

Repaso - Tips para el primer parcial

Practica 1:

Es mas de definiciones

Atributos de diseño de lenguajes de programacion - Atributos de calidad de lenguajes de programacion

◆ Ortogonalidad

Definición:

La ortogonalidad se refiere a que **los componentes del lenguaje pueden combinarse entre sí sin restricciones inesperadas**. Es decir, **una pequeña cantidad de conceptos se combinan de manera coherente** para producir muchas funcionalidades.

Ejemplo práctico:

Si en un lenguaje una estructura como un `if` puede usarse dentro de cualquier otra estructura (`while`, `for`, funciones, etc.) sin limitaciones especiales, el lenguaje es ortogonal.

Ventaja:

Reduce excepciones y hace el lenguaje más predecible y fácil de aprender.

◆ Expresividad

Definición:

La expresividad indica **qué tan fácilmente se pueden expresar ideas complejas** en pocas líneas de código de forma clara.

Ejemplo práctico:

En Python, escribir `sum(lista)` es muy expresivo comparado con escribir un bucle para sumar manualmente los elementos.

Ventaja:

Permite escribir código conciso, claro y potente.

◆ Legibilidad

Definición:

Es la **facilidad con la que otros (o uno mismo en el futuro) pueden leer y entender el código**.

Ejemplo práctico:

Una variable llamada `totalVentas` es más legible que `tvx`.

Ventaja:

Es fundamental para el mantenimiento del código y el trabajo en equipo.

◆ Simplicidad

Definición:

La simplicidad se refiere a que el lenguaje **tiene pocos conceptos pero bien definidos**, sin mucha "magia" oculta ni reglas complicadas.

Ejemplo práctico:

Lenguajes como Go tienen una sintaxis simple y pocas estructuras, lo que facilita su aprendizaje.

Ventaja:

Reduce la curva de aprendizaje y los errores al programar.

✂ ¿Qué son los **Applets**?

Definición:

Un **applet** es un pequeño programa Java que se ejecuta dentro de una **página web en un navegador**. Son aplicaciones **del lado del cliente**, es decir, se ejecutan en la computadora del usuario.


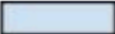



🌐 ¿Qué son los **Servlets**?

Definición:

Un **servlet** es un programa Java que **se ejecuta en un servidor web** y responde a peticiones, usualmente **HTTP**. Son componentes del **backend**, es decir, del lado del servidor.

Practica 2:

Practica, vemos BNF - EBNF, diagramas sintacticos (conway)

Meta símbolos utilizados por		Símbolo utilizado en Diagramas sintácticos	Significado
BNF	EBNF		
palabra terminal	palabra terminal		Definición de un elemento terminal
< >	< >	rectángulo 	Definición de un elemento no terminal
::=	::=	diagrama con rectángulos, óvalos y flechas	Es un metasímbolo y define una producción / regla
	()	flecha que se divide en dos o más caminos	Selección de una alternativa
< p > < p1 >	{ }	flecha que vuelve a la condición	Repetición
	*		Repetición de 0 o más veces
	+		Repetición de 1 o más veces
	[]		Opcional, está presente o no lo está

Apuntes que tengo de esta practica

$G = (N, T, S, P)$

N = Símbolos no terminales => Expresiones

$N = \{ \text{<numero_entero>, <digito> } \}$

ej:

T = Símbolos terminales

ej:

$T = \{ 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 \}$

S = Punto de partida, desde el cual se generan todas las expresiones.

$S = \text{<numero_entero>}$

P = Conjunto de producciones

$\text{<numero_entero> ::= <digito> | <digito> \text{<numero_entero>}$
 $\text{<digito> ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9}$

Es primordial entender la diferencia entre BNF y EBNF

TIPO	
BNF	Tiene recursion
EBNF	NO tiene recursion. Es repetitivo/iterativo.

Practica 3:

Sintaxis:

La sintaxis define las reglas estructurales que determina como escribir expresiones, instrucciones y programas validos. Si un codigo no cumple la sintaxis del lenguaje el compilador o interprete mostrara un error y no lo ejecutara.

