Práctica 3: Semántica

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.

Ejercicio 2: a. ¿Qué significa compilar un programa?

Compilar un programa significa traducir el código fuente escrito en un lenguaje de alto nivel (como C++, Java o Python) a un lenguaje que la computadora pueda entender, generalmente lenguaje de máquina.

Este proceso es realizado por un compilador, que analiza todo el código fuente antes de ejecutarlo y genera un archivo ejecutable o un código intermedio (como ensamblador).

Una vez compilado, el programa se puede ejecutar sin necesidad de volver a traducirlo. Esto lo diferencia de los lenguajes interpretados, donde el código fuente se traduce línea por línea en tiempo de ejecución.

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

1- Etapa de Análisis (Vinculadas al código fuente):

- Análisis léxico (Programa Scanner): Es un proceso que lleva tiempo, se encarga de hacer el análisis a nivel de palabra (LEXEMA) y divide el programa en sus elementos/categorías: identificadores, delimitadores, símbolos especiales, operadores, números, palabras clave, palabras reservadas, comentarios, etc. Además, analiza el tipo de cada uno para ver si son TOKENS válidos y filtra comentarios y separadores (como: espacios en blanco, tabulaciones, etc.)
  Lleva una tabla para la especificación del analizador léxico que incluye cada categoría, el conjunto de atributos y acciones asociadas, además, pone los identificadores en la tabla de símbolos y reemplaza cada símbolo por su entrada en la tabla.
  Genera errores si la entrada no coincide con ninguna categoría léxica y el resultado de este paso será el descubrimiento de los ítems léxicos o tokens y la detección de errores.
- Análisis sintáctico (Programa Parser): El análisis se realiza a nivel de sentencia/estructuras. Tiene como objetivo identificar las estructuras de las sentencias, declaraciones, expresiones, etc. presentes en su entrada usando los tokens del analizador léxico.
  Estas estructuras se pueden representar mediante el árbol de análisis sintáctico o árbol de derivación, que explica cómo se puede derivar la cadena de entrada en la gramática que especifica el lenguaje, validando qué pertenece o no a la gramática en pos de ver que lo que entra es correcto.
  El analizador sintáctico (Programa Parser) se alterna/interactúa con el análisis léxico y análisis semántico. Y usualmente usan técnicas de gramáticas formales.
- Análisis semántico (Programa de Semántica estática): Para realizar este análisis primero se debe pasar correctamente el Scanner y el Parser.
  Esta etapa se considera como una fase medular, es decir, una de las más importantes, se encarga de procesar las estructuras sintácticas, realizar comprobación de tipos (aplica gramática de atributos), agregar a la tabla de símbolos los descriptores de tipos, realizar comprobaciones de duplicados, problemas de tipos, comprobaciones de nombres, etc.
  Además, agrega información implícita y la estructura del código ejecutable continúa tomando forma.
  Esta etapa es el nexo entre etapas inicial y final del compilador (Análisis y Síntesis).

Entre 1- y 2- ocurre *Generación de código intermedio:* Es 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. El código intermedio más habitual es el código de 3-direcciones.

2- Etapa de Síntesis (Vinculadas a características del código objeto, del hardware y la arquitectura):

- Optimización del código: No se hace siempre y no lo hacen todos los compiladores. Estos programas pueden ser herramientas independientes, o estar incluidas en los compiladores e invocarse por medio de opciones de compilación.
  Existen diversas formas y cosas que se pueden optimizar como por ejemplo elegir entre velocidad de ejecución y tamaño del código ejecutable, generar código para un microprocesador específico dentro de una familia de procesadores, eliminar la comprobación de rangos o desbordamientos de pila, evaluación para expresiones booleanas, eliminación de código muerto o no utilizado, eliminación de funciones no utilizadas, etc.
- Generación del código final: El compilador traduce el código fuente y el AST (o alguna representación interna) a código de máquina o código intermedio que la computadora pueda ejecutar.

