

UNIVERSIDAD NACIONAL DE ASUNCIÓN

FACULTAD DE CIENCIAS QUÍMICAS

INGENIERÍA DE ALIMENTOS – INGENIERÍA QUÍMICA

MONOGRAFÍA

UNA APROXIMACIÓN A LOS ALGORITMOS

JOSÉ ROJAS DÁVALOS

SAN LORENZO, PARAGUAY

2015

DEDICATORIA

A mi esposa Gricelda Margarita, por su constante amor, apoyo, paciencia, por el hecho de creer en mi y de alentarme a embarcarme en empresas casi imposibles para mi, pero que al final se concretan.

A mis hijas Margarita María y Clara Margarita, por darme ese soporte emocional tan necesario con sus muestras de amor y afecto.

A mi nieta Ana Giselle, por existir e iluminar mis días con su candidez, espontaneidad y ocurrencias.

A mi madre Rosa Nelly(+) por haberme demostrado con su accionar que todo es posible en esta vida.

A mi padre José Carlos Quinto, por orientarme en la vida con su extraordinaria visión de futuro y por estar siempre cuando le necesité y le necesito.

A mis maestros de la vida, Sindulfo Rolón (+) y José Magno Soler, por haber aprendido con ellos sin que se dieran cuenta que me estaban enseñando las asignaturas no académicas, pero imprescindibles para la realización del ser humano.

RECONOCIMIENTOS Y AGRADECIMIENTOS

A Dios, por hacerme tan afortunado y feliz.

A los estudiantes, docentes y directivos de las facultades Politécnica y de Ciencias Químicas de la Universidad Nacional de Asunción, por su constante apoyo y colaboración para mi desarrollo profesional como docente.

A mi esposa, hijas, padres y hermanas por su afecto y cariño, y darme el soporte emocional que sustenta mi desarrollo personal.

A mis maestros académicos y de la vida, por compartir conmigo sus saberes y experiencias.

Al Banco Central del Paraguay y sus funcionarios, por darme tantas oportunidades de aprendizaje académico y de la vida.

La vida es el periodo de aprendizaje que transcurre entre el nacimiento y la muerte.

RESUMEN

Este trabajo pretende proporcionar nociones básicas para construir algoritmos mediante la herramienta PSeInt, explicando los conceptos mediante ejercicios resueltos.

El contenido se divide en (8) ocho capítulos partiendo de una introducción a conceptos de programación y concluyendo con el manejo de arreglos.

Como se mencionó, se presentan ejercicios resueltos mediante pseudo-código que apoyan la explicación del marco conceptual y se plantean además ejercicios propuestos para que el lector consolide su aprendizaje.

PALABRAS CLAVE: algoritmos, PseInt, programación, pseudo-código.

ÍNDICE

Índice de contenido

DEDICATORIA.....	ii
RECONOCIMIENTOS Y AGRADECIMIENTOS.....	iv
RESUMEN.....	v
ÍNDICE.....	vi
CAPITULO 1: INTRODUCCIÓN A LA PROGRAMACIÓN.....	1
Algorítmica o Algoritmia.....	1
Programas.....	1
Concepto de lenguaje de programación.....	2
Clasificación de los Lenguajes de Programación.....	2
Lenguaje de máquina.....	2
Lenguaje de Bajo Nivel (Ensamblador).....	3
Lenguaje de Alto Nivel.....	3
Pasos para construir un algoritmo.....	4
Estructura general de un algoritmo en PSeInt.....	11
Ejercicios propuestos.....	12
CAPÍTULO 2: ENTRADA Y SALIDA DE DATOS.....	14
Introducción.....	14
Lectura.....	14
Impresión.....	16
Limpiar pantalla.....	18
Esperar tecla.....	19
Esperar.....	19
Ejercicios propuestos.....	21
CAPITULO 3: TIPOS DE DATOS. VARIABLES Y CONSTANTES. OPERADORES Y EXPRESIONES.....	23
Introducción.....	23
Tipos de datos.....	23
Tipo numérico.....	23
Tipo cadena.....	24
Tipo lógico.....	25
Variable.....	27
Constantes.....	29
Constantes literales.....	29
Constantes con nombre.....	30
Operadores y expresiones.....	31
Operadores.....	31
Concatenación de cadenas.....	32
Operadores relacionales y cadenas.....	34

Precedencia de los operadores.....	35
Las expresiones complejas y el uso de parentesis.....	36
Funciones.....	37
Ejercicios propuestos.....	39
CAPITULO 4: ESTRUCTURAS DE DECISIÓN.....	42
Introducción.....	42
Sentencias condicionales.....	42
La sentencia si en su forma más simple.....	43
La sentencia si en su forma completa.....	44
La sentencia según.....	45
Si anidado comparado con según.....	47
El operador lógico &(y) o la conjunción.....	49
El operador lógico (o) o la disyunción.....	50
El operador lógico ~(no) o la negación.....	51
Tablas de verdad.....	51
La conjunción.....	52
La disyunción.....	52
La negación.....	53
La negación de una conjunción.....	53
La negación de una disyunción.....	54
Ejercicios propuestos.....	55
CAPITULO 5: ESTRUCTURAS REPETITIVAS.....	58
Introducción.....	58
Mientras, ¿qué cosa?.....	58
Repetir, ¿hasta cuando?.....	59
¿Desde cuando y hasta cuando?.....	60
Más detalles del ciclo para.....	61
Contadores, ¿para que se usan?.....	62
Sumadores. ¿en qué son útiles?.....	64
¿Cómo finaliza un ciclo?.....	65
El promedio, una herramienta estadística.....	66
Buscando el mayor.....	67
Encontrando el menor.....	70
El rango.....	72
Los ciclos anidados.....	74
El indicador o bandera.....	75
Ejercicios propuestos.....	77
CAPÍTULO 6: OPERACIONES CON CADENAS.....	81
Introducción.....	81
Calculando la longitud.....	81
Recorriendo una cadena.....	82
Convirtiendo a mayúsculas o minúsculas.....	84
Extrayendo una subcadena.....	86

Convirtiendo numérico a cadena.....	87
Convirtiendo cadena a numérico.....	87
Ejercicios propuestos.....	88
CAPITULO 7: SUBROUTINAS.....	90
Introducción.....	90
El algoritmo principal.....	90
La subrutina.....	92
Parámetros.....	94
Ámbito de los identificadores.....	96
Paso de parámetros por valor o referencia.....	98
Paso de parámetros por valor.....	100
Paso de parámetros por referencia.....	103
Tipos de funciones.....	105
De acuerdo al valor que retornan.....	105
Tipo numérico.....	105
Tipo cadena.....	105
Tipo lógico.....	105
Las funciones que retornan o no valor.....	106
Funciones que retornan valor.....	106
Funciones que no retornan valor.....	106
Ejercicios propuestos.....	107
CAPÍTULO 8: ARREGLOS.....	109
Introducción.....	109
Vectores.....	109
Dimensión de un vector.....	110
Dimensión de un vector estático o de dimensión fija.....	110
Dimensión de un vector dinámico o de dimensión variable.....	111
Lectura de un vector.....	111
Estático.....	111
Dinámico.....	112
Proceso de un vector.....	112
Impresión de un vector.....	113
Estático.....	113
Dinámico.....	113
La búsqueda del mayor en un vector.....	113
Matrices.....	114
Dimensión de una matriz.....	116
Dimensión de una matriz estática o de dimensión fija.....	116
Dimensión de una matriz dinámica o de dimensión variable.....	116
Lectura de una matriz.....	117
Estática.....	117
Dinámica.....	117
Proceso de una matriz.....	118

Impresión de una matriz.....	119
Estática.....	119
Dinámica.....	119
La matriz cuadrada.....	120
La diagonal principal.....	121
Elementos situados en la diagonal principal.....	121
Elementos situados por encima de la diagonal principal.....	121
Elementos situados por debajo de la diagonal principal.....	121
La diagonal secundaria.....	124
Ejercicios propuestos.....	126
Bibliografía.....	129
Licencia o derechos de uso.....	130

UNA APROXIMACIÓN A LOS ALGORITMOS

CAPITULO 1: INTRODUCCIÓN A LA PROGRAMACIÓN

Algorítmica o Algoritmia

Un algoritmo (del latín, *dixit algorithmus* y éste a su vez del matemático persa Al Juarims), en el contexto de la Matemática, las Ciencias de la Computación y disciplinas afines, es una lista bien definida, ordenada y finita de operaciones que tiene por objetivo hallar la solución a un problema. Los algoritmos son objeto de estudio de la **Algoritmia** o **Algorítmica**.

En el diario vivir se emplean algoritmos en variadas situaciones para resolver otros tantos problemas. Se consideran algoritmos los manuales de usuario que ayudan a usar un electrodoméstico, o inclusive las instrucciones que recibe un vendedor de parte de su supervisor. Por su supuesto, el ámbito natural y primigenio de uso del algoritmo es el entorno matemático al realizar, por ejemplo, un cálculo o al resolver una ecuación.

Programas

Un programa lo podemos definir como una secuencia de instrucciones, escrita en un determinado lenguaje, para la realización de operaciones por parte de la computadora para llegar a un objetivo previamente trazado. Es el algoritmo traducido a un lenguaje de programación. En el desarrollo de este trabajo

UNA APROXIMACIÓN A LOS ALGORITMOS

estaremos creando software en el momento de escribir los programas en respuesta a los ejercicios resueltos y propuestos. Para ello utilizaremos el lenguaje de pseudo-código PSeInt.

Concepto de lenguaje de programación

Un lenguaje de programación es un conjunto de símbolos, caracteres y reglas que le permiten a las personas comunicarse con la computadora. Los lenguajes de programación tienen un conjunto de instrucciones que nos permiten realizar operaciones de entrada/salida, calculo, manipulación de textos, lógica, comparación y almacenamiento y/o recuperación de información.

Clasificación de los Lenguajes de Programación

Los lenguajes de programación se clasifican en:

Lenguaje de máquina

Son aquellos cuyas instrucciones son directamente entendibles por la computadora ya que sus instrucciones son cadenas binarias y no necesitan traducción posterior para que la CPU pueda comprender y ejecutar el programa. Las instrucciones en lenguaje maquina se expresan en términos de la unidad de memoria más pequeña el bit (dígito binario: 0 o 1).

UNA APROXIMACIÓN A LOS ALGORITMOS

Lenguaje de Bajo Nivel (Ensamblador)

En este lenguaje las instrucciones se escriben en códigos alfabéticos conocidos como nemotécnicos para las operaciones y direcciones simbólicas.

Un programa escrito en lenguaje de Bajo Nivel no puede ser ejecutado directamente por la computadora, sino que requiere una fase de traducción al lenguaje Maquina. El programa original escrito en lenguaje de Bajo Nivel se denomina programa fuente y el programa traducido al lenguaje Maquina se conoce como programa objeto. El traductor de programas fuente a objeto es un programa llamado **Assembler**.

Los lenguajes de bajo nivel permiten crear programas muy rápidos, pero que son a menudo difíciles de aprender. Más importante es el hecho de que los programas escritos en un bajo nivel sean altamente específicos de cada procesador. Si se lleva el programa a otra máquina se debe volver a escribir el programa desde el principio.

Lenguaje de Alto Nivel

Los lenguajes de programación de alto nivel son aquellos en los que las instrucciones o sentencias a la computadora son escritas con palabras similares a los lenguajes humanos, lo que facilita la escritura y comprensión del programa.

UNA APROXIMACIÓN A LOS ALGORITMOS

Sin embargo, no son entendibles directamente por el computador y tienen que ser traducidos por programas traductores que pueden ser los **compiladores** o los **intérpretes**.

Pasos para construir un algoritmo

El hecho de dar “clic” para ejecutar un programa en un entorno gráfico o escribir el nombre y dar “enter” en un entorno de consola o de texto, es extremadamente cómodo comparado con todo lo que tuvo que “sufrir” el programador para llegar a tan feliz resultado: qué el programa funcione!!

Este “sufrimiento” se da en menor o mayor medida dependiendo del grado de acatamiento que haya dado el programador a unos pasos que ayudan a ordenar la concepción de un nuevo programa, empezando por el algoritmo. Sobre estos pasos hay diferentes versiones de acuerdo al autor que se consulte. De esas versiones trataremos de hacer un “punto medio”, agregando nuestras propias observaciones.

Creemos que la mejor manera de empezar a explicar estos pasos es con un ejercicio de comprensión, cuyo enunciado transcribimos a continuación:

Solicite el ingreso de dos números enteros, identificados con a y b, calcule y muestre el resultado de la suma. Ejemplo: Si se ingresa 6 para a y 7 para b se debe mostrar como resultado $6 + 7 = 13$.

UNA APROXIMACIÓN A LOS ALGORITMOS

El **primer paso**, consiste en leer el ejercicio las veces que se necesite para entenderlo claramente. A esto algunos autores y docentes llaman “*Leer el enunciado del ejercicio N veces*”.

Suponiendo se cumplió el primer paso, pasemos al **segundo paso**. Esto sería plantear el ejercicio en un lenguaje coloquial antes de pasarlo a uno más formal, como sería el pseudocódigo, diagrama de flujo u otra representación más estructurada. A esto, nos atrevemos a agregar una herramienta que consideramos fundamental para el correcto planteamiento del ejercicio, la prueba de escritorio o planteamiento con datos. Esta prueba tradicionalmente se usa para evaluar un algoritmo ya resuelto. La experiencia nuestra en el proceso de enseñanza aprendizaje de materias similares nos ha demostrado que ayuda en gran medida a la comprensión del problema a ser resuelto y por lo tanto a su mejor planteamiento. Vayamos a los hechos:

- Para sumar dos números se necesita conocer sus valores. Esto implica que tenemos dos datos de entrada, por lo tanto necesitamos dos variables numéricas enteras para la entrada de datos.
- El proceso a ser aplicado es la suma de dos números enteros que están almacenados en dos variables enteras. El resultado de la suma se debe guardar en una tercera variable.

UNA APROXIMACIÓN A LOS ALGORITMOS

- El resultado o la salida es mostrar el valor de la suma calculada.

Si **a** es la variable que contendrá el primer sumando, **b** el segundo sumando y **s** la suma, una prueba con datos podría verse así:

Obtener a	Obtener b	Realizar s=a+b	Mostrar s
6	7	13	13

Con el problema prácticamente resuelto encaramos el **tercer paso**, que sería escribir el algoritmo, es decir, para nosotros, la solución es:

proceso c3_141a

leer a

leer b

s=a+b

imprimir s

fin proceso

Hemos visto que si trabajamos adecuadamente en el planteamiento el hecho de escribir el algoritmo en un lenguaje más formal, resulta mucho más fácil. Pero aún no podemos estar seguros de que el algoritmo funcione así que vamos al **cuarto paso**, que es la prueba de escritorio:

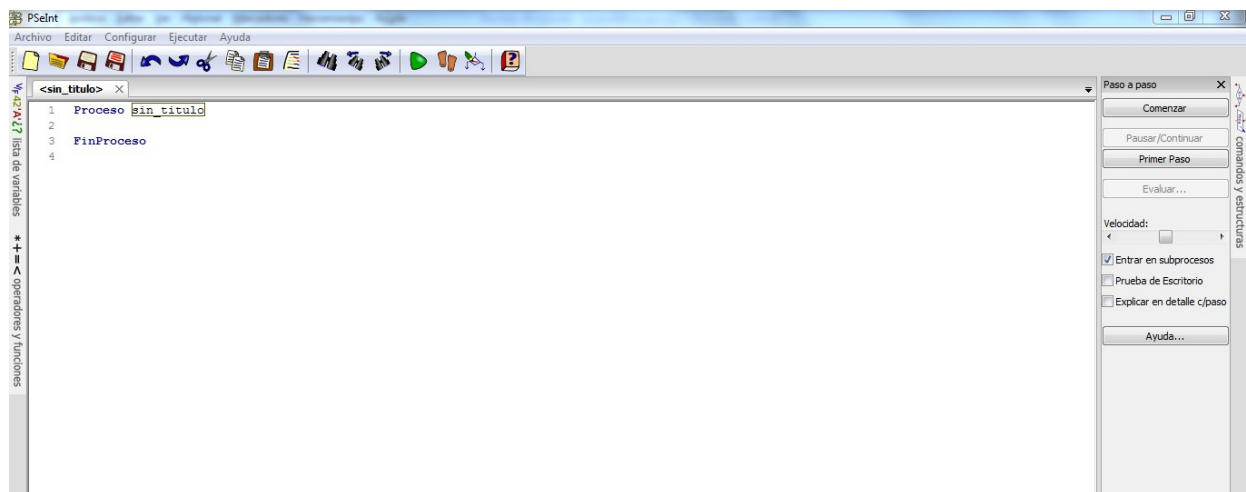
leer(a)	leer(b)	s=a+b	Imprimir(s)
4	6	10	10

UNA APROXIMACIÓN A LOS ALGORITMOS

Una vez escrito el algoritmo, se supone en papel, estamos listos para cargarlo al computador mediante un editor de textos o un Entorno Integrado de Desarrollo (IDE por sus siglas en Ingles) y guardarlo. En resumen, consideramos la carga y guardado del algoritmo como el **quinto paso**.

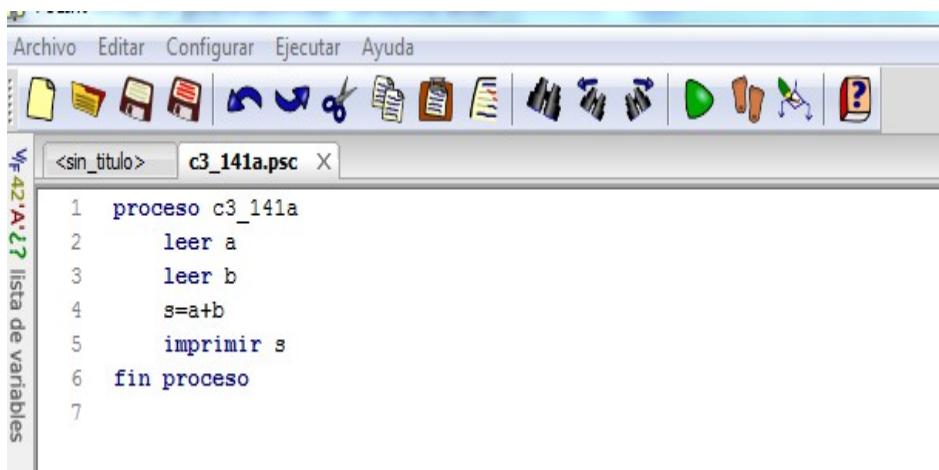
A continuación vemos el proceso de carga y guardado de un algoritmo desde el IDE del PSeInt:

- a) Se ejecuta el programa PseInt.
- b) Se muestra la siguiente pantalla inicial:



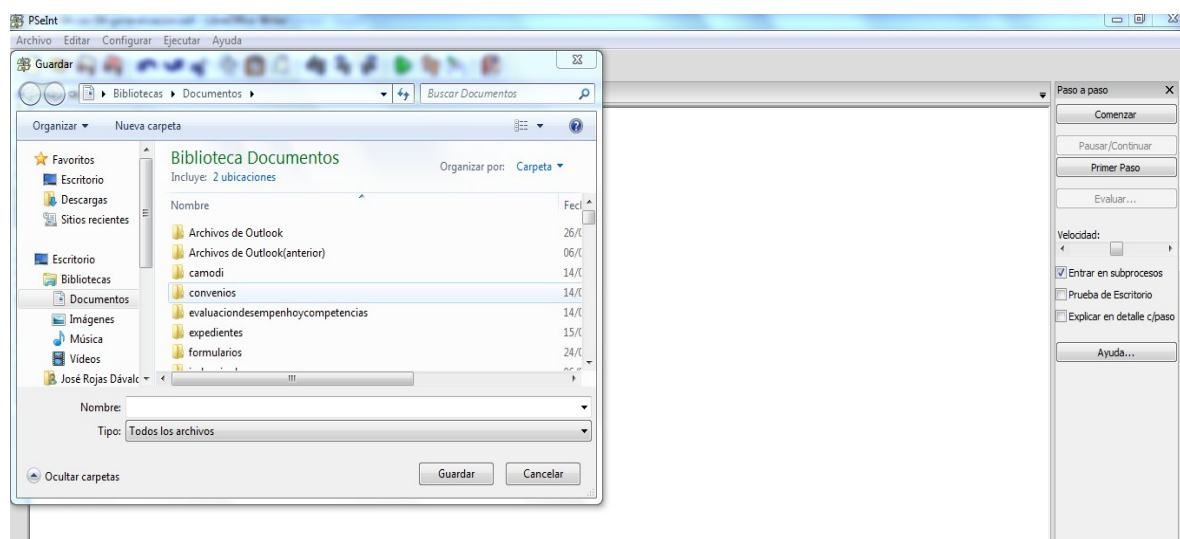
- c) Se carga el algoritmo escribiendo desde la línea identificada con el número 1:

UNA APROXIMACIÓN A LOS ALGORITMOS



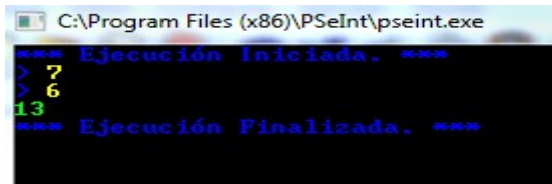
d) Se graba y se le da un nombre seleccionando entre las opciones del IDE

Archivo/Guardar:



El **sexto paso** consiste en la ejecución del algoritmo con datos de prueba. Para ello vamos a la opción Ejecutar/Ejecutar del IDE o presionamos la tecla <F9>:

UNA APROXIMACIÓN A LOS ALGORITMOS



```

C:\Program Files (x86)\PSeInt\pseint.exe
*** Ejecución Iniciada. ***
> 7
> 6
13
*** Ejecución Finalizada. ***
    
```

En un ambiente real se pasaría al **séptimo paso** que consiste en la explotación o uso del algoritmo hasta que uno o varios usuarios finales se den cuenta de las limitaciones o fallas del programa que obligan a realizar las correcciones que correspondan dando lugar al **octavo paso** que sería el de mantenimiento, que podría obligar a volver a cualquiera de los pasos anteriores, **incluso al primer paso**.

