

**UNIVERSIDAD DE ORIENTE**

**NÚCLEO DE ANZOÁTEGUI**

**COORDINACIÓN DE POSTGRADO**

**MAESTRÍA DE INFORMÁTICA GERENCIAL**

**II FASE**

**ASIGNACIÓN INDIVIDUAL**

|  |  |
| --- | --- |
| **Profesora:** | **Realizado por:** |
| Palomo Teresa | Alvino V., José Daniel |

**BARCELONA, JULIO DE 2017**

1. **¿Qué es la Programación?**

Es la actividad para el diseño y desarrollo de programas; conjunto de instrucciones que explicitan un algoritmo escrito en un lenguaje de programación y con las que se pueden realizar un trabajo determinado mediante la ejecución de tales instrucciones mediante la computadora. Este proceso a su vez, determina tres requisitos previos a la actividad de programar:

* La necesidad de resolver un problema mediante el computador.
* Una estrategia de solución con pasos, acciones o procesos a desarrollar para resolver el problema en cuestión (algoritmo).
* Un lenguaje que permita la comprensión y expresión de datos y acciones computacionales por realizar y resolver el problema dado.

1. **Enumere y explique el ciclo de vida de un programa**

El ciclo de vida de un programa es una secuencia estructurada y bien definida de las etapas en Ingeniería de software para desarrollar el producto software deseado. El mismo aporta una serie de pasos a seguir con la finalidad de diseñar y desarrollar un producto software de manera eficiente, el cual veremos a continuación:

1. **Comunicación**

Este es el primer paso donde l usuario inicia la petición de un producto software determinado. Contacta al proveedor de servicios e intenta negociar las condiciones. Presenta su solicitud al proveedor de servicios aportando la organización por escrito.

1. **Recolección de requisitos**

A partir de este paso y en adelante el equipo de desarrollo software trabaja para tirar adelante el proyecto. El equipo se reúne con varios depositarios de dominio del problema, e intentan conseguir la máxima cantidad de información posible sobre lo que requieren. Los requisitos se contemplan y agrupan en requisitos del usuario, requisitos funcionales y requisitos del sistema. La recolección de todos los requisitos se lleva a cabo como se especifica a continuación:

* Estudiando el software y el sistema actual u obsoleto,
* Entrevistando a usuarios y a desarrolladores de Software,
* Consultando la base de datos o
* Recogiendo respuestas a través de cuestionarios.

1. **Estudio de viabilidad**

Después de la recolección de requisitos, el equipo idea un plan para procesar el software. En esta fase, el equipo analiza si el software puede hacerse para cubrir todos los requisitos del usuario y si hay alguna posibilidad de que el software ya no sea necesario. Se investiga si el proyecto es viable a nivel financiero, práctico, y a nivel tecnológico para que la organización acepte la oferta. Hay varios algoritmos disponibles, los cuales ayudan a los desarrolladores a concluir si el proyecto software es factible o no.

1. **Análisis del sistema**

En este paso los desarrolladores trazan su plan e intentan crear el mejor y más conveniente modelo de software para el proyecto. El análisis del sistema inclye el entendimiento de las limitaciones del producto Software; el aprendizaje de los problemas relacionados con el sistema; los cambios que se requieren en sistemas ya existentes con antelación, identificando y dirigiendo el impacto del proyecto a la organización y al personal, etc. El equipo del proyecto analiza las posibilidades del proyecto y planifica la temporalización y los recursos correspondientes.

1. **Diseño**

El siguiente paso es diseñar el producto software con la ayuda de toda la información recogida sobre requisitos y análisis. Los inputs (aportaciones) de los usuarios y los resultados de la recogida de información hecha en la fase anterior serán las aportaciones base de la fase actual. El output (o resultado) de esta etapa toma la forma de 2 diseños; El diseño lógico y el diseño físico. Los ingenieros crean meta-data (Metadatos), Diagramas dilógicos, diagramas de flujo de datos, y en algunos casos pseudocódigos.

1. **Codificación**

Esta fase también se puede denominar 'fase de programación'. La implementación del diseño de software empieza con el lenguaje de programación más conveniente, y desarrollando programas ejecutables y sin errores de manera eficiente.

