

Ciclo Formativo de Grado Superior de Administración de Sistemas Informáticos en red



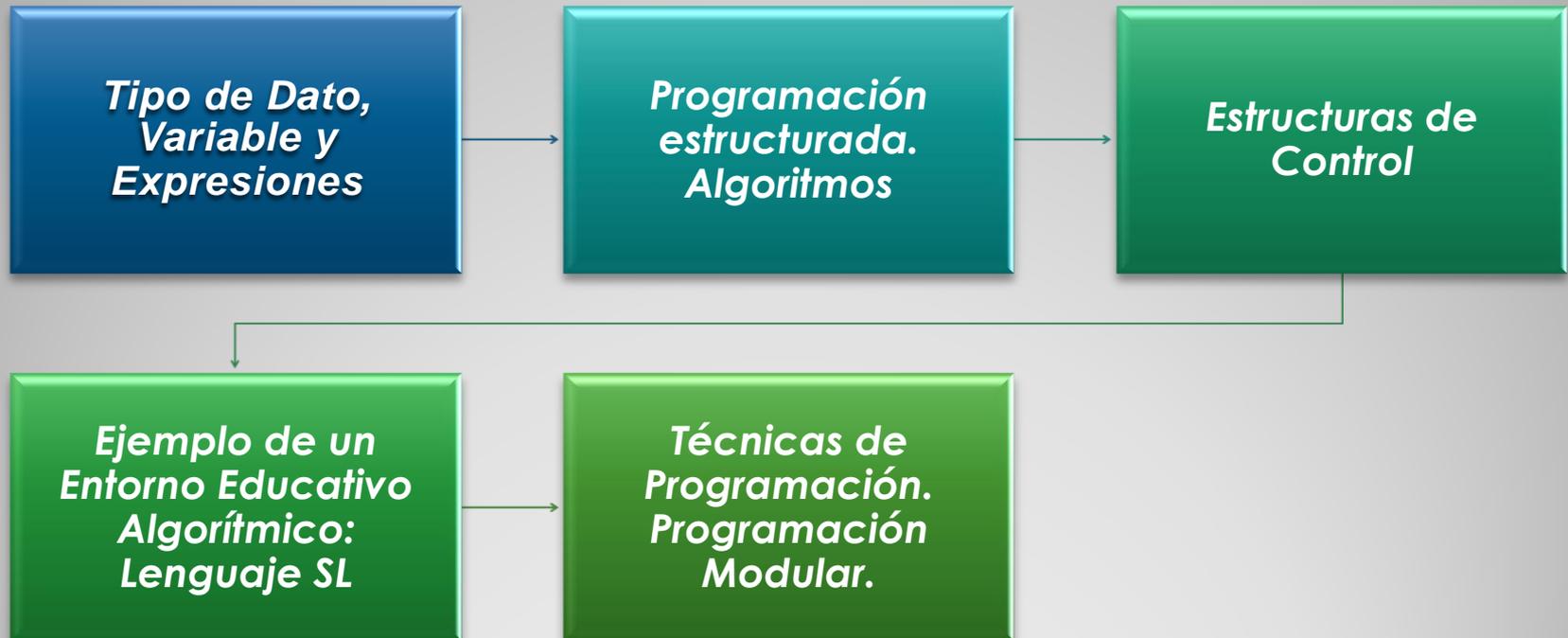
Introducción a la Programación Estructurado/Secuencial y Funcional.

*Departamento de Informática y Comunicación
IES San Juan Bosco (Lorca-Murcia)
Profesor: Juan Antonio López Quesada*

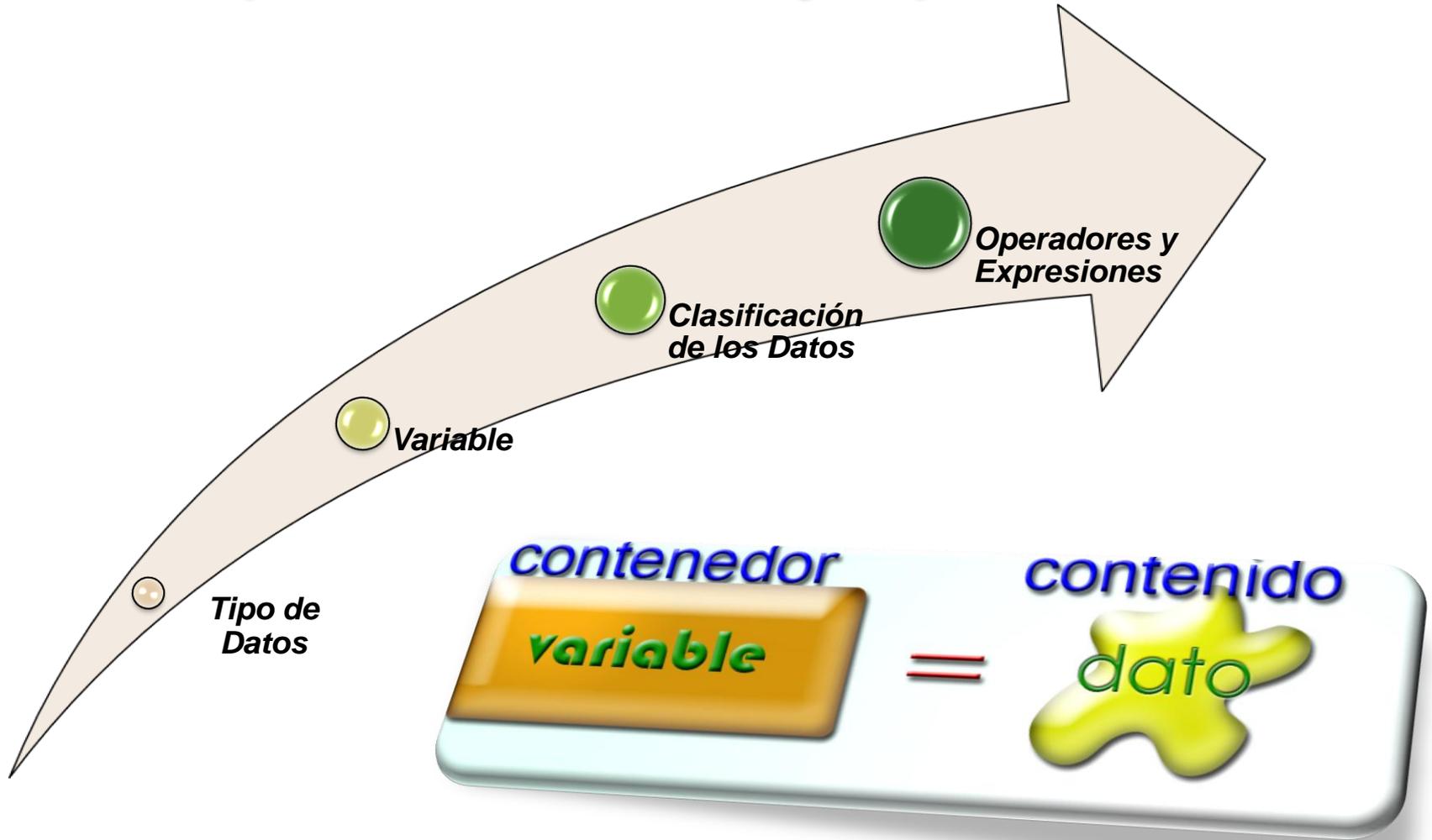




INDICE DE CONTENIDOS



Tipo de Dato, Variable y Expresiones



Tipo de Dato

Es uno de los conceptos fundamentales en el ámbito de los lenguajes de programación y aun más en el marco de los lenguajes estructurados. Un tipo de dato define un conjunto de posibles valores que pueden asociarse, un modo o forma de representación interna y un conjunto de operación en las que puede participar esos posibles valores.

Tenemos que indicar que existen fundamentalmente dos tipos de datos, los denominados predefinidos, que son aquellos que por defecto nos proporciona el lenguaje de programación que se hubiera elegido para el desarrollo/codificación del sistema de información y los tipos creados por el usuario.

Como ejemplo de tipo de dato en C predefinido tenemos int, flota.. etc. y mediante el token/palabra reservada typedef se puede definir nuevos tipos.

