METODOLOGÍA PARA LA ALGORITMIA ORIENTADA A OBJETOS

RICARDO DE JESÚS BOTERO TABARES

Monografía para optar al título de

Especialista en Didáctica Universitaria

Universidad de Antioquia

Facultad de Educación

Departamento de Educación Avanzada

Medellín

Mayo de 2003

RESUMEN

La metodología propuesta permitirá a estudiantes de Ingeniería de Sistemas y áreas afines, docentes de lógica para programación de computadoras y profesionales ingenieros, plantear algoritmos basados en la teoría de la orientación a objetos para el desarrollo de software.

La teoría de la orientación a objetos constituye toda una filosofía de trabajo inmersa en ciertas áreas de las ciencias de la información, como el Análisis y Diseño de Sistemas, las Bases de Datos y la Programación, esencialmente.

Sin embargo, en la algoritmia aún no se han aplicado los conceptos de objeto, clase, herencia y polimorfismo, entre otros, propios de las tecnologías orientadas a objetos. Este trabajo pretende precisamente que dichos conceptos sean aplicados desde la algoritmia, basado en tres aspectos primordiales:

1. Los diagramas N – S (Nassi-Schneiderman)

Estos diagramas, propios del paradigma imperativo para desarrollo de software o programación estructurada, se utilizan para representar algoritmos con base en bloques de construcción rectangulares. Igualmente se podría utilizar cualquier otro tipo de representación, como el pseudocódigo o el diagrama de flujo, sin menoscabo de la aplicabilidad de los conceptos orientados a objetos a la algoritmia.

El uso de estos diagramas también facilitará la aplicación de dos conceptos esenciales de las metodologías estructuradas, también denominadas imperativas: modularidad y reutilización. Además, las estructuras secuencia, ciclo y decisión, así como las estructuras de datos, serán aplicadas con las nuevas herramientas y conceptos que aporta el paradigma de la orientación a objetos.

2. El UML: Lenguaje de Modelado Unificado

UML es un lenguaje estándar para escribir planos de software. En el contexto de este trabajo, se utilizará para representar los diferentes elementos presentes en el ámbito de la solución de un algoritmo: clase, objeto, herencia, encapsulación, etc.

3. La estrategia

Es el conjunto de herramientas lógicas, matemáticas e informáticas, que utiliza el ingeniero de software para diseñar y presentar un algoritmo orientado a objetos. La estrategia constituye el componente esencial de la metodología, dado que implica creatividad para solucionar la situación problemática, basada en las tres etapas siguientes:

- Identificación del problema a resolver

  - Interpretación adecuada.
  - Delimitación del problema.

- Análisis del problema

  - Identificación de clases.
  - Establecimiento de posibles relaciones de herencia.

- Desarrollo de la solución

  - Modelación matemática.
  - Propuesta de solución.

    - ✓ Identificación de entradas de datos y resultados requeridos (salidas).
    - ✓ Definición de constantes, variables y objetos.
    - ✓ Definición de métodos de almacenamiento.

En el esquema general de la metodología, presentado en el apartado 5.1.1, se especifican otras dos etapas (Implementación y Mantenimiento y evaluación), cuyo desarrollo se facilitará con la aplicación de la metodología propuesta.

TABLA DE CONTENIDO

PRÓLOGO

La propuesta desarrollada en esta monografía no pretende cambiar la manera de elaborar los conocimientos necesarios para adentrarse en el mundo de programación de computadoras, aunque presenta algunos aspectos concernientes a la pedagogía, a las nuevas tecnologías aplicadas al contexto educativo y a los procesos de enseñanza, aprendizaje y evaluación de algoritmos, plasmados en las relatorías incluidas en el apéndice F, conferidas de algunos trasfondos conceptuales tratados a lo largo de la Especialización.

La propuesta, sencilla para quienes han abonado los caminos de la programación orientada a objetos o, antagónicamente, para aquellos neófitos que nunca han incursionado en el área de los sistemas de información, presenta una visión distinta de los contenidos a impartir en asignaturas relacionadas con la lógica para programación de computadoras y de la estrategia para comprender dichos contenidos. Cabe anotar que el primer curso para aprender técnicas y métodos de programación de computadoras, ha adoptado diversos nombres al interior de los planes de estudio relacionados con ingeniería de Sistemas y programas afines: Algoritmos, Fundamentos de Programación, Diagramación Estructurada, Análisis de Algoritmos y algunos más. Por tradición, los contenidos de dichas asignaturas conducen a la programación imperativa o estructurada, estrategia que en los tres penúltimos decenios del siglo XX se utilizó para encontrar solución a situaciones posibles de abstraer e implementar en un sistema informático. Desde entonces hasta hoy, y hasta cuando los avances tecnológicos en materia de programación lo permitan, los programadores e ingenieros de software conciben soluciones basados en la reutilización de módulos, donde los grandes volúmenes de datos constituyen una unidad junto con las operaciones que los procesan, proceso ad hoc de la programación orientada a objetos.

El diseño del software moderno se genera a partir del paradigma de la orientación a objetos. Los últimos cambios acaecidos en los lenguajes de programación de propósito general así lo evidencian, donde se ha evolucionado de lo estructurado a lo objetual: de C a C++, de Pascal a Object Pascal, de COBOL a Object COBOL, de Basic a Visual Basic, etc. Incluso, los lenguajes de reciente eclosión en el mercado, como Java de Sun Microsystems y C# de Microsoft, están concebidos para el desarrollo de software orientado a objetos.

Los modelos de proceso de software también han sufrido cambios substanciales con el discurrir del tiempo. El modelo secuencial lineal, denominado también “ciclo de vida clásico” o “modelo en cascada”, incluye una serie de etapas que conducen a la generación de código basado en la programación procedimental, donde el famoso adagio de Maquiavelo “Divide y vencerás” se extiende por analogía a la programación para la creación de software a partir de componentes almacenados en librerías.

Otros modelos más recientes como los basados en prototipos y en ensamblaje de componentes y los modelos en espiral, concurrente e incremental, propugnan por la aplicación de técnicas relacionadas con la ingeniería del software orientada a objetos. ¿Por qué entonces se siguen explicando en las universidades algoritmos para programación netamente imperativa, teniendo en cuenta que áreas como el Análisis y Diseño de sistemas, Bases de Datos y Programación, tienden a la aplicación de la teoría orientada a objetos en sus más recientes adelantos?

Este trabajo, titulado Metodología para la Algoritmia Orientada a Objetos (MAOO), insta a la aplicación de los conceptos fundamentales subyacentes en el paradigma objetual, desde los inicios del plan de estudios de ingeniería de Sistemas, dentro de la asignatura comúnmente denominada Algoritmos y otras relacionadas, evitando el desfase mencionado anteriormente.

El abanico de posibilidades para representar un “algoritmo orientado a objetos” –concepto de uso poco frecuente en el argot académico de la informática, que hace innovadora esta propuesta–, debe ser similar al utilizado para la concepción de algoritmos estructurados: diagrama de flujo, pseudocódigo –el más utilizado entre los docentes universitarios y autores de textos relacionados con el tema– y diagrama rectangular o de bloques, también conocido como diagrama N-S (Nassi-Shneiderman), adoptado en la MAOO debido a múltiples razones:

- Los diagramas N-S son fáciles de entender; su carácter rectangular conduce al orden en la representación del algoritmo.

- Con ellos se puede representar cualquier sentencia de control, siendo claramente interpretables y por consiguiente fáciles de codificar.

- Su carácter gráfico los hace visuales por naturaleza. Con un vistazo general, el analista puede formar una idea acertada concerniente al orden de magnitud del algoritmo.

- El análisis y elaboración de un algoritmo orientado a objetos se facilita al combinar diagramas N-S con notación UML.

El Lenguaje Unificado de Modelado (UML) también se ha escogido en la MAOO para representar todo concepto inherente a la filosofía orientada a objetos: clase, objeto, herencia, generalización, agregación, etc. Esto posibilita que desde el mismo aprendizaje de las estrategias algorítmicas para solucionar problemas, nos iniciemos en el estudio de un lenguaje estándar para modelaje del análisis y diseño de un sistema concebido a partir de clases reutilizables.

Para terminar este esbozo introductorio, agradezco a todas las personas que de una u otra forma aporten observaciones, críticas fundamentadas,

modificaciones o cualquier otro aspecto que sirva para la mejora de la propuesta MAOO. ▲[i]

1. CONTEXTO DEL PROYECTO

1.1. DESCRIPCION DE LA SITUACIÓN PROBLEMÁTICA

Dos problemas se evidencian en la formación básica de los futuros analistas y desarrolladores de software. El primero tiene que ver con el cambio abrupto de paradigma para la modelación computacional, al pasarse en forma inmediata de una lógica basada en algoritmos estructurados –algoritmia imperativa–, a la basada en la teoría de objetos. El segundo se relaciona con la conceptualización pobre o incompleta de los elementos propios de la filosofía de software orientado a objetos.

El desarrollador utiliza los lenguajes orientados a objetos, pero las etapas previas de análisis y diseño de algoritmos las realiza de acuerdo al esquema tradicional estructurada, es decir, emplea herramientas de software orientadas a objetos no por convicción sino por las coyunturas determinadas por su fácil manejo y por las modernas tendencias comerciales que no dan otra alternativa de uso. Como consecuencia de lo anterior, se entregan productos finales a modo de software con reparos en su concepción y utilidad, lo que, como es obvio, se traduce en la indebida utilización de recursos y aumento consecuente de costos para la empresa.

En cuanto al cambio de paradigma de programación, gran parte de las universidades súbitamente pasan de una lógica procedimental estructurada a una programación fundamentada en objetos, lo que implica un cambio de visión

frente a un problema y su solución. Las nuevas tendencias mundiales de la ingeniería de sistemas en lo relacionado al software, han virado hacia el paradigma de programación orientado a objetos, trayendo beneficios a las comunidades académica y empresarial. En los centros educativos estos avances deben conducir, de motu proprio, a implantar desde los primeros semestres una lógica de programación que de una vez tenga el valor agregado de la orientación a objetos, no como un tema adicional, sino como una propuesta cognitiva desde los inicios del plan de formación, para garantizar la calidad del futuro ingeniero en lo relacionado con la construcción de modelos orientados a objetos.

La metodología a proponer constituye una innovación en el área de la Ingeniería de Sistemas relacionada con los fundamentos de programación orientada a objetos y el diseño de algoritmos, haciendo uso del Lenguaje de Modelado Unificado para la construcción de modelos (UML), creado por James Rumbaugh, Ivar Jacobson y Grady Booch y de la simbología para diagramación estructurada ideada por Nassi y Shneiderman.

La innovación radica en la combinación de los diagramas estructurados N-S con el lenguaje de objetos UML, dos herramientas que han impactado positivamente el historial de la producción de software, con el objeto de plantear una metodología que permita la elaboración de algoritmos cimentados en el paradigma de programación orientado a objetos.

En términos generales, el proyecto plantea una nueva forma de desarrollo del pensamiento algorítmico para la solución de problemas modelables en computadora (económicos, matemáticos, comerciales, científicos, etc.), a partir del paradigma orientado a objetos, cuyo pilar fundamental para la solución de situaciones problemáticas gira en torno a la reutilización del software. ▲i

1.2. DEFINICIÓN DEL PROBLEMA

Existe la necesidad de orientar la lógica de programación hacia el desarrollo de soluciones con una metodología de trabajo basada en el paradigma de software orientado a objetos, buscando formar un ingeniero competitivo, consciente de las modernas tendencias para el desarrollo de software.

La actual formación profesional del ingeniero de sistemas implica evidenciar el choque que se puede presentar entre la modelación con lógica imperativa y la lógica orientada a objetos. Se requiere entonces de un método que evite este impasse, mediante la concepción de una lógica que incluya de forma natural toda la teoría intrínseca al paradigma objetual, derivando en una modelación acertada desde los puntos de vista de satisfacción del usuario, uso eficiente de recursos y desarrollo de software acorde con las tendencias mundiales. ▲i

1.3. HIPÓTESIS PLANTEADA

Con una metodología de algoritmia orientada a objetos, se lograría fundamentación en los principios para la modelación y desarrollo del software basado en esta teoría, evitando cambios abruptos con el paradigma tradicional de la lógica de programación basada en procedimientos y funciones, no en métodos u operaciones. ▲[i]

1.4. ÁREAS DE APLICACIÓN

Diseño de algoritmos y desarrollo de programas; en general, ingeniería del software. ▲[i]

1.5. ANTECEDENTES Y JUSTIFICACIÓN

Es amplia la bibliografía difundida relacionada con programación, análisis y diseño orientado a objetos, pero escasa la atinente a la algoritmia sustentada en este paradigma.

Los currículos actuales de las universidades que imparten carreras en ciencias de la computación, incluyen asignaturas relacionadas con las metodologías estructuradas para el diseño de algoritmos, posponiendo para otro período la enseñanza del pilar de la abstracción: el análisis orientado a objetos.

Muchos docentes de ingeniería de sistemas evidencian el desfase que se presenta entre la formación inicial del estudiante en el área de algoritmos y las herramientas para el desarrollo de software que éste utiliza en períodos futuros; el estudiante adquiere formación para el diseño de algoritmos bajo el paradigma estructurado (promulgado por pensadores como Dijkstra, Wirth y Jacopini), pero después se ve enfrentado al desarrollo de software con herramientas que soportan el paradigma orientado a objetos, originando problemas de abstracción que hacen más lento el proceso de aprendizaje.

Este proyecto tratará de erradicar tal desfase mediante la propuesta de un método sencillo, fiable y eficaz, que fundamente a los estudiantes desde sus primeros niveles de formación en todo el bagaje conceptual subyacente en la filosofía orientada a objetos. ▲i

1.6. OBJETIVOS

1.6.1. OBJETIVO GENERAL

Diseñar una metodología que permita a estudiantes y profesionales en Ingeniería de Sistemas y áreas afines, elaborar algoritmos que conduzcan a la solución de problemas inmersos en una amplia gama de dominios, haciendo uso de tecnologías orientadas a objetos para desarrollo de software.

En otros términos, el objetivo de este trabajo consiste en aportar elementos metodológicos que permitan a analistas de software elaborar soluciones,

basados en la teoría orientada a objetos y en los fundamentos de la programación estructurada relacionados con la modularidad. ▲i

1.6.2. OBJETIVOS ESPECÍFICOS

- ✓ Explorar la visión actual de la ingeniería de sistemas y carreras afines, frente a la importancia e incidencia de la algoritmia imperativa tradicional y la orientación a objetos.

- ✓ Comparar las distintas simbologías para representar algoritmos: pseudocódigo, diagramas de flujo y diagramas N-S.

- ✓ Adaptar conceptos propios de la filosofía orientada a objetos para desarrollo de software, a una metodología para el diseño de algoritmos computacionales.

- ✓ Establecer indicadores de evaluación y resultados, para la futura evaluación de la metodología propuesta.

- ✓ Proponer una metodología para el desarrollo de algoritmos orientados a objetos. ▲i

1.7. POSIBLE ALCANCE

Inclusión de la metodología propuesta en los contenidos de asignaturas como Lógica de Programación, Algoritmos y Estructuras de Datos, asociadas a planes de estudio de algunas entidades de educación superior que oferten Ingeniería de Sistemas o áreas afines. ▲[i]

1.8. PERTINENCIA ACADÉMICA Y SOCIAL

La fundamentación académica de los ingenieros de sistemas y profesionales de áreas relacionadas –y en general de todo profesional– debe ser integral en todos los aspectos formativos, desde lo inherente al conocimiento específico hasta lo asociado a la política, la religión, el deporte, la sociedad y la cultura en general.

Uno de los pilares fundamentales en la formación del ingeniero de sistemas lo constituye su capacidad de análisis para plantear soluciones algorítmicas a problemas de diversa índole, con el objeto de diseñar programas fiables y robustos que solucionen los requerimientos del sector productivo. La metodología a proponer desde la MAOO inmiscuye procedimientos y técnicas basadas en la teoría de objetos para el desarrollo de software, que precisamente le aportan al estudiante de ingeniería –o al ingeniero–

herramientas para la solución de una gran gama de problemas de los sectores académico y empresarial.

Por fortuna, muchos centros de educación superior del país utilizan software académico (en esencia compiladores, intérpretes, gestores de bases de datos y sistemas operativos), diseñado y pensado para funcionar con los cimientos de teoría de objetos para el desarrollo de software. Ejemplos de ello son las aplicaciones visuales con interfaces gráficas, los lenguajes orientados a objetos, los sistemas cliente / servidor y el software orientado a la inteligencia artificial. Sin embargo, no educan para la óptima utilización del potencial que ofrecen tales herramientas, porque aún persisten en las viejas técnicas de modelamiento estructurado para el desarrollo de programas. Esto en gran parte se debe al subdesarrollo productivo en el ambiente local relacionado con la producción de software –claro que ya existen empresas nacionales que antagonizan con tal afirmación– , manifiesto en nuestro rol de adaptadores de tecnología en lugar de generadores de ella. ▲i

2. MARCO TEÓRICO[4]

2.1. INTRODUCCIÓN

Actualmente la Tecnología Orientada a Objetos (TOO) no solo se aplica a los lenguajes de programación; también es aplicable a los métodos de análisis y diseño (ADOO) y a otras áreas tales como las bases de datos (BDOO), las comunicaciones (COO) y la inteligencia artificial (IA) (Ver figura 1). Luego, para hacer desarrollo de sistemas de software basados en la TOO, hay que entender lo suficiente algunos conceptos del modelo de objetos y sus antecedentes históricos.

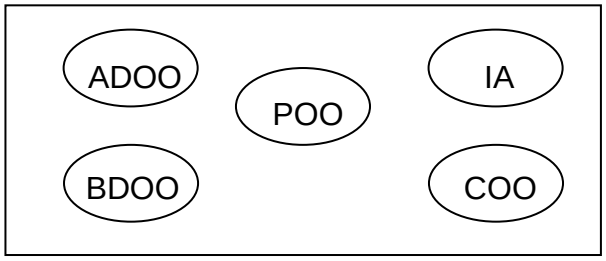


Figura 1. TOO

▲ i

[4] Es extensa la información que se puede hallar en la Red sobre la filosofía orientada a objetos para desarrollo de software. En particular, los numerales 2.1 al 2.4 del presente marco teórico fueron adaptados de http://uxmcc1.iimas.unam.mx/~cursos/Objetos/clase_1.html y http://www.monografias.com/trabajos/objetos/objetos.shtml.

2.2. RESEÑA HISTÓRICA SOBRE LOS LENGUAJES DE PROGRAMACIÓN ORIENTADOS A OBJETOS

El primer lenguaje que introdujo los conceptos de orientación a objetos fue *SIMULA 67* creado en Noruega, por un grupo de investigadores dirigido por O. J. Dahl y K. Nygaard, con el fin de realizar simulaciones discretas de sistemas reales. En esos tiempos no existían lenguajes de programación que se ajustaran a las necesidades del momento, así que se basaron en el lenguaje *ALGOL 60* y lo extendieron con conceptos de objetos, clases, herencia, polimorfismo por inclusión (que se obtiene introduciendo la herencia de clases) y procedimientos virtuales. Estos últimos permiten la sobrecarga de procedimientos, de tal forma que para un solo procedimiento se pueden tener varias implementaciones, dependiendo del nivel de jerarquía de la herencia de clases en el cual está definido un procedimiento. El lenguaje fue utilizado sobre todo en Europa y no tuvo mucho impacto comercial, sin embargo los conceptos que se definieron en él, se volvieron sumamente importantes para el futuro del desarrollo de software.

Alrededor de los años 70´s fue desarrollado el lenguaje de programación orientado a objetos llamado *SMALLTALK* en los laboratorios Xerox de Palo Alto, E. U. A. . Éste lenguaje adoptó los conceptos nombrados anteriormente como su fundamento. El hecho de ser creado en E.U.A., ayudó a que se introdujera a nivel mundial el término de Orientación a Objetos (Object Oriented) y que

cobrara importancia entre los diseñadores de lenguajes de programación. Los puntos importantes de este lenguaje fueron, por un lado, adoptar el concepto de objeto y clase como núcleo del lenguaje, y por el otro la programación interactiva, incorporando las ideas ya conocidas de lenguajes funcionales, es decir, que se tuviese un lenguaje interpretado y no compilado.

En 1985, Bjarne Stroustrup extendió el lenguaje de programación *C* a *C*++, es decir *C* con conceptos de clases y objetos. También por esta fecha se creó desde sus bases el lenguaje *EIFFEL* por B. Meyer. Ambos manejan conceptos de objetos y herencia de clases. La herencia múltiple se introduce pensando en dar mayor flexibilidad a los desarrolladores. Sin embargo, actualmente la herencia múltiple se ha evitado por agregar complejidad en la estructura de clases. Ambos lenguajes tuvieron importancia entre 1985 y la primera mitad de los noventas.

El mas reciente lenguaje orientado a objetos, *JAVA*, fue anunciado formalmente en una conferencia del mes de mayo de 1995; desarrollado por James Golsling y su grupo de colaboradores en *SUN MICROSYSTEMS*, hereda conceptos de *C*++, los simplifica y evita la herencia múltiple. Introduce el término de interfaz y de herencia múltiple de interfaces. JAVA obtiene una rápida aceptación gracias a los *applets*, que son unos programas insertados en páginas *WEB* dentro del código HTML. Estos programas pueden transportarse a través de la Internet para brindarle al usuario mayor interactividad con las páginas WEB. *JAVA* introduce también la programación concurrente y distribuida. El lenguaje es

mitad compilado y mitad interpretado, dando como resultado la portabilidad a distintas plataformas. *JAVA* aun sigue evolucionando y se espera que en los próximos años logre la madurez adecuada para convertirse en el lenguaje de desarrollo de mayor relevancia. ▲[i]

2.3. EL MODELO ORIENTADO A OBJETOS

El modelo orientado a objetos se define en cuatro conceptos básicos:

i) Objeto

ii) Clase

iii) Herencia

iv) Envío o paso de mensajes

Los primeros tres conceptos se refieren a la parte estructural del modelo y el cuarto, que corresponde a mensajes, se refiere a la parte del comportamiento.

Definición de Objeto: En términos generales, un objeto es una abstracción conceptual del mundo real que se puede traducir a un lenguaje computacional o de programación orientada a objetos, (figura 2).

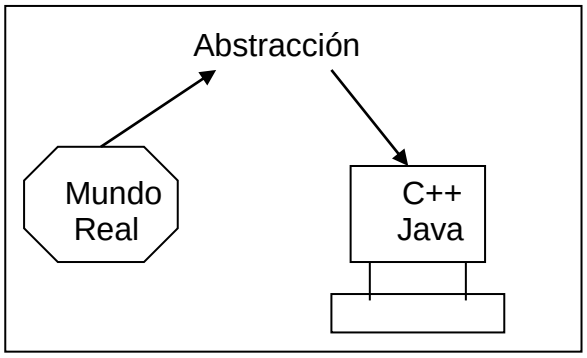


Figura 2. Objeto

La abstracción de objeto se caracteriza por tener una identidad única que lo distingue de otros objetos. También tiene un estado, que permite informar lo que éste representa y su comportamiento, es decir lo que él sabe hacer (figura 3).

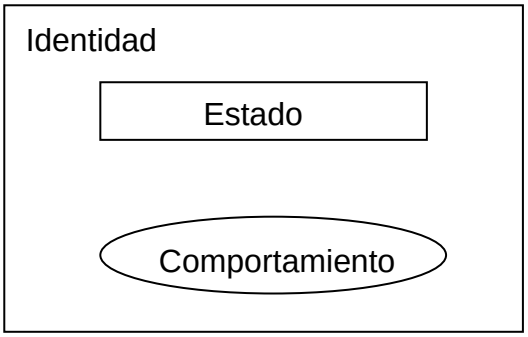


Figura 3. objeto

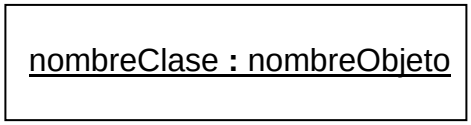
Hablando en términos computacionales, la identidad del objeto se puede interpretar como la referencia. El estado de objeto es una lista de variables conocidas como sus atributos, cuyos valores representan el estado que caracteriza al objeto. El comportamiento es una lista de métodos,

procedimientos, funciones u operaciones que un objeto puede ejecutar a solicitud de otros objetos. Los objetos también se conocen como instancias de clase.

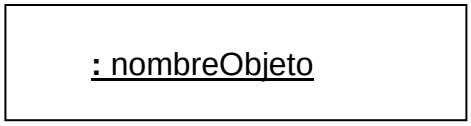
El concepto de objeto es similar al concepto de campo variable, muy utilizado en programación de computadoras. Un objeto puede tener varias referencias, así como una variable puede ser apuntada por otras variables tipo puntero. Una variable puede cambiar su contenido en tiempo de ejecución; un objeto se puede instanciar varias veces en tiempo de ejecución. Toda variable se asocia a un tipo de datos estándar (numérico entero, numérico flotante, carácter, booleano, etc.) o a un tipo de dato abstracto (TDA) definido por el programador mediante un registro (Empleado, Producto, Viaje, Transacción, etc.). Todo objeto se asocia a una clase, ”preempacada” (como las clases Applet, Object ó Graphics de Java) o definida por el programador (Empleado, Producto, Viaje, Transacción, etc.). Sin embargo, un objeto es mucho más que una variable, así como una clase sobrepasa al concepto de registro:

En una variable confluyen sólo datos; en un objeto confluyen al tiempo los datos y las operaciones para actuar sobre dichos datos.

Un objeto se representa en UML así:



A veces se referencia a un objeto genérico (como los de la clase Object de Java) omitiendo el nombre de la clase, así:



El concepto de clase en la teoría de objetos es similar al de registro en las metodologías estructuradas. En resumen:

Clase = datos + operaciones

Las operaciones son subprogramas generalmente a modo de función, no de procedimiento, que en ciertos lenguajes de programación orientados a objetos como C++ y Java adoptan el nombre de métodos.

Definición de Clase: Una clase define un conjunto de objetos que tienen estado y comportamiento. Estos objetos deben tener los mismos atributos con valores posiblemente diferentes asignados a estos, y el mismo conjunto de métodos que definen su proceder al ser ejecutados. Por otro lado, también se puede ver una clase como un molde, esquema o patrón que define la forma de sus objetos o como la estructura estática que define el esquema de estados y el comportamiento que van a tener los objetos y a partir del cual se van a ir creando dinámicamente objetos pertenecientes a dicha clase.

Definición de Herencia: Una clase, llamada clase base o superclase, puede extender sus atributos y métodos a varias clases subalternas denominadas subclases. Esto significa que una subclase, aparte de los atributos y métodos propios, tiene incorporados los atributos y métodos heredados de la superclase. Una subclase puede a su vez comportarse como una superclase y heredar a otras clases, creando de esta manera la jerarquía de herencia. Cuando una clase hereda de varias superclases se presenta la herencia múltiple.

Definición de Envío de Mensajes: ¿ Qué sucede cuando los objetos desean comunicarse entre sí?. Un objeto recibe un estímulo externo debido a la solicitud de un servicio que se traduce en la invocación de un método de éste objeto. Al ejecutarse el método, el objeto puede solicitar servicios de otros objetos, enviándoles mensajes que implican a su vez la invocación de sus métodos, y dentro de estos, nuevamente invocar servicios de otros objetos y así sucesivamente. Este envío de mensajes en cascada representa el comportamiento dinámico del modelo de objetos.

Para terminar este esbozo sobre los componentes básicos del modelo orientado a objetos, conviene diferenciar entre lenguajes orientados a objetos y lenguajes que no lo son. Tenemos lenguajes:

- Base-objetos: Lenguajes que no tienen herencia. Presentan un concepto parecido a clase y alguna forma de crear objetos a partir de ésta.

- Orientados a objetos: Presentan los conceptos de objeto, clase y herencia de clases.

Estos últimos son los más utilizados en la actualidad, dado que soportan el paradigma orientado a objetos en su totalidad. ▲[i]

2.4. BENEFICIOS Y PROBLEMAS DERIVADOS DE LA ORIENTACIÓN A OBJETOS

Beneficios que se obtienen del desarrollo con programación orientada a objetos

Día a día los costos del hardware decrecen. Así surgen nuevas áreas de aplicación cotidianamente: procesamiento de imágenes y sonido, bases de datos multimediales, automatización de oficinas, ambientes de ingeniería de software, etc. Aún en las aplicaciones *tradicionales* encontramos que definir interfaces hombre-máquina 'a-la-Windows' suele ser bastante conveniente.

Sin embargo, no ocurre lo mismo con el software, pues día a día los costos de su producción siguen aumentando; el mantenimiento y la modificación de sistemas complejos suele ser una tarea trabajosa; cada aplicación, (aunque tenga aspectos similares a otra) suele encararse como un proyecto nuevo. Estos problemas aún no han sido solucionados en forma completa. Pero como los objetos son portables (teóricamente) mientras que la herencia permite la reusabilidad del código orientado a objetos, es más sencillo modificar código existente porque los objetos no interaccionan excepto a través de mensajes; en

consecuencia un cambio en la codificación de un objeto no afectará la operación con otro objeto siempre que los métodos respectivos permanezcan intactos. La introducción de tecnología basada en objetos como herramienta conceptual para analizar, diseñar e implementar aplicaciones a partir de componentes reutilizables, permite obtener software modificable, fácilmente extensible. La reutilización del código disminuye el tiempo que se utiliza en el desarrollo y hace que la producción de software sea más intuitiva porque la gente piensa naturalmente en términos de objetos más que en términos de algoritmos de software.

Problemas derivados de la utilización de la programación orientada a objetos en la actualidad

Un sistema orientado a objetos, por lo visto, puede parecer un paraíso virtual. El problema sin embargo surge en la *implementación* de tal sistema. Muchas compañías oyen acerca de los beneficios de un sistema orientado a objetos e invierten gran cantidad de recursos. Luego comienzan a darse cuenta que han impuesto una nueva cultura que es ajena a los programadores actuales (algo que la propuesta MAOO trata de enmendar). Específicamente los siguientes problemas suelen aparecer repetidamente:

Curvas de aprendizaje largas. Un sistema orientado a objetos ve al mundo en una forma única. Involucra la conceptualización de todos los elementos de un programa, desde los subsistemas a los datos, en la forma de objetos. Toda la

comunicación entre los objetos debe realizarse en la forma de mensajes. Esta no es la forma en que están escritos los programas orientados a objetos actualmente; al hacer la transición a un sistema orientado a objetos la mayoría de los programadores deben capacitarse nuevamente antes de poder usarlo.

Si los conceptos de la orientación a objetos se imparten a los estudiantes desde los semestres iniciales de los planes de formación, las curvas de aprendizaje se minimizarían substancialmente.

Dependencia del lenguaje. A pesar de la portabilidad conceptual de los objetos en un sistema orientado a objetos, en la práctica existen muchas dependencias. Muchos lenguajes orientados a objetos están compitiendo actualmente para dominar el mercado. Cambiar el lenguaje de implementación de un sistema orientado a objetos no es una tarea sencilla; por ejemplo C++ soporta el concepto de herencia múltiple mientras que SmallTalk no lo soporta; en consecuencia la elección de un lenguaje tiene ramificaciones de diseño muy importantes.

Determinación de las clases. Una clase es un molde que se utiliza para crear nuevos objetos. En consecuencia es importante crear el conjunto de clases adecuado para un proyecto. La definición de las clases es más un arte que una ciencia. Si bien hay muchas jerarquías de clase predefinidas, usualmente se deben crear clases específicas para la aplicación que se esté desarrollando. Luego, en 6 meses ó 1 año se da cuenta que las clases que se establecieron no

son pertinentes; en ese caso será necesario reestructurar la jerarquía de clases devastando totalmente la planificación original.

Rendimiento. En un sistema donde **todo** es un objeto y **toda** interacción es a través de mensajes, el tráfico de éstos afecta el rendimiento. A medida que la tecnología avanza, la velocidad de microprocesamiento y la potencia de la memoria aumentan, situación que beneficia el rendimiento del sistema; sin embargo, un diseño de una aplicación orientada a objetos que no tenga en cuenta el rendimiento no será viable comercialmente, por ello se deben diseñar algoritmos con estructuras de datos y órdenes de magnitud adecuados.

Idealmente, debería existir una metodología independiente del lenguaje, fácil de aprender y reestructurar, que ataque estos problemas eficientemente, tal como en MAOO se propone. ▲[i]

2.5. COMPARACIÓN DE SIMBOLOGÍAS PARA REPRESENTAR ALGORITMOS

Existen varias técnicas para la representación de algoritmos estructurados: pseudocódigo, diagramas de flujo y diagramas de bloque o N-S. Cada una de ellas es utilizada extensamente por distintos docentes y desarrolladores dado que, usufructuadas en forma debida, cumplen con el objetivo de expresar la lógica algorítmica inherente a la solución de un problema. La diferencia fundamental entre estas técnicas estriba en el impacto visual que ofrecen. En

particular, el pseudocódigo se expresa con líneas de texto que incluyen identificadores y palabras reservadas debidamente organizadas e indentadas. El diagrama de flujo incluye una serie de símbolos estandarizados unidos entre sí por líneas que definen el flujo de las instrucciones, lo que obliga al uso de extensos espacios físicos en el papel para expresar las ideas. Esto último no ocurre con los diagramas N-S, ya que su compactabilidad los hace fáciles de entender e interpretar. Además, cuando el analista visualiza de manera general, con “un abrir y cerrar de ojos”, un algoritmo expresado con diagrama N-S, de inmediato puede formarse una idea sobre la complejidad del mismo. Por ello, en el proyecto MAOO se ha optado por utilizar este tipo de diagramas, que aunados a la simbología básica de UML servirán de fundamento a la propuesta.

El siguiente ejemplo, solucionado de acuerdo a las metodologías estructuradas, explica mejor el porqué se eligen para la MAOO los diagramas N-S.

Ejemplo convencional con métodos estructurados

Escribir algoritmo que lea N valores tipo entero, entre los cuales se pueden presentar datos repetidos. Hallar el valor promedio, el dato mayor y la cantidad de veces que este último se encuentra en la muestra.

