Première année 2008-2009

Projet C

Calcul d'expressions arithmétiques

1 Présentation

Le but de ce projet est de réaliser en langage C, un programme calculant la valeur d'une expression arithmétique, en utilisant la notation polonaise suffixée.

2 Spécifications du projet

2.1 Fonctionnalités du programme

L'interface du programme proposera succesivement et indéfiniment :

1. la saisie d'une expression arithmétique, par exemple :

$$a/23 + b * c$$
$$(a + b) * (c + d)/a$$

2. la saisie des valeurs des identificateurs présents dans l'expression saisie, a, b, c et d, dans le cas de la deuxième expression.

À la suite de cette saisie, le programme affichera la valeur de l'expression calculée en fonction des valeurs des identificateurs.

Dans le cas où une erreur de saisie sera détectée lors de la lecture de l'expression, le programme invitera l'utilisateur à la resaisir.

De même lors de la saisie d'une valeur erronée pour un identificateur, le programme redemandera la saisie de cette valeur.

2.2 Spécifications détaillées

2.2.1 Syntaxe des expressions

Les expressions seront formées des **items** suivants :
- des constantes entières sur un ou plusieurs caractères ;
- des 4 opérateurs binaires + , −, ∗, / ;
- des identificateurs que l'on supposera composés d'un seul caractère ;
- des parenthèses.

Comme dans tous les langages C et Java, les opérateurs sont associatifs à gauche. Les priorités de + et - sont égales mais inférieures à celles de * et /, elles-même de priorités égales.

2.2.2 Représentation interne des expressions en notation polonaise

Dans la notation polonaise postfixée (dûe à Lukasiewicz), les opérateurs suivent immédiatement les opérandes. L'ordre des opérandes par rapport à la notation infixée est le même mais cette notation permet de s'affranchir des parenthèses.

Exemples :

Expression	Notation polonaise
a+b	ab+
a+b*c	abc*+
a+(b*c)	abc*+
(a+b)*c	ab+c*
a*(b+24)	ab24+*
a-b-c	ab-c-
a+b+c*d	abcd*++
(a*((b-a)+d))	aba-d+*
a+b*(c+d-(e+f/g))	abcd+efg/+-*+

2.2.3 Algorithme de conversion

Par la suite on supposera que les **items** de l'expression à convertir sont lus de gauche à droite. L'algorithme de conversion nécessite :

- une pile, vide au départ, pour stocker temporairement certains **items** ;
- une pile pour ranger l'expression en notation polonaise.

Pour préparer la deuxième phase d'évaluation, on stockera dans une liste les identificateurs de l'expression. Un même identificateur peut figurer plusieurs fois dans une expression.

Au fur et à mesure de la lecture des items, l'algorithme procède de la manière suivante :

- les opérandes (entiers ou identificateurs) sont insérés dans la pile dans leur ordre d'arrivée ;
- les parenthèses ouvrantes sont empilées ;
- une parenthèse fermante entraîne le dépilement de tous les éléments de la pile jusqu'à rencontre d'une parenthèse ouvrante. Les éléments dépilés sont insérés dans la pile, dans leur ordre de dépilement. La parenthèse ouvrante est retirée de la pile. Si aucune parenthèse ouvrante n'est rencontrée, c'est la preuve que l'expression est mal parenthésée.
- un opérateur provoque le dépilement, s'il y en a, de tous les opérateurs de priorité supérieure ou égale présents dans la pile jusqu'à une parenthèse ouvrante ou le fond de la pile. Ils sont insérés dans la pile, dans leur ordre de dépilement. Pour simplifier l'algorithme la parenthèse ouvrante peut-être considérée comme un opérateur de priorité plus faible que celle de tous les autres.
  Ensuite l'opérateur est empilé.
- Après le dernier item lu, la pile est entièrement dépilée, et les items dépilés sont insérés dans la pile, dans leur ordre de dépilement. Dans cette phase la pile ne doit pas contenir de parenthèse ouvrante, sinon c'est la preuve que l'expression était mal parenthésée.

Nous vous conseillons d'exécuter cet algorithme à la main sur des exemples avant de le coder.

2.2.4 Calcul de la valeur d'une expression polonaise suffixée

À l'issue de la première phase, on dispose d'une pile contenant l'expression en notation polonaise, et d'une liste des identificateurs. La liste va permettre de saisir au clavier les valeurs de ces identificateurs et de les y enregistrer. L'expression polonaise située dans la pile est dans l'ordre inverse de ce qui est nécessaire pour la suite. Il vous faudra donc la retourner, en l'empilant dans une nouvelle pile.

Le calcul de la valeur d'une expression polonaise suffixée, de type entier, nécessite une pile d'entiers pour stocker les résultats intermédiaires. L'algorithme est le suivant :

- On prélève les items de l'expression polonaise stockée préalablement dans la pile.
- On empile les entiers et les valeurs entières des identificateurs.
- Un opérateur binaire s'applique sur les deux derniers éléments empilés, que l'on dépile, et le résultat est empilé.
- À la fin de l'expression, la pile ne contient plus qu'une valeur, qui est le résultat.

Par exemple pour l'expression $ab24+*$ si a vaut 3 et si b vaut 2, les états successifs de la pile seront $[3]$, $[3,2]$, $[3,2,24]$, $[3,26]$, $[78]$.

2.2.5 Structure de l'application et structures de données utilisées

L'application est composée des modules suivants :
- Module de lecture des expressions (intégralement fourni).
- Un module unique pseudo-générique "structure linéaire" qui vous permettra de créer les piles et la liste nécessaires aux deux algorithmes (à développer). Ce module devra obligatoirement être implanté en utilisant des listes chainées et en suivant le fichier interface **linear_struct.h** fourni.
  Il est interdit de modifier ce fichier sous peine de ne pas pouvoir effectuer les tests lors de la séance d'évaluation.
- Application principale (à développer).

3 Travail demandé

Ce projet sera réalisé en monôme. À l'issue du projet, vous devrez rendre à votre enseignant
- un rapport (**pdf** généré à partir de LaTeX ou openoffice) résumant le travail effectué ;
- une archive `votrenom.tar` contenant l'ensemble des sources **commentées et indentées correctement** de votre projet.

Ce travail sera également testé par votre enseignant en votre présence sur les machine Linux des salles d'enseignement.

Il est impératif d'avoir vérifié que votre projet compile et fonctionne sur ces machines avant cette séance.

3.1 Fournitures

Les fichiers suivants vous sont fournis :
- le module `read_item`, composé des fichiers `read_item.h`, `read_item.o` et `type_item.h`. Il permet la lecture au clavier des items composant une expression.
- une partie du module `linear_struct`, l'interface `linear_struct.h` et la définition du type dans `linear_struct.c`.

3.2 Modules à réaliser

- Vous devez compléter le module `linear_struct`.
- Vous construirez ensuite l'application principale en utilisant les modules `read_item` et `linear_struct`.
- Vous écrirez un fichier `makefile` permettant de mettre à jour votre application.

3.3 Programmation

Vous devrez respecter intégralement les interfaces des modules fournis.

Les choix de codage n'interviendront pas dans la notation, dès lors que les solutions adoptées sont justifiées et commentées. De plus, ces solutions devront être motivées par une exigence de clarté, d'exhaustivité et de concision en priorité absolue par rapport à un souci quelconque d'optimisation.

Le texte source devra bien sûr être clair, lisible, correctement indenté, intelligemment annoté, en respectant les règles évoquées en TD et en TP.

Les fichiers C rendus devront impérativement pouvoir être compilés même si les fonctions demandées ne sont pas totalement développées.

Lorsque votre programme fonctionnera ou plutôt **semblera** fonctionner, il est important que vous le testiez à l'aide de jeux de tests significatifs, qui doivent montrer le bon fonctionnement de votre programme dans tous les cas que vous jugerez utile de distinguer. Ces tests devront bien sûr être joints au rapport.

3.4 Rapport et Tests

Le travail de conception et de programmation effectué devra figurer *in extenso* dans le rapport de projet à rendre. Vous devrez clairement détailler les découpages en fonctions et les algorithmes de votre programme, expliciter et justifier les choix de conception. Le principe est d'expliquer suffisamment clairement les algorithmes, de manière à pouvoir comprendre le programme sans avoir à lire le listing.

Votre programme sera soumis à une batterie de tests qui permettront de juger sa correction par rapport à l'objectif du projet. Vous devrez également proposer vous-même à cette occasion des jeux de tests destinés à prouver la correction de votre application. Enfin vous devrez être capables de répondre aux questions de l'enseignant de façon claire et synthétique.

3.5 Les Dates à retenir

Remise du source et du rapport	mardi 31 mars : 18h
Tests	mercredi 1 avril : 14h