Práctica 3

martes, 2 de abril de 2024 17:22

Ejercicio 1: ¿Qué define 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

Ejercicio 2:

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

a- realiza la traducción a lenguaje de máquina.Pasa por todas las instrucciones antes de la ejecución (ventajas y desventajas). El código que se genera se guarda y se puede reusar ya compilado.
b-

1) Etapa de Análisis
   a) Análisis léxico (Programa Scanner):
      Análisis a nivel palabra. Divide programas en sus elementos o categorías (identificadores, delimitadores, símbolos especiales, etc). Analiza el tipo de cada uno para ver si son tokens válidos. Genera errores si la entrada no coincide con ninguna categoría. **El resultado de este paso será el descubrimiento de los items léxicos o tokens y detección de errores.**
   b) Análisis sintáctico (Programa Parser):
      Análisis a nivel sentencia (estructura). Usa los tokens del analizador léxico, representa las estructuras como un arbol. Se identifican las estructuras de las sentencias mediante los tokens. **El objetivo principal de un árbol de derivación es representar una sentencia del lenguaje y validar de esta forma que pertenece o no a la gramática**
   c) Análisis semántico (Programa de Semántica estática):
      Procesa las estructuras sintácticas (reconocidas por el analizador sintáctico). Realiza la comprobación de tipos (aplica gramática de atributos). Realiza comprobaciones de duplicados, problema de tipos, etc. Realiza comprobaciones de nombres. **Es el nexo entre etapas inicial y final del compilador(Análisis y Síntesis)**

2) Generación de código intermedio:
realizar la transformación del "código fuente" en una representación de "código intermedio" para una máquina abstracta, Queda una representación independiente de la máquina en la que se va a ejecutar el programa.

3) Etapa de Síntesis
   a) Construye el programa ejecutable y genera el código necesario. Si hay traducción separada de otros módulos (librerías por ejemplo) interviene el Linkeditor (Programa) y se enlazan los distintos módulos objeto del programa.
   b) Optimización: No se hace siempre y no lo hacen todos los compiladores. Los optimizadores de código (programas) pueden ser herramientas independientes, o estar incluidas en los compiladores e invocarse por medio de opciones de 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.

Compilador vs intérprete.

Intérprete: ejecuta en ejecucución
Compi:lo hace antes de ejecutar

Intérprete:sigue orden del programa
Compi:ve todo

Intérprete: cada vez que pasa por un loop lo revisa, si tiene muchas iteraciones es medio un bajón
Compi:una sola vez. Capaz tarda más en analizar todo pero una vez hecho ya está } lenta.

Intérprete: más lento, - eficiente
Compi: más rápido porque ya fue compilado } eficiente

Intérprete: ocupa menos espacio
Compilador: pasa por todas las sentencias y escribe en código de máquina, se hacen muchas tablas que capaz no se van a usar. } espacio.

Intérprete: errores se ven directamente, se detectan los errores por donde pasa la ejecución
Compi: referencia al código fuente se pierde en el cod objeto, es mucho más difícil identificar errores } identificación de errores.

Primero interpreto y después compilo → Combinan está bueno pq veo errores + fácil.

Primero compilo y después interpreto → Traducel a cod intermedio q' después interpreto. Código portable.
→ Como Java, me llevo programa → se interpreta en destino.

Compilan todo q' corra + rápido pq ya pasó a un lenguaje + cercano al q' la máquina conoce.

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

Error sintáctico: Cuando el código no sigue la estructura gramatical o reglas de sitaxis del programa.
Error semántico: el código está escrito correctamente pero la lógica o significado son incorrectos.

Error de sintaxis: falta cerrar el paréntesis (python)
print("Hola mundo"

Error de semántica: error de tipos
Int i=10;
i="hbla"

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.
```

Sintáctico: el nombre de una variable no puede arrancan con un nro
Semántico: i no está declarada
Semántico: a no está inicializada
Semántico: error de tipos, sintáctico: sería a:=

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;
   }
```

Sintáctico: el nombre de una variable no puede arrancan con un nro
Semántico: i no está declarada
Semántico/sintáctico: si está comparando sería ==, si estuviera intentando asignar también porque no se guarda así el valor de un elemento de una lista.
Sintáctico: sería Boolean el casting
Semántico: La función pide retornar una String

Ayuda:
Sintácticos 4, Semánticos 3, Lógico 1

Lógico: el for no tiene sentido, genera un bucle 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);
}
```

Semántico: Watafak, Sintáctico: declar
Semántico: La función cuadrado debería estar declarada arriba
Sintáctico: No está la apertura del comentario
Sintáctico: No dice de qué tipo es número
Sintáctico: No está el valor de retorno de la función
Semántico: declara una variable del mismo nombre que una ya declarada
Semántico: la variable suma no está inicializada
Semántico: la variable numero_cuadrado no está inicializada

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)--->> ")
```

Sintáctico: faltan ()
Sintáctico: sería ==
Sintáctico: sería %, and sería &
Sintáctico: sería **
Sintáctico: mod de una string???