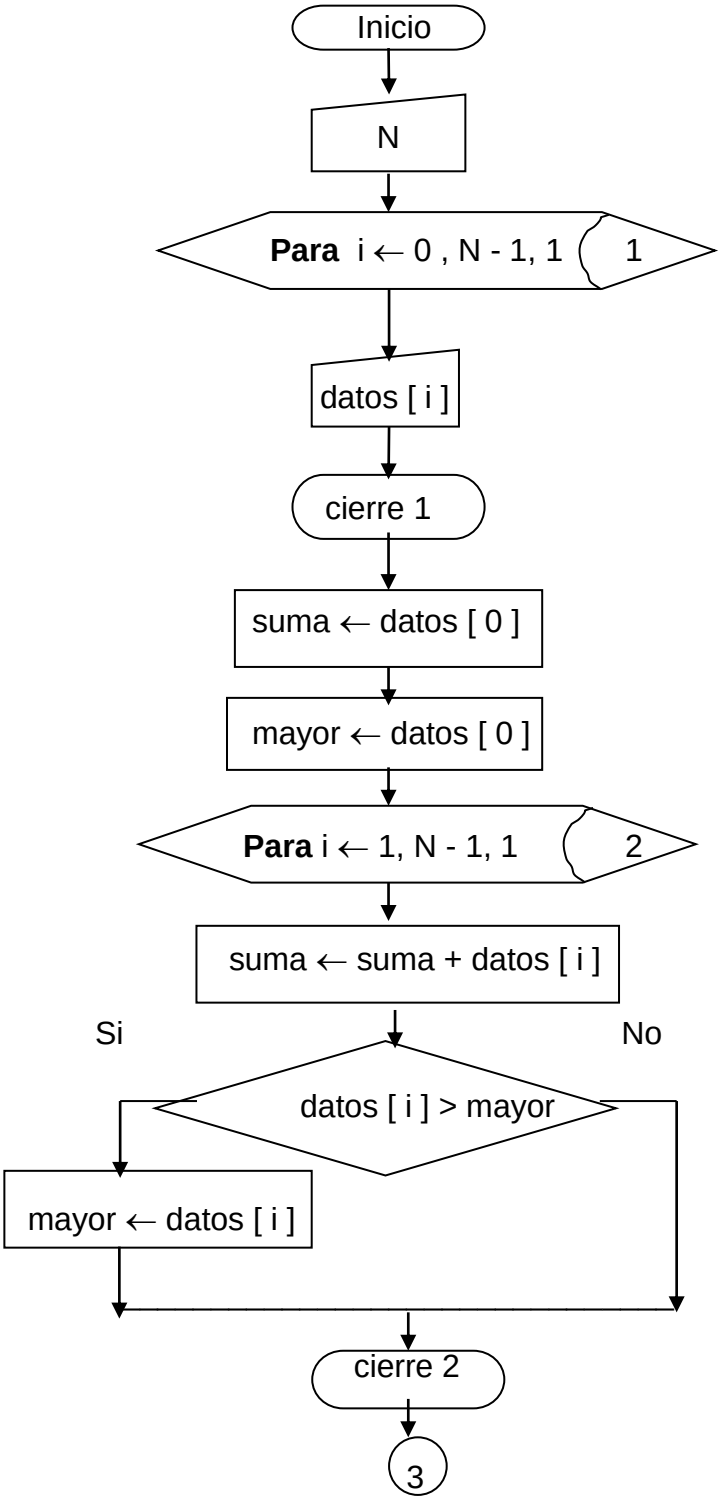
La solución implica el uso del vector *datos* con un rango de posiciones que varía desde cero hasta N – 1, para almacenar los N valores enteros ingresados

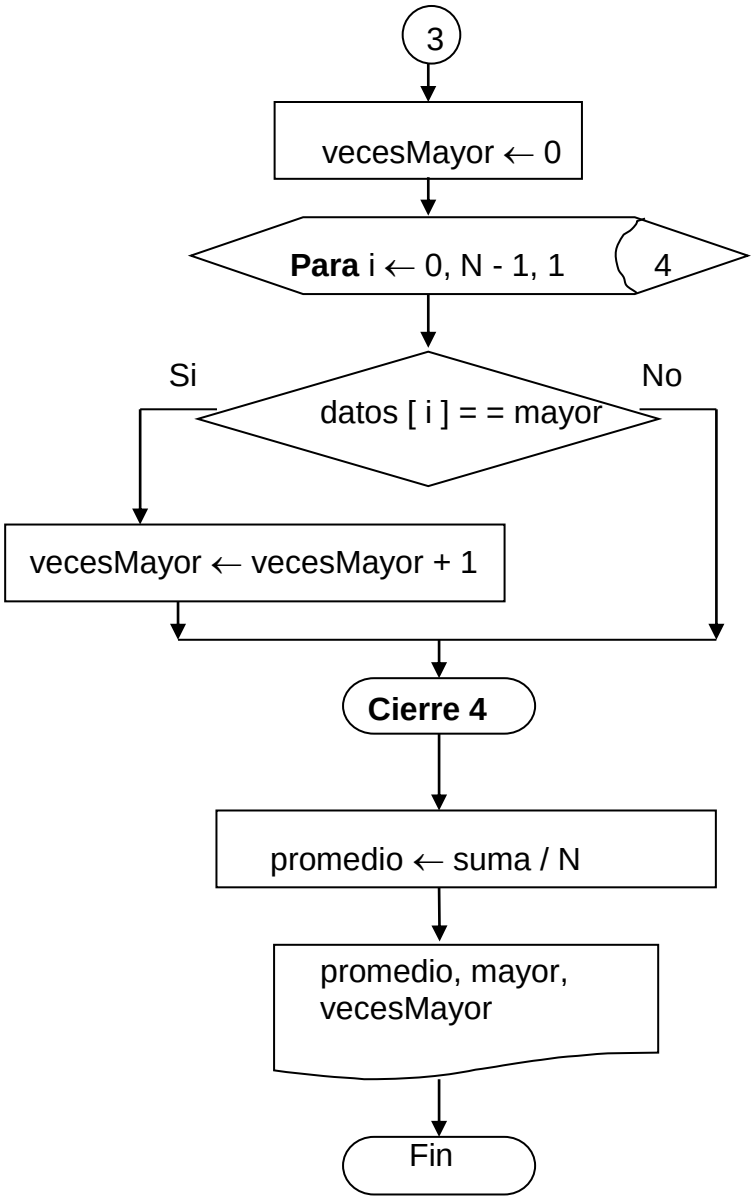
vía teclado o entrada estándar. Dado que los nombres de las variables son mnemónicos, los algoritmos mismos se autodocumentan.

i) Pseudocódigo

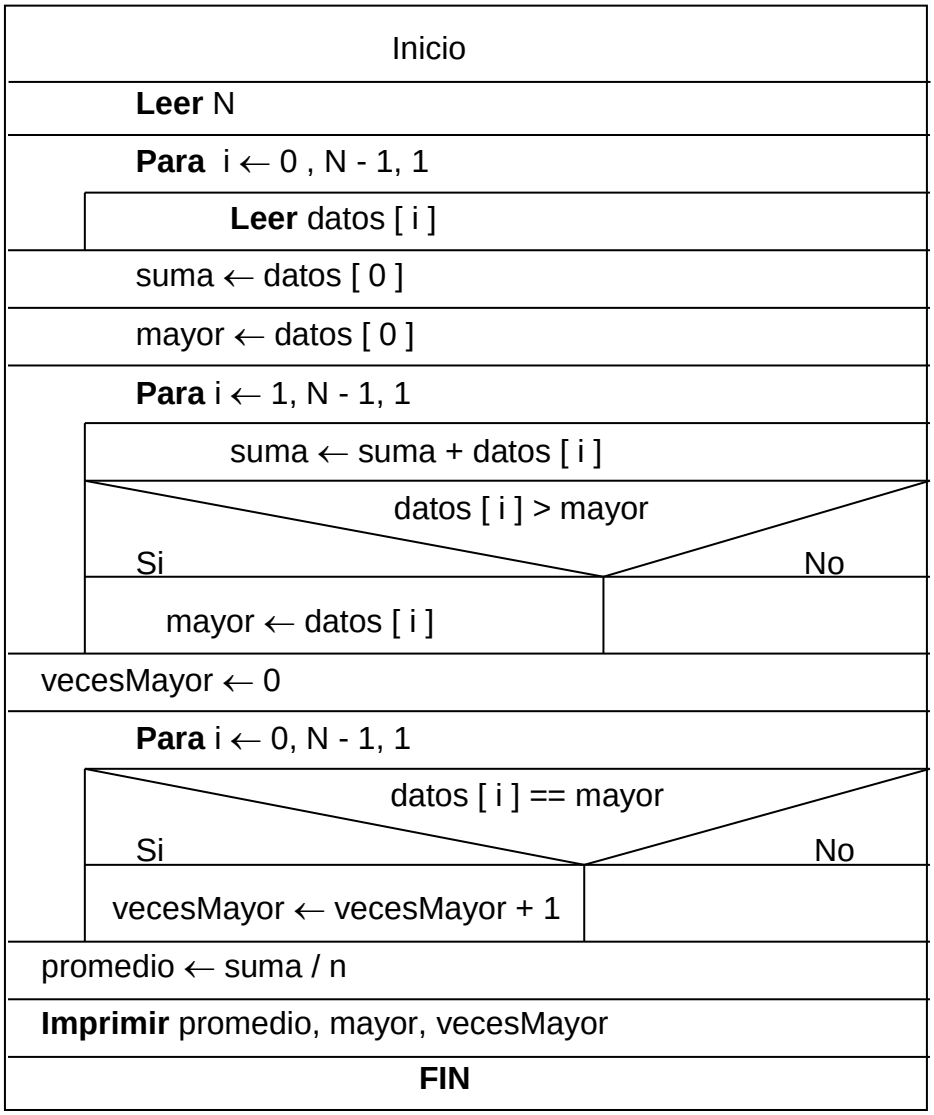
```
Inicio
    Leer N
    Para  i ← 0 , N - 1, 1
            Leer datos [ i ]
    Fin_para
    suma ← datos [ 0 ]
    mayor ← datos [ 0 ]
    Para i ← 1, N -1, 1
            suma ← suma + datos [ i ]
            Si (datos [ i ] > mayor )
                    Entonces     mayor ← datos [ i ]
            Fin_si
    Fin_para
    vecesMayor ← 0
    Para i ← 0, N - 1, 1
            Si ( datos [ i ] = = mayor )
                    Entonces     vecesMayor ← vecesMayor + 1
            Fin_ii
    Fin_para
    promedio ← suma / N
    Imprimir promedio, mayor, vecesMayor
Fin
```

ii) Diagrama de flujo




3
vecesMayor ← 0
Para i ← 0, N - 1, 1
4
Si
No
datos [i] = = mayor
vecesMayor ← vecesMayor + 1
Cierre 4
promedio ← suma / N
promedio, mayor,
vecesMayor
Fin

iii) Diagrama de bloques o N – S



Note el impacto visual, la claridad del diagrama N – S y el espacio reducido que ocupa el algoritmo dentro del papel.

La mejor manera de comprender MAOO es mediante el estudio de la lógica para programación orientada a objetos, el análisis de ejercicios donde sea

aplicado, y la lectura atenta y crítica de la presente propuesta. Las relatorias presentadas en el apéndice F constituyen un acercamiento entre MAOO y ciertos tópicos pedagógicos de interés general, que se pueden profundizar a partir de las fuentes recomendadas en las mismas relatorías. ▲[i]

2.6. NOTACIONES UTILIZADAS EN EL ANÁLISIS Y DISEÑO ORIENTADO A OBJETOS

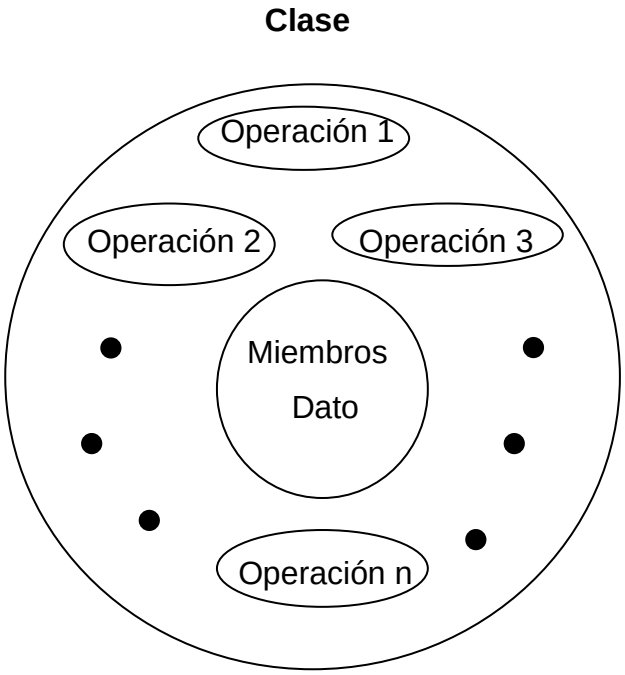
El posicionamiento que ha logrado el paradigma de la orientación a objetos como pilar para la producción de software moderno, ha generado una serie de notaciones gráficas para representar todos los conceptos que subyacen en ésta teoría: clase, clase plantilla, objeto, relación o asociación, generalización, agregación, interfaz, etc.

Estas notaciones, origen de una “torre de Babel” para el modelamiento de sistemas orientados a objetos, fueron propuestas por expertos en el tema y han tenido un impacto positivo en el desarrollo y aplicación de toda la teoría en el ambiente del análisis y diseño de sistemas de información. Las más sobresalientes han sido las notaciones de Martin / Odell, OMT, Booch´93, Coad / Yourdon y UML; este último trata de estandarizar el modelaje recogiendo los aspectos más relevantes de notaciones anteriores y es el que se adopta en la MAOO por su versatibilidad y aceptación en el mercado del software.

A continuación se presenta un breve resumen de las notaciones mencionadas, donde solo se indica la representación de un objeto y un diagrama de clases ilustrativo. Vale anotar que cada notación incluye una amplia gama de símbolos. [▲i]

2.6.1. NOTACIÓN MARTIN / ODELL

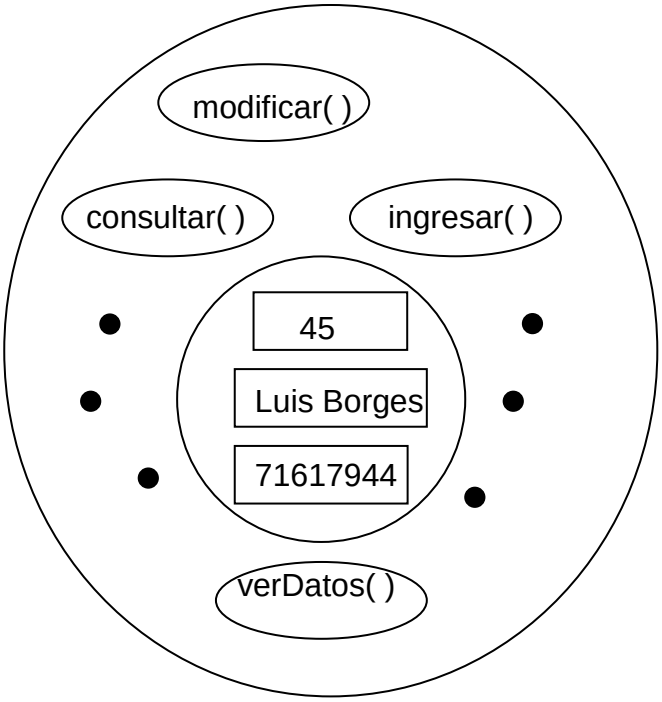
James Martin y James J. Odell presentan la siguiente notación:



Como se observa, Martin y Odell utilizan círculos y elipses para representar una clase y sus componentes.

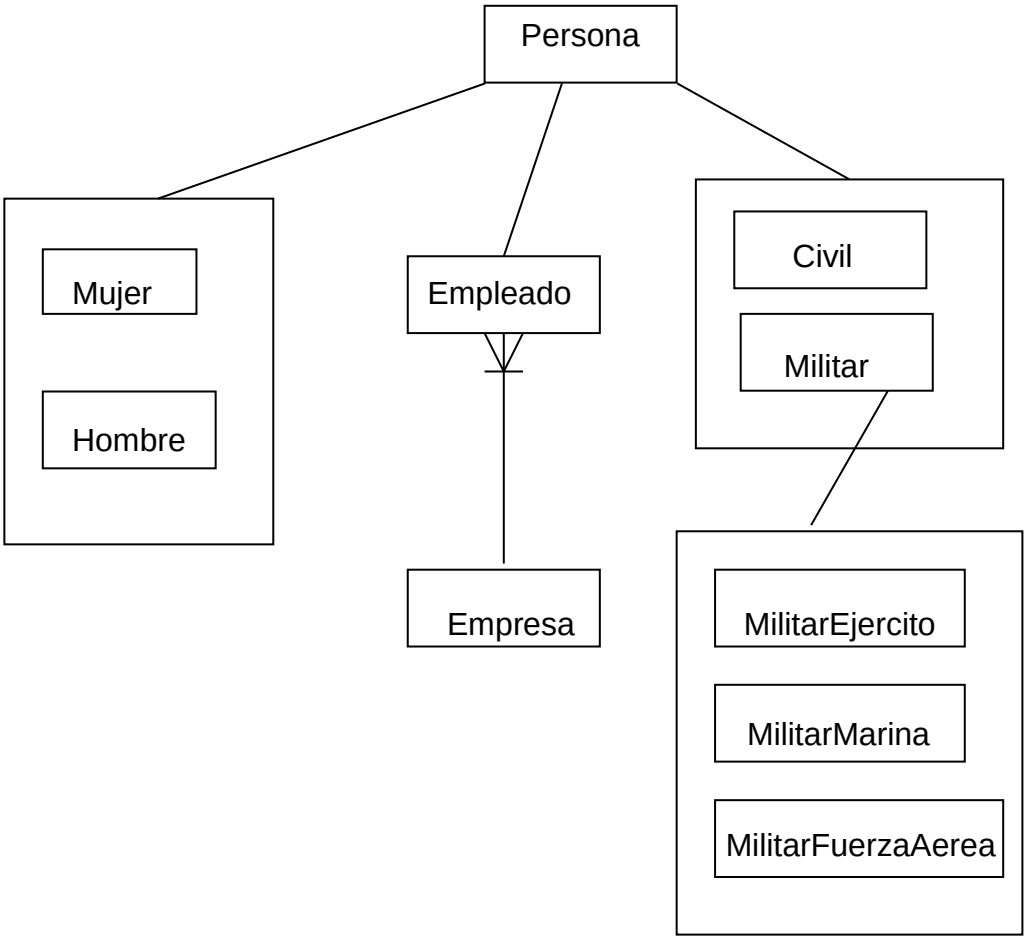
La representación de un objeto es idéntica a la de una clase; a las operaciones se les asigna nombres particulares y los miembros dato son instanciados.

Objeto



El diagrama de clases se basa en rectángulos, algo común a la gran mayoría de las notaciones, tal como lo evidencia la siguiente figura.

Diagrama de clases



▲[i]

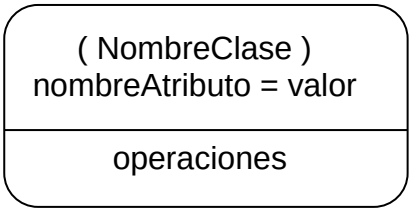
2.6.2. NOTACIÓN OMT

La notación OMT (Object Modeling Technique), propuesta por James Rumbaugh y su grupo de colaboradores, ha sido una de las más populares en el ámbito de la producción de software.

OMT se utiliza en el análisis y diseño de sistemas orientados a objetos. La actividad de análisis crea tres modelos: el modelo de objetos (una representación de objetos, clases, jerarquías y relaciones), el modelo dinámico (una representación del comportamiento de los objetos del sistema) y el modelo funcional (una representación a alto nivel del flujo de información a través del sistema, similar a un diagrama de flujo de datos). Una información más extensa puede encontrarse en la referencia bibliográfica RUMBAUGH, 1996.

Los símbolos en OMT para representar una clase, un objeto, así como un diagrama de clases ilustrativo, son los siguientes:

Clase



Objeto

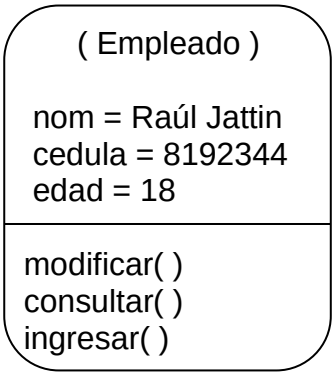
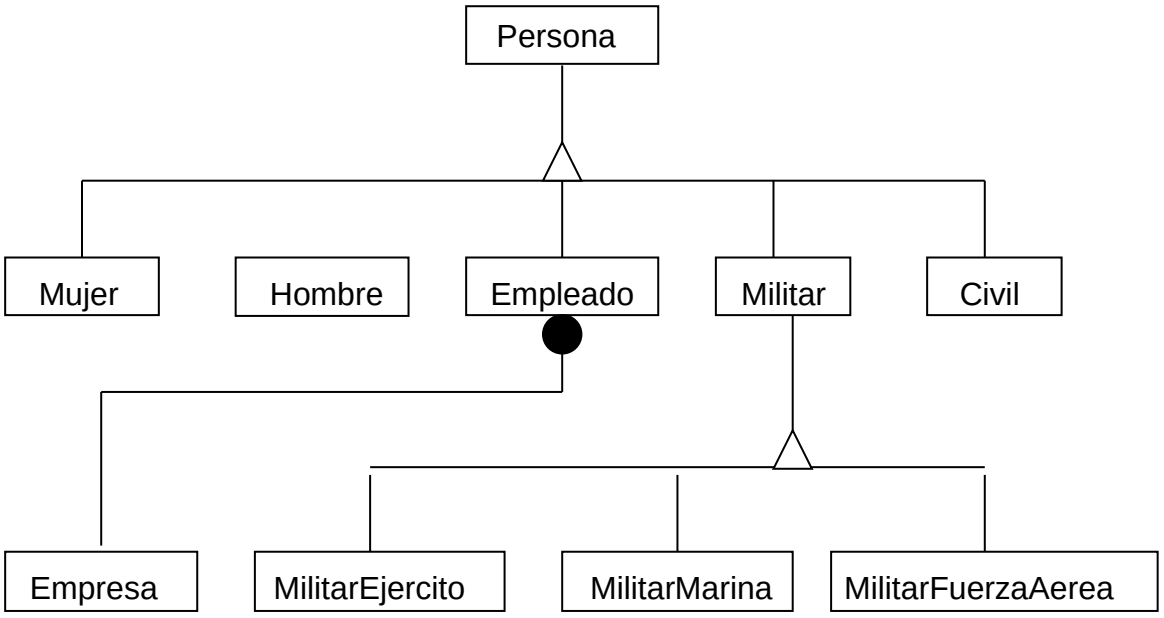


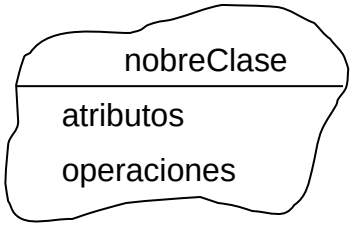
Diagrama de clases



▲[i]

2.6.3. NOTACIÓN BOOCH´93

Grady Booch propuso esta notación en el año de 1993. El método de Booch abarca un micro y un macro proceso de desarrollo. El nivel micro define un conjunto de tareas de análisis que se reaplican en cada etapa en el macro proceso, por esto mantiene un enfoque evolutivo.



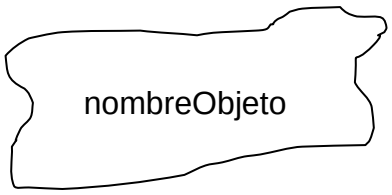
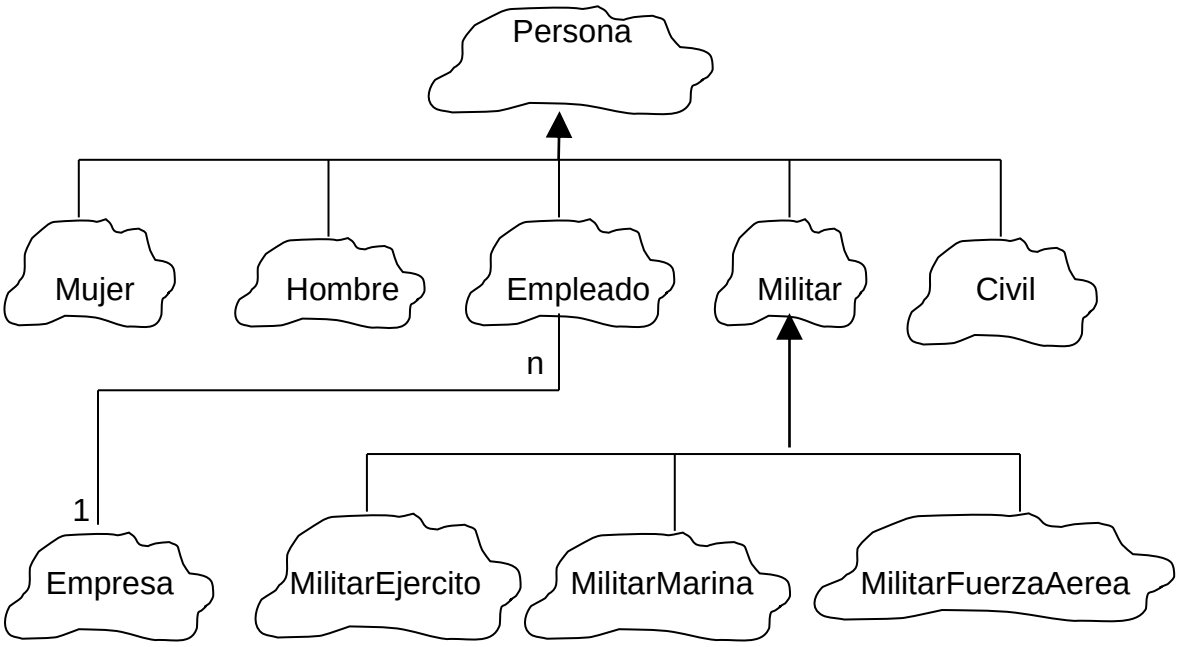


Diagrama de clases Booch'93

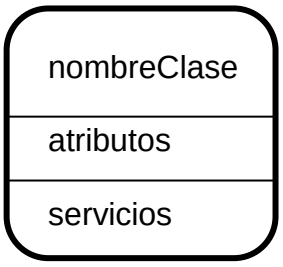


▲[i]

2.6.4. NOTACIÓN COAD / YOURDON

Según PRESSMAN, "el método de P. Coad y E. Yourdon, se considera, con frecuencia, uno de los métodos de análisis y diseño orientado a objetos más sencillo de aprender. La notación del modelado es relativamente simple y las reglas para desarrollar el modelo de análisis son evidentes".

Clase



Objeto

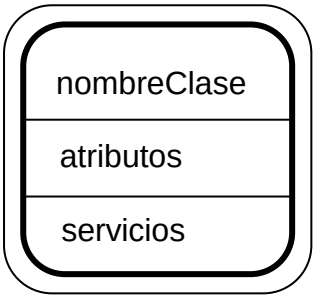
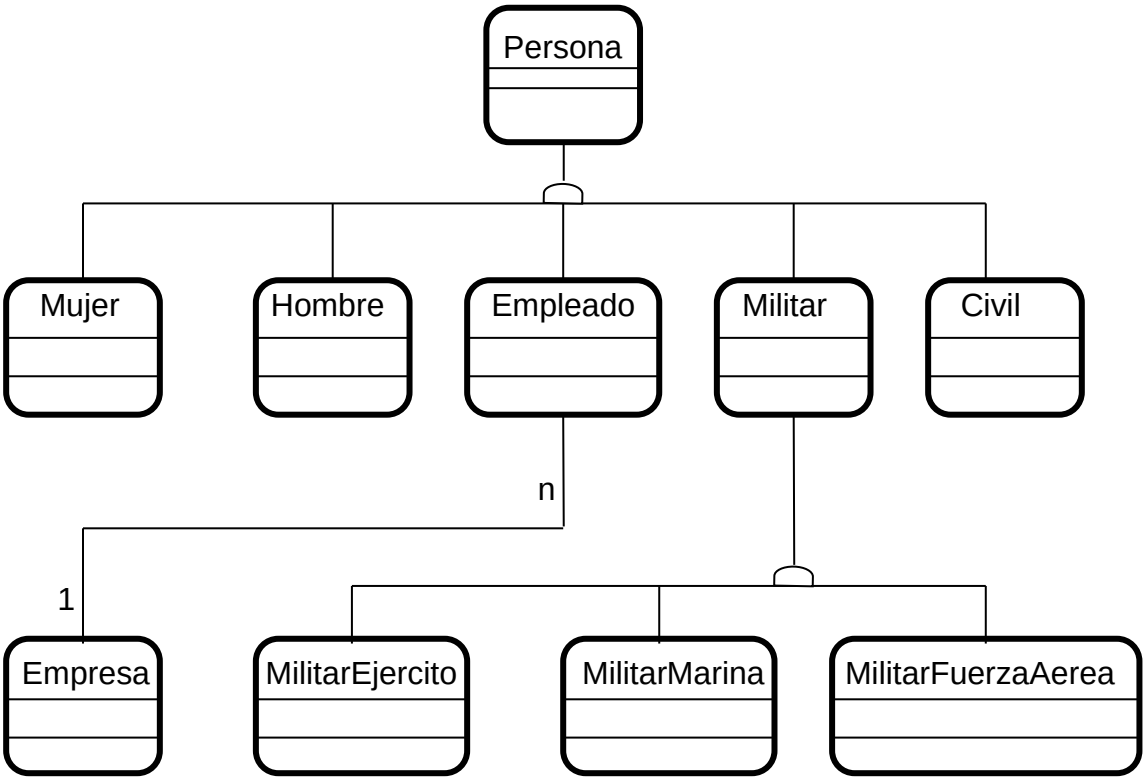


Diagrama de clases Coad / Yourdon



▲[i]

2.6.5. NOTACIÓN UML

El Lenguaje Unificado de Modelado (UML) fue propuesto por la “trilogía de amigos” Grady Booch, James Rumbaugh e Ivar Jocobson, quienes ya habían elaborado las metodologías Booch´93, OMT Y OOSE, respectivamente. Ellos decidieron, a partir de un extenso conocimiento sobre el tema y aprovechando la coyuntura de su trabajo mancomunado en Rational Software Corporation, unificar con un lenguaje de modelado orientado a objetos que cumpliera tres metas fundamentales:

i) Modelar sistemas, desde el concepto hasta los artefactos ejecutables, utilizando técnicas orientadas a objetos.

ii) Cubrir las cuestiones relacionadas con el tamaño inherente a los sistemas complejos y críticos.

iii) Crear un lenguaje de modelado utilizable tanto por las personas como por las máquinas.

Debido a que UML impactó de manera positiva a escala mundial –y aún lo sigue haciendo– la industria del software, como herramienta indispensable para el análisis y diseño orientado a objetos, se ha escogido en la MAOO para representar los aspectos básicos asociados a situaciones problemáticas posibles de modelar mediante algoritmos y en consecuencia almacenables en computadora.

El apéndice B expone una visión condensada de UML. Para mayor información, remitirse a las referencias BOOCH, RUMBAUGH y SCHMULLER. ▲[i]

2.6.6. OTRAS NOTACIONES

Como respuesta de los metodologistas a los avances de la industria del software latentes en unas aplicaciones cada vez más complejas y en los nuevos lenguajes de programación basados en el paradigma de la orientación a

objetos, surgieron enfoques alternativos de modelado importantes en su momento. Fusion, Shlaer-Mellor, R. Edge, Taylor y OMT-2, constituyeron métodos completos, aunque todos tenían sus puntos fuertes y débiles. ▲[i]

3. MARCO REFERENCIAL

Para el análisis del estado de la algoritmia tradicional frente a la orientada a objetos en la Universidad en general, se tomaron algunos planes de estudio e investigaciones de centros educativos superiores de España, México, Argentina y Colombia. La elección de los países y las universidades fue por completo arbitraria, a excepción de Colombia por obvios motivos. Se podrían tomar otras naciones y entidades en la búsqueda de un análisis exhaustivo; sin embargo, las asignaturas tomadas en cuenta son una muestra representativa que permite extraer claras observaciones sobre el tratamiento que se le da a la algoritmia en las instituciones de educación superior del mundo. [▲i]

3.1. COLOMBIA

Universidad de Antioquia – Facultad de Ingeniería – Algoritmos I, Estructuras de Datos I y II

El curso Algoritmos I de la Facultad de Ingeniería de la Universidad de Antioquia, se enfoca al estudio de los componentes funcionales de una computadora, los algoritmos estructurados con manejo de vectores y matrices, subalgoritmos y programación orientada a objetos. Este último tema indica que se tiene en cuenta el paradigma de la orientación a objetos para el diseño de algoritmos.

En el curso Estructuras de Datos I se estudian métodos lineales de disposición de datos en memoria, como las listas ligadas, listas generalizadas, pilas y colas, y métodos no lineales como árboles binarios y grafos, bajo un enfoque por completo estructurado, tal como lo demuestra la bibliografía recomendada para la asignatura.

En Estructura de Datos ll, se someten a estudio técnicas para manejo de la memoria externa en sistemas de archivo, sin el tratamiento explícito de la programación orientada a objetos, aunque se introduce en ella a partir del texto Estructuras de Datos con C y C++, de Langsam / Tenenbaum.

Universidad de Antioquia – Facultad de Ingeniería – Laboratorio Integrado de Sistemas

El grupo Sicosis de la Facultad de Ingeniería de la Universidad de Antioquia viene diseñando, por iniciativa del profesor Rios Castrillón, una herramienta didáctica para la iniciación a los objetos.

Dicha herramienta sería utilizada por estudiantes de la Facultad en asignaturas como Algoritmos I y II, Lenguajes de Programación y laboratorios asociados a las materias mencionadas, además también la usufructuarían otros estudiantes, docentes e investigadores internos y externos a la Universidad.

La principal característica de este producto en elaboración, consiste en un compilador que procesa pseudocódigo orientado a objetos en lenguaje natural hispano, permitiendo al estudiante una comprensión rápida y certera de los fundamentos de la orientación a objetos, sin la necesidad de incursionar en la parafernalia de los lenguajes orientados a objetos de gran circulación en el actual mercado del software.

Universidad Cooperativa de Colombia – Facultad de Ingeniería – Lógica de Programación y Estructuras de Datos I

Los microcurrículos vigentes de las asignaturas Lógica de Programación y Estructuras de Datos I de la Facultad de Ingeniería de la U.C.C., sede Medellín, presentan las temáticas tradicionales de la algoritmia imperativa, basada en la solución de problemas a partir de subprogramas.

La asignatura Lógica de Programación incluye una introducción al paradigma imperativo para el desarrollo de algoritmos, basado en el diseño descendente mediante subrutinas con arreglos de una y dos dimensiones.

Estructura de Datos realiza un repaso general de algoritmos para el tratamiento de vectores y matrices, además profundiza las estructuras lineales pila, cola y lista ligada, y en la no lineal árbol binario. La teoría del paradigma orientado a objetos no aparece en ítem alguno de los contenidos y la bibliografía recomendada tampoco induce a su tratamiento.

Tecnológico de Antioquia – Facultad de Sistemas – Lógica de Programación

En la presentación de contenidos por unidades de la asignatura Lógica de Programación, se nota un énfasis marcado en el paradigma de la programación estructurada (o paradigma imperativo), sin tratamiento alguno del concepto de objeto como instancia de clase.

La bibliografía recomendada es escasa, aunque de calidad para el estudio de técnicas de programación con diseño descendente.

Instituto Tecnológico Metropolitano – Tecnología en Sistemas de Información – Lógica y Programación

El contenido dicotómico de la asignatura Lógica y Programación es ambivalente. Presenta inicialmente las técnicas para resolver problemas con algoritmos estructurados y desemboca en programación con Visual Basic, lenguaje orientado a objetos y a eventos.

El estudio de un lenguaje visual (como alguno de los incluidos en el paquete Visual Studio de Microsoft Corporation) o de lenguajes no visuales (C++ de Borland o Java de Sun Microsystems, por ejemplo), se abordará mejor si se tiene conocimiento a priori de la teoría de objetos. Aprender a programar con un lenguaje orientado a eventos o a objetos, sin la comprensión previa de la filosofía orientada a objetos, es comparable con el aprendizaje de una segunda

lengua por simple audición, sin la observancia de los elementos gramáticos y semánticos del nuevo idioma. ▲[i]

3.2. ESPAÑA

Universidad Complutense de Madrid – Facultad de Ciencias Matemáticas – Adquisición y Tratamiento de Datos

La asignatura Adquisición y Tratamiento de Datos pretende que el alumno adquiera una visión amplia de las técnicas en las que se apoya la programación a partir del estudio inicial de las estructuras de datos básicas y métodos para su manipulación, la introducción al análisis de algoritmos, finalizando con el estudio de las técnicas de diseño de algoritmos.

El contenido de la asignatura muestra la dicotomía entre una algoritmia basada en el paradigma del análisis estructurado (inmersa en la bibliografía dada en la guía: Aho / Hopcroft / Ullman, Brassard / Bratley y Wirth), y la programación orientada a objetos con Java (tratada en los textos de Lafore, Golsling, Weiss y Eckel).

Universidad Complutense de Madrid – Facultad de Informática – Estructura de datos y de la Información

El programa detallado de la asignatura Estructura de Datos y de la Información no permite establecer si se tratan los contenidos con métodos imperativos u orientados a objetos. Sin embargo, al observar la bibliografía básica, aparecen dos textos que presentan programas orientados a objetos con el lenguaje híbrido C ++, escritos por Heileman y Langsam / Augenstein / Tenenbaum. En particular, Heileman presenta algoritmos orientados a objetos expresados en pseudocódigo, en el capítulo relacionado con árboles binarios de búsqueda.

Universidad de La Coruña – Facultad de Ingeniería – Algoritmos

El curso Algoritmos, que inicia con un capítulo sobre análisis de algoritmos mediante el cálculo de su eficiencia y tiempo de ejecución, tiene un claro enfoque estructurado. Se tratan temas complejos relacionados con grafos y técnicas de diseño de algoritmos apoyados en una bibliografía básica de avanzada bajo la autoría de Weiss, Brassard / Bratley y Manber, expertos en el diseño y análisis de algoritmos tomados de aplicaciones como la criptografía, el álgebra lineal, la teoría computacional de números, la teoría de grafos, la investigación operativa, la inteligencia artificial y otras.

La incursión a las técnicas para desarrollo de algoritmos sustentados en la teoría de objetos se presenta en el texto complementario de Heileman, pero no se explicita en el contenido del curso. ▲[i]

3.3. MÉXICO

Universidad Tecnológica de La Mixteca – Facultad de Ingeniería – Exposiciones y publicaciones

Entre la lista de los principales documentos publicados por la Universidad Tecnológica de La Mixteca, un alto porcentaje corresponde a temáticas relacionadas con la metodologías para el análisis, diseño y programación orientada a objetos. Sin embargo, el tópico relacionado con la algoritmia objetual no es tratado, corroborando de esta manera la poca información bibliográfica y en formato electrónico existente sobre el tema.

Universidad Nacional Autónoma de México – Facultad de Ingeniería – Computadoras y Programación

La asignatura Computadoras y Programación dedica aproximadamente el 35% del contenido al tema de la programación estructurada, y más o menos un 43% a las prácticas de laboratorio con el lenguaje de programación C. El porcentaje restante lo dedica a estudios sobre aspectos históricos de la computación y representación interna de la información. El temario de la orientación a objetos

en cualquiera de sus facetas no está incluido en los contenidos del curso ni en la bibliografía propuesta, razón por la cual de debe considerar un capítulo adicional donde se traten los aspectos más relevantes de dicha teoría. ▲i

3.4. ARGENTINA

Universidad Tecnológica Nacional – Facultad de Ingeniería – Algoritmos y Estructuras de Datos

Algoritmos y Estructuras de Datos presenta un interesante enfoque fundamentado en el paradigma de la programación estructurada. El curso inicia con una introducción a la programación, analiza los paradigmas de programación más representativos (incluyendo el paradigma de la orientación a objetos de manera secundaria) y finaliza con el diseño de programas modulares basados en el lenguaje funcional MuLisp y el procedimental Pascal. La bibliografía recomendada no está actualizada y tampoco tiene en cuenta las modernas tendencias inherentes al desarrollo de software reutilizable basado en clases. ▲i

4. ASPECTOS METODOLÓGICOS

La gestación de este trabajo parte de dos causales:

- La experiencia del autor como Ingeniero de Sistemas al servicio de la docencia universitaria.

- Las observaciones realizadas por algunos profesionales del área informática respecto a la inclusión de objetos en un algoritmo.

Antes de plantear estrategias metodológicas conducentes a una algoritmia objetual, se realizó el estudio de las siguientes áreas específicas relacionadas con la ingeniería informática, ordenadas cronológicamente según su aparición en el proceso académico:

Algoritmos estructurados
Programación imperativa
Programación orientada a objetos
Análisis y diseño orientado a objetos.

Este orden generó barreras para la comprensión expedita de la programación orientada a objetos, dado que el paradigma imperativo antes estudiado no inducía a éste tipo de programación. Así entonces, es relevante argüir:

Si un programa[5] estructurado es la codificación de un algoritmo estructurado, entonces un programa orientado a objetos debe ser la codificación de un algoritmo orientado a objetos.

Luego de experimentar con el análisis y el diseño orientado a objetos basado en UML y con la programación en C++ y Java, surge la propuesta aquí sustentada.

▲i

4.1. TIPO DE ESTUDIO

La Metodología para la Algoritmia Orientada a Objetos es un estudio explorativo, por cuanto recoge dos herramientas conceptuales ya existentes para la construcción de planos de software (UML y diagramas N – S) y propone, con un conocimiento sobre la teoría de objetos, solucionar problemas mediante algoritmos para futura implementación en computadora. Además, el estudio es una innovación en el área del diseño de algoritmos: rompe con antiguos esquemas de abstracción usados por los analistas para solucionar problemas con métodos procedimentales.

▲i

[5] En el ámbito de la informática existen lenguajes procedimentales y no procedimentales; los primeros implican algoritmos, pues se debe especificar el CÓMO se logra un resultado, es decir se especifica el

4.2. MÉTODO DE INVESTIGACIÓN

El método de investigación utilizado será deductivo, dado que se observan fenómenos generales – algoritmia imperativa y programación orientada a objetos – con el objeto de explicar y expandir la situación particular propuesta: método para el diseño de algoritmos con orientación a objetos.

