Conceptos y Paradigmas de Lenguajes de Programación - Práctica 3

EJERCICIO 1

La **semántica** describe el significado de los **símbolos, palabras** y **frases** de un lenguaje (ya sea lenguaje natural o informático) que es sintácticamente válido.
Para luego poder darle un significado a una construcción del lenguaje.
Tipos de semántica:
- **Estática**: antes de la ejecución.
- **Dinámica**: durante la ejecución.

EJERCICIO 2

a) Dado un programa escrito en un lenguaje de alto nivel o lenguaje fuente (Ej: Ada, C++, Pascal, etc), hay un programa llamado **compilador** que realiza la **traducción** a lenguaje de máquina (lenguaje objeto). Se traduce/compila **antes** de la ejecución.

b) La compilación puede ejecutarse en 1 o 2 etapas. En ambos casos se cumplen varias sub-etapas, las principales son:

1) Etapa de análisis *(más vinculado al código fuente)*
- **Análisis léxico (programa Scanner):** analiza el tipo de cada uno de sus elementos para ver si son Tokens válidos.
- **Análisis sintáctico (programa Parser):** se identifican las estructuras de las sentencias, declaraciones, expresiones, etc. a partir de los Tokens del analizador léxico.
- **Análisis semántico (programa de semántica estática):** procesa las estructuras sintácticas (reconocidas por el analizador sintáctico)

Puede generarse código intermedio: es realizar la transformación del código fuente en una representación de código intermedio para una máquina abstracta.

2) Etapa de Síntesis *(más vinculado a características del código objeto, del hardware y de la arquitectura)*
- **Optimización del código:** no lo hacen todos los compiladores.
- **Generación del código final:** programa objeto a ejecutar en la computadora.

c) La **semántica** interviene en la etapa de análisis, luego del análisis léxico y el análisis sintáctico *(deben pasar antes bien Scanner y Parser).* Es una de las etapas más importantes, ya que, procesa las estructuras sintácticas y es el nexo entre la etapa inicial y final del compilador. Realiza la comprobación de tipos, nombres, duplicados, etc.

EJERCICIO 3

Comparación entre Compilador e Intérprete

Intérprete	Compilador
Se utiliza en ejecución	Se utiliza antes de la ejecución
Ejecuta el programa línea por línea (no necesariamente recorre todo el código)	Sigue el orden físico de las sentencias (recorre todo)
Dependerá de la acción del usuario, de la entrada de datos y/o alguna decisión del programa.	Produce un programa ejecutable equivalente en lenguaje objeto
El programa fuente será público (necesito ambos)	El programa fuente no será público
Ocupa menos espacio de memoria	Generalmente ocupa más espacio
Es más fácil detectar y corregir errores	Es casi imposible ubicar errores (se pierde la referencia entre el código fuente y el código objeto) -> se deben usar otras técnicas (Ej: semántica dinámica)
Más lento en ejecución	Más rápido en ejecución, pero tarda en compilar

EJERCICIO 4

Error Sintáctico
Un error sintáctico ocurre cuando el código no sigue las reglas gramaticales del lenguaje. Es decir, tiene una estructura incorrecta. Se detecta en tiempo de compilación o de interpretación, antes de que el programa se ejecute.

Ejemplo de error sintáctico en Python

```
def saludar(nombre:
    print("Hola, " + nombre)
```

En este caso el error sintáctico es que falta el paréntesis de cierre en la declaración de la función saludar().

Error Semántico
Un error semántico ocurre cuando el código sigue las reglas gramaticales del lenguaje, es decir, que tiene una sintáxis correcta pero igualmente carece de sentido para el lenguaje.
En el caso de la Semántica Estática, esta se evalúa previo a la ejecución del programa.
En el caso de la Semántica Dinámica, se evalúa durante la ejecución del programa.

