

# Algorithmique et programmation en C

## les fonctions et la portée des variables

François Delbot

Maître de conférences de l'Université Paris-Nanterre  
Membre de l'équipe de recherche opérationnelle du LIP6

26 mars 2020

# Pré-requis

Avant de débiter ce cours, vous devez maîtriser les bases de la programmation, en particulier :

- ❶ Savoir écrire un programme minimum en C.
- ❷ Savoir manipuler des fonctions (printf, scanf, pow ...).
- ❸ Être à l'aise avec les tests (if/else).
- ❹ Être à l'aise avec les boucles (while/for).
- ❺ Savoir manipuler des tableaux.

## 1 Des programmes plus complexes

- Prototype
- Conseils

## 2 Appel de fonction

- Paramètres formels et effectifs
- Portée des variables
- Transmission des paramètres
- Les tableaux, un cas à part... pour le moment.

## 3 Exercices

# Des programmes simples...

Jusqu'à présent, nous avons écrit des programmes permettant de résoudre des problèmes triviaux. Par exemple, déterminer si un nombre est premier, ou non :

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 int main()
5 {
6     int nb,i,premier=1;
7     printf("Entrez votre nombre: ");
8     scanf("%d",&nb);
9     for(i=2;i<nb;i++)
10     {
11         if(nb%i==0) premier = 0;
12     }
13     if(premier==1&&nb>=2)
14     {
15         printf("%d est un nombre premier!",nb);
16     }
17     else
18     {
19         printf("%d n'est pas un nombre premier!",nb);
20     }
21     return EXIT_SUCCESS;
22 }
```

# Des problèmes plus complexes...

## Illustration avec les nombres narcissiques

De nombreux problèmes vont nécessiter un temps de réflexion plus important ainsi qu'une quantité de code conséquente. Voici un exemple d'exercice non trivial :

Un nombre narcissique est un entier naturel  $n$  non nul qui est égal à la somme des puissances  $p$ - $i_{\text{emes}}$  de ses chiffres en base dix, où  $p$  désigne le nombre de chiffres de  $n$ . Par exemple,  $548834 = 5^6 + 4^6 + 8^6 + 8^6 + 3^6 + 4^6$  est un nombre narcissique. Écrire un programme qui demande à l'utilisateur un entier strictement positif et qui affiche si ce nombre est narcissique ou non.

Comment feriez-vous pour réaliser cet exercice ?

# Des problèmes plus complexes...

Illustration avec les nombres narcissiques

A vous de jouer, essayez de réaliser un programme résolvant ce problème.

# Des problèmes plus complexes...

Nous nous intéressons donc à des problèmes plus difficiles à appréhender, que l'on peut qualifier de complexe :

## Définition : complexe

Qui contient plusieurs parties ou plusieurs éléments combinés d'une manière qui n'est pas immédiatement claire pour l'esprit. [larousse]

Mais comment résoudre ces problèmes, si ils sont difficiles à appréhender ? Et comment réaliser le programme correspondant ?

# Diviser pour régner

- Pour résoudre un problème complexe, il est d'usage de le décomposer en sous-problèmes plus simples à résoudre.
- Ces sous-problèmes sont eux-mêmes décomposés en problèmes encore plus simples à résoudre.
- Il s'agit du principe, bien connu, de diviser pour régner !

En reprenant notre exercice sur les nombres narcissiques, il est clair que nous devons être capable de :

- 1 connaître le nombre de chiffres du nombre testé
- 2 déterminer la valeur de chacun des chiffres du nombre testé
- 3 élever un nombre à une puissance



# Diviser pour régner

Indépendamment les unes des autres, ces tâches ne sont pas compliquées :

Exemple de code permettant de calculer  $x^n$  :

```
1  int i, res=1, x=2, n=4;
2  for (i=1; i<=n; i++)
3  {
4      res = res * x;
5  }
```

Exemple de code permettant de compter le nombre de chiffres d'une variable *nb* :

