Chapitre 7

Fonctions III

Il y a deux méthodes pour passer des variables en paramètre dans une fonction : le passage par valeur et le passage par référence. Nous décrivons dans ce cours la deuxième technique qui consiste à passer non plus la valeur des variables comme paramètre, mais à passer les adresses des variables. Il n'y a donc plus de copie, plus de variable locale. Toute modification du paramètre dans la fonction appelée entraîne la modification de la variable passée en paramètre.

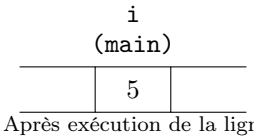
7.1 Passage par référence

Le *passage des paramètres par référence* consiste à passer non plus la valeur des variables comme paramètre, mais à passer les adresses des variables. Il n'y a donc plus de copie, plus de variable locale. Toute modification des paramètres dans la fonction appelée entraîne la modification des variables dont l'adresse est passée en paramètre. Ainsi, si l'on souhaite que la modification d'un paramètre dans la fonction se répercute dans le reste du programme, il faut alors passer le paramètre par référence. Dans ce cas, **la fonction ne travaille plus sur une copie mais directement sur la variable**. Pour cela, il faut rajouter le caractère & entre le type et le nom du paramètre. Ainsi, l'exécution du code :

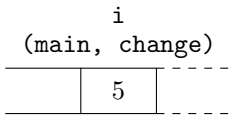
```
void change(int &i)
{
  affichage("i = ", i , "\n");
  i = 3;
  affichage("i = " , i , "\n");
}

int main()
{
  int i;
  i = 5;
  affichage("i = " , i , "\n");
  change(i);
  affichage("i = " , i , "\n");
  return 0;
}
```

affichage le résultat :

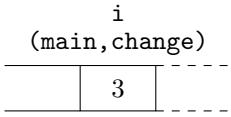
```
i = 5
i = 5
i = 3
i = 3
```



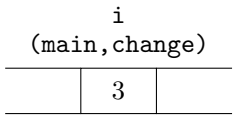
(a) Après exécution de la ligne 11



(b) Avant exécution de la ligne 4



(c) Après exécution de la ligne 4



(d) Après exécution de la ligne 13

FIGURE 7.1 – Passage d'un paramètre par référence

Dans la fonction `main` est déclarée une variable `i`, initialisée à la ligne 11. L'état de la mémoire après l'exécution de cette instruction est décrit dans la figure 7.1(a). Seule la variable `i` de la fonction `main` est présente en mémoire et sa valeur est `5`. L'instruction de ligne 12 affiche `i=5`. La fonction `change` est ensuite appelée avec passage par référence `&i`. La variable locale de la fonction qui s'appelle également `i`, partagera le même espace mémoire que la variable `i` déclarée dans la fonction main (instruction 11). L'état de la mémoire correspond à celui indiqué dans la figure 7.1(b). L'instruction de la ligne 3 affiche donc `i=5`. Puis, la valeur du seul espace mémoire partagé par les deux fonctions est changée à la ligne 4 (voir figure 7.1(c)). La ligne 5 affiche alors `i=3`. À ce moment-là, l'exécution de `change` est terminée et entraîne la modification de la variable passée en paramètre qui porte le même nom `i`. Alors, après l'exécution de la ligne 13, la variable `i` de la fonction main est modifiée comme décrit par la figure 4.2(d). La ligne 14 affiche donc `i=3`.

Le passage par référence est important car il permet de **transmettre plusieurs valeurs de sortie** au programme appelant même si une seule valeur peut être retournée par la fonction. Ainsi, le code d'une fonction calculant l'aire et le périmètre d'un rectangle peut être :

```
1  void airePerimetreRectangle(double longueur, double largeur, double &aire, double &perim)
2  {
3    aire = longueur * largeur;
4    perim = 2 * (longueur + largeur);
5  }
6
7  int main()
8  {
9    double a, p;
10   airePerimetreRectangle(10.0,4.5,a,p);
11   affichage("Aire du rectangle : ",a , ", périmètre : ", p , "\n");
12   return 0;
13 }
```

L'exécution du code précédent affiche : `Aire du rectangle : 45, périmètre : 29`.

Dans l'exemple précédent, une des deux valeurs calculées (l'aire ou le périmètre) aurait pu être retournée par la fonction avec l'instruction `return`. Il y a donc plusieurs manières de coder et le choix est laissé au codeur.

