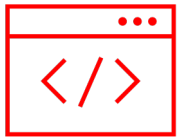
UNIVERSIDAD
NACIONAL
DE COLOMBIA

PROYECTO **CULTURAL, CIENTÍFICO Y COLECTIVO** DE NACIÓN



Generalidades

Prof. Oscar Mauricio Salazar Ospina
Correo: omsalazaro@unal.edu.co

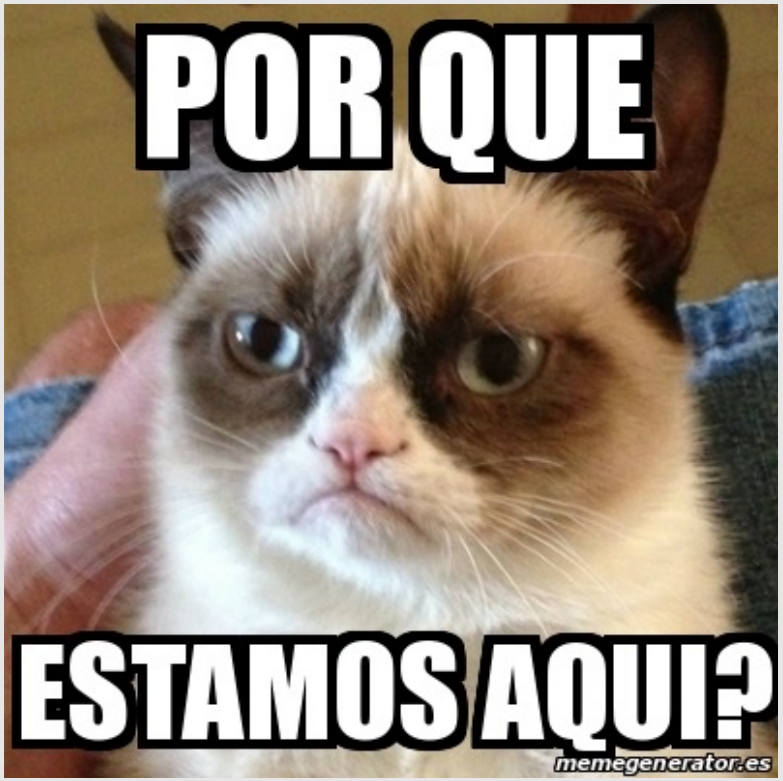
3007743 - Programación Lógica y Funcional
3010426 - Teoría de Lenguajes de Programación

Facultad de Minas
Departamento de ciencias de la computación y la decisión

Universidad Nacional de Colombia
PROYECTO **CULTURAL, CIENTÍFICO Y COLECTIVO** DE NACIÓN

Generalidades

Teoría de lenguajes de programación



Generalidades

Teoría de lenguajes de programación

Rama de las **ciencias** de la **computación**

Generalidades

Teoría de lenguajes de programación

Rama de las **ciencias** de la **computación**

Diseño, implementación, análisis, caracterización y **clasificación** de lenguajes de programación y sus **características**.

Generalidades

Teoría de lenguajes de programación

Rama de las **ciencias** de la **computación**

Diseño, implementación, análisis, caracterización y **clasificación** de lenguajes de programación y sus **características**.

Campo multidisciplinar: matemáticas, ingeniería del software, lingüística, ciencias cognitivas, etc.

Generalidades

Teoría de lenguajes de programación

Rama de las **ciencias** de la **computación**

Diseño, implementación, análisis, caracterización y **clasificación** de lenguajes de programación y sus **características**.

Campo multidisciplinar: matemáticas, ingeniería del software, lingüística, ciencias cognitivas, etc.

Símbolo representativo, modelo computacional.

Generalidades

Teoría de lenguajes de programación

Rama de las **ciencias** de la **computación**

Diseño, implementación, análisis, caracterización y **clasificación** de lenguajes de programación y sus **características**.

Campo multidisciplinar: matemáticas, ingeniería del software, lingüística, ciencias cognitivas, etc.

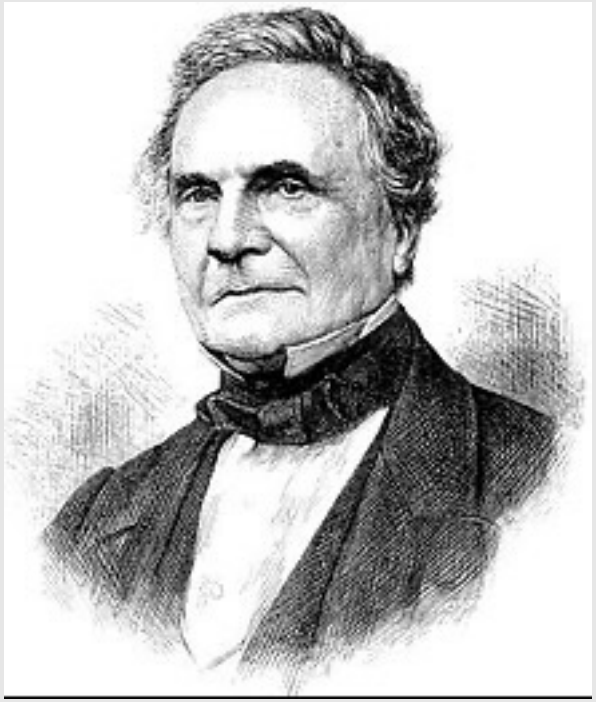
Símbolo representativo, modelo computacional.

