💯

# 1er Parcial 100/100

## ▼ Cite y explique tres características de los lenguajes que pueden afectar a la simplicidad general (+).

La simplicidad de un lenguaje afecta su legibilidad y su escribibilidad. Hay tres factores principales que afectan a la simplicidad general de un lenguaje:

Muchas construcciones básicas: Un lenguaje con muchas construcciones básicas es más difícil de aprender. Un programador a veces solo aprende un subconjunto del lenguaje, lo que puede causar problemas de legibilidad si otro programador conoce otro subconjunto. Los problemas de legibilidad ocurren siempre que el autor del programa ha aprendido un subconjunto diferente de aquel con el que el lector está familiarizado.

Multiplicidad de características: La multiplicidad de características se refiere a tener varias formas de realizar una operación en particular. Por ejemplo, en Java, una variable entera se puede aumentar de 4 formas: `count = count + 1`, `count += 1`, `count++`, `++count`

Sobrecarga de operadores: La sobrecarga de operadores se da cuando un símbolo tiene múltiples significados, aunque esto a menudo es útil, puede reducir la legibilidad si se le permite al los usuarios crear su propia sobrecarga y no lo hacen de manera sensata. Por ejemplo, sobrecargar + para sumar enteros y puntos flotantes es útil, pero definir + para sumar elementos de matrices puede ser confuso.

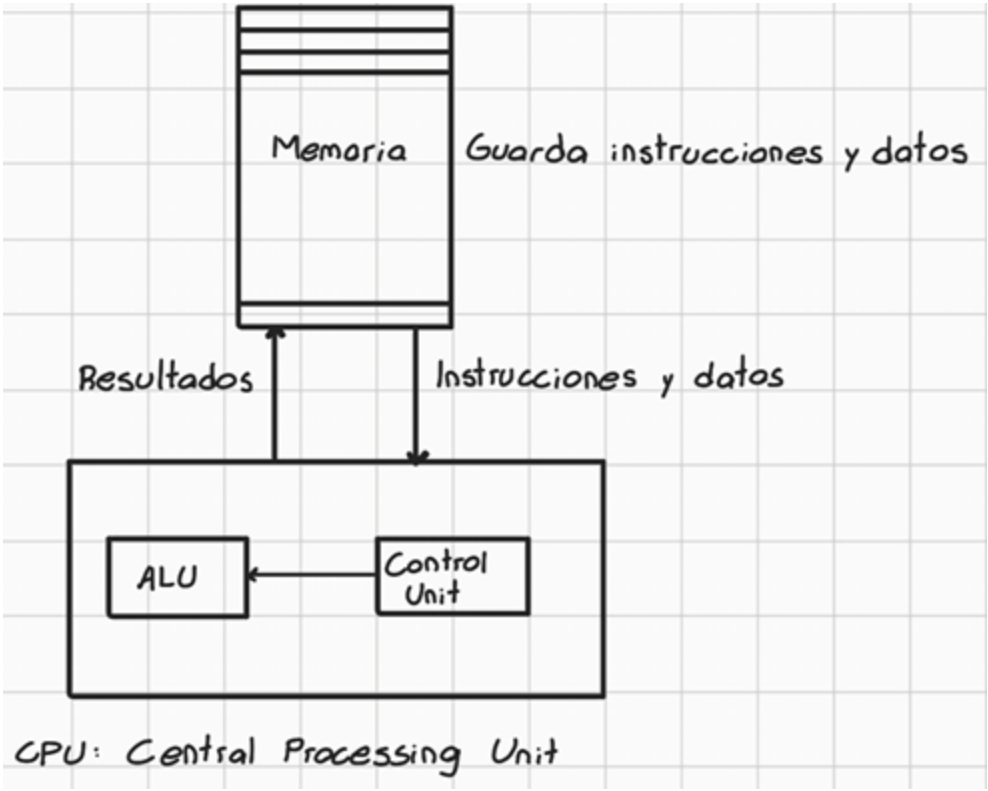
## ▼ Como se produce la ejecución de un código máquina en una computadora basada en Von Neumann.

La ejecución de un programa en este tipo de arquitectura ocurre en un proceso llamado ciclo de búsqueda-ejecución (fetch-execute cycle). Los programas residen en la memoria, pero se ejecutan en el CPU. Cada instrucción que se va a ejecutar debe trasladarse de la memoria al CPU. La dirección de la siguiente

instrucción que se va a ejecutar se mantiene en un registro llamado program counter.

El ciclo de búsqueda-ejecución puede describirse mediante el siguiente algoritmo:

Inicializar el program counter
    Repetir indefinidamente
        Buscar la instrucción apuntada por el program counter
        Aumentar el program counter para apuntar a la siguiente instrucción
        Decodificar la instrucción
        Ejecutar la instrucción
    Fin de la repetición




▼ **Proporcione una tabla indicando los criterios de evaluación de lenguajes y las características deseables de los lenguajes que afectan según Sebesta. Explique detalladamente cada uno de los criterios. Desarrolle brevemente una de las características que se evalúan. (+++++)**

**La legibilidad** se refiere a que tan fácil es leer y comprender un programa escrito en un lenguaje. La legibilidad se debe considerar en el contexto del dominio del problema. Si un programa que describe un cálculo está escrito en un lenguaje no diseñado para tal uso, el programa puede ser antinatural y enrevesado, lo que lo hace difícil de leer.

**La escribibilidad** en un lenguaje se refiere a la facilidad para crear programas en un dominio específico. La mayoría de características que afectan a la legibilidad también afectan a la escribibilidad, ya que escribir un código implica releerlo varias veces. La escribibilidad también debe considerarse en el contexto del dominio del problema. No es justo comparar la facilidad de escritura de dos lenguajes escribiendo una aplicación cuando uno de los lenguajes fue diseñado para esa aplicación y el otro no.

Se dice que un programa es confiable si funciona de acuerdo a sus especificaciones en todas las condiciones.

| Características | Legibilidad | Escribibilidad | Confiabilidad |
|---|---|---|---|
| Simplicidad General | . | . | . |
| Ortogonalidad | . | . | . |
| Tipos de Datos | . | . | . |
| Diseño de Sintaxis | . | . | . |
| Soporte para la Abstracción | | . | . |
| Expresividad | | . | . |
| Chequeo de Tipos | | | . |
| Manejo de Excepciones | | | . |
| Aliasing Restringido | | | . |

Simplicidad General: La simplicidad de un lenguaje puede afectar significativamente la legibilidad de un programa. La simplicidad se ve afectada por 3 factores:

- Muchas construcciones básicas: un lenguaje con muchas construcciones básicas es más difícil de aprender. Puede que un programador se aprenda solo un subconjunto del lenguaje, lo que puede causar problemas de legibilidad si otro usuario conoce otro subconjunto. Los problemas de

legibilidad empiezan siempre que el autor del programa ha aprendido un subconjunto que el lector no conoce.

