Chapitre 4

Fonctions

Une fonction est une portion de code (séquence d'instructions) effectuant une *tâche précise*, un calcul ou un affichage par exemple. Une fois qu'une fonction a été définie, on peut l'appeler aussi souvent qu'on le souhaite et à n'importe quel endroit du programme. Cela permet de

— *réutiliser* les portions de code effectuant la tâche,

— *structurer* le programme et le rendre plus clair.

Par exemple, si l'on souhaite ajouter les valeurs absolues de deux nombres `a` et `b`, il est plus clair d'écrire

```
res = abs(nb1) + abs(nb2);
```

que

```
if (nb1 < 0)
  nb1 *= -1;
if (nb2 < 0)
  nb2 *= -1;
res = nb1 + nb2;
```

La fonction `abs()` fait partie des nombreuses fonctions prédéfinies dans la bibliothèque standard du C++. Nous allons voir comment en définir de nouvelles.

4.1 Définition

Une fonction se définit par :

— un **type de retour**. On verra de quoi il s'agit au chapitre suivant ; pour l'instant on écrira `void`.

— un **nom** suivi d'une paire de parenthèses. Les règles pour le choix du nom sont les mêmes que pour les noms de variables.

— à l'intérieur des parenthèses, la liste des **paramètres** (ou **arguments**) dans l'ordre, avec leur type et leur nom. Il peut n'y en avoir aucun. S'il y en a plusieurs, ils sont séparés par des virgules.

— un **corps** délimité par des accolades. C'est le bloc d'instructions qui sera exécuté lors de l'appel de la fonction. Les paramètres permettent de jouer sur son comportement.

Les trois premiers éléments constituent l'**en-tête**. La définition est donc de la forme :

```
void nom(type1 param1, type2 param2, ...)
{
  // instructions utilisant les paramètres
}
```

L'en-tête de la fonction est la première ligne. Voici un exemple concret :

```
1 void affiche_age(bool est_femme, string nom, int age)
2 {
3   if (est_femme)
4     affichage("Mme ");
5   else
6     affichage("Mr ");
7   affichage(nom, " a ", age, " ans.\n");
8 }
```

Ici l'en-tête est la ligne 1 et le corps est constitué des lignes 2 à 8.

4.2 Appel

Pour appeler une fonction à partir d'un programme, cette fonction doit **avoir été définie précédemment**. Pour l'utiliser, on commence alors par écrire son nom puis, entre parenthèses, des variables de type correspondant aux paramètres. L'appel se déroule ainsi :

1. le **programme appelant** s'arrête,
2. les variables passées entre parenthèses déterminent la valeur des paramètres. On parle aussi de **valeurs d'entrée** ;
3. le corps de la fonction est exécuté avec ces paramètres,
4. le cas échéant, des **valeurs de sortie** sont renvoyées au programme appelant. On verra comment dans les chapitres suivants ;
5. le programme appelant reprend son exécution.

Le processus est représenté dans la figure 1.1.

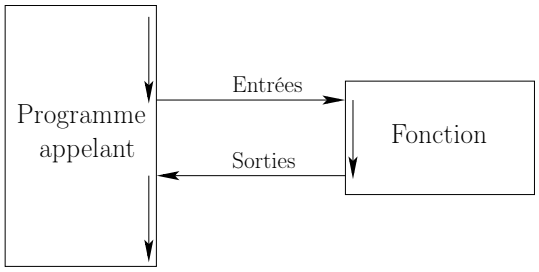


FIGURE 4.1 – Déroulement du programme lors de l'appel d'une fonction

Appelons la fonction `affiche_age()` définie précédemment dans un programme principal :

```
int main()
{
  affiche_age(false,"Martin",23);
  affiche_age(true,"Sophie",32);
  return 0;
}
```

On a appelé la fonction deux fois et produit l'affichage :

```
Mr Martin a 23 ans.
Mme Sophie a 32 ans.
```

Il faut bien distinguer les **paramètres** (ou paramètres formels) des **valeurs des paramètres** (ou paramètres effectifs). Les paramètres n'ont pas de valeur à la définition de la fonction. C'est à l'appel que l'utilisateur fournit les valeurs des paramètres. Celles-ci peuvent donc changer à chaque

appel. Ainsi, dans l'exemple précédent, la fonction `affiche_age()` a trois paramètres formels en entrée :

1. le premier est de type booléen et de nom `est_femme`,
2. le second de type chaîne de caractères et s'appelle `nom`,
3. le troisième de type entier de nom `age`.

Au premier appel (ligne 3), les valeurs des paramètres sont `true`, `"Martin"` et `23` ; au deuxième appel (ligne 4), les valeurs sont `false`, `"Sophie"` et `32`.

Les valeurs des paramètres doivent être du même type que ceux indiqués pour les paramètres dans l'en-tête de la fonction. Dans le cas contraire une **conversion implicite** de type est faite.

Remarque : On peut définir des fonctions ne prenant aucun paramètre.

▶ Sur ce thème : **Exercices 1, 2 et 6, TD4**

4.3 Portée des variables

La **portée** ou visibilité d'une variable est l'endroit dans le code où cette variable est accessible. Une variable déclarée dans le corps d'une fonction n'est visible (ou accessible) qu'à l'intérieur de cette fonction. On dit qu'elle est **locale** à la fonction. Il n'est pas possible d'y accéder depuis le programme principal ou une autre fonction.

