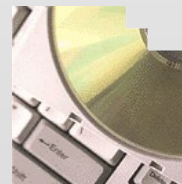


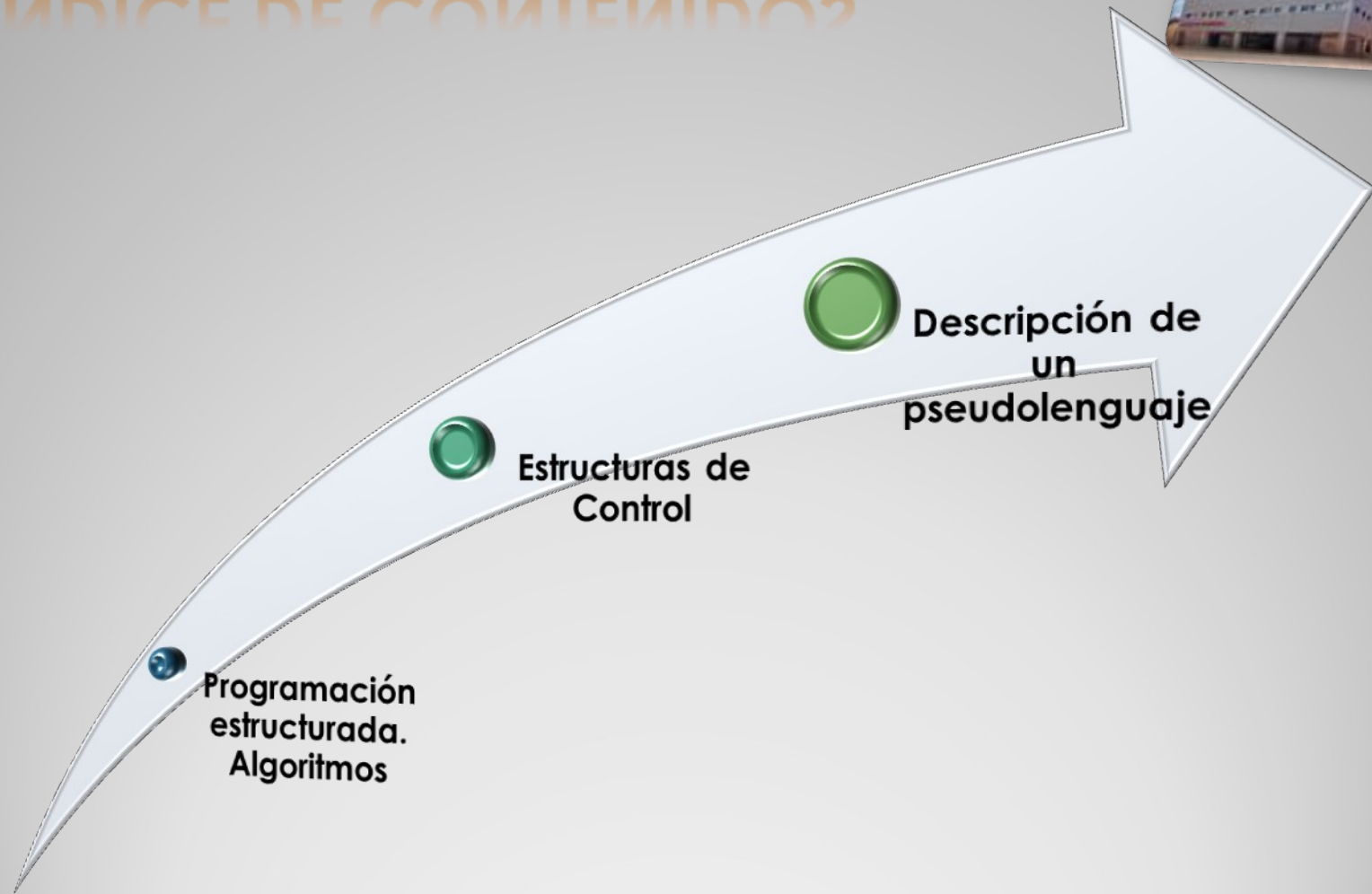
Ciclo Formativo de Grado Superior de Administración de Sistemas Informáticos

Módulo Profesional: *Fundamentos de Programación*
Unidad de Trabajo 5.- *Desarrollo de Algoritmos.*
Programación Secuencial

