SINF 1121 – Algorithmique et structures de données

Mission 1

1 Description générale de la thématique

Plusieurs langages de programmation utilisent de façon explicite une pile d'opérandes. C'est en particulier le cas du langage `PostScript`, un des formats usuels de fichiers utilisés pour l'impression. Un fichier `PostScript` envoyé à une imprimante est en réalité un programme qui définit les opérations que doit effectuer l'imprimante pour produire le document imprimé. La plupart des opérations exécutées en langage `PostScript` consistent à dépiler les opérandes depuis la pile du langage et à empiler le résultat de l'opération. Le langage `forth` communément utilisé par des astronomes pour programmer des télescopes est également basé sur une pile. Les calculateurs de poche `Hewlett-Packard` utilisent aussi une pile d'opérandes.

Dans le cadre de cette mission, vous serez amenés à concevoir et mettre en oeuvre un mini-calculateur basé sur le langage `PostScript`. Ce sera l'occasion d'approfondir votre connaissance du TAD pile et de ses possibilités d'implémentation.

2 A faire au plus vite

Les conditions à remplir pour pouvoir aborder cette mission sont :

- se procurer **au plus vite** l'ouvrage de référence du cours : *Algorithms*, Sedgewick, Wayne 4ème édition (*Algorithms-4*), disponible notamment à la librairie Agora-OIL. Ce livre est **la** ressource **indispensable** pour le bon suivi de ce cours. Des références explicites à des chapitres, sections ou pages sont présentes dans l'énoncé de chacune des missions. Les notions présentées dans le premier chapitre constituent par ailleurs des rappels utiles.
- **s'inscrire au cours** sur Moodle
  `http://moodleucl.uclouvain.be/course/view.php?id=7682`
- **consulter l'agenda du cours** sur Moodle.
- **accéder aux informations spécifiques** pour cette mission sur Moodle.
- s'inscrire au cours sur inginious
  `https://inginious.info.ucl.ac.be/course/LSINF1121-2015`

3 Prérequis

Les prérequis à satisfaire pour aborder ce cours sont :

- avoir une maîtrise de la programmation en Java (ou, à tout le moins, dans un langage orienté objet tel que C++)
- connaître les fonctionnalités de base d'un environnement de développement en Java tel que **Eclipse, IntelliJ**,
- se débrouiller en anglais technique et scientifique (lecture).

Si certains d'entre vous ont des difficultés avec un ou plusieurs de ces prérequis, il est indispensable d'en discuter en groupe et avec votre tuteur avant de commencer à travailler sur cette mission.

4 Objectifs poursuivis

A l'issue de cette mission chaque étudiant sera capable de :

— décrire avec précision les propriétés des types abstraits ***pile*** et ***file***,
— faire la distinction entre un ***type abstrait de données*** et son implémentation,
— mettre en oeuvre et évaluer une implémentation d'une pile par une ***liste simplement ou doublement chaînée***,
— ***concevoir*** et mettre en oeuvre un ***mini-interpréteur*** utilisant une pile,
— utiliser des ***tests unitaires*** (`JUnit`) pour tester et prouver le bon fonctionnement d'un programme.

5 Ressources

— Livre : *Algorithms*, Sedgewick, Wayne 4ème édition (*Algorithms-4*), disponible notamment à la librairie Agora-OIL : sections **1.2, 1.3**, chapitre **1**.
Sur le site iCampus, suivre **`complexité calculatoire`** `dans la section` **`Prérequis`** `sur la page d'accueil.`
— Document : *Spécifications en Java, Préconditions et postconditions : pourquoi ? comment ?*, P. Dupont.
Sur le site iCampus, suivre `Resources/Divers/specif.pdf`

6 Calendrier

— lundi 21 septembre : démarrage de la mission
— vendredi 25 septembre, avant 18h00 : envoi réponses questions Inginious + tâche individuelle Inginious (excepté test interpréteur).
— lundi 28 septembre : séance intermédiaire
— vendredi 02 octobre, avant 18h00 : remise des produits sur Inginious
— lundi 05 octobre : séance de bilan

