**Práctica 3**  
**CPLP**

1. 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
2. 1. Convertir un programa en lenguaje máquina a partir de otro programa de computadora escrito en otro lenguaje.

**Escribir el código fuente**:  
Primero, debes escribir el programa en un editor de texto o un entorno de desarrollo integrado (IDE). Este código se guarda en un archivo con la extensión correspondiente al lenguaje (por ejemplo, .c para C, .cpp para C++, .java para Java).

**Preprocesamiento (si aplica)**:  
En lenguajes como C o C++, el compilador ejecuta un preprocesador que prepara el código. Esto incluye resolver directivas como #include (para importar bibliotecas) o #define (para definir constantes o macros). El resultado es un código intermedio que aún no es ejecutable.

**Compilación**:  
El compilador traduce el código fuente (o el código preprocesado) a un lenguaje de bajo nivel, generalmente código de máquina o un formato intermedio (como bytecode en Java). Este paso verifica la sintaxis y genera un archivo objeto (por ejemplo, .o o .obj en C/C++). Si hay errores de sintaxis o semánticos, el compilador los reportará aquí y deberás corregirlos antes de continuar.

**Enlazado (linking)**:  
Una vez que tienes los archivos objeto, el enlazador (linker) los combina con las bibliotecas necesarias (como funciones estándar o externas) para crear un archivo ejecutable final (por ejemplo, .exe en Windows o un archivo binario en Linux). Este paso resuelve referencias a funciones o variables definidas en otros archivos o bibliotecas.

**Ejecución**:  
Con el archivo ejecutable generado, puedes correr el programa en tu sistema operativo utilizando el comando adecuado (por ejemplo, ./nombrePrograma en Linux o haciendo doble clic en el .exe en Windows).

* 1. La semantica interviene en el paso 3, ya que ahi es donde se checkean los errores semanticos o sintácticos que pueda llegar a tener el programa.



No, a diferencia de la compilacion, cuando se interpreta un programa lo que se hace es mediante un interprete, ejecutar el codigo fuente linea por linea sin la necesidad de compilarlo previamente

Ventajas y desventajas Interpretacion

* Ventajas:
  + Desarrollo y depuración de programas más rápidos y sencillos
  + Ejecución en múltiples plataformas sin necesidad de recompilación
  + Experiencia de desarrollo interactiva
* Desventajas:
  + Ejecución más lenta que los programas compilados
  + Menos eficientes en términos de velocidad de ejecución

Compilacion

Ventajas

1. Mayor velocidad de ejecución:  
   Como el código ya está traducido a lenguaje de máquina, el programa se ejecuta más rápido porque no hay traducción en tiempo real.
2. Optimización:  
   El compilador puede analizar el código y aplicar optimizaciones (como eliminar código muerto o mejorar bucles), lo que mejora el rendimiento.
3. Detección temprana de errores:  
   Los errores de sintaxis y algunos errores semánticos se identifican durante la compilación, antes de que el programa se ejecute.
4. Independencia del intérprete:  
   Una vez compilado, el programa no necesita un intérprete o entorno adicional para ejecutarse (solo el sistema operativo o máquina virtual, en algunos casos como Java).
5. Protección del código fuente:  
   El usuario final recibe un archivo ejecutable, no el código fuente, lo que dificulta la modificación o copia del código original.

#### Desventajas:

1. Tiempo inicial de compilación:  
   Compilar un programa puede tomar tiempo, especialmente en proyectos grandes, lo que retrasa las pruebas rápidas.
2. Falta de flexibilidad:  
   Cualquier cambio en el código requiere recompilar todo el programa, lo que puede ser inconveniente durante el desarrollo.
3. Dependencia de la plataforma:  
   El código compilado suele ser específico para un sistema operativo o arquitectura (a menos que se use un compilador cruzado), lo que puede limitar la portabilidad.
4. Depuración más compleja:  
   Los errores solo se detectan al compilar o al ejecutar el programa, y rastrear problemas puede ser más difícil sin un entorno interactivo.

### 

### **Error sintáctico**

Un error sintáctico ocurre cuando el código no sigue las reglas gramaticales o de estructura del lenguaje de programación. Es como escribir una oración con palabras mal ordenadas o faltantes en un idioma humano.

