Scripts de test Python3 pour les increments 1 et 2 du projet informatique 2022-23

1. Introduction au script python3 `execute_tests.py`

`execute_tests.py` est un script python3 qui exécute une suite de tests spécifiée dans des fichiers d'extension `.test`
et génère un rapport en comparant :

- le code d'erreur (*return code*) du programme testé ;
- les sorties (`stdout` et `stderr`) du programme testé

avec un oracle spécifié dans le fichier `.test` .

Il détecte également les erreurs de segmentation.

Il dispose d'une option permettant d'exécuter le programme à tester dans `valgrind` et de rapporter les erreurs `valgrind` .

2. Installation / prérequis

Ce script nécessite `python3` et le package python3 `pexpect` et `termcolor` .
Il est également conseillé de disposer du comparateur de fichiers texte `meld` .

Si `python3` et `pip3` sont déjà installés, pour configurer sur une machine école ou la machine virtuelle Phelma, exécuter les commandes suivantes :

```
pip3 install pexpect --user
pip3 install termcolor --user
sudo yum install meld
```

Les deux premières commandes installent les package python3 nécessaires au niveau "utilisateur" (ces commandes devront être répétées sur chaque compte utilisateur). La dernière commande installe le comparateur de fichier texte `meld` *.*

3. Structure d'un fichier `.test`

Le script utilise des fichiers d'extension `.test`, qui doivent respecter une syntaxe spécifique. Un fichier `.test` spécifie un ou plusieurs tests à effectuer, déterminés par la liste des arguments à passer au programme à tester, ainsi que les résultats attendus (**"oracles"**) pour chacun de ces tests.
Voici un exemple de fichier `.test` contenant 2 tests, pour le programme de matching d'expressions régulières fourni au début du projet, qui donc prend 2 arguments en ligne de commande :

```
# TEST START ##########################################################################
# Test args              : 'a*'      'aabbbccccddddd'
# Test return code       : EXIT_SUCCESS
# Test stdout and stderr :
The start of 'aabbbccccddddd' is a*, next: 'bbbccccddddd'.
# END Test stdout and stderr
# TEST END   ##########################################################################

# TEST START ##########################################################################
# Test args              : '[a-z'   'abcdef_1234'
# Test return code       : EXIT_FAILURE
# Test stdout and stderr :
Invalid regexp : '[a-z' is an unterminated group of chars.
# END Test stdout and stderr
# TEST END   ##########################################################################
```

Les éléments importants dans ce fichier sont les suivants (attention à respecter la syntaxe des balises !) :

- `# TEST START` et `# TEST END` : indiquent respectivement le début et la fin de la spécification d'un test ;
- `# Test args :` : les arguments à passer au programme à tester pour exécuter le test ;
- `# Test return code` : le *code de retour attendu du programme*, qui sera comparé au *code de retour effectif* obtenu lors de l'exécution du programme à tester avec les arguments du test. Doit valoir :
  - soit `EXIT_SUCCESS` si le programme est censé se terminer sans erreur, c'est à dire avec un code erreur nul, (*e.g.* avec `exit(EXIT_SUCCESS);`) ;
  - soit `EXIT_FAILURE` si le programme est censé se terminer avec un code erreur non nul (*e.g.* avec `exit(EXIT_FAILURE);`).
- `# Test stdout and stderr :` et `# END Test stdout and stderr` : toutes les lignes qui figurent entre ces deux balises constituent la *sortie attendue du programme*. Cette sortie regroupe la sortie et l'erreur standard (`stdout` et `stderr`). Elle sera comparée à la *sortie effective* obtenue lors de l'exécution du programme à tester avec les arguments du test.

4. Exécuter des tests

Le script Python prend en paramètre un **répertoire** contenant des fichiers `.test`.

Il génère une sortie, un **répertoire de même nom, suffixé par** `_result`, qui contient les résultats des tests.

Par exemple, si le répertoire en entrée s'appelle `mes_tests`, le répertoire créé en sortie s'appellera `mes_tests_result`.

Ces deux répertoires peuvent être comparés avec un comparateur de fichier texte, typiquement `meld` sous Linux.

Exemple d'exécution du script de test

On suppose dans cet exemple :

- que le programme à tester est `../../bin/regexp.exe` ;
- que le répertoire contenant les fichiers `.test` est `./00_test_regexp_basic`.

Pour exécuter les tests, il faut lancer la commande :

```
./execute_tests.py runtest ../../bin/regexp.exe ./00_test_regexp_basic
```

Le script s'exécute alors sur chacun des tests trouvés dans chacun des fichiers `.test` du répertoire `./repertoire_des_tests`.

Plus précisément, pour chacun des tests qui se trouve dans l'un des fichiers de test d'extension `.test` du répertoire `./00_test_regexp_basic` :

- Le script exécute le programme `../../bin/regexp.exe` avec les arguments du test ;
- Le script écrit les résultats dans un autre fichier `.test`, créé dans un nouveau répertoire nommé `./00_test_regexp_basic_result` (<= repérer `_result` à la fin du nom du répertoire résultat).

Puis, à l'issue de tous les tests, le script écrit dans le terminal un résumé de tous les tests réalisés, en précisant les tests qui passent (code erreur et sorties conformes à ce qui est attendu), les tests qui ne passent pas (code erreur OU sortie non conforme(s) à ce qui est attendu), les tests qui ont généré une erreur de segmentation, *etc*.

5. Générer un fichier `.test` pour ajouter des tests à la base de tests

Le script permet également de générer un fichier `.test` à partir d'une série de paramètres placés dans un fichier d'extension `.testarg`. Cela est utile pour ajouter de nouveaux tests à votre base de tests.

Ce mode utilise des fichiers d'extension `.testarg`, qui sont très simples : ils contiennent, sur chaque ligne, des arguments à utiliser pour un test.

Exemple de fichier `.testarg` :

```
# fichier .testarg permettant de générer un fichier .test au moyen du script python
# Ceci est un commentaire commençant par # => ignoré.
# Attention : le commentaire doit etre en début de ligne !

# 3 séries de paramètres pour générer
# un fichier .test contenant 3 tests
# avec le script execute_tests.py.


'aa' 'aabbcd'

'a*' 'aabbcd'

'a*b*cde' '0123'
```

La commande à exécuter pour générer le fichier `.test` correspondant ressemble alors à ce qui suit (repérer l'option `gentest`) :

```
./execute_tests.py gentest ../../bin/regexp.exe ./chemin/monfichier.testarg
```

Cette commande va exécuter votre programme `../../bin/regexp.exe` pour chaque série d'arguments (3 dans le cas du fichier `.testarg` qui précède), puis générer un fichier `.test` nommé `./chemin/monfichier.test`.

Ce fichier `.test` peut, ultérieurement, être utilisé comme oracle.

Attention : bien sûr, avant d'utiliser le fichier `.test` ainsi généré comme un oracle, il vous revient de vérifier "à la main", et avec précision, que les sorties générées par votre programme et placées dans le fichier `.test` sont bien correctes !!

6. Options du script de test

Pour plus de détails sur les options disponibles dans le script de test, visitez l'aide en ligne :

```
./execute_tests.py -h
./execute_tests.py runtest -h
./execute_tests.py gentest -h
```