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

Métodos de Programação - 201600

Trabalho 1

Crie uma biblioteca de arvore binária em C ou C++:

A biblioteca deve ter todas as funções necessárias para manipular a arvore, inclusive permitir destruir a arvore.

Para cada função você deve especificar qual o tipo de retorno, dos parâmetros e o que acontece no caso de haver erro.

A seguir a biblioteca dever ser utilizada para implementar o jogo das 20 perguntas em um arquivo jogo_20_pergunta.c (ou .cpp). O jogo funciona da seguinte forma:

1) O usuário escolhe um objeto
2) O programa deve fazer até 20 perguntas cuja a resposta seja SIM ou NÃO.
3) A situação do jogo é armazenada em uma arvore binária.
4) A primeira pergunta que o programa faz é a que está na raiz da arvore binária.
5) Se o usuário responde SIM, o programa continua com o nó a esquerda fazendo a pergunta correspondente.
6) Se o usuário responde NÃO, o programa continua com o nó a direita fazendo a pergunta correspondente.
7) Se o programa chega em uma folha da arvore e ela não é a resposta, então a resposta não está armazenada na árvore. Neste caso, se deve criar uma pergunta nova que seja capaz de diferenciar entre o objeto na folha que já existia e o novo objeto.
8) Dever ter a opção de se remover um nó (ex. a resposta for considerada errada) e de se remover uma pergunta (ex. pergunta não faz sentido).
9) Deve existir a opção de salvar a arvore em arquivo e a arvore pode ser restaurada de um arquivo.

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

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

O programa testa_arvore.c (ou .cpp) deve testar se a implementação da biblioteca de arvore funciona corretamente. Devem ser tratadas as exceções. Ex: O que acontece quando se tenta destruir uma arvore que não existe?

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 deve ser feito orientado a testes utilizando os frameworks Gtest ou Catch. Utilizar um controlador de versões (ex. git, subversion, mercurial) para fazer o log das mudanças no código.

3) 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.

4) Instrumente o código usando o gcov. 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.

5) 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)

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

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

8) Deve ser gerado um documento (pdf ou odt) que diz como cada função foi testada. Para cada caso deve constar:

8.1) Nome da função, parâmetros e significado dos parâmetros. Especificação da função

8.2) Para cada um dos testes em cada função
  8,2.1) Nome de cada teste
  8,2.2) O que vai ser testado
  8,2.3) Qual deve a ser a entrada
  8,2.4) Qual deve ser a saída
  8,2.5) Qual é o critério para passar no teste
  8,2.6) Se a sua função efetivamente passou no teste ou não

8,3) Responda em que casos a função não retorna um resultado válido.
  8,3.1) Existem funções que podem corromper a estrutura de dados? Como?
  8,3.2) O que pode ser feito para evitar este problema

9) Deve ser enviado o log com as alterações no código mostrando que o desenvolvimento foi orientado a teste.

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/ 4 /18
Pela tarefa na página da disciplina no ead.unb.br