# Teste automatizado de algoritmos através de assertivas e da framework da Google®

Rodrigo Ferreira Guimarães
Departamento de Ciência de Computação e Faculdade de Tecnologia
Universidade de Brasília, Brasília
Matrícula 14/0170740
Email: rodrigofegui@aluno.unb.br

## I. Introdução

Este trabalho trata da aplicação de testes automatizados para as funções responsáveis pela manipulação da estrutura de dados denominada *pilha*, tanto na sua implementação como vetor como lista encadeada.

Serão expostos os passos para a execução do trabalho como um todo, desde a instalação do gtest até a execução do *cppcheck* (considerado como o último passo do trabalho).

## II. Embasamento Teórico

Para que haja um correto entendimento sobre o desenvolvimento deste projeto é importante abordar alguns aspectos relevantes, sendo eles descritos nas subseções a seguir.

Vale ressaltar que os comandos executados nos blocos são referentes à execução deste trabalho, podendo haver mudanças para outros trabalho. Onde estiver escrito "*P*", nos blocos, é apenas um espaço entre a linha anterior e a linha atual, pois, devido ao espaço de exibição os códigos podem ficar além da área devida.

### A. Gtest

O *Gtest*[1] é uma *framework* da Google escrita para realizar testes de algoritmos em linguagem de programação C++, em várias plataformas - baseado na arquitetura *xUnit*. Possui suportes para testes automatizados, configurações de assertivas, definições de assertivas, testes fatais, dentre outras coisas.

Faça o *download*, disponibilizado no *site* do *gtest*, do arquivo *gtest-"última versão".zip*, após isso abra o terminal e execute os comandos como mostrados no bloco *Instalacao_Gtest*.

Ao seguir esses passos o *gtest* será instalado num diretório que apenas usuário *root*, administrador, poderá manipular.

Para que seja possível a utilização do *gtest* para um programa expecífico, é importante criação de um arquivo cmake, para a automatização das operações, como demostrado no bloco *CMakeLists.txt*.

Para a execução do *gtest* é necessário seguir os comandos mostrados no bloco *Executando_Gtest*.

```
cd Download/
sudo mv gtest-1.7.0.zip /opt/google/
cd /opt/google/
sudo unzip gtest-1.7.0.zip
cd gtest-1.7.0
sudo ./configure
sudo make
sudo cp -a include/gtest /usr/include
sudo cp -a lib/ .libs
sudo mkdir mybuild
cd mybuild/
sudo cmake -DBUILD_SHARED_LIBS=ON -Dgtest_build_samples=ON
"P" -G"Unix Makefiles" ..
sudo make
ls
sudo ./sample2_unittest
```

Instalacao_Gtest

```
cmake_minimum_required(VERSION 2.6)
project(Trab2_MP_Rodrigo)
enable_testing()
find_package (Threads)
message(STATUS GTEST_ROOT=$ENV{GTEST_ROOT})
include_directories($ENV{GTEST_ROOT}/include)
link_directories($ENV{GTEST_ROOT}/samples)
add_executable(Trab2_MP_Rodrigo Fontes/Pilha.cpp
"P" Fontes/Pilha_teste.cpp)
target_link_libraries(Trab2_MP_Rodrigo gtest gtest_main)
target_link_libraries(Trab2_MP_Rodrigo
"P" ${CMAKE_THREAD_LIBS_INIT})
add_test(NAME Trab2_MP_Rodrigo COMMAND Trab2_MP_Rodrigo)
```

CMakeLists.txt

### B. Gcov

Também foi utilizado o programa *Gcov* [2], que é um programa que oferece cobertura para testes. Sendo possível utilizá-lo como uma ferramenta de análise para ajudar a descobrir onde aplicar uma otimização de processamento, por exemplo.

Para que haja a execução do gcov, juntamente com o *gtest*, é preciso acrescentar *flags* para a execução, sendo elas descritas no bloco *Acrescentando_Gcov*.

Para a execução do *gcov* são necessários os passos descritos no bloco *Executando_Gcov*.