Ejemplo de error semántico en Python

```
int main()
{       int a;
        char cadena;
        resultado = a + cadena;
        printf(resultado);
        return 0;
}
```

En este caso, la sintáxis es correcta para Python, pero hay un error semántico ya que se intenta sumar dos variables de distintos tipos (a de tipo int y cadena de tipo char) y no hay reglas que lo permitan o lo resuelvan.

EJERCICIO 5

a) Programa en Pascal

Programa	**Errores**

Program P
var
  5: integer;
  a: char;
begin
  for i := 5 to 10 do
  begin
  write(a);
  a = a + 1;
  end;
end.
	// Sintáctico: faltan ; después de P
	
	// Sintáctico: 5 no puede ser nombre de variable
	
	// Semántico: variable i no está declarada
	
	// Semántico: variable a no tiene ningún valor asignado
	// Sintáctico: debería ser a:=a+1; // Semántico: no se puede sumar un char

Total:	3 errores sintácticos y 3 semánticos

b) Programa en Java

Programa	**Errores**

public String tabla(int numero, arrayList<Boolean> listado)
{
  String result = null;
  for(i = 1; i < 11; i--) {
 	
  result += numero + "x" + i + "=" + (i*numero) + “\n";
 	
  listado.get(listado.size()-1)=(BOOLEAN) numero>i;
  }
 	
  return true;
}
	// Sintáctico: es ArrayList
	
	// Semántico: la variable i no está declarada
	// Lógico: loop infinito
	
	// Semántico: result tiene valor null. No puede concatenarse
	// Sintáctico: BOOLEAN no existe. Debería ser Boolean.
	
	// Sintáctico: el lado izquierdo de una asignación debe ser una variable (según el profe es semántico porque es un error de tipo)
	
	// Semántico: error de tipo return. Debe retornar un String.

Total:	3 errores sintácticos, 3 semánticos y 1 lógico

c) Programa en C

Programa	Errores

# include <stdio.h> 	
int suma; /* Esta es una variable global */
int main()
{
  int indice;
  encabezado;
  for (indice = 1 ; indice <= 7 ; indice ++)
  cuadrado (indice);
  final(); /* Llama a la función final */
  return 0;
}
 	
cuadrado (numero)
int numero;
{
  int numero_cuadrado;
  numero_cuadrado == numero * numero;
  suma += numero_cuadrado;
  printf("El cuadrado de %d es %d\n", numero, numero_cuadrado);
}
	// Sintáctico: sin espacio entre # y include
	
	// Sintáctico: var encabezado no declarada
	
	// Semántico: cuadrado no existe en este contexto
	// Semántico: la función final no existe
	// Sintáctico: faltan apertura y cierre de comentario
	
	// Sintáctico: falta llaves { } para la función cuadrado
	// Sintáctico: falta tipo de retorno o void de función
	
	// Sintáctico: se usa un solo = para la asignación
	// Semántico: variable numero no inicializada
	// Semántico: variable suma no declarada

Total:	6 sintácticos y 4 semánticos

d) Programa en Python

Programa	Errores

#!/usr/bin/python
print "\nDEFINICION DE NUMEROS PRIMOS"
r = 1
while r = True:
  N = input("\nDame el numero a analizar: ")
  i = 3
  fact = 0
  if (N mod 2 == 0) and (N != 2):
  print "\nEl numero %d NO es primo\n" % N
  else:
  while i <= (N^0.5):
  if (N % i) == 0:
  mensaje = "\nEl numero ingresado NO es primo\n" % N
  msg = mensaje[4:6]
  print msg
  fact = 1
  i += 2
  if fact == 0:
  print "\nEl numero %d SI es primo\n" % N
  r = input("Consultar otro número? SI (1) o NO (0)--->> ")
	
	// Sintáctico: falta el uso de ( )
	
	// Sintáctico: debería ser ==
	// Semántico: se quiere comparar r (1) con True
	// Semántico: input necesita castearse a tipo numérico
	
