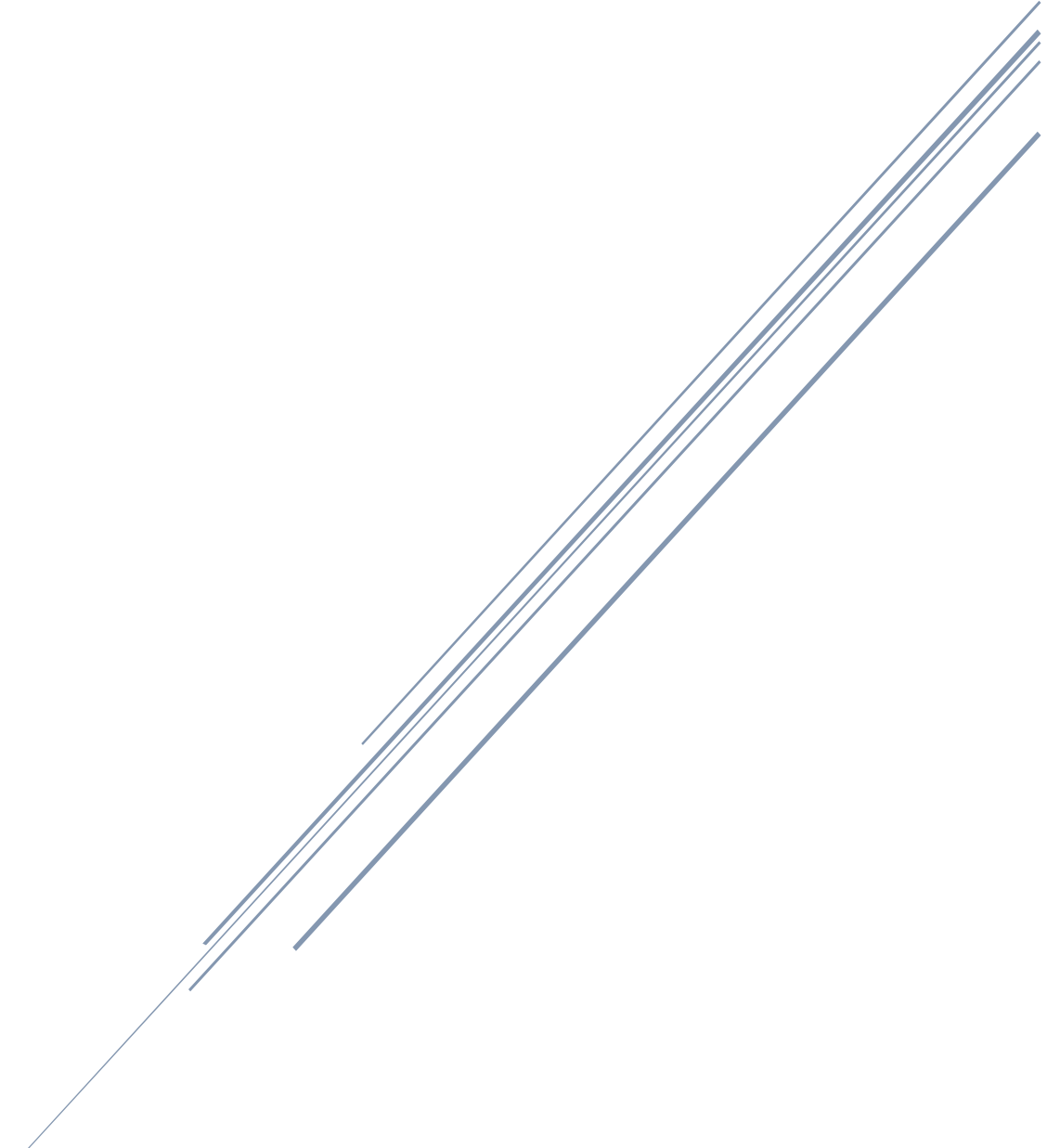


TRABAJO PRACTICO - TEORICO

Sintaxis y Semántica de los Lenguajes



Universidad Tecnológica Nacional Facultad Regional Buenos Aires
Integrantes: Gonzalo Salas Vetre – Juan Ignacio Piroso
Curso: K2006

