Récursivité des actions [rc]
Exercices de cours

Karine Zampieri, Stéphane Rivière

Unisciel algoprog Version 21 mai 2018

Table des matières

alg - Exercices de cours (Solution)

Mots-Clés Récursivité des actions ■
Difficulté ●●○ (3 h) ■

1 Appréhender le cours

1.1 Fonction factoriel / pgfactoriel

Objectif

Le factoriel d'un entier naturel n est défini par la relation de récurrence :

$$\begin{cases} 0! = 1 \\ n! = n \cdot (n-1)! \end{cases}$$

Donnez le type de récursivité, la condition d'arrêt et la récurrence.

Solution simple

C'est une récursivité simple de condition d'arrêt $0! = 1$ et de récurrence :

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

Déduisez une fonction récursive `factoriel(n)` qui calcule et renvoie le factoriel de `n` (entier).

Validez votre fonction avec la solution.

Solution alg @[pgfactoriel.alg]

```
Fonction factoriel ( n : Entier ) : Entier
Début
  |  Si ( n <= 0 ) Alors
  |    |  Retourner ( 1 )
  |  Sinon
  |    |  Retourner ( n * factoriel ( n - 1 ) )
  |  FinSi
Fin
```

Explicitez le calcul de `fac(3)` (abrégé de `factoriel(3)`).

Solution simple

Chaque fonction s'exécute dans son environnement de données associé : lors de l'exécution du calcul de `fac(0)|, il y a quatre variables \lstinline`n@ définies avec quatre valeurs différentes dans chaque environnement de données. Précisons le déroulement des calculs :

1. Le calcul de `fac(3)` est lancé (étape 1).
2. Pour évaluer la valeur de `3*fac(2)`, le calcul de `fac(3)` se suspend pour connaître la valeur de `fac(2)` (étape 2).

3. Le calcul de `fac(2)` se suspend à son tour pour évaluer `fac(1)` (étape 3), lequel se suspend également pour évaluer `fac(0)`.
4. Grâce à la condition d'arrêt, `fac(0)| renvoie 1: cette valeur remplace \lstinline`fac(0)@ dans le calcul suspendu de `fac(1)`.
5. Le calcul de `fac(1)` peut reprendre là où il était suspendu et s'effectuer, `fac(1)` renvoie 1, et le calcul de `fac(2)` peut reprendre et s'effectuer pour produire 2, puis le calcul de `fac(3)` reprend et s'effectue pour produire 6.

Finalement, on a calculé :

```
fac(3) -> 3*fac(2) -> 3*2*fac(1) -> 3*2*1*fac(0) -> 3*2*1*1 -> 6
```

Écrivez une fonction récursive terminale `facRt(n,a)` qui calcule et renvoie le factoriel de `n` (entier), l'entier `a` étant le résultat.

Écrivez une fonction maître `factorielRt(n)` qui lance la fonction récursive terminale.

Validez vos fonctions avec la solution.

Solution alg @[pgfactoriel.alg]

```
Fonction factorielRt ( n : Entier ) : Entier
Début
  |  Retourner ( facRt ( n , 1 ) )
Fin
Fonction facRt ( n : Entier ; a : Entier ) : Entier
Début
  |  Si ( n <= 0 ) Alors
  |    |  Retourner ( a )
  |  Sinon
  |    |  Retourner ( facRt ( n - 1 , a * n ) )
  |  FinSi
Fin
```

Écrivez un algorithme qui saisit un entier et teste les fonctions.

1.2 Boucles récursives / pgboucles

Écrivez le **profil** d'une procédure récursive `afficherBoucleUp(n1,n2)` qui affiche les entiers de `n1` à `n2` en ordre croissant.

Donnez le type de récursivité, la condition d'arrêt et la récurrence.

Solution simple
C'est une récursivité simple de condition d'arrêt `n1>n2` et de récurrence `n1+1`.

Déduisez le corps de la procédure récursive.
Exemple :

```
afficherBoucleUp(-3,5) ==> -3 -2 -1 0 1 2 3 4 5
```

Validez votre procédure avec la solution.

