Trabalho Prático – Script de Teste Automático

O trabalho prático da disciplina será avaliado em duas etapas, sendo a primeira uma avaliação objetiva, com testes automáticos. Foi disponibilizado aos alunos um script para execução de alguns testes automáticos, sendo portanto possível garantir que o trabalho passa nesses testes antes de submetê-lo ao professor.

Este documento explica como preparar seu trabalho para execução do script.

1. Obtendo e executando o script

O script de teste funciona somente em ambientes Linux/MacOS e foi testado no LabGrad. Recomenda-se fortemente que os testes finais do seu trabalho sejam feitos no LabGrad, pois as versões das ferramentas instaladas nas máquinas do laboratório serão consideradas como versões de referência para a correção do trabalho.

Para obter e executar o script Java, siga os passos abaixo:

1. Na página da disciplina, obtenha o arquivo teaching-br-pac-20141-script-java.zip;
2. Descompacte o arquivo em alguma pasta do seu sistema;
3. Abra um terminal, acesse esta pasta;
4. Execute o script: `./test.sh` e o resultado deve ser parecido com a saída abaixo:

```
$ ./test.sh
Script de teste PAC 2014/1 - Trabalho Java

$
```

O script C++ funciona de maneira idêntica, substituindo `java` por `cpp` no nome do arquivo disponibilizado no site.

Ambos os scripts reconhecem trabalhos se forem colocados em pastas no mesmo diretório em que se encontra o script `test.sh` e se os trabalhos seguirem as instruções contidas nas seções 2 e 3, abaixo.

Note que há já uma pasta `testes`, que contém os arquivos de teste executados pelo script. O script já é configurado a não considerar esta pasta como uma pasta de trabalho.

No exemplo abaixo, o comando `ls` mostra que há um diretório professor dentro do qual encontra-se a implementação do trabalho do professor. Em seguida, mostra a execução do script de testes sem erro algum:

```
$ ls -Flpah
total 8
drwxr-xr-x   5 vitor  wheel   170B May 30 18:00 ./
drwxr-xr-x@ 55 vitor  wheel   1.8K May 30 17:54 ../
drwxr-xr-x   4 vitor  wheel   136B May 30 12:30 professor/
-rwxr-xr-x@  1 vitor  wheel   2.0K May 30 17:57 test.sh
drwxr-xr-x   6 vitor  wheel   204B May 30 17:24 testes/

$ ./test.sh
```

```
Script de teste PAC 2014/1 - Trabalho Java

[I] Testando professor...
[I] Testando professor: teste 01
[I] Testando professor: teste 01, tudo OK em 1-apagar.csv
[I] Testando professor: teste 01, tudo OK em 2-areceber.csv
[I] Testando professor: teste 01, tudo OK em 3-vendasprod.csv
[I] Testando professor: teste 01, tudo OK em 4-vendaspgto.csv
[I] Testando professor: teste 01, tudo OK em 5-estoque.csv
[I] Testando professor: teste 02
[I] Testando professor: teste 02, serialização OK!
[I] Testando professor: teste 03
[I] Testando professor: teste 03, serialização OK!
[I] Testando professor: teste 03, tudo OK em 1-apagar.csv
[I] Testando professor: teste 03, tudo OK em 2-areceber.csv
[I] Testando professor: teste 03, tudo OK em 3-vendasprod.csv
[I] Testando professor: teste 03, tudo OK em 4-vendaspgto.csv
[I] Testando professor: teste 03, tudo OK em 5-estoque.csv
[I] Testando professor: pronto!

$
```

2. Testando o trabalho Java

O arquivo compactado do trabalho Java, a ser enviado por e-mail conforme descrito na especificação do trabalho deve conter os arquivos fonte e um arquivo de *build* do Ant. Ele não deve conter classes compiladas (.class). Estas devem ser geradas pelo software de *build* (construção de programas) Apache Ant (http://ant.apache.org).