- Multiplicidad de características: la multiplicidad de características se refiere a tener varias formas de hacer una operación en particular.
- Sobrecarga de operadores: La sobrecarga de operadores se da cuando un símbolo tiene múltiples significados, aunque a menudo es útil, puede ser perjudicial si le permite al usuario crear su propia sobrecarga y estos no lo hacen de manera sensata.

Ortogonalidad: La ortogonalidad en un lenguaje significa que unas pocas construcciones primitivas pueden combinarse de varias formas para crear estructuras de control y datos. Cada combinación es legal y significativa. Por ejemplo, si un lenguaje tiene cuatro tipos de datos (entero, flotante, double, char) y dos operadores de tipo (arreglo y puntero), si los dos operadores de tipo pueden aplicarse entre sí y a los cuatro tipos de datos, se puede definir un gran número de estructuras de datos. Una característica ortogonal es independiente del contexto de su aparición en un programa. La ortogonalidad proviene de la simetría entre primitivas. La falta de ortogonalidad puede llevar a excepciones en las reglas del lenguaje.

Tipos de Datos: La presencia de facilidades adecuadas para definir tipos de datos y estructuras de datos es una ayuda significativa para la legibilidad. Por ejemplo, si se utiliza un tipo numérico para un indicador porque no hay un tipo booleano (timeOut = 1), el significado puede no ser claro, mientras que en un lenguaje con booleanos (timeOut = True), el significado sería perfectamente claro.

Diseño de Sintaxis: La sintaxis tiene gran importancia en la legibilidad de los programas, algunos ejemplos de elecciones de diseño sintáctico que afectan la legibilidad son:

- palabras especiales: la legibilidad de un programa está fuertemente influenciada por las formas de las palabras especiales del lenguaje. Especialmente importante es el método de formar declaraciones compuestas, o grupos de declaraciones. Por ejemplo, en lenguajes como C, usar llaves para agrupar declaraciones puede dificultar la legibilidad, ya que todos los bloques terminan igual, en cambio, en Ada, la legibilidad se

ve aumentada con el uso de más palabras especiales para denotar el cierre de grupos de declaraciones (end if, end loop). Usar más palabras reservadas puede aumentar la legibilidad. Este es un ejemplo del conflicto entre la simplicidad y la mayor legibilidad.

- Forma y significado: Es importante que la forma o significado de las declaraciones sugieran su propósito para mejorar la legibilidad. La semántica o significado debe seguir directamente a la sintaxis o forma. Sin embargo, algunos lenguajes como C, presentan constructos similares en apariencia pero con significados distintos según el contexto, como ocurre con la palabra `Static`.

Soporte para la Abstracción: Es la capacidad de definir y utilizar estructuras complicadas u operaciones de tal manera que los detalles puedan ser ignorados.

- Abstracción de Procesos: utilización de un subprograma para implementar un algoritmo.
- Abstracción de Datos: el lenguaje debería proporcionar facilidades para acercar la solución del problema al dominio del problema.

El ejemplo más simple de abstracción de procesos son los subprogramas, al definir un subprograma, se puede utilizar ignorando como funciona realmente. Elimina la replicación de código. Ignora los detalles de implementación.

Como ejemplo de abstracción de datos, un árbol se puede representar de manera más natural utilizando punteros en los nodos.

Expresividad: La expresividad de un lenguaje puede referirse a varias características diferentes. En un lenguaje como APL, significa que existen operadores muy potentes que permiten realizar una gran cantidad de cálculos con un programa muy pequeño.

Más comúnmente, la expresividad se refiere a que un lenguaje tiene formas relativamente convenientes, en lugar de engorrosas, de especificar cálculos.

En C, la notación `count++` es más conveniente y más corta que `count = count + 1.`
La inclusión de la sentencia

`for` en Java facilita la escritura de bucles en comparación con el uso de `while`. Todas estas características aumentan la facilidad de escritura de un lenguaje.

Chequeo de Tipos: La verificación de tipos consiste en comprobar errores de tipo en un programa, ya sea por el compilador o durante la ejecución de un programa. La verificación de tipos en tiempo de compilación en lugar de en tiempo de ejecución es preferible porque es más económica y permite detectar errores antes.

Un ejemplo de cómo la falta de chequeo de tipos ha causado errores es el uso de parámetros en C. En C, el tipo de un parámetro real en una llamada a función no se verificaba con respecto al tipo de parámetro formal en la función. Por ejemplo, una variable de tipo `int` podía pasarse a una función que esperaba un `float`, y ni el compilador ni el sistema en tiempo de ejecución detectaban esta discrepancia.

Aliasing Restringido: El aliasing es tener dos o más nombres distintos en un programa para acceder a una misma celda de memoria. El aliasing es una característica peligrosa en un lenguaje de programación. La mayoría de los lenguajes de programación permiten algún tipo de aliasing, por ejemplo, dos punteros que apuntan a la misma variable. En un programa así, el programador debe recordar que cambiar el valor a través de uno de los punteros afectará el valor accedido por los demás.

Manejo de Excepciones: El manejo de excepciones se refiere a la capacidad de un programa para interceptar errores en tiempo de ejecución, tomar medidas correctivas y luego continuar. Esto es una ayuda evidente para la confiabilidad. Lenguajes como Ada, C++, Java y C# incluyen amplias capacidades para el manejo de excepciones.

En algunos lenguajes, el aliasing se utiliza para superar deficiencias en las facilidades de abstracción de datos del lenguaje. Otros lenguajes, restringen el aliasing en gran medida para aumentar su confiabilidad.

## ▼ Cite y explique 2 de los ejemplos de decisiones de diseño que afectan al diseño de Sintaxis.

**Palabras especiales:** la legibilidad de un programa esta fuertemente influenciada por las formas de las palabras especiales del lenguaje. Especialmente importante es el método de formar declaraciones compuestas,

o grupo de declaraciones. Por ejemplo, en lenguajes como C, el uso de llaves para agrupar declaraciones puede dificultar la legibilidad, ya que todos los bloques terminan igual. Por otro lado, en lenguajes como Ada, la existencia palabras especiales como end if end loop favorece a la legibilidad. Este es un ejemplo de conflicto entre la simplicidad y la legibilidad.

**Forma y significado:** es importante que la forma o significado de una declaración sugiera su propósito para mejorar la legibilidad. La semántica debe seguir directamente a la sintaxis. Sin embargo, en algunos lenguajes como C, presentan constructos similares en aparecía pero con significados distintos según el contexto, como ocurre con la palabra `Static`

## ▼ Cuales fueron los objetivos de diseño ALGOL? Por quienes fue diseñado? (+)

ALGOL60 se creó como un esfuerzo conjunto de desarrollar un lenguaje de programación universal para aplicaciones científicas. Este esfuerzo surgió en respuesta a la proliferación de lenguajes dependientes de máquinas, que dificultaban el intercambio de programas.

