
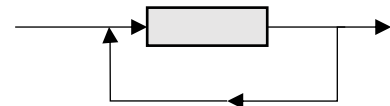
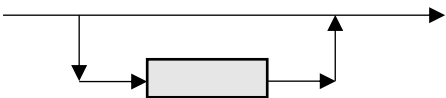


Definiciones teóricas y anotaciones para el parcial

## Practica 1

- [Resumen de la clase 1 de teoría.](#)

## Practica 2

Meta-símbolos utilizados por		Símbolo utilizado en Diagramas sintácticos	Significado
BNF	EBNF		
Palabra terminal	Palabra terminal	Ovalo	Definición de un elemento terminal.
<>	<>	Rectángulo	Definición de un elemento no terminal
::=	::=	Diagrama con rectángulos, óvalos y flechas	Meta-símbolo de definición que indica que el elemento a su izquierda se puede definir según el esquema de la derecha
	( )	Flecha que se divide en dos o más caminos	Meta-símbolo de opción que indica que puede elegirse uno y solo uno de los meta símbolos
< p > < p1 >	{ }		Repetición
	*		Repetición de 0 o más veces.
	+		Repetición de 1 o más veces.
	[ ]		Elemento optativo (puede o no estar).

*Nota: p y p1 son producciones simbólicas.*

### ANOTACIONES:

- Los espacios no los tomo en cuenta.
- Cuando se trata de sentencias o bloques, puedo asumir que ya están hechos, lo único que debo hacer es aclararlo en los no terminales
  - Ejemplo:

```

<bloque> ::= <sentencia> | <sentencia> <bloque> | <bloque> <sentencia> ;
<sentencia> ::= <sentencia_asignacion> | <llamada_a_funcion> | <sentencia_if> |
<sentencia_for> | <sentencia_while> | <sentencia_switch> }

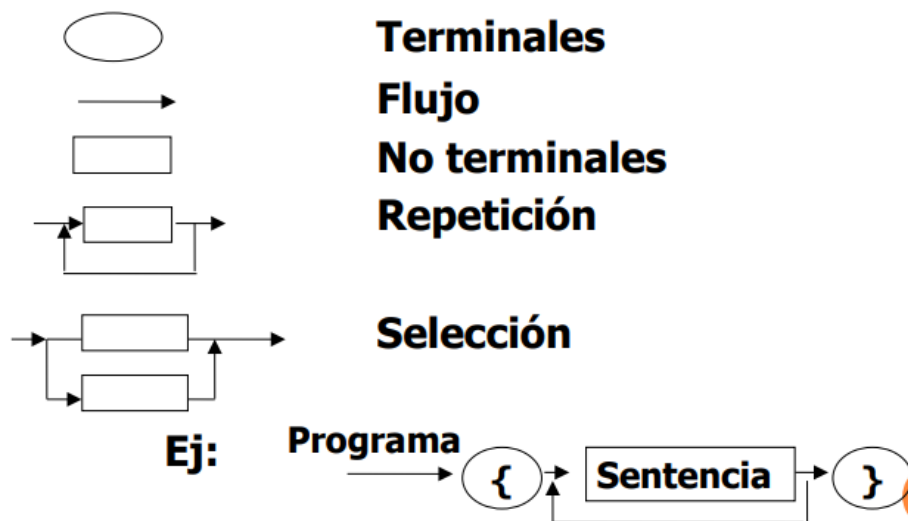
```

Deberían estar la <sentencia\_asignación> y las otras en no terminales. Lo que si debo tomar en cuenta es alguna regla del lenguaje como el ; después de una sentencia o algo de esa índole

