

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*

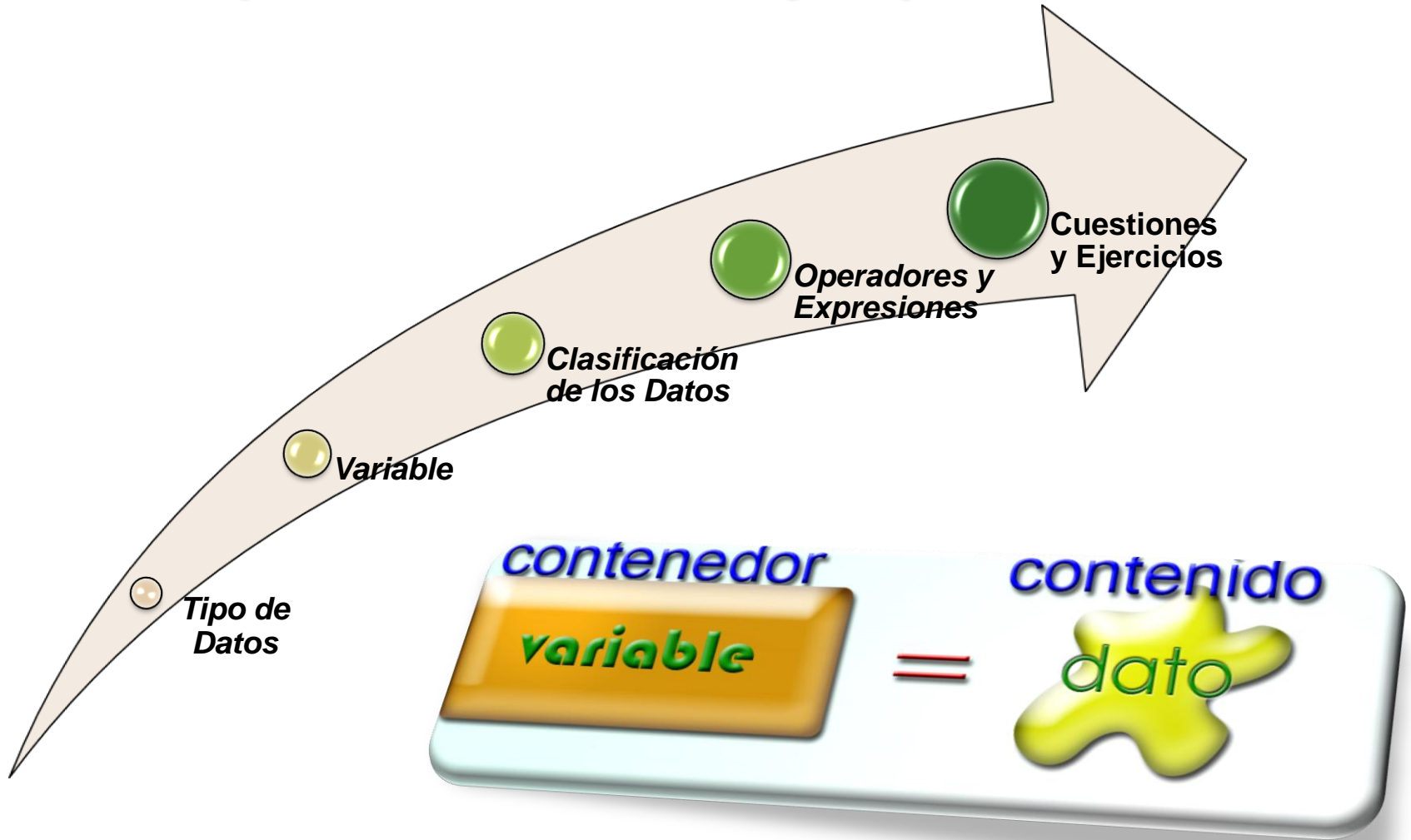




INDICE DE CONTENIDOS



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*:

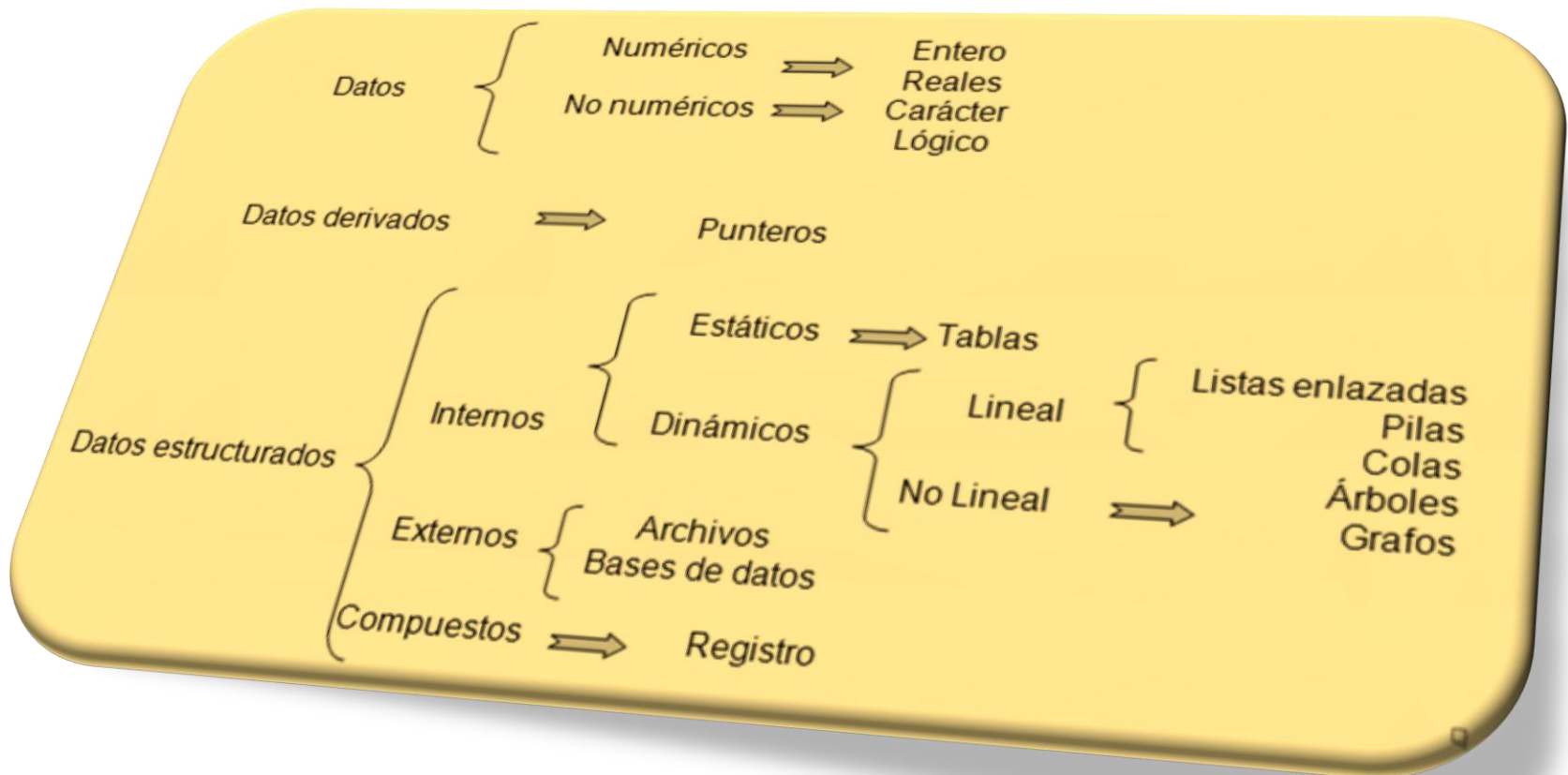
Una constante es un identificador cuyo valor asociado es fijo; realmente lo que ocurre es que en el proceso de pre compilación el identificador se sustituye directamente por el literal asociado. En el caso de C tenemos la palabra reservado DEFINE para la creación de macros/constantes.

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 $\text{contador}+5$, 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

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.

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"

Cuestiones y Ejercicios

Calcula el valor de cada una de las siguientes expresiones aritmética:

$$27 \bmod 4 + 15 / 4$$

$$37 \setminus 4 + 4 + (-7)$$

$$9 + 2 / 3 * 25 * 3$$

$$(7 * 3 - 4) - 45$$

Calcular el resultado de las siguientes expresiones relacionales:

$$(27 \bmod 4) + 15 / 4 \leq 4 + (-7)$$

$$9 + (2 / 3) > 3 * (25 * -3)$$

$$(7 * 3 \bmod 45) - 45 \lt \gt 2 / 3$$

$$'a' < 'z' \quad \text{véase el código ascii}$$

$$"Juan" == "Juan" \text{ véase el código ascii}$$

Calcular el resultado de las siguientes expresiones lógicas:

$$25 \leq 7 \textbf{ y no } (7 = \gt 2)$$

$$\textbf{no} ("Juan_" < "pepe" \textbf{ and } (37 < 4 + (-7))) \textbf{ o } ('3' < 'z')$$

$$24 < 5 \textbf{ y } 10 \geq 10 \textbf{ o } 10 = 5$$

$$(\textbf{no} (6 / 3 > 3) \textbf{ o } 7 < 7) \textbf{ y } (3 \geq 9 \setminus 2 \textbf{ o } 2 + 3 \leq 7 / 2)$$

Cuestiones y Ejercicios

- ✚ ¿Cuáles son los tipos elementales de X, Y Z ...?
- ✚ ¿Cuál es la sintaxis para declarar una variable en X, Y Z ...?
- ✚ ¿Qué tipo de datos se usarían para declarar variables que representan los valores siguientes?

La edad de un niño.

Las medidas de un edificio.

El nombre de una persona.

La letra de su apartamento.

Una variable que indique si alguien tiene hijos o no.

- ✚ Evalúe las siguientes expresiones aritméticas.

a) $6 - 3 * 4 / 4 + 5 * 2$

b) $(2 * 3 + (6 \text{ div } 3 - 1)) ** 2$

c) $6 \text{ mod } 3 - 9 / 4 * 2 + 1$

Cuestiones y Ejercicios

Inserte paréntesis para indicar el orden en que se aplican los operadores.