Índice

Lenguaje PHP	2
Introducción.....	2
Algunas BNF de PHP.....	2
Lenguaje Haskell	3
Introducción.....	3
Algunas BNF de Haskell.....	4
PHP VS HASKELL	5
Algoritmo de búsqueda lineal.....	5
<i>Haskell</i>	5
Tiempo en encontrar el 1	5
Tiempo en encontrar el 8	5
<i>PHP</i>	6
Tiempo en encontrar el 1	6
Tiempo en encontrar el 8	6
Algoritmo de ordenamiento (Burbuja)	6
<i>Haskell</i>	6
<i>PHP</i>	7
Diferencias entre PHP y Haskell	7

Lenguaje PHP

Introducción

Creado por Rasmus Lerdorf en 1994 PHP es un lenguaje de programación interpretado del lado del servidor y de uso general que se adapta especialmente a desarrollo web.

PHP tuvo sus inicios como un CGI en C para interpretar comandos limitados, conocido como Personal Home Page Tools. Su éxito creció cuando otros desarrolladores quisieron usarlo en sus páginas web. Más adelante, se añadió un sistema para procesar formularios llamado FI (Form Interpreter), y esta combinación se convirtió en la primera versión compacta, PHP/FI.

En 1997, se reescribió el analizador sintáctico y se añadieron características como soporte para protocolos web y bases de datos comerciales. Estas mejoras sentaron las bases para PHP versión 3. Aunque aún tenía un largo camino por recorrer, ya ofrecía muchas funcionalidades, lo que atrajo a programadores y contribuyó a su crecimiento como herramienta esencial para el desarrollo web.

Algunas BNF de PHP

```
1  program      ::= statement_list
2
3  statement_list ::= statement
4                  | statement statement_list
5
6  statement     ::= expression_statement
7                  | if_statement
8                  | while_statement
9                  | ...
10
11 expression_statement ::= expression ';'
12
13 if_statement  ::= 'if' '(' expression ')' '{' statement_list '}'
14                  ['else' '{' statement_list '}']
15
16 while_statement ::= 'while' '(' expression ')' '{' statement_list '}'
17
18 expression    ::= variable '=' value
19                  | function_call
20                  | literal_value
21                  | ...
22
23 function_call ::= function_name '(' argument_list ')'
24
25 argument_list ::= expression
26                  | expression ',' argument_list
27
28 variable      ::= '$' identifier
29
30 literal_value  ::= integer
31                  | string
32                  | boolean
33                  | ...
34
35 integer       ::= [0-9]+
36
37 string        ::= '"' characters '"'
38
39 characters    ::= [any characters]
40
41 boolean       ::= 'true' | 'false'
42
43 identifier    ::= [a-zA-Z_][a-zA-Z0-9_]*
```

Lenguaje Haskell

Introducción

Haskell es un lenguaje de programación puramente funcional, cuya primera versión fue lanzada en 1990. Su nombre proviene del matemático Haskell Brooks Curry, que sentó las bases de los lenguajes de programación funcional con su trabajo sobre lógica combinatoria (entre 1920 y 1960). Haskell se basa en el cálculo lambda (lenguaje formal para la investigación de funciones), por lo que el logotipo del lenguaje contiene el símbolo de esta letra griega.

Los programas escritos en Haskell se representan siempre como funciones matemáticas, pero estas funciones nunca tienen efectos secundarios ni derivados. De este modo, cada función utilizada siempre devuelve el mismo resultado con la misma entrada, y el estado del programa nunca cambia. Por esto, el valor de una expresión o el resultado de una función dependen exclusivamente de los parámetros de entrada en el momento. En Haskell no pueden hacerse construcciones de lenguaje imperativo para programar una secuencia de declaraciones.

Haskell destaca sobre todo por las siguientes ventajas:

- La productividad de los desarrolladores puede aumentar considerablemente.
- El código del software de Haskell es breve, claro y fácil de mantener.
- Las aplicaciones de Haskell son menos propensas a errores y ofrecen una gran fiabilidad.
- La brecha “semántica” entre el programador y el lenguaje es mínima.

