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

Métodos de Programação - 201600

Trabalho 1

O objetivo deste trabalho é utilizar o desenvolvimento orientado a testes (TDD) para fazer uma calculadora que utiliza uma string como entrada para fazer a soma dos termos separados por um delimitador.

A função deverá ter o formato:

int soma_string(char * string_entrada)

A string poderá conter qualquer quantidade de números. Em cada linha se pode ter 0, 1, 2 ou 3 números. Os números são separados por exatamente um delimitador que no caso é a vírgula. Qualquer separador diferente é considerado erro a menos que tenha sido adicionado como delimitador.

Ex: 1,2\n (valido retorna 3)
3,2,1\n (valido retorna 6)
1,\n (invalido retorna -1)
,2\n (invalido retorna -1)
1,,2\n (invalido retorna -1)
,\n (invalido retorna -1)
1;2\n (invalido retorna -1)
1,2 (sem '\n' no final da string invalido retorna -1)
1,2 \n (espaço no final invalido retorna -1)
1\n,2\n (válido pois o \n é ignorado se estiver entre números retorna 3)
1\n\n\n,4\n (válido pois o \n é ignorado se estiver entre números retorna 5)
1\n\n\n,\n\n\n3\n (válido pois o \n é ignorado se estiver entre números retorna 4)
1,2,3,4\n (invalido pois estoura limite por linha retorna -1)
1,2,3\n,4\n (valido pois não tem mais de 3 números por linha retorna 10)

Números negativos são proibidos. Se houver um número negativo retorna -1
Os números maiores que 1000 são ignorados. Ex. '3,2000\n' dá resultado 3.

Você pode especificar um novo delimitador na primeira linha com "//[delimitador]\n[numeros…]" ex. "//[;]\n2;3\n" faz com que o ';' passe a ser um separador válido assim, dá resultado 5.

Um delimitador pode ter qualquer tamanho
Ex. "//[***]\n2***3***4\n" retorna 9
"//[***]2***3***4\n" é invalido pois falta '\n' retorna -1

A linha que especifica os separadores é opcional mas se existir ela tem de estar no formato correto.

Pode haver qualquer número de delimitadores especificados e de qualquer tamanho.

Ex. "//[delimitador1][delimitador2]\n" for example "//[**][%%%]\n2**1%%%3\n" retorna 6.

Deve ser gerado um executável testa_soma_string_stdin ele lê a string da entrada padrão, executa a função soma_string segundo a especificação acima e imprime o inteiro na saída padrão.

Depois de compilado o programa deve ser executado com
testa_soma_string_stdin < entrada.txt > saida.txt

Ex.
Se o arquivo entrada.txt tem apenas a string "1\n\n\n,\n\n\n3\n", o arquivo saida.txt terá apenas "4"
Se o arquivo entrada.txt tem apenas a string "1,,2\n", o arquivo saida.txt terá apenas "-1"

1) O programa deverá ser dividido em módulos e desenvolvido em C ou C++. Deverá ser feito um makefile como no exemplo do makefile 5 dado em :
(http://www.cs.colby.edu/maxwell/courses/tutorials/maketutor/)
Deverá haver um *header* string_soma.h ou string_soma.hpp com o protótipo da função
Deverá haver um módulo string_soma.c ou string_soma.cpp com a implementação da função int soma_string(char * string_entrada)
Deverá haver um arquivo testa_string_soma.c (ou .cpp) que faz os testes utilizando o *framework* de teste.
Deverá haver um arquivo testa_soma_string_stdin.c (ou .cpp) que lê a string da entrada padrão e o coloca o resultado na saída padrão (não utiliza o *framework* de teste).

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 deverá ser devidamente comentado facilitando o entendimento do mesmo.

2) Faça um documento dizendo quais testes você fez a cada passo e o que passar neste teste significa.

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

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

4) Deverá ser entregue o histórico do desenvolvimento orientado a testes feito através do github (https://github.com/)

5) 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.

6) 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)

7) 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.

8) O Valgrind (http://valgrind.org/) não é obrigatório mas é interessante que seja utilizado

9) O programa deve ser depurado utilizando o GDB se for necessário

Devem ser enviados para a tarefa no ead.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.

Data de entrega:

20/ 9 /17

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