💯

2do Parcial 100/100

▼ Por examen

▼ 2023-1

▼ Explica en al menos 120 palabras, qué es la transparencia referencial y cómo se relaciona la transparencia referencial con los efectos colaterales [side effects] de las funciones?

▼ Cuáles son tres consideraciones de diseño relacionadas a los Selectores de dos vías según Sebesta?

▼ Cuáles son cinco consideraciones de diseño relacionadas a los selectores múltiples según Sebesta?

▼ Cuáles son las categorías en las que pueden clasificarse las sentencias de control de iteraciones? Cuales son las preguntas que sirven para definir estas clasificaciones?

▼ Cuáles son tres consideraciones de diseño relacionadas a las iteraciones controladas por contador en Sebesta?

▼ Cuáles son las tres características de una subrutina? Cuáles son 11 consideraciones de diseño principales de las subrutinas?

▼ Cite y explique brevemente los 5 tipos de paso de parámetros? Explica en al menos 50 palabras en cada caso.

▼ Cuáles son las complicaciones que existen con el paso de parámetros que son subprogramas? Cuáles son las opciones que se han desarrollado para lidiar con la cuestión de la determinación del entorno en referencia, explique brevemente. Explica en al menos 50 palabras.

▼ Cómo se relacionan los TDA con la encapsulación y el ocultamiento de información? Explica en al menos 50 palabras.

▼ 2023-2 / 2024-1

▼ ¿Cuál es la principal diferencia entre un registro y una tupla?

A) Los registros tienen campos con nombres y las tuplas no

B) Las tuplas tienen campos con nombres y los registros no

C) Los registros pueden contener diferentes tipos de datos, pero las tuplas no

D) No hay diferencia

▼ ¿Cuál es una ventaja de los tipos de datos decimales?

A) Ocupan menos memoria que los tipos enteros

B) Son más rápidos que los tipos enteros

C) Permiten una representación más precisa de números fraccionarios

D) No tienen desventajas

▼ Defina error de tipo.

A) Ocurre cuando se realiza una operación en un tipo de dato que es compatible con esa operación

B) Ocurre cuando se realiza una operación en un tipo de dato que no es compatible con esa operación

C) Ocurre cuando se utiliza un tipo de dato incorrecto en una declaración

D) Ocurre cuando se declara una variable con un tipo incorrecto

▼ ¿Cuáles son los dos problemas comunes con los punteros?

A) Punteros rápidos y punteros lentos

B) Punteros nulos y punteros colgantes

C) Punteros largos y punteros cortos

D) Punteros seguros y punteros inseguros

▼ Defina unión, unión libre y unión discriminada.

A) Unión: varios tipos en el mismo espacio de memoria, Unión libre: unión sin discriminador de tipo, Unión discriminada: unión con discriminador de tipo

B) Unión: unión con discriminador de tipo, Unión libre: unión sin discriminador de tipo, Unión discriminada: varios tipos en el mismo espacio de memoria

C) Unión: unión sin discriminador de tipo, Unión libre: varios tipos en el mismo espacio de memoria, Unión discriminada: unión con discriminador de tipo

D) Unión: varios tipos en el mismo espacio de memoria, Unión libre: unión con discriminador de tipo, Unión discriminada: unión sin discriminador de tipo

▼ ¿Qué lenguajes permiten subíndices negativos para acceder a elementos de un arreglo?

A) C++

B) Python

C) Java

D) C#

▼ ¿Qué tipo de datos tiene un orden secuencial?

A) Enumeración

B) Subrango

C) Ordinales

D) Puntero

▼ ¿Cuál de las siguientes opciones mejor define la precedencia de operadores?

A) Orden en el que se ejecutan los programas

B) Orden en el que se evalúan los operadores en una expresión

C) Cantidad de operadores en una expresión

D) Velocidad de ejecución de los operadores

▼ ¿Qué es un cast?

A) Conversión implícita de un tipo de dato a otro

B) Conversión explícita de un tipo de dato a otro realizada por el programador

C) Error de tipo en un programa

D) Función especial en programación

▼ ¿Qué es una coerción?

A) Error de tipo en un programa

B) Conversión explícita de un tipo de dato a otro

C) Conversión implícita de un tipo de dato a otro

D) Función especial en programación

▼ ¿Qué es la transparencia referencial?

A) Propiedad de las funciones que dependen del estado del programa

B) Propiedad de las funciones que siempre producen el mismo resultado para los mismos argumentos

C) Capacidad de una función para cambiar su entorno

D) Tipo especial de variable

▼ ¿En qué sentido es la declaración `for` de C más flexible que en muchos otros lenguajes?

A) C permite múltiples declaraciones en la inicialización.

B) C solo permite enteros en la declaración `for`.

C) C permite cualquier expresión en las secciones de inicialización, condición y actualización.

D) C no permite bucles `for`.

E) C permite cualquier expresión en las secciones de inicialización, condición y actualización.

▼ ¿Qué ventaja tiene la declaración `break` de Java sobre la declaración `break` de C?

A) Java no tiene declaración `break`.

B) Java permite salir de bloques anidados específicos con etiquetas.

C) Java no permite usar `break` en bloques de `switch`.

D) No hay diferencia entre Java y C en cuanto a la declaración `break`.

▼ Seleccione la opción que describe correctamente tres situaciones en las que se necesita una declaración de bucle combinada de conteo y lógica.

A) 1) Cuando necesitas repetir una acción un número específico de veces, pero también necesitas una condición específica para continuar. 2) Cuando estás procesando elementos de una lista y necesitas parar cuando encuentres un valor específico. 3) Cuando estás esperando a que un recurso esté disponible, pero solo quieres intentarlo por un tiempo limitado.

B) 1) Cuando necesitas repetir una acción un número infinito de veces. 2) Cuando solo necesitas una declaración de bucle basada en una condición lógica. 3) Cuando quieres iterar a través de todos los elementos de una colección sin ninguna condición específica.

C) 1) Cuando estás realizando una búsqueda y quieres detenerte cuando encuentres el elemento o después de un cierto número de intentos. 2) Cuando estás intentando conectarte a un servidor y quieres reintentar un número específico de veces antes de rendirte. 3) Cuando estás validando entrada del usuario y quieres darle un número limitado de intentos para ingresar datos válidos.

