Récursivité

Dr Khadim DRAME
kdrame@univ-zig.sn

Département d'Informatique
UFR des Sciences et Technologies
Université Assane Seck de Ziguinchor

6 janvier 2022

Objectif général du cours

- Définir et utiliser ses propres fonctions récursives en Pascal.

Plan

C'est quoi la récursivité ?

- La **récursivité** correspond à la **récurrence** en Mathématiques.
- C'est un style de programmation relativement simple, intuitif et élégante pour résoudre des problèmes
  - programmes plus lisibles et plus concis ;
  - adapté à certains types de données.
- Une **fonction** (ou procédure) **récursive** est une fonction (ou procédure) qui fait appel à elle même dans sa définition.

Introduction

Exemples de fonctions récursives

- Calcul de la somme des entiers de 1 à n

$$somme(n) = \sum_{i=1}^{n} i = n + \sum_{i=1}^{n-1} i$$

$$somme(n) = n + somme(n-1)$$

- Calcul de la factorielle de n

$$n! = \prod_{i=1}^{n} i = n \times \prod_{i=1}^{n-1} i$$

$$n! = n \times (n-1)!$$

Plan

Méthode de conception

Technique pour définir une fonction récursive

- Identifier tous les cas à prendre en considération.
- Définir d'abord le(s) **cas de base** (ceux qui ne nécessitent pas un appel récursif de la fonction).
- Définir les autres cas, ceux faisant des appels récursifs.
- Généraliser et simplifier.

Méthode de conception

Test d'arrêt

- C'est un cas particulier ne faisant pas d'appel récursif ;
- Il est indispensable, sinon le programme ne s'arrête jamais !
- Il doit être testé en premier.
- On parle aussi de conditions de terminaison.

Appel(s) récursif(s)

- Si le test d'arrêt n'est pas vérifié, on fait des appels récursifs ;
- Ils doivent renvoyer au test d'arrêt.

Méthode de conception

Structure d'une fonction/procédure récursive

```
function <nom_fonction>(<args>):<type_retour>;
begin
  if(<cas particulier>) then
    <traitement direct du cas particulier>
    {pas d'appel récursif}
  else
    <traitement des appels récursifs>
end;
```

Exemples

- Exemple 1

```
1 function somme(n: integer):integer;
2 begin
3   if(n=0) then
4     somme:=0 {test d'arrêt}
5   else
6     somme:= n+somme(n-1);{appel récursif}
7 end;
```

- Exemple 2

```
1 function fact(n: integer):integer;
2 begin
3   if(n=0) then
4     fact:=1 {test d'arrêt}
5   else
6     fact:= n*fact(n-1); {appel récursif}
7 end;
```

Fonctionnement

Pile d'exécution

La Pile d'exécution (call stack) du programme en cours est un emplacement mémoire destiner à mémoriser les paramètres, les variables locales et l'adresse de retour de chaque fonction en cours d'exécution.

Fonctionnement

- Les appels récursifs sont empilés.
- Lorsque la condition d'arrêt est atteinte, les appels précédents sont dépilés un à un.

Fonctionnement

Exemple (factorielle)

fact(5)

$5 \neq 0 \rightarrow 5 * \text{fact}(4)$

$4 \neq 0 \rightarrow 5 * 4 * \text{fact}(3)$

$3 \neq 0 \rightarrow 5 * 4 * 3 * \text{fact}(2)$

$2 \neq 0 \rightarrow 5 * 4 * 3 * 2 * \text{fact}(1)$

$1 \neq 0 \rightarrow 5 * 4 * 3 * 2 * 1 * \text{fact}(0)$

$0 = 0 \rightarrow 5 * 4 * 3 * 2 * 1 * 1$

$\rightarrow 5 * 4 * 3 * 2 * 1$

$\rightarrow 5 * 4 * 3 * 2$

$\rightarrow 5 * 4 * 6$

$\rightarrow 5 * 24$

$\rightarrow 120$

Plan

Récursivité simple vs récursivité multiple

Récursivité simple

- Une fonction est dite récursive simple si un seul appel récursif est activé à chaque fois.

