Les procédures

Variables locales et globales

- Les variables globales a et b sont initialisées respectivement aux valeurs 0 et 2
- a++ fera passer la valeur de a à 1
- b--, fera passer la valeur de b à 1
- b=f(a); va premièrement provoquer l'exécution de la fonction f avec la paramètre effectif a puis l'affectation de la valeur retournée par la fonction à la variable b.
- L'exécution de la fonction f: les instructions a=2, b=7; vont modifier les variables a et b, car celles-ci sont des variables globales. La valeur retournée par f sera 2 *1+7=9
- La valeur 9 est ensuite affectée à b
- Après l'exécution du programme a=2 et b=9.

Les procédures

Variables locales et globales

- Quelles sont les valeurs de a et b après l'exécution du programme suivant:

```
Fonction f (a : entier): entier
k← 1;
écrire (" k= ", k)
Retourne (k*a)
FinFonction
```

```
Algorithme Test_f
  variables k, b : entier
Début
  k ← 0;
  k++
  écrire (" k= ", k)

           k ← 2
            k++
            écrire (" k= ", k)

  k- -
  écrire (" k= ", k)
  k=f(k)
Fin
```

Récurrence et Récursivité

- **Suite récurrente**: la définition d'une suite est la donnée

  - D'un terme général défini en fonction du (ou des) terme(s) précédant(s)

  - D'un terme initial qui permet d'initialiser le calcul

- **Principe de récurrence :**

Soit **P** un prédicat (ou propriété) sur IN qui peut être soit vrai soit faux (on écrira souvent P(n) à la place de P(n) = vrai).

On suppose que

- **P**(0) vrai
- $\forall n \in IN$, **P**(n) $\Rightarrow$ **P**(n+1)

Alors , pour tout n $\in$IN, P(n) est vrai.

Si on considère le prédicat suivant

P(n) : je sais résoudre le problème pour n alors le principe de récurrence nous dit que si je sais résoudre le Pb pour n=0 et que si je sais exprimer la solution pour n en fonction de la solution pour n+1 alors je sais résoudre le Pb pour n'importe quel n.

Récurrence et Récursivité

- Examples:

1. **Puissance**

   $a^0 = 1$

   $a^{n+1} = a\,a^n$

   Ou bien

   $a^0 = 1$

   $a^n = a\,a^{n-1} \quad n>0$

2. **Factoriel**

   $0! = 1$

   $n! = n\,(n-1)! \quad , \quad n \geq 1$

3. **Suite de Fibonacci**

   $F_0 = F_1 = 1$

   $F_n = F_{n-1} + F_{n-2} \quad , \quad n \geq 2$

Récurrence et Récursivité

- Un algorithme (ou fonction) est dit récursif s'il est défini en fonction de lui-même.
  - Exemples
  - Fonction puissance(x : réel, n : entier) : réel

```
début
        si n = 0 alors retourner 1
        sinon retourner (x * puissance(x , n-1))
fin
```

  - Factoriel (n) début

```
        si n = 0 alors retourner(1)
        sinon retourner (n*factoriel(n-1))
fin
```

Récurrence et Récursivité

- *fact(n)*

  début

  si n= 0 alors

  retourner(1) sinon

  *retourner(n*fact(n-1)) fsi*

  fin

- Le déroulement de l'appel de fact(3):

  3*fact(2)

  Fact(3) 3*2=6 2*fact(1)

  6 2*1=2 1*fact(0)

  1*1=1

- La condition n = 0 est appelée test d'arrêt de la récursivité.
- Il est impératif de prévoir un test d'arrêt dans une fonction récursive, sinon l'exécution ne s'arrêtera jamais.
- L'appel récursif est traité comme n'importe appel de fonction.

Récurrence et Récursivité

L'appel d'une fonction (récursive ou non) se fait dans un **contexte d'exécution** propre (pile d'exécution), qui contient :

- L'adresse mémoire de l'instruction qui a appelé la fonction (adresse de retour)
- Les valeurs des paramètres et des variables locales à la fonction.