1. **Pruebas**

Se estima que el 50% de todos los procesos de desarrollo de software deberían ser evaluados. Los errores pueden arruinar el software tanto a nivel crítico y hasta el punto de ser eliminado. Las pruebas de Software se hacen mientras se codifica y suelen hacerlo los desarrolladores y otros expertos evaluadores a varios niveles. Esto incluye evaluación de módulos, evaluación del programa, evaluación del producto, evaluación interna y finalmente evaluación con el consumidor final. Encontrar errores y su remedio a tiempo es la llave para conseguir un software fiable.

1. **Integración**

El Software puede necesitar estar integrado con las bibliotecas, Bases de datos o con otro u otros programas. Esta fase se focaliza en la integración del software con las entidades del mundo exterior.

1. **Implementación**

Aquí se instala el software en máquinas de clientes. A veces, el software necesita instalar configuraciones para el consumidor final con posterioridad. El Software se evalúa por su adaptabilidad y su portabilidad, en cuanto a las cuestiones relacionadas con la integración y conceptos asociados, se resuelven durante la implementación.

1. **Mantenimiento y Funcionamiento**

Esta fase confirma el funcionamiento del software en términos de más eficiencia y menos errores. Si se requiere, los usuarios se forman, o se les presta documentación sobre como operar y como mantenerlo en funcionamiento. El software se mantiene de forma temprana actualizando el código en acorde a los cambios que tienen lugar en entornos del usuario o tecnológicos. Esta fase puede que tenga que encarar retos originados por virus ocultos o problemas no identificados del mundo real.

1. **¿De qué manera los programas pueden resolver problemas?**

El proceso de resolución de un problema con una computadora conduce a la escritura de un programa y a su ejecución en la misma. Aunque el proceso de diseñar programas es “esencialmente”, un proceso creativo, se puede considerar una serie de fases o pasos comunes, que generalmente deben seguir todos los programadores.

Las fases de resolución de un problema con computadora son:

* Análisis del problema: El problema se analiza teniendo presente la especificación de los requisitos dados por el cliente de la empresa o por otra persona que encarga el programa.
* Diseño del algoritmo: una vez analizado el problema, se diseña una solución que conducirá a un algoritmo que resuelva el problema.
* Codificación (implementación): la solución se escribe en la sintaxis del lenguaje de alto nivel (por ejemplo, C ) y se obtiene un programa.
* Ejecución, verificación y depuración: el programa se ejecuta, se comprueba rigurosamente y se elimina todos los errores (denominados “bugs”, en inglés) que puedan aparecer.
* Mantenimiento: El programa se actualiza y modifica, cada vez que sea necesario, de modo que se cumplan todas las necesidades de cambio de sus usuarios.
* Documentación: Escritura de las diferentes fases del ciclo de vida del software, esencialmente el análisis, diseño y codificación, unidos a manuales de usuario y referencia, así como normas para el mantenimiento.

Las dos primeras fases conducen a un diseño detallado escrito en forma de algoritmo. Durante la tercera etapa (codificación) se implementa el algoritmo en un código escrito en un lenguaje de programación, reflejando las ideas desarrolladas en las fases de análisis y diseño.

La fase de ejecución y compilación traduce y ejecuta el programa. En las fases de verificación y depuración el programador busca errores de las etapas anteriores y los elimina. Comprobará que mientras más tiempo se gaste en la fase de análisis y diseño, menos se gastará en la depuración del programa. Por último, se debe realizar la documentación del programa.

1. **Defina cada una de las estructuras de programación y ejemplifíquelas haciendo uso de un algoritmo.**

**Estructuras de control: condicionales y bucles**

Son parte fundamental de cualquier lenguaje. Sin ellas, las instrucciones de un programa solo podrán ejecutarse en el orden en que están escritas (orden secuencial). Las estructuras de control permiten modificar este orden. Hay dos categorías de estructuras de control:

* **Condicionales o bifurcaciones:** permiten que se ejecuten conjuntos distintos de instrucciones, en función de que se verifique o no determinada condición.
* **Bucles o repeticiones:** permiten que se ejecute repetidamente un conjunto de instrucciones, bien un numero predeterminado de veces, o bien hasta que se verifique una determinada condición.

En términos de un lenguaje de programación, que se verifique o no una condición se traduce en que una (adecuada) expresión lógica tome el valor VERDADERO (TRUE) o tome el valor FALSO (FALSE).