### **Error semántico**

Un error semántico ocurre cuando el código es sintácticamente correcto (sigue las reglas del lenguaje), pero no tiene sentido lógico o no hace lo que el programador intende. Es como escribir una oración gramaticalmente correcta pero con un significado absurdo o incorrecto.

1. 1. 1. Usa un número como identificador, en Pascal esto no es posible. (Sintactico, se detecta en compilacion)
      2. Dentro del for, utiliza = como operador de asignacion, en pascal el operador de asignacion es “:=”. (sintactico, se detecta en compilacion)
      3. No define i, como la variable índice del for, si no que define a 5. (Semantico, se detecta en compilacion)
      4. Define a “a” del tipo char, pero luego le asigna valores del tipo integer. (Semantico, se detecta en compilacion)
      5. En el primer ciclo del for, hace un write de a, la cual no contiene nada. (Semantico, se detecta en ejecución)
   2. 1. En los parametros del metodo, define listado del tipo arrayList cuando deberia ser ArrayList. (Semantico, se detecta en compilacion)
      2. Define un for loop, el cual nunca termina. (Logico, se detecta en ejecución)
      3. En el for, no declara el tipo de la variable i. (Sintactico, se detecta en ejecución)
      4. Intenta hacer un casting al tipo BOOLEAN, cuando java lo define como boolean (Semantico, se detecta en compilacion)
      5. Hace un .size, sin considerar que la lista este vacia(Logico, en ejecucion)
      6. El tipo de retorno del metodo esta definido como String, pero retorna un boolean. (Semantico, se detecta en compilacion)
      7. Dentro del for, intenta concatenar un string el cual expresa el resultado de multiplicar numero por i, pero debido a que i fue definido en 1 y se decrementa por cada ciclo, la multiplicacion va a dar erronea (Logico)
      8. Intenta asignar usango .get() (Sintactico, se detecta en compilacion)
      9. Inicializa result en null, pero luego le concatena algo. (Error Semantico, se detecta en ejecución)
   3. 1. En el metodo cuadrado(), intenta asignar usando el ==. (Sintactico, se detecta en compilacion)
      2. La funcion cuadrado, no es ni una definicion(ya que no tiene la llave siguiente) ni una invocacion(ya que no tiene punto y coma);
      3. El parametro de cuadrado no tiene tipo (sintactico, se detecta en compilacion)
      4. Llama a una funcion final(), la cual no esta definida. (Semantico, se detecta en compilacion)
      5. Llama a encabezado, el cual nunca fue definido ni como variable ni como funcion. (Semantico, se detecta en compilacion)
      6. Intenta llamar a una funcion (Encabezado) sin los dos paréntesis que indican que es una funcion (Sintactico, se detecta en compilacion)
      7. La variable global suma no se declara con valor y luego es utilizada. (Semantico, se detecta en Compilacion)
   4. 1. Usa = en el While r = true, el cual es el operador de asignacion (Sintactico, en ejecucion)
      2. Usa MOD, para calcular el resto de una division, pero en python eso se hace usando %. (Sintactico, en ejecución)
      3. Usa Print sin parentesis(Sintactico, en ejecucion)
      4. Hace mal los checkeos, ya que si se ingresa el numero uno, este seria tomado como primo. (Logico)
      5. Intenta hacer potencia con el ^ cuando el operador de potencia es el \*\*(Sintactico, se checkea previo a la ejecucion)
      6. mensaje crea una cadena, pero al imprmir msg solo se imprime una parte la cual no tiene sentido (Logico)
   5. 1. Intenta hacer edad \* 12 pero edad nunca fue definida (Semantico)
      2. Escribe Puts en vez de puts, ruby es case sensitive (Sintactico)
      3. Usa , y + juntos (error sintactico)
      4. Hace días = meses \* 30 cuando no todos los meses tiene 30 días, por lo que la salida de esto sera erronea (Logico)
      5. Hace “meses” \* 30, lo cual generara repeticion de texto y no la multiplicación que se espera (Logico)
      6. En hs guarda una cadena ‘días \* 24’, ya que por las comillas guarda un string (Logico)
      7. La linea puts 'Eso es: meses + ' meses o ' + dias + ' días o ' + hs + ' horas' , toma el primer + dentro del primer string (Sintactico)
      8. Vuelve a intentar usar la variable edad la cual nunca fue definidad. (Semantico)
      9. Utiliza Name en vez de nom (Semantico)
2. public void ordenarArreglo(char[] arregloDeCaracteres, int x) {

boolean ok;

do {

ok = true;

for (int i = 0; i < x - 1; i++) {

if (arregloDeCaracteres[i] > arregloDeCaracteres[i + 1]) {

char aux = arregloDeCaracteres[i];

arregloDeCaracteres[i] = arregloDeCaracteres[i + 1];

arregloDeCaracteres[i + 1] = aux;

ok = false;