D) 1) Cuando solo necesitas un bucle que se ejecute un número fijo de veces. 2) Cuando estás realizando una operación que no requiere condiciones de parada. 3) Cuando estás realizando cálculos matemáticos que no involucran iteraciones.

▼ Presenta tus propios argumentos a favor y en contra de permitir expresiones aritméticas de modo mixto.
▼ ¿Crees que la eliminación de operadores sobrecargados en tu lenguaje de programación favorito sería beneficiosa? ¿Por qué sí o por qué no?
▼ Describe una situación en la cual el operador de suma en un lenguaje de programación no sería asociativo.
▼ Explique de forma detallada la diferencia entre variables de pila dinámicas (stack-dynamic) y variables de montón explícito dinámicas (explicit heap-dynamic). Proporcione ejemplos de lenguajes que utilizan cada tipo y discuta las ventajas y desventajas de cada tipo de variable.

▼ 2020-2

▼ ¿Cuáles son las 3 características generales de los subprogramas?
▼ Describe correctamente el problema de pasar arrays multidimensionales como parámetros.
▼ ¿Cómo se representan las referencias a las variables en el método static-chain?
▼ Explique qué es la dirección de una variable.
▼ Define conversión narrowing y conversión widening. Mencione un ejemplo de cada conversión.
▼ ¿Qué es una definición de subprograma?
▼ Describe correctamente el método shallow-access para implementar el alcance dinámico.
▼ ¿Cuál es la forma general de un selector de dos vías?
▼ Qué es subtipo de polimorfismo?
▼ ¿Qué es un ancestor estático de un subprograma? ¿Qué es un ancestor dinámico de un subprograma?
▼ ¿Qué es el enlace ad hoc?
▼ ¿Qué mecanismo sigue un compilador cuando el número de casos en una declaración de selección es 10 o más para optimizar el tiempo requerido de ejecución?

- ▼ ¿Qué mecanismo se utilizan para almacenar enteros negativos en una computadora?
- ▼ ¿Cuáles son los problemas de diseño para las declaraciones de bucle controladaslógicamente?
- ▼ ¿Cuál es la diferencia entre un registro de activación y una instancia de registro de activación?
- ▼ ¿Cómo un lenguaje funcional implementa la repetición?
- ▼ ¿Qué debe almacenarse para la vinculación (linkage) a un subprograma?
- ▼ ¿Qué es un tipo de dato abstracto?
- ▼ ¿Cuál es el problema de diseño en arrays asociativos?

▼ 2017-1

- ▼ Cuáles son los 6 atributos de las variables? Explique brevemente cada uno de ellos.
- ▼ Qué es un entorno de referencia?
- ▼ En que consiste el entorno de referencia en un lenguaje de alcance estático?
- ▼ Porqué se requiere del entorno de referencia al momento de la compilación?
- ▼ Qué tipo de datos puede ser utilizado para representar de manera precisa el valor e (la base para el logaritmo natural) o el valor de pi? Presente una breve descripción del tipo de datos que corresponde.
- ▼ Cuáles son las 6 cuestiones de diseño fundamentales de las expresiones aritméticas según Sebesta? Explique brevemente cada una de ellas.

▼ 2010

- ▼ Precedencia y asociatividad de operadores.
- ▼ Side-effects de una función. Ejemplo.
- ▼ Por qué los diseñadores de los lenguajes no están de acuerdo con respecto a la coerción en expresiones aritméticas?

▼ Cuáles son las únicas estructuras de control necesarias?
▼ Cuáles son las sentencias break de java y c++?
▼ Cuestiones de diseño para el manejo de excepciones.

▼ Final 2023-2

▼ En términos de aliasing, ¿qué afirmación es cierta respecto a la fiabilidad?

a) El aliasing es deseable para la claridad del código

b) El aliasing aumenta la fiabilidad al permitir múltiples nombres para una variable

c) La restricción del aliasing aumenta la fiabilidad del programa

d) El aliasing no tiene impacto en la fiabilidad del programa

▼ ¿Cuál es una característica clave de las variables en la programación imperativa en relación con la arquitectura de la computadora von Neumann?

a) Simulan el comportamiento del procesador

b) Son abstracciones de las celdas de memoria de la máquina

c) Representan las instrucciones del programa

d) Actúan como puertas lógicas en circuitos

▼ En el contexto de los lenguajes de programación, ¿cómo se define el 'alcance' de una variable?

a) La duración de la memoria asignada a la variable

b) El rango de operaciones posibles en la variable

c) El rango de instrucciones en las que la variable es visible

d) La compatibilidad de la variable con diferentes tipos de datos

▼ En la programación funcional, las expresiones nombradas se asemejan a:

a) Variables mutables

b) Punteros

c) Constantes nombradas

d) Argumentos de funciones

▼ ¿Cuál es una desventaja principal del uso de variables estáticas en la programación?

a) No pueden ser utilizadas en subprogramas recursivos

b) Requieren una mayor cantidad de memoria

c) Son menos eficientes en tiempo de ejecución

d) No son compatibles con la mayoría de los lenguajes modernos

▼ ¿Qué caracteriza a una variable 'heap-dinámica explícita' en programación?

a) Su ciclo de vida y tipo son dinámicamente vinculados durante la ejecución

b) Son liberadas automáticamente por el recolector de basura

c) Son vinculadas durante la compilación del programa

d) Tienen un alcance global y estático en el programa

▼ Final 2016-2

▼ Explique detalladamente qué es el entorno de referencia de una sentencia, cómo se usa?

▼ Cuáles son los tipos de enlaces de almacenamiento (storage bindings). Explique detalladamente cada uno de ellos.

▼ Cite y explique detalladamente los dos enfoques existentes para la equivalencia de tipo.

▼ Qué es la transparencia referencial y cómo se relaciona con los efectos colaterales? (Referential Transparency and side Effects)

▼ Mencione un uso común inadecuado de los resultados de Böhm y Jacopin.

▼ Defina shallow y deep binding para entornos de referencia de subprogramas que han sido pasados como

parámetros. Explique detalladamente.