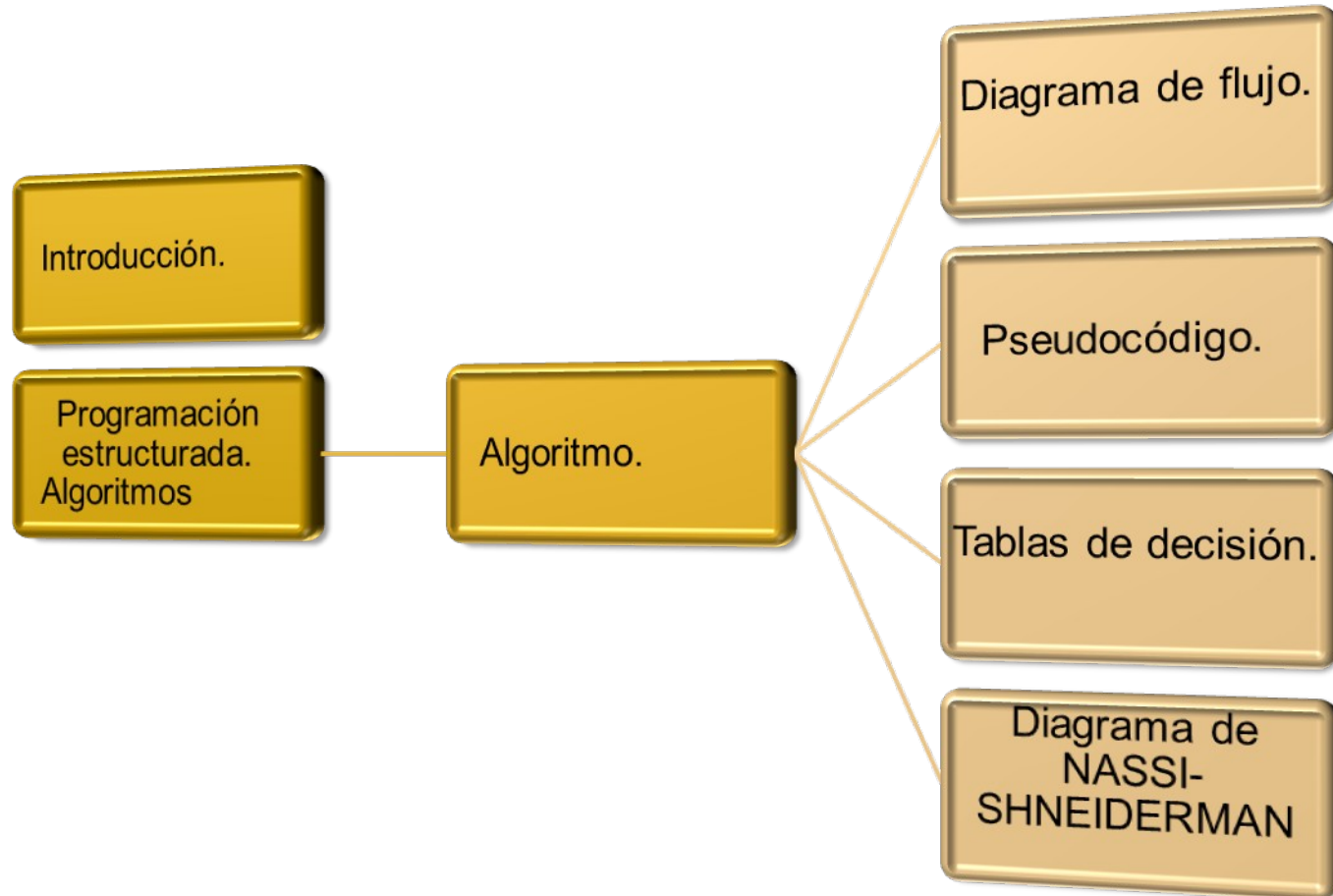
Departamento de Informática y Comunicación
IES San Juan Bosco (Lorca-Murcia)
Profesor: Juan Antonio López Quesada



INDICE DE CONTENIDOS



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:



Un algoritmo puede ser considerado como una caja negra encargada de procesar unos datos de entrada y generar unos datos de salida.

Introducción

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^0 veces sobre los mismos datos de entrada deberá obtenerse n^0 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.



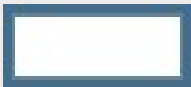




Programación estructurada. Algoritmos

Algoritmo. Diagrama de flujo.

Es una técnica de presentación gráfica para la construcción de algoritmos basada en el uso de símbolos estándar conectados o unidos mediante líneas de flujo que muestran la secuencia lógica de las operaciones que debe de realizar un ordenador. Los diagramas de flujo se pueden clasificar en dos grandes grupos, organigramas y ordinogramas.