Reglas sintacticas:


Define como debe estar estructurado el codigo de un lenguaje para que sea valido. Estan formadas por:

- Simbolos no terminales
- Simbolos terminales
- Simbolos especiales

Ejemplo de regla sintáctica (en BNF)

Supongamos una pequeña parte de la sintaxis de un lenguaje ficticio:

bnf

 Copiar código

```
<instruccion> ::= "if" <expresion> "then" <instruccion> | "print" <expresion>
<expresion>    ::= <numero> | <identificador>
<numero>       ::= "0" | "1" | "2" | ... | "9"
<identificador> ::= "x" | "y" | "z"
```

Reglas lexicograficas

- Se refiere a la formacion de palabras validas en un lenguaje.
- Define como se escriben identificadores, numeros, operadores y palabras claves.
- Se describe mediante expresiones regulares o automatas finitos.

Palabras reservadas:

Son identificadores con un significado especial en el lenguaje y no pueden usarse como nombres de variables, funciones o clases.

Que define la semantica? y la sintaxis?

La semantica define el significado de las construcciones del lenguaje. Mientras que la sintaxis describe como deben estructurarse correctamente las expresiones.

Errores sintacticos vs Errores semanticos

Error Sintactico:

Ocurren cuando no se respetan las reglas gramaticales establecidas por el lenguaje. Es decir, cuando hay un error en la forma que se escribio el codigo y por eso el compilador no puede interpretarlo.

Estos errores se detectan en tiempo de compilacion.

```
// Ejemplo (1) Error de sintaxis en tipo de variable
integ numero3 = 10;
// ✗ Error sintáctico: 'integ' no es una palabra reservada válida.
// ✔ Corrección:
int numero3 = 10;

// Ejemplo (2) Error de sintaxis en condicion for.
for (i:= 1; i<10) i++ {...}
// ✗ Error sintáctico: uso incorrecto de la sintaxis del 'for'.
// ✔ Corrección:
for (int i=1; i<10; i++){....}
```

Error Semantico:

Ocurren cuando el codigo esta bien escrito, es decir, es sintacticamente valido. Pero hay un error de logica. Los errores semanticos estan asociados con errores de logica en el codigo.

Estos errores se detectan en tiempo de ejecucion. Ya que el codigo es valido sintacticamente, pero falla su comportamiento de forma logica.


```
// Ejemplo (1) Error de logica en condicion del for.
for (int i=0; i>-10; i++) {...}
// ✗ Error Semantico: Por mas de que sintacticamente sea correcto. Este for
```

```
tiene un error de logica, no terminaria nunca.
```

```
//  Corrección:
```

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

Conclusion entre error sintactico y error semantico

 En resumen, los errores sintácticos impiden que el código se compile, mientras que los errores semánticos permiten que el programa se ejecute, pero con un comportamiento incorrecto o inesperado.

¿Qué significa compilar un programa

Compilar un programa significa traducir el código fuente escrito en un lenguaje de alto nivel (como C, java, python) a un lenguaje de máquina o código intermedio que la computadora pueda ejecutar.

Pasos necesarios para compilar un programa:

- Primer paso: Etapa de análisis.
 - Análisis léxico: Se detectan errores como caracteres inválidos o estructuras incorrectas.
 - Análisis sintáctico: Verifica si la estructura del código sigue las reglas gramaticales del lenguaje. Detecta errores como parentesis mal cerrados o estructuras incorrectas.
 - Análisis semántico: Debe pasar bien el análisis léxico y sintáctico.
 - Su importancia radica en asegurar que el código no solo sea sintácticamente correcto, sino que también tenga sentido lógico.
 - Generar código intermedio: Implica transformar el código fuente en una representación de un código intermedio para una máquina abstracta.
- Segundo paso: Etapa de síntesis.
 - Construye el programa ejecutable y genera el código necesario.
- Tercer paso: Semántica.

Es lo mismo compilar un programa que interpretarlo?

No es lo mismo compilar un programa que interpretarlo, son dos conceptos totalmente distintos.


Compilar: Traduce todo el código fuente a un lenguaje máquina (binario) antes de ejecutarlo. Los errores se detectan antes de la ejecución, en la fase de compilación.

Interpretar: No traduce el código completo, sino que lo lee y lo ejecuta línea por línea. Traduce la línea a lenguaje máquina y la ejecuta inmediatamente. Repite el proceso hasta completar la ejecución del programa. Los errores se detectan en tiempo de ejecución, lo que puede causar fallos inesperados

 Diferencia clave:

Compilado: El código se traduce una sola vez y luego se ejecuta.

Interpretado: Se traduce y ejecuta al mismo tiempo, línea por línea.

 Ejemplo de lenguajes compilados: C, C++, Rust.

 Ejemplo de lenguajes interpretados: Python, JavaScript, Bash

Binding

◆ Definición:

La ligadura (o binding) es el proceso mediante el cual se asocia un identificador (como una variable, función o método) con una entidad concreta en la memoria o en el código.