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

La semántica aparece en la etapa de análisis semántico (semántica estática) y es una de las partes más importantes ya que es la previa a la generación de código intermedio. Encuentra los últimos errores que pueden haber arrastrado el análisis léxico y sintáctico. Se enfoca en encontrar el significado de las sentencias.

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.

Diferencias	Intérprete	Compilador
Definición	Programa que ejecuta el código fuente línea por línea sin generar un archivo independiente.	Programa que traduce todo el código fuente a lenguaje de máquina antes de ejecutarlo, creando un archivo ejecutable.
Por cómo se ejecuta	- Se utiliza el intérprete en la ejecución. Ejecuta el programa línea por línea. - Por donde pase dependerá de la acción del usuario, de la entrada de datos y/o de alguna decisión del programa. - El programa fuente será público.	- Se utiliza el compilador antes de la ejecución. - Produce un programa ejecutable equivalente en lenguaje objeto - El programa fuente no será publico
Por el orden de ejecución	Sigue el orden lógico de ejecución (no necesariamente recorre todo el código)	Sigue el orden físico de las sentencias (recorre todo)
Por el tiempo consumido de ejecución	- Por cada sentencia que pasa realiza el proceso de decodificación (lee, analiza y ejecuta) para determinar las operaciones y sus operandos. Es repetitivo. - Si la sentencia está en un proceso iterativo, la tarea de decodificación se realiza tantas veces como sea requerido. La velocidad del proceso se ve afectada por esto mismo.	- Pasa por todas las sentencias. - No repite lazos. - Traduce todo de una sola vez. - Genera código objeto ya compilado. - La velocidad de compilación dependerá del tamaño del código.
Por la eficiencia posterior	- Más lento en ejecución. Se repite el proceso cada vez que se	- Más rápido ejecutar desde el punto de vista del hardware porque ya está en

	ejecuta el mismo programa o pasa por las mismas instrucciones. - Para ser ejecutado en otra máquina se necesita tener si o si el intérprete instalado y el programa fuente será público.	un lenguaje de más bajo nivel. - Detectó más errores al pasar por todas las sentencias. - Siempre está listo para ser ejecutado. - Capaz se tardó más en compilar por las verificaciones previas. El programa fuente no será público.
Por el espacio ocupado	- No pasa por todas las sentencias entonces ocupa menos espacio de memoria. - Cada sentencia se deja en la forma original y las instrucciones interpretadas necesarias para ejecutarlas se almacenan en los subprogramas del interprete en memoria. - Tablas de símbolos, variables y otros se generan cuando se usan en forma dinámica.	- Si pasa por todas las sentencias. - Una sentencia puede ocupar decenas o centenas de sentencias de máquina al pasar a código objeto. - Cosas cómo tablas de símbolos, variables, etc. se generan siempre se usen o no. -El compilador en general ocupa más espacio.
Por la detección de errores	- Las sentencias del código fuente pueden ser relacionadas directamente con la sentencia en ejecución entonces se puede ubicar donde se produjo el error. - Es más fácil detectarlos por donde pasa la ejecución. - Es más fácil corregirlos.	- Se pierde la referencia entre el código fuente y el código objeto. - Es casi imposible ubicar el error, pobres en significado para el programador. - Se deben usar otras técnicas (Semántica Dinámica).
Ventajas	Facilidad de desarrollo y depuración, flexibilidad para realizar cambios en el código sin necesidad de recompilación.	Mayor eficiencia en tiempo de ejecución, mayor capacidad de optimización, independencia del código fuente una vez compilado.
Desventajas	Menor eficiencia en tiempo de ejecución, dependencia continua del código fuente durante la ejecución, falta de optimizaciones globales.	Requiere un paso adicional de compilación antes de la ejecución, lo que puede aumentar el tiempo de desarrollo inicial y el tamaño del archivo ejecutable.

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

Error Sintáctico: Un error sintáctico ocurre cuando el código no sigue las reglas gramaticales del lenguaje de programación. Estos errores son detectados por el analizador sintáctico (parser) del lenguaje durante la fase de compilación o interpretación. Ejemplo en Pascal:

```
begin
  writeln("Hola mundo")  // Error: comillas incorrectas, deben ser simples (' ')
end.
```

Error: Uso incorrecto de comillas.

