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

Módulo Profesional: IAW

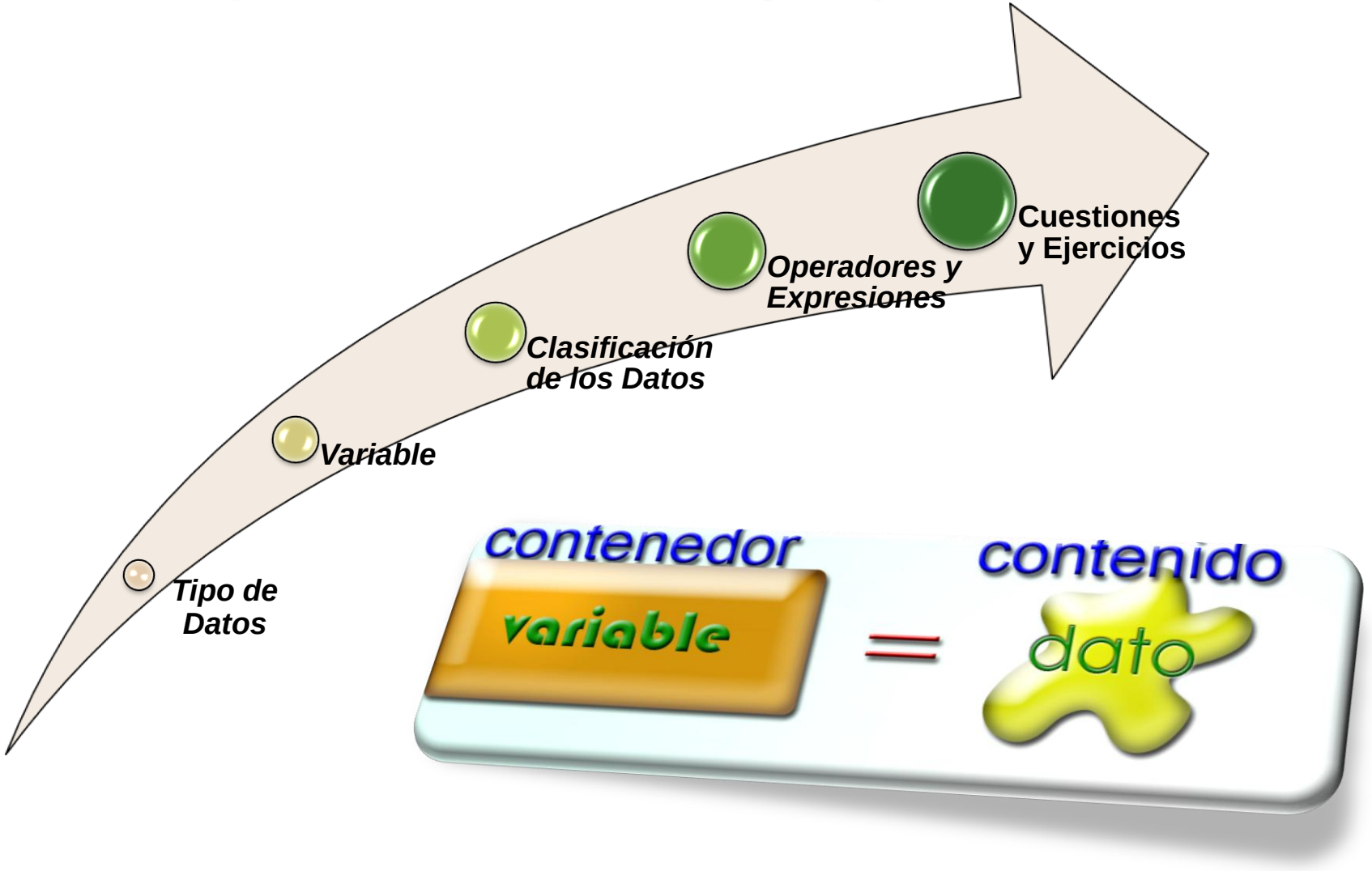
UT 2. Introducción a la Programación estructurada y modular en los lenguajes de <<script>> clientes y servidor.