▼ Qué debe ser almacenado para la vinculación a un subprograma? (What must be stored for the linkage to a subprogram?)

▼ Final 2016-2.2

▼ Defina static scoping y dynamic scoping. Cuáles son las reglas que se aplican en cada caso.

▼ Un array se puede almacenar en row-major-order como se hace en C++ o en column-major-order como ocurre en FORTRAN. Escriba una función de acceso para row-major-order.

▼ Defina named type equivalence y structure type equivalence

▼ Cite las cuestiones de diseño de los selectores de 2 vías.

▼ Qué es un heap-dynamic array? Cuál es su ventaja?. Cuál es su desventaja?. Dar ejemplo.

▼ Cuál es la diferencia entre un static array y un heap dynamic array?

▼ Cite las 6 cuestiones de diseño en las expresiones aritméticas según Sebesta.

▼ Cite las cuestiones de diseño de los registros.

▼ Dibuje la organización del almacenamiento en tiempo de ejecución. Describa brevemente sus partes.

▼ Dibuje un registro de activación. Describa brevemente sus partes.

▼ Los pasos que se siguen al ser llamado un procedimiento.

▼ Los pasos que se siguen cuando el procedimiento sale de escena.

▼ **Describa un ambiente de ejecución basado en pila sin procedimientos locales.**

▼ Por temas

▼ ¿Qué tipo de datos tiene un orden secuencial?

A) Enumeración

B) Subrango

C) Ordinales

D) Puntero

▼ Describe una situación en la cual el operador de suma en un lenguaje de programación no sería asociativo.

▼ U5 - NAMES, BINDINGS, TYPE CHECKING, AND SCOPES

▼ ¿Cuál es una característica clave de las variables en la programación imperativa en relación con la arquitectura de la computadora von Neumann?

a) Simulan el comportamiento del procesador

b) Son abstracciones de las celdas de memoria de la máquina

c) Representan las instrucciones del programa

d) Actúan como puertas lógicas en circuitos

▼ En la programación funcional, las expresiones nombradas se asemejan a:

a) Variables mutables

b) Punteros

c) Constantes nombradas

d) Argumentos de funciones

▼ Explique qué es la dirección de una variable.

La dirección de una variable es la dirección de memoria de la máquina con la que está asociada. La dirección de una variable también se conoce como “l-value”, ya que la dirección es lo que se necesita cuando una variable aparece en el lado izquierdo de una asignación.

▼ En términos de aliasing, ¿Qué afirmación es cierta respecto a la fiabilidad?

a) El aliasing es deseable para la claridad del código

b) El aliasing aumenta la fiabilidad al permitir múltiples nombres para una variable

c) La restricción del aliasing aumenta la fiabilidad del programa

d) El aliasing no tiene impacto en la fiabilidad del programa

▼ Cuáles son los 6 atributos de las variables? Explique brevemente cada uno de ellos.

Nombre: Es una cadena de caracteres utilizada para identificar a la variable.

Dirección: La dirección de una variable es la dirección de memoria de la máquina con la que está asociada.

Tipo: El tipo de una variable determina el rango de valores que la variable puede almacenar y el conjunto de operaciones que están definidas para los valores de este tipo.

Valor: Es el contenido de la o las celdas de memoria asociadas con la variable.

Tiempo de vida: Es el periodo durante el cual una variable esta asociada a una ubicación de memoria especifica.

Alcance: Es el rango de sentencias en las que una variable es visible, es decir, donde puede ser referenciada o asignada.

▼ ¿Cuál es una desventaja principal del uso de variables estáticas en la programación?

a) No pueden ser utilizadas en subprogramas recursivos

b) Requieren una mayor cantidad de memoria

c) Son menos eficientes en tiempo de ejecución

d) No son compatibles con la mayoría de los lenguajes modernos

▼ Explique de forma detallada la diferencia entre variables de pila dinámicas (stack-dynamic) y variables de montón explícito dinámicas (explicit heap-dynamic). Proporcione ejemplos de lenguajes que utilizan cada tipo y discuta las ventajas y desventajas de cada tipo de variable.

▼ ¿Qué caracteriza a una variable 'heap-dinámica explícita' en programación?

a) Su ciclo de vida y tipo son dinámicamente vinculados durante la ejecución

b) Son liberadas automáticamente por el recolector de basura

c) Son vinculadas durante la compilación del programa

d) Tienen un alcance global y estático en el programa

▼ Cuáles son los tipos de enlaces de almacenamiento (storage bindings). Explique detalladamente cada uno de ellos.

Variable estática: Son las variables en las que tanto su almacenamiento como su tipo se asocian en tiempo de compilación. Estos no pueden cambiar a lo largo de la ejecución del programa. Son utilizadas normalmente en variables globales. Su principal ventaja es la eficiencia, ya que permiten direccionamiento directo y no generan sobrecarga en tiempo de ejecución.

Variable dinámica en pila: Las variables dinámicas en pila tienen su tipo asociado en tiempo de compilación, pero su almacenamiento se asocia cuando se elabora su declaración, o sea, cuando la ejecución alcanza el código asociado a la declaración. Por ejemplo, las variables que se declaran al inicio de un método se elaboran cuando se llama al método, y se desasignan cuando el método completa su ejecución. Son útiles para programas recursivos, ya que cada copia activa del subprograma tiene su propia versión de variables locales.

Variable dinámica explicita en el heap: Una variable dinámica explícita en el heap es una variable sin nombre la cual se asigna y desasigna explícitamente mediante instrucciones en tiempo de ejecución. Estas variables solo pueden referenciarse a través de punteros o referencias. Por ejemplo, en C++

```
int *intnode;
intnode = new int;
```

las siguientes declaraciones crean una variable dinámica explicita en el heap.

Estas variables son útiles para construir estructuras dinámicas como listas enlazadas o árboles, las cuales necesitan crecer o disminuirse durante la ejecución.

Variable dinámica implícita en el heap: Son las variables en las que se asocian al almacenamiento en el heap solo cuando se les asigna un valor. De hecho, todos sus atributos se asocian cada vez que se les asigna un valor. Su ventaja es que ofrecen el mayor grado de flexibilidad, permitiendo escribir código altamente genérico.

