

INTRODUCTION À L'ALGORITHMIQUE II

CHAPITRE : POINTEURS - FONCTIONS

A.U: 2020 - 2021

Partie 1 : Pointeurs

Pointeur



- Un pointeur = Adresse

- Déclaration:

variable nom_point : ^ type

- Remarque:

Maison de superficie 60 m²

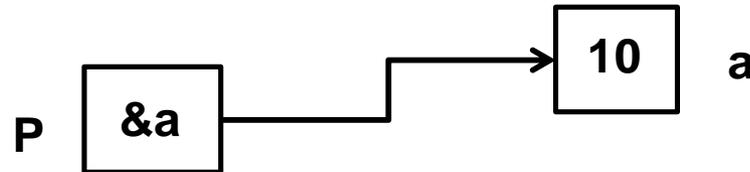


Maison de superficie 300 m²



Rappel: Pointeur

- Dessin: p pointe sur a



variable a,b : entier

variable p : ^ entier

- $p \leftarrow \&a$ // le pointeur p contient l'adresse de la variable a
- Pour accéder au contenu de la variable a via le pointeur p, utiliser la notation p^{\wedge}

Ex: $b \leftarrow p^{\wedge}$ /* équivalent à $b \leftarrow a$; b vaut 10 */

Rappel: Pointeurs et tableaux 1D

- Tout opération effectuée par indexation dans un tableau est réalisable avec des pointeurs.

- **Affectation:**

soient **tab** et **pt** respectivement un tableau d'Entier et pointeur vers type Entier,

`pt ← tab` ou `pt ← &tab[0]` /* pt pointe sur le 1^{er} élément de tab */

Remarque: le **nom d'un tableau** (sans indices à sa suite) est un considéré comme **pointeur constant** sur le début du tableau.

- **Différences entre un tableau et un pointeur:**

- un tableau alloue un espace à la variable (pointeur ne le fait pas)
- un tableau n'est pas une **lvalue** (alors que le pointeur l'est)

Rappel: Arithmétique des Pointeurs

- **Règle d'ajout d'un entier** : Si on additionne un entier à un pointeur pointant sur un tableau, alors le résultat est aussi un pointeur.

Exemple: si **p** est un pointeur vers **tab** un tableau d'Entier, alors:

- $p \leftarrow \text{tab}$ // p pointe sur le 1^{er} élément de tab
- $p+1$ pointe sur le 2^{ème} élément de tab
- valeur de p: adresse de $\text{tab}[0]$
- valeur de $p+1$: adresse de $\text{tab}[0] + \text{taille}(\text{Entier})$
- valeur de $p + i$: adresse de $\text{tab}[0] + i * \text{taille}(\text{Entier})$

- **Question:** et si p pointe vers un tableau T de type long ?
- **Réponse:** valeur de $p + i$: adresse de $T[0] + i * \text{taille}(\text{long})$

- **Correspondance:**

- $pt + i$ correspond à $\&\text{tab}[i]$
- $(pt + i)^\wedge$ ou $pt[i]$ correspond à $\text{tab}[i]$

- **Formule Générale:** $p + i$ correspond à $p + i * \text{taille}(p^\wedge)$

Rappel: Arithmétique des Pointeurs

- **Soustraction de pointeurs de même type**

Si p et q deux pointeurs sur le même type alors

$q - p$ fournit le **nombre d'éléments du type pointé** situés entre les deux adresses correspondantes.

– Exemple :

Variable $tab[5]$: Entier

$p, q : ^{\wedge}\text{Entier}$

$p \leftarrow tab$

$q \leftarrow \&tab[3]$

Alors

$q - p$ vaut 3

Rappel: Arithmétique des Pointeurs

- **Comparaison de pointeurs de meme type**

Soient p et q deux pointeurs sur un meme tableau tab ,

La comparaison $p < q$ est vraie si p pointe sur un élément de tab qui précède celui sur lequel pointe q

— Ex EP1:

```
variable tab[10] : Numérique
```

```
      p,q : ^ Numérique
```

```
Début
```

```
tab ← {1,2,3,4,5,6,7}
```

```
p ← tab
```

```
q ← &tab[5]
```

```
TantQue (p<q) faire
```

```
    Ecrire ("\np[", p-tab, "] : ", p^)
```

```
    p ← p+1
```

```
FinTantQue
```

```
Fin
```

