ALGORITMOS Y ESTRUCTURAS DE DATOS
SEMANA Nº 2

OBJETIVOS DE LA SEGUNDA SEMANA

Clase teórica
- Algoritmos (Concepto, representación gráfica, escritura de algoritmos, estructuras de control)
- Programas (Concepto, Organización, Programas tipos, Elementos básicos y Esquema general).
- Lenguajes de programación.

Clase práctica
- Ejercicios con clases (en diagramas) y diagramas de flujo.

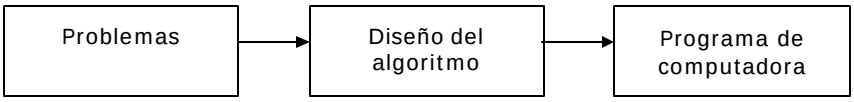
CLASE TEORICA

ALGORITMOS

1. CONCEPTO

El objetivo fundamental de esta materia es enseñar a resolver problemas mediante una computadora. Un programador de computadora es antes que nada una persona que resuelve problemas, por lo que para llegar a ser un programador eficaz se necesita aprender a resolver problemas de un modo riguroso y sistemático.

La resolución de un problema exige el diseño de un algoritmo que resuelva el problema propuesto.



Los pasos para la resolución de un problema son:

1. Diseño del algoritmo que describe la secuencia ordenada de pasos, sin ambigüedades, que conducen a la solución de un problema dado. (Análisis del problema y desarrollo del algoritmo)
2. Escritura del algoritmo como un programa en un lenguaje de programación adecuado.
3. Ejecución y validación del programa por la computadora

Para llegar a la realización de un programa es necesario el diseño previo de un algoritmo, de modo que sin algoritmo no puede existir un programa.

Los algoritmos son independientes tanto del lenguaje de programación en que se expresan como de la computadora que los ejecuta. En cada problema el algoritmo se puede expresar en un lenguaje diferente de programación y, ejecutarse en una computadora distinta; sin embargo, el algoritmo será siempre el mismo. Así por ejemplo, en una analogía con la vida diaria, una receta de un plato de cocina se puede expresar en español, ingles o francés, pero cualquiera que sea el lenguaje, los pasos para la elaboración del plato se realizarán sin importar el cocinero.

En la ciencia de la computación y en la programación los algoritmos son más importantes que los lenguajes de programación o las computadoras. Un lenguaje de programación es tan sólo un medio para expresar un algoritmo y una computadora es sólo un procesador para ejecutarlo. Tanto el lenguaje de programación como la computadora son los medios para obtener un fin: conseguir que el algoritmo se ejecute y se efectúe el proceso correspondiente.

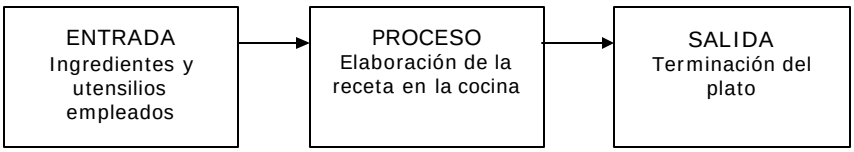
Dada la importancia del algoritmo en la ciencia de la computación, un aspecto muy importante será el diseño de algoritmo,

El diseño de la mayoría de los algoritmos requiere creatividad y conocimientos profundos de la técnica de la programación. En esencia, todo problema se puede describir por medio de un algoritmo.

Las características fundamentales que debe cumplir todo algoritmo son:

- Un algoritmo debe ser preciso e indicar el orden de realización de cada paso.
- Un algoritmo debe estar definido. Si se sigue un algoritmo dos veces, se debe obtener el mismo resultado cada vez.
- Un algoritmo debe ser finito. Si se sigue un algoritmo, se bebe terminar en algún momento; o sea debe tener un número finito de pasos.

La definición de un algoritmo debe describir tres partes: Entrada, Proceso y Salida. En el algoritmo de receta de cocina citada anteriormente se tendrá:



2. REPRESENTACION GRAFICA DE LOS ALGORITMOS

Para representar un algoritmo se debe utilizar algún método que permita independizar dicho algoritmo del lenguaje de programación elegido. Ello permita que un algoritmo pueda ser codificado indistintamente en cualquier lenguaje. Para conseguir este objetivo se precisa que el algoritmo sea representado gráfica o numéricamente, de modo que las sucesivas acciones no dependan de la sintaxis de ningún lenguaje de programación, sino que la descripción pueda servir fácilmente para la transformación en un programa, es decir, su codificación.

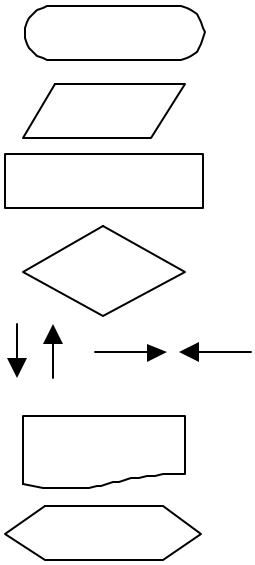
Los métodos para representar un algoritmo son:

- Diagrama de flujo.
- Pseudocódigo.
- Lenguaje español.
- Fórmulas.