Error Semántico: Un error semántico ocurre cuando el código está bien formado desde el punto de vista sintáctico, pero no produce los resultados esperados debido a una interpretación incorrecta del significado del código. Pueden manifestarse como comportamientos inesperados o resultados incorrectos durante la ejecución del programa. Ejemplo en Pascal:

```
var x: integer;
begin
  x := 'Hola';  // Error: se intenta asignar un string a un entero
  writeln(x);
end.
```

Error: Incompatibilidad de tipos (semántica estática: se detecta antes de ejecutar el programa).

```
def recursiva()
        return recursiva()
recursiva()
```

Error: Stackoverflow en recursión infinita (semántica estática: se detecta en ejecución del programa).

La diferencia clave entre un error sintáctico y uno semántico radica en que los errores sintácticos se refieren a problemas con la estructura y la gramática del código, mientras que los errores semánticos se refieren a problemas con el significado o la lógica del código.

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 (1)
var 5: integer; (2)
var a:char;
Begin
        for i:=5 to 10 do begin (3)
                write(a); (4)
                a=a+1; (5)
        end;
End.
```

(1) Error Sintáctico: Falta el ";". Es detectado en compilación.
(2) Error Sintáctico: Una variable no puede comenzar con un número. Es detectado en compilación.
(3) Error Semántico: La variable "i" no está declarada. Es detectado en compilación.
(4) Error Semántico: La variable "a" no está inicializada. Es detectado en ejecución.
(5) Error Sintáctico: La asignación en Pascal es realizada con el símbolo ":=". Es detectado en compilación.
(5) Error Semántico: La variable "a" es de tipo char, y se le intenta sumar uno a la misma, realizando una operación inválida. Es detectado en compilación.

Ayuda: *Sintáctico 2, Semántico 3*

b) Java:

```
public String tabla (int numero, arrayList<Boolean> listado) { (1)
        String result = null;
        for(i = 1; i < 11; i--) { (2)
```

```
                result += numero + "x" + i + "=" + (i*numero) + "\n"; (3)
                listado.get(listado.size()-1)=(BOOLEAN) numero>i; (4)
        }
        return true; (5)
}
```

(1) Error Sintáctico: Debería ser ArrayList<Boolean> con A mayúscula. Es detectado en compilación.
(2) Error Sintáctico: La declaración es correcta pero la variable "i" no tiene definido un tipo. Detectado en compilación.
(2) Error Lógico: Bucle infinito. La variable i al decrementar su valor nunca podrá ser más grande que un valor que ya era mayor previamente.
(3) Error Semántico: Concatenar cadenas con null genera una excepción. Detectada en ejecución.
(4) Error Sintáctico: Uso incorrecto de paréntesis en la asignación. Es detectado en compilación.
(4) Error Semántico: El tipo BOOLEAN no existe, pero el lenguaje no establece reglas para esto. Detectado en compilación.
(4) Error Sintáctico: La asignación es inválida. Es detectado en compilación.
(4) Error Lógico: Si la lista está vacía se genera un error. Detectado en ejecución.
(5) Error Semántico: Intenta devolver true en un método que retorna String.

Ayuda: *Sintácticos 4, Semánticos 3, Lógico 1*

c) C