◆ Importancia: La ligadura es crucial para definir el comportamiento semántico de un programa, ya que determina cuándo y cómo se resuelven los nombres de variables, funciones y objetos. Esto influye en aspectos como el rendimiento, la flexibilidad y la detección de errores en tiempo de compilación o ejecución.

Ligadura Estática vs. Dinámica

Tipo de Ligadura	¿Cuándo se resuelve?	Características	Ejemplo de uso
Estática (early binding)	En tiempo de compilación	Más rápida, menos flexible	Variables globales, métodos final en Java
Dinámica (late binding)	En tiempo de ejecución	Más flexible, pero menos eficiente	Polimorfismo, virtual en C++, dynamic en Python

Ejemplo de ligadura estática.

Por ejemplo, en lenguajes con tipado estático como Java, el tipo de una variable se determina en tiempo de compilación.

```
class Animal {  
    void hacerSonido() {  
        System.out.println("Sonido genérico");  
    }  
}  
  
public class Main
```

```
{ public static void main(String[] args) {  
    Animal a = new Animal(); a.hacerSonido();  
}  
}  
// La ligadura es estática, siempre ejecuta "Sonido genérico" } }
```

Ejemplo ligadura dinamica:

En lenguajes con soporte para **polimorfismo y herencia**, como **Java, C++ o Python**, la ligadura se puede realizar en **tiempo de ejecución**, dependiendo del tipo real del objeto.

```
class Animal {
    void hacerSonido() {
        System.out.println("Sonido genérico");
    }
}

class Perro extends Animal {
    void hacerSonido() {
        System.out.println("Ladrado");
    }
}

public class Main {
    public static void main(String[] args) {
        Animal a = new Perro(); // Ligadura dinámica
        a.hacerSonido(); // Ejecuta "Ladrado" porque se resuelve en tiempo de
ejecución
    }
}
```

✔ En este caso, aunque `a` es del tipo `Animal`, en **tiempo de ejecución** se detecta que es una instancia de `Perro`, por lo que se llama al método sobrescrito.

Diferencias Clave entre Ligadura Estática y Dinámica

Característica	Ligadura Estática	Ligadura Dinámica
Cuándo se resuelve	En tiempo de compilación	En tiempo de ejecución
Velocidad	Más rápida (optimización)	Más lenta (búsqueda en tiempo real)
Flexibilidad	Menos flexible	Más flexible
Uso común	Variables locales, funciones normales	Polimorfismo, sobrescritura de métodos

Conclusión:

- **Ligadura estática** mejora la eficiencia y evita errores en ejecución, pero limita la flexibilidad.
- **Ligadura dinámica** permite la reutilización de código con polimorfismo, aunque con un pequeño costo de rendimiento.

Practica 4:

Apuntes que tengo (estaria bueno chequearlos)

L - Value: Direccion de memoria ligada a la variable

Direccion de memoria ligada a la variable **durante la ejecucion**.

En el L-Value la allocacion de memoria puede ser:

- Estatico: Se hace en compilacion (cuando se carga el programa en memoria y perdura hasta el final de la ejecucion)
- Dinamico: Se hace en tiempo de ejecucion
 - Automatica: Cuando aparece una declaracion de una variable en la ejecucion
 - Explicita: Requerida por el programador con alguna sentencia de creacion

R - Value: Valor asignado en memoria

Cada lenguaje inicializa a las variables con un valor distinto, dependiendo del tipo.

- **Las constantes son estaticas en cuanto a su R-Valor**

Alcance:

Rango de instrucciones en el que es conocido el nombre de la variable o funcion

- Alcance de una funcion: Desde que se define hasta que termina.
- Alcance de una variable: Desde que se la declara hasta que termina el bloque que la contiene (creo que si la variable es estatica cambia).

Nota: Tener en cuenta que para el alcance en esta cursada, se toma a partir de la siguiente linea donde se declaro.

Tiempo de vida:

Intervalo desde que una variable/funcion/procedimiento, existe en memoria hasta que se desaloca de memoria. Es decir, desde que se inicia el bloque que lo contiene, hasta que finaliza.

- Por ejemplo si tenemos un procedimiento, su tiempo de vida va a ser:
 - Desde la linea que se define el procedimiento.
 - Hasta la linea que termina la declaracion del procedimiento.