DIAGRAMA DE FLUJO

Un diagrama de flujo es una de las técnicas de representación de algoritmo más antiguo y a la vez más utilizado. Un diagrama de flujo es un diagrama que utiliza los símbolos (cajas) estándar y que los pasos del algoritmo son escritos en el interior, donde son unidos mediante de flechas, denominadas líneas de flujo, que indican la secuencia en que se deben ejecutar.

Símbolos principales:

SIMBOLOS PRINCIPALES	FUNCION
	**Terminal** representa el comienzo "inicio' y el final "fin", de un programa. **Entrada/Salida**, cualquier tipo de introducción de datos en la memoria desde los periféricos. **Proceso**, cualquier tipo de operación que pueda originar cambio de valor, operaciones aritméticas, transferencia. **Decisión**, indica operaciones lógicas o de comparación entre datos y en función del resultado de la misma determina cuál de los distintos caminos alternativos del programa se debe seguir; normalmente tiene dos salidas. **Indicador de dirección o línea de flujo**, indica el sentido de ejecución de las operaciones. **Impresora**, se utiliza en ocasiones **en** lugar del símbolo de E/S **Subrutina**, indica comienzo de una subrutina o función

Los símbolos estándar normalizados son muy variados, sin embargo, los símbolos más utilizados representan:

- Proceso
- Decisión
- Fin
- Entrada / Salida
- Dirección del flujo.

En un diagrama de flujo:

a. Existe una caja etiquetada "inicio", que es de tipo elíptico.
b. Existe otra caja etiquetada "fin" de igual forma que en a.
c. Si existen otras cajas, normalmente son rectangulares, tipo rombo o paralelogramo

Se puede escribir más de un paso del algoritmo en un sola rectangular. El uso de flechas significa que la caja no necesita ser escrita debajo de su predecesora. Sin embargo, abusar demasiado de esta flexibilidad conduce a diagramas de flujo complicados e ininteligibles.

PSEUDOCODIGO

El pseudocódigo es un lenguaje de especificación de algoritmos. El uso de tal lenguaje hace el paso de codificación final relativamente fácil.

El pseudocódigo nació como un lenguaje, similar al ingles y era un medio de representar básicamente las estructuras de control de programación estructurado. Se considera un primer borrador, dado que el pseudocódigo tiene que traducirse posteriormente a un lenguaje de programación. El pseudocódigo no puede ser ejecutado por una computadora. La ventaja del pseudocódigo es que en su uso en la planificación de un programa, el programador se puede concentrar en la lógica y en las estructuras de control y no preocuparse de las reglas de un lenguaje especifico. Es también fácil modificar el pseudocódigo si se descubren errores o anomalías en la lógica del programa, mientras que en muchas ocasiones suele ser difícil el cambio en la lógica una vez que está codificado en un lenguaje de programación.

El pseudocódigo utiliza para representar las acciones sucesivas palabras reservadas en inglés, similares a sus homónimas en los lenguajes de programación, tales como **end, if-then-else, while, etc.** La escritura del pseudocódigo exige normalmente la indentación de diferentes líneas.

3. ESCRITURA DEL ALGORITMO

La escritura de un algoritmo mediante una herramienta de programación debe ser lo más clara posible y estructura de modo que su lectura facilite considerablemente el entendimiento del algoritmo y su posterior codificación en un lenguaje de programación.

Los algoritmos deben ser escritos en lenguajes similares a los programas. Por ahora, utilizaremos el lenguaje algoritmo basado en pseudocódigo.

Un algoritmo constará de dos componentes: una cabecera y un cuerpo.

La cabecera es una acción simple que comienza con las acciones de declaración que definen o declaran las variables y constantes que se utilizarán en la rutina.

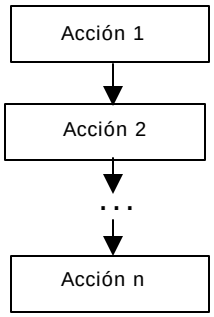
El cuerpo es un bloque algoritmo que consta de las acciones ejecutables donde las acciones ejecutables son las acciones que posteriormente deberá realizar la computadora cuando el algoritmo sea convertido en programa ejecutable.

4. ESTRUCTURAS DE CONTROL

Es la manera como se van encadenando, uniendo entre sí las acciones, dando origen a los distintos tipos de estructuras.

1. Estructura secuencial

La estructura secuencial es aquélla en la que una acción sigue a otra en secuencia. Las tareas se suceden de tal modo que la salida de una es la entrada de la siguiente y así sucesivamente hasta el fin del proceso.



Acción 1
Acción 2
...
Acción n

2. Estructuras selectivas

Cuando el programa desea especificar dos o más caminos alternativos, se deben utilizar estructuras selectivas o de decisión.

Una instrucción de decisión o selección evalúa una condición y en función del resultado de esa condición se bifurcará a un determinado punto.