Además la propuesta metodológica incluirá un método de investigación analítico, porque a partir de la identificación de partes que caracterizan una realidad – desarrollo de software estructurado y orientado a objetos –, se establece una relación causa-efecto entre ellas, que conduce a proponer un cambio en la forma de resolver una situación modelable con software. [i]

4.3. FUENTES Y TÉCNICAS DE RECOLECIÓN DE INFORMACIÓN

Las fuentes primarias para recabar información de utilidad están dadas por la observación y análisis directo de los paradigmas de programación imperativo y orientado a objetos. Fuentes secundarias son los libros de texto, documentos e hipertextos reseñados a lo largo del trabajo.

Las técnicas constituyen los medios empleados para la recolección de información. Los medios utilizados son las redes telemáticas –Internet– y

---

proceso para alcanzar el resultado. En los segundos se indica QUÉ se necesita, no cómo obtenerlo (v.g. SQL en bases de datos): en tales casos no tienen sentido los algoritmos orientados a los procedimientos.

entrevistas a profesionales de la ingeniería para sondear la pertinencia de la propuesta. ▲[i]

4.4. VARIABLES A CONSIDERAR EN EL DESARROLLO DE LA METODOLOGÍA

4.4.1. EN LA FASE DE DIAGNÓSTICO

- La percepción actual de los desarrolladores de software y los docentes de la ingeniería de sistemas frente a la algoritmia.

- La importancia del desarrollo de software dentro de la computación en general.

- El estado del modelamiento por objetos y de la algoritmia tradicional dentro de las universidades.

- El estado del modelamiento por objetos dentro de las empresas. ▲[i]

4.4.2. EN LA FASE DE LA FORMULACIÓN DEL MODELO

- Existencia de conceptos nuevos.

- Adaptación de otros conceptos.

- Validez de la metodología propuesta.

- Importancia del modelo dentro de la computación en general.

- La estructura del modelo desde los ángulos teórico y pedagógico.

- La pertinencia científica de la propuesta.

- La pertinencia tecnológica de la propuesta.

- La pertinencia académica de la propuesta.

- La pertinencia pedagógica y filosófica de la propuesta.

- La adaptabilidad y flexibilidad de la propuesta. ▲[i]

4.4.3. EN LA FASE DE EVALUACIÓN

- Diferencias entre las metodologías estructuradas y orientada a objetos.

- Diferencias en los productos finales derivados de los desarrollos orientados a objetos e imperativos tradicionales.

- Incidencia de la fundamentación orientada a objetos en la visión del desarrollador de aplicaciones en las fases de análisis, diseño y desarrollo de aplicaciones. ▲[i]

5. DESARROLLO DE LA PROPUESTA METODOLÓGICA

La Metodología para la Algoritmia Orientada a Objetos –MAOO, es una propuesta para la abstracción inherente a la teoría de objetos, fundamento del paradigma de programación orientado a objetos y a eventos, con el fin de aplicarla a la lógica de programación.

La aplicación de la metodología implica el uso de los diagramas N–S para escritura de algoritmos y del UML para la representación de objetos implícitos en una solución algorítmica, como objetos, clases y herencias, esencialmente.

Los diagramas N – S son claros e impactantes a nivel visual. La metodología es por completo independiente de dichos diagramas, porque si se escogiera para la exposición de éste trabajo el pseudocódigo o el diagrama de flujo, la esencia de la metodología permanecería intacta.

UML es un lenguaje de reciente aparición en el mercado; es ya un estándar de la industria del software y de toda aquella que requiera la construcción de modelos como condición previa para el diseño y posterior construcción de

prototipos. Este lenguaje fue aceptado en 1997 por el Object Management Group (OMG), como un estándar para modelado de sistemas.

La Metodología para la Algoritmia Orientada a Objetos pretende aportar elementos teóricos que permitan a los desarrolladores de software plantear algoritmos con estructuras de almacenamiento más complejas que las variables, como son los objetos (compuestos por datos y operaciones sobre ellos). La inclusión de objetos en un algoritmo, implicará también el uso de sus conceptos relacionados: clase, herencia, polimorfismo y ocultamiento de datos, entre otros. Algoritmos así concebidos evitarán cambios inadecuados de paradigma, cuando se llegue a programar con lenguajes visuales como Visual Basic (orientados a eventos sobre objetos), lenguajes híbridos como C++ (soportan los paradigmas imperativo y objetual) y puros orientados a objetos como Java. [▲i]

5.1. ELEMENTOS DE LA METODOLOGÍA PARA LA ALGORITMIA ORIENTADA A OBJETOS

La metodología incluye una serie de pasos que garantizan un producto final –algoritmo– con las siguientes características:

- Finito: todo algoritmo consta de una cabecera, un cuerpo y un pié o base que inducen a su finalización. Sin embargo, aunque contenga estos elementos, un algoritmo puede incurrir en un ciclo infinito que impida la

terminación de la tarea para la cual fue destinado. Por tanto, la finitud de un algoritmo se garantiza si en un momento específico del tiempo termina su proceso de ejecución.

Cuando el algoritmo forma parte de las operaciones de una clase –o sea, es un método–, la cabecera constará de un tipo de dato de retorno, el nombre de la clase a la cual se asocia el método, el operador **::** –denominado en C++ "operador de resolución de ámbito"–, el nombre de la clase a la cual se asocia el método, y entre paréntesis una lista de nombres de argumento separados por comas. La cabecera de un algoritmo principal constará sólo de la palabra "Inicio". El tipo de dato de retorno será uno de los siguientes: *entero*, *flotante*, *carácter*, *cadena*, *booleano*, *clase* y *sin valor*; este último tipo –conocido en lenguajes de programación como C, C ++, Java y Perl como *void*–, se utiliza para dar a entender que el algoritmo no retorna valor alguno. El tipo de datos *clase* es un tipo abstracto de datos definido por el analista, equivalente a otra clase que como tal, incluye sus propios miembros dato y operaciones. El pié del método constará tan solo de la sentencia *retorne*, con un valor opcional de retorno encerrado entre paréntesis. El cuerpo es el conjunto de instrucciones ejecutables del algoritmo. Esquemáticamente:

tipo_de_retorno [nombreClase **::**] nombreMétodo([parámetros])
//cuerpo del subalgoritmo ométodo
retorne[(valor)]

Los corchetes no forman parte integral del esquema; se utilizan para indicar que su contenido es opcional.

Ahora, si el algoritmo no está asociado a las operaciones de una clase, es decir, constituye un algoritmo principal, su esquema general es el siguiente:

Inicio
// Cuerpo del algoritmo
Fin

- Claro: las instruciones de un algoritmo no admiten ambigüedad, deben ser claras y precisas.

- Formal: el algoritmo debe utilizar, en la medida de lo posible, modelos matemáticos, ecuaciones, álgebra de Boole, etc., que ayuden a culminar la tarea para la cual está destinado.

- Eficaz: las instrucciones del algoritmo deben consumir el menor tiempo de máquina posible, sin menoscabo del objetivo del mismo. Esto equivale a decir que el contador de frecuencias y el orden de magnitud del algoritmo deben ser los ideales.

- Compacto: todo algoritmo debe ser construido a partir de tres estructuras básicas: secuencia, decisión y ciclo, tal como lo enunció Edsgar Dijkstra

hace algunas décadas. El apéndice A hace referencia a tales bloques de construcción.

- Verificable: capacidad del algoritmo para soportar procedimientos de validación y de aceptar juegos de test.

- Robusto: un algoritmo posee esta propiedad si funciona correctamente para todos los casos posibles, incluso en situaciones anormales.

- Extensible: es la facilidad que tiene el algoritmo de adaptarse a cambios en su especificación.

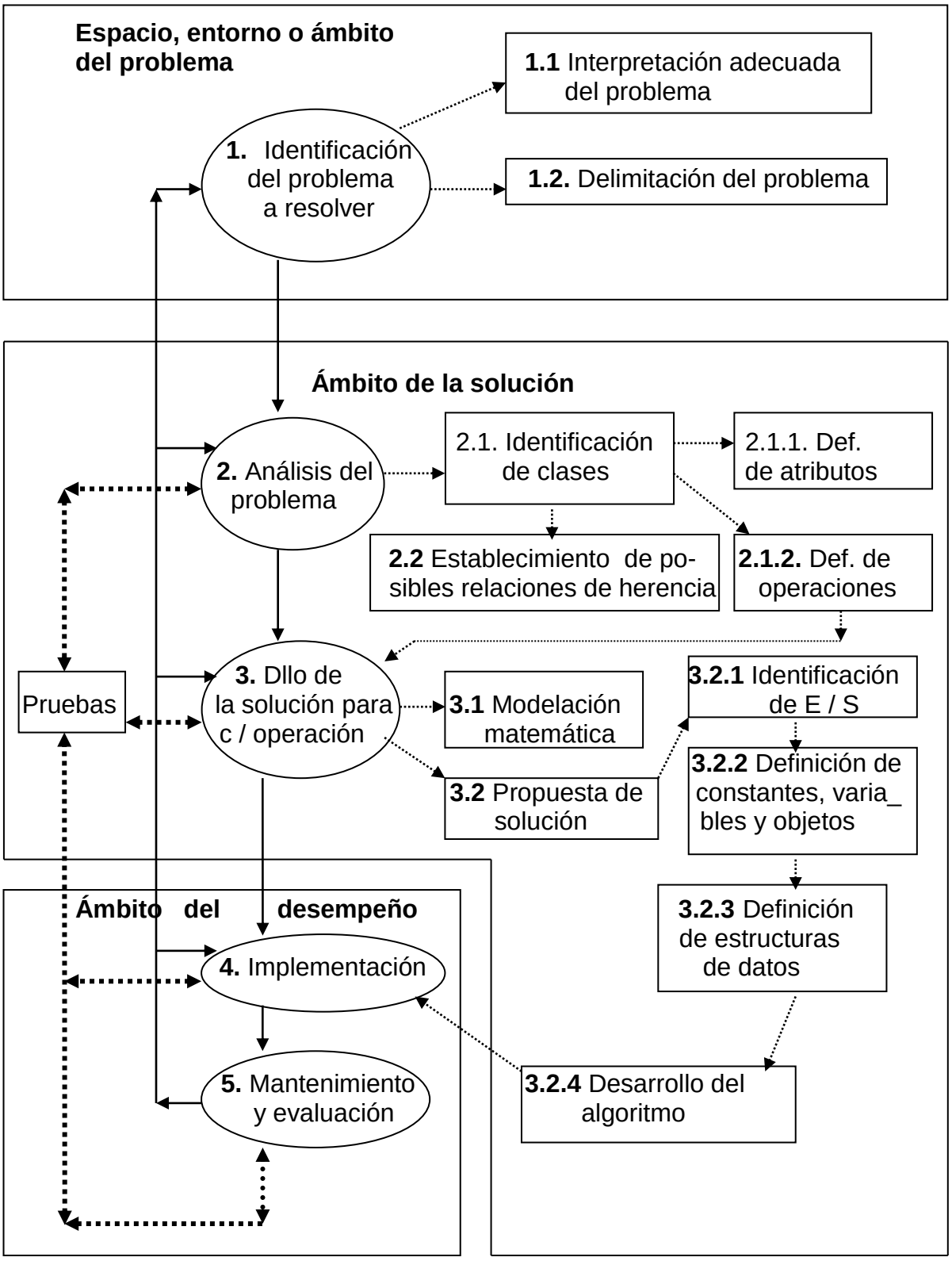
- Reutilizable: capacidad de un algoritmo para ser reutilizado en su totalidad o en parte por nuevos algoritmos. Esto se logra precisamente por la relación que se presenta entre un algoritmo y una operación de alguna clase. ▲[i]

5.1.1. ESQUEMA GENERAL

El siguiente esquema presenta todas las etapas de la metodología, incluidas en tres áreas bien definidas:

- ✓ Espacio, entorno o ámbito del problema.
- ✓ Ámbito de la solución.
- ✓ Ámbito del desempeño.

La Metodología para la Algoritmia Orientada a Objetos hará énfasis especial en el ámbito de la solución, sin descuidar la etapa previa del espacio del problema. El ámbito del desempeño implica codificación del algoritmo en algún lenguaje de programación, aspecto irrelevante para la propuesta.


Espacio, entorno o ámbito del problema
1.1 Interpretación adecuada del problema
1. Identificación del problema a resolver
1.2. Delimitación del problema
Ámbito de la solución
2. Análisis del problema
2.1. Identificación de clases
2.1.1. Def. de atributos
2.2 Establecimiento de po- sibles relaciones de herencia
2.1.2. Def. de operaciones
Pruebas
3. Dllo de la solución para c / operación
3.1 Modelación matemática
3.2.1 Identificación de E / S
3.2 Propuesta de solución
3.2.2 Definición de constantes, varia_ bles y objetos
3.2.3 Definición de estructuras de datos
Ámbito del desempeño
4. Implementación
5. Mantenimiento y evaluación
3.2.4 Desarrollo del algoritmo

▲ i

5.1.1.1. IDENTIFICACIÓN DEL PROBLEMA A RESOLVER

La descripción escrita de una situación problemática, surge por la necesidad de darle una solución lo más óptima posible. Dicha descripción la propone un usuario final (son usuarios finales, por ejemplo, el personal administrativo y operativo de una organización empresarial), un científico (la simulación de modelos matemáticos en computadora es un proceso habitual en las ciencias aplicadas y exactas) o un grupo de personas con intereses comunes (comité, asociación, etc.), quienes establecen como clientes los requisitos del futuro sistema.

La identificación del problema a resolver incluye los siguientes procesos:

- Interpretación adecuada del problema

Una interpretación errónea de la situación problemática, conducirá a una solución carente de validez.

La interpretación correcta de un enunciado incluye elementos semánticos, análisis de proposiciones y demás factores relacionados con el entendimiento humano. Una interpretación del problema a partir de los *casos de uso* (concepto ingenioso del análisis y diseño orientado a objetos, propuesto por Ivar Jacobson), sería una buena alternativa para no desviarse de la situación a resolver.

- Delimitación del problema

El analista debe comprender el dominio del problema, es decir, delimitar su contorno significativo, para no anticipar o sobrepasar los límites de la situación planteada.

La delimitación del problema forma parte también de la interpretación del mismo, son procesos que se solapan. ▲[i]

5.1.1.2. ANÁLISIS DEL PROBLEMA

Luego de una correcta interpretación y delimitación del problema, se pasa a la etapa de análisis, en la cual el ingeniero de software amplía el conocimiento de la situación antes de proceder a plantear una solución. El análisis es un proceso cognoscitivo difícil de describir; si es inadecuado surgirán errores que se propagarán a las etapas siguientes, con el consecuente despilfarro del recurso humano y logística de software y hardware requerida. Los errores se pueden evitar aplicando paralelamente al análisis pruebas de verificación.

- Identificación de clases

El análisis del problema incluye la identificación de clases. Una clase es una abstracción de la realidad que el analista selecciona para, a futuro, modelarla con software.

El concepto de clase lo definen Rumbaugh, Jacobson y Booch[6], así:

> ***Clase***. Una clase representa un concepto discreto dentro de la aplicación que se está modelando: una cosa física (tal como un aeroplano), una cosa negocios (tal como un pedido), una cosa lógica (tal como un horario de difusión), una cosa de una aplicación (tal como un botón de cancelar), una cosa del computador (tal como una tabla hash), o una cosa del comportamiento (tal como una tarea). Una clase es el descriptor de un conjunto de objetos con una estructura, comportamiento, y relaciones similares. Todos los atributos y operaciones están unidos a clases o a otros clasificadores. Las clases son los focos alrededor de los cuales se organizan los sistemas orientados a objetos.

Un conocimiento amplio y certero del problema permitirá reconocer las clases requeridas para desarrollar la solución. Una identificación errónea de clases, así como la omisión de alguna necesaria para el proceso de desarrollo del software, acarreará problemas a lo largo de las demás etapas del ámbito de la solución y el desempeño.

- Definición de atributos y operaciones

Toda clase está formada por unos atributos y unas operaciones, que el analista también debe definir de acuerdo al entorno del problema.

Un atributo es una propiedad de una clase identificada con un nombre, que describe un rango de valores que pueden tomar las instancias de la propiedad.

[6] J. Rumbaugh, I. Jacobson, G. Booch, El Lenguaje Unificado de Modelado. Manual de referencia, Pearson Educación, Madrid, 2000. Traducción de Salvador Sánchez et al, p. 38.

Un atributo representa alguna propiedad del elemento que se está modelando, compartida por todos los elementos de esa clase. Por ejemplo, se podrían modelar los clientes de manera que cada uno tenga una cédula, un nombre, una dirección y un teléfono; toda pared tiene una altura, una anchura y un grosor. Un atributo es, por tanto, una abstracción de un tipo de dato o estado que puede incluir un objeto de la clase. En un momento dado, un objeto de una clase tendrá valores específicos para cada uno de los atributos de su clase.

Una operación es la implementación de un servicio que puede ser requerido a cualquier objeto de la clase para que muestre un comportamiento. En otras palabras, una operación es una abstracción de algo que se puede hacer a un objeto y que es compartido por todos los objetos de la clase. Una clase puede tener cualquier número de operaciones o ninguna. Por ejemplo, en una biblioteca de ventanas como la del paquete *awt* de Java, todos los objetos de la clase *Rectangle* pueden ser movidos, redimensionados o consultados sobre sus propiedades. A menudo (pero no siempre), la invocación de una operación sobre un objeto cambia los datos o el estado del objeto. Las operaciones son precisamente los algoritmos que se deben plantear para el hallazgo de la solución. Se utilizará la sintaxis de UML para representar clases.

- Establecimiento de posibles relaciones de herencia

Al identificar clases se deben establecer las posibles relaciones de herencia existentes, sea entre las clases propias del problema a solucionar o entre otras

clases reutilizables dispuestas en librerías. Los mecanismos de herencia permiten la reutilización de clases antes definidas, enlazando una clase con otra para formar jerarquías, las cuales contienen *clases derivadas* de una *clase base* o *superclase*. ▲[i]

5.1.1.3. DESARROLLO DE LA SOLUCIÓN PARA CADA OPERACIÓN (DISEÑO)

El paso siguiente al análisis del problema es el desarrollo de la solución. A veces, debido a la extensión y complejidad de la situación problemática, se puede ir describiendo la solución en lenguaje natural, a modo de una secuencia de pasos generales que se van detallando a medida que se avanza.

El desarrollo de la solución implica el uso de diferentes técnicas para diseño de algoritmos: dividir para vencer, programación dinámica, algoritmos voraces, algoritmos heurísticos, etc.

Según el diagrama del esquema general, el desarrollo de la solución incluye dos fases:

- Modelación matemática

En lo posible, se deben establecer modelos matemáticos para representar las facetas de la solución propuesta; ello implica el uso de fórmulas, ecuaciones

constantes universales, recurrencias, simulaciones, técnicas de procesamiento como derivadas, etc.

Ciertos algoritmos pueden no requerir de elementos matemáticos, debido seguro a su trivialidad o ámbito de problema ínfimo.

- Propuesta de solución

Esta es la fase primordial del esquema, porque en ella se plantea la solución mediante un algoritmo detallado y documentado.

La propuesta de solución incluye una secuencia de pasos:

- Identificación de E/S (entrada de datos, salida de información)

  Se deben determinar las entradas de datos requeridas para solucionar el problema, provenientes de distintos dispositivos: teclado, memoria interna (RAM) o memoria externa (disquetes, discos duros, CD ROM, etc.). Asimismo, las salidas deben ser identificadas para no desviarse del propósito del algoritmo: encontrar solución a un problema específico.

  La entrada puede ser nula o múltiple y la salida se debe garantizar, es decir, nunca puede ser nula. Una salida puede ser no evidente, como ocurre cuando sólo se modifica un dato al interior de una estructura de datos.

- Definición de constantes, variables y objetos

La definición de clases y el análisis adecuado en el ámbito de la solución, permitirán establecer las constantes, variables y objetos necesarios.

Las constantes son valores que no cambian durante la ejecución de un programa (en este contexto “programa” se puede equiparar con “algoritmo”), y deben tener un tipo de dato asociado: numérico entero, numérico de punto flotante, carácter, cadena o booleano.

Las variables son campos de memoria cuyo contenido puede variar durante la ejecución de un programa. Como las constantes, también tienen asociado un tipo de datos.

En las ciencias exactas se usan nombres simbólicos para referenciar cantidades y constantes universales como Π , la aceleración de la gravedad, el número de Avogadro, etc. Por ejemplo, la fórmula

$$\mathbf{a} = \Pi * \mathbf{r}^2,$$

se usa para calcular el área **a** de un círculo de radio **r**. Los nombres simbólicos **a** y **r** reciben el nombre de variable, y el símbolo Π recibe el apelativo de constante, pues su valor es inmutable 3.1415... (los puntos suspensivos denotan guarismo irracional).

Un objeto es una entidad discreta, con límite bien definido y con identidad, que encapsula el estado y el comportamiento; un objeto es una instancia de una clase, y como las variables, representan una posición de memoria en la computadora.

- Definición de estructuras de datos

La propuesta de solución debe incluir a las estructuras de datos para almacenar los datos del problema.

Una estructura de datos es una clase de datos que se puede caracterizar por su organización y operaciones definidas sobre ella. Una variable de hecho es una estructura de datos donde se almacena sólo un dato a la vez; por lo general, en los algoritmos se requiere del almacenamiento de múltiples datos de igual tipo al mismo tiempo; para lograrlo, se utilizan estructuras de datos simples (cadena, registro), lineales (arreglo, pila, cola, lista ligada), binarias no lineales (árbol binario, árbol binario de búsqueda) y N-arias no lineales (grafos, árboles generales, árboles de búsqueda multicaminos, árboles B, árboles B*, árboles B^+ y trie).

- Desarrollo del algoritmo

  Finalizadas las etapas anteriores se pasa al desarrollo del algoritmo, haciendo uso de alguna representación válida (pseudocódigo, diagrama de flujo, diagrama N–S). Los diagramas N–S se recomiendan por su claridad y laconismo. ▲[i]

5.1.1.4. IMPLEMENTACIÓN

Esta etapa consiste en codificar el algoritmo en un lenguaje de programación que soporte el paradigma de la orientación a objetos. Este proceso resulta casi completamente mecánico, y debe estar ceñido a las reglas sintácticas y semánticas del lenguaje, a la inclusión de bibliotecas de clases reutilizables y a la conservación de ciertos criterios de estilo que garanticen la claridad de los programas. ▲[i]

5.1.1.5. MANTENIMIENTO Y EVALUACIÓN

La puesta en marcha de una aplicación de software en una organización cualquiera, generará necesidades de mantenimiento constante al software, debido a los cambios organizacionales y a los requerimientos de evaluación para verificar su desempeño.

Observación importante:

El proceso de desarrollo de software orientado a objetos debe ser iterativo en lo relacionado con las pruebas.

Una prueba en los ámbitos de la solución o del desempeño, variará de acuerdo a la etapa donde se encuentre el proceso. Para clarificar, véanse los siguientes ejemplos:

- Pruebas en etapa de análisis:

  - Verificar las clases establecidas, en búsqueda de aquellas redundantes o irrelevantes que se puedan desechar, o de otras importantes para el modelo aún no identificadas.

  - Revisar las jerarquías de clase para efectos de reutilización y ocultamiento de datos.

- Pruebas en etapa de desarrollo de la solución para cada operación:

  - Verificar puntos críticos del algoritmo, es decir, garantizar que el algoritmo produce resultados veraces con una cantidad representativa de datos válidos.

- Comprobar la consistencia del modelo matemático con la realidad que simula.

- Calcular la complejidad (notación O grande) y el tiempo de ejecución del algoritmo.

➢ Pruebas en etapa de implementación:

- Revisar interfaces gráficas de usuario para verificar su validez y amigabilidad.

- Verificar correspondencia entre el algoritmo planteado y el programa desarrollado.

➢ Pruebas en etapa de mantenimiento y evaluación:

- Revisar comportamiento del producto (software) en paralelo con los cambios organizacionales.

- Medir la respuesta del programa ante eventuales cambios de plataforma hardware o software. ▲[i]

5.1.2. INDICADORES DE EVALUACIÓN Y RESULTADOS ASOCIADOS A LA METODOLOGÍA

Los indicadores de evaluación y resultados especificados a continuación, serán solidificados o determinados en un período posterior a la aplicación de la metodología con los estudiantes. Dichos indicadores constituyen unidades de medición que permitirán identificar las bondades de la metodología en los procesos docente-educativos anejos a la algoritmia, así como las posibles debilidades o inconvenientes que se puedan presentar, con el objetivo de aplicar a futuro los correctivos necesarios.

Algunos indicadores son difíciles de medir, como los indicadores de beneficio; otros por el contrario, se determinan fácilmente.

Indicadores de beneficio

- Grado de aprendizaje alcanzado por el estudiante en lo referente a técnicas de modelamiento orientado a objetos basadas en algoritmos.

- Número de estudiantes que obtienen logros positivos en su proceso de aprendizaje al solucionar problemas aplicando la metodología.

Indicadores de calidad

- Grado de correspondencia existente entre los algoritmos planteados a partir de la metodología propuesta y los conceptos subyacentes en las tecnologías orientadas a objetos.

Indicadores de cobertura

- Número de instituciones educativas que adaptan la metodología al currículo, es decir, cantidad de instituciones que acogen la propuesta metodológica para el desarrollo de algoritmos sujetos al paradigma de la orientación a objetos, en asignaturas como Lógica de Programación, Algoritmos, Estructuras de Datos y otras relacionadas.

- Cantidad de profesionales, docentes y no docentes, que aceptan y aplican la metodología.

Indicadores de resultados

- Cantidad de algoritmos probados exitosamente, diseñados a partir de estrategias planteadas en la MAOO.

- Número de programas desarrollados con un diseño de algoritmos basado en la MAOO.

- Porcentaje de estudiantes reprobados en signaturas donde la MAOO sea el principal objeto de conocimiento. ▲[i]

6. CONCLUSIONES

La culminación del presente trabajo monográfico conduce a una serie de conclusiones, unas relacionadas con aspectos específicos del diseño de algoritmos, otras asociadas a los procesos de enseñanza –aprendizaje.

i) Los estudiantes de Ingeniería de Sistemas y programas académicos afines, adquieren elementos de raciocinio y lógica propios de las metodologías estructuradas para el desarrollo de software, pero en etapas de implementación y mantenimiento de los programas desarrollados se enfrentan a otro paradigma, el orientado a objetos, generando secuelas negativas en el proceso de enseñanza–aprendizaje, reflejadas a futuro en la calidad del profesional egresado.

ii) En general, el eje temático “Algoritmia orientada a objetos” no se incluye, hasta el momento, en los contenidos de asignaturas como Algoritmos y Estructuras de Datos, impartidas en facultades que ofrezcan programas de ingeniería. Existen algunos logros para incluir la temática en las asignaturas algoritmos I y algoritmos II, inmersas en el plan de estudios vigente de Ingeniería de Sistemas en la Universidad de Antioquia.

En el capítulo 7 de la fuente bibliográfica HEILEMAN, titulado “Árboles binarios de búsqueda”, se desarrollan algoritmos Orientados a objetos presentados en pseudocódigo. Estos antecedentes verifican la

pertenencia del cambio de paradigma que se debe dar en cuanto al análisis y elaboración de algoritmos.

iii) Los conceptos de la orientación a objetos para el desarrollo de software se imparten regularmente en asignaturas asociadas a la programación, el análisis y diseño de sistemas y las bases de datos, sin una fundamentación previa que permita la apropiación de conceptos en dichas áreas de una forma más expedita.

iv) Las entidades educativas, centros de investigación y empresas productoras de software, deben adaptarse a los continuos cambios tecnológicos que les garanticen altos niveles de competitividad. Por ejemplo, en una estructura curricular de un programa académico como Ingeniería de Sistemas u otro afín (que no incluya asignaturas donde se estudien los beneficios del desarrollo de software a partir de componentes reutilizables, donde se continúe manejando el ciclo de desarrollo de software lineal o en cascada como dechado a seguir, donde el estudio de los modelos de proceso de software no incluya los modelos en espiral y de ensamblaje de componentes, entre otras flaquezas académicas), se deben establecer mecanismos de mejora propuestos por los comités curriculares, consejos académicos y otros organismos que velen por la calidad académica.

v) Los indicadores de gestión establecidos, solo se podrán medir en un periodo posterior a la implementación de la metodología. Se espera que dichos indicadores permitan identificar sus posibles debilidades, en cuyo caso, se tomarán los correctivos pertinentes, en búsqueda de una continua mejora de la metodología. Los valores finales de los indicadores también servirán para determinar sus bondades y mostrarán un nuevo viraje (el orientado a objetos) para el fascinante mundo del diseño de algoritmos.

vi) La metodología propuesta no “destruye” los elementos propios de las metodologías estructuradas para desarrollo de software (modularidad, diseño descendente, sentencias de control para toma de decisiones y ciclos, paso de parámetros por valor y referencia, etc.). Por el contrario, todos estos conceptos prevalecen en la propuesta MAOO, son absorbidos por ella.

Se podría afirmar que la metodología para la algoritmia arientada a objetos, es un superconjunto de la convencional metodología para la algoritmia estructurada, así como el lenguaje de programación C++ es un superconjunto del lenguaje C.

vii) Este trabajo es una propuesta que se debe analizar en ambientes académicos a la luz de los recientes adelantos o cambios de los lenguajes de programación que se imponen en los ámbitos científico y

empresarial. El autor se compromete a exponer la propuesta en las instituciones que a bien lo permitan. ▲[i]

A

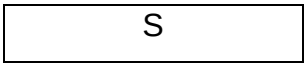
LOS DIAGRAMAS N–S (NASSI–SCHEIDERMAN)

Edsgar Dijkstra afirmó hace aproximadamente cuatro décadas que cualquier problema a modelar mediante computadora, se podía resolver en base a tres estructuras lógicas fundamentales: secuencia, decisión y ciclo.

La estructura secuencia implica linealidad y adyacencia instructiva; la decisión implica la ejecución de ciertas instrucciones de acuerdo al valor de verdad de una expresión lógica; la estructura ciclo indica repetitividad finita de un grupo de instrucciones.

A partir de los postulados de Dijkstra, Nassi y Shneiderman idean una simbología clara y visual, basada en rectángulos –por eso también se conoce como diagramación rectangular–, que se esboza a continuación.

1. Estructura secuencia

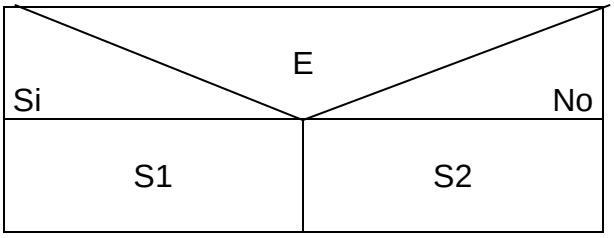


Donde S: secuencia de instrucciones.
Las instrucciones pueden ser de entrada, salida, asignación, decisión o ciclo.

2. Estructura de decisión

Se diferencian dos tipos:

2.1. Selector dual (if)



Donde

E: Expresión lógica o condición

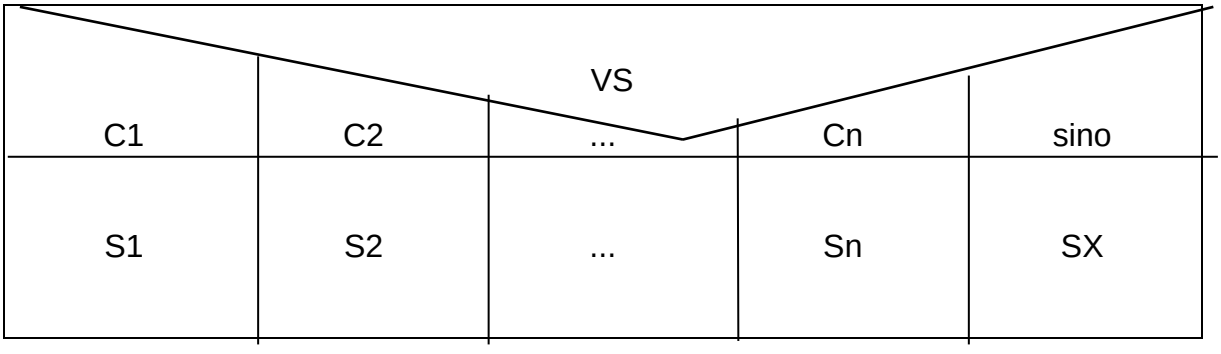
S1: Estructura secuencia

S2: Estructura secuencia

Funcionamiento: Si la expresión E es verdadera se ejecuta la secuencia S1, de lo contrario se ejecuta la secuencia S2.

Recomendación: S1 nunca debe coincidir con la secuencia nula, pero S2 sí.

2.2. Selector múltiple (switch)



Donde:

VS: Variable selectora. Puede ser de tipo entero, carácter, enumerado o

subrango.

Ci: Lista de constantes i

$i \in [i, n]$

Toda Ci debe coincidir con el tipo de VS

S: Estructura secuencia J

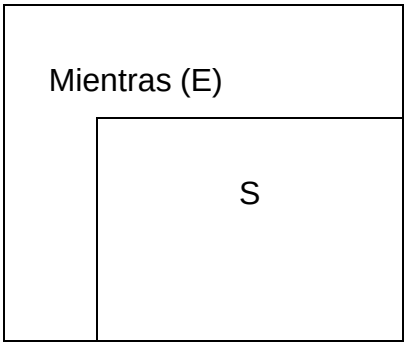
$J \in [i,n] \cup [X]$

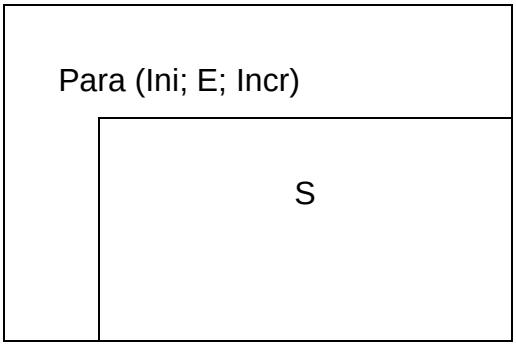
Funcionamiento: Si la variable VS coincide con la lista de constantes Ci se ejecuta la secuencia Si; de lo contrario pasa a efectuarse la secuencia SX, de existir, pues es de carácter opcional.

3. Estructura ciclo

Se presentan dos tipos:

3.1. Ciclos MIENTRAS y PARA





donde:

S: Estructura secuencia

E: Expresión lógica (verdadera o falsa)

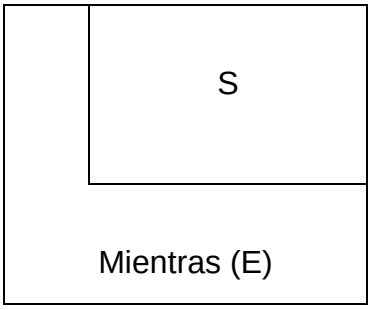
Ini: Inicialización de una o más variables

Incr: Incremento (positivo o negativo) de las variables asociadas a Ini.

Funcionamiento ciclo MIENTRAS. Si la expresión E es verdadera, se ejecuta reiteradamente la secuencia S; en caso contrario el control pasa a la siguiente instrucción después del MIENTRAS.
Dentro de S debe existir al menos una instrucción que modifique a la expresión E para evitar bucle infinito.

Funcionamiento bucle PARA. Se ejecuta la asignación Ini (puede ser múltiple) y se evalúa la expresión E; si ésta posee el valor verdad verdadero se ejecuta el cuerpo del ciclo (secuencia S) y se procede a ejecutar el incremento Incr. El proceso se repite hasta que la expresión E se hace falsa, lo cual ocasiona interrupción del ciclo.

3.2. Ciclo HACER / MIENTRAS



S: Estructura secuencia

E: Expresión booleana

FUNCIONAMIENTO DEL CICLO HACER / MIENTRAS

Funciona exactamente igual al bucle MIENTRAS, solo que la expresión E se evalúa en distinto punto. Se concluye entonces que el cuerpo de un ciclo MIENTRAS se ejecuta cero o más veces, pero el del ciclo HACER / MIENTRAS siempre se ha de repetir al menos una vez.

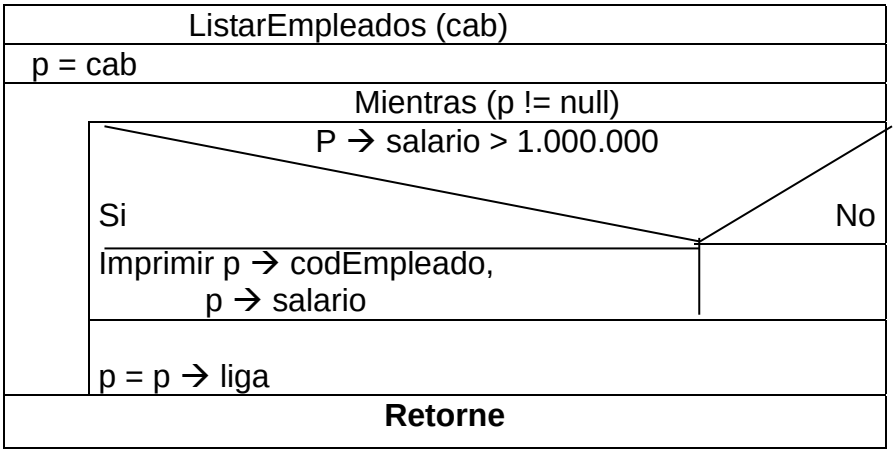
Nota. Los diagramas N-S son claros y visuales (implican bloques rectangulares que globalmente son absorbidos por la estructura secuencia).

Veamos el siguiente ejemplo que visualiza los datos de un conjunto de empleados con salario superior a $1.000.000. La configuración del nodo (donde se almacenan datos ficticios para un empleado) es la siguiente:

nodo

codEmpleado	Salario	Liga

donde **liga** constituye la dirección de memoria para almacenar los datos del siguiente empleado; los demás datos no deben requerir explicación.



Los diagramas N-S son simples, visuales y de gran utilidad para expresar la algoritmia de un problema.