En nuestro caso, el algoritmo al ejecutar no emite ningún mensaje de pedido de datos que ayuden al usuario a entender lo que solicita, pues solo aparece el cursor parpadeante. Tampoco al mostrar el resultado despliega un mensaje asociado al hecho. También faltaría asegurarse de limpiar la pantalla antes de la ejecución. Por consiguiente la nueva versión del algoritmo debería tener el siguiente aspecto:

proceso c3_141b

limpiar pantalla

imprimir sin saltar "Ingrese el valor para el primer sumando"

leer a

imprimir sin saltar "Ingrese el valor para el segundo sumando"

UNA APROXIMACIÓN A LOS ALGORITMOS

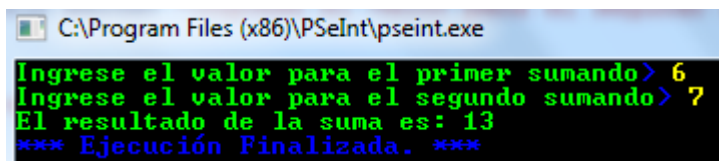
leer b

$s=a+b$

imprimir "El resultado de la suma es: ",s

fin proceso

Para realizar esta modificación se debe volver a la carga y guardado del algoritmo que es el **quinto paso**. Posteriormente se vuelve a hacer la ejecución con datos de prueba conocido como el **sexto paso**:



```
C:\Program Files (x86)\PSeInt\pseint.exe
Ingrese el valor para el primer sumando > 6
Ingrese el valor para el segundo sumando > 7
El resultado de la suma es: 13
*** Ejecución Finalizada. ***
```

Resumiendo, los pasos para la construcción de un algoritmo son:

1. Leer N veces la definición del problema.
2. Plantear el problema con datos de prueba (Prueba de escritorio).
3. Escribir el algoritmo.
4. Hacer la prueba de escritorio con datos de prueba.
5. Cargar y guardar mediante editores de texto o IDEs.
6. Ejecutar con datos de prueba.
7. Usar o explotar el algoritmo
8. Mantener el algoritmo corrigiendo errores o realizando ajustes,

UNA APROXIMACIÓN A LOS ALGORITMOS

pudiendo esta acción hacer que vuelva a cualquiera de los pasos anteriores.

Estructura general de un algoritmo en PSeInt

Lo siguiente muestra esquemáticamente las partes de un algoritmo en PSeInt:

proceso nombre_del_algoritmo

...

sentencias

...

fin proceso

subproceso variable_de_retorno <- nombre_de_la_funcion (argumento_1, argumento_2, ...)

acción 1;

acción 1; .

.

acción n;

finproceso

A continuación se muestra el algoritmo que calcula la suma como ejemplo de aspecto general de un algoritmo en PseInt:

proceso c3_141c

//

UNA APROXIMACIÓN A LOS ALGORITMOS

```
//      Objetivo   :      Sumar dos numeros
//      Autor      :      José Rojas Dávalos
//      Fecha      :      26/08/2013

Limpiar pantalla // Limpia pantalla

imprimir sin saltar "Ingrese valor del primer sumando" // Mensaje

leer a // Pide dato por teclado

imprimir sin saltar "Ingrese valor del segundo sumando"

leer b

s=a+b // Realiza la suma

imprimir a," + ",b," = ",s // Muestra el resultado

fin proceso
```

Ejercicios propuestos

Escriba el algoritmo para:

1. Gestionar la cédula de identidad policial en la Republica del Paraguay, suponiendo que el interesado debe trasladarse desde su domicilio particular hasta el local mas cercano del Dirección de Identificaciones de la Policia Nacional.
2. Solicitar el ingreso de dos números enteros, identificados con x e y, calcule y muestre el resultado de la resta. Ejemplo: Si se ingresa 8 para a y 2 para b se debe mostrar como resultado $8 - 2 = 6$. El operador para la resta es el carácter - (guión).
3. Requerir el ingreso de dos números enteros, identificados con x e y, calcule y muestre el resultado del producto. Ejemplo: Si se ingresa 6 para a

UNA APROXIMACIÓN A LOS ALGORITMOS

y 7 para b se debe mostrar como resultado $6 \times 7 = 42$. El operador para el producto es el carácter * (asterisco).

4. Pedir el ingreso de dos números enteros, identificados con x e y, calcule y muestre el resultado del cociente. Ejemplo: Si se ingresa 4 para a y 2 para b, se debe mostrar como resultado $4 / 2 = 2$. El operador para el cociente es el carácter / (barra).

UNA APROXIMACIÓN A LOS ALGORITMOS

CAPÍTULO 2: ENTRADA Y SALIDA DE DATOS

Introducción

Las instrucciones de entrada y salida, tienen su alto grado de importancia al permitirnos comunicarnos con la computadora, mediante su uso en un algoritmo escrito en PSeInt. En este apartado estaremos revisando como preparar el ingreso de datos a través del teclado, mostrar los resultados y los mensajes de ayuda, limpiar la pantalla y una característica que nos permite hacer que la computadora demore unos segundos o mili-segundos antes de continuar su accionar.

Todo lo expuesto va acompañado de ejemplos de parte de un algoritmo o uno completo.

Disfrútenlo!!!

Lectura

Leer es una instrucción que permite el ingreso de datos desde el teclado y su sintaxis básica es la siguiente;

Leer <variable1> , <variable2> , ... ,<variableN> ;

El formato mostrado permite la asignación de N(unos o más) valores a la misma cantidad de variables, mediante el ingreso de datos por teclado. La variable es creada durante el proceso de lectura si es que no existe aún, y si ya existe pierde

UNA APROXIMACIÓN A LOS ALGORITMOS

su valor anterior, motivo por el cual se suele afirmar que la lectura es "destruktiva", en el sentido de que pierde su valor anterior. Ejemplo:

```
leer nombre // Espera a que se ingrese el dato por teclado
```

La computadora responde mostrando el símbolo de >(mayor) y solicitando el dato:



El separador entre las variables de una lista es configurable desde el cuadro de opciones del PseInt y se puede usar espacio en lugar de la coma, que es la opción por defecto. Así se podría escribir:

```
leer nombre1 nombre2|
```

El ejemplo muestra un algoritmo que lee el nombre de una persona y la saluda:

```
proceso c3_201 // Inicio del algoritmo
//
// Objetivo    : Saludar a una persona cuyo nombre se ingresa por teclado
// Autor       : José Rojas Dávalos
// Fecha       : 08/09/2014
//
limpiar pantalla // Limpia contenido actual de la pantalla cuando se ejecuta
imprimir sin saltar "¿Cuál es tu nombre?" // Solicita dato al usuario
leer nombre // Espera a que se ingrese el dato por teclado
imprimir sin bajar "Presione una tecla para permitirme que le salude..." // Mensaje para el usuario
esperar tecla // Espera que se presione una tecla
imprimir " " // Salto de línea
imprimir " Hola, ",nombre // Imprime el saludo
fin proceso // Fin del algoritmo
```

UNA APROXIMACIÓN A LOS ALGORITMOS

Impresión

Para mostrar datos por pantalla existe la instrucción *Imprimir* cuyo formato general de uso es:

Imprimir <expr1> , <expr2> , ... , <exprN>

Mediante esta instrucción se puede mostrar por pantalla uno o más valores y si hay más de una expresión se mostrarán una a continuación de la otra, por lo que se debe insertar los espacios necesarios para mostrar más de un resultado, si ese es el requerimiento. El ejemplo:

```
mostrar " Hola, ",nombre1," y ",nombre2 // Saluda a ambos nombres
```

El resultado:

Hola, Margarita y Clara

Si se usa *Imprimir* en combinación con las expresiones clave *sin saltar* o *sin bajar*, los valores se despliegan pero no hay avance a la línea siguiente, por lo que la siguiente instrucción de lectura o escritura se realizará en la misma línea. Si no se usa una de las expresiones mencionadas la computadora asume que debe imprimir posicionar el cursor en la siguiente línea. Se muestran dos formatos posibles del uso de *sin saltar*(o lo que es lo mismo, *sin bajar*):

imprimir sin saltar <expr1>,...,<exprN>

imprimir <expr1>,...,<exprN> *sin saltar*

UNA APROXIMACIÓN A LOS ALGORITMOS

Se muestra un ejemplo de pseudocódigo y su ejecución:

```
imprimir sin saltar "¿Cual es el primer nombre?" // Solicita el primer nombre
¿Cual es el primer nombre?> 
```

Vea la posición del cursor (en azul) al lado del texto esperando el dato. No hubo “salto de línea”. Para confirmar esto, observe la posición del nombre una vez ingresado:

```
¿Cual es el primer nombre?> Ana
```

Es indistinto el uso las palabras *Escribir* y *Mostrar* en vez de *Imprimir* si su perfil de lenguaje está configurado como flexible. Se podría, opcionalmente, usar:

```
escribir sin saltar "¿Cual es el segundo nombre?" // Solicita el segundo nombre
```

o

```
mostrar " Hola, ",nombre1," y ",nombre2 // Saluda a ambos nombres
```

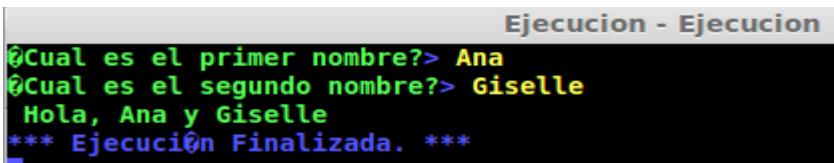
Lo siguiente muestra un ejemplo completo, tanto para solicitar datos al usuario como para mostrar resultados:

UNA APROXIMACIÓN A LOS ALGORITMOS

```

proceso c3_203 // Inicio del algoritmo
//
//  Objetivo      :  Saludar en una sola línea, a dos personas cuyos nombres
//                  se ingresan por teclado, en dos variables
//  Autor         :  José Rojas Dávalos
//  Fecha        :  08/09/2014
//
    limpiar pantalla // Borra el contenido de la pantalla al inciar la ejecucion
    imprimir sin saltar "¿Cual es el primer nombre?" // Solicita el primer nombre
    leer nombrel // Permite cargar el primer nombre
    escribir sin saltar "¿Cual es el segundo nombre?" // Solicita el segundo nombre
    leer nombre2 // Permite cargar el segundo nombre
    mostrar " Hola, ",nombrel," y ",nombre2 // Saluda a ambos nombres
fin proceso // Fin del algoritmo
    
```

Una muestra de la ejecución del algoritmo:



```

Ejecucion - Ejecucion
¿Cual es el primer nombre?> Ana
¿Cual es el segundo nombre?> Giselle
Hola, Ana y Giselle
*** Ejecución Finalizada. ***
    
```

Limpiar pantalla

Permite limpiar la pantalla del contenido que tiene en el momento de ejecutarse el algoritmo y coloca el cursor en el ángulo superior izquierdo. Normalmente se usa al inicio de un programa, pero esto puede variar de acuerdo a los requerimientos.

La forma general de uso es:

Limpiar pantalla

Opcionalmente se puede usar:

Borrar pantalla

Un ejemplo de su uso al inicio de un algoritmo en combinación con la primera

UNA APROXIMACIÓN A LOS ALGORITMOS

instrucción de salida:

```
limpiar pantalla // Borra el contenido de la pantalla al inciar la ejecucion
imprimir sin saltar "¿Cual es el primer nombre?" // Solicita el primer nombre
```

El efecto que causa:

```
¿Cual es el primer nombre?>
```

En este caso se borró el contenido de la pantalla y se mostró el primer mensaje.

Esperar tecla

Interrumpe la ejecución del algoritmo hasta que el usuario presiona una tecla y entonces continua la ejecución. Su formato general:

Esperar tecla

Un ejemplo de su uso:

```
imprimir sin saltar "¿Cuál es tu nombre?" // Solicita dato al usuario
leer nombre // Espera a que se ingrese el dato por teclado
imprimir sin bajar "Presione una tecla para permitirme que le salude..." // Mensaje para el usuario
esperar tecla // Espera que se presione una tecla
imprimir " " // Salto de línea
imprimir " Hola, ",nombre // Imprime el saludo
```

La secuencia de ejecución es:

```
¿Cuál es tu nombre?> Ana
Presione una tecla para permitirme que le salude...
Hola, Ana
*** Ejecución Finalizada. ***
```

Esperar

Se utiliza para realizar una pausa en el algoritmo durante un intervalo de tiempo

UNA APROXIMACIÓN A LOS ALGORITMOS

indicado con dos parámetros, la duración y la unidad de medida. La duración se debe indicar con una expresión numérica que puede ser una constante o una variable. Las unidades de medida utilizables son *segundo*, *segundos*, *milisegundo* o *milisegundos*. El formato general es:

Esperar <expresión numérica> [segundo|segundos|milisegundo|milisegundos]

Un ejemplo de su uso:

```
esperar 5 segundos // Demora la ejecución durante 5 segundos
```

A continuación un algoritmo de ejemplo:

```
proceso c3_205 // Inicio del algoritmo
//
// // Objetivo      :   Luego de una espera de 5 segundos
//                   mostrar la leyenda "Hola", dibujada con asteriscos
// Autor           :   José Rojas Dávalos
// Fecha          :   14/09/2014
//
limpiar pantalla // Borra pantalla al inicio
esperar 5 segundos // Demora la ejecución durante 5 segundos
imprimir "*" * ***** *          ***** // Muestra línea 1 del mensaje
imprimir "*" * * * * * * *          * * // Muestra línea 2 del mensaje
imprimir "***** * * * * *          *****" // Muestra línea 3 del mensaje
imprimir "*" * * * * * * *          * * // Muestra línea 4 del mensaje
imprimir "*" * ***** ***** * * // Muestra línea 5 del mensaje
fin proceso // Fin del algoritmo
```

El resultado:

UNA APROXIMACIÓN A LOS ALGORITMOS



Ejercicios propuestos

Escriba el algoritmo para

1. Solicitar por teclado los datos de una persona según el siguiente diseño de pantalla y ejemplo de carga de datos:

Nombre.....: Margarita

Título.....: Doctora

Peso.....: 49

Estatura.....: 1.60

Las leyendas se deben mostrar desde la línea 4 en adelante sin espacios entre líneas, todas empezando en la columna 4. La entrada de datos en todas las líneas se debe hacer empezando en la columna 16. Al final imprimir un mensaje de saludo en la línea 9, columna 4 del formato “Hola “,titulo,” ” ,nombre,” de “,peso,” kilos y “,estatura,” centímetros”.

Con nuestros datos de ejemplo el saludo debería mostrar:

“Hola Doctora Margarita de 49 kilos y 160 centimeros”.

2. Mostrar 5 asteriscos y pausar 5 segundos.

UNA APROXIMACIÓN A LOS ALGORITMOS

3. Pedir por teclado las longitudes del frente y el fondo de un terreno y mostrar la superficie, según el siguiente ejemplo de ejecución:

Ingrese la longitud del frente: 12

Ingrese la longitud del fondo: 30

El terreno tiene 360 metros cuadrados.

UNA APROXIMACIÓN A LOS ALGORITMOS

CAPITULO 3: TIPOS DE DATOS. VARIABLES Y CONSTANTES. OPERADORES Y EXPRESIONES

Introducción

Los datos procesados y convertidos en información, son herramientas que ayudan a la toma de decisión, y por ende una de las razones fundamentales de la existencia de las computadoras.

Este capítulo trata, justamente, de los datos, sus tipos, de como se almacenan en la memoria volátil (RAM) y cuales son las formas de procesar las expresiones numéricas, de texto o lógicas, mediante los operadores, respondiendo a necesidades de la vida real.

Tipos de datos

Tipo numérico

El tipo de dato de una variable determina que valores pueden ser almacenados en ella. Por ejemplo si necesitamos almacenar el resultado de un cálculo estaríamos guardando un valor numérico. Por lo tanto podemos decir que la variable es de tipo numérico, como se muestra en el siguiente ejemplo:

proceso c3_301 // Inicio del algoritmo

//

// Objetivo : Calcular la cantidad de producto obtenido a partir de un reactivo

// Autor : Jose Anibal Aguero Guillen - Traduccido al PSeInt por Jose Rojas Davalos

UNA APROXIMACIÓN A LOS ALGORITMOS

```
//      Fecha      :      22/09/2014
//
// ncarbonato: Numero de moles de reactivo
// nbicarbonato; Numero de moles de producto
// borrar pantalla // Limpia pantalla
// imprimir sin saltar "Ingrese el numero de moles de Carbonato de Sodio Anhidro" // Computadora pide dato
// leer ncarbonato // Usuario ingresa dato
// nbicarbonato = 2*ncarbonato // Calcula el número de moles de producto
// imprimir "A partir de ",ncarbonato," moles de carbonato de sodio, se obtienen ",nbicarbonato," moles de bicarbonato de sodio"
// Imprime el resultado
fin proceso // Fin del algoritmo
```

Se puede almacenar en las variables numéricas tanto enteros como reales. Es decir puede representar expresiones numéricas con o sin decimales.

Todos los valores y variables tienen asociados un tipo de datos y el traductor verifica que las operaciones realizadas sean consistentes. Si escribimos una línea de código fuente como la siguiente:

```
nbicarbonato = 2*"ncarbonato" // Calcula el número de moles de producto
```

el traductor señalará la línea como errónea, pues si bien. Inicialmente, nbicarbonato y ncarbonato son variables numéricas “ncarbonato” al ser delimitada por comillas ya no es una variable numérica, sino una constante tipo cadena.

A propósito de cadena, es el siguiente tipo de dato que vamos a estudiar.

Tipo cadena

En el algoritmo:

```
proceso c3_307 // Inicio del algoritmo
//
```


UNA APROXIMACIÓN A LOS ALGORITMOS

```
//      Objetivo      : Imprimir el nombre de una sal usando concatenación de cadenas
//      Autor          : Jose Anebal Aguero Guillen - Traducido al PSeInt por Jose Rojas Davalos
//      Fecha          : 22/09/2014
//
// cation: Nombre del cation
// sal: Nombre de la sal

borrar pantalla // Limpia pantalla

imprimir sin saltar "Nombre del cation" // Computadora solicita cation

leer cation // Usuario ingresa nombre de cation

imprimir sin saltar "Nombre del anion: " // Computadora solicita anion

leer anion

sal = anion + " de " + cation // Computadora construye nombre de sal juntando(concatenando) anion y cation

imprimir "La sal formada es el ", sal // Computadora muestra nombre de sal formada

fin proceso // Fin del algoritmo
```

tenemos la siguiente línea:

```
sal = anion + " de " + cation // Computadora construye nombre de sal juntando(concatenando) anion y cation
```

donde la variable sal puede realmente contener, letras mayúsculas, letras minúsculas, los símbolos o caracteres especiales y los dígitos, más allá de las necesidad de solo almacenar el nombres de una sal.

En el programa ejemplo, la línea:

```
imprimir sin saltar "Nombre del anion: " // Computadora solicita anion
```

también contiene una expresión cadena que es caso es la constante "Nombre del anion:". Mas adelante veremos con más detalle el concepto de constantes.