La especificación formal de algoritmo tiene realmente utilidad cuando el algoritmo requiere una descripción más complicada que una lista sencilla de instrucciones. Éste es el caso cuando existe un número de posibles alternativas resultantes de la evaluación de una determinada condición.

Las estructura selectivas se utilizan para tomar decisiones lógicas; de ahí que se suelan denominar también estructuras de decisión o alternativas.

En la estructura selectiva se evalúa una condición y en función del resultado de la misma se realiza una opción u otra. Las condiciones se especifican usando expresiones lógicas. La representación de una estructura selectiva se hace con palabra en pseudocódigo, con una figura geométrica en forma de rombo.

Las estructuras selectivas o alternativas pueden ser:

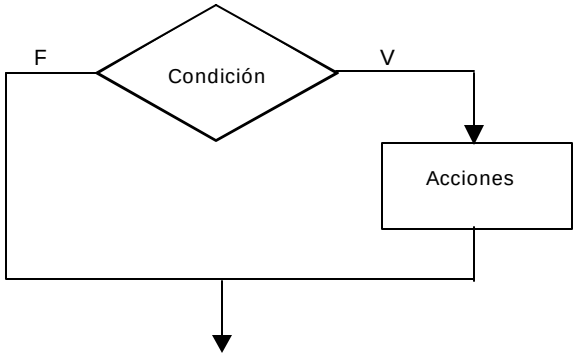
- Simples
- Dobles.
- Múltiples.

Alternativa simple

La estructura alternativa simple **si_entonces**, ejecuta una determinada acción cuando se cumple una determinada condición. La selección **si-entonces** evalúa la condición:

- Si la condición es **verdadera,** entonces ejecuta la acción (simple o contar de varias acciones)
- Si la condición es **falsa,** entonces no hace nada.

La representación gráfica de la estructura condicional simple se ve a continuación.



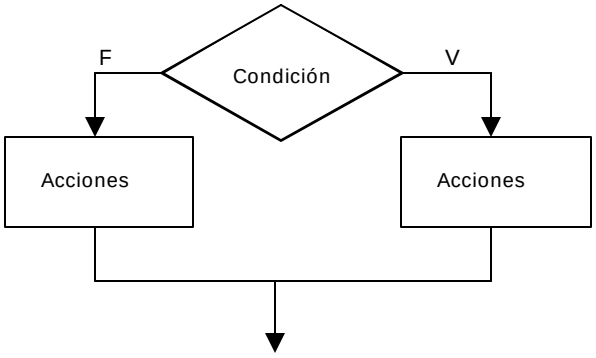
Si condición **Entonces**
Acciones
Fin_Si

Observe que las palabras del pseudocódigo Si y Fin_Si se alinean verticalmente indentando (sangrando) la acción o bloque de acciones.

Alternativa doble

La estructura anterior es muy limitada y normalmente se necesitará una estructura que permita elegir entre dos opciones o alternativas posibles, en función del cumplimiento o no de una determinada condición.

Si la condición es verdadera, se ejecuta la acciónes_1 y si es falsa, se ejecuta la acciónes_2.



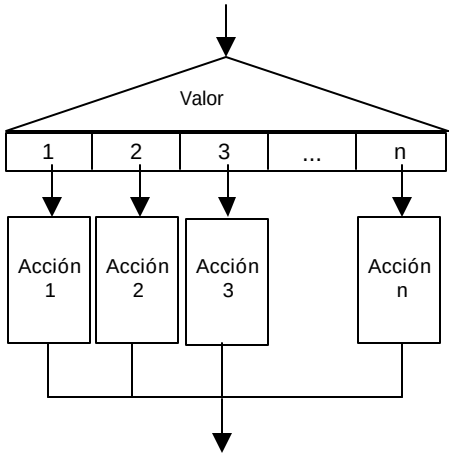
Si Condición **Entonces**
Acciones
Sino
Acciones
Fin_Si

Observe que en el pseudocódigo las acciones que dependen de entonces y sino están indentadas en relación con las palabras Si y Fin_Si; este procedimiento aumenta la legibilidad de la estructura y es el medio idóneo para representar algoritmos.

Alternativa múltiple

Con frecuencia, es necesario que existan más de dos elecciones posibles. Por ejemplo: en la resolución de la ecuación de segundo grado existen tres posibles alternativas o caminos a seguir, según que el discriminante sea negativo, nulo o positivo. Este problema, se podría resolver por estructuras alternativas simples o dobles, anidadas o en cascada; sin embargo, con este método, si el número de alternativas es grande puede plantear serios problemas de escritura del algoritmo y naturalmente de legibilidad.

La estructura de decisión múltiple evaluará una expresión que podrá tomar n valores distintos: 1, 2, 3,..., n. Según que elija uno de estos valores en la condición, se realizará una de las n acciones, o lo que es igual, el flujo del algoritmo seguirá sin determinado camino entre los n posibles.