Rappel: Arithmétique des Pointeurs

- L'exemple EP1 affiche:

```
p[0] : 1  
p[1] : 2  
p[2] : 3  
p[3] : 4  
p[4] : 5
```

Rappel: Arithmétique des Pointeurs

- **Exemple:** affichage de tableau par p sans utiliser indice

Constante NB 10

variable tab[NB] : Réel

p : ^Réel

Début

p ← tab

Pour i allant de 0 à NB-1

 Ecrire ("\nDonner tab[",i,"] : ")

 Lire p^

p ← p+1

FinPour

Ecrire ("\nles elements du tableau: ");

P ← tab

TantQue (p < tab + NB) faire

 Ecrire ("\ntab[", p-tab , "]: ", p^)

FinTantQue

Fin

Rappel: Arithmétique des Pointeurs

- **Pointeur null:** pointeur qui ne pointe sur aucune variable
- **$p \leftarrow \text{NULL}$ ou $p \leftarrow 0$**
- **Comparaison:**
si $(p \neq \text{NULL})$ ou si $(p \neq 0)$

Rappel: Pointeurs - Allocation mémoire

- Allocation dynamique de la mémoire :
 - Réalisée lors de l'exécution du programme
 - Réservée à un pointeur p via la fonction
 - `allocation(p) /* cas d'un élément */`
 - `allocation(p,nb_elem) /* cas de nb_elem éléments */`
- Libération de la mémoire
 - via la fonction `libère(p)`
 - Libère et restitue l'espace pointé par le pointeur p

Rappel: Pointeurs - Allocation mémoire

- **Attention:**

- La fonction **libère()** doit être **appliquée uniquement** à des pointeurs pointant sur une zone allouée dynamiquement (par la fonction **allocation**).
- La fonction libère() **ne libère pas** l'espace mémoire occupé par la variable pointeur.
- La fonction **libère()** rend indéterminée la valeur de p (p pointe vers n'importe quoi).

- **Recommandation:**

- Après l'utilisation de la fonction libère à un pointeur, il vaut mieux le rendre **NULL**.
- Ex:
libère(p)
P ← NULL

Partie 2 : Fonctions

Fonctions ?

- A-t-on utiliser des fonctions ?
 - Lire()
 - Ecrire()
 - mod(a,b)
 - div(n,e)

Fonctions: Exemple - (1/3)

Algorithme: calcul de la factorielle, de l'arrangement et la combinaison.

Variables A,C,F,n,k,i : Entier

Variables NF,NKF,KF : Entier

Ecrire("Calcul de la factorielle:")

Ecrire("Saisir un entier n:")

Lire(n)

F ← 1

Pour i allant de 1 à n

 F ← F * i

FinPour

Ecrire("La factorielle de ",n," : ",F)

Ecrire("-----")

Ecrire("Calcul de l'Arrangement:")

Ecrire("Saisir un entier n:")

Lire(n)

Ecrire("Saisir un entier k (k ≤ n): ")

Lire(k)

NF ← 1

Pour i allant de 1 à n

 NF ← NF * i

FinPour

NKF ← 1

Pour i allant de 1 à n-k

 NKF ← NKF * i

FinPour

A ← NF / NKF

Ecrire("L'arrangement de",k," à ",n")

Ecrire(" vaut :",A)

Ecrire("-----")

Ecrire("Calcul de la Combinaion:")

NF ← 1

Pour i allant de 1 à n

 NF ← NF * i

FinPour

Fonctions: Exemple - suite (2/3)

NKF \leftarrow 1

Pour i allant de 1 à n-k

 NKF \leftarrow NKF * i

FinPour

FK \leftarrow 1

Pour i allant de 1 à k

 FK \leftarrow FK * i

FinPour

C \leftarrow NF / (KF * NKF)

Ecrire("La combinaison de ",k," parmi ",n," vaut :",C)

Question: Et si on voulait commencer le calcul de la factorielle à partir du rang 2:

 FK \leftarrow 1

 Pour i allant de 2 à n

 FK \leftarrow FK * i

 FinPour

Fonctions: Exemple - suite (3/3)