▼ En el contexto de los lenguajes de programación, ¿Cómo se define el 'alcance' de una variable?

a) La duración de la memoria asignada a la variable

b) El rango de operaciones posibles en la variable

c) El rango de instrucciones en las que la variable es visible

d) La compatibilidad de la variable con diferentes tipos de datos

▼ Defina static scoping y dynamic scoping. Cuáles son las reglas que se aplican en cada caso.

El alcance estático permite determinar el alcance de una variable de manera estática, es decir, antes de la ejecución, permitiendo a los lectores y compiladores determinar el tipo de cada variable simplemente examinando su código fuente. Esto se realiza buscando la variable referenciada en el subprograma actual, y si no se encuentra, la búsqueda se extiende a sus padres estáticos. Uno de sus problemas en

que en la mayoría de los casos permite más acceso a variables y a subprogramas del que es necesario.

El alcance dinámico se basa en la secuencia de llamadas de subprogramas, no en su relación espacial entre sí. Por lo tanto, se determina en tiempo de ejecución. Esto se logra buscando la variable referenciada en las declaraciones locales. Si no se encuentra, la búsqueda continúa en el padre dinámico de esa función, y así sucesivamente, hasta que se encuentra la declaración a la variable. El alcance dinámico hace que los programas sean más difíciles de leer y menos confiables, ya que una variable no local puede referirse a diferentes variables según la secuencia de llamadas. Además, no se pueden verificar los tipos de las variables no locales en tiempo de compilación.

▼ Qué es un entorno de referencia?

El entorno de referencia de una sentencia es el conjunto de variables visibles en esa sentencia. Este entorno se necesita al compilar la sentencia para manejar las referencias a variables no locales en tiempo de ejecución.

El entorno de referencia de una sentencia en un lenguaje estático consiste de las variables de su alcance local y las visibles en alcances ascendentes.

En lenguajes de alcance dinámico el entorno de referencia de una sentencia está compuesto por las variables declaradas localmente, más las variables de todos los subprogramas que estén actualmente activos.

▼ Explique detalladamente qué es el entorno de referencia de una sentencia, cómo se usa?

El entorno de referencia de una sentencia es el conjunto de variables que son visibles para esa sentencia. Se utiliza al momento de compilar una sentencia para manejar referencias a variables de otro alcance en tiempo de ejecución.

▼ En que consiste el entorno de referencia en un lenguaje de alcance estático?

En un lenguaje estático, el entorno de referencia de una sentencia consta de las variables locales y las variables en alcances estáticos, o sea, en sus ancestros estáticos.

▼ Porqué se requiere del entorno de referencia al momento de la compilación?

El entrono de referencia al momento de la compilación se necesita para manejar las referencias a variables de otro alcance en tiempo de ejecución.

▼ U6 - DATA TYPES

▼ ¿Qué es una coerción?

A) Error de tipo en un programa

B) Conversión explícita de un tipo de dato a otro

C) Conversión implícita de un tipo de dato a otro

D) Función especial en programación

▼ ¿Qué mecanismos se utilizan para almacenar enteros negativos en una computadora?

Notación signo-magnitud, en donde el bit de signo se establece para indicar negativo, y el resto de la cadena de bits representa el valor absoluto del número.

Notación complemento a dos, en donde la representación de un entero negativo se forma tomando el complemento lógico de la versión positiva del número y sumando uno.

Notación complemento a uno, el entero negativo se almacena como el complemento lógico de su valor absoluto. Desventaja de que tiene dos representaciones de cero.

▼ ¿Cuál es una ventaja de los tipos de datos decimales?

A) Ocupan menos memoria que los tipos enteros

B) Son más rápidos que los tipos enteros

C) Permiten una representación más precisa de números fraccionarios

D) No tienen desventajas

▼ Qué tipo de datos puede ser utilizado para representar de manera precisa el valor e (la base para el logaritmo natural) o el valor de pi? Presente una breve descripción del tipo de datos que corresponde.

No hay ningún tipo de dato con el que se pueda almacenar correctamente los valores e ni pi, ya que estos números tienen infinitos decimales, lo cual sería imposible guardar en las computadoras debido a la naturaleza finita de la memoria.

▼ ¿Qué lenguajes permiten subíndices negativos para acceder a elementos de un arreglo?

A) C++

B) Python

C) Java

D) C#

▼ Cuál es la diferencia entre un static array y un heap dynamic array?

En un arreglo estático, tanto el rango de subíndices como la asignación de almacenamiento son enlazados de manera estática, es decir, antes del tiempo te ejecución. Estos se mantienen fijos y no pueden cambiar durante la ejecución del programa.

En un arreglo dinámico en el montón, los rangos de subíndice y la asignación de almacenamiento son dinámicos y pueden cambiar cualquier cantidad de veces durante la vida útil del arreglo.

▼ Un array se puede almacenar en row-major-order como se hace en C++ o en column-major-order como ocurre en FORTRAN. Escriba una función de acceso para row-major-order.

a[i, j] = a[0, 0] + (i*n+j) * element_size

donde i es el número de filas por encima del elemento deseado, n el tamaño de la fila, y j el número de elementos a la izquierda de la

columna deseada

▼ ¿Cuál es el problema de diseño en arrays asociativos?

La única cuestión de diseño es cual es la forma de referenciar a sus elementos?

▼ Cite las cuestiones de diseño de los registros.

- Cual es la forma sintáctica de las referencias a los campos?
- Se permiten referencias elípticas?

▼ ¿Cuál es la principal diferencia entre un registro y una tupla?

A) Los registros tienen campos con nombres y las tuplas no

B) Las tuplas tienen campos con nombres y los registros no

C) Los registros pueden contener diferentes tipos de datos, pero las tuplas no

D) No hay diferencia

▼ Defina unión, unión libre y unión discriminada.