Tipo lógico

Permite almacenar dos valores posibles, VERDADERO o FALSO. El programa mostrado a continuación es un ejemplo de uso de expresiones lógicas y los valores

UNA APROXIMACIÓN A LOS ALGORITMOS

que retorna:

```
proceso c3_305 // Inicio del algoritmo

//
//      Objetivo      :      Comprobar que un par de puntos pertenece a una recta dada
//      Autor          :      Jose Anibal Aguero Guillen - Traducido al PSeInt por Jose Rojas Davalos
//      Fecha          :      22/09/2014
//
//      sn1 : Si es verdad o falso que x1 e y1 pertenecen a la recta
//      sn2 : Si es verdad o falso que x2 e y2 pertenecen a la recta

limpiar pantalla // Limpia pantalla

imprimir sin saltar "Ingrese coordenada x del primer punto" // Computadora solicita x1
leer x1 // Usuario ingresa valor de x1

imprimir sin saltar "Ingrese coordenada y del primer punto" // Computadora solicita y1
leer y1 // Usuario ingresa valor de y1

imprimir sin saltar "Ingrese coordenada x del segundo punto" // Computadora solicita x2
leer x2 // Usuario ingresa valor de x2

imprimir sin saltar "Ingrese coordenada y del segundo punto" // Computadora solicita y2
leer y2 // Usuario ingresa valor de y2

sn1 = (y1 == (3 - 2*x1)) // Computadora carga verdadero o falso en sn1 si y1 es igual 3 - 2*x1
sn2 = (y2 == (3 - 2*x2)) // Computadora carga verdadero o falso en sn2 si y2 es igual 3 - 2*x2

imprimir "Es ",sn1," que el punto (",x1," ",y1,") ", " pertenece a la recta 2x + y - 3 = 0" // Computadora muestra resultado para
primer punto

imprimir "Es ",sn2," que el punto (",x2," ",y2,") ", " pertenece a la recta 2x + y - 3 = 0" // Computadora muestra resultado
para segundo punto

fin proceso // Fin del algoritmo
```

Las variables *sn1* y *sn2* contendrán VERDADERO o FALSO según las expresiones que se les asigna sean ciertas, en cuyo caso los puntos pertenecen a la recta respectiva . En caso contrario los puntos no pertenecen a la recta. En la impresión se utiliza las variables *sn1* y *sn2* como parte de un mensaje coloquial

UNA APROXIMACIÓN A LOS ALGORITMOS

que muestra el resultado.

Resumiendo:

Tipo de dato básico	Descripción de los valores que puede almacenar
cadena	Cualquier secuencia de caracteres. La cadena vacía se representada con "" y que tiene longitud cero.
logico	Los valores VERDADERO o FALSO. Son 2 identificadores que son constantes predefinidas.
numerico	Valores enteros y reales, con signo o sin signo.

Variable

Una variable en un algoritmo computacional es una posición de memoria donde se puede almacenar información. Por ejemplo, si se debe obtener el módulo de un vector de (3) tres componentes, se debe cargar en memoria el valor de (3) tres variables para poder realizar el cálculo. El resultado también se asigna a una variable luego del cálculo, de tal forma a mostrar el resultado al usuario. Como su nombre lo indica, el valor almacenado en una variable puede ir variando a medida que el algoritmo avanza.

El ejemplo siguiente muestra el uso de 4(cuatro) variables:

proceso c3_303 //Inicio del algoritmo

//

// Objetivo : Calcula el módulo de un vector de tres dimensiones

// Autor : Jose Anibal Aguero Guillen - Traducido al PSeInt y leves modificaciones por Jose Rojas

UNA APROXIMACIÓN A LOS ALGORITMOS

Dávalos

```
//      Fecha      :      22/09/2014

//

// x,y,z: componentes del vector

//      m: modulo del vector

borrar pantalla // limpia pantalla

imprimir sin saltar "Introduzca componente x del del vector " // Computadora solicita componente x
leer x // Usuario digita valor de x

imprimir sin saltar "Introduzca componente y del del vector " // Computadora solicita componente y
leer y // Usuario digita valor de y

imprimir sin saltar "Introduzca componente z del del vector " // Computadora solicita componente z
leer z // Usuario digita valor de z

m = raíz(x^2 + y^2 + z^2) // Computadora aplica formula y calcula m

imprimir "El modulo del vector <"x","y","z"> es: "m // Computadora muestra resultado

fin proceso // Fin dek algoritmo
```

En un algoritmo se hace referencia a una variable mediante un identificador (el nombre de la variable). Un identificador debe comenzar con letras, y puede contener solo letras, números y el guión bajo. No puede contener ni espacios ni operadores, ni coincidir con una palabra reservada o función del lenguaje, para no generar ambigüedad. Ejemplos de identificadores válidos son **x**, **y**, **z** y **m**. En la mayoría de los casos los nombres de variables no pueden contener acentos, ni diéresis, ni eñes.

Las variables tienen un tipo de dato asociado, por lo que durante la ejecución del algoritmo una variable deberá guardar datos siempre del mismo tipo. Por ejemplo, si una variable se utiliza para guardar números, no puede utilizarse luego para guardar texto. Hay dos formas de crear una variable y/o asignarle un valor: la

UNA APROXIMACIÓN A LOS ALGORITMOS

lectura y la asignación. Si se lee o asigna un valor en una variable que no existe, esta se crea. Si la variable ya existía, esta toma el nuevo valor, perdiendo el viejo. Por esto se dice que la asignación y la lectura son acciones destructivas (aunque se debe notar que en la asignación pueden intervenir más de una variable, y solo se destruye el contenido previo de la que se encuentra a la izquierda del signo de asignación). Una vez inicializada, la variable puede utilizarse en cualquier expresión.

Si se intenta asignar a una variable ya definida un dato de un tipo incorrecto se producirá un error en tiempo de ejecución.

Constantes

Las constantes son valores fijos que se utilizan dentro de un programa y que no cambian sus valor durante el proceso. Existen más de un tipo de constante que veremos a continuación.

Constantes literales

Se llaman constantes literales a aquellos números o cadenas que forma parte del texto del programa.

Ejemplos:

imprimir "La sal formada es el", sal // Computadora muestra nombre de sal formada

En este caso “La sal formada es el” es una constante literal tipo cadena.

En la expresión:

UNA APROXIMACIÓN A LOS ALGORITMOS

$m = \text{raiz}(x^2 + y^2 + z^2)$ // Computadora aplica formula y calcula m

El número 2, es una constante literal tipo numérico.

Constantes con nombre

Las constantes con nombres, en PSeInt, se simulan variables a los que asignamos un valor que se espera(pero no es seguro) no cambien durante la ejecución de todo el algoritmo.

El siguiente programa utiliza una constante con nombre:

```
proceso c3_306 // Inicio del algoritmo
//
//      Objetivo      :      Calcular el alcance de un proyectil con movimiento parabólico
//      Autor          :      Jose Anibal Aguero Guillen - Traducido al PSeInt por Jose Rojas Davalos
//      Fecha          :      22/09/2014
//
// Constante

g = 9.8 // aceleración de la gravedad (m/s^2)

//      V0: Velocidad inicial del proyectil (m/s)
//      theta: Angulo de tiro (rad)
//      x: Alcance del proyectil

limpiar pantalla // Borra el contenido de la pantalla

imprimir sin saltar "Introduzca la velocidad inicial del proyectil (m/s)" // Computadora solicita velocidad inicial
leer V0 // Usuario ingresa velocidad inicial

imprimir sin saltar "Introduzca el ángulo de lanzamiento (rad)" // Computadora solicita angulo en radianes
leer theta // Usuario ingresa angulo en radianes

x = (V0^2)*(sen(2*theta))/g // Computadora calcula alcance

imprimir "El alcance del proyectil es de ",x," metros" // Computadora muestra alcance de proyectil
fin proceso // Fin del algoritmo
```

UNA APROXIMACIÓN A LOS ALGORITMOS

Operadores y expresiones

Llamamos expresión a una combinación válida de operadores, constantes, variables y llamadas a funciones. Las expresiones tienen un papel fundamental pues a través de ellas expresamos los cálculos y otras operaciones que deseamos que el programa realice.

El uso de expresiones es muy sencillo porque en su mayoría se componen de operaciones aritméticas de uso cotidiano: sumas, multiplicaciones, potencias, etc.

Algunos ejemplos de expresiones son:

- $x = (V_0^2) * (\sin(2 * \theta)) / g$ // Computadora calcula alcance
- $sn1 = (y1 == (3 - 2 * x1))$ // Computadora carga verdadero o falso en sn1 si y1 es igual $3 - 2 * x1$
- $sal = anion + " de " + cation$ // Computadora construye nombre de sal juntando(concatenando) anion y cation

Operadores

Los operadores son símbolos que usamos para indicar la realización de operaciones con ciertos datos, a los que llamamos operandos. Se dividen en (4) cuatro grupos: relacionales, lógicos, algebraicos y la concatenación de cadenas.

Operador	Significado	Ejemplo
Relacionales		
>	Mayor que	<u>3</u> >2

UNA APROXIMACIÓN A LOS ALGORITMOS

<	Menor que	'ABC'<'abc'
=	Igual que	<u>4</u> =3
<=	Menor o igual que	'a'<='b'
>=	Mayor o igual que	<u>4</u> >=5
<>	Distinto que	'a'<>'b'
Lógicos		
& ó Y	Conjunción (y)	(7>4) & (2=1) //falso
ó O	Disyunción (o)	(1=1 2=1) //verdadero
~ ó NO	Negación (no)	~(2<5) //falso
Algebraicos		
+	Suma	total <- cant1 + cant2
=	Resta	stock <- disp – venta
*	Multiplicación	area <- base * altura
/	División <- 100 * parte / total	
^	Potenciación	sup <- 3.41 * radio ^ 2
% ó MOD	Módulo (resto de la división entera)	resto <- num MOD div

Concatenación de cadenas

Llamamos concatenación a la operación de tomar dos cadenas y generar una tercera que contiene todos los caracteres de la primera cadena seguidos de los de la segunda, y las subsiguientes si hay más. Veamos un ejemplo:

```
proceso c3_307 // Inicio del algoritmo
//
```


UNA APROXIMACIÓN A LOS ALGORITMOS

```
//      Objetivo      : Imprimir el nombre de una sal usando concatenación de cadenas
//      Autor          : Jose Anibal Aguero Guillen - Traducido al PSeInt por Jose Rojas Davalos
//      Fecha          : 22/09/2014
//
// cation: Nombre del cation
// sal: Nombre de la sal

borrar pantalla // Limpia pantalla
imprimir sin saltar "Nombre del cation" // Computadora solicita cation
leer cation // Usuario ingresa nombre de cation
imprimir sin saltar "Nombre del anion: " // Computadora solicita anion
leer anion

sal = anion + " de " + cation // Computadora construye nombre de sal juntando(concatenando) anion y cation
imprimir "La sal formada es el ", sal // Computadora muestra nombre de sal formada
fin proceso // Fin del algoritmo
```

En la línea resaltada se mostrar la unión del contenido de anion, la constante cadena “ de “ y el contenido de cation, como muestra la ejecución del programa:

```
Nombre del cation> sodio
Nombre del anion: > cloruro
La sal formada es el cloruro de sodio
*** Ejecución Finalizada. ***
```

El signo utilizado para concatenar cadenas es el mismo que usamos para sumar números. El PSeInt puede interpretar correctamente lo que deseamos realizar porque inspecciona el tipo de las variables que acompañan al signo ‘+’: Si ambos operandos son numéricos, se trata de una suma; si ambos son cadenas, se requiere una concatenación; de lo contrario hay un error.

Es importante señalar que la concatenación, a diferencia de la suma de números, NO es conmutativa. Es decir, no da lo mismo:

UNA APROXIMACIÓN A LOS ALGORITMOS

sal = anion + " de " + cation

que

sal = cation + " de " + anion.

Mas adelante veremos que existen varias funciones predefinidas y otras operaciones sofisticadas que podemos realizar con cadenas de caracteres.

Operadores relacionales y cadenas

Cuando dos cadenas, sean éstas literales o variables, aparecen como operandos de un operador relacional, cada carácter de la primera cadena es comparado el carácter que se encuentra en la segunda cadena en la misma posición desde la izquierda.

Se considera que "A" es menor que "B" porque la primera letra aparece antes que la segunda en la tabla ASCII. Esta comparación de posiciones en la tabla ASCII determina el funcionamiento de los operadores relacionales aplicados a cadenas.

A continuación algunos ejemplos:

Comparación	Resultado	Explicación
"ABC"<"ACC"	Verdadero	El segundo carácter de la primera cadena es "B", mientras que de la segunda cadena es "C". En la tabla ASCII "B" está antes que "C".
"a" <= "A"	Falso	En la tabla ASCII las letras minúsculas aparecen mucho después que las mayúsculas,

UNA APROXIMACIÓN A LOS ALGORITMOS

		es decir, son “mayores”.
ABA” > “AB”	Verdadero	Los dos primeros caracteres de ambas cadenas coinciden pero la primera cadena tiene un carácter más que la segunda, lo que la convierte en “mayor” que la segunda.
“AbCdE” == “AbCdE”	Verdadero	Los caracteres de ambas cadenas son los mismos, en el mismo orden.
“Z” > “”	Verdadero	Una cadena de longitud uno es mayor que la cadena vacía.
“” == “”	Verdadero	Ambas cadenas tienen longitud 0.

Precedencia de los operadores

Las reglas de precedencia de los operadores determinan en qué orden se evalúan las expresiones. En general, los operadores aritméticos tienen la precedencia usual. La siguiente tabla define las precedencias de los operadores:

Precedencia	Operadores
1 (mayor precedencia)	^ (potenciación)
2	-, + (cambio de signo, identidad de signo)
3	*, /, %
4	+, -
5	==, <, <=, >, >=

UNA APROXIMACIÓN A LOS ALGORITMOS

6	not
7	and
8 (menor precedencia)	or

Los operadores que se encuentran en la misma línea tienen igual precedencia. Si forman parte de una expresión o subexpresión no parentizada, se evaluarán de izquierda a derecha. Siguiendo la tabla anterior, la siguientes sentencias hacen que la variable termine con el valor 31:

a=2

b=4*a*3 // b = 4 * (2 * 3) $\Rightarrow 4 * 6 \Rightarrow 24$

c = a ^ 2 * -b // c = (2 ^ 2) * -24 $\Rightarrow 4 * -24 \Rightarrow -96$

a = b - c % 10 + 1 // a = 24 - (-96 % 10) + 1 $\Rightarrow 24 - (-6) + 1 \Rightarrow 31$

Las expresiones complejas y el uso de parentesis

Podemos utilizar paréntesis con dos propósitos:

Para hacer que las expresiones sean más legibles: La “abundancia” de paréntesis no tiene ningún impacto negativo sobre la ejecución del programa y en cambio puede facilitar su comprensión, no solo por parte de uno mismo sino por otros que deban entender el programa después. Un ejemplo que muestra el uso de paréntesis con este propósito es:

y = v0y * t + 1/2 * -g * t^2 // sin parentización “redundante”

UNA APROXIMACIÓN A LOS ALGORITMOS

$$y = (v_{0y} * t) + ((1/2) * (-g) * (t^2))$$

Para alterar el orden en que se evalúan las expresiones, si la precedencia por defecto no produce el resultado que necesitamos. Veamos dos ejemplos:

a) $3 * 2 + 1$ // es 7, pero

$$3 * (2 + 1) \text{ // es } 9$$

b) -2^2 // es -4, pero

$$(-2)^2 \text{ // es } 4$$

Funciones

Las funciones en el pseudocódigo se utilizan de forma similar a otros lenguajes. Se coloca su nombre seguido de los argumentos para la misma encerrados entre paréntesis (por ejemplo `trunc(x)`). Se pueden utilizar dentro de cualquier expresión, y cuando se evalúe la misma, se reemplazará por el resultado correspondiente. Actualmente, todas las funciones disponibles son matemáticas (es decir que devolverán un resultado de tipo numérico) y reciben un sólo parámetro de tipo numérico. A continuación se listan las funciones integradas disponibles:

Función	Significado
RC(X) o RAIZ(X)	Raíz Cuadrada de X
ABS(X)	Valor Absoluto de X
LN(X)	Logaritmo Natural de X
EXP(X)	Función Exponencial de X
SEN(X)	Seno de X

UNA APROXIMACIÓN A LOS ALGORITMOS

COS(X)	Coseno de X
TAN(X)	Tangente de X
ASEN(X)	Arcoseno de X
ACOS(X)	Arcocoseno de X
ATAN(X)	Arcotangente de X
TRUNC(X)	Parte entera de X
REDON(X)	Entero más cercano a X
AZAR(X)	Entero aleatorio entre 0 y x-1
LONGITUD(S)	Cantidad de caracteres de la cadena S
MAYUSCULAS(S)	Retorna una copia de la cadena S con todos sus caracteres en mayúsculas
MINUSCULAS(S)	Retorna una copia de la cadena S con todos sus caracteres en minúsculas
SUBCADENA(S,X,Y)	Retorna una nueva cadena que consiste en la parte de la cadena S que va desde la posición X hasta la posición Y (incluyendo ambos extremos). Las posiciones utilizan la misma base que los arreglos, por lo que la primer letra será la 0 o la 1 de acuerdo al perfil del lenguaje utilizado.
CONCATENAR(S1,S2)	Retorna una nueva cadena resulta de unir las cadenas S1 y S2.

UNA APROXIMACIÓN A LOS ALGORITMOS

CONVERTIRNUMERO(X)	Recibe una cadena de caracteres que contiene un número y devuelve una variable numérica con el mismo.
CONVERTIRTEXTO(S)	Recibe un real y devuelve una variable numérica con la representación como cadena de caracteres de dicho real.

Notas:

- El módulo (resto de la división entera) no se encuentra entre las funciones ya que fue implementado como operador.
- Las funciones trigonométricas reciben el ángulo en radianes. Para facilitar las conversiones se puede usar la constante PI (Ej: si A es un ángulo en grados, su coseno se obtiene con "cos(A*PI/180)").
- Las funciones que operan sobre cadenas de caracteres solo estarán disponibles si el perfil de lenguaje seleccionado lo permite (ver Opciones del Pseudocódigo).
- Para crear sus propias funciones en pseudocódigo para utilizar desde el procedimiento principal vea la sección SubProcesos.

Ejercicios propuestos

Preparar el algoritmo para:

UNA APROXIMACIÓN A LOS ALGORITMOS

1. Pida el nombre y el año de ingreso de una persona a una empresa además el año actual. A partir de estos datos calcule y muestre la antigüedad de la persona en la empresa. Un ejemplo de la ejecución de este programa podría ser:

Ingrese su nombre: Marga

Ingrese su año de ingreso(aaaa): 2000

Ingrese el año actual(aaaa): 2010

Marga: Ud. tiene 10 años de antigüedad

2. Solicite las edades y nombres de dos personas y muestre un mensaje diciendo si uno de ellos es o no menor que el otro.
3. Analice si un número entero mayor a cero ingresado es divisible o no por x. x también se ingresa por teclado
4. Lea por teclado a y b, ambos numéricos enteros y calcule la raíz cuadrada del producto de ambos.
5. Permita ingresar tres valores cadena y los muestre concatenados.
6. Resuelva la ecuación $y=x^2+2x+6$ para un valor de x.
7. Calcule cuantos metros de cable se necesita para unir dos puntos sabiendo que por cada metro de distancia se debe prever dos metros de cable. El dato que se ingresa es la distancia en metros entre los dos puntos. Se debe

UNA APROXIMACIÓN A LOS ALGORITMOS

mostrar como resultado la cantidad de cable necesaria, expresada en metros.

8. Calcule la superficie de un rectángulo.
9. Verifique si un número es la tercera parte de otro.

UNA APROXIMACIÓN A LOS ALGORITMOS

CAPITULO 4: ESTRUCTURAS DE DECISIÓN

Introducción

Según se menciona en los cursos introductorios la computadora cuenta con una parte llamada unidad aritmética – lógica (ALU, por su sigla en Inglés) , que se encarga de las operaciones de cálculo, por un lado, y de realizar las comparaciones y tomar las decisiones, por otro, de acuerdo a lo que establece el programa que se está ejecutando.

En el capítulo anterior habíamos visto conceptos y ejercicios relacionados a la parte aritmética de la ALU. En esta construiremos algoritmos que harán funcionar el poder de decisión de las computadoras.

Hagamos eso mismo.

Sentencias condicionales

Existen dos sentencias condicionales: sentencia **si** y sentencia **segun**. La primera se ejecutará si cierta condición se cumple, mientras la segunda puede ayudar a mejorar la legibilidad si las condiciones muy largas o complejas. La sintaxis básica de la sentencia si es:

Si expresion_logica **Entonces**

acciones_por_verdadero

UNA APROXIMACIÓN A LOS ALGORITMOS

Sino

acciones_por_falso

Fin Si

La cláusula **sino** se usa cuando el algoritmo necesita realizar acciones si la condición es falsa.

La sentencia si en su forma más simple

Si expresion_logica Entonces

acciones_por_verdadero

Fin Si

El siguiente algoritmo imprime un texto si se cumple una condición dada con un dato numérico ingresado:

```
proceso c3_401 // Inicio del algoritmo
//
//      Objetivo   :      Muestra texto "Conocimiento" si se ingresa valor 1
//      Autor      :      Jose Rojas Davalos
//      Fecha      :      12/10/2014
//
// codigo: de criterio
//
      borrar pantalla // Limpia pantalla
      imprimir sin saltar "Codigo de criterio(1=Conocimiento)" // Computadora pide codigo
      leer codigo // Usuario ingresa codigo
      si codigo == 1 // Si el codigo tiene valor 1
          imprimir "Conocimiento" // Muestra mensaje
```

UNA APROXIMACIÓN A LOS ALGORITMOS

finsi // Cierre del si

finproceso // Fin del algoritmo

En este caso la condición es `codigo == 1`, y no se usa la salida por falso, pues solo interesa cuando se cumple la condición, en cuyo caso se imprime un texto.