Les variables locales sont *créées à chaque appel* de la fonction à leur déclaration dans cette fonction puis *détruites à la fin de l'appel*. Par exemple, la compilation du code

```
void affiche_somme(int x, int y)
{ // Les paramètres en entrée x et y sont aussi des variables locales à la fonction !

  int z = x + y;              // z est une variable locale à la fonction

  affichage ( "La somme de " , x , " et " , y , " vaut " , z , '\n');
}

int main()
{
  affiche_somme(7,5); // Attention ! deux paramètres, 7 et 5
  affichage ( z , '\n');
  return 0;
}
```

produira l'erreur suivante :

```
============================
=== ERREURS DANS LE CODE ===
============================
  In function 'int main()'
Erreur ligne 11 : 'z' was not declared in this scope
```

puisque `z` est une variable locale à `affiche_somme` et n'existe plus après l'appel de cette fonction.

De même il n'est pas possible d'accéder dans une fonction à une variable déclarée dans le programme principal ou une autre fonction. La compilation du code :

```
void f()
{
  affichage ( a , '\n');              // a n'a pas été déclarée
}

int main()
```

```
7  {
8    int a = 12;  // a est une variable locale du main()
9    f();
10   return 0;
11 }
```

produira l'erreur

```
1  ============================
2  === ERREURS DANS LE CODE ===
3  ============================
4    In function 'void f()'
5  Erreur ligne 2 : 'a' was not declared in this scope
```

car, bien que la variable `a` existe durant toute l'exécution du programme, elle est locale au `main` et ne peut donc être utilisée dans une autre fonction.

Il s'ensuit qu'une fonction ne peut communiquer des données avec son programme appelant que par ses paramètres et, on le verra bientôt, ses valeurs de sortie.

4.4 Transmission des paramètres

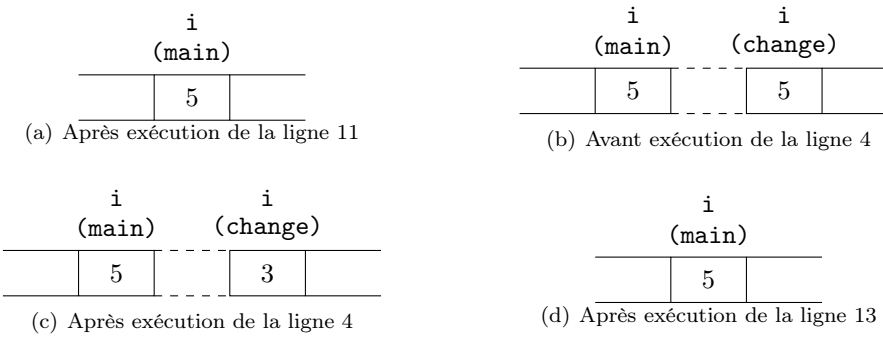
Il y a deux façons de passer des paramètres à une fonction, Une modification des paramètres n'a pas le même effet dans les deux cas sur le reste du programme.

4.4.1 Passage par copie (ou par valeur)

Lors de l'appel, lorsqu'une variable est passée en paramètre, une copie correspondant au paramètre est créée. Le nom de la variable est local à sa fonction, par conséquent une variable `x` de la fonction appelante n'est pas la même (zone mémoire) que la variable `x` de la fonction appelée.
Exemple. Soit le programme suivant :

```
1  void change(int i)
2  {
3    affichage ( "i = " , i , '\n');
4    i = 3;
5    affichage ( "i = " , i , '\n');
6  }
7
8  int main()
9  {
10   int i;
11   i = 5;
12   affichage ( "i = " , i , '\n');
13   change(i);
14   affichage ( "i = " , i , '\n');
15   return 0;
16 }
```

Dans le `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 1.2(a). Seule la variable `i` de la fonction `main` est présente en mémoire et sa valeur est `5`. L'instruction de la ligne 12 affiche `i=5`. La fonction `change` est ensuite appelée avec passage de la variable `i` par copie. Une copie de la variable est donc créée en mémoire et affectée au paramètre de la fonction `change` (qui s'appelle également `i`). L'état de la mémoire correspond à celui indiqué dans la figure 1.2(b). L'instruction de la ligne 3 affiche donc `i=5`. Puis, la valeur de `i` est changée à la ligne 4. Une seule variable `i` est connue de la fonction `change`, et c'est par conséquent celle qui est modifiée (voir figure 1.2(c)). La ligne 5 affiche



(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 4.2 – Passage d'un paramètre par valeur

alors `i=3`. À ce moment-là, l'exécution de `change` est terminée et les variables associées libérées. Après l'exécution de la ligne 13, la mémoire est dans le même état qu'avant l'appel, comme décrit par la figure 1.2(d). La ligne 14 affiche donc `i=5`.

▶ Sur ce thème : **EXERCICES ??, 4 ET 5, TD 4**

4.4.2 Passage par référence

Il fera l'objet d'un autre cours.

TD4 : Fonctions (Corrigé)

✓ Exercice 1 : Test de compréhension*

Question 1.1 : Quelle est la différence entre définition et appel d'une fonction ?
Correction :
- Définition : on donne l'en-tête de la fonction et le corps de la fonction (aucune instruction n'est exécutée).
- Appel : les instructions associées à la fonction sont exécutées pour certaines valeurs données entre parenthèses.