▲ i

B

EL LENGUAJE UNIFICADO DE MODELADO (UML)*

UML (Unified Modeling Language) es un lenguaje gráfico para visualizar, especificar, construir y documentar los artefactos de un sistema con gran cantidad de software. Los artefactos se refieren esencialmente a los requisitos, la arquitectura, el diseño, el código fuente, la planificación de proyectos, las pruebas, los prototipos y las versiones.

UML ha sido utilizado de forma efectiva en dominios tales como los sistemas de información de empresas, bancos y servicios financieros / industrias aeroespaciales, comercio, electrónica médica, ámbito científico y servicios distribuidos basados en la web; incluso es tan expresivo, que se puede utilizar para modelar sistemas que no son software, tales como flujos de trabajo (workflows) en el sistema jurídico, estructura y comportamiento de un sistema de vigilancia médica de un enfermo y el diseño de hardware.

* Resumen adaptado del texto "El Lenguaje Unificado de Modelado", Grady Booch / James Rumbaugh / Ivar Jacobson, Rational Software Corporation, traducción de José Saenz Martínez, Addison Wesley, Madrid, 1999.

La importancia del modelado es radical. La construcción de una caseta para un animal doméstico, de una morada humana o de un vehículo de transporte, requiere de un modelaje que se expresa en planos (en plural). De igual manera, el diseño de software requiere de una serie de planos, casi independientes entre si, pero enlazados lógicamente de alguna manera. Por ejemplo, en la construcción de un edificio, los planos de fontanería, electricidad y división arquitectónica, aunque son distintos en si mismos, deben compactar lógicamente para lograr el edificio ideal.

UML incluye nueve tipos de diagramas para expresar los planos de un sistema con gran cantidad de software: diagramas de clases, de objetos, de casos de uso, de secuencia, de colaboración, de estados, de actividades, de componentes y de despliegue.

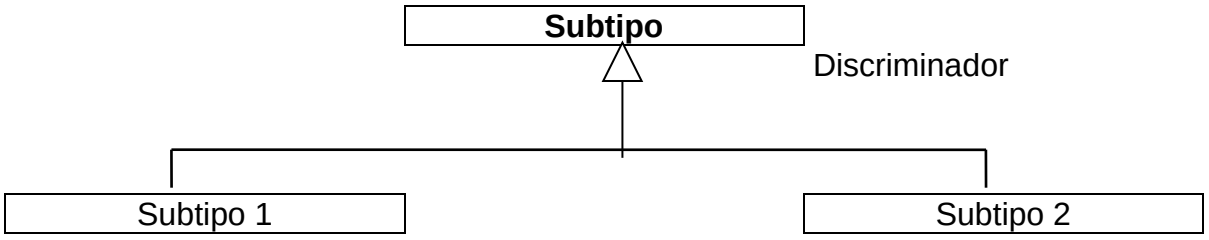
La simbología esencial para construir los anteriores diagramas se resumen a continuación.

Clase

Nombre De la clase

Nombre de la clase
Atributo: Tipo = ValorInicial
Operación (lista argumentos): tipo de devolución

Generalización

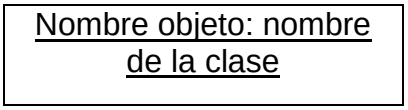


Restricción
(Descripción de la condición)

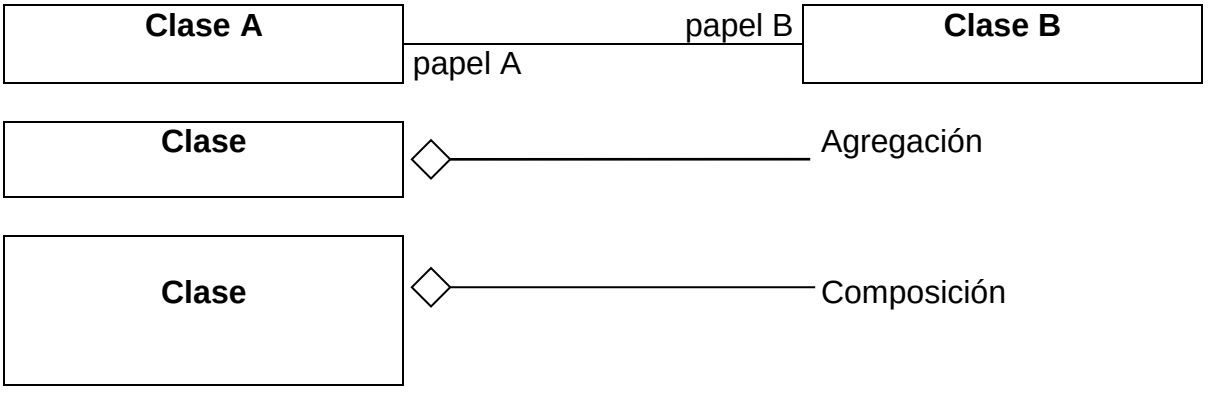
Estereotipo
<<nombre del estereotipo>

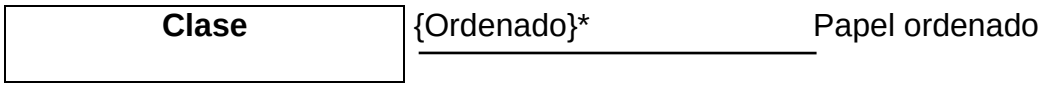
Nota

Objeto

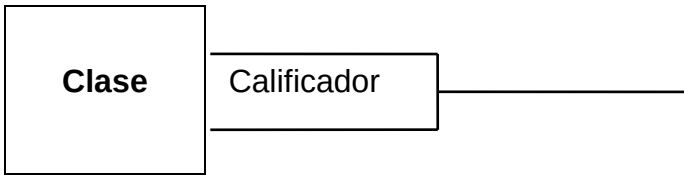


Asociación

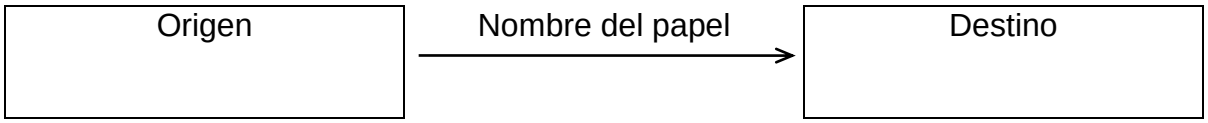




Asociación Calificada



Navegabilidad



Dependencia

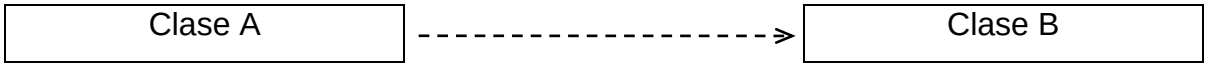
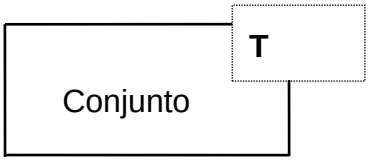


Diagrama de clases: clase parametrizada

Clase de plantilla



Clase de asociación

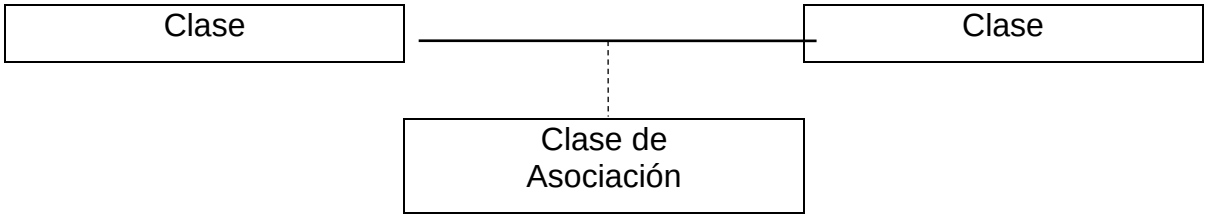


Diagrama de actividad

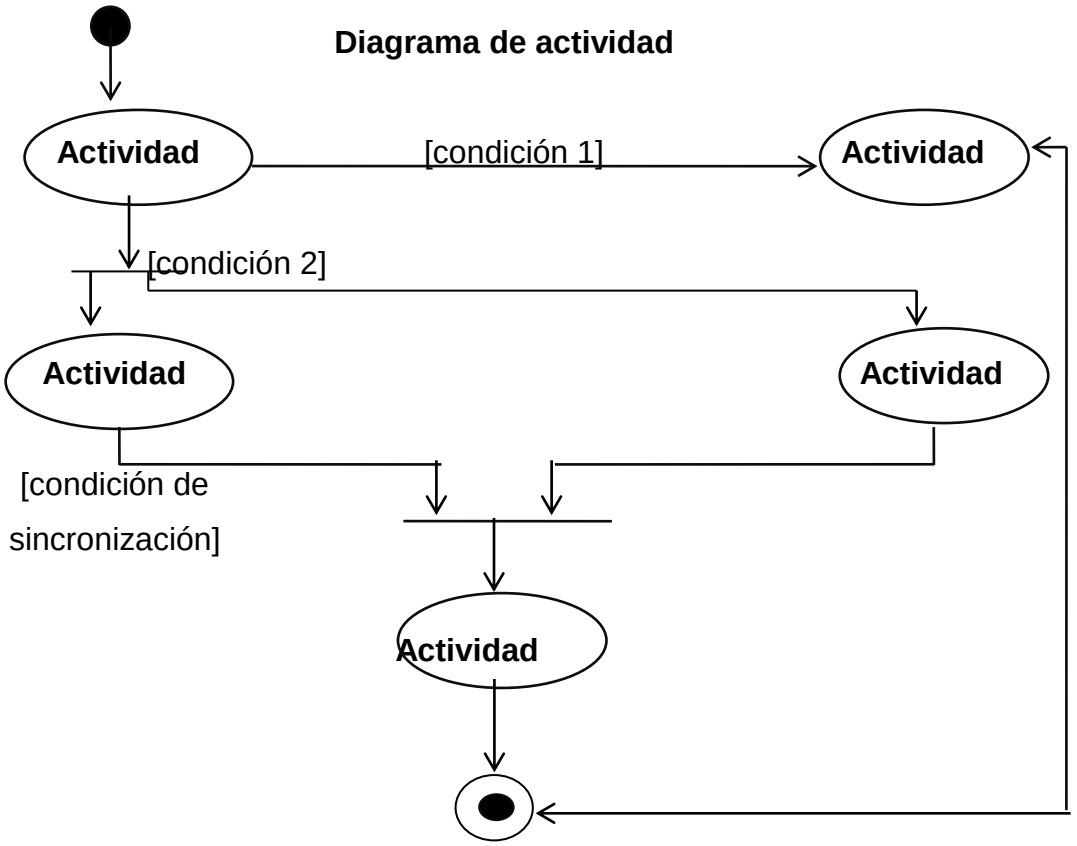


Diagrama de paquetes

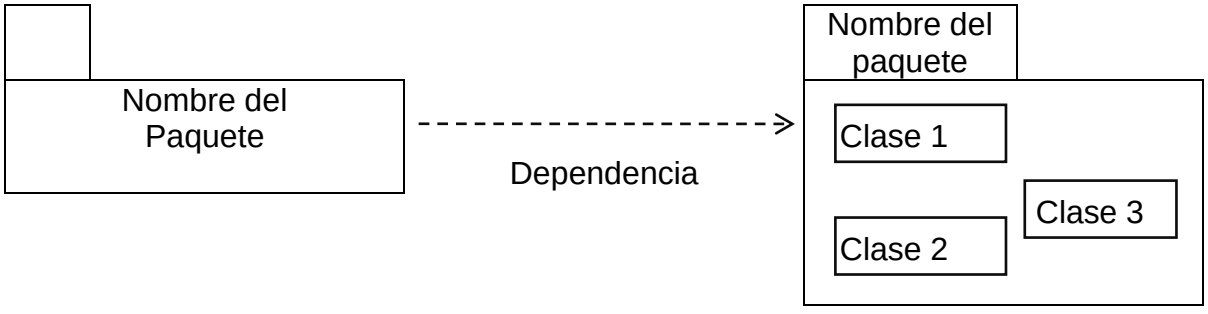


Diagrama de estados

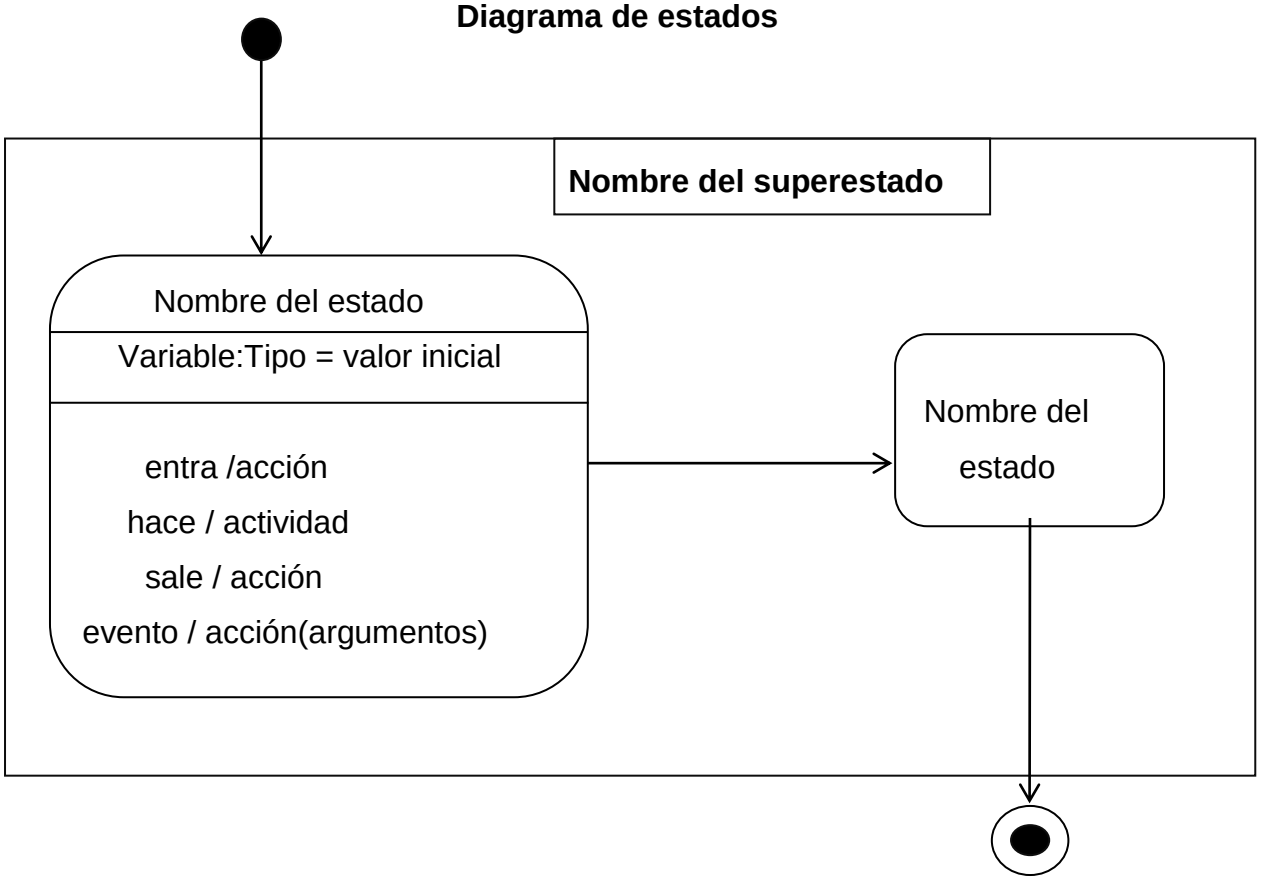
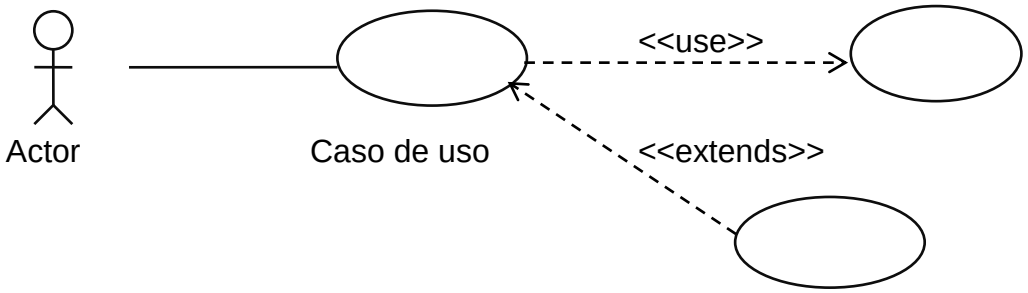


Diagrama de caso de uso



El futuro de UML es promisorio en un mercado de software basado en componentes reutilizables. [▲i]

C

PARADIGMAS DE PROGRAMACIÓN

El resumen presentado a continuación fue extractado del sitio http://www.guiafe.com.ar/sitioaedd/pagina.php3?pagina=guiasdeestudio.php3&descripcion=Guías%20de%20Estudio ,
elaborado para el programa Ingeniería en Sistemas de Información de la Universidad Tecnológica Nacional, Facultad Regional Santa Fe, República Argentina.

Un **paradigma de programación** es una colección de patrones conceptuales que modelan la forma de razonar sobre problemas, de formular algoritmos y, a la larga, de estructurar programas. A veces, un lenguaje contiene o soporta los elementos de un determinado paradigma; por ejemplo, Pascal es un lenguaje diseñado según el paradigma imperativo. Por supuesto, suele haber más de un lenguaje basado en un paradigma dado. También existen lenguajes que combinan elementos de varios paradigmas. De todas formas, y aunque en la práctica muchas veces se identifican, conviene distinguir ambos conceptos, paradigma y lenguaje de programación.

Los dos paradigmas más extendidos y estudiados desde un punto de vista algorítmico, quizá sean el imperativo y el funcional. Veamos los fundamentos de ambos, así como algunas ideas sobre otros paradigmas.

- **Paradigma funcional**

El **paradigma funcional** se basa en el concepto matemático de función. Una **función** es una regla de correspondencia que asocia a cada elemento de un conjunto origen un elemento de un conjunto destino. Los conjuntos origen y destino suelen llamarse, respectivamente, **dominio** y **rango** de la función.

Dada una función **f** y un elemento **x** de su dominio, la notación **f(x)** representa el elemento *z* del rango que **f** asocia a **x**, es decir, **f(x) = z**. La expresión **f(x)** se denomina **aplicación de función**.

Una función se especifica dando su nombre, dominio, rango y regla de correspondencia. Una forma de declarar la regla es por **enumeración**: se da una colección de parejas formadas por un elemento del dominio y su elemento correspondiente en el rango.

Por ejemplo, sea una función que determine si un día de la semana es

laborable (ignorando las fiestas que eventualmente pudiera haber). La función puede llamarse **EsLaborable**. El conjunto de dominio está formado por los siete días de la semana, y el conjunto de rango, por los valores de verdad Cierto y Falso. La regla de correspondencia queda definida con los pares:

(Lunes,Cierto)

(Martes,Cierto)

(Miércoles,Cierto)

(Jueves,Cierto)

(Viernes,Cierto)

(Sábado,Cierto)

(Domingo,Falso)

Sin embargo, la especificación por enumeración sólo sirve para aquellas funciones cuyo dominio tiene un número finito (y no muy grande de elementos). En caso contrario, es más útil otra manera de declarar, denominada **por comprensión**. Una descripción por comprensión da una regla abstracta que describe cómo calcular, para un elemento genérico **x** del dominio, su correspondiente elemento **z** del rango.

Por ejemplo, sea la función de incrementar un número entero en la unidad. Tanto el dominio como el rango son el conjunto de enteros. La regla puede

describirse como:

Incrementar(x) = x+1

De forma que:

Incrementar(4) = 4+1 = 5

Obsérvese en este ejemplo que la aplicación de **Incrementar(4)** se reescribe mediante la expresión escrita en la definición de Incrementar, sustituyendo el elemento genérico *x* por el elemento concreto 4 del dominio al que la función se aplica. A su vez, la expresión resultante es otra aplicación (esta vez, de la función de suma) que produce una tercera expresión, el entero 5. Las tres expresiones son iguales, salvo que la última es más sencilla, la razón por la que se dice que es el **valor** o resultado de **evaluar** las expresiones previas.

La definición dada de función puede generalizarse, de manera que una función puede tener **N** conjuntos de dominio, siendo **N>1**, en cuyo caso transforma una colección de *N* elementos, cada uno perteneciente a un conjunto de dominio distinto, en un elemento del rango.

Se define la **aridad** de una función como el número de dominios sobre los que la función está definida. Si su aridad vale 1, 2, ..., N, se dice que la función es **unaria, binaria, ..., N-aria**, respectivamente.

Veamos un ejemplo algo más complicado en el que se quiere hallar el máximo de tres enteros. Para encontrar la solución, puede usarse la función que halla el mayor de dos enteros. Llamaremos, respectivamente, **Max3** y **Max** a ambas funciones.

Dado que se dispone de la función Max, un modo de hallar el máximo de tres enteros es el siguiente: primero se calcula el máximo de dos de ellos, y después, el máximo entre éste y el entero restante. Es decir:

Max3 (x,y,z) = Max (x, Max (y,z))

Ahora puede aplicarse la función; por ejemplo:

Max3 (4,9,1) = Max (4, Max (9,1)) = Max (4,9) = 9

El ejemplo ilustra cómo se combinan las funciones para hacer cálculos algo complicados: una función se aplica a unos valores y su resultado es usado por otra aplicación de la función. Este anidamiento de aplicaciones de función se llama **composición de funciones**. En el ejemplo, la función **Max3** se realiza componiendo dos veces la función **Max**.

En resumen, la forma básica en que se resuelven problemas con el paradigma funcional consiste en diseñar primero una función y después evaluar expresiones funcionales, en las que la función se aplica a unos valores iniciales para obtener un valor final. La función suele diseñarse como la composición de otras funciones. A su vez, la evaluación consiste en reescribir unas expresiones en otras usando las definiciones de función.

La definición del paradigma funcional se completa con otras construcciones adicionales (tipos de datos, expresión condicional, recursión, etc).

- **Paradigma imperativo**

El **paradigma imperativo** no se inspira en las matemáticas, sino en el modelo de máquina más extendido, el llamado **modelo de von Neumann**. Una máquina de von Nueumann está formada por un procesador, una memoria y otros elementos auxiliares. La máquina funciona de manera cíclica: su procesador realiza una operación, luego otra y así siempre; es decir, realiza una serie de operaciones **secuencialmente**, una tras otra. La computadora tiene un juego de operaciones limitado, pero lo suficientemente versátil como para realizar cualquier cálculo automatizable.

La memoria de la computadora permite tener almacenados, en celdas

numeradas consecutivamente y en forma binaria, tanto las operaciones como los datos por manipular. El orden de realización de las operaciones viene dado por su disposición en la memoria (salvo que alguna operación indique lo contrario): tras realizar una operación, se continúa por la situada en la celda numerada con el número siguiente. Para realizar algún cálculo, se parte de ciertos datos almacenados y se realizan diversas operaciones; al final, el resultado está almacenado en alguna celda de memoria.

El paradigma imperativo permite expresar algoritmos que se realizan de la forma descrita, sólo que su especificación omite detalles de funcionamiento del ordenador concreto donde se va a utilizar. Los datos se almacenan en variables, es decir, una **variable** es una referencia abstracta a una zona de memoria donde se guardan datos. Las operaciones realizables se llaman **instrucciones** y su realización suele llamarse **ejecución**.

La instrucción principal es la de **asignación**, que tiene el siguiente formato:

<variable> := <expresión>

Por ejemplo:

x := 4

o bien

x := y + 1

La expresión situada a la derecha del símbolo de asignación ‘:=’ es una expresión funcional. Una instrucción de asignación expresa que primero se evalúa la expresión funcional y después se almacena su valor en la variable situada a la izquierda del símbolo de asignación.

Conviene tener en cuenta que la aparición de una variable en una instrucción de asignación tiene distinto sentido, dependiendo de que aparezca a la izquierda o a la derecha del símbolo de asignación. Si aparece a la izquierda, representa una celda de memoria; si aparece a la derecha, representa un valor (contenido en una celda de memoria).

El primer ejemplo anterior indica que se almacene el valor 4 en la variable **x**. El segundo ejemplo declara que al valor almacenado en la variable **y** se le sume uno y el valor resultante se almacene en la variable **x**, desapareciendo el valor anteriormente guardado en ella; el valor almacenado en la variable **y** no resulta afectado por la asignación.

Pueden combinarse varias instrucciones para hacer cálculos más complicados. La forma más sencilla es mediante la secuencia. Una **secuencia de instrucciones** es una colección de instrucciones situadas consecutivamente y separadas de alguna manera. La ejecución de una secuencia se realiza ejecutando las instrucciones una detrás de otra según el orden de su escritura,

de izquierda a derecha y de arriba abajo.

Por ejemplo, sea la secuencia en un lenguaje como Pascal:

```
x := 4;

x := y + 1
```

Al ejecutar esta secuencia, primero se realiza la primera asignación, que almacena un 4 en la variable **x**. Posteriormente se ejecuta la segunda instrucción, que almacena en la variable **x** el valor contenido en la variable **y** sumándole 1. Es obvio que, en este ejemplo, la segunda instrucción destruye lo realizado por la primera, así que dicha secuencia no tiene mucho sentido.

Veamos un ejemplo más complicado. Se parte de dos variables **x** e **y**, y se quieren intercambiar los valores contenidos en ellas. El lector puede intentar resolver el problema antes de continuar. (Pista: si necesita más variables, puede usarlas).

Dado que se quiere almacenar en cada variable el valor contenido en la otra, puede pensarse en asignar mutuamente sus valores, quedando:

```
x := y;

y := x;
```

o bien

```
y := x;
```

```
x := y;
```

Sin embargo, estas soluciones son erróneas. Si las asignaciones se realizaran simultáneamente, serían soluciones correctas, pero su ejecución es secuencial, por lo que se pierde alguno de los dos valores iniciales.

Supóngase, por ejemplo, que inicialmente **x = 4, y =1**. Si se ejecuta la primera solución, primero se asigna a **x** el valor contenido en **y**, quedando **x = 1**, **y =1**, y después se asigna a **y** el valor de **x**. El fallo consiste en que la primera asignación hizo desaparecer el valor que inicialmente contenía **x**, con lo que es imposible después recuperarlo para asignarlo a **y**. La segunda solución adolece de un problema análogo.

La solución consiste en usar una variable auxiliar **aux** cuya única misión sea guardar temporalmente el valor antiguo de una de las dos variables:

```
aux := x;

x := y;

y := aux
```

El lector puede comprobar que partiendo de **x = 4**, **y = 1** y ejecutando la secuencia anterior, se tiene finalmente **x = 1**, **y = 4**.

En resumen, la forma básica de expresar algoritmos en el paradigma imperativo

consiste en declarar las variables necesarias y diseñar una secuencia de asignaciones que transformen los valores almacenados. Para ejecutar un algoritmo imperativo, se parte de ciertos valores almacenados en las variables y se ejecutan las instrucciones del algoritmo secuencialmente; al final, los resultados calculados se encuentran almacenados en estas mismas u otras variables.

El paradigma imperativo se completa con otras características que dan mayor flexibilidad de cálculo, aunque siempre siguiendo un modelo abstracto de máquina de von Neumann: tipos de datos, declaraciones, instrucciones de flujo de control, etc.

- **Otros paradigmas**

Existen otros paradigmas distintos del funcional y el imperativo. Aunque no se abordan en esta cátedra, no queremos dejar de dar las características principales de algunos de ellos.

El **paradigma orientado a objetos** es una abstracción todavía mayor que el imperativo de la máquina de von Neumann. El programador ya no ve las variables como celdas de memoria, sino como objetos cuya estructura interna desconoce. Un objeto puede tomar un valor mediante una asignación, pero en

general el programador se ve forzado a utilizar un conjunto finito de operaciones. Un objeto pertenece a cierta clase, que define el conjunto de operaciones utilizables. El paradigma se completa con los conceptos de encapsulamiento, herencia, polimorfismo, etc (Smalltalk, Java).

El **paradigma ensamblador**, al contrario, es una concreción mayor de la máquina de von Neumann. Cada lenguaje ensamblador está asociado a una máquina concreta. Ahora los valores manipulados no son abstractos, sino que se maneja su representación binaria en la memoria de la computadora. Tampoco pueden usarse expresiones funcionales, sino que utilizando zonas de almacenamiento intermedio, llamadas registros, se hace operación tras operación. Otros elementos clave en el paradigma son los modos de direccionamiento, etiquetas, subrutinas, interrupciones, etc. El paradigma imperativo surgió del ensamblador, por lo que sus lenguajes asociados a veces son citados, respectivamente, como ***lenguajes de alto nivel*** **y** ***de bajo nivel***(Assembler).

El ***paradigma lógico***, como el funcional, tiene una base matemática: el concepto de predicado o relación. Un programa lógico está formado por hechos (predicados que se cumplen siempre) y reglas (la implicación de un predicado por otros); ambos elementos pueden contener variables que son conceptualmente iguales a los parámetros formales del paradigma funcional. La ejecución de un programa lógico es un proceso deductivo que intenta

comprobar que cierto predicado se satisface; la deducción tiene éxito si se encuentran algunos valores de las variables que permiten deducir el predicado a partir de los hechos y las reglas. El paradigma lógico es más flexible que los anteriores, porque permite hacer cálculos (de valores de variables) en varias direcciones. Otros elementos de este paradigma son la unificación, resolución, puntos de elección, variables lógicas, etc. (Prolog).

Con frecuencia se intentan clasificar los paradigmas según diversos criterios. Una clasificación bastante extendida agrupa a los paradigmas en dos grupos: ***con efectos laterales y declarativos***. Los primeros están basados en el modelo de Von Neumann y basan sus cálculos en ciertas entidades con un estado que se modifica explícitamente; pueden incluirse aquí los paradigmas ensamblador, imperativo y orientado a objetos. Los segundos expresan las soluciones algorítmicas en términos matemáticos, como funciones, relaciones, restricciones, etc.; e incluyen a los paradigmas funcional y lógico. Otra posible clasificación, que no desarrollaremos aquí, distingue la ***realización secuencial*** o ***paralela*** de las distintas partes de los algoritmos. ▲ i

D

GUÍAS DE ASIGNATURA RELACIONADAS CON ALGORITMOS, DE ALGUNAS UNIVERSIDADES NACIONALES Y EXTRANJERAS

Todas las guías identificadas con la tripleta Institución educativa – Facultad – Tópico (donde tópico puede ser el nombre de una o más asignaturas, un trabajo de investigación o una publicación), fueron obtenidas a partir de búsquedas en la Internet. Por tanto, se especifica el URL de origen para mayor información; si no aparece, significa que dicho contenido fue obtenido directamente en la institución educativa.

Se han omitido muchas universidades de prestigio, tanto locales como foráneas. Una exposición exhaustiva de guías de asignatura, haría este anexo bastante basto.

COLOMBIA

UNIVERSIDAD DE ANTIOQUIA

Departamento de Ingeniería de Sistemas

Algoritmos I

Método: Taller dialogante con construcción colectiva.

Sesiones presenciales de 1 hora y 50 minutos.

1. Presentación del curso: Temas generales, forma de evaluación (0%, 10%, 20%. 30%. 40%), sugerencias sobre un método de estudio. El computador y sus componentes funcionales.
2. Unidades de memoria, cálculo, control, entrada y salida. Manejo estático de la memoria, operandos y operadores.
3. Otras memorias, unidad de control, faces de diseño, traducción y ejecución. Examen 0%.
4. Estructuración, metodología de trabajo y características de los algoritmos.
5. Análisis de ejercicios elementales: cte-cte, cte-var, var-var, cte*(var-var).
6. Análisis de un caso generalizante bajo código rígido var*(var-var).
7. Análisis de problemas y creación de una solución
8. Examen 20%.
9. Arreglos unidimensionales. Análisis de ejercicios.
10. Análisis de problemas y creación de una solución.
11. Análisis de problemas y creación de una solución.

12. Arreglos multidimensionales. Análisis de ejercicios.
13. Análisis de problemas y creación de una solución.
14. Análisis de problemas y creación de una solución.
15. Análisis de problemas y creación de una solución.
16. Análisis de problemas y creación de una solución.
17. Examen 20%.
18. Subalgoritmos. Visión interna. Análisis de ejercicios.
19. Visión externa. Análisis de problemas y creación de una solución.
20. Análisis de problemas y creación de una solución.
21. Análisis de problemas y creación de una solución.
22. Análisis de problemas y creación de una solución.
23. Examen 30%.
24. POO. Análisis de ejercicios.
25. Análisis de problemas y creación de una solución.
26. Análisis de problemas y creación de una solución.
27. Análisis de problemas y creación de una solución.
28. Análisis de problemas y creación de una solución.
29. Análisis de problemas y creación de una solución.
30. Examen 30%.

UNIVERSIDAD DE ANTIOQUIA

Departamento de Ingeniería de Sistemas

Estructuras de Datos I

OBJETIVO GENERAL:

Definir las diferentes estructuras básicas como estructuras algebraicas e implementar a nivel algorítmico, las diferentes operaciones de una estructura de datos.

METODOLOGIA:

Exposición de elementos teóricos por parte del profesor, con solución de ejercicios en clase y ejercicios propuestos para que el estudiante desarrolle.

PROGRAMA DETALLADO DEL CURSO POR MÓDULOS:

1. Presentación de los temas a desarrollar, concertación de la evaluación con sus porcentajes y fechas y bibliografía. Definición de estructuras de datos y axiomas.

2. Estructura números naturales: funciones y axiomas. estructura arreglo: funciones y axiomas.

3. Estructura lista ordenada: funciones y axiomas. Estructura polinomio: funciones y axiomas.

4. Representación de polinomios: en vectores, forma 1 y forma 2. Ejercicios resueltos y ejercicios propuestos.

5. Representación de polinomios como listas ligadas. Ejercicios resueltos y ejercicios propuestos.

6. Matrices dispersas: definición y representaciones. Representación en tripletas y ejercicios.

7. Matrices dispersas: representaciones como listas ligadas, forma 1 y forma 2 y ejercicios.

8. Colas: Definición, formas de representación, ventajas y deseventajas de unas formas frente a otras y algoritmos básicos.

9. Ejercicios con colas y representación de múltiples colas tanto en vectores como en listas ligadas.

10. Listas generalizadas: definición, representación, ejemplos y ejercicios.

11. Representación de polinomios con múltiples variables como lista generalizada. Ejercicios resueltos y propuestos.

12. Arboles: definición y conceptos. Representación de árboles como listas ligadas y como lista generalizada.

13. Ejercicios de árboles representados como lista generalizada.

14. Arboles binarios: definición y conceptos. Representación de árboles binarios en vectores y como listas ligadas.

15. Recorridos sobre árboles binarios.

16. Ejercicios de aplicaciones con árboles binarios.

17. Arboles binarios enhebrados. Conceptos y ejercicios.

18. Arboles AVL. Definición, conceptos y aplicaciones.

19. Construcción de árboles AVL: rotaciones y balanceo.

20. Grafos: definición, conceptos y representaciones.

21. Construcción de grafos y recorridos sobre ellos.

22. Determinación de árboles de costo mínimo con base en grafos dados, algoritmos de Kruskal y de Prim.

23. Determinación de recorridos mínimos sobre grafos. Algoritmos de Dijkstra y Floyd.

24. Expresiones: Infijo, prefijo y posfijo.

25. Manipulación de expresiones en las diferentes formas de representación.

26. Ejercicios con expresiones.

BIBLIOGRAFIA:

Flórez Roberto, Algoritmos y estructuras de datos (texto guía)

Becerra César, estructuras de datos en C.

Aho, ullman, Algoritmos y estructuras de datos.

Horowitz and Sahni, Fundamentals of data structures.

Knuth, Algoritmos fundamentales.

EVALUACION:

Cuatro exámenes, cada uno del 25%. Los temas correspondientes a cada examen son: Axiomatización y polinomios, primer examen; matrices dispersas y colas, segundo examen; listas generalizadas y árboles, tercer examen, y grafos y expresiones, el cuarto examen.

UNIVERSIDAD DE ANTIOQUIA

Departamento de Ingeniería de Sistemas

Estructuras de Datos II

OBJETIVO

Al finalizar el curso el estudiante debe conocer y manejar las estructuras de datos que le permitan definir y manipular archivos de datos en memoria principal y secundaria.

CONTENIDO

ARCHIVOS

METODOS DE ACCESO UNIDIMENSIONAL

1. Introducción,

2. Dispositivos de acceso secuencias y directo (DASD)

3 . Accesos en línea y por lotes

4. Archivos de Registros de longitud fija.

5. Archivos de Registros de longitud variable.

6. Técnicas de indización : método ISAM

7. Técnicas de accesos directos: Hashing estático

S. Técnicas de accesos directos: Hashing dinámico lineal

9. Técnicas de accesos directos: Hashíng dinámico

10. Técnicas de accesos directos: Hashing dinámico extendido

11. Taller

12 Evaluación

13. Arboles m-way y árboles B

14. Indización por árboles B+

15 Árboles trie , árboles doblemente encadenados y archivos tipo anillo

16. Árboles PATRICIA

17. Indices secundarios: multilistas e invertidos

18. Taller

ORDENAMIENTO

19. Introducción

20. Ordenamiento por Inserción

21. Ordenamiento por Intercambio

22. Ordenamiento por Selección

23 Ordenamiento por Mezcla

24 Ordenamiento por Distribución

25. Ordenamiento Externo por Mezcla multi-vía

26. Evaluación

BÚSQUEDA

27. Introducción

28. Búsqueda y Binaria

29. Búsqueda Digital

30. Evaluación.

BIBLIOGRAFÍA

Korth Henry, FUNDAMENTOS DE BASES DE DATOS, ED. McGraw Hill, 2 ed.

Wiederhold Gio, DISEÑO DE BASES DE DATOS, ED. McGraw Hill, 2 ed.

Knuth Donald, EL ARTE DE PROGRAMAR ORDENADORES, Volumen 3, ED. Reverté.

Uliman Jeffrey, DATABASE AND KNOWLEDGE-BASE SYSTEMS, Volume 1, ED.

Computer Science Presa,

Yedidyah Langsam, Tenenbaum Aaron, ESTRUCTURAS DE DATOS CON C Y C++,

ED. Prentice Hall, 2 cd.

---

http://bochica.udea.edu.co/~frios/AOO/clase_Grupo.html

LAB. INTEGRADO DE SISTEMAS, FACULTAD DE INGENIERÍA, U. DE A.