### C. CppCheck

Em paralelo à utilização do *gtes* e do *gcov*, também foi utilizado a ferramenta *cppcheck*.

O *CppCheck* [3] é uma ferramenta de análise estática para códigos C/C++ que foca em analisar o que os compiladores, e muitas outras ferramenteas de análise, não detectam: falsos positivos.

Há a necessidade de instalação do *cppcheck* e para tanto é necessário seguir os comandos do bloco *Instalacao_Cppcheck*.

```
mkdir Construcoes
cd Construcoes
cmake ..
make
ls
./Trab2_MP_Rodrigo
```

Executando_Gtest

```
SET(GCC_COVERAGE_COMPILE_FLAGS
"P" "-fprofile-arcs -ftest-coverage")
SET(GCC_COVERAGE_LINK_FLAGS
"P" "-lgcov")
SET(CMAKE_CXX_FLAGS
"P" "${CMAKE_CXX_FLAGS}
"P" ${GCC_COVERAGE_COMPILE_FLAGS}" )
SET(CMAKE_EXE_LINKER_FLAGS
"P" "${CMAKE_EXE_LINKER_FLAGS}
"P" ${GCC_COVERAGE_LINK_FLAGS}" )
SET(CMAKE_CXX_OUTPUT_EXTENSION_REPLACE 1)
```

Acrescentando_Gcov

A fim de executar o *cppcheck* sobre os códigos desenvolvidos, seguir os comandos no bloco *Executando_Cppcheck*.

### D. Assertivas

Assertivas são expressões lógicas envolvendo dados e estados manipulados, podendo ser definidas em níveis de abstração:

- ***Funções***: devem estar satisfeitas em determinados pontos do corpo da função, comumente denominadas assertivas de *entrada* e de *saída*;
- ***Classes e Módulos***: devem estar satisfeitas ao entrar e ao retornar de funções, assertivas *invariantes* e *estruturais*;
- ***Programas***: devem estar satisfeitas para os dados persistentes.

Podendo ser utilizadas para para a especificicação de funções - com o desenvolvimento dirigido por contratos -, argumentação de corretude de programas - através dos predicados da argumentação -, depuração - facilitando a completa e correta remoção de defeitos -, dentre outras.

## III. Resultados

Com a base teórica construída é possível apresentar os resultados dos testes. Será apresentado os resultados sujeitos ao padrão: nome da função; parâmetros de entrada e seus significados; especificação da função; para uma dada entrada, qual a saída esperada; críterio de aprovação; e se a função passou no teste desenvolvido.

Como há duas formas de implementação da pilha, logo estas partes serão expostas separadamente.

### A. Implementada como Vetor

Esta seção é dedicada ao desenvolvimento do trabalho analisando a implementação da pilha como um vetor, que tem tamanho fixo. Dessa forma, tem-se as funções como descritas a seguir.

```
cd Construcoes/CMakeFiles/Trab2_MP_Rodrigo.dir/Fontes/
gcov *.gcno
```

Executando_Gcov

```
sudo apt-get update
sudo apt-get install cppcheck
```

Instalacao_Cppcheck

*1) Criação:* Para que haja o manuseio da pilha é importante que ocorra a criação da mesma, logo esta função tem a finalidade de conseguir alocar um espaço na memória para o vetor a inserido a lista, com isso, considerando a sua forma de implementação, tem-se o protótipo:

$$bool\ criar\ (int\ tam);$$

A partir disso, atribui-se à variável $tam$ o tamanho do vetor a ser alocado, havendo a ressalva (através de uma assertiva de entrada) de que o valor deve ser maior e igual a $1$, pois não há vetor com $-10$ posições, por exemplo. Há o retorno de um estado lógico positivo se a alocação foi bem sucedida, sendo isto avaliado da seguinte forma: se antes de um dado limite de tentativas, uma memória for alocada, então houve sucesso, caso contrário não houve.

