CHAPITRE 6 : Les fonctions en langage C

Il est souhaitable, pour diverses raisons, de **décomposer** un problème en plusieurs sous-tâches, et de programmer ces sous-tâches comme des blocs indépendants. C'est le concept de la programmation modulaire qui utilise des sous-programmes.

En langage C, il y a une seule sorte de sous-programmes : les fonctions. On commence d'abord avec l'écriture et les appels des fonctions. On verra plus tard les avantages de la programmation modulaire.

A)Fonction naturelle avec return

Une fonction "naturelle" est un sous-programme permettant de calculer et de "retourner" (avec **return**) un seul résultat de type simple (float, char et int dans le cadre du cours IFT 1810) à partir (fréquemment) d'une liste de paramètres (des données).

1) Syntaxe

```
type_résultat_de_retour nom_fonction(liste de paramètre(s))
{
   déclarations locales si nécessaire
   calcul et retour (avec return) du résultat
}
```

Illustration de la syntaxe à l'aide d'un exemple

La fonction suivante permet de calculer et de retourner la plus grande valeur parmi deux réels.

```
float plusGrand(float x, float y)
{
   if (x > y)
      return x;
   else
      return y;
}
```

Les déclarations locales sont "localement" utiles pour le calcul du résultat à partir des données. Pour la fonction "plusGrand" ci-dessus, on n'a besoin d'aucune déclaration locale.

La fonction suivante permet de calculer le prix total à payer d'un article taxable à partir du prix de l'article. Dans le calcul du prix total à payer, on a besoin des informations intermédiaires. On les déclare "localement" à l'intérieur de cette fonction.

```
float prixAPayer(float prix)
{
   /* déclarations locales :  */
   const float TAUX_TPS = 0.06,
               TAUX_TVQ = 0.075;
   float tps, tvq;

   /* calcul du prix total à payer : prix + les 2 taxes */
   tps = prix * TAUX_TPS;
   tvq = (prix+TPS) * TAUX_TVQ;
   return prix + tps + tvq;
}
```

2) Remarques

a) Le nom de la fonction ne contient pas de résultat de retour comme en PASCAL.
b) Il faut utiliser le résultat retourné dans un bon contexte (affichage, affectation, comparaison, ...).
c) L'instruction "return" provoque la fin de la fonction. On revient ensuite à l'endroit où la fonction a été appellée. Il doit avoir au moins une instruction "return" dans la fonction.
d) Dans l'en-tête, on ne peut pas regrouper les paramètres de même type :

```
float plusGrand(float x, y)  /* erroné! */
```

e) Ne pas terminer l'en-tête par un point virgule :

```
float plusGrand(float x, float y); /* erroné! */
```

3) Exemples

Exemple 1 : fonction qui retourne un entier comme résultat

Écrire un programme permettant de saisir un entier n supérieur à zéro (par exemple : 5284).

Le programme calcule (par les fonctions) et affiche à l'écran :

- la somme des chiffres du nombre n (ici 19);

- l'envers du nombre n (ici 4825).

Solution

```
#include <stdio.h>

int somChif(int n)
{
   int somme = 0;

   while (n > 0){
      somme += n % 10;
      n /= 10;
   }
   return somme;
}

int envers(int n)
{
   int k = 0;

   while (n > 0){
       k = k * 10 + n % 10;
       n /= 10;
   }
   return k;
}

void main()
{
   int nombre;

   printf("Entrez un entier positif : ");
   scanf("%d", &nombre);

   printf("La somme des chiffres de %d est %d\n\n", nombre, somChif(nombre));
   printf("L'envers du nombre %d est %d\n\n", nombre, envers(nombre));
   printf("Cliquez sur le bouton de fermeture ");
}
```

Exécution

```
Entrez un entier positif : 875
La somme des chiffres de 875 est 20

L'envers du nombre 875 est 578
```

Exemple 2 : fonction qui retourne un réel comme résultat

Écrire une fonction permettant de calculer le bonus dépendant du poste de travail :

poste 'A' (analyste) : 234.50 $ de bonus
poste 'P' (programmeur) : 210.90 $ de bonus
poste 'O' (opérateur) : 189.00 $ de bonus

Écrire quelques appels (utilisations) valides de cette fonction.