Problema:

Elaborar un algoritmo que permita crear grupos de estudiantes con sus notas, listar un estudiante y listar todos los estudiantes

Dos versiones de solución:

Muy procedimental y poco objetual

Menos procedimental y más objetual

Versión aún muy procedimental

```
clase GRUPO
{
  publicos:
        grupo(IDENTIFICACION)
        recibaporcentajes(DIR_LISTA_PORCENTAJES)
        adicioneesteestudiante(CARNET, DIR_LISTA_NOTAS)
}

clase DATO
{
 publicos:
        DATO(NUM)
        DEMESUVALOR(VAL)
}

inicie
{
 muestre "Solicite el tipo de servicio así: 1: crear   2: listar un estudiante   3: listar
todo el grupo   0: terminar"
 entre SERVICIO
   LG es_una LISTA
```

```
mientras   SERVICIO <> 0   haga
  {
    casos de SERVICIO:
      caso 1:
        {
          muestre "Cuál es la identificación del grupo ?"
          entre IG
              AREA(GRUPO(IG), DIRG)  //se instancia un objeto dinámico tipo
              GRUPO. El SO da su dirección en DIRG.
          muestre "Entre el  % de la evaluación "
          entre POR
         AREA(LISTA, DIRP)  //se instancia un objeto dinámico tipo LISTA. El
         SO da su dirección en DIRP.
                    mientras   POR > 0   haga
                       {
                        AREA(DATO(POR), DIRD)   //se instancia un objeto
                       //dinámico tipo  DATO. El SO da su dirección en DIRD.
                        DIRP^adicione_como_cola(DIRD, it_POR)
                        muestre "Entre el  % de la evaluación "
                        entre POR
                        }
                    DIRG^recibaporcentajes(DIRP)
```

```
        muestre "Entre el  carné del estudiante "
        entre CARNE
        mietras    CARNE > 0     haga
          {
            AREA(LISTA, DIRN)   //se instancia un objeto dinámico tipo
                                  //LISTA. El SO da su dirección en DIRN.
            muestre "Entre una nota del estudiante "
            entre NOTA
            mientras    NOTA >= 0   haga
                {
                    AREA(DATO(NOTA), DIRD)   //se instancia un objeto
                                  dinámico tipo DATO. Su dirección: DIRN.
                    DIRN^adicione_como_cabeza(DIRD, it_NOT)
                    muestre "Entre una nota del estudiante "
                    entre NOTA
                 }
            DIRG^adicioneesteestudiante(CARNE, DIRN)
            muestre "Entre el  carné del estudiante "
            entre CARNE
          }
    }
caso 2:
    {
```

```
                ***** en construccion sep.12 2001 *****
            }
        caso 3:
            {


            }


    }
 muestre "Solicite el tipo de servicio así: 1: crear   2: listar un estudiante   3:
listar todo el grupo   0: terminar"
 entre SERVICIO
}
```

UNIVERSIDAD COOPERATIVA DE COLOMBIA

FACULTAD DE INGENIERIA

MEDELLÍN

PROGRAMA: Ingeniería de Sistemas

ASIGNATURA: Estructura de Datos I

INTENSIDAD HORA: 4

PREREQUISITOS: Programación

TIPO DE CURSO: Teórico X Practica _

1. OBJETIVO GENERAL

Desarrollar un curso de estructuras de datos lineales y no lineales, con los algoritmos necesarios para su tratamiento representados por medio de diagramas de bloques, con independencia de cualquier lenguaje de programación. Presentando en forma clara y explícita los conceptos de funcionalidad, aplicación y operaciones básicas con las diferentes estructuras y las técnicas de trabajo con algoritmos recursivos.

2. OBJETIVOS ESPECIFICOS

Lograr que al finalizar el curso el estudiante este en capacidad de:

- Definir, recorrer, almacenar datos y efectuar las operaciones básicas en las estructuras: arreglo unidimensional (Vector), Arreglos bidimensional (Matriz), lista, pila, cola y árbol binario.
- Realizar operaciones con combinación de estructuras.
- Identificar en un problema determinando el tipo de estructura a apropiada para resolverlo.

- Identificar en un problema determinado, el tipo de estructura apropiado par resolverlo.
- Distinguir como aperar la recursividad y su efectividad en el recorrido de arboles.

3. METODOLOGIA

- Exposiciones de los temas por el catedrático en aula de clase.
- Resolver de problemas típicos y clásicos en clase, por el profesor, a partir de los cuales el alumno podrá resolver los que le serán entregados en talleres.
- Participación en talleres y trabajos teórico_prácticos para afianzar el conocimiento del estudiante.
- Intervenciones de los estudiantes con el objeto de exponer un tema específico o con el fin de aclarar dudas.

4. CONTENIDO

UNIDAD I 10 Horas

1. Conceptos básicos

1.1 Programación

1.2 Algoritmos

1.2.1 Representación gráfica

2. Modularización y generalización

2.1 Programa principal

2.2 Subprogramas.

3. Tipos y estructura de datos

3.1 Tipos de datos

3.2 Arreglos

3.3 Matrices

UNIDAD II 8 Horas

Estructuras Lineales

Conceptos Básicos

1. Puntero
2. Procedimientos traer y liberar
3. La constante nula
4. Registro

UNIDAD III 12 Horas

Estructura de datos: Lista

1. Lista simplemente ligada
2. Lista simplemente ligada circular
3. Lista doblemente ligada
4. Lista doblemente ligada circular

UNIDAD IV 10 Horas

Estructura de datos: Pila

1. Definición
2. Representación
3. Operaciones con filas
4. Aplicaciones.

UNIDAD V 10 Horas

Estructura de datos: Cola

1. Definición
2. Representación de las colas
3. Operaciones con colas
4. Tipos de colas
5. Aplicaciones

UNIDAD VI 12 Horas

Estructura de datos: Arbol

1. Características y propiedades
2. Arboles binarios
3. Representación de expresiones aritméticas mediante arboles.
4. Recorrido de arboles binarios
5. Arboles binarios de búsqueda.

6. BIBLIOGRAFIA

Texto guía:

Estructura de datos. Osvaldo Cairo / Silvia Guardati. Ed. McGraw-Hill

Textos de consulta:

1. Fundamentos de programación, algoritmos y estructuras de datos.

   Luis Joyanes Aguilar.

   McGraw-Hill

2. Estructura de datos en C.

   Cesar Becerra Santamaria.

3. Estructura de datos en Pascal

   Moshe Augenstein

   Aaron Tenembaum

4. Desarrollo de algoritmos y sus aplicaciones en Basic, Pascal, Cobol y C

   Guillermo Correa Uribe

   McGraw-Hill

UNIVERSIDAD COOPERATIVA DE COLOMBIA

FACULTAD DE INGENIERIA

MEDELLÍN

FACULTAD : INGENIERIAS

PROGRAMA : INGENIERÍA DE SISTEMAS

ASIGNATURA : LOGICA DE PROGRAMACION

CODIGO : INF91

INTENSIDAD H/S : 6

1. OBJETIVOS

Utilizar la lógica de programación en la solución de problemas mediante el uso del computador.

Plantear solución de problemas mediante técnicas modulares, aplicando la lógica de la programación con el correcto uso del computador.

2. METODOLOGIA

Se utilizará el método de exposición en el aula de clases y se asignarán prácticas que se desarrollarán en el laboratorio de microcomputadores, para aplicar los conceptos teóricos estudiados.

3. CONTENIDO

3.1 Introducción al curso

3.2 Variables y constantes

3.3 Expresiones lógicas, aritméticas y booleanas

3.4 Entrada y salida de datos

3.4.1 Asignación a variables

3.4.2 Condicionales

3.4.3 Ciclos

3.5 Vectores

3.6 Matrices

3.7 Subprogramas

4**. BIBLIOGRAFIA**

- Correa Uribe , Guillermo. Desarrollo de algoritmos y sus aplicaciones en BASIC, Pascal, COBOL y C.
- Joyanes, Luis. Problemas de metodología de la programación.
- Joyanes, Luis. Fundamentos de programación.
- Lozano, Letvin. Diagramación libre y estructurada.
- Vasquez, Gabriel. Lógica para programación de computadores.
- Ríos, Fabián. Notas sobre algoritmos. U. de A.
- Guarín, Hugo. Lógica simbólica.
- Matemática moderna y Matemáticas discretas.

PROGRAMA:	TECNOLOGIA EN SISTEMAS
ASIGNATURA:	LÓGICA DE PROGRAMACIÓN
CODIGO:	3002015
SEMESTRE:	**I**
INTENSIDAD:	SEIS HORAS SEMANALES
PREREQUISITOS:	
CORREQUISITOS:	INTRODUCCIÓN A LOS SISTEMAS

OBJETIVO GENERAL

Planear y desarrollar algoritmos computacionales basados en conceptos de las matemáticas discretas (álgebra de Boole, teoría de conjuntos, etc.) y en estrategias metodológicas (consecuencia del análisis y el raciocionio) modelables a partir de las sentencias de control (decisión y ciclos).

OBJETIVOS ESPECIFICOS

1. Diferenciar entre los distintos tipos de datos para la evaluación de expresiones aritméticas y booleanas.
2. Aprender a identificar situaciones que impliquen toma de decisiones e iteración de instrucciones.
3. Solucionar problemas con la técnica de diseño de algoritmos “Divide y vencerás” o algoritmia modular, para conformar bibliotecas de subalgoritmos reutilizables.

PRESENTACION DE CONTENIDOS POR UNIDADES

UNIDAD 1: CONCEPTOS BASICOS DE LOGICA

- Definición
- Ejercicios para el razonamiento lógico.
- Conceptos básicos de programación.
- Algoritmos y su clasificación.
    - Cualitativos
    - Cuantitativos
- Clasificación de los lenguajes de programación.
- Definición de compilador.
- Tipos de datos
- Operadores y expresiones.
- El operador de asignación
- Conversiones de tipo.
- Entrada y salida datos.
- Ejercicios de aplicación.

UNIDAD 2: ETAPAS EN LA RESOLUCIÓN DE UN PROBLEMA.

- La resolución de problemas.
- Análisis del problema.
- Diseño de la solución.
- Representación gráfica de los algoritmos.

- Diagrama de Nassi-Schneiderman(N-S).
- Diagrama de flujo
- Pseudocódigo.

UNIDAD 3: ESTRUCTURA GENERAL DE UN ALGORITMO.

- Características de un algoritmo.
- Partes constitutivas de un algoritmo
- Tipos de instrucciones:
  - De asignación
  - De lectura de datos
  - De escritura de resultados
  - De bifurcación
- Elementos básicos de un programa
  - Bucles
  - Contadores
  - Acumuladores
  - Decisiones
  - Interruptores
- Escritura de Algoritmos
  - Cabecera del algoritmo
  - Declaración de variables
  - Declaración de constantes
  - Estilo de escritura

- Ejercicios de aplicación.

UNIDAD 4: PROGRAMACION ESTRUCTURADA

- Paradigmas de programación
- Programación estructurada
- Diseño descendente
- Estructura secuencia
- Estructura selección
  - Simple (SI - ENTONCES / IF - THEN)
  - Doble (SI - ENTONCES- SINO / IF - THEN - ELSE)
  - Múltiple (SEGÚN_SEA, EN_CASO_DE / CASE)
- Estructuras repetitivas.
  - Mientras (WHILE)
  - Repetir (REPEAT)
  - Desde/Para (FOR)
- Estructuras de decisión anidadas.
- Estructuras repetitivas anidadas.
- Ejercicios de aplicación.

UNIDAD 5: SUBPROGRAMAS (SUBALGORITMOS)

- Concepto
- Procedimiento

- Función
- Paso de parámetros por valor y referencia
- Ejercicios de aplicación

UNIDAD 6: VECTORES Y MATRICES

- Operaciones con vectores
    - Formas de creación
    - Recorrido
    - Búsqueda
    - Inserción y eliminación
    - Actualización
    - Ordenamiento
- Operaciones con matrices
    - Creación
    - Inserción de filas y columnas
    - Eliminación de filas y columnas
    - Recorrido
- Ejercicios de aplicación.

METODOLOGIA

- Sesiones magistrales.
- Desarrollo de Talleres Individuales y en grupo.

BIBLIOGRAFIA

- Fundamentos de Programación, Algoritmos y Estructura de Datos.

  Luis Joyanes Aguilar, McGraw Hill

- Programación Estructurada

  Efraín Oviedo

  Universidad de Antioquia

- Problemas de Metodología de la Programación.

  Luis Joyanes Aguilar

  McGraw Hill.

DISEÑO MICROCURRICULAR

1. INFORMACIÓN GENERAL

1.1 Nombre del Programa: Tecnología en Sistemas de Información

1.2 Núcleo o Asignatura: Lógica y Programación LPI-308

1.3 Intensidad horaria: 8 horas semanales 128 horas semestrales

1.4 Prerrequisitos: Introducción a la tecnología en sistemas de información

1.5 Correquisitos: Ninguno

2. PERTINENCIA

2.1 Área o Campo del saber: Tecnología Informática: Algoritmos, estructura de datos y principios de programación utilizando un lenguaje de alto nivel.

2.2 Objeto de Estudio: Representación de soluciones a problemas mediante el diseño de instrucciones lógicas siguiendo modelos estandarizados, representación de los datos y posibles transformaciones y conocimiento y aplicación de un lenguaje de programación de alto nivel.

2.3 Relevancia Pedagógica del Objeto de Estudio:

El tecnólogo en sistemas de información se enfrenta permanentemente a la solución de problemas, los algoritmos, las estructuras de datos y los lenguajes

de alto nivel son el fundamento para la construcción del pensamiento lógico necesario que le permite diseñar y planificar soluciones lógicas de manera rigurosa y sistemática.

2.4 Microcompetencias o habilidades del Núcleo:

Conocer los fundamentos de lógica de programación

Conocer los principios sobre la programación sin interfaz

Conocer los principios sobre la programación visual

Diseñar y elaborar aplicativos sencillos en lenguajes de alto nivel

3. ESTRUCTURA PROGRAMÁTICA

3.1 Eje problémico o temático	3.2 Competencias práctico-experimentales y del saber	3.3 Contenido del eje	3.4 Tiempo (horas)
Lógica de programación	• Analizar problemas en forma lógica • Resolver problemas “estructurados” • Representar la solución de problemas en lenguajes descriptivos	• Lúdica • Definición de algoritmos y variables (Repaso) • Diagrama de Flujo • Diagramación de algoritmos	32

	• Aprender a dividir problemas en subproblemas • Resolver problemas en forma estructurada • Aprender la reutilización de procesos • Manejar la persistencia de la información	• Pseudocódico • Estructuras de decisión • Estructuras de repetición • Funciones • Subrutinas, SubProcedimientos • Archivos	
Estructuras de datos	• Aprender el almacenamiento de datos en forma grupal • Operar datos en forma de grupos • Conocer modelos especiales para el almacenamiento de datos. • Implementar estructuras de datos.	• Arreglos unidimensionales • Arreglos bidimensionales • Arreglos multidimensionales • PIlas • Listas y colas • Árboles • Grafos	48
Programación	• Conocer los diferentes lenguajes de	• Conceptos básicos • Tipos de programación	48

	programación con el cual se pueden escribir, poner a prueba y optimizar programas de computador para la solución de problemas específicos. • Identificar las estructuras Básicas con las cuales se puede escribir un programa. • Conocer el entorno de herramientas visuales y no visuales. • Reconocer la sintaxis de lenguajes de programación comercial como el Visual Basic. • Diseñar interfaces de usuario gráficas.	• Programación Visual • Estructuras de entrada • Estructuras de salida • Variables • Estructuras de decisión • Estructuras de repetición • Arreglos • Funciones • Subrutinas • Archivos	

4. ESTRATEGIAS METODOLÓGICAS

De carácter demostrativo:

Exposición de modelos con análisis paso a paso.

Explicaciones magistrales en procesos estandarizados.

Observación de trabajos de autores.

Seminarios

Exposiciones de los estudiantes

De carácter mental:

Análisis de modelos propuestos

Generalización de conceptos a través de ejercicios de aplicación

Propuesta de problemas

Elaboración de conceptos a partir de la demostración

De carácter práctico:

Solución de ejercicios planteados en clase

Utilización del computador como herramienta de evaluación del diseño propuesto.

Talleres extraclase

De carácter experimental:

Propuesta de nuevos problemas y soluciones

Pruebas de ensayo-error

Comprobaciones haciendo uso del computador

De carácter operativo:

Planteamiento de problemas que a partir de cero, se exige el planteamiento, la discusión y la puesta en práctica de una entre varias soluciones.

5. CRITERIOS DE EVALUACIÓN

Para la evaluación de esta asignatura, se tienen en cuenta las características individuales del desempeño de los estudiantes y para ello, se observará el desarrollo de las competencias en diferentes momentos, uno inicial de diagnóstico, uno segundo de adquisición y asimilación durante todo el proceso y, un tercero de comprobación de la competencia por medio de la aplicación.

Por el carácter teórico - práctico, utiliza en la evaluación de los estudiantes actividades de talleres grupales, pruebas escritas, presentación de trabajos, demostraciones y exposiciones.

En la práctica evaluativa se observará en los alumnos:

Identificación de los conceptos básicos de programación y manejo ético del software y los derechos de autor..

Reconocimiento de las estructuras básicas de un programa y aplicación de las sentencias y estructuras en programas simples. Utilización de controles básicos de la interfaz.

Ordenación de datos con diversos criterios de arreglos y aplicación de la programación modular con funciones y subrutinas.

Diferenciación de la sintaxis de sentencias en varios lenguajes de programación; cambio de instrucciones de un lenguaje a otro y análisis técnico para decidir entre varios lenguajes.

El estudiante, acompañado por el docente, desarrollará un miniproyecto con base en los fundamentos del curso.

6. MODALIDAD PEDAGÓGICA

Cada uno de los ejes temáticos se administrará en forma presencial en un 90%. El estudiante deberá complementarlo en su tiempo y espacio con talleres de práctica diseñados para cada eje. Estos talleres, así como la síntesis teórica de cada eje, se presentarán en la página Web a la cual tendrá acceso el estudiante en cualquier hora y lugar, mediante una terminal conectada a Internet. Si embargo, queda claro que por cada hora presencial, el estudiante debe dedicar dos horas para afianzar su aprendizaje.

7. BIBLIOGRAFÍA ESPECÍFICA

JOYANES AGUILAR, Luis. Fundamentos de programación Algoritmos y Estructura de Datos. McGraw Hill, Colombia, 1999

OVIEDO, Efrain. Algoritmos estructurados.

CORREA, Guillermo. Desarrollo de Algoritmos

Aho, ALFRED V. Estructura de datos y algoritmos. 1988.

ANTONAKOS MANSFIELD, Estructura de datos en C

BECERRA SANTAMARÍA, César Estructura de datos en C.

Cairó, Osvaldo. Estructura de datos. Editorial McGraw-Hill, 1993.

CEVALLOS, Francisco Javier. Programación Orientada a objetos en C++.

Horowitz, Ellis. Fundamentals of data Structures. Editorial Computer Science Press.

JOYANES, Luis y Otros. Estructura de datos – Libro de Problemas. Madrid. McGraw- Hill, 1999

Villalobos S., Jorge A. Diseño y manejo de estructuras de datos en C. Editorial McGraw-Hill, 1996.

Wirth, N. Algoriths + Data Structures = Programs. Editorial Prentice Hall, 1989.

ERSKINE, Robert. Iniciación a la programación. 1986

HEHNER, Eric A Practical Theory of Programming 1995

LÓPEZ. Fundamentos de programación. Paraninfo, 1998

ALCALDE. Metodología de la programación. McGraw-Hill, 1997

QUERO. Programación en Lenguajes estructurados, Paraninfo, 1997

CAIRO. Metodología de la programación. AlfaOmega,1995

McKELY, Mike y otros. Visual Basic 5. Edición Especial. Prentice Hall.1998

Microsoft Visual Basic. Languaje reference. Microsoft Corporation, 1995

Microsoft Visual Basic. Programmer's Guide. Microsoft Corporation, 1995

CEBALLOS, Fco. Javier. Enciclopedia Visual Basic 4. Rama, 1996

Antonakos. Programación estructurada en C, 1987

ESPAÑA

UNIVERSIDAD COMPLUTENSE DE MADRID

FACULTAD DE CIENCIAS MATEMÁTICAS

ADQUISISIÓN Y TRATAMIENTO DE DATOS

http://www.ucm.es/info/dsip/Asignaturas/Matematicas/MAT406.htm

Facultad de Ciencias Matemáticas

Curso académico 2002-2003

Asignatura: Adquisición y tratamiento de datos

Código: 406

Titulación: Licenciatura en Ciencias y Técnicas Estadísticas

Curso: 2º ciclo

Créditos: 9

Departamento: Sistemas Informáticos y Programación

Resumen del programa:

Esta asignatura se presenta como una continuación de otra previa de introducción a la programación. El objetivo es que el alumno adquiera una visión amplia de las técnicas en las que se apoya la programación. En la primera parte se presentan las estructuras de datos básicas y métodos para su manipulación. A continuación se enseñan los tipos de algoritmos más comunes y su eficiencia.

Resumen del programa en inglés:

This subject follows other subject that is an introduction to Programming. The goal is that the student learn the techniques which programming is based on.

The first part introduces basic data structures with the correspondent manipulation algorithms. The second part introduces the different kinds of algorithms and their efficiency.

Programa detallado:

1. Introducción: Principios de Programación en JAVA: Tipos primitivos, operadores básicos, sentencias condicionales e iterativas, entrada y salida, manejo de excepciones. Clases.

2. Estructuras de Datos. Tipos abstractos de datos (TADs). Conceptos y programación

Tipos de datos lineales: Pilas. Colas. Listas. Tablas dispersas, hash. Colas de prioridad

Tipos de datos no lineales: Árboles binarios. Árboles de búsqueda. Grafos.

3. Algoritmos: Estructuras algorítmicas básicas. Diseño recursivo e iterativo.

Introducción: Eficiencia. Análisis de algoritmos. Estudio de complejidad.

Divide y vencerás.

Método voraz.

Programación dinámica.

Búsqueda de soluciones:

Vuelta atrás (Backtracking),

Ramificación y acotación (Branch & Bound)

Bibliografía:

Básica (por orden de prioridad):

Primera parte:

§ Lafore, R. Data Structures & Algorithms in JAVA. Ed. Waite Group 1998

§ K. Arnold, J. Gosling. The Java Programming Language. Addison Wesley, 1996.

§ Weiss, M. A. Estructuras de Datos en Java, 2000.

§ Weiss, M. A. Data structures and problem solving using Java, 1998.

§ Wirth, N. Algoritmos + estructuras de datos = programas, 1994.

§ Cormen, T.H., Leiserson, C.E., Rivest, R.L. Introduction to Algorithms, The MIT Press, 1990.

§ Brassard, G., Bratley, P. Fundamentals of Algorithms, Prentice Hall, 1995.

Versión en castellano: Fundamentos de Algoritmia, Prentice Hall, 1997.

Complementaria:

Primera parte:

§ A.V.Aho, J.E.Hopcroft, J.D.Ullman. Data Structures and Algorithms, Addison Wesley, 1983.

§ G. Brassard, P. Bratley. Algorítmica. Concepción y análisis. Ed. Masson, 1990.

(Versión en castellano, 1992.)

§ Weiss, M. A. Data structures and algorithm analysis in Java, 1999.

§ Wirth, N. Algoritmos y estructuras de datos, 1987.

§ Eckel, Bruce. Thinking in Java : the definitive introduction to object-oriented programming in the language of the World-wide Web Prentice-Hall, 1998.

También en http://www.BruceEckel.com.

Desarrollo de la asignatura:

La asignatura se impartirá con desarrollos teóricos en el aula con una sesión semanal de laboratorio donde se programarán los aspectos vistos en teoría. El lenguaje escogido para la asignatura completa es JAVA.

Método de evaluación:

La entrega de las prácticas en Java es requisito para aprobar la signatura. Se realizará un examen parcial liberatorio en febrero. Examenes finales en Junio y Septiembre. Para aprobar la asignatura es obligatorio entregar las prácticas de laboratorio.

UNIVERSIDAD COMPLUTENSE DE MADRID

FACULTAD DE INFORMÁTICA

ESTRUCTURA DE DATOS Y DE LA INFORMACIÓN

http://www.ucm.es/info/dsip/Asignaturas/Sistemas/FI611.htm

Facultad de Informática

Curso académico 2002-2003

Asignatura: Estructura de Datos y de la Información

Código: 611

Titulación: Ingeniería en Informática de Sistemas

Curso: 2º

Créditos: 12

Departamento: Sistemas Informáticos y Programación

Resumen del programa:

El objetivo perseguido en esta asignatura es proporcionar a los alumnos la capacidad de especificar tipos abstractos de datos; implementarlos en un lenguaje imperativo, eligiendo para ello estructuras de datos adecuadas; razonar sobre la corrección y eficiencia de tales implementaciones; y desarrollar programas que utilicen dichos tipos abstractos de datos.

Resumen del programa en inglés: The aim of this subject is to provide the students with the following abilities: to specify abstract data types; to implement them in an imperative language, on the basis of suitably chosen data structures; to reason on the correctness and efficiency of such implementations; and to develop programs which use abstract data types.

Programa detallado:

1. Análisis de la eficiencia de algoritmos.

2. Diseño de algoritmos iterativos e recursivos.

3. Tipos abstractos de datos: Especificación e implementación.

4. Tipos de datos lineales: Pilas, colas y listas.

5. Árboles binarios y de búsqueda. Montículos.

6. Tablas.

7. Grafos.

Bibliografía básica:

· Peña, R., Diseño de programas. Formalismo y abstracción. Segunda edición. Prentice Hall. 1998.

· Heileman, G.L., *Estructuras de datos, algoritmos y programación orientada a objetos.* McGraw Hill, 1998.

· Langsam, Y., Augenstein, M.J. y Tenenbaum, A.M., *Estructuras de datos con C y C*++. Prentice Hall , 1997

Bibliografía complementaria:

· Brassard, G., Bratley, P., *Fundamentos de Algoritmia*. Prentice Hall, 1997

· Cormen, T.H., Leiserson, C.E., Rivest, R.L., Introduction to algorithms. The MIT Press. 1998.

· Aho, A.V., Hopcroft, J.E., Ullman, J.D., *Data Structures and Algorithms*, Addison Wesley, 1983.

· Horowitz, E., Sahni, S., Mehta, D., *Fundamentals of Data Structures in C++*. W.H. Freeman & Co., 1995

· Franch, X., Estructuras de datos. Especificación, diseño e implementación. Ediciones UPC. 1994.

· Balcázar, J.L., *Programación metódica*. McGraw-Hill, 1993.

Desarrollo de la asignatura:

Clases teóricas y clases prácticas con resolución de ejercicios.

Método de evaluación:

Se realizará un examen parcial en febrero, y exámenes finales en junio y septiembre.

Para aprobar la asignatura, los alumnos tendrán que superar alguno de los dos exámenes finales (junio o septiembre). El examen de febrero tiene un carácter liberatorio de la materia del primer cuatrimestre por lo que respecta al examen final de junio. Existe la posibilidad de examinarse de la asignatura completa, aun habiendo aprobado el examen de febrero, con vistas a mejorar la nota de la asignatura.

FACULTAD DE INGENIERÍA

Algoritmos - Curso 2001 - 2002

Ingeniería Técnica en Informática de Gestión

http://carpanta.dc.fi.udc.es/~valderru/alg/gprog02.html

Programa

1. **Análisis de Algoritmos**

   1.1 Análisis de la eficiencia de los algoritmos

   Notaciones asintóticas, Modelo de computación, Verificación empírica del análisis

   1.2 Cálculo de los tiempos de ejecución

   Análisis de los casos peor y medio, Cálculo de O

2. **Estructuras de datos**

   2.1 Listas, pilas y colas

   2.2 Árboles

   2.3 Dispersión (*hashing*)

   2.4 Colas de prioridad (montículos)

   2.5 Conjuntos disjuntos

3. **Algoritmos de Ordenación**

   3.1 Ordenación por inserción

   3.2 Ordenación de Shell

   3.3 Ordenación por montículos (*heapsort*)

3.4 Ordenación por fusión (*mergesort*)

3.5 Ordenación rápida (*quicksort*)

4. **Algoritmos de grafos**

   4.1 Representación de grafos

   4.2 Ordenación topológica

   4.3 Árbol de recubrimiento mínimo: algoritmos de Kruskal y Prim

   4.4 Algoritmos del camino más corto: algoritmo de Dijkstra

   4.5 Recorridos sobre grafos

5. **Técnicas de diseño de algoritmos**

   5.1 Algoritmos ávidos o voraces

   5.2 Divide y Vencerás

   5.3 Programación dinámica

   5.4 Algoritmos aleatorios

   5.5 Algoritmos con retroceso

   5.6 Algoritmos paralelos

Bibliografía

- M.A. Weiss, *Estructuras de Datos y Algoritmos*, Addison Wesley 1995.
- G. Brassard y P. Bratley, *Fundamentos de Algoritmia*, Prentice Hall 1997.
- G. Brassard y P. Bratley, *Fundamentals of Algorithmics*, Prentice Hall 1996.
- U. Manber, *Introduction to Algorithms - A Creative Approach*, Addison Wesley 1989.

Textos Complementarios:

- G.L. Heileman, *Estructuras de Datos, Algoritmos, y Programación Orientada a Objetos*, McGraw-Hill 1998.
- R. Peña Marí, *Diseño de Programas, Formalismo y Abstracción*, Prentice Hall 1998.
- T.H. Cormen, C.E. Leiserson y R.L. Rivest, *Introduction to Algorithms*, MIT Press 1990.
- R. Sedgewick, *Algorithms*, Addison Wesley 1988.

MÉXICO

UNIVERSIDAD TECNOLÓGICA DE LA MIXTECA

FACULTAD DE INGENIERÍA

EXPOSICIONES Y PUBLICACIONES

http://nuyoo.utm.mx/~caff/doc/Difusion.html

En esta sección encontrarán la lista de los principales documentos generados, muchos de estos colocados en línea. Para visualizar los documentos se recomienda Acrobat Reader 5

Notas de clase

- Apuntes de Análisis y Diseño Orientado a Objetos (Versión 2001, UML y RUP). Tamaño 4.82 Mb
- Apuntes de Análisis y Diseño Orientado a Objetos (Versión 2000, UML). Tamaño: 10.7Mb
- Ingeniería de Requerimientos.
- Tecnologías de Objetos.[1]
- Programación Orientada a Objetos con Java.[2]
- Programación Orientada a Objetos con C++.
- Resumen de notas de Programación Orientada a Objetos con C++.
- Apuntes de Compiladores.

- Estructuras de Datos.
- Lenguaje C.

Exposiciones

- Ponencia **Aplicación del Modelo de Comercio Electrónico para la sistematización de la Biblioteca de la UTM.** ANIEI. XV Congreso Nacional y I Congreso Internacional de Informática y Computación. 23 al 25 de octubre de 2002. Guadalajara, Jalisco. México.
- Ponencia **Manejo de Transacciones en un Subsistema de Almacenamiento de un prototipo de Sistema Manejador de Base de Datos Orientado a Objetos.** SMCC. Tercer Encuentro Internacional de Computación ENC'01. Septiembre, 2001. Aguascalientes, México.
- Seminario **El Proceso Unificado Rational para el Desarrollo de Software.** Octubre, 2000.
- Ponencia **Suite Única de Transacciones Electrónicas (S.U.T.E).** México. XI Simposium Nacional de Informática. 2000.
- Conferencia **Propuesta de un Ambiente para el Modelado de Sociedades de Agentes Reactivos.** México. Memorias del XI Simposium Nacional de Informática. 2000.
- Conferencia **Introducción a la Ingeniería de Software.** 1999. ITAO. Oaxaca, México
- Seminario **Subsistema de Almacenamiento de un prototipo de SMBDOO.** UTM. Oaxaca.1999.

- Seminario **Introducción a Java.** UTM. Oaxaca.1998.

Publicaciones

- Fernández y Fernández, C A; Moreno Rocha, M A. **Evolución del Entorno Tecnológico para la Enseñanza a Distancia en la Universidad Virtual de la UTM.** IV Jornadas Sobre Informática y Sociedad. 11 al 13 de diciembre de 2002. Barcelona, España.
- Fernández y Fernández, Carlos Alberto. **Aplicación del Modelo de Comercio Electrónico para la sistematización de la Biblioteca de la UTM.** ANIEI. XV Congreso Nacional y I Congreso Internacional de Informática y Computación. 23 al 25 de octubre de 2002. Guadalajara, Jalisco. México.
- Fernández y Fernández, Carlos Alberto. **Modelado Visual con UML.** UTM. TEMAS de Ciencia y Tecnología. Vol.6 Número 16. Enero-Abril 2002. México.
- Fernández y Fernández, Carlos Alberto. **Manejo de Transacciones en un Subsistema de Almacenamiento de un prototipo de Sistema Manejador de Base de Datos Orientado a Objetos.** SMCC. Tercer Encuentro Internacional de Computación ENC'01. Septiembre, 2001. Aguascalientes, México.
- Fernández y Fernández, Carlos Alberto; Pedro Luis Martínez Guzmán; Reyes Matamoros Agustín Manuel; Rodríguez Jiménez Noé. **Suite Única de Transacciones Electrónicas (S.U.T.E).** México. Memorias del XI Simposium Nacional de Informática. 2000.

- Héctor Gabriel Acosta Mesa; Fernández y Fernández, Carlos Alberto. **Propuesta de un Ambiente para el Modelado de Sociedades de Agentes Reactivos.** México. Memorias del XI Simposium Nacional de Informática. 2000.
- Alma Rosa García Gaona; Armando Pérez Flores; Epifanía Marcial Sánchez; Carlos Alberto Fernández y Fernández; Suhey Villalobos Hernández; Mayra Velasco Landa; Karina Duran García. **Un Prototipo De Sistema Manejador De Base De Datos Orientado A Objetos. (SMBDDOO_UV).** SMCC. Primer Encuentro de Computación ENC ’97. Septiembre, 1997. Querétaro, México.

COMPUTADORAS Y PROGRAMACIÓN

http://lcp02.fi-b.unam.mx/asignatura/asigna.html

Programa de la Asignatura: Computadoras y Programación.

Objetivo del Curso

El alumno adquirirá una cultura informática básica, y conocerá las técnicas y procedimientos de análisis y diseño de programas para utilizar la computadora digital como herramienta en la solución de problemas relacionados con la ingeniería.

Temas de la materia

NUM	TEMAS DE LA MATERIA:	HORAS
I.	**INTRODUCCION A LA COMPUTACION**	**7.5**
II.	**REPRESENTACION INTERNA DE LA INFORMACION**	**7.5**
III.	**METODOLOGIA DE LA PROGRAMACION ESTRUCTURADA**	**25**
******	**PRACTICAS DE LABORATORIO**	**32**
*********	**********	**TOTAL 72**

Clave:1206

Núm de Créditos: 07

Duración del Curso: 16 semanas (72 horas)

Carrera: ICi, ICo, IEe, IGf, IGI, IIn, IMe, IMm, IPe, Ite, ITg.

Semestre: 2°

Obligatoria: Sí.

Teoría: 2.5

Prácticas: 2

Antecedentes, objetivos y contenidos de los temas

I. Introducción a la computación TEMAS

Objetivo:

El alumno conocerá la evolución de las computadoras y el software; asimismo, podrá describir en lo general los elementos de las configuraciones actuales de las computadoras, los lenguajes que emplean y las interacción entre sus componentes.

Contenido:

- I.1 Historia de la computación
    - I.1.1 La plataforma de arranque.

- I.1.2 Desarrollo de los equipos de cálculo.
- I.1.3 Las primeras computadoras.
  - I.1.3.1 Computadora analógica.
  - I.1.3.2 Computadora digital.
  - I.1.3.3 Computadora híbrida.
- I.1.4 Primera generación de computadoras.
- I.1.5 Segunda generación de computadoras.
- I.1.6 Tercera generación de computadoras.
- I.1.7 Cuarta generación de computadoras.
  - I.1.7.1 Historia de las microcomputadoras.
  - I.1.7.2 Configuración de sistemas de cómputo.
    - I.1.7.2.1 Arquitectura de computadoras: CISC y RISC.
    - I.1.7.2.2 Configuración de Redes.
      - I.1.7.2.2.1 Tipos de Redes (LAN, MAN, WAN).
      - I.1.7.2.2.2 Internet.
- I.1.8 Quinta generación de computadoras.
- I.1.9 Historia panorámica de la computación en México.

        - I.1.9.1 Antecedentes.
        - I.1.9.2 La primera computadora en el País.
        - I.1.9.3 Desarrollo de la computación en el País.

- 1.2 Panorama de la historia del software.
    - I.2.1 Antecedentes.
    - I.2.2 Conceptos generales.
    - I.2.3 Clasificación del software:
        - I.2.3.1 Software de sistemas:
            - I.2.3.1.1 Historia de los sistemas operativos.
            - I.2.3.1.2 Sistema operativos para microcomputadoras.
            - I.2.3.1.3 Software de comunicaciones.
        - I.2.3.2 Lenguajes de Programación.
            - I.2.3.2.1 Lenguajes de bajo nivel.
            - I.2.3.2.2 Lenguajes de alto nivel.
            - I.2.3.2.3 Lenguajes orientados a procedimientos.
            - I.2.3.2.4 Lenguajes orientados a objetos.

II. Representación interna de la información. TEMAS

Objetivo:

El alumno conocerá cómo se representa la información internamente en la computadora para su proceso, los sistemas de numeración posicional, los códigos de codificación de mayor aceptación y los errores que los diferentes tipos de representación generan.

Contenido:

- II.1 Definición de un sistema de numeración posicional.
- II.2 Conversión entre sistemas de numeración posicional (decimal, binario, octal, y hexadecimal).
- II.3 Operaciones básicas entre binario, octal y decimal.
- II.4 Unidades de medida de la información: bit, byte, palabra de computadora y códigos más empleados para la representación de la información (ASCII y EBCDIC).
- II.5 Representación numérica: magnitud y signo, complemento a dos para números enteros y reales.
- II.6 Tipos de errores que se presentan en la manipulación de cantidades.

III. Metodología de la programación estructurada. TEMAS

Objetivo:

El alumno conocerá y aplicará el concepto de algoritmo, las técnicas y herramientas de la programación estructurada para la solución de problemas y su programación en lenguaje 'C' (Visual C).

