Université de Bejaia.

Faculté des Sciences Exactes.

Département Informatique

Cours
Programmation Avancée

Master 1 en Informatique

Options : RN (RS & SIA)

Prof. BOUALLOUCHE Louiza

Chap. 4 La récursivité

- La récursivité est un art d'écrire des algorithmes qui résolvent des problèmes que l'on ne sait pas résoudre. C'est le cas des problèmes de « la tour de Hanoï », « la traversée de la rivière », ...

- Un programme ou algorithme est récursif lorsqu'il est défini en fonction de lui-même.

- Avantage. La récursivité permet de résoudre certains problèmes d'une manière rapide et les procédures correspondantes sont plus simples et concises.

- Inconvénient. Leur exécution est occasionnée par une perte de temps considérable, et fait appel à l'utilisation des piles ; ce qui engendre une utilisation excessive de la mémoire voire un débordement.

Les différents types de récursivité

1. Récursivité simple.

On parle de récursivité simple lorsque l'algorithme contient un seul appel.

Exemple

$$n! = \begin{cases} 1 & si\ n \leq 1; \\ (n-1)! * n & sinon \end{cases}$$

Fonction *factorielle (n:* ***entier****):* ***entier ;***
Début
    Si *n* ≤ *1* ***alors***
        Retourner *1 ;*
    Sinon
        Retourner *n*factorielle (n-1) ;*
    Fin si
Fin

2. Récursivité Multiple

On parle de récursivité multiple lorsque l'algorithme contient plus d'un appel récursif.

Exemple

$$c_n^p = \begin{cases} 1 & si\ p = 0\ ou\ p = n\ ; \\ c_{n-1}^p + c_{n-1}^{p-1} & sinon \end{cases}$$

Fonction *combinaison (n, p:* ***entier****):* ***entier***

Début

Si *p = 0 ou p = n* ***alors***

Retourner *1 ;*

Sinon

Retourner *combinaison (n-1, p) + combinaison (n-1, p-1);*

Fin si

Fin

2. Récursivité Mutuelle

Des procédures sont mutuellement récursives si elles dépendent les unes des autres.

Exemple

$$Pair(n) = \begin{cases} vrai & si\ n = 0 \\ Impair(n-1) & si\ n > 0 \end{cases}$$

$$Impair(n) = \begin{cases} faux & si\ n = 0 \\ Pair(n-1) & si\ n > 0 \end{cases}$$

```
Fonction Pair (n: entier): bool
Début
        Si n = 0 alors
             Retourner vrai ;
         Sinon
           Retourner Impair (n-1);
        Fin si
Fin
```

```
Fonction Impair (n: entier): bool
Début
        Si n = 0 alors
             Retourner faux ;
          Sinon
             Retourner Pair (n-1);
        Fin si
Fin
```

Récursivité Imbriquée

Une procédure est récursive imbriquée si elle contient des appels récursifs imbriqués.

Exemple. Fonction d'Ackermann

$$A(m,n)=\begin{cases} n+1 & si \quad m=0; \\ A(m-1,1) & si \quad m>0 \; et \; n=0; \\ A(m-1,A(m,n-1)) & si \quad m>0 \; et \; n>0 \end{cases}$$

```
Fonction Ackermann (m , n: entier): entier
Début
    Si m = 0 alors
        Retourner n+1 ;
    Sinon
        Si n = 0 alors
            Retourner Ackermann (m-1 ,1);
        Sinon
            Retourner Ackermann (m-1, Ackermann (m, n-1);
        Fin Si
    Fin Si
Fin
```

Le principe de la programmation récursive

La programmation récursive est un art qui s'appuie sur le principe suivant.

- Connaissance des cas triviaux. Il s'agit d'un certain nombre de cas simples dont la résolution est connue (qui constituent les cas d'arrêt de la récursivité) ;

- Avoir un moyen de passer d'un cas compliqué à un cas simple
  - il faut être sûr que l'on retombera toujours sur un cas connu (cas d'arrêt) ; autrement dit, il faut que la fonction soit complètement définie (sur tout son domaine d'application).

  - Existence d'un ordre strict, tel que la suite des valeurs successives des paramètres soit strictement monotone et finit par atteindre une valeur pour laquelle la solution est explicitement définie. Par exemple, l'exécution de la fonction factorielle (n), engendre cet ordre (de la suite) monotone décroissant

n, n-1, n-2, …, 1 *Si n > 0* *et*
0 *Si n = 0*

L'élimination de la récursivité ou la dérécursivation

La dérécursivation est la transformation d'une procédure récursive en une procédure itérative. Il est toujours possible de dérécursiver. Il existe deux types de récursivité, la récursivité terminale et la non terminale.

Récursivité terminale

Un algorithme est dit récursif terminal s'il ne contient aucun traitement après un appel récursif.

```
Procédure P(U)
    début
        si C(U) alors
            D(U);
            P(α(U)) ;
        sinon
            T(U) ;
        Finsi
    Fin
```

- U : liste des paramètres ;
- C(U) : condition portant sur U ;
- D(U) : traitement de base (dépend de U);
- α(U): transformation des paramètres ;
- T(U) : traitement de terminaison ;

```
Procédure P(U)
    Début
        si C(U) alors
            D(U);
            P(a(U)) ;
        Sinon
            T(U) ;
        Fin si
    Fin.
```

- Version Pd dérécursivée de P

```
Procédure Pd(U)
  Début
    Tant que C(U) faire
        D(U);
        U← a (U) ;
    Fin Tant que
    T(U) ;
  Fin.
```

Exemple. Le calcul de la factorielle.

La fonction récursive

```
Fonction Fact (n: entier): entier ;
Début
        Si n > 1 alors
            Retourner n*Fact (n-1) ;
        Sinon
                Retourner 1 ;
         Fin Si
Fin.
```

La fonction dérécursivée

```
Fonction Fact-dR (n: entier): entier ;
            Var F : Int ;
            Début
              F ← 1 ;
             Tant que  n>1  alors
               F ← n*F;
                n ← n-1 ;
             Fin tant que
             Retourner F ;
            Fin.
```

Récursivité non terminale

Un algorithme est dit récursif non terminal s'il contient un traitement après un appel récursif.

Dans ce cas, il va falloir sauvegarder le contexte de l'appel récursif (les paramètres de l'appel engendrant l'appel récursif).

```
Procédure Q(U)
 Début
      Si C(U) alors
           D(U);
          Q(a(U));
          F(U)
       Sinon
          T(U) ;
      Fin si
 Fin.
```

```
Procédure Qd(U)
 Début
  initPileVide()
  Tant que C(U)  faire
        D(U) ;
        Empiler(U) ;
        U← a(U) ;
  Fin Tant que
  T(U) ;
  Tant que non PileVide()  faire
        Dépiler(U) ;
        F(U) ;
  Fin Tant que
Fin.
```

- **Illustration.** Exemple d'exécution (trace) de Q pour U=U0.

Appel Q(U0)
C(U0) vrai
D(U0)
Appel Q(α (U0))
C(α (U0)) vrai
D(α (U0))
Appel Q(α (α (U0)))
C(α (α (U0))) faux
T(α (α (U0)))
F(α (U0))
***F*(U0)**

```
Procédure Q(U)
  Début
       Si C(U) alors
            D(U);
            Q(a(U));
            F(U)
         Sinon
            T(U) ;
       Fin si
   Fin.
```

Trace de Qd pour U=U0
Appel Qd(U0)
C(U0) vrai
D(U0)
Empiler (U) Pile = [U0]
U← α (U)
C(α (U0)) vrai
D(α (U0))
Empiler (α(U0)) Pile = [U0, α(U0)]
U← α(α (U0))
C(α (α (U0))) faux
T(α (α (U0)))
Dépiler(U) U= α (U0)
F(α (U0))
Dépiler(U) U= U0
***F*(U0)**

```
Procédure Qd(U)
 Début
  initPileVide()
  Tant que C(U)  faire
        D(U) ;
        Empiler(U) ;
        U← a(U) ;
  Fin Tant que
  T(U) ;
  Tant que non PileVide()  faire
        Dépiler(U) ;
        F(U) ;
  Fin Tant que
Fin.
```

Finalement les traces d'exécution sont les mêmes

D(U0)
D(α (U0))
T(α (α (U0)))
F(α (U0))
***F*(U0)**

Exemple. Algorithme de parcours d'une liste.

La procédure récursive.

```
Procedure ParcoursListe(P : Pointeur)

 Début

      Si (P<>Null) alors

            ParcoursListe (P->.suivant);

           Ecrire (P->.info) ;
     Fin Si
 Fin.
```

La procédure dérécursivée

```
Procedure ParcoursListe(P : Pointeur)

Début

   initPileVide();

   Tant que (P<>Null)  faire
        Empiler(P) ;   P← P->.suivant  ;
   Fin Tant que

   Tant que non PileVide()  faire
        Dépiler(P) ;   Ecrire (P->.info) ;
   Fin Tant que
 Fin
```

Dans cet exemple

U == P ; C(U) == P<>NULL ; D(U) == Vide ; α(U) == P->.suivant ; F(U) == Ecrire (P->.info) ; T(U) == Vide.

La version de la fonction du calcul de la factorielle suivante est de type non terminal.

```
Fonction Fact (n: entier): entier ;
Début
        Si n > 1 alors
            Retourner Fact (n-1)*n ;
          Sinon
            Retourner 1 ;
        Fin Si
Fin.
```

Autre forme de récursivité non terminale.

```
Procedure P(U)
  Début

        Tant que C(U) faire
              D(U);
              P(α(U));
              F(U)
       Fin Tant que

Fin
```

```
Procedure Pd(U)
 Début
   initPileVide()
   Tant que C(U) ou non PileVide()  faire
       Tant que C(U)  faire
           D(U) ;
           Empiler(U) ;
           U ← α(U) ;
       Fin Tant que
       Dépiler(U) ;
       F(U) ;
   Fin Tant que
Fin
```

Remarque.

Les compilateurs savent la plupart du temps reconnaître les appels récursifs terminaux et ceux-ci n'engendrent pas de surcroît par rapport à la version itérative du même programme.