Conceptos y Paradigmas de Lenguajes de Programación 2023
Práctica Nro. 2
Sintaxis
Objetivo: conocer cómo se define léxicamente un lenguaje de programación y cuales son las herramientas necesarias para hacerlo

Ejercicio 1: ¿Qué define la semántica?

La semántica describe el significado de los símbolos, palabras y frases de un lenguaje ya sea lenguaje natural o lenguaje informático que ya es sintácticamente válido.
Se puede dividir en semántica estática y semántica dinámica.

La semántica estática no está relacionada con el significado del programa, está relacionada con las formas válidas.
Se las llama así porque el análisis para el chequeo puede hacerse en compilación. Para describir la sintaxis y la semántica estática formalmente sirven las denominadas gramáticas de atributos, inventadas por Knuth en 1968. Generalmente las gramáticas sensibles al contexto resuelven los aspectos de la semántica estática.

La semántica dinámica es la que describe el efecto de ejecutar las diferentes construcciones en el lenguaje de programación.
Su efecto se describe durante la ejecución del programa. Los programas solo se pueden ejecutar si son correctos para la sintaxis y para la semántica estática.

Ejercicio 2:

a. ¿Qué significa compilar un programa?

Compilar un programa significa traducir el programa escrito en un lenguaje de alto nivel, a lenguaje de máquina.

b. Describa brevemente cada uno de los pasos necesarios para compilar un programa.

Para compilar correctamente, es decir, poder hacer la traducción, primero se deben validar varias cosas:

- Primero se realiza un analisis lexico (analisis a nivel de palabra o lexema, se divide el programa en sus elementos/ categorías y se analiza el tipo de cada uno para ver si son tokens válidos)

- Lo siguiente es el analisis sintactico, (analisis a nivel de estructuras o sentencias, en este paso se usa una gramática para construir el "árbol sintáctico" / "árbol derivación" del programa y validar de esta forma que pertenece o no a la gramática para ver que lo que entra es correcto)

- En tercer lugar se realiza el **análisis semántico**, la fase más **importante**. (Realiza la comprobación de tipos (aplica gramática de atributos), agrega a la tabla de símbolos los descriptores de tipos, realiza comprobaciones de duplicados, problema de tipos, etc y realiza comprobaciones de nombres y más cosas)

- Por último, se ejecuta la etapa de síntesis (Construye el programa ejecutable y genera el código necesario)

Ejercicio 3: **Con respecto al punto anterior ¿es lo mismo compilar un programa que interpretarlo? Justifique su respuesta mostrando las diferencias básicas, ventajas y desventajas de cada uno.**

Compilación:

- **Traduce todo el código a binario antes de la ejecución.**
- Ventajas: Eficiencia en la ejecución, seguridad durante la compilación, independencia de entorno.
- Desventajas: Proceso lento de compilación, limitada portabilidad del código compilado, necesidad de recompilar para cambios.

Interpretación:

- **Traduce y ejecuta el código línea por línea en tiempo real.**
- Ventajas: Facilidad de depuración, mayor portabilidad del código fuente, agilidad en el desarrollo.
- Desventajas: Menor eficiencia en la ejecución, menor seguridad, dependencia del entorno de ejecución.

Ejercicio 4: **Explique claramente la diferencia entre un error sintáctico y uno semántico. Ejemplifique cada caso**

Error sintáctico:

- Un error sintáctico ocurre cuando una sentencia viola las reglas definidas en la gramática del lenguaje.
- Se detecta durante la etapa del Análisis léxico o en la del Análisis Sintáctico.
- Un error común podría ser la falta de un punto y coma, la mala colocación de un paréntesis o el uso indebido de una palabra clave.

Error semántico:

- Un error semántico refiere al sentido o significado de una sentencia ya sintácticamente válida.
- Se detecta durante la etapa de Análisis Semántico.
- Un error común puede ser una asignación incorrecta como asignar un string a una variable de tipo entero o intentar llamar a una función con una cantidad incorrecta de parámetros

Ejercicio 5: **Sean los siguientes ejemplos de programas. Analice y diga qué tipo de error se produce (Semántico o Sintáctico) y en qué momento se detectan dichos errores (Compilación o Ejecución). Aclaración: Los valores de la ayuda pueden ser mayores.**

a) Pascal

```
Program P
var 5: integer;
var a: char;

Begin
        for i:=5 to 10 do // variable ‘i’ no está declarada
        begin
                write(a);
                a=a+1;

        end;
End.
```

Ayuda: Sintáctico 2, Semántico 3

ERRORES:
Sintáctico:

- Línea 1: falta punto y coma final
- Línea 2: Una variable no puede identificarse con un número
- Línea 9: Faltan los dos puntos en la asignación

Semántico:

- Línea 6: La variable ‘i’ no está declarada
- Línea 9: Suma integer con un char
- Línea 9: La variable ‘a’ no está inicializada

b) Java:

```
public String tabla(int numero, arrayList 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;
}
```

Ayuda: Sintácticos 4, Semánticos 3, Lógico 1

ERRORES:
Sintáctico:

- Linea 1: Array list va con mayus
- Linea 1:Array list le falta el tipo
- Línea 4: Falta el tipo de la variable ‘i’
- Línea 6: Sería Boolean para castear, no BOOLEAN

Semántico:

- Línea 5: Intenta asignar a la lista el boolean
- Línea 6: BOOLEAN no está declarado
- Línea 8: Retorna un boolean, se esperaba un String

Lógico:

- Línea 4: El for entra en loop infinito

c) C

```
# 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);
}
```

ERRORES

Sintáctico:

- Línea 6: Falta la llave para abrir el bloque del For
- Línea 8: Falta abrir comentario en “Llama a la función final */”
- Linea 12: Falta indicar el tipo del parámetro “numero”

Semántico:

- Línea 5: “encabezado” no está declarado
- Línea 7: La función “cuadrado” no está declarada (debe declararse ANTES)
- Línea 8: La función “final” no está declarada
- Línea 15: Para asignar, debería ser “=” en vez de “==”

// Supuestamente me faltan 2 errores semánticos más, pero ni idea, corrigiendo esas cosas que puse, ya compila y funciona perfectamente, así que ni idea que me falta

d) Python

```
#!/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)--->> ")
```

Ayuda: Sintácticos 2, Semánticos 3

ERRORES:

Sintáctico:

- Línea 2: Faltan los paréntesis para el print
- Línea 17: Faltan paréntesis en el print
- Línea 15: Faltan paréntesis para el print
- Línea 4: Intenta comparar con “=”, y sería “==”
- Línea 8: Intenta calcular el módulo con “mod”, en python se usa el operador %

Semántico:

- Línea 3: Compara int con boolean
- Línea 8: Input recibe String, por tanto N en el If es un String, y se intentan operaciones de enteros

//Me falta un simpático, ni idea

e) Ruby

```
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
```

ERRORES
Sintáctico:

- Línea 2: Puts va con minúscula

Semántico:

- Línea 7: No existe variable ‘edad’
- Línea 10: No existen variables ‘meses o’ ni ‘dias o’
- Línea 15: No existe variable ‘name’

Lógico:

- Línea 8: Concatena 30 veces el String “meses”
- Línea 9: Asigna el string “días * 24” a hs

Ejercicio 6:

Dado el siguiente código escrito en pascal. Transcriba la misma funcionalidad de acuerdo al lenguaje que haya cursado en años anteriores. Defina brevemente la sintaxis (sin hacer la gramática) y semántica para la utilización de arreglos y estructuras de control del ejemplo.

```
Procedure ordenar_arreglo(var arreglo: arreglo_de_caracteres;cont:integer);
var
        i:integer; ordenado:boolean;
        aux:char;

begin
        repeat ordenado:=true;
                for i:=1 to cont-1 do
                        if ord(arreglo[i])>ord(arreglo[i+1])
                        then begin
                                aux:=arreglo[i];
                                arreglo[i]:=arreglo[i+1];
                                arreglo[i+1]:=aux; ordenado:=false
                        end;
        until ordenado;
end;
```

Observación: Aquí sólo se debe definir la instrucción y qué es lo que hace cada una; detallando alguna particularidad del lenguaje respecto de ella. Por ejemplo el for de java necesita definir una variable entera, una condición y un incremento para dicha variable.

```
  public static void ordenarArreglo(char[] arreglo) {
      boolean ordenado;
      char aux;

      do {
         ordenado = true;
         for (int i = 0; i < arreglo.length - 1; i++) {
            if (arreglo[i] > arreglo[i + 1]) {
               aux = arreglo[i];
               arreglo[i] = arreglo[i + 1];
               arreglo[i + 1] = aux;
               ordenado = false;
            }
         }
      } while (!ordenado);
   }
```

Declaración de Variables:

- Pascal: En Pascal, las variables se declaran al principio de un bloque de código con su tipo de dato seguido por el nombre de la variable. Por ejemplo, var i: integer;.
- Java: En Java, las variables se pueden declarar en cualquier lugar dentro del bloque de código y siguen la sintaxis tipoDeDato nombreDeVariable. Por ejemplo, int i;.

Arreglos:

- Pascal: Los arreglos en Pascal se declaran con un tipo de dato y un tamaño fijo, por ejemplo, arreglo: array[1..N] of char;.
- Java: En Java, los arreglos se declaran con el tipo de dato seguido por []. Por ejemplo, char[] arreglo.