Solution

```
float montantBonus(char poste)
{
   const float BONUS_A = 234.50,
               BONUS_P = 210.90,
               BONUS_O = 189.00;
   float bonus;

   switch (toupper(poste)){
      case 'A': bonus = BONUS_A;
                break ;
      case 'P': bonus = BONUS_P;
                break ;
      case 'O': bonus = BONUS_O;
   }

   return bonus;
}
```

Quelques appels valides

```
a) dans l'affichage (plus courant)

      printf("bonus d'un analyste : %6.2f\n", montantBonus('A'));

b) dans une affectation :

      float salHebdo, salTotal;
      char  poste;
      ...
      salTotal = salHebdo + montantBonus(poste);

c) dans une comparaison :

      if (montantBonus (poste) > 200.00)
         printf("Pas si mal\n");
```

Exemple 3 : fonction qui retourne un caractère comme résultat

Écrire une fonction permettant de jouer le même rôle que "toupper" (conversion d'un caractère en majuscule).

Écrire un appel valide de cette fonction.

Solution

```
char majuscule(char c)
{
   if (c >= 'a' && c <= 'z')     /* lettre minuscule */
      return c + 'A' - 'a';       /* code ASCII */
   else
      return c;
}
```

Un appel valide :

```
printf("Le caractere %c en majuscule est %c\n", 'e', majuscule('e'));
```

Voici un extrait de la table des codes ASCII :

```
Caractere                    Numero
...                           ...
'0'                           48
'1'                           49
...                           ...
'A'                           65
'B'                           66
...                           ...
'a'                           97
'b'                           98
...                           ...
```

Supposons que la variable c contienne un 'b' minuscule, alors c + 'A' - 'a' vaut 98 + 65 – 97 = 66 (le langage C fait la conversion automatique pour interpréter certains résultats quand il y a un conflit de types). Quel est le caractère dont l'ordre est 66? C'est la lettre 'B' en majuscule. La fonction retourne cette lettre comme résultat de la conversion.

Exemple 4 : fonction qui retourne vrai ou faux comme résultat

Écrire une fonction permettant de retourner vrai (1) ou faux (0) selon qu'un caractère est une voyelle ou non.

Écrire une instruction utilisant cette fonction pour afficher les 20 consonnes en majuscules à l'écran.

Solution

```
int voyelle(char lettre)
{
   int reponse;
   switch (toupper(lettre)){
      case 'A':
```

```
      case 'E':
      case 'I':
      case 'O':
      case 'U':
      case 'Y': reponse = 1; /* VRAI  */
                break;
      default :reponse = 0;  /* Faux */
   }
   return reponse;
}
```

L'affichage des 20 consonnes en majuscules à l'écran se fait comme suit :

```
char lettre;

for (lettre = 'A'; lettre <= 'Z'; lettre++)
   if (!voyelle(lettre))
      printf("%c", lettre);
printf("\n\n");
```

Remarque

Pour plus de clarté, on peut aussi utiliser les #define dans le cas des fonctions booléennes. Exemple : écrire une fonction qui retourne vrai ou faux selon qu'un entier n est pair ou non.

```
int estPair(int n)
{
   #define VRAI 1
   #define FAUX 0

   if (n % 2 == 0)
      return VRAI;
   else
      return FAUX;
}
```

N.B. : Plusieurs autres exemples seront présentés avec les paramètres de type tableau.

B) Fonction de type void (pas de return)

Ce genre de fonction est semblable à "Sub" en Visual Basic (ou "Procedure en PASCAL). Elle réalise une ou quelques tâches (trier, chercher, compter-afficher, calculer, etc.).
Certains livres présentent la fonction principale comme fonction qui retourne un entier comme résultat :

```
int main()
{
   ...
   return 0;
}
```

Cette manière permet d'éviter des avertissements (warning) à la compilation (au lieu d'en-tête main tout court). Cette méthode n'est pas très compréhensible pour les débutants en programmation. Il est préférable d'utiliser une fonction de type void :

```
void main() /* rien à retourner */
{
   ...
}
```

1) Syntaxe

```
void nom_fonction(liste de paramètre(s))
{
   déclarations locales

   suites d'instructions réalisant la tâche confiée à la fonction
   (s'il y a des résultats de retour, ce sont des paramètres
   transmis par pointeurs ou par référence(on verra plus en détail
   plus tard))
}
```

Pour le nom de la fonction (nom_fonction), on suggère d'utiliser le nom d'un verbe (ou venant d'un verbe) qui résume la tâche de la fonction :

```
void calculer(...)
void trier(...)
void lireRemplir(...)
etc.
```

2) Exemples

Premier cas

Quand on utilise des résultats à l'intérieur du corps d'une fonction, on n'a que des paramètres transmis par valeur.

