Exercices sur les structures de données

Programmation 3

Exercices

Pour cette série d'exercices, utilisez vos propres structures de données.

1. Écrivez un programme qui fusionne deux listes chaînées d'entiers ordonnés en une liste chaînée d'entiers ordonnés. Une méthode fusionner() dans une classe FusionListesChainees doit recevoir en paramètres les deux listes à fusionner et doit retourner la liste fusionnée.
2. Écrivez un programme qui lit une ligne de texte et qui utilise une pile pour afficher le texte inversé.
3. Écrivez un programme qui utilise une pile pour déterminer si une chaîne de caractères est un palindrome. Vous devez ignorer la casse, les espaces et la ponctuation.

Exercices

Les piles sont utilisées par les compilateurs pour évaluer des expressions et générer le code machine. Dans cet exercice voyons comment les compilateurs évaluent des expressions arithmétiques simples.

Nous écrivons généralement les opérations arithmétiques comme ceci : 3 + 4 ou 7 / 9. Ceci s'appelle la notation infixée. Opérande, opérateur, opérande. Les processeurs préfèrent une notation postfixée, soit 3 4 + ou 7 9 /. Opérande, opérande, opérateur (voir https://fr.wikipedia.org/wiki/Notation_polonaise_inverse).

Pour évaluer une expression infixée, un compilateur

1. convertit l'expression en notation postfixée
2. évalue ensuite l'expression postfixée.

Chaque algorithme utilise une pile pour parvenir à ses fins.

Exercice 4

Implémentez un algorithme de conversion infixe -> postfixe (l'algorithme est présenté à la page suivante). Écrivez une classe ConversionInfixePostfixe qui permet de convertir une expression *infixe* valide en expression *postfixe* à l'aide de votre pile. Par exemple (6 + 2) * 5 - 8 / 4 devient 6 2 + 5 * 8 4 / -

Les opérations +, -, *, /, ^ (exponentiel) et % sont permises.

Vous pouvez créer les méthodes suivantes :

- convertirInfixePostfixe()
- estOperateur()
- estPrioritaire()

Vous n'avez pas à traiter le cas où un opérande contient plus d'un chiffre (ex : 20) ni le cas où l'opérande est une lettre (calculs algébriques).

Quelques exemples de conversion infixe -> postfixe :

(1 + 2) * 3 -> 1 2 + 3 *	1 + 2 * 3 + 4 -> 1 2 3 * + 4 +	(4 + 8) * (6 - 5) / ((3 - 2) * (2 + 2)) -> 4 8 + 6 5 - * 3 2 – 2 2 + * /
1 + 2 * 3 -> 1 2 3 * +	(1 + 2) * (3 + 4) -> 1 2 + 3 4 + *	(3 + 2) * (4 - 1) / (8 + 7) -> 3 2 + 4 1 - * 8 7 + /

Exercice 4

Voici l'algorithme. Vous aurez besoin d'une pile temporaire dans laquelle les opérateurs et les parenthèses seront empilés. Quelques exemples détaillés de la démarche sont donnés ici : https://cs.nyu.edu/courses/fall09/V22.0102-002/lectures/InfixToPostfixExamples.pdf

Empiler une "(".
Ajouter une ")" à la fin de infixe
Tant que la pile n'est pas vide, lire infixe de gauche à droite et
- Si le caractère courant dans infixe est un chiffre (ou une lettre si on veut prévoir les calculs algébriques), l'ajouter à postfixe.
- Si le caractère courant est une "(", l'empiler
- Si le caractère courant est un opérateur
  - Dépiler les opérateur sur le dessus de la pile (s'il y en a) tant qu'ils ont une plus grande priorité que l'opérateur courant, et ajouter les opérateurs dépilés à postfixe.
  - Empiler le caractère courant.
- Si le caractère courant est une ")"
  - Dépiler les opérateurs du dessus de la pile et les ajouter à postfixe jusqu'à ce qu'une "(" soit sur le dessus de la pile.
  - Dépiler la "("

Exercice 5

Écrivez une classe EvaluateurPostfixe qui évalue une expression postfixée valide à l'aide de votre pile. Voici l'algorithme :

Ajouter une ")" à la fin de postfixe. Quand la ")" est atteinte, c'est la fin de l'algorithme.
Quand la ")" n'est pas atteinte, lire l'expression de gauche à droite.

    Si le caractère courant est un chiffre, empiler sa valeur.
    Sinon, si le caractère courant est un opérateur
        Dépiler les deux premiers éléments de la pile dans des variables x et y.
        Calculer x opérateur y
        Empiler le résultat sur la pile

Quand la ")" est atteinte dans l'expression, dépiler la première valeur de la pile. C'est le résultat de l'expression postfixée.

Les méthodes evaluerExpressionPostfixe() et calculer() peuvent être programmées.

Exercice 6

Vous avez vu comment une façon de "traverser" un arbre binaire. Voici un algorithme qui traverse un arbre binaire niveau par niveau, de gauche à droite. Implémentez-le à l'aide de votre arbre binaire et de votre file.

Insérer la racine dans la file
// Afficher la valeur de la racine
Tant qu'il y a des noeuds dans la file
    // Obtenir le prochain noeud de la file
    // Afficher la valeur du noeud
    Si le noeud a un enfant gauche
        Insérer l'enfant dans la file
        Afficher la valeur du Noeud
    Si le noeud a un enfant droit
        Insérer l'enfant dans la file

        Afficher la valeur du Noeud