La sentencia si en su forma completa

Si *expresion_logica* **Entonces**

acciones_por_verdadero

Sino

acciones_por_falso

Fin Si

El algoritmo:

proceso c3_402 // Inicio del algoritmo

//

// Objetivo : Muestra texto "Conocimiento" si se ingresa valor , "Otro criterio", si se ingresa otro valor

// Autor : Jose Rojas Davalos

// Fecha : 12/10/2014

//

// codigo: de criterio

//

borrar pantalla // Limpia pantalla

imprimir sin saltar "Codigo de criterio(1=Conocimiento,Distinto a 1=Otro criterio)" //

UNA APROXIMACIÓN A LOS ALGORITMOS

Computadora pide...

leer codigo // Usuario ingresa codigo

si codigo == 1 // Si el codigo tiene valor 1

imprimir "Conocimiento" // Muestra mensaje para valor 1

sino

imprimir "Otro criterio" // Muestra mensaje para otro valor

finsi // Cierre del si

finproceso // Fin del algoritmo

La condición es `codigo == 1`, y se utilizan ambas salidas, la de verdadero y la de falso, realizando acciones diferentes en cada caso.

La sentencia según

La sentencia **según** permite analizar varias opciones que dependen de un valor numérico. Su formato es:

Según variable Hacer

opcion_1:

secuencia_de_acciones_1

opcion_2:

secuencia_de_acciones_2

opcion_3:

secuencia_de_acciones_3

De Otro Modo:

UNA APROXIMACIÓN A LOS ALGORITMOS

secuencia_de_acciones_dom

Fin Segun

Un ejemplo de su uso:

```
proceso c3_403 // Inicio del algoritmo
//
//      Objetivo   :      Muestra texto de criterio segun código ingresado
//      Autor      :      Jose Rojas Davalos
//      Fecha      :      12/10/2014
//
// codigo: de criterio
//
      borrar pantalla // Limpia pantalla

      // Computadora pide codigo

      imprimir sin saltar "Codigo de criterio(1=Conocimiento,2=Paciencia y
Confianza,3=Tolerancia,4=Habilidad,5=Estrategia)"

      leer codigo // Usuario ingresa codigo

      segun codigo hacer // Segun sea el valor de codigo

1: // Si es 1
      imprimir "Conocimiento" // Conocimiento

2: // Si es 2
      imprimir "Paciencia y Confianza" // Paciencia y confianza

3: // Si es 3
      imprimir "Tolerancia" // Tolerancia

4: // Si es 4
      imprimir "Habilidad" // Habilidad

5: // Si es 5
      imprimir "Entrategia" // Estrategia
```

UNA APROXIMACIÓN A LOS ALGORITMOS

De Otro Modo: // En otro caso

imprimir "Error: valor fuera de rango!!!" // Mensaje de error

finsegun // Cierre del segun

finproceso // Fin del algoritmo

Si anidado comparado con según

El siguiente algoritmo muestra el uso del llamado **si** anidado, es decir varias preguntas una dentro de otra, En otra parte de la misma solución se muestra que algo que podría resolverse con **si** anidado se implementa con **según**:

proceso c3_404 // Inicio del algoritmo

//

// Objetivo : Muestra nombre de entrevistado, criterio de seleccion y el texto que corresponde a una escala de 1 al 5

// datos resultantes de una entrevista laboral

// Autor : Jose Rojas Davalos

// Fecha : 12/10/2014

//

// nombre: del postulantes

// criterio_cod; codigo del criterio de seleccion

// criterio_txt; descripcion del criterio de seleccion

// puntos_num: puntaje concedido en el criterio por el entrevistador

// puntos_txt: texto que corresónde en la escla al puntaje concedido

borrar pantalla // Limpia pantalla

imprimir sin saltar "Nombre del postulante" // Computadora pide nombre

leer nombre // Usuario ingresa nombre

imprimir sin saltar "Codigo de criterio(1=Conocimiento,2=Paciencia y

Confianza,3=Tolerancia,4=Habilidad,5=Estrategia)"

UNA APROXIMACIÓN A LOS ALGORITMOS

```

leer criterio_cod // Usuario ingresa codigo de criterio

imprimir sin saltar "Puntaje obtenido 1=Insuficiente,2=Aceptable,3=Bueno,4=Muy bueno,5=Excelente" //

Computadora pide criterio

leer puntos_num // Usuario ingresa puntaje como numero

// Computadora determina texto que corresponde al criterio

si criterio_cod == 1 // Si es 1
    criterio_txt="Conocimiento" // Le corresponde Conocimiento
sino
    si criterio_cod == 2 // Si es 2
        criterio_txt="Paciencia y Confianza" // Le corresponde Paciencia y Confianza
    sino
        si criterio_cod == 3 // Si es 3
            criterio_txt="Tolerancia" // Le corresponde Tolerancia
        sino
            si criterio_cod == 4 // Si es 4
                criterio_txt="Habilidad" // Le corresponde Habilidad
            sino
                si criterio_cod == 5
                    criterio_txt="Estrategia" // Le corresponde Estrategia
                sino
                    criterio_txt="Desconocido" // Le corresponde
Desconocido
finsi // Cierre del si
finsi // Cierre del si
finsi // Cierre del si
finsi // Cierre del si
finsi
segun puntos_num // En caso que puntos_num

```


UNA APROXIMACIÓN A LOS ALGORITMOS

```

1: // Sea 1
    puntos_txt="Insuficiente" // Es insuficiente
2: // Sea 2
    puntos_txt="Aceptable" // Es aceptable
3: // Sea 3
    puntos_txt="Bueno" // Es bueno
4: // Sea 4
    puntos_txt="Muy bueno" // Es muy bueno
5: // Sea 5
    puntos_txt="Excelente" // Es excelente
de otro modo: // Sea mayor a 5 o menor 1
    puntos_txt="Desconocido" // Es valor desconocido
finsegun
    imprimir nombre+" tiene un nivel "+puntos_txt+" de "+criterio_txt // Imprime el resultado
finproceso // Fin del algoritmo

```

Como se ve, el uso del **sí** anidado implica más complejidad, por lo que aparentemente se optaría por el **según**, que permite comparaciones con números o cadenas, y es de un formato más legible. La limitación de esta última estructura sería el de no permitir preguntar por valores lógicos.

El operador lógico &(y) o la conjunción

El uso del operador lógico **&(y)** se ve en el ejemplo siguiente:

```

proceso c3_405 // Inicio del algoritmo
//
//      Objetivo   :      Lee nombre y edad. Si edad esta en [20,25] muestra ambos datos

```

UNA APROXIMACIÓN A LOS ALGORITMOS

```
//      Autor      :      Jose Rojas Davalos
//      Fecha      :      12/10/2014
//
// nombre: del postulante
// edad: del postulante

borrar pantalla // Limpia pantalla

imprimir sin saltar "Nombre del postulante" // Computadora pide nombre

leer nombre // Usuario ingresa nombre

imprimir sin saltar "Edad del postulante" // Computadora pide nombre

leer edad // Usuario ingresa nombre

si edad >= 20 & edad <= 25 entonces // Si edad esta en el rango [20,25]

    imprimir nombre, " tiene ", edad, " años" // Imprime el resultado

finSi // Cierre del si

finproceso // Fin del algoritmo
```

El operador lógico |(o) o la disyunción

El programa listado a continuación muestra el uso del operador logico |(o):

```
proceso c3_406 // Inicio del algoritmo
//
//      Objetivo   :      Lee nombre y edad. Si edad es menor a 20 o maypr a 25, muestra ambos datos
//      Autor      :      Jose Rojas Davalos
//      Fecha      :      12/10/2014
//
// nombre: del postulante
// edad: del postulante

borrar pantalla // Limpia pantalla

imprimir sin saltar "Nombre del postulante" // Computadora pide nombre

leer nombre // Usuario ingresa nombre

imprimir sin saltar "Edad del postulante" // Computadora pide nombre
```

UNA APROXIMACIÓN A LOS ALGORITMOS

```

leer edad // Usuario ingresa nombre
si edad < 20 | edad > 25 entonces // Si es menor a 20 o mayor a 25
    imprimir nombre, " tiene ", edad, " años" // Imprime el resultado
finSi // Cierre del si
finproceso // Fin del algoritmo
    
```

El operador lógico ~(no) o la negación

A continuación una muestra del uso del operador lógico ~(no):

```

proceso c3_407 // Inicio del algoritmo
//
//      Objetivo :      Lee nombre y edad. Si edad no es mayor a 40, muestra ambos datos
//      Autor      :      Jose Rojas Davalos
//      Fecha      :      12/10/2014
2//
// nombre: del postulante
// edad: del postulante
    borrar pantalla // Limpia pantalla
    imprimir sin saltar "Nombre del postulante" // Computadora pide nombre
    leer nombre // Usuario ingresa nombre
    imprimir sin saltar "Edad del postulante" // Computadora pide nombre
    leer edad // Usuario ingresa nombre
    si ~ (edad > 40) entonces // Si no es mayor a 40
        imprimir nombre, " tiene ", edad, " años" // Imprime el resultado
    finSi // Cierre del si
finproceso // Fin del algoritmo
    
```

Tablas de verdad

UNA APROXIMACIÓN A LOS ALGORITMOS

La conjunción

El único caso en que la conjunción es verdadera es cuando las proposiciones que la forman son ciertas. En todos los demás casos, es falsa. Se muestra la tabla:

```

*****
Tabla de verdad de la conjuncion
Proposición a: edad >= 20
Proposición b: conocimiento = 5
e: edad
c: conocimiento
*****
Conjunción a & b
=====
e  c  a          b          a & b
=  =  =          =          =====
22 5 VERDADERO VERDADERO VERDADERO
22 4 VERDADERO FALSO      FALSO
19 5 FALSO      VERDADERO FALSO
19 5 FALSO      FALSO      FALSO
    
```

La disyunción

El único caso en que la disyunción es falsa es cuando las proposiciones que la forman son falsas. En todos los demás casos, es verdadera. Esta es la tabla:

UNA APROXIMACIÓN A LOS ALGORITMOS

```

*****
Tabla de verdad de la disyuncion
Proposición a: edad >= 20
Proposición b: conocimiento = 5
e: edad
c: conocimiento
*****

Disyunción a i b
=====
e  c  a      b      a i b
=  =  =      =      =====
22  5  VERDADERO  VERDADERO  VERDADERO
22  4  VERDADERO  FALSO      VERDADERO
19  5  FALSO      VERDADERO  VERDADERO
19  5  FALSO      FALSO      FALSO
    
```

La negación

Cuando una afirmación es verdadera su negación es falsa y cuando la afirmación es falsa, su negación es verdadera. La siguiente tabla ilustra lo explicado:

```

*****
Tabla de verdad de la negacion
Proposición a: edad >= 20
e: edad
*****

Negacion ~a
=====
e  a      ~a
=  =      ==
19 FALSO  VERDADERO
20 VERDADERO  FALSO
    
```

La negación de una conjunción

UNA APROXIMACIÓN A LOS ALGORITMOS

La negación de una conjunción es la disyunción de la negación de cada uno sus miembros. La siguiente imagen explica con un ejemplo:

```

*****
Demostracion de que la negacion de una conjuncion
es una disyuncion con las proposiciones negadas
Proposición a: edad >= 20
Proposición b: conocimiento = 5
e: edad
c: conocimiento
*****

e  c  a          b          ~(a & b)  ~a | ~b
=  =  =          =          =====  =====
22 5 VERDADERO VERDADERO FALSO      FALSO
  
```

La negación de una disyunción

La negación de una disyunción es la conjunción de la negación de cada uno sus miembros. Analice el ejemplo que muestra a continuación:

```

*****
Demostracion de que la negacion de una disyuncion
es una conjuncion con las proposiciones negadas
Proposición a: edad >= 20
Proposición b: conocimiento = 5
e: edad
c: conocimiento
*****

e  c  a          b          ~(a | b)  ~a & ~b
=  =  =          =          =====  =====
22 5 VERDADERO VERDADERO FALSO      FALSO
  
```

La doble negación

La negación de una negación es conocida como doble negación que al final resulta ser una afirmación. Un mejor entendimiento puede lograr analizando el ejemplo:

UNA APROXIMACIÓN A LOS ALGORITMOS

```

*****
Demostracion de la doble negacion
Proposición a: edad >= 20
e: edad
*****

e   a           ~(~a)
=   =           =====
22 VERDADERO VERDADERO

```

Ejercicios propuestos

Preparar el algoritmo que:

1. Pida por teclado dos números , calcule la suma de los números introducidos por el usuario, y muestre por pantalla:
 - "LA SUMA SÍ ES MAYOR QUE CERO.", en el caso de que sí lo sea.
 - "LA SUMA NO ES MAYOR QUE CERO.", en el caso de que no lo sea.
2. Pida por teclado el resultado obtenido al lanzar un dado de seis caras y muestre por pantalla el número en letras de la cara opuesta al resultado obtenido. En las caras opuestas de un dado de seis caras están 1-6,2-5,3-4. Si el número ingresado es menor que 1 o mayor que 6 se debe mostrar el mensaje

UNA APROXIMACIÓN A LOS ALGORITMOS

“ERROR: número incorrecto”. Un ejemplo de ejecución del programa podría ser:

Introduzca el resultado del lanzamiento del dado:5

En la cara opuesta está el dos

Otro ejemplo:

Introduzca el resultado del lanzamiento del dado:0

ERROR: número incorrecto

3. Lea por teclado un numero y verifique si es o no divisible exactamente por 5, imprimiendo el mensaje adecuado a cada caso.
4. Verifique si los valores de tres número ingresados por teclado correspondan a los lados de un triángulo equilátero, isósceles, o escaleno. Debe mostrar el mensaje adecuado para cada caso.
5. Pida por teclado dos números, calcule la suma y multiplicación de ambos números,y muestre según sea el caso:
 - "La suma es mayor.", en caso de que sea mayor que la multiplicación de ambos números.
 - "La multiplicación es mayor.", en caso de que sea

UNA APROXIMACIÓN A LOS ALGORITMOS

mayor que la suma de ambos números.

- "La suma y multiplicación son iguales.",

UNA APROXIMACIÓN A LOS ALGORITMOS

CAPITULO 5: ESTRUCTURAS REPETITIVAS

Introducción

Hasta ahora hemos escrito algoritmos de “una sola corrida”. Esto implica que si queremos repetir la ejecución de un procedimiento debemos empezar “de cero” cada vez. Sin embargo existen herramientas para repetir los procesos desde una cantidad constante de veces, pasando por una cantidad variable y hasta una cantidad indeterminada de veces. Estas herramientas que permiten la repetición, cumplen con necesidades del mundo real, para las que se diseñan algoritmos con estas características.

En este contexto, aparecen los contadores, sumadores y cálculo del promedio, que permiten resumir volúmenes de datos en líneas significativas como base para las tomas de decisiones. Otros procedimientos típicos de análisis de datos que requieren ciclos constituyen la búsqueda del mayor, el menor y el rango. Una forma útil para verificar el paso del proceso por uno u otro lugar del ciclo se implementa con indicadores o banderas. Y finalmente algunos procesos requieren usar un ciclo dentro de otro, “anidan-dolos”. Disfruten de la experiencia de repetir los procesos.

Mientras, ¿qué cosa?

La estructura repetitiva mientras posibilita ejecutar repetidamente un conjunto de

UNA APROXIMACIÓN A LOS ALGORITMOS

sentencias, como su nombre lo dice, mientras se cumpla cierta condición definida.

La sintaxis del ciclo mientras es:

Mientras expresion_logica Hacer
secuencia_de_acciones
Fin Mientras

Un ejemplo de su uso se muestra a continuación:

```
mientras cedula >0 hacer // Permite repetir el proceso si se cumple la condicion. Sino ...

    imprimir sin saltar "Nombre:" // Computadora solicita nombre
    leer nombre // Usuario ingresa nombre

    imprimir sin saltar "Apellido:" // Computadora solicita apellido
    leer apellido // Usuario ingresa apellido

    imprimir apellido+" "+nombre+" "+ConvertirATexto(cedula) // Muestra datos solicitados
    imprimir sin saltar "Cedula(cero o negativo para fin:" // Computadora pide cedula las...
    leer cedula // Usuario ingresa cedula las siguientes veces

finmientras // Sale del mientras
```

Repetir, ¿hasta cuando?

A diferencia del ciclo mientras, el ciclo repetir-hasta ejecuta al menos una vez las sentencias definidas en el cuerpo del ciclo, pues la verificación para continuar o no en el ciclo se realiza al final, como puede apreciarse en la sintaxis:

Repetir
secuencia_de_acciones

UNA APROXIMACIÓN A LOS ALGORITMOS

Hasta Que expresion_logica

Debemos observar que las palabras claves son *repetir* y *hasta que*. Si bien esta forma de ciclo es utilizado con menor frecuencia que el ciclo mientras, hay situaciones en las que con el, el algoritmo resultante es más sencillo y directo. A continuación un ejemplo:

Repetir // Permite repetir el proceso. Se realiza por lo menos una vez

imprimir sin bajar "Teclee texto o <Enter> para finalizar" //

Computadora solicita texto

leer texto // Usuario ingresa texto

mostrar texto // Computadora muestra texto ingresado

Hasta Que texto="" // Hasta que texto sea nulo, consecuencia de presionar <Enter>. Verifica condición ...

¿Desde cuando y hasta cuando?

Utilizamos el ciclo *para* cuando conocemos previamente el número de veces que deseamos ejecutar un conjunto de instrucciones.

La sintaxis es:

Para variable_numerica<-valor_inicial Hasta valor_final Con Paso paso

Hacer

secuencia_de_acciones

UNA APROXIMACIÓN A LOS ALGORITMOS

Fin Para

Al igual que el ciclo *mientras*, el conjunto de sentencias que forman parte de un ciclo *para* puede que nunca se ejecute. Esto ocurrirá si:

- El valor inicial es superior al valor final y el incremento es positivo, ó
- El valor inicial es inferior al valor final y el incremento es negativo.

Se muestra un ejemplo, más abajo:

para i=1 hasta 5 // Incrementa el contador y verifica fin. Si i<=5 continua. Sino salta a finpara

imprimir sin bajar "Teclee texto" // Computadora solicita texto

leer texto // Usuario ingresa texto

mostrar convertirtatexto(i)+"-"+texto // Computadora muestra numero de orden....

finpara // Sale del ciclo

Más detalles del ciclo para

Cuando usamos el *para* desde debemos tener presente algunos detalles:

- El valor de inicio, el valor de fin y el incremento deben ser expresiones numéricas. El primero (inicio) y el último (incremento) son siempre automáticamente truncados a enteros. Ejemplo:

<i>para</i>	<div>Valor inicial <i>i=2</i></div>	<i>hasta</i>	<div>Valor final <i>8</i></div>	<i>con paso</i>	<div>Incremento <i>2</i></div>
-------------	--	--------------	--	-----------------	---

UNA APROXIMACIÓN A LOS ALGORITMOS

- El ciclo termina por un valor más de su incremento. Si el incremento es 1, termina por 1 más, si es 2 termina por dos más, etc. Así el siguiente ciclo:

para contar=1 hasta 10

termina en 11. Ó en

para i=2 hasta 8 paso 2

termina en 10.

- El valor del incremento puede ser negativo (por ejemplo, para ir de n a 1):

para i=n hasta 1 con paso -1 // Recorre desde n hasta 1

en forma inversa

- El ciclo puede realizarse de 0 a n veces.
- La variable de control debe ser numérica, pudiendo ser un elemento de un vector o matriz.
- Si se desea salir del ciclo en forma anticipada, puede asignarse a la variable de control el valor de fin.

Contadores, ¿para que se usan?

En los procesos repetitivos se necesitan, normalmente, contar los sucesos o acciones internas de un ciclo, como por ejemplo, la cantidad de veces que se

UNA APROXIMACIÓN A LOS ALGORITMOS

repite una situación dada, ó simplemente, cuantas veces se repetirá un ciclo, como se muestra en los ejemplos siguientes:

Ejemplo 1

mientras edad > 0 Hacer // Permite repetir el proceso si se cumple la condicion. Sino salta a finmientras

c = c + 1 // Computadora incrementa contador de edades leídas

imprimir sin saltar "Edad(cero o negativo para fin:" // Computadora pide edad las siguientes veces

leer edad // Usuario ingresa edad las siguientes veces

finmientras // Sale del ciclo

imprimir "La cantidad de edades leídas es ",c // Computadora muestra valor final del contador

Ejemplo 2

para i=1 hasta n // Inicia, continua y finaliza el proceso que se ejecuta n veces

imprimir sin bajar "Teclee texto" // Computadora solicita texto

leer texto // Usuario ingresa texto

mostrar convertiratexto(i)+"-"+texto // Computadora muestra numero de orden y ...

fin para // Sale del ciclo

Un contador siempre será una variable numérica pasible de un incremento o un decremento(incremento negativo), mediante una constante numérica.