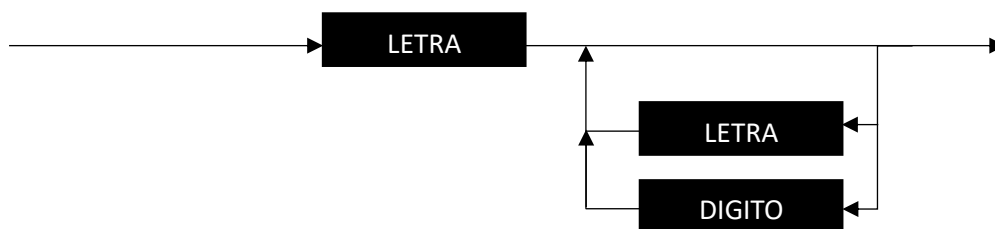
CONWAY

## Sintaxis

### ■ Diagramas sintácticos (CONWAY):



EJEMPLO:



RECORDAR → SINTAXIS ES SOLO PARA FORMAS VALIDAS

Un error sintáctico es cuando un código no se escribe de acuerdo a las reglas de escritura de un lenguaje

### Practica 3

- [Resumen de la clase 2 de teoría.](#)

### Practica 4

Variables y sus atributos

nombre: string de caracteres que se usa para referenciar a la variable.

alcance: rango de instrucciones en el que se conoce el nombre

tipo: valores y operaciones

l-value (tiempo de vida): lugar de memoria asociado con la variable (tiempo de vida)

r-value: valor codificado almacenado en la ubicación de la variable

**Analice en los lenguajes: Java, C, Python y Ruby las diferentes formas de inicialización de variables que poseen. Realice un cuadro comparativo de esta característica.**

Java	<ul style="list-style-type: none"><li>- Declaración e inicialización en la misma línea: <code>int num = 10;</code></li><li>- Declaración e inicialización separadas: <code>int num; num = 10;</code></li><li>- Inicialización por defecto: <code>int num;</code> (inicializada a 0 automáticamente)</li></ul>
C	<ul style="list-style-type: none"><li>- Declaración e inicialización en la misma línea: <code>int num = 10;</code></li><li>- Declaración e inicialización separadas: <code>int num; num = 10;</code></li><li>- Las variables globales y estáticas se inicializan por defecto a 0 si no se les asigna un valor explícitamente.</li><li>- Las variables locales no tienen un valor por defecto y su valor inicial es impredecible, a menos que se les asigne un valor explícitamente.</li></ul>
Python	<ul style="list-style-type: none"><li>- Declaración e inicialización en la misma línea: <code>num = 10</code></li><li>- No es necesario declarar el tipo de variable antes de su inicialización</li></ul>
Ruby	<ul style="list-style-type: none"><li>- Declaración e inicialización en la misma línea: <code>num = 10</code></li><li>- No es necesario declarar el tipo de variable antes de su inicialización</li></ul>

En resumen, Java y C tienen formas similares de inicialización de variables, mientras que Python y Ruby tienen una sintaxis más flexible y sencilla. En Python y Ruby no es necesario declarar el tipo de variable antes de su inicialización, lo que puede hacer que el código sea más fácil de escribir y entender. Por otro lado, Java y C no permiten la asignación de variables sin declarar su tipo previamente. Además, Java tiene la opción de inicialización por defecto, que es útil en ciertos casos, como cuando se desea inicializar un array.

El atributo l-valor de una variable hace referencia a su capacidad de ser referenciada o manipulada mediante una expresión de asignación. En otras palabras, una variable con l-valor puede ser asignada a otra variable o valor.

#### **a. Variable estática.**

Una variable estática es una variable que existe y se inicializa en tiempo de compilación y su valor persiste en memoria durante toda la ejecución del programa. Es decir, se asigna espacio al inicio del programa y se desaloca cuando termina. Puede llegar incluso a durar más que el programa.

#### **b. Variable automática o semiestática.**

Una variable automática o semiestática se declara dentro de un bloque y su ámbito de visibilidad se limita a ese bloque. Su valor se inicializa al entrar al bloque y se destruye al salir de él. Estas variables también se llaman locales, ya que solo son visibles dentro de la función en la que se declaran.

