INGRESO

2023

Cuadernillo del Curso de Nivelación

Instituto de Educación Superior 9-008
Manuel Belgrano

DESARROLLO DE SOFTWARE

Dirección de Educación Superior
Dirección General de Escuelas
Gobierno de Mendoza

Contenido

Estimado Aspirante a 1er. Año: ... 3
Información Ingreso 2023 ... 4
Datos de interés para el ingresante ... 4
PERFIL PROFESIONAL: ... 4
Régimen de Asistencia ... 4
Régimen de Regularidad, Promoción, Evaluación y Acreditación ... 4
Evaluación ... 4
Los siguientes espacios curriculares tienen acreditaci6n directa (promocionales): ... 5
Materias Tecnicatura Superior en Desarrollo de Software ... 6
DOCUMENTACIÓN REQUERIDA PARA INGRESAR AL INSTITUTO ... 7
a) Alumnos con secundario completo finalizado en la Argentina ... 7
b) Alumnos que adeuden materias en la escuela secundaria ... 7
c) Alumnos extranjeros con secundario completo ... 7
d) Alumnos mayores de 25 años sin secundario completo ... 7
DESARROLLO DEL CURSO DE INGRESO (Cronograma a confirmar) ... 9
INICIA EN MARZO 2023 FECHAS A CONFIRMAR ... 9
Curso nivelatorio ... 9
INTRODUCCION A LAS TECNICAS DE PROGRAMACIÓN ... 10
Características del software: ... 10
Estructura Interna de una Computadora ... 10
Redes de Computadoras ... 13
Programación y construcción de Software ... 14
Los sistemas y su enfoque ... 15
¿Por qué hablamos de sistemas? ... 15
¿Qué es un Sistema? ... 15
Características de los sistemas ... 16
Intercambio entre sistemas ... 17
Sistemas tecnológicos ... 18
¿Cómo se construye el Software? ... 18
Diseño de Algoritmos ... 20
Algoritmos ... 21
Concepto ... 21
Características de los algoritmos ... 21
Herramientas para la representación gráfica de los algoritmos ... 22
Diagramas de Flujo ... 23
Pseudocódigo ... 23
Lenguajes de Programación ... 24
Concepto ... 24

Tipos de Lenguajes de Programación 24
El lenguaje máquina 24
El lenguaje de bajo nivel 24
Lenguajes de alto nivel 25
¿Qué es un Programa? 25
Pasos para la construcción de un programa 25
DEFINICIÓN DEL PROBLEMA 25
ANÁLISIS DEL PROBLEMA 25
DISEÑO DEL ALGORITMO 26
CODIFICACIÓN 26
PRUEBA Y DEPURACIÓN 26
Elementos de un Programa 26
Variables y Constantes 26
Variables 27
Constantes 28
Operadores 28
Operador de asignación 28
Los operadores unitarios 29
Operadores Condicionales 30
Operadores relacionales 30
Operadores lógicos 31
Rutinas 32
Ejemplos de rutinas: 32
Desarrollo de programas 32
Estructuras de programación 32
Estructura secuencial 33
Estructura alternativa 33
La estructura alternativa simple 34
Alternativa Doble (si-entonces-sino/if-then-else) 34
Alternativa de Decisión múltiple (según_sea, caso de/case) 35
Estructura repetitiva o iterativa 37
Estructura mientras (while, en inglés) 37
Estructura hacer-mientras (do while, en inglés) 38
Estructura para (for, en inglés) 38
PRACTICA 40

Estimado Aspirante a 1er. Año:

En la actualidad, cada vez más ciudades del país cuentan con importantes centros en tecnologías informáticas y comunicacionales. El desarrollo de esta actividad incluye la presencia de empresas nacionales e internacionales, lo que se refleja en el excelente momento que vive la industria de software.

La demanda de profesionales en tecnologías de la información en el país debe ser vista como una oportunidad para las nuevas generaciones y una razón para impulsar el crecimiento académico en este campo.

Las instituciones educativas cobran singular importancia y es por ello por lo que, el **IES N° 9-008 "Manuel Belgrano"** brinda una propuesta académica de acuerdo con las necesidades del sector y a las exigencias del mercado laboral. Los profesionales de la informática tienen una formación básica sólida, lo que les permitirá seguir capacitándose continuamente al ritmo de la evolución tecnológica.

Información Ingreso 2023

Datos de interés para el ingresante

PERFIL PROFESIONAL:

Competencias Generales El título de Técnico Superior en Desarrollo de Software, habilita al profesional.

El Técnico Superior en Desarrollo de Software estará capacitado para producir artefactos de software, lo que comprende su diseño detallado, construcción -reutilizando elementos existentes o programándolos enteramente- y verificación unitaria, así como su depuración, optimización y mantenimiento; desarrollando las actividades descriptas en el perfil profesional y cumpliendo con los criterios de realización establecidos para las mismas en el marco de un equipo de trabajo organizado por proyecto.

Para ello, será capaz de desempeñarse en las siguientes áreas de competencia:

1. Analizar sistemas de información para determinar especificaciones funcionales estableciendo las características de una solución que las satisfaga, considerando el contexto en que se integra.
2. Modelizar y desarrollar artefactos de software que implementen el modelo diseñado, aplicando patrones o reutilizando código existente.
3. Verificar el correcto funcionamiento y los resultados del desarrollo de los artefactos de software propios o ajenos, teniendo en cuenta las necesidades de cobertura de prueba, diseño y ejecución de estos, y registrando datos junto con los miembros del equipo para realizar acciones correctivas sobre el código.
4. Documentar todas las actividades y productos obtenidos atendiendo al aseguramiento de la calidad en proyectos, productos y procesos, considerando normas y estándares preexistentes.
5. Autogestionar sus actividades, las de su sector dentro de la organización, o emprendimiento propio.
6. Desarrollar en forma sinérgica y creativa el liderazgo, la comunicación activa y el trabajo en equipo, en los múltiples ámbitos de desempeño que plantea el mundo laboral.

Régimen de Asistencia

Cfr. RESOLUCION 258-DGE- RAM G.2.

Régimen de Regularidad, Promoción, Evaluación y Acreditación

Cfr. RESOLUCION 258-DGE- RAM

La regularidad del cursado de cada unidad curricular durara dos años académicos o 7 (siete) turnos ordinarios de examen. Para cursar el segundo año de la carrera, el alumno deberá tener aprobada la mitad más uno de los espacios del primer año. Para cursar el tercer año, el alumno deberá tener aprobado todos los espacios de primer año y la mitad más uno del segundo año. Para acreditar la Práctica Profesional de tercer año, debe haber acreditado todos los espacios curriculares de primero, segundo año y tercer año.

Evaluación

Las unidades curriculares serán evaluadas conforme al formato que posean y a las normas institucionales.
Cfr. RESOLUCION 258-DGE

Los siguientes espacios curriculares tienen acreditaci6n directa:

Se regirán por el sistema de acreditación directa, según lo dispone el Artículo 46 inciso "a" del apartado referido a las trayectorias estudiantiles correspondientes al Reglamento Académico Marco (Res N° 258 –DGE–12), los siguientes espacios curriculares:

1° Año

2. Arquitectura de Computadoras.
4. Álgebra.
5. Inglés Técnico I.
6. Comprensión y Producción de textos.
7. Lógica Computacional.
8. Problemática Sociocultural y del Trabajo.
9. Sistemas Administrativos Aplicados.

2° Año

11.Comunicaciones y Redes
13.Matemática discreta.
14.Análisis Matemático.
15.Inglés Técnico II.
17.Bases de Datos I

3° Año

24.Inglés Técnico III.
25.Bases de Datos II.
26.Probabilidad y Estadística.
27.Legislación Informática.
28.Ética Profesional.
29.Gestión de Proyectos de Software.

Opcionalmente podrán agregarse otros espacios respetando la reglamentación vigente y con la aprobación de las autoridades pertinentes.

Materias Tecnicatura Superior en Desarrollo de Software – PLAN DE ESTUDIOS

PRIMER AÑO	SEGUNDO AÑO	TERCER AÑO
PROGRAMACIÓN I	COMUNICACIONES Y REDES	PROGRAMACIÓN III
ARQUITECTURA DE COMPUTADORAS	PROGRAMACIÓN II	ARQUITECTURA Y DISEÑO DE INTERFACES
REQUERIMIENTOS DE SOFTWARE	MATEMÁTICA DISCRETA	AUDITORÍA Y CALIDAD DE SISTEMAS
ÁLGEBRA	ANÁLISIS MATEMÁTICO	SEGURIDAD INFORMÁTICA
INGLÉS TÉCNICO I	INGLÉS TÉCNICO II	INGLÉS TÉCNICO III
COMPRENSIÓN Y PRODUCCIÓN DE TEXTOS	MODELADO DE SOFTWARE	BASES DE DATOS II
LÓGICA COMPUTACIONAL	BASES DE DATOS I	PROBABILIDAD Y ESTADÍSTICA
PROBLEMÁTICA SOCIOCULTURAL Y DEL TRABAJO	SISTEMAS OPERATIVOS	LEGISLACIÓN INFORMÁTICA
SISTEMAS ADMINISTRATIVOS APLICADOS	PRÁCTICA PROFESIONALIZANTE II	ÉTICA PROFESIONAL
PRÁCTICA PROFESIONALIZANTE I		GESTIÓN DE PROYECTOS DE SOFTWARE
		PRÁCTICA PROFESIONALIZANTE III

DOCUMENTACIÓN REQUERIDA PARA INGRESAR AL INSTITUTO

Una vez realizado el Curso de Nivelación, los alumnos que ingresen al Instituto en el Ciclo Lectivo 2022, tienen que concurrir a la Sede en donde van a cursar con la siguiente documentación necesaria:

a) Alumnos con secundario completo finalizado en la Argentina

2 Fotografías de frente 4 x 4.

2 Fotocopias del DNI certificadas. (Certificadas: debe tener Sello de la Escuela, Sello de "Copia Fiel", firma de la autoridad y Fecha de realización de trámite).

Copia legalizada de la Partida de Nacimiento

Certificado de salud APTO psicofísico (debe decir "**Apto para estudios de nivel superior**", expedido por organismo público-estatal)

Copia certificada del certificado de estudios de enseñanza secundaria. *(Título o Analítico) o la constancia que el mismo se encuentra* en trámite Certificadas: debe tener Sello de la Escuela, Sello de "Copia Fiel", firma de la autoridad y Fecha de realización de trámite).

Constancia de CUIL

Certificado de Vacuna Antitetánica y Hepatitis B (Ley Nº 8972)

Por último, presentar: Adhesión de Colaboración de la Cooperadora, al inicio del ciclo Lectivo 2022

El certificado de aptitud psicofísica se aceptará hasta el mes de julio de 2022. Caso contrario no podrá rendir ninguna materia.

