Universidade de Brasília
Departamento de Ciência da Computação
Disciplina: Técnicas de Programação 2
Código da Disciplina: CIC0198

Técnicas de Programação 2 – CIC0198

Trabalho 2

O objetivo deste trabalho é utilizar o desenvolvimento orientado a testes (TDD) para implementar o sistema de backup descrito nos slides (23-26) sobre tabela de decisão na aula sobre testes 2 (caixa fechada).
O sistema deve permitir fazer o backup de todos os arquivos listados no arquivo "Backup.parm". O sistema deve permitir fazer um backup ou uma restauração para um pendrive (se não possuir pendrive pode ser usado um disco ou diretório especificado).

Você deve usar a tabela de decisão para fazer os testes. Para cada coluna da tabela de decisão, deve ser criado um teste.

Você deve criar arquivos para fazer os testes e fazer teste de cobertura do código usando expressões regulares como visto nos slides sobre teste 1 (caixa aberta).

Para as funções ou métodos do seu código, você deve fazer assertivas de entrada e assertivas de saída. As assertivas devem ser feitas de acordo com os slides sobre assertivas nas seguintes páginas: 14, 15, 16 e 21.

Devem ser colocados os comentários de forma similar nos slides (página 14). Ex:

```
/***************************************************************************
* Função: Converter long para ASCII
* Descrição
* Converte um inteiro long para um string ASCII.
* O string resultado estará alinhado à esquerda no buffer de dimASCII
* caracteres fornecido
* Parâmetros
* dimASCII - número máximo de caracteres do string inclusive
* o caractere zero terminal.
* pNumASCII - ponteiro para o espaço que receberá o string.
* Será truncado à direita caso o string convertido
* exceda a dimensão limite. O primeiro caractere
* será '-' se e somente se número < 0
* Numero - inteiro a ser convertido para string
* Valor retornado
* veja as declarações das condições de retorno
* Assertiva de entrada
* pNumASCII != NULL
* dimensao( *pNumASCII ) >= dimASCII
* dimASCII >= max( 3 , 2 + log10( abs( Numero ))
***************************************************************************/
char * BCD_ConverterLongASCII( int dimASCII ,
char * pNumASCII ,
long Numero ) ;
```

Você deve fazer revisões e inspeções no código C/C++ conforme os documentos de checklists enviados.

O desenvolvimento deverá ser feito passo a passo seguindo a metodologia TDD. A cada passo deve-se pensar qual é o objetivo do teste e o significado de passar ou não no teste.

Para cada teste, seguindo a metodologia TDD você deve criar o teste, passar no teste, e refatorar, fazendo pelo menos 3 commits por teste.

Você deve fazer pelo menos um commit para teste que você fizer. Devem ter sido feitos pelo menos 30 commits

O programa deverá ser dividido em módulos e desenvolvido em C ou C++. Deverá haver um arquivo backup.c (ou .cpp) e um arquivo backup.h (ou .hpp). Deverá haver também um arquivo testa_backup.c (ou .cpp) cujo objetivo é testar o funcionamento da biblioteca de backup.

O programa deve usar um makefile para executar a compilação e outros programas.

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) 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 está 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.

2) O desenvolvimento deverá ser feito utilizando um destes frameworks de teste:

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

3) Deverá ser entregue o histórico do desenvolvimento orientado a testes feitos através do git (https://git-scm.com/docs/gittutorial)

git config --global user.name "Your Name Comes Here"
git config --global user.email you@yourdomain.example.com
git init
git add *
git commit –m “teste 1”
git log

Compactar o diretório “.git” ou equivalente enviando ele junto.

4) Deve ser utilizado um verificador de cobertura
ex. 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 verificador de cobertura 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) Utilize um verificador estático
ex. cppcheck, corrigindo os erros apontados pela ferramenta.
Utilize cppcheck --enable=warning .
para verificar os avisos nos arquivos no diretório corrente (.)

Utilize o verificador estático 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) utilizar o verificador dinâmico Valgrind (*valgrind.org/*)

Utilize o verificador dinâmico sempre e desde o início da codificação pois é mais fácil eliminar os problemas logo quando eles aparecem.

7) Deve ser gerada uma documentação do código usando o programa DoxyGen (http://www.stack.nl/~dimitri/doxygen/): O programa inteiro terá de ser documento usando DoxyGen. Comentários que vão ficar na documentação devem ser do estilo Javadoc.
Para gerar uma documentação mais adequada, rodar
doxygen –g
isto irá gerar um arquivo Doxyfile. Neste arquivo, na linha adequada, colocar:

EXCLUDE = catch.hpp

Isto fará com que o catch.hpp não seja documentado. Uma mudança semelhante deverá ser feita para outro framework se necessário.

Depois de feito isto, para documentar basta rodar : doxygen

Verificar se foi gerada a documentação em html

Devem ser enviados para a tarefa no aprender3.unb.br um arquivo zip onde estão compactados todos os diretórios e arquivos necessários. O documento deve estar na raiz do diretório. Todos os arquivos devem ser enviados compactados em um único arquivo (.zip) e deve ser no formato matricula_primeiro_nome ex: 06_12345_Jose.zip. Deve conter também um arquivo leiame.txt que diga como o programa deve ser compilado.

Deve ser enviado o diretório ".git" compactado junto com o ".zip". É ele que comprova que você fez o trabalho orientado a teste. O desenvolvimento orientado a teste será avaliado por ele. Verifique desde o início se tem este diretório e se tem a informação adequada.

Data de entrega:

16/10 /25

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