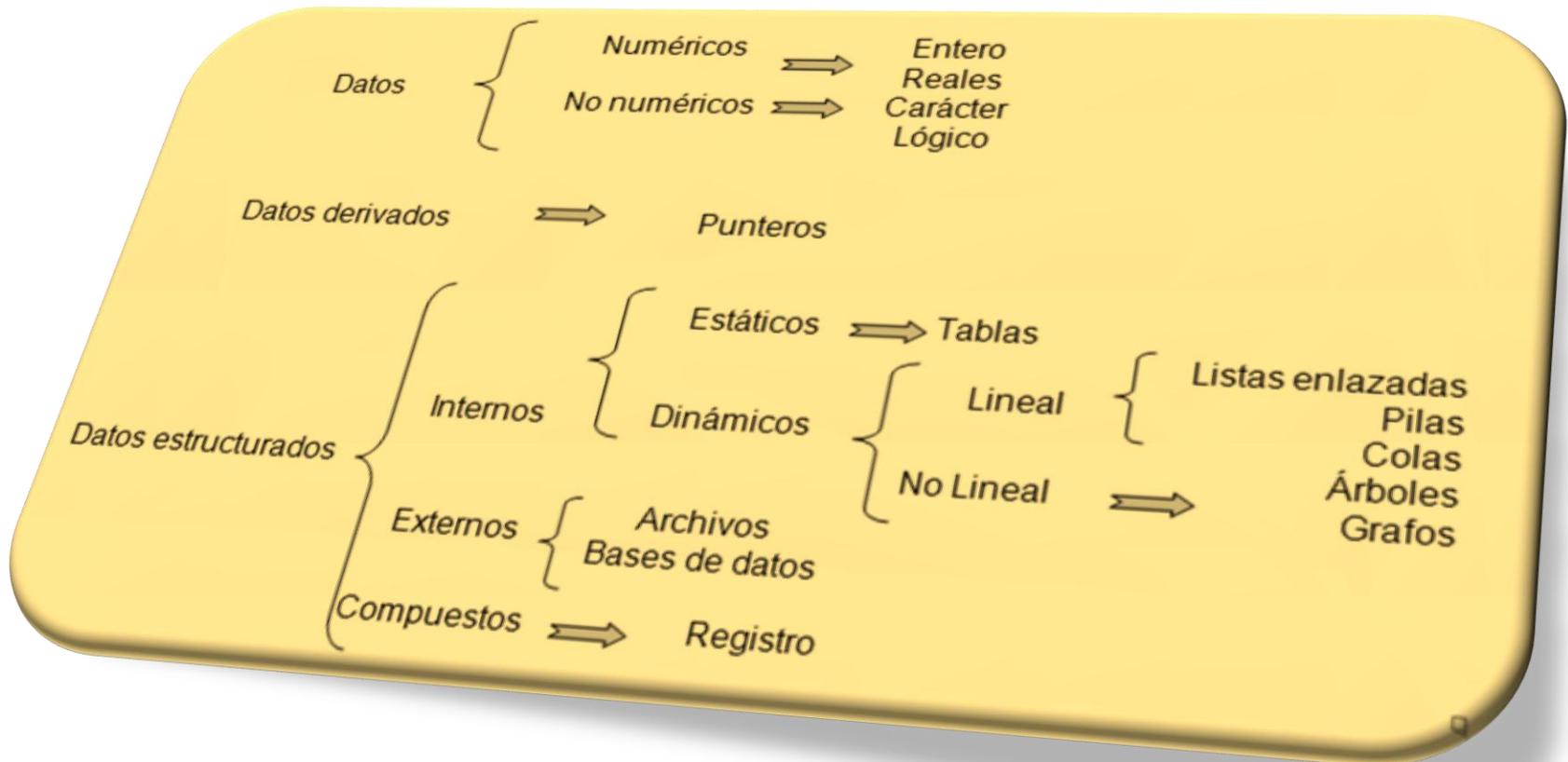
Variable

Para el diseño de un programa es importante determinar que datos voy a utilizar. Estos datos serán almacenados en memoria RAM y para su manipulación necesitaremos poder llamarlos, por lo que debemos definir y utilizar el concepto de variable. Una variable se define como un identificador, un tipo y un valor. Es evidente que el valor que tendrá esa variable será una de los posibles valores en función del tipo y que además dicho valor estará en memoria y por eso el identificador es la manera que tenemos para nombrarlo.

```
package javaapplication;
public class Main {
public static void main(String[] args) {
    int contador=5; // Variable con identificador o nombre
                    contador, tipo entero con signo y valor inicial de 5
}
}
```

Clasificación de los Datos

Una clasificación de los posibles tipos de datos que suelen encontrarse en cualquier lenguaje de programación es la siguiente:



Operadores y Expresiones

Operador

Son símbolos que sirven para conectar los datos facilitando la realización de diversas clases de operaciones.

Tipos de operadores

Significado

Paréntesis () Paréntesis

Aritméticos

** , ^	Potencia
*	Producto
/	División
div. \	División entera
%, mod.	Modulo – Resto de la división entera
+	Suma
-	Resta

Operadores y Expresiones

Alfanuméricos

+ Concatenación
- Concatenación eliminando espacios

Relacionales

= =, = Igual a
i=, <> Distinto a
< Menor
<= Menor o igual
> Mayor
>= Mayor o igual

Lógico

!, NOT, no Negación
&&, AND, y Conjunción
||, OR, o Disyunción

Operadores y Expresiones

Tablas de verdad

En las operaciones lógicas se determina su resultado por medio de las tablas de verdad, suponiendo que "A" y "B" son expresiones lógicas y que "V" es verdadero y "F" es falso, la tabla de verdad de los principales operadores lógicos sería:

A	B	A AND B	A OR B	A XOR B	NOT A	NOT B
V(1)	V(1)	V(1)	V(1)	F(0)	F(0)	F(0)
V(1)	F(0)	F(0)	V(1)	V(1)	F(0)	V(1)
F(0)	V(1)	F(0)	V(1)	V(1)	V(1)	F(0)
F(0)	F(0)	F(0)	F(0)	F(0)	V(1)	V(1)

Operadores y Expresiones

Orden de prioridad de los operadores

El orden de prioridad es el siguiente:

- ✚ Los paréntesis y siempre de los más internos a los más externos.
- ✚ Operador signo
- ✚ Potencia
- ✚ Producto, división y modulo con la misma prioridad, siempre de izquierda a derecha.
- ✚ Suma y resta
- ✚ Concatenación
- ✚ Operadores relacionales
- ✚ Negación
- ✚ Conjunción
- ✚ Disyunción

Operadores y Expresiones

Expresiones

Las expresiones son un conjunto de datos (operandos) y operadores con reglas específicas de construcción. Los operandos pueden ser literales, variables, constantes y expresiones. En la obtención del resultado se debe tener en cuenta el orden de la prioridad de los operadores.

En función del resultado que se obtiene se pueden clasificar en:

Numéricas: Tanto los operandos como los operadores son aritméticos.

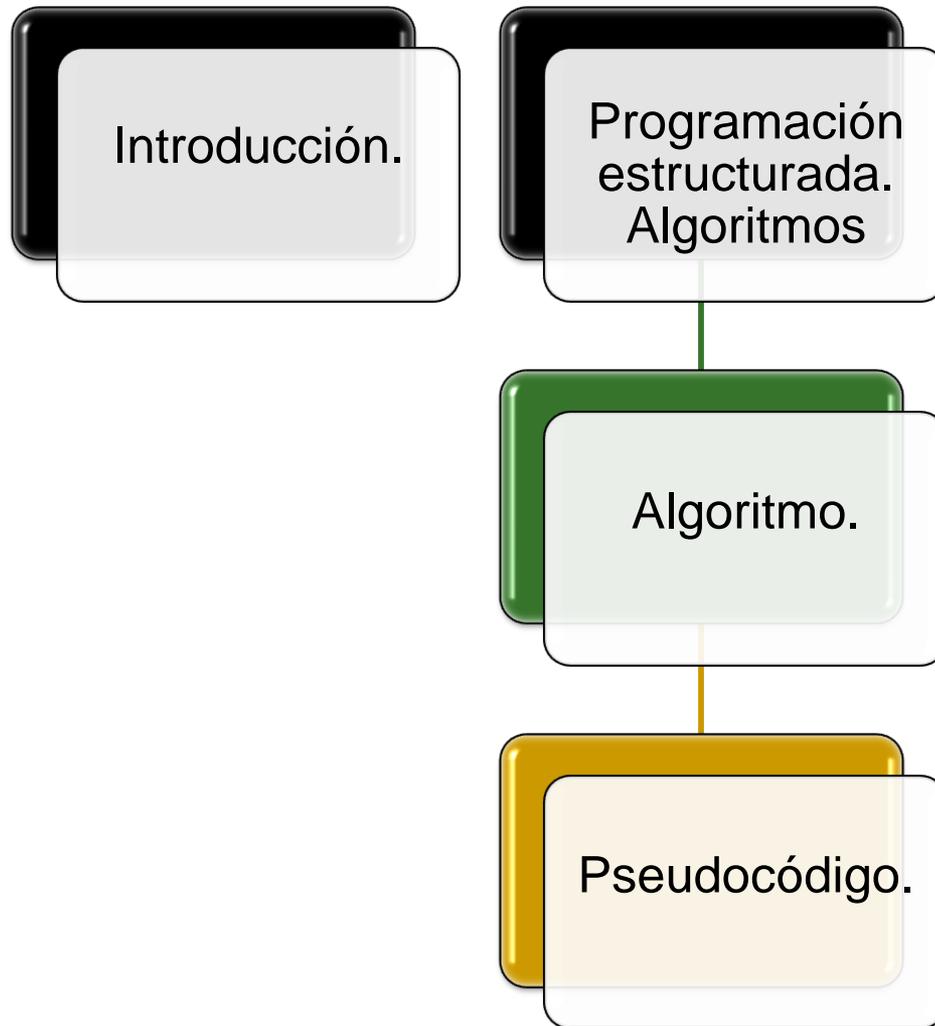
Alfanumérica: Su resultado es una cadena de caracteres y utilizan operadores aritméticos.