Los símbolos básicos que podemos utilizar en un organigrama son los que se muestra en la siguiente figura:

Programación estructurada. Algoritmos

Nombre	Símbolo	Función
Terminal		Representa el inicio y fin de un programa. También puede representar una parada o interrupción programada que sea necesaria realizar en un programa.
Entrada / salida		Cualquier tipo de introducción de datos en la memoria desde los periféricos o registro de información procesada en un periférico.
Proceso		Cualquier tipo de operación que pueda originar cambio de valor, formato o posición de la información almacenada en memoria, operaciones aritméticas, de transformaciones, etc.
Decisión		Indica operaciones lógicas o de comparación entre datos (normalmente dos) y en función del resultado de la misma determina (normalmente si y no) cual de los distintos caminos alternativos del programa se debe seguir
Conector Misma Página		Sirve para enlazar dos partes cualesquiera de un diagrama a través de un conector en la salida y otro conector en la entrada. Se refiere a la conexión en la misma pagina del diagrama
Indicador de dirección o línea de flujo		Indica el sentido de la ejecución de las operaciones
Salida		Se utiliza en ocasiones en lugar del símbolo de salida. El dibujo representa un pedazo de hoja. Es usado para mostrar datos o resultados.

Programación estructurada. Algoritmos

Algoritmo. Diagrama de flujo.

Reglas de los diagramas de flujo

- ✚ Debe de indicar claramente dónde **inicia** y dónde **termina** el diagrama.
- ✚ Cualquier camino del diagrama debe de llevarte siempre al terminal de fin.
- ✚ Organizar los símbolos de tal forma que siga visualmente el flujo de arriba hacia abajo y de izquierda a derecha.
- ✚ No usar lenguaje de programación dentro de los símbolos.
- ✚ Centrar el diagrama en la página.
- ✚ Las líneas deben ser verticales u horizontales, nunca diagonales.
- ✚ No cruzar las líneas de flujo empleando los conectores adecuados sin hacer uso excesivo de ellos.
- ✚ No fraccionar el diagrama con el uso excesivo de conectores.
- ✚ Solo debe llegar una sola línea de flujo a un símbolo. Pero pueden llegar muchas líneas de flujo a otras líneas.
- ✚ y/o izquierda y salir de él por la parte inferior y/o derecha.
- ✚ Evitar que el diagrama sobrepase una página; de no ser posible, enumerar y emplear los conectores correspondientes.
- ✚ Usar lógica positiva, es decir, realizar procesos cuando es verdadera la condición y expresar las condiciones de manera clara (por ej., "no es a \neq de b" \implies "a=b").
- ✚ Comentar al margen únicamente cuando sea necesario.

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.

Programación estructurada. Algoritmos

Algoritmo. Tabla de Decisión.

Muchos procesos de toma de decisiones se pueden solucionar por medio de las tablas de decisión, en las que se representan los elementos característicos de estos problemas:

Los diferentes estados que puede presentar la naturaleza: e_1, e_2, \dots, e_n .

Las acciones o alternativas entre las que seleccionará el decisor: a_1, a_2, \dots, a_m .

Las consecuencias y resultados x_{ij} de la elección de la alternativa a_i cuando la naturaleza presenta el estado e_j .

El formato general de una tabla de decisión es el siguiente:

Forma general de una tabla de decisión					
		Estados de la Naturaleza			
Alternativas		e_1	e_2	\dots	e_n
	a_1	x_{11}	x_{12}	\dots	x_{1n}
	a_2	x_{21}	x_{22}	\dots	x_{2n}
	\dots	\dots	\dots	\dots	\dots
	a_m	x_{m1}	x_{m2}	\dots	x_{mn}

Programación estructurada. Algoritmos

Algoritmo. Diagrama de NASSI-SHNEIDERMAN.

Los diagramas Nassi-Shneiderman, desarrollados por Ike Nassi y Ben Shneiderman, fueron desarrollados para representar la estructura lógica de un flujo de decisiones, como puede ser la ejecución de un programa informático.

Los siguientes son los símbolos más usados en los **D-NS** y sus significados:

1. Indica el inicio del algoritmo.
2. Indica el fin del algoritmo.
3. Indica que se leerán 3 valores y se identificarán como **A, B y C**
4. Indica que se escribirá el valor de **R**
5. Indica un proceso, se evalúa la expresión y se asigna a **R**
6. Indica una **condición** o **pregunta**, si la condición es **verdadera** el flujo se sigue por la rama izquierda, en caso contrario, el flujo seguirá por la rama de la derecha.

