=

Fonctions et procédures

Idée : capitaliser sur le code déjà écrit pour introduire des nouvelles "commandes" et "opérations".

On souhaite donner un nom programme que nous avons déjà écrit, et qui effectue une opération intéressante, pour pouvoir le réutiliser après sans le réécrire. Si ce programme retourne un résultat, on parlera de "fonction", et "procédure" sinon.

De plus, on peut souhaiter choisir les valeurs de certaines variables de la procédure ou fonction, et cela se fait à travers le mechanisme de passage des paramètres.

Procedures: déclaration et appel

Chaque procédure aura:

1. Une *définition* qui dit
   (a) Le type des paramètres.
   (b) comment on la calcule.
2. Un ou plusieurs *appels*: c'est l'utilisation de la procédure.

Exemples (sans paramètres)

Tous les programmes affichent un message d'aide à l'utilisation, et cela à plusieurs occasions

```
procedure  aide ()
  debut aide
        ecrire "Aide du programme Machin."
        ecrire " l'option -o permet de definir le fichier sortie"
        ecrire " l'option -h donne ce message"
  fin aide
...
programme Machin
debut Machin
```



```
        ...
        aide()
        ...
        si (erreur=vrai) alors aide() fin si
        ...
fin Machin
```

Exemples (avec paramètres)

Tous les programmes affichent des messages d'erreur

```
procedure  erreur (valeur s: chaîne de caractères)
  debut erreur
        ecrire "Programme Machin: on a renconter l'erreur" s
  fin erreur
...
programme Machin
debut Machin
        ...
        erreur("Entier trop grand")
        ...
        erreur("Pas assez de memoire")
        ...
fin Machin
```

Fonctions: déclaration et appel

Chaque fonction aura:

1. Une *définition* qui dit
   (a) Le type des paramètres.
   (b) Le type de la valeur rendue.
   (c) comment on la calcule.
2. Un ou plusieurs *appels*: c'est l'utilisation de la fonction.
3. Zero, un ou plusieurs *paramètres*: ce sont les arguments de la fonction.
4. Un *type* et une *valeur*.

Exemples (sans paramètres)

```
fonction pi(): reel
  debut pi
      retourner 3.1415926535897931
  fin pi
```

Exemple avec paramètres

La fonction suivante rend le maximum de deux valeurs de type réél.

```
fonction fmax(valeur a: reel, valeur b: reel) : reel
variables m : un reel
debut fmax
 si (a > b)
 alors
   m <- a
 sinon
   m <- b
 fin si
 retourner m
fin fmax
```

de sorte que l'expression `fmax(pi()*pi(),10.0)` vaut 10.

Paramètres

En général, une fonction ou procédure n'est utile que si elle a des paramètres, comme dans le cas de `fmax`.

Une fonction ou procédure définie avec des paramètres doit être appelée avec des *arguments* (en nombre égal, et du même type!).

définition de fonction	appel de fonction
paramètre	argument


```
fonction fmax(valeur a: reel, valeur b: reel) : reel
...
programme essai
variables x: un reel
          i,j: deux caracteres
debut essai
   lire x
   ecrire fmax(x,x*x)
   /* ecrire fmax(i,j) serait incorrect! */
fin essai
```

Nature des paramètres: déclaration

Dans la declaration

```
fonction fmax(valeur a: reel, valeur b: reel) : reel
...
```

- `a` et `b` sont les paramètres de la fonction
- ils sont déclarés avec leur type
- dans le corps de la fonction, on peut utiliser `a` et `b` comme des variables,
  qui existent exclusivement dans le corps de la fonction
- le mot clé `valeur` indique une *modalité* de passage des paramètres

Nature des paramètres: passage par valeur

Après la déclaration

```
fonction fmax(valeur a: reel, valeur b: reel) : reel
...
```

on peut effectuer un appel de la fonction comme

```
ecrire 3*fmax(x,y)
```

l'expression `fmax(x,y)` est évaluée de la façon suivante:

- les valeurs de `x` et `y` sont calculées

- les paramètres `a` et `b` de `fmax` sont initialisés en utilisant ces valuers
- le corps de la fonction `fmax` est executé, (cela utilise les paramètres `a` et `b` comme des variables)
- le résultat de la fonction est enfin retourné et utilisé pour continuer le calcul de l'expression (ici, on le multiplie par 3 et on l'imprime)

Peu importe les opérations effectuées sur `a` *et* `b` *dans le corps de* `fmax`, *les valeurs des paramètres actuels* `x` *et* `y` *restent inchangés.*

Limite du passage par valeur

On a souvent besoin d'écrire des procédure ou fonctions que, au contraire, *modifient* la valeur de certains paramètres, par exemple, on voudrait écrire une procedure `echange` t.q. `echange(x,y)` échange les valeurs des variables `x` et `y`.

Mais la procédure

```
procédure echange (valeur a: entier, valeur b: entier)
variable t: un entier
debut echange
 t <- a
 a <- y
 b <- t
fin echange
```

n'échange pas les valeurs de ses arguments!
(relisez la description du passage d'arguments par valeur plus haut)

Comment faire?

Passage par référence

Si on écrit

```
procédure echange (reference a: entier, reference b: entier)
variable t: un entier
debut echange
 t <- a
 a <- y
 b <- t
fin echange
```


le passage des paramètres se fait autrement: au moment de l'appel

`echange(i,j)`

- *on ne fait pas une copie des valeurs de i et j dans a et b!*
- au contraire, on transforme a et b en "alias" de i et j, en le faisant pointer sur les memes cases mémoire de i et j
- donc, toute opération effectuée sur les paramètres a et b dans le corps de la procédure ou fonction et aussi effectué sur les arguments i et j

N.B.: on doit passer comme argument d'un paramètre par référence seulement des expressions qui se comportent comme des variables (par exemple, une variable, mais aussi une case d'un tableau etc.)

Résumé

- On déclare "valeur" un paramètre si on ne veut pas que l'appel modifie les arguments
- On déclare "reference" un paramètre si on veut que l'appel modifie les arguments

Bien sûr, on peut melanger paramètres par valeur et par référence.

ATTENTION:

- les tableaux sont *toujours* passé par référence, et *jamais* copiés, même si on déclare le paramètres correspondant "valeur".
(Vous comprendrez les raisons de cela plus avant dans le cursus informatique).

Un exemple complet: le tri par séléction

Voyons comme écrire un programme de tri par séléction beaucoup plus lisible avec les fonctions et les procédures.

Fonction de recherche du minimum à partir d'un indice

```
fonction mintab(valeur i: un entier, reference a: tableau): entier
variable k: un entier
debut mintab
   pour k <- i+1 a  N faire
    si (a[k] < a[i]) alors
        i <- k
    fin si
   fin pour
   retourner i
fin mintab
```

Programme de tri par séléction

```
constante N=10
type tab = tableau de N entiers
procedure echange(reference a, reference b) ...
fonction mintab(valeur i: un entier, reference a: tableau): entier ...

programme triselection
variables a: tab
debut triselection
lire a
  pour i <- 1 a N-1 faire
      echange(a[i],a[mintab(i,a)])
  fin pour
ecrire a
fin triselection
```


Et en C++?

```
#include <iostream.h>

const int N=10;

typedef int tab[N];

void echange (int &a, int &b)
{ int t;
  t=a; a=b; b=t;
}

int mintab(int i, tab &a)
{ int k;

  for(k=i+1;k<N;k++){if (a[k] < a[i]) {i =k;};};
  return(i);
}

void lire(tab &a)
{ int k;

  for (k=0;k<N;k++){cout<<"a["<<k<<"] ? "<<endl; cin>>a[k];};
}

void ecrire(tab &a)
{ int k;
 for (k=0;k<N;k++){cout<<"a["<<k<<"] = "<<a[k]<<endl;};
}

void main(){
 tab a;
 int i;

 lire(a);
 for (i=0;i<N-1;i++){echange(a[i],a[mintab(i,a)]);};
 ecrire(a);
}
```

=

Fonctions et procédures

Idée : capitaliser sur le code déjà écrit pour introduire des nouvelles "commandes" et "opérations".

On souhaite donner un nom programme que nous avons déjà écrit, et qui effectue une opération intéressante, pour pouvoir le réutiliser après sans le réécrire. Si ce programme retourne un résultat, on parlera de "fonction", et "procédure" sinon.

De plus, on peut souhaiter choisir les valeurs de certaines variables de la procédure ou fonction, et cela se fait à travers le mechanisme de passage des paramètres.

Procedures: déclaration et appel

Chaque procédure aura:

1. Une *définition* qui dit
   (a) Le type des paramètres.
   (b) comment on la calcule.
2. Un ou plusieurs *appels*: c'est l'utilisation de la procédure.

Exemples (sans paramètres)

Tous les programmes affichent un message d'aide à l'utilisation, et cela à plusieurs occasions

```
procedure  aide ()
  debut aide
        ecrire "Aide du programme Machin."
        ecrire " l'option -o permet de definir le fichier sortie"
        ecrire " l'option -h donne ce message"
  fin aide
...
programme Machin
debut Machin
```



```
        ...
        aide()
        ...
        si (erreur=vrai) alors aide() fin si
        ...
fin Machin
```

Exemples (avec paramètres)

Tous les programmes affichent des messages d'erreur

```
procedure  erreur (valeur s: chaîne de caractères)
  debut erreur
        ecrire "Programme Machin: on a renconter l'erreur" s
  fin erreur
...
programme Machin
debut Machin
        ...
        erreur("Entier trop grand")
        ...
        erreur("Pas assez de memoire")
        ...
fin Machin
```

Fonctions: déclaration et appel

Chaque fonction aura:

1. Une *définition* qui dit
   (a) Le type des paramètres.
   (b) Le type de la valeur rendue.
   (c) comment on la calcule.
2. Un ou plusieurs *appels*: c'est l'utilisation de la fonction.
3. Zero, un ou plusieurs *paramètres*: ce sont les arguments de la fonction.
4. Un *type* et une *valeur*.

Exemples (sans paramètres)

```
fonction pi(): reel
  debut pi
      retourner 3.1415926535897931
  fin pi
```

Exemple avec paramètres

La fonction suivante rend le maximum de deux valeurs de type réél.

```
fonction fmax(valeur a: reel, valeur b: reel) : reel
variables m : un reel
debut fmax
 si (a > b)
 alors
   m <- a
 sinon
   m <- b
 fin si
 retourner m
fin fmax
```

de sorte que l'expression `fmax(pi()*pi(),10.0)` vaut 10.

Paramètres

En général, une fonction ou procédure n'est utile que si elle a des paramètres, comme dans le cas de `fmax`.

Une fonction ou procédure définie avec des paramètres doit être appelée avec des *arguments* (en nombre égal, et du même type!).

définition de fonction	appel de fonction
paramètre	argument


```
fonction fmax(valeur a: reel, valeur b: reel) : reel
...
programme essai
variables x: un reel
          i,j: deux caracteres
debut essai
   lire x
   ecrire fmax(x,x*x)
   /* ecrire fmax(i,j) serait incorrect! */
fin essai
```

Nature des paramètres: déclaration

Dans la declaration

```
fonction fmax(valeur a: reel, valeur b: reel) : reel
...
```

- `a` et `b` sont les paramètres de la fonction
- ils sont déclarés avec leur type
- dans le corps de la fonction, on peut utiliser `a` et `b` comme des variables, *qui existent exclusivement dans le corps de la fonction*
- le mot clé `valeur` indique une *modalité* de passage des paramètres

Nature des paramètres: passage par valeur

Après la déclaration

```
fonction fmax(valeur a: reel, valeur b: reel) : reel
...
```

on peut effectuer un appel de la fonction comme

```
ecrire 3*fmax(x,y)
```

l'expression `fmax(x,y)` est évaluée de la façon suivante:

- les valeurs de `x` et `y` sont calculées

- les paramètres `a` et `b` de `fmax` sont initialisés en utilisant ces valuers
- le corps de la fonction `fmax` est executé, (cela utilise les paramètres `a` et `b` comme des variables)
- le résultat de la fonction est enfin retourné et utilisé pour continuer le calcul de l'expression (ici, on le multiplie par 3 et on l'imprime)

Peu importe les opérations effectuées sur `a` *et* `b` *dans le corps de* `fmax`, *les valeurs des paramètres actuels* `x` *et* `y` *restent inchangés.*

Limite du passage par valeur

On a souvent besoin d'écrire des procédure ou fonctions que, au contraire, *modifient* la valeur de certains paramètres, par exemple, on voudrait écrire une procedure `echange` t.q. `echange(x,y)` échange les valeurs des variables `x` et `y`.

Mais la procédure

```
procédure echange (valeur a: entier, valeur b: entier)
variable t: un entier
debut echange
 t <- a
 a <- y
 b <- t
fin echange
```

n'échange pas les valeurs de ses arguments!
(relisez la description du passage d'arguments par valeur plus haut)

Comment faire?

Passage par référence

Si on écrit

```
procédure echange (reference a: entier, reference b: entier)
variable t: un entier
debut echange
 t <- a
 a <- y
 b <- t
fin echange
```


le passage des paramètres se fait autrement: au moment de l'appel

`echange(i,j)`

- *on ne fait pas une copie des valeurs de i et j dans a et b!*
- au contraire, on transforme a et b en "alias" de i et j, en le faisant pointer sur les memes cases mémoire de i et j
- donc, toute opération effectuée sur les paramètres a et b dans le corps de la procédure ou fonction et aussi effectué sur les arguments i et j

N.B.: on doit passer comme argument d'un paramètre par référence seulement des expressions qui se comportent comme des variables (par exemple, une variable, mais aussi une case d'un tableau etc.)

Résumé

- On déclare "valeur" un paramètre si on ne veut pas que l'appel modifie les arguments
- On déclare "reference" un paramètre si on veut que l'appel modifie les arguments

Bien sûr, on peut melanger paramètres par valeur et par référence.

ATTENTION:

- les tableaux sont *toujours* passé par référence, et *jamais* copiés, même si on déclare le paramètres correspondant "valeur".
(Vous comprendrez les raisons de cela plus avant dans le cursus informatique).

Un exemple complet: le tri par séléction

Voyons comme écrire un programme de tri par séléction beaucoup plus lisible avec les fonctions et les procédures.

Fonction de recherche du minimum à partir d'un indice

```
fonction mintab(valeur i: un entier, reference a: tableau): entier
variable k: un entier
debut mintab
   pour k <- i+1 a  N faire
    si  (a[k] < a[i]) alors
        i <- k
    fin si
   fin pour
   retourner i
fin mintab
```

Programme de tri par séléction

```
constante N=10
type tab = tableau de N entiers
procedure echange(reference a, reference b) ...
fonction mintab(valeur i: un entier, reference a: tableau): entier ...

programme triselection
variables a: tab
debut triselection
lire a
  pour i <- 1 a N-1 faire
      echange(a[i],a[mintab(i,a)])
  fin pour
ecrire a
fin triselection
```


Et en C++?

```
#include <iostream.h>

const int N=10;

typedef int tab[N];

void echange (int &a, int &b)
{ int t;
  t=a; a=b; b=t;
}

int mintab(int i, tab &a)
{ int k;

  for(k=i+1;k<N;k++){if  (a[k] < a[i]) {i =k;};};
  return(i);
}

void lire(tab &a)
{ int k;

  for (k=0;k<N;k++){cout<<"a["<<k<<"] ? "<<endl; cin>>a[k];};
}

void ecrire(tab &a)
{ int k;
 for (k=0;k<N;k++){cout<<"a["<<k<<"] = "<<a[k]<<endl;};
}

void main(){
 tab a;
 int i;

 lire(a);
 for (i=0;i<N-1;i++){echange(a[i],a[mintab(i,a)]);};
 ecrire(a);
}
```

=

Fonctions et procédures

Idée : capitaliser sur le code déjà écrit pour introduire des nouvelles "commandes" et "opérations".

On souhaite donner un nom programme que nous avons déjà écrit, et qui effectue une opération intéressante, pour pouvoir le réutiliser après sans le réécrire. Si ce programme retourne un résultat, on parlera de "fonction", et "procédure" sinon.

De plus, on peut souhaiter choisir les valeurs de certaines variables de la procédure ou fonction, et cela se fait à travers le mechanisme de passage des paramètres.

Procedures: déclaration et appel

Chaque procédure aura:

1. Une *définition* qui dit
   (a) Le type des paramètres.
   (b) comment on la calcule.
2. Un ou plusieurs *appels*: c'est l'utilisation de la procédure.

Exemples (sans paramètres)

Tous les programmes affichent un message d'aide à l'utilisation, et cela à plusieurs occasions

```
procedure  aide ()
  debut aide
        ecrire "Aide du programme Machin."
        ecrire " l'option -o permet de definir le fichier sortie"
        ecrire " l'option -h donne ce message"
  fin aide
...
programme Machin
debut Machin
```



```
        ...
        aide()
        ...
        si (erreur=vrai) alors aide() fin si
        ...
fin Machin
```

Exemples (avec paramètres)

Tous les programmes affichent des messages d'erreur

```
procedure  erreur (valeur s: chaîne de caractères)
  debut erreur
        ecrire "Programme Machin: on a renconter l'erreur" s
  fin erreur
...
programme Machin
debut Machin
        ...
        erreur("Entier trop grand")
        ...
        erreur("Pas assez de memoire")
        ...
fin Machin
```

Fonctions: déclaration et appel

Chaque fonction aura:

1. Une *définition* qui dit
   (a) Le type des paramètres.
   (b) Le type de la valeur rendue.
   (c) comment on la calcule.
2. Un ou plusieurs *appels*: c'est l'utilisation de la fonction.
3. Zero, un ou plusieurs *paramètres*: ce sont les arguments de la fonction.
4. Un *type* et une *valeur*.

Exemples (sans paramètres)

```
fonction pi(): reel
  debut pi
      retourner 3.1415926535897931
  fin pi
```

Exemple avec paramètres

La fonction suivante rend le maximum de deux valeurs de type réél.

```
fonction fmax(valeur a: reel, valeur b: reel) : reel
variables m : un reel
debut fmax
 si (a > b)
 alors
   m <- a
 sinon
   m <- b
 fin si
 retourner m
fin fmax
```

de sorte que l'expression `fmax(pi()*pi(),10.0)` vaut 10.

Paramètres

En général, une fonction ou procédure n'est utile que si elle a des paramètres, comme dans le cas de `fmax`.

Une fonction ou procédure définie avec des paramètres doit être appelée avec des *arguments* (en nombre égal, et du même type!).

définition de fonction	appel de fonction
paramètre	argument


```
fonction fmax(valeur a: reel, valeur b: reel) : reel
...
programme essai
variables x: un reel
          i,j: deux caracteres
debut essai
   lire x
   ecrire fmax(x,x*x)
   /* ecrire fmax(i,j) serait incorrect! */
fin essai
```

Nature des paramètres: déclaration

Dans la declaration

```
fonction fmax(valeur a: reel, valeur b: reel) : reel
...
```

- `a` et `b` sont les paramètres de la fonction
- ils sont déclarés avec leur type
- dans le corps de la fonction, on peut utiliser `a` et `b` comme des variables,
  qui existent exclusivement dans le corps de la fonction
- le mot clé `valeur` indique une *modalité* de passage des paramètres

Nature des paramètres: passage par valeur

Après la déclaration

```
fonction fmax(valeur a: reel, valeur b: reel) : reel
...
```

on peut effectuer un appel de la fonction comme

```
ecrire 3*fmax(x,y)
```

l'expression `fmax(x,y)` est évaluée de la façon suivante:

- les valeurs de `x` et `y` sont calculées

- les paramètres `a` et `b` de `fmax` sont initialisés en utilisant ces valuers
- le corps de la fonction `fmax` est executé, (cela utilise les paramètres `a` et `b` comme des variables)
- le résultat de la fonction est enfin retourné et utilisé pour continuer le calcul de l'expression (ici, on le multiplie par 3 et on l'imprime)

Peu importe les opérations effectuées sur `a` *et* `b` *dans le corps de* `fmax`*, les valeurs des paramètres actuels* `x` *et* `y` *restent inchangés.*

Limite du passage par valeur

On a souvent besoin d'écrire des procédure ou fonctions que, au contraire, *modifient* la valeur de certains paramètres, par exemple, on voudrait écrire une procedure `echange` t.q. `echange(x,y)` échange les valeurs des variables `x` et `y`.

