ALGORITMOS 2.

GUIA DE ORDENACION DE ARCHIVOS

Mayo 20 de 2024.

Code complete de Steve McConnell

Recomendaciones de los capitulos

- Capitulo 4
- Capitulo 7
- Capitulo 8
- Capitulo 10
- Capitulo 11
- Capitulo 12
- Capitulo 13
- Capitulo 14
- Capitulo 15
- Capitulo 16
- Capitulo 17

Recomendaciones capitulo 4

El capítulo 4 de Code Complete profundiza en las decisiones críticas que impactan el éxito de un proyecto de software. Estas decisiones, tomadas al inicio del desarrollo, determinan la calidad, el rendimiento, la mantenibilidad y la escalabilidad del software final.

Decisiones clave:

- Elección del lenguaje de programación:
    - Características del lenguaje: Considerar sintaxis, manejo de memoria, concurrencia y otros aspectos relevantes.
    - Disponibilidad de bibliotecas y frameworks: Evaluar si existen recursos que satisfagan las necesidades funcionales del proyecto.
- Elección de la arquitectura de software:
    - Monolítica: Un único código base grande y complejo.
    - Modular: Código dividido en módulos independientes y reutilizables.
    - Microservicios: Servicios independientes y escalables que se comunican entre sí.
    - Orientada a eventos: Los componentes interactúan a través de eventos.

Factores a considerar:

- Requisitos del proyecto: Complejidad, necesidades de escalabilidad, entre otros.
- Escalabilidad: Capacidad del software para crecer y manejar más usuarios o datos.
- Mantenibilidad: Facilidad de modificar y actualizar el código.
- Desempeño: Velocidad y eficiencia del software.

Recomendaciones:

- Consultar con expertos: Buscar la opinión de profesionales con experiencia en desarrollo de software.
- Mantenerse actualizado: Estar al día con las últimas tendencias tecnológicas para tomar decisiones acertadas.

- Las decisiones tomadas al inicio del desarrollo de software son cruciales para el éxito del proyecto. Elegir el lenguaje de programación y la arquitectura adecuados, considerando los factores mencionados, es fundamental para crear software de alta calidad, escalable, mantenible y con un buen rendimiento. No dudes en consultar con expertos y mantenerte actualizado en las últimas tecnologías para tomar decisiones informadas que conduzcan al éxito del proyecto.

Recomendaciones del capitulo 7

Las rutinas de software de alta calidad son esenciales para el desarrollo de software exitoso. Estas rutinas son fáciles de entender, mantener y usar, y son menos propensas a errores.

Nombres descriptivos:

- Variables y funciones con nombres claros y precisos.
- Comentarios concisos: Explicando el propósito del código de manera sencilla.
-Formato adecuado: Indentación y organización del código para mejorar la legibilidad.
-Estructura lógica: Código organizado en secciones y bloques con un flujo claro.
-Evitar redundancia: Eliminar código repetitivo y usar técnicas como bucles y funciones.
-Eliminar código innecesario: Quitar código muerto o que no aporta valor.
-Subrutinas para tareas repetitivas: Encapsular tareas repetitivas en funciones reutilizables.

Reutilización:

- Funciones genéricas: Escribir funciones que puedan ser usadas con diferentes tipos de datos.
- Encapsulación en clases: Agrupar funcionalidades relacionadas en clases para mejorar la modularidad.
- Interfaces y abstracciones: Definir interfaces y abstracciones para promover la reutilización del código
- Refactorización: Refactorice el código periódicamente para mejorar su claridad, concisión y mantenibilidad

Escribir rutinas de alta calidad es una habilidad fundamental para los desarrolladores de software. Al seguir estas características y prácticas, los estudiantes de Ingeniería de Sistemas pueden crear código robusto, eficiente y fácil de mantener, lo que contribuye al éxito de los proyectos de software.

Recomendaciones del capitulo 8

La programación defensiva es un conjunto de técnicas y prácticas para escribir código robusto y resistente a errores.
Su objetivo es anticipar y manejar condiciones inesperadas, evitando que estas provoquen fallos en el software.

Principios de la Programación Defensiva:

- Esperar lo peor: Asumir que las entradas del usuario, datos externos y condiciones del sistema pueden ser incorrectas o inesperadas.
- Validar entradas: Verificar que las entradas del usuario y los datos externos sean válidos y estén dentro de los rangos esperados.
- Manejar errores de manera adecuada: No ignorar los errores, sino tratarlos de manera controlada y proporcionar información útil al usuario o al desarrollador.
- Hacer que el código sea resistente a fallas: Implementar mecanismos de seguridad para evitar que los errores causen fallos catastróficos.
- Documentar las suposiciones: Registrar las suposiciones hechas sobre el comportamiento del software y las condiciones externas.