Relacionales: Los operadores son relacionales y los operandos no pueden ser expresiones que al evaluarse den un valor lógico.

Booleanas: Su resultado será un valor de verdad. Ej.: Precio < 250 y total >30000. Precio = 200, Total = 55000. El resultado sería verdadero y verdadero por tanto el valor será verdadero.

Indicar que en cada lenguaje las expresiones tienen sus propias reglas que determinan como se construyen, como se evalúa y cual es tipo de dato resultante en función de sus componentes.

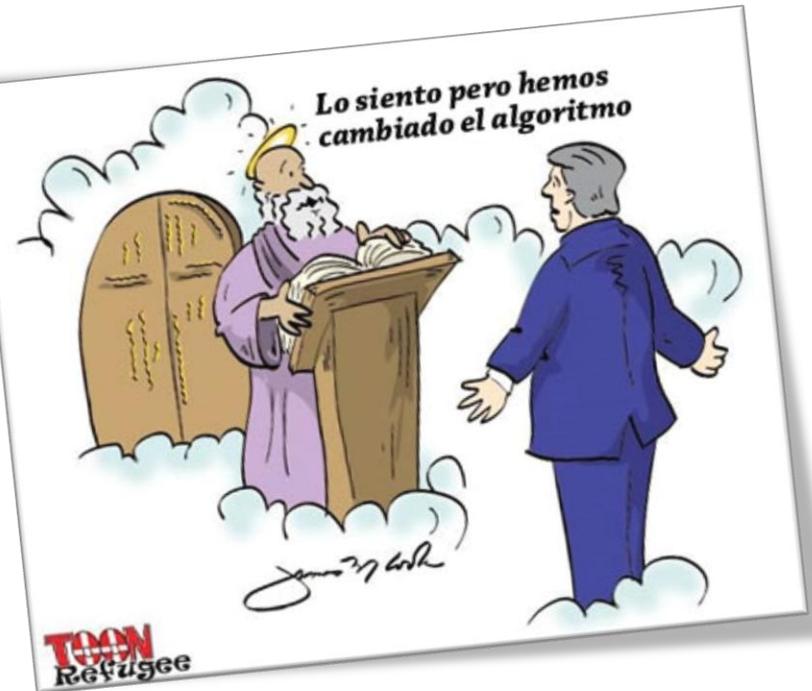
Programación Estructurada. Algoritmos



Abstract/Resumen:

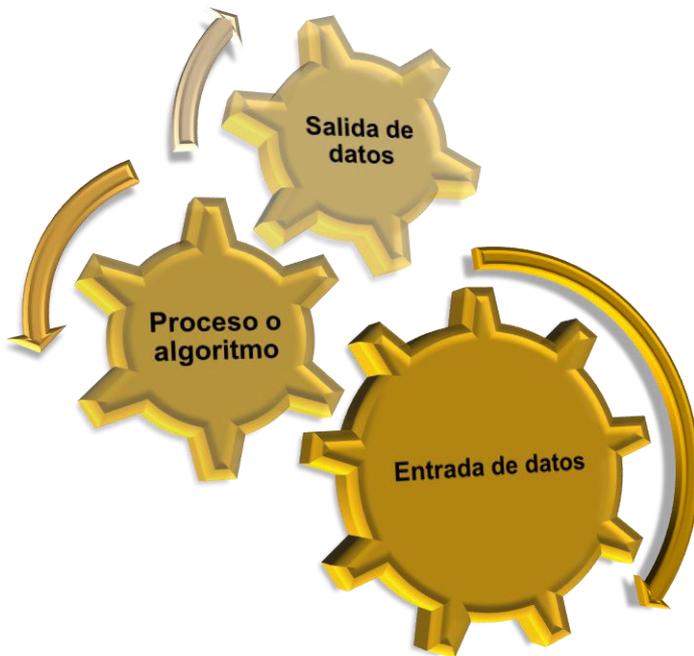
La palabra algoritmo se deriva de la traducción al latín de la palabra árabe alkhwarizmi, nombre de un matemático y astrónomo árabe que escribió un tratado sobre manipulación de números y ecuaciones en el siglo IX. Un algoritmo es una serie de pasos organizados que describe el proceso que se debe seguir, para dar solución a un problema específico.

El desarrollo de algoritmos es un tema fundamental en el diseño de programas o soluciones. En este apartado se pretende afianzar dos elementos fundamentales: conocer, comprender y manejar las diversas ESTRUCTURAS DE CONTROL, y mediante un pseudocódigo resolver problemas reales. Esto permitirá crear en el alumno una buena base que le sirvan para poder programar/codificar de manera fácil y rápida sus programas. La siguiente documentación facilita el desarrollo de su capacidad analítica y creadora, para de esta manera mejorar su destreza en la elaboración de algoritmos que sirven como base para la codificación de los diferentes programas.



Introducción

Cualquier programa esta constituido por un conjunto de órdenes o instrucciones capaces de manipular un conjunto de datos. Cualquier orden o instrucción puede ser dividida en tres grandes bloques claramente diferenciados, correspondientes cada uno ellos a una parte del diseño de un programa:



En el bloque de entrada de datos podemos englobar a todas aquellas instrucciones que toman datos de un dispositivo o periférico externo depositándolos en memoria principal para ser procesados.

El proceso o algoritmo será por tanto todas aquellas instrucciones encargadas de procesar la información o aquellos datos pendientes de elaborar y que previamente habían sido depositados en memoria principal. Finalmente todos los datos obtenidos en el tratamiento de dicha información son depositados nuevamente en memoria principal, quedando de esta manera disponibles.

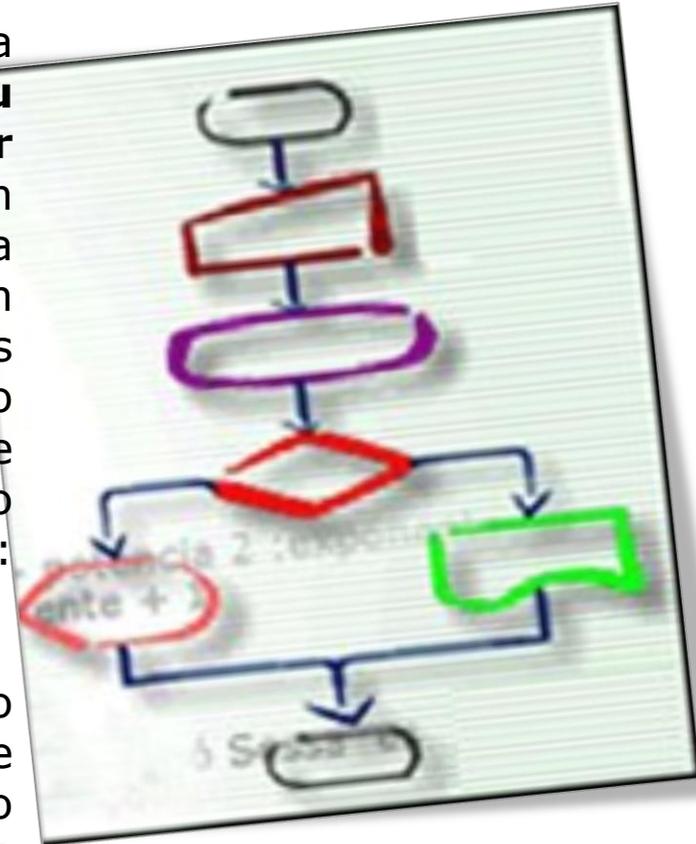
Por último el bloque de salida de datos estará formado por todas aquellas instrucciones que toman los datos depositados en memoria principal una vez procesados los datos de entrada, enviándolos seguidamente a un dispositivo o periférico.

Programación estructurada. Algoritmos

Algoritmo.

Un algoritmo se puede definir como la **descripción abstracta de todas las acciones u operaciones que debe realizar un ordenador de forma clara y detallada**, así como el orden en el que estas deben ejecutarse junto con la descripción de todos aquellos datos que deberán ser manipulados por dichas acciones y que nos conducen a la solución del problema facilitando así su posterior traducción al lenguaje de programación elegido. El diseño de todo algoritmo debe reflejar las tres partes de un programa: entrada, proceso y salida.