```
# include <stdio.h>
int suma; /* Esta es una variable global */
int main() {
        int indice;
        encabezado; (1)
        for (indice = 1 ; indice <= 7 ; indice ++) (2)
        cuadrado (indice); (3)
        final(); Llama a la función final */ (4)
        return 0;
}
cuadrado (numero) (5)
int numero;
{       int numero_cuadrado;
        numero_cuadrado == numero * numero; (6)
        suma += numero_cuadrado; (7)
        printf("El cuadrado de %d es %d\n",
        numero, numero_cuadrado);
}
```

(1) Error Sintáctico: Llamada a "encabezado" sin definición previa. Es detectado en compilación.
(2) Error Sintáctico: No se usan llaves para el for. Es detectado en compilación.
(2) Error Sintáctico: Espacio innecesario en índice++.
(3) Error Semántico: La declaración de "cuadrado" no existe en este contexto. Es detectado en compilación.
(4) Error Semántico: Llamada a "final" sin declaración final. Es detectado en compilación.
(4) Error Sintáctico: Falta la apertura del comentario. Es detectado en compilación.
(5) Error Sintáctico: El parámetro "numero" no tiene definido el tipo. Es detectado en compilación.
(5) Error Sintáctico: En ningún momento se abren llaves.

(6) Error Semántico: La variable “numero” no está inicializada. Es detectado en compilación.
(6) Error Sintáctico: Uso de == en lugar de = para asignación. Es detectado en compilación.
(7) Error Semántico: La variable “suma” no está inicializada explícitamente. Es detectado en compilación.

Ayuda: *Sintácticos 2, Semánticos 6*

d) Python

```
#!/usr/bin/python
print "\nDEFINICION DE NUMEROS PRIMOS" (1)
r = 1
while r = True: (2)
        N = input("\nDame el numero a analizar: ") (3)
        i = 3
        fact = 0
        if (N mod 2 == 0) and (N != 2): (4)
                print "\nEl numero %d NO es primo\n" % N (1)
        else:
                while i <= (N^0.5): (5)
                        if (N % i) == 0:
                                mensaje="\nEl numero ingresado NO es primo\n" % N (6)
                                msg = mensaje[4:6] (6)
                                print msg (1)
                                fact = 1
                        i+=2
                if fact == 0:
                        print "\nEl numero %d SI es primo\n" % N (1)
        r = input("Consultar otro número? SI (1) o NO (0)--->> ")
```

(1) Error Sintáctico: Falta el uso de paréntesis “()” para el print. Es detectado en compilación.
(2) Error Sintáctico: La comparación se realiza con el operador “==”, y no con el de asignación (“=”). Es detectado en compilación.
(3) Error Semántico: input siempre devuelve un String. Se debe convertir la entrada a un tipo número int. Es detectado en ejecución.
(4) Error Sintáctico: No existe el operador MOD, se debe usar “%” para la función de resto. Es detectado en compilación.
(5) Error Sintáctico: Para obtener la raíz cuadrada, se debería usar “**” para exponentes o la función math.sqrt(). Es detectado en compilación.
(6) Error Semántico: El operador % ya está reemplazando % N con el valor de N, por lo que el resultado de la operación es una cadena completa. Hace falta el uso de “%d”. Detectado en ejecución. Podría tomarse también como error lógico la línea siguiente.

Ayuda: *Sintácticos 2, Semánticos 3*

e) Ruby

```
def ej1
        Puts 'Hola, ¿Cuál es tu nombre?' (1)
        nom = gets.chomp
```

```
        puts 'Mi nombre es ', + nom (2)
        puts 'Mi sobrenombre es 'Juan'' (3)
        puts 'Tengo 10 años'
        meses = edad*12 (4)
        dias = 'meses' *30 (5)
        hs= 'dias * 24'
        puts 'Eso es: meses + ' meses o ' + dias + ' días o ' + hs + ' horas' (6)
        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 + '?' (7)
end
```

(1) Error Sintáctico: Es “puts”. Ruby es case sensitive. Es detectado en compilación.
(2) Error Sintáctico: El operador + no puede estar allí en ese contexto, ya que está mal colocado. Para concatenar dos o más cadenas, debemos quitar la “,” que está colocada en esa línea. Es detectado en compilación.
(3) Error Sintáctico: Error en el uso de las comillas. Hay comillas simples dentro de comillas simples sin escaparlas correctamente. Es detectado en compilación.
(4) Error Semántico: La variable “edad” no ha sido definida antes de ser usada. Se detecta en ejecución, ya que Ruby es un lenguaje interpretado.
(5) Error Lógico: No tiene sentido multiplicar una cadena de texto por 30.
(6) Error Semántico y Sintáctico: No se terminan concatenando los Strings ya que el operador “+” está dentro de las comillas simples. Es detectado en ejecución.
(7) Error Semántico: La variable “name” no está definida en el código. Es detectado en tiempo de ejecución.

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;
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.