Recomendaciones del capitulo 10

Elegir el tipo de variable correcto es fundamental para escribir código claro, eficiente y mantenible. Este capítulo explora los diferentes tipos de variables disponibles en la mayoría de los lenguajes de programación, con sus características y aplicaciones:

Factores a considerar:

- Alcance: Limitar el acceso a los datos.
  - Local: Alcance limitado a un bloque de código.
  - Estática: Alcance a toda la función/método.
  - Global: Alcance a todo el programa.
  - De clase: Alcance a todas las instancias de la clase.
- Duración: Mantener el valor entre llamadas o instancias
  - Mutabilidad: Permitir modificar el valor.
  - Local: Se puede modificar dentro del bloque de código.
  - Estática: Se puede modificar en toda la función/método.
  - Global: Se puede modificar en todo el programa (usar con precaución).
  - De instancia: Se puede modificar para cada instancia de la clase.
  - De clase: Se puede modificar para toda la clase (usar con precaución).

Recomendaciones:

- Utilizar variables locales tanto como sea posible.
- Evitar variables globales a menos que sea necesario.
- Documentar el uso de variables estáticas, de instancia y de clase.
- Elegir nombres descriptivos para las variables.
- Utilizar herramientas de análisis estático para identificar errores.

Al seguir estas recomendaciones, podremos elegir el tipo de variable adecuado para cada caso, mejorando la claridad, eficiencia y mantenibilidad de su código.

Recomendaciones del capitulo 11

La elección del tipo de variable adecuado es crucial para escribir código claro, eficiente y mantenible. Este capítulo explora los diferentes tipos de variables disponibles en la mayoría de los lenguajes de programación, con sus características y aplicaciones:

Principios básicos:

- Utilizar nombres significativos: Relacionados con la función o el propósito de la variable.
- Evitar nombres ambiguos o confusos: Evitar interpretaciones múltiples o confusiones.
- Convenciones de nomenclatura:
- Unificar el estilo en todo el proyecto.

Ejemplos:

- CamelCase: `variableNombreUsuario`
- snake_case: `variable_nombre_usuario`
- PascalCase: `VariableNombreUsuario`

Herramientas y documentación:

- Utilizar herramientas de análisis estático para identificar nombres no descriptivos o inconsistentes.
- Documentar las convenciones de nomenclatura utilizadas en el proyecto.
- Beneficios:
- Mejora la calidad, legibilidad y mantenibilidad del código.
- Contribuye al éxito de los proyectos de software.

Al seguir estas recomendaciones, podemos desarrollar habilidades sólidas para nombrar variables, lo que les permitirá escribir código más profesional y efectivo.

Recomendaciones del capitulo 12

Alcance de las Variables

El alcance de las variables es un concepto fundamental en la programación que define la visibilidad de una variable dentro del código, comprender el alcance de las variables es esencial para escribir código claro, organizado y libre de errores

- Utilizar el alcance de bloque tanto como sea posible para limitar el acceso a las variables y mejorar la modularidad.
- Evitar el uso de variables globales a menos que sea absolutamente necesario.
- Elegir nombres descriptivos para las variables y utilizar convenciones de nomenclatura consistentes para evitar confusiones de nombres.
- Documentar el uso de variables con alcance de archivo para facilitar su comprensión.
- Utilizar herramientas de análisis estático para identificar posibles errores relacionados con el alcance de las variables.
- Al seguir estas recomendaciones, podemos comprender y utilizar correctamente el alcance de las variables, mejorando la calidad, organización y seguridad de su código, lo que contribuye a un desarrollo de software exitoso.

Recomendaciones del capitulo 13

Variables Compuestas

Las variables compuestas son una herramienta poderosa en la programación que permite almacenar múltiples valores en una sola unidad. Este capítulo nos muestra los diferentes tipos de variables compuestas y sus aplicaciones

- Elegir la estructura adecuada: Seleccionar el tipo de variable compuesta que mejor se adapte a los datos que se almacenarán. Considerar el tipo de datos, la cantidad de elementos y la necesidad de acceso individual o por grupos.
- Inicializar las variables compuestas: Asignar valores iniciales a los elementos de la variable compuesta al declararla o utilizando asignaciones explícitas.
- Acceder a los elementos de las variables compuestas: Utilizar índices numéricos para acceder a elementos de arreglos o nombres de miembros para acceder a elementos de estructuras y uniones.
- Documentar el uso de variables compuestas: Explicar el propósito de la variable compuesta, el tipo de datos que almacena y cómo se accede a sus elementos.
- Evitar el uso de arreglos de gran tamaño si no es necesario, ya que pueden consumir memoria de manera ineficiente.
- Considerar el uso de listas dinámicas o vectores para almacenar colecciones de datos que pueden cambiar de tamaño durante la ejecución del programa.
- Utilizar funciones y métodos para encapsular el acceso y la manipulación de variables compuestas, mejorando la modularidad y la legibilidad del código.
- Elegir nombres de variables compuestas descriptivos y significativos para facilitar su comprensión.