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


I.E.S.
SAN JUAN BOSCO

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
   }
}
```

Variable

En este punto surge el concepto de *constante y literal*:

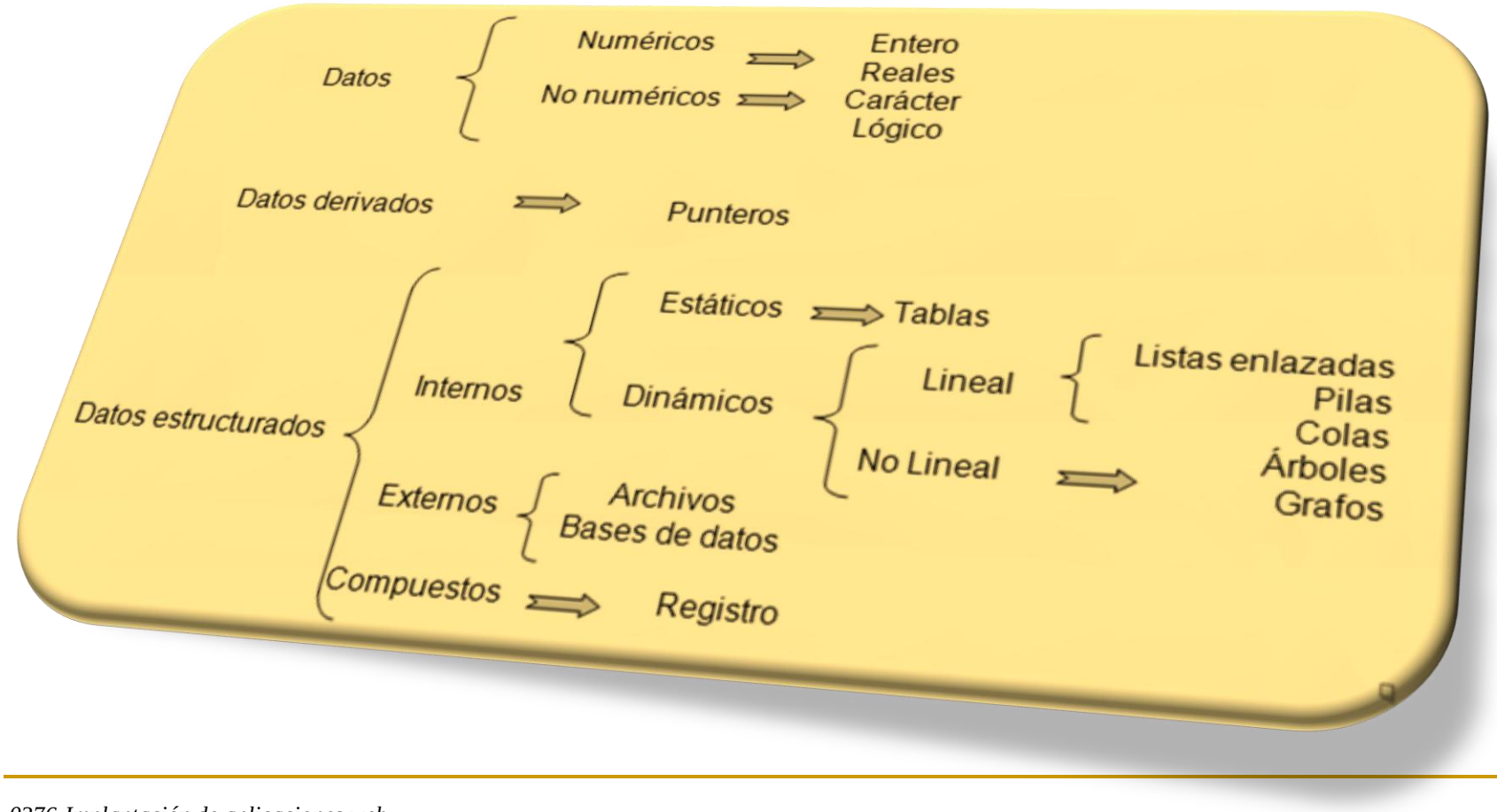
Una constante es un identificador cuyo valor asociado es fijo; realmente lo que ocurre es que en el proceso de pre compilación el identificador se sustituye directamente por el literal asociado. En el caso de C tenemos la palabra reservado DEFINE para la creación de macros/constantes.

Un literal es un dato que aparece explícitamente en el fichero fuente, en el código del programa, por ejemplo si sumo 5+6, es evidente que 5 es un valor entero y que tiene significado por si mismo, mientras que en contador+5, nos encontramos con la variable contador cuyo valor en ese momento no lo conocemos, aunque es uno de los posibles en función del tipo que se hubiera utilizado en la definición.

Por último indicar que tanto las constantes, literales, variables, como la evaluación de las expresiones.. etc., en definitiva todo elemento que es susceptible de ser manipulado/procesado tienen un ***TIPO DE DATO***.

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

Relaciónales

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

Lógico

i, 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.

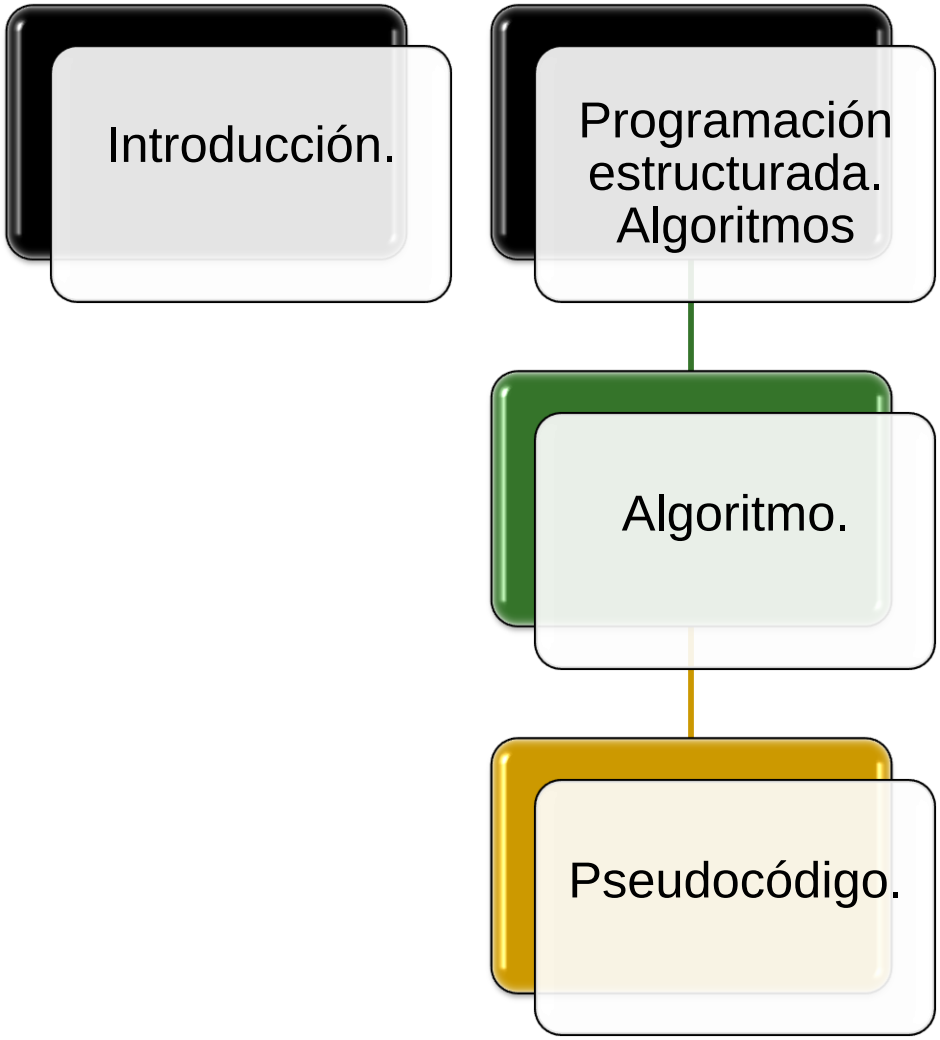
Cuestiones y Ejercicios

- Define la diferencia entre variable y constante.
- Realiza una posible clasificación de los tipos de datos.
- Define con tus palabras el concepto de Tipo de Datos.
- Tipos de datos en java, javascript, pascal y otros lenguajes.
- Define y clasifica las expresiones.
- Describe con conjunto de posibles reglas para la correcta construcción de expresiones.
- Clasifica los tipos de operadores y determina la precedencia que existe entre ellos.
- Indica que variables son correctas o no y motiva la respuesta: Cantidad, contador, %posi, Cant_total, cant de , cant/3, _varia

Indica el tipo de dato de cada literal:

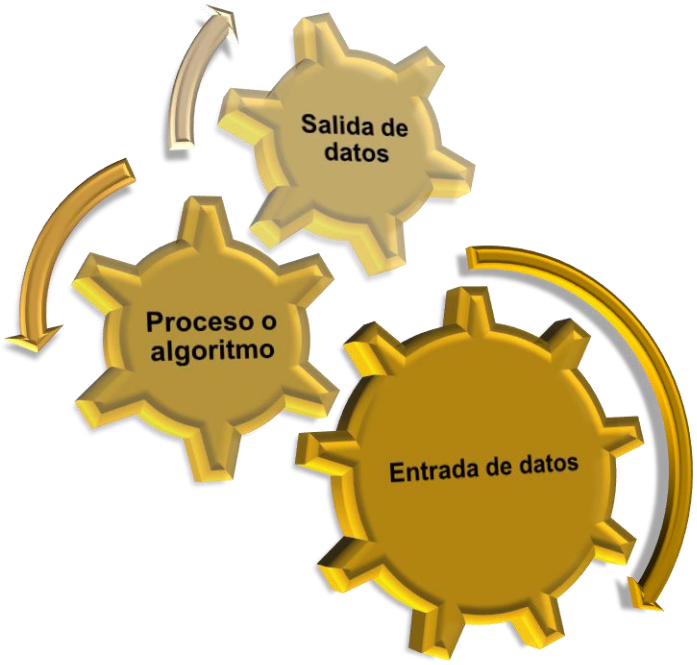
12, 5.7, 3e-12, ‘s’, “dd”, ‘1’, “345”, “fg”

Programación estructurada. Algoritmos



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.

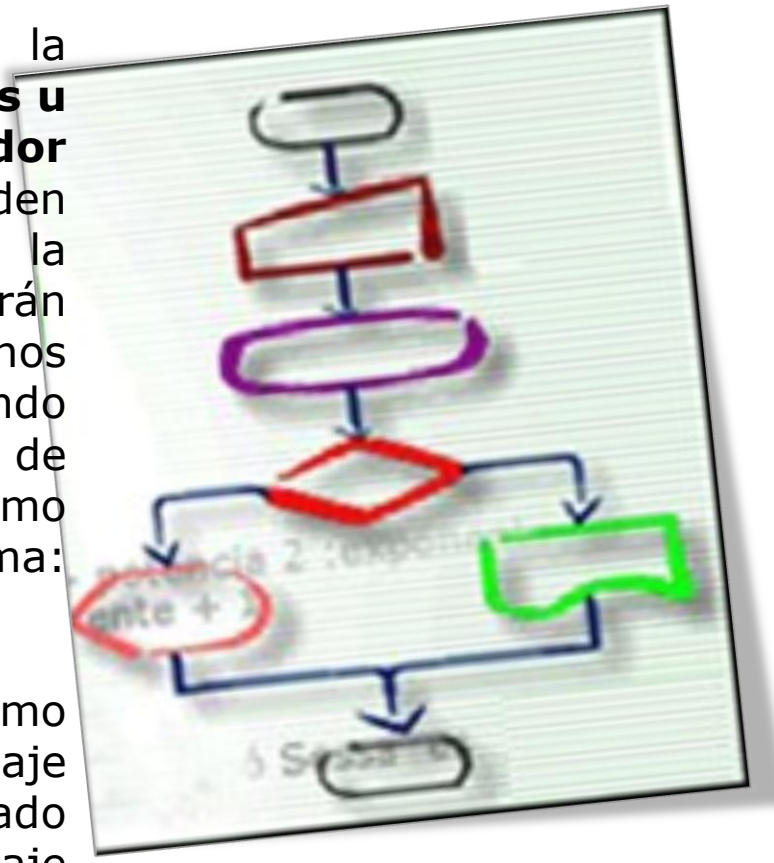
El proceso o algoritmo será por tanto todas aquellas instrucciones encargadas de procesar la información o aquellos datos pendientes de elaborar y que previamente habían sido depositados en memoria principal. Finalmente todos los datos obtenidos en el tratamiento de dicha información son depositados 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 nº veces sobre los **mismos datos de entrada deberá obtenerse nº 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.

