Práctica 3

Ejercicio 1: ¿Qué define la semántica?

Define 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?**
El compilador es un programa que traduce nuestro programa previo a la ejecución. Prepara el programa en el lenguaje máquina para que luego pueda ser ejecutado. El compilador toma todo el programa escrito en un lenguaje de alto nivel que llamamos lenguaje fuente antes de su ejecución. Luego de la compilación va a generar un lenguaje objeto que es generalmente el ejecutable (escrito en lenguaje de máquina .exe) o un lenguaje de nivel intermedio (o lenguaje ensamblador .obj).

b. **Describa brevemente cada uno de los pasos necesarios para compilar un programa.**
Los compiladores pueden ejecutarse en un solo paso o en dos. Dichos pasos cumplen con varias etapas, principalmente se pueden dividir en dos:

- Análisis
    - Análisis léxico (Scanner): Hay un análisis a nivel de palabra (LEXEMA).
        - Se fija que cada elemento del programa se corresponda con un operador, un identificador, etc., básicamente divide el programa en sus elementos: identificadores, delimitadores, símbolos especiales, números, palabras clave, palabras reservadas, comentarios, etc. De esta forma ve si todas las palabras son válidas o no.
        - Es el que lleva más tiempo.
        - Filtra comentarios y separadores como: espacios en blanco, tabulaciones, etc.
        - Genera errores si la entrada no coincide con ninguna categoría léxica.
        - Con esto descubrimos los tokens
        - Pone los identificadores en la tabla de símbolos.
        - Reemplaza cada símbolo por su entrada en la tabla de símbolos
        - El resultado de este paso será el descubrimiento de los items léxicos o tokens.

ID	Componente léxico	Lexema	Patrón
201	IF	if	if
202	OP_RELACIONAL	<, >, !=, ==	<, >, !=, ==
203	IDENTIFICADOR	var, s1, suma, prom	letra (letra | dígito | gb)*
204	ENTERO	2, 23, 5124	(dígito)+

    - Análisis semántico (Parser):

        - Se realiza luego de análisis léxico.
        - Se analiza a nivel de cada sentencia.
        - Busca estructuras, sentencia, declaración, expresiones, variable, ayudándose de los tokens.
        - Se alterna el analizador sintáctico con el análisis semántico.
        - Construye el árbol sintáctico del programa.
    - Análisis semántico (semántica estática):
        - Es la fase más importante.
        - Todas las estructuras sintácticas reconocidas son analizadas.
        - Se realiza una comprobación de tipos y se agrega información implícita (variables no declaradas).
        - Se agregan tablas de símbolos de los descriptores de tipos, etc.
        - Se hace comprobaciones de nombres y duplicados.
        - Nexo entre etapas inicial y final del compilador

Luego de esta etapa se puede realizarla la transformación del código fuente en una representación de código intermedio. Esta representación intermedia llamada código intermedio se parece al código objeto, pero sigue siendo independiente de la máquina. El código objeto es dependiente de la máquina. Este código intermedio debe ser fácil de producir y debe ser fácil de traducir al programa objeto. Hay varias técnicas, la más común es el código de tres dimensiones

- Síntesis: Se construye el programa ejecutable, genera el código necesario y se optimiza el programa generado. Si hay traducción separada de módulos se enlazan los distintos módulos objeto del programa (módulos, unidades, librerías, procedimientos, funciones, subrutinas, macros, etc.) mediante el linkeditor
    - Se genera el módulo de carga. Programa objeto completo.
    - Se realiza el proceso de optimización. (Optativo)

c. ¿En qué paso interviene la semántica y cuál es su importancia dentro de la compilación?
La semántica interviene en el análisis y es de sumamente importancia ya que es el nexo entre las etapas inicial y final del compilador

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, no es lo mismo.