Ejemplos de incremento

- $c = c + 1$ // Computadora incrementa contador de edades leídas
- para $i=1$ hasta n // Inicia, continua y finaliza el proceso que se ejecuta n veces

Ejemplo de decremento

para $i=n$ hasta 1 con paso -1 // Incrementa el contador y verifica

UNA APROXIMACIÓN A LOS ALGORITMOS

fin. Si $i \geq l$ continua. Sino salta a finpara

Para el contador existe un estado inicial:

$c=0$ // Inicia contador a cero

un proceso:

$c = c + 1$ // Computadora incrementa contador de edades leídas

y eventualmente un uso al final de los procesos:

Si $c \neq 0$ // Se verifica si el contador tiene valor cero, para evitar la división por cero

$p=s/c$ // Computadora calcula promedio de edades

FinSi

Por una buena práctica, es conveniente tomar como costumbre, inicializar explícitamente un contador, teniendo en cuenta que esto no es estándar para todas las herramientas algorítmicas.

Sumadores. ¿en qué son útiles?

Un sumador o totalizador tiene como misión almacenar cantidades variables resultantes de sumas sucesivas. Cumple casi el mismo papel que el contador, excepto por el hecho que el incremento o decremento de cada suma es variable, en lugar de constante como es el caso del contador. Un ejemplo:

$s = s + edad$ // Computadora incrementa sumador de edades leídas

UNA APROXIMACIÓN A LOS ALGORITMOS

También un sumador debe ser inicializado con las mismas consideraciones hechas para el contador:

s=0 // Inicia sumador a cero

incrementado en su momento:

s = s + edad // Computadora incrementa sumador de edades leídas

y utilizado para un cálculo final:

Si c<>0 // Se verifica si el contador tiene valor cero, para evitar la division por cero

p=s/c // Computadora calcula promedio de edades

FinSi

¿Cómo finaliza un ciclo?

Se puede dar, de acuerdo a lo que expresa el enunciado las siguientes situaciones de fin de ciclo:

- Cuando la cantidad de veces a repetir es constante:

para i=1 hasta 5 // Incrementa el contador y verifica fin. Si i<=5 continua. Sino salta a finpara

imprimir sin bajar "Teclee texto" // Computadora solicita texto

leer texto // Usuario ingresa texto

mostrar convertirtatexto(i)+"-"+texto // Computadora muestra numero de orden y

textos....

finpara // Sale del ciclo

- Si la cantidad de repeticiones es dada por una variable:

para i=1 hasta n // Inicia, continua y finaliza el proceso que se ejecuta n veces

imprimir sin bajar "Teclee texto" // Computadora solicita texto

leer texto // Usuario ingresa texto

mostrar convertirtatexto(i)+"-"+texto // Computadora muestra numero de orden y

UNA APROXIMACIÓN A LOS ALGORITMOS

```

fin para //Sale deñ ciclo
• Es indefinida y depende de una condición:
imprimir sin saltar "Cedula(cero o negativo para fin:" // Computadora pide cedula la primera vez
leer cedula // Usuario ingresa cedula la primera vez
mientras cedula >0 hacer // Permite repetir el proceso si se cumple la condicion. Sino salta a
finmientras

imprimir sin saltar "Nombre:" // Computadora solicita nombre
leer nombre // Usuario ingresa nombre

imprimir sin saltar "Apellido:" // Computadora solicita apellido
leer apellido // Usuario ingresa apellido

imprimir apellido+" "+nombre+" "+ConvertirATexto(cedula) // Muestra datos
solicitados

imprimir sin saltar "Cedula(cero o negativo para fin:" // Computadora pide cedula las
siguientes...

leer cedula // Usuario ingresa cedula las siguientes veces

finmientras // Sale del mientras
    
```

El promedio, una herramienta estadística

El promedio, desde el punto de vista estadístico, es una herramienta que permite descubrir cierto comportamiento en los datos. En nuestro ejemplo se quiere obtener promedio de edades. Como el concepto de promedio implica que es el resultado que se obtiene a partir del cociente entre la suma y la cantidad, desde la visión algorítmica se necesita calcular antes que el promedio, la suma y cantidad. En palabras más simples el promedio es suma sobre cantidad, y se calcula al final del proceso, fuera del ciclo, con los resultados finales de ambos cálculos. En general, no tiene mucho sentido usar el resultado de la suma, el contador ni el

UNA APROXIMACIÓN A LOS ALGORITMOS

promedio, en forma parcial dentro de un ciclo. Son, por excelencia, resultados finales. Volviendo al tema central de este apartado, el promedio, normalmente su obtención conlleva, en cuanto a los elementos involucrados, las siguientes etapas:

- **Inicialización**

s=0 // Inicia sumador a cero

c=0 // Inicia contador a cero

- **Incremento**

s = s + edad // Computadora incrementa sumador de edades leídas

c = c + 1 // Computadora incrementa contador de edades leídas

- **Cálculo final (fuera del ciclo)**

p=s/c // Computadora calcula promedio de edades

- **Despliegue de resultados finales (fuera del ciclo)**

imprimir "La suma de edades leídas es ",s // Computadora muestra valor final del sumador

imprimir "La cantidad de edades leídas es ",c // Computadora muestra valor final del contador

imprimir "El promedio de edades leídas es ",p // Computadora muestra promedio de edades leídas

Buscando el mayor

Un procedimiento típico y frecuente es el de la búsqueda del mayor, debido que en la vida real es usual que se quiera saber, por ejemplo, el mejor salario, la mejor nota, la edad mayor en un grupo de personas, etc.

En ese contexto el algoritmo de la búsqueda del mayor, sencillamente reproduce

UNA APROXIMACIÓN A LOS ALGORITMOS

lo que cualquier ser humano haría al buscar algo(o algunos) que sea mayor a todos los elementos. Para este menester, debe tomar un primer valor mayor, sin entrar a comparar, mediante una asignación directa:

imprimir sin saltar "Edad" // Computadora solicita la primera edad

leer edad // Usuario ingresa la primera edad

mayor = edad // Se considera la primera edad leída como mayor

En este caso se busca la mayor edad, y para ello se lee y se guarda la primera edad como mayor. Lo que se ha mostrado se realiza en los preliminares, fuera del ciclo.

En el proceso repetitivo se va comparando el valor actual que se tiene como mayor con el nuevo valor que “viene”. Si el valor mayor aparece más de una vez en la serie y quiero “quedarme” con el primer mayor una forma, no la única, de comparar sería:

si edad > mayor entonces // Si la edad nueva es mayor que la guardada

mayor = edad // Cambia el mayor

fin si // Cierre del si

Así en la serie de edades [18,20,19,20,18] se tomaría como mayor el primero que aparece en la serie.

El ejemplo además muestra, que cuando se cumple la condición se guarda el nuevo mayor. Otro caso posible es, cuando también en una serie aparece el valor

UNA APROXIMACIÓN A LOS ALGORITMOS

mayor más de una vez y quiero guardar el último que aparece, debo cambiar la forma de comparar:

si edad >= mayor entonces // Si la edad nueva es mayor o igual que la guardada

mayor = edad // Cambia el mayor actual a una nueva edad mayor

finsi // Cierre del si

Si se aplica esta forma de resolución en la serie [18,20,19,20,18] se tomaría como mayor el último que aparece en la serie.

También es digna de comentar, la situación que se presenta cuando con el mayor se busca un valor asociado. Es decir, que cuando se toma el primer valor para el criterio del mayor, también se debe guardar al valor asociado buscado:

si indicador = 0 entonces // Si la el indicador esta desactivado

indicador = 1 // Se activa el indicador

mayor = cedula // Se guarda la primera cedula como mayor

xnombre = nombre // Se guarda el nombre en xnombre

xapellido = apellido // Se guarda el apellido en xapellido

sino

.....

.....

Esto implica que cuando cambia el mayor en el procedimiento, también debe cambiar el valor asociado:

si cedula >= mayor // Si el valor de cedula que viene tiene mayor o igual valor que el guardado como mayor

mayor = cedula // Nuevo mayor

xnombre = nombre // Se guarda el nombre en xnombre

xapellido = apellido // Se guarda el apellido en xapellido

UNA APROXIMACIÓN A LOS ALGORITMOS

finsi // Cierre del si

Finalmente, la razón de todo el algoritmo, el despliegue de resultados, se produce:

imprimir "El mayor valor de cedula es ", mayor, " y corresponde a "+xnombre+" "+xapellido

En este instante, es importante resaltar que el despliegue de los datos del mayor y asociado, se debe realizar en la mayoría de los casos al final, fuera del proceso repetitivo. Son resultados de fin de proceso por excelencia, pero excepciones siempre hay.

Encontrando el menor

Así como se necesita la búsqueda del mayor, también existen requerimientos de búsqueda de un valor menor, como por ejemplo, el menor salario, la menor nota, la edad menor en un grupo de personas, etc.

El algoritmo de la búsqueda del menor, traduce lo que cualquier persona haría al buscar el menor entre todos los elementos. Para esto, se acostumbra tomar un primer valor menor, sin entrar en comparación alguna:

menor = 9999 // Se guarda edad imposible como menor

En este caso se busca la edad de la persona más joven, y ella nunca podrá ser 9999, pues para este algoritmo se toman las edades de las personas, que en las condiciones de vida actuales, difícilmente sobrepasen la cifra tomada inicialmente como menor. La línea mostrada se encuentra en los inicios del procedimiento,

UNA APROXIMACIÓN A LOS ALGORITMOS

fuera de la repetición.

A continuación se compara el valor actual que se tiene como menor con el nuevo valor que “aparece”. En caso que el valor menor se presente más de una vez en la lista y deseo tener al primero que aparece como menor, una forma, no la única, de comparar sería:

```
si edad < menor entonces // Si la edad nueva es menor que la guardada
```

```
menor = edad // Cambia el menor
```

```
finsi // Cierre del si
```

En la lista de edades [50,80,18,30,70,18,89,77] quedaría como menor el primero que aparece en la lista.

Lo visto en el ejemplo permite deducir, que a condición cumplida, nuevo menor guardado. La otra posibilidad es que, en la lista aparezca el mínimo dos o más veces y mi deseo es guardar el de la última aparición, por lo que debo cambiar la forma de comparar:

```
si edad <= menor entonces // Si la edad nueva es menor o igual que la guardada
```

```
menor = edad // Cambia el menor
```

```
finsi // Cierre del si
```

Si la lista es [50,80,18,30,70,18,89,77] el menor sería el valor 18 resaltado.

Otro aspecto a destacar, es el hecho de buscar un menor y un valor asociado a el. Significa , que al asignar por primera vez el valor del menor, corresponde almacenar al valor asociado buscado:

UNA APROXIMACIÓN A LOS ALGORITMOS

```

imprimir sin saltar "Nombre"

leer nombre // Primera lectura de nombre

imprimir sin saltar "Edad"

leer edad // Primera lectura de edad para guardar como referencia de menor

menor=edad // Guarda como edad del menor por ahora

xnombre=nombre // Guarda como nombre del menor por ahora
    
```

Posteriormente, en el procedimiento, cada vez que cambia el valor del menor, es imprescindible que también cambie el valor del valor asociado:

```

si edad<= _menor // Si edad leída es menor o igual que el actual menor

    menor=edad // Se modifica edad menor actual a nuevo menor

    xnombre=nombre // En consecuencia tambien el nombre

finsi
    
```

Un algoritmo no es completo sino muestra sus resultado:

```

imprimir "El menor valor de edad es ",menor, " y corresponde a "+xnombre
    
```

Como en los casos del sumador, el contador, el promedio y el mayor, el menor y su valor asociado también es resultado de fin de proceso por excelencia.

El rango

Es también un elemento estadístico que se obtiene de la diferencia entre el mayor y el menor valor. Para ello, el algoritmo que calculará el rango debe, de entre una serie de datos, obtener:

- El mayor
- El menor, y

UNA APROXIMACIÓN A LOS ALGORITMOS

- Finalmente el rango.

Es decir que $\text{rango} = \text{mayor} - \text{menor}$. A continuación, un ejemplo de fragmentos del procedimiento que obtiene el rango:

- **Inicializaciones**

```
imprimir sin saltar "¿Cuántas edades va a procesar?" // Computadora solicita valor de x
leer x // Usuario teclea valor de x

imprimir sin saltar "Edad" // Computadora solicita la primera edad
leer edad // Usuario ingresa la primera edad

mayor = edad // Se considera la primera edad leída como mayor
menor = edad // Se considera la primera edad leída como menor
```

- **Proceso**

```
para i=2 hasta x // Para leer desde la segunda edad hasta la edad x

imprimir sin saltar "Edad" // Computadora solicita la siguiente edad
leer edad // Usuario ingresa la siguiente edad

si edad < menor entonces // Si la edad nueva es menor que la guardada
    menor = edad // Cambia el menor
fin si // Cierre del si

si edad > mayor entonces // Si la edad nueva es mayor que la guardada
    mayor = edad // Cambia el mayor
fin si // Cierre del si

finpara // Sale del ciclo
```

- **Finales**

```
rango = mayor - menor // Computador calcula rango

imprimir "El rango entre ", mayor, " y ", menor, " es ", rango // Computadora muestra valor del rango
```

Los ciclos anidados

UNA APROXIMACIÓN A LOS ALGORITMOS

Algunas veces existen necesidades que obligan a usar un ciclo dentro de otro. Por ejemplo que por cada ocurrencia de un ciclo desde, se desarrolle otro en su totalidad:

mientras cedula>0 hacer // Mientras cedula sea mayor a cero

imprimir sin saltar "Nombre" // Computadora solicita nombre

leer nombre // Usuario responde ingresando nombre

imprimir sin saltar "Apellido" // Computadora solicita nombre

leer apellido // Usuario responde ingresando apellido

nota_final=0 // Pone a cero sumadora de notas

para i=1 hasta 5 // Para ingresar notas para los criterios

imprimir sin saltar "Nota en criterio ",i,"(1 al 5)" // Computadora solicita

leer nota_por_criterio // Usuario responde ingresando nota en

nota_final=nota_final+nota_por_criterio

fin para

imprimir apellido," ",nombre," ",cedula," ",nota_final // Imprime datos solicitados

imprimir sin saltar "Cedula(cero o negativo para fin)" // Computadora solicita la cedula

leer cedula // Usuario ingresa la cedula del siguiente postulante

finmientras // Sale del ciclo

En el ejemplo, la indentación o sangría no hace falta, desde el punto de vista de las exigencias de la herramienta. Sin embargo es altamente recomendable, para visualizar que el ciclo está anidado dentro de otro. En este caso, el ciclo que solicita las notas por cada criterio y controlado *por para*, se dice que está dentro del otro ciclo controlado por en el *mientras* con la condición *cedula>0*, que permite que al fin de carga de notas para un postulante se pase al siguiente.

UNA APROXIMACIÓN A LOS ALGORITMOS

Al ciclo que se desarrolla más lentamente, como es el caso del controlado por el *mientras*, vamos a llamarlo ciclo exterior. Al que por cada ocurrencia del ciclo exterior se desarrolla completamente llamaremos ciclo interior, como es el caso del controlado por el *para* que es el ciclo anidado dentro del exterior. A los efectos didácticos, veremos ejemplos de dos ciclos, es decir uno anidado dentro de otro. Los límites posibles para la anidación de los ciclos, están dados por la memoria de la computadora o por las posibilidades de la herramienta algorítmica. En cuanto a las diferentes combinaciones de estructuras cíclicas para construir ciclos anidados, los límites los constituyen solamente la imaginación y las necesidades del usuario. Así, como en este caso, un ciclo *para* puede estar anidado dentro de un *mientras*.

El indicador o bandera

Un indicador o bandera, también conocido como interruptor, conmutador, switch o flag, es una variable que puede tomar diversos valores a lo largo de la ejecución de un algoritmo y permite comunicar información de una parte a otra del mismo (del algoritmo). Los interruptores, por convención, pueden(o deberían de) tomar dos valores: 1(unos) o 0(cero). El primero indica encendido mientras el segundo establece que el indicador está apagado. A continuación un ejemplo de su uso:

```
indicador = 0 // Se pone indicador a cero
```

```
imprimir sin saltar "Cedula(cero o negativo para fin)" // Computadora solicita la primera cedula
```

UNA APROXIMACIÓN A LOS ALGORITMOS

```

leer cedula // Usuario ingresa la primera cedula
mientras cedula>0 hacer // Mientras cedula sea mayor a cero

    imprimir sin saltar "Nombre" // Computadora solicita nombre
    leer nombre // Usuario ingresa nombre
    imprimir sin saltar "Apellido" // Computadora solicita apellido
    leer apellido // Usuario ingresa apellido

    si indicador = 0 entonces // Si la el indicador esta desactivado
        indicador = 1 // Se activa el indicador

        mayor = cedula // Se guarda la primera cedula como mayor
        xnombre = nombre // Se guarda el nombre en xnombre para asegurar que acompanha ...
        xapellido = apellido // Se guarda el apellido en xapellido para asegurar que acompanha...

    sino

        si cedula >= mayor // Si el valor de cedula que viene tiene mayor o igual valor...
            mayor = cedula // Nuevo mayor
            xnombre = nombre // Se guarda el nombre en xnombre para ...
            xapellido = apellido // Se guarda el apellido en xapellido para ...

        fin si // Cierre del si
    fin si // Cierre del si

    .....
    .....
    .....
    
```

En este caso el objetivo del programa es hallar el mayor valor de cédula de identidad entre varios, y se necesita saber cuando se lee el primer valor para guardarlo como primer mayor. Para ello se inicializa *indicador = 0*, luego se consulta su estado, para posteriormente “encenderlo” *indicador = 1*, cumpliéndose de esta manera el ciclo de vida del interruptor.

Universidad Nacional de Asunción

Facultad de Ciencias Químicas

UNA APROXIMACIÓN A LOS ALGORITMOS

Ejercicios propuestos

UNA APROXIMACIÓN A LOS ALGORITMOS

Preparar el algoritmo que:

1. Lea por teclado la edad y el nombre de varias personas, imprima cada registro leído y muestre al final el nombre y la edad de la persona con más edad. El programa finaliza cuando se lee una edad negativa.

Lea por teclado la edad y el nombre de n personas, imprima cada registro leído y muestre al final el promedio de las edades de las personas con mas de 40 años.

2. Resuelva la ecuación $y=x^2+2x+5$ para los valores de x enteros, pares mayores a cero y no superiores a n . El único valor que se ingresa por teclado es el de n . Imprimir el x generado y el y calculado.

3. Dados dos números enteros, muestre por pantalla uno de estos mensajes: “El segundo es el cuadrado exacto del primero”, “El segundo es menor que el cuadrado del primero” o “El segundo es mayor que el cuadrado del primero”, dependiendo de la verificación de la condición correspondiente al significado de cada mensaje. Repetir el proceso para x pares de números.

4. Dados n números enteros, determine cuál de los números es más cercano al primero. Por ejemplo, si el usuario introduce, para $n=5$, los números 2, 6, 4, 1 y 10, el programa responderá que el número más cercano al 2 es el 1.

5. Dados p puntos en el plano, determine cuál de los demás puntos es más

UNA APROXIMACIÓN A LOS ALGORITMOS

cercano al primero. Un punto se representa con dos variables: una para la abscisa y otra para la ordenada. La distancia entre dos puntos (x_1, y_1) y (x_2, y_2) es $\sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$.

6. Muestre todos los múltiplos de n entre n y m , ambos inclusive, donde n y m son números introducidos por el usuario. Se debe validar que m sea mayor o igual a n .

7. Muestre todos los números potencia de 2 entre 2^1 y 2^z , ambos inclusive.

8. Calcule el factorial de un número entero positivo. Si el número es n el factorial de n es igual a $1 * 2 * 3 * \dots * (n-1) * n$.

9. Muestre la tabla de multiplicar de un número introducido por teclado por el usuario. Aquí tiene un ejemplo de cómo se debe comportar el programa:

Ingrese un número: 5

$$5 \times 1 = 5$$

$$5 \times 2 = 10$$

$$5 \times 3 = 15$$

$$5 \times 4 = 20$$

$$5 \times 5 = 25$$

$$5 \times 6 = 30$$

$$5 \times 7 = 35$$

UNA APROXIMACIÓN A LOS ALGORITMOS

$$5 \times 8 = 40$$

$$5 \times 9 = 45$$

$$5 \times 10 = 50$$

UNA APROXIMACIÓN A LOS ALGORITMOS

CAPÍTULO 6: OPERACIONES CON CADENAS

Introducción

Los textos tienen un tratamiento muy importante desde el punto de vista algorítmico, pues, las operaciones con cadenas permiten dar sustento a aquellos. Como ya hemos visto, una de las operaciones básicas con cadenas es la concatenación. Existen además otras operaciones y funciones aplicables a ellas que son de utilidad en muchas situaciones. Las principales son el acceso a los caracteres y las conversiones de tipos de datos. Dispónganse a disfrutar del mundo de las cadenas!!!