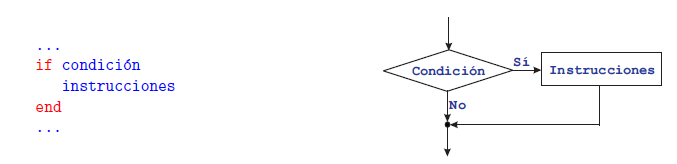
En los casos más sencillos y habituales la condición suele ser una comparación entre dos datos, como, por ejemplo: si a < b hacer una cosa y en caso contrario hacer otra distinta.

A continuación, se describen las distintas estructuras de control. Para cada una de ellas se describe el diagrama de flujo y la sintaxis de la sentencia correspondiente. Obsérvese que todas ellas tienen una única entrada y una única salida.

**Estructura condicional simple: IF**

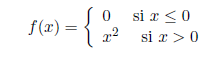
Este es el tipo más sencillo de estructura condicional. Sirve para implementar acciones condicionales del tipo siguiente:

* Si se verifica una determinada condición, ejecutar una serie de instrucciones y luego seguir adelante.
* Si la condición NO se cumple, NO se ejecutan dichas instrucciones y se sigue adelante.



Obsérvese que, en ambos casos (que se verifique o no la condición), los caminos" bifurcados se unen posteriormente en un punto, es decir, el flujo del programa recupera su carácter secuencial, y se continúa ejecutando por la instrucción siguiente a la estructura IF.

*Ejemplo:* Utilización de este tipo de condicional, se considera el cálculo del valor en un punto x de una función definida por partes, como, por ejemplo:



*Algoritmo:* Calculo del valor de la función f(x) = 0 si x · 0, f(x) = x2 si x > 0.

*Inicio*

*1- LEER x*

*2- HACER f=0*

*3- Si x>0*

*HACER f=x2*

*Fin Si*

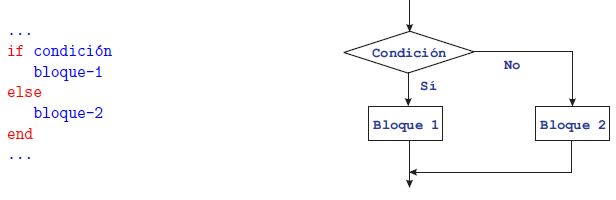
*4- IMPRIMIR 'El valor de la función es: ', f*

*Fin*

**Estructura condicional doble: IF - ELSE**

Este tipo de estructura permite implementar condicionales en los que hay dos acciones alternativas:

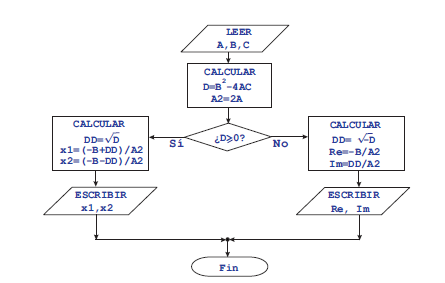
* Si se verifica una determinada condición, ejecutar una serie de instrucciones (bloque 1).
* Si no, esto es, si la condición NO se verifica, ejecutar otra serie de instrucciones (bloque 2).



*Ejemplo:* Utilización de este tipo de estructuras se plantea el problema de calcular las raíces de una ecuación de segundo grado

ax2 + bx + c = 0

distinguiendo dos casos: que las raíces sean reales o que sean complejas (no se contempla, de momento, distinguir entre una o dos raíces reales). Ver a continuación el diagrama de flujo y el pseudocódigo correspondiente.



*Algoritmo:* Cálculo de las raíces de la ecuación de segundo grado Ax2 + Bx + C = 0, distinguiendo los casos de raíces reales y complejas.

*Inicio*

*1- LEER A,B y C*

*2- CALCULAR D=B2-4\*A\*C*

*3- CALCULAR AA=2\*A*

*4- Si D≥0*

*CALCULAR DD=√D*

*x1=(-B+DD)/AA*

*x2=(-B-DD)/AA*

*IMPRIMIR 'La ecuación tiene raíces reales:', x1, x2*

*Si no*

*CALCULAR DD=√-D*

*Re=-B/AA*

*Im=DD/A2*

*IMPRIMIR 'La ecuación tiene raíces complejas conjugadas:'*

*IMPRIMIR 'Parte real:', Re*

*IMPRIMIR 'Parte imaginaria:', Im*

*Fin Si*

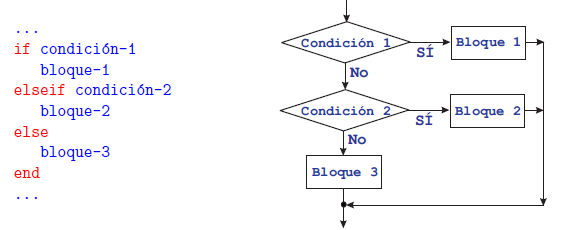
*Fin*

**Estructura condicional múltiple: IF - ELSEIF - ELSE**

En su forma más general, la estructura IF - ELSEIF - ELSE permite implementar condicionales más complicados, en los que se \encadenan" condiciones en la forma siguiente:

* Si se verifica la condición 1, ejecutar las instrucciones del bloque 1.
* Si no se verifica la condición 1, pero SI se verifica la condición 2, ejecutar las instrucciones del bloque 2.
* Si no, esto es, si no se ha verificado ninguna de las condiciones anteriores, ejecutar las instrucciones del bloque 3.

En cualquiera de los casos, el flujo del programa continua por la instrucción siguiente a la estructura IF - ELSEIF - ELSE.



Ejemplo: Determinación del signo de un número: positivo, negativo o nulo.

*Inicio*

*1- LEER X*

*2- Si X>0*

*IMPRIMIR 'El número tiene signo positivo'*

*Si no, si X<0*

*IMPRIMIR 'El número tiene signo negativo'*

*Si no*

*IMPRIMIR 'El número es nulo'*

*Fin*

En la estructura IF - ELSEIF - ELSE se puede multiplicar la cláusula ELSEIF, obteniéndose así una cascada" de condiciones, como se muestra en el organigrama, cuyo funcionamiento es claro.

En este tipo de estructura condicional, la cláusula ELSE junto con su bloque de instrucciones pueden no estar presente.

Las distintas estructuras condicionales descritas pueden ser anidadas, es decir, puede incluirse una estructura IF (de cualquier tipo), como parte de las instrucciones que forman el bloque de uno de los casos de otro IF. Como es lógico, no puede haber solapamiento. Cada estructura IF debe tener su propio fin (end).

Como ejemplo de utilización de este tipo de estructura condicional y de estructuras anidadas:

Dados dos números reales, a y b, y el símbolo, S (carácter), de un operador

aritmético (+, -, \*, /), imprimir el resultado de la operación a S b

*Inicio*

*LEER a*

*LEER b*

*LEER S*

*Si S='+'*

*IMPRIMIR 'El resultado es =', a+b*

*Si no, si S='-'*

*IMPRIMIR 'El resultado es =', a-b*

*Si no, si S='\*'*

*IMPRIMIR 'El resultado es =', a\*b*

*Si no, si b=0*

*Si a=0*

*IMPRIMIR 'El resultado es =', NaN (indeterminación)*

*Si no*

*IMPRIMIR 'El resultado es =', Inf (infinito)*

*Fin Si*

*Si no*

*IMPRIMIR 'El resultado es =', a/b*

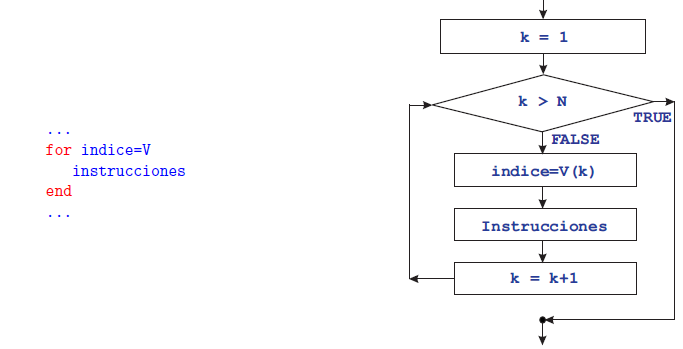
*Fin Si*

*Fin*

**Estructura de repetición indexada: FOR**

Este tipo de estructura permite implementar la repetición de un cierto conjunto de instrucciones un numero predeterminado de veces.

Para ello se utiliza una variable de control del bucle, llamada también índice, que va recorriendo un conjunto prefijado de valores en un orden determinado. Para cada valor del índice en dicho conjunto, se ejecuta una vez el mismo conjunto de instrucciones.

En la Figura se han representado la forma de escribir esta estructura y el organigrama correspondiente: el bloque de instrucciones se ejecuta una vez para cada valor del índice, que va tomando sucesivamente el valor de cada componente del vector V, de longitud N.

Como ejemplo de utilización de la estructura FOR, los siguientes algoritmos para calcular la suma de los n primeros números impares.