Os arquivos fonte podem estar organizados da forma que você achar melhor, desde que o Ant consiga compilá-los, executar as classes geradas e limpar o projeto. Para que isso seja feito de forma automatizada, o arquivo de *build* do Ant deve, obrigatoriamente, encontrar-se na raiz do arquivo compactado e chamar-se `build.xml`. Além disso, ele deve ser feito de forma a responder aos seguintes comandos:

Comando	Resultado esperado
`ant compile`	O código-fonte deve ser compilado, gerando os arquivos .class para todas as classes do trabalho.
`ant run`	O programa deve ser executado especificando as opções `-c clientes.csv -f fornecedores.csv -p produtos.csv -a compras.csv -v vendas.csv` como parâmetro.
`ant run-read-only`	O programa deve ser executado no modo `--read-only` (ver especificação do trabalho), especificando além disso as mesmas opções do comando `ant run` acima.

Comando	Resultado esperado
`ant run-write-only`	O programa deve ser executado no modo `--write-only` (ver especificação do trabalho).
`ant clean`	Todos os arquivos gerados (classes compiladas, relatórios de saída, arquivo de serialização) e eventuais arquivos de entrada de dados devem ser excluídos, sobrando somente o conteúdo original do arquivo compactado (ou seja, o código-fonte e o arquivo de *build*).

Caso você não implemente as opções read-only e write-only, faça com que os respectivos comandos funcionem da mesma forma que o comando run, ou seja, efetuem a execução nromal.

Segue abaixo um exemplo de arquivo build.xml que atende às especificações acima. Em negrito encontram-se marcados os dados que devem ser adaptados dependendo do projeto:

- Subpasta onde encontra-se todo o código-fonte;
- Subpasta onde serão colocadas as classes compiladas;
- Nome da classe principal do programa, ou seja, aquela que possui o método `main()`.

```
<project name="TrabalhoPAC_2014_1" default="compile" basedir=".">
    <description>Arquivo de build do trabalho de PAC, 2014/1.</description>

    <!-- Propriedades do build. -->
    <property name="src" location="src" />
    <property name="bin" location="bin" />
    <property name="mainClass" value="meupacote.MinhaClassePrincipal" />

    <!-- Inicialização. -->
    <target name="init" description="Inicializa as estruturas necessárias.">
        <tstamp/>
        <mkdir dir="${bin}" />
    </target>

    <!-- Compilação. -->
    <target name="compile" depends="init" description="Compila o código-
fonte.">
        <javac includeantruntime="false" srcdir="${src}" destdir="${bin}" />
    </target>

    <!-- Execução normal. -->
    <target name="run" depends="compile" description="Executa o programa
principal, em modo normal.">
        <java classname="${mainClass}">
            <arg value="-c" />
            <arg value="clientes.csv" />
            <arg value="-f" />
            <arg value="fornecedores.csv" />
            <arg value="-p" />
            <arg value="produtos.csv" />
            <arg value="-a" />
            <arg value="compras.csv" />
```

```
                <arg value="-v" />
                <arg value="vendas.csv" />
                <classpath>
                    <pathelement path="${bin}" />
                </classpath>
            </java>
        </target>

        <!-- Execução somente leitura. -->
        <target name="run-read-only" depends="compile" description="Executa o
programa principal, em modo somente leitura.">
            <java classname="${mainClass}">
                <arg value="-c" />
                <arg value="clientes.csv" />
                <arg value="-f" />
                <arg value="fornecedores.csv" />
                <arg value="-p" />
                <arg value="produtos.csv" />
                <arg value="-a" />
                <arg value="compras.csv" />
                <arg value="-v" />
                <arg value="vendas.csv" />
                <arg value="--read-only" />
                <classpath>
                    <pathelement path="${bin}" />
                </classpath>
            </java>
        </target>

        <!-- Execução somente escrita. -->
        <target name="run-write-only" depends="compile" description="Executa o
programa principal, em modo somente escrita.">
            <java classname="${mainClass}">
                <arg value="--write-only" />
                <classpath>
                    <pathelement path="${bin}" />
                </classpath>
            </java>
        </target>

        <!-- Limpeza. -->
        <target name="clean" description="Limpa o projeto, deixando apenas o
código-fonte." >
            <delete dir="${bin}"/>
            <delete><fileset dir="." includes="*.csv"/></delete>
            <delete><fileset dir="." includes="*.dat"/></delete>
        </target>
</project>
```