No se inscribirá a los alumnos que no presenten la documentación detallada precedentemente.

Nota: Quien desee solicitar equivalencias en los Espacios Curriculares deben pedir información en la Sede al momento de inscribirse.

b) Alumnos que adeuden materias en la escuela secundaria

Es imprescindible presentar al momento de la inscripción definitiva toda la documentación solicitada precedentemente en el punto **a)**, excepto el Certificado Analítico, por ello deben presentar Certificado de la Escuela Secundaria al mes de marzo 2017, en el que consten las materias adeudadas y la modalidad del secundario, firmado por la Secretaría y Dirección de la Escuela que lo emite.

c) Alumnos extranjeros con secundario completo

Deberán pasar por Secretaría del Instituto para entregar la documentación necesaria y completar así su inscripción.

d) Alumnos mayores de 25 años sin secundario completo

Al momento de la Preinscripción: Tienen que presentar la siguiente documentación:

2 Fotografías de frente 4 x 4.
2 Fotocopias del DNI certificadas. (Certificadas: debe tener Sello de la Escuela, Sello de "Copia Fiel", firma de la autoridad y Fecha de realización de trámite).
Copia legalizada de la Partida de Nacimiento
Certificado de salud APTO psicofísico (debe decir "**Apto para estudios de nivel superior**", expedido por organismo público-estatal)
Copia certificada del certificado de estudios de enseñanza secundaria. *(Título o Analítico) o la constancia que el mismo se encuentra* en trámite Certificadas: debe tener Sello de la Escuela, Sello de "Copia Fiel", firma de la autoridad y Fecha de realización de trámite).

Constancia de CUIL

Certificado de Vacuna Antitetánica y Hepatitis B (Ley Nº 8972)

En caso de no presentar esta documentación no podrán acceder al Examen de Competencias Básicas.

DESARROLLO DEL CURSO DE INGRESO (Cronograma a confirmar)

La fecha de inicio se confirmará a partir del día __ de febrero de 2023, en cada una de las sedes y está publicada en la página web del instituto. www.iesmb.edu.ar

INICIA EN MARZO 2022 FECHAS A CONFIRMAR

Inicio de curso nivelatorio: 13/02/2023

Finalización de curso nivelatorio: 03/03/2023

Evaluación 13/03/2023 y 15/03/2023

Publicación de resultados: 17/03/2023 (*Los resultados de los exámenes establecen el orden mérito.*)

Período de inscripción a 1er año: 21/03/2023 al 22/03/2023

Inicio regular de clases 27/03/2023

Horarios de Cursado

Curso nivelatorio

Días horarios y sedes para confirmar por e-mail y página web

Cada aspirante debe confirmar su correo electrónico, se le enviará un usuario y password para el ingreso al aula virtual donde se encontrará el resto del material para el curso nivelatorio.

www.iesmb.edu.ar Seleccionar **Aula Virtual**

INTRODUCCION A LAS TECNICAS DE PROGRAMACIÓN

El Software y sus características

El software en sus comienzos era la parte insignificante del hardware, lo que venía como añadidura, casi como regalo. Al poco tiempo adquirió una entidad propia.

En la actualizad, el software es la tecnología individual más importante en el mundo. Nadie en la década de 1950 podría haber predicho que el software se convertiría en una tecnología indispensable en los negocios, la ciencia, la ingeniería; tampoco podría preverse que una compañía de software podría volverse más grande e influyente que la mayoría de las compañías de la era industrial; que una red construida con software, llamada Internet cubriría y cambiaría todo, desde la investigación bibliográfica hasta las compras de los consumidores y los hábitos de las personas. Nadie podría haber imaginado que estaría relacionado con sistemas de todo tipo: transporte, medicina, militares, industriales, entretenimiento, automatización de hogares.

Una definición formal de software según la IEEE (Instituto de Ingeniería Eléctrica y Electrónica) es la siguiente:

Es el conjunto de los programas de cómputo, procedimientos, reglas, documentación y datos asociados, que forman parte de las operaciones de un sistema de computación.

El software puede definirse como "el alma y cerebro de la computadora, la corporización de las funciones de un sistema, el conocimiento capturado acerca de un área de aplicación, la colección de

los programas, y los datos necesarios para convertir a una computadora en una máquina de propósito especial diseñada para una aplicación particular, y toda la información producida durante el desarrollo de un producto de software". El software viabiliza el producto más importante de nuestro tiempo: la información.

Características del software:

1. El software es intangible, es decir, que se trata de un concepto abstracto.
2. Tiene alto contenido intelectual.
3. Su proceso de desarrollo es humano intensivo, es decir que la materia prima principal radica en la mente de quienes lo crean.
4. El software no exhibe una separación real entre investigación y producción.
5. El software puede ser potencialmente modificado, infinitamente.
6. El software no se desgasta
7. La mayoría del software, en su mayoría, aún se construye a medida.
8. El software no se desarrolla en forma masiva, debido a que es único.

Estructura Interna de una Computadora

Una computadora moderna consta de uno o más procesadores, una memoria principal, discos, impresoras, un teclado, un ratón, una pantalla o monitor, interfaces de red y otros dispositivos de entrada/salida. En general es un sistema complejo. Si todos los programadores de aplicaciones tuvieran que comprender el funcionamiento de todas estas partes, no escribirían código alguno. Es más: el trabajo de administrar todos estos componentes y utilizarlos de manera óptima es una tarea muy desafiante. Por esta razón, las computadoras están equipadas con una capa de software llamada sistema operativo, cuyo trabajo es proporcionar a los programas de usuario un modelo de computadora mejor, más simple y pulcro, así como encargarse de la administración de todos los recursos antes mencionados.

La mayoría de las computadoras, grandes o pequeñas, están organizadas como se muestra en la siguiente figura. Constan fundamentalmente de tres componentes principales: Unidad Central de Proceso (UCP) o procesador, la memoria principal o central.
Si a los componentes anteriores se les añaden los dispositivos para comunicación con la computadora, aparece la estructura típica de un sistema de computadora: dispositivos de entrada, dispositivos de salida, memoria externa y el procesador/memoria central.

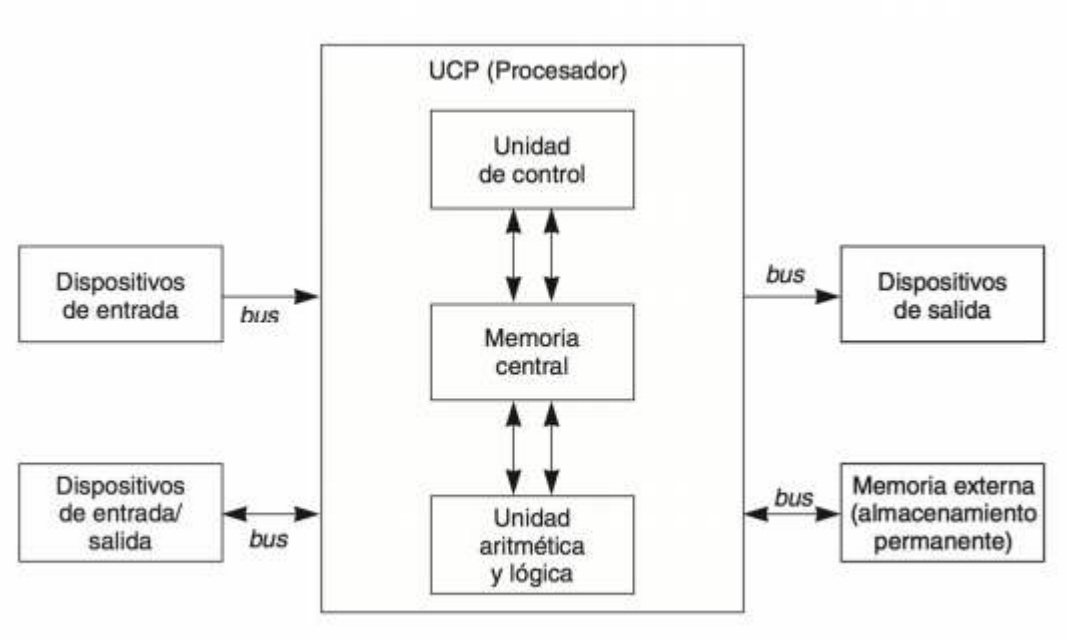

Figura 1: Estructura interna de un sistema de computadora

Los dispositivos de Entrada/Salida (E/S) (en inglés, Input/Output I/O) o periféricos permiten la comunicación entre la computadora y el usuario.

Los dispositivos de entrada, como su nombre indica, sirven para introducir datos en la computadora para su proceso. Los datos se leen de los dispositivos de entrada y se almacenan en la memoria central o interna. Los dispositivos de entrada convierten la información de entrada en señales eléctricas que se almacenan en la memoria central. Dispositivos de entrada típicos son teclados, lápices ópticos, joysticks, lectores de códigos de barras, escáneres, micrófonos, lectores de tarjetas digitales, lectores RFID (tarjetas de identificación por radio frecuencia), etc.

Hoy, tal vez el dispositivo de entrada más popular es el ratón (mouse) que mueve un puntero gráfico (electrónico) sobre la pantalla, o más recientemente las pantallas táctiles, que facilitan la interacción usuario-máquina.

Los dispositivos de salida permiten representar los resultados (salida) del proceso. El dispositivo de salida típico es la pantalla o monitor. Otros dispositivos de salida son: impresoras (imprimen resultados en papel), trazadores gráficos (plotters), reconocedores (sintetizadores) de voz, parlantes, entre otros.

Los dispositivos de entrada/salida y dispositivos de almacenamiento masivo o auxiliar (memoria externa) son: unidad de discos (disquetes, CD-ROM, DVD, discos duros, etc.), videocámaras, memorias

flash, USB, etc.
La memoria central o simplemente memoria (interna o principal) se utiliza para almacenar información (RAM, del inglés Random Access Memory). En general, la información almacenada en la memoria
puede ser de dos tipos: instrucciones de un programa y datos con los que operan las instrucciones. Por ejemplo, para que un programa se pueda ejecutar (correr, funcionar..., en ingle
´s, run), debe ser situado
en la memoria central, en una operación denominada carga (load) del programa. Después, cuando se ejecuta el programa, cualquier dato a procesar se debe llevar a la memoria mediante las instrucciones
del programa. En la memoria central, hay también datos diversos y espacio de almacenamiento temporal que necesita el programa cuando se ejecuta a fin de poder funcionar.
La memoria central de una computadora es una zona de almacenamiento organizada en centenares o millares de unidades de almacenamiento individual o celdas. La memoria central consta de un conjunto de celdas de memoria (estas celdas o posiciones de memoria se denominan también palabras, aunque no guardan analogía con las palabras del lenguaje). El número de celdas de memoria de la memoria central, depende del tipo y modelo de computadora; hoy día el número suele ser millones (512, 1.024, etc.). Cada celda de memoria consta de un cierto número de bits (normalmente 8, un byte).