En 1958 se llevó a cabo la primera reunión de diseño, en Zúrich, en la que participaron 4 miembros de la sociedad GAMM de Europa y 4 miembros de la ACM (Association for computing Machinery), de Estados Unidos. En está reunión se establecieron 3 objetivos principales:

- La sintaxis debería ser lo más cercana posible a la notación matemática estándar, y los programas deberían ser legibles con poca explicación adicional.
- El lenguaje debía poder usarse para describir algoritmos.
- Los programas en el nuevo lenguaje deben ser traducibles mecánicamente a lenguaje máquina.

La reunión de Zúrich logró producir un lenguaje que cumpliera con los objetivos establecidos, pero el proceso de diseño requirió innumerables compromisos.

## ▼ Evaluación de smalltalk.

Smalltalk ha hecho mucho para promover dos aspectos de la informática: las interfaces gráficas de usuario y la programación orientada a objetos. Los

sistemas de ventanas actuales, surgieron a partir de Smalltalk. Hoy en día, las metodologías de diseño de software y los lenguajes más importantes son orientados a objetos. Aunque algunos de los conceptos de la programación orientada a objetos comenzaron con SIMULA67, fue con smalltalk que alcanzaron su madurez, consolidando un impacto duradero en el mundo de la informática.

## ▼ Defina formalmente Expresión Regular (++++).

Una expresión regular es una de las siguientes:

1. Una expresión regular **básica** constituida por un solo carácter **a**, donde a proviene de un alfabeto **∑** de caracteres legales; el metacarácter **ε**; o el metacarácter **ϕ**. En el primer caso, $L(\boldsymbol{a}) = \{ a \}$ ; en el segundo, $L(\boldsymbol{\varepsilon}) = \{ \boldsymbol{\varepsilon} \}$ ; en el tercero, $L(\boldsymbol{\phi}) = \{ \}$.
2. Una expresión de la forma **r | s**, donde r y s son expresiones regulares. En este caso,

   $L(\boldsymbol{r} \mid \boldsymbol{s}) = L(r) \cup L(s)$.
3. Una expresión de la forma **rs**, donde r y s son expresiones regulares. En este caso,

   $L(\boldsymbol{rs}) = L(r)L(s)$.
4. Una expresión de la forma **r***, donde r es una expresión regular. En este caso,

   $L(\boldsymbol{r*}) = L(r)*$.
5. Una expresión de la forma (r), donde r es una expresión regular. En este caso,

   $L((r)) = L(r)$. De este modo, los paréntesis no cambian el lenguaje, sólo se utilizan para ajustar la precedencia de las operaciones.

## ▼ Explique cómo influye el avance general de la informática entre las razones para estudiar los conceptos de lenguajes de programación (+).

Aunque a veces se puede entender porque un lenguaje se volvió popular, muchos creen que los lenguajes más populares no siempre son los mejores. A veces, un lenguaje se usó ampliamente porque quienes elegían no estaban familiarizados con los conceptos de programación.

Por ejemplo, muchos creen que ALGOL60 debió haber reemplazado a Fortran en la década del 60, sin embargo, programadores y managers de esa época no entendían bien el diseño conceptual de ALGO60, encontrando su descripción difícil de leer y entender. No apreciaban beneficios como la estructura de bloques y recursión, por lo que no veían las ventajas de ALGOL60 sobre Fortran.

En general, si quienes eligen lenguajes de programación estuvieran bien informados, tal vez mejores lenguajes reemplazarían a los inferiores.

## ▼ Explique cómo afectan la simplicidad y ortogonalidad de un lenguaje de programación a su facilidad de escritura.

Un lenguaje con demasiados constructos puede resultar en que los programadores no estén familiarizados con todos ellos, lo que puede llevar a un mal uso de ciertas características o al desuso de opciones más eficientes o elegantes. Es preferible contar con un conjunto reducido de constructos primitivos y reglas coherentes para combinarlos, ya que esto facilita el aprendizaje y permite resolver problemas complejos de manera más eficiente.

Sin embargo, un exceso de ortogonalidad, donde casi cualquier combinación de primitivas es válida, puede ser contraproducente. Esto puede permitir que errores en los programas pasen desapercibidos, llevando a absurdos en el código que no pueden ser detectados por el compilador.

## ▼ En que consiste la ortogonalidad en un lenguaje de programación. Provea un ejemplo de ortogonalidad, falta de ortogonalidad y exceso de ortogonalidad (++).

La ortogonalidad significa que unos pocos constructos primitivos se pueden combinar de varias formas para crear estructuras de control y datos. Además, cada combinación de primitivas es legal y significativa.

Por ejemplo, si un lenguaje tiene 4 tipos de datos (entero, float, double, char) y dos operadores de tipo (matriz, puntero), si los operadores pueden aplicarse

entre sí y a los 4 tipos, se pueden definir un gran número de estructuras de datos.

Como ejemplo de falta de ortogonalidad, tenemos a C, que teniendo registros y arreglos, un registro puede ser devuelto por una función, un arreglo no. Un miembro de un arreglo puede ser de cualquier tipo de dato, excepto void o una función.

Un ejemplo de exceso de ortogonalidad encontramos en ALGOL 68, en el que una condicional podía ser el lado izquierdo de una asignación: `(if (A<B) then C else D) := 3`

## ▼ Explique los criterios de evaluación de lenguajes propuestos por Sebesta y presente una evaluación de algún lenguaje de programación que conozca, utilizando los criterios y características, propuestos por Sebesta (+).

**Legibilidad:** es la facilidad con la que se puede leer y comprender el código de un programa. Debe considerarse en el contexto del dominio del problema. Si un programa que describe un cálculo está escrito en un programa no diseñado para eso, el programa puede ser antinatural y enrevesado, lo que lo hace difícil de leer.

**Escribibilidad:** se refiere a que tan fácil es crear programas en un dominio específico. La mayoría de las características que afectan a la legibilidad también afectan a la escribibilidad, ya que escribir un programa implica releerlo varias veces. Al igual que la legibilidad, debe considerarse en el contexto del dominio del problema. No es justo comparar la facilidad de escritura de dos lenguajes escribiendo una aplicación cuando uno de los lenguajes fue diseñado para esa aplicación y el otro no.

**Confiabilidad:** Se dice que un programa es confiable si funciona de acuerdo a sus requerimientos en todas las condiciones. Algunas de las características de la confiabilidad son chequeo de tipos, manejo de excepciones, aliasing restringido.