A) Unión: varios tipos en el mismo espacio de memoria, Unión libre: unión sin discriminador de tipo, Unión discriminada: unión con discriminador de tipo

B) Unión: unión con discriminador de tipo, Unión libre: unión sin discriminador de tipo, Unión discriminada: varios tipos en el mismo espacio de memoria

C) Unión: unión sin discriminador de tipo, Unión libre: varios tipos en el mismo espacio de memoria, Unión discriminada: unión con discriminador de tipo

D) Unión: varios tipos en el mismo espacio de memoria, Unión libre: unión con discriminador de tipo, Unión discriminada: unión sin discriminador de tipo

▼ ¿Cuáles son los dos problemas comunes con los punteros?

A) Punteros rápidos y punteros lentos

B) Punteros nulos y punteros colgantes

C) Punteros largos y punteros cortos

D) Punteros seguros y punteros inseguros

▼ Defina error de tipo.

A) Ocurre cuando se realiza una operación en un tipo de dato que es compatible con esa operación

B) Ocurre cuando se realiza una operación en un tipo de dato que no es compatible con esa operación

C) Ocurre cuando se utiliza un tipo de dato incorrecto en una declaración

D) Ocurre cuando se declara una variable con un tipo incorrecto

▼ Cite y explique detalladamente los dos enfoques existentes para la equivalencia de tipo.

La equivalencia de tipos por nombre significa que dos variables tienen tipos equivalentes si están definidas ya sean en la misma declaración o en declaraciones que usan el mismo nombre de tipo.

La equivalencia de tipos por estructura significa que dos variables tienen tipos equivalentes si sus tipos tienen estructuras idénticas.

▼ Defina named type equivalence y structure type equivalence

La equivalencia de tipos por nombre significa que dos variables tienen tipos equivalentes si están definidas ya sean en la misma declaración o en declaraciones que usan el mismo nombre de tipo.

La equivalencia de tipos por estructura significa que dos variables tienen tipos equivalentes si sus tipos tienen estructuras idénticas.

▼ U7 - EXPRESIONES Y SENTENCIAS DE ASIGNACIÓN

▼ Cite las 6 cuestiones de diseño en las expresiones aritméticas según Sebesta.

- Cuales son las reglas de precedencia de operadores?
- Cuales son las reglas de asociatividad a de operadores?
- Cual es el orden de evaluación de los operandos?
- Existen restricciones en los efectos secundarios en la evaluación de operandos?
- Permite el lenguaje la sobrecarga de operadores definida por el usuario?
- Qué mezcla de tipos esta permitida en las expresiones?

▼ Cuáles son las 6 cuestiones de diseño fundamentales de las expresiones aritméticas según Sebesta? Explique brevemente cada una de ellas.

- **Cuales son las reglas de precedencia de los operadores?** Las reglas de precedencia de los operadores definen el orden en el que se evalúan los operadores en una expresión, con base en la jerarquía de operadores que establezca el diseñador del lenguaje.
- **Cuales son las reglas de asociatividad de los operadores?** Las reglas de asociatividad de los operadores definen en que orden se evaluarán los operadores en una expresión en la que hayan dos operadores adyacentes con la misma precedencia.
- **Cual es el orden de evaluación de los operandos?** El orden de evaluación de los operandos es importante cuando alguno de los operandos de un operador tiene efectos secundarios, ya que que un operando se evalúe primero puede afectar el valor del otro.
- **Existen restricciones en los efectos secundarios de la evaluación de operandos?** Hay dos formas de restringir los efectos secundarios, uno, prohibiendo por completo los efectos secundarios funcionales, por ejemplo, prohibiendo el acceso a variables globales en funciones, y dos, imponiendo un orden de evaluación de los operandos, lo que puede limitar técnicas de optimización utilizadas por los compiladores, ya que algunas de estas implican reordenar las evaluaciones de los operandos.

- **Permite el lenguaje la sobrecarga de operadores definida por el usuario?** La sobrecarga de operadores significa que un un operador puede tener múltiples significados dependiendo del contexto, la sobrecarga de operadores puede beneficiar la legibilidad cuando se utiliza de manera sensata, pero cuando no puede ser bastante perjudicial para esta.
- **Que mezcla de tipos esta permitida en las expresiones?** Una expresión podría tener operandos de diferentes tipos, los lenguajes que permiten tales expresiones, deben definir restricciones y convenciones para las conversiones implícitas de los operados.

▼ ¿Cuál de las siguientes opciones mejor define la precedencia de operadores?

A) Orden en el que se ejecutan los programas

B) Orden en el que se evalúan los operadores en una expresión

C) Cantidad de operadores en una expresión

D) Velocidad de ejecución de los operadores

▼ Precedencia y asociatividad de operadores.

▼ Explica en al menos 120 palabras, qué es la transparencia referencial y cómo se relaciona la transparencia referencial con los efectos colaterales [side effects] de las funciones?

Un programa tiene la propiedad de transparencia referencial si cualquier dos expresiones en el programa que tengan el mismo valor pueden ser sustituidas entre sí sin afectar la acción del programa.

Su relación con los efectos secundarios se demuestra en el siguiente ejemplo:

```
result1 = fun(a) + b;
```

```
temp = fun(a);
```

```
result2 = temp + b;
```

Si fun no tiene efectos secundarios, entonces result1 y result2 serán iguales, ya que las expresiones asignadas a ellas son equivalente. Sin embargo, si fun tiene el efecto secundario de sumar 1 a b, entonces

result1 no sería igual a result2. Así que ese efecto secundario viola la transparencia referencial del programa.

▼ Side-effects de una función. Ejemplo.

Un efecto secundario de una función ocurre cuando la función cambia uno de sus parámetros o una variable global, por ejemplo

```
int fun(a)
    a = a + 1
    return 5
int main
    a = 5
    result = a + fun(a)
```

el siguiente código proporcionaría valores distintos dependiendo del orden de evaluación de los operadores, por ejemplo, si se evalúa de izquierda a derecha, la variable result tendría un valor de 10, pero si se evaluará de derecha a izquierda, tendría un valor de 11

▼ ¿Qué es la transparencia referencial?

A) Propiedad de las funciones que dependen del estado del programa

B) Propiedad de las funciones que siempre producen el mismo resultado para los mismos argumentos

C) Capacidad de una función para cambiar su entorno

D) Tipo especial de variable

▼ Qué es la transparencia referencial y cómo se relaciona con los efectos colaterales? (Referential Transparency and side Effects)

Un programa tiene la propiedad de transparencia referencial si cualquier dos expresiones en el programa que tengan el mismo valor pueden ser sustituidas entre sí sin afectar la acción del programa.