- a) $a \text{ div } b * c \text{ mod } e - f * g$
- b) $a - b * c / d + e * f$
- c) $a * (b + c) + \text{abs}(d * e / f) - g * (h + i / j)$

Elimine los paréntesis innecesarios de las siguientes expresiones.

- a) $((a \text{ mod } b) / c) * (d + e)$
- b) $((a / b) * c) - d$
- c) $(a + (b * (c + d))) - ((e + f) / g * h)$

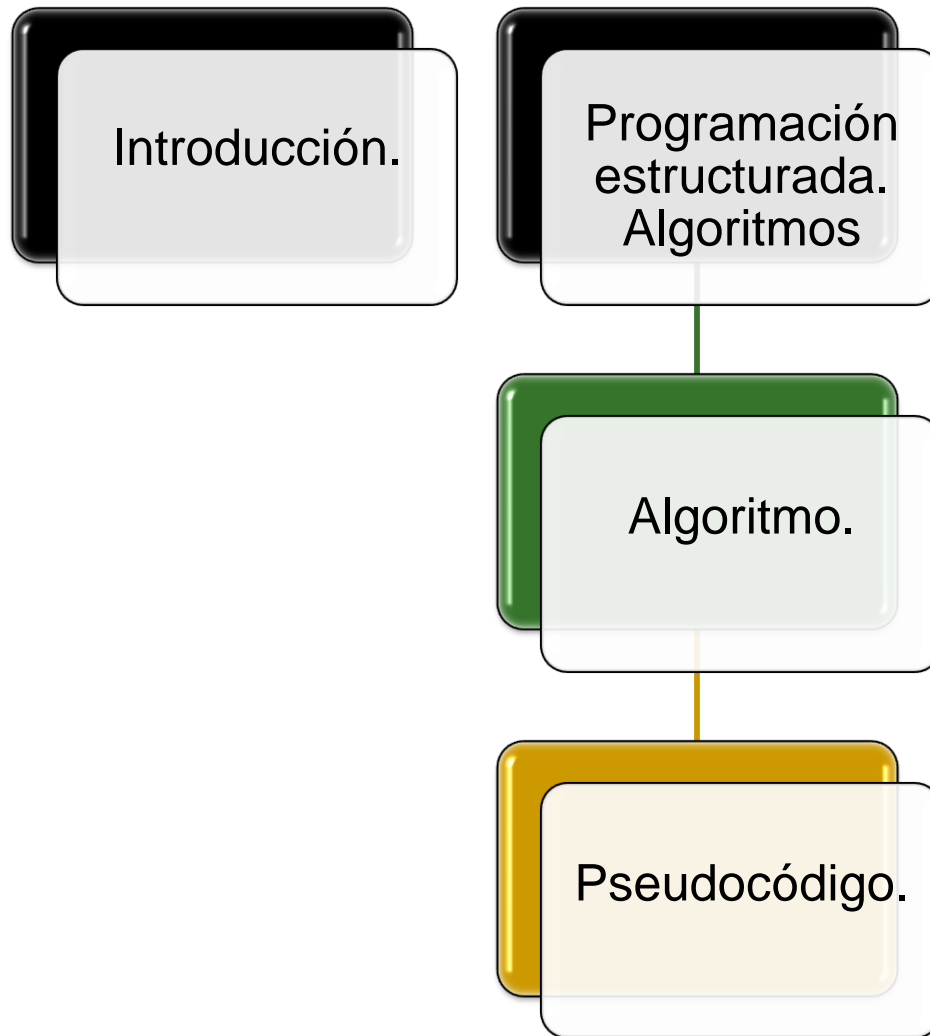
Evalué las siguientes expresiones condicionales.

- a) $(5 / 2 * 3 - 1) = 4 \text{ div } 2 \text{ mod } 2$
- b) $(7 \text{ div } 3) > (3 / 4) ** 2 * (2 + 10 \text{ mod } 5)$
- c) $10 < = 3 + 4 - 2$

Evalué las siguientes expresiones con los operadores aritméticos, relacionales y lógicos.

- a) $\text{no } (15 > = 7 ** 2) \text{ o } (43 - 8 * 2 \text{ div } 4 <> 3 * 2 \text{ div } 2)$
- b) $(15 > = 7 * 3 ** 2 \text{ y } 8 > 3 \text{ y } 15 > 6) \text{ o no } (7 * 3 < 5 + 12 * 2 \text{ div } 3 ** 2)$
- c) $\text{no } ((7 * 3 \text{ div } 2 * 4 < 20) \text{ o } (15 / 2 * 6 < = 15 * 2 / 17))$

Programación estructurada. Algoritmos



Abstract/Resumen:

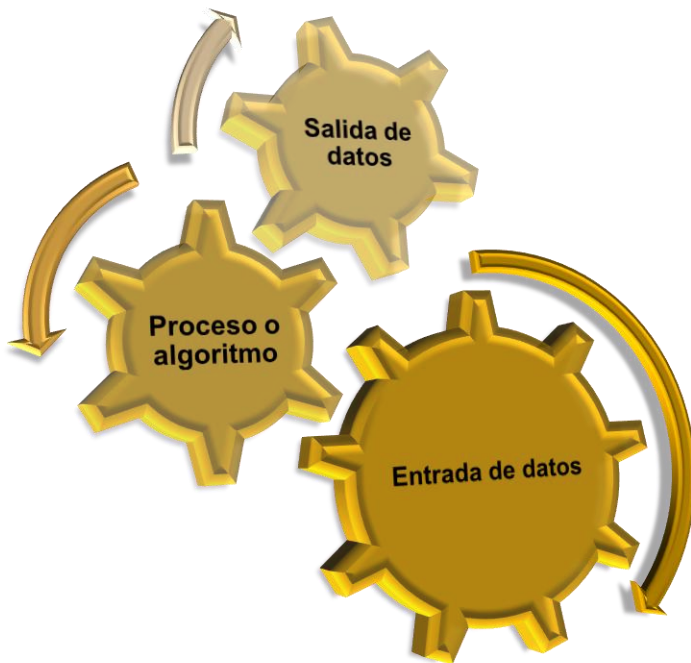
La palabra algoritmo se deriva de la traducción al latín de la palabra árabe alkhwarizmi, nombre de un matemático y astrónomo árabe que escribió un tratado sobre manipulación de números y ecuaciones en el siglo IX. Un algoritmo es una serie de pasos organizados que describe el proceso que se debe seguir,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

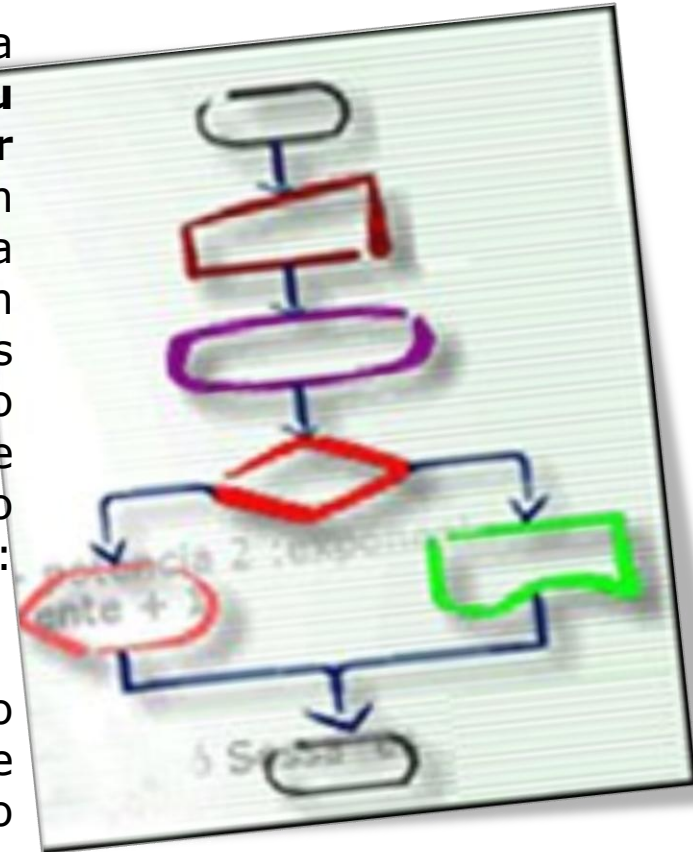
Por último el bloque de salida de datos estará formado por todas aquellas instrucciones que toman los datos depositados en memoria principal una vez procesados los datos de entrada, enviándolos seguidamente a un dispositivo o periférico.

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.