Algorithme: calcul de la factorielle, de l'arrangement et la combinaison.

```
Fonction Fact(Entier : N) : Entier
```

```
Variable j,f : Entier
```

```
Début
```

```
f ← 1
```

```
Pour j allant de 1 à n
```

```
    f ← f * j
```

```
FinPour
```

```
Fin
```

```
variables F,A,C,n,k,i : Entier
```

```
Variables NF,NKF,KF : Entier
```

```
Ecrire("Calcul de la factorielle:")
```

```
Ecrire("Saisir un entier n:")
```

```
Lire(n)
```

```
Ecrire("Factorielle de ",n," : "
```

```
,Fact(n))
```

```
Ecrire("-----")
```

```
Ecrire("Calcul de l'Arrangement:")
```

```
Ecrire("Saisir un entier n:")
```

```
Lire(n)
```

```
Ecrire("Saisir un entier k (k ≤ n): ")
```

```
Lire(k)
```

```
A ← Fact(n) / Fact(n-k)
```

```
Ecrire("L'arrangement de",k," à ",n")
```

```
Ecrire(" vaut :",A)
```

```
Ecrire("-----")
```

```
Ecrire("Calcul de la Combinaion:")
```

```
C ← Fact(n) / (Fact(k) * Fact(n-k))
```

```
Ecrire("La combinaison de ",k," parmi  
",n," vaut :",C)
```

Fonctions

- Efficacité:
 - Lisibilité du code: éviter l'écriture réitérée du même code.
 - Éviter la correction répétitive de la même erreur (En cas de détection d'erreur syntaxique ou logique).
 - Réécriture et amélioration du programme.
 - Fonction = module

Fonctions

- Origine:
 - Esprit d'analyse descendante.
 - La réalisation d'un programme passe par l'analyse descendante du problème:
 - décomposer le problème donné en sous problèmes, et ainsi de suite, jusqu'à **descendre** au niveau des primitives (modules)
 - réussir à trouver les actions élémentaires qui, en partant d'un environnement initial, nous conduisent à l'état final.

Fonctions

- Exemple:

Combinaison k,n

Arrangement k,n

Factorielle k

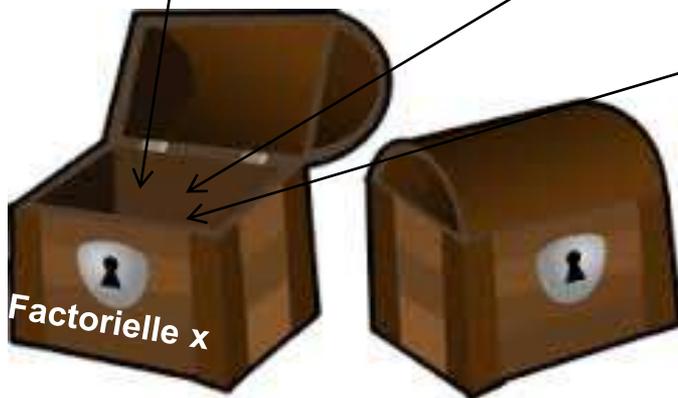
Factorielle n

Factorielle $n-k$

n

$n-k$

k



Fonctions

- Une fonction est connue dans le programme par **son nom**.
- il est nécessaire de connaître les éléments qui vont permettre de l'utiliser (paramètres).
- Comme pour les variables, elle peut être déclarée.

Fonctions: Définition

- La définition se fait au moyen de:
 - Entête de la fonction
 - Déclaration des variables
 - Corps de la fonction

De la manière suivante:

```
Entête  
déclarations des variables  
début  
  
...  
Corps de la fonction  
  
...  
fin
```

Fonctions: Définition

1. L'Entête (appelé aussi prototype ou signature) est écrit ainsi:

```
fonction NomFonction(NomArg_1:TypeArg_1 , ...,  
                    NomArg_k:TypeArg_k) : TypeRetour
```

et composé de:

- nom de la fonction (NomFonction), identificateur standard.
- liste des paramètres (ou arguments) d'appel: Optionnelle, Si la liste est vide, on peut ne rien écrire.
- Du type du résultat (TypeRetour), éventuellement vide. Si la fonction ne rend pas de résultat, on parle de **procédure**.
Remarque: Par précaution, lorsque la fonction ne renvoie rien, vaut mieux écrire dans le type de retour le mot "vide" que de ne rien écrire.