7 Marche à suivre

1. Conformément à l'approche pédagogique proposée, vous répartirez durant la séance de mise en route les questions entre les différents membres du groupe de telle sorte que chaque étudiant soit responsable d'au moins une question. Chaque étudiant préparera la réponse à la (aux) question(s) dont il est responsable. Ceci n'empêche pas de s'informer des réponses apportées aux autres questions d'autant plus que certaines questions sont liées entre elles. En outre, ***chaque étudiant*** se préparera à débattre avec les autres membres du groupe de ***toutes les questions***.
2. Une bonne organisation du travail en groupe suppose une responsabilité partagée entre tous les membres du groupe. Une participation active aux séances tutorées et une collaboration effective entre les membres du groupe pour les documents et programmes à remettre sont donc requises. Une ***note de participation individuelle*** sera prise en compte pour ***20 %*** de la note globale pour ce cours. Une participation active passe par l'implémentation des missions sur Inginious dont certaines parties sont individuelles.

8 Questions

1. Définissez ce qu'est un type abstrait de données [1] (TAD). En java, est-il préférable de décrire un TAD par une classe ou une interface ? Pourquoi ?
2. Comment faire pour implémenter une ***pile*** par une liste simplement chaînée où les opérations `push` et `pop` se font en *fin de liste* ? Cette solution est-elle efficace ? Argumentez.
3. En consultant la documentation sur l'API de Java, décrivez l'implémentation d'une pile par la classe `java.util.Stack`. Aller également voir le code source de l'implémentation `java.util.Stack` (crtl+click depuis votre Eclipse ou IntelliJ).
4. Comment faire pour implémenter le type abstrait de données `Pile` à l'aide de deux files ? Décrivez en particulier le fonctionnement des méthodes *`push`* et *`pop`* dans ce cas. A titre d'exemple, précisez l'état de chacune des deux files après avoir empilé les entiers `1 2 3` à partir d'une pile initialement vide. Décrivez ce qu'il se passe ensuite lorsque l'on effectue l'opération *`pop`*. Quelle est la complexité temporelle de ces méthodes si l'on suppose que chaque opération *`enqueue`* et *`dequeue`* s'exécute en temps constant. Cette implémentation d'une pile est-elle efficace (pour n opérations) par rapport aux autres implémentations présentées dans *Algorithms-4* ?
5. Comment faire en Java pour lire des données textuelles depuis un fichier et pour écrire des résultats dans un fichier ASCII ? Écrivez en Java une méthode ***générique***, c'est-à-dire aussi indépendante que possible de son utilisation dans un contexte particulier, de lecture depuis un fichier texte. Faites de même pour l'écriture dans un fichier ASCII.

1. *abstract data type*, en anglais.

6. Comment faire en Java pour passer des arguments à un programme ? Soyez précis. Donnez un exemple
7. Que signifie les paramètres -Xmx, -Xms que l'on peut passer à la JVM pour l'exécution d'un bytecode ? Est-ce que ces paramètres peuvent influencer la vitesse d'exécution d'un programme Java ? Pourquoi ?
8. Qu'est-ce qu'un iterateur en Java (`java.util.Iterator`). Pourquoi est-ce utile de définir une méthode `iterator()` sur les structures de données ? Que pensez vous de permettre la modification d'une structure de donnée alors qu'on est en train d'itérer sur celle-ci ? Pour vous aidez dans la réflextion, nous vous invitons à lire la spécification de l'API Java concernant la méthode `remove()`. Proposez une modification du code de l'iterateur de Stack qui lance une `java.util.ConcurrentModificationException` si le client modifie la collection avec un `push()` ou `pop()` durant l'itération. Est-ce une bonne idée de laisser l'implémentation de la méthode `remove()` vide si on ne désire pas permettre cette fonctionnalité ?
9. La notation (tilde) est utilisée dans *Algorithms-4*pour l'analyse des temps de calcul des algorithmes. En quoi cette notation diffère ou ressemble aux notations plus classiquement utilisées $\mathcal{O}$ (big Oh), Ω (big Omega) et Θ (big Theta). Expliquez précisément les liens et similitudes entre celles-ci. Que voyez-vous comme avantage à utiliser la notation (tilde) plutôt que $\mathcal{O}$ lorsque c'est possible ?
10. Expliquez comment nous pouvons extraire la caractérisation (tilde) de l'implémentation d'un algorithme à l'aide du test *Doubling ratio*. Comment fonctionne ce test, quelles sont les limites et avantages de ce test ? Supposant que nous mesurons les temps d'exécutions T(N) suivants (en secondes) d'un programme en fonction de la taille de l'entrée N. Comment pouvez-vous caractériser au mieux l'ordre de croissance ? Que serait le temps d'exécution pour 128000.

