INF1413 Teste de Software
Período: 2015-1
Profs. Arndt von Staa

4o. Trabalho

Data de divulgação: 03 de junho (quarta-feira)

Data de entrega: 22 de junho (segunda-feira)

1. Descrição do trabalho

O trabalho visa criar um módulo de teste específico capaz de realizar a geração automática de dados aleatórios de teste e aplicá-los a um programa C++ fornecido. Visa também exercitar, de forma simplificada, a manutenção dirigida por testes.

2. Descrição do quarto trabalho

2.1. Preparação:

1. Extraia o arcabouço Talisman de apoio ao teste de programas escritos em C++ que se encontra zipado no pacote do 4º. trabalho. Descompacte-o em um diretório de trabalho, por exemplo **`c:\Talisman`**. No pacote existe um diretório **`Test\Docs`**. Este diretório contém uma documentação simples e instruções para inicializar as variáveis de ambiente. Siga as instruções deste documento. São necessários um compilador C++ – o pacote está configurado para o Visual Studio – e também o pacote **`Lua`** – compilador e interpretador.
2. Leia a documentação fornecida e ajuste o ambiente da janela **`CMD`** conforme explicado.
3. No diretório **`...\Test\Batches`** existe um batch file **`DoAll.bat`**. Execute este batch file. Caso não funcione, é necessário verificar se todas as variáveis de ambiente estão corretamente inicializadas, se o pacote Lua está devidamente instalado e se os diretórios utilizados por **`DoAll.bat`** são consistentes com a estrutura de diretórios da instalação.

2.2. Trabalho:

1. (**2 pontos**) (Ação: criar um *clone* do módulo a ser testado) Queremos testar o módulo **`SYMBTAB`** (tabela de símbolos) que compõe o arcabouço de teste. Como o teste de regressão do arcabouço de apoio ao teste automatizado utiliza os módulos que o compõem, entre eles **`SYMBTAB`**, torna-se impossível testar mutantes de tais módulos, uma vez que os mutantes gerariam falhas ao executar o arcabouço*. Para resolver este problema é necessário gerar um *clone* do módulo **`SYMBTAB`** bem como do contexto utilizado para testá-lo. Crie uma cópia do módulo **`SYMBTAB`** com o nome **`TSYMBTAB`** e renomeie as classes, **`SMT_CSymbolTable`** (classe que implementa a tabela de símbolos genérica) para **`SMT_TCSymbolTable`** e **`SMTE_CSymbolTableElement`** (classe que define o elemento a ser usado ao testar a tabela de símbolos) para **`SMTE_TCSymbolTableElement`**. Crie uma cópia do módulo de teste específico **`TST_SMT`** e

* Este problema é uma instância de um "Heisenbug" (Wikipedia). O nome é um trocadilho envolvendo Heisenberg, criador da Mecânica Quântica. O Heisenbug é uma ocorrência do "efeito do observador", que estipula que o ato de observar pode alterar o estado do objeto sendo observado. No nosso caso a adição de instrumentos (os mutantes) altera o funcionamento do arcabouço fazendo-o falhar. Porém o arcabouço é exatamente o observador que deveria ser cabaz de observar a falha.

batize-a de **TST_TSMT** Altere o módulo de teste específico **TST_TSMT** para que teste este novo módulo **TSYMBTAB**. Crie uma cópia do descritor de construto de teste **TST_SMT.comp** e batize-a de **TST_TSMT.comp**. Adicione instruções e ajuste o arquivo **TST_TSMT.comp** para que consiga realizar o teste do o módulo cópia (clone) **TSYMBTAB** (são 4 usos de **tst_smt** que precisam ser ajustado). Ajuste o descritor de recompilação geral **TestFramewok.suite** de modo que teste o módulo **SYMBTAB** original como o módulo **TSYMBTAB** clonado. Teste se tudo funciona com o batch **DoAll.bat**. Assegure-se que foram testados tanto **SYMBTAB** como **TSYMBTAB**. Após executar o **DoAll.bat** verifique se a linha de comando **test tsmt 01** funciona. Esse comando gera uma chamada ao programa **TST-TSMT** usando o script de teste **TST-TSMT-01.script**. Observações: a linha de comando **compile TST-tsmt** recompila o programa desde que o *make* esteja correto. A linha de comando **genmake TST-tsmt** cria um arquivo *make* coerente com o descritor do construto.

