Exemple (Rappel fonction)

```
/** ce programme est un exemple rudimentaire
 de manipulation de compte bancaire
**/

/**fonction pour demander un nombre positif **/
fonction pos(valeur c : chaine carac.)
                              résultat réel
  variable s : réel
début pos
  afficher c
  répéter
    afficher ‘‘Entrer un montant : ‘‘
    lire s
  tant que s<=0
  retouner s
fin pos

programme CompteBancaire
 variable solde réelle
début CompteBancaire
 solde <- solde + pos(‘‘Combien déposez-vous ?’’)
 solde <- solde - pos(‘‘Combien retirez-vous ?’’)
 afficher ‘‘Votre solde est : ‘‘, solde
fin CompteBancaire
```

Remarque: on utilise ici le pseudo-langage pour décrire un programme principal

Exemple procédure

Souvent, on veut définir une partie d'un programme qui ne retourne pas de valeurs, mais qui a d'autres effets. Par exemple, on veut afficher un tableau sur l'écran.

```
Constante NbCases : entier 10
Type Tab: Tableau d’entiers
                        indicé de 1 à NbCases

/* Affiche les éléments d’un tableau t entre
les positions début et fin
pré-condition: début et fin indices valides */

procédure afficherTab(valeurs t : Tab,
                            début,fin : entiers)

variables i : entier

début afficherTab
   Pour i <- début à fin faire
        afficher t[i]
   fin pour
fin afficherTab
```


Autre exemple

On veut échanger les valeurs de deux nombres. On ne renvoie pas de résultat, mais on change les valeurs des paramètres.

On utilise une procédure au lieu d'une fonction.

Premier essai:

/* Echange deux nombres*/

procédure échange1(référence x,y : réels)
début échange
    x <- y ; y <- x
fin échange

Pourquoi ça ne marche pas ?

/* Echange deux nombres*/

procédure échange(référence x,y : réels)
variable z:réel
début échange
    z <- x; x <- y ; y <- z
fin échange

Fonctions et procédures

Les utilisations les plus courantes des procédures et fonctions sont :

- structurer le programme
- éviter les répétitions
- constituer des bibliothèques de fonctions usuelles

Second exemple

On veut écrire un programme qui lit un tableau de nombres, affiche le tableau ainsi constitué, trie les nombres en ordre croissant puis affiche le tableau trié.

```
/* Programme pour lire un tableau de nombres,
 * l'afficher,
 * trier les nombres en ordre croissant
 * et afficher le tableau trié
 */
```

constante N : entier 10
type Tab : tableau de N entiers indicé de 0 à N-1

programme tri
variable a:Tab

début tri
    lireTab(a, 0, N-1)
    afficherTab(a, 0, N-1)
    tri_sélection(a, 0, N-1)
    afficherTab(a, 0, N-1)
fin tri

Tri par sélection

- Idée: sélectionner pour chaque case du tableau (ici de gauche à droite) le nombre qui convient.
- Dans la première case, il faut sélectionner le plus petit nombre du tableau.
- Faire de même pour le sous-tableau qui commence à la deuxième case.
- Et ainsi de suite, jusqu'à l'avant-dernière case.
- Le tableau est divisé en une partie déjà triée, et une autre que l'on continue à trier en sélectionnant toujours le minimum.
- Exemple: Avec le tableau

  [12 5 78 23 6]

  on obtient la suite

  [5][12 78 32 61]

  [5 12][78 32 61]

  [5 12 32][78 61]

  [5 12 32 61 78]

Tri par sélection

On peut raffiner la procédure

tri_sélection(a, 0, N-1)

Les tâches :

- échanger deux éléments du tableau et
- rechercher le plus petit élément dans un tableau entre deux bornes.