Récursivité multiple

- Une fonction est dite récursive multiple si deux (récursivité double) ou plusieurs rappels récursif peuvent être activés à chaque fois.
- Exemple (suite de Fibonacci)

$$u_0 = 0$$
$$u_1 = 1$$
$$u_n = u_{n-1} + u_{n-2} \ \forall n \geqslant 2$$

Récursivité mutuelle ou croisée

Fonctions qui s'appellent mutuellement. Utiliser le mot clé **forward**.

```
function impair(n: integer):boolean; forward ;
function pair(n: integer):boolean;
begin
  if(n=0) then
    pair:=true
  else
    pair:= impair(n-1);
end;
```

```
function impair(n: integer):boolean;
begin
  if(n=1) then
    impair:=true
  else
    impair:= pair(n-1);
end;
```

Récursivité imbriquée

Récursivité imbriquée

- On parle de récursivité imbriquée si un appel récursif a comme argument un autre appel récursif à la même fonction.
- Exemple 1 (fonction de McCarthy)

$$f(n) = \begin{cases} n - 10 & si\ n > 100 \\ f(f(n+11)) & si\ n \leq 100 \end{cases}$$

- Exemple 2 (fonction d'Ackermann)

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

Récursivité enveloppée vs récursivité terminale

Récursivité enveloppée

- Une fonction récursive est dite enveloppée si une instruction est exécutée après l'appel de la fonction à elle-même.
- Dans ce cas, les appels récursif sont empilés.
- Toute récursion double est enveloppée.

Récursivité terminale

- Une fonction récursive est dite **terminale** si aucune instruction n'est exécutée après l'appel de la fonction à elle-même.
- Le résultat retourné est celui du dernier appel récursif de la fonction.

Plan

Avantages/inconvénients de la récursivité

- Tout programme écrit avec la boucle **for** peut se transformer en une fonction récursive.

- Version itérative

```
1 function fact_it(n:
    integer):integer;
2 var i, res: integer;
3 begin
4   res:=1;
5   for i:=1 to n do
6     res:=res*i;
7   fact_it:=res;
8 end;
```

- Version récursive

```
1 function fact_r(n: integer)
    :integer;
2 begin
3   if(n=0) then
4     fact_r:=1
5   else
6     fact_r:=n*fact_r(n-1);
7 end;
```

Expressivité de la récursivité

- La plupart des traitements sur les tableaux peuvent se faire sous forme récursive :
  - tris ;
  - recherche séquentielle ;
  - inversion ;
  - etc.
- La plupart des traitements itératifs simples peuvent se traduire facilement sous forme récursive.
- L'écriture sous forme récursive est souvent plus simple.

Avantages/inconvénients de la récursivité

- Avantages
  - c'est un style de programmation simple, intuitif et élégant ;
  - les programmes sont lisibles et concis.
- Inconvénients
  - elle est souvent **coûteuse en temps d'exécution et en mémoire** ;
  - elle n'est pas prise en compte dans certains langages.
- Pour pallier ces inconvénients $\Rightarrow$ **récursion terminale**.

Plan

Exercices d'application

- Exercice 1

  On considère les fonctions f et g ci-dessous :

```
1  function g(x: integer): integer;  forward ;
2  function f(x: integer): integer;
3  begin
4      if x <= 1 then
5          f:=1
6      else
7          f:=g(x+2);
8  end;
9
10 function g(x: integer): integer;
11 begin
12     g:=f(x-3) + 4
13  end;
```

  Quelle est la valeur de f(4) ?

Exercices d'application

- Exercice 2

  On considère la fonction f ci-dessous :

```
1 function f(n: integer): integer;
2 begin
3     if n > 100 then
4         f:=n-10
5     else
6         f:=f(f(n+11));
7 end;
```

  Quelle est la valeur de f(98) ?

Exercices d'application

- Exercice 3
  1. Définir une fonction récursive **puissance** permettant de calculer la puissance d'un entier.
  2. Définir une fonction récursive **modulo** permettant de calculer le reste de la division entre 2 entiers.
  3. Définir une fonction récursive **moyenne** permettant de calculer la moyenne les éléments d'un tableau. triée.

Exercices d'application

- Correction exercice 3

```
function puissance(x: real; n: integer): real;
begin
    if n = 0 then
        puissance:=1
    else
        puissance:=x*puissance(x,n-1);
end;
```

```
function modulo(a,b: integer): integer;
begin
    if a < b then
      modulo:=a
    else
        modulo:=modulo(a-b,b);
end;
```

Exercices d'application

- Correction exercice 3 (suite)

```
1 function moyenne(t: tab; n: integer): real;
2 begin
3     if n = 1 then
4       moyenne:=t[1]
5     else
6        moyenne:=((n-1)*moyenne(t,n-1)+t[n])/n;
7 end;
```

Résumé

- La récursivité est un moyen naturel, élégant et concis de résoudre des problèmes.
- La plupart des problèmes algorithmiques admettent une solution récursive, souvent plus simple.
- Une fonction récursive doit comporter
  - un test d'arrêt dans lequel aucun appel récursif n'est effectué ;
  - un cas général dans lequel un ou plusieurs appel(s) récursif(s) sont effectués.
- Les appels récursifs doivent renvoyer au test d'arrêt.