecuación es ax + b = 0, a y b son los datos, y las posibles soluciones son:

– a <> 0	x = -b/a
– a = 0 b <> 0	entonces "solución imposible"
– a = 0 b = 0	entonces "solución indeterminada"

El algoritmo correspondiente será:

```
Algoritmo RESOL1
1 inicio
2 var
3 real : a, b, x
4 leer (a, b)
5     si a <> 0 entonces
```

```
6      x ← –b/a
7      escribir(x)
8      si_no
9         si b <> 0 entonces
10          escribir ('solución imposible')
11          si_no
12          escribir ('solución indeterminada')
13       fin_si
14    fin_si
15 fin
```

Ejemplo:

Se desea obtener la nómina semanal —salario neto— de los empleados de una empresa cuyo trabajo se paga por horas y del modo siguiente:

– Las horas inferiores o iguales a 35 horas (normales) se pagan a una tarifa determinada que se debe introducir por teclado al igual que el número de horas y el nombre del trabajador,

– Las horas superiores a 35 se pagarán como extras a un promedio de 1,5 horas normales,

– Los impuestos a deducir a los trabajadores varían en función de su sueldo mensual:

— sueldo <= 2.000, libre de impuestos,

— las siguientes 220 euros al 20 por 100,

— el resto, al 30 por 100.

Análisis

Las operaciones a realizar serán:

1. **Inicio**
2. Leer nombre, horas trabajadas, tarifa horaria.
3. Verificar si horas trabajadas <= 35, en cuyo caso salario_bruto = horas * tarifa; en caso contrario, salario_bruto = 35 * tarifa + (horas - 35) * tarifa.

4. Cálculo de impuestos
si salario_bruto <= 2.000, entonces impuestos = 0
si salario_bruto <= 2.220 entonces
impuestos = (salario_bruto - 2.000) * 0.20
si salario_bruto > 2.220 entonces
impuestos = (salario_bruto - 2.220) * 0.30 + (220 * 0.20)
5. Cálculo del salario_neto
salario_neto = salario_bruto - impuestos.
6. **Fin**

Representación del algoritmo en pseudocódigo

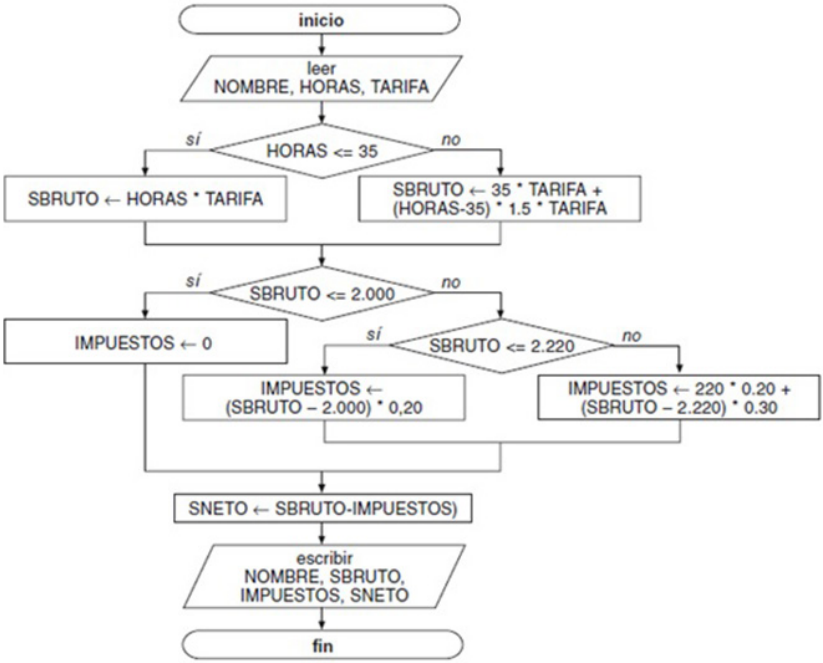
Algoritmo Nómina

```
inicio
var
cadena : nombre
real : horas, impuestos, sbruto, sneto
leer(nombre, horas, tarifa)
        si horas <= 35 entonces
          sbruto ← horas * tarifa
          si_no
          sbruto ← 35 * tarifa + (horas - 35) * 1.5 * tarifa
        fin_si
          si sbruto <= 2.000 entonces
            impuestos ← 0
          si_no
            si (sbruto > 2.000) y (sbruto <= 2.220) entonces
            impuestos ← (sbruto - 2.000) * 0.20
             si_no
               impuestos ← (220 * 0.20) + (sbruto - 2.220)
             fin_si
          fin_si
        sneto ← sbruto - impuestos
        escribir(nombre, sbruto, impuestos, neto)
fin
```

Representación del algoritmo en diagrama de flujo

Figura 30

Diagrama de flujo



Nota. Algoritmo resuelto en diagrama de flujo:

Alternativa Múltiple (según_sea, caso de/case)

(Figueroa Piscoya y otros, 2021) Indican que con frecuencia en la práctica es necesario que existan más de dos elecciones posibles (por ejemplo, en la resolución de la ecuación de segundo grado existen tres posibles alternativas o caminos a seguir, según que el discriminante sea negativo, nulo o positivo).