◇

Question 1.2 : Qu'est-ce que l'en-tête d'une fonction ? le corps d'une fonction ?
Correction :
- En-tête : première ligne contenant le type de retour, le nom de la fonction, les paramètres (entre parenthèses séparés par des virgules) ainsi que leur type.
- Corps : les instructions associées à la fonction. ◇

Question 1.3 : Quelle est la différence entre paramètres et valeurs des paramètres ?
Correction :
- Paramètres : noms donnés entre parenthèses de l'en-tête de la fonction. Les paramètres n'ont pas de valeur et sont utilisées dans le corps de la fonction pour expliquer les instructions.
- Valeurs des paramètres : les valeurs passées à la fonction lors d'un appel. ◇

Question 1.4 : Qu'est-ce qu'un programme appelant ?
Correction :
Partie de code où la fonction est appelée (exécutée). Le programme appelant peut être le programme principal ou une autre fonction. (Il peut y avoir plusieurs programmes appelants.) ◇

Question 1.5 : Les parenthèses sont-elles obligatoires dans la définition d'une fonction s'il n'y a pas de paramètres ? Sont-elles nécessaires dans un appel de fonction s'il n'y a pas de valeurs de paramètre ?
Correction :
Oui, les parenthèses sont nécessaires dans les deux cas pour indiquer à l'ordinateur que l'on définit ou que l'on appelle une fonction. ◇

Question 1.6 : Combien de fois une fonction peut-elle être appelée au minimum ? Au maximum ? Donner deux exemples où une fonction est appelée 100 fois, un avec les mêmes valeurs de paramètre pour tous les appels, puis l'autre avec des différentes valeurs de paramètre à chaque appel.
Correction :
Une fonction peut être appelée 0 fois au minimum. Il n'y a pas de limite maximum (Par exemple une boucle infinie appelant à chaque itération une fonction).

```
1  void afficheCarre(int n)
2  {
3    affichage("Le carré de ", n, " est ", n*n, '\n');
4  }
5
6  int main()
7  {
8    //Meme valeur de paramètre
9    int i=1;
10   while(i<=100)
```

```
11   {
12     afficheCarre(10);
13     i++;
14   }
15  //Avec des valeurs différentes de paramètre
16   i=1;
17   while(i<=100)
18   {
19     afficheCarre(i);
20     i++;
21   }
22 }
```

◇

Question 1.7 : Considérons le code :

```
1  void afficheRes(int nb)
2  {
3    affichage ( "Le résultat est " , nb , '\n');
4  }
5
6  void somme2nb(int a, int b)
7  {
8    affichage ( "Premier nombre = " , a , ", deuxième nombre = " , b , '\n');
9    afficheRes(a+b);
10 }
11
12 int main()
13 {
14   somme2nb(1,2);
15   return 0;
16 }
```

Donner, pour la fonction `somme2nb`, les numéros des lignes associées à l'en-tête, au corps de la fonction, à la définition et à l'appel de la fonction. Indiquer également le programme appelant. Donner finalement ses paramètres et ses valeurs de paramètres lors de son appel. Répondre ensuite aux mêmes questions pour la fonction `afficheRes`.

Correction :

- Fonction `somme2nb` :
  - En-tête : ligne 6
  - Corps : lignes 8-9
  - Définition : lignes 6-10
  - Appel : ligne 14
  - Programme appelant : programme principal (fonction `main`)
  - 2 paramètres `a` et `b` de type `int`. Valeurs des paramètres lors de l'appel : 1 et 2.
- Fonction `afficheRes()` :
  - En-tête : ligne 1
  - Corps : ligne 3
  - Définition : lignes 1-4
  - Appel : ligne 9
  - Programme appelant : fonction `somme2nb`
  - 1 paramètre `nb`. Valeur de paramètre lors de l'appel : 3 (1+2).

◇

✓ Exercice 2 : Nombre d'appels*

Question 2.1 : Considérons le code :

```
1 void f1()
2 {
```

```
  affichage ( "hello \n");
  affichage ( "hello \n");
}

int main()
{
  affichage ( "hello \n");
  return 0;
}
```

À l'exécution, combien de fois la fonction `f1` est-elle appelée ? Combien de fois est affiché `hello` ?

Correction :

La fonction `f1` n'est jamais appelée. `hello` est donc affiché une seule fois. ◇

Question 2.2 : Considérons le code :

```
void f2()
{
  affichage ( "hello \n");
  affichage ( "hello \n");
}

int main()
{
  affichage ( "hello \n");
  f2();
  f2();
  return 0;
}
```

Combien de fois la fonction `f2` est-elle appelée ? Combien de fois est affiché `hello` ?

Correction :

La fonction `f2` est appelée 2 fois. `hello` est donc affiché 5 fois. ◇

Question 2.3 : Considérons le code :

```
void f3()
{
  affichage ( "hello \n");
  affichage ( "hello \n");
}

void f4()
{
  f3();
  affichage ( "hello \n");
  f3();
}

int main()
{
  affichage ( "hello \n");
  f3();
  f4();
  return 0;
}
```

Combien de fois sont appelées `f3` et `f4` ? Combien de fois est affiché `hello` ?