Java, java es un lenguaje el cual su objetivo principal fue la fiabilidad, esto se consiguió incluyendo varias facilidades para el manejo de excepciones, también eliminó los punteros, característica que si tenía C++, de quien estuvo inspirado. Esto fue un sacrificio de flexibilidad por fiabilidad. También, Java

exigió que todas las referencias a elementos de matrices se verifiquen, lo que aumento el costo de ejecución. Esto mejoro la confiabilidad pero redujo la eficiencia.

Al ser un lenguaje orientado a objetos, este lenguaje tiene grandes capacidades para el soporte para la abstracción.

## ▼ Describe como ha influido la arquitectura informática conocida Von Neumann.

La mayoría de los lenguajes populares en los últimos 60 años han sido diseñados en torno a la arquitectura prevalente, llamada arquitectura Von Neumann. Estos lenguajes se llaman lenguajes imperativos. Casi todas las computadoras digitales construidas desde la década de 1940 se han basado en la arquitectura Von Neumann.

Debido a la arquitectura Von Neumann, las características centrales de los lenguajes imperativos son las variables, que modelan celdas de memoria; las declaraciones de asignación, que se basan en la operación de canalización; y la forma iterativa de repetición, que es la forma más eficiente de implementar la repetición en esta arquitectura.

## ▼ Indique un uso de los siguientes métodos de implementación: compilación, interpretación pura y sistemas de implementación híbridos.

Compilación: Los programas se traducen a lenguaje máquina, permitiendo una ejecución muy rápida una vez completada la traducción. C, Cobol, C++

Implementación Pura: Los programas se ejecutan directamente por un intérprete sin realizar ninguna traducción previa. APL, SNOBOL, LISP, PHP, JavaScript

Sistema de implementación híbrida: Traducen programas de lenguajes de alto nivel a un lenguaje intermedio diseñado para facilitar la interpretación. Perl, Java

## ▼ Indique los seis motivos para estudiar las estructuras de los lenguajes de programación según Sebesta.

1. Aumento de la capacidad para expresar ideas: El lenguaje de programación impone límites a las estructuras de control, de datos y abstracciones que

pueden usar, limitando así los algoritmos que pueden crear. Conocer una mayor variedad de características de lenguajes de programación puede reducir estas limitaciones. Al aprender nuevos constructos de lenguaje, los programadores amplían sus procesos de pensamiento en el desarrollo de software.

2. Mejorar la capacidad para elegir lenguajes apropiados: Muchos programadores sin educación formal en ciencias de la computación solo conocen uno o dos lenguajes. Así, al elegir lenguajes para nuevos proyectos, tienden a usar los que ya conocen, aunque no sean los más adecuados. Si estuvieran familiarizados con más lenguajes y sus constructos, podrían seleccionar mejor el lenguaje apropiado.
3. Aumento de la capacidad para aprender nuevos lenguajes: Una comprensión profunda de los conceptos generales de programación facilita el aprendizaje de nuevos lenguajes. Los desarrolladores de software deben estar preparados para aprender varios lenguajes debido a la cantidad de lenguajes en uso y a la naturaleza cambiante de las estadísticas.
4. Comprensión de la implementación: Es esencial abordar los problemas de implementación que afectan a los conceptos de los lenguajes de programación. En algunos casos, esto nos puede ayudar a entender por qué los lenguajes están diseñados de cierta manera y permite usarlos de forma más inteligente. Entender las opciones entre los constructos del lenguaje y sus consecuencias nos hace mejores programadores.
5. Mejor uso de los lenguajes que ya se conocen: La mayoría de lenguajes de programación son grandes y complejos, es poco común que un programador conozca y utilice todas las características de un lenguaje. Al estudiar los conceptos de los lenguajes de programación, el programador puede aprender sobre partes previamente desconocidas y no utilizadas de lenguajes que ya usan para comenzar a utilizar estas características.
6. Avance general de la informática: Aunque normalmente podemos tener una idea del porqué de la popularidad de un lenguaje, muchos piensan que los lenguajes más populares no siempre son los mejores. A veces, un lenguaje se usó ampliamente porque los que elegían no estaban familiarizados con los conceptos de lenguajes de programación. En general, si quienes eligen

lenguajes estuvieran bien informados, tal vez mejores lenguajes reemplazarían a los inferiores.

## ▼ Como afecta la simplicidad general de un lenguaje de programación a su legibilidad? Explique detalladamente.

La simplicidad de un lenguaje afecta su legibilidad. Un lenguaje con muchas construcciones básicas puede ser difícil de aprender. Los programadores a menudo solo aprenden un subconjunto del lenguaje, lo que puede causar problemas de legibilidad si otros conocen otro subconjunto.

Una segunda característica es la multiplicidad de características, es decir, tener múltiples formas de realizar una operación en particular, por ejemplo en Java .....

Un tercer problema potencial es la sobrecarga de operadores, donde un símbolo tiene múltiples significados, aunque esto a menudo es útil, puede causar problemas si se permite al usuario crear su propia sobrecarga y este lo hace de forma no sensata.

Sin embargo, simplificar demasiado también es un problema. Los lenguajes ensambladores son simples, pero menos legibles por la falta de declaraciones de control complejas.

## ▼ Indique 3 criterios de diseño de 3 lenguajes distintos según Sebesta.

Algol 60

- La sintaxis debería ser lo más cercana posible a la notación matemática estándar, y los programas escritos en él deberían ser legibles con poca explicación adicional.
- Debería ser posible usar el lenguaje para la descripción de algoritmos en publicaciones impresas.
- Los programas deben ser traducibles mecánicamente a lenguaje máquina.

Cobol

- Debería usar el inglés tanto como fuera posible.
- El lenguaje debía ser fácil de usar, incluso a costa de ser menos poderoso.

- El diseño no debía estar demasiado restringido por sus problemas de implementación.

## ▼ Describe el contexto histórico, las contribuciones claves, características innovadoras y el objetivo principal de los primeros compiladores FORTRAN.

El desarrollo de Fortran fue impulsado por la introducción de la IBM 704 en 1954, que incluía hardware para punto flotante. Antes de esto, los sistemas que se utilizaban eran interpretados, y como no había hardware para punto flotante, estas operaciones se debían simular mediante software, lo que lo hacía muy lento a estos sistemas interpretativos. Debido a esto, muchos programadores preferían usar lenguaje máquina escrito a mano. Un lenguaje de alto nivel eficiente para aplicaciones científicas era necesario.

El desarrollo de Fortran comenzó incluso antes de la IBM 704. En noviembre de 1954, John Backus y su equipo habían presentado un informe, en el que se detallaba la primera versión de Fortran, Fortran 0. En este informe se afirmaba que el nuevo lenguaje sería tan eficiente como el lenguaje máquina, y tan fácil de escribir como los sistemas interpretativos. También decía que este lenguaje eliminaría los errores de codificación y el proceso de depuración. La eficiencia era uno de los objetivos principales de Fortran.