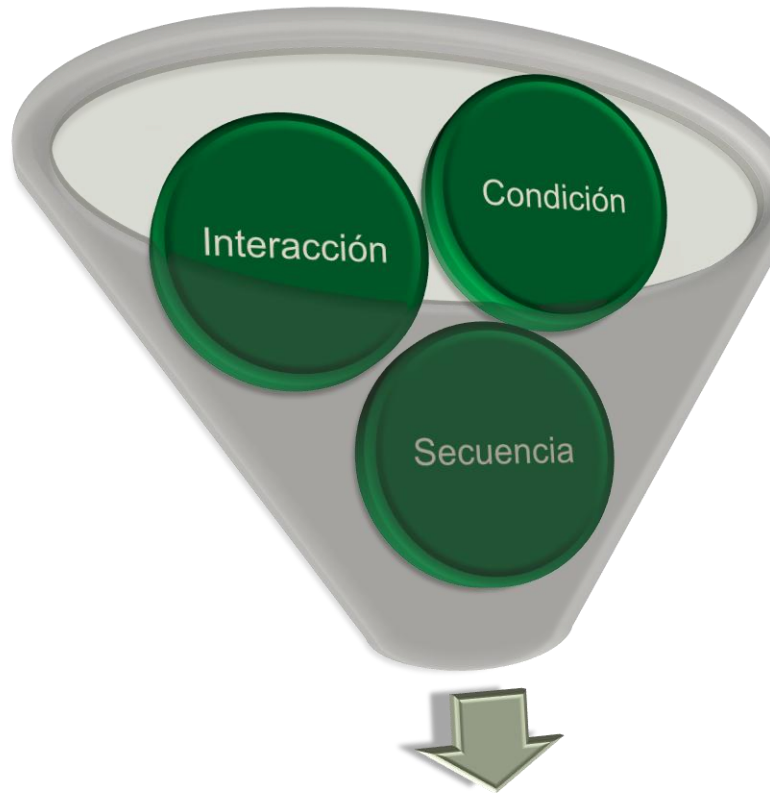
Programación estructurada. Algoritmos

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:

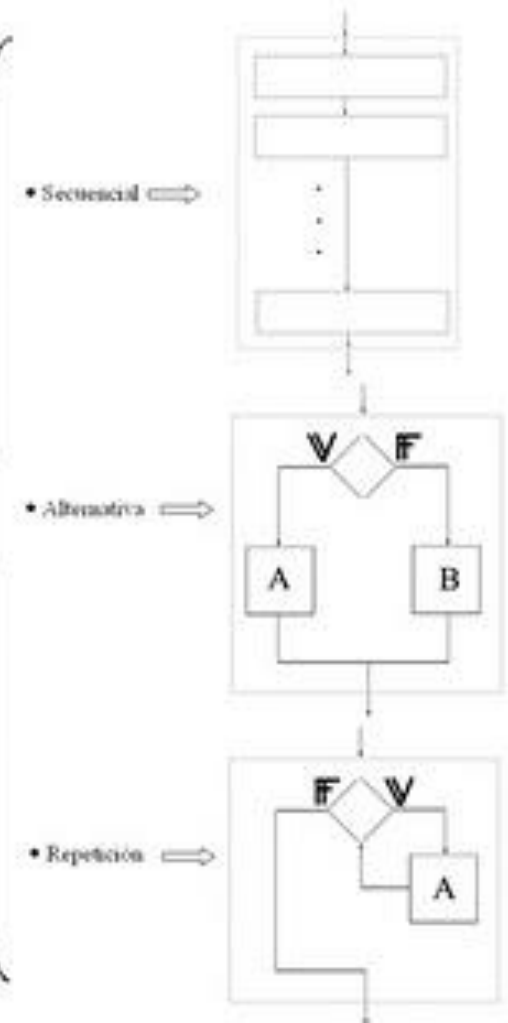
- + No puede ser ejecutada directamente por un ordenador, por lo que tampoco es considerado como un lenguaje de programación propiamente dicho.
- + Ser una forma de representación muy sencilla de aprender y utilizar.
- + Permitir el diseño y desarrollo de algoritmos totalmente independientes del lenguaje 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

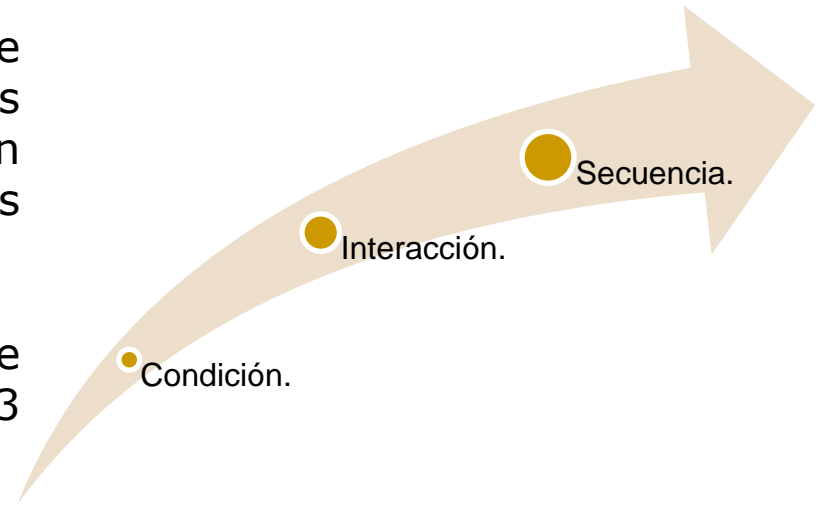


Estructuras de Control

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

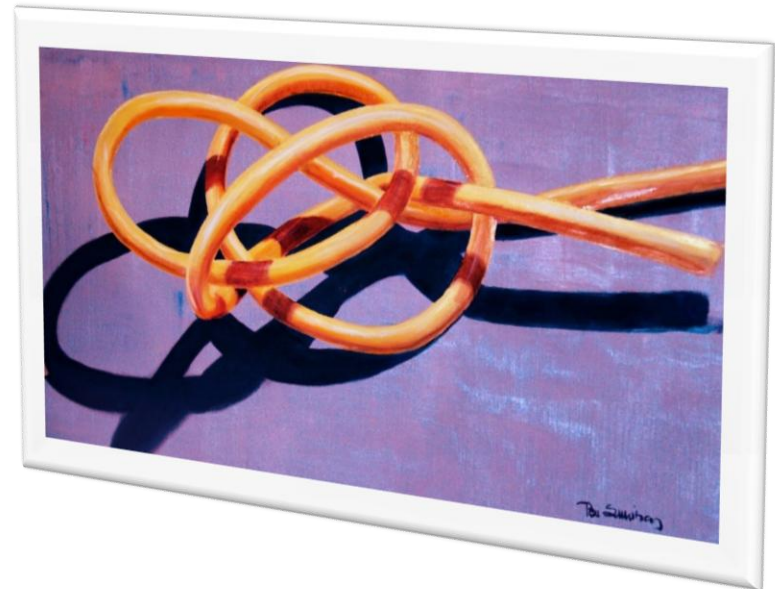
MIENTRAS-HACER / WHILE

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

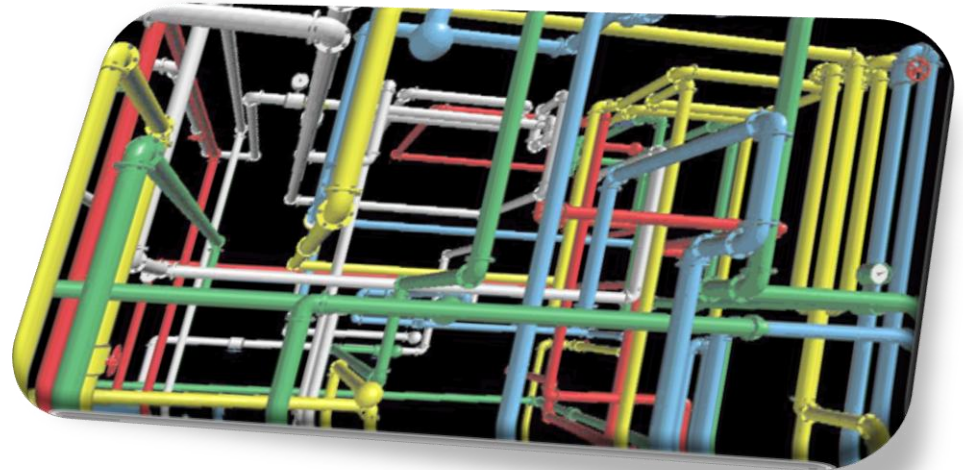
REPETIR-HASTA

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

Do

(Bloque de sentencias)

Until (Condición)



Descripción de un pseudolenguaje 2 ASIR

Comentarios

Se reconoce como comentario cualquier grupo de caracteres situados entre /* y */, aunque estén en diferentes líneas. Por ejemplo:

**/* Este es un comentario que
 ocupa más de una línea */**

Se pueden definir comentarios de una sola línea mediante //.

// Este comentario ocupa una sola línea

En el caso de comentarios de una sola línea no hay indicador de fin de comentario.

Identificadores

Son los nombres dados a variables, funciones, etiquetas u otros objetos definidos por el programador. Un identificador puede estar formado por:

1. Letras minúsculas.
2. Carácter de subrayado

Descripción de un pseudolenguaje 2 ASIR

Operadores y expresiones

Un operador es un símbolo que indica alguna operación sobre uno o varios objetos del lenguaje, a los que se denomina operandos.

Atendiendo al número de operandos sobre los que actúa un operador, estos se clasifican en:

Unarios: actúan sobre un solo operando

Binarios: " " 2 operandos

Atendiendo al tipo de operación que realizan, se clasifican en:

Aritméticos.

Relacionales.

Lógicos.

Descripción de un pseudolenguaje 2 ASIR

Operadores aritméticos

Los operadores aritméticos se exponen en el cuadro siguiente:

	OPERADOR	DESCRIPCIÓN
UNARIOS	-	Cambio de signo
BINARIOS	-	Resta
	+	Suma
	*	Producto
	/	División Real
	\	División Entera
	MOD	Resto de división entera

Descripción de un pseudolenguaje 2 ASIR

Operadores relacionales

Se usan para expresar condiciones y describir una relación entre dos valores.

