TD n° 3 (Correction)

Passage de Paramètres par référence

Le but de ce TD est de comprendre le passage de paramètres par référence en C++ qui utilise le symbole noté &, appelé *ampersand*. Le timing est donné à titre indicatif. La plupart des questions sont courtes, et se traitent en 5 ou 10 minutes.

Correction : *Si les étudiants ont du mal à faire tout seul, ce qui est probable, les exercices seront traités sur un mode plus interactif : question-réponse à toute la classe, ou nominatif.*

Exercice 1. (10mm) AMPERSAND QUI ES-TU, QUE FAIS-TU ? Ce premier exercice très rapide a pour but de comprendre l'importance du symbole "et commercial" également appelé *ampersand* et noté &. Soit le programme suivant :

```
#include <iostream>
using namespace std;
void mystere (int a, int b, int &c, int d){
  c = a+b; d = a*b;
}
int main (){
  int e, f, g, h;
  cout << "donnez une valeur"; cin >> e;
  cout << "donnez une valeur"; cin >> f;
  mystere(e, f, g, h);
  cout << "g vaut " << g << ", h vaut " << h << endl;
  return 0;
}
```

1. Identifier et noter le(s) paramètre(s) formel(s) / le(s) paramètre(s) effectif(s)
2. Identifier et noter le mode de chaque paramètre (donnée, résultat ou donnée-résultat).
3. Qu'est censé faire ce programme ?
4. Corriger ce programme pour qu'il le fasse.

Correction : *Les paramètres formels sont donnés dans la définition, ce sont a b c d, les paramètres effectifs sont donnés à l'appel, ce sont e f g h ; les deux premiers sont des données, les deux derniers sont (supposés être) des résultats. Le programme est censé retourner la somme et le produit, il ne retourne pas le produit à cause d'un ampersand manquant ;*

Exercice 2. (40mm) PROCÉDURE, JE VAIS T'APPELER ! Cet exercice a pour but d'apprendre à appeler une procédure et à récupérer les résultats. Soit la procédure C++ suivante :

```
void classer (int a, int b, int &min, int &max){
  if (a < b) { min = a; max = b; }
  else { min = b; max = a; }
}
```

1. Identifier et noter le(s) paramètre(s) en donnée et le(s) paramètre(s) en résultat.
2. Que fait cette procédure ?
3. Appeler cette procédure depuis un programme principal qui donne directement deux valeurs entières à classer, et qui après l'appel, affiche les valeurs dans l'ordre croissant.
   Correction :

```
int main (){
  int e,f;
  classer(7, 5, e, f);
  cout << e << " plus petit que " << f << endl;
  return 0;
}
```

4. Créer un deuxième appel de cette procédure conçu de telle sorte que deux variables x et y soient modifiées (c'est-à-dire que leurs valeurs soient permutées) si $x > y$; les variables x et y sont donc mises à jour par l'appel. Comme précédemment, après l'appel, on affiche les valeurs dans l'ordre croissant.

   Correction :

```
int main (){
  int x,y;
  cout << " deux entiers please" << endl;
  cin >> x >> y;
  classer(x, y, x, y);
  cout << x << " plus petit que " << y << endl;
  return 0;
}
```

5. Écrire à présent un programme principal qui saisit trois nombres entiers positifs, et appelle "classer" trois fois (en s'inspirant de l'appel précédent) pour les trier. Bien sûr, le programme doit afficher les valeurs dans l'ordre croissant. Que remarquez-vous ?

Correction : *Il faut faire un tableau avec trois colonnes, pour a,b,c, et une ligne pour chaque appel de classer, et remplir le tableau en montrant comment le contenu est mis à jour par les appels.*

```
int main (){
  int a,b,c;
  cout << "trois entiers please" << endl;
  cin >> a >> b >> c;
  classer(a, b, a, b);
  classer(b, c, b, c);
  classer(a, b, a, b);
  cout << a << "<" << b << "<" << c << endl;
  return 0;
}
```

On remarque que c'est toujours les même deux variables qui sont passées en données et en résultat. Donc il y a factorisation possible. On devrait faire une fonction ne prenant que deux paramètres. C'est l'objet de l'exercice suivant.

Exercice 3. (20mm) PROCÉDURE, JE VAIS TE MODIFIER ! Le but de cet exercice est de faire évoluer une procédure existante. On va factoriser un peu plus le code de "classer", en exploitant le fait que dans le programme précédent de tri, les deux paramètres de type données sont aussi les paramètres résultats.

1. Ecrire l'en-tête de la procédure **ordonner** qui a seulement deux paramètres entiers qu'elle échange éventuellement de façon à ce que le deuxième soit toujours le plus grand. Au fait, c'est quoi ce mode de passage ?
2. Programmer le corps de la procédure.
3. Réécrire l'algorithme qui affiche les trois variables lues, dans l'ordre croissant, en utilisant cette nouvelle procédure **ordonner**. Que constatez-vous ?