Mais la procédure

```
procédure echange (valeur a: entier, valeur b: entier)
variable t: un entier
debut echange
 t <- a
 a <- y
 b <- t
fin echange
```

n'échange pas les valeurs de ses arguments!
(relisez la description du passage d'arguments par valeur plus haut)

Comment faire?

Passage par référence

Si on écrit

```
procédure echange (reference a: entier, reference b: entier)
variable t: un entier
debut echange
 t <- a
 a <- y
 b <- t
fin echange
```


le passage des paramètres se fait autrement: au moment de l'appel

`echange(i,j)`

- *on ne fait pas une copie des valeurs de i et j dans a et b!*
- au contraire, on transforme a et b en "alias" de i et j, en le faisant pointer sur les memes cases mémoire de i et j
- donc, toute opération effectuée sur les paramètres a et b dans le corps de la procédure ou fonction et aussi effectué sur les arguments i et j

N.B.: on doit passer comme argument d'un paramètre par référence seulement des expressions qui se comportent comme des variables (par exemple, une variable, mais aussi une case d'un tableau etc.)

Résumé

- On déclare "valeur" un paramètre si on ne veut pas que l'appel modifie les arguments
- On déclare "reference" un paramètre si on veut que l'appel modifie les arguments

Bien sûr, on peut melanger paramètres par valeur et par référence.

ATTENTION:

- les tableaux sont *toujours* passé par référence, et *jamais* copiés, même si on déclare le paramètres correspondant "valeur".
(Vous comprendrez les raisons de cela plus avant dans le cursus informatique).

Un exemple complet: le tri par séléction

Voyons comme écrire un programme de tri par séléction beaucoup plus lisible avec les fonctions et les procédures.

Fonction de recherche du minimum à partir d'un indice

```
fonction mintab(valeur i: un entier, reference a: tableau): entier
variable k: un entier
debut mintab
   pour k <- i+1 a  N faire
    si  (a[k] < a[i]) alors
        i <- k
    fin si
   fin pour
   retourner i
fin mintab
```

Programme de tri par séléction

```
constante N=10
type tab = tableau de N entiers
procedure echange(reference a, reference b) ...
fonction mintab(valeur i: un entier, reference a: tableau): entier ...

programme triselection
variables a: tab
debut triselection
lire a
  pour i <- 1 a N-1 faire
      echange(a[i],a[mintab(i,a)])
  fin pour
ecrire a
fin triselection
```


Et en C++?

```
#include <iostream.h>

const int N=10;

typedef int tab[N];

void echange (int &a, int &b)
{ int t;
  t=a; a=b; b=t;
}

int mintab(int i, tab &a)
{ int k;

  for(k=i+1;k<N;k++){if (a[k] < a[i]) {i =k;};};
  return(i);
}

void lire(tab &a)
{ int k;

  for (k=0;k<N;k++){cout<<"a["<<k<<"] ? "<<endl; cin>>a[k];};
}

void ecrire(tab &a)
{ int k;
 for (k=0;k<N;k++){cout<<"a["<<k<<"] = "<<a[k]<<endl;};
}

void main(){
 tab a;
 int i;

 lire(a);
 for (i=0;i<N-1;i++){echange(a[i],a[mintab(i,a)]);};
 ecrire(a);
}
```

=

Fonctions et procédures

Idée : capitaliser sur le code déjà écrit pour introduire des nouvelles "commandes" et "opérations".

On souhaite donner un nom programme que nous avons déjà écrit, et qui effectue une opération intéressante, pour pouvoir le réutiliser après sans le réécrire. Si ce programme retourne un résultat, on parlera de "fonction", et "procédure" sinon.

De plus, on peut souhaiter choisir les valeurs de certaines variables de la procédure ou fonction, et cela se fait à travers le mechanisme de passage des paramètres.

Procedures: déclaration et appel

Chaque procédure aura:

1. Une *définition* qui dit
   (a) Le type des paramètres.
   (b) comment on la calcule.
2. Un ou plusieurs *appels*: c'est l'utilisation de la procédure.

Exemples (sans paramètres)

Tous les programmes affichent un message d'aide à l'utilisation, et cela à plusieurs occasions

```
procedure  aide ()
  debut aide
        ecrire "Aide du programme Machin."
        ecrire " l'option -o permet de definir le fichier sortie"
        ecrire " l'option -h donne ce message"
  fin aide
...
programme Machin
debut Machin
```



```
        ...
        aide()
        ...
        si (erreur=vrai) alors aide() fin si
        ...
fin Machin
```

Exemples (avec paramètres)

Tous les programmes affichent des messages d'erreur

```
procedure  erreur (valeur s: chaîne de caractères)
  debut erreur
        ecrire "Programme Machin: on a renconter l'erreur" s
  fin erreur
...
programme Machin
debut Machin
        ...
        erreur("Entier trop grand")
        ...
        erreur("Pas assez de memoire")
        ...
fin Machin
```

Fonctions: déclaration et appel

Chaque fonction aura:

1. Une *définition* qui dit
   (a) Le type des paramètres.
   (b) Le type de la valeur rendue.
   (c) comment on la calcule.
2. Un ou plusieurs *appels*: c'est l'utilisation de la fonction.
3. Zero, un ou plusieurs *paramètres*: ce sont les arguments de la fonction.
4. Un *type* et une *valeur*.

Exemples (sans paramètres)

```
fonction pi(): reel
  debut pi
      retourner 3.1415926535897931
  fin pi
```

Exemple avec paramètres

La fonction suivante rend le maximum de deux valeurs de type réél.

```
fonction fmax(valeur a: reel, valeur b: reel) : reel
variables m : un reel
debut fmax
 si (a > b)
 alors
   m <- a
 sinon
   m <- b
 fin si
 retourner m
fin fmax
```

de sorte que l'expression `fmax(pi()*pi(),10.0)` vaut 10.

Paramètres

En général, une fonction ou procédure n'est utile que si elle a des paramètres, comme dans le cas de `fmax`.

Une fonction ou procédure définie avec des paramètres doit être appelée avec des *arguments* (en nombre égal, et du même type!).

définition de fonction	appel de fonction
paramètre	argument


```
fonction fmax(valeur a: reel, valeur b: reel) : reel
...
programme essai
variables x: un reel
          i,j: deux caracteres
debut essai
   lire x
   ecrire fmax(x,x*x)
   /* ecrire fmax(i,j) serait incorrect! */
fin essai
```

Nature des paramètres: déclaration

Dans la declaration

```
fonction fmax(valeur a: reel, valeur b: reel) : reel
...
```

- `a` et `b` sont les paramètres de la fonction
- ils sont déclarés avec leur type
- dans le corps de la fonction, on peut utiliser `a` et `b` comme des variables,
  qui existent exclusivement dans le corps de la fonction
- le mot clé `valeur` indique une *modalité* de passage des paramètres

Nature des paramètres: passage par valeur

Après la déclaration

```
fonction fmax(valeur a: reel, valeur b: reel) : reel
...
```

on peut effectuer un appel de la fonction comme

```
ecrire 3*fmax(x,y)
```

l'expression `fmax(x,y)` est évaluée de la façon suivante:

- les valeurs de `x` et `y` sont calculées

- les paramètres `a` et `b` de `fmax` sont initialisés en utilisant ces valuers
- le corps de la fonction `fmax` est executé, (cela utilise les paramètres `a` et `b` comme des variables)
- le résultat de la fonction est enfin retourné et utilisé pour continuer le calcul de l'expression (ici, on le multiplie par 3 et on l'imprime)

Peu importe les opérations effectuées sur `a` *et* `b` *dans le corps de* `fmax`*, les valeurs des paramètres actuels* `x` *et* `y` *restent inchangés.*

Limite du passage par valeur

On a souvent besoin d'écrire des procédure ou fonctions que, au contraire, *modifient* la valeur de certains paramètres, par exemple, on voudrait écrire une procedure `echange` t.q. `echange(x,y)` échange les valeurs des variables `x` et `y`.

Mais la procédure

```
procédure echange (valeur a: entier, valeur b: entier)
variable t: un entier
debut echange
 t <- a
 a <- y
 b <- t
fin echange
```

n'échange pas les valeurs de ses arguments!
(relisez la description du passage d'arguments par valeur plus haut)

Comment faire?

Passage par référence

Si on écrit

```
procédure echange (reference a: entier, reference b: entier)
variable t: un entier
debut echange
 t <- a
 a <- y
 b <- t
fin echange
```


le passage des paramètres se fait autrement: au moment de l'appel

`echange(i,j)`

- *on ne fait pas une copie des valeurs de i et j dans a et b!*
- au contraire, on transforme a et b en "alias" de i et j, en le faisant pointer sur les memes cases mémoire de i et j
- donc, toute opération effectuée sur les paramètres a et b dans le corps de la procédure ou fonction et aussi effectué sur les arguments i et j

N.B.: on doit passer comme argument d'un paramètre par référence seulement des expressions qui se comportent comme des variables (par exemple, une variable, mais aussi une case d'un tableau etc.)

Résumé

- On déclare "valeur" un paramètre si on ne veut pas que l'appel modifie les arguments
- On déclare "reference" un paramètre si on veut que l'appel modifie les arguments

Bien sûr, on peut melanger paramètres par valeur et par référence.

ATTENTION:

- les tableaux sont *toujours* passé par référence, et *jamais* copiés, même si on déclare le paramètres correspondant "valeur".
  (Vous comprendrez les raisons de cela plus avant dans le cursus informatique).

Un exemple complet: le tri par séléction

Voyons comme écrire un programme de tri par séléction beaucoup plus lisible avec les fonctions et les procédures.

Fonction de recherche du minimum à partir d'un indice

```
fonction mintab(valeur i: un entier, reference a: tableau): entier
variable k: un entier
debut mintab
   pour k <- i+1 a  N faire
    si  (a[k] < a[i]) alors
        i <- k
    fin si
   fin pour
   retourner i
fin mintab
```

Programme de tri par séléction

```
constante N=10
type tab = tableau de N entiers
procedure echange(reference a, reference b) ...
fonction mintab(valeur i: un entier, reference a: tableau): entier ...

programme triselection
variables a: tab
debut triselection
lire a
  pour i <- 1 a N-1 faire
      echange(a[i],a[mintab(i,a)])
  fin pour
ecrire a
fin triselection
```


Et en C++?

```
#include <iostream.h>

const int N=10;

typedef int tab[N];

void echange (int &a, int &b)
{ int t;
  t=a; a=b; b=t;
}

int mintab(int i, tab &a)
{ int k;

  for(k=i+1;k<N;k++){if (a[k] < a[i]) {i =k;};};
  return(i);
}

void lire(tab &a)
{ int k;

  for (k=0;k<N;k++){cout<<"a["<<k<<"] ? "<<endl; cin>>a[k];};
}

void ecrire(tab &a)
{ int k;
 for (k=0;k<N;k++){cout<<"a["<<k<<"] = "<<a[k]<<endl;};
}

void main(){
 tab a;
 int i;

 lire(a);
 for (i=0;i<N-1;i++){echange(a[i],a[mintab(i,a)]);};
 ecrire(a);
}
```

=

Fonctions et procédures

Idée : capitaliser sur le code déjà écrit pour introduire des nouvelles "commandes" et "opérations".

On souhaite donner un nom programme que nous avons déjà écrit, et qui effectue une opération intéressante, pour pouvoir le réutiliser après sans le réécrire. Si ce programme retourne un résultat, on parlera de "fonction", et "procédure" sinon.

De plus, on peut souhaiter choisir les valeurs de certaines variables de la procédure ou fonction, et cela se fait à travers le mechanisme de passage des paramètres.

Procedures: déclaration et appel

Chaque procédure aura:

1. Une *définition* qui dit
   (a) Le type des paramètres.
   (b) comment on la calcule.
2. Un ou plusieurs *appels*: c'est l'utilisation de la procédure.

Exemples (sans paramètres)

Tous les programmes affichent un message d'aide à l'utilisation, et cela à plusieurs occasions

```
procedure  aide ()
  debut aide
        ecrire "Aide du programme Machin."
        ecrire " l'option -o permet de definir le fichier sortie"
        ecrire " l'option -h donne ce message"
  fin aide
...
programme Machin
debut Machin
```



```
        ...
        aide()
        ...
        si (erreur=vrai) alors aide() fin si
        ...
fin Machin
```

Exemples (avec paramètres)

Tous les programmes affichent des messages d'erreur

```
procedure  erreur (valeur s: chaîne de caractères)
  debut erreur
        ecrire "Programme Machin: on a renconter l'erreur" s
  fin erreur
...
programme Machin
debut Machin
        ...
        erreur("Entier trop grand")
        ...
        erreur("Pas assez de memoire")
        ...
fin Machin
```

Fonctions: déclaration et appel

Chaque fonction aura:

1. Une *définition* qui dit
   (a) Le type des paramètres.
   (b) Le type de la valeur rendue.
   (c) comment on la calcule.
2. Un ou plusieurs *appels*: c'est l'utilisation de la fonction.
3. Zero, un ou plusieurs *paramètres*: ce sont les arguments de la fonction.
4. Un *type* et une *valeur*.

Exemples (sans paramètres)

```
fonction pi(): reel
  debut pi
      retourner 3.1415926535897931
  fin pi
```

Exemple avec paramètres

La fonction suivante rend le maximum de deux valeurs de type réél.

```
fonction fmax(valeur a: reel, valeur b: reel) : reel
variables m : un reel
debut fmax
 si (a > b)
 alors
   m <- a
 sinon
   m <- b
 fin si
 retourner m
fin fmax
```

de sorte que l'expression `fmax(pi()*pi(),10.0)` vaut 10.

Paramètres

En général, une fonction ou procédure n'est utile que si elle a des paramètres, comme dans le cas de `fmax`.

Une fonction ou procédure définie avec des paramètres doit être appelée avec des *arguments* (en nombre égal, et du même type!).

définition de fonction	appel de fonction
paramètre	argument


```
fonction fmax(valeur a: reel, valeur b: reel) : reel
...
programme essai
variables x: un reel
          i,j: deux caracteres
debut essai
   lire x
   ecrire fmax(x,x*x)
   /* ecrire fmax(i,j) serait incorrect! */
fin essai
```

Nature des paramètres: déclaration

Dans la declaration

```
fonction fmax(valeur a: reel, valeur b: reel) : reel
...
```

- `a` et `b` sont les paramètres de la fonction
- ils sont déclarés avec leur type
- dans le corps de la fonction, on peut utiliser `a` et `b` comme des variables,
  qui existent exclusivement dans le corps de la fonction
- le mot clé `valeur` indique une *modalité* de passage des paramètres

Nature des paramètres: passage par valeur

Après la déclaration

```
fonction fmax(valeur a: reel, valeur b: reel) : reel
...
```

on peut effectuer un appel de la fonction comme

```
ecrire 3*fmax(x,y)
```

l'expression `fmax(x,y)` est évaluée de la façon suivante:

- les valeurs de `x` et `y` sont calculées

- les paramètres `a` et `b` de `fmax` sont initialisés en utilisant ces valuers
- le corps de la fonction `fmax` est executé, (cela utilise les paramètres `a` et `b` comme des variables)
- le résultat de la fonction est enfin retourné et utilisé pour continuer le calcul de l'expression (ici, on le multiplie par 3 et on l'imprime)

Peu importe les opérations effectuées sur `a` *et* `b` *dans le corps de* `fmax`, *les valeurs des paramètres actuels* `x` *et* `y` *restent inchangés.*

Limite du passage par valeur

On a souvent besoin d'écrire des procédure ou fonctions que, au contraire, *modifient* la valeur de certains paramètres, par exemple, on voudrait écrire une procedure `echange` t.q. `echange(x,y)` échange les valeurs des variables `x` et `y`.

Mais la procédure

```
procédure echange (valeur a: entier, valeur b: entier)
variable t: un entier
debut echange
 t <- a
 a <- y
 b <- t
fin echange
```

n'échange pas les valeurs de ses arguments!
(relisez la description du passage d'arguments par valeur plus haut)

Comment faire?

Passage par référence

Si on écrit

```
procédure echange (reference a: entier, reference b: entier)
variable t: un entier
debut echange
 t <- a
 a <- y
 b <- t
fin echange
```


le passage des paramètres se fait autrement: au moment de l'appel

`echange(i,j)`

- *on ne fait pas une copie des valeurs de i et j dans a et b!*
- au contraire, on transforme a et b en "alias" de i et j, en le faisant pointer sur les memes cases mémoire de i et j
- donc, toute opération effectuée sur les paramètres a et b dans le corps de la procédure ou fonction et aussi effectué sur les arguments i et j

N.B.: on doit passer comme argument d'un paramètre par référence seulement des expressions qui se comportent comme des variables (par exemple, une variable, mais aussi une case d'un tableau etc.)

Résumé

- On déclare "valeur" un paramètre si on ne veut pas que l'appel modifie les arguments
- On déclare "reference" un paramètre si on veut que l'appel modifie les arguments

Bien sûr, on peut melanger paramètres par valeur et par référence.

ATTENTION:

- les tableaux sont *toujours* passé par référence, et *jamais* copiés, même si on déclare le paramètres correspondant "valeur".
(Vous comprendrez les raisons de cela plus avant dans le cursus informatique).

Un exemple complet: le tri par séléction

Voyons comme écrire un programme de tri par séléction beaucoup plus lisible avec les fonctions et les procédures.

Fonction de recherche du minimum à partir d'un indice

```
fonction mintab(valeur i: un entier, reference a: tableau): entier
variable k: un entier
debut mintab
   pour k <- i+1 a  N faire
    si  (a[k] < a[i]) alors
        i <- k
    fin si
   fin pour
   retourner i
fin mintab
```

Programme de tri par séléction

```
constante N=10
type tab = tableau de N entiers
procedure echange(reference a, reference b) ...
fonction mintab(valeur i: un entier, reference a: tableau): entier ...

programme triselection
variables a: tab
debut triselection
lire a
  pour i <- 1 a N-1 faire
      echange(a[i],a[mintab(i,a)])
  fin pour
ecrire a
fin triselection
```


Et en C++?

```
#include <iostream.h>

const int N=10;

typedef int tab[N];

void echange (int &a, int &b)
{ int t;
  t=a; a=b; b=t;
}

int mintab(int i, tab &a)
{ int k;

  for(k=i+1;k<N;k++){if (a[k] < a[i]) {i =k;};};
  return(i);
}

void lire(tab &a)
{ int k;

  for (k=0;k<N;k++){cout<<"a["<<k<<"] ? "<<endl; cin>>a[k];};
}

void ecrire(tab &a)
{ int k;
 for (k=0;k<N;k++){cout<<"a["<<k<<"] = "<<a[k]<<endl;};
}

void main(){
 tab a;
 int i;

 lire(a);
 for (i=0;i<N-1;i++){echange(a[i],a[mintab(i,a)]);};
 ecrire(a);
}
```

=

Fonctions et procédures

Idée : capitaliser sur le code déjà écrit pour introduire des nouvelles "commandes" et "opérations".