Su relación con los efectos secundarios se demuestra en el siguiente ejemplo:

```
result1 = fun(a) + b;
```

```
temp = fun(a);
```

```
result2 = temp + b;
```

Si fun no tiene efectos secundarios, entonces result1 y result2 serán iguales, ya que las expresiones asignadas a ellas son equivalente. Sin embargo, si fun tiene el efecto secundario de sumar 1 a b, entonces result1 no sería igual a result2. Así que ese efecto secundario viola la transparencia referencial del programa.

▼ Define conversión narrowing y conversión widening. Mencione un ejemplo de cada conversión.

una conversión de reducción (narrowing) convierte un valor a un tipo cuyo rango de valores posibles es más pequeño. Por ejemplo, convertir un double a un float es una conversión de reducción, ya que el rango de double es mucho mayor que el de float.

Una conversión de ampliación (widening) convierte un valor a un tipo que puede incluir al menos aproximaciones de todos los valores del tipo original. Por ejemplo, convertir un int a un float.

▼ ¿Qué es una coerción?

A) Error de tipo en un programa

B) Conversión explícita de un tipo de dato a otro

C) Conversión implícita de un tipo de dato a otro

D) Función especial en programación

▼ ¿Qué es un cast?

A) Conversión implícita de un tipo de dato a otro

B) Conversión explícita de un tipo de dato a otro realizada por el programador

C) Error de tipo en un programa

D) Función especial en programación

▼ Por qué los diseñadores de los lenguajes no están de acuerdo con respecto a la coerción en expresiones aritméticas?

Aquellos diseñadores de los lenguajes que se oponen a un amplio rango de coerciones están preocupados por los problemas de confiabilidad que pueden surgir de estas coerciones, ya que reducen los beneficios de la comprobación de tipos.

▼ ¿Crees que la eliminación de operadores sobrecargados en tu lenguaje de programación favorito sería beneficiosa? ¿Por qué sí o por qué no?

▼ Presenta tus propios argumentos a favor y en contra de permitir expresiones aritméticas de modo mixto.

▼ U8 - ESTRUCTURAS DE CONTROL A NIVEL DE SENTENCIA

▼ Cuáles son las únicas estructuras de control necesarias?

Aunque solamente una declaración de control, un goto seleccionable, es mínimamente suficiente, según Böhm y Jacopini se necesitan al menos dos mecanismos adicionales para que las computaciones en los programas sean flexibles y potentes: una para elegir dos caminos de flujo de control y otra para iteraciones controladas lógicamente.

▼ Cuáles son tres consideraciones de diseño relacionadas a los Selectores de dos vías según Sebesta?

- Cual es la forma y el tipo de la expresión que controla la selección?
- Como se especifican las clausulas then y else?
- Como debe especificarse el significado de los selectores anidados?

▼ Cite las cuestiones de diseño de los selectores de 2 vías.

- Cual es la forma y el tipo de la expresión que controla la selección?
- Como se especifican las clausulas then y else?
- Como debe especificarse el significado de los selectores anidados?

▼ ¿Cuál es la forma general de un selector de dos vías?

if *expresion_de_control*

then *clausula*

else *clausula*

▼ Cuáles son cinco consideraciones de diseño relacionadas a los selectores múltiples según Sebesta?

- Cual es la forma y el tipo de la expresión que controla la selección?
- Cómo se especifican los segmentos seleccionables?
- Está restringido el flujo de ejecución para incluir solo un único segmento seleccionable?
- Como se especifican los valores de los casos?
- Como deben manejarse, si es que deben de hacerlo, los valores de expresión de selector no representados?

▼ ¿Qué mecanismo sigue un compilador cuando el número de casos en una declaración de selección es 10 o más para optimizar el tiempo requerido de ejecución?

El compilador construye una tabla hash de las etiquetas de los segmentos, lo que resultaría en un tiempo aproximadamente igual y corto de elegir cualquiera de los segmentos seleccionables. Si el lenguaje permite rangos de valores para las expresiones de los casos, como en Ruby, una tabla hash no es lo adecuado. En este caso es mejor realizar una búsqueda binaria de valores de casos y direcciones de segmentos.

▼ Cuáles son las categorías en las que pueden clasificarse las sentencias de control de iteraciones? Cuales son las preguntas que sirven para definir estas clasificaciones?

Las principales categorías en las que pueden clasificarse las sentencias de control por iteraciones son los bucles controlados por contador y los bucles controlados lógicamente.

Las dos básicas de diseño que sirven para definir estas categorías son:

- Como se controla la iteración?
- Donde debe aparecer el mecanismo de control en la iteración?

▼ Cuáles son tres consideraciones de diseño relacionadas a las iteraciones controladas por contador en Sebesta?

- Cual es el tipo y el alcance de la variable de bucle?
- Debería ser legal que la variable de bucle o los parámetros del bucle sean modificados dentro del bucle, y de ser así, el cambio afecta al control del bucle?
- Debe evaluarse los parámetros de bucle solo una vez o una vez por cada iteración?

▼ ¿En qué sentido es la declaración `for` de C más flexible que en muchos otros lenguajes?

A) C permite múltiples declaraciones en la inicialización.

B) C solo permite enteros en la declaración `for`.

C) C permite cualquier expresión en las secciones de inicialización, condición y actualización.

D) C no permite bucles `for`.

E) C permite cualquier expresión en las secciones de inicialización, condición y actualización.

▼ ¿Cómo un lenguaje funcional implementa la repetición?

Ya que los lenguajes funcionales no tienen variables, lo cual es importante en los lenguajes imperativos para los bucles controlados por contador, controlan la repetición con recursión.

Los bucles con contador pueden ser simulados de la siguiente manera:

El contador puede ser un parámetro de una función que ejecuta repetidamente el cuerpo del bucle, el cual puede estar especificado en una segunda función enviada a la función de bucle como un parámetro. Así, una función de bucle toma la función del cuerpo y el número de repeticiones como parámetros.