	OPERADOR	DESCRIPCIÓN
BINARIOS	>	Mayor que
	>=	Mayor o igual que
	<	Menor que
	<=	Menor o igual que
	=	Igual que
	<>	Diferente que

Descripción de un pseudolenguaje 2 ASIR

Operadores lógicos

Actúan sobre expresiones booleanas, es decir, sobre valores verdadero o falso generados por expresiones como las explicadas en el caso anterior. Son los siguientes:

	OPERADOR
UNARIOS	NOT
BINARIOS	AND
	OR

Operadores [] y ()

Los corchetes se utilizan para acceder a los elementos de un array. Los paréntesis sirven para clarificar una expresión o para modificar las reglas de prioridad entre operadores

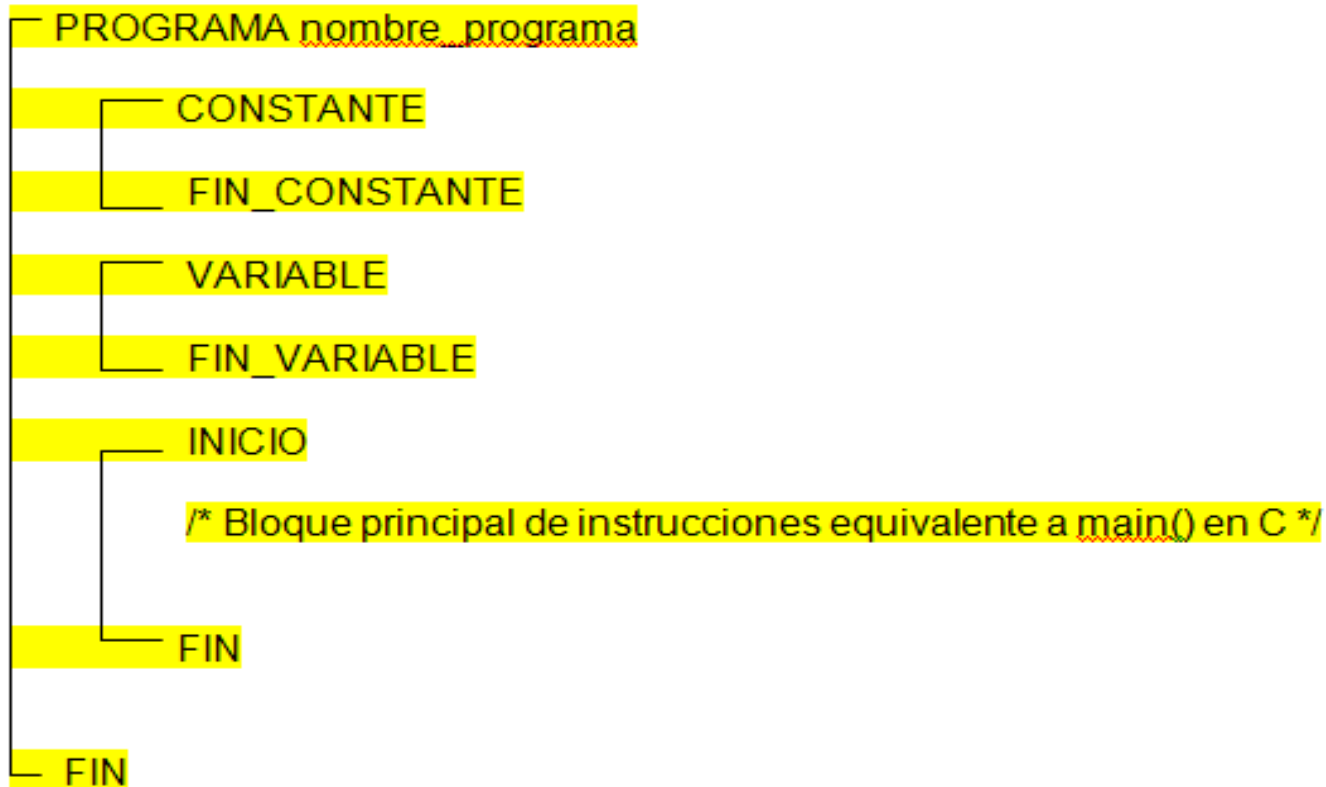
Descripción de un pseudolenguaje 2 ASIR

Prioridad entre los operadores

Nivel de prioridad	Operadores
1º	() []
2º	* / \ MOD
3º	+ -
4º	< <= > >= = <>
5º	NOT
6º	AND
7º	OR

Descripción de un pseudolenguaje 2 ASIR

Estructura del pseudoprograma secuencial



Descripción de un pseudolenguaje 2 ASIR

Constantes

En el primer bloque definiremos las distintas constantes una por línea sin finalizar en punto y coma. La nomenclatura que utilizaremos para la creación de constantes será la siguiente:

nombre_constante literal

Consideraremos los siguientes literales, literal entero, literal carácter, literal real y literal secuencia, siguiendo las mismas reglas que el lenguaje C.

Vease el siguiente ejemplo:

```
CONSTANTE  
FIN        's'  
VALOR     13  
CAMBIO   3.5  
CIUDAD   "lorca"  
FIN_CONSTANTE
```

Descripción de un pseudolenguaje 2 ASIR

Variable

1. Concepto de variable: Posición de memoria que almacena un dato cuyo valor puede cambiar durante la ejecución.
2. En el segundo bloque definiremos las distintas variables que utilizaremos en mi pseudoprograma considerando que dichas variables son globales, es decir, visibles desde cualquier punto de mi fichero fuente/source. En este punto nos surge dos cuestiones, ¿Cómo definimos una variable?, y ¿qué tipos de datos vamos a usar en nuestro pseudolenguaje?.

Definición de una variable:

tipo_variable nombre_variable [=valor_inicial];

Habrà una definición por línea finalizada cada una en un punto y coma. Opcionalmente la variable podrá tener un valor inicial que podrá ser tanto un literal como una constante, esta última previamente definida en su correspondiente bloque.

Descripción de un pseudolenguaje 2 ASIR

Tipos de datos:

Consideramos que en nuestro pseudolenguaje tenemos los siguientes tipos de datos básicos o predefinidos.

- + ENTERO: Cualquier entero positivo o negativo.
- + CARACTER: Cualquier elemento del código ASCII de 8 bits.
- + REAL: Cualquier valor positivo o negativo real.
- + SECUENCIA: Cualquier conjunto finito de caracteres del código ASCII. Reglas de uso de las secuencias:

1. El último carácter de la secuencia es el literal '\0'. Para acceder a cada uno de sus componentes usamos la siguiente nomenclatura:
variable_secuencia (expresión)
2. La expresión que aparece entre paréntesis se evaluará dando como resultado un valor entero. Indicar que la expresión podrá ser: un literal entero, una variable entera o una expresión aritmética que al evaluarse de un valor entero.

Indicar que el primer elemento o carácter quedará referenciado de la siguiente forma: *variable_secuencia (0)*.

Descripción de un pseudolenguaje 2 ASIR

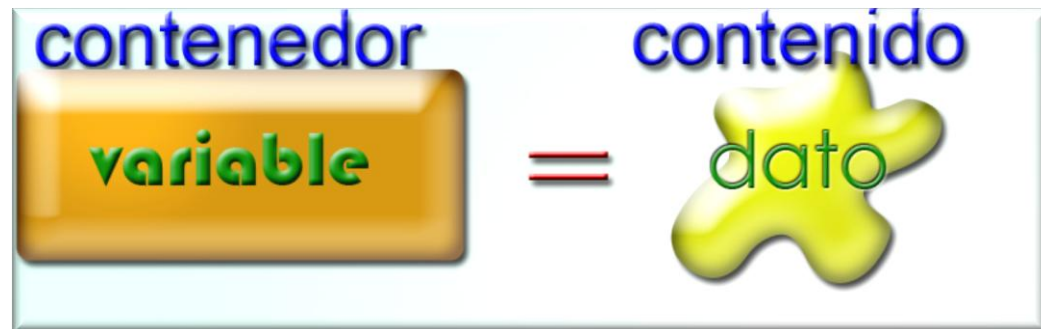
Tipos de datos:

Ejemplo de definición del bloque de variable:

VARIABLE

<i>CARACTER</i>	<i>fin= 'f';</i>
<i>ENTERO</i>	<i>indice=0;</i>
<i>SECUENCIA</i>	<i>nombre= "desconocido";</i>
<i>SECUENCIA</i>	<i>direccion;</i>

FIN_VARIABLE



Descripción de un pseudolenguaje 2 ASIR

Sentencias

Sentencia de asignación

Se produce la evaluación de la expresión que dará un resultado, que deberá ser compatible con el tipo de la variable. Dicho valor se le asignará machacando el dato que tuviera previamente.

Variable = Expresión;

Sentencia de entrada standard

LEER (nombre_variable);

Cuando se ejecute esa sentencia el sistema estará esperando a que introduzcas o pulses una serie de teclas, proceso que finalizará cuando pulses la tecla intro. Ese dato introducido por el teclado se le asignara a la variable que se encuentra entre paréntesis, quedando actualizada a partir de ese momento con dicho valor. En el caso de una variable secuencia el sistema incorporará como último carácter '\0'.

Descripción de un pseudolenguaje 2 ASIR

Sentencia de salida standard

ESCRIBIR (expresión);

Esta sentencia evaluará la expresión y mostrara el resultado por la salida estándar que en nuestro caso será la pantalla. La expresión podrá ser: una variable, un literal, una secuencia o una concatenación de variables y literales y secuencias.

```
ESCRIBIR("Introduce un valor");  
ESCRIBIR(var1*var2);  
ESCRIBIR(var1+" es mayor que"+(var2*3));
```



Descripción de un pseudolenguaje 2 ASIR

Estructuras iterativas

Esta construcción permitirá ejecutar un número n de veces un bloque de sentencias o instrucciones en función de una determina condición. Podemos determinar las siguientes estructuras iterativas:

MIENTRAS (expresión_condicion) HACER
"cuerpo del bucle"
FIN _ MIENTRAS

Funcionamiento:

El cuerpo del bucle formado por n sentencias se ejecutará mientras que se cumpla la condición, es decir, mientras que sea verdadera. Por lo anterior indicar que la condición podrá ser o una expresión relacional o una expresión lógica. Es evidente que si la primera vez la condición es falsa no entrará al cuerpo del bucle, ejecutándose directamente la sentencia que hay a continuación de FIN_MIENTRAS.

Descripción de un pseudolenguaje 2 ASIR

Estructuras iterativas

HACER
 “cuerpo del bucle”
MIENTRAS (condición);

Funcionamiento:

El cuerpo del bucle se ejecutará mientras la condición sea verdadera con la peculiaridad de que dicho cuerpo y por lo tanto sus instrucciones se ejecutarán como mínimo una vez.

Descripción de un pseudolenguaje 2 ASIR

Estructuras iterativas

REPETIR
 "cuerpo del bucle"
HASTA (condición);

Funcionamiento:

El cuerpo del bucle se ejecutará tantas veces hasta que se cumpla la condición, que es decir que el cuerpo del bucle se ejecutará mientras la condición es falsa. Como se observa al menos el cuerpo se ejecutará una vez.