La unidad elemental de memoria se llama byte. Un byte tiene la capacidad de almacenar un carácter de información, y está formado por un conjunto de unidades más pequeñas de almacenamiento denominadas bits, que son dígitos binarios que pueden asumir como valor un 0 o un 1.

Siempre que se almacena una nueva información en una posición, se destruye (desaparece) cualquier información que en ella hubiera y no se puede recuperar. La dirección es permanente y única, el contenido puede cambiar mientras se ejecuta un programa.

La memoria central de una computadora puede tener desde unos centenares de miles de bytes hasta millones de bytes. Como el byte es una unidad elemental de almacenamiento, se utilizan múltiplos de 10 potencia de 2 para definir el tamaño de la memoria central: Kilobyte (KB o Kb) igual a 1.024 bytes (2^{10})— prácticamente se consideran 1.000—; Megabyte (MB o Mb) igual a 1.024 × 1.024 bytes = = 1.048.576 (2^{20}) — prácticamente se consideran 1.000.000; Gigabyte (GB o Gb) igual a 1.024 MB (2 1.073.741.824 (2^{30}) = prácticamente se consideran 1.000 millones de MB.

Byte	**Byte (B)**	*equivale a*	8 bits
Kilobyte	**Kbyte (KB)**	*equivale a*	1.024 bytes
Megabyte	**Mbyte (MB)**	*equivale a*	1.024 Kbytes
Gigabyte	**Gbyte (GB)**	*equivale a*	1.024 Mbytes
Terabyte	**Tbyte (TB)**	*equivale a*	1.024 Gbytes
1 **Tb** = 1.024 **Gb** = 1.024 × 1.024 **Mb** = 1.048.576 **Kb** = 1.073.741.824 **B**			

Tabla 1: Unidades de medida para el almacenamiento en la memoria

La Unidad Central de Proceso UCP, o procesador, dirige y controla el proceso de información realizado por la computadora. La UCP procesa o manipula la información almacenada en memoria; puede recuperar información desde memoria (esta información son datos o instrucciones de programas) y también puede almacenar los resultados de estos procesos en memoria para su uso posterior.

Más adelante veremos en profundidad cómo los programas hacen uso de la memoria para almacenar o leer datos a fin de utilizarlos para el desarrollo de sus funciones.

Redes de Computadoras

La fusión de las computadoras y las comunicaciones ha tenido una profunda influencia en cuanto a la manera en que se organizan los sistemas de cómputo. El concepto una vez dominante del "centro de cómputo" como un salón con una gran computadora a la que los usuarios llevaban su trabajo para procesarlo es ahora totalmente obsoleto, aunque los centros de datos que contienen miles de servidores de Internet se están volviendo comunes).

El viejo modelo de una sola computadora para atender todas las necesidades computacionales de la organización se ha reemplazado por uno en el que un gran número de computadoras separadas pero interconectadas realizan el trabajo. A estos sistemas se les conoce como redes de computadoras. Se dice que dos computadoras están interconectadas si pueden intercambiar información. La conexión no necesita ser a través de un cable de cobre; también se puede utilizar fibra óptica, microondas, infrarrojos y satélites de comunicaciones. Las redes pueden ser de muchos tamaños, figuras y formas, como veremos más adelante.

Por lo general se conectan entre sí para formar redes más grandes, en donde Internet es el ejemplo más popular de una red de redes. Imaginemos el sistema de información de una empresa como si estuviera constituido por una o más bases de datos con información de la empresa y cierto número de empleados que necesitan acceder a esos datos en forma remota.

En este modelo, los datos se almacenan en poderosas computadoras denominadas servidores. A menudo estos servidores están alojados en una ubicación central y un administrador de sistemas se encarga de su mantenimiento. Por el contrario, los empleados tienen en sus escritorios máquinas más simples conocidas como clientes, con las cuales acceden a los datos remotos, por ejemplo, para incluirlos en las hojas de cálculo que desarrollan (algunas veces nos referiremos al usuario humano del equipo cliente como el "cliente", aunque el contexto debe dejar en claro si nos referimos a la computadora o a su usuario). Las máquinas cliente y servidor se conectan mediante una red, como se muestra en la figura 2.

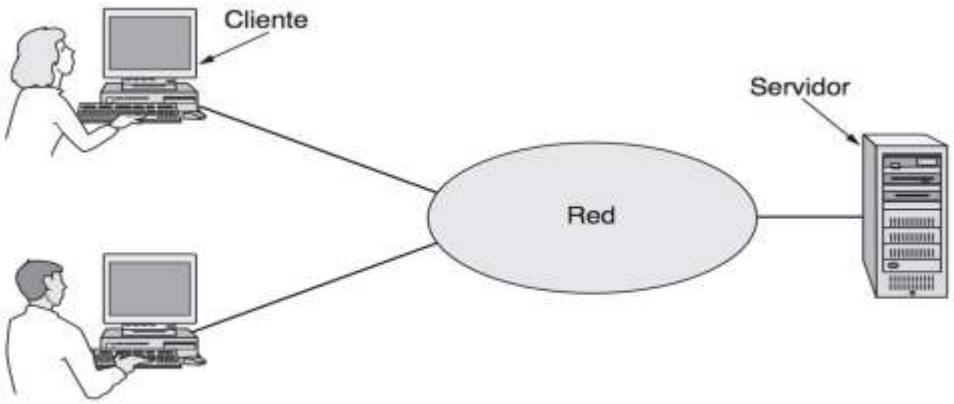

Figura 2: Esquema de una red de computadoras

A esta disposición se le conoce como modelo cliente-servidor. Es un modelo ampliamente utilizado y forma la base de muchas redes. La realización más popular es la de una aplicación web, en la cual el servidor genera páginas web basadas en su base de datos en respuesta a las solicitudes de los clientes que pueden actualizarla. El modelo cliente-servidor es aplicable cuando el cliente y el servidor se encuentran en el mismo edificio (y pertenecen a la misma empresa), pero también cuando están muy alejados. Por ejemplo, cuando una persona accede desde su hogar a una página en Internet se emplea el mismo modelo, en donde el servidor web remoto representa al servidor y la computadora personal del usuario representa al cliente. En la mayoría de las situaciones un servidor puede atender un gran número (cientos o miles) de clientes simultáneamente. La evolución de las comunicaciones y los dispositivos personales, así como las necesidades emergentes de compartir información en tiempo real han posibilitado la expansión de Internet a todos los rincones del mundo. De esta forma cualquier persona puede acceder a sus archivos, compartir datos, comunicarse o buscar información en cualquier momento a través de su computadora, notebook, teléfonos celulares entre otros, tal como se muestra en la siguiente figura.

Figura 3: Integración de tecnología a través de Internet

Programación y construcción de Software

El único tipo de instrucciones que una computadora puede entender es el lenguaje de máquina, o lenguaje de bajo nivel, donde diferentes tipos de procesadores pueden tener distintos lenguajes de máquina. El lenguaje máquina está compuesto de ceros y unos lo que hace que programar en lenguaje máquina sea un proceso tedioso y sujeto a errores. Una alternativa a utilizar lenguaje de máquina es el lenguaje Assembly, Assembler o ensamblador, que es también un lenguaje de bajo nivel que utiliza mnemonics (o abreviaturas) y es más fácil de entender que ceros y unos. Sin embargo, el único lenguaje que una computadora puede entender directamente es el lenguaje máquina, ¿entonces cómo es posible que entienda lenguajes como Assembler? La respuesta es que el lenguaje Assembler es convertido o traducido a lenguaje de máquina mediante un programa llamado ensamblador. Es importante destacar que hay una correspondencia directa entre el lenguaje Assembler y el lenguaje máquina, lo que significa que para cada instrucción de lenguaje assembler existe una instrucción de lenguaje máquina, lo que hace la traducción un proceso directo.

Sin embargo, más allá de que el lenguaje Assembler es más sencillo que el lenguaje máquina, distintos tipos de procesadores tienen diferentes conjuntos de instrucciones lo que se traduce en distintos dialectos de Assembler de una computadora a otra. La solución para hacer la tarea de programación más sencilla y posibilitar a los programas funcionar en distintos tipos de computadoras es utilizar lenguajes de alto nivel, que son más similares al lenguaje natural que utilizamos para comunicarnos diariamente y por motivos históricos estos lenguajes utilizan palabras del idioma inglés. Uno de los primeros lenguajes de programación de alto nivel fue FORTRAN (del inglés FORmula TRANslation, o traducción de fórmulas) que fue desarrollado en los comienzos de los años 50 para ayudar a resolver problemas matemáticos. Desde ese entonces, una gran cantidad de lenguajes de programación de alto nivel han sido creados para abordar distintos tipos de problemas y solucionar las necesidades de distintos tipos de usuarios. Algunos de ellos incluyen a COBOL, también desarrollado en los 50 para abordar aplicaciones empresariales y de negocios; BASIC en los 60 para programadores recién iniciados, Pascal en los 70 para problemas científicos, C, C++ y muchos otros. En este material nos centraremos en el lenguaje Java, también de alto nivel y de propósito general: es decir que puede usarse para una gran variedad de problemas y rubros.

Los sistemas y su enfoque

¿Por qué hablamos de sistemas?

En la primera mitad del siglo XX, surgió la necesidad de diseñar métodos de investigación y estudio de los fenómenos complejos a causa de una acumulación de problemáticas en las que los métodos de investigación de las ciencias particulares se mostraban insuficientes. Por un lado, los nuevos sistemas de producción que incluían varias automatizaciones, el manejo de grandes cantidades de energía (termoeléctrica, nuclear…) que requería de especialistas de variadas ramas, el desarrollo y organización de transporte terrestre, marítimo y aéreo y otros fenómenos.

Por otro, los grandes desarrollos científicos en la física (relatividad, estructura atómica, mecánica cuántica), biología (genética, evolución, estudio de poblaciones), química (teoría del enlace de Lewis, tabla periódica, estructura cristalina), matemática (álgebra de Boole, desarrollo del cálculo, problemas de Hilbert).