Fonctions: Définition

2. Déclarations des variables
3. Corps de la fonction : suite d'instructions, éventuellement une seule, mises entre les mots clé **début** et **fin**.

- Une fonction se définit par la construction:

```
fonction NomFonction(NomArg_1:TypeArg_1 ,...,  
    NomArg_k:TypeArg_k) : TypeRetour
```

```
déclarations des variables
```

```
début
```

```
Corps de la fonction ( Bloc d'instruction(s) )
```

```
fin
```

Fonctions

- Exemple : Factoriel

```
fonction factoriel( a : entier) : entier
```

```
variable f : entier
```

```
Début
```

```
f ← 1
```

```
Tant Que a > 1 faire
```

```
    f ← f * a
```

```
    a ← a-1
```

```
FinTantQue
```

```
retourne f
```

```
Fin
```

Fonctions

- Restriction (convention):
une fonction ne peut pas être définie à l'intérieure d'une autre fonction.
- Remarque: une fonction peut être utilisée (appelée) par une autre fonction:

```
fonction arrangement (k:entier , n:entier) : entier
variable res : entier
Debut
si k > n alors
    res ← 0
sinon
    res ← factoriel(n) / factoriel(n-k)
Finsi
retourne res
Fin
```

Fonctions: Paramètres (arguments)

- Arguments muets (formels):
 - paramètres apparaissant dans l'entete de la fonction lors de sa **définition**
 - Rôle: décrire leurs rôles au sein du corps de la fonction
- Arguments effectifs:
 - paramètres fournis lors de l'appel (utilisation) de la fonction
- Exemples:
 - Dans la définition de fonction factoriel, la variable a est un argument formel
 - Dans la définition de la fonction arrangement, les variables n et k sont des arguments formels
 - Dans la définition de la fonction arrangement, la variable n passée comme paramètre lors de l'appel de la fonction factoriel est un argument effectif.

Fonctions: Exemple complet - 1

- Calculer la somme des puissances p-ième des entiers $S_p(n)$:

$$S_p(n) = 1^p + 2^p + 3^p + \dots + n^p$$

- Ce calcul doit être effectué n fois pour des valeurs de p qui pourraient varier.

→ fabriquer un outil: une fonction **puissance(x,p)**, auquel l'on fournira deux entiers x et p et qui retournera le nombre x^p .

→ On sait que $x^p = x * x * \dots * x$ (p-1 multiplications)

Fonctions: Exemple complet - 2

- D'où la fonction puissance() définie de la manière suivante :
fonction puissance(x : entier, p :entier) : entier
variables z : entier
Début
z ← 1
tant que (p>0) faire
 z ← z*x
 p ← p-1
fintantque
retourne(z)
fin

Fonctions: Exemple complet - 3

- La dernière instruction, retourne(z) est le moyen d'indiquer que le résultat de la fonction est z.
- On peut maintenant écrire le programme qui fait la somme des puissances $P^{\text{ème}}$ des entiers compris entre 1 et n (calcul de $S_p(n)$)

/* la fonction qui calcule x^p */

fonction puissance(x : entier, p :entier) : entier

variables z : entier

Début

z ← 1

tant que (p>0) faire

 z ← z*x

 p ← p-1

fintantque

retourne(z)

fin

Fonctions: Exemple complet - 4

```
fonction menu ()
variables p, n, s, i : entiers
Début
s ← 0
écrire(" entrer la puissance p et l'entier n :")
lire(p,n)
Pour i allant de 1 à n faire
s ← s + puissance(i,p)

/* les variables i et p sont des paramètres réels, la fonction
   puissance(i,p)est appelée pour effectuer le calcul de  $i^p$  */

FinPour
écrire(" La somme est : ",s)
Fin
```

Fonctions

- Exécution

```
entrer la puissance p et l'entier n : 1 10
```

```
La somme est : 55
```

```
=====
```

```
entrer la puissance p et l'entier n : 2 10
```

```
La somme est : 1155
```

```
=====
```

```
entrer la puissance p et l'entier n : 3 10
```

```
La somme est : 3025
```

Fonctions: Prototype (1/3)

- La fonction peut être mise n'importe où dans le programme MAIS,
- Il faut vérifier la correspondance entre paramètres formels et paramètres réels :
 - Même nombre de paramètres
 - Même types respectifs (deux à deux)

→ R1 : il faut que la fonction soit placée **avant son utilisation**.