```
function ordenarArreglo(arreglo, cont) {
  let ordenado;
  let aux;
  do {
    ordenado = true;
    for (let i = 0; i < cont - 1; i++) {
        if (arreglo[i].charCodeAt(0) > arreglo[i + 1].charCodeAt(0)) {
            aux = arreglo[i];
            arreglo[i] = arreglo[i + 1];
            arreglo[i + 1] = aux;
            ordenado = false;
        }
    }
  } while (!ordenado);
}
```

Sobre los arreglos:

- En JavaScript, los arreglos se definen utilizando corchetes [], como let arreglo = [].
- Se puede acceder a los elementos del arreglo utilizando índices, como arreglo[i], comenzando desde el índice 0.

Estructuras de control:

- La estructura for en JavaScript requiere una declaración de la variable, una condición y un incremento o decremento. Sintaxis: for (inicialización; condición; incremento/decremento)
- Sintaxis: El bucle do...while ejecuta el bloque de código al menos una vez, y luego evalúa la condición al final. do { // Código } while (condición);

Comparación de caracteres:

- En JavaScript, no existe el tipo char, por lo que se trabaja con cadenas de texto (strings). Utilizamos el método charCodeAt(0) para obtener el valor numérico de un carácter.

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 especial que hace referencia al objeto actual.

- Dentro de un método de instancia (dentro de una clase), self hace referencia a la instancia del objeto.
- En un método de clase o dentro de una clase misma, self hace referencia a la clase.
- En el contexto de un módulo, self hace referencia al módulo.
- Fuera de cualquier clase o módulo, self hace referencia al objeto principal.

Es utilizado para acceder a los métodos y propiedades del objeto o clase actual. También se utiliza en la definición de métodos de clase y de instancia. Al iniciar el intérprete, “self” tiene el valor main, ya que este es el primer objeto que se crea.

“nil”: Es un objeto especial que representa la ausencia de valor o una referencia vacía. Es un objeto singleton de la clase NilClass. Se utiliza para indicar valores no asignados, ausencia de datos o como valor de retorno por defecto en muchos casos. En las evaluaciones booleanas, nil se comporta como false.

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

Null: Si una variable es null, significa que la variable no tiene valor y que el programador la estableció explícitamente para que no tuviera valor. Una variable nunca será null a menos que en algún lugar del código un programador establezca una variable en null.