Estas grandes revoluciones en el hacer y el pensar hicieron necesario el desarrollo de un enfoque complejo para la investigación de fenómenos complejos. Así nació el enfoque sistémico, sustentado por la Teoría General de los Sistemas (TGS) formulada por Ludwig von Bertalanffy a mediados del siglo XX. Bertalanffy se dedicó especialmente a los organismos como sistemas biológicos, pero luego generalizó su estudio a todo tipo de sistemas. De tal manera que hoy se utiliza el término sistema en todas las áreas del conocimiento humano.

¿Qué es un Sistema?

Llamamos sistema a todo conjunto de elementos relacionados entre sí –puede ser por una finalidad en común-, que tienen un cierto orden u organización y que cumplen una función.

Los sistemas tienen composición (los elementos que lo forman), una estructura interna dada por el conjunto de relaciones entre sus componentes. Y también tienen un entorno o ambiente que es el conjunto de cosas que no pertenecen al sistema pero que actúan sobre él o sobre las que él actúa intercambiando materia, energía e información (MEI).

Figura 4: Elementos de un sistema

Los sistemas están inmersos en un entorno o ambiente, que es el conjunto de elementos que está fuera del sistema, es decir que no pertenecen al sistema pero que actúan sobre él o sobre las que el sistema actúa intercambiando materia, energía e información (MEI).

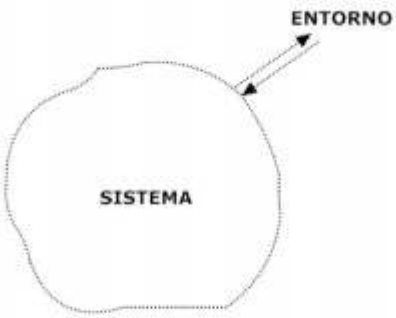

Figura 5: Relación del sistema con su entorno

Características de los sistemas

La característica principal de los sistemas es que poseen una propiedad emergente que no poseen sus componentes particulares. Por ejemplo, la vida es la propiedad emergente de un sistema compuesto por huesos, órganos, etc.; marchar es la propiedad emergente del sistema automóvil compuesto por chapas, motor, luces, etc. Este hecho se suele enunciar con la siguiente afirmación

EL TODO ES MÁS QUE LA SUMA DE LAS PARTES

Otras características de los sistemas son:

a. **Límite o frontera:** Son demarcaciones que permiten establecer qué elementos pertenecen o no al sistema. Los límites pueden ser:
    - Concretos: los que tienen existencia material (ríos que separan países, paredes que definen aulas, etc.)
    - Simbólicos: los que no tienen existencia material y vienen dados por acuerdos, reglas o normas (un alumno pertenece a un curso porque lo establece la escuela, más allá de que pueda hallarse en otro salón o fuera de la misma)

b. **Depósitos o almacenamientos:** son lugares donde se almacena materia, energía o información (MEI). Los depósitos pueden ser:
   - Permanentes: aquellos en que están diseñados para que su contenido no se altere (CDROM, libros, carteles fijos, etc.)
   - Transitorios: aquellos diseñados para que su contenido sufra modificaciones (pizarrón, cartuchera, tanques de agua, etc.)

c. **Canales**: Son lugares o conductos por donde circula materia, energía o información (MEI). Los canales pueden comunicar dos sistemas entre sí o partes de un mismo sistema (las calles pueden ser canales de materia, los cables pueden ser canales de energía si llevan corriente o de información si son telefónicos o de redes, etc.)

d. **Subsistemas:** los sistemas complejos (muchos componentes y relaciones entre ellos) pueden dividirse para su estudio en subsistemas. Esto permite diferentes niveles de estudio de los mismos. Se llama nivel cero al análisis del sistema en su totalidad y su intercambio con el entorno. A partir de allí se define el nivel 1, nivel 2, etc.

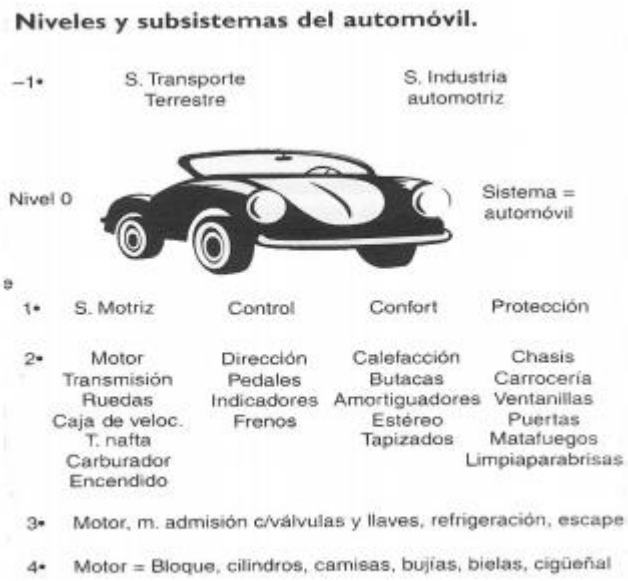

Figura 6: Sistemas y Subsistemas

Intercambio entre sistemas

Los sistemas intercambian entre sí materia, energía e información (MEI). Para que se dé este intercambio es necesario que MEI atraviese los límites del sistema hacia (o desde) el entorno. Si el sistema intercambia con el medio se dice que es abierto, de lo contrario se considera cerrado.

En sistemas cerrados cualquier estado final está determinado por sus condiciones iniciales, ya que no hay modo de que el entorno actúe sobre él. Si un sistema cerrado tampoco intercambia energía se dice que es aislado. En realidad, el único sistema que se considera absolutamente aislado es el universo. De igual modo, muchos sistemas mecánicos e informáticos pueden considerarse razonablemente cerrados.

Los sistemas abiertos, en cambio, pueden, crecer, cambiar, adaptarse al ambiente, incluso algunos reproducirse. Si un sistema posee la organización necesaria para controlar su propio desarrollo, asegurando la continuidad de su composición y estructura (homeostasis) y la de los flujos y transformaciones con que funciona (homeorresis) –mientras las perturbaciones producidas desde su entorno no superen cierto grado–

, entonces el sistema es autopoyético. Los seres vivos, los ecosistemas y organizaciones sociales pueden considerarse sistemas abiertos.

Estos flujos de MEI se pueden representar en diagramas como el siguiente:

Figura 7: Entradas y Salidas

Para clarificar, las líneas de los diferentes flujos pueden representarse por diferentes colores o trazos. Este es el nivel cero de representación de un sistema, con las entradas y salidas de MEI que atraviesan sus límites.

Este tipo de representaciones se denomina diagrama de entrada y salida (E/S o U/O) o diagrama de caja negra, ya que no interesa mostrar qué sucede dentro del sistema.

Sistemas tecnológicos

Los sistemas tecnológicos, son aquellos diseñados por los seres humanos para que cumplan con una finalidad específica. Por eso se dice que son sistemas teleológicos artificiales (del griego telos = fin). La orientación para al fin que se busca suele definir la propiedad emergente del sistema tecnológico. En el ejemplo del automóvil, la propiedad emergente de marchar también se busca como finalidad o propósito del sistema.

Es conveniente aclarar que los sistemas son recortes de la realidad que alguien se propone estudiar o considerar; a ese recorte se le llama Abstracción. En algunos sistemas tecnológicos como un automóvil es sencillo identificar este recorte. Sin embargo, en la red de generación y distribución de energía eléctrica del país no resulta tan sencillo.

Algunos sistemas tecnológicos se caracterizan por procesar materia: son los sistemas de procesamiento de materia (SM). Estos están diseñados para producir, procesar, generar, transformar o distribuir materiales. Las industrias, las huertas, las licuadoras, etc. pueden considerarse SM.

Otros se caracterizan por procesar energía, los sistemas de procesamiento de energía (SE). Estos están diseñados para generar, transformar, distribuir energía. Los ventiladores, automóviles, represas hidroeléctricas, explosivos, etc. pueden considerarse SE. Los que se caracterizan por procesar información se llaman sistemas de información (SI). Están diseñados con el fin de generar, trasformar y distribuir información entre otras tareas. Los sistemas que controlan los automóviles, las redes sociales, los sistemas de punto de venta, el comercio electrónico, por mencionar algunos, son ejemplos de SI.

Desde la aparición del software, los SI han incorporado el software para hacer más eficiente su funcionamiento a un grado tal que se los denomina Sistemas Informáticos, acoplando la palabra "automático" a la palabra "información".

¿Cómo se construye el Software?

El software, como cualquier otro producto, se construye aplicando un proceso que conduzca a un resultado de calidad, que satisfaga las necesidades de quienes lo utilizan. Un proceso de desarrollo de software es una secuencia estructurada de actividades que conduce a la obtención de un producto de software. En definitiva, un proceso define quién está haciendo qué, cuándo y cómo alcanzar un determinado objetivo. En este caso el objetivo es construir un producto de software nuevo o mejorar uno existente.

Figura 9: Proceso de Construcción del Software

Pueden identificarse cuatro actividades fundamentales que son comunes a todos los procesos de software:

1. Especificación del software: donde clientes y profesionales definen el software que se construirá, sus características y las restricciones para su uso.
2. Desarrollo del software, donde se diseña y programa el software.
3. Validación del software, donde se controla que el software satisfaga lo que el cliente quiere.
4. Evolución del software, donde se incorporan mejoras y nuevas características que permitirán a ese producto adaptarse a las necesidades cambiantes del cliente y el mercado.

Si consideramos las características del software que se explicaron anteriormente, determinamos como conclusión que el software no se obtiene por medio de un proceso de manifactura en serie o como líneas de producción, sino que para obtenerlo usamos un proyecto, que se lo puede definir como un esfuerzo planificado, temporal y único, realizado para crear productos o servicios únicos que agreguen valor. Estos proyectos utilizan procesos que definen que tareas deben realizar las personas que trabajan en el proyecto, para obtener los resultados deseados, utilizando como apoyo herramientas que facilitarán su trabajo. En este caso el resultado deseado en el Producto de Software.

Figura 10: Relación entre Proceso, Proyecto y Producto en el desarrollo de Software

Diseño de Algoritmos

El hombre, en el día a día, se enfrenta constantemente a diferentes problemas que debe solucionar y para lograr solucionarlos hace uso de herramientas que le facilitan la tarea. Así, podemos pensar el uso de una calculadora para poder sumar el precio de los productos en un local y así cobrarle al cliente.

Al igual que la calculadora, la computadora también sirve para resolver problemas, pero la diferencia está en la capacidad de procesamiento de las computadoras, que hace que se puedan resolver problemas de gran complejidad, que, si los quisiéramos resolver manualmente, nos llevaría mucho tiempo o ni siquiera podríamos llegar a resolverlos.