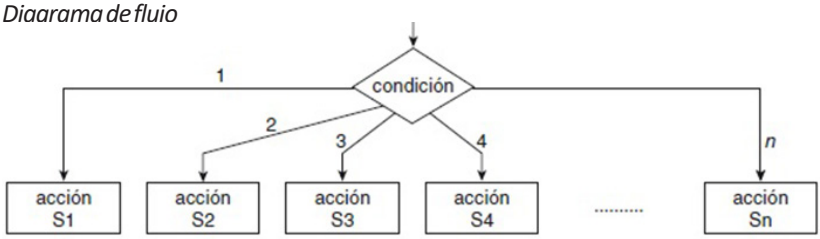
La estructura de decisión múltiple evaluará una expresión que podrá tomar n valores distintos, 1, 2, 3, 4,..., n. (pág. 135)

Según que elija uno de estos valores en la condición, se realizará una de las n acciones, o lo que es igual, el flujo del algoritmo seguirá un determinado camino entre los n posibles. Los diferentes modelos de pseudocódigo de la estructura de decisión múltiple se observan a continuación:

```
según_sea expresion (E) hacer
        e1: accion S11
        accion S12
        .
        .
        accion S1a
        e2: accion S21
        accion S22
        .
        .
        accion S2b
        .
        . (digibug.ugr.es)
        en: accion S31
        accion S32
        .
        .
        accion S3p
        si-no
        accion Sx
fin_según
```

Figura 31

Diagrama de fluio



Nota. Diagrama de flujo Alternativa múltiple.

En inglés la estructura de decisión múltiple se representa:

```
switch (expresión)
{
case valor1:
        sentencia1;
        sentencia2;
        sentencia3;
        .
        .
        break;
case valor2:
        sentencia1;
        sentencia2;
        sentencia3;
        .
        .
        break;
        .
        .
default:
        sentencia1;
        sentencia2;
        sentencia3;
        .
        .
}//fin de la sentencia compuesta
```

Ejemplo:

Se desea diseñar un algoritmo que escriba los nombres de los días de la semana en función del valor de una variable DIA introducida por teclado.

Los días de la semana son 7; por consiguiente, el rango de valores de DIA será 1 .. 7, y caso de que DIA tome un valor fuera de este rango se deberá producir un mensaje de error advirtiendo la situación anómala.

```
algoritmo DiasSemana
inicio
        var entero: DIA
        leer(DIA)
        según_sea DIA hacer
        1: escribir('LUNES')
        2: escribir('MARTES')
        3: escribir('MIERCOLES')
        4: escribir('JUEVES')
        5: escribir('VIERNES')
        6: escribir('SABADO')
        7:escribir('DOMINGO')
        sí-no
          escribir('ERROR')
        fin_según
fin
```

Ejemplo:

Algoritmo que nos indique si un número entero, leído de teclado, tiene 1, 2, 3 o más de 3 dígitos. Considerar los negativos.

Se puede observar que la estructura según_sea <expresion> hacer son varios si <expr.logica> entonces... anidados en la rama si_no. Si se cumple el primero ya no pasa por los demás.

```
algoritmo Digitos
inicio
        var entero : n
        leer(n)
        según_sea n hacer
                -9 .. 9:
                escribir('Tiene 1 digito')
                -99 .. 99:
                escribir('Tiene 2')
                -999 .. 999:
```

```
                escribir('Tiene tres')
                si_no
                escribir('Tiene mas de tres')
        fin_según
fin
```

Estructuras de Decisión Anidadas (en escalera)

"Las estructuras de selección si-entonces y si-entonces-si_no implican la selección de una de dos alternativas. Es posible también utilizar la instrucción si para diseñar estructuras de selección que contengan más de dos alternativas" (Ospina, 2006, pág. 131).

Por ejemplo, una estructura SI-ENTONCES puede contener otra estructura SI-ENTONCES, y esta estructura si-entonces puede contener otra, y así sucesivamente cualquier número de veces; a su vez, dentro de cada estructura pueden existir diferentes acciones.

Las estructuras SI interiores a otras estructuras si se denominan anidadas o encajadas:

```
si <condicion1> entonces
        si <condicion2> entonces
        .
        .
        .
          <acciones>
        fin_si
fin_si
```

Una estructura de selección de N alternativas o de decisión múltiple puede ser construida utilizando una estructura si con este formato:

```
si <condicion1> entonces
  <acciones>
si_no
```

```
            si <condicion2> entonces
            <acciones>
            si_no
                    si <condicion3> entonces
                     <acciones>
                    si_no
                    .
                    .
                    .
                    fin_si
             fin_si
fin_si
```

Una estructura selectiva múltiple constará de una serie de estructuras si, como las estructuras si pueden volverse bastante complejas para que el algoritmo sea claro, será preciso utilizar indentación (sangría o sangrado), de modo que exista una correspondencia entre las palabras reservadas si y fin_si, por un lado, y entonces y si_no, por otro.

La escritura de las estructuras puede variar de unos lenguajes de programación a otros, por ejemplo, una estructura si admite también los siguientes formatos:

```
si <expresion booleana1> entonces
  <acciones>
si_no
            si <expresion booleana2> entonces
              <acciones>
            si_no
              si <expresion booleana3> entonces
                <acciones>
              si_no
                <acciones>
              fin_si
            fin_si
fin_si
```

o bien

```
si <expresion booleana1> entonces
  <acciones>
si_no si <expresion booleana2> entonces
  <acciones>
  fin_si
.
.
.
fin_si
```

Ejemplo:

Diseñar un algoritmo que lea tres números A, B, C y visualice en pantalla el valor del más grande. Se supone que los tres valores son diferentes.