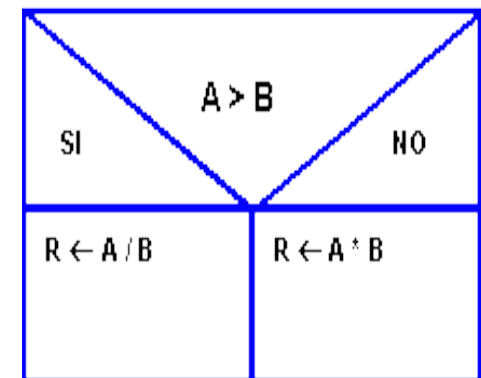
inicio

fin

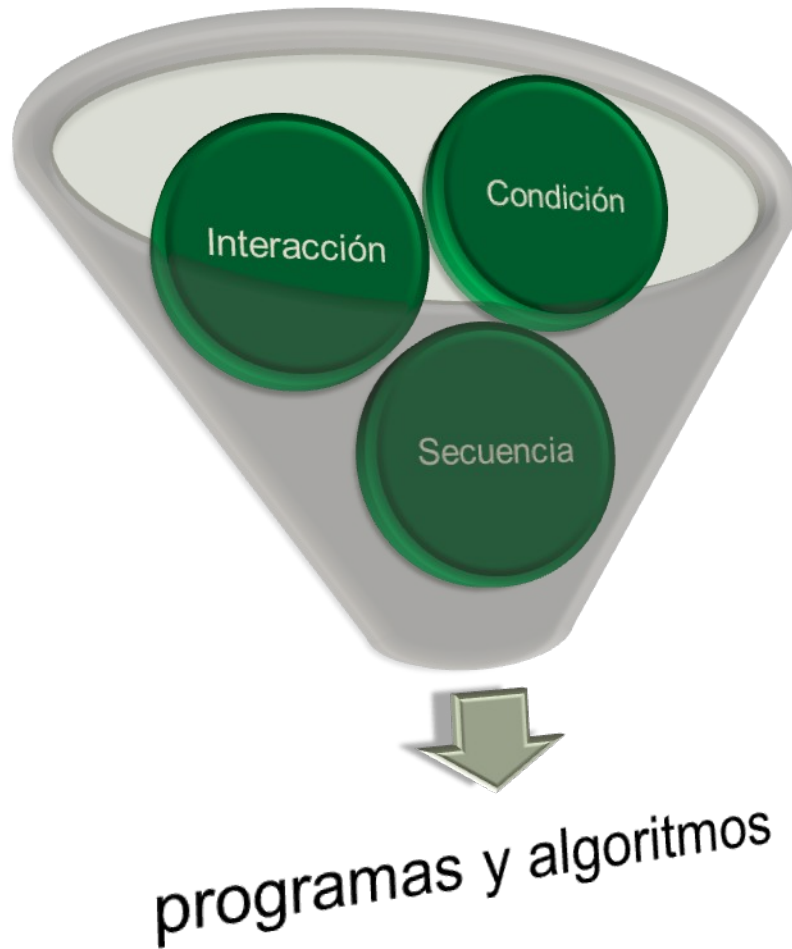
leer A, B, C

escribir R

$R \leftarrow A * B - C$



Estructuras de Control



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. Condición.
2. Interacción.
3. Secuencia.

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

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

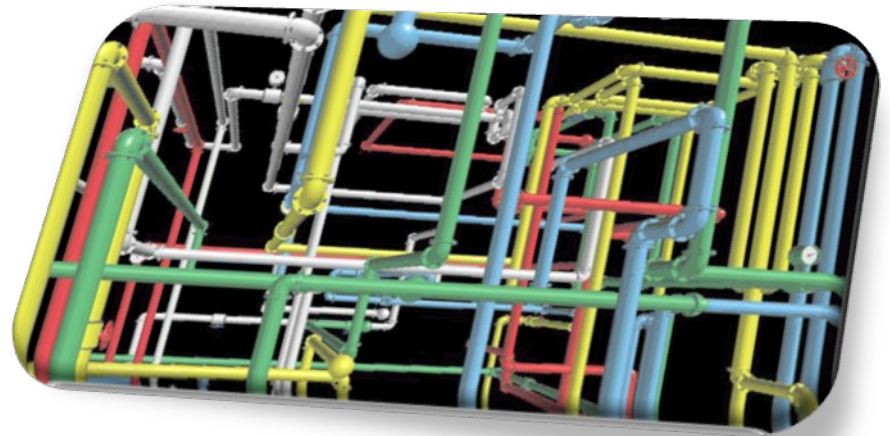
REPETIR-HASTA

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

Do

(Bloque de sentencias)

Until (Condición)



Descripción de un pseudolenguaje. Un Ejemplo.