→ **Prototype :**

- Pour éviter la restriction R1, on peut utiliser un prototype de fonction.
- Permet aux fonctions de s'appeler mutuellement
- Un prototype est constitué de la partie déclaration de la fonction,
- Est placé en général au début du programme.

Fonctions : Prototype (2/3)

- On pourra écrire, par exemple :

```
/* prototypes */  
fonction f(entier,entier):entier  
fonction g(chaine de caractère, entier) ;
```

```
/* corps de la fonction f */  
fonction f(x:entier, y:entier)  
var message : chaine de caractère  
début  
... ..  
g(message,y)  
... ..  
Fin
```

Fonctions : Prototype (3/3)

```
/* corps de la fonction g */  
fonction g(m : chaîne de caractère, y : entier)  
var a,b,c : entier  
début  
... ..  
c ← f(a,b)  
... ..  
Fin
```

- L'utilisation des prototypes n'est pas obligatoire **MAIS**:
 - fortement recommandée
 - meilleur contrôle des paramètres
 - meilleure lisibilité du programme
 - les outils utilisés sont listés au début du texte.

Fonctions: résultat d'une fonction

- Utilisation de l'instruction, retourne(val)
- **retourner** (produire ou rendre) le résultat d'une fonction,
- val doit être du type prévu dans l'entête de la fonction.

- Soit max() une fonction qui **rend** le plus grand des deux nombres entiers x et y, elle peut s'écrire :

```
fonction max ( x : entier, y : entier) : entier
/* maximum de x et y */
si (x < y) retourne(y)
sinon retourne(x)
Finsi
Fin
```

Fonctions: résultat d'une fonction

- L'instruction retourne provoque:
 - la **sortie** immédiate de la fonction ;
 - le retour dans le programme appelant;
 - **les instructions qui suivent dans la fonction ne sont pas exécutées,**
- Dans l'exemple précédent, on peut ne pas mettre "sinon".

```
fonction max ( x : entier, y : entier) : entier
/* maximum de x et y */
si (x<y)
    retourne(y)
Finsi
retourne(x)
Fin
```

