📒 Practica 3

Siguiente

Anterior

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

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 es sintácticamente válido

Para luego poder darle significado a una construcción del lenguaje

Ejercicio 2

a) ¿Qué significa compilar un programa?

Compilar un programa significa convertir el código fuente escrito en un lenguaje de programación de alto nivel en un lenguaje de bajo nivel que pueda ser entendido por la computadora y ejecutado por el procesador.

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

Los pasos necesarios para compilar un programa son los siguientes:

- `Análisis léxico` En esta etapa, el compilador lee el código fuente y lo divide en tokens o palabras clave del lenguaje de programación, como identificadores, constantes, operadores y símbolos.
- `Análisis sintáctico` El compilador utiliza una gramática formal para comprobar si los tokens se combinan de acuerdo con las reglas sintácticas del lenguaje de programación. Si se detecta un error, el compilador emite

un mensaje de error.

- `Análisis semántico` En esta etapa, el compilador verifica que el programa tenga sentido semántico, es decir, que las operaciones y las funciones se estén utilizando correctamente y que el tipo de datos sea coherente.
- `Generación de código intermedio` El compilador crea un código intermedio que es más fácil de traducir al código de máquina. Este código intermedio es a menudo un lenguaje ensamblador de bajo nivel, pero puede ser una representación en un lenguaje de alto nivel, como C.
- `Optimización de código` El compilador intenta mejorar el código intermedio para que se ejecute más rápido o utilice menos memoria.
- `Generación de código de máquina` El compilador convierte el código intermedio en código de máquina que la computadora puede entender y ejecutar.
- `Enlazado` El compilador vincula el código generado con las bibliotecas y los archivos necesarios para producir un programa ejecutable.

`c)` ¿En qué paso interviene la semántica y cual es su importancia dentro de la compilación?

La etapa de análisis semántico es donde se evalúa la corrección semántica del programa. Esta etapa es importante porque garantiza que el programa se comporte como se espera y evita errores comunes como divisiones por cero o intentos de acceso a memoria no asignada. Si un error semántico se pasa por alto, puede ser difícil de detectar y puede causar problemas en el programa en tiempo de ejecución.

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.

No es lo mismo compilar un programa que interpretarlo puesto que son dos formas diferentes de traducir y ejecutar el código de un programa. Un compilador produce un archivo ejecutable a partir del código fuente y lo ejecuta directamente en la computadora, mientras que un interprete traduce el código fuente en tiempo real y lo ejecuta sin generar un ejecutable previamente.

Ejercicio 4

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

Un error sintáctico se produce cuando el código fuente no cumple con las reglas de sintaxis del lenguaje de programación en el que está escrito. En otras palabras, un error sintáctico ocurre cuando se comete un error de gramática en el código fuente que impide que el compilador o el intérprete entienda el significado del código.

Por ejemplo, en el lenguaje de programación Python, si se escribe:

```
if a == 5
    print("a es igual a 5")
```

Se producirá un error sintáctico porque falta el dos puntos (😃 después de la condición del if.

Por otro lado, un error semántico se produce cuando el código fuente cumple con las reglas de sintaxis del lenguaje de programación, pero su significado no es el que se pretendía. Es decir, el código puede compilarse o interpretarse sin errores, pero su resultado es incorrecto porque hay un error en la lógica o en la comprensión del problema que se está resolviendo.

Por ejemplo, supongamos que se escribe un código en Python para calcular el promedio de tres números:

```
a = 5
b = 6
c = 7
promedio = a + b + c / 3
print("El promedio es:", promedio)
```

En este caso, el código se compilará y se ejecutará sin errores, pero el resultado será incorrecto, ya que el operador de división (/) se evalúa antes que la suma. Para obtener el resultado correcto, la expresión debe ser escrita de la siguiente manera:

```
a = 5
b = 6
c = 7
promedio = (a + b + c) / 3
print("El promedio es:", promedio)
```

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

```
1.Program P
2.var
3.    5: integer;
4.var
5.    a:char;
6.Begin
7.    for i:=5 to 10 do begin
8.        write(a);
9.        a=a+1;
10.    end;
11.End.
```

Ayuda: Sintáctico 2, Semántico 3

Sintacticos

- `3.` , se está declarando una variable con un nombre inválido (un número no puede ser un identificador válido en Pascal).
- `9.` , se usa el operador "=" en lugar del operador ":=" para asignar un valor a la variable.