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. Un Ejemplo.

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. Un Ejemplo.

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. Un Ejemplo.

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. Un Ejemplo.

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

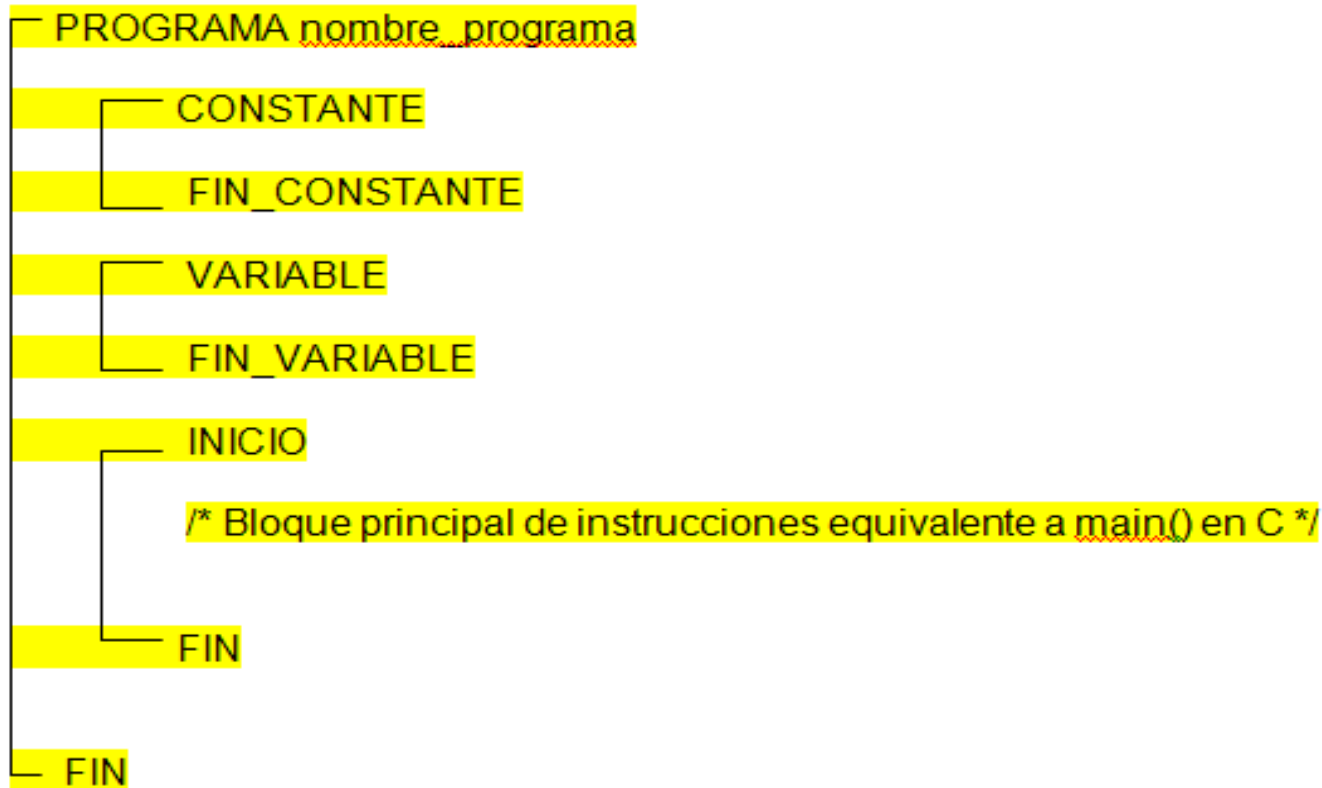
Descripción de un pseudolenguaje. Un Ejemplo.

Prioridad entre los operadores

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

Descripción de un pseudolenguaje. Un Ejemplo.

Estructura del pseudoprograma secuencial



Descripción de un pseudolenguaje. Un Ejemplo.

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. Un Ejemplo.

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. Un Ejemplo.