Fonctions: résultat d'une fonction

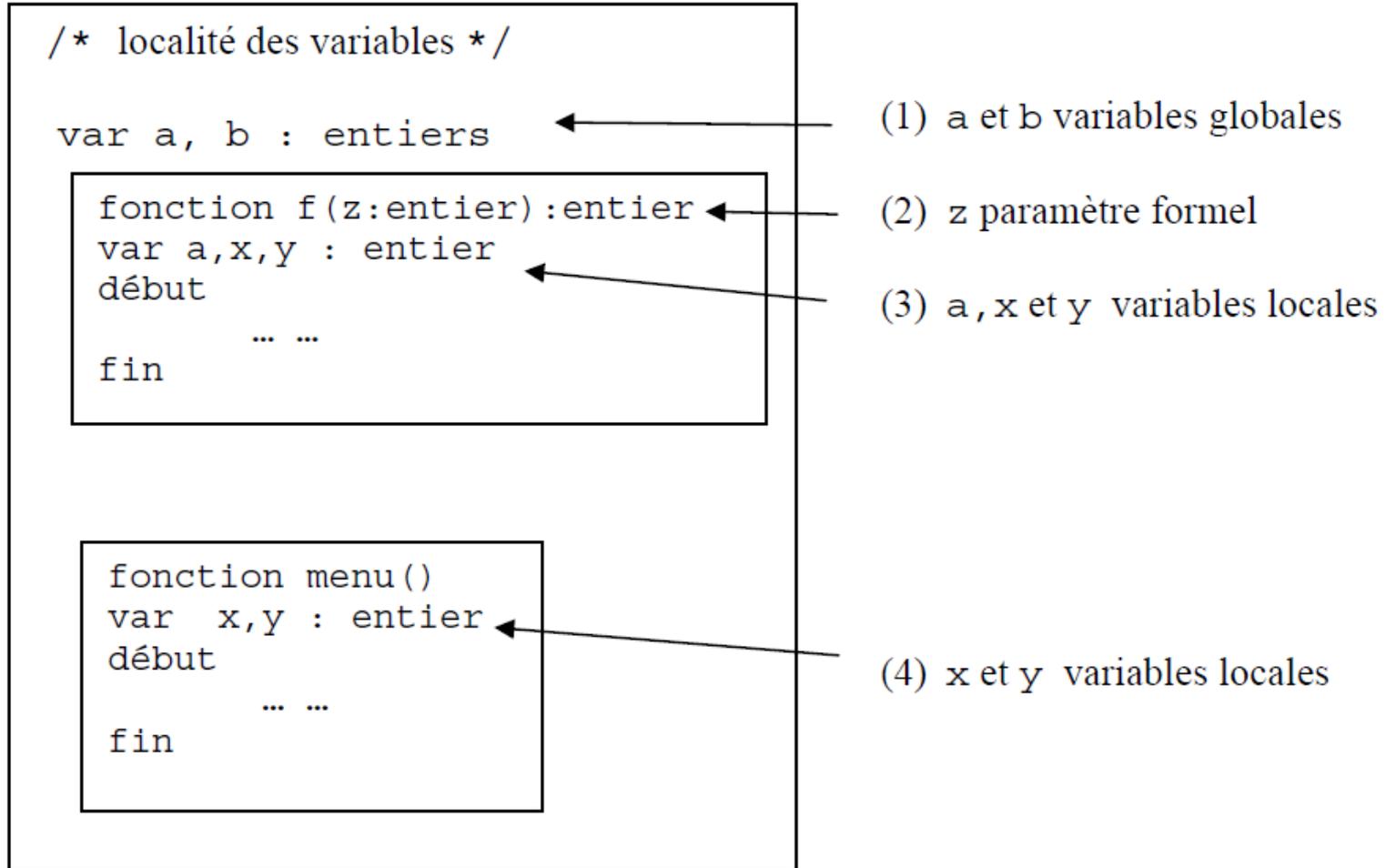
- Exemple de test de nombre premier:

```
fonction Premier ( m : entier) : Boolean
Variables i : Entier
Début
i ← 2
Tantque (i <= sqrt(m))
    si (mod(m,i) = 0)
        Ecrire(m , "n'est pas un nombre premier.")
        retourne(0)
    Finsi
    i ← i + 1
FinTantque
Ecrire(m , "est un nombre premier.")
retourne(1)
Fin
```

Fonctions: variables locales et variables globales

- Il peut y avoir différents niveaux de déclaration de variables.
 - Les variables définies dans le programme hors des fonctions sont dites variables **globales**.
les variables globales sont connues de **tout le programme**.
 - Les variables déclarées dans le corps d'une fonction sont dites variables **locales** ou variables **automatiques**:
 - Connues **uniquement à l'intérieur de la fonction**,
 - Propres à la fonction et invisibles de l'extérieur.
 - Appelées aussi **automatiques** car elles sont **automatiquement détruites** après l'exécution de la fonction et leurs espaces mémoire libérés.
 - Possibilité d'utiliser des variables de même nom dans des fonctions différentes sans craintes d'interférences déplaisantes .

Fonctions: variables locales et variables globales



Fonctions: variables locales et variables globales

- Les variables a et b, (1) sont des variables globales: elles seront connues et utilisables dans toutes les parties du programme.
- La fonction menu() ne peut que:
 - travailler sur les variables globales a et b
 - appeler la fonction f()
- Une fonction connaît:
 - les paramètres formels,
 - ses propres variables,
 - toutes les fonctions définies avant elle
 - **elle se connaît elle-même (récursivité).**
 - les variables globales dont le nom n'est pas redéfini dans la fonction,
→ Conflit locale - globale: La variable locale l'emporte (Ex)

Fonctions: variables locales et variables globales

- La fonction `f()` pourra travailler avec les variables locales `a`, `x`, `y`, `z` et la variable globale `b`.
- La variable globale `a` est inaccessible de l'intérieur de `f()`, toute utilisation de `a` fait référence à la variable locale.
- La fonction `f()` pourra s'appeler elle-même.
- La fonction `menu()` pourra travailler sur les variables locales `x`, `y` et les variables globales `a` et `b`
- la fonction `menu()` pourra appeler `f()`.
- la fonction `menu()` est souvent appelée **fonction principale**

Fonctions: paramètres d'appels

- Passage par valeur :
 - le paramètre d'appel est considéré comme une variable locale,
 - toutes les modifications se feront dans une variable (copie) temporaire dans laquelle est rangée la valeur du paramètre d'appel.
 - On peut appeler la fonction avec comme paramètre, une expression qui sera évaluée et dont la valeur sera le paramètre réel.