Calculando la longitud

La longitud de una cadena puede determinarse con la función predefinida **longitud()**. Por ejemplo:

Escribir 'Ingrese el primer texto' Sin Saltar

Leer cad_1

Escribir 'Ingrese el segundo texto' Sin Saltar

Leer cad_2

long_1<-longitud(cad_1)

long_2<-longitud(cad_2)

Escribir 'El texto ',cad_1,' tiene una longitud igual a ',long_1

Escribir 'El texto ',cad_2,' tiene una longitud igual a ',long_2

Así, un resultado posible es:

UNA APROXIMACIÓN A LOS ALGORITMOS

```
Ingrese el primer texto> Azufre
Ingrese el segundo texto> Potasio
El texto Azufre tiene una longitud igual a 6
El texto Potasio tiene una longitud igual a 7
*** Ejecución Finalizada. ***
```

Recorriendo una cadena

En una cadena cada elemento es un carácter y puede ser referido por su posición dentro del conjunto. Por, lo tanto podemos recorrer una cadena y mostrar cada carácter mediante la función **subcadena()** que tiene 3 parámetros. El primero es la cadena de la cual vamos a extraer la subcadena. El segundo es una expresión numérica que indica la posición inicial desde donde vamos a extraer y el tercero otra expresión numérica que constituye la posición hasta donde vamos extraer. El siguiente es un algoritmo ejemplo:

```
Proceso c3_602

    Borrar Pantalla

    Escribir 'Ingrese un texto' Sin Saltar

    Leer cad

    Escribir concatenar('El texto ingresado es ',cad)

    l longitud_texto<-longitud(cad)

    Escribir 'Los caracteres del texto '+cad+' son:'

    Para i<-1 Hasta longitud_texto Hacer

        car<-subcadena(cad,i,i)

        Escribir i,'-',car

    FinPara

FinProceso
```

UNA APROXIMACIÓN A LOS ALGORITMOS

Una muestra de su ejecución:

```
Ingrese un texto> Ana
El texto ingresado es Ana
Los caracteres del texto Ana son:
1- A
2- n
3- a
*** Ejecución Finalizada. ***
```

También se puede recorrer una cadena en forma inversa:

```
Ingrese un texto> Anita
El texto ingresado es Anita
Los caracteres del texto, en forma invertida, Anita son:
1- a
2- t
3- i
4- n
5- A
*** Ejecución Finalizada. ***
```

Las líneas del algoritmo que realizan la acción mostrada:

Proceso c3_603

Borrar Pantalla

Escribir 'Ingrese un texto' Sin Saltar

Leer cad

Escribir concatenar('El texto ingresado es ',cad)

longitud_texto<-longitud(cad)

Escribir 'Los caracteres del texto, en forma invertida, '+cad+' son:'

Para i<-longitud_texto Hasta 1 Con Paso -1 Hacer

car<-subcadena(cad,i,i)

Escribir (longitud_texto+1)-i,'-',car

FinPara

FinProceso

UNA APROXIMACIÓN A LOS ALGORITMOS

Convirtiendo a mayúsculas o minúsculas

Una cadena se dice que se convierte a mayúsculas cuando se le aplica la función predefinida **mayusculas()**. El efecto de esta función es convertir todas las letras minúsculas de la cadena a mayúsculas, ignorando a todos los otros tipos, es decir mayúsculas, dígitos y caracteres especiales. Un ejemplo:

Proceso c3_604

Borrar Pantalla

Escribir 'Ingrese un texto' Sin Saltar

Leer cad

cad1<-mayusculas(cad)

Escribir cad+' en mayusculas es '+cad1

longitud_texto<-longitud(cad)

Escribir ' '

Escribir 'Caracter original '+'Caracter convertido'

Para i<-1 Hasta longitud_texto Hacer

car<-subcadena(cad,i,i)

car1<-mayusculas(car)

Escribir car+' '+car1

FinPara

FinProceso

Una imagen de la ejecución:

UNA APROXIMACIÓN A LOS ALGORITMOS

```

Ingrese un texto> hola
hola en mayusculas es HOLA

Caracter original Caracter convertido
h                H
o                O
l                L
a                A
*** Ejecución Finalizada. ***
    
```

La conversión a minúsculas se realiza con la función `minusculas()`. Convierte las letras mayúsculas de la cadena a minúsculas, ignorando a todos los otros tipos, es decir minúsculas, dígitos y caracteres especiales. Una muestra de su uso:

Proceso c3_605

Borrar Pantalla

Escribir 'Ingrese un texto' Sin Saltar

Leer cad

cad1<-minusculas(cad)

Escribir cad+' en minusculas es '+cad1

longitud_texto<-longitud(cad)

Escribir ' '

Escribir 'Caracter original '+'Caracter convertido'

Para i<-1 Hasta longitud_texto Hacer

car<-subcadena(cad,i,i)

car1<-minusculas(car)

Escribir car+' '+car1

FinPara

FinProceso

Un resultado posible es:

UNA APROXIMACIÓN A LOS ALGORITMOS

```
Ingrese un texto> HOLA
HOLA en minusculas es hola

Caracter original Caracter convertido
H h
O o
L l
A a
*** Ejecución Finalizada. ***
```

Extrayendo una subcadena

Una cadena está formada por caracteres y dentro de cadena se pueden agrupar las cadenas, en subcadenas. Una de las operaciones básicas es extraer una subcadena de una cadena. Para ello se cuenta con la función cadena **subcadena()**. Así, si la cadena es “paralelo”, la posición inicial es 5 y la posición hasta es de 8, la subcadena resultante debe ser “lelo”, según se muestra a continuación:

```
Ingresar texto> paralelo
¿Desde que posicion quiere extraer?> 5
¿Hasta que posición quiere extraer?> 8
De paralelo se extrajo lelo
*** Ejecución Finalizada. ***
```

Veamos como se implementa esto en el algoritmo:

Proceso c3_606

Borrar Pantalla

Escribir 'Ingresar texto' Sin Saltar

Leer cad

Escribir '¿Desde que posicion quiere extraer?' Sin Saltar

Leer posinicial

Escribir '¿Hasta que posición quiere extraer?' Sin Saltar

Leer posfinal

UNA APROXIMACIÓN A LOS ALGORITMOS

```
subcad<-subcadena(cad,posinicial,posfinal)
```

```
Escribir 'De ',cad,' se extrajo ',subcad
```

```
FinProceso
```

Convirtiendo numérico a cadena

Algunas veces por cuestiones prácticas se necesita almacenar todos los datos como cadena y por ende, convertir datos almacenados como numéricos a cadena. Este tipo de situaciones se soluciona con la función **ConvertirATexto()** cuya ejemplo de uso, en cuanto resultado de ejecución, se muestra a continuación:

```
El numero en formato numerico es 78.135
El numero en formato texto es 78.135
*** Ejecución Finalizada. ***
```

El algoritmo que usa la función es:

```
Proceso c3_608
```

```
Borrar Pantalla
```

```
numero<-78.135
```

```
Escribir 'El numero en formato numerico es ',numero
```

```
num_a_texto<-ConvertirATexto(numero)
```

```
Escribir 'El numero en formato texto es '+num_a_texto
```

```
FinProceso
```

Convirtiendo cadena a numérico

En el tópico anterior, vimos como convertir una expresión numérica a cadena. En esta ocasión presentamos la función con objetivo inverso, que es la de convertir una expresión cadena a una numérica. La función encargada de ese menester es

UNA APROXIMACIÓN A LOS ALGORITMOS

ConvertirANumero(), que toma como parámetro una expresión cadena y retorna su valor en formato numérico. En el ejemplo convierte el texto a numérico y le suma 10 al resultado, como se muestra:

```
El numero en formato texto es 78.135 y el valor numerico es 88.135
*** Ejecución Finalizada. ***
```

Veamos el algoritmo que realiza la conversión:

Proceso c3_609

Borrar Pantalla

txt_nro<-'78.135'

num_nro<-ConvertirANumero(txt_nro)

num_nro<-num_nro+10

Escribir 'El numero en formato texto es '+txt_nro+' y el valor numerico es ',num_nro

FinProceso

Ejercicios propuestos

Construya un algoritmo, que:

1. Dado un texto, lo recorra carácter a carácter y por cada uno de ellos lo identifique cómo vocal minúscula, vocal mayúscula, consonante minúscula, consonante mayúscula u otro si aparece otro tipo de carácter.
2. Solicite la lectura de un texto que no contenga letras mayúsculas. Si el usuario teclea una letra mayúscula, el programa solicitará nuevamente la introducción del texto cuantas veces sea preciso.
3. Lea una cadena y muestre el número de espacios en blanco, letras

UNA APROXIMACIÓN A LOS ALGORITMOS

mayúsculas y letras minúsculas que contiene.

4. Lea *c* cadenas y muestre en pantalla el mensaje, por cada cadena, “Contiene dígito” si contiene algún dígito y “No contiene dígito” en caso contrario.
5. Lea *n* cadenas y un número entero *k* y nos diga cuántas tienen una longitud de *k* caracteres.
6. Muestre cada carácter especial de un texto ingresado por teclado.
7. Dados una cadena *c*, un índice *i* y un número *n*, muestre la subcadena de *c* formada por los *n* caracteres que empiezan en la posición *i*.
8. Lea un texto y cambie todos los dígitos por la letra mayúscula que le corresponde en el alfabeto. Así al dígito 0 le corresponde la A, al 1 la B y así sucesivamente. Imprimir el texto antes y después del cambio.
9. Lea una lista de **cant** nombres y los muestre entre paréntesis.
10. Lea un numero mientras no sea 0 o negativo y concatene todos los números leídos en una sola cadena de caracteres.

UNA APROXIMACIÓN A LOS ALGORITMOS

CAPITULO 7: SUBROUTINAS

Introducción

La subrutina o función es una poderosa herramienta algorítmica que permite dividir los grandes problemas en problemas más pequeños para facilitar su resolución. Acompaña a esto la implementación del método “top-down” (de arriba para abajo) que ayuda a plantear y escribir los algoritmos en forma modular. El hecho de escribir funciones, que son como pequeños algoritmos, con un solo punto de entrada, otro se salida y con una (solamente una) misión específica, facilita la comprensión y el seguimiento en los momentos de problemas o de modificar procedimientos construidos por otros. Si esto se complementa con el uso racional y adecuado de los comentarios, se propicia la legibilidad y el fácil mantenimiento(modificación tendiente a mejoras) de los algoritmos ya existentes y de la creación de nuevos. Encaremos el desafío de las subrutinas.

El algoritmo principal

El algoritmo principal, llamador o invocante es aquel que hace uso de los servicios de una subrutina. Mirado desde el punto de vista de la jerarquía el algoritmo principal es como el “jefe” de la subrutina, pues al invocarla le pasa una orden para que ejecute alguna acción. El ejemplo:

```
// *****
```

UNA APROXIMACIÓN A LOS ALGORITMOS

```
// Zona de algoritmo principal
// *****

proceso c7_01 // Inicio del algoritmo

//

//      Objetivo      :      Ingresa y muestra datos de postulante mientras cedula sea mayor a cero
//
//                                     Escribir la peticion y la lectura de cedula en una
subrutina

//      Autor          :      Jose Rojas Davalos
//      Fecha          :      15/10/2014
//

// cedula: de identidad del postulante
// nombre: del postulante
// apellido: del postulante
// edad: del postulante

borrar pantalla // Limpia pantalla

cedula=leer_cedula() // Invoca funcion de lectura de cedula primera vez

mientras cedula >0 hacer // Permite repetir el proceso si se cumple la condicion. Sino salta a
finmientras

    imprimir sin saltar "Nombre" // Computadora solicita nombre
    leer nombre // Usuario ingresa nombre

    imprimir sin saltar "Apellido" // Computadora solicita apellido
    leer apellido // Usuario ingresa apellido

    imprimir  apellido+" " "+nombre+" " "+ConvertirATexto(cedula) // Muestra  datos
solicitados

cedula=leer_cedula // Si no se envia parametros se puede invocar sin parentesis

finmientras // Sale del mientras

finproceso // Fin del algoritmo principal
```

La subrutina o función es invocada con su nombre y dos paréntesis que significa

UNA APROXIMACIÓN A LOS ALGORITMOS

que el algoritmo principal no envía parámetros a la subrutina. En este caso, cuando no se envía parámetros, la función puede ser invocada sin paréntesis.

La subrutina

El algoritmo subrutina o función , llamado o invocado es aquel que proporciona servicios a un algoritmo principal. Mirado desde el punto de vista de la jerarquía el algoritmo subrutina es como el “empleado” que está a cargo del algoritmo principal, pues al ser invocado le presta un servicio retornando-le un valor. Su formato general es:

SubProceso variable_de_retorno <- Nombre (Argumentos)

instrucción 1

instrucción 2

instrucción 3

.....

instrucción n

Fin SubProceso

Se muestra la función que ayuda al algoritmo principal mencionado antes:

// *****

// Zona de algoritmo subrutina

// *****

funcion ced=leer_cedula() // Inicio del algoritmo funcion

imprimir sin saltar "Cedula(cero o negativo para fin)" // Computadora pide cedula

UNA APROXIMACIÓN A LOS ALGORITMOS

leer ced // Usuario ingresa cedula

finfuncion // Fin del algoritmo funcion

En la primera línea aparece la asignación *ced=leer_cedula()* donde *ced* es la variable donde retorna valor la función y *leer_cedula()* es el nombre. Nótese además que al retornar el control al programa principal el valor de *ced*(variable de la subrutina) se carga en *cedula*(variable del algoritmo principal), pues la subrutina retorna su valor a través de *ced*.

La o las subrutina(s) se pueden escribir antes o después del algoritmo principal.

En el caso mencionado, el esquema sería:

```
// *****

// Zona de algoritmo subrutina

// *****

funcion ced=leer_cedula() // Inicio del algoritmo funcion

    imprimir sin saltar "Cedula(cero o negativo para fin" // Computadora pide cedula

    leer ced // Usuario ingresa cedula

finfuncion // Fin del algoritmo funcion

// *****

// Zona de algoritmo principal

// *****

proceso c7_01 // Inicio del algoritmo

.....

.....

.....

finproceso // Fin del algoritmo principal
```

El otro esquema podría ser:

UNA APROXIMACIÓN A LOS ALGORITMOS

```
// *****

// Zona de algoritmo principal
// *****

proceso c7_01 // Inicio del algoritmo

.....

.....

.....

finproceso // Fin del algoritmo principal

// *****

// Zona de algoritmo subrutina
// *****

funcion ced=leer_cedula() // Inicio del algoritmo funcion

    imprimir sin saltar "Cedula(cero o negativo para fin" // Computadora pide cedula

    leer ced // Usuario ingresa cedula

finfuncion // Fin del algoritmo funcion
```

Parámetros

Los parámetros constituyen valores que se transfieren desde el algoritmo principal hacia las funciones. Así por ejemplo, podría tener un algoritmo principal que permita ingresar el código de un criterio a evaluar en una selección de talentos, se transfiera el valor numérico del código a una función, y que esta retorne el significado en texto del aquel. El valor transferido es un parámetro, enviado por el algoritmo o modulo principal y recibido por la función o módulo subordinado. Así la siguiente línea de un módulo llamador:

```
criterio_txt=criterio_nombre(criterio_cod) // Se invoca criterio_nombre con el parametro criterio_cod
```

UNA APROXIMACIÓN A LOS ALGORITMOS

se corresponde con la primera línea de la siguiente función:

```
funcion txt=criterio_nombre(cod) // Computadora recibe parametro

segun cod hacer // Segun sea el valor de codigo

1: // Si es 1
    txt = "Conocimiento" // Conocimiento

2: // Si es 2
    txt = "Paciencia y Confianza" // Paciencia y confianza

3: // Si es 3
    txt = "Tolerancia" // Tolerancia

4: // Si es 4
    txt = "Habilidad" // Habilidad

5: // Si es 5
    txt = "Estrategia" // Estrategia

finsegun // Cierre del segun

finfuncion // Computadora retorna valor de txt
```

El marco conceptual expresa que el parámetro enviado se debe corresponder en posición y tipo con el parámetro receptor. En este caso, significa que como *criterio_cod* está en la posición 1 de la invocación, *cod* como parámetro receptor de la subrutina también debe estar en la posición 1. En cuanto a tipo se refiere a que ambos parámetros, el enviado y el receptor, deben ser de tipo o de tipo numérico, o de tipo cadena, o lógico. En este caso ambos son numéricos y cumplen la condición. Veamos un ejemplo de cuando hay más de un parámetro. En el ejemplo un algoritmo principal recibe un valor numérico, obtiene una cadena que corresponde , envía ambos parámetros a una función esta retorna un

UNA APROXIMACIÓN A LOS ALGORITMOS

valor tipo cadena con los parámetros recibidos concatenados, previa conversión del valor numérico a cadena. La invocación:

```
escala_cod y txt=escala_concatenada(escala_cod,escala_txt) // Pasa 2 parametros y recibe concatenacion
```

La función invocada:

```
funcion txt=escala_concatenada(cod,cad) // Computadora recibe parametros  
txt=ConvertirATexto(cod)+" "+cad // Concatena  
finfuncion // Computador retorna valor de txt
```

Como se ve en la invocación, *escala_cod* se corresponde con *cod* de la función en la posición 1 y el tipo numérico. Por otro lado *escala_txt* hace lo propio con *cad* con la posición 2 y el tipo cadena.

Los parámetros que se envían se denominan **parámetros actuales**. Los receptores son **parámetros formales**. Así en:

```
Escala_cod y txt=escala_concatenada(escala_cod,escala_txt) // Pasa 2 parametros y recibe concatenacion
```

escala_cod y *escala_txt* son **parámetros actuales** y en:

```
funcion txt=escala_concatenada(cod,cad) // Computadora recibe parametros
```

cod y *cad* son **parámetros formales**.

Ámbito de los identificadores

El algoritmo llamador y las funciones tienen su propia zona de identificadores en la memoria RAM. Por ejemplo *escala_cod* y *escala_txt* son dos variables diferentes a *cod* y *cad* aunque se transfieran los mismos valores. Puede darse el

UNA APROXIMACIÓN A LOS ALGORITMOS

caso de que una variable con el mismo nombre en diferentes ámbitos sean variables diferentes. Por ejemplo, la variable *nro* de un algoritmo principal y una con el mismo nombre en una función tengan valores diferentes y que en un ámbito se modifique su valor no afecte al otro. La siguiente imagen ilustra al respecto:

```
nro en el algoritmo principal antes de la funcion 10
nro en la funcion antes de la asignacion
nro en la funcion despues de la asignacion 15
nro en el algoritmo principal despues de la funcion 10
*** Ejecución Finalizada. ***
```

El algoritmo

```
// *****

// Zona de algoritmo principal

// *****

proceso c7_04 // Inicio del algoritmo principal

//

//      Objetivo      :      Demuestra ambito de variables
//      Autor          :      Jose Rojas Davalos
//      Fecha          :      30/10/2014
//

// nro: a mostrar

borrar pantalla // Limpia pantalla

nro=10 // Asigna valor en algoritmo principal

imprimir "nro en el algoritmo principal antes de la funcion ",nro

muestra // Invoca funcion

imprimir "nro en el algoritmo principal despues de la funcion ",nro

finproceso // Fin del algoritmo principal

// *****
```

UNA APROXIMACIÓN A LOS ALGORITMOS

```
// Zona de algoritmo subrutina
// *****

// nro: a mostrar

funcion muestra() // Inicio del algoritmo funcion

    imprimir "nro en la funcion antes de la asignacion ",nro

    nro=15

    imprimir "nro en la funcion despues de la asignacion ",nro

finfuncion // Fin del algoritmo funcion
```

Un parámetro formal, también pasa a ser una variable propia de la subrutina y las modificaciones que sufre no se llegan a reflejar en los parámetros actuales que le dieron origen. Solamente con un procedimiento especial, que veremos más adelante, se puede hacer que las variables de un algoritmo principal, sean las mismas de una función, en el sentido de tomar los mismos valores.

Finalmente hay que resaltar que las variables solo existen en sus ámbitos. Esto implica que las variables que existen en una función son aquellas que son pasadas como parámetros o se crean en ella; existen mientras se ejecuta la función, después desaparecen. Lo mismo pasa con el algoritmo principal o llamador. Sus variables solo existen en su ámbito y mientras de ejecuta.

Paso de parámetros por valor o referencia

Antes de entrar en tema, debemos aclarar que las variables, aunque en apariencia para nuestra humana comprensión, se identifican por nombre, realmente están determinadas por la dirección de memoria RAM que ocupan. En términos que

UNA APROXIMACIÓN A LOS ALGORITMOS

podamos comprender la variable **nro** de un algoritmo principal es otra totalmente diferente a la variable **nro** de una función, aunque tengan el mismo nombre. En la ejecución del ejemplo anterior, vemos:

```
nro en el algoritmo principal antes de la funcion 10
```

que se corresponde con estas líneas del algoritmo principal:

```
nro=10 // Asigna valor en algoritmo principal
```

```
imprimir "nro en el algoritmo principal antes de la funcion ",nro
```

Al transferir el control de ejecución a la función:

```
muestra // Invoca funcion
```

Y al intentar exhibir el valor de **nro**, se muestra:

```
nro en la funcion antes de la asignacion
```

La línea del algoritmo que ejecuta la impresión es:

```
imprimir "nro en la funcion antes de la asignacion ",nro
```

Como se nota, en el lugar de la variable **nro** se imprime vacío, ni tan siquiera cero. Esto es porque la variable no existe para la función. Sin embargo, esto cambia cuando se ejecutan las siguientes líneas:

```
nro=15
```

```
imprimir "nro en la funcion despues de la asignacion ",nro
```

Y se muestra:

```
nro en la funcion despues de la asignacion 15
```

UNA APROXIMACIÓN A LOS ALGORITMOS

Al asignarse el valor **15** a la variable **nro** empezó a existir para la función. Prueba de ello es que se pudo mostrar su valor. Entonces podemos concluir que **nro** del algoritmo principal y **nro** de la función son dos variables diferentes, pues ocupan ámbitos diferentes. Haciendo una analogía sería como si existiesen dos personas llamadas **Ana Prieto**. Una de ellas vive en Fernando de la Mora y la otra en Lambaré. Es decir tienen el mismo nombre pero diferentes direcciones. Reforzando esto veamos que pasa en el algoritmo principal cuando retorna de la función:

```
muestra // Invoca funcion y transfiere el control de la ejecución
```

```
imprimir "nro en el algoritmo principal despues de la funcion ",nro // Al retornar de la función ejecuta esta línea
```

El resultado es:

```
nro en el algoritmo principal despues de la funcion 10
```

Como se ve, la variable **nro** muestra el valor **10**, que es el valor que se le asigno en algoritmo principal y no se modificó durante su “estancia” en la función.