¿Cómo **hablar** con un computador?.

Generalidades

Teoría de lenguajes de programación - Historia

1833 – Charles Babbage, Ada Lovelace Primer programa, nunca se construyó la máquina de Babbage, primer computador.



Generalidades

Teoría de lenguajes de programación - Historia

1930 – cálculo Lambda (Alonzo Church y Stephen Kleene)
Base de la programación funcional

Generalidades

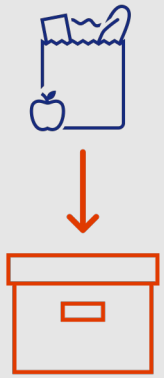
Teoría de lenguajes de programación - Historia

1930 – cálculo Lambda (Alonzo Church y Stephen Kleene)
Base de la programación funcional

Generalidades

Teoría de lenguajes de programación - Historia

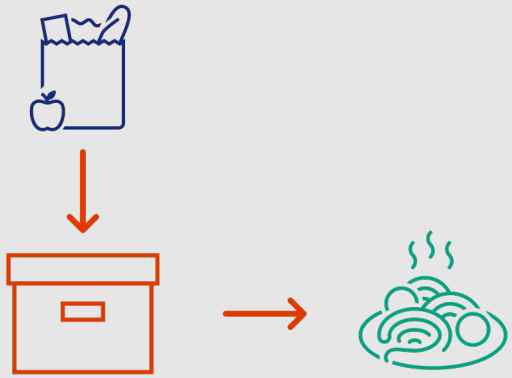
1930 – cálculo Lambda (Alonzo Church y Stephen Kleene)
Base de la programación funcional



Generalidades

Teoría de lenguajes de programación - Historia

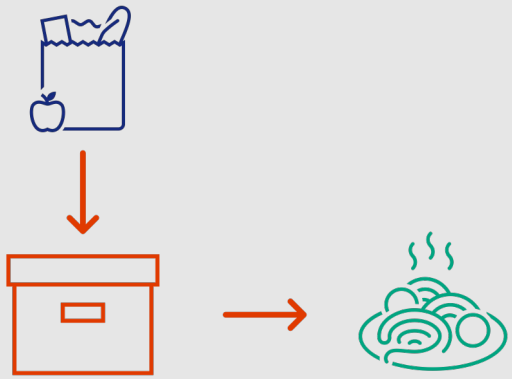
1930 – cálculo Lambda (Alonzo Church y Stephen Kleene)
Base de la programación funcional



Generalidades

Teoría de lenguajes de programación - Historia

1930 – cálculo Lambda (Alonzo Church y Stephen Kleene)
Base de la programación funcional

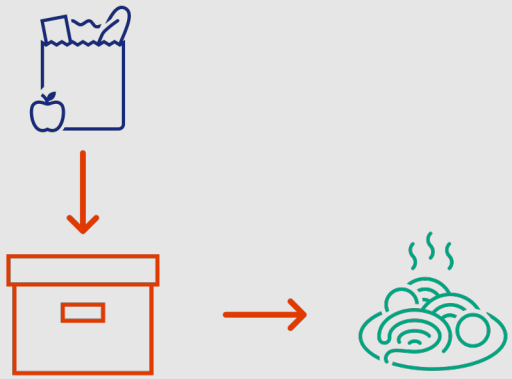


$$f_{(x)} = x^3$$

Generalidades

Teoría de lenguajes de programación - Historia

1930 – cálculo Lambda (Alonzo Church y Stephen Kleene)
Base de la programación funcional



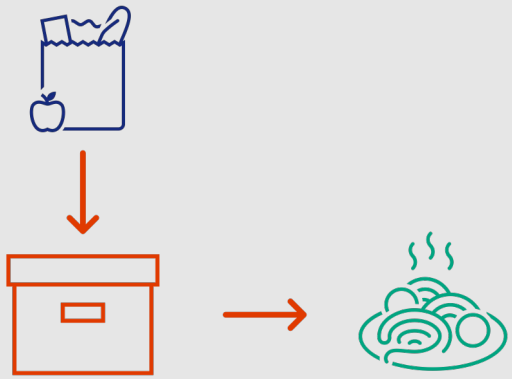
$$f_{(x)} = x^3$$

$$\lambda x.\ x^3$$

Generalidades

Teoría de lenguajes de programación - Historia

1930 – cálculo Lambda (Alonzo Church y Stephen Kleene)
Base de la programación funcional