sont laissées pour plus tard.

```
/* Procédure pour trier un tableau en ordre
 * croissant par sélection.
 * pré-condition : i,j indices valides et i <= j.
 */
```

procédure tri_sélection(référence t:Tab, valeur i,j:entier)
variable k:entier
début tri_sélection
// Invariant : t[i..k-1] est trié et
// t[i..k-1] <= t[k..j] et i <= k <= j
    pour k <- i à j-1 faire
        échanger t[k] avec l'élément le plus petit de t[k..j]
    fin pour fin tri_sélection

Fonctions et procédures : en-tête

- La fonctionnalité et les conditions d'utilisation doivent être très précisément définies par un commentaire.
- L'en-tête d'une procédure ou fonction doit spécifier complètement, par l'intermédiaire de ses paramètres, son interface avec le reste du programme.
- Autres exemples d'en-têtes :

  /* Calcul du pgcd
  * pré-condition a et b positifs et non tous deux nuls
  */

  fonction pgcd(valeur a,b:entier) résultat entier

  /* Echange les valeurs de deux nombres */

  procédure échange(référence x,y:réel)

Bloc d'une procédure ou fonction

Il consiste en

- Déclarations locales (constantes, variables)
- Liste d'instructions.

Dans les instructions on peut utiliser

- les paramètres, les variables locales et les constantes globales
- mais pas les variables locales des autres procédures et fonctions.

Dans une fonction, il doit y avoir une instruction

Retourner expression

qui termine l'exécution de la fonction et définit la valeur du résultat.

Résumé : Une fonction ou procédure se compose d'une en-tête suivi d'un bloc.

Paramètres

1. Paramètre formel

   Le paramètre formel est celui utilisé dans l'en-tête et le bloc de la procédure ou fonction.

   - a et b pour la fonction pgcd
   - x et y pour la procédure échange

   Quels sont les paramètres formels des procédures *tri_sélection* et *pos* ?

2. Paramètre effectif

   Le paramètre effectif est celui utilisé lors de l'appel de la procédure ou fonction.

   - a, 0, N-1 pour la procédure tri_sélection

   !! Les paramètres formels et effectifs correspondants doivent être de même type !!

Modes de passage des paramètres

3. Passage du paramètre par valeur
   - Le paramètre passé par valeur est traité comme une variable locale à la fonction ou procédure. Lors de l'activation de la procédure ou fonction la valeur du paramètre effectif est copiée dans cette variable locale.
   - Le paramètre effectif peut être une variable, une constante ou une expression arithmétique.
   - La valeur du paramètre effectif n'est pas modifiée après l'appel de la fonction ou procédure.

4. Passage du paramètre par référence
   - Le paramètre formel est un alias (synonyme) du paramètre effectif.
   - La procédure/fonction travaille directement sur le paramètre effectif.
   - Le paramètre effectif doit être une variable.
   - La valeur du paramètre effectif peut être modifiée après l'appel de la fonction ou procédure.

Remarque: Si ce n'est pas explicitement indiqué dans l'en-tête de la fonction ou procédure que le passage est par référence, le passage se fait par valeur.

Où est LA différence ?

```
/* Echange deux nombres?!*/
procédure échange1(valeur x,y : réels)
variable z:réel
début échange1
    z <- x; x <- y ; y <- z
fin échange1

/* Echange deux nombres!?*/
procédure échange2(référence x,y : réels)
variable z:réel
début échange2
    z <- x; x <- y; y <- z
fin échange2

programme Exercice
variables a,b: réel
début Exercice
    lire a, b;
    échange1(a,b);
    afficher a,b;
    échange2(a,b);
    afficher a,b;
fin Exercice
```

Exercice: Exécuter à la main ce programme avec $a = 3$ et $b = 5$.

Paramètre valeur et référence

procédure P(valeur x:entier, référence y:entier)

```
variable z:entier
début P
    z <- y div 2; y <- x*x; x <- y*y + z*z
fin P
```

programme Test

```
variables a,b,x,y,z : entiers
début Test
    a <- 3; b <- 0; x <- 5; y <- -2; z <- 8
    P(a,b)
    afficher a, b, x, y, z
    a <- 4; P(a,a)
    afficher a, b, x, y, z
    P(x,y)
    afficher x, y, z
    x <- 2; z <- 3; P(x,y)
    afficher x, y, z
fin Test
```

Exécuter Test à la main.