	// Sintáctico: no se usa mod en Python, se usa %
	
	// Sintáctico: la potencia se realiza con **
	
	// Sintáctico: falta el uso de ( )
	// Semántico: falta %d
	
	// Semántico: se toma subcadena de mensaje y no se imprime bien
	
	// Sintáctico: falta el uso de ( )

Total:	6 sintácticos y 4 semánticos

e) Programa en Ruby

Programa	**Errores**

def ej1
 Puts 'Hola, ¿Cuál es tu nombre?'
  nom = gets.chomp
  puts 'Mi nombre es ', + nom
  puts 'Mi sobrenombre es 'Juan''
  puts 'Tengo 10 años'
  meses = edad*12
  dias = 'meses' *30
  hs= 'dias * 24'
  puts 'Eso es: meses + ' meses o ' + dias + ' días o ' + hs + ' horas'
  puts 'vos cuántos años tenés'
  edad2 = gets.chomp
  edad = edad + edad2.to_i
  puts 'entre ambos tenemos ' + edad + ' años'
  puts '¿Sabes que hay ' + name.length.to_s + ' caracteres en tu nombre, ' + name + '?'
end
	// Sintáctico: es puts
	// Sintáctico: la , está de más
	
	// Semántico: la variable edad no existe
	// Sintáctico: no se puede multiplicar strings
	// Semántico: asigno a hs una cadena de caracteres, no la cuenta (según el profe no es un error semántico. que no tenga sentido para el programador no quiere decir q sea un error)
	
	// Semántico: se concatena de forma incorrecta, las variables meses, dias y hs no deberían estar entre comillas
	// Semántico: edad no está declarada
	
	// Semántico: la variable name no existe

Total:	3 sintácticos y 6 semánticos

***Nota**: si tocara un ejercicio así en el parcial es muy importante **justificar** la respuesta, ya que, hay veces que puede ser muy ambiguo. Dependiendo el lenguaje o la perspectiva con la que se mire. Además, a veces hay más de un error en la misma línea, y eso puede traer diferentes soluciones.*
***Nota 2**: no guiarse mucho por las "Ayudas" del ejercicio que dicen cuántos errores de cada tipo hay en cada código, ya que, es una práctica que tiene muchos años y como dijo el profesor, las soluciones pueden variar depende la persona.*
***Nota 3**: no solo en este tipo de ejercicio, sino en el parcial en general, nos recomiendan justificar cada respuesta y venir a la muestra del parcial a defenderlo, ya que, hay mucha ambigüedad y muchas zonas grises.*

EJERCICIO 6

self: es una palabra clave especial en Ruby que hace referencia al objeto actual en un contexto específico. Su valor depende del contexto: en un método de instancia es la instancia del objeto, en un método de clase es la clase misma. Al iniciar el intérprete, self tiene el valor main ya que es el primer objeto que se crea.

nil: es un objeto especial en Ruby que representa la ausencia de valor o nada (similar al concepto de null en otros lenguajes). Es un objeto único y tiene un único valor que es representado por la clase NilClass. Se utiliza para indicar que no hay un valor definido o válido en una variable o resultado.

EJERCICIO 7

En JavaScript, tanto **null** como **undefined** son valores especiales que representan la ausencia de un valor o la falta de definición. Sin embargo, hay diferencias importantes en su semántica y en su comportamiento.

undefined: es el valor predeterminado cuando una variable es declarada pero no tiene un valor asignado, o cuando una función no retorna nada. Su tipo es un tipo primitivo, y es precisamente **undefined**.

null: es un valor asignado explícitamente para indicar la ausencia intencional de un valor. Su tipo es un tipo de dato **object** (objeto).
La igualdad *null* == *undefined* es true (igualdad no estricta), mientras *null* === *undefined* es false (igualdad estricta).