$$f_{(x)} = x^3$$

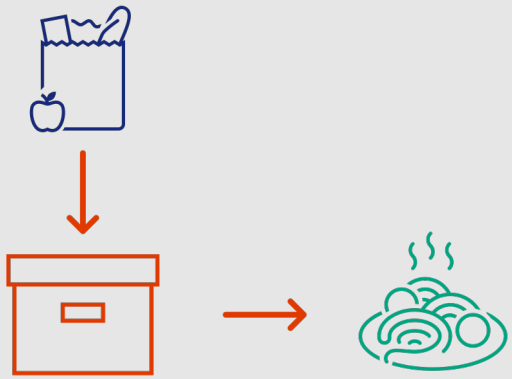
Operador — Variable — Expresión

$$\lambda x.\ x^3$$

Generalidades

Teoría de lenguajes de programación - Historia

1930 – cálculo Lambda (Alonzo Church y Stephen Kleene)
Base de la programación funcional



$$f_{(x)} = x^3$$

Operador — Variable — Expresión

$$\lambda x.\ x^3$$

Funciones anónimas

Generalidades

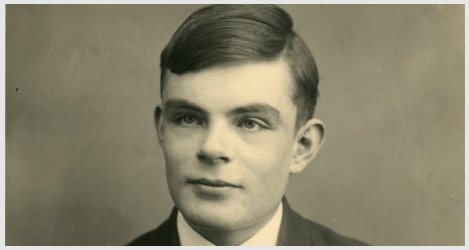
Teoría de lenguajes de programación - Historia

1930 – cálculo Lambda (Alonzo Church y Stephen Kleene)
Base de la programación funcional

El **objetivo** era **modelar** la **computación, NO** ser medio de **comunicación** para **programadores.**

Generalidades

Teoría de lenguajes de programación - Historia



1940 – Plankalkül (Konrad Zuse)
1972

1946 – ENIAC, Tarjetas perforadas, lenguaje ensamblador

1952 – Grace Hooper Ensamblador - Compilador

1954-1957 – Fortran (IBM – John Backus) – Formula Translating

1959 COBOL – Portable (43% de los sistemas bancarios de USA)

ALGOL58

1960 – LISP (MIT – John McCarthy) Académico – Listas - IA

Generalidades

Teoría de lenguajes de programación – Paradigmas de programación

Los **paradigmas** son los diferentes **estilos** de **utilizar** la **programación** para **resolver** un **problema**.

Programación imperativa

Programación declarativa

Generalidades

Teoría de lenguajes de programación – Paradigmas de programación

Los **paradigmas** son los diferentes **estilos** de **utilizar** la **programación** para **resolver** un **problema**.

Programación imperativa

Programación declarativa

- Detallado
- Explícito
- Paso a paso

```
list = []
count = 1while i < 10:
    list.append(i)
    i += 1
```

Generalidades

Teoría de lenguajes de programación – Paradigmas de programación

Los **paradigmas** son los diferentes **estilos** de **utilizar** la **programación** para **resolver** un **problema**.

Programación imperativa

- Detallado
- Explícito
- Paso a paso

```
list = []
count = 1while i < 10:
    list.append(i)
    i += 1
```

Programación declarativa

- Prioriza la claridad del resultado
- Simple

```
list(range(1, 10))
```

Generalidades

Teoría de lenguajes de programación – Paradigmas de programación

¿Cuál es mejor paradigma de programación?