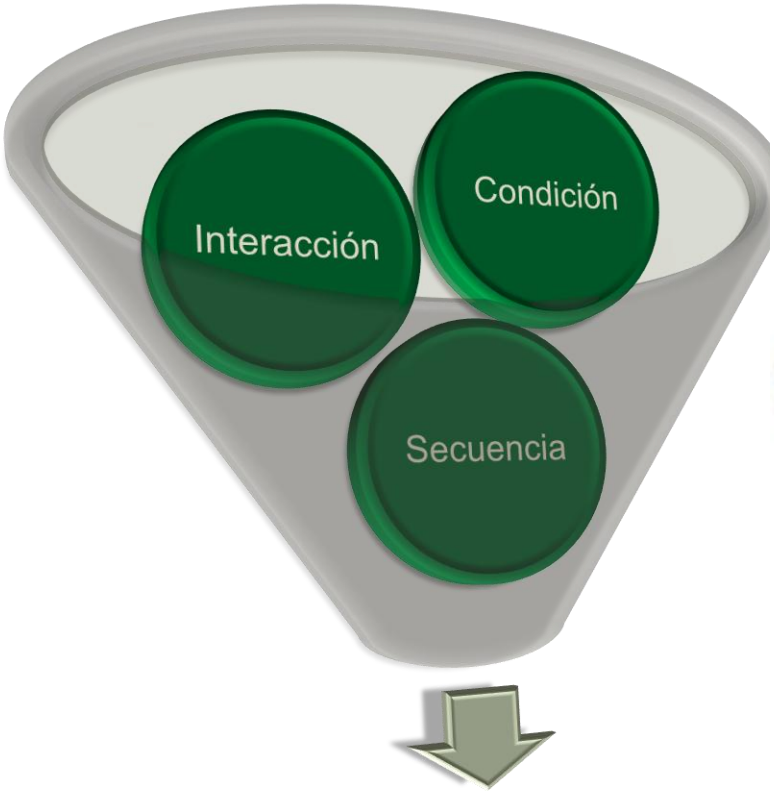
Programación estructurada. Algoritmos

Algoritmo. Pseudocódigo.

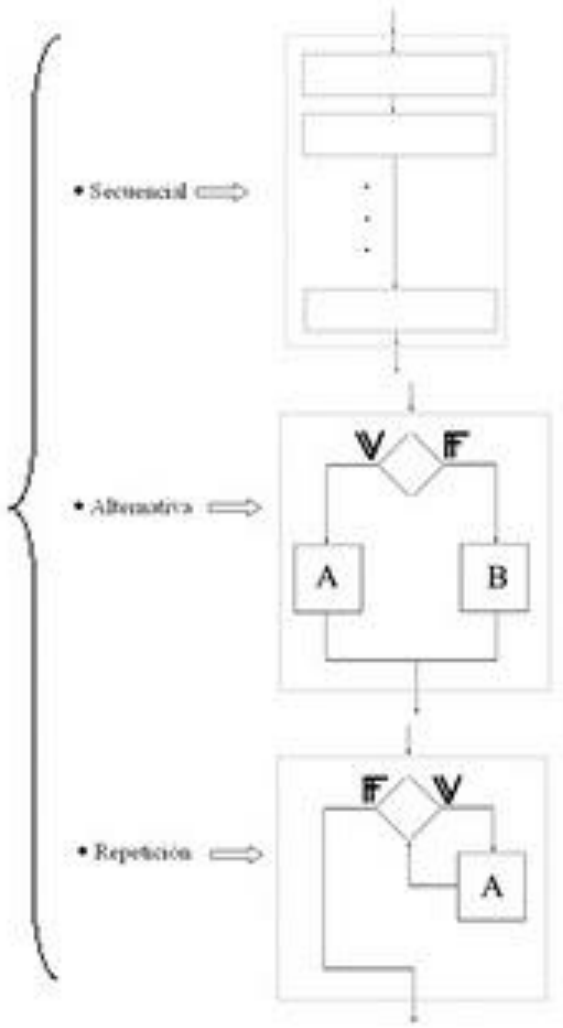
Se puede definir como **el lenguaje intermedio entre el lenguaje natural y el lenguaje de programación seleccionado**. Esta notación se encuentra sujeta a unas determinadas reglas que nos permiten y facilitan el diseño. La notación pseudocodificada surge como método para la representación de instrucciones en una metodología estructurada y nació como un lenguaje similar al ingles 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

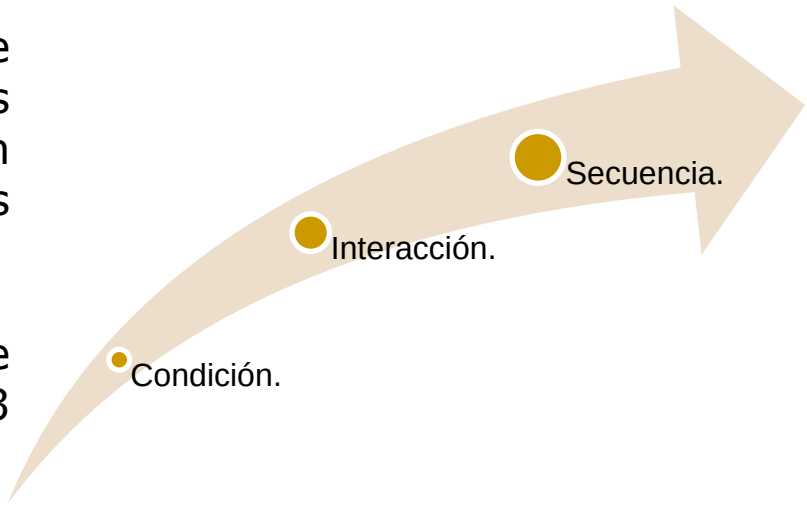


Estructuras de Control

Introducción

Son aquellos elementos con las 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 se sentencias 2)

END IF

Estructuras de Control

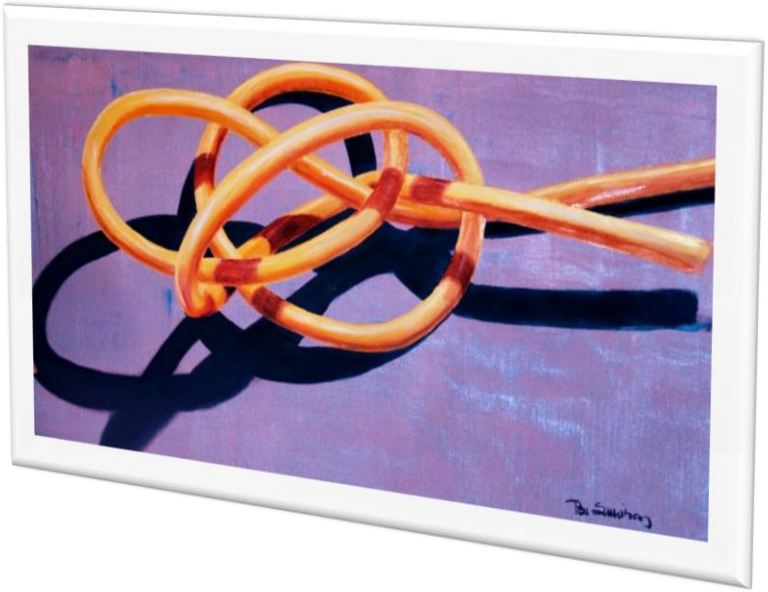
MIENTRAS-HACER / WHILE

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

MIENTRAS (Condición) HACER

(Bloque de sentencias)

FIN-MIENTRAS



Ejemplo de un Entorno Educativo Algorítmico: Lenguaje SL.

www.cnc.una.py/sl/SL-index.html

Más visitados | Comenzar a usar Firef... | Últimas noticias | iSQL*Plus Release 10....

SL

Lenguaje SL — Centro Nacional de Computación CNC

Bienvenida

Novedades

Documentos

Descarga de SLE

Licencia

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 lo hacen apropiado para expresar algoritmos de las etapas iniciales del aprendizaje, pero simultáneamente reúne un rico conjunto de construcciones que posibilitan el tratamiento de tópicos más avanzados de estructuras de datos y programación modular.

Las principales características del lenguaje se describen aquí. También puede encontrar algunos ejemplos sencillos.

El entorno de desarrollo se llama **SLE**. La versión más reciente de SLE está disponible para **Windows** (Windows 95/98, 2000, NT, XP) y **Linux**.

SL es gratuito

Si usted es docente y quiere utilizarlo como apoyo a su actividad educativa, simplemente puede descargar el instalador y realizar las copias para sus alumnos, ya que SL es **gratuito**. Si desea, puede enviar un email a soporte-sl@cnc.una.py con sus sugerencias y comentarios.

Quién usa SL

SL está en uso desde hace ya varios años en varias universidades del Paraguay (entre ellas la UNA y la UCA). Además, varios colegios técnicos lo utilizan también como su lenguaje de introducción a la programación.

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 en el Entorno Educativo Algorítmico: Lenguaje SL.

