Universidad Nacional Autónoma de México
Facultad de Ciencias

Lenguajes de Programación

Karla Ramírez Pulido

Introducción

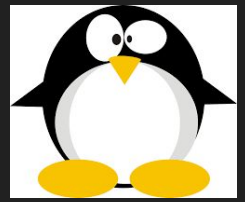
Preguntas:

1. ¿Cuántos Lenguajes de Programación (LP) conocen?
2. ¿En cuántos LP has programado?
3. ¿En cuántos programas llamémosle “BIEN”?
4. ¿En qué LP te siente realmente cómodo programando?
5. Si te enfrentas a un nuevo LP ¿cuánto tiempo transcurre hasta que te sientes más “confiado” “cómodo” en ese lenguaje?
6. ¿En todos los lenguajes que conoces tienes/percibes un mismo nivel de programación?
7. ¿El número de líneas escritas en un LP nos hace “mejores programadores”?

Preguntas:

Entonces hay diferentes estadíos de programación:

- Aprendiendo
- Programando ya con algo de experiencia
- Super experto



Most Popular Programming Languages

1965-2021

Y axis is a relative value to define ranking popularity between all other items.

¿Cómo nos comunicamos con la computadora?

Los programadores: Lenguaje de Programación

Herramienta para comunicarnos que utilizan los programadores para comunicarse con la computadora.

Definición

Un lenguaje de programación es una terna $L = < P_L, D_L, [\![.]\!]_L >$ donde

$P_L \neq \varnothing$ es el conjunto de programas de L

$D_L \neq \varnothing$ es el conjunto de datos (de entrada y salida) de L

$[\![.]\!]_L$ es la función semántica de L tal que

$$[\![.]\!]_L : (P_L \rightarrow D_L) \rightharpoonup D_L$$

¿Qué consta un LP?

1. Sintaxis
2. Semántica
3. Bibliotecas
4. Idioms

Sintaxis

- int
- integer
- INT
- INTEGER
- Integer
- Number

Semántica

Significado

Java: 32 bits / 4 bytes

Sintaxis y Semántica

1. JAVA → a [25]
2. SCHEME → (vector-ref a 25)
3. C → a [25]
4. ML → a [25]

Bibliotecas

Library en inglés, cuya traducción es biblioteca y no librería

Defn 1. Es un **conjunto de subprogramas** utilizados para desarrollar software.

Defn 2. Es un **conjunto de implementaciones funcionales**, programadas en un lenguaje de programación, que ofrece una interfaz bien definida para la funcionalidad que se invoca.

Las bibliotecas contienen código y datos, que proporcionan servicios a programas independientes, es decir, pasan a formar parte de éstos. Esto permite que el código y los datos se compartan y puedan modificarse de forma modular

Idioms

¿Cómo nombramos a nuestros archivos del proyecto final?

1. ProyectoFinal.pdf
2. ProyectoFinalFinal.pdf
3. ProyectoFinalFinalFinal.pdf
4. ProyectoFinalV1.pdf
5. ProyectoFInalFinalVN.pdf
6. ProyectoFinalFinalFinalFinalElBUENO.pdf
7. ProyectoFinalEntregaProfe.pdf
8. ...

En programación

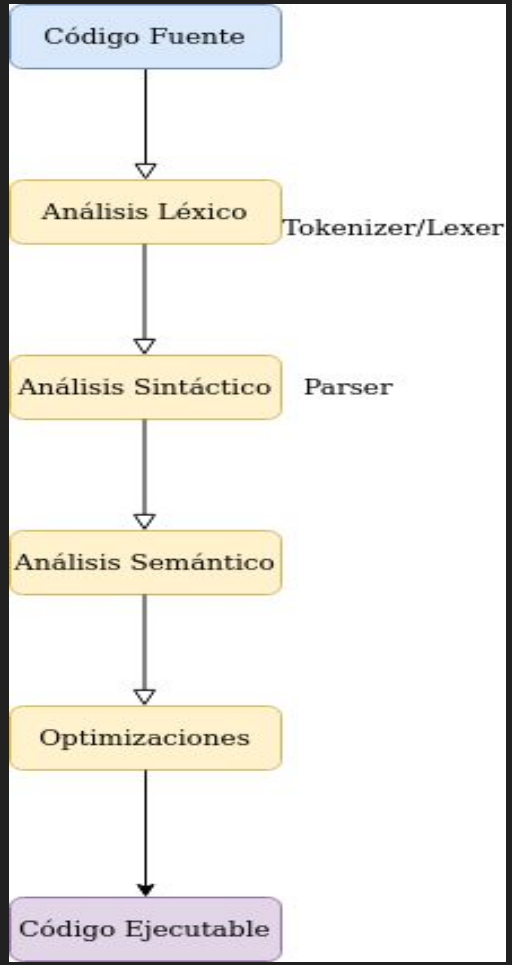
```
int x=0;

int y=0;

int z=0;
```

```
int mi-hermosa-constante = 0;
```

```
for (int i=0;   i<10;   i++)

   for(int j=0;   j<10;   j++)

      for(int k=0;   k<10;   k++)
```

```
for (int omega=0;   omega<10;   omega++)

   for(int phi=0;   phi<10;   phi++)

      for(int a=0;   a<10;   a++)
```