Semanticos

- `7.` , La variable `i` no esta declarada en el for
- `9.` , No fue inicializada la variable `a`
- `9.` , se usa el operador "+" para concatenar caracteres en lugar del operador "succ()" para obtener el siguiente carácter en la tabla ASCII.

b) Java:

```
1.public String tabla(int numero, arrayList<Boolean> listado)
2.{
3.    String result = null;
4.    for(i = 1; i < 11; i--) {
5.        result += numero + "x" + i + "=" + (i*numero) + "\n";
6.        listado.get(listado.size()-1)=(BOOLEAN) numero>i;
7.    }
8.    return true;
9.}
```

Ayuda: Sintácticos 4, Semánticos 3, Lógico 1

Sintacticos

- 1. arrayList empieza con mayuscula
- 6. Estas asignando a un valor, otro valor, ejemplo 20=true
- 6. Ese Boolean esta mal escrito

Semanticos

- 4. falta declarar la variable i
- 5. falta declarar la variable i
- 6. falta declarar la variable i
- 5. Estas multiplicando una variable no declarada con un entero
- 1. El metodo espera un string y devuelve un boolean
- 6. La asignación es incorrecta, se tendria que usar un set

Logico

- 3. Deberia inicializar con cadena vacia en lugar de null, ya que no se puede concatenar un string con null en java
- 4. Se queda en un bucle infinito, ya que nunca encuentra el tope por decrementar

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.    final(); Llama a la función final */
9.    return 0;
10.}
11.cuadrado (numero)
12.int numero;
13.{   int numero_cuadrado;
14.    numero_cuadrado == numero * numero;
15.    suma += numero_cuadrado;
16.    printf("El cuadrado de %d es %d\n",
17.    numero, numero_cuadrado);
18.}
```

Ayuda: Sintácticos 2, Semánticos 6

Sintacticos

- `8.` Hay un comentario de cierre pero no de apertura
- `5.` No esta definido el tipo de variable de `encabezado`

Semanticos

- `8.` La funcion **final** no esta definida
- `15.` Suma no esta inicializada, ya que le estas haciendi +=
- `13.` numero_cuadrado no esta inicializado
- `11.` La funcion cuadrado no tiene un valor asignado
- `11.` Faltan los dos ; de la llamada a la funcion cuadrado
- `11.` El parametro numero no tiene tipo

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.      i = 3
7.      fact = 0
8.      if (N mod 2 == 0) and (N != 2):
9.                      print "\nEl numero %d NO es primo\n" % N
10.     else:
11.             while i <= (N^0.5):
12.                     if (N % i) == 0:
13.                             mensaje="\nEl numero ingresado NO es primo\n" % N
14.                             msg = mensaje[4:6]
15.                             print msg
16.                             fact = 1
17.             i+=2
18.             if fact == 0:
19.                     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 de Sintaxis Se detectan en compilación

- `1.` dentro del `while` tenes un error sintáctico, ya que estas asignando un valor, no comparandolo
- `8.` El `mod` como operación no existe en python, en su lugar se usa el %
- `11.` , El operador de potencia es **

Errores Semanticos

- `8.` estas comparando un string con un entero
- `11 y 12` estas comparando `N` que es un string con un nro

e) Ruby

```ruby
1.def ej1
2.      Puts 'Hola, ¿Cuál es tu nombre?'
3.      nom = gets.chomp
4.      puts 'Mi nombre es ', + nom
5.      puts 'Mi sobrenombre es 'Juan''
6.      puts 'Tengo 10 años'
7.      meses = edad*12
8.      dias = 'meses' *30
9.      hs= 'dias * 24'
10.     puts 'Eso es: meses + ' meses o ' + dias + ' días o ' + hs + ' horas'
11.     puts 'vos cuántos años tenés'
12.     edad2 = gets.chomp
13.     edad = edad + edad2.to_i
14.     puts 'entre ambos tenemos ' + edad + ' años'
15.     puts '¿Sabes que hay ' + name.length.to_s + ' caracteres en tu nombre, ' + name + '?'
16.end
```

Ayuda: Semánticos +4

- `1.` Puts debe ser puts (minúsculas).
- `5.` Nadie sabe lo que es juan, si fuera variable le faltan los + y seguro no tendria la primera en mayuscula.
- `7.` Edad no está definido antes de ser utilizado.
- `8.` Meses es un número pero se usa como una cadena en 'meses' 30.
- `9.` Dias es una cadena pero se usa como un número en 'dias 24'.
- `10.` La concatenación de cadenas en la línea que comienza con 'Eso es: meses + ' es incorrecta.
- `15.` La variable name no está definida antes de ser utilizada.

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;
```

```
def ordenar_arreglo(array):
  print (sorted(array))
ordenar_arreglo([3,2,1])
```

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.