```
1  int nb=12345, tmp, cpt=0;
2  tmp=nb;
3  while(tmp >=10)
4  {
5      cpt++;
6      tmp = tmp / 10;
7  }
8  cpt++;
9  printf("\n%d - %d", nb, cpt);
```

# Diviser pour régner

## Algorithme

A partir de là, il devient facile d'imaginer un algorithme permettant de résoudre notre problème :

Soit  $nb$  le nombre testé :

- ① obtenir  $cpt$  le nombre de chiffres de  $nb$
- ② initialiser une variable  $somme$  à 0
- ③ pour chaque chiffre  $ch$  composant  $nb$  :
  - ① calculer  $ch^{cpt}$
  - ② ajouter le résultat de ce calcul à  $somme$
- ④ si  $somme$  vaut  $nb$  le nombre est narcissique, sinon il ne l'est pas.

# Diviser pour régner

- Malheureusement, réaliser un programme qui implémente cet algorithme n'est pas trivial.
- L'interactions des différentes taches va rendre le code difficile à produire et à lire.
- Des erreurs peuvent se glisser facilement dans le code.

# Diviser pour régner

Exemple d'implémentation affichant si un nombre *nb* est narcissique :

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  int main()
5  {
6      int i,j,nb=370, tmp, cpt=0, ch, somme=0,res;
7      tmp=nb;
8      while(tmp >=10)
9      {
10         cpt++;
11         tmp = tmp / 10;
12     }
13     cpt++;
14     tmp=nb;
15     for(i=0; i<cpt; i++)
16     {
17         ch = tmp%10;
18         tmp = tmp/10;
19         res = 1;
20         for(j=1; j<=cpt; j++)
21         {
22             res = res * ch;
23         }
24         somme = somme + res;
25     }
26     if(somme == nb) printf("\n%d est un nombre narcissique",nb);
27     else printf("\n%d n'est pas un nombre narcissique",nb);
28     return EXIT_SUCCESS;
29 }
```

# Découper son code est une bonne chose

Pour simplifier notre code, nous allons le découper en fonctions. Mais qu'est-ce qu'une fonction ?

## Fonction

- Un ensemble d'instructions, un morceau de code.
- Cet ensemble d'instructions permet d'effectuer une tâche précise (par exemple compter le nombre de chiffres d'un nombre, élever  $x$  à la puissance  $n$  ...)
- Cet ensemble d'instructions est identifié par un nom, et peut être appelé à volonté.

# Les avantages des fonctions

- Meilleure organisation et lisibilité du code : le programme est décomposé en sous-programmes.
- Plus facile à débbugger : chaque fonction peut être testée séparément du reste du programme.
- Réutilisabilité du code : il est possible de réutiliser des fonctions déjà écrites pour résoudre d'autres problèmes (c'est le cas des bibliothèques de fonctions prédéfinies : `printf`, `scanf`, ...).

# Un premier exemple 1/2

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  int main()
5  {
6      int i, fact5, fact9;
7
8      fact5=1;
9      fact9=1;
10
11     for (i=1;i<=5;i++)
12     {
13         fact5 = fact5*i;
14     }
15     printf("La factorielle de 5 vaut : %d\n", fact5);
16     for (i=1;i<=9;i++)
17     {
18         fact9 = fact9*i;
19     }
20     printf("La factorielle de 9 vaut : %d\n", fact9);
21
22     return EXIT_SUCCESS;
23 }
```

## Un premier exemple 2/2

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  int factorielle(int n)
5  {
6      int res=1, i;
7      for(i=1; i<=n; i++)
8      {
9          res = res*i;
10     }
11     return res;
12 }
13 int main()
14 {
15     int i, fact5, fact9;
16
17     fact5=factorielle(5);
18     fact9=factorielle(9);
19     printf("La factorielle de 5 vaut : %d\n", fact5);
20     printf("La factorielle de 9 vaut : %d\n", fact9);
21
22     return EXIT_SUCCESS;
23 }
```



# Principe et définition

## Principe d'une fonction



## Définition d'une fonction

```
1 type_de_retour nom_de_la_fonction(type1 parm1, type2 parm2, ...)  
2 {  
3     // Debut des instructions  
4     instruction1;  
5     instruction2;  
6     ...  
7     ...  
8     instructionN;  
9     return resultat; // Valeur renvoyee par la fonction  
10 }
```

# Détails de la définition

## Type de retour d'une fonction

- `type_de_retour` : Le type de de la valeur renvoyée par la fonction (void, int, float, char, long, ...)
- Une fonction dont le type de retour est `void` est une fonction qui ne retourne rien (également appelées 'procédures' dans d'autres langages, le PASCAL par exemple).
- La valeur retournée peut-être utilisée pour la suite du programme (lignes 17 et 18 des slides de l'exemple avec la fonction factorielle).