Los tres números son A, B y C; para calcular el más grande se realizarán comparaciones sucesivas por parejas.

```
Algoritmo Mayor
inicio
  var real: A, B, C, Mayor
  leer(A, B, C)
  si A > B entonces
        si A > C entonces
        Mayor ← A                    //A > B, A > C
        si_no
          Mayor ← C        //C >= A > B
        fin_si
  si_no
        si B > C entonces
          Mayor ← B        //B >= A, B > C
        si_no
          Mayor ← C        //C >= B >= A
        fin_si
```

```
  fin_si
        escribir('Mayor:', Mayor)
fin
```

Ejemplo:

El siguiente algoritmo lee tres números diferentes, A, B, C, e imprime los valores máximo y mínimo. El procedimiento consistirá en comparaciones sucesivas de parejas de números.

```
Algoritmo Ordenar
inicio
  var real: a,b,c
  escribir ('Deme 3 numeros')
  leer (a, b, c)
        si a > b entonces     // consideramos los dos primeros (a, b)
                              // y los ordenamos
          si b > c entonces // tomo el 3o (c) y lo comparo con el menor
                              // (a o b)
                escribir (a, b, c)
        si-no                 //si el 3o es mayor que el menor averiguo si
        si c > a entonces  // va delante o detrás del mayor
                escribir (c, a, b)
          si_no
                escribir (a, c, b)
          fin_si
        fin_si
si_no
        si a > c entonces
          escribir (b, a, c)
        si_no
          si c > b entonces
        escribir (c, b, a)
          si_no
                escribir (b, c, a)
          fin_si
```

```
            fin_si
        fin_si
fin
```

1.8.3 Estructuras Repetitivas

(Aguilar, 2006) menciona las computadoras están especialmente diseñadas para todas aquellas aplicaciones en las cuales una operación o conjunto de ellas deben repetirse muchas veces. Un tipo muy importante de estructura es el algoritmo necesario para repetir una o varias acciones un número determinado de veces. Un programa que lee una lista de números puede repetir la misma secuencia de mensajes al usuario e instrucciones de lectura hasta que todos los números de un fichero se lean.

Las estructuras que repiten una secuencia de instrucciones un número determinado de veces se denominan bucles y se denomina iteración al hecho de repetir la ejecución de una secuencia de acciones. Un ejemplo aclarará la cuestión.

Supongamos que se desea sumar una lista de números escritos desde teclado —por ejemplo, calificaciones de los alumnos de una clase—. El medio conocido hasta ahora es leer los números y añadir sus valores a una variable SUMA que contenga las sucesivas sumas parciales. La variable SUMA se hace igual a cero y a continuación se incrementa en el valor del número cada vez que uno de ellos se lea. El algoritmo que resuelve este problema es:

```
Algoritmo suma
inicio
        var entero : SUMA, NUMERO
        SUMA ← 0
        leer(numero)
        SUMA ← SUMA + numero
        leer(numero)
        SUMA ← SUMA + numero
        leer(numero)
fin
```

y así sucesivamente para cada número de la lista. En otras palabras, el algoritmo repite muchas veces las acciones.

```
leer(numero)
SUMA ← SUMA + numero
```

Tales opciones repetidas se denominan bucles o lazos. La acción (o acciones) que se repite en un bucle se denomina iteración. Las dos principales preguntas a realizarse en el diseño de un bucle son ¿qué contiene el bucle? y ¿cuántas veces se debe repetir?

Cuando se utiliza un bucle para sumar una lista de números, se necesita saber cuántos números se han de sumar.

Para ello se necesita conocer algún medio para detener el bucle. En el ejemplo anterior se usara la técnica de solicitar al usuario el número que desea, por ejemplo, N. Existen dos procedimientos para contar el número de iteraciones, usar una variable TOTAL que se inicializa a la cantidad de números que se desea y a continuación se decrementa en uno cada vez que el bucle se repite (este procedimiento añade una acción más al cuerpo del bucle: TOTAL ← TOTAL - 1), o bien inicializar la variable TOTAL en 0 o en 1 e ir incrementando en uno a cada iteración hasta llegar al número deseado.

```
Algoritmo suma_numero
inicio

var entero : N, TOTAL
        real : NUMERO, SUMA
        leer(N)
        TOTAL ← N
        SUMA ← 0
        mientras TOTAL > 0 hacer
          leer(NUMERO)
          SUMA ← SUMA + NUMERO
          TOTAL ← TOTAL - 1
    fin_mientras
```

escribir ('La suma de los', N, 'números es', SUMA)
fin

El bucle podrá también haberse terminado poniendo cualquiera de estas condiciones:

– hasta_que TOTAL sea cero
– desde 1 hasta N

Para detener la ejecución de los bucles se utiliza una condición de parada. El pseudocódigo de una estructura repetitiva tendrá siempre este formato:

inicio
//inicialización de variables
repetir
    acciones S1, S2, ...
    salir según condición
    acciones Sn, Sn+1, ...
fin_repetir

Aunque la condición de salida se indica en el formato anterior en el interior del bucle y existen lenguajes que así la contienen expresamente, lo normal es que la condición se indique al final o al principio del bucle, y así se consideran tres tipos de instrucciones o estructuras repetitivas o iterativas generales y una particular que denominaremos iterar, que contiene la salida en el interior del bucle.