Exemple 1

Écrire une fonction permettant de compter et d'afficher le nombre de diviseurs d'un entier n positif donné. Écrire 2 appels permettant d'afficher le nombre de diviseurs de 720 et 984.

Solution

```
void compter(int n)
{
   int k = 1, /* n est un diviseur de lui-même */
       i;     /* utilisé dans la boucle for */

   for (i = 1; i <= n / 2; i++)
      if (n % i == 0)
         k++ ;

   printf("Le nombre de diviseurs de %d est %d\n", n, k);
}
```

Appels

```
compter(720);
compter(984);
```

Exemple 2

Réaliser un programme permettant de :

- déclarer et remplir d'un tableau d'au maximum 10 000 entiers avec des valeurs aléatoires entre 500 et 30 000
- afficher le contenu partiel du tableau : les 10 ers et les 5 derniers éléments du tableau
- déterminer et afficher la valeur minimale et maximale du tableau
- trier le tableau en ordre croissant
- afficher le contenu partiel du tableau : les 7ers et les 3 derniers éléments du tableau après le tri

```
#include <stdio.h>
#include <stdlib.h> /* librairie standard en C */
#include <time.h>   /* pour la fonction time(...)  */



    /* Cette fonction retourne un nombre arbitraire entre 2 bornes */
      int aleatoire(int borneInf, int borneSup) {

         return    rand() % (borneSup - borneInf + 1)  + borneInf;
     }
```

```
 void afficher(int tableau[], int nbElem, int debut, int fin)
 {
      int i;
      printf("Liste des %d premiers et %d derniers elements:\n",
                    debut, fin);
      for (i = 0; i < nbElem ; i++)
        if ( i < debut || i >= nbElem-fin)
             printf("%4d) %10d\n", i, tableau[i]);
        else
           if ( i == debut )
             printf("%4d) etc . . .\n", i);
      printf("\n\n");
 }

 void determinerMinMax(int tableau[], int nbElem)
 {
      int i;
      int mini, maxi ;
      mini = maxi = tableau[0];

      for (i = 0; i < nbElem ; i++)
      {
        if ( tableau[i] < mini)
                mini = tableau[i];
         if ( tableau[i] > maxi)
                maxi = tableau[i];
      }

      printf("Valeur la plus petite : %6d\n", mini);
      printf("Valeur la plus grande : %6d\n", maxi);
      printf("\n\n");
 }

void trier(int tableau[], int nbElem)
  {
      int i, j, indMin;
      int tempo;

       for (i = 0; i < nbElem - 1; i++)
       {
          indMin = i;

           for (j = i + 1; j < nbElem; j++)
              if (tableau[j] < tableau[indMin])
                 indMin = j;

           if (indMin != i){
              tempo      = tableau[i];
              tableau[i] = tableau[indMin];
              tableau[indMin] = tempo;

            }
       }
```

```
       }


int main() {

   #define   BORNE1    10000 /* taille du tableau  */
   const int BORNE2  =  9000, /* tableau contient entre 9 000 et 10 000
entiers */
             VAL_MIN =   500, /* valeur minimale d'un élément */
             VAL_MAX = 30000; /* valeur maximale d'un élément */

   int valeur[BORNE1];
   int nbElem; /* nombre d'éléments du tableau */
   int i;

   srand(time(0));

   nbElem = aleatoire(BORNE2, BORNE1);
   for (i = 0; i < nbElem; i++)
     valeur[i] = aleatoire(VAL_MIN, VAL_MAX);

   afficher(valeur, nbElem, 5, 10); /* afficher les 5 premiers et 10
derniers éléments */
   determinerMinMax(valeur, nbElem);
   trier(valeur, nbElem);
   afficher(valeur, nbElem, 7, 3); /* afficher les 7 premiers et 3
derniers éléments */


   printf("\n");
   system("pause");

   return 0;
}
/* Exécution :

Liste des 5 premiers et 10 derniers elements:
   0)       2202
   1)       4581
   2)       9167
   3)      28226
   4)      22785
   5) etc . . .
9942)       1218
9943)      15135
9944)       3465
9945)      27833
9946)      26386
9947)       7867
9948)       1299
9949)      19411
9950)      19128
```

```
9951)      21296


Valeur la plus petite :    500
Valeur la plus grande :  29988


Liste des 7 premiers et 3 derniers elements:
   0)        500
   1)        500
   2)        501
   3)        502
   4)        507
   5)        508
   6)        510
   7) etc . . .
9949)      29978
9950)      29982
9951)      29988


Appuyez sur une touche pour continuer... */
```