Procédure ou fonction ?

Une procédure est utilisée comme une instruction

si x < y alors échange(x,y) finsi

lireTab(a, 0, N-1)
tri_sélection(a, 0, N-1)
afficherTab(a, 0, N-1)

Une fonction est utilisée comme un facteur dans une expression.

m <- a * b / pgcd(a,b)

Exemple : échange

```
/* Echange deux nombres*/
procédure échange(référence x,y : réels)
variable z:réel
début échange
    z <- x; x <- y ; y <- z
fin échange
```

Exemples d'utilisations

```
a <- 48; b <- 21; z <- 12; échange(a,b)
afficher a, b, z
a <- 48; x <- 21; z <- 12; échange(a,x)
afficher a, x, z
x <- 48; y <- 21; z <- 12;
échange(2*x,3*y)  // Erreur
```


Variable locale, portée, masquage

- Toute variable a une localisation : le bloc où elle a été déclarée.
- Toute variable est accessible uniquement dans son bloc de déclaration. C'est ce qui s'appelle la portée d'une variable.
- La portée d'un paramètre formel est le bloc de sa fonction ou procédure. Une variable locale d'une fonction ou procédure ne peut pas avoir le même nom qu'un paramètre formel.
- Un paramètre formel ou une variable masque une constante globale du même nom.

Exemple

Attention au masquage !

```
constante n : entier 3
procédure P(valeur x:entier, référence y:entier)

variables
z,y:entier // Erreur : y est déjà un paramètre
n: réel    // La variable n masque la constante n
début P
      y <- 3*x+2
      z <- 2
      n <- 0.3    // OK. Ici n est une variable réelle
      b <- 8      // Erreur : b n'existe pas ici
fin P        // fin de la portée de x,y,z,n

programme Q

variables
a,b : entiers
x : réel
début Q
      a <- n        // il s'agit de la constante n
      y <- 8        // erreur : la portée de y et z
      z <- 2*a+b    // se limite à la procédure P
      x <- 0.5      // OK. C'est le x de Q
P(8,n) // erreur : n est une constante
fin Q
```


Procédures et fonctions en C++

Déclaration d'une fonction :

type_résultat nom_fonction (liste de paramètres)

bloc de la fonction

Déclaration d'une procédure :

void nom_procédure (liste de paramètres)

bloc de la procédure

Déclaration d'un paramètre passé par valeur :

type_paramètre nom_paramètre

Déclaration d'un paramètre passé par référence :

type_paramètre &nom_paramètre

Cas particulier des paramètres tableaux

Un tableau ne peut pas être transmis par valeur en C++
La déclaration de paramètre

Tab t

correspond à une transmission par référence
(sans copie du tableau).

Ne pas utiliser la déclaration

Tab &t

Pour interdire la modification d'un paramètre tableau par une procédure ou une fonction on utilise la déclaration de paramètre

const Tab t

Il s'agit toujours d'une transmission par référence (pas de copie) mais le compilateur interdit toute modification du paramètre.
Remarque : on peut aussi utiliser la déclaration const pour un paramètre d'un autre type.

const double &x

Le paramètre est transmis par référence (pas de copie) mais ne peut pas être modifié. Ceci est utile pour éviter de recopier un gros paramètre (e.g. une structure) qui ne doit pas être modifié.

Exercices Cours/TD

1. Écrire une fonction qui prend comme paramètre un tableau de N entiers et ses bornes et qui retourne le minimum du tableau.
2. Écrire une procédure pour lire un tableau d'entiers entre deux bornes.
3. Écrire une procédure pour afficher un tableau d'entiers entre deux bornes.
4. Ecrire une fonction qui étant donné un tableau d'entiers, ses bornes et un entier x retourne `vrai` si x est dans le tableau, `faux` sinon.