# Détails de la définition

## Nom d'une fonction

- ❶ Le nom d'une fonction doit respecter les mêmes règles que pour les noms de variables.
- ❷ Il est d'usage de nommer les fonctions de manière à faciliter son utilisation.
- ❸ Deux fonctions ne peuvent avoir le même nom.

# Détails de la définition

## Paramètres formels d'une fonction

### Paramètres formels Type1 prm1, Type2 prm2,...

- Liste de déclarations des variables (type et nom) associés aux arguments transmis à la fonction lors de son appel.
- Cette liste peut être vide.
- Le type de chaque variable doit être spécifié.
- Les déclarations des variables sont séparées par des **virgules**.
- Les paramètres formels sont des variables qui sont déclarées **automatiquement** lors de l'appel de la fonction et initialisés automatiquement avec les paramètres effectifs.

# Détails de la définition

## Corps de la fonction

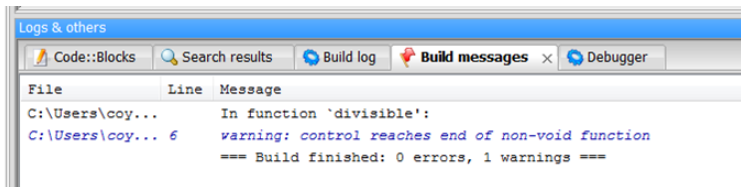
- ❶ Le corps de la fonction définit la suite des instructions nécessaires à la réalisation du sous-programme.
- ❷ Le corps de la fonction est un bloc d'instructions.
- ❸ Les déclarations de variables sont réalisées en tête du bloc de la fonction.
- ❹ Pour toute exécution de fonction dont le type de retour n'est pas « void », chaque branche d'exécution doit se terminer par une instruction de retour de valeur : « return expression ; » où « expression » correspond au résultat retourné par la fonction.
- ❺ Le type de « expression » est le même que le type de retour de la fonction.

# Exercice 1 : ce programme est-il correct ?

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  int divisible(int a, int b)
5  {
6      if (a%b==0) return 1;
7  }
8
9  int main()
10 {
11     int nb1=30, nb2=5;
12     if (divisible(nb1,nb2)==1)
13     {
14         printf("%d est divisible par %d.", nb1, nb2);
15     }
16     else
17     {
18         printf("%d n'est pas divisible par %d.", nb1, nb2);
19     }
20     return EXIT_SUCCESS;
21 }
```

# Exercice 1 : ce programme est-il correct ?

Erreur. Un chemin d'exécution ne retourne rien.



The screenshot shows the 'Logs & others' window of an IDE. It has tabs for 'Code::Blocks', 'Search results', 'Build log', 'Build messages', and 'Debugger'. The 'Build messages' tab is active, displaying the following text:

File	Line	Message
C:\Users\coy...		In function 'divisible':
C:\Users\coy... 6	6	warning: control reaches end of non-void function
=== Build finished: 0 errors, 1 warnings ===		

# Exercice 1 : ce programme est-il correct ?

## Correction

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  int divisible(int a, int b)
5  {
6      if (a%b==0) return 1;
7      return 0;
8  }
9
10 int main()
11 {
12     int nb1=30, nb2=5;
13     if (divisible(nb1,nb2)==1)
14     {
15         printf("%d est divisible par %d.",nb1,nb2);
16     }
17     else
18     {
19         printf("%d n'est pas divisible par %d.",nb1,nb2);
20     }
21     return EXIT_SUCCESS;
22 }
```



