Ejercicio 1: ¿Qué define la semántica? 5
Ejercicio 2: 5
a. ¿Qué significa compilar un programa? 5
b. Describa brevemente cada uno de los pasos necesarios para compilar un programa. 5
c. ¿En qué paso interviene la semántica y cual es su importancia dentro de la compilación? 6
Ejercicio 3: Con respecto al punto anterior ¿es lo mismo compilar un programa que interpretarlo? 7
Justifique su respuesta mostrando las diferencias básicas, ventajas y desventajas de cada uno. 7
Ejercicio 4: Explique claramente la diferencia entre un error sintáctico y uno semántico. Ejemplifique cada caso. 7
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. 8
a) Pascal 8
1. Program P //Error sintáctico (falta ;) 8
2. var 5: integer; var a:char; //Error sintáctico, nombre de variable inicia con un número. 8
3. Begin 8
4. for i:=5 to 10 do begin //Semantico, i no declarado en compilación 8
5. write(a); // Semántico, a no esta inicializado se detecta en compilación 8
6. a=a+1; //Sintattico, va := , Semántico Suma un char a un entero, en compilación 8
7. end; 8
End. Ayuda: Sintáctico 2, Semántico 3. 8
b) Java: 9
1. public String tabla(int numero, arrayList<Boolean> listado) //Sintactico, El tipo es ArrayList 9

2. { 9
3. String result = null; 9
4. for(i = 1; i < 11; i--) { //Semántico, i no esta declarado, en compilación 9
5. result += numero + "x" + i + "=" + (i*numero) + "\n"; //Semántico, result no esta inicializado. En compilación 9
6. listado.get(listado.size()-1)=(BOOLEAN) numero>i; //SintacticoX2, BOOLEAN no existe, lado izquierdo debe ser una variable 9
7. } 9
8. return true; //Semántico, retorna un tipo que no es el de la función, en compilación 9
9. } 9
c) C 9
1. # include <stdio.h> 9
2. int suma; /* Esta es una variable global */ 9
3. int main() 9
4. { int indice; 9
5. encabezado; //Sintactico, falta los paréntesis. 9
6. for (indice = 1 ; indice <= 7 ; indice ++) 9
7. cuadrado (indice); //Semántico, no está declarado cuadrado, en compilación 9
8. final(); Llama a la función final */ // Semántico, final es una palabra reservada, en compilacion. 9
9. return 0; 9
10. } 10
11. cuadrado (numero) // Sintactico X3, falta el tipo de la función, falta el ;, y las llaves 10
12. int numero; 10
13. { int numero_cuadrado; // Sintactico Falta el nombre de la función arriba de las llaves 10
14. numero_cuadrado == numero * numero; //Semantico número no está inicializado, en compilacion 10
15. suma += numero_cuadrado; //Semántico Suma no esta inicializado, en compilacion 10
16. printf("El cuadrado de %d es %d\n", 10

17. numero, numero_cuadrado); 10
18. } 10
Ayuda: Sintácticos 2, Semánticos 6 10
d)Python 10
1. #!/usr/bin/python 11
2. print "\nDEFINICION DE NUMEROS PRIMOS" //Sintactico faltan los () 11
3. r = 1 11
4. while r = True: //Sintactico, asignación en vez de comparacion 11
5. N = input("\nDame el numero a analizar: ") //Semántico, el N se usa como numero pero es un string. En ejecución. 11
6. i = 3 11
7. fact = 0 11
8. if (N mod 2 == 0) and (N != 2): // Sintactico, mod no existe, va % 11
9. print "\nEl numero %d NO es primo\n" % N //Sintáctico faltan los () 11
10. else: 11
11. while i <= (N^0.5): //Sintactico, las potencias se definen usando el operador ** 11
12. if (N % i) == 0: 11
13. mensaje="\nEl numero ingresado NO es primo\n" % N //Semántico, no tiene espacio donde reemplazar N, ejecucion 11
14. msg = mensaje[4:6] 11
15. print msg //Sintactico faltan los () 11
16. fact = 1 11
17. i+=2 11
18. if fact == 0: 11
19. print "\nEl numero %d SI es primo\n" % N 11
20. r = input("Consultar otro número? SI (1) o NO (0)--->> ") 12
Ayuda: Sintácticos 2, Semánticos 3 12
1. def ej1 12
2. Puts 'Hola, ¿Cuál es tu nombre?' //Sintactico, va con P minuscula 12
3. nom = gets.chomp 12