Un ejemplo de un bucle controlado por contador en F#:

```
let rec forLoop loopBody reps =
    if reps <= 0 then
        ()
    else:
        loopBody()
        forLoop loopBody, (reps-1);;
```

▼ ¿Cuáles son los problemas de diseño para las declaraciones de bucle controladas lógicamente?

- Debe el control ser de preevaluación o postevaluación?
- Debe el bucle controlado lógicamente ser una forma especial de un bucle con contador o una instrucción separada?

▼ ¿Qué ventaja tiene la declaración `break` de Java sobre la declaración `break` de C?

A) Java no tiene declaración `break`.

B) Java permite salir de bloques anidados específicos con etiquetas.

C) Java no permite usar `break` en bloques de `switch`.

D) No hay diferencia entre Java y C en cuanto a la declaración `break`.

▼ Seleccione la opción que describe correctamente tres situaciones en las que se necesita una declaración de bucle combinada de conteo y lógica.

A) 1) Cuando necesitas repetir una acción un número específico de veces, pero también necesitas una condición específica para continuar. 2) Cuando estás procesando elementos de una lista y necesitas parar cuando encuentres un valor específico. 3) Cuando estás esperando a que un recurso esté disponible, pero solo quieres intentarlo por un tiempo limitado.

B) 1) Cuando necesitas repetir una acción un número infinito de veces. 2) Cuando solo necesitas una declaración de bucle basada en una condición lógica. 3) Cuando quieres iterar a través de todos los elementos de una colección sin ninguna condición específica.

C) 1) Cuando estás realizando una búsqueda y quieres detenerte cuando encuentres el elemento o después de un cierto número de intentos. 2) Cuando estás intentando conectarte a un servidor y quieres reintentar un número específico de veces antes de rendirte. 3) Cuando estás validando entrada del usuario y quieres darle un número limitado de intentos para ingresar datos válidos.

D) 1) Cuando solo necesitas un bucle que se ejecute un número fijo de veces. 2) Cuando estás realizando una operación que no requiere condiciones de parada. 3) Cuando estás realizando cálculos matemáticos que no involucran iteraciones.

▼ Mencione un uso común inadecuado de los resultados de Böhm y Jacopini.

Un uso indebido del resultado de Böhm y Jacopini es argumentar en contra de la inclusión de otras estructuras de control más allá de las selecciones y los bucles lógicos con preevaluación. Ningún lenguaje de uso general ha dado ese paso todavía, esto debido al impacto negativo en la capacidad de escritura y legibilidad que esto conllevaría. Los programas escritos solo con selecciones y bucles lógicos con preevaluación generalmente tienen una escritura menos natural, son más complejos y, por lo tanto, más difíciles de escribir y leer.

▼ U9 - SUBPROGRAMAS

▼ ¿Cuáles son las 3 características generales de los subprogramas?

Cada subprograma tiene un punto único de entrada.

La unidad de programa que llamó, se mantendrá suspendida hasta que el subprograma llamado terminé, por lo que solo habrá un subprograma en ejecución en un momento dado.

El control siempre regresa a quien hizo la llamada cuando el subprograma termina.

▼ ¿Qué es una definición de subprograma?

Una definición de subprograma describe la interfaz y las acciones de la abstracción del subprograma.

▼ Cuáles son las tres características de una subrutina? Cuáles son 11 consideraciones de diseño principales de las subrutinas?

Características

Los subprogramas deben tener un único punto de entrada.

Cuando un programa llame a un subprograma, este programa debe quedar suspendido hasta que el subprograma termine su ejecución, por lo que habrá solo un subprograma en ejecución en un momento dado.

El control siempre regresa a quien hizo la llamada cuando el subprograma termina.

Cuestiones de Diseño

- Qué método o métodos de paso de parámetros se utilizan?
- Se verifican los tipos de los parámetros reales con respecto a los tipos de los parámetros formales?
- Las variables locales se asignan de manera estática o dinámica?
- Pueden aparecer definiciones de subprogramas dentro de otras definiciones de subprogramas?
- Si los subprogramas pueden pasarse como parámetros y pueden anidarse, cuál es el entorno de referencia de un subprograma pasado como parámetro?
- Se permiten efectos secundarios funcionales?
- Qué tipos de valores pueden devolver las funciones?
- Cuántos valores pueden devolver las funciones?
- Pueden sobrecargarse los subprogramas?
- Pueden ser genéricos los subprogramas?
- Si el lenguaje permite subprogramas anidados, se admiten closures?

▼ Cite y explique brevemente los 5 tipos de paso de parámetros? Explica en al menos 50 palabras en cada caso.

Paso por Valor:

Paso por Resultado:

Paso por Valor-Resultado:

Paso por Referencia:

Paso por Nombre:

▼ Describe correctamente el problema de pasar arrays multidimensionales como parámetros.

La función de mapeo de almacenamiento de una matriz dada con elementos de tamaño 1 es la siguiente:

```
address[i, j] = address[0, 0] + i*n + j
```

Con n siendo el tamaño de fila, o sea, el número de columnas. Como está función debe construirse antes de que el subprograma sea llamado, debe conocer el número de columnas al momento de compilar el subprograma. Para esto, el número de columnas debe incluirse en el parámetro formal.

El problema con esto es que no permite que un programador escriba una función que acepte matrices con diferentes números de columnas, reduciendo así la flexibilidad.

▼ Cuáles son las complicaciones que existen con el paso de parámetros que son subprogramas? Cuáles son las opciones que se han desarrollado para lidiar con la cuestión de la determinación del entorno en referencia, explique brevemente. Explica en al menos 50 palabras.

Para el paso de parámetros que son subprogramas, surgen dos complicaciones:

Primero, la verificación de tipos de los parámetros del subprograma que se pasó como parámetro. En C y C++, las funciones no se pueden pasar directamente como parámetros, pero si se pueden pasar punteros a funciones. El tipo de puntero a función incluye el protocolo de la función. Debido a que el protocolo incluye todos los tipos de los parámetros, estos parámetros pueden ser verificados en cuanto a tipos.