iterar	(loop)
mientras	(while)
hacer-mientras	(do-while)
repetir	(repeat)
desde	(for)

El algoritmo de suma anterior podría expresarse en pseudocódigo estándar así:

```
Algoritmo SUMA_numeros
inicio
        var entero : N, TOTAL
        real: NUMERO, SUMA
        leer(N)
        TOTAL ← N
        SUMA ← 0
        repetir
          leer(NUMERO)
          SUMA ← SUMA + NUMERO
          TOTAL ← TOTAL - 1
        hasta_que TOTAL = 0
        escribir('La suma es', SUMA)
fin
```

Los tres casos generales de estructuras repetitivas dependen de la situación y modo de la condición. La condición se evalúa tan pronto se encuentra en el algoritmo y su resultado producirá los tres tipos de estructuras citadas.

1. La condición de salida del bucle se realiza al principio del bucle (estructura mientras).

```
Algoritmo SUMA1
inicio
  //Inicializar K, S a cero
  K ← 0
  S ← 0
  leer(n)
  mientras K < n hacer
K ← K + 1
S ← S + K
  fin_mientras
  escribir (S)
fin
```

Se ejecuta el bucle mientras se verifica una condición (K < n).

2. La condición de salida se origina al final del bucle; el bucle se ejecuta hasta que se verifica una cierta condición.

```
repetir
  K ← K + 1
  S ← S + K
hasta_que K > n
```

3. La condición de salida se realiza con un contador que cuenta el número de iteraciones.

```
desde i = vi hasta vf hacer
  S ← S + i
fin_desde
```

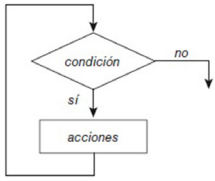
i es un contador que cuenta desde el valor inicial (vi) hasta el valor final (vf) con los incrementos que se consideren; si no se indica nada, el incremento es 1.

Estructura Mientras ("While")

(Ospina, 2006) Menciona que la estructura repetitiva mientras (en inglés while) es aquella en que el cuerpo del bucle se repite mientras se cumple una determinada condición. Cuando se ejecuta la instrucción mientras, la primera cosa que sucede es que se evalúa la condición (una expresión booleana). i se evalúa falsa, no se toma ninguna acción y el programa prosigue en la siguiente instrucción del bucle. Si la expresión booleana es verdadera, entonces se ejecuta el cuerpo del bucle, después de lo cual se evalúa de nuevo la expresión booleana. Este proceso se repite una y otra vez mientras la expresión booleana (condición) sea verdadera. (pág. 191)

Figura 32

Estructura mientras



Nota. Estructura mientras

Ejemplo:

Contar los números enteros positivos introducidos por teclado. Se consideran dos variables enteras número y contador (contará el número de enteros positivos). Se supone que se leen números positivos y se detiene el bucle cuando se lee un número negativo o cero.

```
Algoritmo Cuenta_enteros

inicio
  var entero : numero, contador
  contador ← 0
  leer(numero)
  mientras numero > 0 hacer
    leer(numero)
    contador ← contador + 1
  fin_mientras
  escribir("El numero de enteros positivos es", contador)
fin
```

Ejemplo:

Desarrolle un algoritmo que realice la sumatoria de los números enteros comprendidos entre el 1 y el 10.

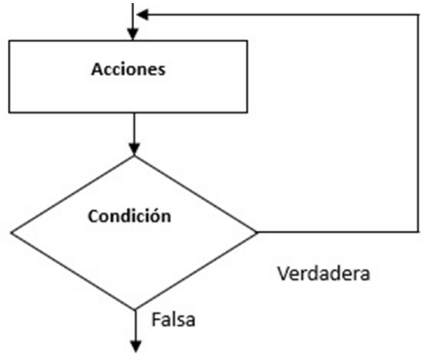
```
Proceso sumanumeros

        Definir a, c Como Entero
        a = 0
        c = 1
        Mientras c <= 10 Hacer
                a <- a + c
                c <- c + 1
                Escribir "La sumatoria es:", a
        Fin Mientras
FinProceso
```

Estructura Hacer-Mientras ("do-while")

El bucle mientras al igual que el bucle desde que se verá con posterioridad evalúa la expresión al comienzo del bucle de repetición; siempre se utilizan para crear bucle pre-test. Los bucles pre-test se denominan también bucles controlados por la entrada. En numerosas ocasiones se necesita que el conjunto de sentencias que componen el cuerpo del bucle se ejecute al menos una vez sea cual sea el valor de la expresión o condición de evaluación.

Figura 33

Hacer-mientras



Nota. Diagrama de flujo de una sentencia hacer-mientras.

Al igual que en el caso del bucle mientras la sentencia en el interior del bucle puede ser simple o compuesta.

Las sentencias en el interior del bucle se ejecutan al menos una vez antes de que la expresión o condición se evalúe. Entonces, si la expresión es verdadera (un valor distinto de cero, en C/C++) las sentencias del cuerpo del bucle se ejecutan una vez más. El proceso continúa hasta que la expresión evaluada toma el valor falso (valor cero en C/C++).