On souhaite donner un nom programme que nous avons déjà écrit, et qui effectue une opération intéressante, pour pouvoir le réutiliser après sans le réécrire. Si ce programme retourne un résultat, on parlera de "fonction", et "procédure" sinon.

De plus, on peut souhaiter choisir les valeurs de certaines variables de la procédure ou fonction, et cela se fait à travers le mechanisme de passage des paramètres.

Procedures: déclaration et appel

Chaque procédure aura:

1. Une *définition* qui dit
   (a) Le type des paramètres.
   (b) comment on la calcule.
2. Un ou plusieurs *appels*: c'est l'utilisation de la procédure.

Exemples (sans paramètres)

Tous les programmes affichent un message d'aide à l'utilisation, et cela à plusieurs occasions

```
procedure  aide ()
  debut aide
        ecrire "Aide du programme Machin."
        ecrire " l'option -o permet de definir le fichier sortie"
        ecrire " l'option -h donne ce message"
  fin aide
...
programme Machin
debut Machin
```



```
        ...
        aide()
        ...
        si (erreur=vrai) alors aide() fin si
        ...
fin Machin
```

Exemples (avec paramètres)

Tous les programmes affichent des messages d'erreur

```
procedure  erreur (valeur s: chaîne de caractères)
  debut erreur
        ecrire "Programme Machin: on a renconter l'erreur" s
  fin erreur
...
programme Machin
debut Machin
        ...
        erreur("Entier trop grand")
        ...
        erreur("Pas assez de memoire")
        ...
fin Machin
```

Fonctions: déclaration et appel

Chaque fonction aura:

1. Une *définition* qui dit
   (a) Le type des paramètres.
   (b) Le type de la valeur rendue.
   (c) comment on la calcule.
2. Un ou plusieurs *appels*: c'est l'utilisation de la fonction.
3. Zero, un ou plusieurs *paramètres*: ce sont les arguments de la fonction.
4. Un *type* et une *valeur*.

Exemples (sans paramètres)

```
fonction pi(): reel
  debut pi
      retourner 3.1415926535897931
  fin pi
```

Exemple avec paramètres

La fonction suivante rend le maximum de deux valeurs de type réél.

```
fonction fmax(valeur a: reel, valeur b: reel) : reel
variables m : un reel
debut fmax
 si (a > b)
 alors
   m <- a
 sinon
   m <- b
 fin si
 retourner m
fin fmax
```

de sorte que l'expression `fmax(pi()*pi(),10.0)` vaut 10.

Paramètres

En général, une fonction ou procédure n'est utile que si elle a des paramètres, comme dans le cas de `fmax`.

Une fonction ou procédure définie avec des paramètres doit être appelée avec des *arguments* (en nombre égal, et du même type!).

définition de fonction	appel de fonction
paramètre	argument


```
fonction fmax(valeur a: reel, valeur b: reel) : reel
...
programme essai
variables x: un reel
          i,j: deux caracteres
debut essai
   lire x
   ecrire fmax(x,x*x)
   /* ecrire fmax(i,j) serait incorrect! */
fin essai
```

Nature des paramètres: déclaration

Dans la declaration

```
fonction fmax(valeur a: reel, valeur b: reel) : reel
...
```

- `a` et `b` sont les paramètres de la fonction
- ils sont déclarés avec leur type
- dans le corps de la fonction, on peut utiliser `a` et `b` comme des variables,
  qui existent exclusivement dans le corps de la fonction
- le mot clé `valeur` indique une *modalité* de passage des paramètres

Nature des paramètres: passage par valeur

Après la déclaration

```
fonction fmax(valeur a: reel, valeur b: reel) : reel
...
```

on peut effectuer un appel de la fonction comme

```
ecrire 3*fmax(x,y)
```

l'expression `fmax(x,y)` est évaluée de la façon suivante:

- les valeurs de `x` et `y` sont calculées

- les paramètres `a` et `b` de `fmax` sont initialisés en utilisant ces valuers
- le corps de la fonction `fmax` est executé, (cela utilise les paramètres `a` et `b` comme des variables)
- le résultat de la fonction est enfin retourné et utilisé pour continuer le calcul de l'expression (ici, on le multiplie par 3 et on l'imprime)

Peu importe les opérations effectuées sur `a` *et* `b` *dans le corps de* `fmax`*, les valeurs des paramètres actuels* `x` *et* `y` *restent inchangés.*

Limite du passage par valeur

On a souvent besoin d'écrire des procédure ou fonctions que, au contraire, *modifient* la valeur de certains paramètres, par exemple, on voudrait écrire une procedure `echange` t.q. `echange(x,y)` échange les valeurs des variables `x` et `y`.

Mais la procédure

```
procédure echange (valeur a: entier, valeur b: entier)
variable t: un entier
debut echange
 t <- a
 a <- y
 b <- t
fin echange
```

n'échange pas les valeurs de ses arguments!
(relisez la description du passage d'arguments par valeur plus haut)

Comment faire?

Passage par référence

Si on écrit

```
procédure echange (reference a: entier, reference b: entier)
variable t: un entier
debut echange
 t <- a
 a <- y
 b <- t
fin echange
```


le passage des paramètres se fait autrement: au moment de l'appel

`echange(i,j)`

- *on ne fait pas une copie des valeurs de i et j dans a et b!*
- au contraire, on transforme a et b en "alias" de i et j, en le faisant pointer sur les memes cases mémoire de i et j
- donc, toute opération effectuée sur les paramètres a et b dans le corps de la procédure ou fonction et aussi effectué sur les arguments i et j

N.B.: on doit passer comme argument d'un paramètre par référence seulement des expressions qui se comportent comme des variables (par exemple, une variable, mais aussi une case d'un tableau etc.)

Résumé

- On déclare "valeur" un paramètre si on ne veut pas que l'appel modifie les arguments
- On déclare "reference" un paramètre si on veut que l'appel modifie les arguments

Bien sûr, on peut melanger paramètres par valeur et par référence.

ATTENTION:

- les tableaux sont *toujours* passé par référence, et *jamais* copiés, même si on déclare le paramètres correspondant "valeur".
(Vous comprendrez les raisons de cela plus avant dans le cursus informatique).

Un exemple complet: le tri par séléction

Voyons comme écrire un programme de tri par séléction beaucoup plus lisible avec les fonctions et les procédures.

Fonction de recherche du minimum à partir d'un indice

```
fonction mintab(valeur i: un entier, reference a: tableau): entier
variable k: un entier
debut mintab
   pour k <- i+1 a  N faire
    si (a[k] < a[i]) alors
        i <- k
    fin si
   fin pour
   retourner i
fin mintab
```

Programme de tri par séléction

```
constante N=10
type tab = tableau de N entiers
procedure echange(reference a, reference b) ...
fonction mintab(valeur i: un entier, reference a: tableau): entier ...

programme triselection
variables a: tab
debut triselection
lire a
  pour i <- 1 a N-1 faire
      echange(a[i],a[mintab(i,a)])
  fin pour
ecrire a
fin triselection
```


Et en C++?

```
#include <iostream.h>

const int N=10;

typedef int tab[N];

void echange (int &a, int &b)
{ int t;
  t=a; a=b; b=t;
}

int mintab(int i, tab &a)
{ int k;

  for(k=i+1;k<N;k++){if (a[k] < a[i]) {i =k;};};
  return(i);
}

void lire(tab &a)
{ int k;

  for (k=0;k<N;k++){cout<<"a["<<k<<"] ? "<<endl; cin>>a[k];};
}

void ecrire(tab &a)
{ int k;
 for (k=0;k<N;k++){cout<<"a["<<k<<"] = "<<a[k]<<endl;};
}

void main(){
 tab a;
 int i;

 lire(a);
 for (i=0;i<N-1;i++){echange(a[i],a[mintab(i,a)]);};
 ecrire(a);
}
```

=

Fonctions et procédures

Idée : capitaliser sur le code déjà écrit pour introduire des nouvelles "commandes" et "opérations".

On souhaite donner un nom programme que nous avons déjà écrit, et qui effectue une opération intéressante, pour pouvoir le réutiliser après sans le réécrire. Si ce programme retourne un résultat, on parlera de "fonction", et "procédure" sinon.

De plus, on peut souhaiter choisir les valeurs de certaines variables de la procédure ou fonction, et cela se fait à travers le mechanisme de passage des paramètres.

Procedures: déclaration et appel

Chaque procédure aura:

1. Une *définition* qui dit
   (a) Le type des paramètres.
   (b) comment on la calcule.
2. Un ou plusieurs *appels*: c'est l'utilisation de la procédure.

Exemples (sans paramètres)

Tous les programmes affichent un message d'aide à l'utilisation, et cela à plusieurs occasions

```
procedure  aide ()
  debut aide
        ecrire "Aide du programme Machin."
        ecrire " l'option -o permet de definir le fichier sortie"
        ecrire " l'option -h donne ce message"
  fin aide
...
programme Machin
debut Machin
```



```
        ...
        aide()
        ...
        si (erreur=vrai) alors aide() fin si
        ...
fin Machin
```

Exemples (avec paramètres)

Tous les programmes affichent des messages d'erreur

```
procedure  erreur (valeur s: chaîne de caractères)
  debut erreur
        ecrire "Programme Machin: on a renconter l'erreur" s
  fin erreur
...
programme Machin
debut Machin
        ...
        erreur("Entier trop grand")
        ...
        erreur("Pas assez de memoire")
        ...
fin Machin
```

Fonctions: déclaration et appel

Chaque fonction aura:

1. Une *définition* qui dit
   (a) Le type des paramètres.
   (b) Le type de la valeur rendue.
   (c) comment on la calcule.
2. Un ou plusieurs *appels*: c'est l'utilisation de la fonction.
3. Zero, un ou plusieurs *paramètres*: ce sont les arguments de la fonction.
4. Un *type* et une *valeur*.

Exemples (sans paramètres)

```
fonction pi(): reel
  debut pi
      retourner 3.1415926535897931
  fin pi
```

Exemple avec paramètres

La fonction suivante rend le maximum de deux valeurs de type réél.

```
fonction fmax(valeur a: reel, valeur b: reel) : reel
variables m : un reel
debut fmax
 si (a > b)
 alors
   m <- a
 sinon
   m <- b
 fin si
 retourner m
fin fmax
```

de sorte que l'expression `fmax(pi()*pi(),10.0)` vaut 10.

Paramètres

En général, une fonction ou procédure n'est utile que si elle a des paramètres, comme dans le cas de `fmax`.

Une fonction ou procédure définie avec des paramètres doit être appelée avec des *arguments* (en nombre égal, et du même type!).

définition de fonction	appel de fonction
paramètre	argument


```
fonction fmax(valeur a: reel, valeur b: reel) : reel
...
programme essai
variables x: un reel
          i,j: deux caracteres
debut essai
   lire x
   ecrire fmax(x,x*x)
   /* ecrire fmax(i,j) serait incorrect! */
fin essai
```

Nature des paramètres: déclaration

Dans la declaration

```
fonction fmax(valeur a: reel, valeur b: reel) : reel
...
```

- `a` et `b` sont les paramètres de la fonction
- ils sont déclarés avec leur type
- dans le corps de la fonction, on peut utiliser `a` et `b` comme des variables,
  qui existent exclusivement dans le corps de la fonction
- le mot clé `valeur` indique une *modalité* de passage des paramètres

Nature des paramètres: passage par valeur

Après la déclaration

```
fonction fmax(valeur a: reel, valeur b: reel) : reel
...
```

on peut effectuer un appel de la fonction comme

```
ecrire 3*fmax(x,y)
```

l'expression `fmax(x,y)` est évaluée de la façon suivante:

- les valeurs de `x` et `y` sont calculées

- les paramètres `a` et `b` de `fmax` sont initialisés en utilisant ces valuers
- le corps de la fonction `fmax` est executé, (cela utilise les paramètres `a` et `b` comme des variables)
- le résultat de la fonction est enfin retourné et utilisé pour continuer le calcul de l'expression (ici, on le multiplie par 3 et on l'imprime)

Peu importe les opérations effectuées sur `a` *et* `b` *dans le corps de* `fmax`*, les valeurs des paramètres actuels* `x` *et* `y` *restent inchangés.*

Limite du passage par valeur

On a souvent besoin d'écrire des procédure ou fonctions que, au contraire, *modifient* la valeur de certains paramètres, par exemple, on voudrait écrire une procedure `echange` t.q. `echange(x,y)` échange les valeurs des variables `x` et `y`.

Mais la procédure

```
procédure echange (valeur a: entier, valeur b: entier)
variable t: un entier
debut echange
 t <- a
 a <- y
 b <- t
fin echange
```

n'échange pas les valeurs de ses arguments!
(relisez la description du passage d'arguments par valeur plus haut)

Comment faire?

Passage par référence

Si on écrit

```
procédure echange (reference a: entier, reference b: entier)
variable t: un entier
debut echange
 t <- a
 a <- y
 b <- t
fin echange
```


le passage des paramètres se fait autrement: au moment de l'appel

`echange(i,j)`

- *on ne fait pas une copie des valeurs de i et j dans a et b!*
- au contraire, on transforme a et b en "alias" de i et j, en le faisant pointer sur les memes cases mémoire de i et j
- donc, toute opération effectuée sur les paramètres a et b dans le corps de la procédure ou fonction et aussi effectué sur les arguments i et j

N.B.: on doit passer comme argument d'un paramètre par référence seulement des expressions qui se comportent comme des variables (par exemple, une variable, mais aussi une case d'un tableau etc.)

Résumé

- On déclare "valeur" un paramètre si on ne veut pas que l'appel modifie les arguments
- On déclare "reference" un paramètre si on veut que l'appel modifie les arguments

Bien sûr, on peut melanger paramètres par valeur et par référence.

ATTENTION:

- les tableaux sont *toujours* passé par référence, et *jamais* copiés, même si on déclare le paramètres correspondant "valeur".
(Vous comprendrez les raisons de cela plus avant dans le cursus informatique).

Un exemple complet: le tri par séléction

Voyons comme écrire un programme de tri par séléction beaucoup plus lisible avec les fonctions et les procédures.

Fonction de recherche du minimum à partir d'un indice

```
fonction mintab(valeur i: un entier, reference a: tableau): entier
variable k: un entier
debut mintab
   pour k <- i+1 a  N faire
    si  (a[k] < a[i]) alors
        i <- k
    fin si
   fin pour
   retourner i
fin mintab
```

Programme de tri par séléction

```
constante N=10
type tab = tableau de N entiers
procedure echange(reference a, reference b) ...
fonction mintab(valeur i: un entier, reference a: tableau): entier ...

programme triselection
variables a: tab
debut triselection
lire a
  pour i <- 1 a N-1 faire
      echange(a[i],a[mintab(i,a)])
  fin pour
ecrire a
fin triselection
```


Et en C++?

```
#include <iostream.h>

const int N=10;

typedef int tab[N];

void echange (int &a, int &b)
{ int t;
  t=a; a=b; b=t;
}

int mintab(int i, tab &a)
{ int k;

  for(k=i+1;k<N;k++){if (a[k] < a[i]) {i =k;};};
  return(i);
}

void lire(tab &a)
{ int k;

  for (k=0;k<N;k++){cout<<"a["<<k<<"] ? "<<endl; cin>>a[k];};
}

void ecrire(tab &a)
{ int k;
 for (k=0;k<N;k++){cout<<"a["<<k<<"] = "<<a[k]<<endl;};
}

void main(){
 tab a;
 int i;

 lire(a);
 for (i=0;i<N-1;i++){echange(a[i],a[mintab(i,a)]);};
 ecrire(a);
}
```

=

Fonctions et procédures

Idée : capitaliser sur le code déjà écrit pour introduire des nouvelles "commandes" et "opérations".

On souhaite donner un nom programme que nous avons déjà écrit, et qui effectue une opération intéressante, pour pouvoir le réutiliser après sans le réécrire. Si ce programme retourne un résultat, on parlera de "fonction", et "procédure" sinon.

De plus, on peut souhaiter choisir les valeurs de certaines variables de la procédure ou fonction, et cela se fait à travers le mechanisme de passage des paramètres.

Procedures: déclaration et appel

Chaque procédure aura:

1. Une *définition* qui dit
   (a) Le type des paramètres.
   (b) comment on la calcule.
2. Un ou plusieurs *appels*: c'est l'utilisation de la procédure.

Exemples (sans paramètres)

Tous les programmes affichent un message d'aide à l'utilisation, et cela à plusieurs occasions

```
procedure  aide ()
  debut aide
        ecrire "Aide du programme Machin."
        ecrire " l'option -o permet de definir le fichier sortie"
        ecrire " l'option -h donne ce message"
  fin aide
...
programme Machin
debut Machin
```



```
        ...
        aide()
        ...
        si (erreur=vrai) alors aide() fin si
        ...
fin Machin
```

Exemples (avec paramètres)

Tous les programmes affichent des messages d'erreur

```
procedure  erreur (valeur s: chaîne de caractères)
  debut erreur
        ecrire "Programme Machin: on a renconter l'erreur" s
  fin erreur
...
programme Machin
debut Machin
        ...
        erreur("Entier trop grand")
        ...
        erreur("Pas assez de memoire")
        ...
fin Machin
```

Fonctions: déclaration et appel

Chaque fonction aura:

1. Une *définition* qui dit
   (a) Le type des paramètres.
   (b) Le type de la valeur rendue.
   (c) comment on la calcule.
2. Un ou plusieurs *appels*: c'est l'utilisation de la fonction.
3. Zero, un ou plusieurs *paramètres*: ce sont les arguments de la fonction.
4. Un *type* et une *valeur*.

Exemples (sans paramètres)

```
fonction pi(): reel
  debut pi
      retourner 3.1415926535897931
  fin pi
```

Exemple avec paramètres

La fonction suivante rend le maximum de deux valeurs de type réél.

```
fonction fmax(valeur a: reel, valeur b: reel) : reel
variables m : un reel
debut fmax
 si (a > b)
 alors
   m <- a
 sinon
   m <- b
 fin si
 retourner m
fin fmax
```

de sorte que l'expression `fmax(pi()*pi(),10.0)` vaut 10.

Paramètres

En général, une fonction ou procédure n'est utile que si elle a des paramètres, comme dans le cas de `fmax`.

Une fonction ou procédure définie avec des paramètres doit être appelée avec des *arguments* (en nombre égal, et du même type!).

définition de fonction	appel de fonction
paramètre	argument


```
fonction fmax(valeur a: reel, valeur b: reel) : reel
...
programme essai
variables x: un reel
          i,j: deux caracteres
debut essai
   lire x
   ecrire fmax(x,x*x)
   /* ecrire fmax(i,j) serait incorrect! */
fin essai
```

Nature des paramètres: déclaration

Dans la declaration

```
fonction fmax(valeur a: reel, valeur b: reel) : reel
...
```

- `a` et `b` sont les paramètres de la fonction
- ils sont déclarés avec leur type
- dans le corps de la fonction, on peut utiliser `a` et `b` comme des variables,
  qui existent exclusivement dans le corps de la fonction
- le mot clé `valeur` indique une *modalité* de passage des paramètres

Nature des paramètres: passage par valeur

Après la déclaration

```
fonction fmax(valeur a: reel, valeur b: reel) : reel
...
```

on peut effectuer un appel de la fonction comme

```
ecrire 3*fmax(x,y)
```

l'expression `fmax(x,y)` est évaluée de la façon suivante:

- les valeurs de `x` et `y` sont calculées

- les paramètres `a` et `b` de `fmax` sont initialisés en utilisant ces valuers
- le corps de la fonction `fmax` est executé, (cela utilise les paramètres `a` et `b` comme des variables)
- le résultat de la fonction est enfin retourné et utilisé pour continuer le calcul de l'expression (ici, on le multiplie par 3 et on l'imprime)

Peu importe les opérations effectuées sur `a` *et* `b` *dans le corps de* `fmax`, *les valeurs des paramètres actuels* `x` *et* `y` *restent inchangés.*

Limite du passage par valeur

On a souvent besoin d'écrire des procédure ou fonctions que, au contraire, *modifient* la valeur de certains paramètres, par exemple, on voudrait écrire une procedure `echange` t.q. `echange(x,y)` échange les valeurs des variables `x` et `y`.

