SINTAXIS Y SEMÁNTICA DE LOS LENGUAJES
2024

TRABAJO PRÁCTICO Nº 2

GRUPO Nº 3

Lenguajes

PHP
Pascal

Integrantes

Guido Solano
Nicolas Fernandez
Sebastian Brusco
Benjamin Soto
Ailén Tamara Olha

SSL 2024

Sintaxis y Semántica de los lenguajes

Trabajo Práctico N°2

Investigar dos lenguajes de programación excluidos: C y C++, buscar las BNF del lenguaje elegido y desarrollar una presentación sobre los siguientes temas:

- Breve historia del lenguaje ("BREVE")
- Comparación centrada en la forma en que los lenguajes manejan LAS FUNCIONES DE ORDENAMIENTO(SORT) con un ejemplo comparando el rendimiento.

La presentación(ppt o similar) debe ser un resumen con las ideas principales, este trabajo se aprueba con la presentación del mismo de manera presencial, todos los integrantes deben exponer.

Pueden agregar código y ejemplos prácticos. La presentación debe finalizar con un breve set de preguntas para sus compañeros.

La entrega debe constar de: la BNF, el entregable completo, la presentación (ppt) resumen y los ejemplos en caso de haberlos.

IMPORTANTE! La presentación debe ser de 15 minutos como máximo, no se debe superar ese plazo, se evaluará este objetivo.

Los pares de lenguajes no deben repetirse en el curso, queda a elección de ustedes cuales usar pero no deben haber dos iguales, la coordinación está a cargo de los alumnos.

LOS TRABAJOS SE DEBEN SUBIR A GIT.

Fecha de entrega: 9/8, fecha de presentación 9 y 16 de agosto, el orden de presentación lo deciden los alumnos pero la lista debe entregarse el día 9 en clase.

PHP (Hypertext Preprocessor)

Historia de PHP

PHP es un lenguaje de programación de código abierto, ampliamente utilizado para el desarrollo web, y que ha experimentado una evolución significativa desde su creación en 1994.

Inicios: PHP/FI
PHP fue creado por Rasmus Lerdorf en 1994 como un conjunto de scripts en C llamado "Personal Home Page Tools" (PHP Tools). Originalmente, estos scripts se utilizaban para rastrear visitas a su currículum online. A medida que sus funcionalidades se expandieron, Lerdorf reescribió PHP Tools, permitiendo la interacción con bases de datos y el desarrollo de aplicaciones web dinámicas. En 1995, publicó el código fuente bajo el nombre PHP/FI ("Forms Interpreter"), lo que permitió a otros desarrolladores usarlo y contribuir a su mejora. Esta versión incluía variables similares a las de Perl y una sintaxis incrustada en HTML, aunque limitada y algo inconsistente.

PHP 3.0
En 1997, Andi Gutmans y Zeev Suraski, insatisfechos con las limitaciones de PHP/FI 2.0, comenzaron a desarrollar una nueva versión del lenguaje. Colaborando con Lerdorf, crearon PHP 3.0, que fue lanzado en 1998. Esta versión introdujo una mayor extensibilidad, permitiendo la integración con múltiples bases de datos, protocolos y APIs, y un soporte básico para la programación orientada a objetos. PHP 3.0 marcó un gran avance en popularidad, siendo instalado en más del 10% de los servidores web en Internet para el momento de su lanzamiento.

PHP 4.0
Poco después del lanzamiento de PHP 3.0, Gutmans y Suraski comenzaron a trabajar en un nuevo núcleo para PHP, con el objetivo de mejorar la ejecución de aplicaciones complejas y la modularidad del código. El resultado fue el motor Zend, introducido en 1999, que sirvió como base para PHP 4.0, lanzado en 2000. Esta versión mejoró significativamente el rendimiento, añadió soporte para sesiones HTTP, buffers de salida, y mecanismos más seguros para manejar entradas de usuario.

PHP 5.0
PHP 5.0 fue lanzado en 2004, impulsado por Zend Engine 2.0, que introdujo un nuevo modelo de objetos, mejorando significativamente el soporte para la programación orientada a objetos. Esta versión también añadió muchas otras características nuevas, consolidando a PHP como uno de los lenguajes más utilizados para el desarrollo web.