Es importante tener en cuenta que todo algoritmo debe ser totalmente independiente del lenguaje de programación, es decir, el algoritmo diseñado deberá permitir su traducción a cualquier lenguaje con independencia del ordenador.



Programación estructurada. Algoritmos

Algoritmo.

Las características que debe cumplir el diseño de todo algoritmo son:

Debe ser **conciso y detallado**, es decir, debe de reflejar adecuadamente el flujo de control.

Todo algoritmo se caracteriza por tener un **inicio y un final**, es decir, debe ser finito.

Al aplicar el algoritmo n° veces sobre los **mismos datos de entrada deberá obtenerse n° veces los mismos resultados**.

Todo algoritmo debe **ser flexible** para permitir y facilitar futuras modificaciones.

Debe ser lo **más claro y sencillo** posible para facilitar su entendimiento y comprensión por parte del personal informático.

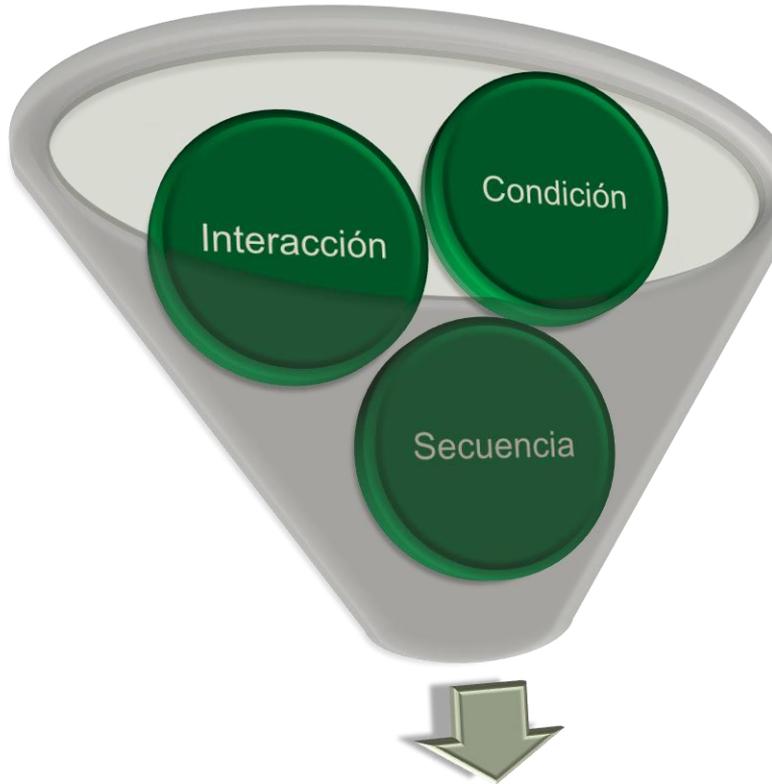
Programación estructurada. Algoritmos

Algoritmo. Pseudocódigo.

Se puede definir como **el lenguaje intermedio entre el lenguaje natural y el lenguaje de programación seleccionado**. Esta notación se encuentra sujeta a unas determinadas reglas que nos permiten y facilitan el diseño. La notación pseudocodificada surge como método para la representación de instrucciones en una metodología estructurada y nació como un lenguaje similar al inglés que utilizaba palabras reservadas de este idioma y que posteriormente se fue adaptando a otros lenguajes de lengua hispana. La notación pseudocodificada se caracteriza por:

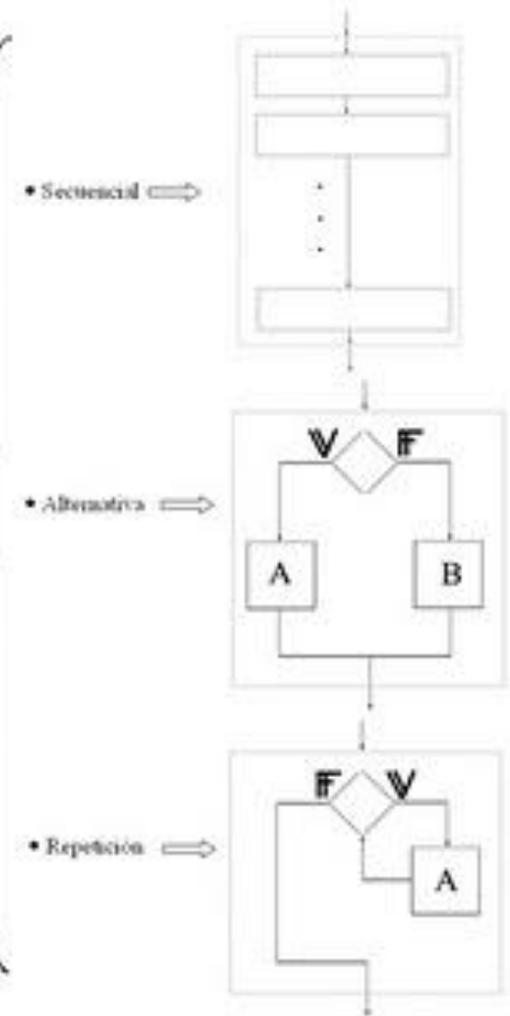
- + No puede ser ejecutada directamente por un ordenador, por lo que tampoco es considerado como un lenguaje de programación propiamente dicho.
- + Ser una forma de representación muy sencilla de aprender y utilizar.
- + Permitir el diseño y desarrollo de algoritmos totalmente independientes del lenguaje de programación que se utilice en la fase de codificación.
- + Facilitar el paso del algoritmo al correspondiente lenguaje de programación.
- + Facilitar la realización de futuras correcciones o actualizaciones.

Estructuras de Control



programas y algoritmos

ESTRUCTURAS DE CONTROL

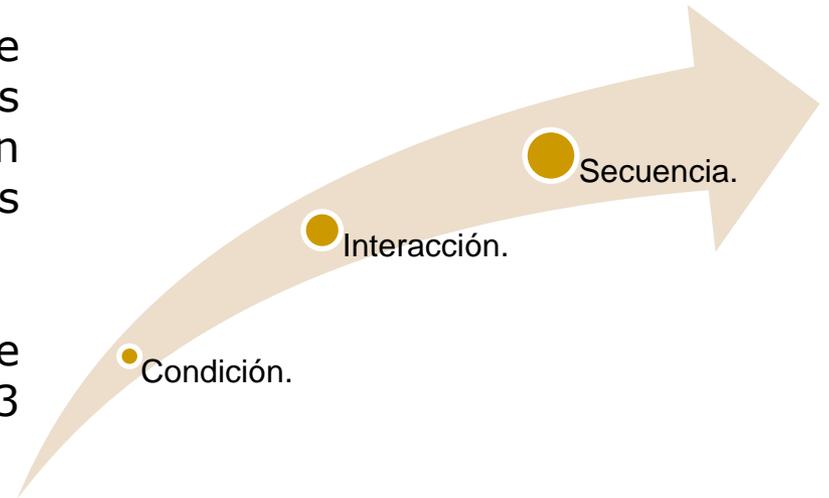


Estructuras de Control

Introducción

Son aquellos elementos con los que podemos llevar a cabo nuestros programas y algoritmos, y permiten modificar el flujo de ejecución de las instrucciones.

Se puede realizar cualquier tipo de problema con la única aplicación de 3 estructuras de control



1.- *Estructuras condición*: Es el punto en el algoritmo en el que se condiciona el estado del proceso y se tienen dos o una alternativa.

2.- *Estructuras de iteración*: Es un mecanismo de lazo. Permite repetir varias veces un grupo de pasos, hasta que se satisfaga una condición.

3.- *Estructura de control de secuencia*: Es un grupo de instrucciones que se ejecuta en orden, de la primera a la última.

Estructuras de Control

IF-THEN-ELSE / SI-ENTONCES-SINO

Si la condición es verdadera, se ejecuta el bloque de sentencias 1, de lo contrario, se ejecuta el bloque de sentencias 2. La condición se corresponde a una expresión relacional o lógica.

IF (Condición) THEN

(Bloque de sentencias 1)

ELSE

(Bloque de sentencias 2)

END IF



Estructuras de Control

Select-Case / switch

Se evalúa la expresión, dando como resultado un valor. Luego, se recorren los "Case" dentro de la estructura buscando la coincidencia. Cuando se encuentra la primera coincidencia, se ejecuta el bloque de sentencias correspondiente y se sale de la estructura Select-Case. Si no se encuentra ninguna coincidencia, se ejecuta el bloque de sentencias de la sección "Case Else/default" si la hubiera.