Mais la procédure

```
procédure echange (valeur a: entier, valeur b: entier)
variable t: un entier
debut echange
 t <- a
 a <- y
 b <- t
fin echange
```

n'échange pas les valeurs de ses arguments!
(relisez la description du passage d'arguments par valeur plus haut)

Comment faire?

Passage par référence

Si on écrit

```
procédure echange (reference a: entier, reference b: entier)
variable t: un entier
debut echange
 t <- a
 a <- y
 b <- t
fin echange
```


le passage des paramètres se fait autrement: au moment de l'appel

`echange(i,j)`

- *on ne fait pas une copie des valeurs de i et j dans a et b!*
- au contraire, on transforme a et b en "alias" de i et j, en le faisant pointer sur les memes cases mémoire de i et j
- donc, toute opération effectuée sur les paramètres a et b dans le corps de la procédure ou fonction et aussi effectué sur les arguments i et j

N.B.: on doit passer comme argument d'un paramètre par référence seulement des expressions qui se comportent comme des variables (par exemple, une variable, mais aussi une case d'un tableau etc.)

Résumé

- On déclare "valeur" un paramètre si on ne veut pas que l'appel modifie les arguments
- On déclare "reference" un paramètre si on veut que l'appel modifie les arguments

Bien sûr, on peut melanger paramètres par valeur et par référence.

ATTENTION:

- les tableaux sont *toujours* passé par référence, et *jamais* copiés, même si on déclare le paramètres correspondant "valeur".
(Vous comprendrez les raisons de cela plus avant dans le cursus informatique).

Un exemple complet: le tri par séléction

Voyons comme écrire un programme de tri par séléction beaucoup plus lisible avec les fonctions et les procédures.

Fonction de recherche du minimum à partir d'un indice

```
fonction mintab(valeur i: un entier, reference a: tableau): entier
variable k: un entier
debut mintab
   pour k <- i+1 a  N faire
    si  (a[k] < a[i]) alors
        i <- k
    fin si
   fin pour
   retourner i
fin mintab
```

Programme de tri par séléction

```
constante N=10
type tab = tableau de N entiers
procedure echange(reference a, reference b) ...
fonction mintab(valeur i: un entier, reference a: tableau): entier ...

programme triselection
variables a: tab
debut triselection
lire a
  pour i <- 1 a N-1 faire
      echange(a[i],a[mintab(i,a)])
  fin pour
ecrire a
fin triselection
```


Et en C++?

```
#include <iostream.h>

const int N=10;

typedef int tab[N];

void echange (int &a, int &b)
{ int t;
  t=a; a=b; b=t;
}

int mintab(int i, tab &a)
{ int k;

  for(k=i+1;k<N;k++){if  (a[k] < a[i]) {i =k;};};
  return(i);
}

void lire(tab &a)
{ int k;

  for (k=0;k<N;k++){cout<<"a["<<k<<"] ? "<<endl; cin>>a[k];};
}

void ecrire(tab &a)
{ int k;
 for (k=0;k<N;k++){cout<<"a["<<k<<"] = "<<a[k]<<endl;};
}

void main(){
 tab a;
 int i;

 lire(a);
 for (i=0;i<N-1;i++){echange(a[i],a[mintab(i,a)]);};
 ecrire(a);
}
```

=

Fonctions et procédures

Idée : capitaliser sur le code déjà écrit pour introduire des nouvelles "commandes" et "opérations".

On souhaite donner un nom programme que nous avons déjà écrit, et qui effectue une opération intéressante, pour pouvoir le réutiliser après sans le réécrire. Si ce programme retourne un résultat, on parlera de "fonction", et "procédure" sinon.

De plus, on peut souhaiter choisir les valeurs de certaines variables de la procédure ou fonction, et cela se fait à travers le mechanisme de passage des paramètres.

Procedures: déclaration et appel

Chaque procédure aura:

1. Une *définition* qui dit
   (a) Le type des paramètres.
   (b) comment on la calcule.
2. Un ou plusieurs *appels*: c'est l'utilisation de la procédure.

Exemples (sans paramètres)

Tous les programmes affichent un message d'aide à l'utilisation, et cela à plusieurs occasions

```
procedure  aide ()
  debut aide
        ecrire "Aide du programme Machin."
        ecrire " l'option -o permet de definir le fichier sortie"
        ecrire " l'option -h donne ce message"
  fin aide
...
programme Machin
debut Machin
```



```
        ...
        aide()
        ...
        si (erreur=vrai) alors aide() fin si
        ...
fin Machin
```

Exemples (avec paramètres)

Tous les programmes affichent des messages d'erreur

```
procedure  erreur (valeur s: chaîne de caractères)
  debut erreur
        ecrire "Programme Machin: on a renconter l'erreur" s
  fin erreur
...
programme Machin
debut Machin
        ...
        erreur("Entier trop grand")
        ...
        erreur("Pas assez de memoire")
        ...
fin Machin
```

Fonctions: déclaration et appel

Chaque fonction aura:

1. Une *définition* qui dit
   (a) Le type des paramètres.
   (b) Le type de la valeur rendue.
   (c) comment on la calcule.
2. Un ou plusieurs *appels*: c'est l'utilisation de la fonction.
3. Zero, un ou plusieurs *paramètres*: ce sont les arguments de la fonction.
4. Un *type* et une *valeur*.

Exemples (sans paramètres)

```
fonction pi(): reel
  debut pi
      retourner 3.1415926535897931
  fin pi
```

Exemple avec paramètres

La fonction suivante rend le maximum de deux valeurs de type réél.

```
fonction fmax(valeur a: reel, valeur b: reel) : reel
variables m : un reel
debut fmax
 si (a > b)
 alors
   m <- a
 sinon
   m <- b
 fin si
 retourner m
fin fmax
```

de sorte que l'expression `fmax(pi()*pi(),10.0)` vaut 10.

Paramètres

En général, une fonction ou procédure n'est utile que si elle a des paramètres, comme dans le cas de `fmax`.

Une fonction ou procédure définie avec des paramètres doit être appelée avec des *arguments* (en nombre égal, et du même type!).

définition de fonction	appel de fonction
paramètre	argument


```
fonction fmax(valeur a: reel, valeur b: reel) : reel
...
programme essai
variables x: un reel
          i,j: deux caracteres
debut essai
   lire x
   ecrire fmax(x,x*x)
   /* ecrire fmax(i,j) serait incorrect! */
fin essai
```

Nature des paramètres: déclaration

Dans la declaration

```
fonction fmax(valeur a: reel, valeur b: reel) : reel
...
```

- `a` et `b` sont les paramètres de la fonction
- ils sont déclarés avec leur type
- dans le corps de la fonction, on peut utiliser `a` et `b` comme des variables,
  qui existent exclusivement dans le corps de la fonction
- le mot clé `valeur` indique une *modalité* de passage des paramètres

Nature des paramètres: passage par valeur

Après la déclaration

```
fonction fmax(valeur a: reel, valeur b: reel) : reel
...
```

on peut effectuer un appel de la fonction comme

```
ecrire 3*fmax(x,y)
```

l'expression `fmax(x,y)` est évaluée de la façon suivante:

- les valeurs de `x` et `y` sont calculées

- les paramètres `a` et `b` de `fmax` sont initialisés en utilisant ces valuers
- le corps de la fonction `fmax` est executé, (cela utilise les paramètres `a` et `b` comme des variables)
- le résultat de la fonction est enfin retourné et utilisé pour continuer le calcul de l'expression (ici, on le multiplie par 3 et on l'imprime)

Peu importe les opérations effectuées sur `a` *et* `b` *dans le corps de* `fmax`, *les valeurs des paramètres actuels* `x` *et* `y` *restent inchangés.*

Limite du passage par valeur

On a souvent besoin d'écrire des procédure ou fonctions que, au contraire, *modifient* la valeur de certains paramètres, par exemple, on voudrait écrire une procedure `echange` t.q. `echange(x,y)` échange les valeurs des variables `x` et `y`.

Mais la procédure

```
procédure echange (valeur a: entier, valeur b: entier)
variable t: un entier
debut echange
 t <- a
 a <- y
 b <- t
fin echange
```

n'échange pas les valeurs de ses arguments!
(relisez la description du passage d'arguments par valeur plus haut)

Comment faire?

Passage par référence

Si on écrit

```
procédure echange (reference a: entier, reference b: entier)
variable t: un entier
debut echange
 t <- a
 a <- y
 b <- t
fin echange
```


le passage des paramètres se fait autrement: au moment de l'appel

`echange(i,j)`

- *on ne fait pas une copie des valeurs de i et j dans a et b!*
- au contraire, on transforme a et b en "alias" de i et j, en le faisant pointer sur les memes cases mémoire de i et j
- donc, toute opération effectuée sur les paramètres a et b dans le corps de la procédure ou fonction et aussi effectué sur les arguments i et j

N.B.: on doit passer comme argument d'un paramètre par référence seulement des expressions qui se comportent comme des variables (par exemple, une variable, mais aussi une case d'un tableau etc.)

Résumé

- On déclare "valeur" un paramètre si on ne veut pas que l'appel modifie les arguments
- On déclare "reference" un paramètre si on veut que l'appel modifie les arguments

Bien sûr, on peut melanger paramètres par valeur et par référence.

ATTENTION:

- les tableaux sont *toujours* passé par référence, et *jamais* copiés, même si on déclare le paramètres correspondant "valeur".
  (Vous comprendrez les raisons de cela plus avant dans le cursus informatique).

Un exemple complet: le tri par séléction

Voyons comme écrire un programme de tri par séléction beaucoup plus lisible avec les fonctions et les procédures.

Fonction de recherche du minimum à partir d'un indice

```
fonction mintab(valeur i: un entier, reference a: tableau): entier
variable k: un entier
debut mintab
   pour k <- i+1 a  N faire
    si  (a[k] < a[i]) alors
        i <- k
    fin si
   fin pour
   retourner i
fin mintab
```

Programme de tri par séléction

```
constante N=10
type tab = tableau de N entiers
procedure echange(reference a, reference b) ...
fonction mintab(valeur i: un entier, reference a: tableau): entier ...

programme triselection
variables a: tab
debut triselection
lire a
  pour i <- 1 a N-1 faire
      echange(a[i],a[mintab(i,a)])
  fin pour
ecrire a
fin triselection
```


Et en C++?

```
#include <iostream.h>

const int N=10;

typedef int tab[N];

void echange (int &a, int &b)
{ int t;
  t=a; a=b; b=t;
}

int mintab(int i, tab &a)
{ int k;

  for(k=i+1;k<N;k++){if (a[k] < a[i]) {i =k;};};
  return(i);
}

void lire(tab &a)
{ int k;

  for (k=0;k<N;k++){cout<<"a["<<k<<"] ? "<<endl; cin>>a[k];};
}

void ecrire(tab &a)
{ int k;
 for (k=0;k<N;k++){cout<<"a["<<k<<"] = "<<a[k]<<endl;};
}

void main(){
 tab a;
 int i;

 lire(a);
 for (i=0;i<N-1;i++){echange(a[i],a[mintab(i,a)]);};
 ecrire(a);
}
```

=

Fonctions et procédures

Idée : capitaliser sur le code déjà écrit pour introduire des nouvelles "commandes" et "opérations".

On souhaite donner un nom programme que nous avons déjà écrit, et qui effectue une opération intéressante, pour pouvoir le réutiliser après sans le réécrire. Si ce programme retourne un résultat, on parlera de "fonction", et "procédure" sinon.

De plus, on peut souhaiter choisir les valeurs de certaines variables de la procédure ou fonction, et cela se fait à travers le mechanisme de passage des paramètres.

Procedures: déclaration et appel

Chaque procédure aura:

1. Une *définition* qui dit
   (a) Le type des paramètres.
   (b) comment on la calcule.
2. Un ou plusieurs *appels*: c'est l'utilisation de la procédure.

Exemples (sans paramètres)

Tous les programmes affichent un message d'aide à l'utilisation, et cela à plusieurs occasions

```
procedure  aide ()
  debut aide
        ecrire "Aide du programme Machin."
        ecrire " l'option -o permet de definir le fichier sortie"
        ecrire " l'option -h donne ce message"
  fin aide
...
programme Machin
debut Machin
```



```
        ...
        aide()
        ...
        si (erreur=vrai) alors aide() fin si
        ...
fin Machin
```

Exemples (avec paramètres)

Tous les programmes affichent des messages d'erreur

```
procedure  erreur (valeur s: chaîne de caractères)
  debut erreur
        ecrire "Programme Machin: on a renconter l'erreur" s
  fin erreur
...
programme Machin
debut Machin
        ...
        erreur("Entier trop grand")
        ...
        erreur("Pas assez de memoire")
        ...
fin Machin
```

Fonctions: déclaration et appel

Chaque fonction aura:

1. Une *définition* qui dit
   (a) Le type des paramètres.
   (b) Le type de la valeur rendue.
   (c) comment on la calcule.
2. Un ou plusieurs *appels*: c'est l'utilisation de la fonction.
3. Zero, un ou plusieurs *paramètres*: ce sont les arguments de la fonction.
4. Un *type* et une *valeur*.

Exemples (sans paramètres)

```
fonction pi(): reel
  debut pi
      retourner 3.1415926535897931
  fin pi
```

Exemple avec paramètres

La fonction suivante rend le maximum de deux valeurs de type réél.

```
fonction fmax(valeur a: reel, valeur b: reel) : reel
variables m : un reel
debut fmax
 si (a > b)
 alors
   m <- a
 sinon
   m <- b
 fin si
 retourner m
fin fmax
```

de sorte que l'expression `fmax(pi()*pi(),10.0)` vaut 10.

Paramètres

En général, une fonction ou procédure n'est utile que si elle a des paramètres, comme dans le cas de `fmax`.

Une fonction ou procédure définie avec des paramètres doit être appelée avec des *arguments* (en nombre égal, et du même type!).

définition de fonction	appel de fonction
paramètre	argument


```
fonction fmax(valeur a: reel, valeur b: reel) : reel
...
programme essai
variables x: un reel
          i,j: deux caracteres
debut essai
   lire x
   ecrire fmax(x,x*x)
   /* ecrire fmax(i,j) serait incorrect! */
fin essai
```

Nature des paramètres: déclaration

Dans la declaration

```
fonction fmax(valeur a: reel, valeur b: reel) : reel
...
```

- `a` et `b` sont les paramètres de la fonction
- ils sont déclarés avec leur type
- dans le corps de la fonction, on peut utiliser `a` et `b` comme des variables,
  qui existent exclusivement dans le corps de la fonction
- le mot clé `valeur` indique une *modalité* de passage des paramètres

Nature des paramètres: passage par valeur

Après la déclaration

```
fonction fmax(valeur a: reel, valeur b: reel) : reel
...
```

on peut effectuer un appel de la fonction comme

```
ecrire 3*fmax(x,y)
```

l'expression `fmax(x,y)` est évaluée de la façon suivante:

- les valeurs de `x` et `y` sont calculées

- les paramètres `a` et `b` de `fmax` sont initialisés en utilisant ces valuers
- le corps de la fonction `fmax` est executé, (cela utilise les paramètres `a` et `b` comme des variables)
- le résultat de la fonction est enfin retourné et utilisé pour continuer le calcul de l'expression (ici, on le multiplie par 3 et on l'imprime)

Peu importe les opérations effectuées sur `a` *et* `b` *dans le corps de* `fmax`*, les valeurs des paramètres actuels* `x` *et* `y` *restent inchangés.*

Limite du passage par valeur

On a souvent besoin d'écrire des procédure ou fonctions que, au contraire, *modifient* la valeur de certains paramètres, par exemple, on voudrait écrire une procedure `echange` t.q. `echange(x,y)` échange les valeurs des variables `x` et `y`.

Mais la procédure

```
procédure echange (valeur a: entier, valeur b: entier)
variable t: un entier
debut echange
 t <- a
 a <- y
 b <- t
fin echange
```

n'échange pas les valeurs de ses arguments!
(relisez la description du passage d'arguments par valeur plus haut)

Comment faire?

Passage par référence

Si on écrit

```
procédure echange (reference a: entier, reference b: entier)
variable t: un entier
debut echange
 t <- a
 a <- y
 b <- t
fin echange
```


le passage des paramètres se fait autrement: au moment de l'appel

`echange(i,j)`

- *on ne fait pas une copie des valeurs de i et j dans a et b!*
- au contraire, on transforme a et b en "alias" de i et j, en le faisant pointer sur les memes cases mémoire de i et j
- donc, toute opération effectuée sur les paramètres a et b dans le corps de la procédure ou fonction et aussi effectué sur les arguments i et j

N.B.: on doit passer comme argument d'un paramètre par référence seulement des expressions qui se comportent comme des variables (par exemple, une variable, mais aussi une case d'un tableau etc.)

Résumé

- On déclare "valeur" un paramètre si on ne veut pas que l'appel modifie les arguments
- On déclare "reference" un paramètre si on veut que l'appel modifie les arguments

Bien sûr, on peut melanger paramètres par valeur et par référence.

ATTENTION:

- les tableaux sont *toujours* passé par référence, et *jamais* copiés, même si on déclare le paramètres correspondant "valeur".
(Vous comprendrez les raisons de cela plus avant dans le cursus informatique).

Un exemple complet: le tri par séléction

Voyons comme écrire un programme de tri par séléction beaucoup plus lisible avec les fonctions et les procédures.

Fonction de recherche du minimum à partir d'un indice

```
fonction mintab(valeur i: un entier, reference a: tableau): entier
variable k: un entier
debut mintab
   pour k <- i+1 a  N faire
    si (a[k] < a[i]) alors
        i <- k
    fin si
   fin pour
   retourner i
fin mintab
```

Programme de tri par séléction

```
constante N=10
type tab = tableau de N entiers
procedure echange(reference a, reference b) ...
fonction mintab(valeur i: un entier, reference a: tableau): entier ...

programme triselection
variables a: tab
debut triselection
lire a
  pour i <- 1 a N-1 faire
      echange(a[i],a[mintab(i,a)])
  fin pour
ecrire a
fin triselection
```


Et en C++?

```
#include <iostream.h>

const int N=10;

typedef int tab[N];

void echange (int &a, int &b)
{ int t;
  t=a; a=b; b=t;
}

int mintab(int i, tab &a)
{ int k;

  for(k=i+1;k<N;k++){if (a[k] < a[i]) {i =k;};};
  return(i);
}

void lire(tab &a)
{ int k;

  for (k=0;k<N;k++){cout<<"a["<<k<<"] ? "<<endl; cin>>a[k];};
}

void ecrire(tab &a)
{ int k;
 for (k=0;k<N;k++){cout<<"a["<<k<<"] = "<<a[k]<<endl;};
}

void main(){
 tab a;
 int i;

 lire(a);
 for (i=0;i<N-1;i++){echange(a[i],a[mintab(i,a)]);};
 ecrire(a);
}
```

=

Fonctions et procédures

Idée : capitaliser sur le code déjà écrit pour introduire des nouvelles "commandes" et "opérations".