Según Valor **Hacer**

Valor 1: Acción 1

Valor 2: Acción 2

Valor 3: Acción 3

...

Valor n: Acción n

Fin Según

La estructura de decisión múltiple en pseudocódigo se puede representar de diversas formas, pudiendo ser las acciones, simples o compuestas.

3. Estructuras repetitivas

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

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

Un bucle o lazo (loop) es un segmento de un algoritmo o programa, cuyas instrucciones se repiten un número determinado de veces mientras se cumple una determinada *condición.* Se debe establecer un mecanismo para determinar las tareas repetitivas. Este mecanismo es una condición que puede ser verdadera o falsa y que se comprueba una vez a cada paso o iteración del bucle.

Un bucle consta de tres partes:

- Decisión.
- Cuerpo del bucle.
- Salida del bucle.

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

```
Inicio
   suma = 0
   leer número
   suma = suma
   leer número
   suma =suma
      ...
fin
```

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

Las dos principales preguntas a realizarse en el diseño de un bucle son: ¿Qué contiene el bucle? y ¿Cuántas veces se debe repetir?

Cuando se utiliza un bucle para sumar una lista de números, se necesita saber cuántos números se han de sumar. Para ello necesitaremos conocer algún medio para detener el bucle. Existen dos

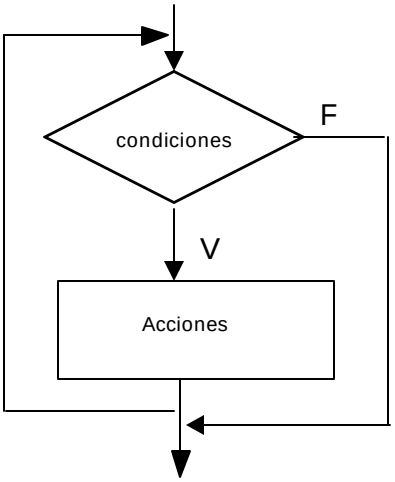
procedimientos para contar el número de iteraciones, usar una variable que se inicializa a la cantidad de números que se desea y a continuación se decrementa en uno cada vez que el bucle se repite**,** o bien inicializar la variable en cero o en uno, e ir incrementando en uno a cada iteración hasta llegar al número deseado.

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

1. La condición de salida del bucle se realiza al principio del bucle (estructura mientras)
2. La condición de salida se origina al Final del bucle; el bucle se ejecuta mientras o hasta que se verifique una cierta condición.
3. La condición de salida se realiza con un contador que cuenta el número de iteraciones.

Estructura mientras

La estructura repetitiva mientras es aquella en la que el bucle se repite mientras se cumple una determinada condición. La representación gráfica es:

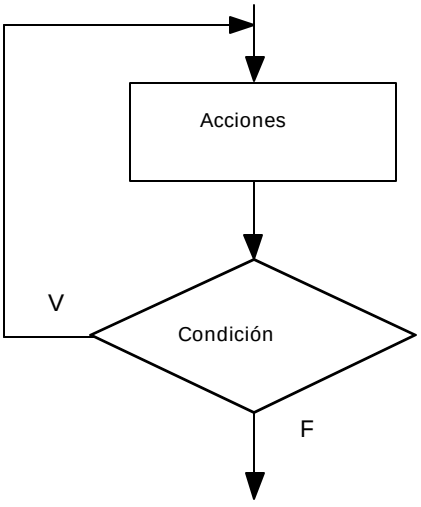


Mientras condición **Hacer**
Acciones
Fin_Mientras

Cuando se ejecuta la instrucción mientras, la primera cosa que sucede es que se evalúa la condición (una expresión booleana). Si se evalúa falsa, ninguna acción se toma y el programa prosigue en la siguiente instrucción del bucle. Si la expresión booleana es *verdadera,* entonces se ejecuta el cuerpo del bucle, después de lo cual se evalúa de nuevo la expresión booleana. Este proceso se repite una y otra vez mientras la expresión **(condición)** sea verdadera.

Estructura repetir o hacer mientras

Existen muchas situaciones en las que se desea que un bucle se ejecute al menos una vez antes de comprobar la condición de repetición. En la estructura **mientras** si el valor de la expresión booleana es inicialmente falsa, el cuerpo del bucle no se ejecutará; Por ello se necesitan otros tipos de estructuras repetitivas. La estructura **repetir** o **hacer mientras** se ejecuta hasta ó mientras que se cumpla una condición determinada que se comprueba al final del bucle.



