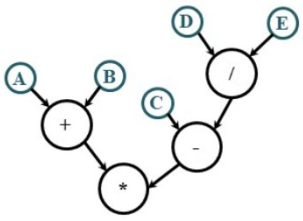
TP SE

18 ربوتك أ, 2025 11:39

Exemple d'expression

E=((A + B) * (C - (D / E)))

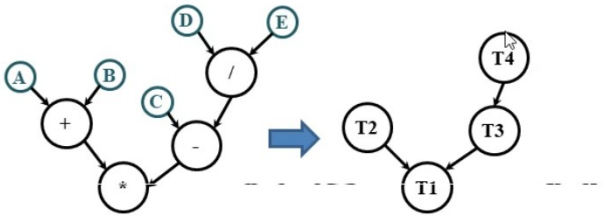


Arbre syntaxique

TP 1: Concurrence

Exemple d'expression

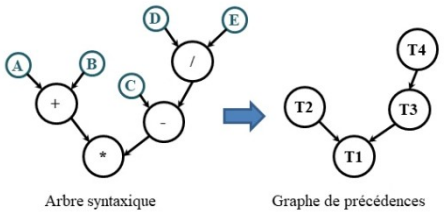
E=((A + B) * (C - (D / E)))



Arbre syntaxique Graphe de précédences

Exemple d'expression

E=((A + B) * (C - (D / E)))



Arbre syntaxique Graphe de précédences

On doit générer (afficher sur écran):

- La liste des noms des taches
- Le calcul fait par chaque tache
- Les précédences entre les taches

T1: M1:= M2*M3
T2: M2:= A+B
T3: M3:=C-M4
T4: M4:=D/E

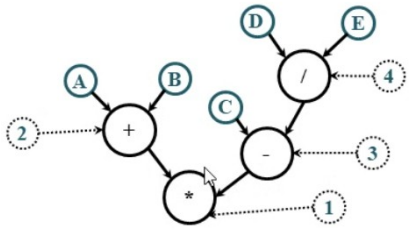
T2< T1
T3<T1
T4<T3

((A + B) * (C - (D / E)))

On s interesst que opertation binaire +−∗/□<operande gauche> <operateur><operande droit>

Multiplication c la dernier operation a realiser

On doit utiliser des variables temporaire
Chaque operation va corressepondre a une tache de la forme:□Ti:Mi=<OpG><OP><OpG>
Dans algo seq on numerotera les taches de maniere monotone croissnate
De la dernier operation realiser jusqu'a le premiere



E=((A + B) * (C - (D / E)))
2 1 3 4

T1: M1:= M2*M3
T2: M2:= A+B
T3: M3:=C-M4
T4: M4:=D/E

Generation mono-processus:
((A + B) * (C - (D / E)))
On emploi une fonction **genere() recursive** qui fonctionera sur une sous expression passe en parametre ,initiallemnt ,on utilise toute l'expression,eele travaillera sur l'operateur **centrale** de cette sous expression pour genere entre autres:
Num de tache
Contenue de la tache :Mi=<OpG><OP><OpG>
Precendence entre tache et la tache qui la precede ti<tj

Comment chercher l'operateur centrale:
Parcourire l'expression de gauche a droite en comencant just apres le premier parenthese et de **compter** les parenthese fermantes **decompter**
Jusqu a on recontre l'operande centrale

```
genere (expression, i,…);
{j:= operateur_central (expression);

--
}
main()
{
genere (E, 1,…); // E est l'expression
      entière; 1 veut dire que la fonction
      travaillera sur  la tache N° 1)
}
```

2-Generer numero de la tache
3-apple de genere()pour travailler sur les sous expression de gauche

```
genere (expression, i,…);
{j:= operateur_central (expression);
generer_numero_tache (i);
s_exG:=s_expression_G(expression, j);
Si existe (s_exG) Alors genere(s_exG, i+1)
      Fsi;
--
}
main()
{
genere (E, 1,…); // E est l'expression entière; 1
      veut dire que la fonction travaillera sur la
      tache N° 1)
}
```

Génération mono-processus

E=((A + B) * (C - (D / E)))

- **Appel de genere () pour travailler sur la sous-expression de gauche**

  Pour l'expression ci-dessus, la sous-expression gauche est

  (A + B)

```
genere (expression, i,…);
{j:= operateur_central (expression);
generer_numero_tache (i);
s_exG:=s_expression_G(expression, j);
Si existe (s_exG) Alors genere(s_exG, i+1)
      Fsi;
--
}
main()
{
genere (E, 1,…); // E est l'expression entière; 1
      veut dire que la fonction travaillera sur la
      tache N° 1)
}
```

