Refatoração de código em CPP

por Guilherme

Verificação Cuidadosa de Alocação de Memória

Tratar Falhas de Alocação

Não é suficiente apenas exibir uma mensagem e encerrar o programa. É necessário implementar um tratamento robusto, com tentativas de recuperação ou uma alternativa segura.

Verificação Explícita

Verifique sempre o resultado da chamada a malloc() ou calloc() para garantir que a alocação foi bem-sucedida antes de usar o ponteiro retornado.

Lógica de Recuperação

Caso a alocação falhe, considere opções como reduzir a quantidade de memória solicitada ou liberar recursos antes de tentar alocar novamente.

Mensagens Informativas

Forneça mensagens de erro claras e úteis para ajudar a diagnosticar e resolver problemas de alocação de memória.

Validação de Estruturas de Dados

Verificar Ponteiros Nulos

Antes de acessar qualquer estrutura de dados, certifique-se de que os ponteiros não são nulos para evitar erros de segmentação.

Validar Estados da Pilha

Verifique se a pilha está vazia antes de realizar operações, para não acessar memória inválida.

Verificações Holísticas

Não se limite a checagens pontuais. Valide a integridade geral da estrutura de dados após cada operação importante.

Tratamento de Entrada do Usuário

Verificação de Erros

Sempre verifique o resultado de funções de entrada, como scanf(), para garantir que a leitura foi bem-sucedida.

Mensagens Claras

Forneça mensagens de erro detalhadas e úteis quando ocorrerem problemas com a entrada do usuário.

Tratamento Robusto

Implemente um loop de entrada com validações para garantir que o usuário forneça dados válidos.

Documentação e Comentários

1

Descrição de Funções

Documente claramente o propósito, parâmetros e retorno de cada função implementada.

2

Explicação de Lógica

Adicione comentários explicando a motivação e o raciocínio por trás de trechos de código complexos.

3

Instruções de Uso

Forneça orientações sobre como usar corretamente o programa, especialmente para funcionalidades não triviais.

Benefícios da Atenção aos Detalhes

1

Código Robusto

Ao tratar adequadamente erros e validar estruturas de dados, você cria um código mais resistente a falhas.

2

Depuração Simplificada

Mensagens de erro claras e validações abrangentes facilitam a identificação e correção de problemas.

3

Maior Confiança

Usuários e colegas desenvolvedores terão mais confiança no seu programa quando ele apresentar um comportamento consistente e previsível.

Conclusão

Dedicar atenção a detalhes importantes, como a verificação de alocação de memória e a validação completa de estruturas de dados, é fundamental para a criação de software robusto, confiável e fácil de manter. Esse esforço extra no desenvolvimento irá recompensar você e seus usuários com um programa de alta qualidade.