L'exécution d'une fonction récursive se fait par des appels successifs à la fonction jusqu'à ce que la condition d'arrêt soit vérifiée, et à chaque appel, les valeurs des paramètres et l'adresse de retour sont mis (empilés) dans la pile d'exécution.

Récurrence et Récursivité

- L'ordre des instructions par rapport à un appel récursif est important.
- Exemple:

afficher(n)

début

si n> 0 alors

١

afficher(n div 10)

٢

écrire(n mod 10)

fsi

fin

- *L*'algorithme récursif afficher(n) permet d'afficher les chiffres d'un entier, strictement positif, selon la disposition de l'instruction écrie(n mod 10):

-Si l'instruction est placée en ١ *, les chiffres sont affichés dans l'ordre inverse*
-Si elle est placée en ٢*, alors les chiffres seront affichés dans le bon ordre Pour n = 123, on a :*

١ $\rightarrow$ *3 2 1*

٢ $\rightarrow$ *1 2 3*

Type de Récursivité

- **<u>Récursivité simple</u>**: Une fonction récursive contient un seul appel récursif.
- **<u>Récursivité multiple</u>**: une fonction récursive contient plus d'un appel récursif (exemple suite de Fibonacci).
- **<u>Récursivité mutuelle</u>**(ou croisée): Consiste à écrire des fonctions qui s'appellent l'une l'autre.

  Exemple

Récursivité Mutuelle

```
Pair(n)
début
   si  n=  0 alors
         retourner vrai
   sinon
    retourner (impair(n-1))
   fsi
fin
```

```
Impair(n)
début
   si  n=  0 alors
         retourner (faux)
   sinon
      retourner (pair(n-1))
   fsi
fin
```

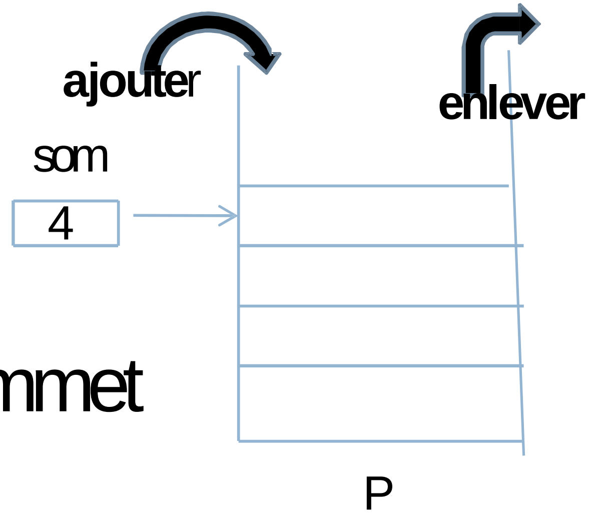
Notion de pile

- **Notion de pile**.

  Une pile est une structure pour représenter une suite d'éléments avec la contrainte qu'on ne peut ajouter, ou enlever, un élément que d'un même côté de la pile (dit sommet de la pile).

- Exemple pile d'assiettes.
- Une pile peut être représentée

  par un tableau et un indice de sommet



Notion de pile

- **<u>Opérations définies sur les piles:</u>**

  - **initialiser(p : Pile)** //Crée une pile vide.
  - **sommet(p : Pile) : élément**// Renvoie l'élément au sommet de la pile p, <u>sous la condition que p soit non vide.</u>
  - **empiler(x : élément, p : Pile)** / / ajoute x au sommet de la pile p.
  - **dépiler(p : Pile)** / / supprime l'élément au sommet de la pile p, <u>sous la condition que p soit non vide</u>.
  - **pileVide(p : Pile) : booléen** / / retourne vrai si p est vide.

Notion de pile

- **Exemple.**

  Une expression e est dite bien parenthésée (on se limite au '(' et ')') si :

1. Le nombre de parenthèses ouvrantes ($|e|_($) est égal au nombre de parenthèses fermantes ($|e|_)$) dans e.
2. Tout préfixe (partie gauche) u de e vérifie: $|u|_(- |u|_(\geq 0$.