4. puts 'Mi nombre es ', + nom //Sintactico, hay una coma de mas 12
5. puts 'Mi sobrenombre es 'Juan'' //Sintactico, o hay un comilla de mas o falta un + ". 12
6. puts 'Tengo 10 años' 12
7. meses = edad*12 //Semántico edad no esta declarado, en ejecución 12
8. dias = 'meses' *30 12
9. hs= 'dias * 24' 12
10. puts 'Eso es: meses + ' meses o ' + dias + ' días o ' + hs + ' horas' //Sintactico falta comilla después de meses 12
11. puts 'vos cuántos años tenés' 12
12. edad2 = gets.chomp 12
13. edad = edad + edad2.to_i 12
14. puts 'entre ambos tenemos ' + edad + ' años' 12
15. puts '¿Sabes que hay ' + name.length.to_s + ' caracteres en tu nombre, ' + name + '?' //Semántico no existe nombre, es nom, en ejecucion 12
16. end 12
Ayuda: Semánticos +4 12
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. 13
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? 13
Ejercicio 7: Determine la semántica de null y undefined para valores en javascript. ¿Qué diferencia hay entre ellos? 14
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 15
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) 16

Ejercicio 1: ¿Qué define la semántica?

Semántica: Describe el significado de las instrucciones de un programa sintácticamente correcto. Varía según lenguaje.

Ejercicio 2:

a. ¿Qué significa compilar un programa?

Compilar es el paso siguiente al chequeo sintáctico y semántico (estático), en el cual el código es pasado a código objeto y luego a código de máquina.

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

1. Análisis léxico: El compilador lee el código fuente y divide el texto en unidades léxicas significativas llamadas "tokens". Estos tokens pueden ser palabras clave, identificadores, símbolos especiales, etc.
2. Análisis sintáctico: El compilador utiliza la gramática del lenguaje de programación para analizar la estructura del programa y determinar si sigue las reglas sintácticas del lenguaje. Esto implica la construcción de un árbol de sintaxis abstracta (AST) que representa la estructura jerárquica del programa.
3. Análisis semántico: Una vez que se ha establecido la estructura del programa, el compilador verifica si las expresiones y las declaraciones tienen un significado coherente dentro del contexto del programa. Se asegura de que las variables sean declaradas antes de ser utilizadas, que los tipos de datos sean compatibles en las operaciones, entre otros aspectos.

4. Generación de código intermedio: En algunos compiladores, se genera un código intermedio que es una representación de más bajo nivel y más cercana al código de máquina que el código fuente original. Este código intermedio es más fácil de optimizar y puede ser utilizado como entrada para diferentes etapas del proceso de compilación.
5. Optimización de código: En esta etapa, el compilador realiza diversas transformaciones en el código intermedio para mejorar su eficiencia en términos de velocidad de ejecución o uso de recursos. Estas optimizaciones pueden incluir la eliminación de código muerto, la reordenación de instrucciones para minimizar el número de accesos a memoria, entre otras técnicas.
6. Generación de código objeto: Finalmente, el compilador genera el código objeto, que es el código de máquina específico para la arquitectura de la computadora de destino. Este código objeto todavía no es ejecutable por sí mismo, ya que puede depender de otras funciones o bibliotecas externas.
7. Enlazado: En el caso de lenguajes como C o C++, el compilador no solo produce un archivo ejecutable directamente, sino que genera un archivo objeto por cada archivo fuente. El enlazador (linker) se encarga de combinar estos archivos objeto junto con cualquier biblioteca externa necesaria para crear un archivo ejecutable final.

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

Interviene en el análisis semántico, que sirve para verificar si el programa tiene un significado coherente.

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.

Falso, esto no es verdadero ya que un programa compilado es pasado a código de máquina en tiempo real, aunque puede ser que tenga un paso de precompilación a un código intermedio para optimizar y chequear errores (no todos hacen esto).
Lo bueno es que un código interpretado es más rápido de desarrollar porque no hay que compilarlo, pero suelen ser más lentos en velocidad de ejecución.

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

Los errores sintacticos son aquellos relacionados con la estructura o gramatica del lenguaje. Detectados durante el analisis sintactico. Los errores gramaticos suceden sobre codigo sintacticamente correcto pero que su significado es erroneo o no su comportamiento es erratico. Detectado durante el analisis semantico y ejecucion.