Consideraciones a tener en cuenta:

Si estamos en pascal, y en el main declaramos la variable p de tipo puntero a entero.:

Elemento	Alcance	Tiempo de Vida
p: ^integer	Linea siguiente a su declaracion, hasta que termine el bloque main.	Desde la entrada hasta la salida del bloque
^p	Accesible mientras tengas p	Desde new(p) hasta dispose(p)

Particularidades de algunos lenguajes.

Tipos de variables al momento de inicializacion.

◆◆ En C: Las variables globales tienen tiempo de vida todo el programa Las variables estaticas tienen tiempo de vida todo el programa y van entre < > Las variables estaticas numericas, tienen R-Valor 0 Las variables globales numericas tienen R-Valor 0

Lenguaje	Tipo de Variable	L-Valor	R-Valor	Notas a considerar	
C	Global numérica	automática	Toma valor por defecto (0)	Se inicializa si es global o estática, no si es local	
	Global puntero	automática	Null por defecto	Igual que arriba, punteros globales se inicializan	
	Global estática numérica	estático	Toma valor por defecto (0)		
	Local estática numérica	automática	Toma valor por defecto	Solo en variables estáticas	
	Local (tipo: int i=4)	automática	Definido (4)		
	Puntero local sin inicializar	automática	Indefinido	Peligroso usar sin inicializar	
	Local tipo: int v1	automatica	0		
ADA	Constante numérica (c : constant integer := 10)	automática	10 (definido)	Es constante común	
	Constante sin := (c :	automática	Definido al compilar		

Lenguaje	Tipo de Variable	L-Valor	R-Valor	Notas a considerar	
	<code>constant := 10)</code>				
	Arreglo tipo (1..10)	automática	Indefinido		
	Arreglo (L..C) definido con constante	semi-dinámica	Indefinido		
	Arreglo de tipo constante	automática	Indefinido		
	Puntero	automática	Null	ADA parece siempre inicializar en Null los punteros	
Pascal	Variable global (<code>var x: integer</code>)	estático	Toma valor por defecto (0)	Variables globales sí se inicializan por defecto	
	Variable local (<code>var x: integer</code>)	automática	Indefinido	Si no se inicializa, el valor es basura	
	Puntero (<code>p: ^integer</code>)	automática	Indefinido o Nil	No siempre se inicializa, depende del compilador (FreePascal lo hace a veces)	
	Contenido de puntero (<code>p^</code>)	dinámico	Indefinido si no se usó <code>new(p)</code>		
Java	Variable primitiva (int, float)	estático	0 (si es global o de clase) / indefinido (si es local)	Variables miembro sí se inicializan; locales no	
	Objeto (<code>String s;</code>)	estático	null (si es miembro), indefinido si local sin inicializar	Java siempre requiere inicializar variables locales antes de usarlas	

Lenguaje	Tipo de Variable	L-Valor	R-Valor	Notas a considerar	
	Arreglo	automática	null (sin inicializar), con elementos a 0/null si se inicializa	El arreglo en sí es null al principio; sus elementos se inicializan si se instancia	
Python	Variable numérica (<code>x = 5</code>)	dinámico	5 (definido)	No hay tipado explícito; todo es objeto y las referencias son automáticas	
	Objeto (<code>obj = Clase()</code>)	dinámico	Referencia a objeto	Siempre se inicializa explícitamente	
	Lista/Dict (<code>lista = [1,2,3]</code>)	dinámico	Definido	Python administra memoria automáticamente	

Variables, tipos de variables y como trabaja cada tipo de variable.

Tipo	Qué es	Vida útil (duración)	Ejemplos y comportamiento
Estática	Se reserva una sola vez en memoria	Desde que el programa inicia hasta que termina	C: <code>static int x;</code> dentro de funciones o global Java: variables de clase (con <code>static</code>)
Automática	Se crean al entrar en un bloque (función, procedimiento) y se destruyen al salir	Mientras dure el bloque/función	C/Pascal: variables locales Ada: variables locales sin <code>new</code>
Dinámica	Se reserva memoria manualmente con <code>new</code> o similar	Mientras el programador no la libere	C: <code>malloc</code> , <code>free</code> Pascal: <code>new(p)</code> y <code>dispose(p)</code> Java: <code>new Clase()</code> (aunque el garbage collector lo maneja)
Semidinámica	Tipo especial donde una parte es dinámica (por ej. límites de un arreglo),	Al entrar al bloque, pero su forma depende de una	Ada: arreglos definidos por parámetros de entrada o constantes (<code>(1..L)</code>)

Tipo	Qué es	Vida útil (duración)	Ejemplos y comportamiento
	pero no se usa <code>new</code> directamente	constante o parámetro	

d. Indique qué diferencia hay entre una variable estática respecto de su l-valor y una constante.