Fortran I se lanzó en 1957, con características innovadoras como el formato de entrada/salida, nombres de variables de hasta 6 caracteres, subrutinas y declaraciones If y Do.

Su éxito se debió a la notable optimización del compilador, que lograba producir código casi tan eficiente como el escrito a mano. Aproximadamente la mitad del código que se estaba escribiendo en la 704 se estaba escribiendo en Fortran.

Una de las características de Fortran I y de todos sus sucesores antes de los 90, eran que los tipos y el almacenamiento de todas las variables estaban fijos antes del tiempo de ejecución. No se podían asignar nuevas variables durante la ejecución. Esto fue un sacrificio de flexibilidad en favor a la simplicidad y eficiencia.

El efecto que ha tenido Fortran en todas las computadoras, junto con el hecho de que todos los lenguajes de programación posteriores le deben algo a

Fortran, es realmente impresionante a la luz de los modestos objetivos de sus diseñadores.

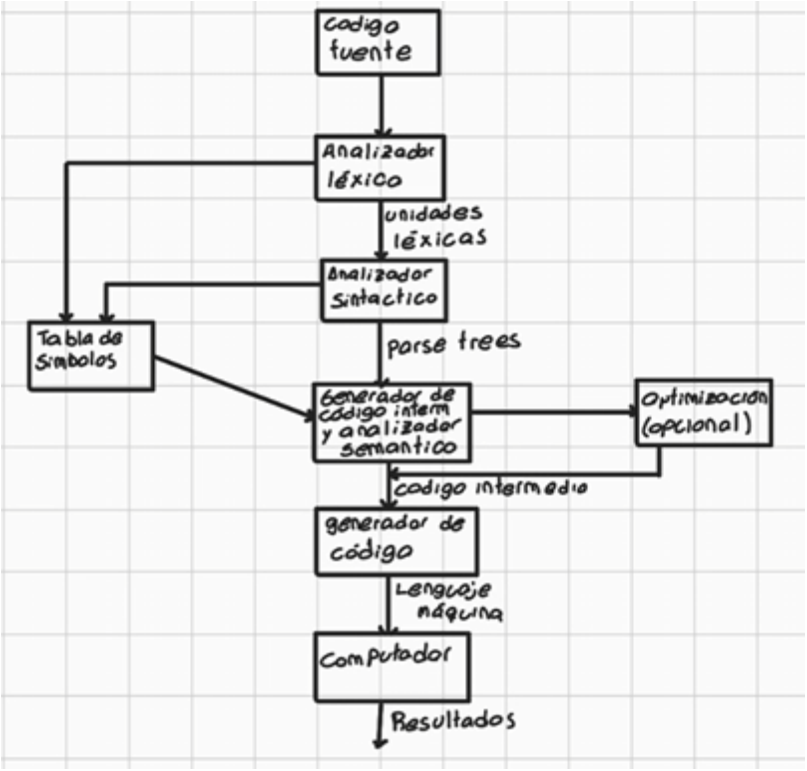
El éxito general de Fortran es difícil de exagerar: cambio drásticamente la forma en la que se usan las computadoras. Esto se debe principalmente a que fue el primer lenguaje de alto nivel ampliamente utilizado.

Alan Perlis, uno de los diseñadores de Algol, dijo: “Fortran es la lingua franca del mundo de la computación”.

## ▼ Presente una breve evaluación del impacto que ha tenido el lenguaje LISP sobre el diseño de los lenguajes.

Lisp como pionero de los lenguajes funcionales, ha sido el principal predecesor de la mayoría de los lenguajes funcionales desde 1960, habiéndose creado muchos dialectos inspirados en Lisp. Algunos de estos son Scheme, Common Lisp, ML, Miranda, Haskell

## ▼ Describe las diferentes etapas del proceso de compilación y realice un gráfico del proceso de compilación. (+++)




El lenguaje que traduce un compilador se llama lenguaje fuente.

El **analizador léxico** agrupa los caracteres del programa fuente en unidades léxicas como identificadores y operadores, ignorando los comentarios.

El **analizador sintáctico** utiliza estas unidades para construir parse trees, que representan la estructura sintáctica del programa.

El **generador de código intermedio** produce un programa en un lenguaje a nivel intermedio entre el programa fuente y el lenguaje máquina. Durante este proceso, el **analizador semántico** verifica errores como los de tipo, que son difíciles de detectar en fase de análisis sintáctico.

La **optimización** mejora los programas haciéndolos más pequeños y/o más rápidos.

La **tabla de símbolos** actúa como base de datos durante la compilación, almacenando el tipo y la información de cada variable definida por el usuario. Esta información es colocada en la tabla de símbolos por los analizadores léxico y sintáctico y es utilizada por el analizador semántico y el generador de código intermedio.

El **generador de código** traduce la versión optimizada del código intermedio al lenguaje máquina, listo para ser ejecutado.

## ▼ Describe elementos del desarrollo de uno de los siguientes lenguajes: Fortran, Lisp, Cobol, Algol. Presenta los siguientes puntos: Principales actores, proceso de diseño, características generales del lenguaje, objetivos, evaluación.

### Fortran

#### Antecedentes

El desarrollo de Fortran fue impulsado por la necesidad de un lenguaje científico de alta eficiencia, esto debido a la creación de la IBM 704, que contaba con hardware de punto flotante. Antes de esto, todos los sistemas utilizados eran interpretativos, ya que no había hardware de punto flotante, por lo que todas las operaciones tenían que ser simuladas en software, lo que los hacía muy lentos, llevando a los programadores a utilizar lenguaje máquina antes que los sistemas interpretativos. La inclusión de hardware de punto flotante eliminó la justificación para el costo de la interpretación.

#### Proceso de Diseño

El desarrollo de Fortran comenzó incluso antes de la IBM 704, liderado por John Backus en IBM. Para noviembre de 1954, John Backus habían presentado un informe que describía la primera versión de Fortran, Fortran 0. Este informe afirmaba que Fortran proporcionaría la eficiencia de los programas codificados en lenguaje máquina y la facilidad de programación de los sistemas interpretativos. También afirmaba que Fortran eliminaría los errores de codificación y el proceso de depuración. La eficiencia era uno de los objetivos principales de Fortran.

Características

Fortran I se lanzó en **1957**, con características innovadoras como el formato de entrada/salida, nombres de variables de hasta 6 caracteres, subrutinas, y declaraciones If y Do.
Su éxito se debió a la notable optimización del compilador, que logró producir código casi tan eficiente como el escrito a mano.
Aproximadamente la mitad del código que se estaba escribiendo para los 704 se estaba escribiendo en Fortran. A lo largo de los años, el lenguaje fue evolucionando con nuevas versiones, como Fortran II, Fortran IV, 77, 90, 95, 2003 y 2008.