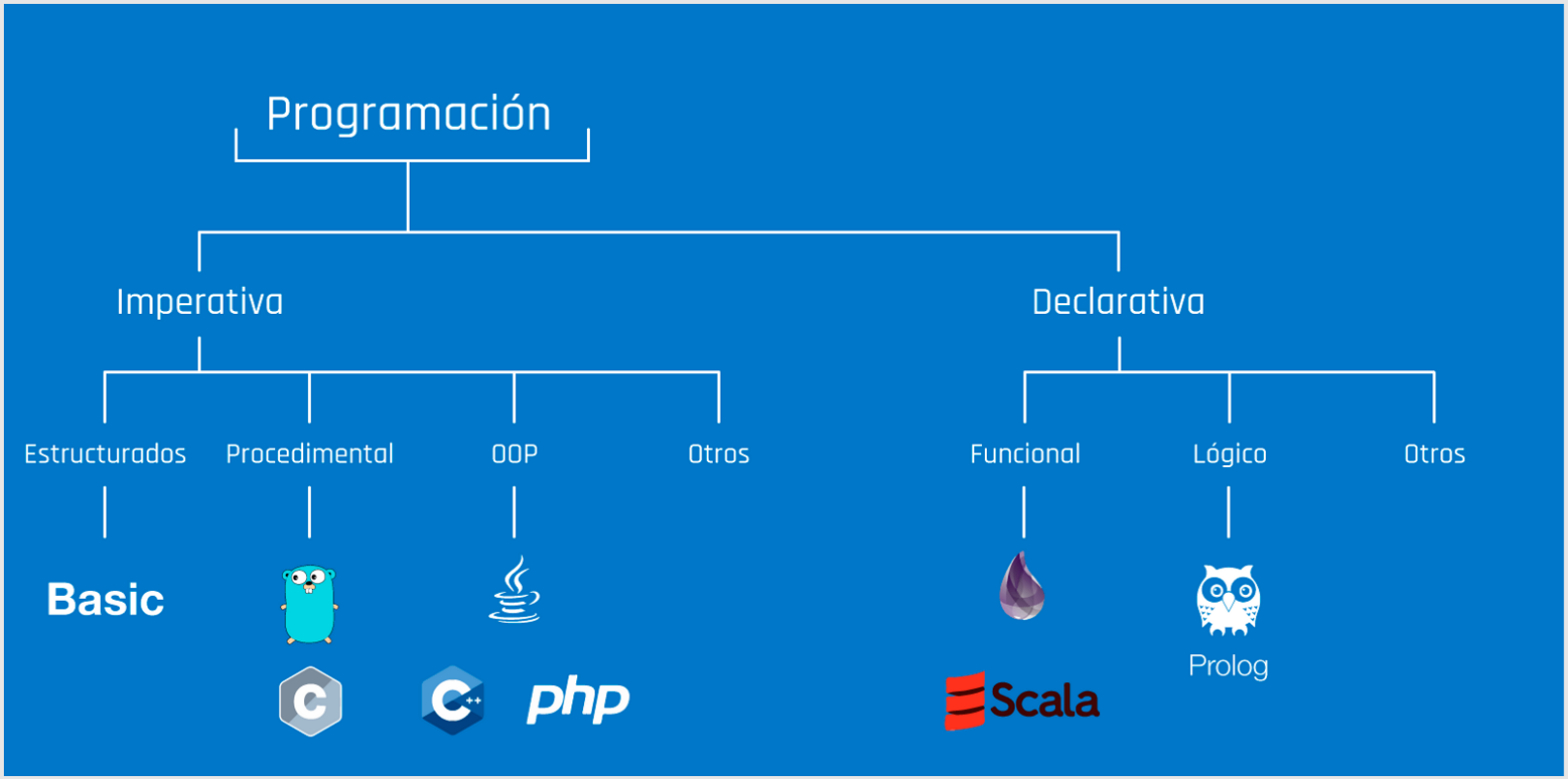
Generalidades

Teoría de lenguajes de programación – Paradigmas de programación

¿Cuál es mejor paradigma de programación?

Generalidades

Teoría de lenguajes de programación – Paradigmas de programación



Generalidades

Teoría de lenguajes de programación – Lenguajes compilados vs interpretados

Tanto **compiladores** como **interpretadores** son **programas** que **convierten** el **código** del programa a código objeto o **lenguaje de máquina**.

```
tope = int(input("¿Tope máximo? "))
def esprimo(n):
        for primo in range(2,n-1):
            if n % primo == 0:
                return False
        return True
print(list(filter(esprimo, range(1,tope))))
```

Lenguaje **de alto nivel** que entiende el programador (**programa fuente**)

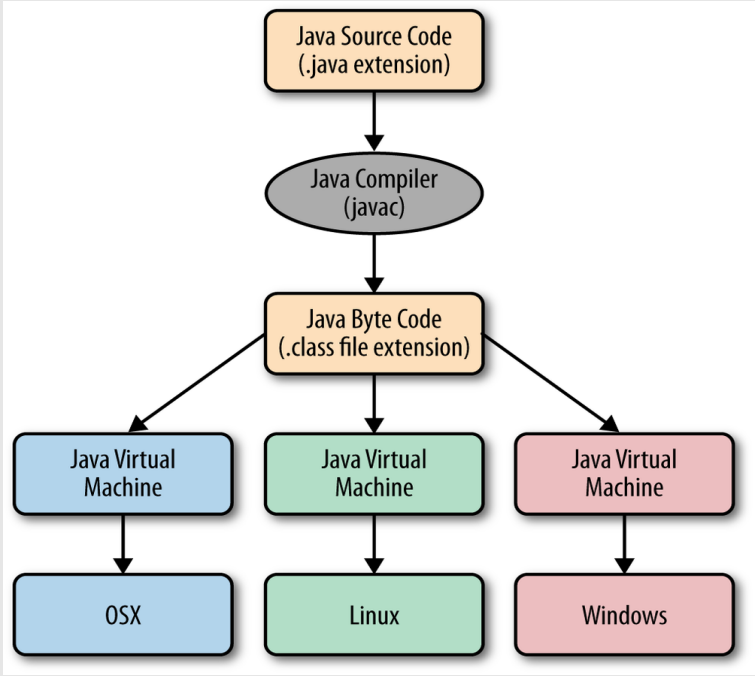
Lenguaje de máquina que entiende el procesador (código objeto)

Generalidades

Teoría de lenguajes de programación – Lenguajes compilados vs interpretados

La **compilación** es el proceso de **conversión** de código de un lenguaje de **alto nivel** (Java, Scala, etc.) en otro, entendible por la **máquina**, en un paso **previo a su ejecución**.

Los programas en **Java** se convierten en **códigos de bytes**.