## Exercice 2 : Modifiez ce programme pour le simplifier

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  int main()
5  {
6      int i;
7      for (i=1; i<11; i++)
8          printf ("\n%d_x_3_=%d", i, i*3);
9      for (i=1; i<11; i++)
10         printf ("\n%d_x_4_=%d", i, i*4);
11     for (i=1; i<11; i++)
12         printf ("\n%d_x_5_=%d", i, i*5);
13     for (i=1; i<11; i++)
14         printf ("\n%d_x_6_=%d", i, i*6);
15     for (i=1; i<11; i++)
16         printf ("\n%d_x_7_=%d", i, i*7);
17     for (i=1; i<11; i++)
18         printf ("\n%d_x_8_=%d", i, i*8);
19     return EXIT_SUCCESS;
20 }
```

## Exercice 2 : Modifiez ce programme

Plus simple, non ? Apportez encore une petite modification...

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  void table(int n)
4  {
5      int i;
6      for (i=1; i<11; i++)
7      {
8          printf ("\n%d x %d = %d", i, n, i*n);
9      }
10 }
11 int main()
12 {
13     table(3);
14     table(4);
15     table(5);
16     table(6);
17     table(7);
18     table(8);
19     return EXIT_SUCCESS;
20 }
```

## Exercice 2 : Modifiez ce programme

Plus simple, non ?

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  void table(int n)
4  {
5      int i;
6      for (i=1; i<11; i++)
7      {
8          printf("\n%d x %d = %d", i, n, i*n);
9      }
10 }
11 int main()
12 {
13     int i;
14     for (i=3; i<9; i++)
15     {
16         table(i);
17     }
18     return EXIT_SUCCESS;
19 }
```

# Le prototype d'une fonction

## Définition d'une fonction

Pour appeler une fonction, il est nécessaire qu'elle soit déclarée dans le code qui précède l'appel.

Par exemple, dans le code du slide précédent, la fonction `table` est définie (et donc déclarée) des lignes 3 à 10. Au moment où cette fonction est appelée, ligne 16, le compilateur connaît la définition de la fonction, ce qui permet de l'utiliser.

# Le prototype d'une fonction

Exemple de fonction non déclarée au moment de son appel

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  int main()
5  {
6      int i;
7      for (i=3; i<9; i++)
8      {
9          table(i);
10     }
11     return EXIT_SUCCESS;
12 }
13
14 void table(int n)
15 {
16     int i;
17     for (i=1; i<11; i++)
18     {
19         printf("\n%d x %d = %d", i, n, i*n);
20     }
21 }
```

# Le prototype d'une fonction

## Définition et déclaration

Il n'est pas nécessaire que la fonction soit définie au moment de son appel. Uniquement qu'elle soit déclarée. La déclaration décrit comment utiliser cette fonction (type de retour, nom, paramètres formels). On pourra utiliser les termes suivants de manière équivalente :

- Signature de fonction.
- En-tête de fonction.
- Déclaration de fonction.
- Prototype de fonction.

Pour déclarer une fonction, il suffit de placer la première ligne et d'ajouter un point virgule.

# Le prototype d'une fonction

## Exemple de déclaration de fonction

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  void table(int n);
5
6  int main()
7  {
8      int i;
9      for (i=3; i<9; i++)
10     {
11         table(i);
12     }
13     return EXIT_SUCCESS;
14 }
15 void table(int n)
16 {
17     int i;
18     for (i=1; i<11; i++)
19     {
20         printf("\n%d x %d = %d", i, n, i*n);
21     }
22 }
```

# Le prototype d'une fonction

## Exemples

Déterminez, uniquement à partir du prototype de différentes fonctions, ce que font ces fonctions :

```
1 int somme_i(int a, int b);  
2 float somme_f(float a, float b);  
3 int est_un_nombre_premier(int n);  
4 void affiche_table_ASCII(void);  
5 int tirage_aleatoire(int min, int max);
```



# Le prototype d'une fonction

## Exercices

- 1 Donnez le prototype d'une fonction permettant de calculer le cube d'un entier.
- 2 Donnez le prototype d'une fonction qui affiche le message 'coucou' à l'écran.
- 3 Donnez le prototype d'une fonction calculant une approximation de Pi (3,14...)
- 4 Donnez le prototype d'une fonction calculant  $x$  à la puissance  $n$ , avec  $x$  et  $n$  des nombres à virgule.

# Le prototype d'une fonction

## Exemples

Déterminez, uniquement à partir du prototype de différentes fonctions, ce que font ces fonctions :