Descripción de un pseudolenguaje 2 ASIR

Estructuras condicionales

SI (CONDICION) ENTONCES
“bloque de sentencias”
FIN _ SI

Funcionamiento:

Se evalúa la condición que será una expresión relacional o lógica, si el resultado es verdadero el flujo se introduce en el cuerpo, por el contrario si es falsa salta a la instrucción siguiente a FIN _ SI.

Descripción de un pseudolenguaje 2 ASIR

Estructuras condicionales

```
SI (CONDICION) ENTONCES  
    <<BLOQUE V>>  
SINO  
    << BLOQUE F>>  
FIN _ SI
```

Funcionamiento:

Se evalúa la condición, si el resultado es verdadero se ejecuta el bloque V y no el bloque F, siguiendo el flujo por la instrucción que haya después de FIN _ SI. Si la expresión se evalúa como falsa se ejecuta el bloque F, siguiendo el flujo por la instrucción que hay después de FIN_SI.

Descripción de un pseudolenguaje 2 ASIR

Estructuras condicionales

```
SI (CONDICION_1) ENTONCES  
    <<BLOQUE_1>>  
SINO SI (CONDICION_2) ENTONCES  
    <<BLOQUE_2>>  
SINO SI  
    .....  
    SINO SI (CONDICION _N) ENTONCES  
        <<BLOQUE_N>>  
FIN _ SI
```

Funcionamiento:

Se evalúa la condición_1, si el resultado es V se ejecuta el bloque_1 siguiendo el flujo por la instrucción siguiente a FIN _ SI. Si la condición_1 es falsa se evalúa la condicion_2, si el resultado es V se ejecuta el bloque_2 siguiendo el flujo por la instrucción siguiente a FIN _ SI y así hasta llegar a evaluar una condición como verdadera. Si todas las condiciones son falsas no se ejecuta ningún bloque de sentencias.

Descripción de un pseudolenguaje 2 ASIR

Estructuras condicionales

```
SI (CONDICION_1) ENTONCES  
    <<BLOQUE_1>>  
SINO SI (CONDICION_2) ENTONCES  
    <<BLOQUE_2>>  
SINO SI (CONDICION_i)  
    .....  
SINO SI (CONDICION_N) ENTONCES  
    <<BLOQUE_N>>  
SINO  
    << BLOQUE_POR_DEFECTO>>  
FIN _ SI
```

Funcionamiento:

Equivalente a la anterior estructura con la peculiaridad de que si no hay condición que se evalúe con resultado verdadero, se ejecutará el BLOQUE_POR_DEFECTO.

Descripción de un pseudolenguaje 2 ASIR

Estructuras condicionales

```
SELECTOR (EXPRESION)  
  CASO_1: (BLOQUE_1)  
    FIN:  
  CASO_2: (BLOQUE_2)  
    FIN:  
  CASO_3: (BLOQUE_3)  
    FIN:  
  CASO_N: (BLOQUE_N)  
    FIN:  
  DEFECTO: (BLOQUE DEFECTO)  
FIN:  
FIN_SELECTOR
```

Funcionamiento:

Se evalúa la expresión y dado el valor obtenido se determina si coincide con alguno de los distintos casos, ejecutándose su bloque de instrucciones, opcionalmente después del bloque tenemos la instrucción FIN, si existe esa instrucción el flujo prosigue por la instrucción que haya después del FIN _ SELECTOR, si no tenemos esa instrucción se ejecutará el bloque siguiente hasta llegar a un FIN o a FIN _ SELECTOR.

El caso defecto es opcional y si aparece el bloque se ejecuta en 2 ocasiones:

1. Cuando el valor de la expresión no coincide con los valores dados en los distintos casos.
2. Cuando entre en un bloque y no se encuentre en la ejecución una instrucción de FIN.

Descripción de un pseudolenguaje 2 ASIR

Vectores y Matrices

Vectores

Son estructuras estáticas por lo que su tamaño no variará a lo largo del tiempo. Es un conjunto finito de datos homogéneos y por lo tanto del mismo tipo, almacenados en memoria dinámica.

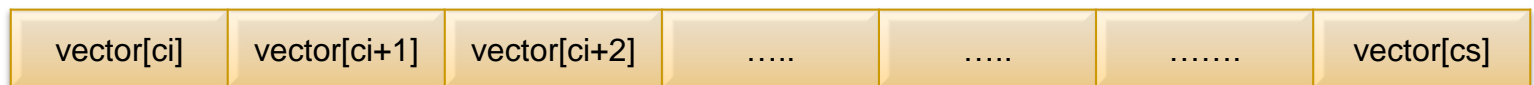
Un vector como una variable tiene asociado un nombre pero no hace referencia a un solo dato sino a 'n' datos.

Un dato de un vector queda referenciado mediante 2 elementos:

1. El nombre del vector.
2. Un entero denominado índice que identifica la posición de dicho dato dentro del vector.

Definición

Array o vector:



Descripción de un pseudolenguaje 2 ASIR

Vectores y Matrices: *Vectores*

Los vectores tienen:

1. Nombre: numero[x], donde x es una expresión.
2. Tamaño: 20 (nº máximo de elementos.)
3. Dimensión: nº de índices que se necesitan para referenciar un elemento.
4. Cota_inf: índice del primer elemento.
5. Cota_sup: índice del último elemento.

Un vector es una estructura unidimensional, que para acceder a cualquier posición habrás que indicar el nombre del vector y el índice asociado al elemento a referenciar/manipular.

Los vectores se utilizan para guardar temporalmente datos del mismo tipo con el fin de manipularlos.

Tipo _ Base nombre_vector [ci... cs]

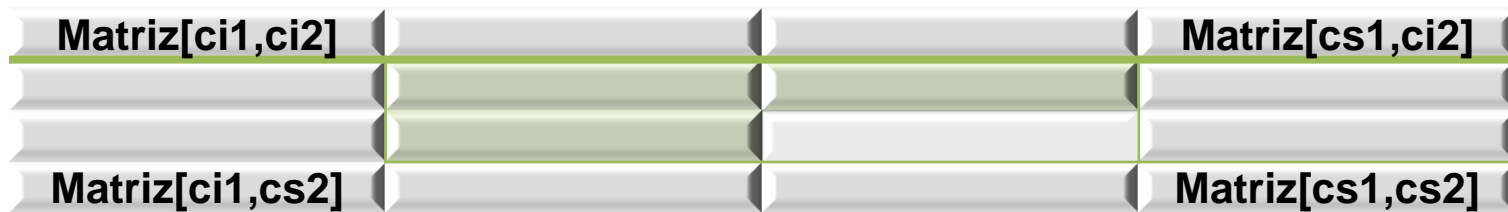
El tipo base será un de los tipos predefinidos para el pseudolenguaje, indicando así que los n elementos asociados al vector son de dicho tipo. Las cotas serán literales o constantes enteras. Indicar que estas estructuras estáticas podrán ser inicializadas.

Descripción de un pseudolenguaje 2 ASIR

Vectores y Matrices

Matrices

Son estructuras de datos estáticas donde un elemento queda identificado mediante dos números enteros el primero que representa lo que sería FILA, y el segundo que conocemos como COLUMNA. Esto significa que una matriz es una estructura bidimensional.



Descripción de un pseudolenguaje 2 ASIR

Vectores y Matrices: *Matrices*

Considerando lo mencionado respecto a la dimensión de un vector y a las dimensiones de una matriz, podemos generalizar las definiciones de la siguiente manera:

Un array 'n' dimensional es una estructura estática de información que contiene 'm' elementos y que cada elemento queda identificado por 2 cosas:

1. Nombres del array.
2. Índices que identifican su posición (tantos como dimensiones).

Tipo_base nombre_matriz [ci1...cs1, ci2...cs2] ;

Ejemplo de definición de matrices:

```
VARIABLE  
          ENTERO            vector[1..2,1..2]={12,456},{12,-9}};  
          SECUENCIA        nombres[1..10,10..20];  
          SECUENCIA        ciudades[1..10];  
FIN_VARIABLE  
  
// LEER(ciudades[indice]); ciudades[1](indice)  
// LEER(nombres[f][c]); nombres[f][c](indice)
```

Descripción de un pseudolenguaje 2 ASIR

Ejemplo 1

Leer un número y mostrar por la salida estándar si dicho número es o no es par.

```
PROGRAMA Ejercicio1
  CONSTANTE
  FIN_CONSTANTE

  VARIABLE
    ENTERO numero=0;
    ENTERO div;
  FIN_VARIABLE

  INICIO

    ESCRIBIR ( "Introduzca un numero" );
    LEER (numero);
    ESCRIBIR ( "Introduzca un divisor" );
    LEER (div);
    SI ( numero MOD div = 0 ) ENTONCES
      ESCRIBIR ( " El numero" + numero + "es divisible por" + div );
    SINO
      ESCRIBIR ( "El numero no es divisible por" + div );
    FIN_SI

  FIN_INICIO
FIN_PROGRAMA
```

Descripción de un pseudolenguaje 2 ASIR

Ejemplo 2

Leer dos números y mostrar el producto de ellos

PROGRAMA ejercicio 2

CONSTANTE

FIN_CONSTANTE

VARIABLE

ENTERO n1;

ENTERO n2;

ENTERO prod;

FIN_VARIABLE

INICIO

ESCRIBIR ("introduzca un numero");

LEER (n1);

ESCRIBIR ("introduzca otro numero");

LEER (n2);

*prod = n1 * n2;*

ESCRIBIR ("El producto es:" +prod);

FIN

FIN_PROGRAMA

Descripción de un pseudolenguaje 2 ASIR

Ejemplo 3

Leer dos números y determinar el mayor de ellos

```
PROGRAMA ejercicio 3
  CONSTANTE
  FIN_CONSTANTE

  VARIABLE
    ENTERO n1;
    ENTERO n2;
  FIN_VARIABLE

  INICIO
    ESCRIBIR ("introduzca un numero");
    LEER (n1);
    ESCRIBIR ("introduzca otro numero");
    LEER (n2);

    Si (n1 > n2 ) ENTONCES
      ESCRIBIR ("El número mayor es:" +n1);
    SiNO
      ESCRIBIR ("El número mayor es:" +n2);
    FIN_SI
  FIN
FIN_PROGRAMA
```

Descripción de un pseudolenguaje 2 ASIR

Ejemplo 4

Leer una secuencia de 30 números y mostrar la suma y el producto de ellos.