7.2 Avantages et inconvénients des deux méthodes

Les passages par références sont plus rapides et plus économes en mémoire que les passages par valeur, puisque les étapes de la création de la variable locale et la copie de la valeur ne sont pas faites. Il faut donc éviter les passages par valeur dans les cas de plusieurs appels de fonction ou de fonctions travaillant avec de grands volumes de données. Par contre les passages par valeurs permettent d'éviter de détruire par mégarde les variables passées en paramètre.

TD7 : Fonctions

✓ Exercice 1 : Compréhension de programme*

Question 1.1 : Est-il vrai que les passages par référence sont plus économes en mémoire que les passages par valeur.

Question 1.2 : Quel est le rôle du symbole & utilisé dans les paramètres d'une fonction donnée.

Question 1.3 : Déterminez ce qu'affiche ce programme.

```
void affiche2int(int a, int b)
{
  affichage(a , "-->" , b , "\n");
}
void incr1(int x)
{
  x=x+1;
}
void incr2(int &x)
{
  x=x+1 ;
}
void decr1(int x)
{
  x=x-1;
}
void decr2(int &x)
{
  x=x-1;
}

int main()
{
  int i = 1;
  int j = 1;
  affiche2int ( i , j);
  incr2(i);
  affiche2int ( i , j);
  decr1(j);
  affiche2int ( i , j);
  decr2(j);
  affiche2int ( i , j);

  while (i != j)
  {
    incr1(j);
    decr2(i);
  }
  affiche2int ( i, j);
  return 0;
}
```

Question 1.4 : Que se passe t-il si l'en-tête de la fonction `decr2` devient `void decr2(int x)` ?

Question 1.5 : Qu'affiche le programme suivant ?

```
int somme2nb(int &a, int &b)
{
  return a+b;
}

int main()
{
  int n = 4;
  affichage("res = " , somme2nb(n,3) , "\n");
  return 0;
}
```

® Exercice 2 : Opération arithmétique*

Écrire une fonction qui prend deux 2 nombres entiers en paramètre et qui calcule et stocke la somme , la différence, le produit de ces 2 entiers.

✓ Exercice 3 : Moyenne, minimum et maximum de deux nombres*

Question 3.1 : Écrire la fonction qui reçoit 2 nombres entiers en paramètre et stocke, le maximum, le minimum et la moyenne des deux variables.

Question 3.2 : On veut maintenant avoir uniquement deux variables qui utilisent le passage par référence (le minimum et le maximum). Que proposez-vous pour renvoyer la valeur de la moyenne ?

✓ Exercice 4 : Échange de variables**

Question 4.1 : Écrire une fonction `echange(a,b)` qui sert à échanger les valeurs de 2 entiers `a` et `b` passés en référence.

Question 4.2 : Écrivez la fonction `void tri(int &a, int & b, int &c)` qui permute les valeurs de `a`, `b` et `c` de sorte que $a \leq b \leq c$. On utilisera la fonction `echange()`.

® Exercice 5 : Opérateurs logiques*

Écrire une fonction qui calcule et stocke le résultat d'un OU d'un ET entre deux variables booléennes passées en paramètres.

® Exercice 6 : Surface et circonférence**

Écrire une fonction qui calcule la surface et la circonférence d'un cercle de rayon R.

TP7 : Fonction

Exercice 7 : Manipulation de l'heure

Définir une heure nécessite 3 champs de type entier appelés `h`, `min` et `sec` correspondant aux heure, minute et seconde.

Question 7.1 : Définir la fonction `void ajouteUneSeconde(...)` ajoutant une seconde à une heure décrite par les trois variables passées en paramètre.

Question 7.2 : Définir la fonction `afficheHeure` prenant en paramètre une heure (décrite par les trois variables) et l'affichant sous la forme `heure:minute:seconde`.

Question 7.3 : Écrire un programme principal déclarant les trois variables `h, min, sec` initialisée à 0 heure, 0 minute et 0 secondes. Ajouter 100 fois une seconde et afficher les différentes heures obtenues.