Ejemplo:

Obtener un algoritmo que lea un número (por ejemplo, 195) y obtenga el número inverso (por ejemplo, 591).

```
Algoritmo invertirnummero
inicio
  var entero: num, digitoSig
  num ← 195
  escribir ('Número: ← ', num)
  escribir ('Número en orden inverso: ')
  hacer
        digitoSig = num MOD 10
        escribir(digitoSig)
        num = num DIV 10
  mientras num > 0
fin
```

Análisis del ejemplo anterior:

Con cada iteración se obtiene el dígito más a la derecha, ya que es el resto de la división entera del valor del número (num) por 10. Así en la primera iteración digitoSig vale 8 ya que es el resto de la división entera de 198 entre 10 (cociente 19 y resto 8). Se visualiza el valor 8. A continuación se divide 198 entre 10 y se toma el cociente entero 19, que se asigna a la variable num.

En la siguiente iteración se divide 19 por 10 (cociente entero 1, resto 9) y se visualiza, por consiguiente, el valor del resto, digitoSig, es decir el dígito 9; a continuación, se divide 19 por 10 y se toma el cociente entero, es decir, 1.

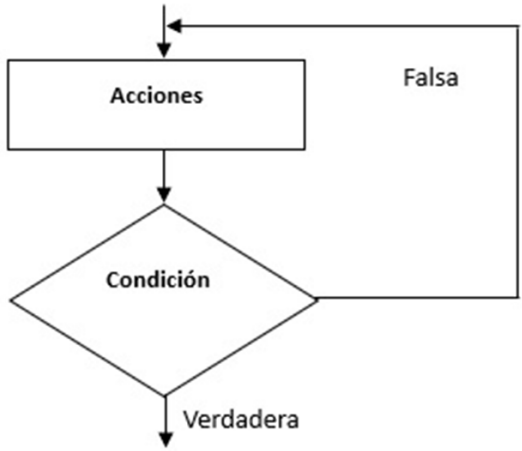
En la tercera y última iteración se divide 1 por 10 y se toma el resto (digitoSig) que es el dígito 1. Se visualiza el dígito 1 a continuación de 89 y como resultado final aparece 891. A continuación, se efectúa la división de nuevo por 10 y entonces el cociente entero es 0 que se asigna a num que al no ser ya mayor que cero hace que se termine el bucle y el algoritmo correspondiente.

Estructura Repetir ("repeat")

Existen muchas situaciones en las que se desea que un bucle se ejecute al menos una vez antes de comprobar la condición de repetición. En la estructura mientras si el valor de la expresión booleana es inicialmente falso, el cuerpo del bucle no se ejecutará; por ello, se necesitan otros tipos de estructuras repetitivas.

Figura 34

Estructura repetir



Nota. Diagrama de flujo estructura repetir.

La estructura repetir (repeat) se ejecuta hasta que se cumpla una condición determinada que se comprueba al final del bucle (Figura 30).

El bucle repetir-hasta_que se repite mientras el valor de la expresión booleana de la condición sea falsa, justo la opuesta de la sentencia mientras.

Ejemplo:

```
Algoritmo repetir
inicio
 var real : numero
 entero: contador
 contador ← 1
 repetir
        leer(numero)
        contador ← contador + 1
 hasta_que contador > 30
 escribir ("Numeros leidos 30")
fin
```

En el ejemplo anterior el bucle se repite hasta que el valor de la variable contador exceda a 30, lo que sucederá después de 30 ejecuciones del cuerpo del bucle.

Con una estructura repetir el cuerpo del bucle se ejecuta siempre al menos una vez. Cuando una instrucción repetir se ejecuta, lo primero que sucede es la ejecución del bucle y, a continuación, se evalúa la expresión booleana resultante de la condición. Si se evalúa como falsa, el cuerpo del bucle se repite y la expresión booleana se evalúa una vez. Después de cada iteración del cuerpo del bucle, la expresión booleana se evalúa; si es verdadera, el bucle termina y el programa sigue en la siguiente instrucción a hasta que.

Ejemplo:

```
Escribir los números 1 a 100.
algoritmo uno_cien
inicio
  var entero : num
  num ← 1
  repetir
    escribir(num)
    num ← num + 1
  hasta_que num > 100
fin
```

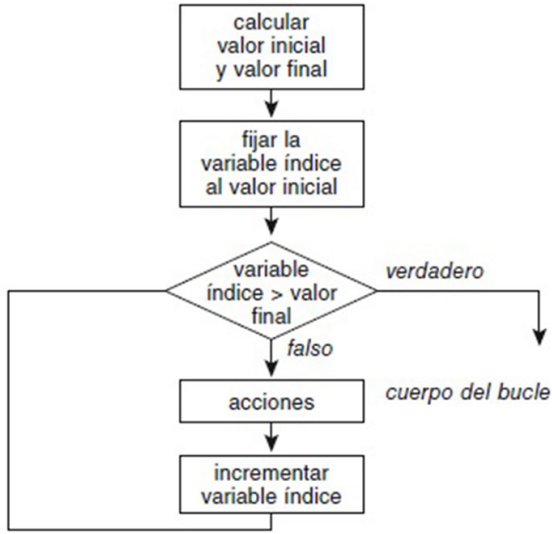
Estructura Desde/Para ("for")

Se conoce de antemano el número de veces que se desean ejecutar las acciones de un bucle. En estos casos, en el que el número de iteraciones es fijo, se debe usar la estructura desde o para (for, en inglés). (Figueroa Piscoya y otros, 2021)

La estructura desde ejecuta las acciones del cuerpo del bucle un número especificado de veces y de modo automático controla el número de iteraciones o pasos a través del cuerpo del bucle. Las herramientas de programación de la estructura desde o para se muestran en la siguiente figura.

