Resumo de Aprendizagem - Clínica de Bugs C#

Problema 1 – Cadastro Simples:

Neste problema, aprendi a declarar e inicializar variáveis de diferentes tipos corretamente, como string, int, char e double. Entendi a importância de usar ponto em literais decimais no C# e de diferenciar aspas simples de duplas para char e string. Testar o código e interpretar as mensagens do compilador me ajudou a identificar erros de sintaxe e tipagem rapidamente. Também percebi que interpolação de strings facilita a exibição de dados formatados de forma clara. Um erro interessante foi tentar dividir o salário por uma letra, que me ensinou a refletir sobre operações válidas. Aprendi a importância de comentar decisões não óbvias e de manter o código legível. Vou levar como regra sempre verificar tipos e operações antes de rodar o programa.

Problema 2 – Escolha de Tipos de Dados:

Trabalhar com este problema me mostrou como a escolha correta do tipo de dado influencia o comportamento do programa. Percebi que usar int para valores inteiros, float ou double para decimais, e bool para status lógico evita erros de conversão. Também aprendi que strings devem ser delimitadas por aspas duplas, enquanto char exige aspas simples. Testar o programa ajudou a compreender a importância da tipagem forte em C#. Um erro curioso foi tentar atribuir string a um int, que evidenciou a necessidade de conversão explícita. Esse exercício reforçou o valor de planejar os tipos antes de codificar. Documentar decisões no código tornou o raciocínio mais claro para futuras revisões.

Problema 3 – Verificar Maioridade:

Este exercício reforçou meu entendimento de operadores de comparação e fluxo condicional. Inicialmente, confundi = com ==, o que causava comportamentos inesperados. Aprendi a sempre verificar a semântica de cada operador e testar casos limites, como 17 e 18 anos. Também percebi a importância de capturar a entrada do usuário com int.Parse corretamente para evitar erros de conversão. O teste de regressão ajudou a confirmar que a lógica estava correta após ajustes. Esse problema mostrou que pequenas confusões de sintaxe podem gerar grandes impactos no resultado. Comentei meu código para explicar decisões e prevenir futuros enganos.

Problema 4 – Nome do Dia da Semana:

Neste problema, aprendi a trabalhar com switch e validar entradas do usuário. Identificar números inválidos e fornecer mensagens claras mostrou a importância de tratar casos fora da faixa esperada. Um erro interessante foi confundir tipos de dados ao comparar string com int, reforçando a atenção à tipagem. Também percebi que organizar casos em blocos facilita a manutenção e a leitura do código. Testar entradas variadas, como 0, 1 e 8, ajudou a garantir

que todos os cenários fossem cobertos. Aprendi que loops e condicionais devem ser claros e documentados. Esse exercício reforçou a importância de planejamento antes de implementar.

Problema 5 – Confirmação de Saída (S/N):

Esse exercício mostrou a importância de loops condicionais e normalização de entrada. Aprendi a usar ToUpper() para aceitar respostas maiúsculas ou minúsculas e evitar erros de comparação. Testar o programa várias vezes me ensinou a lidar com repetição de entradas e lógica de interrupção. Inicialmente, usei operadores incorretos, o que me fez repensar a condição do do-while. Corrigir esses erros reforçou minha atenção a detalhes pequenos que impactam a execução. Comentei decisões não óbvias para futuras referências. Aprendi a sempre validar entradas antes de processá-las.

Problema 6 – Economizando até atingir a meta:

Trabalhar nesse problema me ajudou a consolidar conceitos de acumulação e operadores de atribuição. Inicialmente, usei =+ em vez de +=, o que sobrescrevia valores e gerava resultados incorretos. Testar casos como 10, 20, 30 ajudou a confirmar que a lógica de soma estava correta. Também aprendi a formatar números com Replace e CultureInfo para lidar com decimais. Esse exercício reforçou a necessidade de inicializar variáveis e pensar em loops que terminam corretamente. Documentar o raciocínio ajudou a entender o impacto de cada linha no total final.

Problema 7 – Tabuada de um número:

Gerar a tabuada mostrou a importância de configurar corretamente os loops. Inicialmente, usei decremento, o que criava loops infinitos. Percebi que variáveis de controle precisam ser bem planejadas. Testar todos os números de 1 a 10 me ajudou a identificar erros e garantir que os resultados fossem consistentes. Esse problema reforçou a atenção à lógica matemática e à sintaxe do for. Comentar a função do loop ajudou a manter o código legível. Aprendi a sempre revisar limites e passos de repetição antes de executar.

Problema 8 – Armazenar e Exibir Notas com Situação:

Neste exercício, percebi a importância de arrays e limites de índice. Inicialmente, tentei acessar posições fora do array, causando IndexOutOfRangeException. Aprendi a usar operadores ternários para simplificar condições de aprovação e reprovado. Testar entradas variadas ajudou a garantir que todos os cenários fossem tratados. Documentar decisões de lógica tornou o código mais compreensível. Esse problema reforçou a importância de validar dados de entrada e planejar estruturas de armazenamento antes de implementar.

Problema 9 – Matriz 3×3: diagonais, maiores por linha e transposta:

Trabalhar com matrizes reforçou meu entendimento sobre índices e loops aninhados. Inicialmente, inverti linhas e colunas ao preencher a matriz, o que gerava resultados errados. Testar visualmente cada operação me ajudou a identificar inconsistências. Aprendi a calcular somas, diagonais e valores máximos usando laços estruturados. A transposição da matriz

mostrou a importância de pensar cuidadosamente sobre posições e acesso a elementos. Comentei operações matemáticas para manter a clareza do código. Esse problema destacou a atenção a detalhes e o planejamento prévio.

Problema 10 – Procurar número e exibir posição (3×3):

Esse último problema me ensinou sobre flags booleanas e controle de fluxo. Inicialmente, a variável encontrado não era atualizada corretamente, fazendo com que a posição não fosse exibida. Testar diferentes casos me ajudou a validar a lógica de busca e a atualização de variáveis de controle. Aprendi a trabalhar com loops aninhados e condições de interrupção de maneira eficiente. Comentar cada decisão tornou o código mais legível e fácil de revisar. Esse exercício reforçou a importância de validar todos os cenários antes de considerar o programa concluído.