Correction :

`f4` est appelée 1 fois et `f3` 3 fois. `hello` est donc affichée 8 fois. ◇

Question 2.4 : Considérons le code :

```
void f5()
{
  affichage ( "hello \n");
  affichage ( "hello \n");
}

void f6(int n)
{
  int i = 0;
  while (i<n)
  {
    f5();
    i++;
  }
  affichage ( "hello \n");
}

int main()
{
  affichage ( "hello \n'');
  f5();
  f6(2);
  f6(4);
  return 0;
}
```

Combien de fois sont appelées `f5()` et `f6()` ? Combien de fois est affiché `hello` ?

Correction :

`f6` est appelée 2 fois et `f5` 7 fois. `hello` est affichée 17 fois. ◇

✓ Exercice 3 : Valeur de paramètres*

Question 3.1 : Qu'affiche le code :

```
void f(int a)
{
  affichage ( "a = " , a , '\n');
}

int main()
{
  int a = 3;
  int b = 5;
  a = 18;
  f(b);
  return 0;
}
```

Correction :

`a = 5` ◇

Question 3.2 : Qu'affiche le code :

```
void f(int a, int b)
{
  affichage ( "a = " , a , ", b = " , b , '\n');
}

int main()
{
  int a = 3;
  int b = 4;
  f(a,b);
  f(b,a);
  f(a,a);
  f(b+2,b-3);
  return 0;
}
```

Correction :

```
a = 3, b = 4
a = 4, b = 3
a = 3, b = 3
a = 6, b = 1
```

✓ Exercice 4 : Portée des variables*

Question 4.1 : Qu'affiche le code :

```

void f()
{
  int a = 18;
  affichage ( "a = " , a , '\n');
}

int main()
{
  int a = 5;
  f();
  affichage ( "a = " , a , '\n');
  return 0;
}
```

Correction :

```
a = 18
a = 5
```

✓ Exercice 5 : Paramètres et variables locales*

Question 5.1 : Qu'affiche le code :

```
void f(int a)
{
  a = 10;
  a +=1;
```

```
  affichage ( "a = " , a , '\n');
}

int main()
{
  f(3);
  f(6);
  return 0;
}
```

Ne peut-on pas l'améliorer ?

Correction :

```
a = 11
a = 11
```

Il y a un problème lors de la conception de la fonction `f()`. En effet, cette fonction prend un paramètre mais sa valeur n'est jamais utilisée. Ce n'est donc pas un paramètre mais une variable locale. Certains compilateurs peuvent même considérer ceci comme une erreur de compilation. Il faut donc simplifier le code en supprimant le paramètre.

```
void f()
{
  int varLocale = 11;
  affichage ( "a = " , varLocale , '\n');
}
```

® Exercice 6 : Comparer deux nombres

Écrire la fonction `compare` qui reçoit deux nombres entiers en paramètre `a` et `b` et affiche un message adapté selon que `a` est supérieur, inférieur ou égal à à `b`.

Correction :

```
void compare(int a, int b)
{
  if (a==b)
    affichage ( a , " est égal à " , b , '\n');
  else if (a < b)
    affichage ( a , " est inférieur à " , b , '\n');
  else
    affichage ( a , " est supérieur à " , b , '\n');
}

int main()
{
  compare(12,18);
  return 0;
}
```

TP4 : Fonctions (Corrigé)

Afin de mettre en pratique vos nouvelles connaissances sur les fonctions, nous allons utiliser un animal, bien sympathique, nommé Farida.

Mais qui est Farida [1] ?

Farida est une tortue graphique qui permet de dessiner des figures géométriques dans un bac à sable de **600 pixels [2] x 600 pixels**.

Mais comment se faire comprendre par Farida ?

Farida obéit à des ordres implémentés par des fonctions particulières. Vous devez respecter **scrupuleusement** le cahier de charges de ces fonctions afin que la tortue puisse comprendre ce que vous cherchez à lui faire faire.

Cependant, **avant de dialoguer avec la tortue**, il faut d'abord l'**initialiser** (au début du programme) et la **libérer** (à la fin de votre programme).

1. Pour commencer, **au début de votre programme**, il est nécessaire d'initialiser le bac à sable où va dessiner Farida. **Pour cela, il faut obligatoirement appeler la fonction :** `void initTortue();`
2. À la fin de votre programme, il faut libérer la tortue ; **pour cela, utiliser la fonction :** `void fermerTortue();`

Exemple simple d'un programme initialisant et libérant la tortue :

```
int main()
{
  /**
   * Initialisation de la tortue.
   * Le bac à sable où dessine la tortue à les caractéristiques suivantes :
   * hauteur = 600 pixels
   * largeur = 600 pixels
   * La tortue se trouve au milieu de son bac à sable et regarde vers le haut.
   */
  initTortue();

  /** La tortue est prête à vous écouter et dessiner dans son bac à sable
   * dans la limite des dimensions de ce dernier
   */

  /**
   * Important : A appeler a la fin de votre programme pour libérer la tortue
   */
  fermerTortue();

  /*
   * Le programme se termine correctement
   */
  return 0;
}
```

1. L'application Farida a été développée par M. KHAFIF ; tout clonage de cette dernière est interdit pour quelque usage que ce soit.
2. Un pixel pouvant être considéré comme un point graphique élémentaire sur l'écran.

Attention : Entre l'initialisation et la libération de la tortue, **seuls les ordres décrits ci-dessous sont autorisés !** Toutes les autres fonctions d'entrées/sorties sont à proscrire, par exemple `affichage()`, `saisie()`, etc.

Ordres compréhensibles par Farida :

prototype de la fonction	description
`void initTortue()`	Permet d'initialiser la tortue ainsi que la fenêtre graphique : la taille de la fenêtre graphique est de 600 pixels en largeur et 600 pixels en hauteur, la tortue est positionnée au milieu de son bac à sable et regarde vers le haut, et le crayon est baissé.
`void fermerTortue()`	La tortue est libérée.
`void leverCrayon()`	Permet de lever le crayon, la tortue ne dessine pas pendant son déplacement.
`void baisserCrayon()`	Permet de baisser le crayon, la tortue dessine pendant son déplacement.
`void changerCouleur(string color)`	Change la couleur de tracé de la tortue. Les couleurs possibles pour `color` sont `"blanc"`, `"rouge"`, `"vert"`, `"bleu"` et `"noir"`.
`void avancer(int dist)`	Permet de faire avancer la tortue de `dist` pixel(s) dans la direction courante.
`void droite(int ang)`	Permet de faire tourner la tortue dans le sens des aiguilles d'une montre de `ang` degrés.
`void gauche(int ang)`	Permet de faire tourner la tortue dans le sens inverse des aiguilles d'une montre de `ang` degrés.
`void positionnerTortue(int x, int y)`	Permet de déplacer la tortue à la position de coordonnées (`x`,`y`).
`void pause()`	Attend la fermeture de la fenêtre pour quitter le programme.

Exemple : Le programme suivant permet de dessiner la lettre T en utilisant 3 couleurs différentes.

```
int main()
{
  /**
    * Initialisation de la tortue
    * Le bac à sable où dessine la tortue à les caractéristiques suivantes :
    * hauteur = 600 pixels
    * largeur = 600 pixels
    *
    * La tortue se trouve au milieu de son bac à sable et regarde vers le haut
    */

  // Initialisation de la tortue
  initTortue();

  // Dessin de du "tronc" de la lettre T en bleu
  baisserCrayon();
  changerCouleur("bleu");
  avancer(80);

  // Dessin de la branche gauche de la lettre T en rouge
  gauche(90);
  changerCouleur("rouge");
  avancer(40);
  droite(180);

  // Dessin de la branche droite de la lettre T en vert
```

```
27    leverCrayon();
28    avancer(40);
29    baisserCrayon();
30    changerCouleur("vert");
31    avancer(40);
32    leverCrayon();
33
34    // Attente de la fermeture de la fenêtre pour quitter le programme
35    pause();
36
37    /**
38     * Libération de la tortue
39     */
40    fermerTortue();
41
42    /**
43     * Le programme s'est terminé correctement
44     */
45    return 0;
46  }
```

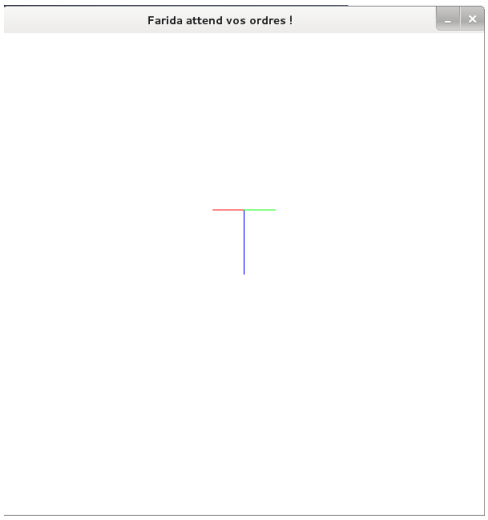


FIGURE 4.3 – Affichage du programme précédent

Exercice 1 : Fonctions et dessins

Question 1.1 : Écrire une fonction, `carre()`, permettant à la tortue de dessiner un carré de côté n à partir de la position courante de la tortue. **Testez et valider votre fonction**.

Exemple d'intéraction homme-machine :

```
Taille du cote du carré ? 60
```

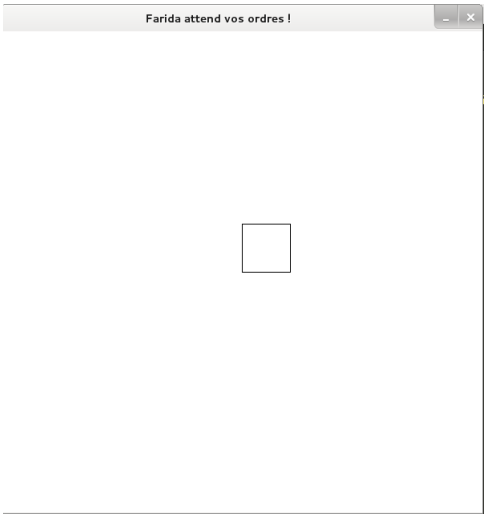


FIGURE 4.4 – Affichage obtenu lors du test de la fonction *carre*.

Correction :

```
/*
 * Dessiner un carre de cote  n
 */
void carre(int n)
{
  // Dessin du carre
  int i = 0;
  while (i < 4)
  {
    avancer(n);
    droite(90);
    i++;
  }
}

int main()
{
  int cote;

  affichage("Taille du cote du carré ?");
  saisie(cote);

  initTortue();

  baisserCrayon();

  carre(cote);

  leverCrayon();

```

```
31    // Attente de la fermeture de la fenêtre pour quitter le programme
32    pause();
33
34    /**
35     * Libération de la tortue
36     */
37    fermerTortue();
38
39    /**
40     * Le programme s'est terminé correctement
41     */
42    return 0;
43  }
```

Question 1.2 : Écrire une fonction, `carrePositionne()`, permettant à la tortue de dessiner un carré de côté n, dont le coin supérieur a pour coordonnées (`x`, `y`) et de couleur `coulCarre`. **Testez et valider votre fonction.**

Note : Vous utiliserez la fonction précédente.

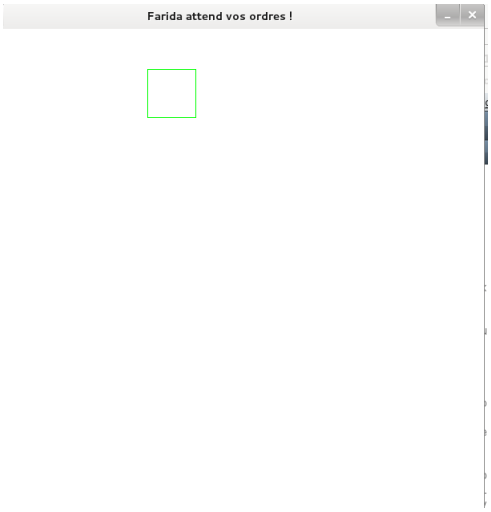


Figure 4.5 – Affichage obtenu lors du test de la fonction *carrePositionne*()

Exemple d'intéraction homme-machine :

```
Taille du cote du carré ? 45
Abscisse du coin supérieur gauche du
carre ? 187
Ordonnee du coin supérieur gauche du
carre ? 70
Couleur R (rouge), V (vert), B
(bleu), N (noir) F
Couleur R (rouge), V (vert), B
(bleu), N (noir) B
```

Correction :
Les fonctions définies dans les questions précédentes ne sont pas réécrites.

```
1   /*
2    * Dessiner un carre de cote n et de couleur coulCarre, dont le coin
3    * supérieur gauche est positionne aux coordonnées (x,y)
4    */
5   void carrePositionne(int n, int x, int y, string coulCarre)
6   {
7     changerCouleur(coulCarre);
8     leverCrayon();
9     // positionnement du coin supérieur du carré
10    positionnerTortue(x, y);
11    baisserCrayon();
12    droite(90);
```

```
13
14    // dessin du carre
15    carre(n);
16    leverCrayon();
17  }
18
19  int main()
20  {
21    int cote;
22
23    affichage("Taille du cote du carré ? ");
24    saisie(cote);
25
26    int xG, yG;
27    affichage("Abscisse du coin supérieur gauche du carre ? ");
28    saisie(xG);
29
30    affichage("Ordonnee du coin supérieur gauche du carre ? ");
31    saisie(yG);
32
33    char choix;
34    do
35    {
36      affichage("Couleur R (rouge), V (vert), B (bleu), N (noir) ");
37      saisie(choix);
38    }
39    while (choix != 'R'
40          && choix != 'r'
41          && choix != 'V'
42          && choix != 'v'
43          && choix != 'B'
44          && choix != 'b'
45          && choix != 'N'
46          && choix != 'n'
47         );
48
49    string couleurChoisie;
50    if ( choix == 'R' || choix == 'r')
51      couleurChoisie = "rouge";
52    else if ( choix == 'V' || choix == 'v' )
53      couleurChoisie = "vert";
54    else if ( choix == 'B' || choix == 'b' )
55      couleurChoisie = "bleu";
56    else
57      couleurChoisie = "noir";
58
59    initTortue();
60
61    carrePositionne(cote, xG, yG, couleurChoisie);
62
63    // Attente de la fermeture de la fenêtre pour quitter le programme
64    pause();
65
66    /**
67     * Libération de la tortue
68     */
69    fermerTortue();
70
```

```
71
72    /**
73     * Le programme s'est terminé corrcetement
74     */
75    return 0;
76  }
```

Question 1.3 : Écrire une fonction, `carres4Coins()`, permettant à la tortue de dessiner 4 carrés de couleurs différentes, de côtés n et situés aux 4 coins de la fenêtre graphique. **Testez et valider votre fonction.**

Note : Vous utiliserez la fonction précédente.

Exemple d'intéraction homme-machine :

`Taille du cote des carrés ? 80`

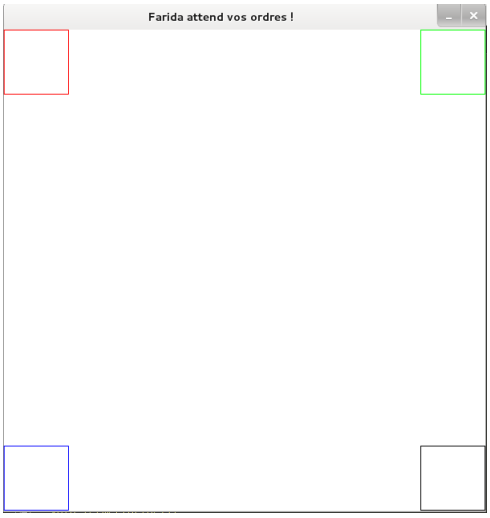


FIGURE 4.6 – Affichage obtenu lors du test de la fonction *carres4coins*()