3. Testando o trabalho C++

O arquivo compactado do trabalho C++, a ser enviado por e-mail conforme descrito na especificação do trabalho deve conter os arquivos fonte e um arquivo de *build* do Make (ou seja, um Makefile). Ele não deve conter arquivos objeto (.o). Estes devem ser gerados pelo make (http://www.gnu.org/software/make/).

Os arquivos fonte podem estar organizados da forma que você achar melhor, desde que o make consiga compilá-los, executar as classes geradas e limpar o projeto. Para que isso seja feito de forma automatizada. O script espera que o seu Makefile seja preparado para responder aos seguintes comandos:

Comando	Resultado esperado
`make`	O código-fonte deve ser compilado, gerando o programa executável.
`make run`	O programa deve ser executado especificando as opções `-c clientes.csv -f fornecedores.csv -p produtos.csv -a compras.csv -v vendas.csv` como parâmetro.
`make clean`	Todos os arquivos gerados (classes compiladas, relatórios de saída) e eventuais arquivos de entrada de dados devem ser excluídos, sobrando somente o conteúdo original do arquivo compactado (ou seja, o código-fonte e o Makefile).

Para o trabalho de C++ será considerado para a correção subjetiva se o aluno utilizou compilação separada. Por exemplo, suponha que o trabalho seja composto pelo arquivo main.cpp e uma classe chamada MinhaClasse (dividida em MinhaClasse.h e MinhaClasse.cpp). Este seria um exemplo de Makefile sem compilação separada (que acarretará perda de pontos no critério subjetivo):

```
all:
	g++ -o main main.cpp MinhaClasse.cpp

run:
	./main -c clientes.csv -f fornecedores.csv -p produtos.csv -a compras.csv -
v vendas.csv

clean:
	rm -rf main *.csv *.o
```

Enquanto este seria um exemplo de Makefile com compilação separada (OK):

```
all: main

main: main.cpp MinhaClasse.h MinhaClasse.o
	g++ -o main main.cpp MinhaClasse.o

MinhaClasse.o: MinhaClasse.h MinhaClasse.cpp
	g++ -c MinhaClasse.cpp

run:
	./main -c clientes.csv -f fornecedores.csv -p produtos.csv -a compras.csv -
v vendas.csv

clean:
	rm -rf main *.csv MinhaClasse.o
```

4. Pontos extra[1]

Será dado 1 ponto extra ao grupo que preparar e enviar ao professor até o prazo do trabalho 1 (Java) um conjunto de arquivos de entrada (clientes.csv, fornecedores.csv, produtos.csv, compras.csv e vendas.csv) que atenda aos seguintes critérios:

- Não conter trechos iguais a outros arquivos de teste disponíveis no site;
- Conter o cadastro de pelo menos 10 clientes, 5 fornecedores e 15 produtos;
- Conter o registro de ao menos 30 compras e 30 vendas, sendo que nestes registros devem estar associados ao menos 7 clientes, 4 fornecedores, 10 produtos e 3 modos de pagamento.

Os arquivos de teste enviados poderão, a critério do professor, ser disponibilizados aos demais alunos como parte do script de testes. Atualizações do script serão divulgadas em sala de aula.

5. Observações finais

Caso haja algum erro neste documento, serão publicadas novas versões e divulgadas erratas em sala de aula. É responsabilidade do aluno manter-se informado, frequentando as aulas ou acompanhando as novidades na página da disciplina na Internet.

[1] Apesar dos pontos extras permitirem que a nota do trabalho Java ultrapasse o valor máximo de 10 pontos, no cálculo da média parcial do aluno, a nota máxima continua sendo 10, não podendo ser ultrapassada.