Análisis


Código Fuente
Análisis Léxico
Tokenizer/Lexer
Análisis Sintáctico
Parser
Análisis Semántico
Optimizaciones
Código Ejecutable

Análisis léxico

int x = 0;

Lexemas

int x = 0 ;

5 lexemas

Análisis sintáctico

```
if ( n < 100 )
    then (n + 2);
    else “error”;
```

```
i  f ( n < 100 )
     t h en  (n + 2);
      else “error”;
```

Análisis Semántico

```
if ( n < 100 )
    then (n + 2);
    else “error”;
```

Evaluar la condicional del if

(n < 100)

si se evalúa a un valor Verdadero entonces se evaluará la expresión **then**

y si no (i.e. se evalúa a un valor Falso) entonces se evaluará la expresión **else**

Primera decisión:

Lenguaje compilado

Realiza todo los pasos anteriores todo el código fuente completo.

Lenguaje interpretado

Realiza todos los pasos anteriores, pero por cada línea, cada vez que ejecutamos el programa.

Ejemplos

```
1.    (define (mult-por-2  n)
2.              (*  n   2))
```

```
> (mult-por-2    4)

8

> (mult-por-2    2)

4
```

Lenguaje Interpretado

1. Análisis léxico para líneas 1 y 2
2. Análisis sintáctico para líneas 1 y 2
3. Análisis semántico para líneas 1 y 2
4. Optimizaciones
5. Generar código ejecutable

1. Análisis léxico para líneas 1 y 2
2. Análisis sintáctico para líneas 1 y 2
3. Análisis semántico para líneas 1 y 2
4. Optimizaciones
5. Generar código ejecutable

Ejemplos

1. (define (mult-por-2 n)
2. (* n 2))

> (mult-por-2 4)

8

> (mult-por-2 2)

4

Lenguaje Compilado

1. Análisis léxico para líneas 1 y 2
2. Análisis sintáctico para líneas 1 y 2
3. Análisis semántico para líneas 1 y 2
4. Optimizaciones
5. Generar código ejecutable

Cuando llamamos a (mult-por-dos 4) y (mult-por-2 2) si ya tenemos el código ejecutable solo lo mandamos llamar

Notación puede ser:

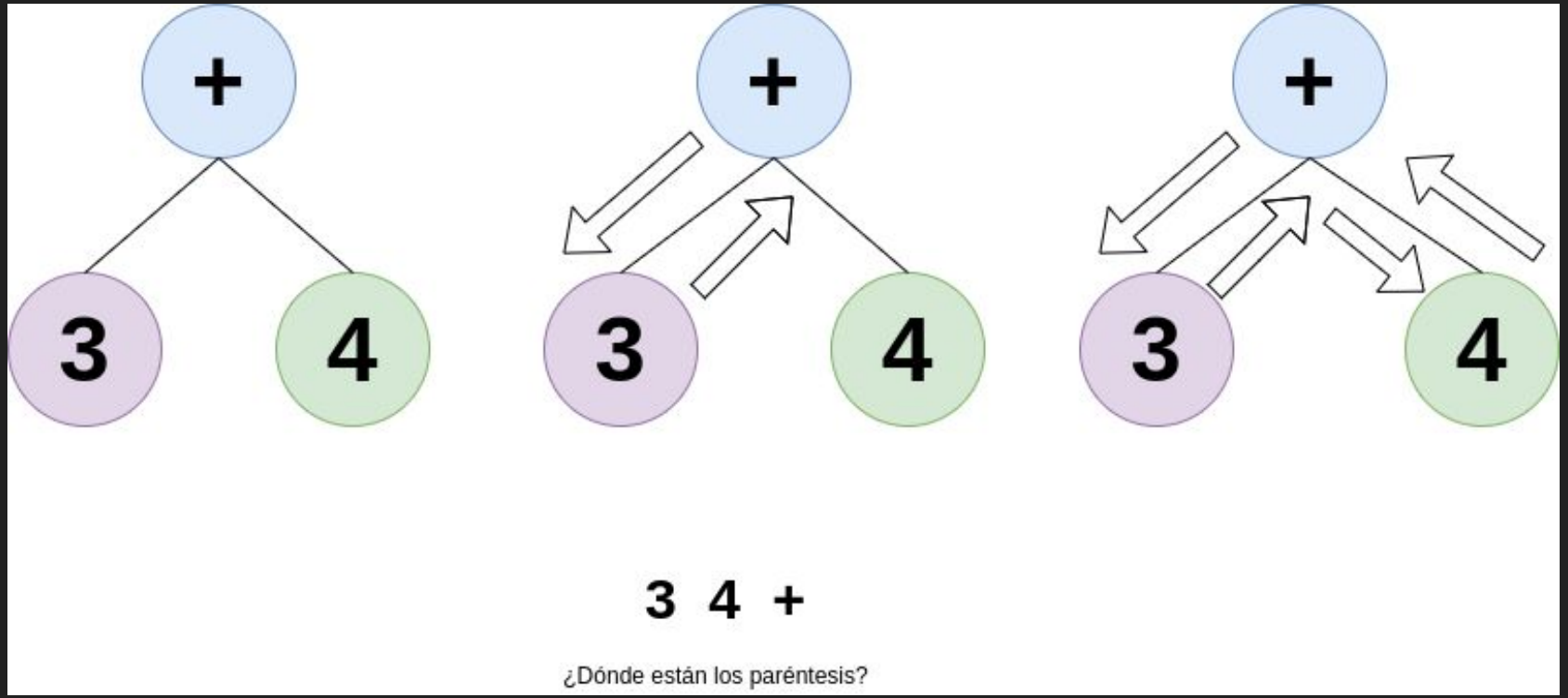
3 + 4 (infija)

3 4 + (postfija)

(+ 3 4) (prefija parentizada)

Notación de árbol

Recorridos



Recorrido

Notación infija:

3 + 4

lado izquierdo -- raíz --- lado derecho

Notación prefija:

\+ 3 4

raíz -- lado izquierdo -- lado derecho

Notación postfija:

3 4 +

lado izquierdo -- lado derecho -- raíz

Sintaxis

(+ 3 4)

(add (num 3) (num 4))

Etiquetas: add para denotar a la suma “+”

num para denotar un número

Nosotros usaremos “{“ “}” y notación prefija

1. 3
2. {+ 3 4}
3. {+ {- 3 4} 7}

Notación BNF

<AE> : := <num>

| {+ <AE> <AE>}

| {- <AE> <AE>}

Ejemplos:

A. 34
B. 454000
C. {+ 3 5}
D. {- {+ 3 5} 9 }
E. {- {+ 3 5} { - 2 2} }

Constructor

```
(define-type AE
    [num (n number?)]
    [add (lhs AE?) (rhs AE?)]
    [sub (lhs AE?) (rhs AE?)] )
```

Parse

```
;; parse : sexp → AE

;; to convert s-expressions into AEs

(define (parse  sexp)

      (cond    [ (number?  sexp) (num  sexp)]

                  [ (list?  sexp)

                        (case (first sexp)
```

Parser

[(+) (add (parse (second sexp)) (parse (third sexp)))]

[(-) (sub (parse (second sexp)) (parse (third sexp)))])]))

Ejemplos

(parse ‘3) sexp = 3
(num 3)

(parse ‘{+ 3 4})
sexp = {+ 3 4}
(add (parse 3) (parse 4))
= (add (num 3) (parse 4))
= (add (num 3) (num 4))