Ex : $F(a = b)$

Fonctions: Passage par valeur

- Exemple : Soit le programme suivant

```
fonction menu ()  
var a,b,c : entier          /* déclaration de a et b*/  
début  
    écrire("entrer les valeurs de a et b :")  
    lire(a,b)  
    écrire(" avant l'échange a = ",a," et b = ",b)  
    c ← a  
    a ← b                    /* l'échange */  
    b ← c  
    écrire(" après l'échange a = ",a," et b = ",b)  
Fin
```

- Execution :

entrer les valeurs de a et b : 3 8

avant l'échange a=3 et b=8

après l'échange a=8 et b=3

Fonctions: Passage par valeur

- On met l'exemple sous forme de fonction

```
fonction echange(x:entier, y:entier) /* x,y : paramètres formels */  
var tmp :entier /* tmp variable locale */  
début  
tmp ← x  
x ← y          /* l'échange */  
y ← tmp  
écrire("dans la fonction echange(): x = ",x," et y = ",y)  
fin
```

Fonctions: Passage par valeur

- On définit une fonction menu() pour faire appel à echange()

```
fonction menu()
```

```
var a,b : entier /* déclaration de a et b*/
```

```
début
```

```
écrire("entrer les valeurs de a et b :")
```

```
lire(a,b)
```

```
écrire(" avant l'échange a = ",a," et b = ",b)
```

```
echange(a,b) /* appel de echange() */
```

```
écrire(" après l'échange a = ",a," et b = ",b)
```

```
Fin
```

- Exécution : entrer les valeurs de a et b : 5 7

avant l'échange a=5 et b=7

dans la fonction echange() : x=7 et y=5

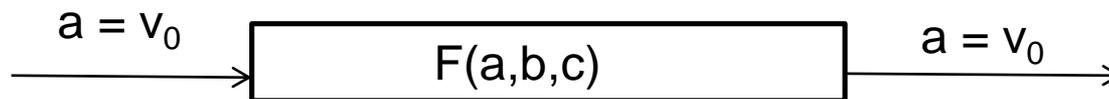
après l'échange a=5 et b=7

L'ÉCHANGE N'A PAS EU LIEU !!

Fonctions: Passage par valeur

- Conclusion:

Lors d'un passage par valeur d'un paramètre, on travaille avec une copie temporaire du paramètre (variable locale) et non avec le paramètre lui même.



Fonctions: Passage par Reference de paramètre

- Au cas où on veut modifier la valeur d'un paramètre à l'intérieur d'une fonction.
- La modification est faite sur la variable elle-même et non pas sur sa copie.
- La valeur de la variable change éventuellement après l'appel à la fonction.
- Le seul moyen de modifier une variable externe à une fonction est indirect : on transmet sa référence (ou adresse).
- L'argument de la fonction doit être une référence de la variable
→ d'où la notion de **pointeur**
- Pour indiquer la référence d'une variable, on la précède par le caractère & lors de l'appel de la fonction



Fonctions: Passage par Reference

- Exemple:

```
fonction echange( px: ^ entier, py: ^ entier)
var tmp :entier
début
tmp ← px^
px^ ← py^          /* l'échange */
py^ ← tmp
écrire("dans la fonction échange(): x = ", px^," et y =
      ",py^)
fin
```

Fonctions: Passage par Reference

- Exemple:

```
fonction menu()
```

```
var a,b : entier /* déclaration de a et b*/
```

```
Début
```

```
  écrire("entrer les valeurs de a et b :")
```

```
  lire(a,b)
```

```
  écrire(" avant l'échange a = ",a," et b = ",b)
```

```
  echange(&a,&b) /* on appel echange() avec les adresses  
                de a et b */
```

```
  écrire(" après l'échange a = ",a," et b = ",b)
```

```
Fin
```

- **Exécution:**

entrer les valeurs de a et b : 5 7

avant l'échange a=5 et b=7

dans la fonction echange() : x=7 et y=5

après l'échange a=7 et b=5

L'ÉCHANGE A EU LIEU !!