Un programador es antes que nada una persona que resuelve problemas; el programador procede a resolver un problema, a partir de la definición de un algoritmo y de la traducción de dicho algoritmo a un programa que ejecutará la computadora.

En la oración anterior se nombran algunos conceptos que debemos profundizar:

Algoritmo: un algoritmo es un método para resolver un problema, que consiste en la realización de un conjunto de pasos lógicamente ordenados tal que, partiendo de ciertos datos de entrada, permite obtener ciertos resultados que conforman la solución del problema. Así, como en la vida real, cuando tenemos que resolver un problema, o lograr un objetivo, por ejemplo: "Tengo que atarme los cordones", para alcanzar la solución de ese problema, realizamos un conjunto de pasos, de manera ordenada y secuencial. Es decir, podríamos definir un algoritmo para atarnos los cordones de la siguiente forma:

1. Ponerme las zapatillas.
2. Agarrar los cordones con ambas manos.
3. Hacer el primer nudo.
4. Hacer un bucle con cada uno de los cordones.
5. Cruzar los dos bucles y ajustar.
6. Corroborar que al caminar los cordones no se sueltan y la zapatilla se encuentra correctamente atada.

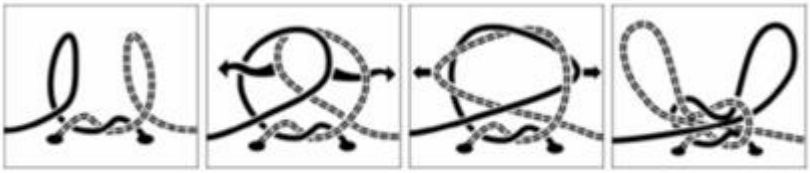

Figura 11: Algoritmo gráfico para atarse los cordones

El concepto de algoritmo es fundamental en el proceso de programación de una computadora, pero si nos detenemos a observar a nuestro alrededor, así como el ejemplo anterior podemos descubrir muchos otros: nos están dando un algoritmo cuando nos indican la forma de llegar a una dirección dada, seguimos algoritmos cuando conducimos un automóvil o cualquier tipo de vehículo. Todos los procesos de cálculo matemático que normalmente realiza una persona en sus tareas cotidianas, como sumar, restar, multiplicar o dividir, están basados en algoritmos que fueron aprendidos en la escuela primaria. Como se ve, la ejecución de algoritmos forma parte de la vida moderna.

Por otro lado, la complejidad de los distintos problemas que podamos abordar puede variar desde muy sencilla a muy compleja, dependiendo de la situación y la cantidad de elementos que intervienen. En casos de mayor complejidad suele ser una buena solución dividir al problema en diferentes subproblemas que puedan ser resueltos de manera independiente. De esta forma la solución final al problema inicial será determinada por las distintas soluciones de los problemas más pequeños cuya resolución es más sencilla.

Programa: luego de haber definido el algoritmo necesario, se debe traducir dicho algoritmo en un conjunto de instrucciones, entendibles por la computadora, que le indican a la misma lo que debe hacer; este conjunto de instrucciones conforma lo que se denomina, un programa.

Para escribir un programa se utilizan lenguajes de programación, que son lenguajes que pueden ser entendidos y procesados por la computadora. Un lenguaje de programación es tan sólo un medio para expresar un algoritmo y una computadora es sólo un procesador para ejecutarlo. Tanto el lenguaje de programación como la computadora son los medios para obtener un fin: conseguir que el algoritmo se ejecute y se efectúe el proceso correspondiente.

Algoritmos

Concepto

Es un método para resolver un problema, que consiste en la realización de un conjunto de pasos lógicamente ordenados, tal que, partiendo de ciertos datos de entrada, permite obtener ciertos resultados que conforman la solución del problema.

Características de los algoritmos

Las características fundamentales que debe cumplir todo algoritmo son:

- Un algoritmo debe ser preciso e indicar el orden de realización de cada paso.
- Un algoritmo debe estar específicamente definido. Es decir, si se ejecuta un mismo algoritmo dos veces, con los mismos datos de entada, se debe obtener el mismo resultado cada vez.
- Un algoritmo debe ser finito. Si se sigue un algoritmo, se debe terminar en algún momento; o sea, debe tener un número finito de pasos. Debe tener un inicio y un final.
- Un algoritmo debe ser correcto: el resultado del algoritmo debe ser el resultado esperado.
- Un algoritmo es independiente tanto del lenguaje de programación en el que se expresa como de la computadora que lo ejecuta.

Como vimos anteriormente, el programador debe constantemente resolver problemas de manera algorítmica, lo que significa plantear el problema de forma tal que queden indicados los pasos necesarios para obtener los resultados pedidos, a partir de los datos conocidos. Lo anterior implica que un algoritmo básicamente consta de tres elementos: Datos de Entrada, Procesos y la Información de Salida.

Figura 12: Estructura de un programa, datos de entrada y salida

Cuando dicho algoritmo se transforma en un programa de computadora:

- Las entradas se darán por medio de un dispositivo de entrada (como los vistos en el bloque anterior), como pueden ser el teclado, disco duro, teléfono, etc. Este proceso se lo conoce como entrada de datos, operación de lectura o acción de leer.
- Las salidas de datos se presentan en dispositivos periféricos de salida, que pueden ser pantalla, impresora, discos, etc. Este proceso se lo conoce como salida de datos, operación de escritura o acción de escribir.

Dado un problema, para plantear un algoritmo que permita resolverlo, es conveniente entender correctamente la situación problemática y su contexto, tratando de deducir del mismo los elementos ya indicados (entradas, procesos y salida). En este sentido entonces, para crear un algoritmo:

1- Comenzar identificando los resultados esperados, porque así quedan claros los objetivos a cumplir.
2- Luego, individualizar los datos con que se cuenta y determinar si con estos datos es suficiente para llegar a los resultados esperados. Es decir, definir los datos de entrada con los que se va a trabajar para lograr el resultado.
3- Finalmente si los datos son completos y los objetivos claros, se intentan plantear los procesos necesarios para pasar de los datos de entrada a los datos de salida.

Para comprender esto, veamos un ejemplo:

Problema:

Obtención del área de un rectángulo:

Altura: 5 cm

Base: 10 cm

1. **Resultado esperado:** área del rectángulo. Salida: área
   - Fórmula del área: base x altura.
2. Los datos con los que se dispone, es decir las entradas de datos son:
   - **Dato de Entrada 1**: altura: 5 cm
   - **Dato de Entrada 2**: base: 10 cm 3.
3. **El proceso para obtener el área del rectángulo es**: área = base * altura] área = 50

Herramientas para la representación gráfica de los algoritmos

Como se especificó anteriormente, un algoritmo es independiente del lenguaje de programación que se utilice. Es por esto, que existen distintas técnicas de representación de un algoritmo que permiten está diferenciación con el lenguaje de programación elegido. De esta forma el algoritmo puede ser representado

en cualquier lenguaje. Existen diversas herramientas para representar gráficamente un algoritmo. En este material presentaremos dos:

1.- Diagrama de flujo.
2.- Lenguaje de especificación de algoritmos: pseudocódigo

Diagramas de Flujo

Un diagrama de flujo hace uso de símbolos estándar que, unidos por flechas, indican la secuencia en que se deben ejecutar.

Estos símbolos son, por ejemplo:

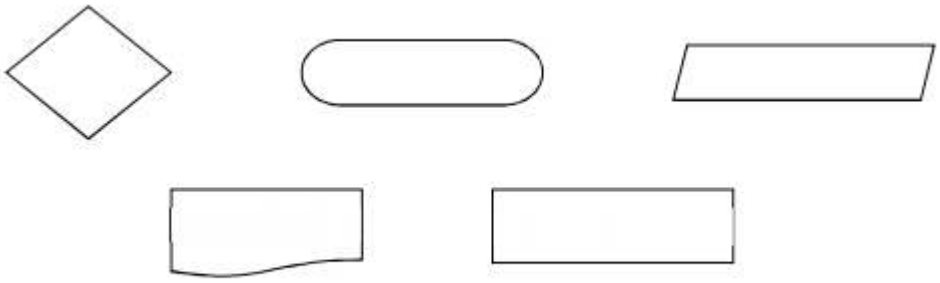

Figura 13: Símbolos utilizados en Diagramas de Flujo

Más adelante, a medida que profundizamos en los temas, retomaremos el uso de esta herramienta y mostraremos algunos ejemplos de su uso.

Pseudocódigo

Conocido como lenguaje de especificación de algoritmos, el pseudocódigo tiene una estructura muy similar al lenguaje natural y sirve para poder expresar algoritmos y programas de forma independiente del lenguaje de programación. Además, es muy utilizado para comunicar y representar ideas que puedan ser entendidas por programadores que conozcan distintos lenguajes. El pseudocódigo luego se traduce a un lenguaje de programación específico ya que la computadora no puede ejecutar el pseudocódigo. Su uso tiene ventajas porque permite al programador una mejor concentración de la lógica y estructuras de control y no preocuparse de las reglas de un lenguaje de programación específico.

Un ejemplo básico de pseudocódigo, considerando el ejemplo utilizado anteriormente, es el siguiente:

```
INICIO FUNCION CALCULAR_AREA
        DEFINIR BASE: 5
        DEFINIR ALTURA: 10
        DEFINIR AREA: BASE * ALTURA
        DEVOLVER AREA
```

FIN

Lenguajes de Programación

Concepto

Los lenguajes de programación son lenguajes que pueden ser entendidos y procesados por la computadora.

Tipos de Lenguajes de Programación

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

1. Lenguajes máquina
2. Lenguaje de bajo nivel (ensamblador)
3. Lenguajes de alto nivel

La elección del lenguaje de programación a utilizar depende mucho del objetivo del software. Por ejemplo, para desarrollar aplicaciones que deben responder en tiempo real como por ejemplo, el control de la velocidad crucero en un sistema de navegación de un auto1 , debemos tener mayor control del hardware disponible, por lo que privilegiaremos lenguajes de más bajo nivel que nos permitan hacer un uso más eficiente de los recursos. En cambio, para aplicaciones de escritorio como sistemas de gestión de productos, calendarios, correo electrónico, entre otras privilegiaremos la elección de lenguajes de más alto nivel que nos permitan ser más eficientes en cuanto a la codificación ya que, en términos generales, es necesario escribir menos líneas de código en los lenguajes de alto nivel, que para su equivalente en bajo nivel.

El lenguaje máquina

Los lenguajes máquina son aquellos que están escritos en lenguajes cuyas instrucciones son cadenas binarias (cadenas o series de caracteres -dígitos- O y 1) que especifican una operación, y las posiciones (dirección) de memoria implicadas en la operación se denominan instrucciones de máquina o código máquina. El código máquina es el conocido código binario.

