Test de programmes

Aujourd'hui c'est vous l'enseignant : cinq cent étudiants vous ont chacun rendu un programme pour résoudre un problème que vous leur avez posé. Il va falloir déterminer quels étudiants vous ont fourni un programme correct !

Prologue - Objectif

L'objectif de cette apnée est de conduire une réflexion sur la manière de tester le bon fonctionnement d'un programme et de proposer un ensemble de procedures de tests automatisées adaptées à un jeu de programmes fournis. La mise en place de telles procedures de test automatique est indispensable dans le développement de logiciels de grande taille ou critiques. Dans cette apnée, le nombre de programmes fournis est volontairement trop important pour permettre le test manuel de chacun d'entre eux, il faudra donc écrire un ensemble de programmes réalisant ces tests automatiquement.

1 - Un problème de parenthésage

Nous disposons dans cette partie de programmes (rappel: `unzip programs.zip`) qui cherchent à résoudre le problème du parenthésage correct : étant donné un texte en entrée, ils doivent dire si le texte contient autant de parenthèses fermantes que de parenthèses ouvrantes et si tout préfixe du texte contient autant ou plus de parenthèses ouvrantes que de parenthèses fermantes. La spécification du comportement de ces programmes est la suivante :

- lit les caractères qui lui sont donnés en entrée standard jusqu'à la fin de fichier.
- si l'ensemble des caractères lus forme un texte correctement parenthésé, alors affiche :

  ```
  Bon parenthesage
  ```

  sinon affiche :

  ```
  Mauvais parenthesage
  ```

Pour tester le bon comportement d'un programme, il faudra l'executer avec plusieurs sortes de textes d'entrée (couvrant si possible toutes les variantes de textes bien ou mal parenthésés) et vérifier pour chacune que le programme affiche bien le résultat attendu. Pour notre problème du parenthésage, nous penserons en particulier à tester nos programmes avec :

- une séquence avec trop de parenthèses ouvrantes
- une séquence avec trop de parenthèses fermantes
- des séquences avec un ou plusieurs groupes de parenthèses imbriquées
- une séquence vide

- ...

En plus des parenthèses usuelles, nos programmes gèrent deux autres types de parenthèses : les crochets et les accolades. Non seulement le texte en entrée doit comporter pour chaque type le même nombre de parenthèses fermantes que de parenthèses ouvrantes mais l'imbrication doit également être correcte.

De manière formelle, nous pouvons définir recursivement un texte correctement parenthésé lorsque nous utilisons plusieurs types de parenthèses, il s'agit d'un texte vérifiant l'une des conditions suivantes :

- il est constitué d'une paire de parenthèses de même type entourant un texte correctement parenthésé (une ouvrante au début et une fermante à la fin)
- il est constitué par la concaténation de textes correctement parenthésés
- c'est un texte sans parenthèses

A titre d'exemple, les textes suivants sont bien parenthésés :

- (salut)[(){()}]fin
- ({[({[]})]})

et les suivants sont mal parenthésés :

- (salut)[(){()]}fin
- ({[({[]})]})

2 - Travail à réaliser

Votre rôle principal est, pour l'ensemble des programmes fournis, de déterminer s'ils résolvent correctement le problème du parenthésage exposé dans la section précédente. Parmi ces programmes, certains se comportent tous de la même manière, dans le cas de programmes incorrects, cela signifie qu'ils contiennent le même bug. En plus de la correction, vous devrez déterminer combien de sortes de comportements différents existent parmis les programmes fournis.

EXEMPLE ELEMENTAIRE : Pour vous aider à démarrer dans l'implémentation d'un moteur de tests, nous vous fournissons l'exemple suivant. Supposons que nous disposions d'un programme `programme1`, que nous ayons écrit un texte `test1.txt` à lui fournir en entrée pour le tester et que nous connaission l'affichage que doit produire le programme s'il est correct (`Entree trop longue` par exemple), alors le script suivant :

```
echo "Entree trop longue" >resultat_attendu
./programme1 <test1.txt >resultat
if diff resultat resultat_attendu
then
    echo OK
else
    echo XXXX
fi
```

teste si `programme1` affiche bien le résultat attendu lorsqu'on lui fourni le texte `test1.txt` en entrée. Le script précédent n'est bien entendu qu'un exemple, et vous êtes fortement encouragés à écrire un programme plus général permettant d'automatiser les tests. Une bonne automatisation des tests pourra en particulier tester un programme donné pour un ensemble d'entrées distinctes facilement extensible.