4-apple de genere() pour travailler sur l'expression de droite

```
genere (expression, i,…);
{j:= operateur_central (expression);
generer_numero_tache (i);
s_exG:=s_expression_G(expression, j);
x=calculer_nbop (s_exG);
Si existe (s_exG) Alors genere(s_exG, i+1) Fsi;
s_exD:=s_expression_D(expression, j);
Si existe (s_exD) Alors genere(s_exD, i+x+1)
      Fsi;
--
}
main()
{
genere (E, 1,…); // E est l'expression entière; 1
      veut dire que la fonction travaillera sur la
      tache N° 1)
}
```

Il vas compter combien il ya d'operande il va retourner

Génération mono-processus

E=((A + B) * (C - (D / E)))

4- Appel de *genere ()* pour travailler sur la sous expression de droite

- Pour l'expression ci-dessus, la sous-expression de droite est

(C - (D / E))

```
genere (expression, i,…);
{j:= operateur_central (expression);
generer_numero_tache (i);
s_exG:=s_expression_G(expression, j);
x=calculer_nbop (s_exG);
Si existe (s_exG) Alors genere(s_exG, i+1) Fsi;
s_exD:=s_expression_D(expression, j);
Si existe (s_exD) Alors genere(s_exD, i+x+1)
    Fsi;
--
}
main()
{
genere (E, 1,…); // E est l'expression entière; 1
    veut dire que la fonction travaillera sur la
    tache N° 1)
}
```

Génération mono-processus

E=((A + B) * (C - (D / E)))

4- Appel de *genere ()* pour travailler sur la sous expression de droite

- Pour l'expression ci-dessus, la sous-expression de droite est

(C - (D / E))

- Lors du précédant appel, *genere ()* devrait travailler sur toute la s-expression de gauche en faisant si nécessaire d'autres appel récursifs. Cela signifie que plusieurs numéro de taches auraient été éventuellement affectés.
- Pour l'appel concernant la partie de droite, on doit continuer **après le dernier numéro** déjà affecté. Pour simplifier, on va **calculer** alors de nombre d'opération du coté gauche (soit **x**) pour définir le **numéro à utiliser (soit i+x+1)** pour l'appel concernant la partie droite. Pour le cas de l'expression ci-dessus, **x=1**. Donc la numérotation commencera à partir de **3**

```
genere (expression, i,…);
{j:= operateur_central (expression);
generer_numero_tache (i);
s_exG:=s_expression_G(expression, j);
x=calculer_nbop (s_exG);
Si existe (s_exG) Alors genere(s_exG, i+1) Fsi;
s_exD:=s_expression_D(expression, j);
Si existe (s_exD) Alors genere(s_exD, i+x+1)
    Fsi;
--
}
main()
{
genere (E, 1,…); // E est l'expression entière; 1
    veut dire que la fonction travaillera sur la
    tache N° 1)
}
```

5-generation du cntenu de la tache

Génération mono-processus

E=((A + B) * (C - (D / E)))

5- Génération du contenu de la tache.

Dans l'exemple ci-dessus et pour l'expression entière, le contenu de la tache **T1** est : **M1 := M2*M3**

```
genere (expression, i,…);
{j:= operateur_central (expression);
generer_numero_tache (i);
s_exG:=s_expression_G(expression, j);
x=calculer_nbop (s_exG);
Si existe (s_exG) Alors genere(s_exG, i+1) Fsi;
s_exD:=s_expression_D(expression, j);
Si existe (s_exD) Alors genere(s_exD, i+x+1)
    Fsi;
generer_contenu_tache (expression, i, x, j);
--
}
main()
{
genere (E, 1,…); // E est l'expression entière; 1
    veut dire que la fonction travaillera sur la
    tache N° 1)
}
```

TP 1: Concurrence

Génération mono-processus

E=((A + B) * (C - (D / E)))

5- Génération du contenu de la tache.

Dans l'exemple ci-dessus et pour l'expression entière, le contenu de la tache **T1** est : **M1 := M2*M3**

- Quatre cas existent selon que les opérandes de gauche et de droites sont des sous-expressions ou des variables/constantes:
  - Ti: Mi:= Mi+1 <OP> Mi+x+1, où <OP> est à la case j
  - Ti: Mi:= varG <OP> Mi+1, où varG est à la case j-1
  - Ti: Mi:= Mi+1 <OP> varD, où varD est à la case j+1
  - Ti: Mi:= varG <OP> varD

```
genere (expression, i,…);
{j:= operateur_central (expression);
generer_numero_tache (i);
s_exG:=s_expression_G(expression, j);
x=calculer_nbop (s_exG);
Si existe (s_exG) Alors genere(s_exG, i+1) Fsi;
s_exD:=s_expression_D(expression, j);
Si existe (s_exD) Alors genere(s_exD, i+x+1)
    Fsi;
generer_contenu_tache (expression, i, x, j);
--
}
main()
{
genere (E, 1,…); // E est l'expression entière; 1
    veut dire que la fonction travaillera sur la
    tache N° 1)
}
```

5-generation de la precedence avec la tache generee dans la fonction appelante s'il y a lieu

E=((A + B) * (C - (D / E)))

5- Génération de la précédence avec la tache générée dans la fonction appelante s'il y a lieu.