Choix de type de fonction (return vs void):

Pour le niveau du IFT 1810 :

> La fonction a t-elle une seule tâche de calculer (déterminer, compter, …) un seul résultat de type simple ?
>
> Si oui => choisir une fonction avec return
>
> Autres cas : choisir une fonction de type void

Exemples de justification :

1. Écrire une fonction pour calculer la taille moyenne:

   A-t-on une seule tâche de calculer un seul résultat de type simple?

- Oui : la taille moyenne est le seul résultat de type réel

On peut confier cette tâche à une fonction avec return :

```
float moyenne ( float taille[], int nbPers) {
        float somme = 0 .0 ;
        int i ;

        for ( i = 0 ; i < nbPers ; i++)
                somme += taille[i];

        return somme / nbPers;
}
```

2. Écrire une fonction pour compter le nombre de personnes dont la taille dépasse 1.90 mètre

A-t-on une seule tâche de calculer un seul résultat de type simple?

- Oui : compter le nombre de personnes qui satisfont une condition. Ce seul résultat est de type entier. (simple).

On peut confier cette tâche à une fonction avec return :

```
int  leNombre ( float taille[], int nbPers, float borne) {
        int n = 0, i ;

        for ( i = 0 ; i < nbPers ; i++)
                if ( taille[i] > borne)
                        n++; /* une personne de plus dont
                                la taille dépasse la borne */
        return n ;
}
```

3. Écrire une fonction pour déterminer la taille la plus grande

A-t-on une seule tâche de calculer un seul résultat de type simple?

Oui : déterminer la taille la plus grande qui est le seul résultat de type réel (type simple).

On peut confier cette tâche à une fonction avec return :

```
float taillePG ( float taille[], int nbPers) {
        float tailleMax = 0.0 ; /* une taille suffisamment petite */
        int i;

        for ( i = 0 ; i < nbPers ; i++)
                if ( taille[i] > tailleMax)
                        tailleMax = taille[i];

        return tailleMax;
}
```

4. Écrire une fonction pour afficher le contenu des 3 tableaux : sexe, taille et poids :

   A-t-on une seule tâche de calculer un seul résultat de type simple?
   Non : choisir une fonction de type void.
   Que fait la fonction ? – elle affiche le contenu des 3 tableaux

   On peut confier cette tâche à une fonction de type void :

```
void afficher ( char sexe[], float taille[], float poids[], int nbPers)
{
        int i;
        printf("Liste des personnes :\n");

        for ( i = 0 ; i < nbPers ; i++)
                printf("%3c %6.2f %7.1f\n", sexe[i], taille[i],
                                poids[i]);
}
```

5. Écrire une fonction pour trier le contenu des 3 tableaux : sexe, taille et poids selon les poids

A-t-on une seule tâche de calculer un seul résultat de type simple?
Non : choisir une fonction de type void.
Que fait la fonction ? – elle trie les 3 tableaux selon les poids

On peut confier cette tâche à une fonction de type void :

```
void trier ( char sexe[], float taille[], float poids[], int nbPers) {
        . . . etc . . .
}
```

6. Écrire une fonction pour lire un fichier, remplir les 3 tableaux sexe, taille et poids, compter le nombre effectif de personnes lues.

A-t-on un résultat de type simple?
Oui : le nombre effectif de personnes lues
Mais: est-elle la seule tâche de la fonction ?
pas du tout : elle lit, elle remplit, elle compte
Selon notre modèle, on confie cette tâche à une fonction de type void.
Comme ce n'est pas une fonction avec return, comment "retourner" le compteur du nombre de personnes lues ?

On ne le retourne pas! on transmet via pointeur :

```
void lireRemplir( char sexe[], float taille[], float poids[], int * p)
{
        . . . etc . . .
}
```

Conseils d'écriture une fonction avec return:

1. Allure d'une fonction avec return :

```
type    nom  ( liste de paramètre(s) ) {
```

```
        déclaration locale si nécessaire
        instruction(s) permettant de calculer et
    de
        retourner le résultat
    }
```

2. type : c'est le type du résultat à retourner (int, float, char, . . .)