Contenido:

- III.1 Diagramas de flujo.
    - III.1.1 Conceptos generales:
        - III.1.1.1 Conceptos de algoritmo y programa.
        - III.1.1.2 Algoritmo no numérico y algoritmo numérico.
    - III.1.2 Elementos de los algoritmos:
        - III.1.2.1 Operadores.
        - III.1.2.2 Declaraciones.
        - III.1.2.3 Constantes.
        - III.1.2.4 Variables.
        - III.1.2.5 Instrucción de reemplazo o concepto de almacén.
        - III.1.2.6 Funciones de biblioteca.

- III.1.3 Simbología básica de los diagramas de flujo.
- III.1.4 El ciclo iterativo.
- III.1.5 Aplicaciones del ciclo iterativo:
    - III.1.5.1 Primera aplicación: el concepto de contador.
    - III.1.5.2 Segunda aplicación: el concepto de sumador (sumatorias).
    - III.1.5.3 Tercera aplicación: el concepto de multiplicación reiterada (factoriales).
- III.1.6 Diagramación estructurada:
    - III.1.6.1 Elementos estructurados de repetición (ciclos):
        - III.1.6.1.1 Ciclo iterativo controlado por un controlador (FOR).
        - III.1.6.1.2 Ciclos anidados(con contador incluido).
        - III.1.6.1.3 Ciclos iterativos controlados por condición:
        - III.1.6.1.3.1 Ciclo con condición al inicio (ciclo WHILE o mientras).

- III.3.1 Orígen y propósito del languaje C.
- III.3.2 Características del lenguaje C. Módulos a través de funciones.
- III.3.3 Configuración de un programa:
    - III.3.3.1 Declaraciones.
    - III.3.3.2 Instrucciones ejecutables.
- III.3.4 Directivas de preprocesamiento.
    - III.3.4.1 Las directivas include y define
- III.3.5 Tipos de declaraciones de variables y su ámbito de acción para el diseño modular o estructurado de programas.
- III.3.6 Elementos de programación del lenguaje C:
    - III.3.6.1 Constantes de C.
    - III.3.6.2 Variables:
        - III.3.6.2.1 Variables enteras.
        - III.3.6.2.2 Variables reales.
        - III.3.6.2.3 Variables con índice(arreglos).
        - III.3.6.2.4 Variables alfanuméricas o de cadena.

- III.3.6.2.5 Variables globales, automáticas, estáticas y register.
- III.3.6.2.6 Operadores aritméticos (+, -, *, /, %).

- III.3.7 Funciones.
  - III.3.7.1 Conceptos básicos de funciones.
  - III.3.7.2 Tipos de funciones y la proposición return.
  - III.3.7.3 Variables locales y globales.
  - III.3.7.4 Funciones de biblioteca.
- III.3.8 Codificación de un primer programa.
- III.3.9 Tipos de datos.
  - III.3.9.1 Modificadores unsigned y long.
  - III.3.9.2 Operadores SIZEof.
  - III.3.9.3 Formatos de entrada-salida.
  - III.3.9.4 Conversión de tipos Implícita y explícita (CAST).
- III.3.10 Operadores.
  - III.3.10.1 Operadores de incremento y decremento.

- III.3.10.2 Operadores de asignación y expresiones.
- III.3.10.3 Operadores de relación y lógicos.
- III.3.10.4 Operador de coma (,).
- III.3.10.5 Operadores de manejo de bits.

- III.3.11 Elementos estructurados de selección.
  - III.3.11.1 Elemento estructurado if-else, encadenamientos y expresiones de condición.
  - III.3.11.2 Operador condición (?).
  - III.3.11.3 Elemento estructurado de ramificación múltiple switch.
- III.3.12 Elementos estructurados de repetición.
  - III.3.12.1 Elementos estructurados while y for.
  - III.3.12.2 Elementos estructurados do-while.
- III.3.13 Apuntadores y arreglos.
  - III.3.13.1 Concepto de apuntador y direcciones.

- III.3.13.2 Argumentos a funciones por referencia.
- III.3.13.3 Arreglos unidimensionales numéricos y aritmética de apuntadores.
- III.3.13.4 Arreglos bidimensionales.
- III.3.13.5 Arreglos de caractéres (cadenas).

- III.3.14 Estructuras.
  - III.3.14.1 Conceptos básicos de estructuras.
  - III.3.14.2 Estructuras y funciones.
  - III.3.14.3 Uso de typedef.
- III.3.15 Argumentos al programa principal.
- III.3.16 Archivos.
  - III.3.16.1 Proposiciones de entrada-salida para archivos.
  - III.3.16.2 Proposición fopen.
  - III.3.16.3 Proposición close.
- III.3.17 Interface de C con ambientes WINDOWS MS, XWINDOWS

- III.3.17.1 Generación de ventanas y menúes.
- III.3.17.2 Adquisición y presentación de datos.

Prácticas del laboratorio. TEMAS

Objetivo:

El alumno conocerá las herramientas básicas para utilizar una computadora como son: un sistema operativo en ambiente gráfico que le permita desarrollar aplicaciones en el lenguaje C, un navegador para Internet, una hoja de cálculo electrónica y un paquete matemático.

Contenido:

- I. SISTEMA OPERATIVO: WINDOWS MS O XWINDOWS UNIX. (1.5 hrs)
- III. HOJA ELECTRÓNICA DE CÁLCULO. (3.5 hrs)
- IV. PAQUETE MATEMÁTICO. (4.5 hrs)
- V. USO DE CORREO ELECTRÓNICO Y NAVEGADOR PARA INTERNET (4.5 hrs)

ARGENTINA

UNIVERSIDAD TECNOLÓGICA NACIONAL

FACULTAD REGIONAL SANTA FE

http://www.guiafe.com.ar/sitioaedd/pagina.php3?pagina=guiasdeestudio.php3&descripcion=Guías%20de%20Estudio

"ALGORITMOS Y ESTRUCTURAS DE DATOS"

INGENIERÍA EN SISTEMAS DE INFORMACIÓN

RESENTACIÓN

La asignatura que antecede a esta dentro del área Programación (MATEMÁTICA DISCRETA -MAD), se ha introducido el concepto de Algoritmo (como un proceso o método paso a paso para resolver un problema) y se han estudiado algunas Estructuras Matemáticas (grafos, árboles) como una colección de elementos con relaciones o vinculaciones que permiten realizar.

Con referencia a los algoritmos, no sólo se presentaron sus características esenciales, sino que además se dieron algunas formas de especificación de algoritmos y se incluyeron algoritmos solución para resolver cuestiones en distintos temas de la materia (Ejemplo: reconocer si una matriz es simétrica, saber si un nodo de un grafo es aislado, recorrer un árbol con el criterio de inorden, etc.).

Con respecto a las estructuras, además de definirlas, se dieron formas de caracterizarlas (Ejemplo: un grafo se caracteriza conociendo el conjunto de nodos y el conjunto de arcos orientados que unen los nodos, pero también puede definirse como un conjunto de nodos y una matriz booleana que describe las vinculaciones entre esos nodos). También se analizaron las operaciones básicas (Ejemplo: determinar si hay un camino entre dos nodos, reconocer los nodos aislados, etc.).

Al analizar las aplicaciones, se estudió que dichas estructuras son modelos matemáticos que pueden emplearse para representar diferentes situaciones reales (Ejemplo: un grafo puede representar rutas entre ciudades, secuencias de tareas en un proyecto, pasos o acciones de un algoritmos, etc.).

En esta asignatura (ALGORITMOS Y ESTRUCTURAS DE DATOS – AED), estos conceptos (modelos y abstracciones matemáticas ya vistas) serán tomados como base para abocarnos al estudio de Algoritmos Computacionales (Abordaremos problemas que van a resolverse empleando computadoras) y de Estructuras de Datos (Modelos para representar datos -información- que serán tratados –almacenados, manipulados- por computadoras).

El objetivo de esta asignatura es desarrollar una Introducción al Diseño de Algoritmos y al Manejo de Estructuras de Datos simples.

Al decir Introducción al diseño de algoritmos, nos referimos a presentar, analizar y aplicar conceptos, metodologías y técnicas básicas, asociadas a la construcción de algoritmos eficaces (es decir que resuelvan los problemas de acuerdo a los requerimientos que se han especificado) y teniendo en cuenta aspectos de eficiencia (que lo resuelvan de la mejor forma posible, es decir considerando el uso de recursos como uso de memoria, tiempo de proceso, etc.)

Cuando indicamos Manejo de Estructuras de Datos simples, nos referimos al tratamiento de información con vinculaciones sencillas (tablas, listas, etc.), que en general se consideran como estructuras básicas, porque se aplican para resolver una amplia variedad de problemas simples y además sirven de base para estructuras de datos más complejas , como por ejemplo las bases de datos que se estudiarán en asignaturas siguientes.

Tal como se expresó en un párrafo anterior, hablamos de información (estructuras) y procesos (algoritmos) que serán tratados por una computadora. Es de conocimiento general que para que una computadora lleve a cabo un proceso, es necesario definir dicho proceso (diseñarlo) y además expresarlo de una manera (lenguaje) que la computadora comprenda.

Cuando nos referimos a procesos, no estamos hablando de acciones operativas para que una máquina se ponga en funcionamiento, o nos permita emplear sus utilitarios y herramientas, sino que, como se ha dicho, nos centramos en procesos que tienen como finalidad resolver un problema determinado, manejando la información concerniente.

Es ampliamente reconocido y difundido el criterio que, para resolver un problema con una computadora, primero hay que saber resolver el problema. Es decir, lo esencial es poder comprenderlo, seleccionar o definir una estrategia de solución y diseñar el proceso, por lo que la expresión del algoritmo en un lenguaje que la máquina comprenda (Lenguaje de Programación) pasa a ser una etapa posterior.

No obstante, existe una relación entre los patrones bajo los cuales se encuadre el modelado del problema a resolver (Paradigma de programación) , las métodos que se determinen para resolverlo y los lenguajes de programación que se dispongan.

Por otra parte desde el punto de vista educativo y formativo, se entiende que es conveniente poder concretar la resolución de los problemas tratados, es decir poder construir los Programas correspondientes y ejecutarlos en una computadora.

Acompañaremos el estudio de diseño de algoritmos y tratamiento de información a través de estructuras de datos, con el conocimiento elemental de un par de lenguajes de programación: muLisp y Pascal y sus correspondientes entornos para poder ingresar los programas y llegar a su ejecución (incluyendo la detección y depuración de errores)

Tabla de Contenidos

3. Introducción a los Algoritmos y Programas

3.1 Algoritmos

3.2 Algoritmos computacionales

3.3 Sintaxis y semántica de un lenguaje de programación

3.4 Paradigmas de programación

3.5 Fases en el desarrollo de un algoritmo

3.6 Corrección y eficiencia de algoritmos

3.7 Diseño de algoritmos

3.8 El estilo en la algorítmica

4. Diseño de Programas

4.1 Análisis del problema

4.2 Diseño y verificación de algoritmos

4.3 Codificación

4.4 Compilación y ejecución

4.5 Verificación y depuración

4.6 Documentación

5. Programación Modular

MATERIAL DE ESTUDIO

BIBLIOGRAFÍA	6. GALVE-GONZALEZ y otros. Algorítmica (Diseño y Análisis de algoritmos funcionales e imperativos) Ed. Ra-ma. (Serie Paradigma) /Madrid/1993. 7. JOYANES AGUILAR. Pascal y Turbo Pascal (Enfoque Práctico). Ed. Mc. Graw Hill/1994.
LECTURAS OBLIGATORIAS	8. Documentación de la Cátedra.
LECTURAS DE APOYO	9. GHEZZI, CARLO y Mehdi Jazayeri. Conceptos de lenguajes de programación. Ediciones Diaz de Santos, S.A. 1986. Capítulo 1. Introducción.

▲ i

E

APLICACIONES DE LA METODOLOGÍA PARA LA ALGORITMIA ORIENTADA A OBJETOS

Este anexo presenta tres problemas básicos solucionados con la metodología propuesta. Como se verá, las soluciones planteadas están relacionadas, dado

que se aplican los mecanismos de herencia propios del paradigma de programación orientado a objetos.

El primer ejemplo requiere de un objeto de la clase **Empleado**, el segundo necesita de **n** objetos de la misma clase, y el tercero utiliza una cantidad desconocida de objetos de la clase derivada **EmpleadoADestajo**, cuya clase base es **Empleado**.

Los casos donde la metodología para la algoritmia orientada a objetos pueda aplicarse, son tan variados como aquellos donde la algoritmia imperativa también lo sea. Este documento es una referencia para la MAOO y no un manual de aplicaciones, por tanto se omiten otras situaciones problemáticas interesantes, tan extensas que seria material para otra publicación.

Los ejemplos demuestran que la metodología es de fácil aplicación, acorde a la teoría de la orientación a objetos. No se pretende abordar situaciones complejas y/o extensas. Sólo se demostrará que la metodología es de fácil aplicación.

1. PROCESAMIENTO DE UN EMPLEADO

Obtener el salario neto de un empleado **x**, sobre el cual se conoce su cédula, nombre, número de horas trabajadas, valor hora, deducciones y bonificaciones.

Solución

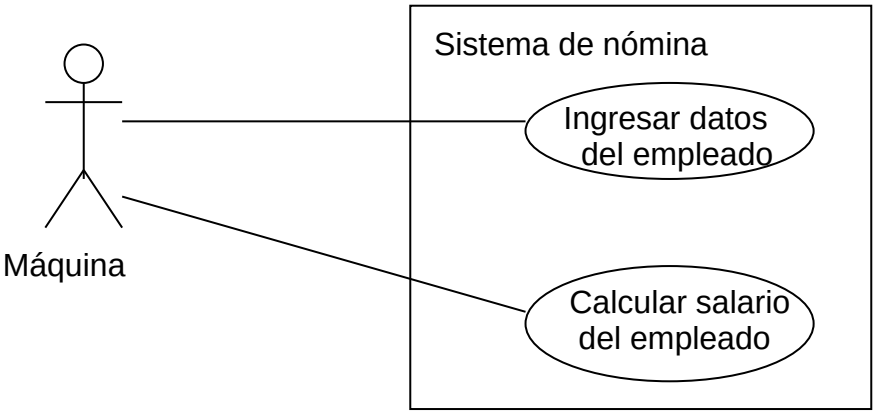
Identificación y delimitación del problema a resolver

El enunciado es corto y trivial. Su interpretación no admite ambigüedades, como podría ocurrir en casos complejos o extensos. El problema está bien delimitado: sólo se procesará un empleado.

Análisis del problema

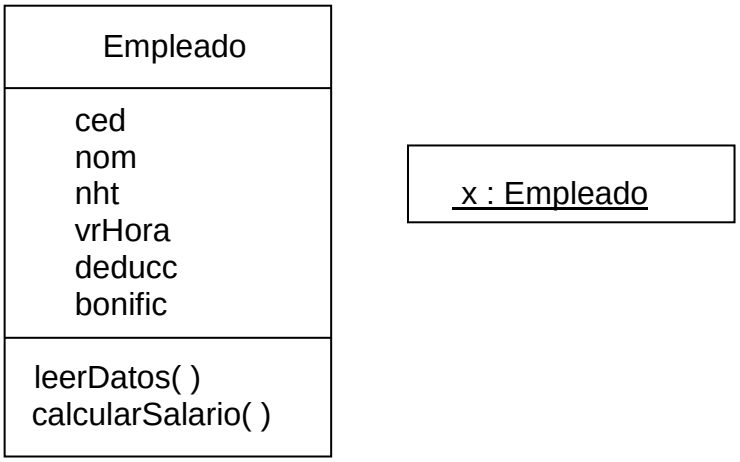
Identificación de clases

Se presentan los casos de uso *Ingresar datos* y *Calcular salario neto*, representados en UML así:



Los casos de uso permiten identificar una clase: Empleado

Se requiere solo de una instancia de clase para dar solución a la situación planteada. La instancia de clase u objeto se denominará *x*, y la clase asociada será Empleado. La representación UML es la siguiente.



Como se observa, la clase *Empleado* tiene un total de seis miembros dato y dos operaciones: leerDatos( ) captura los datos de un empleado desde un dispositivo de entrada estándar como un teclado; calcularSalario( ) calcula el salario neto de un empleado de acuerdo a los datos de entrada ya conocidos.

Documentación de los miembros dato de la clase:

ced: cédula del empleado. Tipo cadena.

nom: nombre del empleado. Tipo cadena.

nht: número de horas trabajadas. Tipo entero.

vrHora: Valor devengado por hora. Tipo real o de punto flotante.

deducc: deducciones. Tipo real.

bonific: bonificaciones. Tipo real.

Desarrollo de la solución para cada operación

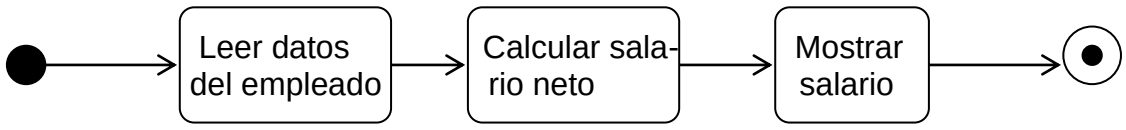
Modelación matemática

Para este caso se requiere de un cálculo simple:

sNeto = (nht * vrHora) + bonific - deducc

Identificación de entradas y salidas de datos

Las entradas y salidas se pueden ilustrar con un diagrama de actividades de UML.



Por tanto, los datos de entrada están dados por los miembros dato de la clase Empleado. La información de salida es única: elsalario neto (sNeto) devengado por el empleado, valor de tipo real.

Desarrollo del algoritmo

Se debe calcular el salario neto de un empleado, por tanto sólo se requiere de un objeto denominado x, tal como se especificó anteriormente.

Algoritmo principal:

Inicio
Empleado x
x.leerDatos()
sNeto = x.calcularSalario()
Imprimir x.ced, x.nom, sNeto
Retorne

Operaciones:

Empleado::leerDatos()
Leer ced, nom, nht, vrHora, bonific, deducc
Retorne

Empleado::calcularSalario()
salario = (nht * vrHora) + bonific - deducc
Retorne (salario)

Nota:

En el método calcularSalario() y en el programa principal se han utilizado dos símbolos propios del lenguaje C++: el operador de resolución de ámbito, denotado por **::**, y el operador de asignación **=**. Esta elección se debe a la aceptación del lenguaje en los ámbitos comercial y científico, aunque se podría haber elegido cualquier otro conjunto de operadores.

2. PROCESAMIENTO DE N EMPLEADOS

Calcular e imprimir el salario neto para una cantidad de **n** empleados. Hallar además los totales pagados por concepto de bonificaciones y recaudados por concepto de retenciones. Por cada empleado se conoce su cédula, nombre, número de horas trabajadas, valor hora, deducciones y bonificaciones

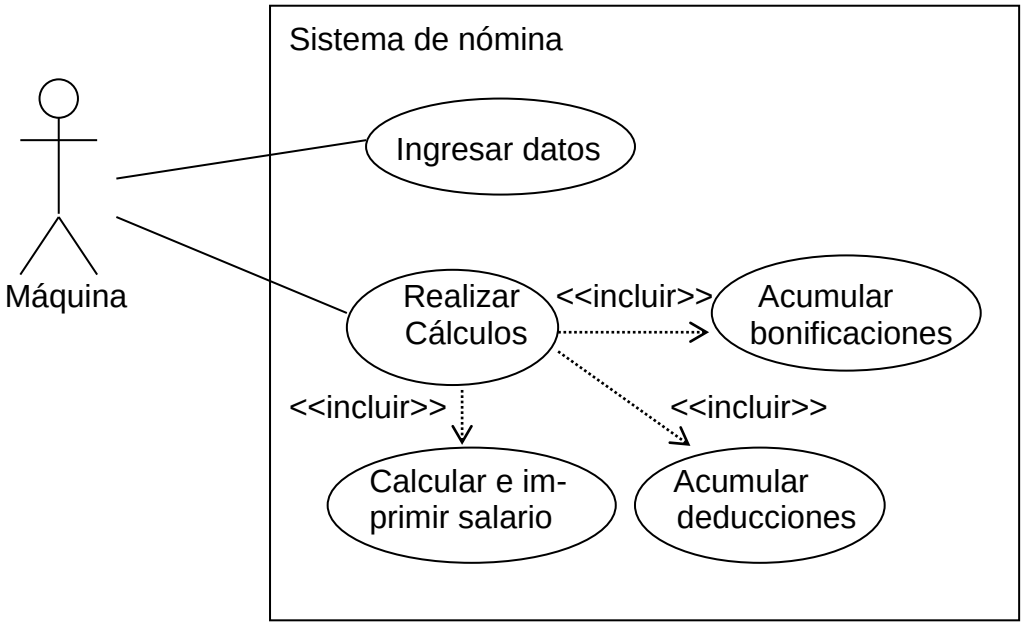
Solución

Identificación del problema a resolver

El problema consiste en el cálculo del salario neto para una cantidad conocida de empleados. No admite ambigüedad y está bien delimitado.

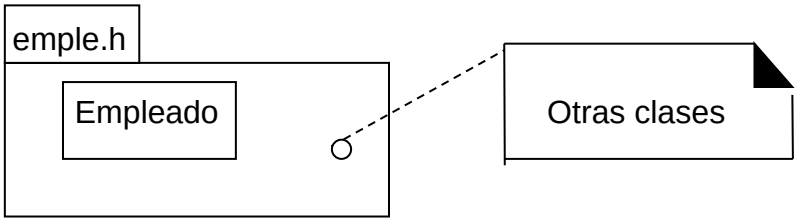
Análisis del problema

El enunciado induce al siguiente diagrama de casos de uso.



Identificación de clases

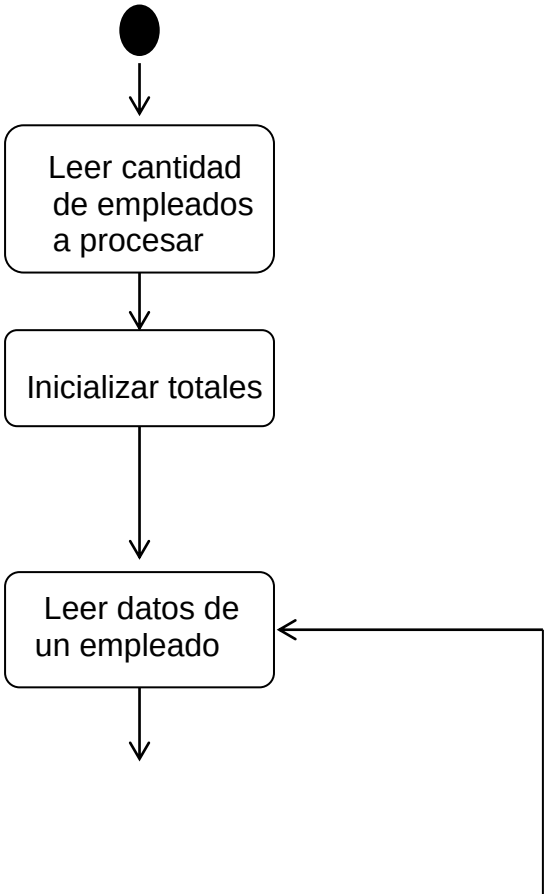
Se reutilizará la clase anterior, es decir, se supone que la clase Empleado se halla almacenada en la librería emple.h; la extensión .h viene del inglés *header*, y se utiliza para denotar a los archivos de cabecera en lenguajes como C y C++.

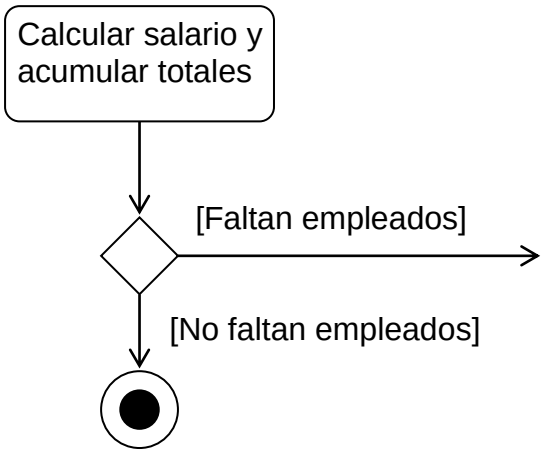


Desarrollo de la solución

Identificación de entradas y salidas de datos:

Las entradas de datos y salidas de información se resumen en el siguiente diagrama de actividades.





Por tanto:

- Datos de entrada (encapsulados en la clase Empleado):

  ced: cédula del empleado. Tipo cadena.

  nom: nombre del empleado. Tipo cadena.

  nht: número de horas trabajadas. Tipo entero.

  vrHora: Valor devengado por hora. Tipo real o de punto flotante.

  deducc: deducciones. Tipo real.

  bonific: bonificaciones. Tipo real.

- Datos de salida

  Por cada empleado:

  sNeto: salario neto devengado por el empleado. Tipo real.

  En general:

  totBonific: total bonificaciones. Tipo real.

  totDeducc: total deducciones. Tipo real.

Definición de estructuras de datos

Los datos de los empleados serán almacenados en un vector de objetosde clase Empleado, denominado *vec*.

vec[ ] : Empleado

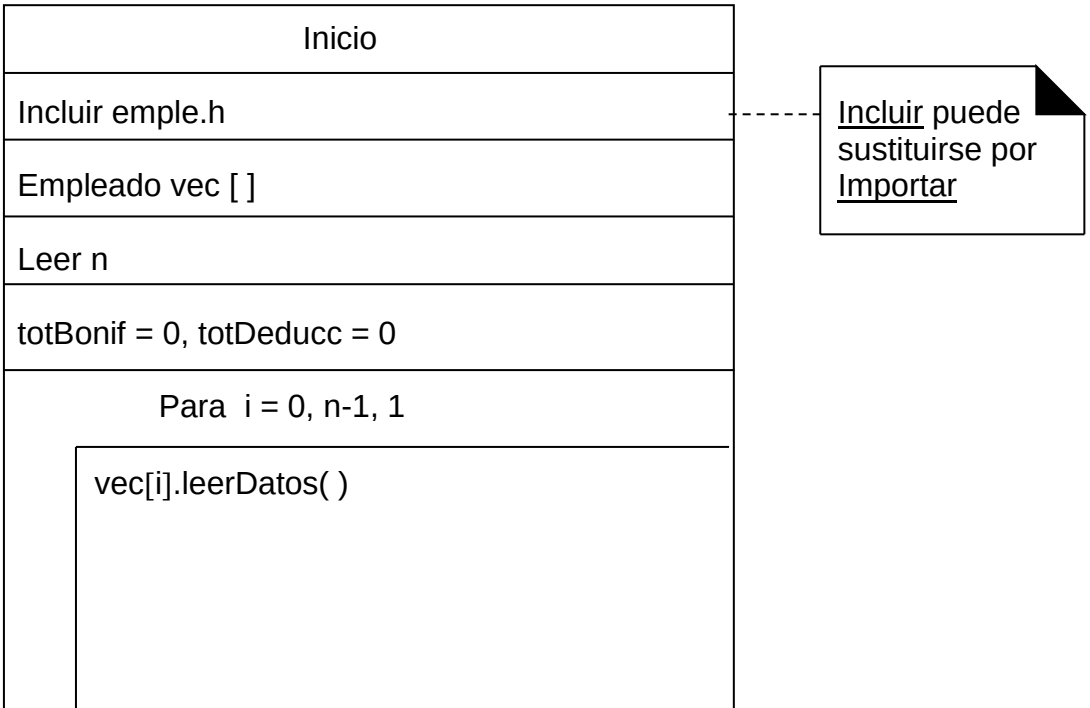
Modelación matemática

La reutilización de la clase empleado conlleva el cálculo implícito en la operación calcularSalario():

sNeto = (nht * vrHora) + bonific – deducc

Desarrollo del algoritmo

Tal como se observará en el algoritmo, antes de cualquier instrucción ejecutable se debe incluir la librería emple.h. Luego:



```
        sNeto = vec[i].calcularSalario( )
        totBonif = totBonif + vec[i].bonific
        totDeducc = totDeducc + vec[i].deducc
        Imprimir vec[i].ced, vec[i].nom, sNeto
Imprimir totBonif, totDeducc
Fin
```

3. PROCESAMIENTO DE UNA CANTIDAD DESCONOCIDA DE EMPLEADOS A DESTAJO

En cierta empresa los trabajadores son contratados por horas y adicionalmente se les paga un rubro ***r*** por la fabricación de una pieza de un producto determinado. De cada empleado se conoce su cédula, nombre, número de horas trabajadas, valor pagado por hora, bonificaciones, deducciones y cantidad de piezas fabricadas. Calcular e imprimir:

- Salario neto de cada empleado.
- Total pagado a todos los empleados por las piezas fabricadas.
- Los empleados que devengan el mayor salario neto.

Se desconoce la cantidad de empleados a procesar.

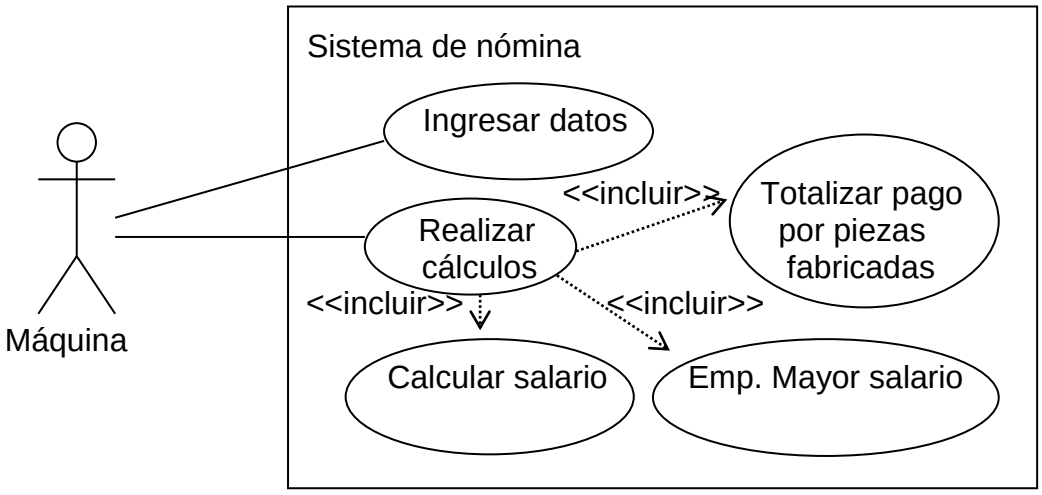
Solución

Identificación del problema

Según la situación expuesta, se debe tener en cuenta que no se conoce la cantidad de empleados a procesar (uno de los aspectos de la delimitación del problema); además, al interpretarla, se debe comprender que se trata del procesamiento de empleados que laboran a destajo.

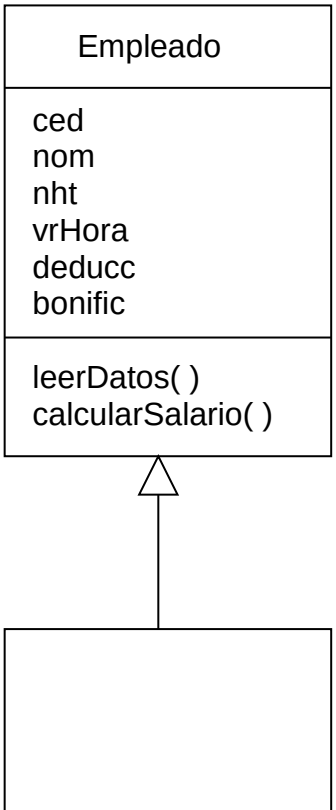
Análisis del problema

El uso del sistema se resume según el siguiente diagrama:



Identificación de clases

Se define la clase **EmpleadoADestajo** a partir de la clase **Empleado**, previamente definida en el ejercicio 1; esto significa que **EmpleadoADestajo** es una subclase de **Empleado**, o que esta última es una superclase o clase base para la primera.



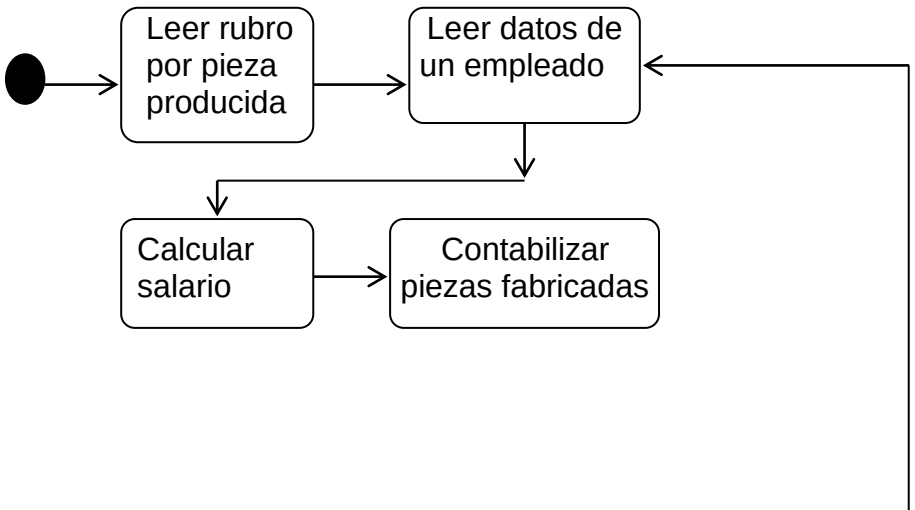
EmpleadoADestajo
cantPF liga
leerDatos( ) calcularSalario( )

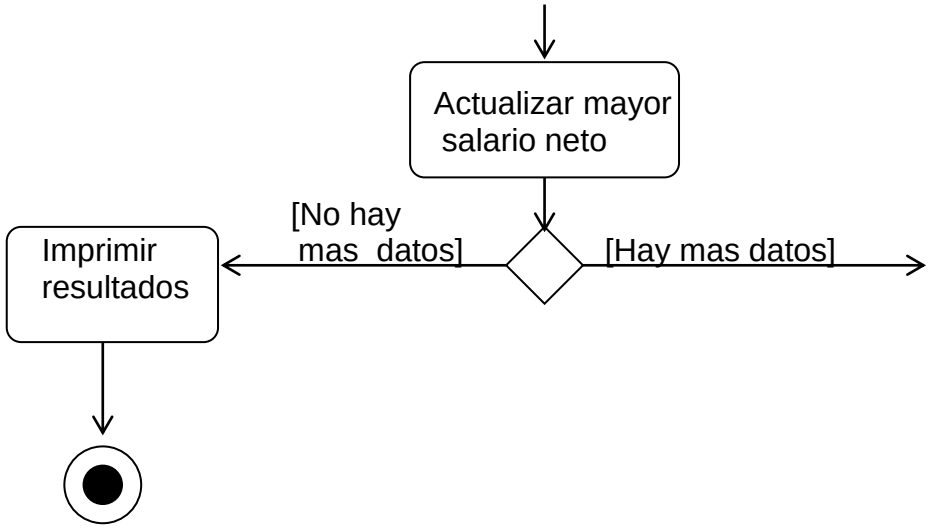
La clase EmpleadoADestajo tiene dos nuevos miembros dato: cantPF –cantidad de piezas producidas– y liga –puntero a un objeto tipo EmpleadoADestajo–; las operaciones leerDatos() y calcularSalario() de la subclase EmpleadoADestajo deben ser redefinidas, es decir, sobrecargadas, aplicando así uno de los conceptos importantes de la orientación a objetos: el polimorfismo.

Desarrollo de la solución para cada operación

Identificación de entradas y salidas

Las entradas de datos y salidas de información se visualizan en el siguiente diagrama de actividades.





Luego:

- Datos de entrada

  En general:

  r: rubro que se paga por pieza fabricada. Tipo real.

  Por cada empleado:

  ced: cédula del empleado. Tipo cadena.

  nom: nombre del empleado. Tipo cadena.

  nht: número de horas trabajadas. Tipo entero.

  vrHora: Valor devengado por hora. Tipo real o de punto flotante.

  deducc: deducciones. Tipo real.

  bonific: bonificaciones. Tipo real.

  cantPF: cantidad de piezas fabricadas. Tipo entero.

- Datos de salida

  Por cada empleado:

sNeto: salario neto devengado por el empleado. Tipo real.

En general:

totPiezasFab: total pagado por piezas fabricadas. Tipo real.

Las cédulas de los empleados que devengan el mayor salario neto.

Modelación matemática

El cálculo a realizar para determinar el salario neto de cada empleado es:

sNeto = (nht * vrHora) + bonific – deducc + (cantPF * r)

Definición de constantes, variables y objetos

En el algoritmo se utilizan las siguientes variables, además de las ya indicadas como datos de entrada y de salida.

c*pf* : contador de piezas fabricadas. Tipo entero.

mayor : salario neto mayor. Tipo real.

cab: puntero al nodo cabeza de la lista liga da de empleados. Tipo puntero a un registro con la estructura especificada a continuación.

ced	*nom*	*nht*	*vrHora*	*bonific*	*deducc*	*cantPF*	*liga*

donde *liga* es otro puntero al siguiente nodo de la lista ligada; los demás campos del nodo ya fueron descritos anteriormente.

p: puntero a un nodo cualquiera de la lista enlazada.

u: puntero al último nodo de la lista actual.

resp: respuesta a la pregunta “¿Existen más empleados? S/N”.

Tipo carácter.

Definición de estructuras de datos

La solución de este problema implica el uso de una estructura de datos, preferiblemente lineal. Se utilizará una lista simplemente ligada de objetos tipo **EmpleadoADestajo**.