Algunas BNF de Haskell

<u>varid</u>	→ (small {small large digit ' })
<u>conid</u>	→ large {small large digit ' }
<u>reservedid</u>	→ case class data default deriving do else foreign if import in infix <u>infixl</u> <u>infixr</u> instance let module newtype of then type where _
<u>varsym</u>	→ (symbol _c {symbol}) _(reservedop dashes)
<u>consym</u>	→ ({symbol}) _(reservedop)
<u>reservedop</u>	→ .. : :: = \ <- -> @ ~ =>
<u>varid</u>	(variables)
<u>conid</u>	(constructors)
<u>tyvar</u>	→ varid (type variables)
<u>tycon</u>	→ conid (type constructors)
<u>tycls</u>	→ conid (type classes)
<u>modid</u>	→ {conid .} conid (modules)
<u>qvarid</u>	→ [modid .] varid
<u>qconid</u>	→ [modid .] conid
<u>qtycon</u>	→ [modid .] tycon
<u>qtycls</u>	→ [modid .] tycls
<u>qvarsym</u>	→ [modid .] varsym

<u>whitestuff</u>	→ whitechar comment <u>ncomment</u>
<u>whitechar</u>	→ newline <u>vertab</u> space tab uniWhite
<u>newline</u>	→ return linefeed <u>formfeed</u>
<u>return</u>	→ a carriage return
<u>linefeed</u>	→ a line feed
<u>vertab</u>	→ a vertical tab
<u>formfeed</u>	→ a form feed
<u>space</u>	→ a space
<u>tab</u>	→ a horizontal tab
<u>uniWhite</u>	→ any Unicode character defined as whitespace
<u>comment</u>	→ dashes [any _(symbol) {any}] newline
<u>dashes</u>	→ -- { }
<u>opencom</u>	→ { -
<u>closecom</u>	→ - }
<u>ncomment</u>	→ opencom ANY seq {ncomment ANY seq} closecom
<u>ANY seq</u>	→ {ANY} _{(ANY) (opencom closecom) (ANY)}
<u>ANY</u>	→ graphic whitechar
<u>any</u>	→ graphic space tab
<u>graphic</u>	→ small large symbol digit special " '
<u>small</u>	→ ascSmall uniSmall _
<u>ascSmall</u>	→ a b ... z
<u>uniSmall</u>	→ any Unicode lowercase letter
<u>large</u>	→ ascLarge uniLarge
<u>ascLarge</u>	→ A B ... Z
<u>uniLarge</u>	→ any uppercase or <u>titlecase</u> Unicode letter
<u>symbol</u>	→ ascSymbol uniSymbol _(special _ '*')
<u>ascSymbol</u>	→ ! # \$ % & * + . / < = > ? @

<u>qconsym</u>	→ [modid .] consym
<u>decimal</u>	→ digit {digit}
<u>octal</u>	→ <u>octit</u> {octit}
<u>hexadecimal</u>	→ <u>hexit</u> {hexit}
<u>integer</u>	→ decimal 0o octal 0O octal 0x hexadecimal 0X hexadecimal
<u>float</u>	→ decimal . decimal [exponent] decimal exponent
<u>exponent</u>	→ (e E) [+ -] decimal
<u>char</u>	→ ' {graphic _c space escape _(\a) } '
<u>string</u>	→ " {graphic _c space escape gap } "
<u>escape</u>	→ \ (charesc ascii decimal o octal x hexadecimal)
<u>charesc</u>	→ a b f n r t v \ " ' &
<u>ascii</u>	→ ^cntrl NUL SOH STX ETX EOT ENQ ACK BEL BS HT LF VT FF CR SO SI DLE DC1 DC2 DC3 DC4 NAK SYN ETB CAN EM SUB ESC FS GS RS US SP DEL
<u>cntrl</u>	→ ascLarge @ [\] ^ _
<u>gap</u>	→ \ whitechar {whitechar} \
<u>program</u>	→ { lexeme whitespace }
<u>lexeme</u>	→ qvarid qconid qvarsym qconsym literal special <u>reservedop</u> <u>reservedid</u>
<u>literal</u>	→ integer float char string
<u>special</u>	→ () , ; [] ` { }
<u>whitespace</u>	→ whitestuff {whitestuff}

PHP VS HASKELL

Algoritmo de búsqueda lineal

Haskell

```
1 busquedaLineal :: Eq a => a -> [a] -> Bool
2 busquedaLineal [] _ = False
3 busquedaLineal (x:xs) elemento | x == elemento = True
4                               | otherwise = busquedaLineal xs elemento
```