Hacer
Acciones
Mientras Condición **Repetir**
ó
Repetir
Acciones
Hasta Condición

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

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

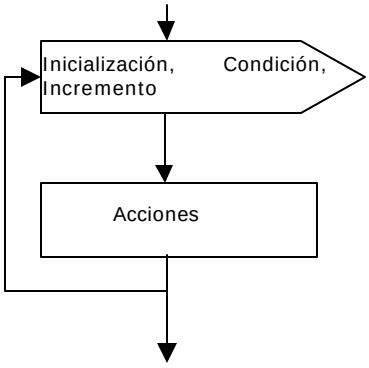
DIFERENCIAS DE LAS ESTRUCTURAS MIENTRAS Y REPETIR

Estructura Mientras	Estructura Hacer o Repetir
Es un ciclo de 0 a N, porque si la condición inicial no se cumple no se entra al ciclo, por lo tanto existe la posibilidad de que las aciones del ciclo no sean ejecutadas.	Es un ciclo de 1 a N, porque siempre se ejecuta al menos una vez las instrucciones del ciclo, si la primera vez la condición inicial es falsa, al evaluar la condición al final, se termina el ciclo pero ya se ejecutaron por lo menos una vez las acciones.

Estructura desde /para

En muchas ocasiones se conoce de antemano el número de veces que desean ejecutar las acciones de un bucle. En estos casos, en que el número de iteraciones es fija, se debe usar la estructura **desde** o **para.**

La estructura desde ejecuta las acciones del cuerpo del bucle un número especificado de veces y de modo automático controla el número de iteraciones o pasos a través del cuerpo del bucle.



Desde inicialización, Condición, imcremento **Hacer**

Acciones

Fin_Desde

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

El incremento de la variable índice siempre es 1 si no se indica expresamente lo contrario. Es posible que el incremento sea distinto de uno, positivo o negativo.

La variable índice o de control normalmente será de tipo entero y es normal emplear como nombre las letras: 1, **J,** K.

El formato de la estructura desde varia si se desea un incremento distinto a 1, bien positivo, bien negativo (decremento).

Si el valor inicial de la variable índice es menor que el valor **final,** los incrementos deben ser positivos, ya que en caso contrario la secuencia de acciones no se ejecutaría. De igual modo si el valor **inicial** es mayor que el valor final, el incremento debe de ser en este caso **negativo,** es decir, **decremento.** Al **incremento** se le suele denominar también paso.

PROGRAMAS

1. CONCEPTO DE PROGRAMA

Un programa de computadora es un conjunto de instrucciones, ordenes dadas a la maquina, que producirán la ejecución de una determinada tarea.

2. ORGANIZACION DE LOS PROGRAMAS

en proceso...

3. PROGRAMAS TIPOS

Existen programas tipos que se encuentran en todas las aplicaciones y consisten en programas que responden siempre al mismo comportamiento, ellos son:

Altas

Realizan las operaciones de ingreso de datos, como alta de productos, clientes, proveedores, etc. Generalmente estas operaciones consisten en una carga de datos, verificación de esa carga y grabación de la misma en un archivo de datos o en una estructura en memoria.

Modificaciones

Realizan las operaciones de modificaciones de datos cargados en un archivo de datos o en una estructura en memoria, por ejemplo un archivo de productos, clientes, proveedores, etc. Generalmente estas operaciones consisten en una búsqueda del elemento a modificar, carga de las modificaciones, verificación de esa carga y grabación de las mismas en el archivo o estructura correspondiente.

Bajas

Realizan las operaciones de eliminación de datos cargados en un archivo de datos o en una estructura en memoria, por ejemplo un archivo de productos, clientes, proveedores, etc. Generalmente estas operaciones consisten en una búsqueda del elemento a eliminar, verificación del elemento y eliminación o borrado del mismo en el archivo correspondiente.

Consultas

Realizan las operaciones de consulta de datos cargados en un archivo de datos o en una estructura en memoria. Dichas consultas se pueden mostrar en pantalla, impresora o archivo, y pueden ser individuales o grupales.

Individuales: Realizan la consulta de un elemento en particular. Generalmente estas operaciones consisten en una búsqueda del elemento a consultar y muestra de los todos los datos del mismo o de los que solicite el usuario.

Grupales: Realizan la consulta de todos los elementos existentes en un archivo de datos o en una estructura en memoria. Generalmente estas operaciones consisten en la elección del archivo de datos a consultar y muestra de los todos los datos del mismo o de los que solicite el usuario.

Listados

Realizan las mismas operaciones que las consultas pero los resultados sólo se muestran en impresora.

Movimientos o transacciones

Realizan las operaciones de carga o modificación de datos de movimientos regulares, tales como, planillas de compras, ventas, asistencias, etc.

Carga: Implica la grabación de ese movimiento en un archivo de datos o en una estructura en memoria. Generalmente estas operaciones consisten en una carga de datos comunes al movimiento (número, fecha, etc.), carga del detalle del movimiento (ítem, cantidad, precio, etc.), verificación de esa carga y grabación de la misma en un archivo de datos o en una estructura en memoria.

Modificación: Implica la modificación de ese movimiento en un archivo de datos o en una estructura en memoria. Generalmente estas operaciones consisten en la búsqueda del movimiento a modificar, la verificación de esa modificación y la grabación de la misma en un archivo de datos o en una estructura en memoria. Las modificaciones de los movimientos o transacciones no siempre se realiza, depende del movimiento de que se trate.

Mantenimiento

Son programas auxiliares o adicionales que se agregan al programa principal o aplicación, sirve para realizar aquellas operaciones que el programador considere de importancia para completar la aplicación, tales como, programas de seguridad, de resguardo (Backup), etc.