En los primeros tiempos del desarrollo de los ordenadores era necesario programarlos directamente de esta forma, sin embargo, eran máquinas extraordinariamente limitadas, con muy pocas instrucciones por lo que aún era posible; en la actualidad esto es completamente irrealizable por lo que es necesario utilizar lenguajes más fácilmente comprensibles para los humanos que deben ser traducidos a código máquina para su ejecución.

Ejemplo de una instrucción:
1110 0010 0010 0001 0000 0000 0010 0000

El lenguaje de bajo nivel

Los lenguajes de bajo nivel son más fáciles de utilizar que los lenguajes máquina, pero, al igual, que ellos, dependen de la máquina en particular. El lenguaje de bajo nivel por excelencia es el ensamblador o

assembler. Las instrucciones en lenguaje ensamblador son instrucciones conocidas como mnemotécnicas (mnemonics). Por ejemplo, nemotécnicos típicos de operaciones aritméticas son: en inglés, ADD, SUB, DIV, etc.; en español, SUM, para sumar, RES, para restar, DIV, para dividir etc.

Lenguajes de alto nivel

Los lenguajes de alto nivel son los más utilizados por los programadores. Están diseñados para que las personas escriban y entiendan los programas de un modo mucho más fácil que los lenguajes máquina y ensambladores. Otra razón es que un programa escrito en lenguaje de alto nivel es independiente de la máquina; esto es, las instrucciones del programa de la computadora no dependen del diseño del hardware o de una computadora en particular. En consecuencia, los programas escritos en lenguaje de alto nivel son portables o transportables, lo que significa la posibilidad de poder ser ejecutados con poca o ninguna modificación en diferentes tipos de computadoras; al contrario que los programas en lenguaje máquina o ensamblador, que sólo se pueden ejecutar en un determinado tipo de computadora. Esto es posible porque los lenguajes de alto nivel son traducidos a lenguaje máquina por un tipo de programa especial denominado "compilador". Un compilador toma como entrada un algoritmo escrito en un lenguaje de alto nivel y lo convierte a instrucciones inteligibles por el ordenador; los compiladores deben estar adaptados a cada tipo de ordenador pues deben generar código máquina específico para el mismo.

Ejemplos de Lenguajes de Alto Nivel:
C, C++, Java, Python, VisualBasic, C#, JavaScript

¿Qué es un Programa?

Es un algoritmo escrito en algún lenguaje de programación de computadoras.
Pasos para la construcción de un programa

DEFINICIÓN DEL PROBLEMA

En este paso se determina la información inicial para la elaboración del programa. Es donde se determina qué es lo que debe resolverse con el computador, el cual requiere una definición clara y precisa.

Es importante que se conozca lo que se desea que realice la computadora; mientras la definición del problema no se conozca del todo, no tiene mucho caso continuar con la siguiente etapa.

ANÁLISIS DEL PROBLEMA

Una vez que se ha comprendido lo que se desea de la computadora, es necesario definir:
- **Los datos de entrada.**
- **Los datos de salida**
- **Los métodos y fórmulas que se necesitan para procesar los datos.**

Una recomendación muy práctica es la de colocarse en el lugar de la computadora y analizar qué es lo que se necesita que se ordene y en qué secuencia para producir los resultados esperados.

DISEÑO DEL ALGORITMO

Se puede utilizar algunas de las herramientas de representación de algoritmos mencionadas anteriormente. Este proceso consiste en definir la secuencia de pasos que se deben llevar a cabo para conseguir la salida identificada en el paso anterior.

CODIFICACIÓN

La codificación es la operación de escribir la solución del problema (de acuerdo a la lógica del diagrama de flujo o pseudocódigo), en una serie de instrucciones detalladas, en un código reconocible por la computadora. La serie de instrucciones detalladas se conoce como código fuente, el cual se escribe en un lenguaje de programación o lenguaje de alto nivel.

PRUEBA Y DEPURACIÓN

Se denomina prueba de escritorio a la comprobación que se hace de un algoritmo para saber si está bien realizado. Esta prueba consiste en tomar datos específicos como entrada y seguir la secuencia indicada en el algoritmo hasta obtener un resultado, el análisis de estos resultados indicará si el algoritmo está correcto o si por el contrario hay necesidad de corregirlo o hacerle ajustes.

Elementos de un Programa

Variables y Constantes

A la hora de elaborar un programa es necesario usar datos; en el caso del ejemplo del cálculo del área del rectángulo, para poder obtener el área del mismo, necesitamos almacenar en la memoria de la computadora el valor de la base y de la altura, para luego poder multiplicar sus valores.

Recordemos que no es lo mismo grabar los datos en memoria que grabarlos en el disco duro. Cuando decimos grabar en memoria nos estaremos refiriendo a grabar esos datos en la RAM. Ahora bien, para grabar esos datos en la RAM podemos hacerlo utilizando dos elementos, llamados: variables y constantes. Los dos elementos funcionan como fuentes de almacenamiento de datos, la gran diferencia entre los dos es que en el caso de las constantes su valor dado no varía en el transcurso de todo el programa.

Podría decirse que tanto las variables como las constantes, son direcciones de memoria con un valor, ya sea un número, una letra, o valor nulo (cuando no tiene valor alguno, se denomina valor nulo). Estos elementos permiten almacenar temporalmente datos en la computadora para luego poder realizar cálculos y operaciones con los mismos. Al almacenarlos en memoria, podemos nombrarlos en cualquier parte de nuestro programa y obtener el valor del dato almacenado, se dice que la variable nos devuelve el valor almacenado.
A continuación, se muestra un esquema, que representa la memoria RAM, como un conjunto de filas, donde, en este caso, cada fila representa un byte y tiene una dirección asociada.

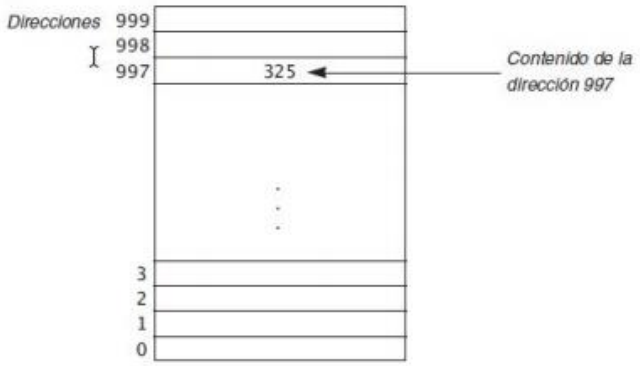

Figura 14: Representación de la memoria RAM

Variables

Son elementos de almacenamiento de datos. Representan una dirección de memoria en donde se almacena un dato, que puede variar en el desarrollo del programa. Una variable es un grupo de bytes asociado a un nombre o identificador, y a través de dicho nombre se puede usar o modificar el contenido de los bytes asociados a esa variable.

En una variable se puede almacenar distintos tipos de datos. De acuerdo al tipo de dato, definido para cada lenguaje de programación, será la cantidad de bytes que ocupa dicha variable en la memoria.

Type	Size	Range
byte	1 byte	−128 to 127
short	2 bytes	−32,768 to 32,767
int	4 bytes	−2,147,483,648 to 2,147,483,647
long	8 bytes	−9,223,372,036,854,775,808 to 9,223,372,036,854,775,807
float	4 bytes	$-3.40282347 \times 10^{38}$ to 3.4028347×10^{38}
double	8 bytes	$-1.79769313486231570 \times 10^{308}$ to $1.79769313486231570 \times 10^{308}$
char	2 bytes	one character
String	2 or more bytes	one or more characters

Tabla 1: Tabla de equivalencias de las unidades de memoria

Lo más importante de la definición de las variables y la elección del tipo de datos asociados es el significado de la variable, o su semántica: ya que en base al tipo de datos seleccionado serán las operaciones que podamos realizar con esa variable, por ejemplo: si tenemos la variable edad deberíamos seleccionar un tipo de datos como integer (número entero) ya que las operaciones relacionadas serán de comparación, sumas o restas y no es necesario tener una profundidad de decimales.

A continuación, se listan algunos de los tipos de datos más comunes y sus posibles usos:

Tipo de Datos	Significado	Ejemplos de uso
Byte	Número entero de 8 bits. Con signo	Temperatura de una habitación en grados Celsius
Short	Número entero de 16 bits. Con signo	Edad de una Persona
Int	Número entero de 32 bits. Con signo	Distancia entre localidades medida en metros
Long	Número entero de 64 bits. Con signo	Producto entre dos distancias almacenadas en variables tipo int como la anterior
Float	Número Real de 32 bits.	Altura de algún objeto
Double	Número Real de 64 bits.	Proporción entre dos magnitudes
Boolean	Valor lógico: true (verdadero) o false (falso)	Almacenar si el usuario ha aprobado un examen o no

Tabla 2: Tipos de dato

Para algunos lenguajes de programación es posible especificar si las variables numéricas tendrán o no signo (es decir que puedan tomar valores negativos y positivos, o sólo positivos). La elección de tener o no signo (definidas con la palabra reservada unsigned) depende del significado de la variable ya que al no tenerlo podremos almacenar el doble de valores, por ejemplo, una variable shortsin signo posee valores desde el 0 al 65535.

Constantes

Elementos de almacenamiento de datos. Representan una dirección de memoria en donde se almacena un dato pero que no varía durante la ejecución del programa. Se podría pensar en un ejemplo de necesitar utilizar en el programa el número pi, como el mismo no varía, se puede definir una constante pi y asignarle el valor 3.14.

Operadores

Los programas de computadoras se apoyan esencialmente en la realización de numerosas operaciones aritméticas y matemáticas de diferente complejidad. Los operadores son símbolos especiales que sirven para ejecutar una determinada operación, devolviendo el resultado de la misma.

Para comprender lo que es un operador, debemos primero introducir el concepto de Expresión. Una expresión es, normalmente, una ecuación matemática, tal como 3 + 5. En esta expresión, el símbolo más (+) es el operador de suma, y los números 3 y 5 se llaman operandos. En síntesis, una expresión es una secuencia de operaciones y operandos que especifica un cálculo.

Existen diferentes tipos de operadores:

Operador de asignación

Es el operador más simple que existe, se utiliza para asignar un valor a una variable o a una constante.

El signo que representa la asignación es el = y este operador indica que el valor a la derecha del = será asignado a lo que está a la izquierda del mismo.