Una de las características de Fortran I, y de todos sus sucesores antes del 90, era que los tipos y el almacenamiento de todas las variables estaban fijos antes del tiempo de ejecución. No se podían asignar nuevas variables durante la ejecución. Esto fue un sacrificio de flexibilidad en favor de la simplicidad y eficiencia.

Evaluación

El efecto que Fortran ha tenido en el uso de las computadoras, junto con el hecho de que todos los lenguajes de programación posteriores le deben algo a Fortran, es realmente impresionante a la luz de los modestos objetivos de sus diseñadores.

El éxito general de Fortran es difícil de exagerar: cambio drásticamente la forma en la que se usan las computadoras. Esto se debe en gran parte a que fue el primer lenguaje de alto nivel ampliamente utilizado.

Alan Perlis, uno de los diseñadores de ALGO 60, dijo: “Fortran es la lingua franca del mundo de la computación”.

## Lisp

Lisp fue el primer lenguaje de programación funcional, que surgió a partir del interés en el área de la inteligencia artificial.

### Antecedentes

El interés en la inteligencia artificial surgió a mediados de la década de los 50, investigadores llegaron a la conclusión de que era necesario desarrollar métodos para que las computadoras pudieran procesar datos simbólicos en listas enlazadas, en lugar de los cálculos numéricos en matrices que eran comunes en ese momento.

En 1956, se publicó un artículo en el que se desarrollo el concepto de procesamiento de listas, y un lenguaje en el cual podía implementarse, IPL. Sin embargo, debido a su bajo nivel y a su implementación en la poco conocida máquina Johnniac, no lograron un uso generalizado. IBM también se interesó en la IA a mediados de los 50 y creó el Fortran List Processing Language (FLPL) como una extensión de Fortran.

### Proceso de Diseño

En 1958, John McCarthy investigó la computación simbólica y desarrolló una lista de requerimientos para realizar dichas computaciones, entre ellos estaba la recursión, las expresiones condicionales, la necesidad de listas enlazadas asignadas dinámicamente y algún tipo de desasignación implícita de listas abandonadas.

Como el lenguaje existente, FLPL, no cumplía con los requisitos, McCarthy se propuso desarrollar un nuevo lenguaje: Lisp, un lenguaje puramente funcional diseñado para procesar listas.

### Características

Lisp solo utiliza dos tipos de estructuras de datos: átomos y listas. Los átomos pueden ser símbolos o literales numéricos, y las listas son estructuras de datos enlazadas, que pueden incluir átomos o sublistas.

Las listas se representan como listas enlazadas simples, donde cada nodo contiene dos punteros: uno al elemento de la lista y otro al siguiente nodo. Este diseño permite la inserción y eliminación en cualquier punto de la lista.

Toda la computación se realiza aplicando funciones a argumentos. Ni las sentencias de asignación ni las variables son necesarias en los lenguajes funcionales. Además, permite que los procesos repetitivos se especifiquen mediante llamadas recursivas a funciones, haciendo innecesario los bucles.

La sintaxis de Lisp es un modelo de simplicidad. El código de programa y los datos tienen exactamente la misma forma: listas entre paréntesis. (A B C D).

Evaluación

Lisp dominó las aplicaciones de inteligencia artificial durante un cuarto de siglo. Gran parte de la reputación de Lisp de ser altamente ineficiente ha sido eliminada. Además de su éxito en la IA, Lisp fue pionero en la programación funcional, que ha sido un área vibrante de investigación en lenguajes de programación. Muchos investigadores de lenguajes de programación creen que la programación funcional es un enfoque mucho mejor para el desarrollo de software que la programación procedural utilizando lenguajes imperativos.

## Cobol

Aunque se ha utilizado durante 65 años, COBOL ha tenido poco efecto en el diseño de lenguajes posteriores, excepto en el caso de PL/I. Puede que aún sea el lenguaje más utilizado, aunque es difícil estar seguro de esto. Quizás la razón más importante por la que COBOL ha tenido poca influencia es que pocos han intentado diseñar un nuevo lenguaje para aplicaciones comerciales desde que apareció. Esto se debe en parte a lo bien que las capacidades de COBOL satisfacen las necesidades de su área de aplicación.

Antecedentes

En 1959, el estado de la informática empresarial era similar a la de la informática científica hace varios años, cuando se estaba diseñado Fortran. Un lenguaje compilado para aplicaciones comerciales, FLOW-MATIC, se había implementado en 1957, pero pertenecía solo a UNIVAC y estaba diseñado para las computadoras de esta empresa. Otro lenguaje, AIMACO, estaba siendo utilizado por la fuerza área de los EEUU, pero era solo una

variación menor de FLOW-MATIC. IBM también había diseñado un lenguaje para aplicaciones comerciales, COMTRAN, pero este aún no se había implementado.

Proceso de Diseño

La primera reunión para crear un lenguaje común para aplicaciones comerciales, patrocinado por el DoD de EEUU fue en mayo de 1959. EL consenso fue que el lenguaje debería tener las siguientes características:
Debía usar tanto el inglés como fuera posible.
El lenguaje debía ser fácil de usar, incluso a costa de ser menos poderoso.
El diseño no debía estar demasiado restringido por los problemas de su implementación.

Sobre está base, se decidió hacer un estudio rápido de los lenguajes existentes. Para esta tarea, se conformó el Comité de Corto Alcance.

La especificación del lenguaje, denominado COBOL 60, fue publicado en abril de 1960.

Características

Cobol introdujo varios conceptos novedosos como el verbo DEFINE, utilizado para macros, y las estructuras de datos jerárquicas, que han sido incluidas en la mayoría de los lenguajes imperativos desde entonces. Fue pionero en permitir que los nombres fueran verdaderamente connotativos. Su división de datos era la parte fuerte, ideal para tareas contables, pero su división de procedimientos era débil inicialmente, careciendo de funciones y soporte para subprogramas.

Evaluación

Fue el primer lenguaje de programación cuyo uso fue obligatorio por parte del Departamento de Defensa. A pesar de sus méritos, COBOL probablemente no habría sobrevivido sin este mandato. El bajo rendimiento de los primeros compiladores simplemente hacía que el lenguaje fuera demasiado costoso de usar. Eventualmente, los compiladores se volvieron más eficientes y las computadoras se volvieron más rápidas y baratas. Juntos, estos factores permitieron que COBOL tuviera éxito dentro y fuera del DoD. Su aparición condujo a la mecanización electrónica de la contabilidad.