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?

- `self` Es una variable predefinida que hace referencia al objeto actual. En otras palabras, self se refiere al objeto que está siendo ejecutado en ese momento. Se usa principalmente para acceder a los métodos y propiedades de ese objeto.
- `nil` Es un valor especial que representa la ausencia de un valor o una referencia nula. En otras palabras, nil se utiliza para indicar que una variable no tiene un valor asignado. Se puede pensar en nil como una especie de "nada" en Ruby. También se utiliza como un valor de retorno por defecto para los métodos que no devuelven nada.

Ejercicio 7

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

En JavaScript, null y undefined son dos valores que se utilizan para representar la ausencia de un valor o un valor indefinido. A pesar de que ambos valores se usan para denotar una ausencia de valor, tienen diferencias importantes en cuanto a su semántica y uso:

- `null` Es un valor primitivo que se utiliza para indicar la ausencia de cualquier objeto o valor. Se puede asignar a una variable como un valor explícito.

- `let variable = null;`
- También se puede utilizar para inicializar una variable que se utilizará más adelante en el código:

```
let variable;
variable = null;
```

Se considera que null es un valor asignado por el programador, ya que se utiliza para indicar que se ha asignado un valor ausente de forma intencional.

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

C

La sentencia break se utiliza para interrumpir la ejecución de un bucle for, while o do-while. Debe estar dentro del cuerpo del bucle, y si se utiliza dentro de un bucle anidado, solo se interrumpe el bucle más cercano. Se utiliza comúnmente con una condición if para salir del bucle cuando se cumple una cierta condición.

PHP

La sentencia break se utiliza principalmente para interrumpir la ejecución de un bucle foreach. Al igual que en C, debe estar dentro del cuerpo del bucle y se utiliza comúnmente con una condición if para salir del bucle cuando se cumple una cierta condición.

JavaScript

La sentencia break se utiliza para interrumpir la ejecución de un bucle for, while, do-while o switch. Al igual que en C y PHP, debe estar dentro del cuerpo del bucle y se utiliza comúnmente con una condición if para salir del bucle cuando se cumple una cierta condición.

Ruby

La sentencia break se utiliza para interrumpir la ejecución de un bucle for, while o until. También se puede utilizar para salir de un bloque de código. En Ruby, la sentencia break puede devolver un valor, lo que la hace más versátil que en otros lenguajes. Debe estar dentro del cuerpo del bucle y se utiliza comúnmente con una condición if para salir del bucle cuando se cumple una cierta condición.

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)

Existen dos tipos de ligadura: la ligadura estática y la ligadura dinámica. La ligadura estática se produce en tiempo de compilación, donde los nombres de las variables y funciones se asocian con su ubicación en memoria antes de

la ejecución del programa. En cambio, la ligadura dinámica se produce en tiempo de ejecución, donde los nombres se asocian con su ubicación en memoria a medida que se van necesitando durante la ejecución del programa.

Un ejemplo de ligadura estática sería en el lenguaje C, donde la declaración de una variable debe preceder a su uso en el programa. Por ejemplo:

```
int main() {
int x = 5;
int y = x + 2;
return 0;
}
```

En este caso, la variable "x" se declara antes de su uso en la expresión "y = x + 2", lo que permite al compilador asignar su dirección de memoria correspondiente en tiempo de compilación.

Un ejemplo de ligadura dinámica sería en el lenguaje Python, donde una función puede recibir un argumento de cualquier tipo y su tipo puede cambiar durante la ejecución del programa. Por ejemplo:

```
def multiply(x, y):
return x * y

a = 5
b = "hello"

print(multiply(a, 2)) # output: 10
print(multiply(b, 2)) # output: "hellohello"
```

En este caso, la función "multiply" puede recibir argumentos de cualquier tipo y su tipo puede cambiar durante la ejecución del programa, lo que permite una mayor flexibilidad en el código.