A lo largo de su historia, PHP ha evolucionado de un conjunto de scripts básicos a un lenguaje de programación robusto y ampliamente adoptado, gracias a su flexibilidad, extensibilidad y la capacidad de adaptarse a las necesidades cambiantes del desarrollo web.

Ventajas de PHP:

1. **Código Abierto:** PHP es gratuito y tiene una gran comunidad de desarrolladores que contribuyen con recursos y soporte.
2. **Fácil de Aprender:** Tiene una curva de aprendizaje baja, lo que lo hace accesible para principiantes.
3. **Compatibilidad:** Funciona en casi cualquier servidor y sistema operativo.
4. **Integración con HTML:** Se puede incrustar fácilmente en HTML, lo que simplifica la creación de páginas web dinámicas.
5. **Flexibilidad y Escalabilidad:** Es adecuado tanto para pequeños proyectos personales como para grandes aplicaciones empresariales.

Desventajas De PHP:

1. **Problemas de Seguridad:** Si no se configura correctamente, puede ser vulnerable a ataques.
2. **Rendimiento:** Puede ser menos eficiente en comparación con otros lenguajes para aplicaciones muy grandes.
3. **Limitaciones en el Desarrollo de Grandes Aplicaciones:** Aunque es flexible, puede no ser la mejor opción para aplicaciones extremadamente complejas.

Algoritmos de ordenamiento en PHP y su rendimiento:

Quicksort:

Ordena el arreglo dividiéndolo en sublistas con respecto a un pivote y luego ordena recursivamente las sublistas.

```php
<?php
// QuickSort Implementation
function quicksort(array &$array, int $low, int $high): void {
    if ($low < $high) {
        $pivotIndex = partition($array, $low, $high);
        quicksort($array, $low, $pivotIndex - 1);
        quicksort($array, $pivotIndex + 1, $high);
    }
}

function partition(array &$array, int $low, int $high): int {
    $pivot = $array[intval(($low + $high) / 2)];
    $i = $low;
    $j = $high;

    while ($i <= $j) {
        while ($array[$i] < $pivot) {
            $i++;
        }
        while ($array[$j] > $pivot) {
            $j--;
        }
        if ($i <= $j) {
            // Swap values
            $temp = $array[$i];
            $array[$i] = $array[$j];
            $array[$j] = $temp;
            $i++;
            $j--;
        }
    }
    return $i;
}

// Generate and initialize the array
$arraySize = 100000;
$array = range(1, $arraySize);
shuffle($array); // Shuffle array to ensure it's unsorted

// Measure the time of sorting
$startTime = microtime(true);
quicksort($array, 0, $arraySize - 1);
$endTime = microtime(true);

// Calculate the duration in milliseconds
$durationMilliseconds = ($endTime - $startTime) * 1000;

// Output the time taken and the size of the array
```

```
Duración del QuickSort: 69.5720 milisegundos con un arreglo de 100000 elementos
```

CPU Time: 0.08 sec(s) | Memory: 21120 kilobyte(s)

MergeSort:

Divide el arreglo en mitades hasta llegar a listas simples y luego fusiona estas listas ordenadamente.

```php
<?php
// Implementación de MergeSort
function mergeSort(array $array): array {
    // Si el arreglo tiene menos de dos elementos, ya está ordenado
    if (count($array) < 2) {
        return $array;
    }

    // Encuentra el punto medio del arreglo
    $middle = intval(count($array) / 2);
    // Divide el arreglo en dos mitades
    $left = array_slice($array, 0, $middle);
    $right = array_slice($array, $middle);

    // Ordena recursivamente ambas mitades y las combina
    return merge(mergeSort($left), mergeSort($right));
}

// Función para combinar dos arreglos ordenados en uno solo
function merge(array $left, array $right): array {
    $result = []; // Arreglo para almacenar el resultado combinado
    $i = $j = 0; // Índices para los arreglos izquierdo y derecho

    // Combina los elementos de ambos arreglos mientras ambos tengan elementos
    while ($i < count($left) && $j < count($right)) {
        if ($left[$i] <= $right[$j]) {
            $result[] = $left[$i];
            $i++;
        } else {
            $result[] = $right[$j];
            $j++;
        }
    }

    // Añade los elementos restantes del arreglo izquierdo, si los hay
    while ($i < count($left)) {
        $result[] = $left[$i];
        $i++;
    }

    // Añade los elementos restantes del arreglo derecho, si los hay
    while ($j < count($right)) {
        $result[] = $right[$j];
        $j++;
    }

    return $result;
}

// Generar e inicializar el arreglo
$arraySize = 100000; // Tamaño del arreglo
$array = range(1, $arraySize); // Crea un arreglo con números del 1 al 100000
shuffle($array); // Mezcla el arreglo para asegurarse de que esté desordenado

// Medir el tiempo de ordenamiento
$startTime = microtime(true); // Tiempo de inicio
$sortedArray = mergeSort($array); // Ordena el arreglo usando MergeSort
$endTime = microtime(true); // Tiempo de finalización

// Calcular la duración en milisegundos
$durationMilliseconds = ($endTime - $startTime) * 1000;

// Imprimir el tiempo tomado y el tamaño del arreglo
echo "Duración del MergeSort: " . number_format($durationMilliseconds, 4) . " milisegundos con un arreglo de " . $arraySize . " elementos\n";
?>
```

```
Duración del MergeSort: 158.8380 milisegundos con un arreglo de 100000 elementos
```