3. nom : c'est le nom pour "appeler" plus tard cette fonction
exemples :

```
                        float moyenne( ….)
                        int   leNombre( . . .)
                        char enMajuscule ( . . .)
```

4. liste de paramètre(s) :
Informations à partir desquelles on calcule le résultat. Pour IFT 1810,
il est très rare que cette liste soit vide.

à partir de quoi on calcule la taille moyenne ?
tableau des tailles, le nombre de personne

```
                  float moyTaille ( float taille[],
int nbPers) { . . .
```

Par contre, après réflexions : on calcule aussi le poids moyen, le
salaire moyen, la note moyenne du final, etc . . .
il est préférable d'utiliser un en-tête plus général comme le suivant:

```
               float moyenne ( float tableau[], int
nbElements ) { . . .
```

a) paramètres de type simple : entier, réel, char , . . .

```
                        type  nom
```

b) paramètres de type tableau à un seul indice (pour IFT 1810):

```
                        type nom[]
```

c) paramètres de type tableau à 2 indices :

```
                         type nom [] [BORNE]
```

exemples :

```
    /* cette fonction calcule et retourne la
somme des valeurs sur
         une diagonale d'un carré magique n x n
(n lignes, n colonnes)
    */
    int somDiagonale ( int carreMagique[]
[MAX_ORDRE], int n)
    etc . . .

    /* cette fonction détermine et retourne la
valeur la plus grande
         d'un tableau de n réels
    */
    float plusGrand ( float tableau[], int n )
```

5. déclarations locales (qui sont localement utiles pour calculer le résultat) : dépendant de votre logique et de degrés de complexité de la fonction. Pour traiter des tableaux, on a besoin souvent de boucle for, on déclare `int i ;` localement (entre autres) pour parcourir les indices des tableaux.

Exemple1 : écrire une fonction permettant de calculer le périmètre d'un carré

```
          float perimetre ( float cote ) {
```

```
            return  4 * cote ;
        }
```

Le calcul est si simple qu'on n'a pas besoin d'informations intermédiaires, donc pas de déclarations locales.

Exemple 2 : écrire une fonction permettant de calculer la valeur la plus grande parmi 3 entiers.

```
    int plusGrand (int val1, int val2, int val3) {
            int maximum ;
            /* maximum entre 2 premiers entiers
    */
            if ( val1 > val2)
                    maximum = val1;
            else
                    maximum = val2 ;

            /* ajustement si nécessaire */
            if (val3 > maximum)
                        maximum = val3;

            return maximum ;
    }
```

Exemple 3 : écrire une fonction permettant de retourner vrai (1) ou faux (0) selon qu'un tableau t de n entiers ne contient que des valeurs paires ou non.

```
int tousPairs ( int t [], int n ) {
      int i ; /* pour for */
      for ( i = 0 ; i < n ; i++ )
                  if (  t[i] % 2 != 0 ) /* impair
*/
                              return 0 ;

      return 1 ;
}
```

6. instructions pour calculer et retourner le résultat : c'est votre logique pour
   ce calcul. Notez qu'il est possible qu'on rencontre plus d'un "return" dans
   une fonction. L'exécution d'un return provoque le retour du résultat
   retourné et aussi la fin de la fonction (voir l'exemple précédent).

```
               float plusPetit  ( float taille1,
float taille2 ) {
                           if (taille1 < taille2)
                                    return taille1;
                           else   return taille2;
               }
```

Conseils d'écriture une fonction de type void:

On rappelle qu'une fonction de type **void** fait souvent une seule ou quelques petites tâches.

1. Allure d'une fonction de type void :

```
     void    nom  ( liste de paramètre(s) ) {
          déclaration locale si nécessaire

          instruction(s) permettant de réaliser
          les petites tâches confiées à la fonction.
     }
```

2. nom : pour IFT 1810, choisir un verbe ou un nom venant des verbes pour résumer les tâches confiées à la fonction :

```
void trier ( . . . ) /* trier des tableaux */

void afficher ( . . . ) /* afficher le contenu . . .*/

/* lire un fichier, remplir les tableaux, . . . */
void lireRemplir ( . . . )
```