2. **(2 pontos**) (Ação: completar o gerador/executor de casos de teste) No módulo de teste específico **TST-SMT** complete a classe **GenerateTest** que é utilizada para gerar e executar automaticamente os casos de teste. A alteração visa completar a geração e a execução de comandos de teste de *inserção*, *busca* e *destruição* de símbolos. A sintaxe dos comandos para o gerador será apresentada mais adiante. Compile e teste usando o script **TST-tsmt-02.script**, assegurando-se que a alteração está sintaticamente correta. Este teste somente verifica se a classe **GenerateTest** está funcionando. Certifique-se que o número de ações realizadas é igual à soma das estatísticas de cada tipo de ação.

3. (**2 pontos**) (Ação: completar a medição). No módulo **TSYMBTAB** adicione um método para computar as estatísticas básicas de uma tabela de símbolos. O módulo deve listar o número de símbolos para cada uma das listas de colisão existentes na tabela. Adicione ao módulo de teste específico uma chamada a esse método. Compile e execute o teste.

4. (**1 ponto**) (Ação: medir cobertura do resultado) Modifique os parâmetros de geração no script de teste de modo que cada contador de ação e de lista de colisão tenha um valor maior do que 10. Quais foram os critérios que você utilizou para assegurar isso? Por que isso foi difícil, ou não foi?

5. (**1 ponto**) (Ação: medir eficácia) Defina pelo menos três operadores de mutação. Usando esses operadores crie cinco mutantes no método **InsertSymbol**. Os mutantes devem ser criados de forma similar ao que está na *transparência 7 da aula 21*. Mutantes equivalentes e/ou escolhas simplórias causam perda de pontos. Deixo de propósito ambíguo o termo "simplório". Verifique se os mutantes são mortos pelo script original **tst-smt-01.script**. Se sobrar algum mutante não "morto", explique as adições realizadas neste script para que passe a "matar" todos os mutantes.

6. (**1 ponto**) (Ação: meça a eficácia do script usando o gerador) Verifique se todos mutantes são "mortos" pelo script de teste **tst-smt-02.script**. Ajuste o script caso não sejam. Explique o ajuste feito.

7. (**1 ponto**) (Ação estudar o efeito da escolha do tamanho da tabela de randomização) Adicione um script de geração gerando 500 símbolos e executando 1000 ações. Enfatize a inserção de símbolos tornando a frequência de inserção maior do que 90%. Execute o teste usando vários tamanhos de tabela de randomização. Devem ser usados 31 , 32 , 512 e 519. O que você observou nas estatísticas da tabela de símbolos?

O bloco de dados do gerador e executor automatizado de teste deve obedecer à seguinte sintaxe:

```
=GenerateTest    condRetorno
   +Controle          dimTabela  numSimbolos  numParmFreq  modoGeracao
   +Generate          numAcoes   numVerifica
   +ActionDistrib     inserção   busca   exclusão
   +SizeDistrib       frequência  limiteSuperior
   +ParameterListEnd
```

=GenerateTest é o comando que dispara a execução do teste com geração automática

+Control determina como é controlada a verificação durante a execução

dimTabela é o número de listas de colisão que devem existir na tabela. Experimente com números primos e com expoentes inteiros de 2.