CPU Time: 0.16 sec(s) | Memory: 29344 kilobyte(s)

Burbuja:

Compara e intercambia elementos adyacentes repetidamente hasta que el arreglo esté ordenado.

```php
<?php
// Implementación de Bubble Sort
function bubbleSort(array &$array): void {
    $n = count($array);

    // Recorre el arreglo
    for ($i = 0; $i < $n - 1; $i++) {
        // Compara cada par de elementos adyacentes
        for ($j = 0; $j < $n - $i - 1; $j++) {
            if ($array[$j] > $array[$j + 1]) {
                // Intercambia los elementos si están en el orden incorrecto
                $temp = $array[$j];
                $array[$j] = $array[$j + 1];
                $array[$j + 1] = $temp;
            }
        }
    }
}

// Generar e inicializar el arreglo
$arraySize = 100; // Tamaño del arreglo
$array = range(1, $arraySize); // Crea un arreglo con números del 1 al 100000
shuffle($array); // Mezcla el arreglo para asegurarse de que esté desordenado

// Medir el tiempo de ordenamiento
$startTime = microtime(true); // Tiempo de inicio
bubbleSort($array); // Ordena el arreglo usando Bubble Sort
$endTime = microtime(true); // Tiempo de finalización

// Calcular la duración en milisegundos
$durationMilliseconds = ($endTime - $startTime) * 1000;

// Imprimir el tiempo tomado y el tamaño del arreglo
echo "Duración del Bubble Sort: " . number_format($durationMilliseconds, 4) . " milisegundos con un arreglo de " . $arraySize . " elementos\n";
?>
```

```
Duración del Bubble Sort: 0.2952 milisegundos con un arreglo de 100 elementos

CPU Time: 0.00 sec(s) | Memory: 19456 kilobyte(s)
```

A partir de las pruebas realizadas en este trabajo podemos observar que los algoritmos Quicksort y MergeSort son generalmente más eficientes que el Burbuja.

La duración del Quicksort para un array de 100.000 elementos fue de 69.5720 milisegundos, para la del MergeSort, con 100.000 también, fue de 158.8380 milisegundos, y la duración del Burbuja para un array de 100 elementos fue de 0.2952 milisegundos.

Si comparamos el rendimiento de cada uno, Quicksort suele ser el más rápido en la práctica debido a su eficiencia y por su bajo uso de memoria adicional. El MergeSort es también muy eficiente y tiene la ventaja de ser estable, pero puede requerir más memoria. El Burbuja es significativamente más lento y generalmente no se recomienda para grandes conjuntos de datos ya que como vimos en las pruebas puede llegar a romper en la compilación al querer ordenar un array de 100.000 elementos.

Pascal

Historia de Pascal

Pascal fue creado por Nikolaus Wirth, un informático suizo a finales de los años 60. Su objetivo era crear un lenguaje que fuera útil para enseñar programación, promover la organización del código y las buenas prácticas de programación.