Solution alg @[pgboucles.alg]

```
Action afficherBoucleUp ( n1 : Entier ; n2 : Entier )
Début
  |  Si ( n1 <= n2 ) Alors
  |    |  Afficher ( n1 )
  |    |  afficherBoucleUp ( n1 + 1 , n2 )
  |  FinSi
Fin
```

En utilisant les mêmes principes, écrivez une procédure récursive `afficherBoucleDn(n1,n2)` qui affiche les entiers de `n1` à `n2` en ordre décroissant. Exemple :

```
afficherBoucleDn(-3,5) ==> 5 4 3 2 1 0 -1 -2 -3
```

Validez votre procédure avec la solution.

Solution alg @[pgboucles.alg]

```
Action afficherBoucleDn ( n1 : Entier ; n2 : Entier )
Début
  |  Si ( n1 <= n2 ) Alors
  |    |  afficherBoucleDn ( n1 + 1 , n2 )
  |    |  Afficher ( n1 )
  |  FinSi
Fin
```

Écrivez un algorithme qui saisit deux entiers et teste les procédures.

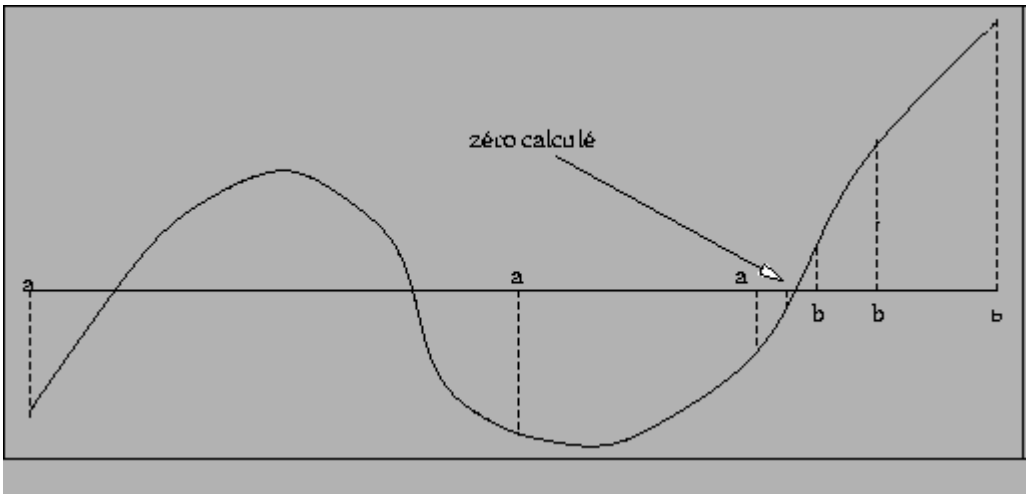
Validez votre algorithme avec la solution.

Solution alg @[pgboucles.alg]

```
Algorithme pgboucle
Variable n1 , n2 : Entier
Début
  |  Afficher ( "Deux entiers n1 <= n2? " )
  |  Saisir ( n1 , n2 )
  |  afficherBoucleUp ( n1 , n2 )
  |  afficherBoucleDn ( n1 , n2 )
Fin
```

1.3 Recherche d'un zéro / pgzerodich

Objectif
On cherche à calculer un zéro d'une fonction réelle continue f sur un intervalle $[a, b]$, prenant des valeurs de signes opposés aux extrémités. Le théorème des valeurs intermédiaires assure l'existence d'un zéro. L'idée de la dichotomie est de chercher un zéro sur $[a, (a+b)/2]$ ou bien sur $[(a+b)/2, b]$, selon le signe de $f((a+b)/2)$.



Écrivez une fonction récursive `zerodich(a,b,epsilon)` qui calcule et renvoie un zéro d'une fonction réelle continue `f` sur un intervalle `[a,b]` à `epsilon` près.

Validez votre fonction avec la solution.