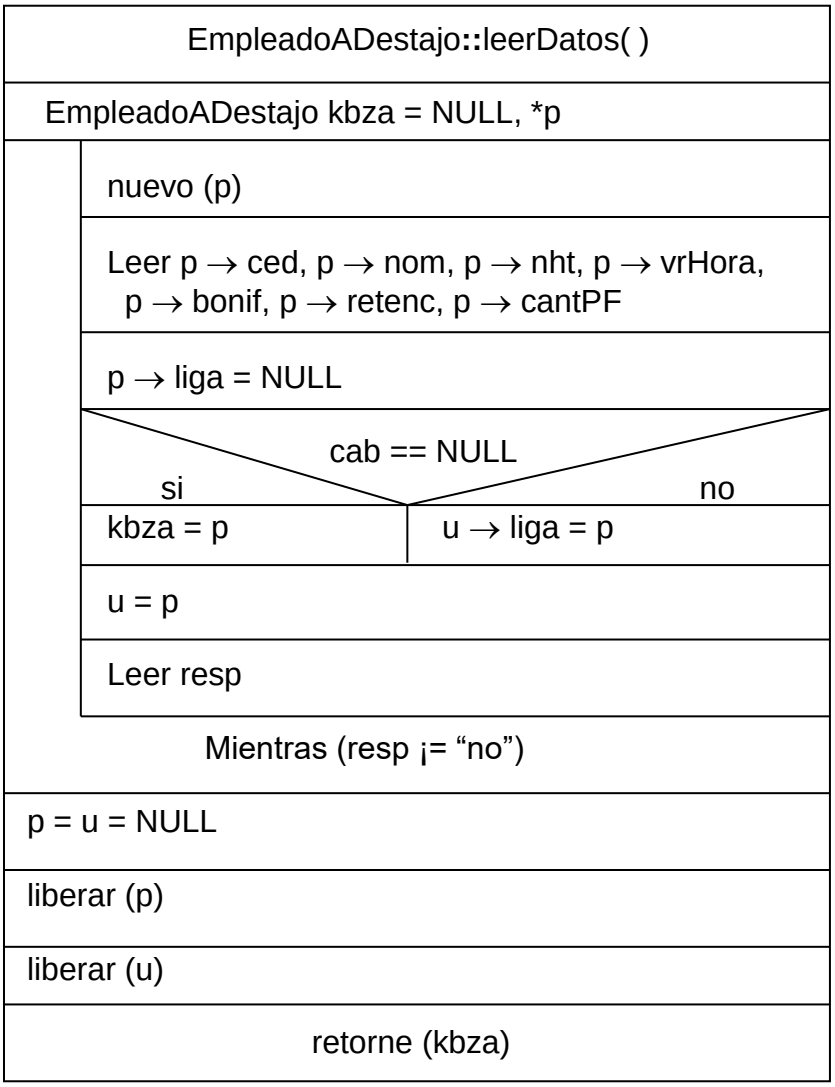
Algoritmos

Se debe definir un conjunto de objetos tipo **EmpleadoADestajo** y almacenarlos en una estructura de datos preferiblemente lineal y dinámica como una lista ligada.

Ahora se solucionará el problema mediante una lista ligada de objetos tipo **EmpleadoADestajo**, teniendo en cuenta que esta clase ahora se encuentra empaquetada en el archivo **emple.h**.

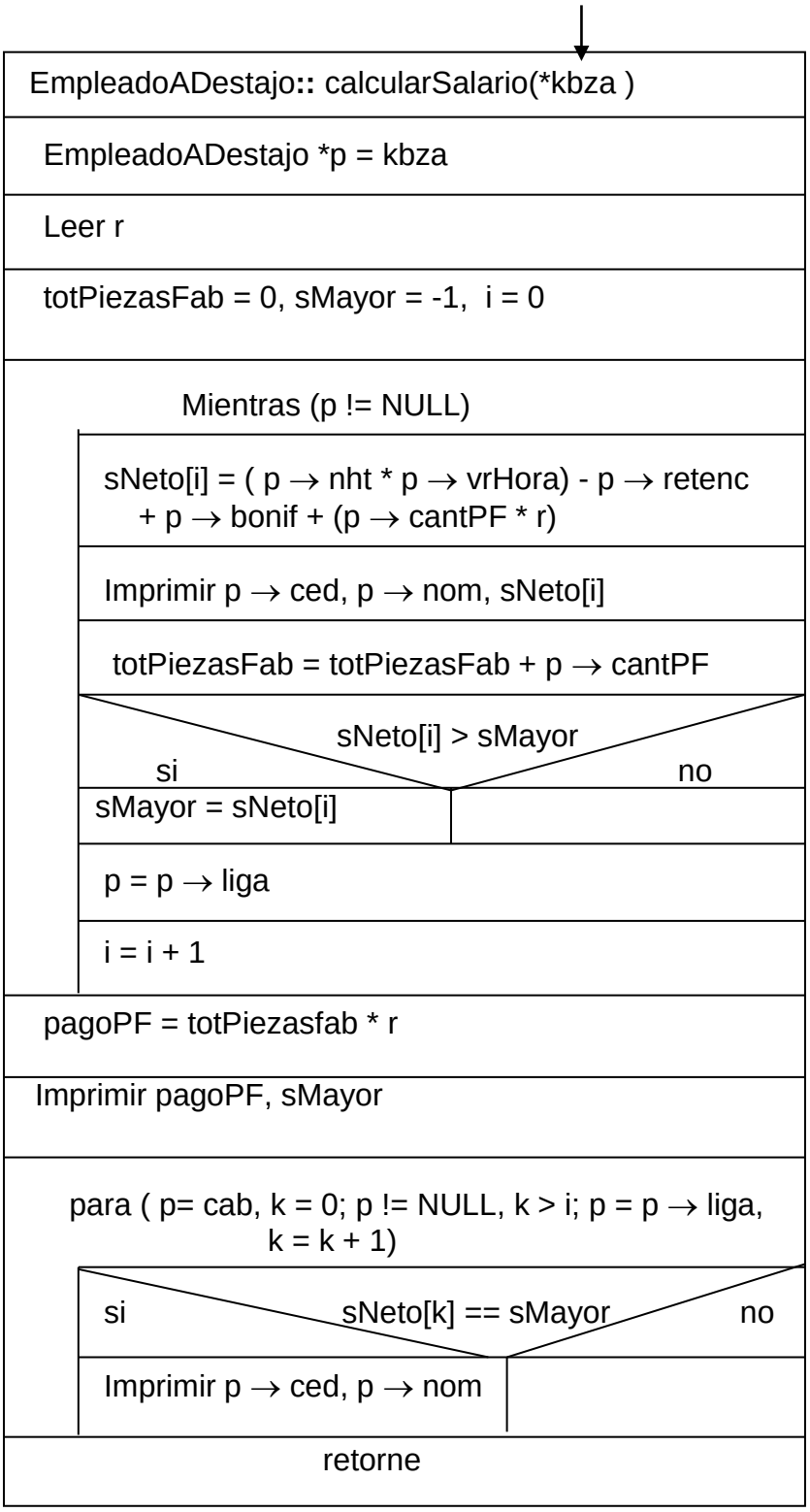
Inicio
Importar emple.h
EmpleadoADestajo *cab
cab = leerDatos( )
calcularSalario(cab)
Fin

El método leerDatos( ) crea la lista ligada de empleados a destajo añadiendo los nuevos nodos al final de la lista y devuelve un puntero al primer objeto o nodo de la lista. El método calcularSalario( ) procesa a cada empleado de la lista y cumple con los requerimientos solicitados en el enunciado.



Notas:

- **nuevo( )** y **liberar( )** son macroinstrucciones para generar y liberar memoria en tiempo de ejecución y equivalen en C++ a **new( )** y **delete( )**. Es de observar que en el lenguaje Java la liberación de memoria es automática, realizada por el recolector de basura (garbage collector).
- Las variables *kbza*, *p* y *u* son punteros a nodos de la lista. El apuntador *kbza* no debe ser liberado, porque es el que precisamente contiene la dirección del primer nodo y debe ser devuelto al programa principal.


EmpleadoADestajo:: calcularSalario(*kbza)
EmpleadoADestajo *p = kbza
Leer r
totPiezasFab = 0, sMayor = -1, i = 0
Mientras (p != NULL)
sNeto[i] = (p → nht * p → vrHora) - p → retenc
+ p → bonif + (p → cantPF * r)
Imprimir p → ced, p → nom, sNeto[i]
totPiezasFab = totPiezasFab + p → cantPF
sNeto[i] > sMayor
si
no
sMayor = sNeto[i]
p = p → liga
i = i + 1
pagoPF = totPiezasfab * r
Imprimir pagoPF, sMayor
para (p= cab, k = 0; p != NULL, k > i; p = p → liga,
k = k + 1)
si
sNeto[k] == sMayor
no
Imprimir p → ced, p → nom
retorne

Además de los miembros dato de la clase **EmpleadoADestajo**, en el algoritmo anterior se han utilizado las siguientes variables y estructuras de datos locales:

p: puntero al primer nodo u objeto de la lista de empleados a destajo.

r: rubro pagado por cada pieza fabricada. Tipo numérico flotante.

totPiezasFab: total de piezas fabricadas por todos los empleados. Tipo numérico entero.

sMayor: salario mayor devengado. Tipo numérico flotante.

i: contador de empleados. A la vez sirve como índice para recorrer el vector. . Tipo numérico entero.

SNeto*[ ]*: arreglo unidimensional tipo entero que almacena los salarios netos de los empleados.

pagoPF: pago total por piezas fabricadas. Tipo numérico flotante.

k: contador para recorrer el vector *sNeto[ ].* Tipo numérico entero. ▲ i

F

RELATORÍAS*

* Las relatorías son ensayos relacionados con algunos temas tratados en el transcurso de la Especialización, aplicados –si es necesario– al proyecto de aula MAOO. ▲[i]

F.1. EDUCACIÓN SIGLO XXI: PEDAGODÍA, INVESTICACIÓN Y PROYECTO DE AULA

CONTEXTUALIZACIÓN

El pertinaz desarrollo científico evidenciado en las últimas décadas, originario de la denominada *tercera revolución industrial* y plasmado en el surgimiento de nuevas ciencias (inteligencia artificial, ingeniería genética, teleinfomática, entre otras) y tecnologías (robótica, sistemas láser, RISC, etc.), conlleva al sistema educativo a una actualización curricular continua, a la dinamización de la didáctica y a la adopción de modelos pedagógicos al interior de las universidades que garanticen la apropiación de los nuevos logros.

Teniendo en cuenta que desde Latinoamérica no eclosionan en general los últimos avances científico-tecnológicos, la velocidad de propagación de los nuevos saberes en el Tercer Mundo no es la misma que en los países donde se genera; en particular, mientras que la robótica tiene ya cierto grado de avance en países como Japón y Estados Unidos, en Colombia u otra nación en desarrollo apenas se logran los primeros avances. El estado actual de las telecomunicaciones como eje cervical de la *aldea global,* hace que las fuentes de información sobre los nuevos descubrimientos sean ahora asequibles desde prácticamente cualquier punto del globo terráqueo, por toda persona y con un mínimo de esfuerzo. No obstante, surgen varios

obstáculos que retardan la llegada de dichas fuentes a los lejanos países adaptadores de tecnología y generalmente hegemonizados en cultura:

- El proceso de comprensión y posterior adaptación de los más recientes avances científicos.

- La multiplicación o transmisión (valga decir, enseñanza) de los nuevos saberes en universidades, centros de investigación y empresas de carácter público, privado o mixto cuya competitividad gire alrededor de la tecnología de punta.

- La falencia económica que impide el desarrollo de la ciencia y la tecnología, debida a la corrupción burocrática o a factores de planificación política (por ejemplo, se invierte minoritariamente en educación y gran parte del rubro financiero se destina a robustecer el poder militar para perpetuar los vestigios del imperialismo, tal como lo evidencia el Plan Colombia).

- Además, deben existir ciertas “reservas científicas” no divulgables ipso facto, análogas a los denominados secretos de estado, que resguarden la ética del científico o cualquier otra eventualidad.

A partir de lo anterior surge el interrogante:

¿Qué estrategias metodológicas se deben implantar para impartir los avances más recientes en ciencia y tecnología a los estudiantes de ingeniería –u otro programa académico– de los últimos niveles del plan de formación?

MARCO DE REFERENCIA

De acuerdo a los antecedentes conceptuales descritos en los textos de apoyo[7], el interrogante planteado en el numeral anterior tiene relación directa con la *sociedad del conocimiento*, a la que el pensador Alvin Toffler denomina *tercera ola.* Ello es evidente porque las comunidades basan hoy su crecimiento y desarrollo en un bien intangible –el conocimiento– demandante, cada vez en mayor grado, de personal altamente capacitado en tecnología, propiciador de una cualificación científica continua que contrarreste la "caducidad académica" del docente universitario, factor negativo para la calidad del futuro egresado.

La apropiación de los aún párvulos conocimientos y la generación de éstos, exige la convergencia de aspectos enmarcados en:

---

7 - Buendía Gómez, Hernando. Educación: la agenda del siglo XXI, hacia un desarrollo humano. Santafé de Bogotá, Tercer Mundo Editores, 1998.
- Gómez B., Hernando y Jaramillo S., Hernán. (Compiladores). 37 modos de hacer ciencia en América Latina. Santafé de Bogotá, Tercer Mundo Editores, 1997.
- González A., Elvia María. El Proyecto de aula. Medellín, Universidad de Antioquia (Facultad de Educación).

- Un personal científico altamente calificado (para nuestro contexto, docentes con posgrado magistral o doctoral).

- Una línea de investigación que imprima particular fisonomía al quehacer científico.

- Un grupo de intelectuales garantes de la legitimidad del proyecto de saber, aceleradores de su ejecución, afianzadores de las funciones de la universidad: multiplicar, distribuír, aplicar y producir conocimiento.

- Una institución que aporte recursos logísticos y financieros para facilitar su desarrollo ponderado, sobre todo en investigación y ciencia conceptualizadas como proyecto de vida.

- Un conjunto de agentes externos expresados en instituciones pares que acrediten el logro científico, la valoración del trabajo foráneo y propio, el apoyo político, las relaciones con comunidades científicas transnacionales, etc.

NUEVO PLANTEAMIENTO

Las innovaciones tecnológicas de la informática robótica, las telecomunicaciones y la genética y los nuevos materiales en fibra óptica o rayos láser, por ejemplo, deben someterse a estudio con la incorporación de

modelos pedagógicos desarrollistas que incentiven la inteligencia en los cuerpos docente y dicente, motiven la iniciativa y propicien descubrimientos en un área específica del conocimiento. El medio empresarial debe actuar como “laboratorio”, evitando el distanciamiento entre los sectores productivo y educativo imperan0tes.

El currículo debe permear continuamente el progreso de la humanidad en todas sus facetas, asegurando una formación integral que apunte a un desarrollo ecuánime de la sociedad en los niveles educativo, económico, político y cultural.

La didáctica ha de transformar el proceso docente-educativo en una actividad placible para quienes la vivencian, plasmada en los roles activos del profesor emisor y el estudiante receptor –o viceversa por la reciprocidad de la comunicación–, con la ejecución, entre otras actividades, de proyectos de investigación multidisciplinares y el uso de la Red como medio más propicio para consonar con la globalización, sin incurrir en vacíos existenciales o descuidos de la esencia humana: transformadora de una sociedad por el hombre y para todos los seres, acentuando el asevero recursivo “pensar sobre lo que se piensa”.

PERTINENCIA

Varios aspectos de la temática tratada contribuyen con el proyecto de aula Metodología para la Algoritmia Orientada a Objetos –MAOO.

- El plan de formación para Ingenieros de Sistemas, Tecnólogos en Sistemas y demás carreras afines debe reestructurarse en lo referente al paradigma de programación a considerar desde los primeros niveles de estudio. En este sentido, la preparación actual en la mayoría de centros de educación superior del país se orienta a la perogrullada programación estructurada, no a la programación orientada a objetos. La esencia de MAOO gira en torno a la filosofía objetual, respondiendo así al ritmo cambiante en materia de ingeniería del software, una de las incontables facetas de la *sociedad del conocimiento*.

- El momento evaluativo (implementación) de la MAOO requiere de un proceso docente-educativo que adopte un modelo pedagógico simbiótico donde el mayor porcentaje lo absorbería el modelo desarrollista, tendiente a virar hacia otro modelo aún no propuesto o desconocido.

- La investigación certera implica creatividad, transición de lo ignoto a lo verificable, socialización del conocimiento, formulación de nuevos interrogantes. Estos son objetivos primigenios de MAOO, ad portas del siglo XXI. ▲[i]

F.2. Tendencias pedagógicas contemporáneas; la Universidad, su historia y su objeto de estudio; la abducción como modelo investigativo

CONTEXTUALIZACIÓN

La condición mutante del conocimiento humano plantea a las instituciones del orden superior un interrogante constante sobre su quehacer pedagógico. La historia evidencia que la universidad pública y privada, generalmente formadora de élites, ha difundido pensamientos míticos, filosóficos, científicos y dialécticos a partir de tendencias pedagógicas cimentadas en las Ciencias de la Educación, donde confluyen disciplinas como la Antropología, la Sociología, la Administración Educativa y la Sicología Cognitiva, entre otras. La pedagogía, considerada en términos generales como lo acometido en el aula –el término "aula" implica una macro cobertura que incluye salones de clase, auditorios, salas de cine, zonas verdes, superautopistas informáticas y en general cualquier espacio donde se pueda reflexionar sobre los variados aspectos formativos para el género humano–, no puede prescindir de la lógica para lograr su objetivo crucial: formar integralmente individuos para el desarrollo social en todas sus facetas. El vocablo "lógica" adopta una semántica múltiple basada en cuatro descripciones: lógica deductiva o analítica (Charles S. Pierce), lógica orientada al objeto (Immanuel Kant), lógica dialéctica (Jürgen Hegel), y lógica orientada a un fin (Herbert A. Simon). Según la primera de ellas –con mayoría

de adeptos en la comunidad científica- existen tres clases elementales de razonamiento: la abducción, la inducción y la deducción; en esta tríada el par inducción–deducción se asocia a la lógica formal ampliamente aplicada en los mundos matemático, lingüístico y filosófico; el componente abductivo se asocia a la lógica no formal, ofrece una mayor complejidad y se ha extendido a la literatura y al campo de los sistemas expertos en la inteligencia artificial. La abducción como método investigativo y manera de razonamiento ha generado controversias, hasta el punto de plantearnos el interrogante:

¿Existe una lógica de la abducción?

MARCO DE REFERENCIA

El concepto “abducción” no se trata directamente en los textos de apoyo, pero está implícito en la primera parte de *El territorio del vacío* de Alain Corbin –dicho apartado titula *La ignorancia y los balbuceos del deseo*, asociado al capítulo *Las raíces del miedo y la repulsión*–, en la obra del literato Edgar Allan Poe –aunque las lecturas recomendadas fueron *El escarabajo de oro* y *Criptografía*, la literatura investigativa de Poe se evidencia también en otras obras como *La carta robada*, *Los crímenes de la calle Morgue* y *El misterio de Marie Roget*; el pensamiento abductivo discurre en sus obras “siniestras”, verbi *gratia La máscara de la muerte roja*, *El hundimiento de la casa Usher*, *Metzengerstein*, *Eleonora* y varios textos adicionales del maestro–, en las *Anotaciones sobre El nombre de la rosa* de Umberto Eco redactadas por la profesora Consuelo

Posada y en los *Apuntes para una aproximación entre estética e investigación*, escritos por el profesor Rodrigo Argüello.

El razonamiento abductivo se resume en la siguiente inferencia.

i) Se observa el hecho sorprendente H1.

ii) Pero si H2 fuera verdadero, H1 sería cosa corriente.

Por tanto:

iii) Hay razón para sospechar que H2 es verdadero.

La abducción constituye entonces un silogismo en el cual la premisa mayor es evidente (H1) y la menor probable (H2). En aras de develar el arcano abductivo, identifiquemos algunas premisas H2 incluidas en las fuentes mencionadas.

(1) Las raíces del miedo y la repulsión.

> “La faz de la tierra antes del diluvio era suave, regular y uniforme, sin montañas y sin mar… tenía la belleza de la Juventud y la naturaleza llena de flores, fresca y fecunda, sin una sola arruga, cicatriz ni fractura en todo el cuerpo; no había rocas ni montañas, ni cavernosos agujeros, ni hendiduras... El aire era sosegado y sereno.”.

(2) El escarabajo de oro –Tomado de *Historias extraordinarias / Poemas*. Plaza & Janés, S.A., Editores. Traducción de Diego Navarro.

> “Como nuestro signo predominante es el 8, empezaremos por atribuirle la e del alfabeto natural. Para comprobar esta suposición, observamos si el 8 se ve a menudo por pares (la e se duplica con gran frecuencia en el inglés), en palabras tales como, por ejemplo, *meet*, *fleet*, *speed*, *seen*, *been*, *agree*, etc. En el presente caso vemos que se duplica no menos de cinco veces, aunque el criptograma es breve.”.

> Las palabras de Guillermo de Baskerville, el detective medieval de Umberto Eco en *El nombre de la rosa,* "La primera regla al descifrar un mensaje es adivinar lo que significa… pueden formarse algunas hipótesis sobre las posibles primeras palabras del mensaje, y luego ves si la regla, que infieres a partir de ellas, puede aplicarse al resto del texto", constituyen una explicación de la interpretación que se aplica al proceso de comprender en general: se comienza con una conjetura hipotética que se transforma en una argumentación, base del razonamiento abductivo y del proceso de interpretación de códigos y de generación inventiva de éstos en un determinado contexto de comprensión.

NUEVO PLANTEAMIENTO

Toda diligencia universitaria debe desarrollarse en un ambiente democrático que garantice la participación pluralista de los diversos actores de la sociedad, propendiendo por la búsqueda continua de una educación emancipadora garante de la promulgación y aplicación del pensamiento racional, en

consonancia con Richard Peters –R.S. Peters: *Ethics and Education*. London: Allen and unwin, 1966–, cuando escribe: "una sociedad democrática defiende el uso de la razón en la vida social y la autonomía personal como objetivo educativo". Una Razón aplicada a los procesos básicos (entre ellos la observación, la clasificación, y la elaboración), a los procesos superiores (resolución de problemas, creatividad, comprensión, etc), a los procesos complejos (pensamiento crítico; razonamiento inductivo, deductivo y abductivo; pensamiento científico) y a la metacognición.

PERTINENCIA

El razonamiento abductivo, tal como se especificó en la contextualización de la presente relatoría, ha sido utilizado para resolver variabilidad de problemas relacionados con la inteligencia artificial. En particular, el sistema experto Permaid (Rolston, 1987) –este sistema experto se emplea en el diagnóstico y mantenimiento de aprox. 10000 grandes subsistemas de disco con cabeza fija, componentes de computadoras HONEYWELL. Permaid realiza tres funciones principales: localiza problemas observados, realiza mantenimiento predictivo y recupera medios de información y archivos. Para mayor información al respecto, remitirse a *Principios de Inteligencia Artificial y Sistemas expertos*, David W. Rolston, McGraw Hill–, usa la inferencia abductiva en el núcleo de determinación de fallas, donde las causas específicas de ellas son hipotetizadas para explicar los eventos observados.

En muchos centros de educación superior del mundo se vienen realizando, desde hace un par de décadas, investigaciones sobre sistemas basados en conocimiento; en particular, en el Instituto Tecnológico y de Estudios Superiores de Monterrey -ITESM, resalta la tesis de Olivia Barrón Cano titulada *Una metodología para el prototipaje de sistemas expertos mediante representaciones intermedias del conocimiento y abducción* –Acceder el URL: http://www-cia.mty.itesm.mx/~rbrena/Proyectos.html.

Es indudable que la implementación del proyecto de aula MAOO -Metodología para la Algoritmia Orientada a Objetos- requerirá de la lógica abductiva para la solución de cierta gama de problemas. ▲[i]

F.3. PEDAGOGÍA, SOCIEDAD, ESTADO E INVESTIGACIÓN

Entre el estudiantado universitario es común el adagio "El profesor X sabe mucho pero no sabe enseñar". Es una afirmación bajo la cual subyacen muchas falencias de una sociedad que ocupa a personas para el quehacer docente con una excelente formación a nivel específico, pero escasa o nula a nivel pedagógico. El docente, consciente o no de tal hecho, a veces lo ignora, asumiendo una posición facilista de continuismo educativo donde el estudiante se tiene que amoldar a sus exigencias y estilo, respaldado en ocasiones por la propia institución educativa (obvia es la existencia de un conjunto de docentes excluido de tal afirmación, profesionales con una clara visión de la educación que a diario forjan valores para el desarrollo integral del personal dicente.).

¿Cómo nos integramos todos los que somos parte del proceso educativo, para trabajar con esa unidad que es el alumno?

El Estado y la sociedad son los responsables de la distribución de saberes a través de la formación primaria, secundaria y superior. El alumno recibe lo que le ofrendan desde la casa, la escuela y la sociedad; tres mensajes simultáneos la mayoría de las veces contrapuestos o diferentes, originarios de ciertas deficiencias que es menester enmendar.

Desde un principio no podemos considerar a una familia como un todo integrado y con un buen nivel educativo de los padres, tíos, abuelos o

hermanos mayores en nuestro país. El Estado obedece a intereses políticos del momento y la sociedad -monitoreada por los medios masivos de comunicación que son controlados por "formadores de opinión" que, aprovechando el bajo nivel de educación de la población utilizan su poder para beneficio de los intereses que representan- se convulsiona con la violencia de los actores del conflicto armado.

Todo esto hace que la situación actual sea de una profunda crisis educacional, en donde la escuela representa los ideales de la modernidad y los alumnos son postmodernos. Entre la escuela y los alumnos no se produce el diálogo y la comunicación que se requieren para el progreso social; los medios no facilitan la tarea educativa, en la mayoría de los casos la obstruyen; la sociedad se debate en una serie de devaneos y búsqueda de una nueva escala de valores éticos y morales en medio de una crisis alienante caracterizada por el descreimiento, el individualismo y la falta de cooperación. La familia, imbuida en esta maraña de desencuentros ya no influye tanto como formadora del individuo, muchas veces cae en la intrascendencia, la incomunicación y el aislamiento.

El valor que tiene la escuela es –a decir de *Braslavsky*– la de distribuir saberes útiles para el desarrollo económico autosostenido, el desarrollo nacional integrado y la construcción de un modelo político democrático. Surge entonces el interrogante: ¿quiénes son los responsables por la selección y distribución equitativa de saberes, de contenidos pertinentes para el desarrollo económico

autosostenido, el desarrollo nacional integrado y la construcción de un modelo político democrático?.

Los responsables son el Estado y la sociedad, o deberían serlo. Estamos asistiendo a un proceso de desintegración social, económica, regional, cultural, productiva y política en la cual emergen con toda virulencia los actores armados. Es el sálvese quién pueda. Los grupos sociales se cierran en defensa de sus propios intereses debido a la política llevada adelante por los gobiernos colombianos desde el Frente Nacional. Si hubo una denominada *Patria Boba,* la de hoy es *Hiperboba*: linda con los límites de la corrupción política, generadora de los vejámenes de violencia entre la milicia, paramilicia y guerrilla.

Ante esta situación nos encontramos con otro debate: *educación* estatal-educación privada. ¿Subsidiar o no subsidiar?. Si la educación pasa a ser privada... ¿cada escuela impartirá los conocimientos que se le ocurran? ¿el docente seguirá impartiendo sus pláticas sin una la formación pedagógica suficiente?. De concretarse tal situación sería como volver a la *Edad Media Feudal*.

La feudalización de la sociedad se caracteriza por los siguientes considerandos:

1. Dominio económico, social, cultural y político de un minúsculo grupo de personas que abusan de su poder.

2. Destrucción de la forma de vida democrática.

3. La escuela como instructora de valores, actitudes y aptitudes totalitarias al servicio del poder.

La universidad, sea pública o privada, debe invertir en la capacitación continua del personal docente a nivel de posgrado, exigiendo formación pedagógica que complemente el saber específico. La feudalización de la sociedad se evita con la investigación realizada en las universidades, liderada sobretodo por docentes y estudiantes de los últimos niveles de un plan de estudios.

La universidad debe establecer un espacio para el debate y la reflexión rigurosa sobre temas pertinentes a la ciencia, la educación y el desarrollo tecnológico y hacer de la investigación una actividad central en la vida institucional que a su vez contribuya al desarrollo del conocimiento y a la solución de problemas relevantes para el país.

De igual forma la universidad debe tener como objetivos prioritarios los siguientes:

- Apoyar la consecución de fuentes de financiación para la investigación.

- Implementar un sistema de información, seguimiento y evaluación de las investigaciones realizadas.

- Promover el desarrollo de investigaciones interdisciplinarias.

- Diseñar estrategias para articular las actividades investigativas con las de docencia con miras a garantizar un mayor impacto de la ciencia en la educación.

- Implementar mecanismos que hagan visible los resultados de la investigación realizada en la Universidad.

El Estado solo no puede asumir la responsabilidad de seleccionar y distribuir todos los saberes, pero *debe* ser el rector de las políticas educativas a nivel nacional, debe canalizar una serie de saberes fundamentales y básicos a toda la población nacional. El Estado tiene el deber de trazar el camino a seguir y controlar que se siga. El margen social para ciertos saberes locales y regionales no puede ser llevado a cabo sin límites y pautas generales establecidas por el Estado y la sociedad.

Nunca tendremos un sistema educativo en serio si antes todos los sectores sociales y políticos no se ponen de acuerdo en un proyecto de país por y para la educación, por y para la paz.

FUENTES CONSULTADAS

- ✓ Los conceptos fundantes de la pedagogía

  Vladimir Zapata Villegas

- ✓ Educación y pedagogía: una diferencia necesaria

  Olga Lucía Zuluaga, Alberto Echeverry, Alberto Martínez, Stella Restrepo, Humberto Quiceno.

- ✓ http://www.research.uniandes.edu.co

- ✓ http://www.nbci.com/mamaglio/Pedagogia/integracion.html

▲[i]

F.4. NUEVAS TECNOLOGÍAS APLICADAS A LA EDUCACIÓN

El desarrollo de las nuevas tecnologías de la información y comunicación en los últimos decenios ha suscitado en la sociedad necesidades a las que el sistema educativo debe responder con ahínco.

La eclosión de los multimedia, los hipermedios y los hipertextos en el ambiente virtual de la Internet, ha generado nuevas formas de ocio, comunicación, entretenimiento, delito, comercio y aprendizaje. En este último sentido, el sistema educativo del estado moderno debe generar directrices que faciliten la incursión de las nuevas tecnologías en la educación media y superior, con el objeto de:

a) Dotar a los alumnos de nuevos hábitos de lectura, debido a que la utilización de los hipertextos supone contar con un lector más activo que el habitual de textos impresos, con una mayor capacidad de análisis y de asociación de ideas.

b) Ofrecer a los ciudadanos desde la escuela, nuevas estrategias de acceso a la información. Otear videoconferencias, foros de discusión, chats, programas de educación virtual y demás avances tecnológicos en las áreas de la informática y telecomunicaciones, debe ser continua y adecuadamente administrado desde la escuela.

c) Incorporar la educación en valores como un complemento ineludible en la formación de personas con un alto compromiso para enfrentar las crisis sociales.

El primer objetivo, relacionado con una nueva forma de organizar cualquier clase de información conocida como hipertexto, se basa en la programación orientada al objeto –eje cervical del proyecto MAOO: Metodología para la Algoritmia Orientada a Objetos– y en la capacidad de la pantalla gráfica –que incluye iconos animados, gif inánimes, vídeo, audio,…, realidad virtual– para acceder a una gran cantidad de información con el objetivo de darle un tratamiento análogo al carácter asociativo de ideas que se atribuye normalmente a la mente humana. De esta forma, la estructura de la información es de naturaleza no lineal o no secuencial, permitiendo su "lectura" o consulta con una gran libertad, mediante la navegación o "browsing".

Conklin define el hipertexto ideal de acuerdo a las siguientes características:

- Se puede consultar la base de datos siguiendo los enlaces y visualizando los contenidos de las páginas o bien haciendo búsquedas mediante palabras claves o cualquier otro requisito similar.

- Es una base de datos en red formada por "páginas" o "nodos" de información textual o gráfica.

- Las páginas de las bases de datos se visualizan en la pantalla del ordenador en "ventanas", programadas para responder a eventos –los clics sobre el ratón, por ejemplo– que minimizan, maximizan o cierran la ventana.

Algo que caracteriza al hipertexto es el dinamismo y la interactividad que conlleva frente a lo estático del texto impreso; el hipertexto se presenta como un medio de organización más dinámico, ya que la información puede cambiar de lugar o presentación, estructurarse en varios niveles y sobre todo ofrecer diversos caminos de lectura en función del interés del usuario. Además es interactivo, en cuanto permite "dialogar" con el texto superando las limitaciones del papel, aumentando las posibilidades de comunicación entre usuarios y ordenadores. En particular, el proceso de vinculación de las tecnologías informáticas como apoyo a los procesos educativos permite una comunicación profesor-alumno y alumno-alumno más continua, asincrónica, en tiempo real o virtual, buscando que los estudiantes desarrollen las habilidades básicas para el trabajo en equipo y que puedan cotejar ideas de manera libre y tolerante.

El segundo objetivo atañe por antonomasia a Internet, donde confluyen extensas posibilidades para la educación en todos los niveles. La red ofrece la posibilidad de manifestar dudas a operadores distantes a menudo dispuestos a responder, constituyéndose un formidable apoyo al ejercicio del aprendizaje. Cabe anotar que lo anterior, aunado a los servicios de los buscadores y

metabuscadores, permite a los estudiantes encontrar sus trabajos pendientes ya recabados. En muchos casos, ni siquiera leen el contenido, sino que copian y pegan la información en el trabajo, con la consecuente transgresión de los derechos de autor. Esta situación nos lleva a reflexionar sobre los procesos de evaluación como un proceso administrativo –de quién aprueba y quién reprueba una asignatura– y mirarlos más bien como parte integral del proceso de aprendizaje que posibilite al alumno identificar sus errores y corregirlos.

Internet se ha convertido en un excelente medio de comunicación para realizar proyectos comparativos entre estudiantes y docentes de diferentes centros educativos, a través de los cuales se consigue que la creación del conocimiento pueda ser evaluada, discutida y, simultáneamente, propagada al instante. Este es uno de los objetivos de la llamada Universidad Virtual, una nueva modalidad de educación a distancia que hace uso de tecnologías de vanguardia: World Wide Web, correo electrónico, grupos de discusión (Hypernews, Netscape news, etc), conferencias en línea (chat), FTP, teclados interactivos (One Touch), Sistema de Integración Remota (SIR), sesiones satelitales, videos y videoconferencias.

La virtualización de la educación es una de las múltiples consecuencias del desarrollo de la telemática. Cada vez más estudiantes y empleados realizan, desde sus casas, tareas que ya no hace falta que cumplan en sus universidades y oficinas. El teletrabajo influye directamente sobre la economía por el abaratamiento de servicios que se pueden realizar a distancia y que le

permitan a un empleado tener contrato sólo por horas o por tarea realizada o de plano fungir como trabajador independiente o *free lance*. Todo ello tiende a modificar las ideas tradicionales sobre la fuerza laboral y la membresía de los sindicatos. El concepto de *estabilidad laboral* y principio como los de *jornada, tiempos de descanso* y *prestaciones*, se alteran con esta todavía sui generis pero en países desarrollados cada vez más frecuente modalidad de empleo. El teletrabajo plantea concepciones nuevas en términos urbanos, cotidianos y sociológicos en ese nuevo contexto doméstico. El español Javier Echavarría ha denominado *Telépolis* a ese entorno configurado por nuevas relaciones urbanas y domésticas, que afectan al orden de los asuntos civiles y personales.

En tercera y última instancia, las nuevas tecnologías inmersas en la educación deben permitir el crecimiento humano desde el conocimiento mismo hasta el arte, la ética y la formación humanista.

La plástica y la estética han adquirido nuevas dimensiones con las recientes familias de microprocesadores, tarjetas de vídeo, software de diseño gráfico y demás adelantos en microelectrónica.

La formación valorativa cimentada en la familia y la educación básica, madurada en la educación media y consolidada en la universidad, también se puede impartir desde el ciberespacio con los grupos de discusión, el correo electrónico y otras herramientas de la Internet ya mencionadas. En este sentido, el Computer Ethics Institute formuló, en 1972, el siguiente decálogo a

modo, diríase, de emulación de los diez mandamientos reguladores de la conducta cristiana:

1. No deberás usar tu computadora en agravio de otras personas.
2. No deberás interferir con el trabajo de la computadora de otra gente.
3. No deberás estar husmeando en los archivos de la computadora de otras personas.
4. No deberás usar la computadora para robar.
5. No deberás usar la computadora para levantar falsos testimonios.
6. No deberás copiar o usar el software ajeno por el que no has pagado.
7. No deberás usar los recursos de la computadora de otra persona sin autorización, o sin la compensación adecuada.
8. No deberás apropiarte de la producción intelectual de otros.
9. Deberás pensar acerca de las consecuencias sociales del programa que estas escribiendo, o del sistema que estas diseñando.
10. Siempre deberás usar una computadora de manera que asegure consideración y respeto para tus colegas humanos.

Desde el luego, la ética en el espacio computacional, a la que podríamos llamar *cibern-ética*, no se agota en el esquemático decálogo anterior. Esos mismos diez apartados sugieren la necesidad de profundizar al respecto.

BIBLIOGRAFÍA

- La nueva alfombra mágica.

  Usos y mitos de Internet, la red de redes.

  Raúl Trejo Delarbre

  Editorial Diana. México, 1996

- Guía para el alumno de la Universidad Virtual.

  Tecnológico de Monterrey –ITESM

- La formación de valores en la Universidad.

  Marta Lorena Salinas Salazar.

  Profesora Facultad de Educación, Universidad de Antioquia.

- En Internet se consultaron los siguientes URLs:

  http://nti.uji.es/docs/nti/tarragona.html

  http://huitoto.udea.edu.co/uso-tes

▲[i]

F.5. EL PROCESO DE APRENDIZAJE EN MAOO

Todo desarrollador de software debe pasar por un proceso de aprendizaje de métodos y técnicas que le permitan el diseño de programas óptimos para los usuarios finales y claramente documentados para el personal que presta soporte. La optimización se mide en términos de eficiencia, integridad, robustez, extensibilidad, reutilización, transportabilidad, amigabilidad, compatibilidad y verificabilidad, aspectos que se testean a través de todo el proceso de desarrollo de una manera iterativa.

El aprendizaje del ingeniero de software incluye, entre otros aspectos, técnicas para abordar el análisis de un problema, estrategias para encontrar la solución con una conceptualización adecuada de abstracciones, diseño de estructuras de datos y conocimiento de algún lenguaje de programación para implementar el diseño en una computadora.

Los aspectos relacionados con la conceptualización de abstracciones y diseño de estructuras de datos atañen directamente a MAOO, ya que las operaciones asociadas a cada clase se expresan por medio de algoritmos.