```
/*
 * Lee un vector de 10 elementos numéricos, y visualizar el mayor.
 * (c) lopezquesada@iessanjuanbosco.es
 */
var
        valores : vector [10] 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
```

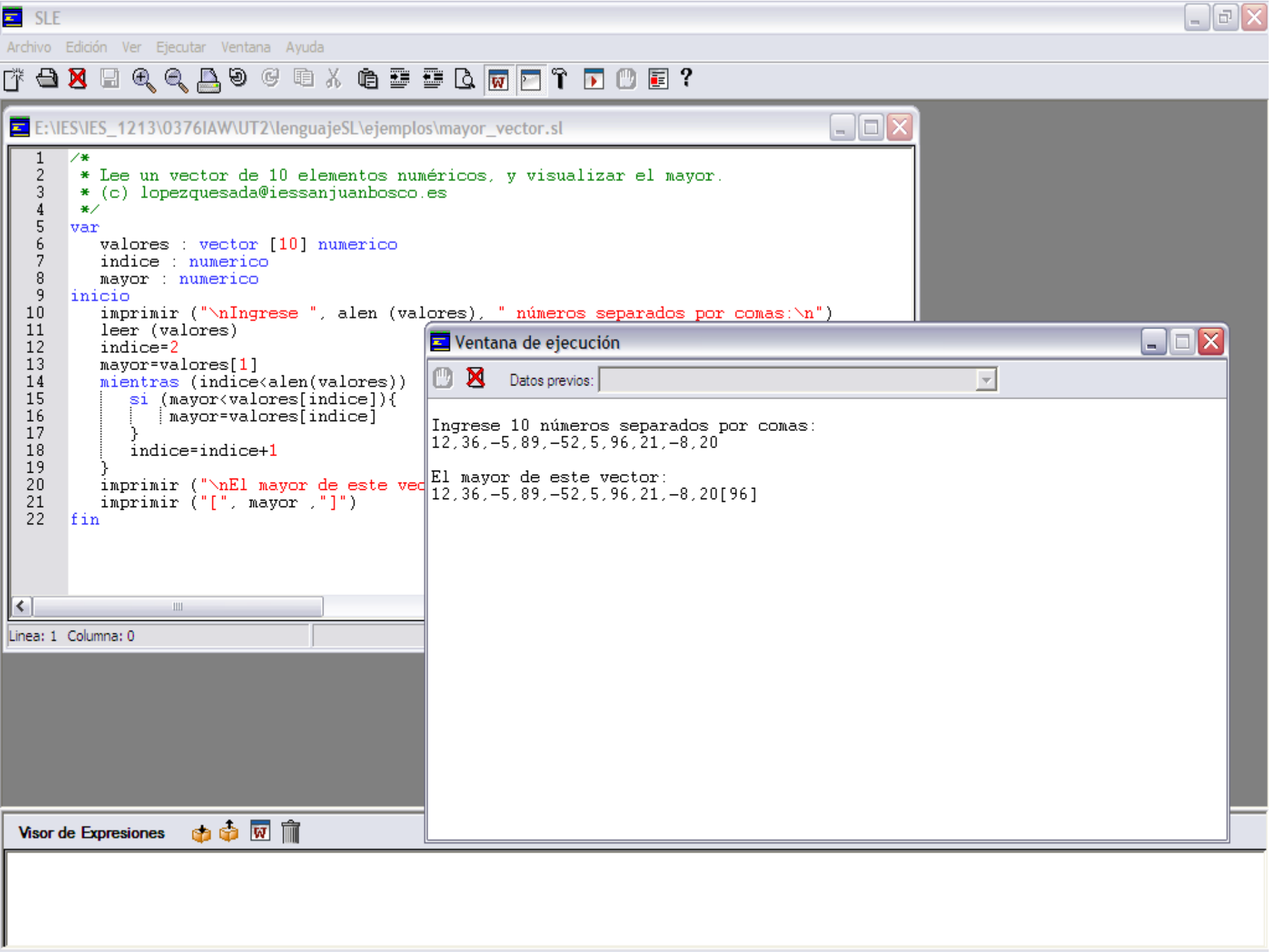
Mensaje

Compilación exitosa

Aceptar

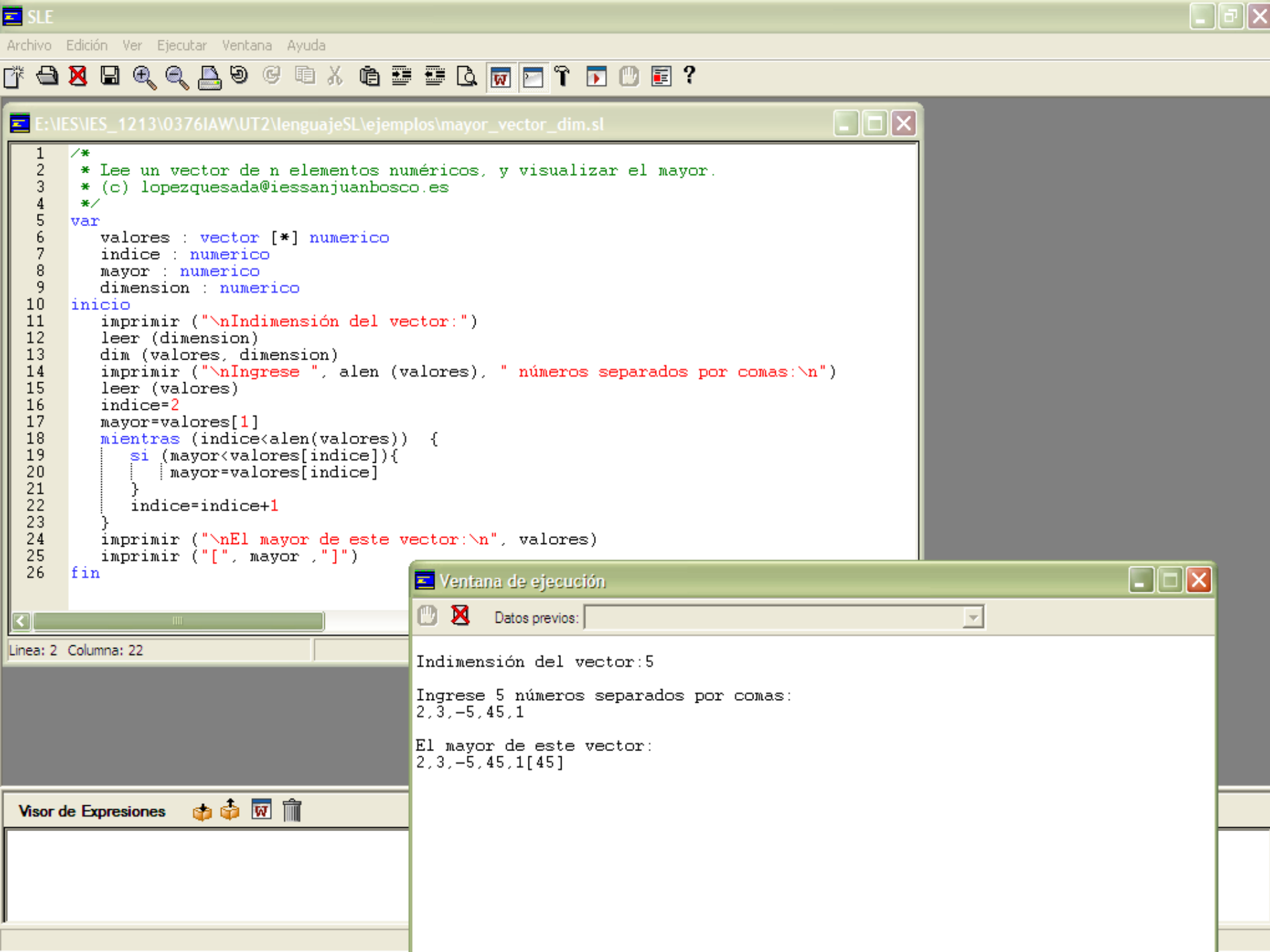
Visor de Expresiones

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

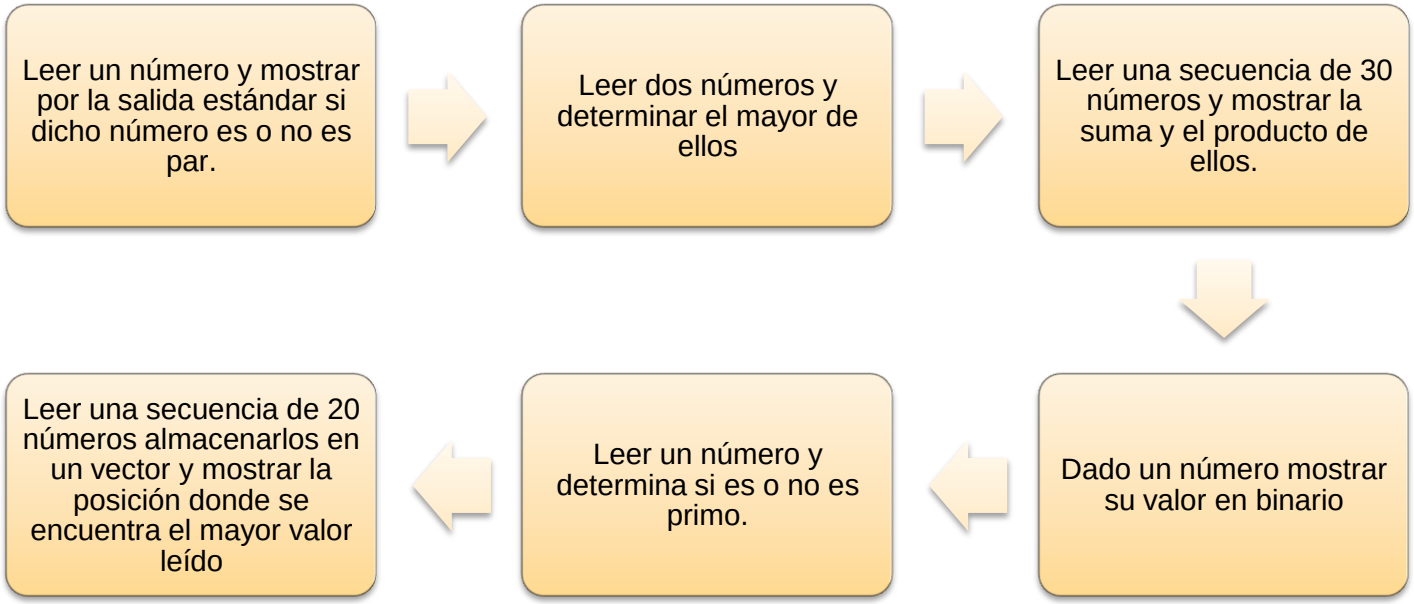

SLE
Archivo Edición Ver Ejecutar Ventana Ayuda
E:\IES\IES_1213\0376IAW\UT2\lenguajeSL\ejemplos\mayor_vector.sl
/*
* Lee un vector de 10 elementos numéricos, y visualizar el mayor.
* (c) lopezquesada@iessanjuanbosco.es
*/
var
valores : vector [10] numerico
indice : numerico
mayor : numerico
inicio
imprimir ("\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 vec
imprimir ("[", mayor ,"]")
fin
Linea: 1 Columna: 0
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]
Visor de Expresiones

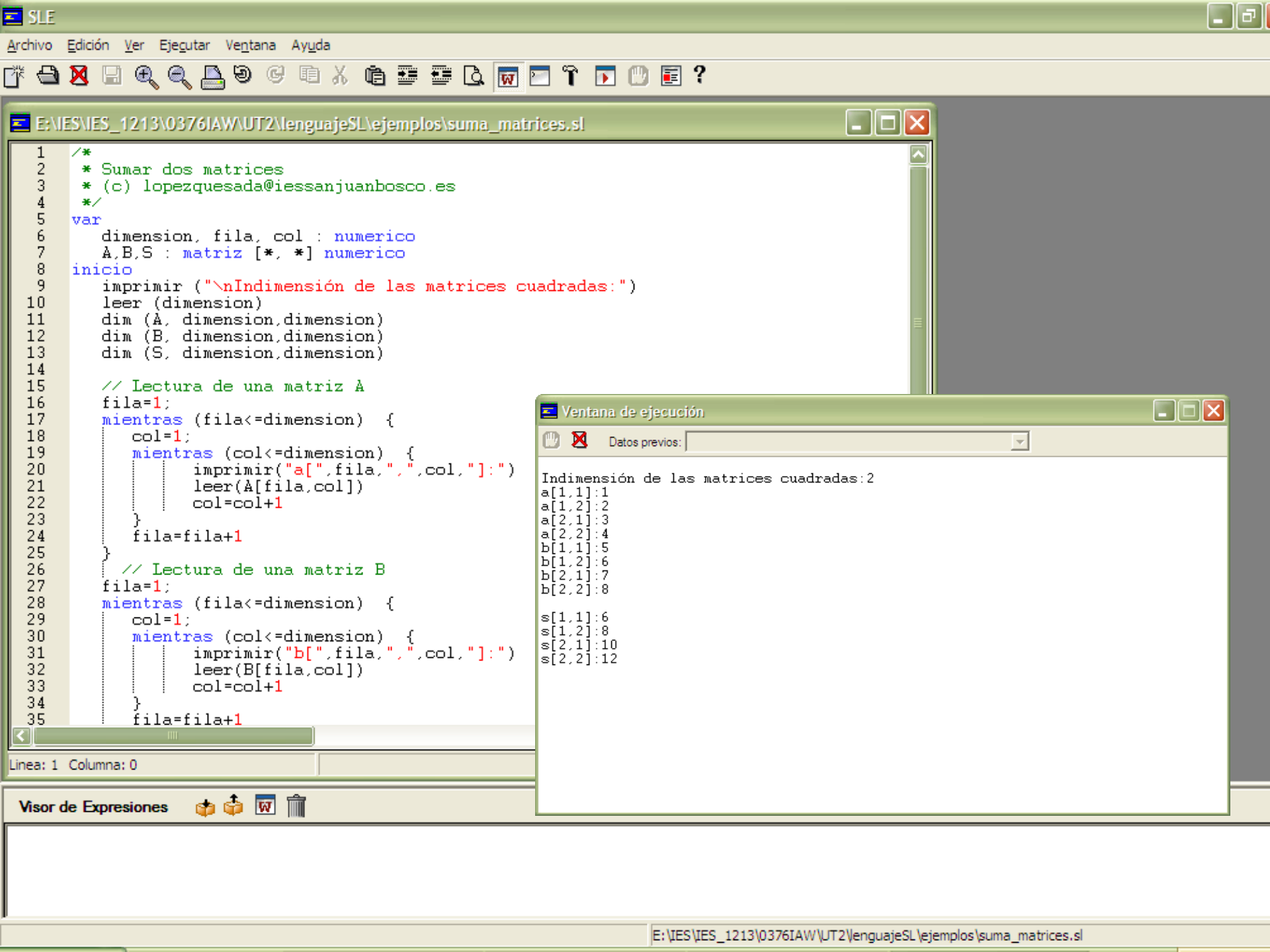