El intérprete y el compilador se puede comparar de diferentes formas

- En como ejecuta
    - Interprete: Lo hace durante la ejecución. Ejecuta sentencia a sentencia, lo traduce y lo ejecuta. Pasa un montón de veces. Para ser ejecutado en otra máquina se necesita tener si o si el intérprete instalado.
    - Compilador: Ocurre antes de ejecutar. Compila todo y genera un código objeto de un lenguaje de un modelo más bajo. El programa fuente no será publico
- Orden que lo ejecuta
    - Interprete: Sigue el orden lógico de ejecución (ya que va sentencia por sentencia del código)
    - Compilador: Sigue el orden físico de las sentencias.
- Tiempo de ejecución

    - Interprete: Por cada sentencia se realiza el proceso de decodificación para determinas las operaciones a ejecutar y sus operandos. Si la sentencia está en un proceso iterativo, se realiza la tarea tantas veces como sea requerido. Puede afectar la velocidad de proceso.
    - Compilador: Genera código de máquina para cada sentencia. No repite lazos, se decodifica una sola vez.
- Eficiencia
    - Interprete: Mas lento en ejecución. Se repite el proceso cada vez que se ejecuta el programa.
    - Compilador: Mas rápido desde el punto de vista del hardware, pero tarda más en compilar. Detectó más errores al pasar por todas las sentencias. Está listo para ser ejecutado. Ya compilado es más eficiente.
- Espacio ocupado
    - Interprete: Ocupa menos espacio de memoria ya que cada sentencia se deja en la forma original y las instrucciones necesarias para ejecutarlas se almacenan en los subprogramas del interprete en memoria.
    - Compilador: Una sentencia puede ocupar decenas o centenas de sentencias de máquina al pasar a código objeto
- Detección de errores
    - Interprete: Las sentencias del código fuente puede ser relacionadas directamente con la que se está ejecutando. Se puede ubicar el error, es más fácil detectarlos por donde pasa la ejecución y es más fácil corregirlos
    - Compilador: Le cuesta más determinar los errores ya que cualquier referencia al código fuente se pierden en el código objeto. Se pierde la referencia entre el código fuente y el código objeto. Es casi imposible ubicar el error, pobres en significado para el programador.

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

Un error sintáctico ocurre cuando el programador escribe código que no va de acuerdo con las reglas de escritura del lenguaje de programación. El formato puede estar especificado en documentos BNF/EBNF.

Ejemplo: String s = “abc”;

Un error semántico ocurre cuando, si bien la sintaxis es correcta, pero la semántica o significado no es el que se pretendía. Hay errores semánticos que se detectan en compilación (semántica estática) y otros durante la ejecución (semántica dinámica).

Ejemplo: String s; s = 9;

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

Sintáctico

1ro: Falta ; luego de Program P

2do: No se puede tener un numero como nombre de variable.

3ro: a=a+1 debería ser a:=a+1

Semántico:

1ro: La variable i no está declarada. Se detecta en compilación.

2do: No se puede realizar a=a+1 puesto que a un char no se le puede sumar un entero. Se detecta en compilación.

3ro: a no esta inicializado, por lo que no se puede imprimir. Se detecta en compilación.

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

Lógico

1ro: i siempre será menor a 11 ya que empieza en 1 y le voy restando, por lo que se trata de un loop infinito.

Sintácticos

1ro: debe ser ArrayList, no arrayList

2do: BOOLEAN deberia ser Boolean o boolean

3ro: listado.get(listado.size()-1)=(BOOLEAN) numero>i; listado.get(listado.size()-1) deberia ser una variable

Semántico

1ro: la variable i no está declarada. Se detecta en compilación.

2do no se puede realizar return true cuando no se especificó que la función iba a retornar un dato de tipo boolean. . Se detecta en compilación.

3ro: (BOOLEAN) numero>i; No puedo transformar un int en un boolean. Se detecta en compilación.

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

Sintácticos

1ro: encabezado; si se quiere llamar a una función faltan los ().

2do: Llama a la función final */ falta abrir el comentario

3ro: cuadrado (numero) faltan las llaves en caso de que sea la declaración de una función, estas llaves están puestas incorrectamente más abajo.

Semánticos

1ro: encabezado (); no es una función declarada. Se detecta en compilación.

