Universidade de Brasília
Departamento de Ciência da Computação
Disciplina: Métodos de Programação
Código da Disciplina: 201600

Métodos de Programação - 201600

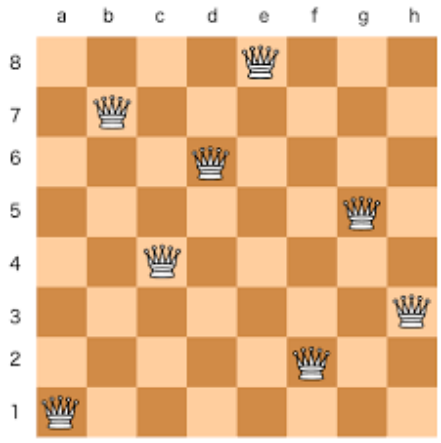
Trabalho 2

O objetivo deste trabalho é utilizar o desenvolvimento orientado a testes (TDD) para **verificar** se um tabuleiro contém a solução para o problema das 8 rainhas.

Não é necessário encontrar uma solução. O objetivo é apenas verificar se a entrada é uma solução

Neste problema se verifica alguma das rainhas ataca outra como na solução abaixo.

A solução do problema é quando nenhuma das 8 rainhas ataca outra como no exemplo abaixo.



A entrada é uma sequência de caracteres.

00000100
01000000
00010000
00000010
00100000
00000001
00001000
10000000

Cada linha representa uma fileira do tabuleiro
‘0’ representa a posição vazia
‘1’ representa a rainha na posição

Estes caracteres podem ser lidos de arquivos texto (.txt) com o formato de acima.

Estou enviando também um arquivo “teste_8_rainhas.txt” para mostrar o formato

A função deverá retornar:
1 caso seja uma solução para o problema
0 caso não seja uma solução para o problema
-1 caso a entrada não seja válida. Ex. não é um tabuleiro 8x8, não tem 8 rainhas, etc.

No caso de não ser solução para o problema, o programa deve salvar um arquivo “ataques.txt” com as posições das rainhas que se atacam. Um ataque por linha.

Ex.
1,3 4,3

significa rainha na posição 1,3 (linha, coluna) ataca rainha na posição 4,3

Você deve fazer assertivas de entrada e de saída para todas as funções do programa de forma semelhante ao que foi visto em sala e similar ao slide 14 de assertivas

Você deve fazer revisões e inspeções no código C/C++ conforme os documentos de checklists enviados.

O desenvolvimento deverá ser feito passo a passo seguindo a metodologia TDD. A cada passo deve-se pensar qual é o objetivo do teste e o significado de passar ou não no teste.

Você deve fazer pelo menos um commit para teste que você fizer. Devem ter sido feitos pelo menos 30 commits

O programa deverá ser dividido em módulos e desenvolvido em C ou C++. Deverá haver um arquivo rainhas.c (ou .cpp) e um arquivo rainhas.h (ou .hpp). Deverá haver também um arquivo testa_rainhas.c (ou .cpp) cujo objetivo é testar o funcionamento da biblioteca que testa o problema das 8 rainhas.

O programa deve usar um makefile para executar a compilação e outros programas.

1) Utilize o padrão de codificação dado em: https://google.github.io/styleguide/cppguide.html

quando ele não entrar em conflito com esta especificação. O código dever ser claro e bem comentado. O código deve ser verificado se está de acordo com o estilo usando o cpplint (https://github.com/cpplint/cpplint).

Utilize o cpplint desde o início da codificação pois é mais fácil adaptar o código no início.

2) O desenvolvimento deverá ser feito utilizando um destes frameworks de teste:

gtest (https://code.google.com/p/googletest/)
catch (https://github.com/philsquared/Catch/blob/master/docs/tutorial.md)

3) Deverá ser entregue o histórico do desenvolvimento orientado a testes feitos através do git (https://git-scm.com/docs/gittutorial)

git config --global user.name "Your Name Comes Here"
git config --global user.email you@yourdomain.example.com
git init
git add *
git commit –m “teste 1”
git log

Compactar o diretório “.git” ou equivalente enviando ele junto.

4) Deve ser utilizado um verificador de cobertura
ex. gcov. (http://gcc.gnu.org/onlinedocs/gcc/Gcov.html). O makefile deve ser modificado de forma incluir as flags -ftest-coverage -fprofile-arcs. Depois de rodar o executável rode gcov nomearquivo e deverá ser gerado um arquivo .gcov com anotação.

O verificador de cobertura é utilizado para saber qual percentual do código é coberto pelos testes. Neste caso os testes devem cobrir pelo menos 80% do código por módulo.

5) Utilize um verificador estático
ex. cppcheck, corrigindo os erros apontados pela ferramenta.
Utilize cppcheck --enable=warning .
para verificar os avisos nos arquivos no diretório corrente (.)

Utilize o verificador estático sempre e desde o início da codificação pois é mais fácil eliminar os problemas logo quando eles aparecem. Devem ser corrigidos apenas problemas no código feito e não em bibliotecas utilizadas (ex. gtest, catch)

utilizar o verificador dinâmico Valgrind (*valgrind.org/*)

6) Deve ser gerada uma documentação do código usando o programa DoxyGen (http://www.stack.nl/~dimitri/doxygen/): O programa inteiro terá de ser documento usando DoxyGen. Comentários que vão ficar na documentação devem ser do estilo Javadoc.
Para gerar uma documentação mais adequada, rodar
doxygen –g
isto irá gerar um arquivo Doxyfile. Neste arquivo, na linha adequada, colocar:

EXCLUDE = catch.hpp

Isto fará com que o catch.hpp não seja documentado. Uma mudança semelhante deverá ser feita para outro framework se necessário.

Depois de feito isto, para documentar basta rodar : doxygen

Devem ser enviados para a tarefa no aprender3.unb.br um arquivo zip onde estão compactados todos os diretórios e arquivos necessários. O documento deve estar na raiz do diretório. Todos os arquivos devem ser enviados compactados em um único arquivo (.zip) e deve ser no formato matricula_primeiro_nome ex: 06_12345_Jose.zip. Deve conter também um arquivo leiame.txt que diga como o programa deve ser compilado.

Deve ser enviado o diretório ".git" compactado junto com o ".zip"

Data de entrega:

4/ 6 /23

Pela tarefa na página da disciplina no aprender3.unb.br