Ejemplo en el Entorno Educativo Algorítmico: Lenguaje SL.

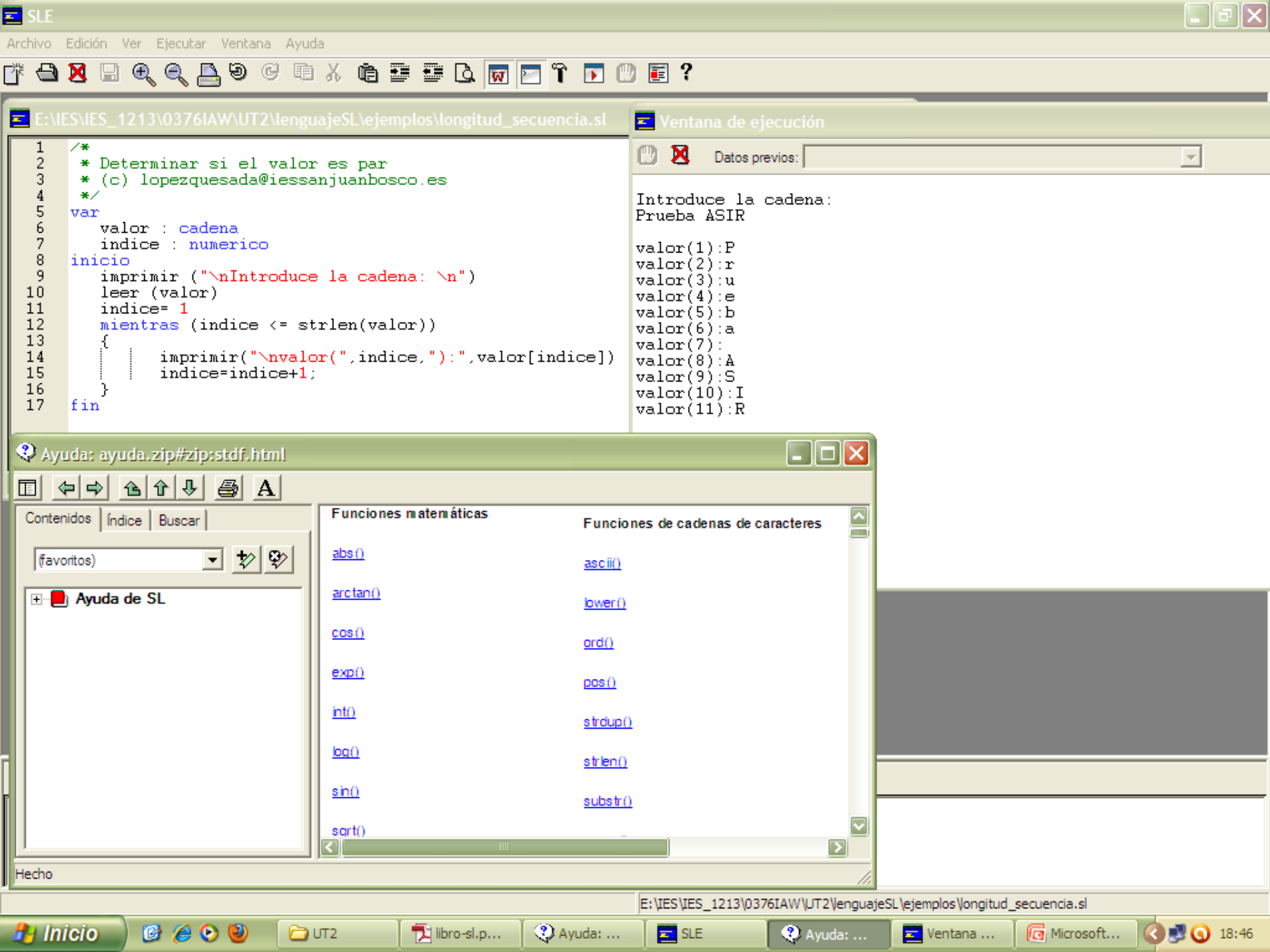
```
var
        valores : vector [*] numerico
        indice : numerico
        mayor : numerico
        dimension : numerico
inicio
        imprimir ("\nIndimensión del vector:")
        leer (dimension)
        dim (valores, dimension)
        imprimir ("\nIngrese ", alen (valores), " números separados 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_dim.sl
/*
* Lee un vector de n elementos numéricos, y visualizar el mayor.
* (c) lopezquesada@iessanjuanbosco.es
*/
var
valores : vector [*] numerico
indice : numerico
mayor : numerico
dimension : numerico
inicio
imprimir ("\nIndimensión del vector:")
leer (dimension)
dim (valores, dimension)
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
Linea: 2 Columna: 22
Visor de Expresiones
Ventana de ejecución
Datos previos:
Indimensión del vector:5
Ingrese 5 números separados por comas:
2,3,-5,45,1
El mayor de este vector:
2,3,-5,45,1[45]