IMPORTANT :

- le choix du langage à utiliser pour cette apnée est libre. Cependant, pour automatiser vos procedures de test, nous vous déconseillons d'utiliser un langage comme le C, qui n'est pas très adapté à la manipulation de fichiers. Utilisez plutôt un langage de script comme le bourne shell qui est plus approprié à ce genre de choses. Faire la même chose que l'exemple précédent en C est nettement plus compliqué.
- **Tous les programmes fournis ont été compilés sur `mandelbrot` et fonctionnent donc sur cette machine. Ils doivent également fonctionner sur la plupart des linux.**

Pensez à organiser votre travail :

- pour chaque test que vous allez concevoir ajoutez en commentaire un descriptif du test : il faudra en particulier commenter l'entrée fournie au programme et expliquer quel mauvais comportement elle cherche à identifier ;
- si vos programmes de test automatique sont bien conçus, il ne doit pas être nécessaire de les modifier pour ajouter un nouveau test ou appliquer l'ensemble des tests disponibles à de nouveaux programmes ;
- regroupez de manière synthétique les résultats de vos test, c'est-à-dire :
  - la description des différents comportements que vous avez pu identifier dans les programmes fournis
  - pour chacun des comportements identifiés (y compris le comportement correct, répondant à la specification), le nombre de programmes exhibant ce comportement.

3 - Tests stochastiques

Dans cette dernière partie, si vous ne l'avez pas déjà spontanément fait dans la partie 2, nous allons nous intéresser à la génération automatique de séquences d'entrée pour le problème du parenthésage simple. En effet, même pour des programmes relativement simples, il est rapidement trop difficile d'imaginer et d'écrire manuellement suffisamment de cas pour tester de manière satisfaisante le respect d'une spécification. Le recours à la génération aléatoire de couples (entrée, sortie attendue) peut permettre alors de couvrir une partie plus large de l'ensemble des entrées possible et d'indentifier des bugs qui seraient passés inaperçus autrement. Ceci est particulièrement vrai pour les bugs associés aux séquences de grande taille en entrée. Dans cette partie nous nous contentons de manipuler des caractères, le choix du langage reste libre, mais le C redevient un bon candidat.

Proposez des méthodes pour :

- générer un texte aléatoire (longueur et contenu determinés aléatoirement) correctement parenthésé.
- générer un texte aléatoire contenant trop de parenthèses ouvrantes.
- générer un texte aléatoire contenant autant de parenthèses ouvrantes que de parenthèses fermantes mais dont un préfixe contient un excès de parenthèses fermantes.

Proposez une implémentation dans le langage de votre choix de chacune de vos méthodes. Si vous utilisez le C, vous aurez probablement besoin de la fonction `random` :

- chaque appel à la fonction `random` produit une valeur pseudo-aléatoire différente de celle du précédent appel.
- une seule fois dans le programme, avant d'utiliser `random`, il faut appeler la fonction `srandom` en lui donnant une graine. Pour une graine donnée, la séquence des valeurs produites par les appels successifs à `random` sera toujours la même, cela permet de reproduire une expérience incluant de la génération aléatoire.

Que pensez-vous de la pertinence de vos méthodes de génération, que peuvent-elles garantir sur les séquences d'entrées qu'elles génèrent ?

Epilogue - Et après ... ?

Le fond du problème dans le domaine du test est qu'un test ne peut que prouver le mauvais fonctionnement d'un programme, pas son bon fonctionnement. Toutefois, la preuve formelle d'un programme est, dans le cas général, impossible, le test reste donc la seule alternative. Pour qu'ils soit satisfaisant, un ensemble de tests doit donc couvrir suffisamment de cas pour nous convaincre que la probabilité qu'un bug n'ait pas été détecté est faible. Dans les programmes qui vous sont donnés aujourd'hui, 27 sont corrects et 16 bugs différents se retrouvent dans les programmes incorrects. Les avez vous tous trouvés ?