Select (Expresión)

Case Valor1 (Bloque de sentencias 1)

Case Valor2 (Bloque de sentencias 2)

Case Valor n (Bloque de sentencias n)

Case else/default (Bloque de sentencias "Else")



End

Estructuras de Control

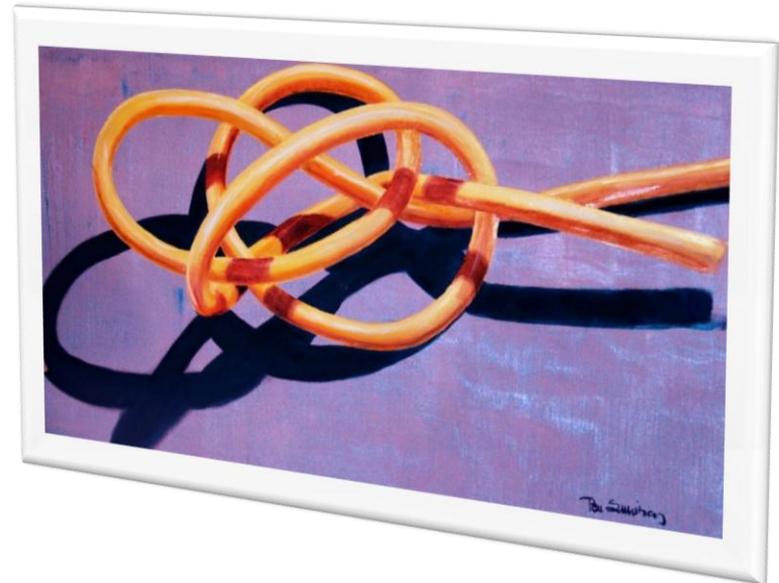
MIENTRAS-HACER / WHILE

Mientras la condición (expresión relacional o lógica) sea verdadera, se ejecutarán las sentencias/estructuras del bloque.

MIENTRAS (Condición) HACER

(Bloque de sentencias)

FIN-MIENTRAS



Estructuras de Control

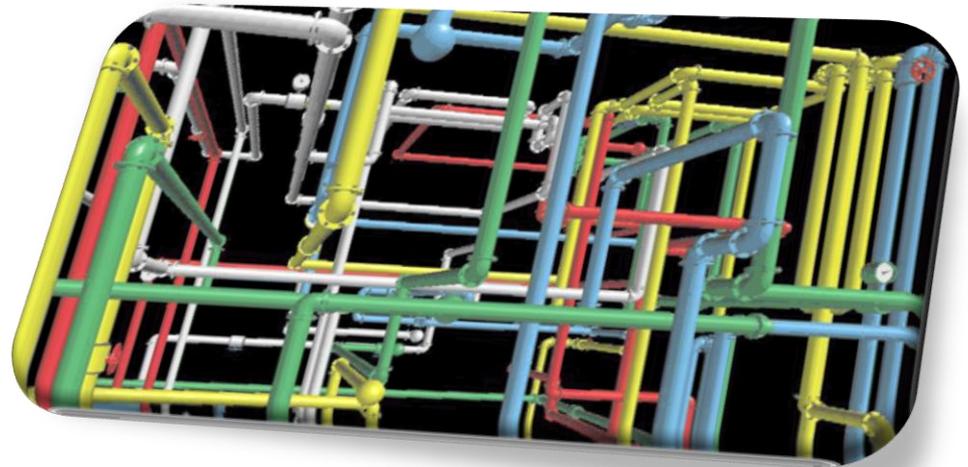
REPETIR-HASTA

Se ejecuta el bloque de sentencias, hasta que la condición sea verdadera.

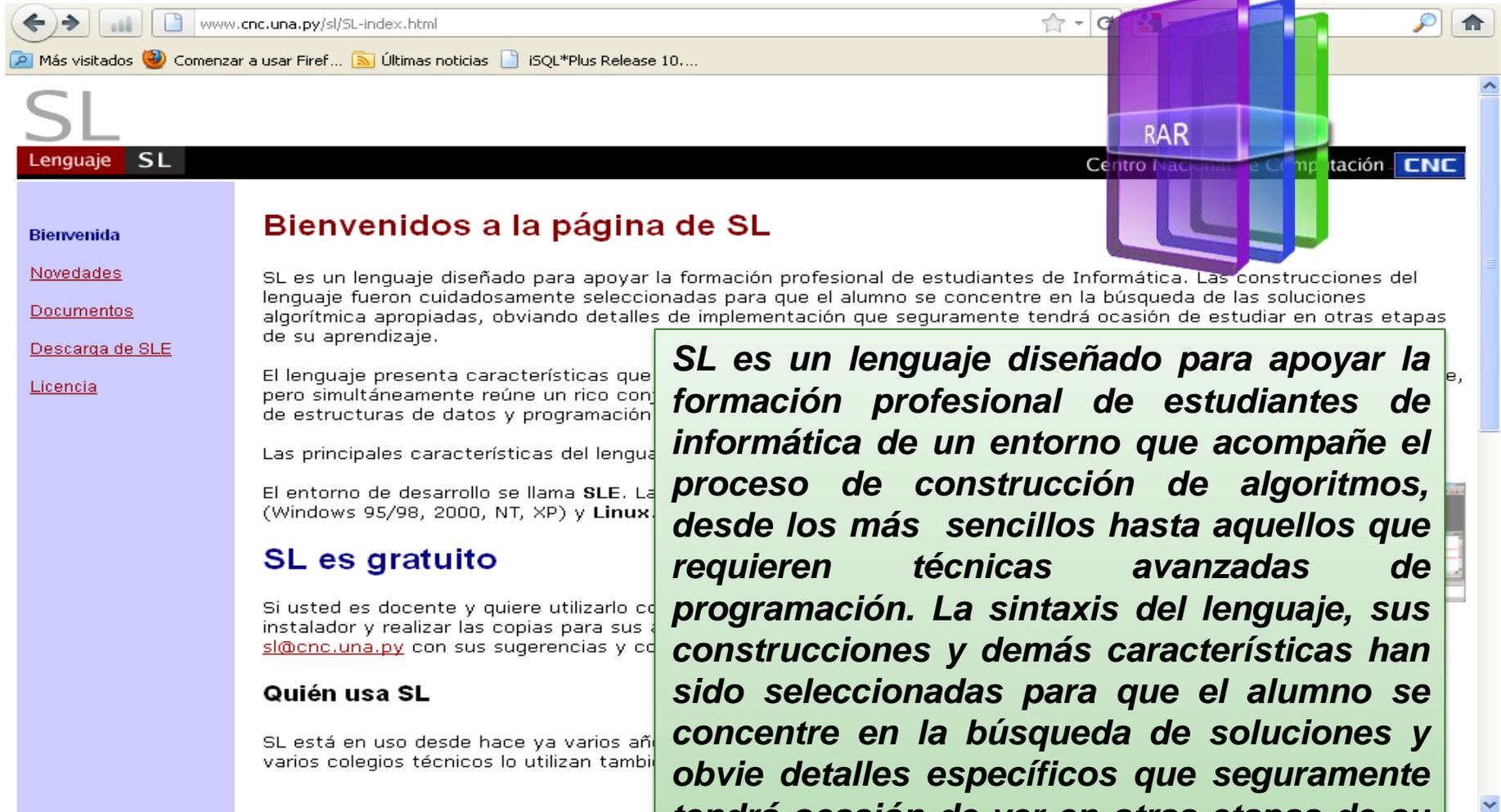
Do

(Bloque de sentencias)

Until (Condición)



Ejemplo de un Entorno Educativo Algorítmico: Lenguaje SL.



The image shows a screenshot of a web browser displaying the website for the SL programming language. The browser's address bar shows the URL www.cnc.una.py/sl/SL-index.html. The website has a navigation menu with links for "Lenguaje SL", "Bienvenida", "Novedades", "Documentos", "Descarga de SLE", and "Licencia". The main content area is titled "Bienvenidos a la página de SL" and contains introductory text about the language. A large, semi-transparent green box with a black border is overlaid on the right side of the page, containing a bolded summary of the language's purpose. A 3D-style icon for a RAR file is positioned in the upper right corner of the browser window.

Bienvenidos a la página de SL

SL es un lenguaje diseñado para apoyar la formación profesional de estudiantes de Informática. Las construcciones del lenguaje fueron cuidadosamente seleccionadas para que el alumno se concentre en la búsqueda de las soluciones algorítmica apropiadas, obviando detalles de implementación que seguramente tendrá ocasión de estudiar en otras etapas de su aprendizaje.

El lenguaje presenta características que pero simultáneamente reúne un rico conjunto de estructuras de datos y programación.

Las principales características del lenguaje

El entorno de desarrollo se llama **SLE**. La (Windows 95/98, 2000, NT, XP) y **Linux**.

SL es gratuito

Si usted es docente y quiere utilizarlo con su clase, puede solicitar el instalador y realizar las copias para sus alumnos escribiendo a sl@cnc.una.py con sus sugerencias y comentarios.