Variable estática	Constante
L-Valor: El l-valor de una variable estática es la dirección de memoria donde se almacena su valor.	L-Valor: Las constantes no tienen un l-valor en el sentido tradicional. El l-valor se refiere a la dirección de memoria de una variable, pero las constantes no ocupan una posición en la memoria de la misma manera que las variables.
Comportamiento: Una variable estática conserva su valor entre las llamadas a la función en la que se declara. Esto significa que su valor persiste a lo largo de la ejecución del programa, pero su valor puede modificarse mediante operaciones de asignación dentro de la función.	Comportamiento: Una constante tiene un valor que no puede cambiar durante la ejecución del programa. Se define mediante la palabra clave "const", y una vez que se le asigna un valor, ese valor no puede ser modificado posteriormente.
En términos de l-valor, mientras que una variable estática tiene una dirección de memoria asociada donde se almacena su valor y que puede ser modificada, una constante no tiene una dirección de memoria explícita, ya que su valor es tratado directamente como un valor literal durante la compilación.	
En términos de comportamiento, una variable estática se utiliza cuando se necesita preservar su valor entre las llamadas a una función, pero aun así permitir que su valor cambie dentro del contexto de la función. Por otro lado, una constante se utiliza cuando se necesita un valor que no cambie en absoluto durante la ejecución del programa y que sea conocido en tiempo de compilación.	

¿Qué pasa con la alocaación de memoria?

Variable	¿Cuándo se aloca?	¿Dónde se guarda en memoria?
Global	En tiempo de carga (inicio del programa)	En la sección de datos globales (data segment)
Static local	También en tiempo de carga	Igual que una global, en sección estática
Local normal	En cada llamada (tiempo de ejecución)	En la pila (stack) del programa

RESUMEN — Variables y Funciones en C

A chequear, esto es segun chatgpt

Elemento	Tipo de almacenamiento	R-valor	Alcance	Tiempo de vida	Acceso entre archivos
<code>int x;</code> dentro de función	Automática (stack)	Indefinido	Solo dentro de la función	Desde que se entra a la función hasta que se sale	✗ No
<code>static int x;</code> en función	Estática interna (stack persistente)	0 si no se inicializa	Solo dentro de la función	Durante todo el programa	✗ No
<code>int x;</code> global (fuera de func)	Estática global	0 si no se inicializa	Desde el punto de declaración	Durante todo el programa	✓ Sí (extern)
<code>static int x;</code> global	Estática con linkage interno	0 si no se inicializa	Solo visible en el archivo	Durante todo el programa	✗ No (extern falla)
<code>extern int x;</code>	Depende del original	No define valor	Donde se declara extern	tiempo de vida Igual a la original	✓ Solo si el original no es static
<code>int *p = malloc(...)</code>	Dinámica (heap)	Dirección devuelta por malloc	Según dónde se declare el puntero	Hasta que se hace <code>free()</code> (si se hace)	✓ (si se pasa puntero)
Función común (<code>int f()</code>)	Código (no datos)	No aplica	Global por defecto	Siempre disponible durante todo el programa	✓ Sí (si no es static)
<code>static int f()</code>	Código con linkage interno	No aplica	Solo en ese archivo	Siempre disponible	✗ No se puede usar desde otro archivo

En C,:

- Las variables globales tienen tiempo de vida todo el programa
- Las variables estaticas tienen tiempo de vida todo el programa y van entre < >

- Las variables estaticas numericas, tienen R-Valor 0
- Las variables globales numericas tienen R-Valor 0

Las variables extern tienen tiempo de vida todo el programa (su alcance puede ir variando ya que se usa en distintas partes del programa.)

Prestar atencion a los extern

Cuando veamos una variable no extern, pero que despues en otro archivo se hace un extern, sabemos que el tiempo de vida es todo el programa, y el alcance puede ir variando, puede tener mas de una opcion de alcance.