On souhaite donner un nom programme que nous avons déjà écrit, et qui effectue une opération intéressante, pour pouvoir le réutiliser après sans le réécrire. Si ce programme retourne un résultat, on parlera de "fonction", et "procédure" sinon.

De plus, on peut souhaiter choisir les valeurs de certaines variables de la procédure ou fonction, et cela se fait à travers le mechanisme de passage des paramètres.

Procedures: déclaration et appel

Chaque procédure aura:

1. Une *définition* qui dit
   (a) Le type des paramètres.
   (b) comment on la calcule.
2. Un ou plusieurs *appels*: c'est l'utilisation de la procédure.

Exemples (sans paramètres)

Tous les programmes affichent un message d'aide à l'utilisation, et cela à plusieurs occasions

```
procedure  aide ()
  debut aide
        ecrire "Aide du programme Machin."
        ecrire " l'option -o permet de definir le fichier sortie"
        ecrire " l'option -h donne ce message"
  fin aide
...
programme Machin
debut Machin
```



```
        ...
        aide()
        ...
        si (erreur=vrai) alors aide() fin si
        ...
fin Machin
```

Exemples (avec paramètres)

Tous les programmes affichent des messages d'erreur

```
procedure  erreur (valeur s: chaîne de caractères)
  debut erreur
        ecrire "Programme Machin: on a rencontrer l'erreur" s
  fin erreur
...
programme Machin
debut Machin
        ...
        erreur("Entier trop grand")
        ...
        erreur("Pas assez de memoire")
        ...
fin Machin
```

Fonctions: déclaration et appel

Chaque fonction aura:

1. Une *définition* qui dit
   (a) Le type des paramètres.
   (b) Le type de la valeur rendue.
   (c) comment on la calcule.
2. Un ou plusieurs *appels*: c'est l'utilisation de la fonction.
3. Zero, un ou plusieurs *paramètres*: ce sont les arguments de la fonction.
4. Un *type* et une *valeur*.

Exemples (sans paramètres)

```
fonction pi(): reel
  debut pi
      retourner 3.1415926535897931
  fin pi
```

Exemple avec paramètres

La fonction suivante rend le maximum de deux valeurs de type réél.

```
fonction fmax(valeur a: reel, valeur b: reel) : reel
variables m : un reel
debut fmax
 si (a > b)
 alors
   m <- a
 sinon
   m <- b
 fin si
 retourner m
fin fmax
```

de sorte que l'expression `fmax(pi()*pi(),10.0)` vaut 10.

Paramètres

En général, une fonction ou procédure n'est utile que si elle a des paramètres, comme dans le cas de `fmax`.

Une fonction ou procédure définie avec des paramètres doit être appelée avec des *arguments* (en nombre égal, et du même type!).

définition de fonction	appel de fonction
paramètre	argument


```
fonction fmax(valeur a: reel, valeur b: reel) : reel
...
programme essai
variables x: un reel
          i,j: deux caracteres
debut essai
   lire x
   ecrire fmax(x,x*x)
   /* ecrire fmax(i,j) serait incorrect! */
fin essai
```

Nature des paramètres: déclaration

Dans la declaration

```
fonction fmax(valeur a: reel, valeur b: reel) : reel
...
```

- `a` et `b` sont les paramètres de la fonction
- ils sont déclarés avec leur type
- dans le corps de la fonction, on peut utiliser `a` et `b` comme des variables,
  qui existent exclusivement dans le corps de la fonction
- le mot clé `valeur` indique une *modalité* de passage des paramètres

Nature des paramètres: passage par valeur

Après la déclaration

```
fonction fmax(valeur a: reel, valeur b: reel) : reel
...
```

on peut effectuer un appel de la fonction comme

```
ecrire 3*fmax(x,y)
```

l'expression `fmax(x,y)` est évaluée de la façon suivante:

- les valeurs de `x` et `y` sont calculées

- les paramètres `a` et `b` de `fmax` sont initialisés en utilisant ces valuers
- le corps de la fonction `fmax` est executé, (cela utilise les paramètres `a` et `b` comme des variables)
- le résultat de la fonction est enfin retourné et utilisé pour continuer le calcul de l'expression (ici, on le multiplie par 3 et on l'imprime)

Peu importe les opérations effectuées sur `a` *et* `b` *dans le corps de* `fmax`, *les valeurs des paramètres actuels* `x` *et* `y` *restent inchangés.*

Limite du passage par valeur

On a souvent besoin d'écrire des procédure ou fonctions que, au contraire, *modifient* la valeur de certains paramètres, par exemple, on voudrait écrire une procedure `echange` t.q. `echange(x,y)` échange les valeurs des variables `x` et `y`.

Mais la procédure

```
procédure echange (valeur a: entier, valeur b: entier)
variable t: un entier
debut echange
 t <- a
 a <- y
 b <- t
fin echange
```

n'échange pas les valeurs de ses arguments!
(relisez la description du passage d'arguments par valeur plus haut)

Comment faire?

Passage par référence

Si on écrit

```
procédure echange (reference a: entier, reference b: entier)
variable t: un entier
debut echange
 t <- a
 a <- y
 b <- t
fin echange
```


le passage des paramètres se fait autrement: au moment de l'appel

`echange(i,j)`

- *on ne fait pas une copie des valeurs de i et j dans a et b!*
- au contraire, on transforme a et b en "alias" de i et j, en le faisant pointer sur les memes cases mémoire de i et j
- donc, toute opération effectuée sur les paramètres a et b dans le corps de la procédure ou fonction et aussi effectué sur les arguments i et j

N.B.: on doit passer comme argument d'un paramètre par référence seulement des expressions qui se comportent comme des variables (par exemple, une variable, mais aussi une case d'un tableau etc.)

Résumé

- On déclare "valeur" un paramètre si on ne veut pas que l'appel modifie les arguments
- On déclare "reference" un paramètre si on veut que l'appel modifie les arguments

Bien sûr, on peut melanger paramètres par valeur et par référence.

ATTENTION:

- les tableaux sont *toujours* passé par référence, et *jamais* copiés, même si on déclare le paramètres correspondant "valeur".
  (Vous comprendrez les raisons de cela plus avant dans le cursus informatique).

Un exemple complet: le tri par séléction

Voyons comme écrire un programme de tri par séléction beaucoup plus lisible avec les fonctions et les procédures.

Fonction de recherche du minimum à partir d'un indice

```
fonction mintab(valeur i: un entier, reference a: tableau): entier
variable k: un entier
debut mintab
   pour k <- i+1 a  N faire
    si  (a[k] < a[i]) alors
        i <- k
    fin si
   fin pour
   retourner i
fin mintab
```

Programme de tri par séléction

```
constante N=10
type tab = tableau de N entiers
procedure echange(reference a, reference b) ...
fonction mintab(valeur i: un entier, reference a: tableau): entier ...

programme triselection
variables a: tab
debut triselection
lire a
  pour i <- 1 a N-1 faire
      echange(a[i],a[mintab(i,a)])
  fin pour
ecrire a
fin triselection
```


Et en C++?

```
#include <iostream.h>

const int N=10;

typedef int tab[N];

void echange (int &a, int &b)
{ int t;
  t=a; a=b; b=t;
}

int mintab(int i, tab &a)
{ int k;

  for(k=i+1;k<N;k++){if (a[k] < a[i]) {i =k;};};
  return(i);
}

void lire(tab &a)
{ int k;

  for (k=0;k<N;k++){cout<<"a["<<k<<"] ? "<<endl; cin>>a[k];};
}

void ecrire(tab &a)
{ int k;
 for (k=0;k<N;k++){cout<<"a["<<k<<"] = "<<a[k]<<endl;};
}

void main(){
 tab a;
 int i;

 lire(a);
 for (i=0;i<N-1;i++){echange(a[i],a[mintab(i,a)]);};
 ecrire(a);
}
```

=

Fonctions et procédures

Idée : capitaliser sur le code déjà écrit pour introduire des nouvelles "commandes" et "opérations".

On souhaite donner un nom programme que nous avons déjà écrit, et qui effectue une opération intéressante, pour pouvoir le réutiliser après sans le réécrire. Si ce programme retourne un résultat, on parlera de "fonction", et "procédure" sinon.

De plus, on peut souhaiter choisir les valeurs de certaines variables de la procédure ou fonction, et cela se fait à travers le mechanisme de passage des paramètres.

Procedures: déclaration et appel

Chaque procédure aura:

1. Une *définition* qui dit
   (a) Le type des paramètres.
   (b) comment on la calcule.
2. Un ou plusieurs *appels*: c'est l'utilisation de la procédure.

Exemples (sans paramètres)

Tous les programmes affichent un message d'aide à l'utilisation, et cela à plusieurs occasions

```
procedure  aide ()
  debut aide
        ecrire "Aide du programme Machin."
        ecrire " l'option -o permet de definir le fichier sortie"
        ecrire " l'option -h donne ce message"
  fin aide
...
programme Machin
debut Machin
```



```
        ...
        aide()
        ...
        si (erreur=vrai) alors aide() fin si
        ...
fin Machin
```

Exemples (avec paramètres)

Tous les programmes affichent des messages d'erreur

```
procedure  erreur (valeur s: chaîne de caractères)
  debut erreur
        ecrire "Programme Machin: on a renconter l'erreur" s
  fin erreur
...
programme Machin
debut Machin
        ...
        erreur("Entier trop grand")
        ...
        erreur("Pas assez de memoire")
        ...
fin Machin
```

Fonctions: déclaration et appel

Chaque fonction aura:

1. Une *définition* qui dit
   (a) Le type des paramètres.
   (b) Le type de la valeur rendue.
   (c) comment on la calcule.
2. Un ou plusieurs *appels*: c'est l'utilisation de la fonction.
3. Zero, un ou plusieurs *paramètres*: ce sont les arguments de la fonction.
4. Un *type* et une *valeur*.

Exemples (sans paramètres)

```
fonction pi(): reel
  debut pi
      retourner 3.1415926535897931
  fin pi
```

Exemple avec paramètres

La fonction suivante rend le maximum de deux valeurs de type réél.

```
fonction fmax(valeur a: reel, valeur b: reel) : reel
variables m : un reel
debut fmax
 si (a > b)
 alors
   m <- a
 sinon
   m <- b
 fin si
 retourner m
fin fmax
```

de sorte que l'expression `fmax(pi()*pi(),10.0)` vaut 10.

Paramètres

En général, une fonction ou procédure n'est utile que si elle a des paramètres, comme dans le cas de **fmax**.

Une fonction ou procédure définie avec des paramètres doit être appelée avec des *arguments* (en nombre égal, et du même type!).

définition de fonction	appel de fonction
paramètre	argument


```
fonction fmax(valeur a: reel, valeur b: reel) : reel
...
programme essai
variables x: un reel
          i,j: deux caracteres
debut essai
   lire x
   ecrire fmax(x,x*x)
   /* ecrire fmax(i,j) serait incorrect! */
fin essai
```

Nature des paramètres: déclaration

Dans la declaration

```
fonction fmax(valeur a: reel, valeur b: reel) : reel
...
```

- `a` et `b` sont les paramètres de la fonction
- ils sont déclarés avec leur type
- dans le corps de la fonction, on peut utiliser `a` et `b` comme des variables,
  qui existent exclusivement dans le corps de la fonction
- le mot clé `valeur` indique une *modalité* de passage des paramètres

Nature des paramètres: passage par valeur

Après la déclaration

```
fonction fmax(valeur a: reel, valeur b: reel) : reel
...
```

on peut effectuer un appel de la fonction comme

```
ecrire 3*fmax(x,y)
```

l'expression `fmax(x,y)` est évaluée de la façon suivante:

- les valeurs de `x` et `y` sont calculées

- les paramètres `a` et `b` de `fmax` sont initialisés en utilisant ces valuers
- le corps de la fonction `fmax` est executé, (cela utilise les paramètres `a` et `b` comme des variables)
- le résultat de la fonction est enfin retourné et utilisé pour continuer le calcul de l'expression (ici, on le multiplie par 3 et on l'imprime)

Peu importe les opérations effectuées sur `a` *et* `b` *dans le corps de* `fmax`*, les valeurs des paramètres actuels* `x` *et* `y` *restent inchangés.*

Limite du passage par valeur

On a souvent besoin d'écrire des procédure ou fonctions que, au contraire, *modifient* la valeur de certains paramètres, par exemple, on voudrait écrire une procedure `echange` t.q. `echange(x,y)` échange les valeurs des variables `x` et `y`.

Mais la procédure

```
procédure echange (valeur a: entier, valeur b: entier)
variable t: un entier
debut echange
 t <- a
 a <- y
 b <- t
fin echange
```

n'échange pas les valeurs de ses arguments!
(relisez la description du passage d'arguments par valeur plus haut)

Comment faire?

Passage par référence

Si on écrit

```
procédure echange (reference a: entier, reference b: entier)
variable t: un entier
debut echange
 t <- a
 a <- y
 b <- t
fin echange
```


le passage des paramètres se fait autrement: au moment de l'appel

`echange(i,j)`

- *on ne fait pas une copie des valeurs de i et j dans a et b!*
- au contraire, on transforme a et b en "alias" de i et j, en le faisant pointer sur les memes cases mémoire de i et j
- donc, toute opération effectuée sur les paramètres a et b dans le corps de la procédure ou fonction et aussi effectué sur les arguments i et j

N.B.: on doit passer comme argument d'un paramètre par référence seulement des expressions qui se comportent comme des variables (par exemple, une variable, mais aussi une case d'un tableau etc.)

Résumé

- On déclare "valeur" un paramètre si on ne veut pas que l'appel modifie les arguments
- On déclare "reference" un paramètre si on veut que l'appel modifie les arguments

Bien sûr, on peut melanger paramètres par valeur et par référence.

ATTENTION:

- les tableaux sont *toujours* passé par référence, et *jamais* copiés, même si on déclare le paramètres correspondant "valeur".
(Vous comprendrez les raisons de cela plus avant dans le cursus informatique).

Un exemple complet: le tri par séléction

Voyons comme écrire un programme de tri par séléction beaucoup plus lisible avec les fonctions et les procédures.

Fonction de recherche du minimum à partir d'un indice

```
fonction mintab(valeur i: un entier, reference a: tableau): entier
variable k: un entier
debut mintab
   pour k <- i+1 a  N faire
    si  (a[k] < a[i]) alors
        i <- k
    fin si
   fin pour
   retourner i
fin mintab
```

Programme de tri par séléction

```
constante N=10
type tab = tableau de N entiers
procedure echange(reference a, reference b) ...
fonction mintab(valeur i: un entier, reference a: tableau): entier ...

programme triselection
variables a: tab
debut triselection
lire a
  pour i <- 1 a N-1 faire
      echange(a[i],a[mintab(i,a)])
  fin pour
ecrire a
fin triselection
```


Et en C++?

```
#include <iostream.h>

const int N=10;

typedef int tab[N];

void echange (int &a, int &b)
{ int t;
  t=a; a=b; b=t;
}

int mintab(int i, tab &a)
{ int k;

  for(k=i+1;k<N;k++){if (a[k] < a[i]) {i =k;};};
  return(i);
}

void lire(tab &a)
{ int k;

  for (k=0;k<N;k++){cout<<"a["<<k<<"] ? "<<endl; cin>>a[k];};
}

void ecrire(tab &a)
{ int k;
 for (k=0;k<N;k++){cout<<"a["<<k<<"] = "<<a[k]<<endl;};
}

void main(){
 tab a;
 int i;

 lire(a);
 for (i=0;i<N-1;i++){echange(a[i],a[mintab(i,a)]);};
 ecrire(a);
}
```

=

Fonctions et procédures

Idée : capitaliser sur le code déjà écrit pour introduire des nouvelles "commandes" et "opérations".

On souhaite donner un nom programme que nous avons déjà écrit, et qui effectue une opération intéressante, pour pouvoir le réutiliser après sans le réécrire. Si ce programme retourne un résultat, on parlera de "fonction", et "procédure" sinon.

De plus, on peut souhaiter choisir les valeurs de certaines variables de la procédure ou fonction, et cela se fait à travers le mechanisme de passage des paramètres.

Procedures: déclaration et appel

Chaque procédure aura:

1. Une *définition* qui dit
   (a) Le type des paramètres.
   (b) comment on la calcule.
2. Un ou plusieurs *appels*: c'est l'utilisation de la procédure.

Exemples (sans paramètres)

Tous les programmes affichent un message d'aide à l'utilisation, et cela à plusieurs occasions

```
procedure  aide ()
  debut aide
        ecrire "Aide du programme Machin."
        ecrire " l'option -o permet de definir le fichier sortie"
        ecrire " l'option -h donne ce message"
  fin aide
...
programme Machin
debut Machin
```



```
        ...
        aide()
        ...
        si (erreur=vrai) alors aide() fin si
        ...
fin Machin
```

Exemples (avec paramètres)

Tous les programmes affichent des messages d'erreur

```
procedure  erreur (valeur s: chaîne de caractères)
  debut erreur
        ecrire "Programme Machin: on a renconter l'erreur" s
  fin erreur
...
programme Machin
debut Machin
        ...
        erreur("Entier trop grand")
        ...
        erreur("Pas assez de memoire")
        ...
fin Machin
```

Fonctions: déclaration et appel

Chaque fonction aura:

1. Une *définition* qui dit
   (a) Le type des paramètres.
   (b) Le type de la valeur rendue.
   (c) comment on la calcule.
2. Un ou plusieurs *appels*: c'est l'utilisation de la fonction.
3. Zero, un ou plusieurs *paramètres*: ce sont les arguments de la fonction.
4. Un *type* et une *valeur*.

Exemples (sans paramètres)

```
fonction pi(): reel
  debut pi
      retourner 3.1415926535897931
  fin pi
```

Exemple avec paramètres

La fonction suivante rend le maximum de deux valeurs de type réél.

```
fonction fmax(valeur a: reel, valeur b: reel) : reel
variables m : un reel
debut fmax
 si (a > b)
 alors
   m <- a
 sinon
   m <- b
 fin si
 retourner m
fin fmax
```

de sorte que l'expression `fmax(pi()*pi(),10.0)` vaut 10.

Paramètres

En général, une fonction ou procédure n'est utile que si elle a des paramètres, comme dans le cas de `fmax`.

Une fonction ou procédure définie avec des paramètres doit être appelée avec des *arguments* (en nombre égal, et du même type!).

définition de fonction	appel de fonction
paramètre	argument


```
fonction fmax(valeur a: reel, valeur b: reel) : reel
...
programme essai
variables x: un reel
          i,j: deux caracteres
debut essai
   lire x
   ecrire fmax(x,x*x)
   /* ecrire fmax(i,j) serait incorrect! */
fin essai
```

Nature des paramètres: déclaration

Dans la declaration

```
fonction fmax(valeur a: reel, valeur b: reel) : reel
...
```

- `a` et `b` sont les paramètres de la fonction
- ils sont déclarés avec leur type
- dans le corps de la fonction, on peut utiliser `a` et `b` comme des variables,
  qui existent exclusivement dans le corps de la fonction
- le mot clé `valeur` indique une *modalité* de passage des paramètres

Nature des paramètres: passage par valeur

Après la déclaration

```
fonction fmax(valeur a: reel, valeur b: reel) : reel
...
```

on peut effectuer un appel de la fonction comme

```
ecrire 3*fmax(x,y)
```

l'expression `fmax(x,y)` est évaluée de la façon suivante:

- les valeurs de `x` et `y` sont calculées

- les paramètres `a` et `b` de `fmax` sont initialisés en utilisant ces valuers
- le corps de la fonction `fmax` est executé, (cela utilise les paramètres `a` et `b` comme des variables)
- le résultat de la fonction est enfin retourné et utilisé pour continuer le calcul de l'expression (ici, on le multiplie par 3 et on l'imprime)

Peu importe les opérations effectuées sur `a` *et* `b` *dans le corps de* `fmax`, *les valeurs des paramètres actuels* `x` *et* `y` *restent inchangés.*

Limite du passage par valeur

On a souvent besoin d'écrire des procédure ou fonctions que, au contraire, *modifient* la valeur de certains paramètres, par exemple, on voudrait écrire une procedure `echange` t.q. `echange(x,y)` échange les valeurs des variables `x` et `y`.