Figura 35

Estructura desde (for)



Nota. Diagrama de flujo estructura desde (for).

La estructura desde comienza con un valor inicial de la variable índice y las acciones especificadas se ejecutan, a menos que el valor inicial sea mayor que el valor final. La variable índice se incrementa en uno y si este nuevo valor no excede al final, se ejecutan de nuevo las acciones. Por consiguiente, las acciones específicas en el bucle se ejecutan para cada valor de la variable índice desde el valor inicial hasta el valor final con el incremento de uno en uno.
Ejemplo:

Obtener mediante un pseudocódigo la tabla de multiplicar de un número utilizando la estructura PARA.

```
Proceso tabla
        Definir i, resul, n Como Entero
        Escribir "Tabla de multiplicar de un numero";
        Escribir "ingrese un numero";
        Leer n;
        Para i <- 1 Hasta 10 Con Paso 1 Hacer
```

```
            resul <- n * i;
            Escribir "La multiplicación de ", n " X " i, " es ", resul;
        Fin Para
    FinProceso
```

Según (Aguilar, 2006) C++ proporciona tres sentencias para el control de bucles: while, for y do-while. While el uso más frecuente es cuando la repetición no está controlada por contador; el test de condición precede a cada repetición del bucle; el cuerpo del bucle puede no ser ejecutado. Se debe utilizar cuando se desea saltar el bucle si la condición es falsa.

For bucle de conteo cuando el número de repeticiones se conoce por anticipado y puede ser controlado por un contador; también es adecuado para bucles que implican control no contable del bucle con simples etapas de inicialización y de actualización; el test de la condición precede a la ejecución del cuerpo del bucle.

Do-while es adecuada cuando se debe asegurar que al menos se ejecuta el bucle una vez.

12. Actividad de aprendizaje

Tema: Partes del computador
¿Cuáles son las partes de un disco duro?

Tema: Modelos de ciclo de vida del software
Enuncie las fases del modelo de ciclo de vida en cascada:

Tema: Pseudocódigo
Realice un algoritmo para calcular y visualizar la longitud de la circunferencia y el área de un círculo de radio dado.

Tema: Estructura secuencial
Resuelva el siguiente ejercicio, realizar el pseudocódigo y el diagrama de flujo correspondiente para el cálculo de la suma y producto de dos números.
La suma S de dos números es S = A+B y el producto P es P = A*B.

Tema: Estructura alternativa simple

Se solicita realizar un algoritmo para leer una temperatura en grados centígrados e indique si es superior a 100 mostrar el mensaje "por encima del punto de ebullición".

Tema: Estructura alternativa doble

Realizar un algoritmo, se solicita la resolución de una ecuación de primer grado.

Si la ecuación es ax + b = 0, a y b son los datos, y las posibles soluciones son:

– a <> 0 x = -b/a

– a = 0 b <> 0 entonces "solución imposible"

– a = 0 b = 0 entonces "solución indeterminada"

Tema: Estructura alternativa múltiple

Realizar un algoritmo el cual, leída una fecha, decir el día de la semana, suponiendo que el día 1 de dicho mes es lunes.

Tema: Estructura de decisión anidadas

Se solicita diseñar un algoritmo, que lea tres números enteros y visualice en pantalla el valor del más grande. Se supone que los tres valores son diferentes.

Tema: Estructura mientras ("while")

Se solicita realizar un algoritmo el cual mostrando el número mayor de una serie de números ingresados por teclado. Salir del bucle sólo cuando se ingresa un número negativo.

Tema: Estructura hacer-mientras ("do-while")

Mediante un algoritmo mostrar los números pares del 2 al 50, usando estructura repetitiva "Hacer-Mientras".

Tema: Estructura Para (for)

Mediante un algoritmo obtener el promedio de edades de 3 personas, cuyas edades son ingresadas por teclado, utilizando la estructura PARA.

13. Autoevaluación

Autoevaluación 1

1. ¿Cuáles son dispositivos de almacenamiento?

a) Las unidades CD-ROM, CD-RW, DVD.
b) Entre estos se encuentran: el mouse, teclado, lápiz óptico.
c) Entre estos se encuentran: las impresoras, las pantallas o monitores, parlantes.
d)El monitor, impresora, teclado.

2. ¿Quién realiza las labores de enviar y recibir datos a través de los buses de datos?

a)Fuente de poder
b) Disco Duro
c) Memoria ROM
d) El microprocesador

3. ¿Qué es la torre del computador?

a) La gran mayoría de los usuarios de ordenadores la llaman CPU.
b) Es el dispositivo primario de salida donde se visualizan las imágenes generadas por el ordenador.
c) Es simplemente el soporte para una serie de componentes albergados dentro de ella.
d) Procesa o manipula la información almacenada en memoria; puede recuperar información desde memoria volátil.

4. ¿Qué es el Software?

a) El software es la parte tangible del computador.
b) Es un componente electrónico que contiene el procesador, la memoria RAM, los buses de datos y otros elementos.
c) Se emplea para labores de procesos que se efectúan con el procesador y otros periféricos, guardado datos mientras se estén usando.
d) Es la parte lógica de un sistema de cómputo, que no es tangible, pero se hace visible cuando se saca algún provecho a un ordenador.