}

}

} while (!ok);

}

El For de java necesita definir una variable entera, una condicion y el incremento, el repeat until en java se realiza usando la palabara clave Do, abriendo llaves y luego al cerrarlas la condicion de corte

Los arreglos en java, se definen mediante el tipo seguido de “[]”, y al ser referenciadas, se indica la posicion dentro del arreglo usando los [] y la posicion dentro. Las posiciones de un arreglo en java van de 0 a n

1. SELF: La semantica de self es sensible al contexto y dinamica, esto debido a que por la naturaleza de esta variable, toma el valor dependiendo del contexto donde es llamada. Por ejemplo, si se la llama desde dentro de una clase, self hace referencia a esta misma clase, mientras que si se la llama dentro de un modulo, esta tomara el valor del modulo mismo.

NIL: Nil en ruby es un objeto especial que representa la ausencia de

valor.

Cuando se usa nil en un programa, se considera que hay una sola

instancia de nil para todo el programa. Por lo que comparar Nil == Nil

da siempre true.

Cuando se instancian variables y no se les asigna contenido, metodos sin retorno, acceso a elementos inexistentes (como una posicion inexistente de un arreglo) o una condicion no cumplida, el valor por defecto es NIL

1. NULL: En javaScript, Null es un tipo de dato primitivo, el cual es asignado voluntariamente por el programador para indicar la ausencia de un valor.

Se puede usar para liberar una referencia de un objeto, como valor inicial de una variable, para una funcion que no retorna nada, etc.

UNDEFINED: El undefined en javaScript, es un valor primitivo que representa que una variable no fue inicializada, una funcion sin retorno explicito, acceso a posiciones no existentes o parametros de una funcion no proporcionados.

1. El Break, es una sentencia utilizada para salir de un bucle o una estructura de control

C: En C, se usa para terminal la ejecución de While’s, For’s, Do-While’s y Switch. Cuando esta se ejecuta se pasa al siguiente bucle o a la siguiente instruccion

PHP: En PHP, tiene la misma funcionalidad que en C pero con la particularidad de que se puede especificar cuatos niveles de estructuras anidadas a interrumpir.

JavaScript: En JS, tiene la misma funcionalidad que en C pero en este, se pueden usar Labels para salir de bucles especificos en estructuras anidadas.

Ruby: En ruby similarmente, se usa para salir de bucles como While, For, Until, Each. Pero no se usa con el Case, debido a que Ruby no tiene Fall Through.

1. Ligadura o Binging, hace referencia al proceso de asociar identificadores, con valores, direcciones de memoria, tipos o definiciones.

Este concepto es fundamental para entender cómo un lenguaje

gestiona sus elementos y cómo se resuelven las referencias durante la

compilación o ejecución.

Hay 2 tipos de Ligadura, los cuales se diferencian dependiendo de en que momento se realiza:

* Binding Estatico: El cual se realiza en compilacion, antes de la ejecución del programa.
  + Por ejemplo, en C o en Java, si se escribe:  
    C: int suma(int a, int b) { return a + b; }
  + El tipo de Suma y sus parametros se ligan en tiempo de ejecución.
* Binding Dinamico: Es la ligadura que se realiza en tiempo de ejecución, tipico de lenguajes como Python o JavaScript .

Por ejemplo:

* + let x;
  + x = 10;
  + La ligadura de x a 10 ocurre en tiempo de ejecución

Estática:

* Más rápida porque se resuelve antes de ejecutar el programa.
* Menos flexible, ya que no permite cambios en tiempo de ejecución.
* Ejemplo: C, C++.

Dinámica:

* Más flexible, permite redefinir tipos o valores durante la ejecución.
* Puede ser más lenta y propensa a errores en tiempo de ejecución.
* Ejemplo: Python, Ruby.