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 : *Data Structures and Algorithms in Java*, Goodrich and Tamassia, 5ème édition [a] (*DSAJ-5*), 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. En outre, c'est le seul document autorisé à l'examen et les photocopies ne sont pas autorisées [b].
- **s'inscrire au cours** sur iCampus `icampus.uclouvain.be/claroline/course/index.php?cid=SINF1121`
- **consulter l'agenda du cours** sur iCampus.
- **accéder aux informations spécifiques** pour cette mission sur iCampus.

a. Les étudiants peuvent utiliser la nouvelle édition (6ème) mais ils doivent être conscients que les références explicites (numéros de pages, sections, chapitres, propositions, ...) peuvent différer entre les éditions. Il est fortement ***déconseillé*** d'utiliser une édition ***antérieure*** à la 5ème.

b. Les photocopies sont illégales dans la mesure où nous couvrirons plus de la moitié de cet ouvrage. Les notions présentées dans les premiers chapitres constituent par ailleurs des rappels utiles.

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**,
- 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,
– mettre en oeuvre des opérations d'***entrées/sorties par fichiers***.

5 Ressources

– Livre : *Data Structures and Algorithms in Java*, Goodrich and Tamassia, 5ème édition (*DSAJ-5*) : sections **3.2, 3.3**, chapitre **5**. [1]
– Le chapitre 4 de *DSAJ-5* constitue aussi un rappel important sur les notions de complexités temporelle et spatiale. En complément, une présentation sur ces notions est également disponible. [2]
Sur le site iCampus, suivre **`complexité calculatoire`** `dans la section` **`Prérequis`** `sur la page d'acceuil`.
– Document : *Spécifications en Java, Préconditions et postconditions : pourquoi ? comment ?*, P. Dupont.
Sur le site iCampus, suivre `Documents et liens/pdf/specif.pdf`

1. sections **3.2, 3.4** et chapitre **6** dans *DSAJ-6*
2. chapitre **4** dans *DSAJ-6*

6 Calendrier

- lundi 22 septembre : démarrage de la mission
- vendredi 26 septembre, avant ***18h00*** : ***envoi réponses questions***
- lundi 29 septembre : séance intermédiaire
- vendredi 03 octobre, avant ***18h00*** : ***remise des produits***
- lundi 06 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.

8 Questions

1. Définissez ce qu'est un type abstrait de données [3] (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`. Cette classe peut-elle convenir comme implémentation de l'interface `Stack` [4] décrite dans *DSAJ-5* ? Pourquoi ?
4. Proposez une implémentation de la classe `DNodeStack`. Il s'agit d'une classe similaire à `NodeStack` (décrite dans *DSAJ-5*) qui propose une implémentation *générique* d'une pile. Votre classe `DNodeStack` doit utiliser une implémentation en liste **doublement** chaînée **générique**. Elle reposera donc sur une classe `DNode<E>` (similaire à `Node<E>`, décrite dans *DSAJ-5*) que vous devez définir. Ajoutez, dans la classe `DNodeStack`, une méthode `public String toString()` qui renvoie une chaîne de caractères représentant le contenu de la pile. Commentez votre code.

3. *abstract data type*, en anglais.
4. Les fragments de code présentés dans *DSAJ-5* sont disponibles à partir du descriptif du cours ; sur le site iCampus, suivre `Description du cours`.

5. Complétez l'interface `Queue` (décrite dans *DSAJ-5*) contenant une interface pour le type abstrait ***file*** en ajoutant des préconditions et postconditions pour chacune des méthodes. Votre spécification correspond-elle à une programmation défensive ?
6. 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 par rapport aux autres implémentations présentées dans *DSAJ-5* ?
7. 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.

Les réponses aux questions doivent être soumises sur iCampus ***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 ASCII ou PDF) reprenant les réponses aux questions devra être soumis sur iCampus ***au plus tard*** pour le vendredi 26 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 $1 - 3$:

```
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 votre interpréteur prendra en entrée un fichier en `mini-PostScript` et produira en sortie un nouveau fichier résultant de l'évaluation des instructions lues. En particulier, les impressions du contenu de la pile se feront vers le fichier de sortie. Par convention, un passage à la ligne sera effectué après chaque exécution de l'instruction **pstack**.
Par exemple, à la lecture du fichier contenant les lignes suivantes :

```
3 3 div pstack
1 sub 0 eq pstack pop
```

votre programme produira un fichier contenant

```
1
true
```

Vous devrez notamment implémenter les commandes `PostScript` suivantes :
pstack, add, sub, mul, div, dup, exch, eq, ne, def, pop

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 votre programme sur ce fichier. Libre à vous de définir également de nouveaux fichiers de test et d'utiliser toutes techniques que vous jugeriez pertinentes pour vous assurer de la correction de votre code (par exemple, par le biais d'assertions).
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.
- 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 (***sources***[a] des programmes Java et leur documentation) sont à soumettre sur iCampus pour le vendredi 03 octobre, à ***18h00*** **dernier délai**. Le tuteur est déchargé de toute correction pour tout produit non remis dans ce délai. Le rapport de programme[b] contenant un commentaire (1 ou 2 page(s) max.) sur la solution apportée au problème doit être disponible au format `PDF`[c]. Les produits de l'ensemble du groupe doivent être soumis sur iCampus en ***un seul*** fichier archive (au format `zip` ou `tar.gz`).

a. ***PAS de `.class`***
b. incluant les réponses aux questions spécifiquement liées au problème traité.
c. en ***un seul*** document.