Undefined: Significa que no hay ningún valor porque todavía no se ha establecido ningún valor. Por ejemplo, si creas una variable y no le asignas un valor, entonces estará undefined. También se puede establecer a una variable con este valor, una razón por la cual querrías hacer esto es esencialmente para restablecer o resetear una variable.

Es importante destacar que undefined es un valor primitivo y no un objeto, a diferencia de null, que es un objeto primitivo.

Si una variable se establece en null o undefined, no tiene valor y si una función devuelve null o undefined, entonces nos está diciendo que no tiene ningún valor para devolver. La diferencia principal entre null y undefined es que null se utiliza para indicar explícitamente la ausencia de valor, mientras que undefined se utiliza para representar la ausencia de valor cuando una variable aún no ha sido asignada o cuando se accede a una propiedad que no existe en un objeto.

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

Lenguaje	Propósito	Características importantes
C	Salir de un ciclo o de un `switch`	- Interrumpe el ciclo o `switch` más cercano.
PHP	Salir de un ciclo o de un `switch`	- Acepta un argumento opcional para interrumpir múltiples niveles de ciclos anidados.
JavaScript	Salir de un ciclo o de un `switch`	- Similar a C, permite usar un número para interrumpir múltiples ciclos anidados.
Ruby	Salir de un ciclo (y puede devolver un valor)	- Puede devolver un valor al salir del ciclo.

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)

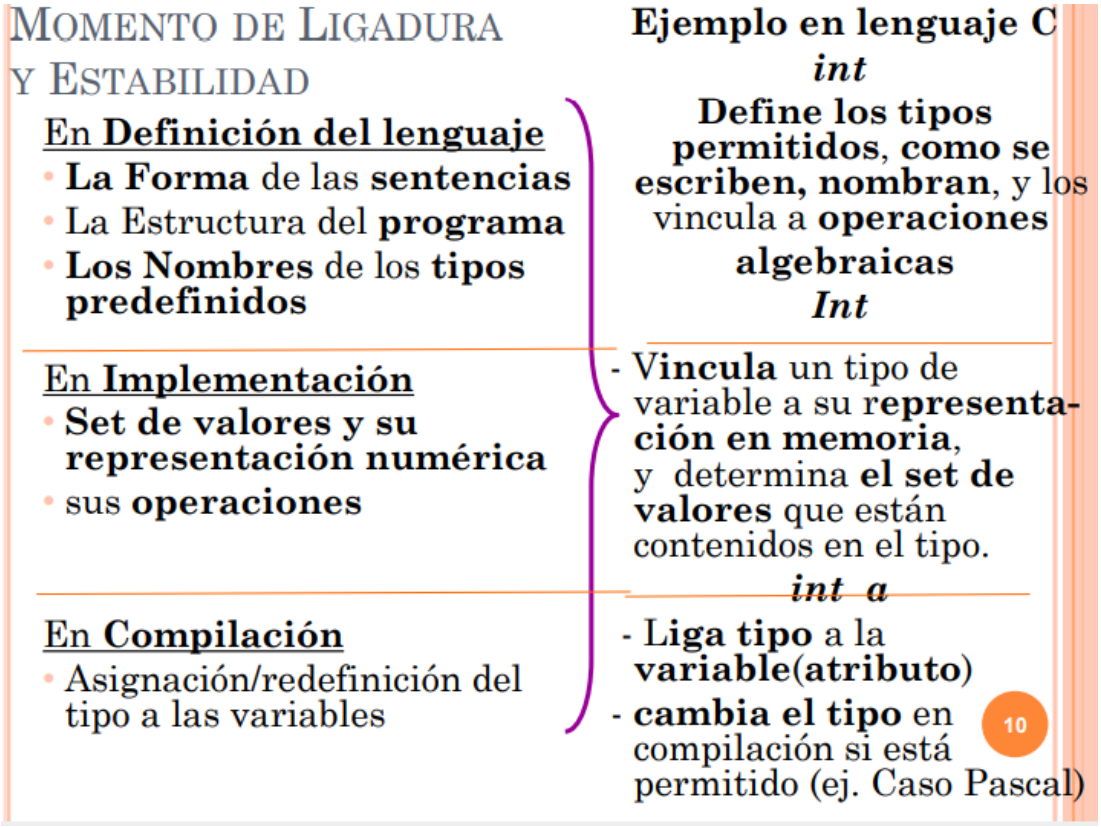
Es el momento en el que a un atributo de una entidad se le asocia un valor determinado. La importancia de la ligadura en la semántica de un programa radica en que permite al compilador o intérprete conocer el tipo y la ubicación de las variables y funciones, lo que es esencial para el correcto funcionamiento del programa.

Ligadura Estática: Se establece antes de la ejecución del programa, ya sea en la definición del lenguaje, la implementación o en la compilación. En cuanto a su estabilidad, este tipo de ligadura no se puede modificar.

Ligadura Dinámica: Se establece durante la ejecución. En cuanto a su estabilidad, se puede modificar durante la ejecución del programa de acuerdo a alguna regla específica del lenguaje. Existe una excepción en cuanto

a la estabilidad con las constantes, su ligadura se produce en tiempo de ejecución, pero esta no puede ser modificada luego de ser establecida.

Estático:



Dinámico: