



UNIVERSIDADE DE BRASÍLIA  
2º SEMESTRE DE 2018

201600 – MÉTODOS DA PROGRAMAÇÃO

PROFESSOR: Jan Corrêa

TURMA: A

ALUNO: Marcos Vinícius Prescendo Tonin – 140153233.

**Trabalho 3 - Checklist para verificação de programa em C/C++**

- *Link github : [https://github.com/M-Tonin/trabalho3\\_tdd-MarcosTonin](https://github.com/M-Tonin/trabalho3_tdd-MarcosTonin).*
- *As assertivas foram feitas usando doxygen como padrão.*
- *Testes das especificações.*
- *Gtest*

```
mtonin@mTonin-X510UR:~/Documents/GTest-master$ ./runTests
[=====] Running 6 tests from 5 test cases.
[-----] Global test environment set-up.
[-----] 1 test from abrirArquivo
[ RUN   ] abrirArquivo.teste_abrir
sdasd.c
Arquivo aberto
sdASF.c
Arquivo aberto
sdASF1234154.c
Error ao abrir arquivo! Arquivo não existe.
Exemplo24Stack.cpp
Arquivo aberto
Exemplo18Enum2.cpp
Arquivo aberto
Exemplo17Friend.cpp
Arquivo aberto
[      OK ] abrirArquivo.teste_abrir (0 ms)
[-----] 1 test from abrirArquivo (1 ms total)

[-----] 1 test from cont_total
[ RUN   ] cont_total.contando
[      OK ] cont_total.contando (0 ms)
[-----] 1 test from cont_total (0 ms total)

[-----] 1 test from cont_branco
[ RUN   ] cont_branco.contando_branco
[      OK ] cont_branco.contando_branco (0 ms)
[-----] 1 test from cont_branco (0 ms total)

[-----] 2 tests from cont_comentario
[ RUN   ] cont_comentario.contando_com
[      OK ] cont_comentario.contando_com (0 ms)
[ RUN   ] cont_comentario.contando2
Exemplo18Enum.cpp
Arquivo aberto
[      OK ] cont_comentario.contando2 (0 ms)
[-----] 2 tests from cont_comentario (0 ms total)

[-----] 1 test from cont_final
[ RUN   ] cont_final.cont_sem_branco_coment
Todas linhas contadas : 29
Todas linhas em branco : 3
Todas linhas comentadas : 3
Total de linhas uteis : 23
Todas linhas contadas : 33
Todas linhas em branco : 2
Todas linhas comentadas : 3
Total de linhas uteis : 28
Todas linhas contadas : 64
Todas linhas em branco : 6
Todas linhas comentadas : 11
Total de linhas uteis : 47
Todas linhas contadas : 42
Todas linhas em branco : 5
Todas linhas comentadas : 7
Total de linhas uteis : 30
[      OK ] cont_final.cont_sem_branco_coment (0 ms)
[-----] 1 test from cont_final (0 ms total)

[-----] Global test environment tear-down
[=====] 6 tests from 5 test cases ran. (1 ms total)
[ PASSED ] 6 tests.
mtonin@mTonin-X510UR:~/Documents/GTest-master$
```

- **Gcov em teste\_contador e contador (.cpp)**

```
mtonin@mTonin-X510UR:~/trabalho3_tdd /trabalho3_tdd/conta_linha/source$ cd obj/
mtonin@mTonin-X510UR:~/trabalho3_tdd /Trabalho3_tdd/conta_linha/source/obj$ gcov teste_contador.cpp -b
File 'teste_contador.cpp'
Lines executed:100.00% of 30
Branches executed:100.00% of 96
Taken at least once:50.00% of 96
Calls executed:82.09% of 67
Creating 'teste_contador.cpp.gcov'
Cannot open source file teste_contador.cpp

File '/usr/include/c++/5/iostream'
Lines executed:100.00% of 1
No branches
Calls executed:100.00% of 2
Creating 'iostream.gcov'

mtonin@mTonin-X510UR:~/trabalho3_tdd /Trabalho3_tdd/conta_linha/source/obj$ gcov contador.cpp -b
File 'contador.cpp'
Lines executed:85.23% of 88
Branches executed:81.93% of 166
Taken at least once:58.00% of 166
Calls executed:84.42% of 77
Creating 'contador.cpp.gcov'
Cannot open source file contador.cpp

File '/usr/include/c++/5/iostream'
Lines executed:100.00% of 1
No branches
Calls executed:100.00% of 2
Creating 'iostream.gcov'

mtonin@mTonin-X510UR:~/trabalho3_tdd /Trabalho3_tdd/conta_linha/source/obj$
```

- **Cpplint**

Deu apenas um erro, no arquivo *contador.hpp*, referente ao uso *std namespace*.  
Porém, não foi mudado.

- **Cppcheck**

```
+ gitkraken ...st-master: ./runTests x source: cppcheck
mtonin@mTonin-X510UR:~/trabalho3_tdd /Trabalho3_tdd/conta_linha/source$ cppcheck --enable=warning teste_contador.cpp
Checking teste_contador.cpp...
mtonin@mTonin-X510UR:~/trabalho3_tdd /Trabalho3_tdd/conta_linha/source$ cppcheck --enable=warning contador.cpp
Checking contador.cpp...
mtonin@mTonin-X510UR:~/trabalho3_tdd /Trabalho3_tdd/conta_linha/source$
```

Furthermore, you should specify "--enable=all" if you want to see errors classified as error, not as style. Unused variable (as given in your example)

- **Laudo**

d) **Como fazer o código**

- 1) As variáveis têm seus nomes bem determinados ? Os nomes das variáveis devem ser claras e concisas.

**Os nomes das variáveis tendem a indicar o que fazem.** ✓ ✗

- 2) As variáveis estão espalhadas pelo código, sendo declaradas em qualquer local ?

**Foi feito na medida do possível.** ✓

- 3) A variável não pode assumir valores negativo ? Se sim, deve-se por optar usar *unsigned*.

**Não foi utilizado.** ✓

- 4) As variáveis seguem alguma convenção ? Se não, as variáveis devem seguir uma convenção.

**Sim, seguem.** ✓

- 5) A alguma possibilidade referência nula (*null*) sem tratamento específico ? Se houver deve ser tratado.

**Sim, foi tratado.** ✓

- 6) A acusação e algum erro ou *project warnings*?

**Não.** ✓

- 7) Há alguma parte do código usada pelo desenvolvedores para testes?

**Não, todas foram tiradas.** ✓

- 8) O código em si está claro?

**Sim!** ✓

- 9) Todos ponteiros estão sendo liberados após ser usado? Se não, devem ser liberado para evitar estouro de memória.

**Sim!** ✓

- 10) Há *if's* que estão aninhado ? Se sim, deve-se procurar usar o *switch* para uma maior concisão do código.

**Não há if aninhados.** ✓

- 11) Todas as variáveis começam com o valor zerado?

**Apriori, todas foram zerado.** ✓

- 12) As funções possuem nome padrão ?

**Não.** X

- 13) Há código inalcançável (unreachable code) ?

**Não.** ✓

- 14) As indentações do código são plausível?

**Sim.** ✓

- 15) Há variáveis que fazem a mesma coisa sem ter necessidade ? Se sim, excluir a variável repetida.

**Não houve uso de contador.** ✓

- e) **Como comentar o código e escrever o código com “design by contract” com assertivas**

- 1) Há comentários inúteis no código ?

**Não.** ✓

- 2) Há comentários depois de cada *if* ?

**Não.** X

- 3) As partes difíceis do código estão comentadas ?

**Sim.** ✓

- 4) Os comentários estão desenvolvido de forma a gerar documentação pelo *doxygen*?

**Sim.** ✓

- 5) Há comentários depois de cada *for* ?

**Não.** X

- 6) O programa utiliza a técnica *design by contract* ?

**Usa.** ✓

- 7) O programa utiliza assertiva ? Se não usar, considerar usar.

**Sim.** ✓

- 8) As funções segue um contrato de funções baseado no “Design by contract” ?

**Sim.** ✓

- 9) As funções especificam quais seus pré requisitos/condições ? As funções devem especificar quais são seus pré requisito para funcionamento.

**Sim.** ✓

- 10) As estruturas principais do código estão comentados ?

**Não, não há estrutura.** ✓

- 11) As funções especificam quais seus pós-requisitos/condições ?

**Sim.** ✓

- 12) Verificar se há comentários que desobedecem a regra do UTF-8.

**Sim.** ✓

- 13) Os comentários são muito denso e acabam por se tornar de difícil leitura?

**Não.** ✓

- 14) Há comentários que especifiquem bugs ?

**Não.** ✓

- 15) Os comentários estão apropriado e de forma que outras pessoas possam entender ?

**Sim.** ✓

- **de entrada, saída, invariantes e como comentários de argumentação do código**

- 1) As funções tem seu retorno especificado? Se não, deve-se comentar e tentar explicar o que significa.

**Sim.** ✓

- 2) As funções tem seu parâmetros especificado? Se não, deve-se comentar e tentar explicar o que significa.

**Não.** ✓

- 3) As invariantes da função estão bem protegida ?

**Sim.** ✓

- 4) As entradas das funções correspondem ao tipo definido pela própria função ?

**Sim.** ✓

- 5) Como tratar as saídas que são retornadas por referência ?

**Sim.** ✓

- 6) Há alguma invariante que se altera quando não devia?

**Sim.** ✓

- 7) Quando ocorre erro o programa termina ?

**Não.** X

- 8) Há entradas da função ou do sistema que são inúteis ? **Não.** ✓

- 9) Há entradas da função que dá algum problema na função? **Não.** ✓

- 10) Há entradas de pelo menos duas funções/sistemas diferentes que representam a mesma coisa com nomes diferente ?

**Sim.** ✓

- 11) As funções que tem saída *void*, quando ocorre um erro deve especificar o que deu errado, e não apenas sair haja visto que não há retorno. **Não.** ✓

- 12) Há invariantes no código que são variáveis globais ? **Não.** ✓

- 13) Há entradas em demasiada quantidade ? **Não.** ✓

- 14) Os argumentos das funções tem seu nome de forma a indicar o que possa significar ? **Não.**

- 15) As saídas são, por si, fazem o que função sugere fazer ? **Sim.** ✓

- g) Como testar o código
- 1) Foi feito um código de teste ? **Sim.** ✓
- 2) O teste é feito de forma automática usando algum software ? **Sim, gtest.** ✓
- 3) O desenvolvimento foi feito baseado em testes ? **Sim, baseado em TDD.** ✓
- 4) Os módulos existentes foram testados de forma separadas ? **Sim, na medida da possível.**
- 5) Os testes feitos cobrem pelo menos 80 % do código ? ✓
- 6) Os testes foram feitos independentes ? ✓
- 7) Os testes seguiram a prioridade das funcionalidades ? **Sim.** ✓
- 8) Os testes produzem um laudo ? **Sim, pelo próprio gtest.** ✓
- 9) Os testes são revisados de forma periódica? **Não foi necessário.**
- 10) Sabe-se exatamente o que se quer testar de cada sistema/funcionalidade ? ✓
- 11) Há testes que estão defasados ? **Não há.** ✓
- 12) Há casos especiais que necessitem ser tratados? **Não há.** ✓
- 13) Foram verificadas as restrições de máquina ? **Não foi.**
- 14) Foi testada a abrangência dos dados de entrada ? ✓
- 15) A massa de dados contempla todas as funcionalidades e estruturas internas do sistema? ✓

- 17 - Responda

**Função:** `int abrir_arquivo(string nome_arq, ifstream& arq);`

**Descrição:** Função que abre o arquivo de nome "nome\_arq" e retorna em arq seu ponteiro.

**Parâmetro :**

**nome\_arq** : String contendo o nome do arquivo.

**arq** : Representa o tipo stream que será retornado.

**Retorno :** Retorna um se deu certo e zero caso dê errado.

**Teste :** Pelo gtest, passando nomes de arquivo que existam e retornando 1 e nome de arquivo que não existe retornando zero, passou no teste.

**Função:** `int conta_total_linhas(ifstream& arq);`

**Descrição:** Conta o número total de linhas do arquivo.

**Parâmetro**

**arq** : Representa o tipo stream que será usado.

**Retorno :** Retorna o total de linhas do arquivo em questão.

**Teste :** Pelo gtest, arquivos e contando o total de linha, passando no teste.

**Função :** `int conta_linhas_branco(ifstream& arq);`

**Descrição** Conta o numero total de linhas em branco

**Parâmetro**

**arq** : Representa o tipo stream que será usado.

**Retorno :** Retorna o total de linhas em branco do arquivo em questão.

**Teste** : Pelo *gtest*, arquivos e contando o total de linha em brancos, passando no teste.

**Função** : *int eh\_vazio(const char st);*

**Descrição** : Dado uma linha verifica se eh vazia ou se não é.

**Parâmetro** *st* -> string que contém a linha toda.

**Retorno** 1: linha está vazia "em branco" e 0: não está em branco.

**Teste** : Pelo uso *indireto por outras funções*, sendo bem sucedido, passou no teste.

**Função** *int nao\_eh\_vazio\_indice(const char st);*

**Descrição** Dado uma linha verifica qual o primeiro char que não eh nulo "" ou tab.

**Parâmetro** *st* -> string que contém a linha toda.

**Retorno** : -1 caso não seja encontrada, ou retorna o índice que foi encontrado algum char.

**Teste** : Pelo uso *indireto por outras funções*, sendo bem sucedido, passou no teste.

**Função** : *int conta\_comments(ifstream& arq);*

**Descrição** : Conta o número total de linhas que são comentadas.

**Parâmetro**

*arq* : Representa o tipo stream que será usado.

**Retorno** : Retorna o total de linhas que foram comentadas do arquivo em questão.

**Teste** : Pelo *gtest*, arquivos e contando o total de linha comentadas, passando no teste.

**Função** : *int conta\_final(ifstream& arq);*

**Descrição** : Calcula o número de linhas realmente usadas.

**Parâmetro**

*arq* : Representa o tipo stream que será usado.

**Retorno** : Retorna o total de linhas realmente úteis.

**Teste** : Pelo de *gtest*, correspondendo o valor certo, passando no teste.

**Função** *int fecha\_arquivo((ifstream& arq);*

**Descrição:** Fecha o arquivo

**Parâmetro:**

*arq* : Representa o tipo stream que será usado.

**Retorno** : 1- se conseguiu fechar e 0 se não deu certo.

**Teste** : Não foi testado por sua simplicidade.

- **Perguntas**

- **Existem funções que podem corromper a estrutura de dados ? Como?**

Sim, há funções que podem corromper a estrutura de dados, mesmo fazendo todos os procedimentos de testes e tudo mais, ainda assim é

possível violar a estrutura de dados. Esse corrompimento pode ocorrer principalmente em funções que mexem com estrutura em si e no caso deste programa as que manipulam ponteiros (*strings*) como *eh\_vazio()* e *nao\_eh\_vazio\_indice()*.

Neste caso, podem então violar a memória apontando para lugares errado da memória, causando o erro *segmetion\_fault*, pode ocorrer também o estouro de memória em casos muito extremo.

- **O que pode ser feito para evitar tal problema.**

Portanto, para evitar o problema deve-se sempre tomar cuidado com a utilização do ponteiro, alguns cuidados também foram tomados para que não fossem passados valores negativos para criação da pilha.

- **Bibliografia**

- Slides vistos em sala de aula.
- Especificações do trabalho disponibilizado pelo professor.
- Links disponível na mesma especificação.