

Conceptos y Paradigmas de Lenguajes de Programación 2024

Práctica Nro. 3

Semántica

Objetivo: Interpretar el concepto de semántica de los lenguajes de programación.

Ejercicio 1: ¿Qué define la semántica?

Ejercicio 2:

- ¿Qué significa compilar un programa?
- Describe brevemente cada uno de los pasos necesarios para compilar un programa.
- ¿En qué paso interviene la semántica y cual es su importancia dentro de la compilación?

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.

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

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 begin

 write(a);

 a=a+1;

 end;

End.

Ayuda: Sintáctico 2, Semántico 3

b) Java:

```
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;
```

```
}
```

Ayuda:

Sintácticos 4, Semánticos 3, Lógico 1

Conceptos y Paradigmas de Lenguajes de Programación 2024

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);
}
```

Ayuda: *Sintácticos 2, Semánticos 6*

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*

Conceptos y Paradigmas de Lenguajes de Programación 2024

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
```

Ayuda: *Semánticos +4*

Ejercicio 5: 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

Conceptos y Paradigmas de Lenguajes de Programación 2024

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.

Ejercicio 6: Explique cuál es la semántica para las variables predefinidas en lenguaje Ruby **self** y **nil**. ¿Qué valor toman; cómo son usadas por el lenguaje?

Ejercicio 7: Determine la semántica de null y undefined para valores en javascript. ¿Qué diferencia hay entre ellos?

Ejercicio 8: Determine la semántica de la sentencia break en C, PHP, javascript y Ruby. Cite las características más importantes de esta sentencia para cada lenguaje

Ejercicio 9:

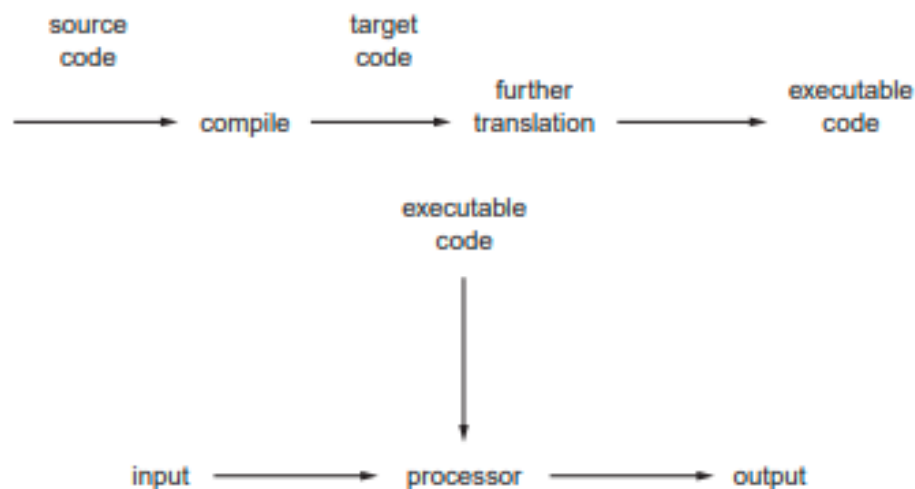
Defina el concepto de ligadura y su importancia respecto de la semántica de un programa. ¿Qué diferencias hay entre ligadura estática y dinámica? Cite ejemplos (proponer casos sencillos)

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.

2. Compilación

- a) La compilación, por otro lado, es al menos un proceso de dos pasos: el programa original (o programa fuente) se introduce en el compilador, y un nuevo programa (o programa objetivo) se genera a partir del compilador. Este programa objetivo puede luego ser ejecutado, si está en una forma adecuada para la ejecución directa (es decir, en lenguaje máquina). Más comúnmente, el lenguaje objetivo es el lenguaje ensamblador, y el programa objetivo debe ser traducido por un ensamblador en un programa objeto, y luego vinculado con otros programas objeto, y cargado en ubicaciones de memoria apropiadas antes de que pueda ser ejecutado. A veces, el lenguaje objetivo es incluso otro lenguaje de programación, en cuyo caso se debe utilizar un compilador para ese lenguaje para obtener un programa objeto ejecutable.



b) Pasos necesarios para compilar un programa:

1) Etapa de Análisis (Vinculadas al código fuente)