```
1 int cube(int n);  
2 void affiche_message(void);  
3 float PI_approx(void);  
4 float PI_approx(int n);  
5 float puissance(float x, float n);
```

# Les fonctions

## Conseils d'écriture

- Écrire le plus possible de fonctions, les plus petites possibles (réutilisation future).
- Toujours regarder s'il n'existe pas une fonction dans une bibliothèque qui résout le problème.
- Toujours commencer par se demander quels sont les paramètres formels et le type de retour de la fonction.
- Différencier les fonctions qui réalisent des affichages et les fonctions qui retournent des résultats.
- Mettre un commentaire au-dessus de la déclaration pour expliquer ce que fait la fonction et comment elle s'utilise !

# Les fonctions

## Appel de fonction

- Très important : La définition d'une fonction **n'exécute pas** la suite d'instructions qui la compose.
- Pour exécuter les instructions, il faut appeler la fonction.

# Les fonctions

## Paramètres formels vs. paramètres effectifs

- Les paramètres formels sont utilisés lors de la définition de la fonction.
- Lors de l'appel, des valeurs doivent être données à ces paramètres effectifs.
- Un paramètre effectif est une expression.
- Cette expression peut être une constante, une variable, le résultat d'un calcul ou le retour d'un appel de fonction.

# Les fonctions

## Appel de fonction : exemple

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  int cube(int x)
5  {
6      return x*x*x;
7  }
8
9  void afficheCube(int x)
10 {
11     int c = cube(x);
12     printf("Le cube de %d vaut %d", x, c);
13 }
14
15 int main()
16 {
17     int res = cube(3);
18     printf("%d", res);
19     afficheCube(4);
20     return EXIT_SUCCESS;
21 }
```

# Les fonctions

## Liste des paramètres formels

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  int cube(int x)
5  {
6      return x*x*x;
7  }
8
9  void afficheCube(int x)
10 {
11     int c = cube(x);
12     printf("Le cube de %d vaut %d", x, c);
13 }
14
15 int main()
16 {
17     int res = cube(3);
18     printf("%d", res);
19     afficheCube(4);
20     return EXIT_SUCCESS;
21 }
```

# Les fonctions

## Liste des paramètres formels et des paramètres effectifs

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  int cube(int x)
5  {
6      return x*x*x;
7  }
8
9  void afficheCube(int x)
10 {
11     int c = cube(x);
12     printf("Le cube de %d vaut %d", x, c);
13 }
14
15 int main()
16 {
17     int res = cube(3);
18     printf("%d", res);
19     afficheCube(4);
20     return EXIT_SUCCESS;
21 }
```



# Les fonctions

Liste des paramètres formels et des paramètres effectifs.

Exemple d'appel de la fonction cube.

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  int cube(int x)
5  {
6      return x*x*x;
7  }
8
9  void afficheCube(int x)
10 {
11     int c = cube(x);
12     printf("Le cube de %d vaut %d", x, c);
13 }
14
15 int main()
16 {
17     int res = cube(3);
18     printf("%d", res);
19     afficheCube(4);
20     return EXIT_SUCCESS;
21 }
```

# Les fonctions

## Résumé

- La déclaration d'une fonction doit toujours précéder son appel.
- L'appel d'une fonction peut se faire uniquement à l'intérieur d'une autre fonction.
- La seule fonction qu'il n'y a pas besoin d'appeler explicitement est la fonction `main`.
- Le nombre, l'ordre et le type des paramètres formels et des paramètres effectifs d'une fonction doivent être identiques.

# Structure globale d'un fichier source

```
1  /* Incusion de librairies */
2  #include <stdio.h>
3  #include <stdlib.h>
4
5  /* Macros du pre processeur*/
6  #define ...
7
8  /* Declaration des variables globales */
9
10 /* Declaration des fonctions*/
11
12 /* Definition des fonctions */
13
14 /* Definition de la fonction principale */
15 int main(void)
16 {
17     /* Declaration des variables locales */
18
19     /* Suite des instruction de la fonction */
20
21     /* Retour de la fonction */
22     return EXIT_SUCCESS;
23 }
```

# Structure globale

Exemple. Quelle est la valeur affichée à l'écran ?

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  int a = 2;
5
6  int somme(int y)
7  {
8      int x = 3;
9      return a + x + y;
10 }
11
12 int main(void)
13 {
14     int res = somme(7);
15     printf("%d",res);
16     return EXIT_SUCCESS;
17 }
```

# Structure globale

Exemple. Portée des variables.