Nota:

a) El valor de la variable de control índice puede ser utilizado o no dentro del conjunto de instrucciones que forman parte del cuerpo del FOR, pero no debe ser modificado.

b) El conjunto de valores que debe recorrer el índice puede ser vacío (N=0). En ese caso, el bloque de instrucciones no se ejecuta ninguna vez.

c) Las estructuras FOR e IF pueden anidares", es decir, incluir una dentro de la otra, con la restricción (de sentido común) de que la interior tiene que estar completamente contenida en uno de los bloques de instrucciones de la otra. Véase, como ejemplo, el Algoritmo 7.

*Algoritmo:* Dado un entero, n, calcular la suma de los n primeros números impares.

*Inicio*

*LEER n*

*HACER suma=0*

*Para i= 1, 3, 5, ..., 2\*n-1*

*HACER suma=suma + i*

*Fin Para*

*IMPRIMIR 'La suma vale: ', suma*

*Fin*

*Algoritmo:* Dado un entero, n, calcular

*Inicio*

*LEER n*

*HACER suma=1*

*HACER ter=1*

*Para k= 1, 2, ..., n*

*HACER ter=ter/2*

*HACER suma=suma + ter*

*Fin Para*

*IMPRIMIR 'La suma vale: ', suma*

*Fin*

*Algoritmo:* Dado un numero natural, n, imprimir la lista de sus divisores, en orden decreciente.

Inicio

LEER n

IMPRIMIR ' Lista de divisores del número: ', n

Para i=ParteEntera(n/2) hasta 2 (incremento -1)

Si resto(n/i)=0

IMPRIMIR i

Fin Si

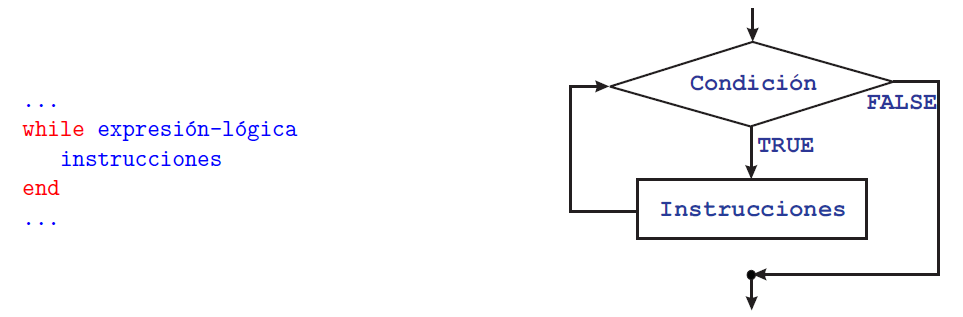
Fin Para

IMPRIMIR 1

Fin

**Estructura repetitiva condicional: WHILE**

Permite implementar la repetición de un mismo conjunto de instrucciones mientras que se verifique una determinada condición: el número de veces que se repetirá el ciclo no está definido a priori.



Su funcionamiento es evidente, a la vista del diagrama:

1. Al comienzo de cada iteración se evalúa la expresión-lógica.

2. Si el resultado es VERDADERO, se ejecuta el conjunto de instrucciones y se vuelve a iterar, es decir,

se repite el paso 1.

3. Si el resultado es FALSO, se detiene la ejecución del ciclo WHILE y el programa se sigue ejecutando

por la instrucción siguiente al END.

Ejemplo: Imprimir de forma ascendente los 100 primeros números naturales.

*Inicio*

*i=1*

*Mientras que i · 100*

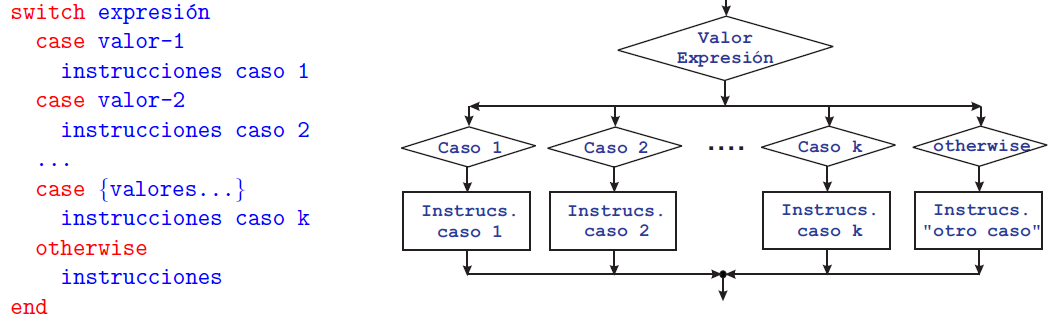
*IMPRIMIR i*

*HACER i=i+1*

*Fin Mientras*

*Fin*

**Estructura de elección entre varios casos: SWITCH**

Este tipo de estructura permite decidir entre varios caminos posibles, en función del valor que tome una determinada instrucción.