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 //Error sintáctico (falta ;)
2. var 5: integer; var a:char; //Error sintáctico, nombre de variable inicia con un número.
3. Begin
4. for i:=5 to 10 do begin //Semantico, i no declarado en compilación
5. write(a); // Semántico, a no esta inicializado se detecta en compilación
6. a=a+1; //Sintattico, va := , Semántico Suma un char a un entero, en compilación
7. end;

End. Ayuda: Sintáctico 2, Semántico 3.

b) Java:

1. public String tabla(int numero, arrayList<Boolean> listado) //Sintactico, El tipo es ArrayList
2. {
3. String result = null;
4. for(i = 1; i < 11; i--) { //Semántico, i no esta declarado, en compilación
5. result += numero + "x" + i + "=" + (i*numero) + "\n"; //Semántico, result no esta inicializado. En compilación
6. listado.get(listado.size()-1)=(BOOLEAN) numero>i; //SintacticoX2, BOOLEAN no existe, lado izquierdo debe ser una variable
7. }
8. return true; //Semántico, retorna un tipo que no es el de la función, en compilación
9. }

c) C

1. # include <stdio.h>
2. int suma; /* Esta es una variable global */
3. int main()
4. { int indice;
5. encabezado; //Sintactico, falta los paréntesis.
6. for (indice = 1 ; indice <= 7 ; indice ++)
7. cuadrado (indice); //Semántico, no está declarado cuadrado, en compilación
8. final(); Llama a la función final */ // Semántico, final es una palabra reservada, en compilacion.
9. return 0;

```
10.  }
11.  cuadrado (numero) // Sintactico X3, falta el tipo de
     la función, falta el ;, y las llaves
12.  int numero;
13.  { int numero_cuadrado; // Sintactico Falta el
     nombre de la función arriba de las llaves
14.  numero_cuadrado == numero * numero;
     //Semantico número no está inicializado, en
     compilacion
15.  suma += numero_cuadrado; //Semántico Suma no
     esta inicializado, en compilacion
16.  printf("El cuadrado de %d es %d\n",
17.  numero, numero_cuadrado);
18.  }
```

Ayuda: Sintácticos 2, Semánticos 6

d)Python

```
1. #!/usr/bin/python
2. print "\nDEFINICION DE NUMEROS PRIMOS"
   //Sintactico faltan los ()
3. r = 1
4. while r = True: //Sintactico, asignación en vez de
   comparacion
5. N = input("\nDame el numero a analizar: ")
   //Semántico, el N se usa como numero pero es un
   string. En ejecución.
6. i = 3
7. fact = 0
8. if (N mod 2 == 0) and (N != 2): // Sintactico, mod no
   existe, va %
9. print "\nEl numero %d NO es primo\n" % N
   //Sintáctico faltan los ()
10.  else:
11.  while i <= (N^0.5): //Sintactico, las potencias se
   definen usando el operador **
12.  if (N % i) == 0:
13.  mensaje="\nEl numero ingresado NO es primo\n" %
   N //Semántico, no tiene espacio donde reemplazar N,
   ejecucion
14.  msg = mensaje[4:6]
15.  print msg //Sintactico faltan los ()
16.  fact = 1
17.  i+=2
18.  if fact == 0:
19.  print "\nEl numero %d SI es primo\n" % N
```

20. r = input("Consultar otro número? SI (1) o NO (0)--->> ")

Ayuda: Sintácticos 2, Semánticos 3

1. def ej1
2. Puts 'Hola, ¿Cuál es tu nombre?' //Sintactico, va con P minuscula
3. nom = gets.chomp
4. puts 'Mi nombre es ', + nom //Sintactico, hay una coma de mas
5. puts 'Mi sobrenombre es 'Juan'' //Sintactico, o hay un comilla de mas o falta un + “.
6. puts 'Tengo 10 años'
7. meses = edad*12 //Semántico edad no esta declarado, en ejecución
8. dias = 'meses' *30
9. hs= 'dias * 24'
10. puts 'Eso es: meses + ' meses o ' + dias + ' días o ' + hs + ' horas' //Sintactico falta comilla después de meses
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 + '?' //Semántico no existe nombre, es nom, en ejecucion
16. end

Ayuda: Semánticos +4

Ni idea este

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.

```
def ordenar_arreglo(arreglo, cont):
    ordenado = False
    while not ordenado:
        ordenado = True
        for i in range(cont - 1):
            if ord(arreglo[i]) > ord(arreglo[i + 1]):
                aux = arreglo[i]
                arreglo[i] = arreglo[i + 1]
                arreglo[i + 1] = aux
                ordenado = False
```