N	1000	2000	4000	8000	16000	32000	64000
T(N)	0	0	0.1	0.3	1.3	5.1	20.5

11. Qu'est-ce qu'un bon ensemble de tests unitaires pour vérifier l'exactitude d'une structure de données ? En quoi la génération de données aléatoire peut être utile pour tester les structures de données ? Pourquoi est-ce important de travailler avec une semence (seed) fixée ? En quoi un outil d'analyse de couverture de code peut être utile (tel que `http://eclemma.org/jacoco/`) pour vous aidez à concevoir des tests. Comment vérifier expérimentalement que l'implémentation d'une structure de données ou un algorithme a bien la complexité temporelle théorique attendue ?

Les réponses aux questions doivent être soumises sur INGInious ***avant*** la séance ***intermédiaire***[a]. Cela suppose une étude individuelle et une mise en commun en groupe (sans tuteur) préalablement à cette séance. Un document (au format PDF) reprenant les réponses aux questions devra être soumis sur INGInious ***au plus tard*** pour le vendredi 25 septembre à ***18h00***. Les réponses seront discutées en groupe avec le tuteur durant la séance intermédiaire. Ces réponses ne doivent pas explicitement faire partie des produits remis en fin de mission. Néanmoins, si certains éléments de réponse sont essentiels à la justification des choix d'implémentation et à l'analyse des résultats du programme, ils seront brièvement rappelés dans le *rapport de programme*.

a. à l'exception, le cas échéant, de question(s) spécifiquement liée(s) au problème traité.

9 Problème

Dans le langage `PostScript`, le programme suivant calcule $1 + 1$:

```
1 1 add pstack
```

Chaque élément est un *token* numérique, symbolique ou booléen.
Dans l'exemple ci-dessus, deux tokens numériques **`1 1`** sont suivis de deux tokens symboliques **`add pstack`**. L'évaluation de ce programme se passe comme suit. Chaque token numérique est empilé sur la pile du langage. Le token **`add`** dépile deux opérandes de la pile, calcule leur somme et empile le résultat. Le token **`pstack`** provoque l'impression du contenu de *toute* la pile.

Les autres opérateurs arithmétiques de base sont représentés respectivement par les tokens **`sub, mul`** et **`div`**. Le token **`pop`** correspond précisément à la méthode pop du TAD pile. Le programme suivant calcule donc la valeur 2 et imprime une chaîne vide (la pile étant vide à ce stade) :

```
1 1 add pop pstack
```

Les deux programmes suivant calculent et impriment respectivement $1 + (3 * 4)$ et $(1 + 3) * 4$, sans que des parenthèses soient nécessaires (pouvez-vous dire pourquoi ?) :

```
1 3 4 mul add pstack
1 3 add 4 mul pstack
```

L'opérateur **`dup`** permet de dupliquer une valeur. Le programme suivant calcule et imprime donc 5.5^2, en laissant une pile vide :

```
5.5 dup mul pstack pop
```

L'opérateur **`exch`** permet d'échanger deux valeurs. Le programme suivant calcule et imprime le résultat de l'opération $3 - 1$:

```
1 3 exch sub pstack
```

Les programmes suivants manipulent également des symboles logiques. Leur résultat est l'impression d'une valeur booléenne, respectivement **`false`** et **`true`**, en laissant une pile vide :

```
1 2 eq pstack pop
1 2 ne pstack pop
```