- Análisis léxico (Programa Scanner): Es un proceso que lleva tiempo, se encarga de hacer el análisis a nivel de palabra (LEXEMA) y divide el programa en sus elementos/categorías: identificadores, delimitadores, símbolos especiales, operadores, números, palabras clave, palabras reservadas, comentarios, etc. Además, analiza el tipo de cada uno para ver si son TOKENS válidos y filtra comentarios y separadores (como: espacios en blanco, tabulaciones, etc.) Lleva una tabla para la especificación del analizador léxico

que incluye cada categoría, el conjunto de atributos y acciones asociadas, además, pone los identificadores en la tabla de símbolos y reemplaza cada símbolo por su entrada en la tabla. Genera errores si la entrada no coincide con ninguna categoría léxica y el resultado de este paso será el descubrimiento de los ítems léxicos o tokens y la detección de errores.

- Análisis sintáctico (Programa Parser): El análisis se realiza a nivel de sentencia/estructuras. Tiene como objetivo identificar las estructuras de las sentencias, declaraciones, expresiones, etc. presentes en su entrada usando los tokens del analizador léxico, estas estructuras se pueden representar mediante el árbol de análisis sintáctico o árbol de derivación, que explica cómo se puede derivar la cadena de entrada en la gramática que especifica el lenguaje, validando qué pertenece o no a la gramática en pos de ver que lo que entra es correcto. El analizador sintáctico (Programa Parser) se alterna/interactúa con el análisis léxico y análisis semántico. Y usualmente usan técnicas de gramáticas formales.
 - Análisis semántico (Programa de Semántica estática): Para realizar este análisis primero se debe pasar correctamente el Scanner y el Parser. Esta etapa se considera como una fase medular, es decir, una de las más importantes, se encarga de procesar las estructuras sintácticas, realizar comprobación de tipos (aplica gramática de atributos), agregar a la tabla de símbolos los descriptores de tipos, realizar comprobaciones de duplicados, problemas de tipos, comprobaciones de nombres, etc. Además, agrega información implícita y la estructura del código ejecutable continúa tomando forma. Esta etapa es el nexo entre etapas inicial y final del compilador (Análisis y Síntesis).
- 2) *Etapa de Síntesis (Vinculadas a características del código objeto, del hardware y la arquitectura)*
- Optimización del código: No se hace siempre y no lo hacen todos los compiladores. Estos programas pueden ser herramientas independientes, o estar incluidas en los compiladores e invocarse por medio de opciones de compilación. Existen diversas formas y cosas que se pueden optimizar como por ejemplo elegir entre velocidad de ejecución y tamaño del código ejecutable, generar código para un microprocesador específico dentro de una familia de procesadores, eliminar la comprobación de rangos o desbordamientos de pila, evaluación para expresiones

booleanas, eliminación de código muerto o no utilizado, eliminación de funciones no utilizadas, etc.

- Generación del código final: El compilador traduce el código fuente y el AST (o alguna representación interna) a código de máquina o código intermedio que la computadora pueda ejecutar.

c) La semántica interviene en el paso de Análisis semántico (semántica estática) y es de suma importancia dentro de la compilación ya que es el nexo entre las etapas de Análisis y las de Síntesis.