4. ELEMENTOS BASICOS DE UN PROGRAMA

En programación se debe separar la diferencia entre el diseño del algoritmo y su implementación en un lenguaje específico. Por ello se debe distinguir claramente entre los conceptos de programación y el medio en que ellos se implementan en un lenguaje específico. Sin embargo, una vez que se comprendan los conceptos de programación y cómo utilizarlos, la enseñanza de un nuevo lenguaje es relativamente fácil.

Los lenguajes de programación tienen elementos básicos que se utilizan como bloques constructivos, así como reglas para las que esos elementos se combinan. Estas reglas se denominan *sintaxis* del lenguaje. Solamente las instrucciones sintácticamente correctas pueden ser interpretadas por la computadora y los programas que contengan errores de sintaxis son rechazados por la máquina.

Nosotros ya vimos que el elemento o unidad básica de un programa orientado a objetos es la clase, pero también existen ciertos elementos que se encuentran en todo programa y que veremos brevemente a continuación:

1. Palabras reservadas.
2. Comentarios.
3. Tipos de datos
4. Variables.
5. Operadores
6. Expresiones
7. Instrucciones.

8. Funciones.

Además de estos elementos básicos existen otros elementos que forman parte de los programas, cuya comprensión y funcionamiento será vital para el correcto diseño de algoritmo y naturalmente la codificación del programa.

9. Contadores.
10. Acumuladores.
11. Banderas.

El amplio conocimiento de todos los elementos de programación y el modo de su integración en los programas constituyen las técnicas de programación que todo buen programador debe conocer.

1. PALABRAS RESERVADAS: Son palabras propias ó nativas de cada lenguaje que el programador debe utilizar según las especificaciones del mismo.

2. COMENTARIOS: Forman parte de la documentación interna de un programa, se encuentran acompañando al código fuente en los lugares que se necesite un comentarios significativo del mismo.

3. TIPOS DE DATOS: Es una descripción del conjunto de valores que puede tomar un dato.

Los tipos de datos que se utiliza en los programas, en general, son los siguientes:

DATOS CARÁCTER: El tipo carácter es una sucesión de caracteres que se encuentran delimitados por una comilla o dobles comillas, según el tipo de lenguaje de programación.

DATOS NUMERICOS: El tipo numérico es el conjunto de los valores numéricos. Se representan como enteros y reales.

DATOS LOGICOS (BOOLEANOS) El tipo lógico, también denominado booleano, es aquel dato que sólo puede tomar uno de dos valores: cierto o verdadero(true) y falso (false)

4. VARIABLES: Una variable es una unidad de almacenamiento identificada por un nombre y un tipo de dato, tiene una dirección (ubicación) dentro de la memoria y puede almacenar información según el tipo de dato que tenga asociado. Es necesario definir las variables de un programa antes de que puedan ser utilizadas.

5. OPERADORES: Símbolos de operación que se utilizan para procesar, comparar o relacionar distintos valores.

OPERADORES ARITMÉTICOS: +, -, *, /
OPERADORES LÓGICOS: and, or.
OPERADORES RELACIONALES: >, <, >=, =<, =, !=
OPERADORES DE ASIGNACIÓN: =

6. EXPRESIONES: Es la combinación de operadores y operandos. Una expresión es evaluada por el lenguaje dando lugar principalmente a dos tipos de resultados, basándose en los cuales podemos dividir las expresiones en aritméticas y condicionales. Las primeras se caracterizan por generar un resultado numérico, mientras que las segundas sólo pueden devolver como resultado los valores verdadero (true) y falso (false).

Las expresiones son combinaciones de constantes, variables, símbolos de operación, paréntesis y nombre de funciones especiales. Las mismas ideas son utilizadas en notación matemática tradicional; por ejemplo:

$a + b*(b + 3) + (a / 10)$

Cada expresión toma un valor que se determina tomando los valores de las variables y constantes implicadas y la ejecución de las operaciones indicadas.

7. INSTRUCCIONES: Son las acciones que se ejecutan en un computador, generalmente hay instrucciones de inicio y fin, de asignación, de lectura, de escritura y de salto o bifurcación.

8. FUNCIONES : Una función, llamada también subrutina, es un conjunto de instrucciones o bloque de código independiente. En general poseen un nombre y un cuerpo que es donde se escriben las instrucciones. Estas funciones son lo que conforman los métodos de una clase.

9. CONTADORES: Es una variable que a partir de un valor inicial, aumenta o disminuye en una cantidad constante.

forma general

contador = valor inicial
contador = contador + paso *paso* es el valor constante que se suma o resta en cada repetición

ejemplo		
	c = 0	c=120
	c = c + 1	c = c + 10

Los procesos repetitivos son la base del uso de las computadoras. En estos procesos se necesitan normalmente contar los sucesos o acciones internas del bucle, como pueden ser los elementos de un fichero, el numero de iteraciones a realizar por el bucle, etc. Una forma de controlar un bucle es mediante un contador.

