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

Métodos de Programação - 201600

Trabalho 1

Utilizando o laboratório 1:

Desenvolver o módulo pilha em C ou C++. Devem ser escritos os arquivos (pilha.h e pilha.c e testa_pilha.c) ou (pilha.hpp e pilha.cpp e testa_pilha.cpp), devem ser implementadas e testadas as funções:

Push (pilha, elemento) - coloca elemento no topo da pilha
Pop (pilha) - retira elemento do topo da pilha – retorna um elemento
Top (pilha) – retorna elemento do topo da pilha sem modificar a pilha – retorna um elemento
Size (pilha) – retorna int tamanho da pilha
SetSize (pilha, int) – muda o tamanho da pilha – retorna True se deu certo, False caso contrário
IsFull (pilha) – retorna verdadeiro se a pilha está cheia e falso caso contrário
IsEmpty (pilha) - retorna verdadeiro se a pilha está vazia e falso caso contrário
CreateStack () – cria pilha – retorna pilha
DestroyStack (pilha) – destrói pilha

(podem ser feitas outras funções se necessário)

A pilha deve guardar um tipo de dado abstrato (struct ou classe) ItemType que inicialmente contem um inteiro mas que pode ser modificado para conter outro tipo de dado com facilidade. Nenhuma das funções listadas devem usar a suposição que ItemType contem um inteiro. Isto é, as funções empilham e desempilham ItemType sem levar em consideração que tipo ItemType contem.

A pilha deve ser implementada de duas formas:

1) Vetor de ItemType
2) Lista encadeada onde cada nodo contem um ItemType

O arquivo pilha.h deve prover as funções de forma mais genérica possível, devendo ser fácil mudar a implementação de vetor para lista encadeada apenas modificando a implementação em "pilha.c" (ou .cpp) e sem mudar o protótipo das funções em "pilha.h" (ou .hpp) e mudando a compilação.

Este trabalho tem como objetivo também avaliar a capacidade do aluno de pensar as funções que são necessárias para compor a biblioteca bem como testá-la.

Faça um programa executável chamado "testa_pilha.c" (ou .cpp) que utiliza o modulo pilha.c que implementa uma biblioteca de pilha usando a interface definida em pilha.h (ou .hpp).

O programa testa_pilha.c (ou .cpp) deve testar se a implementação da biblioteca de pilha funciona corretamente. Devem ser tratadas as exceções. Ex: O que acontece quando se tenta retirar um elemento de uma pilha vazia?

O mesmo arquivo "testa_pilha.c" ou "testa_pilha.cpp" deve servir como teste para a implementação em vetor e em lista encadeada.

O arquivo "testa_pilha.c" ou "testa_pilha.cpp" pode depender do tipo de ItemType

São dois arquivos pilha.c (ou pilha.cpp) um para a implementação com vetor e outro com a implementação em lista encadeada. Os dois vão idealmente usar o mesmo pilha.h.

O programa e o módulo devem ser depurados utilizando o GDB.
(http://heather.cs.ucdavis.edu/~matloff/UnixAndC/CLanguage/Debug.html)
(https://www.cs.umd.edu/~srhuang/teaching/cmsc212/gdb-tutorial-handout.pdf)

1) Faça um Makefile utilizando o exemplo de makefile 5 dado em:

(http://www.cs.colby.edu/maxwell/courses/tutorials/maketutor/)

Não se esqueça de criar os diretórios correspondentes.

2) **O desenvolvimento deverá ser feito orientado a testes como visto em aula utilizando um destes frameworks de teste**:

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

3) Deverá ser entregue o histórico do desenvolvimento orientado a testes feito através de um gerenciador de versões como o svn ou github (https://github.com/), Tortoise svn ou outro que mostre todas as versões

4) 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 esta 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.

5) Instrumente o código usando o 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 gcov 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.

6) Faça a análise estática do programa utilizando o cppcheck, corrigindo os erros apontados pela ferramenta.

Utilize no diretório do código fonte:

```
cppcheck --enable=warning .
```

para verificar os avisos nos arquivos no diretório corrente (.)

Utilize o cppcheck 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)

7) Deve ser gerada uma documentação do código usando o programa DoxyGen (http://www.stack.nl/~dimitri/doxygen/): O programa inteiro deverá de ser documentado usando o DoxyGen.

8) É interessante utilizar o Valgrind (*valgrind.org/*), embora não seja obrigatório.

Devem ser enviados para a tarefa no aprender3.unb.br um arquivo zip onde estão compactados todos os diretórios e arquivos necessários. Todos os arquivos devem ser enviados compactados em um único arquivo (.zip) e deve ser no formato matricula_primeiro_nome.zip. ex: 06_12345_Jose.zip. Deve ser enviado um arquivo dizendo como o programa deve ser compilado e rodado.

Data de entrega:

14/ 10 /20

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