Ejemplo en el Entorno Educativo Algorítmico: Lenguaje SL.

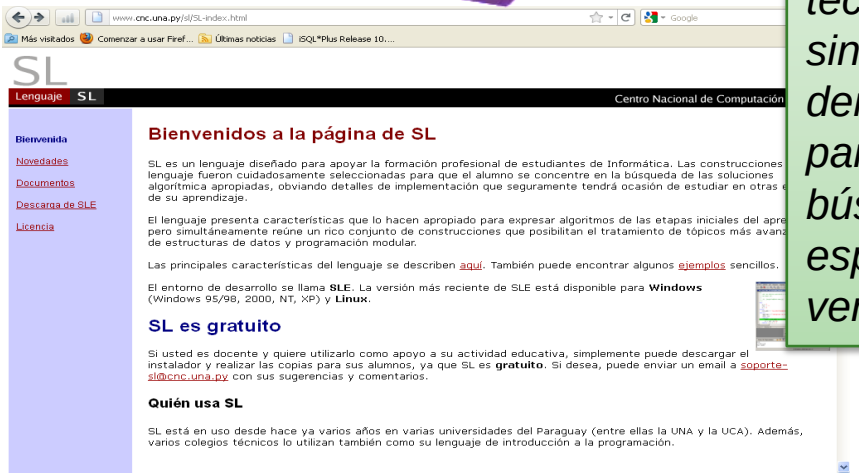



SLE
Archivo Edición Ver Ejecutar Ventana Ayuda
E:\IES\IES_1213\0376IAW\UT2\lenguajeSL\ejemplos\suma_matrices.sl
/*
* Sumar dos matrices
* (c) lopezquesada@iessanjuanbosco.es
*/
var
dimension, fila, col : numerico
A,B,S : matriz [*, *] numerico
inicio
imprimir ("\nIndimensión de las matrices cuadradas:")
leer (dimension)
dim (A, dimension,dimension)
dim (B, dimension,dimension)
dim (S, dimension,dimension)
// Lectura de una matriz A
fila=1;
mientras (fila<=dimension) {
col=1;
mientras (col<=dimension) {
imprimir("a[",fila,",",col,"]:")
leer(A[fila,col])
col=col+1
}
fila=fila+1
}
// Lectura de una matriz B
fila=1;
mientras (fila<=dimension) {
col=1;
mientras (col<=dimension) {
imprimir("b[",fila,",",col,"]:")
leer(B[fila,col])
col=col+1
}
fila=fila+1
Linea: 1 Columna: 0
Ventana de ejecución
Datos previos:
Indimensión de las matrices cuadradas:2
a[1,1]:1
a[1,2]:2
a[2,1]:3
a[2,2]:4
b[1,1]:5
b[1,2]:6
b[2,1]:7
b[2,2]:8
s[1,1]:6
s[1,2]:8
s[2,1]:10
s[2,2]:12
Visor de Expresiones
E:\IES\IES_1213\0376IAW\UT2\lenguajeSL\ejemplos\suma_matrices.sl


SLE
Archivo Edición Ver Ejecutar Ventana Ayuda
E:\IES\IES_1213\0376IAW\UT2\lenguajeSL\ejemplos\longitud_secuencia.sl
1 /*
2 * Determinar si el valor es par
3 * (c) lopezquesada@iessanjuanbosco.es
4 */
5 var
6 valor : cadena
7 indice : numerico
8 inicio
9 imprimir ("\nIntroduce la cadena: \n")
10 leer (valor)
11 indice= 1
12 mientras (indice <= strlen(valor))
13 {
14 imprimir("\nvalor(",indice,"):",valor[indice])
15 indice=indice+1;
16 }
17 fin
Ventana de ejecución
Datos previos:
Introduce la cadena:
Prueba ASIR
valor(1):P
valor(2):r
valor(3):u
valor(4):e
valor(5):b
valor(6):a
valor(7):
valor(8):A
valor(9):S
valor(10):I
valor(11):R
Ayuda: ayuda.zip#zip:stdf.html
Contenidos Índice Buscar
(favoritos)
Ayuda de SL
Funciones matemáticas
abs()
arctan()
cos()
exp()
int()
log()
sin()
sqrt()
Funciones de cadenas de caracteres
ascii()
lower()
ord()
pos()
strdup()
strlen()
substr()
Hecho
E:\IES\IES_1213\0376IAW\UT2\lenguajeSL\ejemplos\longitud_secuencia.sl
Inicio
UT2
libro-sl.p...
Ayuda: ...
SLE
Ayuda: ...
Ventana ...
Microsoft...
18:46

Cuestiones y Ejercicios

- Realizar los siguientes problemas utilizando el Pseudocódigo asociado al ***Entorno Educativo Algorítmico SL***.



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 obvie detalles específicos que seguramente tendrá ocasión de ver en otras etapas de su aprendizaje.

Cuestiones y Ejercicios

1. Leer un número y mostrar por la salida estándar si dicho número es o no es par.
2. Leer 2 números y mostrar el producto de ellos.
3. Leer 2 números y determinar el mayor de ellos.
4. Leer 3 números y mostrar el mayor de ellos.
5. Leer un número y mostrar su tabla de multiplicar.
6. Leer una secuencia de 30 números y mostrar la suma y el producto de ellos.
7. Leer una secuencia de números, hasta que se introduce un número negativo y mostrar la suma de dichos números.
8. Leer dos números y realizar el producto median sumas.
9. Leer dos números y realizar la división mediante restas mostrando el cociente y el resto.
10. Lee una secuencia de números y determina cual es el mayor de ellos.
11. Dado un número mostrar su valor en binario.
12. Generar la sucesión de Fibonacci de N términos.
13. Leer una secuencia se números y mostrar cuales de ellos es el mayor y el menor, el proceso finalizará cuando se introduzca un número impar.
14. Leer una secuencia de números y mostrar la suma de los pares.
15. Leer una secuencia de N valores enteros y visualizar la media.
16. Dado dos valores enteros proporcionados por el usuario (x,y) , visualizar la tabla de multiplicar de los valores comprendidos entre [x,y]

Cuestiones y Ejercicios

17. Leer un número y determinar su factorial.
18. Leer un año y determinar si es o no bisiesto.
19. Leer un número y determinar si es o no es primo.
20. Dado un entero visualizar cuantos divisores (entre 1 y dicho número) están comprendidos entre dos valores proporcionados por el usuario.
21. Leer un entero y un entero de un dígito (0..9), visualizar el número de veces que aparece.
22. Leer un entero y determinar si es o no perfecto.
    - *Un número perfecto es un número natural que es igual a la suma de sus divisores, sin incluirse él mismo.*
    - *Así, 6 es un número perfecto, porque sus divisores propios son 1, 2 y 3; y 6 = 1 + 2 + 3. Los siguientes números perfectos son 28, 496 y 8128.*
23. Leer una secuencia de 30 números y mostrar la suma de los primos.
24. Leer una secuencio de 30 números y mostrar la suma de su factorial.
25. Leer una secuencia de números y mostrar la suma de los pares y el producto de los que son múltiplo de 5.
26. Lee una secuencia de N números, visualizando el producto de los comprendidos entre [x,y]
27. Leer una secuencia de números y determinar el mayor de los pares leídos.
28. Leer una secuencia de enteros y almacenarlos en un vector. Dado 4 enteros proporcionados por el usuario, determinar cuántos elementos del vector están en el intervalo [var1 , var2] o en el intervalo [var3 , var4].
29. Dado un vector de edades, determinar cuántos se encuentran en el intervalo [0,9], cuantos en el intervalo [10,19], … y cuantos en [90,99]