Mais la procédure

```
procédure echange (valeur a: entier, valeur b: entier)
variable t: un entier
debut echange
 t <- a
 a <- y
 b <- t
fin echange
```

n'échange pas les valeurs de ses arguments!
(relisez la description du passage d'arguments par valeur plus haut)

Comment faire?

Passage par référence

Si on écrit

```
procédure echange (reference a: entier, reference b: entier)
variable t: un entier
debut echange
 t <- a
 a <- y
 b <- t
fin echange
```


le passage des paramètres se fait autrement: au moment de l'appel

`echange(i,j)`

- *on ne fait pas une copie des valeurs de i et j dans a et b!*
- au contraire, on transforme a et b en "alias" de i et j, en le faisant pointer sur les memes cases mémoire de i et j
- donc, toute opération effectuée sur les paramètres a et b dans le corps de la procédure ou fonction et aussi effectué sur les arguments i et j

N.B.: on doit passer comme argument d'un paramètre par référence seulement des expressions qui se comportent comme des variables (par exemple, une variable, mais aussi une case d'un tableau etc.)

Résumé

- On déclare "valeur" un paramètre si on ne veut pas que l'appel modifie les arguments
- On déclare "reference" un paramètre si on veut que l'appel modifie les arguments

Bien sûr, on peut melanger paramètres par valeur et par référence.

ATTENTION:

- les tableaux sont *toujours* passé par référence, et *jamais* copiés, même si on déclare le paramètres correspondant "valeur".
(Vous comprendrez les raisons de cela plus avant dans le cursus informatique).

Un exemple complet: le tri par séléction

Voyons comme écrire un programme de tri par séléction beaucoup plus lisible avec les fonctions et les procédures.

Fonction de recherche du minimum à partir d'un indice

```
fonction mintab(valeur i: un entier, reference a: tableau): entier
variable k: un entier
debut mintab
   pour k <- i+1 a  N faire
    si  (a[k] < a[i]) alors
        i <- k
    fin si
   fin pour
   retourner i
fin mintab
```

Programme de tri par séléction

```
constante N=10
type tab = tableau de N entiers
procedure echange(reference a, reference b) ...
fonction mintab(valeur i: un entier, reference a: tableau): entier ...

programme triselection
variables a: tab
debut triselection
lire a
  pour i <- 1 a N-1 faire
      echange(a[i],a[mintab(i,a)])
  fin pour
ecrire a
fin triselection
```


Et en C++?

```
#include <iostream.h>

const int N=10;

typedef int tab[N];

void echange (int &a, int &b)
{ int t;
  t=a; a=b; b=t;
}

int mintab(int i, tab &a)
{ int k;

  for(k=i+1;k<N;k++){if (a[k] < a[i]) {i =k;};};
  return(i);
}

void lire(tab &a)
{ int k;

  for (k=0;k<N;k++){cout<<"a["<<k<<"] ? "<<endl; cin>>a[k];};
}

void ecrire(tab &a)
{ int k;
 for (k=0;k<N;k++){cout<<"a["<<k<<"] = "<<a[k]<<endl;};
}

void main(){
 tab a;
 int i;

 lire(a);
 for (i=0;i<N-1;i++){echange(a[i],a[mintab(i,a)]);};
 ecrire(a);
}
```

=

Fonctions et procédures

Idée : capitaliser sur le code déjà écrit pour introduire des nouvelles "commandes" et "opérations".

On souhaite donner un nom programme que nous avons déjà écrit, et qui effectue une opération intéressante, pour pouvoir le réutiliser après sans le réécrire. Si ce programme retourne un résultat, on parlera de "fonction", et "procédure" sinon.

De plus, on peut souhaiter choisir les valeurs de certaines variables de la procédure ou fonction, et cela se fait à travers le mechanisme de passage des paramètres.

Procedures: déclaration et appel

Chaque procédure aura:

1. Une *définition* qui dit
   (a) Le type des paramètres.
   (b) comment on la calcule.
2. Un ou plusieurs *appels*: c'est l'utilisation de la procédure.

Exemples (sans paramètres)

Tous les programmes affichent un message d'aide à l'utilisation, et cela à plusieurs occasions

```
procedure  aide ()
  debut aide
        ecrire "Aide du programme Machin."
        ecrire " l'option -o permet de definir le fichier sortie"
        ecrire " l'option -h donne ce message"
  fin aide
...
programme Machin
debut Machin
```



```
        ...
        aide()
        ...
        si (erreur=vrai) alors aide() fin si
        ...
fin Machin
```

Exemples (avec paramètres)

Tous les programmes affichent des messages d'erreur

```
procedure  erreur (valeur s: chaîne de caractères)
  debut erreur
        ecrire "Programme Machin: on a rencontrer l'erreur" s
  fin erreur
...
programme Machin
debut Machin
        ...
        erreur("Entier trop grand")
        ...
        erreur("Pas assez de memoire")
        ...
fin Machin
```

Fonctions: déclaration et appel

Chaque fonction aura:

1. Une *définition* qui dit
   (a) Le type des paramètres.
   (b) Le type de la valeur rendue.
   (c) comment on la calcule.
2. Un ou plusieurs *appels*: c'est l'utilisation de la fonction.
3. Zero, un ou plusieurs *paramètres*: ce sont les arguments de la fonction.
4. Un *type* et une *valeur*.

Exemples (sans paramètres)

```
fonction pi(): reel
  debut pi
      retourner 3.1415926535897931
  fin pi
```

Exemple avec paramètres

La fonction suivante rend le maximum de deux valeurs de type réél.

```
fonction fmax(valeur a: reel, valeur b: reel) : reel
variables m : un reel
debut fmax
 si (a > b)
 alors
   m <- a
 sinon
   m <- b
 fin si
 retourner m
fin fmax
```

de sorte que l'expression `fmax(pi()*pi(),10.0)` vaut 10.

Paramètres

En général, une fonction ou procédure n'est utile que si elle a des paramètres, comme dans le cas de `fmax`.

Une fonction ou procédure définie avec des paramètres doit être appelée avec des *arguments* (en nombre égal, et du même type!).

définition de fonction	appel de fonction
paramètre	argument


```
fonction fmax(valeur a: reel, valeur b: reel) : reel
...
programme essai
variables x: un reel
          i,j: deux caracteres
debut essai
   lire x
   ecrire fmax(x,x*x)
   /* ecrire fmax(i,j) serait incorrect! */
fin essai
```

Nature des paramètres: déclaration

Dans la declaration

```
fonction fmax(valeur a: reel, valeur b: reel) : reel
...
```

- `a` et `b` sont les paramètres de la fonction
- ils sont déclarés avec leur type
- dans le corps de la fonction, on peut utiliser `a` et `b` comme des variables,
  qui existent exclusivement dans le corps de la fonction
- le mot clé `valeur` indique une *modalité* de passage des paramètres

Nature des paramètres: passage par valeur

Après la déclaration

```
fonction fmax(valeur a: reel, valeur b: reel) : reel
...
```

on peut effectuer un appel de la fonction comme

```
ecrire 3*fmax(x,y)
```

l'expression `fmax(x,y)` est évaluée de la façon suivante:

- les valeurs de `x` et `y` sont calculées

- les paramètres `a` et `b` de `fmax` sont initialisés en utilisant ces valuers
- le corps de la fonction `fmax` est executé, (cela utilise les paramètres `a` et `b` comme des variables)
- le résultat de la fonction est enfin retourné et utilisé pour continuer le calcul de l'expression (ici, on le multiplie par 3 et on l'imprime)

Peu importe les opérations effectuées sur `a` *et* `b` *dans le corps de* `fmax`, *les valeurs des paramètres actuels* `x` *et* `y` *restent inchangés.*

Limite du passage par valeur

On a souvent besoin d'écrire des procédure ou fonctions que, au contraire, *modifient* la valeur de certains paramètres, par exemple, on voudrait écrire une procedure `echange` t.q. `echange(x,y)` échange les valeurs des variables `x` et `y`.

Mais la procédure

```
procédure echange (valeur a: entier, valeur b: entier)
variable t: un entier
debut echange
 t <- a
 a <- y
 b <- t
fin echange
```

n'échange pas les valeurs de ses arguments!
(relisez la description du passage d'arguments par valeur plus haut)

Comment faire?

Passage par référence

Si on écrit

```
procédure echange (reference a: entier, reference b: entier)
variable t: un entier
debut echange
 t <- a
 a <- y
 b <- t
fin echange
```


le passage des paramètres se fait autrement: au moment de l'appel

`echange(i,j)`

- *on ne fait pas une copie des valeurs de i et j dans a et b!*
- au contraire, on transforme a et b en "alias" de i et j, en le faisant pointer sur les memes cases mémoire de i et j
- donc, toute opération effectuée sur les paramètres a et b dans le corps de la procédure ou fonction et aussi effectué sur les arguments i et j

N.B.: on doit passer comme argument d'un paramètre par référence seulement des expressions qui se comportent comme des variables (par exemple, une variable, mais aussi une case d'un tableau etc.)

Résumé

- On déclare "valeur" un paramètre si on ne veut pas que l'appel modifie les arguments
- On déclare "reference" un paramètre si on veut que l'appel modifie les arguments

Bien sûr, on peut melanger paramètres par valeur et par référence.

ATTENTION:

- les tableaux sont *toujours* passé par référence, et *jamais* copiés, même si on déclare le paramètres correspondant "valeur".
  (Vous comprendrez les raisons de cela plus avant dans le cursus informatique).

Un exemple complet: le tri par séléction

Voyons comme écrire un programme de tri par séléction beaucoup plus lisible avec les fonctions et les procédures.

Fonction de recherche du minimum à partir d'un indice

```
fonction mintab(valeur i: un entier, reference a: tableau): entier
variable k: un entier
debut mintab
   pour k <- i+1 a  N faire
    si  (a[k] < a[i]) alors
        i <- k
    fin si
   fin pour
   retourner i
fin mintab
```

Programme de tri par séléction

```
constante N=10
type tab = tableau de N entiers
procedure echange(reference a, reference b) ...
fonction mintab(valeur i: un entier, reference a: tableau): entier ...

programme triselection
variables a: tab
debut triselection
lire a
  pour i <- 1 a N-1 faire
      echange(a[i],a[mintab(i,a)])
  fin pour
ecrire a
fin triselection
```


Et en C++?

```
#include <iostream.h>

const int N=10;

typedef int tab[N];

void echange (int &a, int &b)
{ int t;
  t=a; a=b; b=t;
}

int mintab(int i, tab &a)
{ int k;

  for(k=i+1;k<N;k++){if (a[k] < a[i]) {i =k;};};
  return(i);
}

void lire(tab &a)
{ int k;

  for (k=0;k<N;k++){cout<<"a["<<k<<"] ? "<<endl; cin>>a[k];};
}

void ecrire(tab &a)
{ int k;
 for (k=0;k<N;k++){cout<<"a["<<k<<"] = "<<a[k]<<endl;};
}

void main(){
 tab a;
 int i;

 lire(a);
 for (i=0;i<N-1;i++){echange(a[i],a[mintab(i,a)]);};
 ecrire(a);
}
```

=

Fonctions et procédures

Idée : capitaliser sur le code déjà écrit pour introduire des nouvelles "commandes" et "opérations".

On souhaite donner un nom programme que nous avons déjà écrit, et qui effectue une opération intéressante, pour pouvoir le réutiliser après sans le réécrire. Si ce programme retourne un résultat, on parlera de "fonction", et "procédure" sinon.

De plus, on peut souhaiter choisir les valeurs de certaines variables de la procédure ou fonction, et cela se fait à travers le mechanisme de passage des paramètres.

Procedures: déclaration et appel

Chaque procédure aura:

1. Une *définition* qui dit
   (a) Le type des paramètres.
   (b) comment on la calcule.
2. Un ou plusieurs *appels*: c'est l'utilisation de la procédure.

Exemples (sans paramètres)

Tous les programmes affichent un message d'aide à l'utilisation, et cela à plusieurs occasions

```
procedure  aide ()
  debut aide
        ecrire "Aide du programme Machin."
        ecrire " l'option -o permet de definir le fichier sortie"
        ecrire " l'option -h donne ce message"
  fin aide
...
programme Machin
debut Machin
```



```
        ...
        aide()
        ...
        si (erreur=vrai) alors aide() fin si
        ...
fin Machin
```

Exemples (avec paramètres)

Tous les programmes affichent des messages d'erreur

```
procedure  erreur (valeur s: chaîne de caractères)
  debut erreur
        ecrire "Programme Machin: on a renconter l'erreur" s
  fin erreur
...
programme Machin
debut Machin
        ...
        erreur("Entier trop grand")
        ...
        erreur("Pas assez de memoire")
        ...
fin Machin
```

Fonctions: déclaration et appel

Chaque fonction aura:

1. Une *définition* qui dit
   (a) Le type des paramètres.
   (b) Le type de la valeur rendue.
   (c) comment on la calcule.
2. Un ou plusieurs *appels*: c'est l'utilisation de la fonction.
3. Zero, un ou plusieurs *paramètres*: ce sont les arguments de la fonction.
4. Un *type* et une *valeur*.

Exemples (sans paramètres)

```
fonction pi(): reel
  debut pi
      retourner 3.1415926535897931
  fin pi
```

Exemple avec paramètres

La fonction suivante rend le maximum de deux valeurs de type réél.

```
fonction fmax(valeur a: reel, valeur b: reel) : reel
variables m : un reel
debut fmax
 si (a > b)
 alors
   m <- a
 sinon
   m <- b
 fin si
 retourner m
fin fmax
```

de sorte que l'expression `fmax(pi()*pi(),10.0)` vaut 10.

Paramètres

En général, une fonction ou procédure n'est utile que si elle a des paramètres, comme dans le cas de `fmax`.

Une fonction ou procédure définie avec des paramètres doit être appelée avec des *arguments* (en nombre égal, et du même type!).

définition de fonction	appel de fonction
paramètre	argument


```
fonction fmax(valeur a: reel, valeur b: reel) : reel
...
programme essai
variables x: un reel
          i,j: deux caracteres
debut essai
   lire x
   ecrire fmax(x,x*x)
   /* ecrire fmax(i,j) serait incorrect! */
fin essai
```

Nature des paramètres: déclaration

Dans la declaration

```
fonction fmax(valeur a: reel, valeur b: reel) : reel
...
```

- `a` et `b` sont les paramètres de la fonction
- ils sont déclarés avec leur type
- dans le corps de la fonction, on peut utiliser `a` et `b` comme des variables, *qui existent exclusivement dans le corps de la fonction*
- le mot clé `valeur` indique une *modalité* de passage des paramètres

Nature des paramètres: passage par valeur

Après la déclaration

```
fonction fmax(valeur a: reel, valeur b: reel) : reel
...
```

on peut effectuer un appel de la fonction comme

```
ecrire 3*fmax(x,y)
```

l'expression `fmax(x,y)` est évaluée de la façon suivante:

- les valeurs de `x` et `y` sont calculées

- les paramètres `a` et `b` de `fmax` sont initialisés en utilisant ces valuers
- le corps de la fonction `fmax` est executé, (cela utilise les paramètres `a` et `b` comme des variables)
- le résultat de la fonction est enfin retourné et utilisé pour continuer le calcul de l'expression (ici, on le multiplie par 3 et on l'imprime)

Peu importe les opérations effectuées sur `a` *et* `b` *dans le corps de* `fmax`, *les valeurs des paramètres actuels* `x` *et* `y` *restent inchangés.*

Limite du passage par valeur

On a souvent besoin d'écrire des procédure ou fonctions que, au contraire, *modifient* la valeur de certains paramètres, par exemple, on voudrait écrire une procedure `echange` t.q. `echange(x,y)` échange les valeurs des variables `x` et `y`.

Mais la procédure

```
procédure echange (valeur a: entier, valeur b: entier)
variable t: un entier
debut echange
 t <- a
 a <- y
 b <- t
fin echange
```

n'échange pas les valeurs de ses arguments!
(relisez la description du passage d'arguments par valeur plus haut)

Comment faire?

Passage par référence

Si on écrit

```
procédure echange (reference a: entier, reference b: entier)
variable t: un entier
debut echange
 t <- a
 a <- y
 b <- t
fin echange
```


le passage des paramètres se fait autrement: au moment de l'appel

`echange(i,j)`

- *on ne fait pas une copie des valeurs de i et j dans a et b!*
- au contraire, on transforme a et b en "alias" de i et j, en le faisant pointer sur les memes cases mémoire de i et j
- donc, toute opération effectuée sur les paramètres a et b dans le corps de la procédure ou fonction et aussi effectué sur les arguments i et j

N.B.: on doit passer comme argument d'un paramètre par référence seulement des expressions qui se comportent comme des variables (par exemple, une variable, mais aussi une case d'un tableau etc.)

Résumé

- On déclare "valeur" un paramètre si on ne veut pas que l'appel modifie les arguments
- On déclare "reference" un paramètre si on veut que l'appel modifie les arguments

Bien sûr, on peut melanger paramètres par valeur et par référence.

ATTENTION:

- les tableaux sont *toujours* passé par référence, et *jamais* copiés, même si on déclare le paramètres correspondant "valeur".
(Vous comprendrez les raisons de cela plus avant dans le cursus informatique).

Un exemple complet: le tri par séléction

Voyons comme écrire un programme de tri par séléction beaucoup plus lisible avec les fonctions et les procédures.

Fonction de recherche du minimum à partir d'un indice

```
fonction mintab(valeur i: un entier, reference a: tableau): entier
variable k: un entier
debut mintab
   pour k <- i+1 a  N faire
    si  (a[k] < a[i]) alors
        i <- k
    fin si
   fin pour
   retourner i
fin mintab
```

Programme de tri par séléction

```
constante N=10
type tab = tableau de N entiers
procedure echange(reference a, reference b) ...
fonction mintab(valeur i: un entier, reference a: tableau): entier ...

programme triselection
variables a: tab
debut triselection
lire a
  pour i <- 1 a N-1 faire
      echange(a[i],a[mintab(i,a)])
  fin pour
ecrire a
fin triselection
```


Et en C++?

```
#include <iostream.h>

const int N=10;

typedef int tab[N];

void echange (int &a, int &b)
{ int t;
  t=a; a=b; b=t;
}

int mintab(int i, tab &a)
{ int k;

  for(k=i+1;k<N;k++){if  (a[k] < a[i]) {i =k;};};
  return(i);
}

void lire(tab &a)
{ int k;

  for (k=0;k<N;k++){cout<<"a["<<k<<"] ? "<<endl; cin>>a[k];};
}

void ecrire(tab &a)
{ int k;
 for (k=0;k<N;k++){cout<<"a["<<k<<"] = "<<a[k]<<endl;};
}

void main(){
 tab a;
 int i;

 lire(a);
 for (i=0;i<N-1;i++){echange(a[i],a[mintab(i,a)]);};
 ecrire(a);
}
```

=

Fonctions et procédures

Idée : capitaliser sur le code déjà écrit pour introduire des nouvelles "commandes" et "opérations".