Generalidades

Teoría de lenguajes de programación –
Lenguajes compilados vs interpretados

La **versión** más **directa** y **tangible** de la **compilación** es aquella que nos **produce** un **código objeto binario ejecutable** como salida (ejm. Scala, C, C++)


```
1  def factorial(x: BigInt): BigInt =
2  if (x == 0) 1 else x * factorial(x - 1)
3  println(factoria(6))
4
```

→

```
ScalaFiddle.scala:3: error: not found: value factoria
  println(factoria(6))
          ^
```

↓ Editar, corregir

```
1  def factorial(x: BigInt): BigInt =
2  if (x == 0) 1 else x * factorial(x - 1)
3  println(factorial(6))
4
```

←

```
720
```

Generalidades

Teoría de lenguajes de programación – Lenguajes compilados vs interpretados

Un **lenguaje interpretado** se caracteriza por ser **convertido** a un **lenguaje de máquina** a **medida que es ejecutado** (Ruby, Python y JavaScript)

```
In [2]: print("Hola Mundo")
        print('letras de un texto')
        for i in "TEXTO":
            print(i)
            i = 1
        print(lograd0)
        println('FIN')
```

```
Hola Mundo
letras de un texto
T
E
X
T
O

---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-2-b35dda80017a> in <module>
      4     print(i)
      5     i = 1
----> 6 print(lograd0)
      7 println('FIN')

NameError: name 'lograd0' is not defined
```

Generalidades

Teoría de lenguajes de programación –
Lenguajes compilados vs interpretados

Ventajas y desventajas

- El **ciclo** de **desarrollo del programa** (tiempo transcurrido entre escribir el código y probarlo) es más **rápido** en un lenguaje **interpretado**.
- En los **lenguajes compilados** es necesario realizar el **proceso de compilación** cada vez que se **cambia** el código **fuente**.
- Una **desventaja** de un **lenguaje compilado** es que cuando se **compila** un programa se deben **crear ejecutables** para cada uno de **los sistemas operativos** en los que se va a utilizar (Un ejecutable creado para Linux no va a servir en Windows).
- Un **lenguaje compilado** es mucho más **rápido** que uno **interpretado**, debido a que cuando es ejecutado **ya se encuentra en código de máquina** y eso también le permite hacer algunas **optimizaciones** que no son posibles con un lenguaje interpretado.

Generalidades

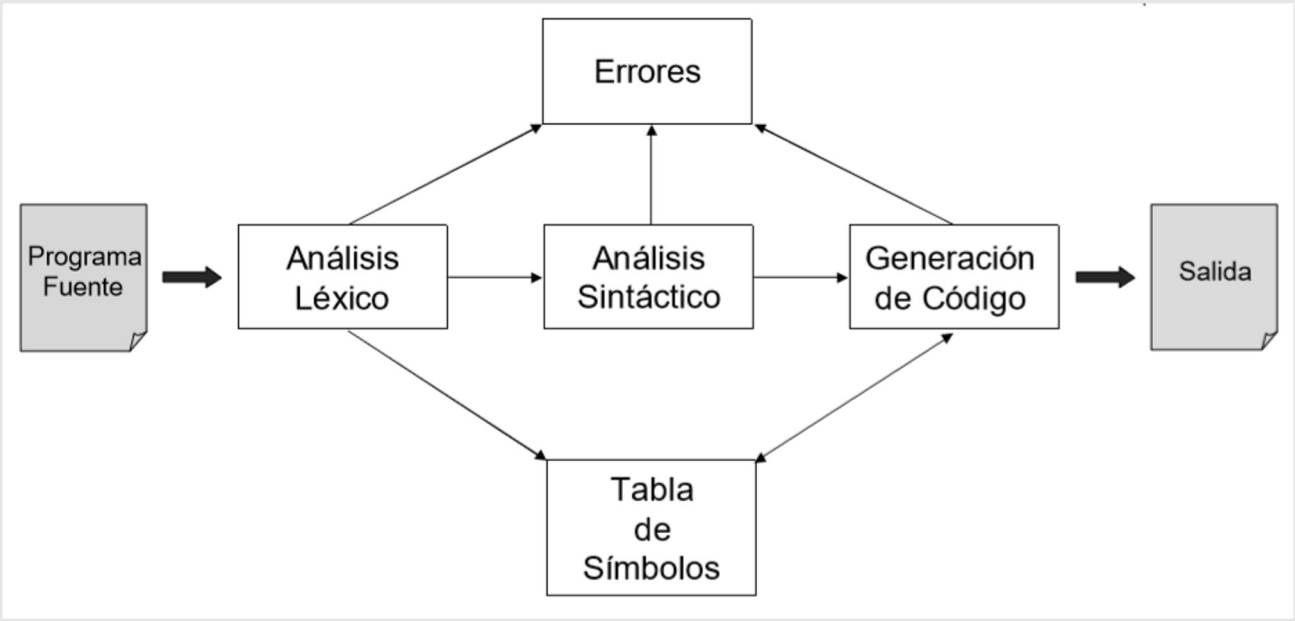
Teoría de lenguajes de programación –
Lenguajes compilados vs interpretados

Ventajas y desventajas

- Una **desventaja** de un **lenguaje interpretado** es que, para ser ejecutado, se debe **tener instalado el interpretador**. Esto no es necesario en un lenguaje compilado que es convertido a lenguaje de máquina.
- Un lenguaje **compilado** está **optimizado** para el momento de la **ejecución**.
- Un lenguaje **interpretado** está optimizado para hacerle la **vida más fácil** al **programador**.

