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

Métodos de Programação - 201600

# Trabalho 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- coloca elemento no topo da pilha
Pop - retira elemento do topo da pilha
Top – retorna elemento do topo da pilha sem modificar a pilha.
Size – retorna tamanho da pilha
SetSize – muda o tamanho da pilha
IsFull – retorna verdadeiro se a pilha está cheia
IsEmpty - retorna verdadeiro se a pilha está vazia
CreateStack – cria pilha
DestroyStack – destrói pilha

(podem ser feitas outras funções se necessário)

A pilha deve guardar um tipo de dado abstrato que inicialmente é um inteiro mas que pode ser modificado para outro tipo de dado com facilidade.

A pilha deve ser implementada de duas formas:

1) Vetor de inteiros
2) Lista encadeada

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 o “pilha.c” e a compilação.

Não devem ser usadas funções de biblioteca que resolvam o problema como por exemplo uma função de pilha.

Faça um programa executável chamado “testa_pilha.c” (ou .cpp) que utiliza o modulo arvore.c que implementa uma biblioteca de arvore 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 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) 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.

3) Utilize um framework de teste: gtest ou catch.

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

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

Devem ser enviados para a tarefa no ead.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:

**11/ 9 /18**

**Pela tarefa na página da disciplina no ead.unb.br**