Solution alg @[pgzerodich.alg]

```
Remarque Suppose que f(a)*f(b) < 0
Fonction zerodicho ( a , b , epsilon : Réel ) : Réel
Variable c , fc : Réel
Début
  |  c <- ( a + b ) / 2
  |  fc = f ( c )
  |  Si ( Abs ( fc ) < epsilon ) Alors
  |    |  Retourner ( c )
  |  Sinon
  |    |  Si ( f ( a ) * fc < 0 ) Alors
  |    |    |  Retourner ( zerodicho ( a , c ) )
  |    |  Sinon
  |    |    |  Retourner ( zerodicho ( c , b ) )
  |    |  FinSi
  |  FinSi
Fin
```

1.4 Carré d'un entier / pgcarrei

Écrivez une fonction récursive `carrei(n)` qui calcule et renvoie le carré d'un entier positif `n` par la méthode des impairs.

Orientation

Analysons le problème :

Étape 1 Commençons par chercher l'expression commune à tous les appels récursifs de la fonction considérée. Prenons quelques exemples :

```
4^2 = 1 + 3 + 5 + 7
3^2 = 1 + 3 + 5
2^2 = 1 + 3
1^2 = 1
```

Nous pouvons les réécrire de la manière suivante :

```
4^2 = 3^2 + 7
3^2 = 2^2 + 5
2^2 = 1^2 + 3
1^2 = 1
```

ce qui donne l'expression commune à chaque étape : le carré d'un valeur `n` est `(2*n-1)` additionné au carré de `(n-1)`.

Vérifions :

```
carrei(4) = carrei(3) + 7
carrei(3) = carrei(2) + 5
```

Étape 2 Trouvez le point d'arrêt de la récursivité.
On connaît la valeur de 1^2 qui vaut 1. Donc 1 est notre point d'arrêt.

Étape 3 Vérifiez qu'à chaque appel, la fonction se rapproche du point d'arrêt.
Le point de départ de notre calcul est une valeur n strictement positive. A chaque étape, on diminue n de 1, donc on finira bien par atteindre la valeur $n = 1$.

Validez votre fonction avec la solution.

Écrivez un algorithme qui teste votre fonction.

Testez.

Validez votre algorithme avec la solution.

2 Appliquer le cours

2.1 Fonction divisible / pgdivisible

Propriété
Un entier n est divisible par un entier p, si (et seulement si) le reste de la division entière de n par p (c.-à-d. le modulo) est nul.

Dans le cas où $0 < b \leq a$, écrivez une fonction récursive `divisible1(a,b)` qui teste et renvoie `Vrai` si un entier `a` est divisible par un entier `b`.

De même, dans le cas où $a < b$ (avec $b > 0$), écrivez une fonction récursive `divisible2(a,b)` qui teste et renvoie `Vrai` si un entier `a` est divisible par un entier `b`.

Validez vos fonctions avec la solution.

Solution alg @[pgdivisible.alg]

```
Fonction divisible1 ( a , b : Entier ) : Booléen
Début
  |  Si ( a <= b ) Alors
  |    |  Retourner ( a = b )
  |  Sinon
  |    |  Retourner ( divisible1 ( a - b , b ) )
  |  FinSi
Fin
```

```
Fonction divisible2 ( a , b : Entier ) : Booléen
Début
  |  Si ( a >= b ) Alors
  |    |  Retourner ( a = b )
  |  Sinon
  |    |  Retourner ( divisible2 ( a + b , b ) )
  |  FinSi
Fin
```

Déduisez une fonction maître `divisible(n,p)` qui teste et renvoie `Vrai` si un entier `n` est divisible par un entier `p`.

Validez votre fonction avec la solution.

Solution alg @[pgdivisible.alg]

```
Fonction divisible ( n , p : Entier ) : Booléen
Variable a , b : Entier
Début
  |  a <- n
```

```
  |  b <- p
  |  Si ( b < 0 ) Alors
  |    |  a <- - a
  |    |  b <- - b
  |  FinSi
  |  Si ( b = 0 ) Alors
  |    |  Retourner ( Faux )
  |  Sinon
  |    |  Si ( a >= b ) Alors
  |    |    |  Retourner ( divisible1 ( a , b ) )
  |    |  Sinon
  |    |    |  Retourner ( divisible2 ( a , b ) )
  |    |  FinSi
  |  FinSi
Fin
```

Écrivez un algorithme qui saisit deux entiers puis teste votre fonction.