Generalidades

Teoría de lenguajes de programación – fases de compilación



Generalidades

Teoría de lenguajes de programación – fases de compilación

Análisis Léxico: Primera etapa de la compilación en donde **se lee el programa fuente**, se remueven los espacios en blanco, tabulaciones, saltos de línea, se remueve los comentarios y se agrupan los caracteres en unidades llamadas ***tokens***.

Tokens

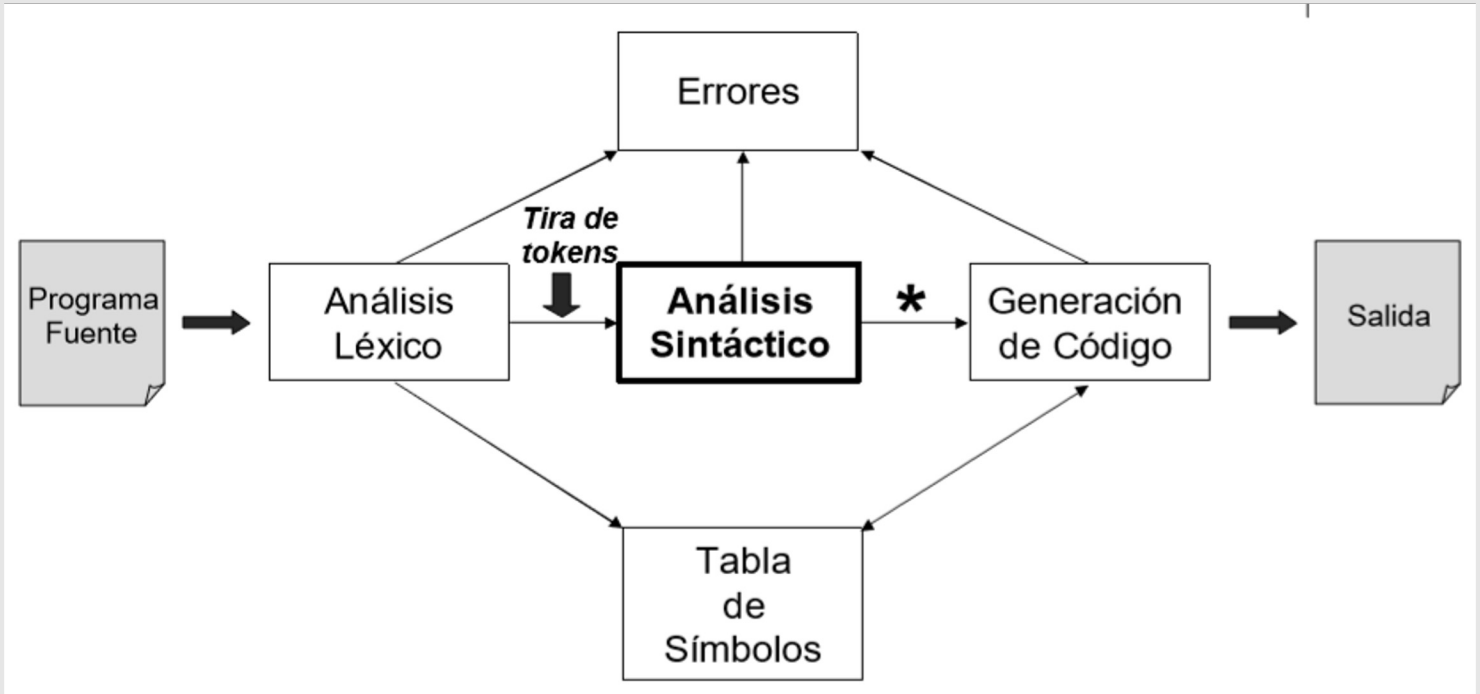
- Palabras reservadas (IF, THEN, ELSE)
- Operadores (‘+’, ‘>=‘, ‘:=‘)
- Cadenas de múltiples caracteres (Identificador, Constante, etc.)

Token	Identificación del token
ID	27
CTE	28
IF	59
THEN	60
ELSE	61
+	70
/	73
>=	80
:=	85

Lexemas

Generalidades

Teoría de lenguajes de programación – fases de compilación



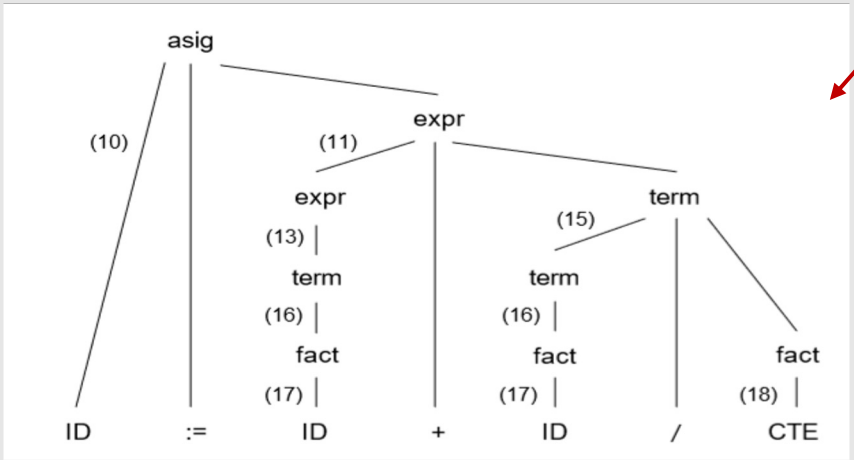
Nota. Cuando el **análisis léxico** detecta un ***token*** de tipo **identificador** (ejm. Variable nomina), lo **ingresa** en la **Tabla de Símbolos**.

