TRABALHO PRÁTICO 1 – TP1

Trabalho idealizado pelo Prof. Dr. Rafael Garibotti

INFORMAÇÕES GERAIS

O objetivo desse trabalho é utilizar na prática os conceitos e ferramentas de teste de software estudados durante o semestre.

- Entrega: 29/09/2023, 17:30 (***Hard deadline***)
- Trabalho deve ser realizado individualmente.

ESPECIFICAÇÃO DO PROBLEMA

Vocês devem testar a implementação de diversos algoritmos de ordenamento desenvolvidos na linguagem C. Foi escolhido os algoritmos desenvolvidos por *Danilo Novakovic* e disponíveis publicamente no GitHub (https://github.com/DaniloNovakovic/sorting-algorithms-in-c), onde as implementações foram separadas em pastas por complexidade (O(n), O(n^2) e O(nlogn)). A fim de facilitar o tratamento dos mesmos, foi desenvolvido uma função (chamada ***sort***) que recebe alguns parâmetros e chama a implementação correspondente do algoritmo de ordenamento em questão. A estrutura básica está disponível no Material de Apoio e o seu protótipo é mostrado abaixo:

- int ***sort***(int* a, int length, char* type, int algorithm);

Note que a função ***sort*** recebe 4 argumentos, o ***a*** é um ponteiro para o vetor de inteiros a ser ordenado. O vetor deve conter de 2 a 20 elementos. O argumento ***length*** mostra o comprimento do vetor passado. O ***type*** indica qual é a complexidade do algoritmo a ser utilizado, as *strings* válidas são: "On", "On2" e "Onlogn". Por fim, o argumento ***algorithm*** indica qual algoritmo de ordenação se pretende utilizar. Note que os algoritmos foram definidos em um *enumerate*, os quais constam as seguintes opções: COUNTING, RADIX, BUBBLE, INSERTION, SELECTION, HEAP, MERGE, QUICK. A função ***sort*** retorna '0' sempre que os parâmetros estão válidos e '1' quando encontrado alguma configuração inválida. Note que a complexidade passada tem que corresponder ao algoritmo escolhido, como definido originalmente por Danilo Novakovic.

RECURSOS UTILIZADOS

Vocês devem utilizar como base o projeto disponível no Material de Apoio. Este projeto apresenta um **Makefile** básico, onde vocês devem alterá-lo para automatizar a compilação, usando como compilador o **clang/gcc**. Primeiramente, vocês devem criar um projeto no *GitHub* e utilizar/configurar a ferramenta de *Continuous Integration* disponível nele. Além disso, vocês devem incrementar o **Makefile** utilizando o **gcov** como ferramenta de análise de cobertura de código, além das seguintes ferramentas de teste: **cppcheck**, **valgrind** e **sanitizer**.

Também devem usar uma ferramenta para descrever os testes. É sugerido o uso do **Unity**, já mostrado em aula, mas este ponto fica de livre escolha caso alguns grupos queiram pesquisar e usar outras ferramentas. Neste sentido, o grupo pode decidir qual ferramenta utilizar para este propósito. Algumas alternativas conhecidas são: **gtest**, **cpptest**, **catch**. Entretanto existem dezenas de outras opções.

Os alunos **podem propor ferramentas adicionais** e isso vai ser altamente valorizado na avaliação.

ENTREGÁVEIS

- Um arquivo zip com o código fonte do repositório e o relatório, **ambos postados no Moodle ANTES do prazo**.
- Os alunos terão que **apresentar** o seu ambiente de projeto, seus testes e os resultados.
- O repositório deve ser totalmente automatizado através do **Makefile** em termos de: compilação, execução do relatório de cobertura, e execução dos testes. A cada *commit*, a ferramenta de *Continuous Integration* deve ser executada para verificar se os testes passaram.
- O relatório deve ser entregue no formado PDF, e conter:
  - Uma tabela com as classes de equivalências e valores limites, além dos testes resultantes do uso destes 2 critérios.
  - Ao descrever os testes no relatório, especifiquem no seguinte formato:

Número do Teste	Nome do Teste	Casos de Teste
1	<nome do teste1>	[{entrada},{saida esperada}]
2	<nome do teste2>	[{entrada},{saida esperada}]
...	...	...
N	<nome do testeN>	[{entrada},{saida esperada}]

  - Especifiquem separadamente os testes adicionais, por exemplo, testes incluídos para aumentar a cobertura de código. Neste caso, além de adotar o mesmo formato para descrever o teste, inclua também uma frase justificando a inclusão desse teste. Por exemplo, ele cobre que parte do código? Que caso relevante que vocês perceberam que foi necessário adicionar e que os critérios de equivalência e valor limite não contemplavam?
- Cobertura: utilizar o **gcov** para cobrir 100% da *cobertura de linhas* e da *cobertura de branch*. Lembrem-se de incluir os resultados obtidos de cobertura no relatório. Note que como estamos utilizando uma implementação "as cegas" de um repositório público do GitHub, as vezes não é possível chegar nessa cobertura, devido a má implementação do desenvolvedor. **Neste caso, os 100% serão adaptados para o maior percentual encontrado pelos colegas.**
- Incluam no relatório as ferramentas adicionais de verificação utilizadas, se este for o caso.
- **Evitem modificações no código fonte original**. Qualquer mudança deve ser relatada e pedido a liberação para prosseguir com a mudança para o Professor, sem a anuência do mesmo, nenhuma modificação é permitida. Contudo, se tais mudanças ocorrerem, as mesmas devem ser relatadas no código fonte com comentários e também no relatório.

AVALIAÇÃO

- [3 pontos] Relatório e apresentação
- [2 pontos] Automação dos scripts
- [3 pontos] Qualidade e completude dos testes
- [2 pontos] Uso adequado das ferramentas indicadas

PS.: A **apresentação é obrigatória!** Logo, não apresentação do trabalho será considerado como **trabalho não entregue**.