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 hace referencia al objeto receptor actual, es decir, al objeto sobre el cual se está ejecutando el código en un determinado momento. Puede cambiar de contexto dependiendo de dónde se encuentre siendo utilizado.

nil: En Ruby, nil es un objeto que representa la ausencia de un valor. Se utiliza para indicar que una variable no tiene ningún valor asignado. Es importante destacar que en Ruby, nil es un objeto de

la clase NilClass. Se comporta como un valor falso en contextos booleanos. Por ejemplo:

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

null:

- null es un valor primitivo en JavaScript que representa la ausencia intencional de cualquier valor o un valor nulo.
- Se utiliza para indicar explícitamente que una variable no contiene ningún valor o que no apunta a ningún objeto o referencia válida.
- null es de tipo object, lo que puede ser confuso, pero es técnicamente un valor primitivo.

undefined:

- undefined también es un valor primitivo en JavaScript que se asigna automáticamente a una variable cuando se declara pero no se le asigna ningún valor.
- Representa la ausencia de valor debido a que una variable no ha sido inicializada o a que una propiedad de un objeto no está definida.
- undefined es una forma de "valor por defecto" en JavaScript.

Diferencias clave:

- null es asignado por el programador para indicar la ausencia intencional de valor, mientras que undefined es asignado automáticamente por JavaScript para indicar la falta de inicialización o de definición.
- null es un valor primitivo, mientras que undefined es tanto un valor primitivo como un tipo de datos.
- null es igual a undefined en términos de comparación de valor (==), pero no es igual en términos de identidad (===). Por lo tanto, null == undefined es true, pero null === undefined es false.

- null se utiliza comúnmente para reiniciar o restablecer valores de variables, mientras que undefined indica que la variable no ha sido definida o inicializada.

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, do-while) o un switch.
Características importantes:
Puede utilizarse dentro de bucles anidados para salir del bucle más cercano al break.
No tiene un efecto fuera del bucle o el switch en el que se encuentra.
Si se usa dentro de un switch, interrumpe la ejecución del switch, saliendo del bloque switch.
No se puede usar fuera de un bucle o switch.

PHP:

Semántica: En PHP, break se usa para salir de un bucle (for, while, do-while) o un switch.
Características importantes:
Similar a C, puede salir de bucles anidados, pero se puede indicar con un valor numérica la cantidad de anidaciones a salir.

Javascript:

En JavaScript, break se utiliza para salir de un bucle (for, while, do-while) o un switch o label.
Características importantes:
Se comporta de manera similar a C y PHP, saliendo del bucle más cercano o del switch.

Puede salir de bucles anidados, pero solo afecta al bucle más cercano.
Puede saltar a una etiqueta por ejemplo.

Ruby:

En Ruby, break se utiliza para salir de un bucle (for, while, until, loop) o un case dentro de un switch.
Características importantes:
Puede salir de bucles anidados, afectando solo al bucle más cercano. Además permite que salga si tiene un condicion estilo break if (cosa).

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)

Importancia respecto a la semántica del programa:

La ligadura es crucial para la semántica de un programa, ya que establece las relaciones entre los identificadores y los valores que representan. Esto permite que el programa acceda y manipule datos de manera coherente y predecible. Sin ligadura adecuada, el programa no podría interpretarse correctamente ni ejecutarse de manera efectiva.

Diferencias entre ligadura estática y dinámica:

Ligadura estática: En la ligadura estática, la asociación entre el identificador y su valor se determina en tiempo antes de la ejecucion y permanece constante durante la ejecución del programa. Esto significa que el valor al que se refiere un identificador se establece antes de que el programa se ejecute y no puede cambiarse en tiempo de ejecución.

Ejemplo: En un lenguaje como C, cuando declaramos una variable global int x = 5;, la ligadura estática vincula el nombre x con el valor 5 en tiempo de compilación. Esto significa que x siempre se referirá a 5 durante la ejecución del programa, a menos que se redefina en tiempo de compilación.

Ligadura dinámica: En la ligadura dinámica, la asociación entre el identificador y su valor se determina en tiempo de ejecución y puede cambiar durante la ejecución del programa. Esto permite una mayor flexibilidad y adaptabilidad en el comportamiento del programa.

Ejemplo: En lenguajes con asignación dinámica de memoria como Python, cuando asignamos un valor a una variable x = 5, la ligadura dinámica vincula el nombre x con el valor 5 en tiempo de ejecución. Más tarde, podemos cambiar el valor de x a otro valor, como x = 10, y la ligadura dinámica se ajustará para reflejar este cambio.