Hitos clave:

- Primera versión de Pascal en 1970, inspirada en el lenguaje Algol 60 pero con una sintaxis más sencilla y clara.
- Pascal fue popular en los años 80 debido a compiladores como Turbo Pascal de Borland, que ofrecían un entorno de desarrollo integrado (IDE) muy intuitivo y accesible.Dado que Turbo Pascal sólo estaba disponible para una arquitectura, traducía directamente a código máquina del Intel 8088, logrando construir programas que se ejecutaban mucho más rápidamente que los producidos en los esquemas interpretados.
- Pascal sirvió como base para el desarrollo de otros lenguajes como Modula-2 y Oberon también creados por Wirth. Además, sus principios de programación estructurada influyeron en lenguajes modernos como C y C++.
- Se utilizaba en la enseñanza de la programación y en el desarrollo de software para sistemas embebidos.Por ejemplo, controladores lógicos programables (PLC) en automatización industrial, equipos médicos como monitores de signos vitales y ultrasonidos, etc.

¿Qué tan importante fue Pascal?

- Enseñanza: Su sintaxis clara y estructurada lo convirtieron en un lenguaje ideal para enseñar los fundamentos de la programación.
- Programación estructurada: Fomenta el uso de procedimientos, funciones y bloques de código, lo que mejoró la legibilidad y el mantenimiento del software.
- Base para otros lenguajes: Sus conceptos y principios influyeron en el diseño de muchos lenguajes posteriores.

Orígenes y Desarrollo

Niklaus Wirth desarrolla Pascal como una mejora sobre el lenguaje ALGOL, que ya promovía la programación estructurada pero era considerado complicado y poco accesible para los estudiantes. Wirth quería un lenguaje que fuera simple, eficiente y fácil de aprender, pero que también pudiera ser utilizado para desarrollar software real.

¿Dónde se usa Pascal?

1. Aplicaciones Educativas y Herramientas de Aprendizaje.
2. Desarrollo de Software Antiguo.
3. Desarrollo de Compiladores.

4. Delphi (Object Pascal).
5. Robótica y Sistemas Embebidos.

Ventajas de Pascal

1. **Fácil de Aprender**: Pascal fue diseñado con la enseñanza en mente, lo que lo hace un lenguaje ideal para principiantes. Su sintaxis es clara y concisa, lo que facilita la comprensión de los conceptos básicos de programación.
2. **Promueve Buenas Prácticas de Programación**: Pascal fomenta la programación estructurada y modular. Esto ayuda a los programadores a desarrollar hábitos de codificación que son útiles en otros lenguajes y en proyectos más grandes.
3. **Tipado Estricto**: El tipado estricto de Pascal ayuda a prevenir errores comunes relacionados con los tipos de datos, lo que puede mejorar la confiabilidad y robustez del código.
4. **Eficiencia**: Los compiladores de Pascal, especialmente Turbo Pascal, son conocidos por su rapidez y eficiencia en la generación de código ejecutable.
5. **Buena Documentación y Comunidad de Soporte**: Existe una gran cantidad de recursos educativos, libros y documentación disponible para aprender y trabajar con Pascal.
6. **Entorno de Desarrollo Integrado (IDE)**: Herramientas como Turbo Pascal ofrecieron un entorno de desarrollo integrado fácil de usar, lo que facilitaba la edición, compilación y depuración del código.

Desventajas de Pascal

1. **Popularidad Decreciente**: Pascal no es tan popular en la industria del software actual como otros lenguajes modernos como Python, Java o JavaScript, lo que puede limitar sus aplicaciones y oportunidades laborales.
2. **Limitaciones en Programación Orientada a Objetos (OOP)**: Aunque Object Pascal (una extensión de Pascal) introdujo características de OOP, el lenguaje original de Pascal no fue diseñado con la programación orientada a objetos en mente, lo que puede ser una limitación para algunos tipos de aplicaciones modernas.
3. **Menos Flexibilidad para Aplicaciones Comerciales**: Mientras que Pascal es excelente para la enseñanza y aplicaciones científicas, no ha sido tan ampliamente adoptado para el desarrollo de aplicaciones comerciales y empresariales.
4. **Ecosistema y Librerías Limitadas**: En comparación con otros lenguajes modernos, Pascal tiene un ecosistema más limitado de bibliotecas y frameworks, lo que puede dificultar el desarrollo de aplicaciones complejas.
5. **Menor Soporte para Desarrollos Web y Móviles**: Otros lenguajes tienen mejores herramientas y bibliotecas para el desarrollo de aplicaciones web y móviles, áreas donde Pascal no se destaca.
6. **Compatibilidad con Hardware y Sistemas Operativos Modernos**: Aunque existen versiones modernas de Pascal, no siempre son tan compatibles o eficientes en hardware y sistemas operativos modernos como los lenguajes de programación más actuales.