5. Un ciclo de vida para un proyecto se compone de:

a) Entregables permiten evaluar la marcha del proyecto mediante comprobaciones de su adecuación o no a los requisitos funcionales.
b) Fases sucesivas compuestas por tareas que se pueden planificar.
c) Describir los estados por los que pasa el producto.

d) La comprobar que el sistema es lo que el cliente quiere.

Autoevaluación 2

1. ¿Cuál es la diferencia entre la estructura Do while y While?

a. Una sentencia do-while es similar a una sentencia while, cuando el número de repeticiones se conoce por anticipado y puede ser controlado por un contador
b. Una sentencia do-while es similar a una sentencia while, excepto que el cuerpo del bucle se ejecuta siempre al menos una vez.
c. Una sentencia while es similar a una sentencia do-while ya que se debe asegurar que al menos se ejecuta el bucle una vez.
d. Una sentencia do-while es similar a una sentencia while cuando se desea saltar el bucle si la condición es falsa.

2. ¿Cuál es la estructura de decisión múltiple?

a. Si entonces..... fin_si
b. según_sea expresion (E) hacer....fin_según
c. mientras...hacer....fin_mientras
d. hacer...mientras

3. ¿Cuál es la diferencia entre las estructuras mientras y repetir?

a. La estructura mientras termina cuando la condición es verdadera, mientras que repetir termina cuando la condición es falsa.
b. La estructura mientras termina cuando la condición es falsa, mientras que repetir termina cuando la condición es verdadera.
c. La estructura mientras termina cuando la condición es falsa, mientras que repetir termina cuando la condición es verdadera.
d. Una sentencia repetir se ejecuta cuando el número de repeticiones se conoce por anticipado y puede ser controlado por un contador.

4. ¿Qué se utiliza para obtener el módulo de una división?

a. El operador de concatenar
b. El operador de asignación
c. El operador Mod
d. El operador Div

5. ¿Cómo es el proceso del ciclo para ... finpara?

a. Esta estructura repetitiva está diseñada generalmente para cuando no se conoce cuantas iteraciones o veces se debe repetir el ciclo.
b. Se diseño para cuando no se conoce cuantas iteraciones o veces se debe

repetir el ciclo, se emplea también cuando se conocen la cantidad de iteraciones como en el ciclo anterior.
c. Esta estructura repetitiva está diseñada para cuando se conoce el valor inicial (en donde empieza el ciclo) y el final (en donde termina).
d. Este proceso permite reproducir una serie instrucciones un número infinito de veces.

14. Evaluación final

La evaluación final se realizará presencialmente. El alumno deberá resolver 2 ejercicios planteados aplicando toda su destreza para realizar el pseudocódigo y el diagrama de flujo correspondiente a cada uno de los problemas planteados. La evaluación final será sobre 10 puntos.

15. Solucionario de las autoevaluaciones

Autoevaluación 1

1. ¿Cuáles son dispositivos de almacenamiento?

a) Las unidades CD-ROM, CD-RW, DVD.
b) Entre estos se encuentran: el mouse, teclado, lápiz óptico.
c) Entre estos se encuentran: las impresoras, las pantallas o monitores, parlantes.
d) El monitor, impresora, teclado.

2. ¿Quién realiza las labores de enviar y recibir datos a través de los buses de datos?

a) Fuente de poder
b) Disco Duro
c) Memoria ROM
d) El microprocesador

3. ¿Qué es la torre del computador?

a) La gran mayoría de los usuarios de ordenadores la llaman CPU.
b) Es el dispositivo primario de salida donde se visualizan las imágenes generadas por el ordenador.
c) Es simplemente el soporte para una serie de componentes albergados dentro de ella.
d) Procesa o manipula la información almacenada en memoria; puede recuperar información desde memoria volátil.

4. ¿Qué es el Software?

a) El software es la parte tangible del computador.
b) Es un componente electrónico que contiene el procesador, la memoria RAM, los buses de datos y otros elementos.
c) Se emplea para labores de procesos que se efectúan con el procesador y otros periféricos, guardado datos mientras se estén usando.
d) Es la parte lógica de un sistema de cómputo, que no es tangible pero se hace visible cuando se saca algún provecho a un ordenador.

5. Un ciclo de vida para un proyecto se compone de:

a) Entregables permiten evaluar la marcha del proyecto mediante comprobaciones de su adecuación o no a los requisitos funcionales
b) Fases sucesivas compuestas por tareas que se pueden planificar.
c) Describir los estados por los que pasa el producto.
d) La comprobar que el sistema es lo que el cliente quiere.

Autoevaluación 2

1. ¿Cuál es la diferencia entre la estructura Do while y While?

a. Una sentencia do-while es similar a una sentencia while, cuando el número de repeticiones se conoce por anticipado y puede ser controlado por un contador
b. Una sentencia do-while es similar a una sentencia while, excepto que el cuerpo del bucle se ejecuta siempre al menos una vez.
c. Una sentencia while es similar a una sentencia do-while ya que se debe asegurar que al menos se ejecuta el bucle una vez.
d. Una sentencia do-while es similar a una sentencia while cuando se desea saltar el bucle si la condición es falsa.