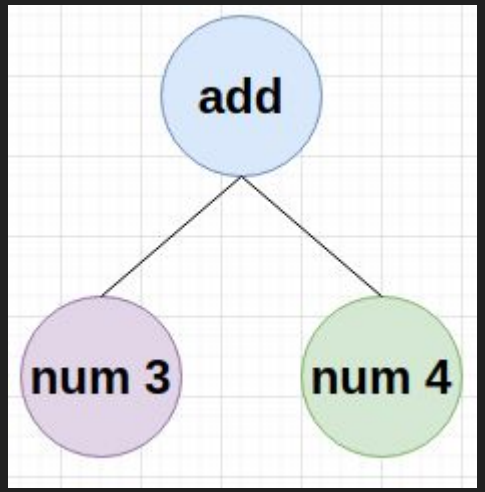
Parser

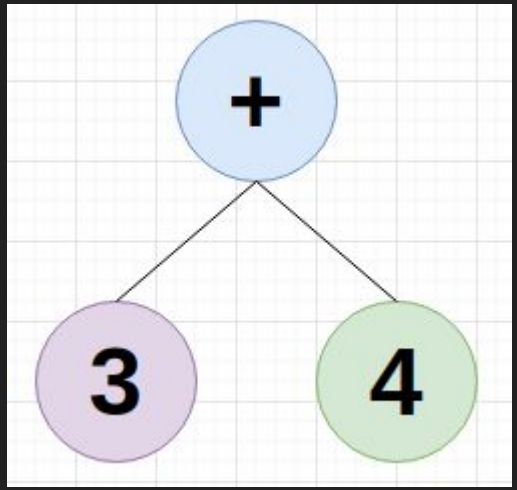
```
(define (parse  sexp)
   (cond   [ (number?  sexp) (num  sexp)]
           [ (list?  sexp)
                (case (first sexp)
                 [ (+) (add (parse (second sexp))
                            (parse (third sexp)))]
                [ (-) (sub (parse (second sexp))
                           (parse (third sexp)))]  ) ] ) )
```

```
> (parse ‘3)
(num 3)

> (parse ‘{-  1  4} )
   (sub (parse 1) (parse 4))
= (sub (num 1) (parse 4))
= (sub (num 1) (num 4))
```

(add (num 3) (num 4))

Árbol generado por el PARSE se llama ÁRBOL de SINTAXIS ABSTRACTA, ASA.



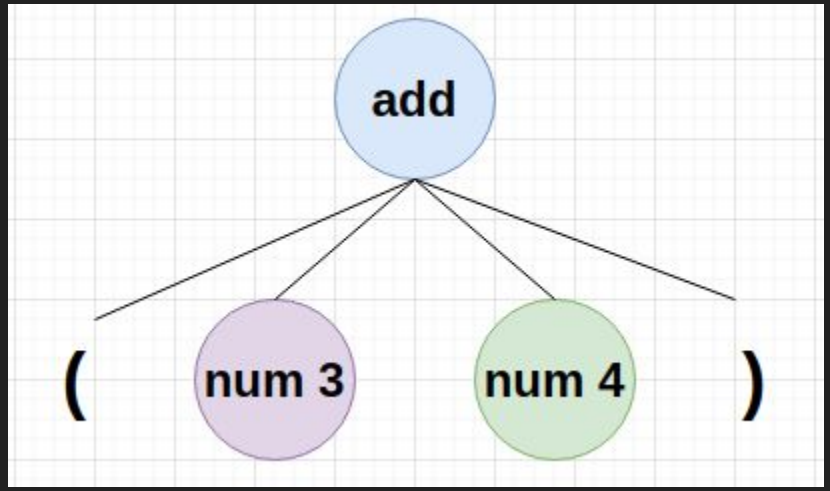


Tipos de árboles

Árbol de Sintaxis Abstracta, ASA.

Árbol de Sintaxis Concreta, ASC.

(add (num 3) (num 4))



Lenguaje Anfitrión y Lenguaje Objetivo

“Vamos a compilar código escrito en Java, el compilador está escrito en C”

Lenguaje anfitrión: C

Lenguaje objetivo: Java

“El intérprete de Scheme está escrito en LISP”

Lenguaje anfitrión: LISP

Lenguaje objetivo: Scheme

Paradigmas de lenguajes de programación

A. Imperativo

i. Estructurado
ii. Orientado a Objetos

B. Declarativo

iii. Funcional
iv. Lógico

Paradigmas de LP

1. Funcional
   a. Funciones o procedimientos.
   b. Haskell, Scheme, Racket, LISP, ...

2. Orientado a Objetos
   c. Objetos, clases, métodos, herencia, polimorfismo, ...
   d. C++, Ada, Java, Python, Ruby, Smalltalk,...

Paradigmas de LP

3. Estructurado

    a. Funciones, bloques.

    b. C, Pascal, Algol, FORTRAN, etc.

4. Lógico

    a. Hechos y reglas.

    b. PROLOG

Gracias