Métodos de ordenamiento en Pascal

- **Burbuja:** Es posible que este sea el algoritmo más sencillo de escribir, y también el menos eficiente. La idea es sencilla: se van desplazando los números más pequeños hacia atrás hasta su posición correcta.

```
for i := maximo downto 2 do            { De final a principio }
  if datos[i] < datos[i-1] then        { Si estÃ¡ colocado al revÃ©s }
    swap(datos[i], datos[i-1]);        { Le da la vuelta }
```

-

```
procedure swap(var a,b: integer);          { Intercambia dos datos }
var
  tmp: integer;
begin
  tmp := a;
  a := b;
  b := tmp;
end;
```

-

- **MergeSort:** En este caso, se trata de dividir nuestra lista en "sublistas" cada vez más pequeñas (se hará de forma recursiva). Finalmente, volvemos a juntar todas esas listas en una sola.

```
funciÃ³n MERGESORT (a: lista): lista
  opcion
    |a| = 1: devolver a                { TamaÃ±o 1: fin de recursiÃ³n }
    |a| <> 1:
       (x1,x2) := DESCOMPONER(a)       { Si no, divide en dos trozos }
       s1 := MERGESORT (x1)            { Aplica recursivamente a los }
       s2 := MERGESORT (x2)            {   dos trozos }
       devolver MERGE(s1,s2)           { Y finalmente combina en una }
  fopc                                 {   Ãºnica lista grande }
ff
```

-

- **QuickSort:** Lo que este hace es mirar el valor del centro de la lista. Mueve a su derecha todos los valores menores y a la izquierda todos los mayores, pero no los ordena aún, sino que luego recursivamente vuelve a hacer lo mismo en ambos trozos. Así finalmente queda ordenado.

```
procedure Sort(l, r: Integer);          { Esta es la parte recursiva }
var
  i, j, x, y: integer;
begin
  i := l; j := r;                       { LÃmites por los lados }
  x := datos[(l+r) DIV 2];              { Centro de la comparaciones }
  repeat
    while datos[i] < x do i := i + 1;   { Salta los ya colocados }
    while x < datos[j] do j := j - 1;   {   en ambos lados }
    if i <= j then                      { Si queda alguno sin colocar }
      begin
      swap(datos[i], datos[j]);         { Los cambia de lado }
      i := i + 1; j := j - 1;           { Y sigue acercÃ¡ndose al centro }
      end;
  until i > j;                          { Hasta que lo pasemos }
  if l < j then Sort(l, j);             { Llamadas recursivas por cada }
  if i < r then Sort(i, r);             {   lado }
end;
```