PROGRAMA Ejercicio4

CONSTANTES

C_I = 1;

C_S = 30;

FIN_CONSTANTES

VARIABLE

ENTERO numero;

ENTERO indice;

ENTERO suma=0;

ENTERO multiplicar;

FIN_VARIABLE

Descripción de un pseudolenguaje 2 ASIR

Ejemplo 4

.....

INICIO

```
multiplicar = 1;  
indice=C_I;  
MIENTRAS (indice<=C_S) HACER  
    ESCRIBIR ("introduce un numero");  
    LEER (numero );  
    suma =suma + numero;  
    multiplicar = multiplicar * numero ;  
    indice=indice+1;
```

FIN_MIENTRAS

```
ESCRIBIR ("La suma es "+ suma );  
ESCRIBIR ("El producto es " + multiplica);
```

FIN

FIN_PROGRAMA

Descripción de un pseudolenguaje 2 ASIR

Ejemplo 5

Leer una secuencia de números, hasta que se introduce un número negativo y mostrar la suma de dichos números.

```
PROGRAMA ejercicio 7
  CONSTANTE
  FIN_CONSTANTE

  VARIABLE
  ENTERO n;
  ENTERO suma =0;
  FIN_VARIABLE

  INICIO
  HACER
    ESCRIBIR ("introduzca un numero");
    LEER (n);
    SI (n >= 0) ENTONCES
      suma = suma + n;
    FIN_SI
  MIENTRAS (n >= 0);
  ESCRIBIR ("la suma es:" +suma);
  FIN
FIN_PROGRAMA
```

Descripción de un pseudolenguaje 2 ASIR

Ejemplo 6

Dado un número mostrar su valor en binario

```
PROGRAMA ejercicio6
  CONSTANTE
  FIN_CONSTANTE

  VARIABLE
  ENTERO n;
  ENTERO resto;
  FIN_VARIABLE

  INICIO
  ESCRIBIR ("introduzca un número para pasarlo a binario");
  LEER (n);
  MIENTRAS (n >= 1) hacer
    resto = n mod 2;
    n = n / 2;
    ESCRIBIR (resto);
  FIN_MIENTRAS
  ESCRIBIR ("el binario esta escrito inversamente");
  FIN
FIN_PROGRAMA
```

Descripción de un pseudolenguaje 2 ASIR

Ejemplo 7

Generar enteros de 3 en 3 comenzando por 2 hasta el valor máximo menor que 30. Calculando la suma de los enteros generados que sean divisibles por 5.

PROGRAMA ejercicio7

CONSTANTE

CS=30;

CI=2;

SALTO=3;

DIV=5;

FIN_CONSTANTE

VARIABLE

ENTERO n = CI;

ENTERO suma = 0;

FIN_VARIABLE

INICIO

MIENTRAS (n <= CS) HACER

SI (n mod DIV = 0) ENTONCES

suma = suma + n;

FIN_SI

n = n + SALTO;

FIN_MIENTRAS

ESCRIBIR ("La suma es: "+suma);

FIN

FIN_PROGRAMA

Descripción de un pseudolenguaje 2 ASIR

Ejemplo 8

Leer un número y determina si es o no es primo.

```
PROGRAMA ejercicio8
  CONSTANTE
  FIN_CONSTANTE

  VARIABLE
  ENTERO n;
  ENTERO indice = 2;
  FIN_VARIABLE

  INICIO
  ESCRIBIR ("introduzca un numero");
  LEER (n);
  MIENTRAS (indice < n) AND (n MOD indice <> 0) HACER
    indice = indice + 1;
  FIN_MIENTRAS
  Si (indice = n ) entonces
    ESCRIBIR ("El número es primo");
  SINO
    ESCRIBIR ("El número no es primo");
  FIN_SI
  FIN
  FIN_PROGRAMA
```

Descripción de un pseudolenguaje 2 ASIR

Ejemplo 9

Calcular el valor del número $E = \sum(1/n!)$

PROGRAMA ejercicio9

CONSTANTE

FIN_CONSTANTE

VARIABLE

ENTERO n_termino;

ENTERO indice = 1;

ENTERO fact = 1;

ENTERO cont = 0;

ENTERO suma = 0;

FIN_VARIABLE

INICIO

ESCRIBIR ("introduzca un numero");

LEER (n_termino);

MIENTRAS (cont < n_termino) hacer

fact=1;

indice=1;

MIENTRAS (indice <= cont) hacer

*fact = fact * indice;*

indice = indice + 1;

FIN_MIENTRAS

suma = suma + (1/ fact);

cont=cont+1;

FIN_MIENTRAS

FIN

FIN_PROGRAMA

Descripción de un pseudolenguaje 2 ASIR

Ejemplo 10

Leer una secuencia de 20 números almacenarlos en un vector y mostrar la posición donde se encuentra el mayor valor leído

PROGRAMA ejercicio10

CONSTANTE

CI = 1;

CS = 20;

FIN_CONSTANTE

VARIABLE

ENTERO mayor_posicion = LOW_VALUE;

ENTERO mayor_valor = LOW_VALUE;

ENTERO indice = CI;

ENTERO numeros [CI...CS];

FIN_VARIABLE

.....

Descripción de un pseudolenguaje 2 ASIR

.....

INICIO

```
MIENTRAS (indice <= CS) hacer  
  ESCRIBIR ("introduzca un numero");  
  LEER (numeros [indice]);  
  indice = indice +1;  
FIN_MIENTRAS
```

indice = CI

```
MIENTRAS (indice <= CS) HACER  
  SI (numeros[indice] > mayor_valor) ENTONCES  
    mayor_valor = numeros[indice];  
    mayor_posicion = indice;  
  FIN_SI  
  indice = indice +1;
```

FIN_MIENTRAS

```
  ESCRIBIR ("el mayor se encuentra en la posicion: +mayor_posicion+");
```

FIN

FIN_PROGRAMA

Ejemplo de un Entorno Educativo Algorítmico: Lenguaje SL.



The screenshot shows a web browser window displaying the homepage of the SL programming language. The browser's address bar shows the URL `www.cnc.una.py/sl/SL-index.html`. The page features a navigation menu on the left with links for [Novidades](#), [Documentos](#), [Descarga de SLE](#), and [Licencia](#). The main content area is titled "Bienvenidos a la página de SL" and contains several paragraphs of text. A small image of the SLE development environment is visible on the right side of the page.

SL

Lenguaje SL Centro Nacional de Computación CNC

Bienvenida

[Novidades](#)

[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



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

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
```




E:\MESVIES_1213\0376IAW\UT2\lenguajeSL\ejemplos\mayor_vector_dim.sl

```

1  /*
2  * Lee un vector de n elementos numéricos, y visualizar el mayor.
3  * (c) lopezquesada@iessanjuanbosco.es
4  */
5  var
6  valores : vector [*] numerico
7  indice : numerico
8  mayor : numerico
9  dimension : numerico
10 inicio
11 imprimir ("\nIndimensión del vector:")
12 leer (dimension)
13 dim (valores, dimension)
14 imprimir ("\nIngreso ", alen (valores), " números separados por comas:\n")
15 leer (valores)
16 indice=2
17 mayor=valores[1]
18 mientras (indice<alen(valores)) {
19     si (mayor<valores[indice]){
20         mayor=valores[indice]
21     }
22     indice=indice+1
23 }
24 imprimir ("\nEl mayor de este vector:\n", valores)
25 imprimir ("[" , mayor ,"]")
26 fin

```

Línea: 2 Columna: 22

Ventana de ejecución

Datos previos:

```

Indimensión del vector:5
Ingreso 5 números separados por comas:
2,3,-5,45,1
El mayor de este vector:
2,3,-5,45,1[45]

```

Visor de Expresiones



Descripción de un pseudolenguaje 2 ASIR

Ejemplo 11

Dado una secuencia de números leídos y almacenados en un vector A mostrar dichos números en orden.

```
PROGRAMA ejercicio11
  CONSTANTE
    CI = 1
    CS = 50
  FIN_CONSTANTE

  VARIABLE
    ENTERO vector[CI...CS];
    ENTERO aux ;
    ENTERO intercambio = 0 ;
    ENTERO indice = CI;
  FIN_VARIABLE
```

.....

Descripción de un pseudolenguaje 2 ASIR

....

INICIO

HACER

intercambio = 1;

MIENTRAS (indice <=(CS-1)) HACER

SI (vector [indice] > vector [indice + 1]) ENTONCES

intercambio = 0;

aux = vector [indice]

vector [indice] = vector [indice + 1];

FIN_SI

indice = indice + 1;

FIN_MIENTRAS

MIENTRAS(intercambio = 1);

indice = 1;

MIENTRAS (indice <=CS) HACER

ESCRIBIR(vector[indice]);

indice = indice + 1;

FIN_MIENTRAS

FIN

FIN_PROGRAMA

Descripción de un pseudolenguaje 2 ASIR

Ejemplo 12

Dado dos matrices A y B obtener la suma

PROGRAMA ejercicio12

CONSTANTE

CI1 = 1

CI2 = 1

CS1 = 10

CS2 = 10

FIN_CONSTANTE

VARIABLE

ENTERO matrizA [CI1..CS1][CI2..CS2];

ENTERO matrizB [CI1..CS2][CI2..CS2];

ENTERO matrizC [CI1..CS1,CI2..CS2];

ENTERO indiceA;

ENTERO indiceB;

FIN_VARIABLE

.....

Descripción de un pseudolenguaje 2 ASIR

INICIO