Estructura de Control - Bucle:

- Pascal: Pascal utiliza el bucle repeat-until, que ejecuta un bloque de código al menos una vez antes de verificar la condición de salida.
- Java: Java utiliza el bucle do-while que sigue una lógica similar al repeat-until de Pascal, ejecutando un bloque de código al menos una vez antes de verificar la condición de salida.

Estructura de Control - Bucle for:

- Pascal: En Pascal, el bucle for sigue una estructura con una variable de control, el valor de inicio y el límite de salida.
- Java: Java utiliza el bucle for con una sintaxis diferente a la de Pascal, con una variable de control, una condición de salida y un incremento o decremento.

Acceso a Elementos de un Arreglo:

- Pascal: En Pascal, se accede a los elementos de un arreglo mediante su índice entre corchetes. Por ejemplo, arreglo[i].
- Java: En Java, el acceso a los elementos de un arreglo sigue la misma lógica que en Pascal, utilizando corchetes. Por ejemplo, arreglo[i].

Ejercicio 7: Explique cuál es la semántica para las variables predefinidas en lenguaje Ruby self y nil. ¿Qué valor toman; cómo son usadas por el lenguaje?

Self es una palabra clave de ruby que permite acceder a la instancia actual de un objeto, de manera similar al “this” de Java. Toman el valor de la dirección del objeto que esté ejecutando el método en un momento dado. Cambia automáticamente.

Nil representa la ausencia de valor o la falta de un objeto. Es un objeto único en Ruby que representa la nada. El valor de nil es el propio objeto nil. En términos de evaluación booleana, nil se considera falso, lo que significa que en contextos donde se espera una expresión booleana (if, unless, while, etc.), nil se considera falso.

Ejercicio 8: Determine la semántica de null y undefined para valores en javascript.¿Qué diferencia hay entre ellos?

En JavaScript, null y undefined son valores que indican la ausencia de valor.

- **null** se utiliza cuando se desea indicar explícitamente que una variable no tiene un valor válido. Es intencional

- **undefined** se utiliza en situaciones en las que una variable no ha sido inicializada o cuando se intenta acceder a una propiedad inexistente en un objeto. Probablemente hay un error del programador

Ejercicio 9: Determine la semántica de la sentencia break en C, PHP, javascript y Ruby. Cite las características más importantes de esta sentencia para cada lenguaje

- **C:** La instrucción break finaliza la ejecución de la instrucción do, for, switch o while más próxima que la incluya. El control pasa a la instrucción que hay a continuación de la instrucción finalizada

- **PHP:** break finaliza la ejecución de la estructura for, foreach, while, do-while o switch en curso. Break acepta un argumento numérico opcional que indica de cuántas estructuras anidadas circundantes se debe salir. El valor predeterminado es 1, es decir, solamente se sale de la estructura circundante inmediata

- **Javascript:** Termina el bucle actual, sentencia switch o label y transfiere el control del programa a la siguiente sentencia a la sentencia de terminación de éstos elementos. La sentencia break incluye una etiqueta opcional que permite al programa salir de una sentencia etiquetada. La sentencia break necesita estar anidada dentro de la sentencia etiquetada. La sentencia etiquetada puede ser cualquier tipo de sentencia; no tiene que ser una sentencia de bucle

- **Ruby:** usamos una declaración break para interrumpir la ejecución del bucle en el programa. En Ruby la sentencia break lleva una condición: El loop se rompe solo si la condición se cumple

Ejercicio 10: Defina el concepto de ligadura y su importancia respecto de la semántica de un programa. ¿Qué diferencias hay entre ligadura estática y dinámica? Cite ejemplos (proponer casos sencillos)

CONCEPTO DE LIGADURA (BINDING)

El binding es un concepto central en la definición de la semántica de los lenguajes de programación

- Los programas trabajan con entidades
- Las entidades tienen atributos
- Estos atributos tienen que establecerse antes de poder usar la entidad
- LIGADURA: es el momento en el que el atributo se asocia con un valor

LIGADURA ESTÁTICA

- Se establece antes de la ejecución.
- No se puede modificar: El término estática hace referencia al binding time y a su **estabilidad**.

Ejemplo:

```
int n;
// En compilación, se liga la variable con su TIPO y no se puede modificar
```

LIGADURA DINÁMICA

- Se establece durante la ejecución
- Se puede modificar durante la ejecución de acuerdo a alguna regla específica del lenguaje.
- Excepción: constantes (el binding es en runtime pero no puede ser modifica luego de establecida)

Ejemplo:

```
n = persona.getEdad();
// En ejecución, se le liga a la variable ‘n’ su R-Valor, y se puede modificar todas las veces que sean necesarias en ejecución
```