Quién usa SL

SL está en uso desde hace ya varios años y en la actualidad varios colegios técnicos lo utilizan también.

SL es un lenguaje diseñado para apoyar la formación profesional de estudiantes de informática de un entorno que acompañe el proceso de construcción de algoritmos, desde los más sencillos hasta aquellos que requieren técnicas avanzadas de programación. La sintaxis del lenguaje, sus construcciones y demás características han sido seleccionadas para que el alumno se concentre en la búsqueda de soluciones y obvие detalles específicos que seguramente tendrá ocasión de ver en otras etapas de su aprendizaje.

Ejemplo de un Entorno Educativo Algorítmico: Lenguaje SL.

Descarga e instalación de SLE versión 2

Si usa **Microsoft Windows** en cualquiera de sus versiones, puede descargar el siguiente instalador, que puede ejecutarse directamente desde el navegador o ser guardado en el disco local para su posterior ejecución. [setup.exe](#) [5694 KB] del **2004.04.30**

Aviso importante

2004.04.30 Fueron descubiertos algunos problemas en cálculos aritméticos, donde ocurre una pérdida de precisión, y en la lectura de valores numéricos desde un archivo. Si usted tiene instalado SLE Versión 2 previa a la publicada el día de hoy, le sugerimos que actualice su instalación descargando el ejecutable [sle2.exe](#) [1000 KB], reemplazando el que fue instalado originalmente.

Por lo general, `sle2.exe` se ubica en `c:\Archivos de programa\sle2\bin`, o en `c:\Program Files\sle2\bin` si su Windows está en inglés.

Una vez realizada la instalación, en el “Menú inicio” y en el “escritorio” encontrará íconos para activar SLE versión 2. Además, todos los archivos con extensión “.sl” quedarán asociados a SL. Si usa **Linux**, puede descargar [sle-2.0-4.i386.rpm](#) para instalarlo usando `rpm -i` desde la línea de comandos.

Nota: Necesitará acceso a **root** para realizar esta operación, pues los programas y demás archivos se instalan en `/usr/local/sle2`. Esta versión fue probada sobre Red Hat Linux 9.

Ejemplo de un Entorno Educativo Algorítmico: Lenguaje SL.

```
/*
 * Lee un vector de 10 elementos numéricos, y visualizar el mayor.
 * (c) lopezquesada@iessanjuanbosco.es
 */
var
    valores : vector [5] numerico
    indice : numerico
    mayor : numerico
inicio
    imprimir ("\nIngrese ", alen (valores), " números separados por comas:\n")
    leer (valores)
    indice=2
    mayor=valores[1]
    mientras (indice<alen(valores)) {
        si (mayor<valores[indice]){
            mayor=valores[indice]
        }
        indice=indice+1
    }
    imprimir ("\nEl mayor de este vector:\n", valores)
    imprimir ("[" , mayor ,"]")
fin
```

SLE

Archivo Edición Ver Ejecutar Ventana Ayuda

E:\IES\IES_1213\0376IAW\UT2\lenguajeSL\ejemplos\mayor_vector.sl

```
1 /*
2  * Lee un vector de 10 elementos numéricos, y visualizar el mayor.
3  * (c) lopezquesada@iessanjuanbosco.es
4  */
5 var
6  valores : vector [10] numerico
7  indice : numerico
8  mayor : numerico
9 inicio
10 imprimir ("\nIngrese ", alen (valores), " números separados por comas:\n")
11 leer (valores)
12 indice=2
13 mayor=valores[1]
14 mientras (indice<alen(valores)) {
15     si (mayor<valores[indice]){
16         mayor=valores[indice]
17     }
18     indice=indice+1
19 }
20 imprimir ("\nEl mayor de este vect")
21 imprimir ("[" . mayor . "]")
22 fin
```

Mensaje

Compilación exitosa

Aceptar

Visor de Expresiones

E:\IES\IES_1213\0376IAW\UT2\lenguajeSL\ejemplos\mayor_vector.sl



E:\IES\IES_1213\0376IAW\UT2\lenguajeSL\ejemplos\mayor_vector.sl

```

1  /*
2  * Lee un vector de 10 elementos numéricos, y visualizar el mayor.
3  * (c) lopezquesada@iessanjuanbosco.es
4  */
5  var
6  valores : vector [10] numerico
7  indice : numerico
8  mayor : numerico
9  inicio
10 imprimir ("\nIngrese ", alen (valores), " números separados por comas:\n")
11 leer (valores)
12 indice=2
13 mayor=valores[1]
14 mientras (indice<alen(valores))
15     si (mayor<valores[indice]){
16         mayor=valores[indice]
17     }
18     indice=indice+1
19 }
20 imprimir ("\nEl mayor de este vector es: ", mayor)
21 imprimir ("[" , mayor , "]" )
22 fin

```

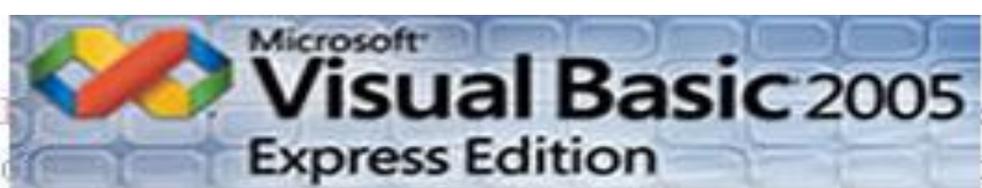
Ventana de ejecución

Datos previos:

Ingrese 10 números separados por comas:
12,36,-5,89,-52,5,96,21,-8,20

El mayor de este vector:
12,36,-5,89,-52,5,96,21,-8,20[96]

Linea: 1 Columna: 0



Perl



```
var x = 2; var y = 10; var z = 10;
document.write(eval("x * y + z + 1"))
```

C++

Se obtiene: 2051

javascript



pythonTM

```
rray("Enero")
es es un objeto, y r
```

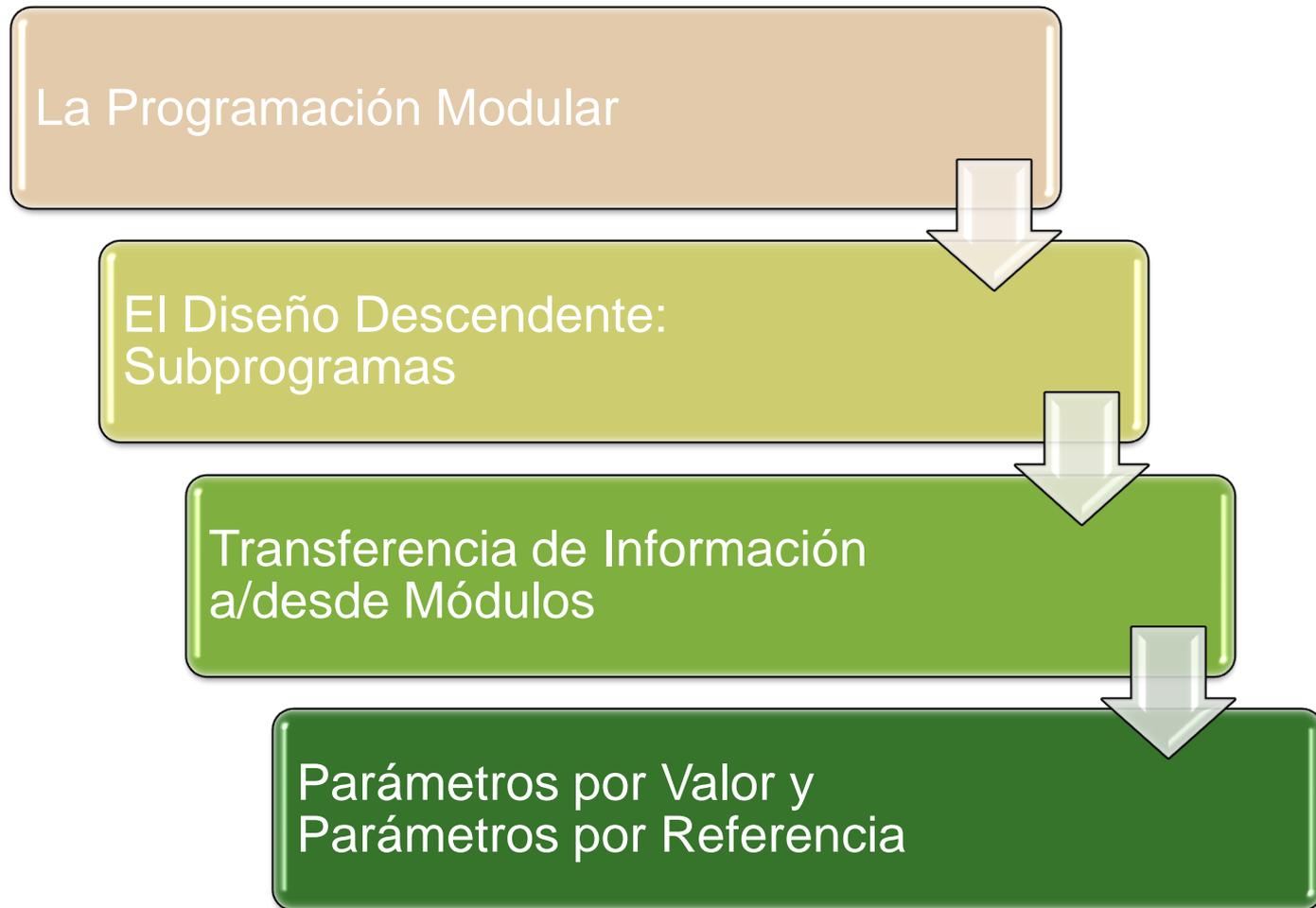


Ruby
A Programmer's Best Friend

```
es son
nero, I
```



Técnicas de Programación. Programación Modular.