#### **c. Variable dinámica.**

Una variable dinámica es una variable que se crea y se destruyen explícitamente en tiempo de ejecución. En la mayoría de los lenguajes de programación, estas variables se crean utilizando funciones o métodos específicos como `malloc()` o `new`. El valor de una variable dinámica puede cambiar a lo largo de la ejecución del programa. Utilizan la memoria HEAP para alocarse.

#### **d. Variable semidinámica.**

Una variable semidinámica es aquella cuyo tamaño se fija durante la ejecución pero en la compilación se guarda algún tipo de información previa, como la ubicación en memoria. Un ejemplo de una variable semidinámica en ADA podría ser una matriz cuyo tamaño se puede ajustar en tiempo de ejecución, pero solo en un punto de control específico del programa.

ANOTACION: Un puntero consta de dos variables, la automática que es el puntero con nombre que conocemos y la dinámica que es una variable anónima cuyo tiempo de vida empieza cuando se inicializa (`new`)

En el lenguaje de programación C, se pueden clasificar las variables según su l-valor de la siguiente manera:

1. Variable estática: son variables que se almacenan en la memoria estática y mantienen su valor entre llamadas a funciones. Estas variables se declaran con la palabra clave "static" y se inicializan automáticamente en cero.
2. Variable automática: son variables que se almacenan en la memoria de la pila y se eliminan automáticamente al salir de su ámbito de definición. Estas variables se declaran sin la palabra clave "static" y su vida útil está limitada a su función o bloque de código.
3. Variable dinámica: son variables que se asignan en tiempo de ejecución y se almacenan en la memoria dinámica. Estas variables se crean utilizando las funciones "malloc" o "calloc" y se liberan utilizando la función "free". Estas variables tienen una vida útil que se extiende más allá de la función o bloque de código en el que se crean.

En el lenguaje de programación Ada, las variables se clasifican según su l-valor de la siguiente manera:

1. Variable estática: se definen usando la palabra clave "constant" o "static" y se mantiene en memoria durante toda la ejecución del programa.

2. Variable automática o semiestática: se definen dentro de un bloque y se eliminan de la memoria cuando el bloque termina.
3. Variable dinámica: se asignan en tiempo de ejecución usando la palabra clave "new". Estas variables se mantienen en la memoria hasta que se libera explícitamente usando la palabra clave "delete".
4. Variable semidinámica: una matriz cuyo tamaño se puede ajustar en tiempo de ejecución, pero solo en un punto de control específico del programa.

En **ADA**, la clasificación de las constantes en numéricas y comunes se debe a que las constantes numéricas son aquellas que se refieren a valores numéricos, mientras que las constantes comunes son aquellas que pueden contener caracteres y símbolos, como cadenas de texto.

En **JAVA** los identificadores se pueden clasificar:

Globales (variables estáticas de clase, aquellos que se declaran a nivel de la clase y pueden ser accedidos sin necesidad de crear una instancia de la clase)

Variables de instancia (no estáticas, son variables de instancia que se declaran dentro de la clase y solo pueden ser accedidas a través de una instancia de la clase)

Locales (son aquellos que se declaran dentro de un método y solo pueden ser accedidos dentro del mismo)

La diferencia entre variables globales, variables de instancia y variables locales en Java se basa en su alcance y su duración. Las variables globales son visibles en toda la clase y tienen una duración de toda la vida de la aplicación, mientras que las variables de instancia son específicas de cada objeto y duran tanto como el objeto exista. Por último, las variables locales solo son visibles en el método o bloque de código en el que se declaran y tienen una duración limitada a la ejecución de ese método o bloque.

**El nombre de una variable puede condicionar:**

**a) Su tiempo de vida:** El nombre de una variable no condiciona su tiempo de vida.

**b) Su alcance:** El nombre una variable puede condicionar su alcance de, ya que el alcance de una variable es el rango de instrucciones en el que se conoce el nombre.

**c) Su r-valor:** El nombre de una variable no condiciona su r-valor, ya que el r-valor se determina por la asignación de un valor a la variable (al nombre). Lo que si permite el nombre es poder acceder, mediante ese nombre, a la variable.