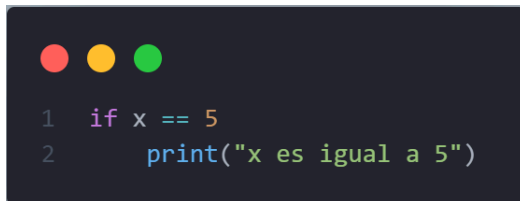
3. Compilación e Interpretación

| <i>Compilación</i> | <i>Interpretación</i> |
|--|--|
| Es un proceso de dos pasos donde el código fuente se traduce completamente a lenguaje de máquina antes de la ejecución. | La interpretación es un proceso de un solo paso donde el código fuente se traduce línea por línea o instrucción por instrucción mientras se ejecuta. |
| Produce un archivo ejecutable independiente del código fuente, que puede ser ejecutado sin la necesidad del código fuente original. | No se genera un archivo ejecutable independiente del código fuente original; el código fuente debe estar presente durante la ejecución. |
| Los programas compilados tienden a ser más rápidos en términos de ejecución, ya que el código se traduce directamente a lenguaje de máquina. | Los programas interpretados tienden a ser más lentos en términos de ejecución, ya que cada instrucción se traduce y ejecuta en tiempo real. |
| Los compiladores pueden realizar optimizaciones durante la compilación, lo que puede resultar en un mejor rendimiento del programa. | Los programas interpretados pueden ser más flexibles ya que el código fuente puede ser modificado y ejecutado directamente sin necesidad de recompilación. |
| Tiene como ventajas: Mayor eficiencia en tiempo de ejecución, mayor capacidad de optimización, independencia | Tiene como ventajas: Facilidad de desarrollo y depuración, flexibilidad para realizar cambios en el código sin |

| | |
|---|--|
| del código fuente una vez compilado. | necesidad de recompilación. |
| Tiene como desventajas: Requiere un paso adicional de compilación antes de la ejecución, lo que puede aumentar el tiempo de desarrollo inicial y el tamaño del archivo ejecutable. | Tiene como desventajas: Menor eficiencia en tiempo de ejecución, dependencia continua del código fuente durante la ejecución, falta de optimizaciones globales. |

4. Errores Sintácticos y Semánticos

- Error Sintáctico: Un error sintáctico ocurre cuando el código no sigue las reglas gramaticales del lenguaje de programación. Estos errores son detectados por el analizador sintáctico (parser) del lenguaje durante la fase de compilación o interpretación. Ejemplo de esto en Python:



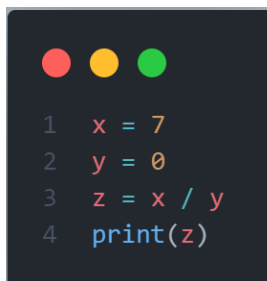
```

1  if x == 5
2      print("x es igual a 5")

```

En este caso, el error sintáctico se produce debido a la falta de dos puntos (:) al final de la declaración if. La salida del intérprete de Python sería algo como "SyntaxError: invalid syntax", indicando que hay un error en la sintaxis del código.

- Error Semántico: Un error semántico ocurre cuando el código está bien formado desde el punto de vista sintáctico, pero no produce los resultados esperados debido a una interpretación incorrecta del significado del código. Pueden manifestarse como comportamientos inesperados o resultados incorrectos durante la ejecución del programa. Ejemplo de esto en Python:



```

1  x = 7
2  y = 0
3  z = x / y
4  print(z)

```

En este ejemplo, el código es sintácticamente correcto, pero produce un error semántico porque intentamos dividir x por y, donde y tiene el valor de 0. Esto resultará en un error durante la ejecución, ya que no es posible dividir entre cero. El intérprete de

Python no detectará este error durante la fase de compilación o interpretación, pero generará un mensaje de error como "ZeroDivisionError: division by zero" durante la ejecución del programa.

La diferencia clave entre un error sintáctico y uno semántico radica en que los errores sintácticos se refieren a problemas con la estructura y la gramática del código, mientras que los errores semánticos se refieren a problemas con el significado o la lógica del código.

5. Análisis de programas:

a) Pascal

```
1 Program P                                // Sintáctico: Falta ;
2     var 5: integer;                       // Sintáctico: No se puede tener un número como nombre de una variable
3     var a:char;
4 Begin
5     for i:=5 to 10 do begin               // Semántico: La variable "i" no está declarada. Detectado en Compilación.
6         write(a);                         // Semántico: La variable "a" no está inicializada. Detectado en Compilación.
7         a=a+1;                            {
8                                           Sintáctico: Es a:=a+1;
9                                           Semántico: La variable "a" es de tipo "char", sumarle
10                                          1 no es una operación válida. Detectado en Compilación.
11                                          }
12     end;
13 End.
```

b) Java

```
1 public String tabla(int numero, arrayList<Boolean> listado) { // Sintáctico: Es ArrayList.
2     String result = null;
3     for(i = 1; i < 11; i--) {
4
5                                     /*
6                                     Lógico: Loop infinito.
7                                     Semántico: La variable "i" no está
8                                     declarada. Detectado en Compilación.
9                                     */
10
11                                     /*
12                                     Semántico: Concatenación de null en la
13                                     cadena. Detectado en Compilación.
14                                     */
15
16                                     /*
17                                     Sintáctico: El lado izquierdo de la
18                                     asignación debe ser una variable.
19                                     Sintáctico: "BOOLEAN" no existe, debería
20                                     ser Boolean o boolean.
21                                     */
22
23                                     /*
24                                     Semántico: Error de tipo en el return.
25                                     Detectado en Compilación.
26                                     */
27
28                                     */
29
30     }
31     return true;
32 }
```