Correction :

```
1  /*
2   * Dessiner 4 carres de cote  n, de couleurs différentes
3   * aux quatres coins de la fenêtre graphique
4   */
5  void carres4coins(int n)
6  {
7    carrePositionne(n, 0, 0, "rouge");
8    carrePositionne(n, DIM_X - n - 1, 0, "vert");
9    carrePositionne(n, 0, DIM_Y - n - 1, "bleu");
10   carrePositionne(n, DIM_X - n - 1, DIM_Y - n - 1, "noir");
11 }
12
13 int main()
14 {
15   int cote; // longueur d'un cote du carre
16
17   // saisi de la longueur d'un cote du carre
18   affichage("Taille du cote des carrés ? ");
19   saisie(cote);
20
```

```
21    initTortue();
22
23    carres4coins(cote);
24
25    // Attente de la fermeture de la fenêtre pour quitter le programme
26    pause();
27
28    /**
29     * Libération de la tortue
30     */
31    fermerTortue();
32    return 0;
33  }
```

Question 1.4 : Écrire une fonction, `rectangle()`, permettant de dessiner un rectangle à partir de la position courante de la tortue. Pour le test, la largeur et la hauteur du rectangle seront choisies par l'utilisateur avant d'appeler la fonction `rectangle()`.

Exemple d'intéraction homme-machine :

```
largeur du rectangle ? 40
longueur du rectangle ? 70
```

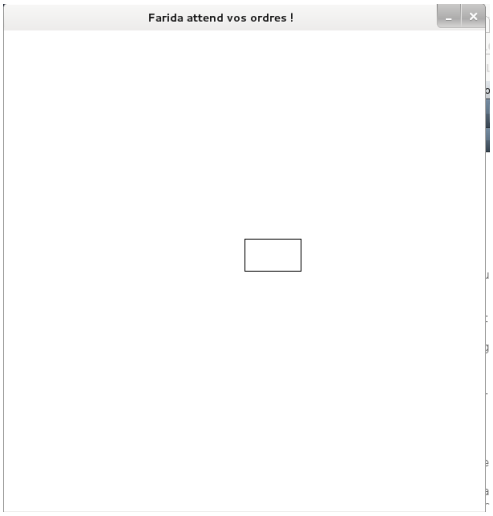


FIGURE 4.7 – Affichage obtenu lors du test de la fonction *rectangle*()

Correction :

```
1  /*!
2   * Dessiner un rectangle
3   */
4  void rectangle(int largeur, int longueur)
5  {
6    // Dessin du rectangle
7    int i = 0;
8    while (i < 2)
9    {
10     avancer(largeur);
11     droite(90);
12     avancer(longueur);
13     droite(90);
14     i++;
```

```
15    }
16  }
17
18
19  int main()
20  {
21    int largR, longR;
22
23    affichage("largeur du rectangle ? ");
24    saisie(largR);
25
26    affichage("longueur du rectangle ? ");
27    saisie(longR);
28
29    // Initialisation de la tortue
30    initTortue();
31
32    baisserCrayon();
33    rectangle(largR, longR);
34    leverCrayon();
35
36    // Attente de la fermeture de la fenêtre pour quitter le programme
37    pause();
38
39    /**
40     * Libération de la tortue
41     */
42    fermerTortue();
43
44    return 0;
45  }
```

Question 1.5 : Écrire une fonction, `croix()`, permettant de dessiner la figure suivante.

Note : La croix est constituée de deux rectangles tel que $longueurRectangle = 3 * largeurRectangle$.

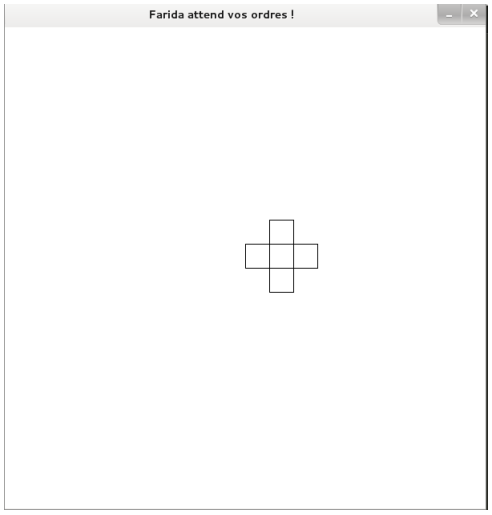


FIGURE 4.8 – Affichage obtenu lors du test de la fonction $croix()$

Correction :

```
/*
 * Dessiner une croix (largeur n)
 */
void croix(int n)
{
  baisserCrayon();
  rectangle(n, 3 * n);
  leverCrayon();
  avancer(2 * n);
  droite(90);
  avancer(n);

  baisserCrayon();
  rectangle(n, 3 * n);
}

int main()
{
  int n;

  affichage("Donnez la largeur du rectangle ? ");
  saisie(n);

  // Initialisation de la tortue
  initTortue();

  baisserCrayon();
  croix(n);
  leverCrayon();
  // Attente de la fermeture de la fenêtre pour quitter le programme
  pause();

  fermerTortue();
  return 0;
}
```

Question 1.6 : Carrés emboîtés.