Testez.

```
Vos deux entiers? 125632 256
==>divisible(a,b) vaut Faux
```

Validez votre algorithme avec la solution.

Solution alg @[pgdivisible.alg]

```
Algorithme pgdivisible
Variable a , b : Entier
Début
  |  Afficher ( "Vos deux entiers? " )
  |  Saisir ( a , b )
  |  Afficher ( "==> divisible(a,b) vaut " , divisible ( a , b ) )
Fin
```

2.2 Jeu ”devinez le nombre” / pgdeviner

Objectif
Le jeu « devinez le nombre de 1 à 100 » est le suivant :

- La personne #1 choisit secrètement un entier X de 1 à 100
- La personne #2 propose à la personne #1 un entier P de 1 à 100, qui lui répond
  - C'est exact si $X = P$
  - Plus bas si $X < P$
  - Plus haut si $X > P$
- La personne #2 doit répéter jusqu'à ce que le nombre est deviné.

Quelle est la stratégie la plus efficace pour trouver l'entier mystère ?

Solution simple
Utiliser une recherche dichotomique.

Écrivez le profil et le début de la procédure récursive `deviner(vmin,vmax,mystere)` du jeu : elle demande à l'utilisateur un entier compris dans `[vmin..vmax]`, l'entier `mystere` étant le nombre à deviner.

Écrivez les trois cas de la récursion.
Affichez l'invite :

```
Devinez l'entier entre [vmin] et [vmax]?
```

Validez votre procédure avec la solution.

Solution alg @[pgdeviner.alg]

```
Action deviner ( vmin , vmax , mystere : Entier )
Variable nombre : Entier
Début
  |  Afficher ( "Devinez l'entier entre " , vmin , " et " , vmax , "? " )
  |  Saisir ( nombre )
  |  Si ( mystere = nombre ) Alors
  |     |  Afficher ( "C'est exact" )
  |  Sinon
  |     |  Si ( mystere < nombre ) Alors
  |     |     |  Afficher ( "Plus bas" )
  |     |     |  deviner ( vmin , nombre - 1 , mystere )
  |     |  Sinon
  |     |     |  Afficher ( "Plus haut" )
  |     |     |  deviner ( nombre + 1 , vmax , mystere )
  |     |  FinSi
  |  FinSi
Fin
```

Écrivez un algorithme qui lance le jeu entre 1 et 100 en tirant au hasard l'entier mystère.

Outil alg
La fonction `Aléatoire(n)` renvoie un entier pseudo-aléatoire dans `[0..n-1]`.

Testez.

Validez votre algorithme avec la solution.

Solution alg @[pgdeviner.alg]

```
Algorithme pgdeviner
Début
  |  deviner ( 1 , 100 , Aléatoire ( 100 ) + 1 )
Fin
```

2.3 Procédure quorest / pgquorest

Propriété
La relation de la division entière est :

$$a = q_{n^*}\, b + r_{n^*} \text{ avec } 0 \leq r_{n^*} < b$$

les suites étant définies par :

$$\begin{cases} q_n = q_{n-1} + 1 \\ r_n = r_{n-1} - b \\ q_0 = 0,\ r_0 = a \end{cases}$$

Prouvez la relation.

Solution simple
En effet, à l'initialisation $a = 0 \cdot b + a = a$. Et si l'on suppose la relation vraie jusqu'à l'ordre k, à l'ordre suivant :

$$\begin{aligned} a &= (q_k + 1)\, b + (r_k - b) \\ &= q_k\, b + r_k \text{ qui est vraie} \end{aligned}$$

Écrivez une procédure récursive `quorest(a,b,quotient,reste)` qui calcule dans `quotient` le quotient entier de la division d'un entier `a` par un entier `b` et dans `reste` le reste de cette division. Les entiers `a` et `b` sont supposés positifs.

Validez votre procédure avec la solution.

Solution alg @[pgquorest.alg]

```
Action quorest ( a : Entier ; b : Entier ; R quotient : Entier ; R reste : Entier )
Début
  |  Si ( a < b ) Alors
  |     |  quotient <- 0
  |     |  reste <- a
  |  Sinon
  |     |  quorest ( a - b , b , quotient , reste )
  |     |  quotient <- quotient + 1
  |  FinSi
Fin
```

Écrivez un algorithme qui saisit deux entiers positifs puis calcule et affiche l'opération de la division entière.

Testez. Exemple d'exécution :

```
Deux entiers positifs? 50 6
50 = 8 * 6 + 2
```

Validez votre algorithme avec la solution.

Solution alg @[pgquorest.alg]

```
Algorithme pgquorest
Variable a , b : Entier
Variable quotient , reste : Entier
Début
  |  Afficher ( "Deux entiers positifs? " )
  |  Saisir ( a , b )
  |  quorest ( a , b , quotient , reste )
  |  Afficher ( a , " = " , quotient , " * " , b , " + " , reste )
Fin
```

2.4 Nombre encadré / pgcnombre

Objectif
Cet exercice demande deux entiers puis affiche le premier entier, entouré d'autant de paires de crochets `"["` et `"]"` qu'indiqué par la valeur du deuxième nombre (supposé positif ou nul). Exemples d'exécution :

```
42 3
==> [[[42]]]
```

```
24 0
==> 24
```

Quelle est la condition d'arrêt ?

Solution simple
C'est que `nc` (nombre de crochets) vaut zéro.

Et la récurrence ?

Solution simple
C'est :

1. Affichez un crochet ouvrant
2. Appelez récursivement avec `nc-1` paires de crochets
3. Affichez un crochet fermant

Écrivez une procédure récursive `afficherCNombre(n,nc)` qui affiche un entier `n` entouré de `nc` (entier) de paires de crochets `"["` et `"]"`.

Validez votre procédure avec la solution.

Solution alg @[pgcnombre.alg]

```
Action afficherCNombre ( n : Entier ; nc : Entier )
Début
  |  Si ( nc = 0 ) Alors
  |    |  Afficher ( n )
  |  Sinon
  |    |  Afficher ( "[" )
  |    |  afficherCNombre ( n , nc - 1 )
  |    |  Afficher ( "]" )
  |  FinSi
Fin
```

Écrivez un algorithme qui saisit deux entiers et teste la procédure.

Testez.

Validez votre algorithme avec la solution.

Solution alg @[pgcnombre.alg]

```
Algorithme pgcnombre
Variable n , nc : Entier
Début
  |  Afficher ( "Un entier? " )
  |  Saisir ( n )
  |  Afficher ( "Nombre de crochets? " )
  |  Saisir ( nc )
  |  afficherCNombre ( n , nc )
Fin
```

2.5 Coefficients binomiaux / binome

Propriété

Les coefficients du binôme $\binom{n}{k}$ $(1 \leq k < n)$ sont calculables récursivement selon :

- si $k = 0$ ou $k = n$: retourner 1
- sinon retourner $\binom{n-1}{k-1} + \binom{n-1}{k}$

Donnez le type de récursivité, la condition d'arrêt et la récurrence.

Solution simple

C'est une récursivité multiple de conditions d'arrêts $p = 0$ ou $p > n$ et de récurrence la formule ci-dessus.

Écrivez une fonction récursive `binome(n,p)` qui calcule et renvoie le coefficient binomial $\binom{n}{p}$.

Validez votre fonction avec la solution.

Solution alg @[binome.alg]

```
Fonction binome ( n , p : Entier ) : Entier
Début
  |  Si ( p = 0 ) Alors
  |     |  Retourner ( 1 )
  |  Sinon
  |     |  Si ( p > n ) Alors
  |     |     |  Retourner ( 0 )
  |     |  Sinon
  |     |     |  Retourner ( binome ( n - 1 , p ) + binome ( n - 1 , p - 1 ) )
  |     |  FinSi
  |  FinSi
Fin
```

Quelle est la complexité de la traduction directe de cet algorithme ?

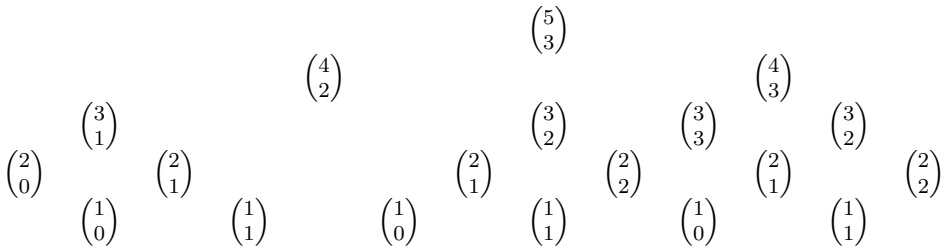
Solution simple

Il est exponentiel car pour chaque étape de calcul il faut recalculer plusieurs fois les coefficients binomiaux d'ordre inférieur.

Proposez un autre algorithme **récursif** ayant cette fois une complexité polynomiale.

Aide simple

Essayez de mettre en évidence une autre relation de récurrence en vous aidant du fait que $\binom{n}{k} = \frac{n!}{k!(n-k)!}$.



Solution simple

En développant la relation :

$$\begin{aligned}\binom{n}{k} &= \frac{n!}{k! \cdot (n-k)!} = \frac{n \cdot (n-1) \cdot \ldots \cdot (n-k+1)}{k \cdot (k-1) \cdot \ldots \cdot 1} \\ &= \frac{n}{k} \cdot \frac{(n-1) \cdot \ldots \cdot (n-k+1)}{(k-1) \cdot \ldots \cdot 1}\end{aligned}$$

En terme de récurrence, on peut établir la nouvelle relation :

$$\binom{n}{k} = \binom{n-1}{k-1} \cdot \frac{n}{k} \text{ si } k > 0 \quad \text{et} \quad \binom{n}{0} = 1$$

La double récursion est ainsi supprimée et il y a au plus $k+1$ appels de fonction. L'algorithme est polynomial en $O(k)$.

Écrivez une fonction récursive `binomeRec(n,k)` qui calcule et renvoie le coefficient binomial $\binom{n}{k}$ qui évite la double récursion.

Validez votre fonction avec la solution.

2.6 Chiffres d'un entier / pgchiffres

Objectif
Cet exercice demande un entier puis affiche la suite de ses chiffres de la droite vers la gauche puis de la gauche vers la droite. Exemple d'exécution :

```
Votre entier? 12354
==> De droite à gauche
45321
==> De gauche à droite
12354
```

Écrivez une procédure récursive `afficherChiffreDG(n)` qui affiche la suite des chiffres d'un entier `n` de la droite vers la gauche.

Validez votre procédure avec la solution.

Solution alg @[pgchiffres.alg]

```
Action afficherChiffreDG ( n : Entier )
Début
  |  Si ( n > 0 ) Alors
  |    |  Afficher ( Modulo ( n , 10 ) )
  |    |  afficherChiffreDG ( DivEnt ( n , 10 ) )
  |  FinSi
Fin
```

De même, écrivez une procédure récursive `afficherChiffreGD(n)` qui affiche la suite des chiffres d'un entier `n` de la gauche vers la droite.

Validez votre procédure avec la solution.

Solution alg @[pgchiffres.alg]

```
Action afficherChiffreGD ( n : Entier )
Début
  |  Si ( n > 0 ) Alors
  |    |  afficherChiffreGD ( DivEnt ( n , 10 ) )
  |    |  Afficher ( Modulo ( n , 10 ) )
  |  FinSi
Fin
```

Écrivez un algorithme qui saisit un entier et teste vos procédures.

Testez.

Validez votre algorithme avec la solution.

Solution alg @[pgchiffres.alg]

```
Algorithme pgchiffres
Variable n : Entier
Début
  |  Afficher ( "Votre entier? " )
  |  Saisir ( n )
  |  Afficher ( "==> De droite à gauche " )
  |  afficherChiffreDG ( n )
  |  Afficher ( "==> De gauche à droite " )
  |  afficherChiffreGD ( n )
Fin
```

2.7 Code du coffre-fort / pgcoffrefort

Afin de protéger l'accès au coffre-fort, un nombre aléatoire est généré pour servir de code à la prochaine ouverture. Ce nombre aléatoire fait moins de 10 chiffres. De plus, le nombre de chiffres pairs de ce nombre est rajouté comme chiffre des unités.

Écrivez une version récursive `nbChiffPairs(nb)` qui calcule le dernier chiffre, à rajouter au nombre donné, pour former le code du coffre-fort.

Solution simple

Si `nb` vaut zéro, alors c'est 1. Sinon il faut analyser chaque chiffre de `nb` et regarder s'il est pair : si oui, il y en a un en plus. D'où il convient de définir une fonction récursive `nbChiffPairsRec(nb)` qui calcule le nombre de chiffres pairs d'un nombre donné, supposé non nul positif pour faire ce calcul.

Validez votre fonction avec la solution.

Testez.

Validez votre algorithme avec la solution.

3 Approfondir le cours

3.1 Fonction 91 de Mac-Carthy / carthy

Définition
Introduite par MAC-CARTHY pour montrer certains pièges de la récursivité, considérons la fonction récursive :

$$F(n) = \begin{cases} n - 10 & \text{si } n > 100 \\ F(F(n+11)) & \text{sinon} \end{cases}$$

Écrivez une fonction `carthy(n)` de la définition.

Validez votre fonction avec la solution.

Solution alg @[carthy.alg]

```
Fonction carthy ( n : Entier ) : Entier
Début
  |  Si ( n > 100 ) Alors
  |    |  Retourner ( n - 10 )
  |  Sinon
  |    |  Retourner ( carthy ( carthy ( n + 11 ) ) )
  |  FinSi
Fin
```

Trouvez la valeur de F pour tout n.

3.2 Fonction de Morris / morris

Définition

La fonction dite « de Morris » est définie par (m et n entiers naturels) :

$$\text{morris}(m,n) = \begin{cases} 1 & \text{si } m = 0 \\ \text{morris}(m-1, \text{morris}(m,n)) & \text{sinon} \end{cases}$$

Écrivez une fonction `morris(m,n)` de la définition.

Validez votre fonction avec la solution.

Solution alg @[morris.alg]

```
Remarque Suppose m et n entiers naturels
Fonction morris ( m , n : Entier ) : Entier
Début
  |  Si ( m = 0 ) Alors
  |    |  Retourner ( 1 )
  |  Sinon
  |    |  Retourner ( morris ( m - 1 , morris ( m , n ) )
  |  FinSi
Fin
```

Quel est le problème de cette fonction ?

Solution simple

Une preuve trop rapide de terminaison de cette fonction conduirait à écrire :

- On considère l'ordre lexicographique sur $\mathbb{N}^2$.
- Le calcul de `morris(m,n)` fait appel au calcul de `morris(m-1,X)` et $(m-1, X) \prec (m, n)$ pour tout X.
- La fonction termine lorsque son premier paramètre est nul.

Mais $X = morris(m,n)$. Alors l'exécution de `morris(1,0)` ne termine pas car ce calcul conduit à l'exécution de `morris(0,morris(1,0))`, donc à nouveau au calcul de `morris(1,0)` et ainsi de suite. En effet, dans la plupart des langages de programmation, les paramètres sont évalués avant d'être passés à la fonction (on parle « d'appel par valeurs ») : le langage évalue en priorité les expressions les plus profondes d'une expression donnée. Bref, il faut être attentif !

3.3 Identité et fonction unité / identite

Définition

Voici une « définition » farfelue mais correcte de l'application identité Id sur les entiers naturels :

$$\begin{cases} Id(0) = 0 \\ Id(n) = n * Un(n-1), \forall n \geq 1 \end{cases}$$

où Un désigne la fonction constante qui a tout n associe 1, que l'on peut définir de la manière suivante :

$$\begin{cases} Un(0) = 1 \\ Un(n) = Id(n) - Id(n-1), \forall n \geq 1 \end{cases}$$

Écrivez des fonctions mutuellement récursives `Id(n)` et `Un(n)` avec `n` entier positif ou nul.