Pero, ¿qué pasa si tengo necesidad de transferir valor(es) a la función para que pueda ser usada en ella para un cálculo u otro tipo de operación?. Existen dos casos posibles.

Paso de parámetros por valor

Uno de los casos es cuando necesito que el(los) parámetro(s) enviados estén

UNA APROXIMACIÓN A LOS ALGORITMOS

disponibles para la función pero no sean afectados en su valor original. Es decir se envía una copia de los valores a la función y ella trabaja con las copias sin afectar el valor original. El ejemplo es el del cálculo del área de un triángulo, donde el programa principal envía como parámetros al función los datos **base** y **altura** y una vez que retorna carga el resultado en **area**. Esta es la línea donde se invoca la función:

```
area=superficie(base,altura)// Se invoca superficie(). El resultado se carga en area
```

La función invocada es:

```
funcion area=superficie(base,altura) // Inicio del algoritmo funcion
```

```
area = (base * altura)/2 // Computadora calcula area
```

```
finfuncion // Fin del algoritmo funcion
```

Las variables **base** y **altura** al ser enviadas tienen los valores que se ingresaron por teclado. Al llegar a la función, son declaradas implícitamente como variables pertenecientes a ella, por la primera línea:

```
funcion area=superficie(base,altura) // Inicio del algoritmo funcion
```

Los valores actuales son los mismos que la del algoritmo principal, pues fueron copiados las variables de la subrutina, que a pesar de llevar el mismo nombre no son las mismas por los motivos ya explicados de direcciones de memoria diferentes. Recurriendo a la analogía de **Ana Prieto** de Fernando d ella Mora y **Ana Prieto** de San Lorenzo, podemos decir que por un lado existen **base** y

UNA APROXIMACIÓN A LOS ALGORITMOS

altura del proceso **c3_705**, el algoritmo principal, y **base** y **altura** de la función **superficie()**, la función.

Ejecutemos el algoritmo, por ende la función, y veamos el resultado:

```
Valor de la base(cero o menos para fin)> 7
Valor de la altura0> 3
El triangulo de base 7 y altura 3 tiene como superficie 10.5
Valor de la base(cero o menos para fin)> 8
Valor de la altura3> 2
El triangulo de base 8 y altura 2 tiene como superficie 8
Valor de la base(cero o menos para fin)> 0
*** Ejecución Finalizada. ***
```

Tan solo para demostrar queda las variables, aunque tengan el mismo nombre, son de diferentes ámbitos y los valores solo se copian desde el algoritmo principal hacia la función, agreguemos algunas líneas a la función:

```
funcion area=superficie(base,altura) // Inicio del algoritmo funcion

    area = (base * altura)/2 // Computadora calcula area

    base=89 // Descomentar para demostrar

    altura = 90 // Descomentar para demostrar

    imprimir "Base=",base," Altura=",altura, " valores modificados en la funcion" // Descomentar para
demostrar

finfuncion // Fin del algoritmo funcion
```

El resultado de la ejecución con los mismos valores anteriores:

```
Valor de la base(cero o menos para fin)> 7
Valor de la altura0> 3
Base=89 Altura=90 valores modificados en la funcion
El triangulo de base 7 y altura 3 tiene como superficie 10.5
Valor de la base(cero o menos para fin)> 8
Valor de la altura3> 2
Base=89 Altura=90 valores modificados en la funcion
El triangulo de base 8 y altura 2 tiene como superficie 8
Valor de la base(cero o menos para fin)> 0
*** Ejecución Finalizada. ***
```

Se resalta con el subrayado rojo la modificación de los valores en la función y en

UNA APROXIMACIÓN A LOS ALGORITMOS

azul los valores del algoritmo principal al retornar de la función, notándose que las modificaciones en la subrutina no alteraron los valores del módulo principal.

Con esto se demuestra que la función, utiliza solamente la copia de los valores del algoritmo principal, y por ende se produce el **Paso de parámetros por valor**.

Paso de parámetros por referencia

El otro caso a analizar es cuando se quiere que el algoritmo principal y las funciones se transmitan valores en ambos sentidos. Es decir que ambos módulos, el principal y el subordinado, compartan la misma variable, entendiéndose que la “misma variable” significa no precisamente el mismo nombre(que se puede dar), sino la misma dirección de memoria. Así si usamos la analogía de **Ana Prieto**, nos referimos solamente a la de Fernando de la Mora.

En nuestro caso necesitamos compartir, entre el algoritmo principal y la función los valores de **base** y **altura**. Es decir tenemos un proceso denominado **c3_705** que invoca las funciones **leer_base()** y **leer_altura()**, y que necesita que las variables base y altura sean las mismas en todos los ámbitos. Se muestran las dos invocaciones:

```
leer_base(base_p) // Computadora solicita base y el usuario responde, la primera vez
```

```
leer_altura(altura_p) // Computadora solicita altura y el usuario responde
```

A continuación cada una de las funciones:

```
funcion leer_base(base_s por referencia) // Inicio del algoritmo funcion
```

UNA APROXIMACIÓN A LOS ALGORITMOS

```

imprimir sin saltar "Valor de la base(cero o menos para fin)" // Computadora solicita valor
leer base_s // Usuario ingresa valor

finfuncion // Fin del algoritmo funcion

funcion leer_altura(altura_s por referencia) // Inicio del algoritmo funcion

imprimir sin saltar "Valor de la altura",altura // Computadora solicita valor
leer altura_s // Usuario ingresa valor

finfuncion // Fin del algoritmo funcion
    
```

Es importante resaltar que las variables del módulo principal tienen en su nombre como sufijo la letra **p** mientras la de los módulos llamados usan el sufijo **s**. Esto es con el fin de diferenciar el ámbito. También que aparece la expresión “**por referencia**” después de mencionar el parámetro formal, en la primera línea de las funciones invocadas. Todo esto es para que si a **base_p** y **altura_p** se les asignó cero en el llamador, al transferir el control, sean recibidos en las variables **base_s** y **altura_s**, respectivamente. Mediante cada lectura se cargan **7** y **3** respectivamente y estos valores modifican a **base** y **altura**, tanto en el ámbito de la función como en el algoritmo principal. Esto se comprobará cuando se calcule el área que si funciona el **Paso de parámetros por referencia** debe arrojar un resultado de **10.5**. Según la secuencia, el cálculo se hace después de haber invocado las funciones de lectura, con lo que se demuestra que los valores son compartidos entre todos los módulos. Otro hecho destacable es que las variables **base_p** y **base_s**, son las mismas aunque tengan distintos nombres, comparten la

UNA APROXIMACIÓN A LOS ALGORITMOS

misma dirección de memoria RAM. Lo mismo pasa con **altura_p** y **altura_s**. Une ejemplo de la ejecución:

```
Valor de la base(cero o menos para fin)> 7
Valor de la altura> 3
El triangulo de base 7 y altura 3 tiene como superficie 10.5
Valor de la base(cero o menos para fin)> 0
*** Ejecución Finalizada. ***
```

Tipos de funciones

De acuerdo al valor que retornan

Las funciones pueden ser consideradas de tipo numérico, cadena o lógico.

Tipo numérico

Así la función mostrada a continuación es de tipo numérico:

```
funcion ced=leer_cedula() // Inicio del algoritmo funcion es de tipo numérico al ser ced una variable
numérica.
```

Tipo cadena

Sin embargo la función:

```
funcion txt=criterio_nombre(cod) // Computadora recibe parametro
```

es de tipo cadena al retornar un valor del mencionado tipo.

Tipo lógico

Por su parte la subrutina:

```
funcion cod_ok=criterio_ok(cod) // Computadora recibe parametro
cod_ok=(cod>=1 & cod<=5) // Verifica que cod este en el rango y carga verdadero o falso en la
variable
```

UNA APROXIMACIÓN A LOS ALGORITMOS

```
finfuncion // Computador retorna valor de cod_ok
```

es de tipo lógico.

Las funciones que retornan o no valor

También el hecho de que las **funciones retornan o no valor** genera dos tipos. Para comprender estas dos posibilidades hay que entender que se quiere decir con **retornar valor**.

Funciones que retornan valor

Se dice que una **función retorna valor**, cuando se asocia el resultado de la función a una variable. Por ejemplo:

```
funcion area=superficie(base,altura) // Inicio del algoritmo funcion  
  
    area = (base * altura)/2 // Computadora calcula area  
  
finfuncion // Fin del algoritmo funcion
```

En este caso la función, denominada **superficie()** retorna valor mediante la variable **area**.

Funciones que no retornan valor

Se considera que una **función no retorna valor** cuando el resultado de la función no está asociada a variable alguna, como se muestra a continuación:

```
funcion leer_base(base_s por referencia) // Inicio del algoritmo funcion  
  
    imprimir sin saltar "Valor de la base(cero o menos para fin)" // Computadora solicita valor  
  
    leer base_s // Usuario ingresa valor  
  
finfuncion // Fin del algoritmo funcion
```

UNA APROXIMACIÓN A LOS ALGORITMOS

El resultado de la función `leer_base()` no está asociada a ninguna variable.

Ejercicios propuestos

Preparar el algoritmo que:

1. Lea por teclado un número entero y verifique si es par o no. La verificación debe realizarse en una subrutina.
2. Permita leer varios nombres por teclado. Cada nombre ingresado debe pasar por el siguiente proceso:
 - a) La primera letra se debe convertir a mayúsculas
 - b) Las demás letras se debe convertir a minúsculas.

El proceso de cada nombre se debe realizar en una subrutina y para ello el programa principal debe pasar como parámetro a la subrutina, el nombre ingresado, y aquella(la subrutina), debe retornar el valor recibido, convertido según las condiciones precedente-mente establecidas.

Se deber imprimir el nombre, antes y después de la conversión y la lectura termina cuando se ingresa “/*” en las dos primeras posiciones de la cadena leída.

3. Invoque una subrutina que analice un texto ingresado por teclado y retorne la cantidad de vocales que contiene. Se debe procesar t textos.
4. Invoque una subrutina que analice un texto ingresado por teclado y retorne

UNA APROXIMACIÓN A LOS ALGORITMOS

la cantidad de consonantes que contiene. Se debe procesar t textos.

5. Invoque una función que retorne un texto pasado como parámetro en forma inversa. Así si el texto ingresado es “hola”, la función debe retornar, “aloh”. El proceso se debe repetir mientras el texto ingresado tenga longitud mayor a uno.
6. Permita invocar una función de sorteo de una rifa. El programa principal debe pedir el numero mas alto de las boletas vendidas, que debe ser mayor a cero, y emitir el mensaje correspondiente a los números ganadores de los 10 primeros premios.

UNA APROXIMACIÓN A LOS ALGORITMOS

CAPÍTULO 8: ARREGLOS

Introducción

Un arreglo es un conjunto finito y ordenado de elementos homogéneos. La propiedad ordenado significa que el elemento primero, segundo, tercero...n-ésimo de un arreglo puede ser identificado y son homogéneos, es decir, del mismo tipo de datos. Un arreglo puede estar compuesto por elementos tipo cadena, lógico o numérico.

Los arreglos son genéricamente conocidos como matrices, que cuando tienen una sola dimensión son conocidos como **matriz unidimensional o vector**. También existen los arreglos de más de una dimensión en cuyo se los denomina como **matriz multidimensional o simplemente matriz**. La más usada es la **matriz bidimensional** o de dos dimensiones .

En este texto tomaremos las siguientes convenciones:

- Usaremos matrices de hasta dos dimensiones.
- Denominaremos **vector** a la matriz unidimensional.
- Distinguiremos como **matriz** al arreglo bidimensional.

Vectores

Un ejemplo de vector:

a[1] a[2] a[.....] a[i] a[n]

UNA APROXIMACIÓN A LOS ALGORITMOS

100	90	50	70
-----	----	-------	----	-------	----

En el ejemplo, **a** es el nombre del vector y cada elemento está identificado por el nombre del vector y un número entero entre corchetes indicando el subíndice, índice o posición de cada elemento. Un vector representa a un conjunto de variables simples, que ocupan espacios contiguos de memoria, denominadas con el mismo nombre y diferenciadas por el subíndice. Como debe ser homogéneo debe guardar datos del mismo tipo para cada elemento. En este caso cada elemento debe ser numérico, pues guarda datos numéricos. Se dice, entonces, que el vector **a** es **numérico**.

Así **a[1]=100** implica que el valor **100** está guardado en la posición **1** del vector **a** o dicho de otra forma la variable simple **a[1]** contiene el valor **100**. Un vector tiene una dimensión que puede ser **fija** o **variable**. En nuestro ejemplo el vector tiene dimensión **n** o sea **variable**. La dimensión **fija** se da cuando se sabe exactamente cuantos elementos tendrá el vector. El hecho que un vector sea de dimensión **fija** o **variable** da lugar a dos tipos: los estáticos con dimensión **fija** y los dinámicos con dimensión **variable** también conocidos como vectores **abiertos**.

Dimensión de un vector

Dimensión de un vector estático o de dimensión fija

UNA APROXIMACIÓN A LOS ALGORITMOS

Para declarar un vector numérico de tamaño o dimensión (5) cinco se escribe:

```
dimension a[5] // Dimensiona vector de tamaño 5
```

Es un vector estático pues la dimensión se establece a través de un valor constante.

Dimensión de un vector dinámico o de dimensión variable

Para declarar un vector numérico de dimensión **n**, se debe realizar la siguiente secuencia de pasos:

```
imprimir sin saltar "Ingrese dimensión del vector" // Computadora solicita cantidad de elementos del vector
```

```
leer n // Usuario ingresa cantidad de elementos del vector
```

```
dimension a[n] // Dimensiona vector de tamaño n
```

Es un vector dinámico pues la dimensión se establece a través de una variable, cuyo valor se lee por teclado.

Lectura de un vector

Estático

Se debe ingresar el valor para cada elemento del vector por medio de un ciclo y con un mensaje amigable para el usuario final:

```
para k=1 hasta 5 // Repite cinco veces
```

```
imprimir sin saltar "Valor para a[" ,k, "]" // Computadora solicita valor del elemento del vector
```

```
leer a[k] // Usuario ingresa valor del elemento del vector
```

```
finpara // Sale del ciclo
```

Una muestra de la ejecución de la lectura:

UNA APROXIMACIÓN A LOS ALGORITMOS

```
Valor para a[1]> 8
Valor para a[2]> 5
Valor para a[3]> 3
Valor para a[4]> 4
Valor para a[5]> 9
```

Dinámico

El proceso es casi idéntico a la lectura de vectores estáticos. La diferencia radica a que se debe ingresar la dimensión del vector que indica cuantos elementos se debe leer. El ejemplo:

// Lectura de valores

para k=1 hasta n // Repite n veces

imprimir sin saltar "Valor para a["k,k"]" // Computadora solicita valor del elemento del vector

leer a[k] // Usuario ingresa valor del elemento del vector

finpara // Sale del ciclo

La porción de código que acabamos de ver, necesita de las siguientes líneas previas:

imprimir sin saltar "Ingrese dimensión del vector" // Computadora solicita cantidad de elementos del vector

leer n // Usuario ingresa cantidad de elementos del vector

dimension a[n] // Dimensiona vector de tamaño n

Una muestra de la ejecución de la lectura:

```
Ingrese dimensión del vector> 4
Valor para a[1]> 49
Valor para a[2]> 37
Valor para a[3]> 59
Valor para a[4]> 62
```

Proceso de un vector

Normalmente para procesar un vector hay que recorrerlo elemento a elemento

UNA APROXIMACIÓN A LOS ALGORITMOS

como en el siguiente ejemplo:

```
para k=1 hasta n // Repite n veces
    a[k]=a[k]*2 // Computadora multiplica por 2 cada valor del elemento
finpara // Sale del ciclo
```

Impresión de un vector

Estático

```
imprimir "El vector resultante es: "

para k=1 hasta 5 // Repite cinco veces
    imprimir sin saltar a[k], " " // Computadora muestra el valor de cada elemento
finpara // Sale del ciclo

imprimir " " // Salto de linea
```

Dinámico

```
imprimir "El vector resultante es: "

para k=1 hasta n // Repite n veces
    imprimir sin saltar a[k], " " // Computadora muestra el valor de cada elemento
finpara // Sale del ciclo

imprimir " " // Computadora salta a siguiente linea de impresion
```

La búsqueda del mayor en un vector

La estructura de datos llamada vector facilita la búsqueda del mayor valor en ella.

A continuación la parte de código donde se realiza la búsqueda del mayor y su posición dentro del vector:

```
// Proceso

mayor=a[1] // Guarda el primer valor como mayor
```

UNA APROXIMACIÓN A LOS ALGORITMOS

```

p=1 // Asume 1 como posicion del mayor
para k=2 hasta n // Recorre desde posicion 2 para comparar
    si a[k]>=mayor entonces // Si el elemento actual es mayor o igual ...
        mayor=a[k] // Nuevo mayor
        p=k // Posicion del nuevo mayor
    fin si // Fin del si
finpara // Sale del ciclo
    
```

El algoritmo empleado es el mismo que habíamos usado cuando desarrollamos estructuras repetitivas. Se guarda el primer elemento como mayor, al igual que su posición, y se recorre con un ciclo desde el segundo elemento, comparando siempre contra el mayor actual. Si el elemento que aparece es mayor que el mayor guardado se procede al cambio almacenando el nuevo mayor y la nueva posición. El operador de relación en la comparación es de mayor o igual, para que en caso que el mayor valor aparezca más de una vez se guarde el último que aparece, junto con su posición. Esto es, si el mayor valor es “Margarita” y aparece en las posiciones 2 y 4, se guarda el de la posición 4 como el mayor. A continuación un ejemplo de la ejecución del algoritmo:

```

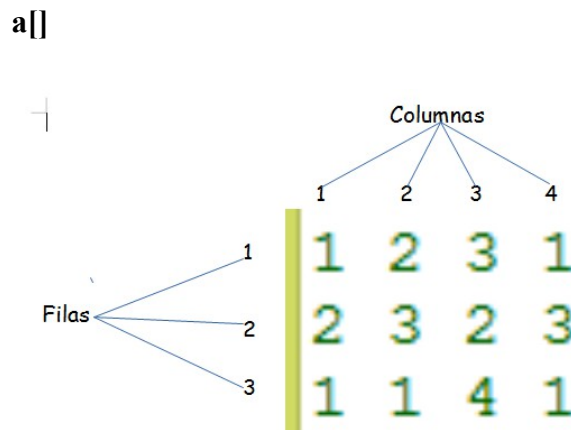
Dimension del vector> 5
Valor para a[1]> Gricelda
Valor para a[2]> Margarita
Valor para a[3]> Ana
Valor para a[4]> Margarita
Valor para a[5]> Clara
El mayor valor es Margarita y ocupa la posicion 4
    
```

Matrices

Un ejemplo de matriz, que suponiendo que el arreglo es de orden 3 x 4, podría

UNA APROXIMACIÓN A LOS ALGORITMOS

verse como se muestra:



En el ejemplo, **a** es el nombre de la matriz y cada elemento está identificado por el nombre de la matriz y dos números enteros entre corchetes separados por coma indicando los subíndices, índices o la posición de cada elemento en forma de coordenadas. Es decir un elemento de una matriz está identificado por una posición fila y otra posición columna, en ese orden. Por ejemplo, leyendo la matriz sabemos que el elemento **a[3,2]** contiene el valor **1**.

Una matriz representa a un conjunto de variables simples que ocupan espacios contiguos de memoria, denominadas con el mismo nombre y diferenciadas por los subíndices. Como debe ser homogéneo debe guardar datos del mismo tipo (numérico, lógico o cadena), para cada elemento. En este caso cada elemento debe almacenar números y se dice, entonces, que la matriz **a** es **numérica**.

Resumiendo:

UNA APROXIMACIÓN A LOS ALGORITMOS

- Una matriz es un arreglo **bidimensional** que guarda datos en c/u de sus elementos.
- Una matriz es una disposición tabular en **filas** y **columnas**.
- Los elementos se identifican por con el nombre de la matriz, entre corchetes la posición fila y la posición columna como en **a[3,2]**.
- La matriz del ejemplo tiene 3 **filas** y 4 **columnas**.
- Se puede decir que la matriz es de **dimensión o de orden 3x4**.
- Cada elemento de la matriz es de tipo numérico, por lo tanto la matriz es del mismo tipo.

