Conceptos y paradigmas de lenguajes de programación – Practica 3

1) ¿Qué define la semántica?

La semántica describe el significado de los símbolos, palabras y frases de un lenguaje ya sea natural o lenguaje informático. Se reconocen 2 tipos de semántica: estática y dinámica.

* Estática: No está relacionado con el significado del programa, está relacionado con las formas válidas. Se las llama así porque el análisis para el chequeo puede hacerse en compilación.
* Dinámicas: Es la que describe el efecto de ejecutar las diferentes construcciones en el lenguaje de programación. Su efecto se describe durante la ejecución del programa. Los programas solo se pueden ejecutar si son correctos para la sintaxis y para la semántica estática.

2)a) ¿Que significa compilar un programa?

Compilar un programa significa realizar una traducción del código fuente (escrito en lenguaje de programación de alto nivel) a lenguaje de maquina (código objeto) para que pueda ser ejecutado por la computadora. La herramienta encargada de la traducción es el compilador.

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

Primero está el *Análisis* del programa fuente, lo cual consta de las siguientes etapas:

* Análisis léxico (Scanner): Es el que lleva mas tiempo. Hace el análisis a nivel de palabra y divide el programa en sus elementos consecutivos: palabras clave, delimitadores, números, etc.  
  Filtra comentarios y espacios. Genera errores si la entrada no coincide con ninguna categoría léxica.
* Análisis sintáctico (Parser): Este análisis se realiza a nivel de sentencia. Se identifican las estructuras, sentencias, declaraciones, etc. Este analizador se alterna con el análisis semántico y aplica una gramática para construir el árbol sintáctico del programa.
* Análisis semántico (Semántica estática): Es la fase mas importante. Las estructuras sintácticas reconocidas por el analizador sintáctico son procesadas y la estructura del código ejecutable toma forma. Se realizan comprobación de tipos. Se agrega información implícita (variables no declaradas). Se hacen comprobaciones de nombres (por ejemplo, toda variable debe estar declarada).

La otra etapa es la de *Síntesis*, está compuesta por:

* Generación del código intermedio: Las características de esta representación es que este debe ser fácil de producir y debe ser fácil de traducir al lenguaje objeto.
* Optimización del código: Este proceso es optativo.
* Generación del código (máquina): Es la traducción del código intermedio a lenguaje de máquina.

2)c) ¿En qué paso interviene la semántica y cuál es su importancia dentro de la compilación?  
  
La semántica interviene en el Análisis Semántico. Es la fase más importante dado que las estructuras sintácticas reconocidas por el analizador sintáctico son procesadas y la estructura del código ejecutable toma forma. También se realizan comprobación de tipos, se agrega información implícita (variables no declaradas), se hacen comprobaciones de nombres (por ejemplo, toda variable debe estar declarada), y es el nexo entre el análisis y la síntesis.

3) Con respecto al punto anterior ¿es lo mismo que interpretarlo? Justifique su respuesta

No es lo mismo interpretar un programa que compilarlo, ya que el intérprete solo realiza la traducción a medida que sea necesario, típicamente instrucción por instrucción y normalmente no guarda el resultado de dicha traducción.

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

Un error sintáctico se produce cuando se escribe código de una forma no admitida por las reglas del lenguaje. Los errores de sintaxis son detectados casi siempre por el compilador o intérprete, que muestra un mensaje de error que informa del problema. Ejemplo: falta de un “;” al final de una sentencia que lo requiere.  
Por otro lado, los errores semánticos son más sutiles. Un error semántico se produce cuando la sintaxis del código es correcta, pero el significado no es el que se pretendía. La construcción obedece las reglas del lenguaje, y por ello el compilador o intérprete no detectan los errores semánticos. Los compiladores e intérpretes sólo se ocupan de la estructura del código que se escribe, y no de su significado. Ejemplo: imprimir una variable que no tiene valor asignado.

Anexo: Error lógico:  
Un programa puede continuar en ejecución después de haberse producido errores semánticos, pero su estado interno puede ser distinto del esperado. Puede que las variables no contengan los datos correctos, o bien es posible que el programa siga un camino distinto del pretendido. Eventualmente, la consecuencia será un resultado incorrecto. Estos errores se denominan lógicos, ya que, aunque el programa no se bloquea, la lógica que representan contiene un error.

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.

<?php

function ordenarArreglo(array $arreglo, $cont) {

do {

$ordenado = true;

for ($i=1; $i <= $cont - 1; $i++) {

if ($arreglo[$i] > $arreglo[$i+1]) {

$aux = $arreglo[$i];

$arreglo[$i] = $arreglo[$i+1];

$arreglo[$i+1] = $arreglo[$i];

$ordenado = false;

}

}

} while ($ordenado);

return $arreglo;

}

?>

* La sintaxis de un arreglo …

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?

Dado que todo en Ruby es un objeto, cada pieza de código que se escribe "pertenece" a algún objeto. “Self” es una variable especial que apunta al objeto que "posee" el código que se está ejecutando actualmente, o en otras palabras, el objeto que se está utilizando en ese instante.  
Por otro lado “Nil” es una clase en Ruby. Se implementa con la finalidad de representar literalmente nada. La utilidad que brinda principalmente es la de desasignar una variable y que no referencie a ningún otro objeto.

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

“null” es un literal de JavaScript que representa un valor nulo o "vacío", es decir, una variable nula no tiene valor. Es uno de los valores primitivos de JavaScript.  
Por otro lado, “undefined” significa que una variable ha sido declarada, pero no definida. La principal diferencia entre estas dos palabras en JavaScript, es que “null” es un objeto, y “undefined” es un tipo en si mismo.

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

En C: La instrucción de salto break se usa para interrumpir (romper) la ejecución normal de un bucle, es decir, la instrucción break finaliza (termina) la ejecución de un bucle y, por tanto, el control del programa se transfiere (salta) a la primera instrucción después del bucle.  
  
En PHP: finaliza la ejecución de la estructura for, foreach, while, do-while o switch en curso.  
  
En JavaScript: Termina el bucle actual, sentecia switch o label y transfiere el control del programa a la sentencia posterior al bucle en cuestión.

En Ruby: Idem

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 tratan con entidades, como variables, rutinas, declaraciones, etc.

Las entidades del programa tienen ciertas propiedades llamadas atributos. Por ejemplo, una variable tiene un nombre, un tipo, un área de almacenamiento donde se almacena su valor; una rutina tiene un nombre, parámetros formales de cierto tipo, ciertas convenciones de paso de parámetros; Una declaración tiene acciones asociadas. Los atributos deben especificarse antes de que se elabore una entidad. El concepto de ligadura se refiere a la especificación de la naturaleza exacta de un atributo. Para cada entidad, la información de atributos está contenida en un repositorio llamado descriptor.  
Su importancia respecto a la semántica de un programa es que es un concepto central en la definición de esta ya que los lenguajes de programación difieren en el número de entidades con las que pueden tratar, en el número de atributos que deben vincularse a las entidades, en el momento en que se producen tales vinculaciones (tiempo de vinculación) y en la estabilidad de la vinculación (es decir, si una La vinculación establecida es fija o modificable).

La diferencia principal entre la ligadura estática y la dinámica es que la primera no se puede modificar y la segunda es modificable.  
Ejemplo: en programación orientada a objetos la ligadura dinámica se utiliza cuando múltiples clases, en una jerarquía de clases, contienen diferentes implementaciones del mismo método.