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  int a = 2;
5
6  int somme(int y)
7  {
8      int x = 3;
9      return a + x + y;
10 }
11
12 int main(void)
13 {
14     int res = somme(7);
15     printf("%d",res);
16     return EXIT_SUCCESS;
17 }
```

↓ Portée de la variable locale x

# Structure globale

Exemple. Portée des variables.

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  int a = 2;
5
6  int somme(int y)
7  {
8      int x = 3;
9      return a + x + y;
10 }
11
12 int main(void)
13 {
14     int res = somme(7);
15     printf("%d",res);
16     return EXIT_SUCCESS;
17 }
```

↓ Portée de la variable locale x

↓ Portée de la variable locale y

# Structure globale

Exemple. Portée des variables.

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  int a = 2;
5
6  int somme(int y)
7  {
8      int x = 3;
9      return a + x + y;
10 }
11
12 int main(void)
13 {
14     int res = somme(7);
15     printf("%d",res);
16     return EXIT_SUCCESS;
17 }
```

Diagram illustrating variable scope:

- Blue arrow pointing to line 8: Portée de la variable locale x
- Orange arrow pointing to line 9: Portée de la variable locale y
- Red arrow pointing to line 14: Portée de la variable locale res

# Structure globale

Exemple. Portée des variables.

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  int a = 2;
5
6  int somme(int y)
7  {
8      int x = 3;
9      return a + x + y;
10 }
11
12 int main(void)
13 {
14     int res = somme(7);
15     printf("%d",res);
16     return EXIT_SUCCESS;
17 }
```

The diagram illustrates the scope of variables in the provided C code. It features three vertical arrows pointing downwards, each associated with a text label:

- A blue arrow points from the opening curly brace of the `somme` function (line 7) to the label "Portée de la variable locale x" (Scope of the local variable x).
- An orange arrow points from the opening curly brace of the `somme` function (line 7) to the label "Portée de la variable locale y" (Scope of the local variable y).
- A red arrow points from the opening curly brace of the `main` function (line 13) to the label "Portée de la variable locale res" (Scope of the local variable res).
- A black arrow points from the `int a = 2;` declaration (line 4) to the label "Portée de la variable globale a" (Scope of the global variable a).



# Portée des noms de variables

## Variables locales et paramètres formels

Dans une fonction, les paramètres formels et les variables locales déclarées explicitement doivent porter des noms différents.

```
1 int somme(int y)
2 {
3     int y = 3;
4     return y + y;
5 }
```

**Erreur de compilation : 'y' redeclared as different kind of symbol**

# Portée des noms de variables

## Variables locales, paramètres formels et variables globales

Dans une fonction, les paramètres formels et les variables locales déclarées explicitement peuvent avoir le même nom qu'une variable globale. Dans ce cas, c'est la variable locale qui prends le dessus.

```
1 int a = 5;  
2  
3 int somme(int y)  
4 {  
5     int a = 3;  
6     return a + y;  
7 }
```

- ❶ La fonction somme va retourner la valeur  $y + 3$
- ❷ La variable globale a possède toujours la valeur 5 après l'exécution de la fonction somme.

# Les fonctions

## Résumé 2/2

- Une variable déclarée à l'intérieur d'une fonction est une variable locale.
- Une variable déclarée en début de programme (après les « include » et les « # define ») est une variable globale.
- Portée d'une variable : désigne la partie du programme dans laquelle on peut utiliser une variable.
- Une variable locale n'est utilisable qu'à l'intérieur de la fonction dans laquelle elle est déclarée. Plus précisément, elle n'est utilisable que dans le bloc dans lequel elle est déclarée.
- Une variable globale est utilisable dans tout le programme.
- Les paramètres formels d'une fonction correspondent à des variables locales initialisées lors de l'appel de la fonction.