Tri par sélection détaillé

```
constante N : entier 10
type Tab = tableau de N entiers
           indicé par [0...N-1]

  // Echange les valeurs de x et y
procédure échange(référence x,y: entiers)
 variable z:entier
début échange
   z <- x; x <- y; y <- z
fin échange

// Retourne la position du minimum de t[i..j]
// On doit avoir i,j indices valides et i <= j
fonction posmin(valeur t:Tab,
                valeur i,j:entier) résultat entier
variables k,pos : entiers
début posmin
   pos <- i
   //Invariant: t[pos] <= t[i..k-1] et k <= j+1
   pour k <- i+1 à j faire
       si t[k] < t[pos] alors pos <- k finsi
   fin pour
   retourner pos
fin posmin
```



```
// tri de t[i..j]
// On doit avoir i,j indices valides et i <= j
procédure tri_séléction(référence t:Tab,
                        valeur i,j:entier)
variable k:entier
début tri
     // Invariant : t[i..k-1] est trié et
     // t[i..k-1] <= t[k..j] et i <= k <= j
   pour k <- i à j-1 faire
         échange(t[k],t[posmin(t,k,j)])
   fin pour
fin tri

programme tri
variable a:Tab
début tri
  lire a        //  abbréviation en Pseudo
  tri_sélection(a, 0, N-1)
  afficher a    //  abbréviation en Pseudo
fin tri
```


Tri par sélection en C++

```
#include <iostream.h>
const int N=10;
typedef int Tab[N];

// Echange les valeurs de x et y
void echange(int &x, int &y)
{ // echange
  int z;
  z = x; x = y; y = z;
} // echange

// Retourne la position du minimum de t[i..j]
// On doit avoir i,j indices valides et i <= j
int posmin(const Tab t, int i, int j)
{ // posmin
  int k;
  int pos;
  pos = i;
  for(k=i+1;k<=j;k++)
      if (t[k] < t[pos]) pos = k;
  return pos;
} //fin posmin
```



```
// tri de t[i..j]
void tri(Tab t, int i, int j)
{ // tri
  int k;
  for(k=i;k<j;k++)
    echange(t[k],t[posmin(t,k,j)]);
} // fin tri


// lecture de t[i..j]
// On doit avoir i,j indices valides et i <= j
void lireTab(Tab t, int i, int j)
{ // lireTab
  int k;
  for(k=i;k<=j;k++) {
      cin >> t[k];
  } // for
} // fin lireTab
```

```
// écriture de t[i..j]
// On doit avoir i,j indices valides et i <= j
void ecrireTab(const Tab t, int i, int j)
{ // ecrireTab
  int k;
  for(k=i;k<=j;k++) {
      cout << t[k] << " ";
  } // for
  cout << endl;
} // fin ecrireTab

void main()
{ // tri_sélection
  Tab a;
  lireTab(a,0,N-1);
  tri(a,0,N-1);
  ecrireTab(a,0,N-1);
} // fin tri_sélection
```


Tri par insertion

- Idée: comme pour trier un paquet de cartes. On insère chaque élément du tableau un par un de gauche à droite à la place qui convient.
- Dans la première case, on laisse l'élément inchangé.
- On prend le deuxième élément et on l'insère à la bonne place dans la partie gauche du tableau.
- Le troisième est inséré à la bonne place parmi les éléments déjà triés dans la partie gauche du tableau.
- Et ainsi de suite, jusqu'au dernier élément.
- Le tableau est divisé en une partie déjà triée et une avec les éléments qui doivent encore être insérés.
- Exemple: Avec le tableau

  [12 5 78 23 6]

  on obtient

  [5 12] [78 23 6]

  [5 12 78] [23 6]

  [5 12 23 78] [6]