Estas operaciones involucran toda la teoría subyacente en la filosofía orientada a objetos e implican creatividad por parte de quienes las diseñan, por que se deben aplicar los artificios de la lógica expresados en los razonamientos inductivo, deductivo y abductivo.

Según lo anterior surgen varios interrogantes estrechamente relacionados:

- ¿Que propuestas didácticas se deben adoptar para la enseñanza de la algoritmia orientada a objetos?

- Si se aplica la estrategia de aprendizaje por descubrimiento con estudiantes de lógica de programación o estructuras de datos, ¿se operacionaliza mejor la estrategia con el método de casos, el método de proyectos, el método de problemas o el seminario investigativo?

- Entre las pedagogías expositiva y cognitiva, ¿cual será la más idónea para el aprendizaje de la algoritmia objetual?

Las respuestas a tales preguntas deben tener en cuenta la naturaleza de la lógica y las características propias del personal inmerso en el proceso docente – educativo del área en cuestión.

La naturaleza inherente a la lógica para programación de computadoras involucra la construcción del conocimiento a partir del desarrollo de los procesos cognitivos, las habilidades del pensamiento, la inteligencia y la creatividad. Ello conlleva a que la estrategia para la enseñanza de la lógica presuponga una mezcla de aprendizaje por descubrimiento y por construcción.

La forma de implementar mejor la estrategia de aprendizaje de la lógica esta dada por el PBL[8] (Problem-Based Learning) a partir de casos que los estudiantes deben resolver siguiendo unas etapas claramente definidas: abordar la situación problemática, definir el problema, explorar el problema, plantear la solución, llevar a cabo el plan y evaluar el proceso. El caso debe ser formulado por el profesor, quien a partir de su experiencia elige un problema con una gran gama de posibilidades analíticas que colmen las expectativas de aprendizaje y garanticen el logro de los objetivos trazados; por lo general un solo caso no bastará y se tendrán que abordar otros adicionales para que el proceso docente - educativo sea completo.

Las características de los participantes han de coadyuvar al desarrollo de habilidades interpersonales para el trabajo en grupo, de autoconfianza, de autodirección, de autoevaluación y de aprendizaje continuo como estrategia para el manejo del cambio. Por ello el docente debe:

- Mostrar preocupación por el estudiante como persona.

- Demostrar interés por el éxito de los estudiantes.

- Incluir actividades que preparen al estudiante para el campo profesional.

[8] Modelo de Aprendizaje que la Universidad de McMaster en Ontario, Canadá, ha venido desarrollando en sus diferentes escuelas con óptimos resultados.

- Responsabilizar al estudiante de su proceso de aprendizaje.

- Clarificar metas de aprendizaje.

- Lograr coherencia entre el proceso de evaluación y las metas de aprendizaje.

- Incluir actividades que ayuden a los estudiantes a crear una estructura de conocimiento útil, recuperable y aplicable en diferentes situaciones.

- Proporcionar retroalimentación inmediata.

- Motivar a los estudiantes.

Análogamente, el estudiante debe:

- Ser activo.

- Trabajar cooperativamente.

- Entender claramente la tarea y el tiempo necesario para realizarla.

- Dedicarse diligentemente a la tarea.

- Ser consciente de su propio estilo de aprendizaje para utilizarlo con efectividad.

- Ser consciente de la necesidad de crear una estructura de conocimiento.

- Ser responsable de su proceso de aprendizaje.

En el área especifica de la ingeniería de software, el modelaje de un problema a implementar en una computadora con un lenguaje de programación orientado a objetos se comienza a dilucidar y a plantear con base en los denominados *casos de uso,* planteados por Ivar Jacobson en 1992, y que consisten esencialmente en interacciones típicas entre los usuarios y un sistema computacional. Un caso de uso es una descripción de un conjunto de secuencias de acciones, incluyendo variantes, que ejecuta un sistema para producir un resultado observable de valor para un actor, donde este último podría ser una persona o un dispositivo de hardware que desempeña determinado rol respecto al sistema.

Los casos de uso constituyen una manera de enfrentar la complejidad asociada al desarrollo de software en la etapa de análisis de requerimientos. Las etapas y actividades en el desarrollo orientado a objetos basado en UML se circunscriben al análisis de requerimientos, el diseño del sistema, el diseño detallado, la implementación y las pruebas, todas ellas enmarcadas en un ciclo iterativo e incremental a través del tiempo. Cabe preguntarse si Jacobson al

proponer sus famosos *use cases,* lo hizo basado en el método de casos –vía de operacionalización para la estrategia de aprendizaje por descubrimiento– sistematizado por la universidad de Harvard en la década del 60 a partir de las teorías de John Dewey. De hecho, existen muchas similitudes entre lo que es un caso de uso en el UML y un caso problemático propuesto a un estudiante para que éste aplique la teoría aprendida sobre determinado item de conocimiento, ya que ambos:

- Desarrollan y posibilitan capacidad de comunicación entre estudiante/profesor, estudiante/estudiante, ingeniero de software/usuario final, ingeniero de software/personal directivo.

- Ejercitan habilidades para procesar información, evaluar alternativas de solución y tomar decisiones.

- Propician y consolidan el trabajo en grupo.

- Desarrollan la capacidad de argumentación en el estudiante y en el ingeniero de software.

- Suponen un procedimiento ágil, participativo, dinámico, flexible y riguroso de análisis.

- Permiten identificar actores –usuarios, dispositivos hardware, personal administrativo, personal operativo, etc– que intervengan en el caso, incluyendo los roles que éstos desempeñan.

- Clarifican las relaciones con el entorno.

- Sirven para establecer cuadros comparativos entre los posibles casos que ofrezcan cierta similitud.

- Posibilitan el análisis de errores y excepciones.

- Siempre son el eslabón esencial para el logro de un objetivo: aprendizaje o desarrollo de software.

El aprendizaje de la lógica para programación de computadoras con técnicas modernas o algoritmia orientada a objetos involucra, en definitiva, el empleo de casos y problemas específicos que permitan desarrollar en el estudiante o el profesional todo su potencial de razonamiento. La estrategia de aprendizaje por descubrimiento y construcción basada en problemas encaja had hoc dentro del proceso de diseño de algoritmos con un alto grado de desempeño para los usuarios finales.

BIBLIOGRAFIA

- Estrategias y métodos de enseñanza en la universidad.

  Ponencia presentada al Primer encuentro de profesores universitarios de cursos básicos de química. 7 y 8 de nov. De 1996.

  CORREA URIBE, SANTIAGO. Profesor Facultad de Educación. Universidad de Antioquia.

- Aprendizaje Basado en Problemas. Documento.

  BERNARDO RESTREPO GÓMEZ. Ph. D. En Educación.

- Aprendizaje Basado en Problemas: una alternativa educativa.

  LAURA LIVAS. Master en Enseñanza Superior. Docente Escuela de Educación. Universidad Regiomontana, Monterrey. ▲[i]

F.6. LA EVALUACIÓN DE UN ALGORITMO

Antes de emitir conceptos sobre todo lo que implica la evaluación de un algoritmo presentado por un estudiante como respuesta al planteamiento de un problema particular propuesto por el docente, se enumerarán brevemente las características generales de la evaluación, dilucidadas a partir de la fuente bibliográfica ARR0YAVE.

1. El desarrollo de la evaluación inmersa en el proceso de enseñanza-aprendizaje debe partir de los interrogantes filosóficos: evaluar a quién, cómo, por qué, para qué, cuándo, dónde, qué, con cuáles medios.

2. La evaluación como proceso es un asunto que no se puede dejar solamente en instrumentos de evaluación descontextualizados y sin claridad. Hay que dejar claro desde qué perspectiva referencial se plantea.

3. El proceso de la evaluación posibilita un mejoramiento continuo en los procesos de aprendizaje del alumno y de enseñanza del docente.

4. El proceso de la evaluación dentro del componente método de la didáctica tratada como un sistema complejo, propone cuatro niveles evaluativos: la autoevaluación, la coevaluación, la heteroevaluación y la interevaluación, los cuales en su proceso dinámico de aplicabilidad conciben la evaluación de una manera íntegra, integral e integradora para los actores del proceso formativo.

5. La evaluación es un proceso permanente, continuo, aplicado desde el inicio de las actividades académicas propias a un objeto de conocimiento específico, evitando el facilismo de la evaluación terminal del modelo educativo clásico; una evaluación distribuida a lo largo de todo el proceso formativo, alivia la presión del tiempo, permite a los sujetos en formación ver varios enfoques de las tareas propuestas y los compromete a dar y recibir retroalimentación sobre los trabajos que se están realizando.

   El proceso de evaluación alude a la importancia práctica de ofrecer, a los sujetos formadores y en formación, una frecuente evaluación informativa-formativa en todo momento, no tanto con fines calificativos, sino para hacer avanzar los conocimientos significativos asimilados, comprendidos y contextualizados, sus habilidades estimuladas desarrolladas y aplicadas en la solución de problemas cotidianos y sus valores cultivados y formados para múltiples interacciones dentro de la cultura intersubjetiva.

6. La evaluación posibilita un proceso de enseñanza efectivo, en cuanto a que la valoración del propio desempeño se vuelve casi automática, puesto que constantemente se esta comparando el desempeño actual con el anterior y con aquel al que se quiere llegar. Además cambia el equilibrio de poder y autoridad en el aula, en la medida que se transforma desde una herramienta para el control del docente hacia un proceso por el cual docentes y alumnos

aprenden a usar criterios explícitos para evaluar y mejorar sus conocimientos, las habilidades y los valores del proceso formativo.

La efectividad del proceso también se basa en criterios abiertos vinculados directamente con los conocimientos, habilidades y valores de alumnos y docentes; si una mayoría de estudiantes falla al aplicar el instrumento evaluativo, el docente, a partir de un análisis objetivo de la situación, deberá generar estrategias que conduzcan al mejoramiento de los resultados basado en criterios académicos, éticos y ecuánimes.

7. Un proceso evaluativo presenta desafíos porque en su desarrollo se perturba la cultura del secreto, propia de la mayoría de las modalidades de examen en las instituciones educativas, y hace que los actores formadores y en formación asuman nuevos roles y relaciones, conscientes de las tensiones locales, regionales y globales que los puedan afectar.

Todas estas características generales de la evaluación son también aplicables a la evaluación de un algoritmo orientado a objetos, con algunas consideraciones adicionales.

Cuando el docente se propone evaluar un problema planteado a, por ejemplo, 20 estudiantes, debe tener en cuenta que no necesariamente va encontrar algoritmos idénticos dado que el proceso de razonamiento de un individuo difiere respecto al de los demás, aunque es claro que todas las soluciones

expuestas deben confluir a un mismo objetivo. El modelo de solución del docente ha de ser flexible y abierto a las distintas posibilidades que exponen los estudiantes, sin olvidar aspectos importantes en desempeño del algoritmo:

- La eficiencia en el uso de recursos de memoria o espacio de almacenamiento, es decir, el algoritmo debe utilizar la memoria estrictamente necesaria que conlleve a la solución del problema. Sin embargo se puede aducir que el uso de la memoria en un algoritmo ya no es tan importante como en el pasado, si se tienen en cuenta los grandes adelantos en microelectrónica que han conducido a la fabricación de computadoras con gran capacidad de memoria RAM. A pesar de ello, el uso de variables, objetos, o estructuras de datos redundantes se deben evitar en todo algoritmo.

- La conservación de los formalismos matemáticos para la construcción de expresiones aritméticas y lógicas, así como las reglas asociadas a la sintaxis general de las sentencias de control para manejar decisiones, ciclos y asignaciones en los diagramas N-S, y la simbología básica del UML para representar clases, objetos, herencias y demás conceptos fundamentados en el paradigma orientado a objetos usados en la solución del problema.

- El tiempo de computación o de ejecución equivalente con la rapidez con que se ejecuta el algoritmo. Dicho tiempo es independiente de la computadora

que se vaya a utilizar y del lenguaje en el que se vaya a desarrollar, y se puede calcular utilizando técnicas formales como la notación O grande que conduce a tiempos de ejecución constantes, lineales, cuadráticos, cúbicos, logarítmicos, exponenciales, etc. Si dos algoritmos que solucionan el mismo problema tienen órdenes de magnitud lineal y cuadrátrico, se debe seleccionar el primero porque el algoritmo de complejidad lineal es más rápido.

- El comportamiento del algoritmo con respecto del caso medio y el caso peor. El caso medio se refiere a la situación normal bajo la cual funcionaria el algoritmo, con datos de entrada acordes a situaciones reales rutinarias. El caso peor es aquel bajo el cual el algoritmo requiere más tiempo, y se considera para algoritmos cuyos tiempos de respuesta sean críticos. Por ejemplo, si se trata de controlar una planta nuclear, es crucial saber el límite superior del tiempo de respuesta del sistema, independiente del ejemplar concreto que se vaya a resolver. [BRASSARD].

- El nivel de reutilización de clases ya definidas, tratando siempre de escribir algoritmos basados en componentes para madurar la idea de “chip de software”, evitando la monotonía de “reinventar la rueda”.

Dejo a consideración de los profesionales de la Ingeniería de Sistemas y ciencias de la computación, entre los que se encuentran analistas, desarrolladores, gerentes de la información y docentes universitarios, entre

otros, anotaciones adicionales sobre la evaluación de un algoritmo y sobre la propuesta que subyace en el proyecto de aula MAOO. Toda crítica será recibida con beneplácito, en búsqueda de una metodología que permita soportar los fundamentos de la filosofía orientada a objetos desde los primeros niveles de formación en los diferentes planes de estudio de las carreras de ciencias e ingeniería.

BILIOGRAFÍA

ARROYAVE G., DORA INÉS. La didáctica como un sistema complejo. Medellín, 2001.

BRASSARD, G. / BRATLEY, T. Fundamentos de Algoritmia. Prentice Hall, Madrid, 1997.

▲i

F.7. Métodos de enseñanza para la algoritmia orientada a objetos

Es común entre estudiantes de Ingeniería de Sistemas y programas afines, que asignaturas como Lógica de Programación, Algoritmos y Estructuras de Datos, sean estudiadas en varios semestres consecutivos por efectos de pérdida o cancelación, debido a dos factores fundamentales:

- La naturaleza misma de las asignaturas, que exige del estudiante una alta capacidad de abstracción y razonamiento.

- El método de enseñanza y por ende de aprendizaje, adoptado por el docente.

Como es de esperarse, existen excepciones a la mortandad académica dado que cierto porcentaje de estudiantes aprueba dichas asignaturas con alta calificación, mientras que otros las reprueban, las aprueban con notas mínimas o simplemente las cancelan en espera de una nueva oportunidad para el semestre venidero. Respecto a los primeros cabe preguntarse:

¿Realmente lograron los objetivos planteados?

La respuesta a esta pregunta implica la consideración de factores endógenos y exógenos al proceso académico reflejado en los métodos de enseñanza del docente y en las estrategias de aprendizaje de los estudiantes, las cuales implican un exhaustivo y extenso análisis que no se desarrollará en estas líneas pues el interés gira en torno a una de las categorías de la Didáctica, el Método,

concretizado en el método de enseñanza idóneo para impartir el objeto de conocimiento Algoritmia Orientada a Objetos.

En esencia, tanto la algoritmia tradicional –donde lo procedimental basado en una fase de análisis fundamentada en los requerimientos de usuario, es lo primordial– como la orientada a objetos –eje fundamental del proyecto de aula MAOO– requieren de métodos de enseñanza dinámicos allende de lo tradicional, donde los dogmas no tienen mucha cabida dado que la Lógica de Programación se fundamenta en temarios propios de las Matemáticas Discretas –álgebra booleana, teoría de conjuntos, cálculo proposicional, cálculo de predicados, etc.–, combinados con las reglas del razonamiento inductivo y deductivo. En algoritmos voraces, paralelos, probabilistas y heurísticos, el razonamiento abductivo puede desempeñar un rol importante.

El dinamismo impreso al método adoptado para explicar la lógica computacional objetual incluye los siguientes aspectos:

1. Una fuente de adquisición del conocimiento basado en textos, sean libros, ensayos, artículos de revista o hipertextos y en explicaciones magistrales que conduzcan al estudiante a discernir sobre el objeto del conocimiento, aún mas allá de lo aportado por el docente en la clase. El trabajo de motivación debe inducir a desarrollar en el estudiante sus propias

capacidades de razonamiento aplicadas a la algoritmia, dado que el tutor no puede proveer inteligencia para lograr el aprendizaje por ser esta inherente al individuo.

La función del profesor universitario no puede limitarse solo a la transmisión del conocimiento sino a la estimulación en los alumnos del propio deseo de adquirirlos, despertando un espíritu crítico y reflexivo. El trabajo motivacional cobra relevancia si se tiene en cuenta la masificación de ciertos programas académicos, entre ellos Ingeniería de Sistemas, donde un seguimiento individual del proceso de aprendizaje se torna a veces inviable. En general, las motivaciones deben ir dirigidas a concienciar al alumno sobre la labor que está realizando, haciéndole ver que el conocimiento en construcción es de vital importancia para su futuro profesional.

2. La relación entre la actividad del profesor y la de los estudiantes se debe flexibilizar para permitir los trabajos expositivos del profesor, independientes del estudiante y de elaboración conjunta entre ambos. En este último caso, sería interesante simular la lógica inherente a un software construido con base a componentes reutilizables, donde cada estudiante o grupo de ellos construye sus propios módulos para después ensamblarlos en un producto único que solucione cierta situación problemática. Este ejercicio serviría además para explicar el concepto de interfaz externa o prototipo de función.

La relación docente-dicente se puede dinamizar con la presentación de impulsos didácticos a modo de preguntas que incentiven la actividad consciente productiva del estudiante, en lugar de la reproductiva. Las preguntas deben permitir al aprendiz retomar conceptos ya estudiados que faciliten procesar activamente la información en búsqueda de desarrollo de nuevos significados y habilidades.

3. El carácter de la actividad cognoscitiva propia de la algoritmia orientada a objetos ha de conducir a una apropiación del conocimiento de una manera reflexiva que promueva el razonamiento y la creatividad, evitando la memorización y solución de problemas a partir de modelos preestablecidos. Para lograrlo, el profesor planteará problemas o casos que conduzcan a una actividad y búsqueda independiente del conocimiento por parte de los estudiantes, conocida como búsqueda parcial o heurística. La heurística se ha utilizado en filosofía y lógica para referirse a la rama de la ciencia que estudia el razonamiento, e incluye una serie de principios, reglas y estrategias, perfectamente aplicables a la enseñanza de la algoritmia.

El principio de analogía “sugiere la utilización de contenido o forma para lograr inferencias nuevas sobre la base de las propiedades o relaciones conocidas”; según esta definición de Lourdes Valverde Ramírez (Los métodos de enseñanza y aprendizaje, Facultad de Educación, Universidad de Antioquia. Febrero 2001), se observa que cierta gama de algoritmos se puede resolver por analogía con respecto a otros, sin incurrir en el problema

de la memorización o de los modelos preestablecidos mencionados anteriormente.

El principio de reducción es aplicable en el sentido de plantear soluciones algorítmicas mínimas pero eficientes, que optimicen el uso de recursos de memoria y minimicen la redundancia de instrucciones mediante técnicas adecuadas de recurrencia, inducción, deducción y generalización.

Las reglas heurísticas expresan acciones y operaciones a realizar en la búsqueda de medios y vías para resolver un problema; por ejemplo, dos reglas heurísticas comunes en la solución de un problema por medio de un algoritmo serían “identificar, si existen, los datos de entrada en información de salida” y “pensemos sobre la estructura de datos más apropiada para este caso”.

Las estrategias heurísticas, clasificadas en dos grupos, son tácticas que posibilitan encontrar los métodos necesarios para resolver un ejercicio. En el primer grupo se encuentra la estrategia de trabajo hacia atrás y en el segundo la estrategia de trabajo hacia delante, ambas de uso extendido en la enseñanza de la matemática.

En el ambiente de la enseñanza de algoritmos, el trabajo hacia atrás no es aplicable dado que con esta estrategia se supone conocido lo buscado, técnica que tiene validez en el álgebra relacional y el cálculo relacional de tuplas y dominios, inherentes al área de las bases de datos como sustento

lógico del SQL (Structured Query Language: lenguaje inmerso en los manejadores de bases de datos relacionales y orientados a objetos, de uso generalizado entre los desarrolladores de software).

El trabajo hacia adelante es aplicable a la enseñanza de la algoritmia objetual, porque al solucionar un problema se puede partir de unos datos de entrada a los cuales se les aplica un proceso de razonamiento para conformar un plan de solución que permita organizar, buscar, insertar, eliminar o modificar datos o emitir resultados que conduzcan a toma de decisión gerencial o de usuario final.

En conclusión, la utilización de un método mixto es lo ideal para enseñar los fundamentos de la algoritmia orientada a objetos, donde se combinen la lección magistral debidamente estructurada, la motivación que permita la constante iniciativa del estudiante en busca del objeto de conocimiento, los multimedios como apoyo externo al proceso docente-educativo, la propuesta de casos que incentive la creatividad y el empleo de estrategias heurísticas para forjar ingenieros de software capaces de responder adecuadamente a las exigencias cambiantes del sector productivo.

BIBLIOGRAFÍA

- Valverde Ramírez, Lourdes. Los métodos de enseñanza y aprendizaje. Universidad de Antioquia. Facultad de educación. Febrero de 2001.

- Uso de nuevas tecnologías en la educación superior. http://www.edudistan.com/Zara%20Stella%20Sierra%20Avila.htm

- Blanco, Francisco José. Los métodos docentes y las nuevas tecnologías: hacia un método mixto. http://www.ciberaula.es/quaderns/hemeroteca/quaderns/sumario19/blanco.html

▲ i

G

GLOSARIO

Este glosario, por demás pequeño, define algunos términos relevantes utilizados en el presente trabajo. La semántica dada coincide a veces con lo literario, otras con lo técnico referido a la informática. Las entradas están organizadas alfabéticamente, escritas en **negrita** y en minúscula, a excepción de las siglas y los nombres propios que se presentan o comienzan en mayúsculas. Al final de cada significado se pueden presentar algunas de las siguientes convenciones:

- Contraste: <término>. Hace referencia a un término que tiene un significado opuesto o substancialmente diferente.
- Ver: <término>. Hace referencia a un término que tiene significado similar, pero no sinónimo.
- Sinónimo: <término>. Indica que ambos términos tienen el mismo significado.

abducción. Movimiento por el cual un miembro u otro órgano se aleja del plano medio que divide imaginariamente el cuerpo en dos partes simétricas: abducción del brazo, del ojo. ll Silogismo cuya premisa mayor es evidente y la menor menos evidente o solo probable.

acción. Un procedimiento, algorítmico o computacional.

agregación. Acción y efecto de agregar o agregarse. || Forma especial de asociación entre clases que especifica una relación todo-parte entre el agregado (todo) y una parte que lo compone.

agregar. Unir o juntar unas personas o cosas a otras.

algoritmia. Ciencia del cálculo aritmético y algebraico; teoría de los números.

algoritmo. Conjunto ordenado y finito de operaciones que permite hallar la solución de un problema.

aprender. Adquirir el conocimiento de algo por medio del estudio o de la experiencia.

argumento. Valor específico correspondiente a un parámetro. Sinónimo: parámetro real. Contraste: parámetro.

arte. Virtud, disposición y habilidad para hacer algo. || Manifestación de la actividad humana mediante la cual se expresa una visión personal y desinteresada que interpreta lo real o imaginario con recursos plásticos, lingüísticos o sonoros || Maña, astucia.

C++. Lenguaje de programación desarrollado por Bjarne Stroustrup a principios de la década de los 80, bajo el cual se puede programar de manera estructurada u orientada a objetos; por eso se dice que es un lenguaje híbrido.

ciencia. Conjunto de conocimientos obtenidos mediante la observación y el razonamiento, sistemáticamente estructurados y de los que se deducen principios y leyes grandes.

clase. Orden en que, con arreglo a determinadas condiciones o calidades, se consideran comprendidas diferentes personas o cosas. || Descripción de un conjunto de objetos que comparten los mismos atributos, operaciones, métodos, relaciones y semántica. Una clase es una implementación de un tipo abstracto de datos.

cognición. Conocimiento.

comportamiento. Efectos visibles de una operación o evento, incluyendo sus resultados.

conocer. Averiguar por el ejercicio de las facultades intelectuales, la naturaleza, cualidades y relaciones de las cosas. || Entender, advertir, saber, echar de ver. || Percibir el objeto como distinto de todo lo que no es él.

conocimiento. Acción y efecto de conocer. Entendimiento, inteligencia, razón natural.

diagrama N-S. Forma de representación de un algoritmo, basada en bloques de construcción rectangulares. El apéndice A contiene más detalles al respecto.

didáctico, **ca**. Perteneciente o relativo a la enseñanza. Propio, adecuado para enseñar o instruir. Arte de enseñar. || Según Muller, la didáctica tiene como origen el verbo griego *didaskein*, que se utiliza tanto en activo, enseñar, como en pasivo, aprender o ser enseñado, y también transitivo, en el sentido de aprender por sí mismo; el sustantivo derivado *didáxis* significa enseñanza, y *didaktiké téchne* el arte de enseñar. En un sentido amplio la didáctica puede concebirse como la reflexión científica sobre la enseñanza y el aprendizaje.

educación. Acción y efecto de educar. Instrucción por medio de la acción docente.

educar. Dirigir, encaminar, doctrinar, desarrollar o perfeccionar las facultades intelectuales y morales del niño o del joven por medio de preceptos, ejercicios, ejemplos, etc.

enseñanza. Acción y efecto de enseñar. || Sistema y método de dar instrucción. || Ejemplo, acción o suceso que sirve de experiencia, enseñando o advirtiendo cómo se debe obrar en casos análogos.

enseñar. Instruir, doctrinar, amaestrar con reglas o preceptos. Dar advertencia, ejemplo o escarmiento que sirva de experiencia y guía para obrar en lo sucesivo.

estado. Condición o situación en la vida de un objeto, durante la cual satisface una condición, realiza una actividad, o está esperando un evento.

estrategia. Arte de dirigir las operaciones militares. || Habilidad para dirigir un asunto.

evento. Acontecimiento significativo. Un evento tiene ubicación en el tiempo y en el espacio y puede tener parámetros. Un usuario interactúa con una GUI a través de eventos sobre el ratón, tales como clic, doble clic, clic derecho, etc. o con eventos sobre cajas de texto como la tecla *enter*.

función. Capacidad de acción o acción propia de los seres vivos y de sus órganos y de las máquinas o instrumentos. || Subprograma que siempre retorna un valor único al programa principal. Ver: método.

generalización. Acción y efecto de generalizar. || Relación taxonómica entre un elemento más general y uno más específico. El elemento más específico es totalmente consistente con el más general y contiene más información. En los lugares donde están permitidas instancias del objeto más general pueden utilizarse instancias del más específico. Ver: herencia.

generalizar. Hacer pública o común una cosa. Abstraer lo que es común y esencial a muchas cosas, para formar un concepto general que las comprenda a todas.

GUI. (Graphic User Interface)

heredar. Suceder por disposición testamentaria o legal en los bienes y acciones que otro tenía al tiempo de su muerte.

herencia. Derecho de heredar. || Mecanismo por el cual elementos más específicos incorporan la estructura y el comportamiento de elementos más generales. Ver: generalización.

implementación. Definición de cómo está construído o compuesto algo. Por ejemplo: una clase es una implementación de un tipo, un método es una implementación de una operación.

instancia. Individuo descripto por una clase o un tipo. Nota de uso: de acuerdo con la interpretación estricta del metamodelo un individuo de un tipo es una instancia y un individuo de una clase es un objeto. Es aceptable, en un contexto menos formal, referirse a un individuo de una clase como un objeto o una instancia. Ver: tipo. Contraste: objeto.

Interface. Utilización de un tipo para describir el comportamiento visible de una clase, objeto u otra entidad. En el caso de una clase o un objeto, la interface incluye la signatura de las operaciones. Ver: tipo.

Java. Lenguaje de programación orientado a objetos creado en Sun Microsystems por James Gosling y su grupo de colaboradores. Java incluye extensas bibliotecas de clases con componentes para multimedia, conectividad de red, subprocesos múltiples, computación distribuida, acceso a bases de datos, etc.

mensaje. Comunicación entre objetos que transmite información con la expectativa de desatar una acción. La recepción de un mensaje es, normalmente, considerada un evento.

metamodelo. Modelo que define el UML para expresar un modelo.

método. Modo de decir o hacer con orden. || Obra que enseña los lineamientos de una ciencia o arte. || Implementación de una operación. Algoritmo, función o procedimiento que permite llegar al resultado de una operación.

metodología. Ciencia del método. || Conjunto de métodos que se siguen en una investigación científica o en una exposición doctrinal.

objeto. Todo lo que puede ser materia de conocimiento o sensibilidad de parte del sujeto, incluso este mismo. || Entidad delimitada precisamente y con identidad, que encapsula estado y comportamiento. El estado es representado por sus atributos y relaciones; el comportamiento es representado por sus operaciones y métodos. Un objeto es una instancia de una clase. Ver. Clase, instancia.

OMT. Sigla de Object Modelling Techniques. Metodología para el análisis y el diseño orientado a objetos propuesta por James Rumbaugh

operación. Servicio que puede ser requerido a un objeto para producir un comportamiento. Una operación tiene una signatura, que puede restringir sus parámetros posibles.

orden de magnitud. Medida que establece la complejidad de un algoritmo. Un orden de magnitud puede ser constante, logarítmico, semilogarítmico, exponencial y otros mas.

pedagogía. Ciencia que se ocupa de la educación y la enseñanza. En general, lo que enseña y educa por doctrina o ejemplos.

parámetro. Especificación de una variable que puede ser pasada, cambiada y/o devuelta. Un parámetro puede incluir nombre, tipo y dirección. Los parámetros son utilizados para operaciones, mensajes y eventos. Sinónimo: parámetro formal. Contraste: argumento.

parámetro formal. Sinónimo: parámetro.

parámetro real. Denominado también parámetro actual. Sinónimo: argumento.

proceder. Originarse una cosa de otra. Modo de portarse una persona.

procedimiento. Acción de proceder. || Subprograma que retorna cero, uno, dos o más valores al programa principal. Ver: método.

relatar. Referir o dar a conocer un hecho.

relator. Que relata o refiere una cosa. Persona que en un congreso o asamblea hace relación de los asuntos tratados, así como de las deliberaciones y acuerdos correspondientes.

relatoría. Empleo u oficina de relator. En el contexto de este trabajo, relatoría es sinónimo de relato.

signatura. Nombre y parámetros de una operación, mensaje o evento. Opcionalmente los parámetros pueden incluir un parámetro devuelto como resultado de la operación, mensaje o evento.

subalgoritmo. Algoritmo que al ser combinado con otros algoritmos, sirve para la solución de un problema determinado. Ver: subprograma. Sinónimo: algoritmo.

subprograma. En informática, conjunto de instrucciones que realizan una función particular dependiente del programa principal. Segmento de programa que realiza una tarea específica bien definida. En lenguajes de programación como pascal, los subprogramas se dividen en procedimientos y funciones. Subalgoritmo codificado.

técnico, **ca**. Perteneciente o relativo a las aplicaciones de las ciencias y las artes. || Persona que posee los conocimientos especiales de una ciencia o arte. || Conjunto de procedimientos y recursos de que se sirve una ciencia o un arte.

tecnología. Conjunto de teorías y de técnicas que permiten el aprovechamiento práctico del conocimiento científico. || Lenguaje propio de una ciencia o de un arte.

tipo. Descripción de un conjunto de instancias que comparten las mismas operaciones, atributos abstractos, relaciones abstractas y semántica. Un tipo puede definir la especificación de una operación (como su signatura) pero no su implementación (método). Nota sobre el uso: a veces tipo e interface se usan como sinónimos pero no son términos equivalentes. Ver: clase, instancia. Contraste: interface.

UML. Sigla de Unified Modeling Language. Lenguaje gráfico para visualizar, especificar, construir y documentar los artefactos de un sistema con gran cantidad de software. Para mayor información, ver el apéndice B. ▲i

REFERENCIAS BIBLIOGRÁFICAS

AHO, Alfred V., Jhon E. Hopcroft y Jeffrey D. Ullman*, Estructuras de Datos y Algoritmos*, Addisson-Wesley Iberoamericana, Wilmington, Delaware, E.U.A., 1998.

ARROYAVE, Dora Inés. *La Didáctica como un sistema complejo*, Medellín, 2001.

BLANCO, Francisco José. *Los métodos docentes y las nuevas tecnologías: hacia un método mixto*. http://www.ciberaula.es/quaderns/hemeroteca/quaderns/sumario19/blanco.html

BECERRA Santamaría, César, *Estructuras de datos en C++*, Por Computador Ltda, Santafé de Bogotá, 1999.

BOOCH, Grady, James Rumbaugh e Ivar Jacobson, *El Lenguaje de Modelado Unificado*, Addison Wesley Iberoamericana, Madrid, 1999.

BISHOP, Judy, *Java: Fundamentos de Programación*, Addison Wesley, Madrid, 1999.

BRASSARD, G. y P. Bratley, *Fundamentos de Algoritmia*, Prentice Hall, Madrid, 1997.

BUENDÍA Gómez, Hernando. *Educación: la agenda del siglo XXI, hacia un desarrollo humano*, Tercer Mundo Editores, Santafé de Bogotá, 1998.

CAIRÓ, Osvaldo, y Silvia Guardati, *Estructuras de Datos*, McGraw Hill, México, 1993.

CORBIN, Alain. *El territorio del vacío*, Biblioteca Mondadori, Madrid, 1993.

CORREA Uribe, Santiago. *Estrategias y métodos de enseñanza en la universidad*, Universidad de antioquia, Facultad de Educación, Ponencia presentada en el Primer encuentro de profesores universitarios de cursos básicos de Química, 7 y 8 de de noviembre de 1996.

DEITEL, H. M. y P. J. Deitel, *Cómo Programar en C/C++*, Prentice Hall Hispanoamericana, México, 1997.

DEITEL, H. M. y P. J. Deitel, *Cómo Programar en Java*, Prentice Hall Hispanoamericana, México, 1998.

DOMÍNGUEZ Giraldo, Gerardo. *Indicadores de Gestión y Resultados*, Biblioteca Jurídica Diké, Medellín, 2002.

FOWLER, Martin y Kendall Scott, *UML Gota a Gota*, Addison Wesley Longman de México, S.A. de C.V., México, 1999.

GARCíA DE SOLA , Juan F. y Vicente Garcerán Hernández, *Lenguaje C y Estructura de Datos: aplicaciones generales y de gestión*, McGraw Hill, México, 1993.

GONZÁLEZ Agudelo, Elvia María. *El Proyecto de Aula*, Universidad de Antioquia, Facultad de Educación, Medellín, 2000.

GOTTFRIED, Byron, *Programación en C*, Mc Graw Hill, Madrid, 1991.

JARAMILLO S., Hernán y Hernando Gómez B. (Compiladores). *37 formas de hacer ciencia en América Latina*, Tercer Mundo Editores, Santafé de Bogotá, 1997.

JOYANES Aguilar, Luis, *Programación Orientada a Objetos,* Osborne McGraw-Hill, 1998.

KRUSE, Robert L., *Estructuras de Datos y Diseño de Programas*, Prentice Hall, México, 1988.

LANGSAM, Y., M.J. Augenstein y A. M. Tenenbaum, *Estructuras de Datos con C y C*++, Prentice Hall Hispanoamericana, México, 1997.

LARMAN, Craig, *UML y Patrones: Introducción al Análisis y Diseño Orientado a Objetos*, Prentice Hall, México, 1999.

LIPSCHUTZ, Seymour, *Estructura de Datos*, McGraw Hill, Madrid, 1987.

LOOMIS, Mary S., *Estructura de Datos y Organización de Archivos*, Prentice Hall, México, 1991.

MARTIN, James y James J. Odell, *Análisis y Diseño Orientado a Objetos*, Prentice Hall Hispanoamericana, 1994.

PEÑA Tamayo, Jhon Jairo y Martha Nelly Mesa Granda, *Propuesta de guía metodológica para la elaboración de trabajos de grado (Cuarta versión)*, Universidad Cooperativa de Colombia, Facultad de Ingeniería, Medellín, 2001.

PETERS, R.S. *Ethics and Education*, Allen and Unwin, London, 1996.

POE, Edgar Allan. *Historias Extraordinarias/ Poemas*, Plaza & Janés Editores, Barcelona, 1973.

PRESSMAN, Roger S., *Ingeniería del software Un enfoque práctico*, McGraw-Hill, 1998.

REAL ACADEMIA ESPAÑOLA, *Diccionario de la Lengua Española*, Vigésima segunda edición, Espasa Calpe S.A., Madrid, 2001.

RIOS Castrillón, Fabián, *Apuntes de diagramación*, Facultad de Ingeniería, Universidad de Antioquia, 1982.

ROLSTON, David W. *Principios de Inteligencia Artificial y Sistemas Expertos*, McGraw Hill, México, 1997.

RUMBAUGH, James, Ivar Jacobson y Grady Booch, *El lenguaje de Modelado Unificado. Manual de referencia*, Pearson Educación, Madrid, 2000.

RUMBAUGH, James, Michael Blaha, William Premerlani, Eddy Frederick y William Lorensen, *Modelado y Diseño Orientado a Objetos*, Prentice Hall, Madrid, 1996.

SALINAS Salazar, Marta Lorena. *La formación en valores en la Universidad*, Universidad de Antioquia, Facultad de Educación, 2000.

SANFORD, Leetsma y Larry Nyhoff, *Programación en Pascal*, Prentice Hall, Madrid, 1999.

SCHMULLER, Joseph, *Aprendiendo UML en 24 horas*, Pearson Educación, México, 2000.

SEDGEWICK, Robert, *Algoritmos en C++*, Addison-Wesley / Díaz de Santos, Wilmington, Delaware, E.U.A., 1995.

WEISS, Mark Allen, *Estructuras de Datos y Algoritmos*, Addison-Wesley Iberoamericana, Wilmington, Delaware, E.U.A., 1995.

VALVERDE Ramírez, Lourdes. *Los métodos de enseñanza y aprendizaje*, Universidad de Antioquia. Facultad de Educación, Medellín, Febrero de 2001.

VILLALOBOS, Jorge A., *Diseño y Manejo de Estructuras de Datos en C*, Mc Graw Hill, Santafé de Bogotá, 1996. ▲ i