c) C

```

1  # include <stdio.h>
2  int suma; /* Esta es una variable global */
3  int main()
4  {   int indice;
5      encabezado;
6      for (indice = 1 ; indice <= 7 ; indice ++)
7          cuadrado (indice);
8
9
10
11      final(); Llama a la función final */
12
13
14
15
16      return 0;
17  }
18  cuadrado (numero)
19
20
21
22
23  int numero;
24
25
26
27  {   int numero_cuadrado;
28      numero_cuadrado == numero * numero;
29
30
31
32
33
34      suma += numero_cuadrado;
35
36
37
38      printf("El cuadrado de %d es %d\n",
39             numero, numero_cuadrado);
40  }

```

// Sintáctico: La variable "encabezado" no está declarada.
 // Sintáctico: Falta uso de llaves "{}" para el for.
 /*
 Semántico: La declaración de "cuadrado" no existe en ese contexto.
 Detectado en Compilación.
 */
 /*
 Sintáctico: Falta la apertura del comentario.
 Semántico: La declaración de "final" no existe.
 Detectado en Compilación.
 */
 /*
 Sintáctico: Se debe especificar el tipo de retorno o void.
 Sintáctico: La función "cuadrado(numero)" necesita el uso
 de llaves "{}".
 */
 /*
 Semántico: La sentencia "int numero;" rompe la declaración
 de la función "cuadrado". Detectado en Compilación.
 */
 /*
 Semántico: La variable "numero" no está inicializada.
 Detectado en Compilación.
 Sintáctico: Para asignar la sentencia debería ser
 "numero_cuadrado = numero * numero".
 */
 /*
 Semántico: Se asigna una variable que no está inicializada.
 Detectado en Compilación.
 */

d) Python

```

1  #!/usr/bin/python
2  print "\nDEFINICION DE NUMEROS PRIMOS"
3  r = 1
4  while r = True:
5      N = input("\nDame el numero a analizar: ")
6
7
8
9      i = 3
10     fact = 0
11     if (N mod 2 == 0) and (N != 2):
12         print "\nEl numero %d NO es primo\n" % N
13     else:
14         while i <= (N^0.5):
15             if (N % i) == 0:
16                 mensaje="\nEl numero ingresado NO es primo\n" % N
17
18
19
20                 msg = mensaje[4:6]
21                 print msg
22                 fact = 1
23                 i+=2
24             if fact == 0:
25                 print "\nEl numero %d SI es primo\n" % N
26  r = input("Consultar otro número? SI (1) o NO (0)---> ")

```

Sintáctico: Falta el uso de "()".
 # Sintáctico: La comparación se hace con "==".
 ""
 Semántico: El input necesita el casteo a un tipo numérico.
 Detectado en Ejecución.
 ""
 # Sintáctico: "mod" no existe, es "%".
 # Sintáctico: Falta el uso de "()".
 # Sintáctico: La potencia se realiza con "**".
 ""
 Semántico: Hace falta el uso de "%d".
 Detectado en Ejecución.
 ""
 # Sintáctico: Falta el uso de "()".
 # Sintáctico: Falta el uso de "()".

e) Ruby

```
1 def ej1
2   puts 'Hola, ¿Cuál es tu nombre?' # Sintáctico: Es "puts".
3   nom = gets.chomp
4   puts 'Mi nombre es ', + nom      # Sintáctico: La "," está de más.
5   puts 'Mi sobrenombre es 'Juan''
6   puts 'Tengo 10 años'
7   meses = edad*12                  # Semántico: La variable "edad" no existe. Detectado en Ejecución.
8   días = 'meses' *30               # Sintáctico: No es posible multiplicar strings.
9   hs= 'días * 24'
10  puts 'Eso es: meses + ' meses o ' + días + ' días o ' + hs + ' horas' # Semántico: Los "+" para concatenar van por fuera de las comillas.
11                                     # Detectado en Ejecución.
12  puts 'vos cuántos años tenés'
13  edad2 = gets.chomp
14  edad = edad + edad2.to_i
15  puts 'entre ambos tenemos ' + edad + ' años'
16  puts '¿Sabes que hay ' + name.length.to_s + ' caracteres en tu nombre, ' + name + '?' #Sintáctico: la variable "name" no está definida.
17 end
18
```

- **NOTA:** No entendí bien el punto de tener que rehacer el código.