   Correction : *Le mode de passage s'appelle donnée/résultat, à la fois on lit et on écrit dans les paramètres. On constate que c'est encore plus court, et encore plus clair, parce qu'on a factorisé encore plus de code. Yes.*

```
void ordonner (int &a, int &b){
  int temp;
  if (a>b) {temp=a; a=b; b=temp;}
}

int main (){
  int a,b,c;
  cout << "trois entiers please" << endl;
  cin >> a >> b >> c;
  ordonner(a,b); ordonner(b,c); ordonner(a,b);
  cout << a << "<" << b << "<" << c << endl;
  return 0;
}
```

Exercice 4. (40mm) PROCÉDURE JE VAIS TE CRÉER!! On va maintenant programmer des procédures qui font quelque chose de significatif (avec des boucles), et qui calculent deux résultats, donc nécessitent d'en transmettre au moins un par référence. Pour ne pas que l'exercice prenne trop de temps, on vous donne la majorité du code. On considère donc la fonction **find** qui renvoie vrai si une lettre donnée se trouve dans une chaîne de caractères. Le programme suivant itère sur tous les caractères, et compare le caractère numéro i de la chaîne avec le caractère recherché.

```
bool find(string maChaine, char monChar){
    bool result = false;
    for (int i = 0; i < maChaine.length(); i++)
        if (maChaine[i] == monChar) result = true;
    return result;
}
```

1. Appeler cette fonction depuis un programme principal qui donne directement les valeurs d'une chaîne et d'un caractère.

   Correction : *Le but du jeu est d'insister sur les différentes formes de syntaxe des appels et des retours entre fonctions et procédures.*

   ```
   int main(){
       if (find("bonjour",'n')) cout << "ca a l'air ok" << endl;
       return 0;
   }
   ```

2. Écrire l'en-tête d'une procédure **findAndWhere** qui, en plus, calcule à quelle position se trouve le caractère s'il s'y trouve. Si le caractère cherché est présent plusieurs fois, n'importe quelle occurrence convient. On passera deux paramètres par référence pour récupérer les deux résultats.
3. Écrire le corps de cette procédure en vous inspirant de **find**. Tester cette procédure avec un appel simple sans faire de saisie.

   Correction :

   ```
   void findAndWhere(string maChaine, char monChar, bool &result, int &ouca){
       result = false; ouca = -1;
       for (int i = 0; i < maChaine.length(); i++) {
           if (maChaine[i] == monChar)  {
               result = true; ouca = i;
           }
       }
   }
   int main(){
   ```

```
    bool f; int pos;
    findAndWhere("bonjour", 'n', f, pos);
    if (f)
        cout << "on a trouve n dans bonjour en position " << pos << end;
    return 0;
}
```

4. Modifier le code précédent pour que le booléen soit retourné sous forme fonctionnelle, i.e. avec une instruction `return`. Commencer par modifier l'en-tête. Tester cette deuxième procédure avec un appel simple.

   Correction :

```
bool findAndWhere2(string maChaine, char monChar, int &ouca){
    bool result = false; ouca = -1;
    for (int i = 0; i < maChaine.length(); i++)
        if (maChaine[i] == monChar) { result = true; ouca = i;};
    return result;
}
int main(){
    bool f; int pos;
    if (findAndWhere2("bonjour", 'n', pos))
        cout << "on a trouve n dans bonjour en position " << pos << endl;
    return 0;
}
```

5. OPTIONNEL : utiliser cette dernière procédure pour programmer une autre procédure qui remplace toutes les lettres minuscules d'une chaîne de caractères donnée par des majuscules, en itérant sur les lettres minuscules de l'alphabet.

   Correction :

```
void toUpperCase(string &maChaine){
    int pos;
    for (char monChar = 'a'; monChar <= 'z'; monChar++) {
        while (findAndWhere2(maChaine, monChar, pos)) {
            maChaine[pos] = maChaine[pos] + 'A' - 'a';
        }
    }
}
int main () {
    string ch="bonjour";
    toUpperCase(ch);
    cout << ch <<endl;
    return 0;
}
```

Exercice 5. (10mm) PROCÉDURE PARFOIS IL VAUT MIEUX SE PASSER DE TOI ?!
On suppose écrite une fonction qui renvoie le max de deux entiers.

```
int max2(int a, int b){
    if (a>b) return a;
    else return b;
}
```

Écrire le code d'une fonction qui calcule le max de 3 entiers, cela prend deux lignes ; le but de ce dernier tout petit exercice est de clarifier quand utiliser plutôt une fonction, et quand utiliser plutôt une procédure. Plus précisément, quand est-il préférable de renvoyer un résultat via un paramètre passé par référence, ou avec un return ?

Correction :

```
#include <iostream>
using namespace std ;
int max2(int a, int b){
    if (a>b) return a;
    else return b;
}
int max3(int a, int b, int c){
    return max2(a,max2(b,c));
}
int main(){
    cout << max3(5,7,6);
    return 0;
}
```

Dans la grande majorité des cas, on utilise le passage par référence seulement s'il y a deux résultats, s'il n'y en a qu'un seul on utilise un return. En effet, dans ce dernier cas, on peut enchaîner les appels de fonction sans avoir à stocker de résultats dans des variables intermédiaires. Exception : si on transmet des résultats qui prennent de la place, pour ne pas les recopier, voir cours.