Ayuda: *Sintácticos 2, Semánticos 3*

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
```

Sintáctico: No se pueden usar así las comillas
Semántico: edad no está definido
Sintáctico: concatenación mal hecha
Semántico: edad no inicializada
Semántico: edad no es string

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;
```

```
import java.util.Arrays;
import java.util.Random;

public class Main {
    public static boolean estaOrdenado(char[] arreglo) {
        for (int i = 0; i < arreglo.length - 1; i++) {
            if (arreglo[i] > arreglo[i + 1]) {
                return false;
            }
        }
        return true;
    }

    public static void bogosort(char[] arreglo) {
        Random rand = new Random();
        while (!estaOrdenado(arreglo)) {
            for (int i = 0; i < arreglo.length; i++) {
                int j = rand.nextInt(arreglo.length);
                char temp = arreglo[i];
                arreglo[i] = arreglo[j];
                arreglo[j] = temp;
            }
        }
    }

    public static void main(String[] args) {
        char[] arreglo = {'b', 'c', 'a', 'f', 'd', 'e'};
        System.out.println("Arreglo desordenado: " +
Arrays.toString(arreglo));
        bogosort(arreglo);
        System.out.println("Arreglo ordenado: " +
Arrays.toString(arreglo));
    }
}
```

- `estaOrdenado(char[] arreglo)`: Esta función verifica si el arreglo está ordenado.<
- `bogosort(char[] arreglo)`: Este método implementa el algoritmo bogosort. En cada iteración, mezcla aleatoriamente los elementos del arreglo hasta que estén ordenados.
- `Random rand = new Random();`: Se utiliza para generar números aleatorios.
- `while (!estaOrdenado(arreglo)) { ... }`: Este bucle se ejecuta hasta que el arreglo esté ordenado.
- `for (int i = 0; i < arreglo.length; i++) { ... }`: Este bucle itera sobre cada elemento del arreglo y lo intercambia con otro elemento aleatorio.
- `System.out.println("Arreglo desordenado: " + Arrays.toString(arreglo));`: Imprime el arreglo antes de ordenarlo.
- `System.out.println("Arreglo ordenado: " + Arrays.toString(arreglo));`: Imprime el arreglo después de ordenarlo.

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: En Ruby, self se refiere al objeto actual en el contexto en el que se encuentra. Dentro de métodos de instancia, self hace referencia a la instancia de la clase. Fuera de los métodos, self se refiere al objeto principal (main).
- nil: En Ruby, nil es un objeto especial que representa la ausencia de valor o la nada. Se utiliza para indicar que una variable no tiene ningún valor asignado. Por ejemplo, cuando un método no devuelve explícitamente un valor, implícitamente devuelve nil.

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

- null: En JavaScript, null es un valor primitivo que representa la ausencia intencionada de cualquier objeto o valor. Se utiliza para indicar que una variable no contiene ningún valor.
- undefined: En JavaScript, undefined indica que una variable ha sido declarada pero no inicializada, o que una propiedad no está definida en un objeto. También es devuelto por funciones que no devuelven un valor explícito.

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: En C, break se utiliza para salir de un bucle for, while o do-while de forma prematura.
- PHP: En PHP, break se utiliza de manera similar a C para salir de bucles for, while, do-while y switch.
- JavaScript: En JavaScript, break se utiliza para salir de bucles for, while, do-while o switch.
- Ruby: En Ruby, break se utiliza dentro de bucles while, until, for y bloques do..end o {}.

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)

La ligadura se refiere a la asociación de un nombre de variable o identificador con un valor en un contexto particular. La ligadura puede ser estática, donde la asociación se establece en tiempo de compilación y no cambia durante la ejecución, o dinámica, donde la asociación se

resuelve en tiempo de ejecución y puede cambiar durante la ejecución del programa. Las diferencias entre ligadura estática y dinámica afectan cómo se comportan los programas y cómo se gestionan las variables en el codigo.