PARIS DESCARTES IUT

DUT Informatique - 1ère année
SDA - Structures de Données et Algorithmes

TRAVAUX DIRIGES – Semaine n°4

Thèmes

– Résolution de problème et type abstrait de données (Pile)
– Algorithmique de pile

Spécifications

Les algorithmes seront à implémenter en C++ en utilisant le composant de pile du cours n° 4 ou à exprimer en pseudo-langage.

Primitives (fonctions) permettant la manipulation de variable de type `Pile` en pseudo-langage :

Primitives	*Rôle*	*Précondition*
`Pile ← empiler(Pile, Item)`	Empilement d'un élément au sommet de la pile	
`Pile ← dépiler(Pile)`	Dépilement de l'élément situé au sommet de la pile	La pile n'est pas vide
`Item ← sommet(Pile)`	Accès à l'élément situé au sommet de la pile	La pile n'est pas vide
`Booléen ← estVide(Pile)`	Indicateur de pile vide	

De plus, la constante `PileVide` permet d'obtenir une pile vide.

Extrait des primitives du type Pile du Cours n°4 :

```
/** @brief Initialiser une pile vide p de capacité c (>0) en mémoire dynamique */
void initialiser(Pile& p, unsigned int c);
/** @brief Désallouer une pile p en mémoire dynamique */
void detruire(Pile& p);
/** @brief retourne l'item au sommet d'une pile p */
Item sommet(const Pile& p);
/** @brief Empiler un item it sur une pile p */
void empiler(Pile& p, const Item& it);
/** @brief Dépiler l'item au sommet d'une pile p */
void depiler(Pile& p);
/** @brief Indicateur de pile vide (true si vide, false sinon)  */
bool estVide(const Pile& p);
```


Exercice 1. Inversion du contenu d'un tableau

L'algorithme demandé utilisera une pile pour inverser un tableau d'éléments de type quelconque. Après inversion, le tableau d'entier { 1, 2, 3 } est transformé en { 3, 2, 1 }.

1.1. Prototypez la fonction `inverser` dont le rôle est d'inverser le contenu d'un tableau de type `Item` suivant l'algorithme trouvé. Le tableau originel ne sera pas conservé lors de cette inversion.

1.2. Codez la fonction `inverser`.

1.3. Testez l'inversion d'un tableau d'entier initialisé de façon explicite par { 1, 2, 3, 4, 5 }.

Exercice 2. Evaluation d'une expression arithmétique postfixée

L'algorithme demandé permettra d'évaluer une expression arithmétique en notation postfixée. Dans cette notation, l'opérateur est placé après ses opérandes.

Par exemple : 7 * 2 s'écrit 7 2 * (3+7) * 2 s'écrit 3 7 + 2 * - 3 s'écrit 3 -

L'expression est évaluée à partir d'un balayage de gauche à droite de l'expression. En présence d'un opérateur, l'opération est évaluée avec les deux précédents opérandes (dans l'ordre, l'opérande gauche et l'opérande droit).

L'expression 4 2 - 3 * a pour valeur 6 = (4 - 2) * 3
alors que l'expression 4 2 3 - * a pour valeur -4 = 4 * (2 - 3).

Nous ferons l'hypothèse que l'expression est syntaxiquement correcte et que nous disposons de primitives permettant de parcourir les différents composants d'une expression :

Primitives	*Rôle*	*Précondition*
`début(Expression)`	Initialisation du parcours de l'expression	
`Token ← suivant(Expression)`	Accès au composant suivant dans le parcours	Le parcours n'est pas fini
`Booléen ← fin(Expression)`	Indicateur de fin de parcours	

Une variable `t` de type `Token` pourra contenir soit un entier, soit l'un des quatre opérateurs (+, -, ×, ÷). Si `t` contient un entier, la variable pourra être employée partout où un entier est attendu et il est possible de tester si `t` contient l'opérateur + par l'expression logique (`t = '+'`).

2.1. Donnez l'algorithme de l'évaluation d'une expression arithmétique postfixée en pseudo-langage. L'algorithme utilisera une pile.

2.2. Prototypez et codez la fonction `evaluer` dont le rôle est d'évaluer une expression arithmétique postfixée suivant l'algorithme trouvé.

2.3. Donnez un programme de test de l'évaluation d'une expression arithmétique postfixée.