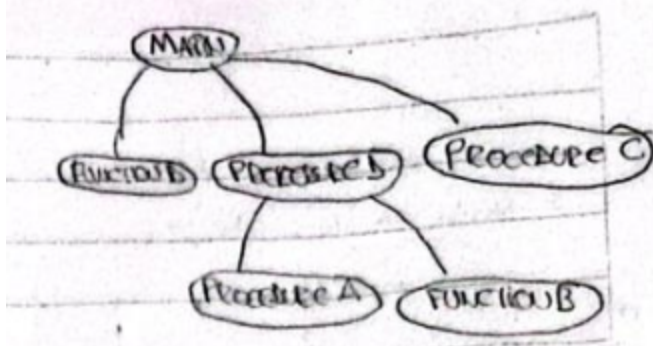
Funciones no tienen Tipo, ni R-Valor.

Practica 5:

Para esta practica, hay que ir prestando atencion a la ejecucion y dependiendo que cadena estemos siguiendo, ver como se maneja el flujo de ejecucion.

Tips para arrancar a hacer los ejercicios:

1. Armar arbol para saber quien modulo define a quien



2. Armar la tabla de ejecucion

Siempre poner en el medio las variables primero y despues los modulos que defina el modulo actual.

nombrePrograma - PR

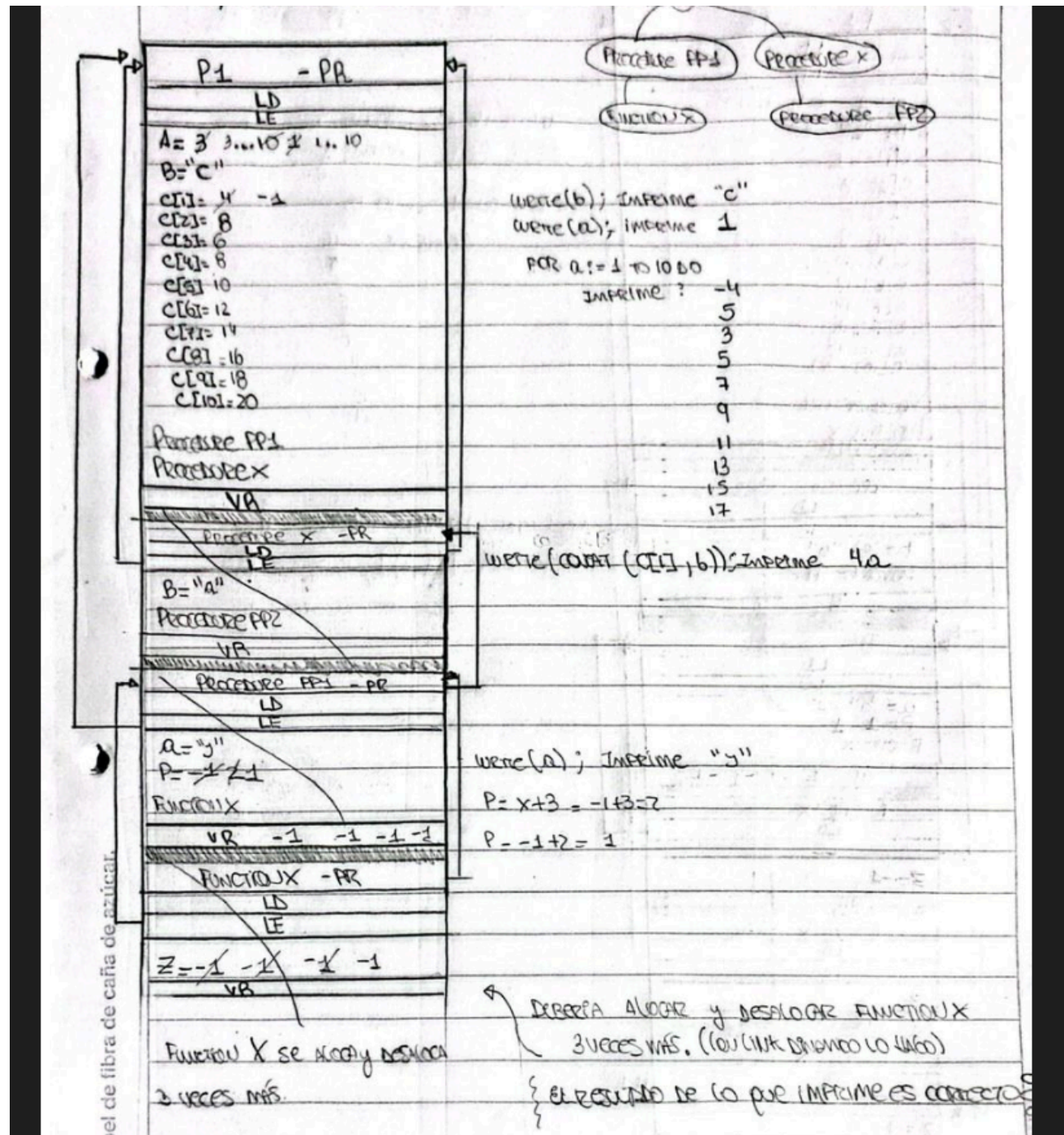
LD

LE

{variables y otros modulos
que el modulo actual declare}

VR

Ejemplo algo así:



3) Cuando termina el modulo actual, y se desaloca, lo tachamos para indicar que termino su ejecucion .