Écrire une fonction, `carresEmboites()` qui réalise, grâce à la tortue, le dessin suivant : sachant que

- la tortue, au départ, est placée dans le coin supérieur gauche du carré interne et qu'elle regarde vers le haut,
- le nombre de carrés à tracer est un paramètre en entrée pour la fonction (4 dans l'exemple ci-dessus),
- la longueur du coté du carré le plus petit (à l'intérieur) est un paramètre en entrée de la fonction et la longueur du coté de chaque autre carré est égale au double de la longueur du coté du carré juste en dessous. On utilisera pour dessiner les carrés la fonction `carre()`,
- le segment qui relie chaque carré est de longueur fixe et égale au coté du plus petit carré et forme un morceau de la diagonale de chaque carré.

Correction :

```
/*
 * Dessiner des carrés emboités.
 * cote : longueur du cote du carre
 * nbCarres : nombre de carres emboités
 */
void carreEmboite(int cote, int nbCarres)
```

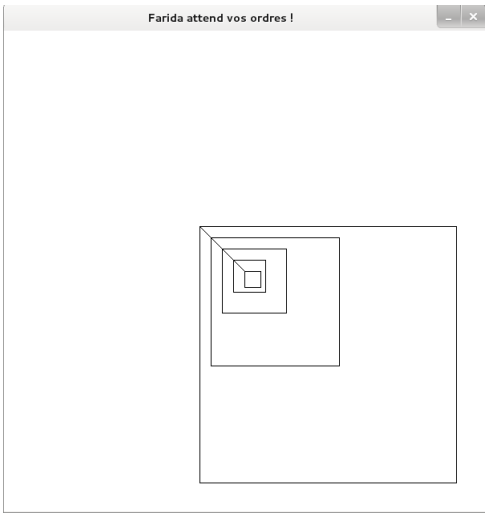


FIGURE 4.9 – Affichage obtenu lors du test de la fonction *carresEmboites*()

```
7  {
8    int longueur = cote;
9
10   // positionnement initial
11   droite(90);
12   baisserCrayon();
13
14   int i = 0;
15   while (i < nbCarres - 1)
16   {
17     carre(longueur);
18     gauche(135);
19     avancer(cote);
20     droite(135);
21     longueur = longueur * 2;
22     i++;
23   }
24   carre(longueur);
25   leverCrayon();
26 }
27
28 int main()
29 {
30   int coteC, nbCarres;
31
32   affichage("Longueur d'un côté du carre? ");
33   saisie(coteC);
34
35   affichage("nombre de carres emboites ? ");
36   saisie(nbCarres);
37
38   initTortue();
39
40   carreEmboite(coteC, nbCarres);
41
42   // Attente de la fermeture de la fenêtre pour quitter le programme
43   pause();
```

```
44
45   fermerTortue();
46   return 0;
47 }
```

Question 1.7 : Carré de carrés.

1. Écrire une fonction, `carreCarres()` permettant de dessiner p carrés de coté n les uns à la suite des autres. On utilisera pour dessiner chaque petit carré la fonction `carre()`.
2. Écrire un programme qui permet de dessiner avec la tortue la figure 2 :

   Affichage obtenu :

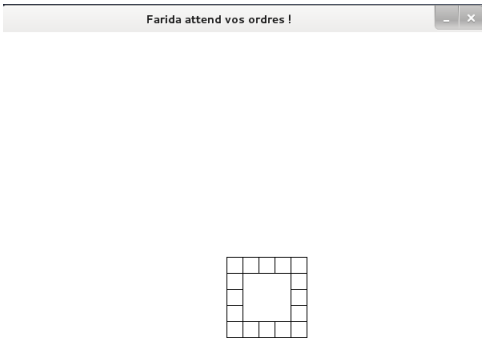


FIGURE 4.10 – Affichage obtenu lors du test de la fonction *carreCarres*()

— La longueur du coté des petits carrés est un paramètre de la fonction.
— Le nombre de petits carrés nécessaires pour former un coté du grand carré est un paramètre de la fonction (4 dans l'exemple ci-dessus).
— La tortue, au départ, est positionnée au coin inférieur gauche et regarde vers le haut.

Correction :

```
1  /*
2   * Dessiner un carre de carres
3   * cote : longueur d'un côté du petit carre
4   * nbCarres : nombre de petits carrés pour
5   *         former un côté du grand carré
6   */
7  void carreCarres(int cote, int nbCarres)
8  {
9    baisserCrayon();
10   int i = 0;
11   while (i < 4)
12   {
13     int j = 0;
14     while (j < nbCarres)
15     {
16       baisserCrayon();
```

```
17        carre(cote);
18        leverCrayon();
19        droite(90);
20        avancer(cote);
21        gauche(90);
22        j++;
23      }
24      gauche(90);
25      avancer(cote);
26      droite(180);
27      i++;
28    }
29  }
30
31
32  int main()
33  {
34    int coteC, nbCarres;
35
36    affichage("Longueur d'un côté du petit carré ? ");
37    saisie(coteC);
38
39    affichage("nombre de carres par cote du grand carre ? ");
40    saisie(nbCarres);
41    initTortue();
42
43    baisserCrayon();
44    carreCarres(coteC, nbCarres);
45    leverCrayon();
46
47    // Attente de la fermeture de la fenêtre pour quitter le programme
48    pause();
49
50    fermerTortue();
51    return 0;
52  }
```