2. ¿Cuál es la estructura de decisión múltiple?

a. Si entonces….. fin_si
b. según_sea expresion (E) hacer….fin_según
c. mientras…hacer….fin_mientras
d. hacer…mientras

3. ¿Cuál es la diferencia entre las estructuras mientras y repetir?

a. La estructura mientras termina cuando la condición es verdadera, mientras que repetir termina cuando la condición es falsa.
b. La estructura mientras termina cuando la condición es falsa, mientras que repetir termina cuando la condición es verdadera.

c. La estructura mientras termina cuando la condición es falsa, mientras que repetir termina cuando la condición es verdadera.
d. Una sentencia repetir se lejecuta cuando el número de repeticiones se conoce por anticipado y puede ser controlado por un contador

4. ¿Qué se utiliza para obtener el módulo de una división?

a. El operador de concatenar
b. El operador de asignación
c. El operador Mod
d. El operador Div

5. ¿Cómo es el proceso del ciclo para ... finpara?

a. Esta estructura repetitiva está diseñada generalmente para cuando no se conoce cuantas iteraciones o veces se debe repetir el ciclo.
b. Se diseño para cuando no se conoce cuantas iteraciones o veces se debe repetir el ciclo, se emplea también cuando se conocen la cantidad de iteraciones como en el ciclo anterior.
c. Esta estructura repetitiva está diseñada para cuando se conoce el valor inicial (en donde empieza el ciclo) y el final (en donde termina).
d. Este proceso permite reproducir una serie instrucciones un número infinito de veces.

16. Glosario

A

Algoritmo: Descripción precisa de los pasos que nos llevan a la solución de un problema planteado.
ATX: Advanced Technology Extended.
B
Bios: (Basic Input/Output System) Es un estándar de facto que define la interfaz de firmware para computadoras.
Bucle: Serie de instrucciones que se repiten indefinidamente mientras no se cumpla una condición previamente establecida.
C
Ciclo de vida: conjunto de fases por las que pasa el sistema que se está desarrollando desde que nace la idea inicial hasta que el software es retirado o remplazado.
G
GNU: Es un sistema operativo de software libre.

I

Ingeniería del software: disciplina de ingeniería preocupada por todos los aspectos de la producción de software desde las primeras etapas de especificación del sistema hasta el mantenimiento del sistema después de que éste se haya puesto en uso. Es la aplicación de un enfoque sistemático, disciplinado y cuantificable para el desarrollo, operación y mantenimiento del software, que es la aplicación de la ingeniería del software.

Iteración: significa repetir varias veces un proceso con la intención de alcanzar una meta deseada, objetivo o resultado.

M

MAC: Es la línea de ordenadores personales diseñada, desarrollada y comercializada por Apple Inc.

P

Pseudocódigo: Es una forma de escribir los pasos que va a realizar un programa de la forma más cercana al lenguaje de programación que vamos a utilizar posteriormente.

S

Software: son los programas y la documentación asociada tal como requisitos, modelos de diseño y manuales de usuario.

17.Referencias bibliográficas

– Aguilar, L. J. (2006). Programación en C++. Algoritmos, estructuras de datos y objetos (2.ª edición ed.). España: McGRAW-HILL/INTERAMERICANA DE ESPAÑA. Disponible en: https://doi.org/84-481-4645-X

– Díaz, J., Harari, I., & Amadeo, A. P. (2013). Guía de recomendaciones para diseño de software centrado en el usuario (1a edición ed.). Buenos Aires, Argentina: Universidad Nacional de La Plata.

– Eckel, B. (2011). Pensar en C++. La Nación Disponible en: https://doi.org/http://arco.esi.uclm.es/~david.villa/pensarC++.html

– Figueroa Piscoya , E. N., Maldonado Ramirez, I., & Santa Cruz Acosta, R. C. (2021). Fundamentos de programación (Primera Edición ed.). Bagua: Biblioteca Nacional del Perú N° 2021.

– INTECO. (2009). CURSO DE INTRODUCCIÓN A LA INGENIERÍA DEL SOFTWARE. Instituto Nacional de Tecnologías de la comunicación. Disponible en: https://doi.org/http://creativecommons.org/licenses/by-nc-sa/2.5/es/

– Mathieu, M. J. (2014). Introducción a la programación. México: Grupo editorial Patria.
– Ospina, C. A. (2006). Fundamentos de programación, guía de autoenseñanza. Colombia: RA-MA editorial. Disponible en: https://doi.org/75-089-060
– Patiño, C. A. (2017). Lógica de programación. Fondo editorial Areandino. Disponible en: https://doi.org/978-958-5455-95-5
– Vacas, F. S. (2009). Complejidad y Tecnologías de la Información. España: Fundación Rogelio Segovia. Disponible en: https://doi.org/978-84-7402-365-7

18. Anexos o recursos

– https://libros.metabiblioteca.org/items/74874a48-41f9-45f2-a1a5-4cea-7b5b59c4
– http://repositoriokoha.uner.edu.ar/fing/pdf/5224.pdf
– https://digitk.areandina.edu.co/bitstream/handle/areandina/1281/Algoritmos%20y%20programacio%cc%81n.pdf?sequence=1&isAllowed=y
– https://d1wqtxts1xzle7.cloudfront.net/55545337/Libro_Algoritmos-libre.pdf?1516044956=&response-content-disposition=inline%3B+filename%3D-Libro_Algoritmos.pdf&Expires=1690850279&Signature=TE1Y

Instituto Superior Tecnológico
VICENTE LEÓN

Guía general de estudio de la asignatura

Diciembre 2023

ISBN: 978-9942-7211-6-7