Exercice 8 : Jeu du 21*

Le jeu du 21 est une version simplifiée du jeu de dés appelé le 421. Le jeu du 21 se joue avec deux dés et deux joueurs. À chaque combinaison de dés est associé un score. Chaque joueur a 3 lancés de dés pour obtenir le meilleur score possible (associé à la combinaison de dés obtenue). Le perdant est celui qui ne parvient pas à obtenir un score au moins égal à celui obtenu précédemment par l'adversaire.

Les scores sont obtenus par combinaison des deux dés. La meilleure combinaison est un 2 et un 1. Les combinaisons suivantes sont les doubles (double 6, double 5, ..., double 1). Le reste des combinaisons est ordonné par le chiffre le plus grand puis par le deuxième chiffre. Voici la liste complète des combinaisons possibles (de la meilleure à la moins bonne) :
2-1, 6-6, 5-5, 4-4, 3-3, 2-2, 1-1, 6-5, 6-4, 6-3, 6-2, 6-1, 5-4, 5-3, 5-2, 5-1, 4-3, 4-2, 4-1, 3-2.

Le joueur dont c'est le tour commence par lancer les deux dés. Il peut alors décider d'effectuer une relance, c'est-à-dire de relancer un des deux dés ou les deux. Son tour se termine après maximum deux relances et on ne considère que la combinaison de dés obtenue à la fin de son tour.

Question 8.1 : Écrire une fonction permettant à l'utilisateur de saisir les dés qu'il souhaite relancer : 0 s'il ne veut faire aucune relance, 1 s'il souhaite relancer uniquement le premier dé, 2 s'il souhaite relancer uniquement le deuxième dé et 3 s'il souhaite relancer les deux dés. Le choix de l'utilisateur sera retourné par la fonction.

Question 8.2 : En utilisant la fonction `lancerDe`, écrire une fonction `lancerDesChoix` qui permet de communiquer au plus deux lancés de dés à l'appelant en fonction de la valeur de choix (nombre compris en entre 0 et 3) reçu en paramètre.

Cartouche de la fonction `lancerDesChoix` :

```
/*!
 *  \fn void lancerDesChoix()
 * Communique au plus deux lancés de dés au programme appelant en fonction de
 * la valeur de choix
 * \param de1 (sortie) : Premier lancé de dé
 * \param de2 (sortie) : Deuxième lancé de dé
 * \param choix (entrée) : nombre compris entre 0 et 3 et permettant d'indiquer
 *                         à la fonction comment "remplir" de1 et de2
 */
```

Le tableau suivant décrit le fonctionnement de `lancerDesChoix` :

choix	de1	de2
0	aucun lancé	aucun lancé
1	lancé	aucun lancé
2	aucun lancé	lancé
3	lancé	lancé

Question 8.3 : Écrire une fonction `afficherUnDe` qui affiche le dé passé en paramètre sous forme "graphique". Ainsi, si la valeur du dé est 5, la fonction doit afficher :

```
/////////
/ *   * /
/   *   /
/ *   * /
/////////
```

Question 8.4 : Écrire la fonction `afficherDes` qui affiche sous forme graphique le résultat d'un lancer de dés.

Question 8.5 : Il est nécessaire, pour comparer les combinaisons de dés adversaires, d'associer à toute combinaison de dés un score. On rappelle la liste combinaisons possibles (de la meilleure à la moins bonne) :
2-1, 6-6, 5-5, 4-4, 3-3, 2-2, 1-1, 6-5, 6-4, 6-3, 6-2, 6-1, 5-4, 5-3, 5-2, 5-1, 4-3, 4-2, 4-1, 3-2.

Ce score doit respecter l'ordre des combinaisons, c'est-à-dire qu'une combinaison meilleure qu'une autre doit avoir un score plus grand. Par exemple, le score associé à la combinaison 1-1 doit être supérieur au score associé à la combinaison 6-5.

Déterminer une manière simple de calculer un tel score pour une combinaison de dés donnée. Vous pouvez pour cela associer un très gros score (10^3 par exemple) à la combinaison 2-1, puis des scores de l'ordre des centaines aux doubles et des scores correspondant à la valeur qu'elles décrivent (5-4 et 54) aux combinaisons restantes.

Définir une fonction retournant le score associé à une combinaison de dés passée en paramètre.

