Ce travail est à faire en binôme. Le compte rendu (un par binôme sur moodle, après inscription des deux membres dans le binôme) est à déposer sur la page Moodle du cours.

Cette apnée est la suite de l'apnée Programmation «Tests de programmes». Il est conseillé de relire le prologue et la section 1 de cette apnée (et de l'avoir terminée bien entendu).

On s'intéresse au problème de la génération automatique de tests. On souhaite donc réaliser un (ou plusieurs) programme(s) permettant de :

1. générer aléatoirement un texte correctement parenthésé (avec parenthèses multiples)
2. générer aléatoirement un texte mal parenthésé

Lors de l'apnée Programmation, vous avez dû remarquer que ce problème n'est pas si simple. Une chaîne générée de manière entièrement aléatoire est en général peu intéressante du point de vue du problème considéré : elle contient peu de parenthèses, ou peu de profondeur d'imbrication.

Nous proposons dans cette apnée de partir de l'algorithme de vérification du parenthésage, afin d'obtenir dans un premier temps un algorithme de génération de textes correctement parenthésés. Cet algorithme pourra ensuite être modifié et utilisé pour la génération de textes mal parenthésés.

Exercice 1

On rappelle l'algorithme de vérification du parenthésage :

```
p = PileVide()
tant que il reste des caractères
   c = caractère suivant
   si c est une parenthèse ouvrante alors
      Empiler(p , c)
   si c est une parenthèse fermante alors
      si EstVide(p) ou Sommet(p) ne correspond pas à c alors
         retourner Erreur de parenthésage
      sinon
         Dépiler(p)
si EstVide(p) alors
   retourner Parenthésage correct
sinon
retourner Erreur de parenthésage
```

Cet algorithme peut être modifié afin non plus de vérifier le parenthésage, mais de générer une chaîne correcte. On reprend donc la structure de l'algorithme, en utilisant la pile pour sauvegarder l'état courant du parenthésage :

```
p = PileVide()

tant que non EstVide(p) ou il reste des caractères à imprimer
   choisir parmi :
   { c est une parenthèse ouvrante }
       c = parenthèse ouvrante
       Empiler(p , c)
   { c est une parenthèse fermante }
       si non EstVide(p) alors
          c = parenthèse fermante correspondant à Sommet(p)
          Dépiler(p)
   { c est un caractère quelconque }
       c = caractère non parenthèse
   Imprimer c
```

Cet algorithme n'est pas déterministe : il y a deux choix à faire à chaque pas de l'itération :

- imprime-t-on encore des caractères ou bien s'arrête-t-on ?
- si on continue, quel (type de) caractère imprime-t-on ?

La manière de faire ces choix n'est ici pas indiquée. On peut pour cela utiliser un générateur aléatoire (tout en s'assurant que la chaîne générée est correctement parenthésée).

1. Préciser les choix effectués dans l'algorithme ci-dessus afin de générer une chaîne aléatoire correctement parenthésée d'une **longueur minimum** donnée.
2. Préciser les choix effectués dans l'algorithme ci-dessus afin de générer une chaîne aléatoire correctement parenthésée d'un **niveau d'imbrication minimum** donné.

Programmez ces deux versions de l'algorithme. Utilisez les chaînes générées pour tester les programmes fournis lors de l'apnée Programmatio... ?
Avez-vous trouvé de nouveaux bugs ? Pensez éventuellement à varier les paramètres de votre générateur aléatoire.

Votre compte-rendu devra présenter explicitement :

- les algorithmes programmés, avec les différentes valeurs de paramètres utilisées
- les bugs détectés dans les différentes situations, et la stratégie que vous avez mise en œuvre pour en détecter le plus possible

Fournissez également les codes sources compilables de vos programmes implémentant ces algorithmes.

Exercice 2

Modifiez les algorithmes de l'exercice 1 afin qu'ils génèrent aléatoirement une chaîne mal parenthésée. Quels sont les différents cas à prévoir ?

Programmez l'algorithme ainsi modifié. Utilisez les chaînes générées pour tester les programmes fournis lors de l'apnée Programmation. Avez-vous trouvé de nouveaux bugs ?

À nouveau, votre compte-rendu devra présenter explicitement :

- les algorithmes utilisés, avec les différentes valeurs de paramètres utilisées
- les bugs détectés dans les différentes situations, et la stratégie que vous avez mise en œuvre pour en détecter le plus possible

Fournissez également les codes sources compilables de vos programmes implémentant ces algorithmes.

Exercice 3 (en bonus s'il reste du temps)

Dans certaines situations, il n'est pas possible de connaître à l'avance, pour un test donné, la réponse attendue par la spécification de l'algorithme. On peut alors utiliser un oracle, fonction permettant de décider si un test a échoué ou non.

Dans notre cas, l'oracle est une fonction prenant en paramètre la chaîne lue, et un booléen, indiquant la réponse du programme testé (vrai pour «Bon parenthésage», faux sinon). Cette fonction calcule alors quelle devrait être la réponse correcte, et renvoie un booléen indiquant si la réponse du programme est correcte ou non pour la chaîne considérée.

Complétez la fonction OracleParenthesage du programme oracle.c. Ce programme prend deux arguments : le nom du fichier contenant la chaîne à tester, et le nom du fichier contenant la réponse du programme testé.

Utilisez l'oracle pour tester les programmes fournis sur ces tests supplémentaires. Avez-vous trouvé de nouveaux bugs ?

Compte-rendu

Le compte-rendu doit être déposé au format PDF. Il devra comporter, **sur 4 pages maximum** :

- une introduction résumant le travail effectué ;
- les algorithmes obtenus aux exercices 1 et 2, la manière dont ils ont été utilisés (valeur des paramètres, nombre et propriétés des chaînes générées) ;
- le nombre de programmes incorrects découverts à chaque étape.

Outre ce PDF, vous devez déposer :

- tous les fichiers sources nécessaires pour compiler vos programmes implémentant les divers algorithmes mis au point (donc y compris les fichiers d'en-tête, les structures de données implémentées...) ;
- une archive compressée contenant les chaînes générées ;
- si vous avez abordé l'exercice 3, le fichier oracle.c complété.

Annexe : utilisation de la fonction C random

```
#include

long int random(void);

void srandom(unsigned int seed);
```

La fonction C random permet de générer une séquence de nombres pseudo-aléatoire. Chaque appel à cette fonction renvoie un entier compris entre 0 et RAND_MAX.

La fonction srandom doit être appelée une seule fois, au début du programme, avant la première utilisation de random. Elle prend en paramètre une graine (*seed*), permettant d'initialiser la séquence pseudo-aléatoire.

Last modified: Thursday, 22 August 2024, 4:41 PM

?