Lista usada = [2,3,1,2,3,6,5,8]

Tiempo en encontrar el 1

```
benchmarking busquedaLineal/1
time          36.08 ns   (35.69 ns .. 36.72 ns)
              0.993 R²   (0.985 R² .. 0.999 R²)
mean          37.88 ns   (36.74 ns .. 40.44 ns)
std dev       5.418 ns   (2.926 ns .. 9.732 ns)
variance introduced by outliers: 96% (severely inflated)
```

Tiempo en encontrar el 8

```
benchmarking busquedaLineal/8
time          77.55 ns   (75.68 ns .. 80.07 ns)
              0.993 R²   (0.987 R² .. 0.999 R²)
mean          76.78 ns   (75.62 ns .. 79.52 ns)
std dev       5.441 ns   (2.839 ns .. 9.077 ns)
variance introduced by outliers: 83% (severely inflated)
```

PHP

```
1 function busquedaLineal($elemento, $array) {
2     foreach ($array as $valor) {
3         if ($valor == $elemento) {
4             return true;
5         }
6     }
7     return false;
8 }
```

Lista usada = [2,3,1,2,3,6,5,8]

Tiempo en encontrar el 1

```
PS C:\Users\juani\OneDrive\Escritorio\Programacion\PHP> php -f algoritmoSSL.php
Tiempo transcurrido: 0.000006 segundos
PS C:\Users\juani\OneDrive\Escritorio\Programacion\PHP> |
```

Tiempo en encontrar el 8

```
PS C:\Users\juani\OneDrive\Escritorio\Programacion\PHP> php -f algoritmoSSL.php
Tiempo transcurrido: 0.000007 segundos
PS C:\Users\juani\OneDrive\Escritorio\Programacion\PHP> php -f algoritmoSSL.php
```

Algoritmo de ordenamiento (Burbuja)

Haskell

```
1 ordenamientoBurbuja :: Ord a => [a] -> [a]
2 ordenamientoBurbuja lista = foldl cambiarHasta [] lista
3
4 cambiarHasta :: Ord b => [b] -> b -> [b]
5 cambiarHasta [] elemento = [elemento]
6 cambiarHasta (siguienteElemento:cola) elemento =
7     min elemento siguienteElemento : cambiarHasta cola (max elemento siguienteElemento)
```

Lista usada = [9,8,7,6,5,4,3,2,1]

```
benchmarking Burbuja
time                186.6 ns   (181.1 ns .. 194.3 ns)
                    0.992 R²   (0.987 R² .. 0.998 R²)
mean                185.0 ns   (182.3 ns .. 189.4 ns)
std dev             11.08 ns   (5.666 ns .. 17.38 ns)
variance introduced by outliers: 77% (severely inflated)
```

PHP

```
1 function ordenamientoBurbuja($array)
2 {
3     $arraySize = count($array);
4     for ($i = 0; $i < $arraySize - 1; $i++) {
5         for ($j = 0; $j < $arraySize - $i - 1; $j++) {
6             if ($array[$j] > $array[$j + 1]) {
7                 $temp = $array[$j];
8                 $array[$j] = $array[$j + 1];
9                 $array[$j + 1] = $temp;
10            }
11        }
12    }
13    return $array;
14 }
```

Lista usada = [9,8,7,6,5,4,3,2,1]

```
PS C:\Users\juani\OneDrive\Escritorio\Programacion\PHP> php -f algoritmoSSL.php
```

```
Tiempo transcurrido: 0.000018 segundos
```

```
PS C:\Users\juani\OneDrive\Escritorio\Programacion\PHP> |
```

Diferencias entre PHP y Haskell

- **Paradigma:**

PHP: Imperativo y orientado a objetos.

Haskell: Funcional pura.

- **Tipado:**

PHP: Dinámico (tiempo de ejecución).

Haskell: Estático (tiempo de compilación).

- **Evaluación:**

PHP: Estricta (evaluación cuando se necesita).

Haskell: Perezosa (evaluación postergada).

- **Sintaxis/Estilo:**

PHP: Sintaxis más permisiva, estilo variado.

Haskell: Sintaxis funcional compacta y estructurada.

- **Casos de uso:**

PHP: Desarrollo web principalmente.

Haskell: Amplia gama, incluyendo investigación y matemáticas.