10. ACUMULADORES: Es una variable que a partir de un valor inicial, aumenta o disminuye en una cantidad no constante. Se utilizan para sumar o restar variables afines.

forma general

```
contador = valor inicial
contador = contador + cantidad        cantidad se suma o resta en cada repetición
```

ejemplo

```
c = 0                cant=120
c = c + cant
```

Un acumulador es una variable cuya misión es almacenar cantidades variables resultantes de sumas sucesivas. Realiza la misma función que un contador con la diferencia de que el incremento o decremento de cada suma es variable en lugar de constante como en el caso del contador.

11. BANDERAS: Es una variable que puede tomar diversos valores a lo largo de la ejecución del programa y que permite comunicar información de una parte a otra del mismo. Por lo general toman dos valores (ej. 0 y 1) y se utilizan para saber si se ha ejecutado o no una parte del proceso. De acuerdo al valor que tenga en el momento en que se le testea se puede determinar si se ejecuta o no esa parte del proceso.

LENGUAJES DE PROGRAMACION

Los lenguajes de programación son los medios de comunicación entre los programadores y la computadora, con ellos se construyen los programas que luego serán ejecutados por la misma. Como ya vimos, un programa es un conjunto de instrucciones que, al ser ejecutadas, llevan a cabo una tarea o función específica. Existen dos categorías de lenguajes:

Lenguajes de Bajo Nivel: En estos cada línea de programa que se escribe con ellos es una orden para la computadora.

Lenguaje ensamblador: Cada línea de programa se construye con unas instrucciones y símbolos específicos, cada una de las cuales tiene un nombre.

Lenguaje de maquina: Las instrucciones o líneas de programas son cadenas de 0 y o de caracteres hexadecimales.

Lenguajes de Alto Nivel: Lenguaje de programación simbólico, parecido a las lengua naturales, lo que facilita su manejo y aprendizaje.

Los lenguajes de alto nivel y el lenguaje ensamblador, antes de ser ejecutados, se traducen a lenguaje de máquina, único lenguaje entendido por la computadora.

TRADUCTORES

La computadora sólo puede ejecutar instrucciones escritas en lenguaje formado por secuencias de ceros y unos, al que normalmente se denomina lenguaje de máquina. Por ello, cualquier lenguaje de programación que no sea lenguaje de máquina, para poder ejecutarse, necesitará un proceso de traducción. La forma en que se ejecutará un programa está referida a los términos **intérpretes** y **compiladores**. En teoría, cualquier lenguaje de programación puede ser compilado o interpretado, pero algunos lenguajes se ejecutan normalmente de una forma o de otra. La forma en que se ejecuta un programa no está definida por el lenguaje en el que se escribe. Los intérpretes y compiladores simplemente son programas sofisticados que funcionan sobre su código de programa fuente.

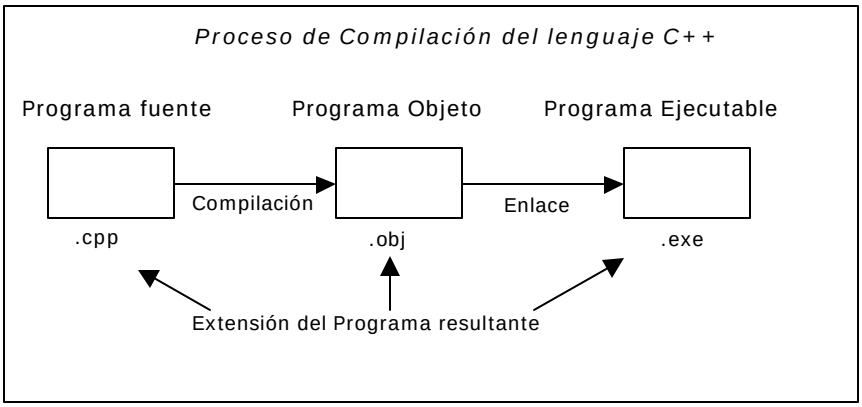
Intérpetre: Lee el código fuente de su programa de línea en línea, traduce a código de máquina y ejecuta las instrucciones específicas que están en esa línea. Lenguajes que utilizan intérpretes son Basic, Fox, etc.

Compilador: Lee el programa entero y lo convierte en código objeto, que es una traducción del código fuente, de forma que el compilador pueda ejecutarlo directamente. Lenguajes que utilizan compiladores son C y C++, Pascal, etc. Con esta forma utilizaremos dos términos que describiremos a continuación:

Tiempo de compilación: se refiere a los sucesos que ocurren durante el proceso de compilación.

Tiempo de ejecución: se refiere a los sucesos que ocurren mientras se ejecuta el programa.

Por ejemplo, en el gráfico siguiente se muestra el proceso de compilación del lenguaje C++



CLASE PRACTICA

Para la clase práctica se sugiere ejercicios que identifiquen clases y que grafiquen el cuerpo de los métodos. Por ejemplo tomando el ejercicio de la primera semana

PROBLEMA

Dado el valor de los tres lados de un triángulo, calcular el perímetro.