## Algol

### Antecedentes

ALGOL60 fue el resultado de esfuerzos para diseñar un lenguaje de programación universal para aplicaciones científicas. Luego de que fortran se hiciera realidad en 1957, se empezaron a desarrollar varios otros lenguajes de alto nivel. La proliferación de lenguajes dificultaba el intercambio de programas entre los usuarios. Además, los nuevos lenguajes estaban creciendo en torno a arquitecturas específicas. Debido a esto, varios grupos importantes de usuarios de computadores de Estados Unidos, presentaron una petición a la ACM para formar un comité que estudiara y recomendara acciones para crear un lenguaje de programación científica independiente de la máquina.

Anteriormente, en 1955, la asociación GAMM había formado un comité para diseñar un lenguaje algorítmico universal, independiente de la máquina. Esto debido al temor de los europeos de ser dominados por IBM. Sin embargo, a finales de 1957, la aparición de varios lenguajes de alto nivel en los Estados Unidos convenció al GAMM de que su esfuerzo debía ampliarse para incluir a los estadounidenses, y se envió una carta de invitación a ACM. En abril de 1958, ambos grupos acordaron oficialmente un proyecto conjunto de diseño de lenguaje.

### Proceso de Diseño

La primera reunión de diseño se celebro en Zúrich en mayo de 1958, donde participaron cuatro miembros de GAMM y cuatro miembros de la ACM. Se establecieron tres objetivos principales:

- La sintaxis del lenguaje debería ser lo más cercana posible a la notación matemática estándar, y los programas escritos en él deberían ser legibles con poca explicación adicional.
- Debería ser posible usar el lenguaje para la descripción de algoritmos en publicaciones impresas.
- Los programas en el nuevo lenguaje deben ser traducibles mecánicamente a lenguaje máquina.

La reunión de Zúrich logró producir un lenguaje que cumpliera con los objetivos establecidos, pero el proceso de diseño requirió de innumerables compromisos.

Este lenguaje fue inicialmente nombrado International Algoritmic Language (IAL), sin embargo, durante el año siguiente, el nombre se cambió a ALGOL, siendo posteriormente conocido como ALGOL 58.

Características

En muchos aspectos, ALGOL58 fue un descendiente de fortran, generalizando muchas de sus características y añadiendo nuevas construcciones y conceptos. Algunas de las generalizaciones tenían que ver con el objetivo de no vincular el lenguaje a ninguna máquina en particular, y otras fueron intentos de hacer el lenguaje más flexible y poderoso. De este esfuerzo surgió una rara combinación de simplicidad y elegancia.

ALGOL58 formalizó el concepto de tipo de datos, añadió la idea de sentencias compuestas, permitió características como identificadores de longitud ilimitada, cualquier número de dimensiones en matrices, y sentencias de selección anidadas.

El informe de ALGOL 58, se veía más como una colección de ideas para el diseño de lenguajes que como un lenguaje estándar universal. En realidad, el informe ALGO58 no estaba concebido para ser un producto final, sino como un documento preliminar para la discusión internacional.

Al principio, tanto IBM como SHARE parecían adoptar ALGOL58, sin embargo, para la primavera de 1959, pronto abandonaron ALGOL58 en favor de Fortran, debido a las dificultades y costos asociados con la implementación y adopción de un nuevo lenguaje.

En enero de 1960, se celebró la segunda reunión de ALGOL, el propósito fue debatir las 80 sugerencias que habían sido presentadas para su consideración. Peter Naur jugó un papel crucial al adaptar BNF para describir el nuevo lenguaje propuesto, ALGOL60.

En la reunión de 1960, se realizaron modificaciones importantes a ALGOL58, dando lugar a ALGOL60. Entre los desarrollos más importantes estaban los siguientes:

- El concepto de estructura de bloques, que permitía la localización de partes de los programas.
- Dos métodos para pasar parámetros: paso por valor y paso por nombre.
- Se permitió que los procedimientos fueran recursivos.
- Se permitieron matrices dinámicas en pila.

Evaluación

En algunos aspectos, ALGOL60 fue un gran éxito; en otros, fue un rotundo fracaso. Logró convertirse casi de inmediato en el único medio formal aceptable para comunicar algoritmos, y se mantuvo así por más de 20 años. Todos los lenguajes de programación imperativos diseñados desde 1960 deben algo a ALGOL 60, de hecho, la mayoría son descendientes directos o indirectos.

Fue pionero en muchos aspectos, fue la primera vez que un grupo internacional intentó diseñar un lenguaje de programación, fue el primer lenguaje que fue diseñado para ser independiente de la máquina, también fue el primer lenguaje cuya sintaxis fue descrita formalmente.

Sin embargo, su implementación fue complicada, y la falta de soporte de IBM y la predominancia de Fortran en EEUU contribuyeron a su fracaso para lograr un uso generalizado. Algunas de sus características resultaron ser demasiado flexibles; dificultaban la comprensión y hacían ineficiente la implementación. Además, la complejidad percibida del BNF dificultaron su adopción.

## ▼ Evaluación de ALGOL 60.

En algunos aspectos, ALGOL60 fue un éxito; en otros, fue un rotundo fracaso,. Logro consolidarse casi de inmediato como el único lenguaje aceptado para describir algoritmos, y se mantuvo así por más de 20 años. Todos los lenguajes de programación imperativos diseñados desde 1960 deben algo a ALGOL60, de hecho, la mayoría son descendientes directos o indirectos.

Fue pionero en muchos aspectos, fue la primera vez que un grupo internacional intentó desarrollar un lenguaje de programación, fue el primer lenguaje independiente de la máquina, también fue el primer lenguaje cuya sintaxis fue formalmente descrita.

Sin embargo, su implementación fue complicada, la falta de soporte de IBM y el uso arraigado de Fortran en los EEUU contribuyeron a su fracaso para lograr un uso generalizado. Algunas de sus características resultaron ser demasiado flexibles; dificultaban la comprensión y hacían ineficiente la implementación. Además, la dificultad percibida del BNF dificultó su adopción.

## ▼ Nombre una persona que ha tenido una gran influencia en el desarrollo del:

**El modelo imperativo:** John Backus

**El modelo funcional:** John McCarthy

**La orientación a objetos:** Alan Kay

**El modelo de programación lógica:** Robert Kowalski

## ▼ Evaluación de LISP.

Lisp dominó completamente las aplicaciones de inteligencia artificial por al menos un cuarto de siglo. Gran parte de su reputación de ser altamente ineficiente ha sido eliminada. Además de su éxito en IA, Lisp fue pionero en la programación funcional, que ha demostrado ser un área vibrante de investigación en lenguajes de programación. Muchos investigadores de lenguajes de programación creen que la programación funcional es un enfoque mucho mejor para el desarrollo de software que la programación procedural de los lenguajes imperativos.