Question 8.6 : Définir une fonction `tour` simulant le tour d'un joueur. Cette fonction prendra en paramètre un booléen valant `true` si c'est le tour du premier joueur, et `false` sinon. La fonction devra afficher `Tour du joueur` suivi du numéro du joueur (1 ou 2). Elle permettra ensuite au joueur de lancer les dés, d'effectuer les relances si nécessaire puis retournera le score obtenu par le joueur.

Question 8.7 : Définir la fonction `partie` permettant de jouer au jeu du 21.

Question 8.8 : Écrire la fonction principale permettant de jouer au jeu du 21.

Exercice 9 : Tablette de chocolat [1]**

Une tablette (ou plaquette) de chocolat est une matrice de carreaux de chocolat que l'on peut couper selon les lignes ou les colonnes qui séparent les carreaux. La hauteur et la largeur se comptent en nombre de carreaux. On a peint en vert un carreau dans le coin d'une tablette. Deux joueurs coupent tour à tour la tablette, au choix, selon une ligne ou une colonne (pas nécessairement près du bord). L'objectif est de ne pas être le joueur qui se retrouve avec le carreau vert (la tablette de hauteur 1 et de largeur 1). Le but de cet exercice est de programmer ce jeu.

On suppose dans un premier temps que deux joueurs humains jouent l'un contre l'autre. Une tablette de chocolat sera stockée en mémoire à l'aide de deux entiers correspondant au nombre de lignes et de colonnes (en carreaux de chocolat). Le carré de chocolat peint en vert sera sur la première ligne et la première colonne.

Question 9.1 : Écrire une fonction `Saisie()` effectuant une saisie répétée d'un entier, jusqu'à ce que cet entier soit un nombre entier compris entre `min` et `max`, `min` et `max` étant deux paramètres de la fonction.

1. Exercice issu du cours de C de Pierre Boudes

Remarque : la majuscule à `Saisie()` est importante, la fonction `saisie` ayyant déjà été définie et prenant comme paramètre la variable dont la valeur doit êre saisie.

Question 9.2 : Écrire une fonction permettant d'initialiser le nombre de lignes et de colonnes de la tablette avec des valeurs aléatoires comprises entre 5 et 20 pour le nombre de lignes et entre 10 et 30 pour le nombre de colonnes. De plus, le nombre de lignes doit être différent du nombre de colonnes. Pour générer un nombre aléatoire, utiliser la fonction `rand()` (c.f., TD précédent, exercice nombre magique).

Question 9.3 : Écrire une fonction `afficheLigneTablette` prenant en paramètre le numéro de la ligne à afficher et le nombre de colonnes. Les carreaux "normaux" seront représentés par la lettre `'X'` et le carré peint en vert par la lettre `'N'`.

Question 9.4 : Écrire une fonction `afficheTablette()` prenant en paramètre le nombre de lignes et de colonnes et affichant la tablette de chocolat. Cette fonction appelera la fonction définie précédemment.

Question 9.5 : Écrire une fonction `perdu()` qui prend en paramètre le nombre de lignes et de colonnes de la tablette et retourne 1 si le joueur a perdu (le nombre de lignes et le nombre de colonnes sont égaux à 1), 0 sinon.

Question 9.6 : À chaque tour, un joueur doit choisir s'il supprime des lignes ou des colonnes. Pour exprimer ce choix, l'utilisateur devra saisir l'entier 0 s'il choisit de supprimer des lignes, ou l'entier 1 pour les colonnes. Écrire une fonction `saisieLigneOuColonnes()` prenant en paramètre le nombre de lignes et de colonnes de la tablette. Cette fonction effectuera une saisie répétée (utilisation de la fonction `Saisie()`) jusqu'à ce que l'utilisateur ait saisi 0 ou 1. De plus, s'il ne reste qu'une seule ligne ou qu'une seule colonne, alors aucun choix ne sera proposé au joueur. La fonction renverra 0 si l'utilisateur a choisi de supprimer des lignes, et 1 sinon.

Question 9.7 : Écrire le programme principal permettant de jouer au jeu de la tablette.

Question 9.8 : Il existe une stratégie gagnante pour ce problème. Déterminer cette stratégie gagnante et programmer une version du jeu de la tablette pour un joueur de manière à ce que le joueur perde systématiquement, quel que soit le nombre de lignes ou de colonnes qu'il retire durant la partie (On suppose que l'ordinateur commence à jouer).