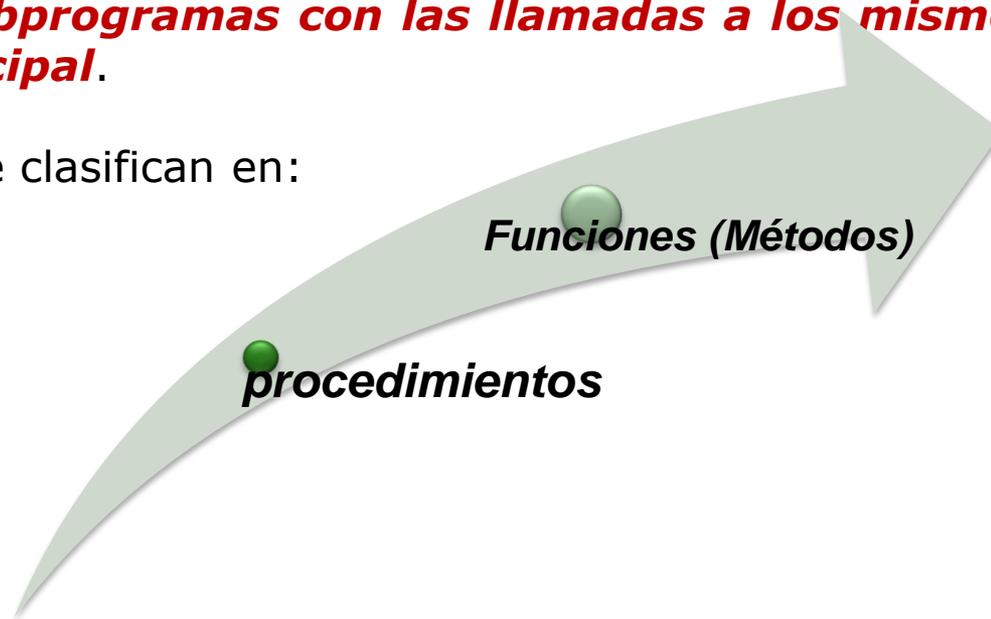


La Programación Modular

Una estrategia muy utilizada para la resolución de problemas complejos es la división del problema en otros problemas más pequeños o subproblemas. ***Estos subproblemas se implementan mediante módulos o subprogramas.***

Los subprogramas son una herramienta importante para el desarrollo de algoritmos y programas de modo que normalmente un proyecto de programación está compuesto generalmente de un ***programa principal y un conjunto de subprogramas con las llamadas a los mismos dentro del programa principal.***

Los subprogramas se clasifican en:



El Diseño Descendente: Subprogramas

Uno de los métodos fundamentales para resolver un problema como se ha mencionado en el apartado anterior es dividirlo en problemas más pequeños. Estos *subproblemas* a su vez pueden ser divididos repetidamente en problemas más pequeños hasta que los problemas más pequeños puedan ser solucionados.

Esta técnica de dividir el problema principal en subproblemas se denomina frecuentemente *divide y vencerás*. El método de diseño se denomina diseño descendente, debido a que se comienza en la parte superior con un problema general y se diseñan soluciones específicas a sus subproblemas.

El problema principal se resuelve con el programa principal (también llamado conductor/main() del programa), y los subproblemas (módulos) mediante subprogramas: procedimientos y funciones.

Un subprograma realiza una tarea concreta que se describe con una serie de instrucciones.

Transferencia de Información a/desde Módulos

Los Parámetros

Los módulos o subprogramas sirven para ejecutar tareas concretas, pero no utilizan ningún tipo de dato del resto del programa. Sin embargo, una de las características importantes de los subprogramas/módulos es la posibilidad de comunicación.

Esta comunicación se realiza a través de una lista de parámetros.

Un parámetro es un método para pasar información (valores a variables) del programa principal a un módulo o viceversa.

Así pues, los módulos se clasifican en:

Módulos sin parámetros (no existe comunicación entre el programa principal y los módulos o entre módulos).

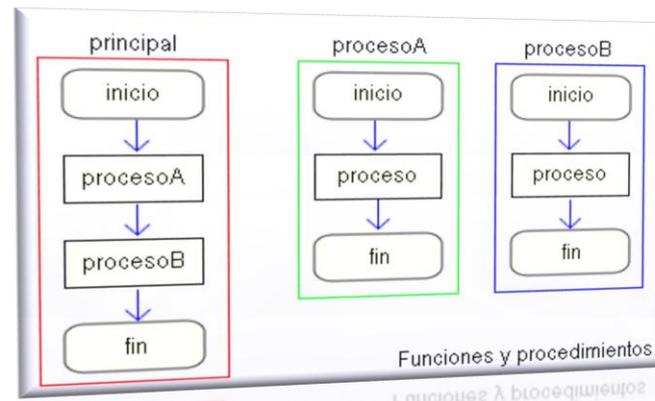
Módulos con parámetros (existe comunicación entre el programa principal y los módulos, y entre ellos).

Transferencia de Información a/desde Módulos

Los Parámetros

Un parámetro es prácticamente, una variable cuyo valor debe ser o bien proporcionado por el programa principal al módulo o ser devuelto desde el módulo hasta el programa principal. Por consiguiente hay dos tipos de parámetros: *entrada* y *salida*.

Los parámetros de entrada son aquellos cuyos valores deben ser proporcionados por el programa principal, y los de salida son aquellos cuyos valores se calcularán en el subprograma o módulo y se deben devolver al programa principal para su proceso posterior.



Transferencia de Información a/desde Módulos

Los Parámetros

Las sentencias llamadas a subprogramas constan de dos partes: *un nombre de subprograma y una lista de parámetros llamados actuales.*

nombreSubprograma (pa1,pa2,...)

En la declaración de un subprograma, cuando se incluyen parámetros, estos se denominan parámetros formales. Ellos sirven para contener los valores de parámetros actuales cuando se llama al subprograma.

Procedimiento o funcion nombresubprograma(pf1,pf2,...)

Los parámetros actuales en la llamada al subprograma deben coincidir en número, orden y tipo con los parámetros formales de la declaración del subprograma. Es decir, debe existir una correspondencia entre parámetros actuales y actuales.

Parámetros por Valor y Parámetros por Referencia

Existen dos tipos de parámetros, como se indicó anteriormente, que nos ayudan a transferir/recibir información de otros subprogramas, o del programa principal, a saber: *parámetros de entrada (por valor)* y *parámetros de salida o de entrada/salida (referencia)*.

Parámetros Valor

Son parámetros unidireccionales que se utilizan para proporcionar información a un subprograma, pero no pueden devolver valores, al programa llamador.

Se les llama parámetros de entrada, ya que en la llamada al subprograma el valor del parámetro actual se pasa a la variable que representa el parámetro formal. Este valor puede ser modificado dentro del subprograma pero el valor modificado no es devuelto al programa o subprograma llamador, realmente lo que ocurre es que se pasa al parámetro formal una copia del parámetro actual, imposibilitando que en el módulo llamador se refleje cualquier cambio efectuado.

Parámetros por Valor y Parámetros por Referencia

Parámetros Referencia

Se utilizan tanto para recibir como para transmitir valores entre el subprograma y el programa llamador.

Este puede actuar como parámetro de salida o de entrada/salida, para eso el parámetro formal recibe una referencia o puntero del parámetro actual, por lo que cualquier cambio de dicho parámetro quedará automáticamente reflejado en el parámetro actual y por lo tanto en el módulo en que se ha realizado la llamada.

Variables Locales y Variables Globales

Las variables utilizadas en un programa con subprogramas pueden ser de dos tipos: *locales* y *globales*.