Prestar atencion a:

VR: Valor que retorna a la unidad actual, otra unidad.

- Si una unidad genera un valor **no se escribe en la unidad que lo genera/retorna**, se escribe en la **unidad que lo llama/invoca**, es decir en la **unidad que retornó el dato, o le pidio el dato**.

- El valor se registra en quien llamó , que es el que o va a usar,

Para armar el arbol sintactico, tenemos que arrancar desde program, y ver los procedimientos y funciones, si los define program, o si los define algun otro funcion/procedimiento.

Liks Dinamicos y Estaticos:

Link estatico: Quien lo declara (El padre en el arbol de anidamiento)

Link dinamico: Quien lo llamó/ Modulo que lo llama/invoca.

siempre tienen que estar las cuentas, y tienen que estar al costado del registro.

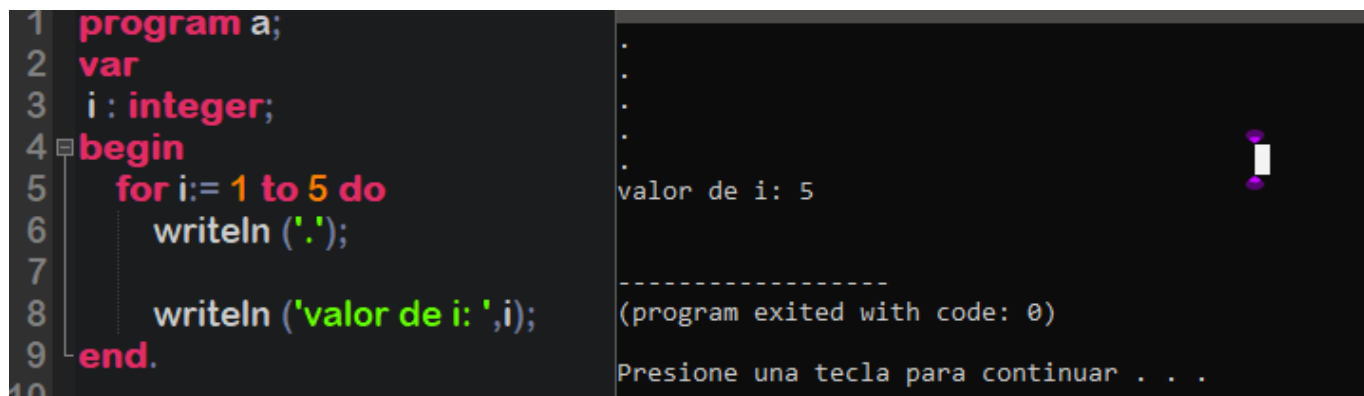
primero **siempre buscamos las variables dentro del entorno en el que estamos ejecutando**. Si no la encontramos la buscamos en otros modulos a traves de link estatico o dinamico.

Para hacer las busquedas de variables y despues para asignar valor, buscamos de abajo hacia arriba. Por ejemplo si en nuestro modulo no tenemos la variable, lo buscamos arriba, y asi sucesivamente.

Cuando tenemos que buscar un valor, tenemos que tener en cuenta si tenemos que hacerlo por cadena estatica o cadena dinamica

Importantisimo: El valor de retorno es hacia el modulo que invoca, NO SE HACE DESDE EL MODULO QUE ENCUENTRA/GENERA EL VALOR. Sino el VR le pertenece al modulo donde invoca.

Pilas de ejecucion



The image shows a code editor on the left and a terminal window on the right. The code in the editor is a Pascal program named 'a' that declares an integer variable 'i' and uses a 'for' loop to print the value of 'i' from 1 to 5. The terminal window shows the output of the program, which is a series of dots followed by the text 'valor de i: 5', a separator line, and the message '(program exited with code: 0)'. The prompt 'Presione una tecla para continuar . . .' is visible at the bottom of the terminal.

```

1  program a;
2  var
3  i : integer;
4  begin
5      for i:= 1 to 5 do
6          writeln ('.');
7
8          writeln ('valor de i: ',i);
9  end.