Cuestiones y Ejercicios

30. Leer una secuencia de N enteros y almacenarlos en un vector, determinar posteriormente el mayor pero que cumple la condición de ser primo
31. Leer una secuencia de m palabras y determinar la de mayor en longitud y realizar para cada una de ellas su visualización de forma inversa.
32. Leer una secuencia de 20 números almacenarlos en un vector y mostrar la posición donde se encuentra el mayor valor leído.
33. Leer una secuencia de enteros y almacenarlos en un vector. Dado un entero proporcionado por el usuario determinar cuántos son: mayores, menores e iguales.
34. Leer una secuencia de enteros y almacenarlos en un vector. Dado dos enteros proporcionados por el usuario [X,Y] determinar cuántos están comprendidos entre esos dos valores.
35. Dado dos vectores A y B de 15 elementos cada uno, obtener un vector C donde la posición i se almacene la suma de A[i]+B[i].
36. Dado un vector de enteros y un valor proporcionado por el usuario determinar si dicho valor se encuentra o no en el vector..
37. Leer una secuencia de 20 números y almacenar en un vector sus factoriales.
38. Dado dos matrices A y B obtener la suma.
39. Leer una secuencia de enteros y almacenarlos en una matriz. Se pedirá al usuario sobre que fila se quiere trabajar de la cual se obtendrán los factoriales de los enteros que los componen.
40. Dado una matriz determinar la posición (i,j) del mayor.
41. Dada una matriz cuadrada visualizar el mayor de la diagonal.

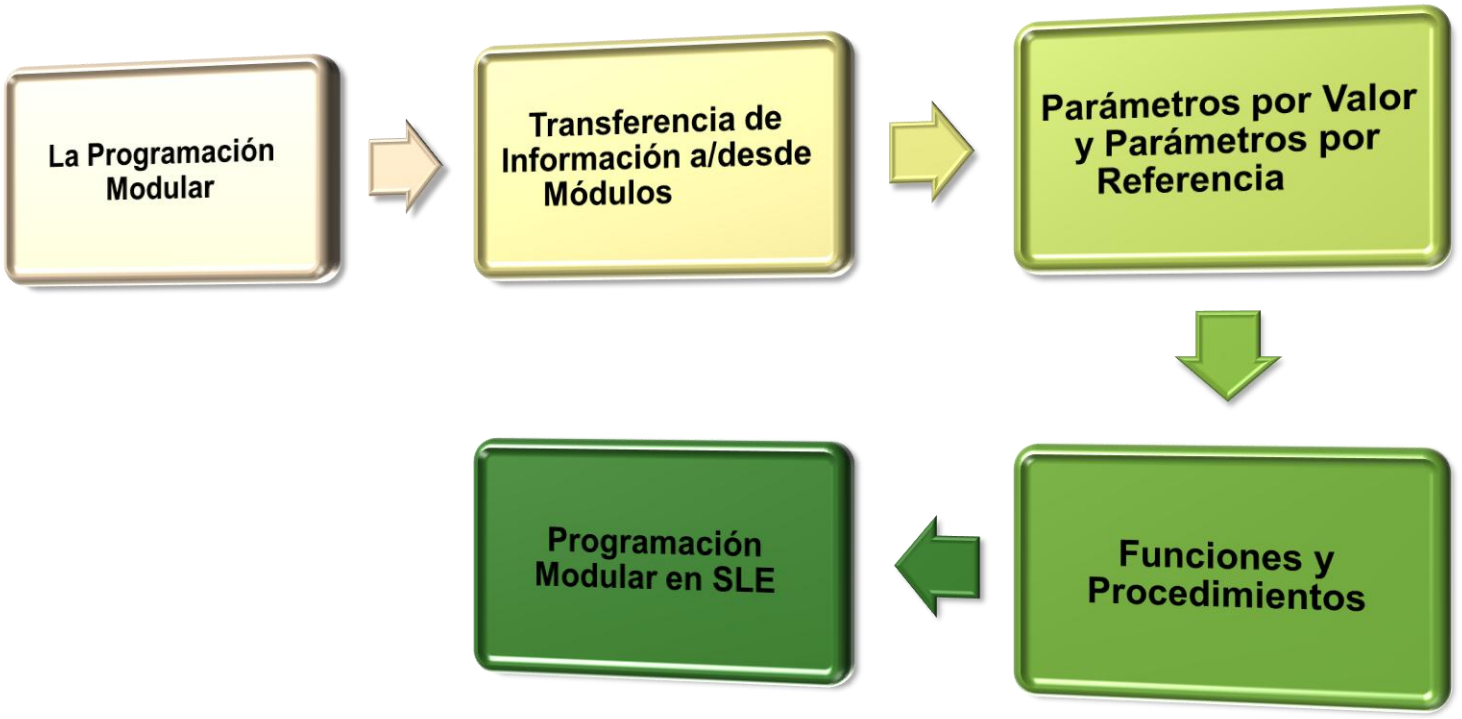
Cuestiones y Ejercicios

42. Dada una matriz de enteros visualizar el vector resultado de almacenar los mayores de cada fila.
43. Dada una matriz [x,y] y un vector[y] determina si hay una fila que sus valores coincidan con los valores del vector proporcionado.
44. Leer una secuencia de 20 números almacenarlos en un vector A[1..20] y mostrar la suma de los elementos que ocupan posiciones pares y el mayor de los que ocupan posiciones impares.
45. Dado un vector de números determina aquellos que sea primos.
46. Dada una matriz determina la fila con mayor número de valores primos.
47. Dado una secuencia de caracteres determina su longitud.
48. Dado una secuencia de caracteres determina si es o no palíndromo.
49. Dado un vector de secuencias determinar las que son palíndromo.
50. Dado una secuencia de caracteres determinar cuantas letras minúsculas tiene.
51. Dada dos secuencias de caracteres determinar el orden lexicográfico.

- Ejercicios realizados por nuestro ex alumno Ismael García utilizando el Pseudocódigo asociado al ***Entorno Educativo Algorítmico SL***.


php
Microsoft
Visual Basic 2005
Express Edition
C++
Perl
javascript
python™
Ruby
A Programmer's Best Friend
Java

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:

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).

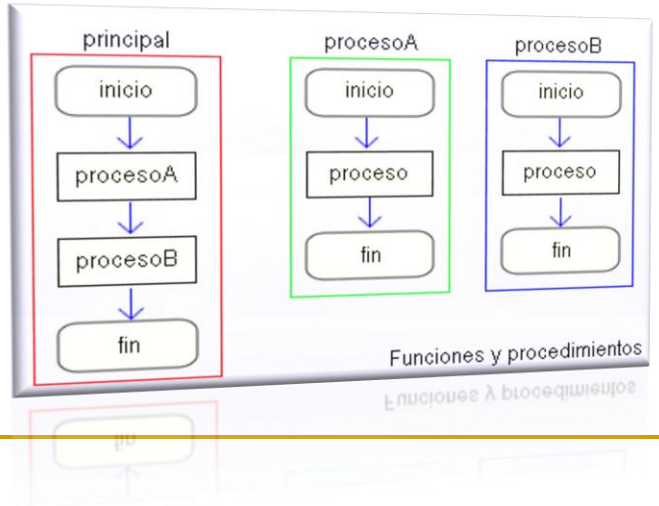
Módulos con parámetros (existe comunicación entre el programa principal y los módulos, y entre ellos).