Generalidades

Teoría de lenguajes de programación – fases de compilación

Análisis Sintáctico: En esta fase se toma el conjunto de ***tokens*** producidos por la fase de **análisis léxico** y se **genera un árbol de sintaxis** (**parsing**). Se valida si el árbol producido es sintácticamente correcto.

Total := Base + Recargo / 100
ID := ID + ID / CTE



Generalidades

Teoría de lenguajes de programación – fases de compilación

Análisis Semántico: valida si el **árbol sintáctico** construido **concuerda** con las **reglas del lenguaje formal**. Por ejemplo, asignaciones de valores entre tipos de datos que son compatibles.

En esta fase también es muy necesario mantener un control de los identificadores con sus respectivos tipos y expresiones, por ejemplo si una **variable es asignada sin haber sido declarada**, y produce como salida un árbol de sintaxis anotado.

Generalidades

Teoría de lenguajes de programación – fases de compilación

Generación de código intermedio: el compilador genera un **código intermedio** entre el **código fuente** y el **código de la maquina** objetivo (unos y ceros). Este representa un programa para una **máquina abstracta**.

Este **código intermedio** debe ser generado de tal manera que es **fácilmente traducido** a un lenguaje de máquina de **bajo nivel**.

Generalidades

Teoría de lenguajes de programación – fases de compilación

Optimización: puede asumirse como algo que **quita líneas de código innecesarias**, y **ordena una secuencia de declaraciones** que aceleran la ejecución del programa **sin desperdiciar recursos** de CPU o memoria RAM.

```
int valorA = 0;
int valorB = 10;
int temp = valorA + ValorB;
```

Generalidades

Teoría de lenguajes de programación – fases de compilación

Optimización: puede asumirse como algo que **quita líneas de código innecesarias**, y **ordena una secuencia de declaraciones** que aceleran la ejecución del programa **sin desperdiciar recursos** de CPU o memoria RAM.

```
int valorA = 0;
int valorB = 10;
int temp = valorA + ValorB;
```

```
int temp = 0 + 10;
```

Generalidades

Teoría de lenguajes de programación – fases de compilación

Generación de código: toma la **versión optimizada** del código intermedio y se mapea a la **lenguaje** de **máquina objetivo**.

El **sistema operativo** tomara estas instrucciones y les **asignará un espacio** en la **memoria** para así poder funcionar.

Generalidades

Teoría de lenguajes de programación – fases de compilación

Ejemplo:

Fuente:

precio = costo + impuesto * 60

Generalidades

Teoría de lenguajes de programación – fases de compilación

Ejemplo:

Fuente:

precio = costo + impuesto * 60

↓ AL

id1 = id2 + id3 * 60

Generalidades

Teoría de lenguajes de programación – fases de compilación

Ejemplo:

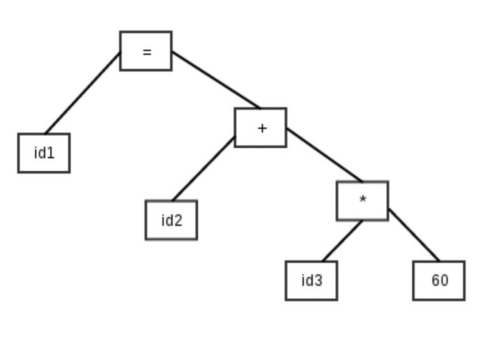
Fuente:

precio = costo + impuesto * 60

↓ AL

id1 = id2 + id3 * 60

↓ ASi



Generalidades

Teoría de lenguajes de programación – fases de compilación

Ejemplo:

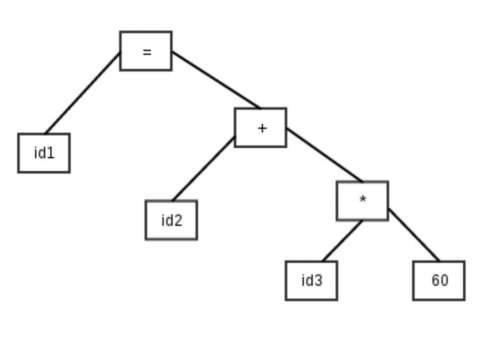
Fuente:

precio = costo + impuesto * 60

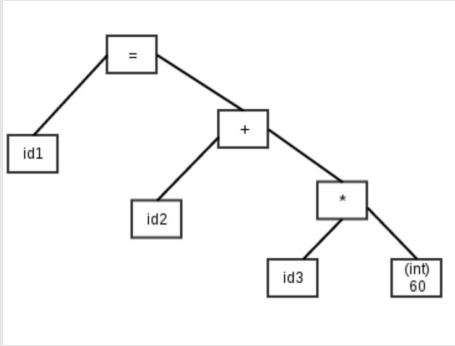
↓ AL

id1 = id2 + id3 * 60

↓ ASi



ASe →



Generalidades

Teoría de lenguajes de programación – fases de compilación

Ejemplo:

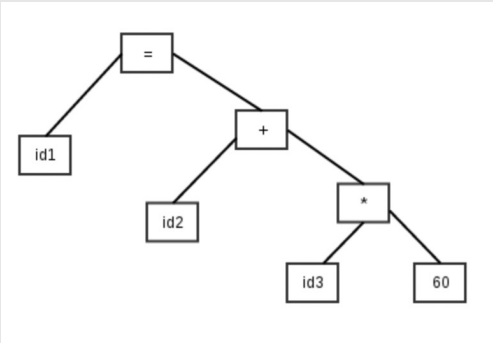
Fuente:

precio = costo + impuesto * 60

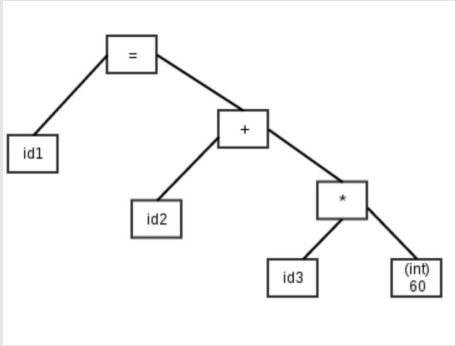
↓ AL

id1 = id2 + id3 * 60

↓ ASi



ASe →



Cod. Inter. ↑

temp1 = int(60)
temp2 = id3 * temp1
temp3 = id2 + temp2
id1 = temp3

Generalidades

Teoría de lenguajes de programación – fases de compilación

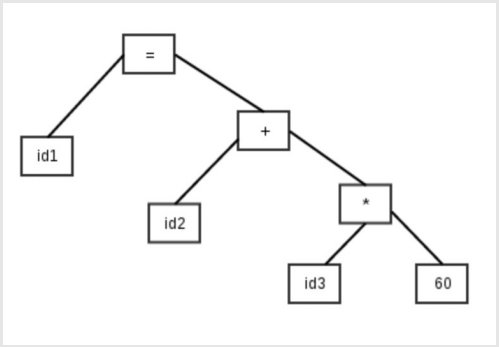
Ejemplo:

Fuente:
precio = costo + impuesto * 60

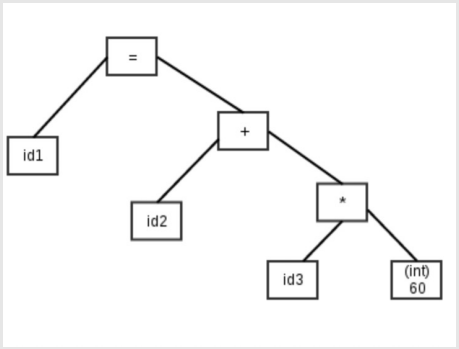
↓ AL

id1 = id2 + id3 * 60

↓ ASi



ASe →



Cod. Inter. ↑

temp1 = id3 * 60
id1 = id2 + temp1

Optimización ↑

temp1 = int(60)
temp2 = id3 * temp1
temp3 = id2 + temp2
id1 = temp3

Generalidades

Teoría de lenguajes de programación – fases de compilación

Ejemplo:

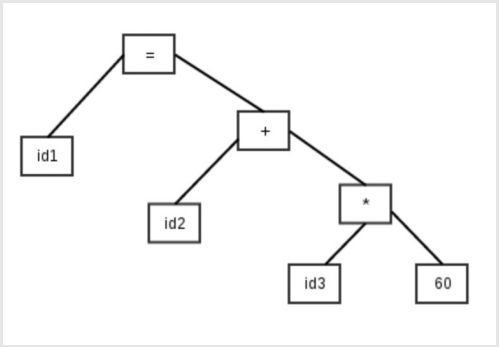
Fuente:

precio = costo + impuesto * 60

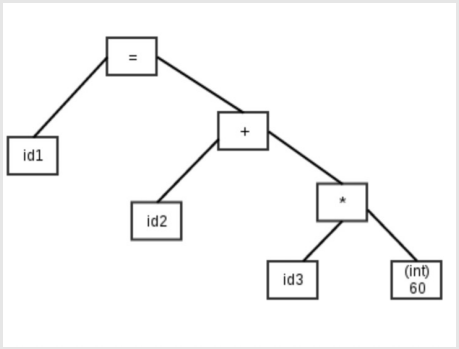
↓ AL

id1 = id2 + id3 * 60

↓ ASi



ASe →



Cod. Inter. ↑

temp1 = id3 * 60
id1 = id2 + temp1

Optimización ↑

temp1 = int(60)
temp2 = id3 * temp1
temp3 = id2 + temp2
id1 = temp3

Gen. Cod. ↘

MOVF id3, R2
MULF #60, R2
MOVF id2, R1
ADDF R2, R1
MOVF R1, id1

Código ensamblador

Gracias