```

```

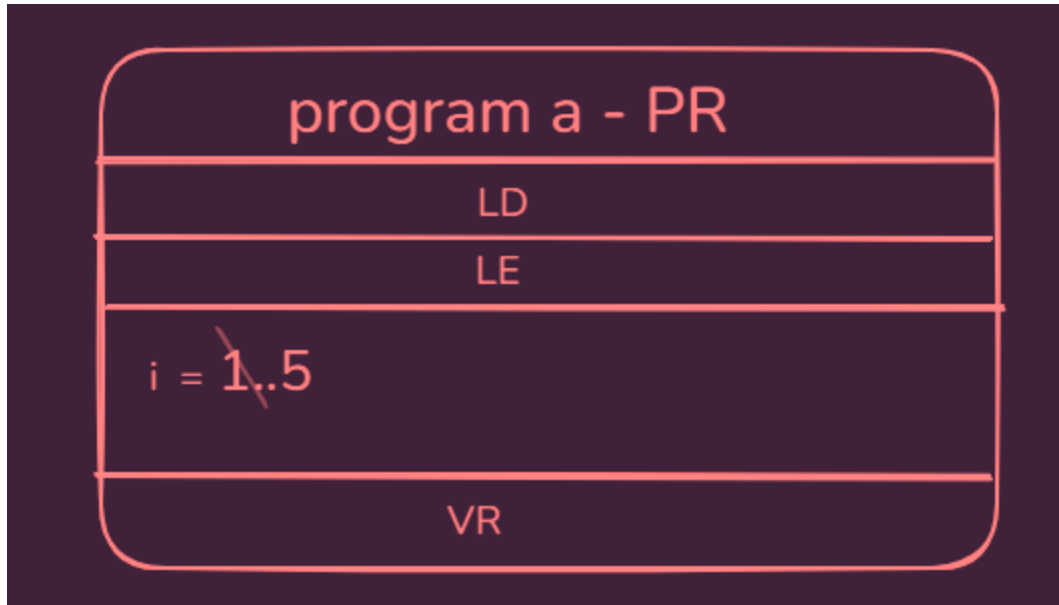
.
.
.
.
.
valor de i: 5
-----
(program exited with code: 0)
Presione una tecla para continuar . . .

```

En un for, la variable indice termina con el ultimo valor de la iteracion.

En una pila de ejecucion, debemos expresarlo de esta manera:

- Expresamos que i va de 1 hasta 5 (1..5) y luego para indicar que termina con el valor 5, tachamos el 1..



Preguntas teoricas de verdadero o falso con justificacion.

(V / F)

1. Los lenguajes dinámicamente tipados permiten detectar errores de tipo en tiempo de compilación.
 - FALSA
 - Los lenguajes dinamicamente tipados no detectan errores en tiempo de compilacion, sino en tiempo de ejecucion. En estos lenguajes no es necesario declarar el tipo de una variable, y su tipo puede cambiar durante la ejecucion del programa.
 - En cambio, los lenguajes estaticamente tipados, verifican los tipos en tiempo de compilacion y marcarian un error por ejemplo si se quisiese hacer esto:

```
int edad = "hola";
```

Modo de parámetro	Qué significa	Cómo se comporta
in	Solo entrada	El parámetro se usa dentro, pero no se modifica fuera . Es como una constante local .
out	Solo salida	El parámetro no tiene valor útil al entrar , y se le debe asignar un valor dentro del procedimiento.
in out	Entrada y salida	El parámetro entra con un valor y puede salir modificado .

✓ 2. Técnicas de paso de parámetros

Esto se refiere a **cómo** se pasa el valor internamente (lo que hace el lenguaje), y afecta si se pueden o no modificar los valores originales.

Técnica	Cómo funciona	Ejemplo de lenguaje
Por valor	Se pasa una copia . Cambios dentro del procedimiento no afectan al original .	C, Java (para tipos primitivos)
Por referencia	Se pasa una referencia (o dirección) . Cambios afectan al original.	C++ (&), Python (para objetos mutables), Ada (in out)
Por resultado	El parámetro se ignora al entrar , pero se le asigna algo dentro.	Ada (out)
Por nombre (menos común)	Se pasa el código literal del argumento como si fuera copiado dentro del procedimiento.	Lenguajes antiguos como ALGOL (muy raro hoy en día)