

# CONTENIDO

PROLOGO por David M. Moore, Director de Desarrollo de Microsoft	xiii
PREFACIO _____	xix
INTRODUCCION _____	xxiii

Con el crecimiento en complejidad del software y el ascenso del porcentaje de errores que lleva asociado, se hace cada vez más necesario que los programadores produzcan código sin errores mucho más pronto en el ciclo de desarrollo, antes de que el código se envíe al grupo de pruebas. La clave para escribir código sin errores es conocer mejor la forma en la que aparecen los errores. Los programadores pueden cultivar este conocimiento haciéndose a sí mismos dos preguntas simples sobre cada uno de los errores que encuentren: “¿Cómo podría preveer este error?” y “¿Cómo podría haber detectado automáticamente este error?” Las normas de este libro son el resultado de hacerse regulamente estas dos preguntas durante varios años.

CAPITULO UNO. UN COMPILADOR IMAGINARIO _____	1
--	---

Si su compilador pudiese detectar cualquier error de su programa, sin importar de qué tipo fuera, y le diese un mensaje de error, librar su código de los errores sería fácil. No existe este compilador omnisciente, pero habilitando las opciones de advertencia de su compilador, utilizando las comprobaciones de sintaxis y portabilidad, y empleando las pruebas de unidad automáticas, puede incrementar el número de errores que se detecten automáticamente.

CAPITULO DOS. VALIDE SU PROPIO CODIGO _____	11
---	----

Una buena estrategia de desarrollo es mantener dos versiones de su programa: una para la comercialización y otra que se utiliza para depurar el código. Utilizando instrucciones de validación para la depuración, puede detectar los errores provocados por argumentos erróneos, utilizaciones accidentales de comportamientos indefinidos, suposiciones erróneas hechas por otros programadores y condiciones imposibles que sin embargo aparecen de algún modo. Los algoritmos de respaldo para depuración sólo ayudarán a verificar los resultados de las funciones y de los algoritmos usados por ellas.

**CAPITULO TRES. PROTEJA SUS SUBSISTEMAS \_\_\_\_\_ 41**

Las validaciones esperan calladamente hasta que aparecen los errores. Las comprobaciones de integridad de los subsistemas son incluso más poderosas, verifican los subsistemas activamente y avisan de los errores antes de que afecten al programa. Las comprobaciones de integridad para el gestor de memoria estándar del C pueden detectar punteros colgados, bloques de memoria perdidos, y la utilización de memoria que no ha sido inicializada o que ya se ha liberado. Las comprobaciones de integridad también pueden emplearse para eliminar comportamientos extraños, que son responsables de situaciones no probadas, y a forzar a que los errores del subsistema sean reproducidos para que puedan ser localizados y corregidos.

**CAPITULO CUATRO. HAGA UN SEGUIMIENTO DEL CODIGO \_\_\_ 67**

La mejor forma de encontrar los errores es hacer un seguimiento paso a paso de todo el código nuevo con el depurador. Haciendo el seguimiento de cada instrucción y centrándose en el flujo de datos, puede detectar rápidamente los problemas de las expresiones y de los algoritmos. Centrando su atención en los datos y no en las instrucciones, obtendrá una visión muy diferente de su código. Seguir paso a paso el código lleva tiempo, pero no tanto como lo que creen la mayoría de los programadores.

**CAPITULO CINCO. INTERFACES DE MAQUINAS AUTOMATICAS 77**

No es suficiente que las funciones no tengan errores; deben ser fáciles de utilizar sin introducir errores inesperados. Si se ha de reducir el porcentaje de errores, cada función necesita tener un propósito bien definido, tener entradas y salidas con un único propósito, ser legible desde donde se la llama, e idealmente no devolver nunca una condición de error. Las funciones con estos atributos son fáciles de verificar empleando validaciones y código de depuración, y minimizan el total de código de gestión de errores que se tiene que escribir.

**CAPITULO SEIS. RIESGOS DEL OFICIO \_\_\_\_\_ 97**

Dado el gran número de implementaciones posibles para una función concreta, no es sorprendente que algunas implementaciones sean más propensas a errores que otras. La clave para escribir funciones robustas es cambiar los algoritmos y las expresiones del lenguaje arriesgados por alternativas más seguras que hayan demostrado tener una eficiencia comparable. Por un lado, esto puede significar utilizar tipos de datos no ambiguos; por otro puede significar eliminar completamente un diseño simple porque es difícil, o imposible, probarlo.

**CAPITULO SIETE. CAMINOS EQUIVOCADOS \_\_\_\_\_ 129**

Algunas técnicas de programación son tan arriesgadas que nunca deberían usarse. La mayoría de ellas son obviamente arriesgadas, pero algunas parecen muy seguras, incluso deseables, porque cubren una necesidad sin peligro aparente. Estos modos equivocados de codificación son lobos con piel de cordero. ¿Por qué no puede referenciar memoria que acaba de liberar? ¿Por qué es arriesgado pasar datos en almacenamientos globales o estáticos? ¿Por qué se deben evitar las funciones parásitas? ¿Por qué es imprudente confiar en cualquier detalle muy concreto no incluido en el estándar ANSI?

**CAPITULO OCHO. LO IMPORTANTE ES LA ACTITUD \_\_\_\_\_ 151**

Un programador puede seguir todas las normas de este libro, pero sin la actitud apropiada y sin un conjunto de buenos hábitos en la programación, escribir código sin errores le resultará mucho más difícil de lo que debiera serlo. Si un programador cree que se puede “escapar” un error simplemente, o que corregir los errores “más tarde” no es perjudicial para el producto, los errores persistirán. Si un programador regularmente “reforma” el código, permite una flexibilidad innecesaria en las funciones, da la bienvenida a cualquier utilidad “gratuita” que salga del diseño, o simplemente “intenta” soluciones peligrosas, esperando tropezarse con algo que funciona, escribir código sin errores será una batalla ardua. Tener un buen conjunto de hábitos y actitudes posiblemente es el requisito más importante para escribir consistentemente código sin errores.

<b>EPILOGO. ¿CUAL ES EL SIGUIENTE PASO? _____</b>	<b>173</b>
<b>APENDICE A. LISTAS DE COMPROBACION PARA EL PROCESO DE CODIFICACION _____</b>	<b>175</b>
<b>APENDICE B. RUTINAS DE REGISTRO DE MEMORIA _____</b>	<b>181</b>
<b>APENDICE C. RESPUESTAS _____</b>	<b>189</b>
<b>REFERENCIAS _____</b>	<b>245</b>
<b>INDICE _____</b>	<b>247</b>