EJERCICIO 8

La sentencia break en diferentes lenguajes tiene el propósito de salir de bucles o estructuras switch, pero con algunas variaciones según el lenguaje:

C: sale de un bucle o switch más cercano. No tiene soporte para salir de múltiples niveles de anidamiento.

PHP: similar a C, pero permite especificar cuántos niveles de anidamiento romper (break n).

JavaScript: similar a C, sale de un bucle o switch más cercano, sin soporte para múltiples niveles.

Ruby: además de salir de bucles, puede devolver un valor desde un bucle o bloque. No tiene soporte para múltiples niveles de anidamiento.

En resumen, la principal diferencia radica en cómo se manejan los niveles de anidamiento (solo PHP puede especificarlo), y Ruby tiene además la capacidad de devolver un valor cuando se usa **break**.

EJERCICIO 9

Ligadura: es el proceso de asociar un nombre (variable o función) con su valor o dirección de memoria en un programa. Es crucial para la semántica de un programa, ya que determina cómo y cuándo se resuelven las referencias a variables y funciones.
En otras palabras, la ligadura es el proceso de asociar cada entidad con sus atributos.

Ligadura Estática
- Ocurre antes de la ejecución (ya sea en la definición del lenguaje, la implementación o la compilación)
- El valor y tipo de las variables son fijos antes de la ejecución
- Común en lenguajes como C y Java
- Más eficiente pero menos flexible

Ligadura Dinámica
- Ocurre en tiempo de ejecución

- Los valores y referencias pueden cambiar mientras el programa se ejecuta
- Común en lenguajes como JavaScript y Python
- Menos eficiente pero más flexible

Ejemplos en C

En Definición • Forma de las sentencias • Estructura del programa • Nombres de los tipos predefinidos	**int** para denominar a los enteros
En Implementación • Set de valores y su representación numérica • Sus operaciones	**int** su representación en memoria, set de valores contenidos en el tipo y sus operaciones
En Compilación • Asignación del tipo a las variables	**int a** se liga el tipo a la variable (atributo)
En Ejecución • Variables se enlazan con sus valores • Variables se enlazan con su lugar de almacenamiento	**int a** **a = 10** **a = 5** el valor de la variable se liga y puede cambiar en ejecución

Otros ejemplos
En C y Java no se puede cambiar el tipo de una variable en ejecución pero si puede cambiar su valor. Por eso se dice que tienen ligadura estática.
En JavaScript y Python tanto el tipo como el valor de una variable puede cambiarse en ejecución. Por eso se dice que tienen ligadura dinámica.

CONSULTAR !!!

*Nota: es ambigua la definición y diferencia entre ligadura estática y dinámica. Se podría decir que la **ligadura estática** es cuando los valores y los tipos de las variables se asignan antes de la ejecución del programa y la **ligadura dinámica** es cuando los valores y tipos de variables se asignan y pueden cambiar durante la ejecución del programa.*
*Sin embargo, en el caso del lenguaje **C y Java**, que se dice que tienen ligadura estática, los tipos de las variables son definidos antes de la ejecución del programa pero su valor puede cambiar durante la ejecución.*
*Y en lenguajes como **Python y JavaScript**, que se dice que tiene ligadura dinámica, tanto el tipo como el valor de las variables puede cambiar en tiempo de ejecución del programa.*

***Respuesta del ayudante:** ligadura estática se refiere más que todo a que las variables se les asigna un tipo previo a la ejecución del programa, pero luego durante la ejecución pueden cambiar su valor. En cambio la ligadura dinámica se refiere a que las variables pueden cambiar tanto su valor como su tipo durante la ejecución del programa.*
Luego también me dijo que la cátedra considera que solamente C tiene variables estáticas puras, que son aquellas a las cuales se les asigna un lugar de memoria previo a la ejecución del programa y que persiste tanto previo como posterior a ese ejecución.

***Nota**: donde se encuentre un error de tipo es un error semántico.*