On souhaite donner un nom programme que nous avons déjà écrit, et qui effectue une opération intéressante, pour pouvoir le réutiliser après sans le réécrire. Si ce programme retourne un résultat, on parlera de "fonction", et "procédure" sinon.

De plus, on peut souhaiter choisir les valeurs de certaines variables de la procédure ou fonction, et cela se fait à travers le mechanisme de passage des paramètres.

Procedures: déclaration et appel

Chaque procédure aura:

1. Une *définition* qui dit
   (a) Le type des paramètres.
   (b) comment on la calcule.
2. Un ou plusieurs *appels*: c'est l'utilisation de la procédure.

Exemples (sans paramètres)

Tous les programmes affichent un message d'aide à l'utilisation, et cela à plusieurs occasions

```
procedure  aide ()
  debut aide
        ecrire "Aide du programme Machin."
        ecrire " l'option -o permet de definir le fichier sortie"
        ecrire " l'option -h donne ce message"
  fin aide
...
programme Machin
debut Machin
```



```
        ...
        aide()
        ...
        si (erreur=vrai) alors aide() fin si
        ...
fin Machin
```

Exemples (avec paramètres)

Tous les programmes affichent des messages d'erreur

```
procedure  erreur (valeur s: chaîne de caractères)
  debut erreur
        ecrire "Programme Machin: on a renconter l'erreur" s
  fin erreur
...
programme Machin
debut Machin
        ...
        erreur("Entier trop grand")
        ...
        erreur("Pas assez de memoire")
        ...
fin Machin
```

Fonctions: déclaration et appel

Chaque fonction aura:

1. Une *définition* qui dit
   (a) Le type des paramètres.
   (b) Le type de la valeur rendue.
   (c) comment on la calcule.
2. Un ou plusieurs *appels*: c'est l'utilisation de la fonction.
3. Zero, un ou plusieurs *paramètres*: ce sont les arguments de la fonction.
4. Un *type* et une *valeur*.

Exemples (sans paramètres)

```
fonction pi(): reel
  debut pi
      retourner 3.1415926535897931
  fin pi
```

Exemple avec paramètres

La fonction suivante rend le maximum de deux valeurs de type réél.

```
fonction fmax(valeur a: reel, valeur b: reel) : reel
variables m : un reel
debut fmax
 si (a > b)
 alors
   m <- a
 sinon
   m <- b
 fin si
 retourner m
fin fmax
```

de sorte que l'expression `fmax(pi()*pi(),10.0)` vaut 10.

Paramètres

En général, une fonction ou procédure n'est utile que si elle a des paramètres, comme dans le cas de `fmax`.

Une fonction ou procédure définie avec des paramètres doit être appelée avec des *arguments* (en nombre égal, et du même type!).

définition de fonction	appel de fonction
paramètre	argument


```
fonction fmax(valeur a: reel, valeur b: reel) : reel
...
programme essai
variables x: un reel
          i,j: deux caracteres
debut essai
   lire x
   ecrire fmax(x,x*x)
   /* ecrire fmax(i,j) serait incorrect! */
fin essai
```

Nature des paramètres: déclaration

Dans la declaration

```
fonction fmax(valeur a: reel, valeur b: reel) : reel
...
```

- `a` et `b` sont les paramètres de la fonction
- ils sont déclarés avec leur type
- dans le corps de la fonction, on peut utiliser `a` et `b` comme des variables,
  qui existent exclusivement dans le corps de la fonction
- le mot clé `valeur` indique une *modalité* de passage des paramètres

Nature des paramètres: passage par valeur

Après la déclaration

```
fonction fmax(valeur a: reel, valeur b: reel) : reel
...
```

on peut effectuer un appel de la fonction comme

```
ecrire 3*fmax(x,y)
```

l'expression `fmax(x,y)` est évaluée de la façon suivante:

- les valeurs de `x` et `y` sont calculées

- les paramètres `a` et `b` de `fmax` sont initialisés en utilisant ces valuers
- le corps de la fonction `fmax` est executé, (cela utilise les paramètres `a` et `b` comme des variables)
- le résultat de la fonction est enfin retourné et utilisé pour continuer le calcul de l'expression (ici, on le multiplie par 3 et on l'imprime)

Peu importe les opérations effectuées sur `a` *et* `b` *dans le corps de* `fmax`, *les valeurs des paramètres actuels* `x` *et* `y` *restent inchangés.*

Limite du passage par valeur

On a souvent besoin d'écrire des procédure ou fonctions que, au contraire, *modifient* la valeur de certains paramètres, par exemple, on voudrait écrire une procedure `echange` t.q. `echange(x,y)` échange les valeurs des variables `x` et `y`.

Mais la procédure

```
procédure echange (valeur a: entier, valeur b: entier)
variable t: un entier
debut echange
 t <- a
 a <- y
 b <- t
fin echange
```

n'échange pas les valeurs de ses arguments!
(relisez la description du passage d'arguments par valeur plus haut)

Comment faire?

Passage par référence

Si on écrit

```
procédure echange (reference a: entier, reference b: entier)
variable t: un entier
debut echange
 t <- a
 a <- y
 b <- t
fin echange
```


le passage des paramètres se fait autrement: au moment de l'appel

`echange(i,j)`

- *on ne fait pas une copie des valeurs de i et j dans a et b!*
- au contraire, on transforme a et b en "alias" de i et j, en le faisant pointer sur les memes cases mémoire de i et j
- donc, toute opération effectuée sur les paramètres a et b dans le corps de la procédure ou fonction et aussi effectué sur les arguments i et j

N.B.: on doit passer comme argument d'un paramètre par référence seulement des expressions qui se comportent comme des variables (par exemple, une variable, mais aussi une case d'un tableau etc.)

Résumé

- On déclare "valeur" un paramètre si on ne veut pas que l'appel modifie les arguments
- On déclare "reference" un paramètre si on veut que l'appel modifie les arguments

Bien sûr, on peut melanger paramètres par valeur et par référence.

ATTENTION:

- les tableaux sont *toujours* passé par référence, et *jamais* copiés, même si on déclare le paramètres correspondant "valeur".
  (Vous comprendrez les raisons de cela plus avant dans le cursus informatique).

Un exemple complet: le tri par séléction

Voyons comme écrire un programme de tri par séléction beaucoup plus lisible avec les fonctions et les procédures.

Fonction de recherche du minimum à partir d'un indice

```
fonction mintab(valeur i: un entier, reference a: tableau): entier
variable k: un entier
debut mintab
   pour k <- i+1 a  N faire
    si  (a[k] < a[i]) alors
        i <- k
    fin si
   fin pour
   retourner i
fin mintab
```

Programme de tri par séléction

```
constante N=10
type tab = tableau de N entiers
procedure echange(reference a, reference b) ...
fonction mintab(valeur i: un entier, reference a: tableau): entier ...

programme triselection
variables a: tab
debut triselection
lire a
  pour i <- 1 a N-1 faire
      echange(a[i],a[mintab(i,a)])
  fin pour
ecrire a
fin triselection
```


Et en C++?

```
#include <iostream.h>

const int N=10;

typedef int tab[N];

void echange (int &a, int &b)
{ int t;
  t=a; a=b; b=t;
}

int mintab(int i, tab &a)
{ int k;

  for(k=i+1;k<N;k++){if  (a[k] < a[i]) {i =k;};};
  return(i);
}

void lire(tab &a)
{ int k;

  for (k=0;k<N;k++){cout<<"a["<<k<<"] ? "<<endl; cin>>a[k];};
}

void ecrire(tab &a)
{ int k;
 for (k=0;k<N;k++){cout<<"a["<<k<<"] = "<<a[k]<<endl;};
}

void main(){
 tab a;
 int i;

 lire(a);
 for (i=0;i<N-1;i++){echange(a[i],a[mintab(i,a)]);};
 ecrire(a);
}
```

=

Fonctions et procédures

Idée : capitaliser sur le code déjà écrit pour introduire des nouvelles "commandes" et "opérations".

On souhaite donner un nom programme que nous avons déjà écrit, et qui effectue une opération intéressante, pour pouvoir le réutiliser après sans le réécrire. Si ce programme retourne un résultat, on parlera de "fonction", et "procédure" sinon.

De plus, on peut souhaiter choisir les valeurs de certaines variables de la procédure ou fonction, et cela se fait à travers le mechanisme de passage des paramètres.

Procedures: déclaration et appel

Chaque procédure aura:

1. Une *définition* qui dit
   (a) Le type des paramètres.
   (b) comment on la calcule.
2. Un ou plusieurs *appels*: c'est l'utilisation de la procédure.

Exemples (sans paramètres)

Tous les programmes affichent un message d'aide à l'utilisation, et cela à plusieurs occasions

```
procedure  aide ()
  debut aide
        ecrire "Aide du programme Machin."
        ecrire " l'option -o permet de definir le fichier sortie"
        ecrire " l'option -h donne ce message"
  fin aide
...
programme Machin
debut Machin
```



```
        ...
        aide()
        ...
        si (erreur=vrai) alors aide() fin si
        ...
fin Machin
```

Exemples (avec paramètres)

Tous les programmes affichent des messages d'erreur

```
procedure  erreur (valeur s: chaîne de caractères)
  debut erreur
        ecrire "Programme Machin: on a renconter l'erreur" s
  fin erreur
...
programme Machin
debut Machin
        ...
        erreur("Entier trop grand")
        ...
        erreur("Pas assez de memoire")
        ...
fin Machin
```

Fonctions: déclaration et appel

Chaque fonction aura:

1. Une *définition* qui dit
   (a) Le type des paramètres.
   (b) Le type de la valeur rendue.
   (c) comment on la calcule.
2. Un ou plusieurs *appels*: c'est l'utilisation de la fonction.
3. Zero, un ou plusieurs *paramètres*: ce sont les arguments de la fonction.
4. Un *type* et une *valeur*.

Exemples (sans paramètres)

```
fonction pi(): reel
  debut pi
      retourner 3.1415926535897931
  fin pi
```

Exemple avec paramètres

La fonction suivante rend le maximum de deux valeurs de type réél.

```
fonction fmax(valeur a: reel, valeur b: reel) : reel
variables m : un reel
debut fmax
 si (a > b)
 alors
   m <- a
 sinon
   m <- b
 fin si
 retourner m
fin fmax
```

de sorte que l'expression `fmax(pi()*pi(),10.0)` vaut 10.

Paramètres

En général, une fonction ou procédure n'est utile que si elle a des paramètres, comme dans le cas de `fmax`.

Une fonction ou procédure définie avec des paramètres doit être appelée avec des *arguments* (en nombre égal, et du même type!).

définition de fonction	appel de fonction
paramètre	argument


```
fonction fmax(valeur a: reel, valeur b: reel) : reel
...
programme essai
variables x: un reel
          i,j: deux caracteres
debut essai
   lire x
   ecrire fmax(x,x*x)
   /* ecrire fmax(i,j) serait incorrect! */
fin essai
```

Nature des paramètres: déclaration

Dans la declaration

```
fonction fmax(valeur a: reel, valeur b: reel) : reel
...
```

- `a` et `b` sont les paramètres de la fonction
- ils sont déclarés avec leur type
- dans le corps de la fonction, on peut utiliser `a` et `b` comme des variables,
  qui existent exclusivement dans le corps de la fonction
- le mot clé `valeur` indique une *modalité* de passage des paramètres

Nature des paramètres: passage par valeur

Après la déclaration

```
fonction fmax(valeur a: reel, valeur b: reel) : reel
...
```

on peut effectuer un appel de la fonction comme

```
ecrire 3*fmax(x,y)
```

l'expression `fmax(x,y)` est évaluée de la façon suivante:

- les valeurs de `x` et `y` sont calculées

- les paramètres `a` et `b` de `fmax` sont initialisés en utilisant ces valuers
- le corps de la fonction `fmax` est executé, (cela utilise les paramètres `a` et `b` comme des variables)
- le résultat de la fonction est enfin retourné et utilisé pour continuer le calcul de l'expression (ici, on le multiplie par 3 et on l'imprime)

Peu importe les opérations effectuées sur `a` *et* `b` *dans le corps de* `fmax`, *les valeurs des paramètres actuels* `x` *et* `y` *restent inchangés.*

Limite du passage par valeur

On a souvent besoin d'écrire des procédure ou fonctions que, au contraire, *modifient* la valeur de certains paramètres, par exemple, on voudrait écrire une procedure `echange` t.q. `echange(x,y)` échange les valeurs des variables `x` et `y`.

Mais la procédure

```
procédure echange (valeur a: entier, valeur b: entier)
variable t: un entier
debut echange
 t <- a
 a <- y
 b <- t
fin echange
```

n'échange pas les valeurs de ses arguments!
(relisez la description du passage d'arguments par valeur plus haut)

Comment faire?

Passage par référence

Si on écrit

```
procédure echange (reference a: entier, reference b: entier)
variable t: un entier
debut echange
 t <- a
 a <- y
 b <- t
fin echange
```


le passage des paramètres se fait autrement: au moment de l'appel

`echange(i,j)`

- *on ne fait pas une copie des valeurs de i et j dans a et b!*
- au contraire, on transforme a et b en "alias" de i et j, en le faisant pointer sur les memes cases mémoire de i et j
- donc, toute opération effectuée sur les paramètres a et b dans le corps de la procédure ou fonction et aussi effectué sur les arguments i et j

N.B.: on doit passer comme argument d'un paramètre par référence seulement des expressions qui se comportent comme des variables (par exemple, une variable, mais aussi une case d'un tableau etc.)

Résumé

- On déclare "valeur" un paramètre si on ne veut pas que l'appel modifie les arguments
- On déclare "reference" un paramètre si on veut que l'appel modifie les arguments

Bien sûr, on peut melanger paramètres par valeur et par référence.

ATTENTION:

- les tableaux sont *toujours* passé par référence, et *jamais* copiés, même si on déclare le paramètres correspondant "valeur".
  (Vous comprendrez les raisons de cela plus avant dans le cursus informatique).

Un exemple complet: le tri par séléction

Voyons comme écrire un programme de tri par séléction beaucoup plus lisible avec les fonctions et les procédures.

Fonction de recherche du minimum à partir d'un indice

```
fonction mintab(valeur i: un entier, reference a: tableau): entier
variable k: un entier
debut mintab
   pour k <- i+1 a  N faire
    si  (a[k] < a[i]) alors
        i <- k
    fin si
   fin pour
   retourner i
fin mintab
```

Programme de tri par séléction

```
constante N=10
type tab = tableau de N entiers
procedure echange(reference a, reference b) ...
fonction mintab(valeur i: un entier, reference a: tableau): entier ...

programme triselection
variables a: tab
debut triselection
lire a
  pour i <- 1 a N-1 faire
      echange(a[i],a[mintab(i,a)])
  fin pour
ecrire a
fin triselection
```


Et en C++?

```
#include <iostream.h>

const int N=10;

typedef int tab[N];

void echange (int &a, int &b)
{ int t;
  t=a; a=b; b=t;
}

int mintab(int i, tab &a)
{ int k;

  for(k=i+1;k<N;k++){if (a[k] < a[i]) {i =k;};};
  return(i);
}

void lire(tab &a)
{ int k;

  for (k=0;k<N;k++){cout<<"a["<<k<<"] ? "<<endl; cin>>a[k];};
}

void ecrire(tab &a)
{ int k;
 for (k=0;k<N;k++){cout<<"a["<<k<<"] = "<<a[k]<<endl;};
}

void main(){
 tab a;
 int i;

 lire(a);
 for (i=0;i<N-1;i++){echange(a[i],a[mintab(i,a)]);};
 ecrire(a);
}
```

=

Fonctions et procédures

Idée : capitaliser sur le code déjà écrit pour introduire des nouvelles "commandes" et "opérations".

On souhaite donner un nom programme que nous avons déjà écrit, et qui effectue une opération intéressante, pour pouvoir le réutiliser après sans le réécrire. Si ce programme retourne un résultat, on parlera de "fonction", et "procédure" sinon.

De plus, on peut souhaiter choisir les valeurs de certaines variables de la procédure ou fonction, et cela se fait à travers le mechanisme de passage des paramètres.

Procedures: déclaration et appel

Chaque procédure aura:

1. Une *définition* qui dit
   (a) Le type des paramètres.
   (b) comment on la calcule.
2. Un ou plusieurs *appels*: c'est l'utilisation de la procédure.

Exemples (sans paramètres)

Tous les programmes affichent un message d'aide à l'utilisation, et cela à plusieurs occasions

```
procedure  aide ()
  debut aide
        ecrire "Aide du programme Machin."
        ecrire " l'option -o permet de definir le fichier sortie"
        ecrire " l'option -h donne ce message"
  fin aide
...
programme Machin
debut Machin
```



```
        ...
        aide()
        ...
        si (erreur=vrai) alors aide() fin si
        ...
fin Machin
```

Exemples (avec paramètres)

Tous les programmes affichent des messages d'erreur

```
procedure  erreur (valeur s: chaîne de caractères)
  debut erreur
        ecrire "Programme Machin: on a renconter l'erreur" s
  fin erreur
...
programme Machin
debut Machin
        ...
        erreur("Entier trop grand")
        ...
        erreur("Pas assez de memoire")
        ...
fin Machin
```

Fonctions: déclaration et appel

Chaque fonction aura:

1. Une *définition* qui dit
   (a) Le type des paramètres.
   (b) Le type de la valeur rendue.
   (c) comment on la calcule.
2. Un ou plusieurs *appels*: c'est l'utilisation de la fonction.
3. Zero, un ou plusieurs *paramètres*: ce sont les arguments de la fonction.
4. Un *type* et une *valeur*.

Exemples (sans paramètres)

```
fonction pi(): reel
  debut pi
      retourner 3.1415926535897931
  fin pi
```

Exemple avec paramètres

La fonction suivante rend le maximum de deux valeurs de type réél.

```
fonction fmax(valeur a: reel, valeur b: reel) : reel
variables m : un reel
debut fmax
 si (a > b)
 alors
   m <- a
 sinon
   m <- b
 fin si
 retourner m
fin fmax
```

de sorte que l'expression `fmax(pi()*pi(),10.0)` vaut 10.

Paramètres

En général, une fonction ou procédure n'est utile que si elle a des paramètres, comme dans le cas de **fmax**.

Une fonction ou procédure définie avec des paramètres doit être appelée avec des *arguments* (en nombre égal, et du même type!).

définition de fonction	appel de fonction
paramètre	argument


```
fonction fmax(valeur a: reel, valeur b: reel) : reel
...
programme essai
variables x: un reel
          i,j: deux caracteres
debut essai
   lire x
   ecrire fmax(x,x*x)
   /* ecrire fmax(i,j) serait incorrect! */
fin essai
```

Nature des paramètres: déclaration

Dans la declaration

```
fonction fmax(valeur a: reel, valeur b: reel) : reel
...
```

- `a` et `b` sont les paramètres de la fonction
- ils sont déclarés avec leur type
- dans le corps de la fonction, on peut utiliser `a` et `b` comme des variables,
  qui existent exclusivement dans le corps de la fonction
- le mot clé `valeur` indique une *modalité* de passage des paramètres

Nature des paramètres: passage par valeur

Après la déclaration

```
fonction fmax(valeur a: reel, valeur b: reel) : reel
...
```

on peut effectuer un appel de la fonction comme

```
ecrire 3*fmax(x,y)
```

l'expression `fmax(x,y)` est évaluée de la façon suivante:

- les valeurs de `x` et `y` sont calculées

- les paramètres `a` et `b` de `fmax` sont initialisés en utilisant ces valuers
- le corps de la fonction `fmax` est executé, (cela utilise les paramètres `a` et `b` comme des variables)
- le résultat de la fonction est enfin retourné et utilisé pour continuer le calcul de l'expression (ici, on le multiplie par 3 et on l'imprime)

Peu importe les opérations effectuées sur `a` *et* `b` *dans le corps de* `fmax`, *les valeurs des paramètres actuels* `x` *et* `y` *restent inchangés.*

Limite du passage par valeur

On a souvent besoin d'écrire des procédure ou fonctions que, au contraire, *modifient* la valeur de certains paramètres, par exemple, on voudrait écrire une procedure `echange` t.q. `echange(x,y)` échange les valeurs des variables `x` et `y`.