2do: cuadrado (indice) no es una función declarada. Se detecta en compilación.

3ro: cuadrado (numero) e int numero; no está dentro de ningún bloque. Se detecta en compilación.

4to: final() la función final no esta declarada. Se detecta en compilación.

5to: numero_cuadrado == numero * numero; numero y numero_cuadrado no estan inicializado. Se detecta en compilacion

6to: suma += numero_cuadrado; suma no esta declarada. Se detecta en compilación.

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

Sintácticos

1ro: Faltan paréntesis en todos los print

2do: if (N mod 2 == 0) and (N != 2): el mod en Python es %

3ro: while i <= (N^0.5): la potencia es **

4to: while r = True la comparación se hace con ==

Semánticos

1ro: while r = True se esta comparando un int con un boolean. Se detecta en compilacion

2do: if (N mod 2 == 0) and (N != 2): hay que parsear el n a int con (int). Se detecta en compilacion

3ro: "\nEl numero ingresado NO es primo\n" % N falta el d%. Se detecta en ejecución

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

Sintáctico

1ro: Puts 'Hola, ¿Cuál es tu nombre?' puts va con mayuscula

2do: puts 'Mi sobrenombre es 'Juan'' se cierra el ‘’ y se sigue escribiendo, falta un “

Semánticos

1ro: meses = edad*12 nunca se declaró la edad. Se detecta en compilación.

2do: puts 'Eso es: meses + ' meses o ' + dias + ' días o ' + hs + ' horas' no existen las variables meses o y días o . Se detecta en compilación.

3ro: edad = edad + edad2.to_i edad no está declarada. Se detecta en compilación.

4to: puts '¿Sabes que hay ' + name.length.to_s + ' caracteres en tu nombre, ' + name + '?' name no esta definido. Se detecta en compilación.

Ejercicio 5: Dado el siguiente código escrito en pascal. Transcriba la misma funcionalidad de acuerdo con el 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.

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: el objeto que se está usando en ese instante. Al iniciar el intérprete self tiene el valor main, ya que este es el primer objeto que se crea. En una clase o definición de módulo, self es la clase o el módulo al que pertenece el objeto

nil: valor se utiliza para expresar la noción de “falta de un objeto”. Es un objeto Ruby especial que se utiliza para representar un valor "vacío" o "predeterminado". También es un valor "falso", lo que significa que se comporta como falso cuando se usa en una declaración condicional. Muchos métodos pueden devolver cero como resultado. Esto sucede cuando solicita un valor, pero ese valor no está disponible.

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

null: representa intencionalmente un valor nulo o "vacío"

undefined: una variable a la que no se le ha asignado valor, o no se ha declarado en absoluto (no se declara, no existe)

La diferencia es que null es intencionalmente declarada como vacio o nulo, undefined no, es asignado sólo por Javascript de forma automática como valor inicial cuando se define una variable y no se le dá un valor.

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

break

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. La instrucción break se usa con frecuencia para finalizar el procesamiento de un caso concreto en una instrucción switch. Si no existe una instrucción iterativa o una instrucción switch incluyente, se genera un error.

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, sentecia switch o label y transfiere el control del programa a la siguiente sentencia a la sentecia 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: Usado para terminar un bucle actual, normalmente usado en bucles while true. Lo simpático de ruby es que podemos definir una condición que se debe cumplir para que se ejecute el break (en la misma línea de este) → break if condición. Otro uso potente es que break puede recibir argumentos que devolverá al terminar el bucle.

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)

Los programas trabajan con entidades, estas entidades tienen atributos que tienen que establecerse antes de poder usar la entidad. La ligadura es la asociación entre la entidad y el atributo.

- Una ligadura es estática si se establece antes de la ejecución y no se puede cambiar. El termino estática referencia al momento del binding y a su estabilidad.
- Una ligadura es dinámica si se establece en el momento de la ejecución y puede cambiarse de acuerdo con alguna regla especifica del lenguaje. Excepción: constantes

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.