6. Variables predefinidas en Ruby

- **“self”:** Es una variable especial que hace referencia al objeto actual. En un contexto de clase, “self” hace referencia a la clase misma. Esta se puede usar dentro de métodos de clase para hacer referencia a la clase en la que se está definiendo el método. Al iniciar el intérprete, “self” tiene el valor main, ya que este es el primer objeto que se crea.
- **“nil”:** Es un objeto especial que representa la ausencia de valor o la nada. Es el equivalente a “null” en otros lenguajes. Se utiliza para indicar que una variable o expresión no tiene un valor asignado o no retorna nada. En contextos booleanos, “nil” es considerado como un valor falso.

7. Null y undefined en JavaScript

- **“undefined”:** Es un valor primitivo que se asigna automáticamente a las variables que se han declarado pero no se les ha asignado ningún valor. También se devuelve cuando se accede a las propiedades de un objeto que no existen.
- **“null”:** Es un valor primitivo en JavaScript. Se utiliza para representar la ausencia intencional de valor. Es decir, se puede asignar a una variable para indicar explícitamente que no hay ningún valor presente.
- La diferencia principal entre null y undefined es que null se utiliza para indicar explícitamente la ausencia de valor, mientras que undefined se utiliza para representar la ausencia de valor cuando una variable aún no ha sido asignada o cuando se accede a una propiedad que no existe en un objeto.

8. Sentencia break

- **C:** En C, la sentencia break se utiliza para salir de un bucle (for, while o do-while) o de un switch. Cuando se encuentra una instrucción break dentro de un bucle o un switch, el control del programa salta inmediatamente fuera del bucle o del switch y la ejecución continúa con la primera línea de código después del bucle o del switch.
- **PHP:** En PHP, la sentencia break se utiliza principalmente dentro de bucles (for, while, do-while) y de un switch. Es igual que en C.
- **JavaScript:** En JavaScript, break se utiliza dentro de bucles (for, while, do-while) y de un switch para salir de ellos. La sintaxis y la semántica de break en JavaScript son similares a las de C y PHP.
- **Ruby:** En Ruby, la sentencia break también se utiliza dentro de bucles (for, while, until) para salir de ellos. A diferencia de C, PHP y JavaScript, Ruby no tiene un switch nativo, por lo que break se utiliza principalmente para salir de bucles.

9. Concepto de Ligadura

- Es el momento en el que a un atributo de una entidad se le asocia un valor determinado. Es un concepto central en la definición de la semántica de los lenguajes de programación. Se puede dividir en 2 tipos: Estática y Dinámica.
- **Ligadura Estática:** Se establece antes de la ejecución del programa, ya sea en la definición del lenguaje, la implementación o en la compilación. En cuanto a su Estabilidad, este tipo de Ligadura no se puede modificar.
- **Ligadura Dinámica:** Se establece durante la ejecución. En cuanto a su Estabilidad, se puede modificar durante la ejecución del programa de acuerdo a alguna regla específica del lenguaje. Existe una excepción en cuanto a la Estabilidad con las Constantes, su ligadura se produce en tiempo de ejecución pero esta no puede ser modificada luego de ser establecida.
- **Ejemplo:**

Ejemplos:

◦ En **Definición**

- Forma de las sentencias
- Estructura del programa
- Nombres de los tipos predefinidos

◦ En **Implementación**

- Representación de los números y sus operaciones

◦ En **Compilación**

- Asignación del tipo a las variables

En lenguaje C

int

Para denominar a los enteros

int

- Representación
- Operaciones que pueden realizarse sobre ellos

int a

- Se liga tipo a la variable

◦ En **Ejecución**

- Variables con sus valores
- Variables con su lugar de almacenamiento

int a

- el valor de una variable entera se liga en ejecución y puede cambiarse muchas veces.