```
indiceA=CI1;
MIENTRAS (indiceA <=CS1) hacer //Leer la matriz A
    indiceB = CI2;
    MIENTRAS (indiceB <= CS2)
        ESCRIBIR ("introduzca un número para el A");
        LEER (matrizA [indiceA, indiceB]);
        indiceB = indiceB +1;
    FIN_MIENTRAS
    indiceA = indiceA +1;
FIN_MIENTRAS
```

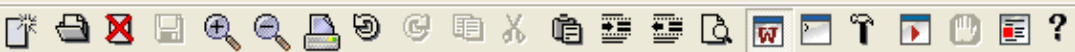
```
indiceA = CI1;
MIENTRAS (indiceA < =CS1) hacer // Leer la matriz B
    indiceB = CI2;
    MIENTRAS (indiceB < =CS2) hacer
        ESCRIBIR ("introduzca un número para el B");
        LEER (matrizB [indiceA, indiceB]);
        indiceB = indicB +1;
    FIN_MIENTRAS
    indiceA = indiceA +1;
FIN_MIENTRAS
```

Descripción de un pseudolenguaje 2 ASIR

```
indiceA=CI1;
MIENTRAS (indiceA <= CS1) hacer           // Suma de las matrices A y B
    indiceB= CI2;
    MIENTRAS (indiceB <= CS2) hacer
        matriz C [indiceA,indiceB] = matriz A [indiceA,indiceB] + matriz B [indiceA,indiceB]
        indiceB= indiceB + 1;
    FIN_MIENTRAS
    indiceA= indiceA + 1;
FIN_MIENTRAS
```

```
indiceA = CI1;
MIENTRAS (indiceA < CS1) hacer           // mostrar la matriz C
    indiceB= CI2;
    MIENTRAS (indiceB < CS2) hacer
        ESCRIBIR (matriz C[indiceA,indiceB])
        indiceB= indiceB + 1;
    FIN_MIENTRAS
    indiceA= indiceA + 1;
FIN_MIENTRAS
```

```
FIN
FIN_PROGRAMA
```



```

1 /*
2  * Sumar dos matrices
3  * (c) lopezquesada@iessanjuanbosco.es
4  */
5 var
6   dimension, fila, col : numerico
7   A,B,S : matriz [*,*] numerico
8 inicio
9   imprimir ("\nIndimensión de las matrices cuadradas:")
10  leer (dimension)
11  dim (A, dimension,dimension)
12  dim (B, dimension,dimension)
13  dim (S, dimension,dimension)
14
15  // Lectura de una matriz A
16  fila=1;
17  mientras (fila<=dimension) {
18    col=1;
19    mientras (col<=dimension) {
20      imprimir("a[" ,fila,".",col,"]:")
21      leer(A[fila,col])
22      col=col+1
23    }
24    fila=fila+1
25  }
26  // Lectura de una matriz B
27  fila=1;
28  mientras (fila<=dimension) {
29    col=1;
30    mientras (col<=dimension) {
31      imprimir("b[" ,fila,".",col,"]:")
32      leer(B[fila,col])
33      col=col+1
34    }
35    fila=fila+1

```

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

```

Descripción de un pseudolenguaje 2 ASIR

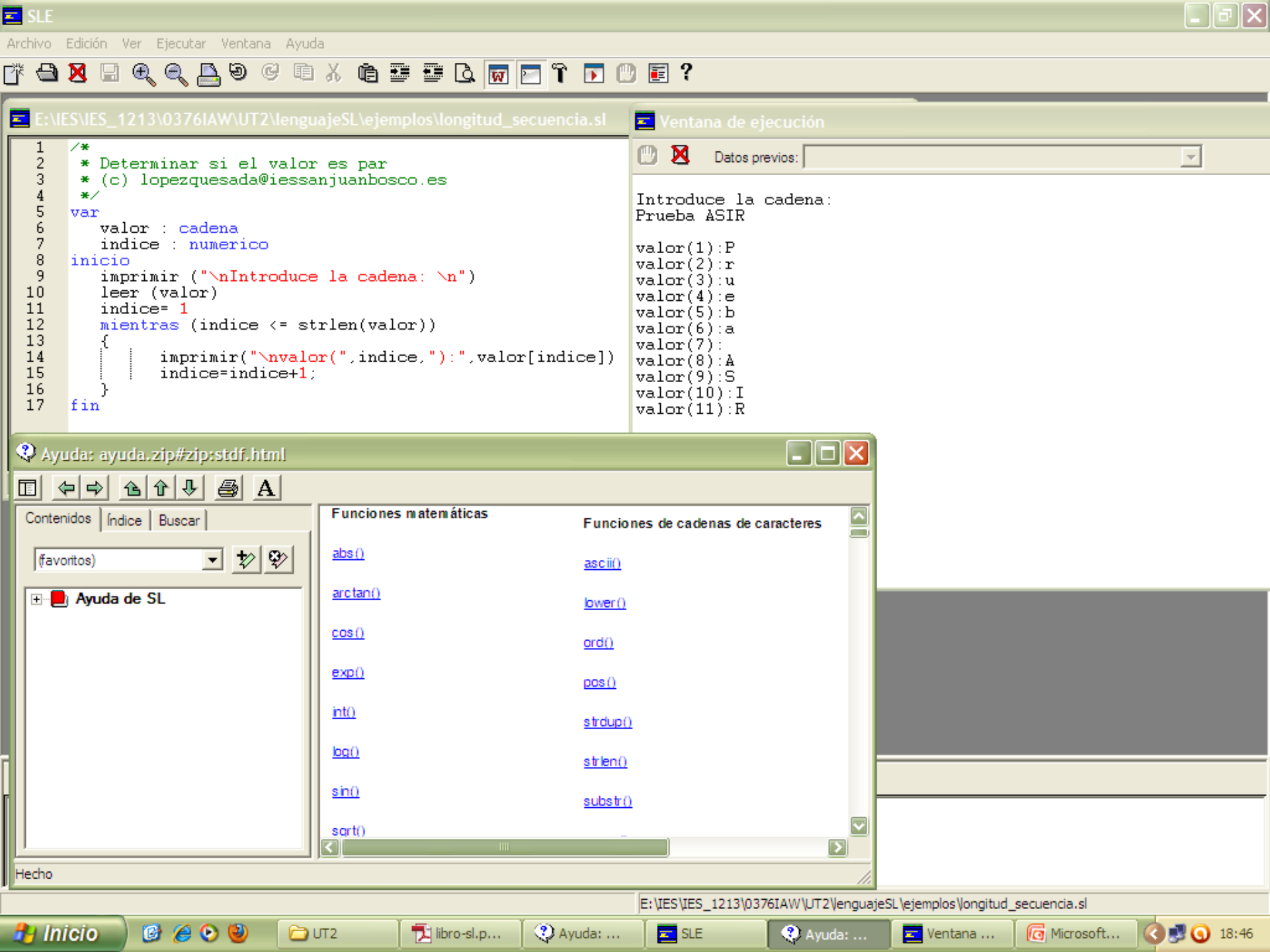
Ejemplo 13

Leer una secuencia y visualizar por pantalla su longitud

```
PROGRAMA ejercicio13
  CONSTANTE
  FIN_CONSTANTE

  VARIABLE
    SECUENCIA nombre;
    ENTERO indice ;
  FIN_VARIABLE

  INICIO
    LEER(nombre);
    indice = 0;
    MIENTRAS (nombre (indice) <> '\0')
      indice = indice+ 1;
    FIN_MIENTRAS
    ESCRIBIR ("La longitud de la secuencia: "+nombre +"es "+indice);
  FIN
FIN_PROGRAMA
```

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. Las construcciones 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 sis@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.

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.

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 de 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 secuencia 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*.





Perl



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

C++

Se obtiene: 2051

javascript



python™

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

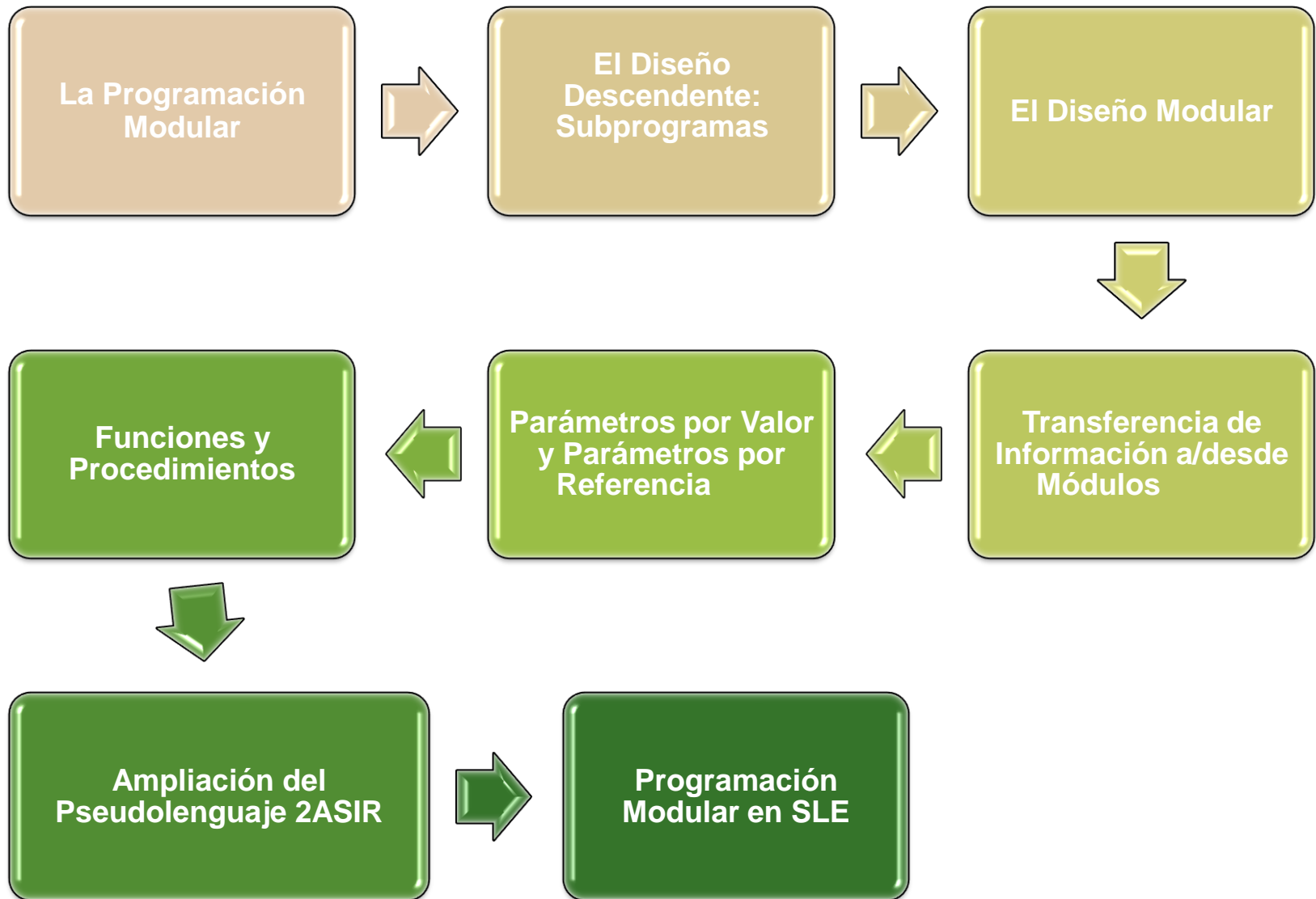


Ruby
A Programmer's Best Friend

```
es son
nero, I
```



Técnicas de Programación. Programación Modular.