ANALISIS

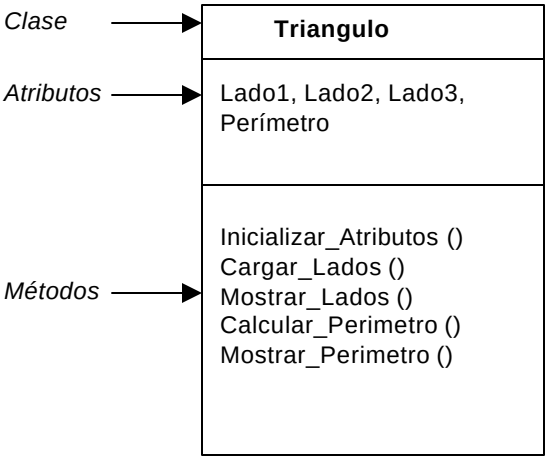
1. Identificar el objeto: identificar la entidad que engloba al problema, en nuestro ejemplo es el triángulo.

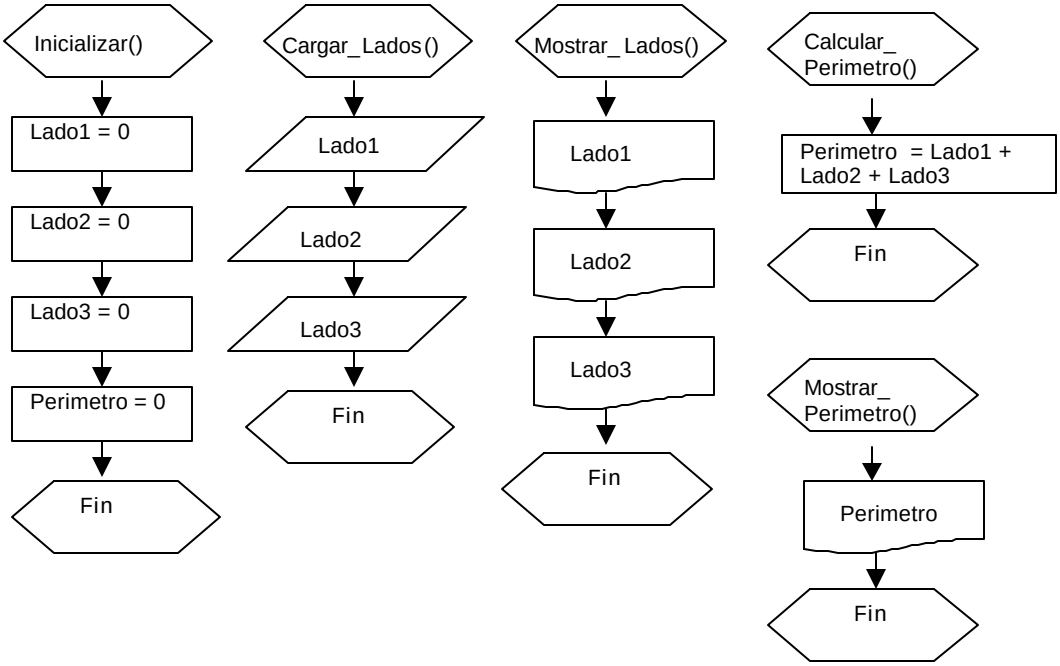
2. Identificar sus atributos: podemos decir que todo triángulo tiene como datos esenciales los tres lados, además, nuestro problema en particular necesitará el valor del perímetro, que es un datos calculable a partir de los tres lados y que puede figurar como atributo o no dependiendo si se quiere guardar su estado en el objeto. Como nuestro enunciado no nos restringe la forma de implementarlo vamos a definir como atributos los tres lados y el perímetro.

3. Identificar sus métodos: los métodos a elegir siempre dependerán de las operaciones que podemos realizar con los atributos, entonces podemos enumerar:

Inicializar los atributos: Inicializar en 0, por ejemplo, los lados y el perímetro.
Ingresar los lados: Se ingresa el valor de cada lado.
Mostrar los lados: Mostrar el valor de los lados.
Calcular el perímetro: formula que devuelve el perímetro del triángulo.
Mostrar el perímetro: Mostrar el valor del perímetro.

Graficando el resultado


Clase
Triangulo
Atributos
Lado1, Lado2, Lado3,
Perímetro
Métodos
Inicializar_Atributos ()
Cargar_Lados ()
Mostrar_Lados ()
Calcular_Perimetro ()
Mostrar_Perimetro ()


Inicializar()
Lado1 = 0
Lado2 = 0
Lado3 = 0
Perimetro = 0
Fin
Cargar_Lados()
Lado1
Lado2
Lado3
Fin
Mostrar_Lados()
Lado1
Lado2
Lado3
Fin
Calcular_
Perimetro()
Perimetro = Lado1 +
Lado2 + Lado3
Fin
Mostrar_
Perimetro()
Perimetro
Fin