Ejemplo en pseudocódigo:

Entero edad = 20
Decimal precio = 25.45

Operadores aritméticos Son operadores binarios (requieren siempre dos operandos) que realizan las operaciones aritméticas habituales: suma (+), resta (-), multiplicación (*), división (/) y resto de la división entera (%), por ejemplo 50%8=2 porque necesitamos obtener un número entero que queda luego de determinar la cantidad de veces que 8 entra en 50.

Ejemplo:

Expresión	Operador	Operandos	Resultado arrojado
5 * 7	*	5 , 7	35
6 + 3	+	6 , 3	9
20 - 4	-	20 , 4	16
50 % 8	%	50 , 8	2
45/5	/	45,5	9

Tabla Resumen Operadores Aritméticos:

Operador	Significado
+	Suma
-	Resta
*	Producto
/	División
%	Resto de la división entera

Los operadores unitarios

Los operadores unitarios requieren sólo un operando; que llevan a cabo diversas operaciones, tales como incrementar/decrementar un valor de a uno, negar una expresión, o invertir el valor de un booleano.

Operador	Descripción	Ejemplo	Resultado
++	operador de incremento; incrementa un valor de a 1	int suma=20; suma++;	suma=21
--	operador de decremento; Reduce un valor de a 1	int resta=20; resta--;	resta=19
!	operador de complemento lógico; invierte el valor de un valor booleano	boolean a=true; boolean b= !a;	b=false

Operadores Condicionales

Son aquellos operadores que sirven para comparar valores. Siempre devuelven valores booleanos: TRUE O FALSE. Pueden ser Relacionales o Lógicos.

Operadores relacionales

Los operadores relacionales sirven para realizar comparaciones de igualdad, desigualdad y relación de menor o mayor.

Los operadores relacionales determinan si un operando es mayor que, menor que, igual a, o no igual a otro operando. La mayoría de estos operadores probablemente le resultará familiar. Tenga en cuenta que debe utilizar " == ", no " = ", al probar si dos valores primitivos son iguales.

Operador	Significado
==	Igual a
!=	No igual a
>	Mayor que
>=	Mayor o igual que
<	Menor que
<=	Menor o igual que

Expresión	Operador	Resultado
a > b	>	true: si a es mayor que b false: si a es menor que b
a >= b	>=	true:si a es mayor o igual que b false: si a es menor que b
a < b	<	true: si a es menor que b false: si a es mayor que b
a <= b	<=	true: si a es menor o igual que b false: si a es mayor que b.
a == b	==	true: si a y b son iguales. false: si a y b son diferentes.
a != b	!=	true: si a y b son diferentes false: si a y b son iguales.

Operadores lógicos

Los operadores lógicos (AND, OR y NOT), sirven para evaluar condiciones complejas. Se utilizan para construir expresiones lógicas, combinando valores lógicos (true y/o false) o los resultados de los operadores relacionales.

Expresión	Nombre Operador	Operador	Resultado
a && b	AND	&&	true: si a y b son verdaderos. false: si a es falso, o si b es falso, o si a y b son falsos
a || b	OR	||	true: si a es verdadero, o si b es verdadero, o si a y b son verdaderos. false: si a y b son falsos.

Debe notarse que en ciertos casos el segundo operando no se evalúa porque no es necesario (si ambos tienen que ser true y el primero es false ya se sabe que la condición de que ambos sean true no se va a cumplir).

Rutinas

Las rutinas son uno de los recursos más valiosos cuando se trabaja en programación ya que permiten que los programas sean más simples, debido a que el programa principal se compone de diferentes rutinas donde cada una de ellas realiza una tarea determinada.

Una rutina se define como un bloque, formado por un conjunto de instrucciones que realizan una tarea específica y a la cual se la puede llamar desde cualquier parte del programa principal. Además, una rutina puede opcionalmente tener un valor de retorno y parámetros. El valor de retorno puede entenderse como el resultado de las instrucciones llevadas a cabo por la rutina, por ejemplo si para una rutina llamada sumar(a, b) podríamos esperar que su valor de retorno sea la suma de los números a y b. En el caso anterior, a y b son los datos de entrada de la rutina necesarios para realizar los cálculos correspondientes. A estos datos de entrada los denominamos parámetros y a las rutinas que reciben parámetros las denominamos funciones, procedimientos o métodos, dependiendo del lenguaje de programación.

Ejemplos de rutinas:

SumarPrecioProductos(precioProducto1, precioProducto2)
Rutina que realiza la suma de los precios de los productos comprados por un cliente y devuelve el monto total conseguido.

AplicarDescuento(montoTotal)
Rutina que a partir de un monto total aplica un descuento de 10% y devuelve el monto total con el descuento aplicado.

Entonces nuestro programa puede hacer uso de dichas rutinas cuando lo necesite. **Por ejemplo** cuando un cliente realice una compra determinada, podemos llamar a la rutina **sumarPrecioProductos** y cobrarle el monto devuelto por la misma. En el caso que el cliente abonara con un cupón de descuento, podemos entonces llamar a la rutina **aplicarDescuento** y así obtener el nuevo monto con el 10% aplicado.

Desarrollo de programas

Estructuras de programación

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

a) secuenciales
b) selectivas o de decisión
c) repetitivas

Las Estructuras de Control determinan el orden en que deben ejecutarse las instrucciones de un algoritmo: si serán recorridas una luego de la otra, si habrá que tomar decisiones sobre si ejecutar o no alguna acción o si habrá repeticiones.

Estructura secuencial

Es la estructura en donde una acción (instrucción) sigue a otra de manera secuencial. Las tareas se dan de tal forma que la salida de una es la entrada de la que sigue y así en lo sucesivo hasta cumplir con todo el proceso. Esta estructura de control es la más simple, permite que las instrucciones que la constituyen se ejecuten una tras otra en el orden en que se listan. Por ejemplo, considérese el siguiente fragmento de un algoritmo:

En este fragmento se indica que se ejecute la operación 1 y a continuación la operación 2

Figura 15: Diagrama de Flujo Secuencial

Estructura alternativa

Estas estructuras de control son de gran utilidad para cuando el algoritmo a desarrollar requiera una descripción más complicada que una lista sencilla de instrucciones. Este es el caso cuando existe un número de posibles alternativas que resultan de la evaluación de una determinada condición.

Este tipo de estructuras son utilizadas para tomar decisiones lógicas, es por esto que también se denominan estructuras de decisión o selectivas.

En estas estructuras, se realiza una evaluación de una condición y de acuerdo al resultado, el algoritmo realiza una determinada acción. Las condiciones son especificadas utilizando expresiones lógicas.

Las estructuras selectivas/alternativas pueden ser:

- Simples
- Dobles
- Múltiples

Alternativa simple (si-entonces/if-then)

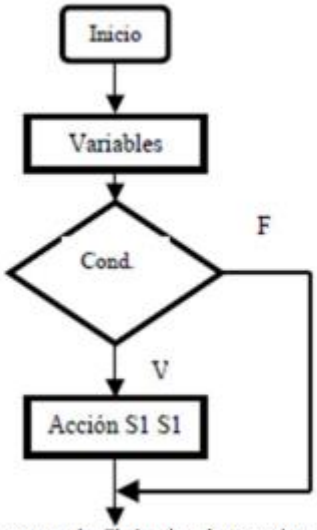

Figura 16: Diagrama de Flujo de alternativa simple

La estructura alternativa simple si-entonces (en inglés if-then) lleva a cabo una acción al cumplirse una determinada condición. La selección si-entonces evalúa la condición y:

- Si la condición es verdadera, ejecuta la acción S1
- Si la condición es falsa, no ejecuta nada.

En español:	En Inglés:
Si <condición> Entonces <acción S1> Fin_si	If <condición> Then <acción S1> End_if

Ejemplo:

INICIO
    ENTERO edad = 18
    SI (edad > 18)
    ENTONCES:
    puede manejar un auto
    FIN_SI
FIN

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

Existen limitaciones en la estructura anterior, y se necesitará normalmente una estructura que permita elegir dos opciones o alternativas posibles, de acuerdo al cumplimiento o no de una determinada condición:

- Si la condición es verdadera, se ejecuta la acción S1
- Si la condición es falsa, se ejecuta la acción S2

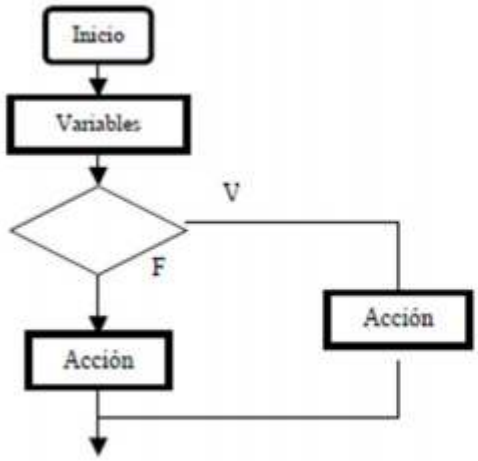

Figura 17: Diagrama de Flujo de alternativa doble

En español:

```
Si <condición>
entonces <acción S1>
sino <acción S2>
Fin_Si
```

En inglés:

```
If <condición>
then<acción>
else<acción S2>
End_if
```

Ejemplo:

INICIO
BOOLEANO afueraLlueve = verdadero
SI (afueraLlueve es verdadero)
ENTONCES:
me quedo viendo películas
SINO:
salgo al parque a tomar mates
FIN_SI
FIN

Alternativa de Decisión múltiple (según_sea, caso de/case)

Se utiliza cuando existen más de dos alternativas para elegir. Esto podría solucionarse por medio de estructuras alternativas simples o dobles, anidadas o en cascada. Sin embargo, se pueden plantear serios problemas de escritura del algoritmo, de comprensión y de legibilidad, si el número de alternativas es grande.

En esta estructura, se evalúa una condición o expresión que puede tomar n valores. Según el valor que la expresión tenga en cada momento se ejecutan las acciones correspondientes al valor.