Fonctions: Passage par Reference

- Commentaires:
 - Dans la fonction `echange()` :
 - les deux paramètres formels sont des pointeur sur des entiers (des adresses).
 - le contenu(x) est la valeur qui se trouve à l'adresse x (puisque x est une adresse).
 - l'instruction $\text{tmp} \leftarrow x^{\wedge}$ signifie mettre dans la variable `tmp` (de type entier) la valeur de la variable qui se trouve à l'adresse x .
 - Dans la fonction `menu()` :
 - l'instruction `echange(&a,&b)` signifie qu'on appelle la fonction `echange()` en lui transmettant les adresses des variables x et y .

Fonctions: Passage par Reference

- Conclusion:

si la modification des paramètres est souhaitée → passage par reference

- **Remarque Finale: Le passage par reference d'un paramètre dans une fonction est un passage par valeur de sa reference (adresse - pointeur)**

Fonctions Prédéfinies

Fonctions Prédéfinies: Texte

- **Len(chaine)** : renvoie le nombre de caractères d'une chaine
- **Mid(chaine,pos,lg)** : renvoie un extrait de la chaine, commençant au caractère **pos** et faisant **lg** caractères de long. (commence à 0)
- **Left(chaine,n)** : renvoie les n caractères les plus à gauche dans chaine.
- **Right(chaine,n)** : renvoie les n caractères les plus à droite dans chaine
- **Find(chaine1,chaine2)** :
 - Renvoie la position de chaine2 dans chaine1,
 - Commence par 0,
 - Si chaine2 n'est pas comprise dans chaine1, la fonction renvoie -1

Fonctions Prédéfinies: Texte

- **Exemples:**

Len("Bonjour")	vaut	7
Len("")	vaut	0
Mid("Ahmed is back", 3, 7)	vaut	"ed is b"
Mid("Ahmed is back", 11, 1)	vaut	"c"
Left("Et pourtant...", 8)	vaut	"Et pourt"
Right("Et pourtant...", 4)	vaut	"t..."
Find("Un pur bonheur", "pur")	vaut	3
Find("Un pur bonheur", "vie")	vaut	-1

Fonctions Prédéfinies: numériques

- **Partie Entière: Ent()**

Fonction qui permet de récupérer la partie entière d'un nombre :

$A \leftarrow \text{Ent}(5,28)$ A vaut 5

- **Modulo: Mod()**

– Cette fonction permet de récupérer le reste de la division d'un nombre par un deuxième nombre. Par exemple :

– $A \leftarrow \text{Mod}(17,3)$ A vaut 2 car $17 = 5*3 + 2$

$B \leftarrow \text{Mod}(12,2)$ B vaut 0 car $12 = 6*2$

Fonctions Prédéfinies: numériques

- **Génération de nombres aléatoires**

Toto \leftarrow rand()

On a : $0 \leq \text{Toto} < 1$

- Par exemple, pour générer un nombre entre 2,25 et 2,65:

l'intervalle mesure 0,40 de large;

Donc : $0 \leq \text{rand}() * 0,40 < 0,40$

Il suffit dès lors d'ajouter 2,25 pour obtenir l'intervalle voulu:

Toto \leftarrow rand()*0,40 + 2,25

Fonctions Prédéfinies

- conversion
 - Numérique vers caractère: `convnc()`
A ← `convnc(4)` résultat A vaut '4'
 - caractère vers Numérique : `convcn()`
N ← `convcn('9')` : résultat N vaut le chiffre 9
- Tableaux dynamiques: `Redim()`
 - **Redim Tab(nb)** : redimensionne le tableau Tab en (nb + 1) éléments
 - **Attention:** nb est la position du dernier élément

Fonctions Prédéfinies

- Examens (et controle) :

Sauf indication, vous **avez le droit** d'utiliser les fonctions prédéfinies.