## ▼ Define formalmente un DFA, un NFA, cuales son las diferencias entre ambos?

Un DFA M se compone de un alfabeto Z, un conjunto de estados S, una función de transición T = S x Z → S, un estado inicial so \in S, y un conjunto de estados de aceptación A \in S. El lenguaje aceptado por M escrito como L (M), se define como el conjunto de cadenas de caracteres c1, c2, ... cn con cada ci \in Z, tal que existen estados s1 = T(s0, c1), s2 = T(s1, c2), ... , sn = T(sn-1, cn) con sn como un elemento de A.

Un NFA M se compone de un alfabeto Z, un conjunto de estados S, una función de transición T = S x ( Z \cup {e}) → S, un estado inicial s0 \in S, y un conjunto de estados de aceptación A \in S. El lenguaje aceptado por M, escrito como L(M), se define como el conjunto de cadenas de caracteres c1, c2, ... cn,

con cada ci \in Z\cup{e}, tal que existen estados s1 en T(s0, c1), s2 en T(s1, c2), ... , sn en T(sn-1, cn) con sn como un elemento de A.

## ▼ Cite y explique brevemente los criterios que afectan a la legibilidad.

Simplicidad General: la simplicidad general afecta en gran parte a la legibilidad. Un lenguaje con muchas construcciones básicas lo hace más difícil de aprender. Puede que un programador conozca solo un subconjunto del lenguaje, lo que dificulta la legibilidad si otro conoce otro subconjunto de este. Un segundo problema es la multiplicidad de características, o sea, tener varias formas de realizar una operación en particular. En java....
Un tercer problema potencial es la sobrecarga de operadores, que se refiere a asignarle distintos significados a un mismo símbolo. Aunque esto a menudo es bueno, puede ser contraproducente si se le permite al usuario crear su propia sobrecarga y este no lo hace de manera sensata. Sobrecargar el símbolo + para sumar enteros y flotantes puede ser útil, pero usar + para sumar elementos de matrices puede ser confuso.

Ortogonalidad: La ortogonalidad en un lenguaje se consigue estableciendo un grupo reducido de constructos primitivos con reglas coherentes para combinarlos. Toda combinación es legal y significante. Por ejemplo, si tenemos 4 tipos de datos y 2 operadores de tipo, si combinamos estos 2 operadores de tipos entre si y a los 4 tipos de datos, podemos crear un gran número de estructuras de datos.

Tipos de Datos: Es importante ofrecer facilidades para el uso de distintos tipos de datos. Por ejemplo, en un lenguaje sin booleanos, se podría usar un entero para indicar el estado de una bandera, TimeOut = 1, esto puede no tener un significado claro o ser confuso, en cambio, en un lenguaje con booleanos, el significado sería más claro. TimeOut = True.

Diseño de Sintaxis: Hay dos consideraciones importantes en el diseño de sintaxis para mejorar la legibilidad de un lenguaje.

> Palabras especiales: El uso de más palabras especiales podría ser de ayuda para la legibilidad. Por ejemplo, en lenguajes como C, donde se usan llaves para agrupar declaraciones, se puede llegar a dificultar la legibilidad, ya que todos los bloques terminan igual. En cambio, el Ada, el uso de palabras reservadas como End If o End Loop para denotar cierres de bloques es de

gran ayuda para la legibilidad. Esto es un ejemplo de sacrificio de simplicidad por legibilidad.

Forma y significado: La forma y el significado de una declaración debería sugerir su propósito. La semántica o significado debe seguir directamente a la sintaxis o forma. Sin embargo, algunos lenguajes como C, presentan constructos de apariencia similar pero con significado distinto según su contexto, como es el caso del Static.

## ▼ Explique como influye el manejo de excepciones en la confiabilidad de un lenguaje para crear programas.

El manejo de excepciones se refiere a las facilidades de un lenguaje para detectar errores en tiempo de ejecución, corregirlos y continuar con el programa. Esto es una ayuda evidente para la confiabilidad. Lenguajes como Ada, C++, Java y C# ofrecen varias facilidades para el manejo de excepciones.

## ▼ Definición formal de DFA.

Un DFA (por las siglas de autómata finito determinístico en inglés) $M$ se compone de un alfabeto $\sum$, un conjunto de estados $S$, una función de transición $T : S \times \sum \to S$, un estado de inicio $s_0 \in S$ y un conjunto de estados de aceptación $A \subset S$. El lenguaje aceptado por M escrito como $L(M)$ , se define como el conjunto de cadenas de caracteres $c_1, c_2, \dots, c_n$ con cada $c_i \in \sum$, tal que existen estados $s_1 = T(s_0, c_1), s_2 = T(s_1, c_2), \dots, s_n = T(s_{n-1}, c_n)$, con $s_n$ como un elemento de A (es decir, un estado de aceptación).

## ▼ Definición formal de NFA.

Un NFA (por las siglas de autómata finito no determinístico en inglés) $M$ se compone de un alfabeto $\sum$, un conjunto de estados $S$, una función de transición $T : S \times (\sum \cup \{\epsilon\}) \to \delta(S)$, un estado de inicio $s_0 \in S$ y un conjunto de estados de aceptación $A \subset S$. El lenguaje aceptado por M escrito como $L(M)$, se define como el conjunto de cadenas de caracteres $c_1, c_2, \dots, c_n$ con cada $c_i \in \sum \cup \{\epsilon\}$, tal que existen estados $s_1$ en $T(s_o, c_1)$, $s_2$ en $T(s_1, c_2)$, ... , $s_n$ en $T(s_{n-1}, c_n)$, con $s_n$ como un elemento de A (es decir, un estado de aceptación).

## ▼ Cuales son tres de los “language design trade-offs” señalados por Sebesta.

Fiabilidad por costo de ejecución. Java. Por ejemplo, Java exige que todas las referencias a elementos de matrices se verifiquen, lo que aumenta el costo de ejecución, mientras que C no realiza estas verificaciones, haciendo que sus programas sean más rápidos pero menos fiables. Los diseñadores de Java sacrificaron la eficiencia por la fiabilidad.

Escribibilidad por Legibilidad. APL. APL tiene un conjunto muy poderoso de operadores, lo que resulta en un alto grado de expresividad, por lo que, APL es muy fácil de escribir. Sin embargo, los programas en APL tienen una legibilidad muy pobre. Una expresión compacta y concisa tiene una cierta belleza matemática, pero es difícil de entender para cualquiera que no sea programador. El diseñador de APL sacrificó legibilidad por facilidad de escritura.

Escribibilidad por Fiabilidad. C++. C++ permite la manipulación flexible de punteros, lo que no se incluye en Java debido a problemas potenciales de fiabilidad.