# Transmission des paramètres

## Exemple introductif (1/11)

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 void change(int y)
5 {
6     y = 3;
7 }
8
9 int main(void)
10 {
11     int y = 7;
12     printf("%d\n", y);
13     change(y);
14     printf("%d\n", y);
15     return EXIT_SUCCESS;
16 }
```

Pile des  
appels de  
fonctions



# Transmission des paramètres

## Exemple introductif (2/11)

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 void change(int y)
5 {
6     y = 3;
7 }
8
9 int main(void) ←
10 {
11     int y = 7;
12     printf("%d\n", y);
13     change(y);
14     printf("%d\n", y);
15     return EXIT_SUCCESS;
16 }
```



Pile des  
appels de  
fonctions

# Transmission des paramètres

## Exemple introductif (3/11)

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 void change(int y)
5 {
6     y = 3;
7 }
8
9 int main(void)
10 {
11     int y = 7;
12     printf("%d\n", y);
13     change(y);
14     printf("%d\n", y);
15     return EXIT_SUCCESS;
16 }
```



Pile des  
appels de  
fonctions

# Transmission des paramètres

## Exemple introductif (4/11)

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 void change(int y)
5 {
6     y = 3;
7 }
8
9 int main(void)
10 {
11     int y = 7;
12     printf("%d\n", y);
13     change(y);
14     printf("%d\n", y);
15     return EXIT_SUCCESS;
16 }
```



Pile des  
appels de  
fonctions

# Transmission des paramètres

## Exemple introductif (5/11)

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 void change(int y)
5 {
6     y = 3;
7 }
8
9 int main(void)
10 {
11     int y = 7;
12     printf("%d\n", y);
13     change(y);
14     printf("%d\n", y);
15     return EXIT_SUCCESS;
16 }
```



Pile des  
appels de  
fonctions



# Transmission des paramètres

## Exemple introductif (6/11)

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  void change(int y) ←
5  {
6      y = 3;
7  }
8
9  int main(void)
10 {
11     int y = 7;
12     printf("%d\n", y);
13     change(y); ←
14     printf("%d\n", y);
15     return EXIT_SUCCESS;
16 }
```

Pile des  
appels de  
fonctions



# Transmission des paramètres

## Exemple introductif (7/11)

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  void change(int y)
5  {
6      y = 3;
7  }
8
9  int main(void)
10 {
11     int y = 7;
12     printf("%d\n", y);
13     change(y);
14     printf("%d\n", y);
15     return EXIT_SUCCESS;
16 }
```

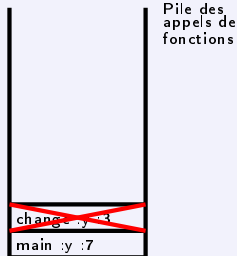


Pile des  
appels de  
fonctions

# Transmission des paramètres

## Exemple introductif (8/11)

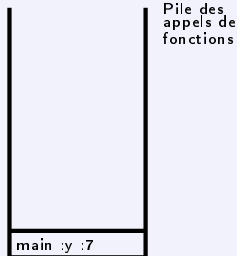
```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 void change(int y)
5 {
6     y = 3;
7 }
8
9 int main(void)
10 {
11     int y = 7;
12     printf("%d\n", y);
13     change(y);
14     printf("%d\n", y);
15     return EXIT_SUCCESS;
16 }
```



# Transmission des paramètres

## Exemple introductif (9/11)

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  void change(int y)
5  {
6      y = 3;
7  }
8
9  int main(void)
10 {
11     int y = 7;
12     printf("%d\n", y);
13     change(y);
14     printf("%d\n", y); ←
15     return EXIT_SUCCESS;
16 }
```



# Transmission des paramètres

## Exemple introductif (10/11)

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 void change(int y)
5 {
6     y = 3;
7 }
8
9 int main(void)
10 {
11     int y = 7;
12     printf("%d\n", y);
13     change(y);
14     printf("%d\n", y);
15     return EXIT_SUCCESS; ←
16 }
```



Pile des  
appels de  
fonctions

# Transmission des paramètres

## Exemple introductif (11/11)

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 void change(int y)
5 {
6     y = 3;
7 }
8
9 int main(void)
10 {
11     int y = 7;
12     printf("%d\n", y);
13     change(y);
14     printf("%d\n", y);
15     return EXIT_SUCCESS;
16 }
```



Pile des  
appels de  
fonctions

# Les tableaux, un cas à part... pour le moment.

La modification des valeurs des cases d'un tableau passé en argument persiste, même après la fin de la fonction.

Une histoire de pointeurs, mais on verra cela plus tard.