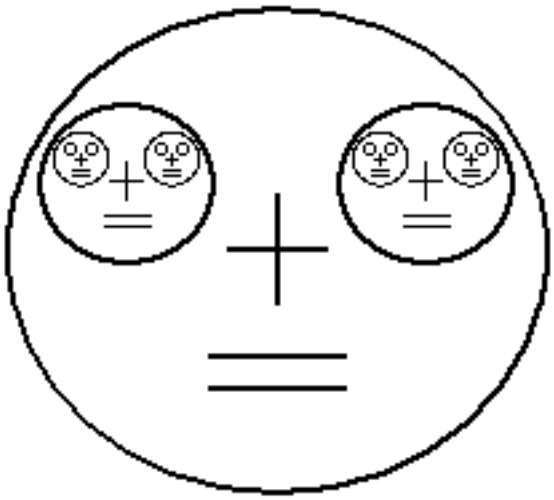
Validez votre fonction avec la solution.

Solution alg @[identite.alg]

```
Fonction Id ( n : Entier ) : Entier
Début
  |  Si ( n = 0 ) Alors
  |    |  Retourner ( 0 )
  |  Sinon
  |    |  Retourner ( n * Un ( n - 1 ) )
  |  FinSi
Fin
Fonction Un ( n : Entier ) : Entier
Début
  |  Si ( n = 0 ) Alors
  |    |  Retourner ( 1 )
  |  Sinon
  |    |  Retourner ( Id ( n ) - Id ( n - 1 ) )
  |  FinSi
Fin
```

3.4 0 + 0 = la tête à Gogo / pgtete

Objectif
Comme vous le savez, 0+0=0.



On pourrait aussi dire 0=(0+0). Dans ce cas, on peut aussi aller un peu plus loin, et puisque 0 vaut (0+0), remplacer les 0 de (0+0) par leur valeur, et obtenir :

```
0 = ((0 + 0) + (0 + 0))
```

Rien n'empêche de continuer et d'écrire :

```
0 = (((0 + 0) + (0 + 0)) + ((0 + 0) + (0 + 0)))
```

Cela devient vite fatiguant de le faire à la main. Cet exercice le fait pour vous. Exemples d'exécution :

```
Un entier? 0
==> 0 = 0
```

```
Un entier? 2
==> 0 = ((0 + 0) + (0 + 0))
```

Quelle est la condition d'arrêt ?
Quelle est la récurrence ?

Solution simple
La condition d'arrêt est que `n` vaut zéro et la récurrence est :

1. Affichez une parenthèse ouvrante
2. Appelez récursivement avec `n-1`
3. Affichez un plus
4. Appelez récursivement avec `n-1`
5. Affichez une parenthèse fermante

Écrivez une procédure récursive `afficherTeteToto(n)` qui affiche `n` fois les zéros à droite de l'égalité « 0=0 » par leur valeur « (0+0) ».

Validez votre procédure avec la solution.

Solution alg @[pgtete.alg]

```
Action afficherTeteToto ( n : Entier )
Début
  |  Si ( n = 0 ) Alors
  |    |  Afficher ( "0" )
  |  Sinon
  |    |  Afficher ( "(" )
  |    |  afficherTeteToto ( n - 1 )
  |    |  Afficher ( "+" )
  |    |  afficherTeteToto ( n - 1 )
  |    |  Afficher ( ")" )
  |  FinSi
Fin
```

Écrivez un algorithme qui saisit un entier et lance la procédure.

Testez.

Validez votre algorithme avec la solution.

Solution alg @[pgtete.alg]

```
Algorithme pgtete
Variable n : Entier
Début
  |  Afficher ( "Un entier? " )
  |  Saisir ( n )
  |  Afficher ( "==> 0 = " )
  |  afficherTeteToto ( n )
Fin
```

4 Références générales

Comprend [Felea-PG1 :c3 ;ex85], [Franceioi-AL1 :c6 :ex1], [Franceioi-AL1 :c6 :ex4] ■