- Dans l'exemple ci-dessus et pour l'expression entière, la tache correspondante à l'opération '*' (T1) n'a pas de tache qui la précède. Par contre, lors du traitement concernant l'opération '+', sa tache correspondante (i.e. **T2**) précède **T1** (i.e. **T2<T1** est donc générée).

```
genere (expression, i, pere);
{j:= operateur_central (expression);
generer_numero_tache (i);
s_exG:=s_expression_G(expression, j);
x=calculer_nbop (s_exG);
Si existe (s_exG) Alors genere(s_exG, i+1, i)
    Fsi;
s_exD:=s_expression_D(expression, j);
Si existe (s_exD) Alors genere(s_exD, i+x+1, i)
    Fsi;
generer_contenu_tache (expression, i, x, j);
generer_precedence (i, pere)
}
main()
{
genere (E, 1, -1); /* E est l'expression entière; 1
    veut dire que la fonction travaillera sur la
    tache N° 1); -1 signifie qu'il s'agit du
    premier appel de genere () */
}
```

Algorithme général

```
genere (expression, i, pere);
{
j:= operateur_central (expression);
generer_numero_tache (i);
s_exG:=s_expression_G(expression, j);
x=calculer_nbop (s_exG);
Si existe (s_exG) Alors genere(s_exG, i+1, i) Fsi;
s_exD:=s_expression_D(expression, j);
Si existe (s_exD) Alors genere(s_exD, i+x+1, i) Fsi;
generer_contenu_tache (expression, i, x, j);
generer_precedence (i, pere)
}
main()
{ Lire (E);
genere (E, 1, -1); /* E est l'expression entière; 1 veut dire que la fonction travaillera sur la
    tache N° 1); -1 signifie qu'il s'agit du premier appel de genere () */
}
```

Déroulement sur l'exemple : E=((A + B) * (C - (D / E)))

2 1 3 4

```
genere (((A+B)*(C-(D/E))), 1, -1)
{j:= operateur_central (((A+B)*(C-(D/E))));          j=5
generer_numero_tache (1);                            T1;
s_exG:=s_expression_G(((A+B)*(C-(D/E))), 5);         s_exG=(A+B)
x=calculer_nbop ((A+B));                             x=1
genere((A+B), 2, 1)
{j:= operateur_central ((A+B));                      j=2
generer_numero_tache (2);                            T2
s_exG:=s_expression_G((A+B), 2);                     s_exG=A
x=calculer_nbop ((A+B));                             x=0
s_exD:=s_expression_D((A+B), 2);                     s_exD=B
generer_contenu_tache ((A+B), 2,0, 2);               T2: M2:=A+B
generer_precedence (2, 1)                            T2<T1
}

s_exD:=s_expression_D(((A+B)*(C-(D/E))), 5);         s_exD=(C-(D/E))
 genere((C-(D/E)), 1+1+1, 1)
{j:= operateur_central ((C-(D/E)));                  j=2
generer_numero_tache (3);                            T3
s_exG:=s_expression_G((C-(D/E)), 2);                 s_exG=C
x=calculer_nbop (C);                                 x=0
s_exD:=s_expression_D((C-(D/E)), 2);                 s_exD=(D/E)
genere((D/E), 3+0+1, 3)
{j:= operateur_central ((D/E));                      j=2
generer_numero_tache (4);                            T4
s_exG:=s_expression_G((D/E), 2);                     s_exG=D
x=calculer_nbop (D);                                 x=0
s_exD:=s_expression_D((D/E), 2);                     s_exD=D
generer_contenu_tache ((D/E), 4,0 , 2);                T4:=D/E
generer_precedence (4, 3)                            T3<T4
}
```

```
generer_contenu_tache ((C-(D/E)), 3, 0, 2);          T3: M3:=C-M4
generer_precedence (3, 1)                            T3<T1
}

generer_contenu_tache (((A+B)*(C-(D/E))), 1, 1, 5);  T1: M1:=M2*M3
generer_precedence (1, -1)
}
```

Error:

j=5 ✗ --> j=6 ☑

Violet: s-exG=s_expression_G((D/E),2) s_exD=D ✗ --->Violet: s-exG=s_expression_G((D/E),2) s_exD=C ☑

Finir tp0-->fichier

Ecrire le programme sequentiel de ce projet