Des symboles peuvent être définis par l'instruction **`def`**. Le programme suivant calcule et imprime l'aire d'un cercle de rayon 1.6 :

```
/pi 3.141592653 def
/radius 1.6 def
pi radius dup mul mul pstack pop
```

9.1 Analyse, conception et production d'une solution

Vous êtes en charge de concevoir et d'implémenter en Java un interpréteur du mini langage `PostScript` décrit ci-dessus. Concrètement, vous devrez créer une classe `Interpreter` avec une méthode `interpret` qui prendra en entrée un `String` contenant les instructions (sur une seule ligne) et qui retournera un nouveau `String` contenant l'état de la stack lors de l'instruction `pstack`. S'il n'y a pas d'instruction `pstack` ou si la stack est vide lors de `pstack`, le `String` vide doit être retourné.
Par exemple, lors de l'appel suivant : `myInterpreter.interpret("3 3 div pstack")`, votre méthode doit retourner `1`. Si ensuite on effectue un second appel avec `"1 sub 0 eq pstack pop"`, le résultat attendu est `true`.
Vous devrez notamment implémenter les commandes `PostScript` suivantes :
`pstack, add, sub, mul, div, dup, exch, eq, ne, def, pop`

Ce problème sera divisé en trois étapes :
— Création de tests unitaires avec JUnit pour tester le bon fonctionnement de votre stack et de votre interpréteur (tâche individuelle, à faire **avant** la phase d'implémentation) ;
— Implémentation d'une stack générique (tâche individuelle) ;
— Implémentation de l'interpréteur de PostScript (par groupes).
Pour plus de détails, voir l'énoncé des différentes tâches sur INGInious.

Un fichier ***data.mps*** comportant des instructions en `mini-PostScript` est disponible sur le site iCampus, suivre `Documents et Liens/missions/m1/`.
Vous testerez notamment (mais pas uniquement !) votre programme sur les différentes lignes de ce fichier.
Il vous est demandé d'implémenter la pile de votre interpréteur `mini-PostScript` par une ***liste simplement chaînée***.

Il serait particulièrement utile de vous poser un ensemble de questions ***avant*** d'aborder le travail de conception et de programmation dans ce problème. Parmi celles-ci, on peut citer :

- Quelles sont les fonctionnalités du programme demandé ? Comment répartir ces fonctionnalités dans différentes parties de votre programme et comment répartir le travail de programmation entre vous ?
- Quelles sont les fonctionnalités du programme demandé qui ne dépendent pas spécifiquement du problème traité ? Comment isoler ces fonctionnalités de manière à ce qu'elles soient réutilisables dans un autre programme résolvant un autre problème ?
- Que faudrait-il faire si l'on désirait changer d'implémentation de pile dans votre interpréteur ? Comment garantir que ce changement nécessite le moins de modifications possibles dans votre programme ?
- Est-il possible que certaines instructions écrites en `mini-Postscript` soient non valides ? Si oui, comment proposez-vous de gérer ces cas ? Sinon, en quoi votre programme utilise-t-il cette propriété ?

10 Indications méthodologiques

- Veillez à répondre aux questions de manière ***précise*** et ***concise***. Citez vos sources.
- Ne vous précipitez pas sur la résolution du problème mais veillez à répondre aux questions avant les phases d'analyse et de conception.
- Écrivez les tests unitaires avant de commencer à programmer.
- Demandez-vous comment un **diagramme de classes** peut aider à répartir efficacement la tâche de programmation entre vous.
- Demandez-vous comment la réponse aux questions et votre analyse initiale du problème peut influencer le travail de programmation.
- Demandez-vous comment répartir le travail de programmation pour que tous les membres du groupe aient l'occasion de perfectionner leur maîtrise de Java pendant ce quadrimestre.

11 Remise des produits

Les produits de la mission (code, tests et rapport) sont à soumettre sur INGInious (https://inginious.info.ucl.ac.be/course/LSINF1121-2015) pour le vendredi 02 octobre, à ***18h00*** **dernier délai**. Passé ce délai, il sera impossible d'effectuer une nouvelle soumission.