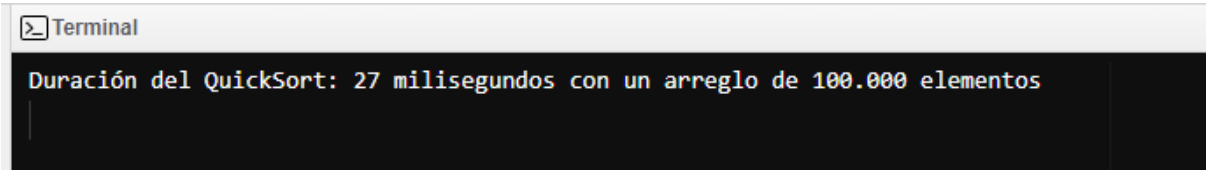
-

Ejemplo rendimiento Algoritmo QuickSort en Pascal

Ejemplo de rendimiento utilizando el algoritmo QuickSort en compilador online
https://www.tutorialspoint.com/compile_pascal_online.php

```
program SortComparison;

uses
  SysUtils, DateUtils;

type
  TDoubleArray = array of Double;

// Procedimiento QuickSort
procedure QuickSort(var A: TDoubleArray; Low, High: NativeInt);
var
  i, j: NativeInt;
  pivot, temp: Double;
begin
  if Low >= High then Exit;
  i := Low;
  j := High;
  pivot := A[(Low + High) div 2];
  repeat
    while A[i] < pivot do Inc(i);
    while A[j] > pivot do Dec(j);
    if i <= j then
    begin
      temp := A[i];
      A[i] := A[j];
      A[j] := temp;
      Inc(i);
      Dec(j);
    end;
  until i > j;
  if Low < j then QuickSort(A, Low, j);
  if i < High then QuickSort(A, i, High);
end;

// Procedimiento para ordenar y medir el tiempo
procedure SortAndMeasure(var A: TDoubleArray; var Duration: TDateTime);
var
  Start, Finish: TDateTime;
```

```
35  // Procedimiento para ordenar y medir el tiempo
36  procedure SortAndMeasure(var A: TDoubleArray; var Duration: TDateTime);
37  var
38    Start, Finish: TDateTime;
39  begin
40    Start := Now;
41    QuickSort(A, 0, High(A));
42    Finish := Now;
43    Duration := Finish - Start;
44  end;
45
46  var
47    LargeArray: TDoubleArray;
48    i: NativeInt;
49    Duration: TDateTime;
50
51  begin
52    Randomize;
53    SetLength(LargeArray, 100000);  // Arreglo grande
54    for i := 0 to High(LargeArray) do
55      LargeArray[i] := Random * 100000;
56
57    SortAndMeasure(LargeArray, Duration);
58    Writeln('Duración del QuickSort: ', MilliSecondsBetween(0, Duration), ' milisegundos con un arreglo
          de 100.000 elementos');
59  end.
```



Tipo de dato utilizado para el arreglo: NativeInt.

En sistemas de 64 bits, NativeInt es un entero con signo de 64 bits, lo que significa que puede representar valores en el rango de -9,223,372,036,854,775,808 a 9,223,372,036,854,775,807.

Comparación entre PHP y Pascal:

Basándonos en la investigación de ambos lenguajes y en las pruebas realizadas de los algoritmos, identificamos los siguientes 4 puntos claves para realizar la comparación.

- **Simplicidad**: PHP destaca por su sintaxis sencilla y directa, lo que facilita la escritura y comprensión del código. En contraste, Pascal tiene una estructura más rígida y formal, lo que aunque puede resultar en un código más explícito y organizado, también puede hacer que la escritura sea más detallada y menos flexible.
- **Manipulación de Arrays**: En PHP, la manipulación de arrays es intuitiva y flexible, con funciones incorporadas que facilitan su uso sin necesidad de definiciones

detalladas. En Pascal, sin embargo, los arrays requieren una definición más precisa y suelen involucrar el uso de procedimientos específicos para su manipulación, lo que puede agregar complejidad al desarrollo.

- **Manejo de Tiempos**: PHP proporciona funciones nativas como `microtime()` que permiten medir fácilmente el tiempo de ejecución de scripts, lo que es útil para realizar pruebas de rendimiento. En Pascal, medir el tiempo de ejecución es menos directo, ya que generalmente se requiere utilizar bibliotecas adicionales o escribir código adicional para lograrlo.
- **Desempeño**: Aunque PHP es ágil para desarrollar y probar pequeños algoritmos, es menos eficiente en operaciones de cálculo intensivo debido a su naturaleza interpretada y orientada a la web. Pascal, por otro lado, es un lenguaje compilado que está más cerca del hardware, lo que le permite ejecutar tareas con mayor eficiencia y ofrecer un rendimiento superior en aplicaciones que demandan un uso intensivo de recursos.

Como conclusión podemos decir que, aunque PHP es ideal para desarrollar rápidamente y probar pequeños algoritmos gracias a su facilidad de uso y flexibilidad, Pascal es la mejor opción cuando se requiere un rendimiento superior o un mayor control sobre los recursos del sistema. Pascal es especialmente preferible en aplicaciones que demandan alta eficiencia y velocidad de ejecución.

Bibliografía

PHP

https://www.php.net/manual/es/history.php.php

Ventajas y desventajas del lenguaje PHP » BaulPHP

Ventajas y Desventajas de PHP: Una Guía Completa para Programadores (dongee.com)

Pascal

https://lenguajesprogramacionblog.wordpress.com/2017/04/22/lenguaje-de-programacion-pascal/

https://programacionpro.com/lenguaje-de-programacion-pascal-todo-lo-que-necesitas-saber/

https://www.nachocabanes.com/pascal/curso/cupasamp03.php