# Les tableaux, un cas à part... pour le moment.

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 void exemple(int tab[10])
4 {
5     int i;
6     for(i=0;i<10;i++)
7     {
8         tab[i] = i*i;
9     }
10 }
11 int main()
12 {
13     int i;
14     int tableau[10];
15     for(i=0;i<10;i++)
16     {
17         tableau[i] = i;
18     }
19     //exemple(tableau);
20     for(i=0;i<10;i++)
21     {
22         printf("%d\u",tableau[i]);
23     }
24     return 0;
25 }
```

"C:\Users\FrançoisDelbot\Desktop\Mes projets C\exemple\bin\Debug\exemple.exe"

0 1 2 3 4 5 6 7 8 9

Process returned 0 (0x0) execution time : 0.024 s  
Press any key to continue.



# Les tableaux, un cas à part... pour le moment.

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 void exemple(int tab[10])
4 {
5     int i;
6     for(i=0;i<10;i++)
7     {
8         tab[i] = i*i;
9     }
10 }
11 int main()
12 {
13     int i;
14     int tableau[10];
15     for(i=0;i<10;i++)
16     {
17         tableau[i] = i;
18     }
19     exemple(tableau);
20     for(i=0;i<10;i++)
21     {
22         printf("%d\u",tableau[i]);
23     }
24     return 0;
25 }
```

"C:\Users\FrançoisDelbot\Desktop\Mes projets C\exemple\bin\Debug\exemple.exe"

0 1 4 9 16 25 36 49 64 81

Process returned 0 (0x0) execution time : 0.025 s

Press any key to continue.

# Exercise 1

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  int f1(int a, int b)
5  {
6      return a*b;
7  }
8  int f2(int x, int y)
9  {
10     int res;
11     res = x + f1(x,y);
12     return res;
13 }
14 int f3(int u, int v, int w)
15 {
16     int res;
17     res = f2(u,w) + f2(v,w);
18     return res;
19 }
20 int main(void)
21 {
22     printf("%d\n", f3(3,2,4));
23     return EXIT_SUCCESS;
24 }
```

- 1 Qu'affiche ce programme ?
- 2 Modifiez les fonctions f2 et f3 pour qu'elles n'utilisent plus de variables locales autres que les paramètres formels (elles ne doivent pas non plus utiliser de variables globales!).

# Exercice 1

## Retour sur les nombres narcissiques

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  int nombre_de_chiffres(int n);
5  int puissance(int x, int n);
6  int get_chiffre(int nombre, int i_chiffre);
7  int est_narcissique(int nb);
8
9  int main()
10 {
11     int nb;
12
13     printf("Entrez un entier : ");
14     scanf("%d", &nb);
15
16     if(est_narcissique(nb) == 1) printf("\n%d est un nombre narcissique", nb);
17     else printf("\n%d n'est pas un nombre narcissique", nb);
18
19     return EXIT_SUCCESS;
20 }
```

# Exercice 1

## Retour sur les nombres narcissiques (suite)

Ce code se trouve dans le même fichier source, en dessous de la fonction main.

```
1 int nombre_de_chiffres(int n)
2 {
3     int cpt = 0;
4     while(n >= 10)
5     {
6         cpt++;
7         n = n / 10;
8     }
9     return cpt + 1;
10 }
11
12 int puissance(int x, int n)
13 {
14     int i, res=1;
15     for(i=1; i<=n; i++)
16     {
17         res = res * x;
18     }
19     return res;
20 }
```

# Exercice 1

## Retour sur les nombres narcissiques (suite)

Ce code se trouve dans le même fichier source, en dessous des fonctions précédentes.

```
1 int get_chiffre(int nombre, int i_chiffre)
2 {
3     int i;
4     for(i=1; i<i_chiffre; i++)
5     {
6         nombre = nombre / 10;
7     }
8     return nombre%10;
9 }
10
11
12 int est_narcissique(int nb)
13 {
14     int i, nb_c, ch, somme;
15     somme = 0;
16     nb_c = nombre_de_chiffres(nb);
17     for(i=1; i<=nb_c; i++)
18     {
19         ch=get_chiffre(nb,i);
20         somme = somme + puissance(ch, nb_c);
21     }
22     if(somme == nb) return 1;
23     return 0;
24 }
```