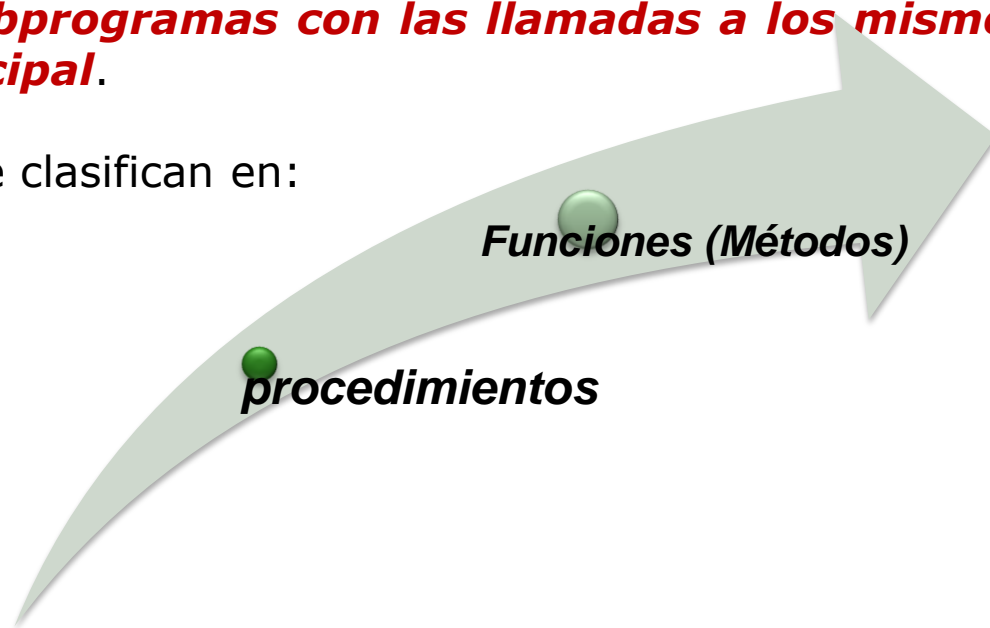


La Programación Modular

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 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.

El Diseño Modular

Los subproblemas o módulos se diseñan con subprogramas, que a su vez se clasifican en procedimientos y funciones.

Los procedimientos y las funciones son unidades de programas (subprogramas/módulos) diseñadas para ejecutar una tarea específica. Por ejemplo, los módulos predefinidos, `printf()` y `scanf()` de C ANSI, están diseñados para realizar operaciones de entrada y salida.

El proceso de descomposición de un problema en módulos se conoce como programación modular o descomposición modular.

Los procedimientos y funciones son similares, aunque presentan notables diferencias entre ellos:

1. Los identificadores de las funciones pueden participar en una expresión, dependiendo del tipo de valor que devuelve.
2. Las funciones no cambian el estado del sistema de información.
3. Los identificadores asociados a un procedimiento no pueden participar en una expresión dado que no pueden devolver un tipo de dato.
4. Tanto las funciones como los procedimientos tiene cero, uno o n parámetros formales.
5. Los parámetros formales pueden ser por Valor o por Referencia.

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).

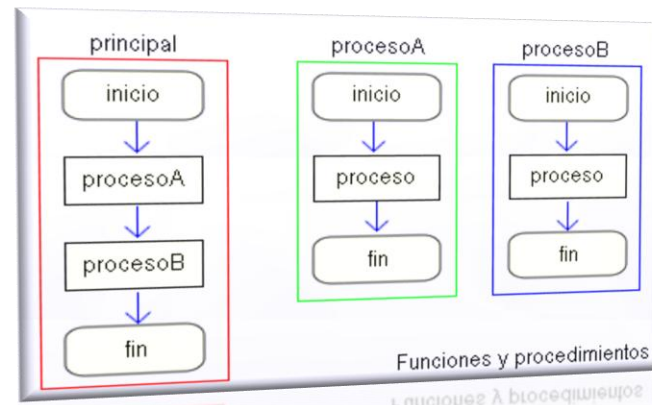
Módulos con parámetros (existe comunicación entre el programa principal y los módulos, y entre ellos).

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));

IMPORTAR "path_librería_[.J]"; (camino donde se encuentra nuestro fichero).

PROGRAMA nombre_programa

CONSTANTE // Constantes globales

FIN_CONSTANTE

VARIABLE // Variables globales

FIN_VARIABLE

[Tipo] FUNCION ~~o función~~ (VAL[REF] ~~épo_~~ parámetro nombre_parametro[,...])

CONSTANTE // Constantes locales

FIN_CONSTANTE

VARIABLE // Variables locales

FIN_VARIABLE

Inicio

..... // bloque de código de la rutina

DEVOLVER (expresión)

Fin

Fin_función

.....
Tipo FUNCION nombre_función_i

FIN_FUNCIÓN

PROCEDIMIENTO ~~o procedimiento~~ (VAL[REF] ~~épo_~~ parámetro nombre_

parametro[,...])

CONSTANTE // Constantes locales

FIN_CONSTANTE

VARIABLE // Variables locales

FIN_VARIABLE

INICIO

// bloque de código de la rutina

FIN

FIN_PROCEDIMIENTO

INICIO ~~o main()~~ módulo principal del programa/pseudocódigo

FIN

FIN_PROGRAMA

Ampliación del pseudolenguaje 2ASIR

Ampliación del pseudolenguaje 2ASIR

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

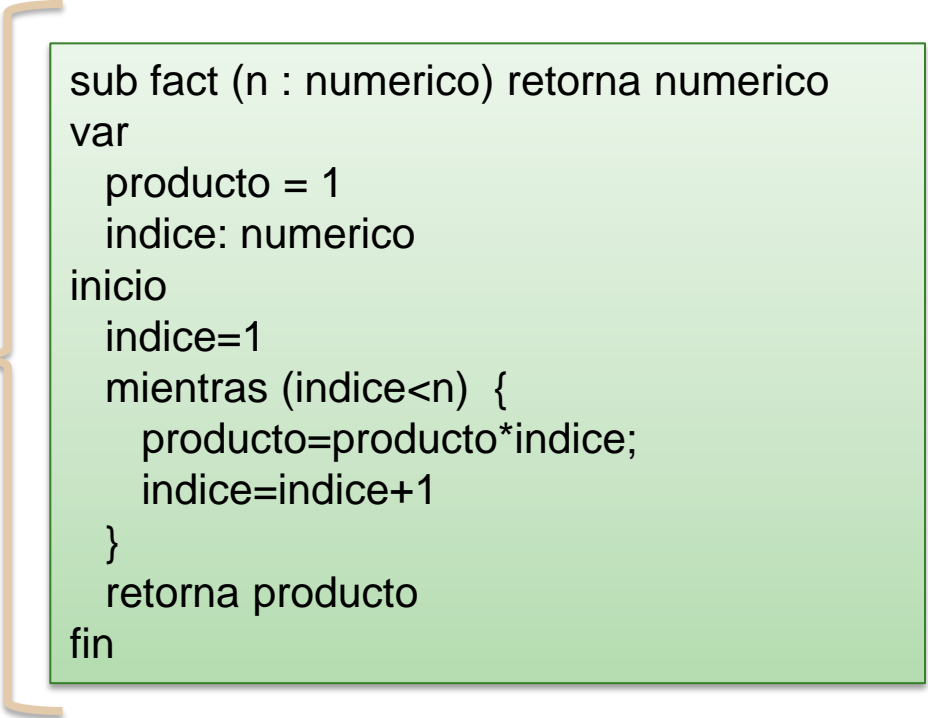
```
PROGRAMA Ejercicio
    CONSTANTES // Constantes globales
        C_I = 1;
        C_S= 30;
    FIN_CONSTANTES
    VARIABLE
        ENTERO numero;
        ENTERO indice;
        ENTERO suma=0;
        ENTERO multiplicar=1;
        ENTERO resul;
    FIN_VARIABLE // Variables globales

    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
```

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
```



```

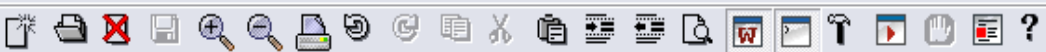
1  /*
2  * Este es el clásico ejemplo del cálculo del factorial
3  * de un entero positivo n.
4  *
5  * (c) lopezquesada@iessanjuanbosco.es
6  */
7  var
8  n : numerico
9  inicio
10 imprimir ("\nCALCULO DE FACTORIAL",
11           "\n-----",
12           "\nIngrese un numero (0-n):")
13 leer (n)
14 si ( 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 ("nCALCULO DE FACTORIAL",
11           "-----n-----",
12           "\nIngrese un numero (0-n):")
13 leer (n)
14 si ( n >= 0 && n == int (n) ) {
15     imprimir ("\n\nn", 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:

CALCULO DE FACTORIAL

Ingrese un numero (0-n):5

5!=24

CALCULO DE FACTORIAL

Ingrese un numero (0-n):-8

No definido para -8

Linea: 1 Columna: 0