numSimbolos é o número de símbolos que devem ser gerados na tabela dos símbolos a serem utilizados. A cada ação é escolhido randomicamente um desses símbolos a ser usado ao realizar a ação. A tabela contendo os símbolos gerados também mantém o fato do símbolo estar ou não na tabela de símbolos sendo testada. Dessa forma pode-se verificar se o resultado da ação é o resultado esperado.

numParmFreq é o número de linhas contendo parâmetros de frequência de tamanho dos símbolos.

modoGeracao seleciona a semente do gerador de números aleatórios. Zero corresponde a usar a semente padrão; 1 corresponde a usar uma semente gerada e diferente a cada vez que o módulo de números aleatórios for inicializado. Se for > 1 o número fornecido é utilizado como semente.

+Generate indica quantas ações devem ser executadas e com que frequência devem ser executadas as verificações das assertivas estruturais

numAcoes é o número de ações a serem geradas

numVerifica é o número de ações a realizar, ao final das quais deve ser ativada o método **Verify**. Este método deve ser ativado também ao terminar a geração. Antes de iniciar a verificação deve ser gerada uma linha contendo o número de ações efetuadas até o momento. Além de evidenciar o progresso do processamento, também permite determinar a última verificação bem sucedida antes da observação de uma falha, caso ocorra algum problema.

+ActionDistrib controla a frequência com que são selecionadas ações de inserção, busca e exclusão. A soma das frequências deve dar 100.

inserção frequência com que será escolhida a ação de inserção – **InsertSymbol**.

busca frequência com que será escolhida a ação de busca – **SearchSymbol**.

exclusão frequência com que será escolhida a ação de exclusão – **DeleteSymbol**.

+SizeDistrib é um conjunto de uma ou mais linhas que descrevem a distribuição de tamanhos de cada símbolo. No script de teste devem aparecer tantos comandos **+SizeDistrib** quanto indicado no parâmetro **numParmFreq** do comando **+Control**.

frequência indica a frequência com que o descritor deve ser selecionado. A soma das frequências do conjunto de descritores deve dar 100

`limiteSuperior` determina o limite superior do intervalo de tamanho correspondente ao descritor. O limite inferior é dado pelo limite superior da linha anterior mais um. A primeira linha tem 0 como limite inferior. Dentro do intervalo [limite inferior, limite superior] é selecionado um tamanho segundo a distribuição uniforme. Este tamanho é utilizado para criar um símbolo de **`tamanho`** caracteres, em que cada caractere é escolhido aleatoriamente segundo a distribuição uniforme do conjunto de letras ASCII minúsculas.

`+ParameterListEnd` este comando sinaliza o término da lista de distribuições de tamanho.

3. Entrega do Trabalho

O trabalho deve ser realizado em grupos de 2 ou 3 alunos.

Devem ser entregues **os códigos fonte e executáveis do módulo sob teste e do módulo de teste específico**, e, ainda, os *scripts* e resultados de teste gerados no decorrer da elaboração de cada um dos passos do trabalho. Deve ser entregue também um arquivo **`.DOC`** (ou **`.DOCX`**) contendo as observações e comentários para cada um dos passos acima.

Além dos arquivos acima, deve ser entregue um relatório de tempo despendido no trabalho. O relatório consta de uma tabela de registro de trabalho do grupo organizada como a seguir:

Data	Horas Trabalhadas	Quem trabalhou	Tipo Tarefa	Descrição da Tarefa Realizada

A coluna *Quem trabalhou* identifica 1 ou mais membros do grupo que realizaram a tarefa. Na descrição da tarefa redija uma explicação breve sobre o que foi feito. Esta descrição deve estar de acordo com o Tipo Tarefa. Cada Tipo Tarefa identifica uma natureza de atividade que deverá ser discriminada explicitamente, mesmo que, durante uma mesma sessão de trabalho tenham sido realizadas diversas tarefas. **Crie e justifique a lista de naturezas de tarefas**.

4. Critérios de correção básicos

- Clareza da exposição e qualidade do código produzido.