GM3

TD11 - Piles

1. Analyse syntaxique

Par exemple, problème de la reconnaissance des expressions bien parenthèses. Ecrire un algorithme qui :

- accepte les mots comme (), ()(()) ou ((()())())
- rejette les mots comme)(, ()(() ou ((()))))

Laisser chercher.
Stocker les parenthèses ouvrantes non encore refermées dans une pile de caractères,
Le programme lit caractère par caractère le mot entré :

- si c'est une ouvrante, elle est empilée ;
- si c'est une fermante, l'ouvrante correspondante est dépilée.

Le mot est accepté si

- la pile n'est jamais vide à la lecture d'une fermante ; et si
- la pile est vide lorsque le mot a été lu.

```
Fonction parenthèse (c : chaine) : booléen
Var p : pile
    res : booléen
Début
res←vrai
p←créer-pile
i←1
TantQue i≤lg(c) et res Faire
    Si c[i]='('
         Alors empiler(p,c[i])
         Sinon Si c[i]=')'
                  Alors Si pile-vide(p) alors res←faux
                                        sinon dépiler(p)
                        FinSi
               FinSi
    FinSi
    i←i+1
FinTantQue
Si res et ¬pile-vide(p)
      alors res←faux
FinSi
Retourner(res)
Fin
```

2. Calcul d'une expression postfixée

La notation postfixée (polonaise) consiste à placer les opérandes devant l'opérateur. La notation infixée (parenthèsée) consiste à entourer les opérateurs par leurs opérandes. Les parenthèses sont nécessaires uniquement en notation infixée.
Les notations préfixée et postfixée sont d'un emploi plus facile puisqu'on sait immédiatement combien d'opérandes il faut rechercher.
(3 + 5) * 2 s'écrira en notation postfixée (notation polonaise) : 3 5 + 2 *
alors que 3 + (5 * 2) s'écrira: 3 5 2 * +
Pour 6 5 2 3 + 8 * + 3 + * = 6*(5 + ((2+3) * 8) + 3) = 6*48 = 288

```
Type  op-binaire = {+, -, *, /}
      op-unaire = {racine_2, sin, cos, exp}

effectuer1(op,x) ; effectuer2(op,x,y) ; C2R : chaine → réel ; mot-suiv(c,i,m)
```

```
Fonction calculer (c : chaine) : réel
Type  op-binaire = {+, -, *, /}
      op-unaire = {racine_2, sin, cos, exp}
Var   p : pile
      x, y : réel
      i : entier
      m : chaine
Début
p←créer-pile
i←1
TantQue i≤lg(c) Faire
    mot-suiv(c,i,m)
    Si m dans op-binaire
      Alors x←sommet(p)
            dépiler(p)
            y←sommet(p)
            dépiler(p)
            empiler(p,effectuer2(m,x,y))
      Sinon Si m dans op-unaire
                  Alors x←sommet(p)
                        dépiler(p)
                        empiler(p,effectuer1(m,x))
                  Sinon empiler(p,C2R(m))
            FinSi
    FinSi
FinTantQue
Si ¬pile-vide(p)
      Alors x ← sommet(p)
            dépiler(p)
      Sinon x ← 0
FinSi
Retourner(x)
Fin
```

3. Transformation d'une expression infixée en postfixée

Pour éviter le parenthésage, il est possible de transformer une expression infixée en une expression postfixée en faisant "glisser" les opérateurs arithmétiques à la suite des expressions auxquelles ils s'appliquent.
Précédence des opérateurs (pour cet algorithme) :
3 : opérateurs unaires
2 : / *
1 : + -
0 : (
L'algorithme passe les opérandes à la forme postfixe, mais sauvegarde les opérateurs dans la pile jusqu'à ce que tous les opérandes soient tous traduits.

Exemple : a+b*c+(d*e+f)*g = abc*+de*f+g*+

GM3

```
Fonction traduire (c : chaine) : chaine
Var p : pile
    post, m, x : chaine
    i : entier
Début
p←créer-pile
post←''
i←1
TantQue i≤lg(c) Faire
   mot-suiv(c,i,m)
   Si est-opérande(m)
      Alors post←concaténer(post,m)
      Sinon Si m ='('
              Alors empiler(p,m)
              Sinon Si m =')'
                      Alors x←sommet(p)
                            dépiler(p)
                            TantQue x≠'(' Faire
                              post←concaténer(post,x)
                              x←sommet(p)
                              dépiler(p)
                            FinTantQue
                      Sinon {opérateur}
                        TantQue ¬pile-vide(p) et prec(sommet(p))>=prec(m) Faire
                               x←sommet(p)
                               dépiler(p)
                               post←concaténer(post,x)
                        FinTantQue
                        empiler(p,m)
                  FinSi
            FinSi
   FinSi
FinTantQue
TantQue ¬pile-vide(p) Faire
    x←sommet(p)
    dépiler(p)
    post←concaténer(post,x)
FinTantQue
Retourner(post)
Fin
```