En cada uno de los casos, el valor correspondiente puede ser o bien un solo valor, o bien un conjunto de valores, en cuyo caso se indican entre llaves. La cláusula OTHERWISE y su correspondiente conjunto de instrucciones puede no estar presente.

El funcionamiento es el siguiente:

1. Al comienzo se evalúa la expresión.

2. Si expresión toma el valor (los valores) especificados junto a la primera clausula CASE, se ejecuta el

conjunto de instrucciones de este caso y después se abandona la estructura SWITCH, continuando por la instrucción siguiente al END.

3. Se repite el procedimiento anterior, de forma ordenada, para cada una de las clausulas CASE que siguen.

4. Si la cláusula OTHERWISE está presente y la expresión no ha tomado ninguno de los valores anteriormente especificados, se ejecuta el conjunto de instrucciones correspondiente.

Obsérvese que se ejecuta, como máximo el conjunto de instrucciones de uno de los casos, es decir, una vez que se ha verificado un caso y se ha ejecutado su conjunto de instrucciones, no se testea el resto de casos, ya que se abandona la estructura. Obviamente, si la cláusula OTHERWISE no está presente, puede ocurrir que no se dé ninguno de los casos.

Ejemplo: Dados dos números reales, a y b, y el símbolo, S (carácter), de un operador aritmético (+, -, \*, /), imprimir el resultado de la operación a S b

*LEER a , b , S*

*Elegir caso S*

*Caso '+'*

*IMPRIMIR 'El resultado es =', a+b*

*Caso '-'*

*IMPRIMIR 'El resultado es =', a-b*

*Caso '\*'*

*IMPRIMIR 'El resultado es =', a\*b*

*Caso '\*'*

*IMPRIMIR 'El resultado es =', a\*b*

*Caso '/'*

*Si a6=0*

*IMPRIMIR 'El resultado es =', a/b*

*Si no, si b6=0*

*IMPRIMIR 'El resultado es =', Inf (infinito)*

*Si no*

*IMPRIMIR 'El resultado es =', NaN (indeterminación)*

*Fin Si*

*En otro caso*

*IMPRIMIR 'El operador no se reconoce'*

*Fin Elegir caso*

1. **Establezca las diferencias entre la Programación Estructurada y la Programación Orientada a Objetos.:**

* La programación orientada a objetos o POOes un paradigma de programación que usa objetos y sus interacciones, para diseñar aplicaciones y programas informáticos. Está basado en varias técnicas, incluyendo herencia, abstracción, polimorfismo y encapsulamiento.
* A diferencia del paradigma estructurado, que propone modelar a la realidad como una serie de procedimientos secuenciales, la orientación a objetos propone representar todo lo que conocemos en términos de entidades (objetos) que interactúan y se relacionan entre sí. Estas entidades pueden representar absolutamente cualquier cosa, desde algo físico y tangible como una persona, una factura o un auto, hasta cosas intangibles como la imaginación, un proceso químico o un algoritmo matemático.
* La POO difiere de la programación estructurada tradicional, en la que los datos y los procedimientos están separados y sin relación, ya que lo único que se busca es el procesamiento de unos datos de entrada para obtener otros de salida. La programación estructurada anima al programador a pensar sobre todo en términos de procedimientos o funciones, y en segundo lugar en las estructuras de datos que esos procedimientos manejan.
* En la programación estructurada sólo se escriben funciones que procesan datos. Los programadores que emplean POO, en cambio, primero definen objetos para luego enviarles mensajes solicitándoles que realicen sus métodos por sí mismos.

En la actualidad, el paradigma de orientación a objetos es sin lugar a dudas el más utilizado por las empresas de todo el mundo a la hora de encarar desarrollos de aplicaciones de software, ya que permite representar de manera relativamente simple modelos y realidades muy complejas y esto hace que el software sea más fácil de programar, comprender y mantener.