**d) Su tipo:** El nombre de una variable no condiciona directamente su tipo, ya que el tipo se define explícitamente en la declaración de la variable. Sin embargo, es importante elegir un nombre descriptivo para la variable que refleje el tipo de dato que almacena, lo que puede hacer que sea más fácil para los programadores entender y trabajar con el código que utiliza la variable

#### ANOTACIONES:

- Si se tiene un puntero, se tienen dos variables (generalmente)
  - Una automática que sería el puntero (el que almacena el l-valor de la otra variable). Su tiempo de vida es como una variable normal, es decir, el tiempo de vida del bloque que la contiene.
  - Una dinámica, la cual es anónima, su l-valor es almacenado por la automática. Su tiempo de vida es desde el new.
    - Según lo que pregunte en práctica cuando una automática apunta a otra variable igual esta la variable anónima (medio raro se me hace pero bueno xd)
  - Si dos automáticas apuntan a la misma variable, habrá igualmente DOS variables anónimas, con sus respectivos tiempos de vida (sería como el tiempo de vida del enlace con la heap)
- El tiempo de vida de las variables es desde el inicio del bloque (excepto si es una static, el tiempo de vida siempre es todo el programa, y mas)
- El alcance de una función es desde donde la declaro hasta abajo.
- El tiempo de vida de una función básicamente es toda su definición
- Si se trata de 2 programas en C que están compilando al mismo tiempo como por ejemplo:

## **ARCHIVO1.C**

```
1.      int v1;
2.      int *a;
3.      Int fun2 ()
4.      { int v1, y;
5.          for(y=0; y<8; y++)
6.          { extern int v2;
7.              ...}
8.      }
9.      main()
10.     {static int var3;
11.     extern int v2;
12.     int v1, y;
13.     for(y=0; y<10; y++)
14.     { char var1='C';
15.       a=&v1;}
16.     }
```

## **ARCHIVO2.C**

```
17.     static int aux;
18.     int v2;
19.     static int fun2( )
20.     { extern int v1;
21.       aux=aux+1;
22.       ...
23.     }
24.     int fun3( )
25.     { int aux;
26.       aux=aux+1;
27.       ...
28.     }
```

- Las variables globales (todas) tendrán tiempo de vida del inicio al final (1-28).
  - Línea 1 int v1 1-28
  - Línea 2 \*a 1-28
  - Línea 17 static int aux línea <1-28> → es global y estática
  - Línea 18 int v2 1-28
- Las variables estáticas tendrán tiempo de vida desde el inicio del programa hasta el final.
  - Línea 10 static int var3 tendrá tiempo de vida desde <1-28>

## Practica 5

### Modelo de registro de activación

Head (prog principal)
Pto retorno
EE (enlace estático)
ED (enlace dinámico)
Variables...
...
Parámetros ...
....
Procedimientos ....
....
Funciones ...
....
Valor de retorno

- HEAD
  - Current: Dirección base del registro de activación de la unidad que se esté ejecutando actualmente
  - Free: Próxima dirección libre en la pila
- Punto de retorno: la siguiente dirección a ejecutar después de que termine la subrutina



- Enlace estático: apunta al registro de activación de la unidad que estáticamente la contiene
- Enlace dinámico: Contiene un puntero a la dirección base del registro de activación de la rutina llamadora
- Variables: las variables definidas dentro de la unidad.
- Procedimientos: los procedimientos definidos dentro de la unidad.
- Funciones: las funciones definidas dentro de la unidad.
- Valor de retorno: Al terminar una rutina se desaloca su RA, por lo tanto la rutina llamante debe guardar en su RA el valor de retorno de la rutina llamada.

Si no tengo una variable y tengo que buscarla:

- Si se trata de cadena estática, voy al RA del que me contiene a buscar la variable (sigo enlace estático).
- Si se trata de cadena dinámica, voy al RA del que me llamo a buscar la variable (sigo enlace dinámico).

TIP → Hacer árbol sintáctico determinando las estructuras y quien contiene a quien