O teste sobre está função foi subdividido em duas parte: a primeira com a tentativa da criação com um número menor do que $1$ e a segunda com um valor inteiro positivo qualquer; como previsto, a assertiva fez com que o programa fosse encerrado para o primeiro teste, visto que foi violada a condição básica, enquanto que para a segunda houve o retorno do estado lógico positivo. Dessa forma, a função passou no teste.

*2) Inserção:* Para a função de inserção tem-se o protótipo:

$$int\ inserir\ (int\ info)$$

A partir disso, atribui-se à variável $info$ a informação a ser inserida na pilha, havendo as ressalvas: a pilha já deve estar alocada e a pilha não pode estar cheia, ou seja, deve haver, pelo menos, uma posição livre no vetor. Passando pelas assertivas de entrada, há o retorno do próprio valor inserido, pois este será comparado com o valor registrado no topo da pilha.

O teste foi subdividido em três partes: a primeira consiste em tentar inserir sem ter a pilha alocada, a segunda uma inserção em condições normais de trabaho e a terceira tentando inserir elemento além da capacidade da pilha; como previsto, para o primeiro caso houve o retorno de condição falsa, visto que o vetor não tinha sido alocado, para a segunda também teve êxito, pois o valor retornado coincidiu com o valor registrado no topo e para o último caso, também houve êxito, pois retornou o valor falso, uma vez que o vetor já estava cheio.

*3) Remoção:* Para a função de remoção tem-se o protótipo:

$$int\ remover()$$

A partir disso, percebe-se que não há necessidade de parâmetros de entrada, embora haja assertivas de entrada: a

```
cd ../../../../Fontes/
cppcheck *.cpp
```

Executando_Cppcheck

pilha deve estar alocada e não deve estar vazia; passando por esta etapa, há o decremento do topo e o retorno do valor que estava no topo.

O teste foi subdividido em duas partes análogo à inserção, mas respeitando as assertivas da remoção.

*4) Topo:* Para a função de visualização do topo tem-se o protótipo:

$$int\ topo()$$

Possui as mesmas características que a remoção, divergindo apenas quanto ao decremento do topo, pois o mesmo eh mantido.

### *B. Implementada como Lista Encadeada*

Esta seção é dedicada ao desenvolvimento do trabalho analisando a implementação da pilha como uma lista encadeada. Dessa forma, tem-se as funções como descritas a seguir.

*1) Criação:* Tem-se a função para criação da pilha, pelo mesmo motivo da implementação como vetor, mas, principalmente, pela correta inicialização das variáveis de controle da pilha., sendo que está possui o seguinte protótipo:

$$bool\ criar\ ()$$

A partir disso, percebe-se que não há a necessidade da atribuição de parâmetros para função; há apenas o uso de uma assertiva de saída para garantir que o ponteiro foi inicializado corretamente, sendo positivo há o retorno do estado positivo, caso contrário do negativo.

### *C. Semelhantes*

Há funções que possuem o mesmo procedimento de teste, com passos semelhantes a serem seguidos, dentre elas: a inserção, a remoção e o retorno do topo. Estas possuem os mesmos protótipos e características semelhantes, quanto ao desenvolvimento dos testes.

Ocorre diferença na funão de inserção, uma vez que é possível inserir elementos na pilha antes de uma criação formal da pilha, ou seja, as variáveis de controle não transmitem a realidade quanto seu estado corrente.

## IV. Conclusão

Com o desenvolvimento dos testes através do *Google Test*, foi obtido êxito na execução; em relação ao *Gcov*, o mesmo pode ser afirmado, uma vez que foi encontrado o valor de 79,82% de cobertura dos teste, sendo próximo ao valor recomendado pela expecificação do projeto.

## Referências


[1] GoogleTest. *Google C++ Testing Framework*. Disponível em: $https://code.google.com/p/googletest/$, acessado em 2015.

[2] Gcov. *Gcov - A Test Coverage Program*. Disponível em: $https://gcc.gnu.org/onlinedocs/gcc/Gcov.html$, acessado em 2015.

[3] Cppcheck. *Cppcheck - A Tool for static C/C++ code analysi*. Disponível em: $http://cppcheck.sourceforge.net/$, acessado em 2015.