Mais la procédure

```
procédure echange (valeur a: entier, valeur b: entier)
variable t: un entier
debut echange
 t <- a
 a <- y
 b <- t
fin echange
```

n'échange pas les valeurs de ses arguments!
(relisez la description du passage d'arguments par valeur plus haut)

Comment faire?

Passage par référence

Si on écrit

```
procédure echange (reference a: entier, reference b: entier)
variable t: un entier
debut echange
 t <- a
 a <- y
 b <- t
fin echange
```


le passage des paramètres se fait autrement: au moment de l'appel

`echange(i,j)`

- *on ne fait pas une copie des valeurs de i et j dans a et b!*
- au contraire, on transforme a et b en "alias" de i et j, en le faisant pointer sur les memes cases mémoire de i et j
- donc, toute opération effectuée sur les paramètres a et b dans le corps de la procédure ou fonction et aussi effectué sur les arguments i et j

N.B.: on doit passer comme argument d'un paramètre par référence seulement des expressions qui se comportent comme des variables (par exemple, une variable, mais aussi une case d'un tableau etc.)

Résumé

- On déclare "valeur" un paramètre si on ne veut pas que l'appel modifie les arguments
- On déclare "reference" un paramètre si on veut que l'appel modifie les arguments

Bien sûr, on peut melanger paramètres par valeur et par référence.

ATTENTION:

- les tableaux sont *toujours* passé par référence, et *jamais* copiés, même si on déclare le paramètres correspondant "valeur".
  (Vous comprendrez les raisons de cela plus avant dans le cursus informatique).

Un exemple complet: le tri par séléction

Voyons comme écrire un programme de tri par séléction beaucoup plus lisible avec les fonctions et les procédures.

Fonction de recherche du minimum à partir d'un indice

```
fonction mintab(valeur i: un entier, reference a: tableau): entier
variable k: un entier
debut mintab
   pour k <- i+1 a  N faire
    si  (a[k] < a[i]) alors
        i <- k
    fin si
   fin pour
   retourner i
fin mintab
```

Programme de tri par séléction

```
constante N=10
type tab = tableau de N entiers
procedure echange(reference a, reference b) ...
fonction mintab(valeur i: un entier, reference a: tableau): entier ...

programme triselection
variables a: tab
debut triselection
lire a
  pour i <- 1 a N-1 faire
      echange(a[i],a[mintab(i,a)])
  fin pour
ecrire a
fin triselection
```


Et en C++?

```
#include <iostream.h>

const int N=10;

typedef int tab[N];

void echange (int &a, int &b)
{ int t;
  t=a; a=b; b=t;
}

int mintab(int i, tab &a)
{ int k;

  for(k=i+1;k<N;k++){if  (a[k] < a[i]) {i =k;};};
  return(i);
}

void lire(tab &a)
{ int k;

  for (k=0;k<N;k++){cout<<"a["<<k<<"] ? "<<endl; cin>>a[k];};
}

void ecrire(tab &a)
{ int k;
 for (k=0;k<N;k++){cout<<"a["<<k<<"] = "<<a[k]<<endl;};
}

void main(){
 tab a;
 int i;

 lire(a);
 for (i=0;i<N-1;i++){echange(a[i],a[mintab(i,a)]);};
 ecrire(a);
}
```

=

Fonctions et procédures

Idée : capitaliser sur le code déjà écrit pour introduire des nouvelles "commandes" et "opérations".

On souhaite donner un nom programme que nous avons déjà écrit, et qui effectue une opération intéressante, pour pouvoir le réutiliser après sans le réécrire. Si ce programme retourne un résultat, on parlera de "fonction", et "procédure" sinon.

De plus, on peut souhaiter choisir les valeurs de certaines variables de la procédure ou fonction, et cela se fait à travers le mechanisme de passage des paramètres.

Procedures: déclaration et appel

Chaque procédure aura:

1. Une *définition* qui dit
   (a) Le type des paramètres.
   (b) comment on la calcule.
2. Un ou plusieurs *appels*: c'est l'utilisation de la procédure.

Exemples (sans paramètres)

Tous les programmes affichent un message d'aide à l'utilisation, et cela à plusieurs occasions

```
procedure  aide ()
  debut aide
        ecrire "Aide du programme Machin."
        ecrire " l'option -o permet de definir le fichier sortie"
        ecrire " l'option -h donne ce message"
  fin aide
...
programme Machin
debut Machin
```



```
        ...
        aide()
        ...
        si (erreur=vrai) alors aide() fin si
        ...
fin Machin
```

Exemples (avec paramètres)

Tous les programmes affichent des messages d'erreur

```
procedure  erreur (valeur s: chaîne de caractères)
  debut erreur
        ecrire "Programme Machin: on a rencontér l'erreur" s
  fin erreur
...
programme Machin
debut Machin
        ...
        erreur("Entier trop grand")
        ...
        erreur("Pas assez de memoire")
        ...
fin Machin
```

Fonctions: déclaration et appel

Chaque fonction aura:

1. Une *définition* qui dit
   (a) Le type des paramètres.
   (b) Le type de la valeur rendue.
   (c) comment on la calcule.
2. Un ou plusieurs *appels*: c'est l'utilisation de la fonction.
3. Zero, un ou plusieurs *paramètres*: ce sont les arguments de la fonction.
4. Un *type* et une *valeur*.

Exemples (sans paramètres)

```
fonction pi(): reel
  debut pi
      retourner 3.1415926535897931
  fin pi
```

Exemple avec paramètres

La fonction suivante rend le maximum de deux valeurs de type réél.

```
fonction fmax(valeur a: reel, valeur b: reel) : reel
variables m : un reel
debut fmax
 si (a > b)
 alors
   m <- a
 sinon
   m <- b
 fin si
 retourner m
fin fmax
```

de sorte que l'expression `fmax(pi()*pi(),10.0)` vaut 10.

Paramètres

En général, une fonction ou procédure n'est utile que si elle a des paramètres, comme dans le cas de `fmax`.

Une fonction ou procédure définie avec des paramètres doit être appelée avec des *arguments* (en nombre égal, et du même type!).

définition de fonction	appel de fonction
paramètre	argument


```
fonction fmax(valeur a: reel, valeur b: reel) : reel
...
programme essai
variables x: un reel
          i,j: deux caracteres
debut essai
   lire x
   ecrire fmax(x,x*x)
   /* ecrire fmax(i,j) serait incorrect! */
fin essai
```

Nature des paramètres: déclaration

Dans la declaration

```
fonction fmax(valeur a: reel, valeur b: reel) : reel
...
```

- `a` et `b` sont les paramètres de la fonction
- ils sont déclarés avec leur type
- dans le corps de la fonction, on peut utiliser `a` et `b` comme des variables,
  qui existent exclusivement dans le corps de la fonction
- le mot clé `valeur` indique une *modalité* de passage des paramètres

Nature des paramètres: passage par valeur

Après la déclaration

```
fonction fmax(valeur a: reel, valeur b: reel) : reel
...
```

on peut effectuer un appel de la fonction comme

```
ecrire 3*fmax(x,y)
```

l'expression `fmax(x,y)` est évaluée de la façon suivante:

- les valeurs de `x` et `y` sont calculées

- les paramètres `a` et `b` de `fmax` sont initialisés en utilisant ces valuers
- le corps de la fonction `fmax` est executé, (cela utilise les paramètres `a` et `b` comme des variables)
- le résultat de la fonction est enfin retourné et utilisé pour continuer le calcul de l'expression (ici, on le multiplie par 3 et on l'imprime)

Peu importe les opérations effectuées sur `a` *et* `b` *dans le corps de* `fmax`, *les valeurs des paramètres actuels* `x` *et* `y` *restent inchangés.*

Limite du passage par valeur

On a souvent besoin d'écrire des procédure ou fonctions que, au contraire, *modifient* la valeur de certains paramètres, par exemple, on voudrait écrire une procedure `echange` t.q. `echange(x,y)` échange les valeurs des variables `x` et `y`.

Mais la procédure

```
procédure echange (valeur a: entier, valeur b: entier)
variable t: un entier
debut echange
 t <- a
 a <- y
 b <- t
fin echange
```

n'échange pas les valeurs de ses arguments!
(relisez la description du passage d'arguments par valeur plus haut)

Comment faire?

Passage par référence

Si on écrit

```
procédure echange (reference a: entier, reference b: entier)
variable t: un entier
debut echange
 t <- a
 a <- y
 b <- t
fin echange
```


le passage des paramètres se fait autrement: au moment de l'appel

`echange(i,j)`

- *on ne fait pas une copie des valeurs de i et j dans a et b!*
- au contraire, on transforme a et b en "alias" de i et j, en le faisant pointer sur les memes cases mémoire de i et j
- donc, toute opération effectuée sur les paramètres a et b dans le corps de la procédure ou fonction et aussi effectué sur les arguments i et j

N.B.: on doit passer comme argument d'un paramètre par référence seulement des expressions qui se comportent comme des variables (par exemple, une variable, mais aussi une case d'un tableau etc.)

Résumé

- On déclare "valeur" un paramètre si on ne veut pas que l'appel modifie les arguments
- On déclare "reference" un paramètre si on veut que l'appel modifie les arguments

Bien sûr, on peut melanger paramètres par valeur et par référence.

ATTENTION:

- les tableaux sont *toujours* passé par référence, et *jamais* copiés, même si on déclare le paramètres correspondant "valeur".
(Vous comprendrez les raisons de cela plus avant dans le cursus informatique).

Un exemple complet: le tri par séléction

Voyons comme écrire un programme de tri par séléction beaucoup plus lisible avec les fonctions et les procédures.

Fonction de recherche du minimum à partir d'un indice

```
fonction mintab(valeur i: un entier, reference a: tableau): entier
variable k: un entier
debut mintab
   pour k <- i+1 a  N faire
    si  (a[k] < a[i]) alors
        i <- k
    fin si
   fin pour
   retourner i
fin mintab
```

Programme de tri par séléction

```
constante N=10
type tab = tableau de N entiers
procedure echange(reference a, reference b) ...
fonction mintab(valeur i: un entier, reference a: tableau): entier ...

programme triselection
variables a: tab
debut triselection
lire a
  pour i <- 1 a N-1 faire
      echange(a[i],a[mintab(i,a)])
  fin pour
ecrire a
fin triselection
```


Et en C++?

```
#include <iostream.h>

const int N=10;

typedef int tab[N];

void echange (int &a, int &b)
{ int t;
  t=a; a=b; b=t;
}

int mintab(int i, tab &a)
{ int k;

  for(k=i+1;k<N;k++){if (a[k] < a[i]) {i =k;};};
  return(i);
}

void lire(tab &a)
{ int k;

  for (k=0;k<N;k++){cout<<"a["<<k<<"] ? "<<endl; cin>>a[k];};
}

void ecrire(tab &a)
{ int k;
 for (k=0;k<N;k++){cout<<"a["<<k<<"] = "<<a[k]<<endl;};
}

void main(){
 tab a;
 int i;

 lire(a);
 for (i=0;i<N-1;i++){echange(a[i],a[mintab(i,a)]);};
 ecrire(a);
}
```

=

Fonctions et procédures

Idée : capitaliser sur le code déjà écrit pour introduire des nouvelles "commandes" et "opérations".

On souhaite donner un nom programme que nous avons déjà écrit, et qui effectue une opération intéressante, pour pouvoir le réutiliser après sans le réécrire. Si ce programme retourne un résultat, on parlera de "fonction", et "procédure" sinon.

De plus, on peut souhaiter choisir les valeurs de certaines variables de la procédure ou fonction, et cela se fait à travers le mechanisme de passage des paramètres.

Procedures: déclaration et appel

Chaque procédure aura:

1. Une *définition* qui dit
   (a) Le type des paramètres.
   (b) comment on la calcule.
2. Un ou plusieurs *appels*: c'est l'utilisation de la procédure.

Exemples (sans paramètres)

Tous les programmes affichent un message d'aide à l'utilisation, et cela à plusieurs occasions

```
procedure  aide ()
  debut aide
        ecrire "Aide du programme Machin."
        ecrire " l'option -o permet de definir le fichier sortie"
        ecrire " l'option -h donne ce message"
  fin aide
...
programme Machin
debut Machin
```



```
        ...
        aide()
        ...
        si (erreur=vrai) alors aide() fin si
        ...
fin Machin
```

Exemples (avec paramètres)

Tous les programmes affichent des messages d'erreur

```
procedure  erreur (valeur s: chaîne de caractères)
  debut erreur
        ecrire "Programme Machin: on a renconter l'erreur" s
  fin erreur
...
programme Machin
debut Machin
        ...
        erreur("Entier trop grand")
        ...
        erreur("Pas assez de memoire")
        ...
fin Machin
```

Fonctions: déclaration et appel

Chaque fonction aura:

1. Une *définition* qui dit
   (a) Le type des paramètres.
   (b) Le type de la valeur rendue.
   (c) comment on la calcule.
2. Un ou plusieurs *appels*: c'est l'utilisation de la fonction.
3. Zero, un ou plusieurs *paramètres*: ce sont les arguments de la fonction.
4. Un *type* et une *valeur*.

Exemples (sans paramètres)

```
fonction pi(): reel
  debut pi
      retourner 3.1415926535897931
  fin pi
```

Exemple avec paramètres

La fonction suivante rend le maximum de deux valeurs de type réél.

```
fonction fmax(valeur a: reel, valeur b: reel) : reel
variables m : un reel
debut fmax
 si (a > b)
 alors
   m <- a
 sinon
   m <- b
 fin si
 retourner m
fin fmax
```

de sorte que l'expression `fmax(pi()*pi(),10.0)` vaut 10.

Paramètres

En général, une fonction ou procédure n'est utile que si elle a des paramètres, comme dans le cas de `fmax`.

Une fonction ou procédure définie avec des paramètres doit être appelée avec des *arguments* (en nombre égal, et du même type!).

définition de fonction	appel de fonction
paramètre	argument


```
fonction fmax(valeur a: reel, valeur b: reel) : reel
...
programme essai
variables x: un reel
          i,j: deux caracteres
debut essai
   lire x
   ecrire fmax(x,x*x)
   /* ecrire fmax(i,j) serait incorrect! */
fin essai
```

Nature des paramètres: déclaration

Dans la declaration

```
fonction fmax(valeur a: reel, valeur b: reel) : reel
...
```

- `a` et `b` sont les paramètres de la fonction
- ils sont déclarés avec leur type
- dans le corps de la fonction, on peut utiliser `a` et `b` comme des variables,
  qui existent exclusivement dans le corps de la fonction
- le mot clé `valeur` indique une *modalité* de passage des paramètres

Nature des paramètres: passage par valeur

Après la déclaration

```
fonction fmax(valeur a: reel, valeur b: reel) : reel
...
```

on peut effectuer un appel de la fonction comme

```
ecrire 3*fmax(x,y)
```

l'expression `fmax(x,y)` est évaluée de la façon suivante:

- les valeurs de `x` et `y` sont calculées

- les paramètres `a` et `b` de `fmax` sont initialisés en utilisant ces valuers
- le corps de la fonction `fmax` est executé, (cela utilise les paramètres `a` et `b` comme des variables)
- le résultat de la fonction est enfin retourné et utilisé pour continuer le calcul de l'expression (ici, on le multiplie par 3 et on l'imprime)

Peu importe les opérations effectuées sur `a` *et* `b` *dans le corps de* `fmax`, *les valeurs des paramètres actuels* `x` *et* `y` *restent inchangés.*

Limite du passage par valeur

On a souvent besoin d'écrire des procédure ou fonctions que, au contraire, *modifient* la valeur de certains paramètres, par exemple, on voudrait écrire une procedure `echange` t.q. `echange(x,y)` échange les valeurs des variables `x` et `y`.

Mais la procédure

```
procédure echange (valeur a: entier, valeur b: entier)
variable t: un entier
debut echange
 t <- a
 a <- y
 b <- t
fin echange
```

n'échange pas les valeurs de ses arguments!
(relisez la description du passage d'arguments par valeur plus haut)

Comment faire?

Passage par référence

Si on écrit

```
procédure echange (reference a: entier, reference b: entier)
variable t: un entier
debut echange
 t <- a
 a <- y
 b <- t
fin echange
```


le passage des paramètres se fait autrement: au moment de l'appel

`echange(i,j)`

- *on ne fait pas une copie des valeurs de i et j dans a et b!*
- au contraire, on transforme a et b en "alias" de i et j, en le faisant pointer sur les memes cases mémoire de i et j
- donc, toute opération effectuée sur les paramètres a et b dans le corps de la procédure ou fonction et aussi effectué sur les arguments i et j

N.B.: on doit passer comme argument d'un paramètre par référence seulement des expressions qui se comportent comme des variables (par exemple, une variable, mais aussi une case d'un tableau etc.)

Résumé

- On déclare "valeur" un paramètre si on ne veut pas que l'appel modifie les arguments
- On déclare "reference" un paramètre si on veut que l'appel modifie les arguments

Bien sûr, on peut melanger paramètres par valeur et par référence.

ATTENTION:

- les tableaux sont *toujours* passé par référence, et *jamais* copiés, même si on déclare le paramètres correspondant "valeur".
  (Vous comprendrez les raisons de cela plus avant dans le cursus informatique).

Un exemple complet: le tri par séléction

Voyons comme écrire un programme de tri par séléction beaucoup plus lisible avec les fonctions et les procédures.

Fonction de recherche du minimum à partir d'un indice

```
fonction mintab(valeur i: un entier, reference a: tableau): entier
variable k: un entier
debut mintab
   pour k <- i+1 a  N faire
    si  (a[k] < a[i]) alors
        i <- k
    fin si
   fin pour
   retourner i
fin mintab
```

Programme de tri par séléction

```
constante N=10
type tab = tableau de N entiers
procedure echange(reference a, reference b) ...
fonction mintab(valeur i: un entier, reference a: tableau): entier ...

programme triselection
variables a: tab
debut triselection
lire a
  pour i <- 1 a N-1 faire
      echange(a[i],a[mintab(i,a)])
  fin pour
ecrire a
fin triselection
```


Et en C++?

```
#include <iostream.h>

const int N=10;

typedef int tab[N];

void echange (int &a, int &b)
{ int t;
  t=a; a=b; b=t;
}

int mintab(int i, tab &a)
{ int k;

  for(k=i+1;k<N;k++){if (a[k] < a[i]) {i =k;};};
  return(i);
}

void lire(tab &a)
{ int k;

  for (k=0;k<N;k++){cout<<"a["<<k<<"] ? "<<endl; cin>>a[k];};
}

void ecrire(tab &a)
{ int k;
 for (k=0;k<N;k++){cout<<"a["<<k<<"] = "<<a[k]<<endl;};
}

void main(){
 tab a;
 int i;

 lire(a);
 for (i=0;i<N-1;i++){echange(a[i],a[mintab(i,a)]);};
 ecrire(a);
}
```