Variables Locales

Una variable local es una variable que está declarada dentro de un subprograma y se dice que es local al subprograma. Una variable local solo está disponible durante la ejecución del subprograma, al terminar se pierde el valor que se encontraba guardado en dicha la variable local.

Variables Globales

Al contrario que las variables locales cuyos valores se pueden utilizar solo dentro del subprograma en que fueron declaradas, las variables globales se pueden utilizar en todo el programa principal y en todos los subprogramas, donde se haga referencia al identificador de esta variable.

Procedimientos y Funciones

Los procedimientos y funciones son la base principal en la programación modular, estudiaremos aquí su funcionamiento y su sintaxis, aunque debo recordarles que es solamente pseudocódigo.

Procedimientos

Un procedimiento es un subprograma que realiza una tarea específica. Puede recibir cero o n parámetros del programa que llama y devolver cero o más valores a dicho programa.

Un procedimiento está compuesto de un grupo de sentencias a las que se asigna un nombre (identificador) y constituye una unidad de programa. La tarea determinada al procedimiento se ejecuta siempre que se encuentra el nombre del procedimiento.

Procedimientos y Funciones

La declaración indica las instrucciones a ejecutar. Su sintaxis es:

```
procedimiento nombreproc (lista de parámetros formales  
<<por valor y/o referencia>>)  
  declaraciones locales  
inicio  
  cuerpo del procedimiento (instrucciones)  
fin
```

Un procedimiento es llamado en un programa o dentro de otro procedimiento directamente por su nombre en cualquiera de las dos formas:

```
nombreproc();
```

```
nombreproc (lista parámetros actuales);
```

Procedimientos y Funciones

Funciones

Una función es un subprograma que recibe como argumentos un conjunto de n parámetros formales los cuales pueden ser por valor y/o por referencia, pudiendo devolver un resultado a través de una sentencia cuyo tipo estará indicado en la cabecera de la función. Esta característica le diferencia esencialmente de un procedimiento.

Su formato es el siguiente:

```
funcion nombrefuncion (lista de parámetros formales <<por  
valor y/o referencia>>) : tipo a devolver  
    declaraciones locales  
inicio  
    cuerpo de la función  
    nombrefuncion <- valor a devolver  
fin
```

Procedimientos y Funciones

Una función es llamada por medio de su nombre, en una sentencia de asignación, en una sentencia de salida, o podrá participar como operando de una expresión considerando siempre el tipo del valor que devuelve.

Se puede llamar a una función en cualquiera de las siguientes formas:

nombrefuncion o nombrefuncion(lista parámetros actuales);

identificador = Expresion(nombrefuncion(lista parámetros actuales));

```
/*  
 * Este es el clásico ejemplo del cálculo del factorial  
 * de un entero positivo n mediante un función.  
 *  
 * (c) lopezquesada@iessanjuanbosco.es  
 */
```

```
var
```

```
  n : numerico
```

```
inicio
```

```
  imprimir ("\nCALCULO DE FACTORIAL",  
            "\n-----",  
            "\nIngrese un numero (0-n):")
```

```
  leer (n)
```

```
  si ( n >= 0 && n == int (n) ) {
```

```
    imprimir ("\n\n\n", n, "!=" , fact (n))
```

```
  sino
```

```
    imprimir ("\nNo definido para ", n)
```

```
  }
```

```
fin
```

```
sub fact (n : numerico) retorna numerico  
var  
  producto = 1  
  indice: numerico  
inicio  
  indice=1  
  mientras (indice<n) {  
    producto=producto*indice;  
    indice=indice+1  
  }  
  retorna producto  
fin
```



```

1  /*
2  * Este es el clásico ejemplo del cálculo del factorial
3  * de un entero positivo n.
4  *
5  * (c) lopezquesada@iessanjuanbosco.es
6  */
7  var
8  n : numerico
9  inicio
10 imprimir ("\nCALCULO DE FACTORIAL",
11           "\n-----",
12           "\nIngrese un numero (0-n):")
13 leer (n)
14 si ( n >= 0 && n == int (n) ) {
15     imprimir ("\n\n", n, "!=" , fact (n))
16 } sino
17     imprimir ("\nNo definido para ", n)
18 }
19 fin
20
21
22 sub fact (n : numerico) retorna numerico
23 var
24     producto = 1
25     indice: numerico
26 inicio
27     indice=1
28     mientras (indice<n) {
29         producto=producto*indice;
30         indice=indice+1
31     }
32     retorna producto
33 fin

```

Mensaje

Compilación exitosa

Aceptar



E:\IES\IES_1213\0376IAW\UT2\lenguajeSL\ejemplos\factorial.sl

```

1  /*
2  * Este es el clásico ejemplo del cálculo del factorial
3  * de un entero positivo n.
4  *
5  * (c) lopezquesada@iessanjuanbosco.es
6  */
7  var
8      n : numerico
9  inicio
10     imprimir ("nCÁLCULO DE FACTORIAL",
11              "\n-----",
12              "\nIngrese un numero (0-n):")
13     leer (n)
14     si ( n >= 0 && n == int (n) ) {
15         imprimir ("\n\n", n, "!=" , fact (n))
16     sino
17         imprimir ("\nNo definido para ", n)
18     }
19 fin
20
21 sub fact (n : numerico) retorna numerico
22 var
23     producto = 1
24     indice: numerico
25 inicio
26     indice=1
27     mientras (indice<n) {
28         producto=producto*indice;
29         indice=indice+1
30     }
31     retorna producto
32 fin
33

```

Ventana de ejecución

Datos previos:

CÁLCULO DE FACTORIAL

Ingrese un numero (0-n):5

5!=24

CÁLCULO DE FACTORIAL

Ingrese un numero (0-n):-8

No definido para -8

Linea: 1 Columna: 0

Ejemplo: Leer una secuencia de 30 números y mostrar la suma de los primos y el producto de los factoriales.

PROGRAMA Ejercicio

CONSTANTES // Constantes globales

C_I = 1;

C_S = 30;

FIN_CONSTANTES

VARIABLE

ENTERO numero;

ENTERO indice;

ENTERO suma=0;

ENTERO multiplicar=1;

ENTERO resul;

FIN_VARIABLE // Variable sglobales

FUNCION ENTERO factorial(VAL ENTERO fact) // Consideramos el caso general n!

CONSTANTES // Constantes locales a esta función

FIN_CONSTANTES

VARIABLE

ENTERO indice;

ENTERO suma=0;

ENTERO mul=1;

FIN_VARIABLE // Variables locales a esta función

INICIO

indice=1;

MIENTRAS (indice<=fact) HACER

 multiplicar = multiplicar * indice ;

 indice=indice+1;

FIN_MIENTRAS

DEVOLVER(mul);

FIN

FIN_FUNCION

.....

```

PROCEDIMIENTO primo(VAL ENTERO var, REF ENTERO es_primo) // Consideramos el caso general
    CONSTANTES // Constantes locales
    FIN_CONSTANTES
    VARIABLE
        ENTERO indice;
    FIN_VARIABLE // Variables locales
    INICIO
        indice=2;
        MIENTRAS ((indice<var) AND (var MOD indice <>0)) HACER
            indice=indice+1;
        FIN_MIENTRAS
        SI (indice=var) ENTONCES
            es_primo=1;
        SINO
            es_primo=0;
        FIN_SIN
    FIN
FIN_PROCEDIMIENTO
INICIO // módulo principal mail()
    multiplicar = 1;
    indice=C_I;
    MIENTRAS (indice<=C_S)
        ESCRIBIR ("introduce un numero");
        LEER (numero );
        primo(numero,resul);
        SI (resul=1) ENTONCES
            suma =suma + numero;
        FIN_SI
        multiplicar = multiplicar * factorial(numero) ;
        indice=indice+1;
    FIN_MIENTRAS
    ESCRIBIR ("La suma es "+ suma );
    ESCRIBIR ("El producto es " + multiplica);
    FIN
FIN_PROGRAMA

```

IMPORTAR mate_lib.h

```
PROGRAMA Ejercicio
    CONSTANTES // Constantes globales
        C_I = 1;
        C_S= 30;
    FIN_CONSTANTES
    VARIABLE
        ENTERO numero;
        ENTERO indice;
        ENTERO suma=0;
        ENTERO multiplicar=1;
        ENTERO resul;
    FIN_VARIABLE // Variables globales

    INICIO // módulo principal mail()
        multiplicar = 1;
        indice=C_I;
        MIENTRAS (indice<=C_S)
            ESCRIBIR ("introduce un numero");
            LEER (numero );
            primo(numero,resul);
            SI (resul=1) ENTONCES
                suma =suma + numero;
            FIN_SI
            multiplicar = multiplicar * factorial(numero) ;
            indice=indice+1;
        FIN_MIENTRAS
        ESCRIBIR ("La suma es "+ suma );
        ESCRIBIR ("El producto es " + multiplica);

    FIN
FIN_PROGRAMA
```