Recomendaciones del capitulo 14

Diseño de Funciones

- El diseño de funciones es un aspecto fundamental de la programación que impacta directamente en la calidad, mantenibilidad y legibilidad del código. Este capítulo explora los principios clave para diseñar funciones efectivas
- Planificar las funciones antes de escribirlas: Definir el propósito de cada función, sus entradas, salidas y comportamiento.
- Utilizar nombres descriptivos para las funciones: Los nombres deben reflejar claramente la tarea que realiza la función.
- Documentar las funciones: Explicar el propósito de la función, sus parámetros, valores de retorno y posibles excepciones.
- Evitar funciones demasiado grandes: Dividir funciones grandes en funciones más pequeñas y manejables.
- Utilizar parámetros para evitar variables globales: Pasar datos entre funciones mediante parámetros en lugar de usar variables globales.
- Considerar el uso de funciones recursivas para resolver problemas de manera elegante y eficiente

Teniendo en cuenta las recomendaciones estaremos listos para diseñar funciones efectivas que mejoren la calidad, mantenibilidad y legibilidad de su código, lo que contribuye al éxito de los proyectos de software

Recomendaciones del capitulo 15

Este capítulo explora en profundidad la implementación de funciones, un aspecto fundamental de la programación que permite modularizar el código y organizar la lógica de manera eficiente

- Implementación de Funciones
- Seguir las convenciones de nomenclatura y estilo de código para las funciones.
- Documentar las funciones, incluyendo su propósito, parámetros, valores de retorno y posibles excepciones.
- Escribir funciones pequeñas y manejables, evitando funciones demasiado grandes o complejas.
- Utilizar comentarios para explicar el código dentro de las funciones.

Recomendaciones del capitulo 16

Depuración de Funciones

La depuración es una habilidad fundamental para los programadores, ya que permite identificar y corregir errores en el código. Este capítulo explora técnicas efectivas para depurar funciones y solucionar problemas de código

- Desarrollar una metodología de depuración sistemática.
- Comenzar con las pruebas unitarias para identificar errores tempranos.
- Utilizar las herramientas de depuración disponibles en el entorno de desarrollo.
- Imprimir mensajes de diagnóstico para rastrear el flujo de ejecución del programa.
- Validar entradas y salidas de funciones.
- Manejar errores y excepciones de forma adecuada.
- Practicar la depuración regularmente para mejorar las habilidades.
- Ser paciente y persistente al depurar código

Con estas recomendaciones claras deduciremos que ya podemos desarrollar habilidades de depuración efectivas que nos permitirán identificar y corregir errores en sus funciones y mejorar la calidad de su código, contribuyendo al éxito de los proyectos de software

Recomendaciones del capitulo 17

Las funciones especiales ofrecen funcionalidades adicionales para escribir código más elegante, eficiente y reutilizable. Este capítulo explora cuatro tipos de funciones especiales:

1. Funciones recursivas:

- Permiten resolver problemas de forma recursiva, dividiendo el problema en subproblemas más pequeños del mismo tipo.
- Recomendación: Utilizarlas para problemas que se descomponen naturalmente en subproblemas similares.
- Ejemplo: Cálculo del factorial de un número.

2. Funciones en línea:

- Funciones pequeñas que se definen y ejecutan dentro de una expresión.
- Recomendación: Utilizarlas para simplificar expresiones cortas y mejorar la legibilidad.
- Ejemplo: Ordenar una lista de números usando una función en línea.

3. Funciones de plantilla:

- Funciones genéricas que pueden trabajar con diferentes tipos de datos sin necesidad de reescribir el código.
- Recomendación: Utilizarlas para evitar código repetitivo y mejorar la reutilización.
- Ejemplo: Función para intercambiar dos valores de cualquier tipo.

4. Funciones lambda:

- Funciones anónimas que se definen sin un nombre y se utilizan en contextos específicos.
- Recomendación: Utilizarlas para encapsular código corto en contextos específicos, como callbacks o expresiones lambda.
- Ejemplo: Ordenar una lista de números usando una función lambda.

5. Funciones de orden superior:

- Funciones que reciben otras funciones como argumentos o devuelven funciones como resultado.
- Recomendación: Utilizarlas para crear algoritmos más flexibles y reutilizables.
- Ejemplo: Función que aplica una función a cada elemento de una lista.

Recomendaciones generales:

Comprender el funcionamiento de cada tipo de función especial antes de usarla.

Elegir el tipo de función adecuado para cada caso específico.

No abusar de las funciones especiales, ya que un uso excesivo puede dificultar la lectura del código.

Documentar el uso de funciones especiales para mejorar la comprensión del código.

Con estos puntos ya claros podremos aprovechar las ventajas de los tipos especiales de funciones para escribir código más efectivo, elegante y reutilizable, mejorando la calidad de sus proyectos de software.

Integrantes

- Juan Jose Arias Murillo
- Jhonny Benitez Caro
- Sebastian Zapata Naranjo
- Jean Paul Ortiz Restrepo

UCO
Universidad Católica de Oriente
Vigilada Mineducación

Renovación Acreditación Institucional en ALTA CALIDAD

A la **Verdad,**
por la **fe**
y la **ciencia**

#TransformandoVidas

¡Gracias!