  [5 6 12 23 78]

Tri par insertion

```
/* Procédure pour trier un tableau en ordre
 * croissant par insertion.
 * On doit avoir i,j indices valides et i <= j.
 */

procédure tri_insertion(référence t:Tab,
                        valeur i,j:entier)
```

```
variable k:entier
début tri_insertion
// Invariant : t[i..k-1] est trié et i + 1 <= k <= j
    pour k <- i+1 à j faire
     insérer(t,i,k)
    fin pour
fin tri_insertion
```

Exercices Cours/TD

- Qu'est-ce que doit faire la procédure insérer ?
- Détailler la procédure insérer.

Exercices Cours/TD

1. Que fait le programme suivant ?

```
#include <iostream.h>

int f(int x){
  x++; // x=x+1;
  return x;
}

int g( int &x){
  x=x+1;  // x++;
  return x;
}

int main(){
  int n =5;
  cout << f(n) << endl;
  cout << f(n) << endl;
  cout << g(n) << endl;
  cout << g(n) << endl;
}
```

2. Reprenons l'exemple du compte bancaire : on veut représenter le compte par trois champs : **numéro du compte** (entier), **type du compte** (chéque ou épargne) et **solde** (entier). Définir un type `compte` qui permet de gérer ses trois informations. Ecrire ensuite des *accesseurs* et *modificateurs* pour chacun de ces trois champs. Un accesseur permet de savoir ce qu'un champ donné contient. Un modificateur permet de modifier le contenu d'un champ donné. A chaque fois se poser la question : fonction ou procédure ? Paramètre valeur, paramètre référence ?

Solutions

Tri par insertion:

```
/* insère l'élément t[j] à la bonne place dans
   le tableau t[i] à t[j-1]
   préconditions:  t[i] à t[j-1] est trié
                   et i < j                    */

procédure inserer(référence t:Tab,
                  valeur i,j: deux entiers)

variables k, cle : entier

début inserer
   k <- j
   cle <- t[k]
   tant que i < k et t[k-1] > cle faire
      t[k] <- t[k-1]
      k <- k - 1
   fin tant que
   t[k] <- cle
fin inserer
```

Invariant de la boucle:
Le tableau $t[i \ldots j]$ sans l'élément $t[k]$ est trié,
$t[k+1 \ldots j] > cle$ et $i \leq k$

Solutions

1.
```
#include <iostream.h>

int f(int x){
  x++; // x=x+1;
  return x;
}

int g( int &x){
  x=x+1;  // x++;
  return x;
}

int main(){
  int n =5;
  cout << f(n) << endl;
  cout << f(n) << endl;
  cout << g(n) << endl;
  cout << g(n) << endl;
}
```

Affiche $6\ 6\ 6\ 7$.

2. Compte bancaire :

```
#include <iostream.h>

const int max = 6;

typedef int cnum[max];
typedef enum ctype {cheque,epargne};

typedef struct compte {
  cnum num;
  ctype type;
  int solde;
  };

/* ici accesseur */
int acc_solde(compte c){
  return c.solde;
}

/* affiche le numero du compte */
void  affiche_num(compte c){
  int i;

  cout << "numero du compte : ";
  for (i=0; i<max ; i++)
    cout << c.num[i];
  cout << endl;
}

/* accesseur */
ctype acc_type(compte c){
```

```
  return c.type;
}

void affiche_type(ctype t){
  cout << "type du compte : ";
  switch(t) {
   case cheque   : cout << "cheque";    break;
   case epargne   : cout << "epargne";   break;
  }
  cout << endl;
}

void retrait(compte &c){
  int m;
  cout << "Combien retirez-vous ?" << endl;
  cin >> m;
  c.solde -= m;
}

void modif_num(compte &c, cnum n){
  int i;

  for (i=0; i<max ; i++)
  c.num[i] = n[i];
}

void modif_type(compte &c, ctype t){
  c.type = t;
}
```