PSEUDOCÓDIGO:

```
Según sea <expresión>
        <Valor1>: <acción1>
        <valor2>: <acción2>
        ...............................
        [<otro>: <acciones>]
fin según
```

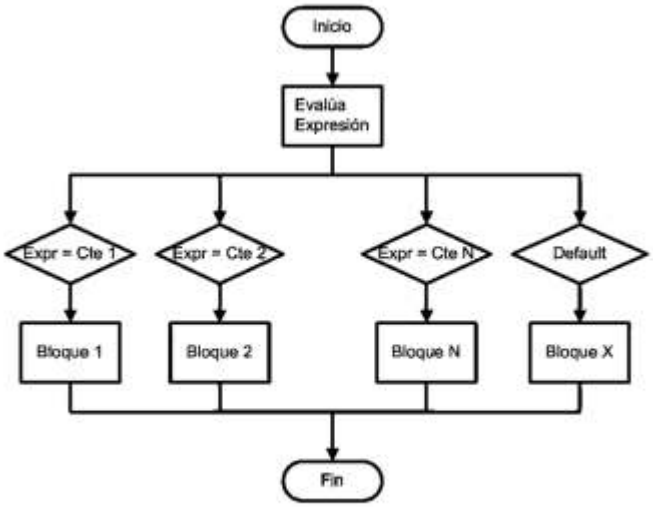

Figura 18: Diagrama de Flujo de Decisión Múltiple

Ejemplo en pseudocódigo:

```
INICIO
ENTERO posicionDeLlegada = 3
        SEGUN SEA posicionDeLlegada
                1: entregar medalla de oro
                2: entregar medalla de plata
                3: entregar medalla de bronce
        otro: entregar mención especial
FIN
```

Es importante mencionar que la estructura anterior puede ser escrita usando los condicionales vistos anteriormente de la siguiente forma:

```
INICIO
ENTERO posicionDeLlegada = 3
        SI (posicionDeLlegada = 1)
                ENTONCES:
                 entregar medalla de oro
                 SINO:
                SI (posicionDeLlegada = 2)
                        ENTONCES:
                        entregar medalla de plata
                 SINO:
                        SI (posicionDeLlegada = 3)
                        ENTONCES:
                                entregar medalla de bronce
                        SINO:
                                entregar mención especial
                        FIN_SI
                 FIN_SI
         FIN_SI
FIN
```

Podemos ver que usar condiciones anidadas podemos resolver el mismo problema, pero la estructura resultante es mucho más compleja y difícil de modificar.

Estructura repetitiva o iterativa

Durante el proceso de creación de programas, es muy común, encontrarse con que una operación o conjunto de operaciones deben repetirse muchas veces. Para ello es importante conocer las estructuras de algoritmos que permiten repetir una o varias acciones, un número determinado de veces.

Las estructuras que repiten una secuencia de instrucciones un número determinado de veces se denominan BUCLES. Y cada repetición del bucle se llama iteración.

Todo bucle tiene que llevar asociada una condición, que es la que va a determinar cuándo se repite el bucle y cuando deja de repetirse.

Un bucle se denomina también lazo o loop. Hay que prestar especial atención a los bucles infinitos, hecho que ocurre cuando la condición de finalización del bucle no se llega a cumplir nunca. Se trata de un fallo muy típico, habitual sobre todo entre programadores principiantes.

Hay distintos tipos de bucles:

- Mientras, en inglés: While
- Hacer Mientras, en inglés: Do While.
- Para, en inglés: For

Estructura mientras (while, en inglés)

Esta estructura repetitiva "mientras", es en la que el cuerpo del bucle se repite siempre que se cumpla una determinada condición.

Diagrama de Flujo

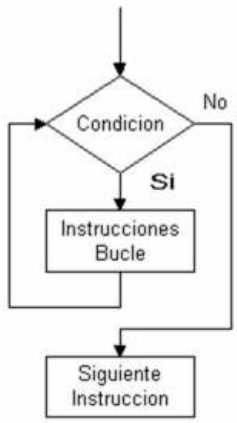

Figura 19: Diagrama de Flujo de la estructura "mientras"

Pseudocódigo:

```
Mientras <cond> hacer
  <acciones>
fin mientras
```

Ejemplo:

```
INICIO
 BOOLEANO tanqueLleno = falso
 MIENTRAS (tanqueLleno == falso)
 HACER:
            llenar tanque
 FIN_MIENTRAS
 // el tanque ya está lleno :)
FIN
```

Estructura hacer-mientras (do while, en inglés)

Esta estructura es muy similar a la anterior, sólo que a diferencia del while el contenido del bucle se ejecuta siempre al menos una vez, ya que la evaluación de la condición se encuentra al final. De esta forma garantizamos que las acciones dentro de este bucle sean llevadas a cabo, aunque sea una vez independientemente del valor de la condición.

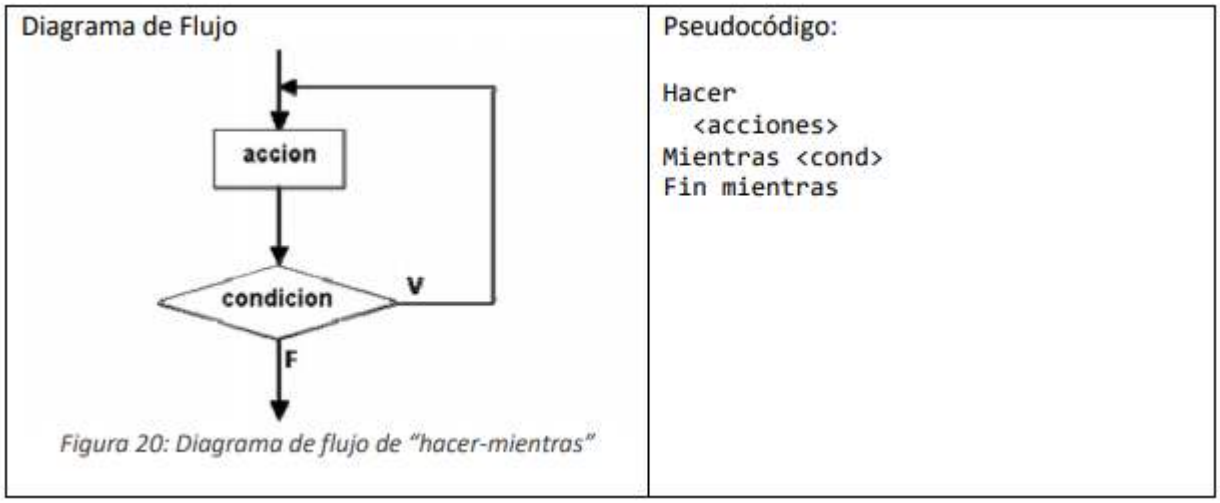

Figura 20: Diagrama de flujo de "hacer-mientras"

Ejemplo:

```
INICIO
BOOLEANO llegadaColectivo=false;
 HACER: esperar en la parada
   MIENTRAS (llegadaColectivo == false)
   FIN_MIENTRAS
FIN
```

Estructura para (for, en inglés)

La estructura for es un poco más compleja que las anteriores y nos permite ejecutar un conjunto de acciones para cada elemento de una lista, o para cada paso de un conjunto de elementos. Su implementación depende

del lenguaje de programación, pero en términos generales podemos identificar tres componentes: la inicialización, la condición de corte y el incremento.

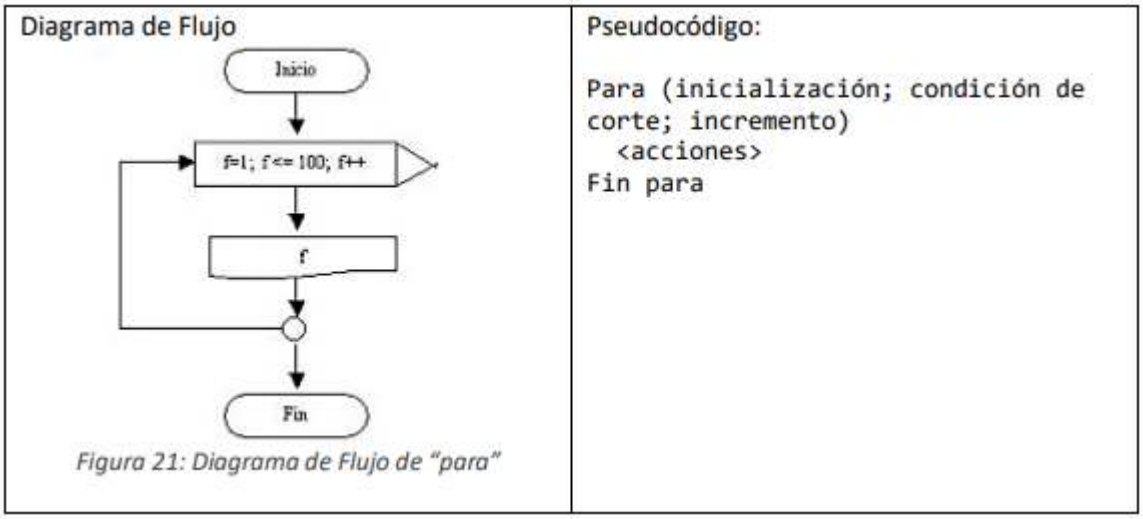

Figura 21: Diagrama de Flujo de "para"

Ejemplo:

```
INICIO
  PARA (ENTERO RUEDA = 1; RUEDA <= 4; RUEDA++)
      inflar_rueda (RUEDA)
  FIN_PARA
FIN
```

La ejecución del pseudocódigo anterior dará como resultado las siguientes llamadas a la función inflar():

1. inflar_rueda (1)
2. inflar_rueda (2)
3. inflar_rueda (3)
4. inflar_rueda (4)

Luego de esto podríamos suponer que hemos inflado las 4 cubiertas del auto y estamos listos para seguir viaje.

PRACTICA

1. Escriba una definición formal de software según la IEEE
2. Mencione, cuáles son las características del software
3. ¿Cuáles son los tres componentes principales de una computadora?
4. ¿Menciones tres dispositivos de entrada?
5. ¿En qué unidad se mide el tamaño de memoria de un computador?
6. ¿Cuándo es aplicable el modelo cliente-Servidor?
7. ¿Qué es un Sistema?
8. ¿Cuál es la característica principal de los sistemas?
9. ¿Cómo se representan los flujos de MEI?
10. ¿Cuáles son cuatro actividades fundamentales que son comunes a todos los procesos de software?
11. ¿Qué es un algoritmo?
12. ¿Qué características debe tener un algoritmo?
13. ¿Cuáles son las herramientas de representación gráfica de algoritmos?
14. Mencione los tres tipos de lenguajes de programación utilizados en la actualidad
15. De dos ejemplos de lenguajes de alto nivel
16. ¿Qué es un programa?
17. ¿Cuáles son los tres puntos principales que se debe definir durante el Análisis del Problema?
18. ¿Qué es una variable?
19. ¿Qué es una constante?
20. ¿Qué es un operador de asignación?
21. Mencione los operadores condicionales, relacionales y operadores lógicos
22. ¿Cuáles son las estructuras de programación?
23. ¿Cuántos tipos de estructuras alternativas hay?
24. De un ejemplo de estructura secuencial con diagrama de flujo
25. De un ejemplo de estructura alternativa simple con diagrama de flujo
26. De un ejemplo de estructura alternativa doble con diagrama de flujo
27. De un ejemplo de estructura repetitiva usando (hacer mientras) con diagrama de flujo