Algorithme: on parcourt l'expression e (de gauche à droite). A chaque rencontre d'une parenthèse ouvrante on l'empile, et à chaque rencontre d'une parenthèse fermente on dépile.

Si on arrive à la fin de e avec une pile vide, l'expression e est bien parenthésée sinon e n'est pas bien parenthésée.

- l'expression (()())() est bien parenthée.
- l'expression ())(n'est pas bien parenthésée.

Transformation du récursif en itératif : « Dérécursivation »

- **Schéma d'un algorithme récursif**:

```
algoR(X)
début
        A
    si C(X) alors
        B;
        algoR(φ(X));
        D;
    sinon
        E;
    fsi;
fin
```

Où :
X : liste de paramètres
C : condition d'arrêt portant sur X
A, B, D, E : bloc d'instructions (éventuellement vide)
$\varphi(X)$: transformation des paramètres

Transformation du récursif en itératif : « Dérécursivation »

```
algoR(X)
Début
        A
   si C(X) alors
        B;
        algoR(φ(X));
        D ;
   sinon
        E;
   fsi;
fin
```

- **Algorithme itératif équivalent.**

```
algoI(X)
   p : Pile
début
   initialiser(p);
        A ;
   tantque C(X) faire
        B;
        empiler(X, p);
        X := φ (X);
        A ;
   ftantque;
        E;
   tantque (non pileVide(p)) faire
        X := sommet(p);
        dépiler(p);
        D ;
   ftantque
   fin
```

Transformation du récursif en itératif : « Dérécursivation »

- Exemple.

```
afficherR(n)
    début
        si  n > 0  alors
            afficher(n div 10)
            écrire(n mod10);
        fsi
    fin
```

```
afficherI(n)
p : Pile;
Début
    initialiser(p);
    tanque n > 0 faire
        empiler(n, p);
        n := n div 10;
    ftantque
    tantque (non pileVide(p)) faire
        n := sommet(p);
        dépiler(p);
        écrire(n mod 10);
    ftanque
fin
```

Transformation du récursif en itératif : « Dérécursivation »

Récursivité terminale:

- La récursivité d'une fonction **F**(X) est aussi dite terminale lorsqu'elle se termine par l'instruction retourner(**F**(φ(X))). On ajoute, dans ce cas, un paramètre à la liste X pour contenir le résultat de retour d'un appel récursive, comme le montre l'exemple suivant:

Exemple

fonction FACR(*n*) ;
début
si *n*=0 alors retourner (1)
sinon retourner (*n** FACR(*n*-1)) ;
fin.
RÉCURSIVE

fonction FACR'(*n, r*)
début
si *n*=0 **alors retourner** (*r*)
sinon retourner (FACR'(*n-1, n*r*)) ;
fin.
RÉCURSIVITÉ TERMINALE

fonction FACI(*n*) ;
début
r: = 1 ;
tant que *n* > 0 **faire**
{ *r* := *n* * *r* ; *n* := *n*-1 ; }
retourner (*r*) ;
fin.
ITÉRATIVE

fonction FACI'(*n, r*) ;
début
tant que *n* > 0 **faire**
{ *r* : = *n* * *r* ; *n* : = *n*-1 ; }
retourner (*r*) ;
fin.

Exercices

1. Ecrire une fonction récursive qui permet de calculer la suite suivante:

$$U_1 = 1$$
$$U_n = U_{n-1} + N \quad \text{tel que } N \geq 1$$

2. Ecrire une fonction booléenne rend une valeur vrai si un nombre "b" est diviseur d'un nombre "a" ou faux dans le cas contraire.

3. Ecrire une fonction récursive permettant de calculer la puissance pour x réel et n entier en utilisant la définition récursive suivante:

$$X^n = 1 \text{ pour } n = 0,$$

$$X^n = (X^{n/2})^2 \quad \text{pour n pair,}$$

$$X^n = X^{n-1} . X \quad \text{pour n impair.}$$