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)`
1. La expresión que aparece entre paréntesis se evaluara 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. Un Ejemplo.

Tipos de datos:

Ejemplo de definición del bloque de variable:

VARIABLE

CARACTER

ENTERO

SECUENCIA

SECUENCIA

fin= 'f';

indice=0;

nombre= "desconocido";

direccion;

FIN_VARIABLE

Descripción de un pseudolenguaje. Un Ejemplo.

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 asignará 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. Un Ejemplo.

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 secuencia o una concatenación de variables y literales secuencias.

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



Descripción de un pseudolenguaje. Un Ejemplo.

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. Un Ejemplo.

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. Un Ejemplo.

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. Un Ejemplo.

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. Un Ejemplo.

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. Un Ejemplo.

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. Un Ejemplo.

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. Un Ejemplo.

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. Un Ejemplo.

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. Un Ejemplo.

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. Un Ejemplo.

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.

Matriz[ci1,ci2]			Matriz[cs1,ci2]
Matriz[ci1,cs2]			Matriz[cs1,cs2]



Descripción de un pseudolenguaje. Un Ejemplo.

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];
ENTERO	matriz_dinamica[][];

FIN_VARIABLE

Descripción de un pseudolenguaje. Un Ejemplo.

Ejemplo 1

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

```
PROGRAMA Ejercicio
  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. Un Ejemplo.

Ejemplo 2

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

PROGRAMA Ejercicio

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. Un Ejemplo.

Ejemplo 2

INICIO

multiplicar = 1;

indice=C_I;

MIENTRAS (indice<=C_S)

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

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. Leer una secuencia de números y mostrar su producto, el proceso finalizará cuando el usuario pulse a la tecla F.
11. Lee una secuencia de números y determina cual es el mayor de ellos.
12. Dado un número mostrar su valor en binario.
13. 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.
14. Calcular la media de una secuencia de números, el proceso finalizará cuando
15. 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.
16. Leer una secuencia de números y sumar solo los pares mostrando el resultado del proceso.
17. Leer una secuencia de números y mostrar la suma de los 30 números que ocupan posiciones de lectura par.

Cuestiones y Ejercicios

18. Leer un número y determinar su factorial.
19. Leer un número y determinar si es o no es primo.
20. Leer una secuencia de 30 números y mostrar la suma de los primos.
21. Leer una secuencia de 30 números y mostrar la suma de su factorial.
22. Leer una secuencia de números y mostrar la suma de los pares y el producto de los que son múltiplo de 5.
23. Leer una secuencia de números y determinar el mayor de los pares leídos.
24. Leer una secuencia de números y mostrar el mayor de los múltiplos de 5 leídos y el menor de los múltiplos de 3 leídos.
25. Leer una secuencia de 20 números almacenarlos en un vector y mostrar la posición donde se encuentra el mayor valor leído.
26. Dado dos vectores A y B de 15 elementos cada uno, obtener un vector C donde la posición i se almacene la suma de $A[i] + B[i]$.
27. Dado dos vectores A y B de 15 elementos cada uno, obtener un vector C donde la posición i se almacene la suma de $A[i] + B[i]$ y mostrar el mayor de los $C[i]$.
28. Dado una secuencia de número leídos y almacenados en un vector A mostrar dichos números en orden.
29. Dado una secuencia de número leídos y almacenados en un vector A y un número leído determinar si dicho número se encuentra o no en el vector.
30. Leer una secuencia de 20 números y almacenar en un vector sus factoriales.
31. Leer 20 números y almacenarlos de manera ordenada en un vector.
32. Dado dos matrices A y B obtener la suma.
33. Dado una matriz determinar la posición (i,j) del mayor.
34. Dado una matriz determinar la posición (i,j) del mayor y menor.

Cuestiones y Ejercicios

35. Leer un número y una letra si la letra es B mostrar el valor en binario, si es O en octal y si es H en hexadecimal.
36. 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.
37. Dada una matriz $A[1..4][1..5]$ realiza la ordenación de la misma.
38. Dada una matriz $A[1..4][1..5]$ realiza el proceso de ordenar solo por filas.
39. Dado un vector de números determina aquellos que sea primos.
40. Realiza un algoritmo que lea un conjunto de secuencias y determine dada una leída si se encuentra en ese conjunto.
41. Dado un párrafo leído por el teclado determine cuantas palabras contiene.
42. Dado un párrafo leído por el teclado determine la palabra de mayor tamaño.
43. Dado una secuencia determina si es palíndromo.

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

