Université du Québec **vendredi 19 octobre 2018.**
à Chicoutimi - DIM

Devoir 2
Structure de données
(8INF259)

Ce devoir doit être remis au plus tard
le vendredi 16 novembre avant 16h.

Instructions

- Pour faciliter la correction de vos programmes, il est recommandé de bien les commenter.
- Dans votre compte sur le sunens, créez un répertoire dont le nom est devoir2.
- Travaillez en équipe au plus de deux étudiant(e)s, remettez une seule copie par équipe. Il est strictement interdit pour une équipe de copier le travail d'une autre équipe.
- Envoyez votre travail à votre chargée de TD/TP : asma.razgallah1@uqac.ca

Objectifs du travail demandé

Étant donnée une expression arithmétique, le but de ce devoir est de la transformer en une expression postfixe et ensuite l'évaluer. L'expression arithmétique de départ est une suite de caractères contenant des valeurs numériques et les opérateurs **(** , **)** , ***** , /, **%**, **+**, et **-** . Tout autre caractère est considéré comme invalide pour cette expression. Les priorités des opérateurs, listés ci-dessus, sont dans l'ordre décroissant comme suit. Notez que l'opérateur **/** réfère à une division entière.

Priorité haute : (et)
Priorité moyenne : * / et %
Priorité basse : + et -

Bien entendu, à l'entrée, ces éléments sont lus comme une chaîne de caractères. Il vous revient à vous de les transformer aux types voulus. Pour simplifier les choses, on peut supposer les valeurs numériques composant cette chaine de caractères sont sur un seul digit.

Écrire un programme C++ implantant les fonctions de passage de l'écriture infixe à postfixe et d'évaluation d'une expression postfixe, en utilisant les STL Stack et Vector de C++. Pour ce faire, on utilise une classe POSTFIXE définie comme suit :

```
Template <element class>

Class POSTFIX {
 Private:
   Stack<element> Pile;
   Vector<element> Tableau;

 Public:
  bool postfix (vector<element> tableau );// lit l'expression à évaluer, à partir du clavier, dans
tableau et valide si l'expression ne contient que les caractères ci-dessus, à savoir les nombres entiers
composés de caractères nombres, et les opérateurs ci-dessus.

  bool valider_expression (vector<element> tableau); // teste si l'expression lue contient un nombre
valide de parenthèses
```

void transformerennombres (vector <element> tableau); // transforme les nombres lus en caractères en valeurs numériques

void transformerenpostfixe(stack<element> Pile, vector<element> tableau);// transforme l'expression lue en une expression postfixe.

int evaluer_expression(stack<element> Pile, vector<element> tableau); // affiche la valeur de l'expression lue.
}

Un bonus de 10% est accordé aux étudiants qui :

1. Prennent en compte le fait qu'un nombre peut-être sur plusieurs digits. (5%)

2. Utilisent une déclaration dynamique du tableau et de la pile. Dans ce cas, il est nécessaire de déclarer un destructeur. De plus, les itérateurs begin et end sont utilisés avec ->. Par exemple, si on utilise **vector<int> *tableau = new vector<int> (n)** pour déclarer d'une manière dynamique le vecteur tableau de n entiers, alors pour accéder au kème élément de tableau, on fait (*tableau)[k], au premier élément, on fait tableau->begin( ), etc. (5%)