El hecho de una matriz sea de dimensión **fija** o **variable** da lugar a dos tipos: las estáticas con dimensiones **fijas** y las dinámicas con dimensión **variable** también conocidos como matrices **abiertas**.

Dimensión de una matriz

Dimensión de una matriz estática o de dimensión fija

Por ejemplo, para dimensionar una matriz numérica de **3 filas** y **4 columnas** escribiremos:

dimension a[3,4] // Dimensiona matriz a con 3 filas y 4 columnas

Dimensión de una matriz dinámica o de dimensión variable

Primero se debe obtener las dimensiones , en el ejemplo, mediante las siguientes

UNA APROXIMACIÓN A LOS ALGORITMOS

líneas:

```
imprimir sin saltar "Ingrese dimensión fila de la matriz" // Computadora solicita valor de m

leer m // Usuario ingresa valor de m

imprimir sin saltar "Ingrese dimensión columna de la matriz" // Computadora solicita valor de n

leer n // Usuario ingresa valor de n
```

Luego, el dimensionamiento:

```
dimension a[m,n] // Dimensiona matriz a con m filas y n columnas
```

Lectura de una matriz

Estática

Se ingresan los datos para la matriz por medio de ciclos anidados:

```
// Lectura de valores

para k=1 hasta 3 // Ciclo para filas
    para l=1 hasta 4 // Ciclo para columnas
        imprimir sin saltar "Valor para a["k","l"]" // Solicita valor del elemento
        leer a[k,l] // Lee valor del elemento
    finpara // Sale del ciclo
finpara // Sale del ciclo
```

Parte de la ejecución se vería así:

```
Valor para a[1,1]> 44
Valor para a[1,2]> 22
Valor para a[1,3]> 11
Valor para a[1,4]> 99
Valor para a[2,1]> 22
Valor para a[2,2]> 12
Valor para a[2,3]> 11
Valor para a[2,4]> 22
Valor para a[3,1]> 33
Valor para a[3,2]> 20
Valor para a[3,3]> 30
Valor para a[3,4]> 40
```

Dinámica

UNA APROXIMACIÓN A LOS ALGORITMOS

El proceso es casi idéntico a la lectura de matrices estáticas. La diferencia radica a que se debe ingresar las dimensiones de la matriz que me indica cuantos elementos debe leer. Por supuesto también se debe usar ciclos para leer los elementos de una matriz, como se ve en la siguiente porción de código:

```
// Lectura de valores

para k=1 hasta m // Ciclo para filas

    para l=1 hasta n // Ciclo para columnas

        imprimir sin saltar "Nombre para a["k","l"]" // Solicita valor del elemento

        leer a[k,l] // Lee valor del elemento

    finpara // Sale del ciclo

finpara // Sale del ciclo
```

Una muestra del funcionamiento sería:

```
Ingrese dimensión fila de la matriz> 2
Ingrese dimensión columna de la matriz> 3
Nombre para a[1,1]> Marga
Nombre para a[1,2]> Clara
Nombre para a[1,3]> Gricelda
Nombre para a[2,1]> Ana
Nombre para a[2,2]> Longina
Nombre para a[2,3]> Nidia
```

Proceso de una matriz

Normalmente para procesar una matriz hay que recorrerlo elemento a elemento como en el siguiente ejemplo con una **matriz de dimensión fija**:

```
// Proceso

para k=1 hasta 3 // Ciclo para filas

    para l=1 hasta 4 // Ciclo para columnas
```

UNA APROXIMACIÓN A LOS ALGORITMOS

```
a[k,l]=a[k,l]-1 // Resta uno a cada elemento
```

```
finpara // Sale del ciclo
```

```
finpara // Sale del ciclo
```

Otro ejemplo con una matriz de dimensión variable:

```
// Proceso
```

```
para k=1 hasta m // Ciclo para filas
```

```
para l=1 hasta n // Ciclo para columnas
```

```
a[k,l]="@"+a[k,l]+"@" // Concatena @ al comienzo y al final
```

```
finpara // Sale del ciclo
```

```
finpara // Sale del ciclo
```

Impresión de una matriz

Estática

```
para k=1 hasta 3 // Ciclo para filas
```

```
para l=1 hasta 4 // Ciclo para columnas
```

```
imprimir sin saltar a[k,l], " " // Imprime valor de cada elemento
```

```
fin para // Sale del ciclo
```

```
imprimir " " // Salto de linea
```

```
finpara // Sale del ciclo
```

Una vista de su ejecución:

```
Los elementos de la matriz resultante son:
32 11 76 41
77 89 10 85
21 98 29 23
```

Dinámica

```
// Resultados
```

```
imprimir "Los elementos de la matriz resultante son:" // Mensaje de encabezado
```

UNA APROXIMACIÓN A LOS ALGORITMOS

```

para k=1 hasta m // Ciclo para filas
    para l=1 hasta n // Ciclo para columnas
        imprimir sin saltar a[k,l], " " // Imprime valor de cada elemento
    fin para // Sale del ciclo
    imprimir " " // Salto de linea
finpara // Sale del ciclo
    
```

La ejecución se presenta de la siguiente manera:

```

Los elementos de la matriz resultante son:
@juan@ @mara@
@pepe@ @toia@
    
```

La matriz cuadrada

Se conoce como matriz cuadrada a aquella que tiene la dimensión fila igual a la dimensión columna. Así si se dice que una matriz es de orden 4 se entiende que tiene 4 filas y 4 columnas. Si se expresa la dimensión de una matriz como $n \times n$ (ene por ene) también se entiende que es cuadrada. En el primer caso es con dimensiones fijas, en el segundo con dimensiones variables. Como las dimensiones son iguales se puede expresar como matriz de dimensión u orden 4 en el primer caso, o como matriz de dimensión u orden n , en el segundo. Supongamos que la siguiente matriz, sea la representación de un aula y que los ceros en los elementos indican ausencia de sillas en esas coordenadas, mientras que los unos guardados en los elementos indican que hay sillas en esos lugares:

UNA APROXIMACIÓN A LOS ALGORITMOS

1	1	0	1
1	1	1	0
1	0	0	1
0	1	1	1

La diagonal principal

Elementos situados en la diagonal principal

Tomemos los valores siguientes: $\{1,1,0,1\}$. Suponiendo que nuestra matriz se denomina **aula[]** serían los elementos `aula[1,1]`, `aula[2,2]`, `aula[3,3]` y `aula[4,4]`. ¿Qué tienen de común estos elementos? Sus coordenadas o subíndices son iguales desde el punto de vista posición fila y posición columna. En este caso se dice que estos elementos están en la **diagonal principal de la matriz**.

Elementos situados por encima de la diagonal principal

Si ahora tomamos los siguientes valores: $\{1,0,1,1,0,1\}$. Vemos que se guardan en los elementos `aula[1,2]`, `aula[1,3]`, `aula[1,4]`, `aula[2,3]`, `aula[2,4]` y `aula[3,4]`. ¿Qué tienen de común estos elementos? Sus coordenadas filas son menores que sus coordenadas columnas. Se dice que estos elementos están **por encima de la diagonal principal**.

Elementos situados por debajo de la diagonal principal

Finalmente, si tenemos en cuenta los valores de los elementos : $\{1,1,0,0,1,1\}$. Vemos que se guardan en los elementos `aula[2,1]`, `aula[3,1]`, `aula[3,2]`, `aula[4,1]`,

UNA APROXIMACIÓN A LOS ALGORITMOS

aula[4,2] y aula[4,3]. ¿Qué tienen de común estos elementos? Sus coordenadas filas son mayores que sus coordenadas columnas. Se dice que estos elementos están **por debajo de la diagonal principal**.

A continuación ejemplos de selección de elementos de una matriz cuadrada según su relación con la diagonal principal. La parte del algoritmo que se exhibe, muestra las coordenadas de los elementos con valor 1 que se encuentran **en la diagonal principal**:

```
// Resultados

imprimir "Coordenadas en diagonal principal donde existen sillas:" // Encabezado

para k=1 hasta fc // Se usa un solo ciclo, pues cuando una matriz es cuadrada sus coordenadas fila y columna ...

    si aula[k,k]==1 // Si tiene valor 1 en la diagonal principal

        imprimir sin saltar "[" ,k, ", ",k, "]" " // Existe la silla e imprime coordenadas

    fin si // Sale del si

finpara // Sale del ciclo

imprimir " " // Salto de linea
```

La ejecución se ve así:

```
Tamaño(orden) del aula:4
La posición 1,1, ¿tiene silla(0=no,1=si)? 1
La posición 1,2, ¿tiene silla(0=no,1=si)? 1
La posición 1,3, ¿tiene silla(0=no,1=si)? 0
La posición 1,4, ¿tiene silla(0=no,1=si)? 1
La posición 2,1, ¿tiene silla(0=no,1=si)? 1
La posición 2,2, ¿tiene silla(0=no,1=si)? 1
La posición 2,3, ¿tiene silla(0=no,1=si)? 1
La posición 2,4, ¿tiene silla(0=no,1=si)? 0
La posición 3,1, ¿tiene silla(0=no,1=si)? 1
La posición 3,2, ¿tiene silla(0=no,1=si)? 0
La posición 3,3, ¿tiene silla(0=no,1=si)? 0
La posición 3,4, ¿tiene silla(0=no,1=si)? 1
La posición 4,1, ¿tiene silla(0=no,1=si)? 0
La posición 4,2, ¿tiene silla(0=no,1=si)? 1
La posición 4,3, ¿tiene silla(0=no,1=si)? 1
La posición 4,4, ¿tiene silla(0=no,1=si)? 1
Coordenadas donde existen sillas:
[1,1] [2,2] [4,4]
```

Las siguientes líneas de código realizan la suma de las edades de los alumnos

UNA APROXIMACIÓN A LOS ALGORITMOS

guardadas en una matriz cuadrada, seleccionando los elementos **por debajo de la diagonal principal**:

```
// Proceso

para k=1 hasta r // Ciclo para filas

    para l=1 hasta r // Ciclo para columnas

        si(k>l) // El elemento esta por debajo de la diagonal principal

            se=se+alu[k,l]// Incrementa el sumador de edades

        fin si // Sale del si

    finpara // Sale del ciclo

finpara // Sale del ciclo
```

El gráfico muestra la ejecución del algoritmo:

```
Dimension de la matriz cuadrada> 3
Ingresar edad del alumno [1,1]> 20
Ingresar edad del alumno [1,2]> 18
Ingresar edad del alumno [1,3]> 19
Ingresar edad del alumno [2,1]> 18
Ingresar edad del alumno [2,2]> 19
Ingresar edad del alumno [2,3]> 20
Ingresar edad del alumno [3,1]> 17
Ingresar edad del alumno [3,2]> 18
Ingresar edad del alumno [3,3]> 19
La suma de las edades situadas por debajo de la diagonal principal es 53
... ..
```

Otra zona de una matriz cuadrada es la ocupada por los elementos de que están por encima de la diagonal principal. La parte del algoritmo mostrada, cuenta y suma las edades de los alumnos guardadas en una matriz cuadrada, seleccionando **los elementos por encima de la diagonal principal**:

```
para k=1 hasta r // Ciclo para filas

    para l=1 hasta r // Ciclo para columnas

        si(k<l) // El elemento esta por encima de la diagonal principal

            se=se+alu[k,l]// Incrementa el sumador de edades
```

UNA APROXIMACIÓN A LOS ALGORITMOS

```

ce=ce+1 // Incrementa el contador de edades
finsi // Sale del si
finpara // Sale del ciclo
finpara // Sale del ciclo

```

Se muestra una instantánea de la ejecución del algoritmo:

```

Dimension de la matriz cuadrada> 3
Ingresar edad del alumno [1,1]> 20
Ingresar edad del alumno [1,2]> 10
Ingresar edad del alumno [1,3]> 12
Ingresar edad del alumno [2,1]> 18
Ingresar edad del alumno [2,2]> 20
Ingresar edad del alumno [2,3]> 18
Ingresar edad del alumno [3,1]> 17
Ingresar edad del alumno [3,2]> 16
Ingresar edad del alumno [3,3]> 18
40 El promedio de las edades situadas por encima de la diagonal principal es 13.3333333333
*** Ejecución Finalizada ***

```

La diagonal secundaria

Suponiendo la siguiente matriz, sea la representación de un aula y que una “N” en los elementos indican ausencia de sillas en esas coordenadas, mientras que una “S” guardada en los elementos indican que hay sillas en esos lugares:

S	S	N	S
S	S	S	N
S	N	N	S
N	S	S	S

Los valores de los elementos marcados son: {“S”, “S”, “N”, “N”} que se guardan en aula[1,4], aula[2,3], aula[3,2] y aula[4,1]. ¿Qué tienen de común estos elementos?

Si nos fijamos con atención en las coordenadas y en el orden en que fueron citados notaremos que las coordenadas de las filas van de menor a mayor y las

UNA APROXIMACIÓN A LOS ALGORITMOS

coordenadas de las columnas van de mayor a menor. Por lo tanto con un solo ciclo de 1 a 4 puedo generar el valor de las filas y tengo que buscar por medio de algún truco de cálculo generar en paralelo el valor para los columnas. Descubra el truco usado en la parte del algoritmo que se lista a continuación, para recorrer la diagonal secundaria con un solo ciclo y analizar si los elementos contienen “N” o “S”:

```

imprimir "Coordenadas de la diagonal secundaria donde existen sillas:" // Encabezado

l=fc // El contador de columnas se inicializa con fc, pues ira en orden inverso al contador de filas ...

para k=1 hasta fc // Repetir fc veces

    si aula[k,l]=="S" // Si tiene valor "S"

        imprimir sin saltar "[" ,k, ", ",l, "]" // Existe la silla e imprime coordenadas

    fin si // Sale del si

    l=l-1 // Genera columna

finpara // Sala del ciclo

imprimir " " // Salto de linea
    
```

A continuación, una muestra de la ejecución:

```

Tamaño(orden) del aula> 4
La posición 1,1 ¿tiene silla (n=no,s=si)? > s
La posición 1,2 ¿tiene silla (n=no,s=si)? > s
La posición 1,3 ¿tiene silla (n=no,s=si)? > n
La posición 1,4 ¿tiene silla (n=no,s=si)? > s
La posición 2,1 ¿tiene silla (n=no,s=si)? > s
La posición 2,2 ¿tiene silla (n=no,s=si)? > s
La posición 2,3 ¿tiene silla (n=no,s=si)? > s
La posición 2,4 ¿tiene silla (n=no,s=si)? > n
La posición 3,1 ¿tiene silla (n=no,s=si)? > s
La posición 3,2 ¿tiene silla (n=no,s=si)? > n
La posición 3,3 ¿tiene silla (n=no,s=si)? > n
La posición 3,4 ¿tiene silla (n=no,s=si)? > s
La posición 4,1 ¿tiene silla (n=no,s=si)? > n
La posición 4,2 ¿tiene silla (n=no,s=si)? > s
La posición 4,3 ¿tiene silla (n=no,s=si)? > s
La posición 4,4 ¿tiene silla (n=no,s=si)? > s
Coordenadas de la diagonal secundaria donde existen sillas:
[1,4] [2,3]
    
```

UNA APROXIMACIÓN A LOS ALGORITMOS

Ejercicios propuestos

Preparar el algoritmo que:

1. Lea dos vectores de dimensión d conteniendo lista de nombres y los convierta a una sola lista utilizando un tercer vector. Imprimir el vector resultante.
2. Lea en un vector los nombres de clientes de un banco y en otro el saldo de cada cliente. Imprimir la lista de clientes con saldo rojo. La cantidad de clientes está dada por la variable c .
3. Cargue en un vector numérico la cosecha en hectáreas de d productores. De imprimir como resultado la cantidad de mayor cosecha, la de menor cosecha y el rango entre las cantidades, sabiendo que rango es mayor – menor.
4. Cargue en un vector el nombre de los departamentos del Paraguay y en otro la cantidad anual de lluvia caída en cada departamento, e imprima la lista con nombre de departamento y cantidad anual de lluvia caída, ordenada numérica y ascendente-mente por cantidad anual de lluvia caída.
5. Lea un vector numérico de m posiciones y verifique si cada elemento es divisible o no por el valor q , cargando en cada elemento de un vector auxiliar tipo lógico en la posición que le corresponde verdadero si no es

UNA APROXIMACIÓN A LOS ALGORITMOS

divisible y falso si es . Posteriormente al proceso se debe recorrer el vector tipo lógico e imprimir el mensaje apropiado a la situación, mencionando el elemento numérico analizado. El análisis de si un numero es divisible o no debe realizarse en una subrutina.

6. Cargue en una matriz de dimensión 3x3 los números comprendidos entre [1,9] en orden descendente. No se debe ingresar datos por teclado ni usar arreglos con valores constante. Imprimir la matriz generada.
7. Cree una representación plana de una sección de un cristal de cloruro de sodio usando una matriz de 4x4. Para ello cargue la matriz de la siguiente forma:
 - Si la fila es par: llenar las columnas pares con iones sodio y las columnas impares con iones cloruro.
 - Si la fila es impar: llenar las columnas pares con iones cloruro y las columnas impares con iones sodio.
8. Realice el ejercicio anterior para un cristal de yoduro de potasio utilizando una matriz mxm.
9. Cargue e imprima en una matriz de 3x4 las 3 reacciones principales del proceso Ostwald para la obtención del ácido nítrico a partir del amoníaco.
10. Calcule el determinante de una matriz de orden 3x3 utilizando la Regla de

UNA APROXIMACIÓN A LOS ALGORITMOS

Sarrus. Imprima la matriz ingresada y su determinante.

11. Genere una matriz de orden n con el texto “IA” en la diagonal principal, “IQ” en la diagonal secundaria y asteriscos en las demás posiciones. Imprimir la matriz generada en un proceso aparte. El único valor que debe ser ingresado por teclado es el de n .
12. Pida introducir los datos para una matriz de orden x , luego imprima el mayor valor, el menor valor y el rango de los datos introducidos en la matriz.
13. Cargue la letra “u” en los elementos de la matriz por encima de la diagonal principal, “d” en los elementos de la matriz por debajo de la diagonal principal y “c” en los elementos de la diagonal principal. El único valor que se debe introducir es el orden de la matriz. Imprimir la matriz resultante.
14. Permita determinar la cantidad de habitantes que viven en tres zonas de una ciudad. La ciudad se representa con matriz de orden o y cada elemento representa a una manzana. El valor que se carga en cada elemento de la matriz es la cantidad de habitantes por manzana. Las tres zonas son: la central, situada en la diagonal principal, la zona alta por encima de la diagonal principal y la zona baja, por debajo de la diagonal principal.

UNA APROXIMACIÓN A LOS ALGORITMOS

15. Sabiendo que el servicio de comedor de un restaurante es atendido por 3 mozos que sirven v diferentes variedades de comidas, permita elaborar un informe, relacionando la cantidad de cada variedad de comida servida con cada uno de los mozos según el siguiente formato

Mozo	1	2	3	Total
Variedad comida				
1				
2				
....				
v				
Total				

Bibliografía

Joyanes Aguilar, Luis. (1990). Fundamentos de Programación. Algoritmos y Estructura de Datos. Madrid, España: McGRAW-HILL/INTERAMERICANA DE ESPAÑA, S.A.

Silvero, Juan. (1999). Introducción al lenguaje SL. Asunción, Paraguay: Centro Nacional de Computación(CNC) de la Universidad Nacional de Asunción(UNA).

PseInt. (2014). *Pablo Novara . El Pseudocódigo*. Recuperado el 15 de octubre de

UNA APROXIMACIÓN A LOS ALGORITMOS

2014 de <http://pseint.sourceforge.net/index.php?page=pseudocodigo.php>

Algoritmo. (2015). En Wikipedia, la enciclopedia libre. Recuperado el 20 de febrero de 2015 de <http://es.wikipedia.org/wiki/Algoritmo>.

Rodríguez Almeida, Miguel Ángel. (1991). METODOLOGÍA DE LA PROGRAMACIÓN. A través de Pseudocódigo . Madrid, España: McGRAW-HILL/INTERAMERICANA DE ESPAÑA, S.A.

Correa Uribe, Guillermo. (1992). DESARROLLO DE ALGORITMOS Y SUS APLICACIONES EN BASIC, PASCAL, COBOL Y C .Santafé de Bogota, Colombia: McGRAW-HILL/INTERAMERICANA DE ESPAÑA, S.A.

Licencia o derechos de uso

Este trabajo se libera bajo el tipo de licencia **PP**, de la expresión guaraní "**Pejapo Pejaposeva**" y en castellano sería, "**Hagan Lo Que Quieran**"(HLQQ). Con una licencia **PP** ya saben que pueden copiar "in extenso", duplicar, reutilizar, adaptar, re-mezclar y redistribuir, sin rubores ni remordimientos. Para mí, un docente que no comparte sus conocimientos, obras u otros recursos y servicios, y piensa solo en lo que le "costó" adquirir y como "recuperar" eso, no merece la distinción de llamarse tal. Con eso no digo que se muera de hambre y no cobre por sus servicios. Pero si empieza a (pensar en) lucrar, ahí ya no nos entendemos.