La segunda complicación se da cuando hay subprogramas anidados, y es cual es el entorno de referencia que debe usarse para ejecutar el subprograma pasado. Para esto hay tres opciones:

Usar el entorno de la instrucción de llamada al subprograma pasado (shallow binding)

Usar el entorno de la definición del subprograma pasado (deep binding)

Usar el entorno de la instrucción que pasó el subprograma como parámetro real (ad hoc binding)

▼ Defina shallow y deep binding para entornos de referencia de subprogramas que han sido pasados como parámetros. Explique detalladamente.

Shallow binding se refiere a utilizar el entorno de referencia de la instrucción de llamada al subprograma pasado como parámetro.

Deep binding se refiere a usar el entorno de referencia de la definición del subprograma pasado.

▼ ¿Qué es el enlace ad hoc?

Ad hoc binding es una solución al problema de cual entorno de referencia debe usarse para un subprograma pasado como parámetro. En programas con ad hoc binding, el entorno utilizado por un subprograma que fue pasado como parámetro es el entorno de la llamada que pasó el subprograma como un parámetro

▼ Qué es polimorfismo de subtipo?

El polimorfismo de subtipo significa que una variable de tipo T puede acceder a cualquier objeto de tipo T o de cualquier tipo derivado de T.

▼ U10 - IMPLEMENTACIÓN DE SUBPROGRAMAS

▼ Los pasos que se siguen al ser llamado un procedimiento.

En un subprograma simple, es decir, en el que los su programas no pueden anidarse y todas las variables son locales estáticas, los pasos al realizar una llamada son:

1. Guardar el estado de ejecución del programa actual.

2. Calcular y pasar los parámetros.
3. Pasar la dirección de retorno al subprograma llamado.
4. Transferir el control al subprograma llamado.

Pasos a realizar en la llamada de un subprograma en un lenguaje con variables locales dinámicas en pila

Programa que llama

1. Crear una instancia del registro de activación
2. Guardar el estado de ejecución del programa actual
3. Calcular y pasar los parámetros
4. Pasar la dirección de retorno al subprograma llamado
5. Transferir el control al subprograma llamado

Subprograma llamado

1. Guardar el antiguo EP en la pila como el enlace dinámico y crear el nuevo EP
2. Asignar las variables locales

▼ Los pasos que se siguen cuando el procedimiento sale de escena.

Los pasos que se siguen cuando un subprograma simple sale de la escena son los siguientes:

1. Si hay parámetros de modo salida o entrada salida, se copian los valores actuales de los parámetros formales a los parámetros reales
2. Si el subprograma es una función, el valor funcional se coloca en un lugar accesible para el programa que hizo la llamada.
3. Se restaura el estado de ejecución del programa que llama.
4. El control se transfiere de vuelta al programa que llama.

Lenguajes con variables locales dinámicas en pila

Subprograma llamado

1. Si hay parámetros en modo de salida o de valor resultado, mover los valores de los parámetros formales a los parámetros reales.
2. Si el subprograma es una función, colocar el valor funcional en un lugar accesible para el subprograma que llamo
3. Restaurar el puntero de la pila configurándolo al valor del EP actual menos uno y configurar el EP al antiguo enlace dinámico
4. Restaurar el estado de ejecución del programa que llama
5. Transferir el control de vuelta al programa que llama

▼ ¿Qué debe almacenarse para la vinculación (linkage) a un subprograma?

Para la vinculación de un subprograma debe almacenarse:

1. Información sobre el estado de ejecución del programa que llama.
2. Parámetros.
3. Valores devueltos por funciones.
4. Dirección de retorno.
5. Variables temporales utilizadas por los subprograma.

▼ ¿Cuál es la diferencia entre un registro de activación y una instancia de registro de activación?

Un registro de activación es el formato de la parte no relacionada con el código de un subprograma, la cual describe datos que son relevantes solo durante la activación o ejecución del subprograma. Una instancia de registro de activación es un ejemplo concreto de un registro de activación, el cual se crea cada que un subprograma es llamado.

▼ Dibuje un registro de activación. Describa brevemente sus partes

Subprograma Simple

Variables Locales	Variables temporales utilizadas dentro del subprograma llamado

Parámetros	Son los valores o direcciones proporcionados por el programa que llama
Dirección de Retorno	puntero a la instrucción que sigue a llamada

Lenguajes con variables locales dinámicas en pila

Variables Locales	Variables temporales utilizadas dentro del subprograma llamado
Parámetros	Son los valores o direcciones proporcionados por el programa que llama
Enlace Dinámico	Puntero a la base de la instancia de la instancia del registro de activación del que llama
Dirección de Retorno	puntero a la instrucción que sigue a llamada

▼ ¿Cómo se representan las referencias a las variables en el método static-chain?

Las referencias a variables no locales en el método static-chain puede ser representada por un par de ordenado de enteros (chain_offset, local_offset), donde chain offset es el número de enlaces a la instancia correcta de registro de activación y local offset es desplazamiento necesario para llegar a la variable desde el inicio del registro de activación.

▼ Describe correctamente el método shallow-access para implementar el alcance dinámico.

▼ U14 - EXCEPTION HANDLING and EVENT HANDLING

▼ Cuestiones de diseño para el manejo de excepciones.

- Cómo y donde se especifican los manejadores de excepciones y cuál es su alcance?
- Cómo se vincula una ocurrencia a un manejador de excepciones?
- Se puede pasar información sobre una excepción al manejador?
- Dónde continúa la ejecución, si es que continúa, después de que un manejador de excepciones completa su ejecución?

- Se proporciona alguna forma de finalización?
- Cómo se especifican las excepciones definidas por el usuario?
- Si hay excepciones predefinidas, deberían existir manejadores de excepciones predeterminados para los programas que no proporcionen los suyos?
- Pueden levantarse explícitamente las excepciones predefinidas?
- Se tratan los errores detectables por hardware como excepciones que pueden ser manejadas?
- Hay alguna excepción predefinida?

▼ Louden U7 - AMBIENTES DE EJECUCIÓN

▼ Dibuje la organización del almacenamiento en tiempo de ejecución. Describa brevemente sus partes.

área de código	
área global/estatica	
pila	
espacio libre	
heap	

▼ Describa un ambiente de ejecución basado en pila sin procedimientos locales.