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 indico 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));

Programación modular en SLE.

```
/*
 * Este es el clásico ejemplo del cálculo del factorial
 * de un entero positivo n.
 *
 * (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
```

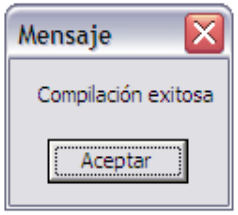
```
sub fact (n : numerico) retorna numerico
var
   producto = 1
   indice: numerico
inicio
   indice=1
   mientras (indice<n)  {
      producto=producto*indice;
      indice=indice+1
   }
   retorna producto
fin
```

SLE

Archivo Edición Ver Ejecutar Ventana Ayuda

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 ("\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", 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
```

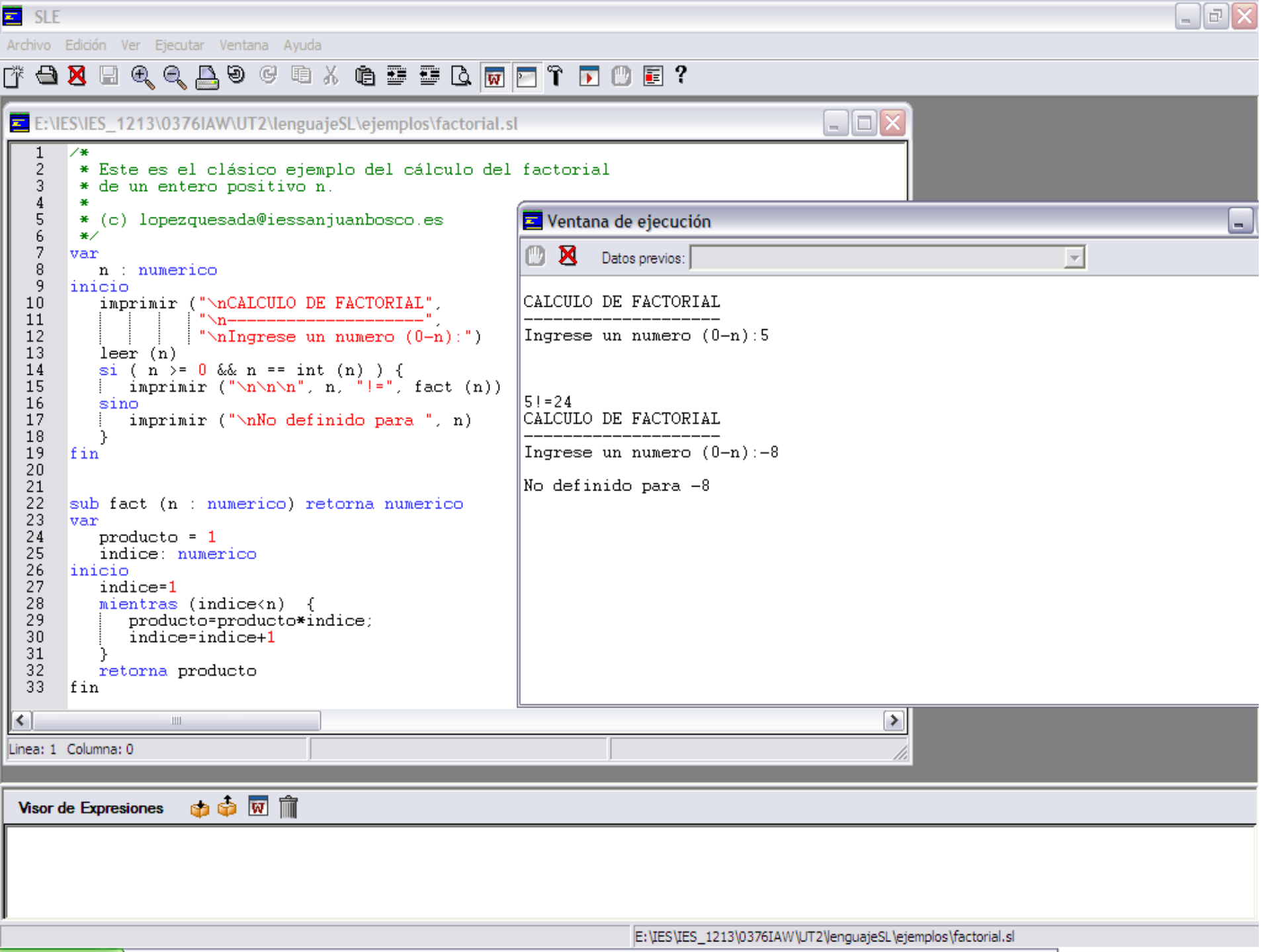


Linea: 28 Columna: 21

Visor de Expresiones

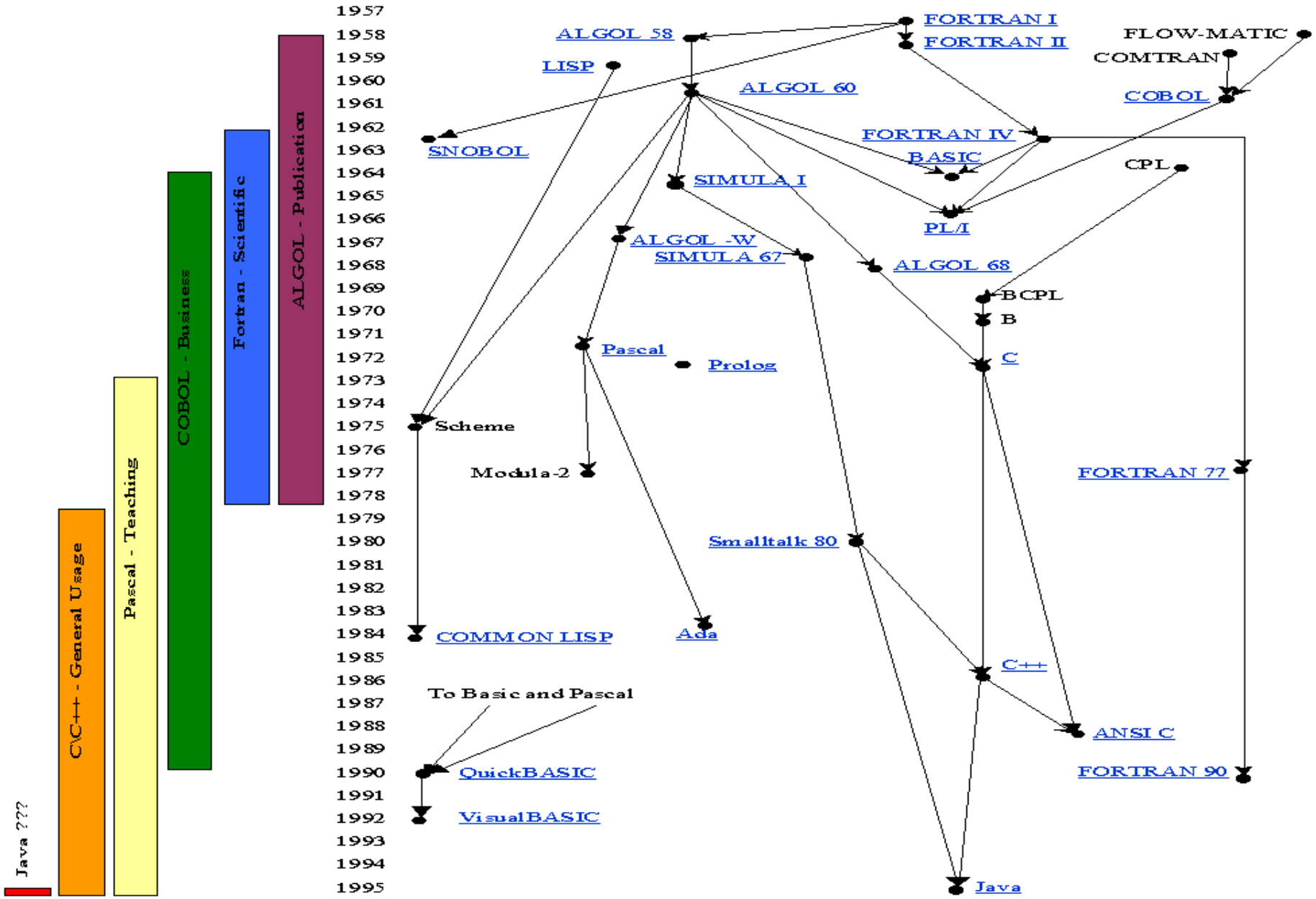
Compilar

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


SLE
Archivo Edición Ver Ejecutar Ventana Ayuda
E:\IES\IES_1213\0376IAW\UT2\lenguajeSL\ejemplos\factorial.sl
/*
* Este es el clásico ejemplo del cálculo del factorial
* de un entero positivo n.
*
* (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
Linea: 1 Columna: 0
Ventana de ejecución
Datos previos:
CALCULO DE FACTORIAL

Ingrese un numero (0-n):5
5!=24
CALCULO DE FACTORIAL

Ingrese un numero (0-n):-8
No definido para -8
Visor de Expresiones
E:\IES\IES_1213\0376IAW\UT2\lenguajeSL\ejemplos\factorial.sl


ALGOL - Publication
Fortran - Scientific
COBOL - Business
Pascal - Teaching
C\C++ - General Usage
Java ???
1957
1958
1959
1960
1961
1962
1963
1964
1965
1966
1967
1968
1969
1970
1971
1972
1973
1974
1975
1976
1977
1978
1979
1980
1981
1982
1983
1984
1985
1986
1987
1988
1989
1990
1991
1992
1993
1994
1995
ALGOL 58
FORTRAN I
FORTRAN II
FLOW-MATIC
COMTRAN
LISP
ALGOL 60
COBOL
SNOBOL
FORTRAN IV
BASIC
CPL
SIMULA I
PL/I
ALGOL -W
SIMULA 67
ALGOL 68
BCPL
B
Pascal
Prolog
C
Scheme
Modula-2
FORTRAN 77
Smalltalk 80
COMMON LISP
Ada
C++
To Basic and Pascal
ANSI C
QuickBASIC
FORTRAN 90
VisualBASIC
Java