Module ITC313 - Informatique

Partie C / C++

TD1 Fonctions, nombres aléatoires, fonctions récursives

Benoît Darties - benoit.darties@u-bourgogne.fr
Université de Bourgogne

Année universitaire 2016-2017

1 Variables en C : affichage, lecture, manipulation

Les exercices de cette section présentent des concepts élémentaires pour interagir avec l'utilisateur avec des fonctions `C` et non `C++`, via les fonctions `printf` et `scanf`. Ces fonctions sont plus délicates à manier que leurs équivalent `C++` `cin` et `cout`.

Exercice 1 : *Affichage de variables en C avec la fonction printf*

L'objectif de cet exercice est de créer un premier programme manipulant quelques variables et les affiche en utilisant la fonction `printf`, et de comprendre l'utilisation de cette fonction qui est plus déroutante au premier abord que son équivalent en `C++`. Cette fonction se révèle extrêmement complète et permet d'afficher des variables de différents types et sous différents formats. Nous ne ferons qu'effleurer ses possibilités ici.

Sur cet exercice, nous travaillerons avec le fichier `echauffement.c` présenté ci-après :

Listing 1 – Programme `echauffement.c`

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  int main() {
5    int a = 50;
6    char b = 'W' ;
7    char c;
8
9    printf("Ceci est un echauffement\n");
10
11   printf("nous sautons une ligne\npuis une autre\n");
12   printf("la valeur de la variable a est %i\n", a);
13   printf("la valeur de la variable b est %c\n", b);
14   printf("la valeur de la variable c est %c\n", c);
15
16   exit(0);
17 }
```

1. Récupérez ou recopiez le fichier `echauffement.c`. En vous positionnant dans le répertoire contenant ce fichier, quelle ligne de commande devez-vous exécuter pour compiler ce fichier en un programme exécutable nommé `echauffProg` ? Quelle commande devez-vous exécuter pour lancer ce programme ?

2. Compilez ce programme en un fichier exécutable `echauffProg` et exécutez ce programme. Que fait la fonction `printf()` ? Quel est son équivalent en `C++` ?
3. En vous aidant du code et du manuel de la fonction `printf()` accessible via la commande `man -3 printf`, répondez aux questions suivantes :
   (a) A quoi sert la fonction `printf()` ?
   (b) Comment affiche-t'on un retour à la ligne avec la fonction `printf()` ?
   (c) Dans la ligne , par quoi est remplacé le spécificateur de chaine '`%i`' à l'affichage ?
   (d) Dans la ligne , par quoi est remplacé le spécificateur de chaine '`%c`' à l'affichage ?
   (e) Dans cette même ligne, remplacez spécificateur de chaine '`%c`' par '`%i`. Recompilez, et exécutez à nouveau le programme. Quelle valeur est affichée et à quoi correspond-elle ?
   (f) Pour afficher la valeur d'un flottant, quel spécificateur de chaine faut-il utiliser ?
   (g) Rajoutez une instruction qui affiche sur la même ligne toutes les valeurs des variables `a`, `b`, et `c` dans cet ordre.
   (h) Créez une variable `d` de type `float`. Pour respecter les règles de rédaction d'un programme, l'ensemble des variables d'une fonction doit être déclarée en début de programme. Affectez-lui la valeur `25.57`, et ajoutez une instruction permettant d'afficher cette valeur à l'écran.

Exercice 2 : *Lecture de variables en C avec la fonction scanf*

Dans cet exercice, nous utilisons la fonction `scanf` pour lire une chaine au clavier, analyser ses différents éléments, et les stocker dans différentes variables.

1. **Lecture d'un entier :**
   (a) Recopiez ou récupérez le programme `scanf_1.c` présenté ci-après :

Listing 2 – Programme `scanf_1.c`

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  int main() {
5      int ello ;
6
7      printf("saisissez un entier et appuyez sur entree\n");
8      scanf("%i", &ello);
9
10     printf("vous avez saisi %i\n", ello);
11
12   exit(0);
13 }
```

   (b) Testez le programme plusieurs fois en donnant différentes valeurs à chaque exécution. Notez que l'on ne peut saisir des nombres au delà d'une certaine valeur, sans que ces derniers soient ramenés à une toute autre valeur du domaine. Notez que `scanf` prend un premier paramètre qui est un spécificateur de chaine entrée. Ici, ce spécificateur de chaine ne contient qu'un marqueur `%i`, qui correspond à un entier. Le paramètre suivant est alors l'adresse de la variable dans laquelle on doit stocker la valeur lue au clavier.
2. **Lecture de plusieurs valeurs à la suite :**
   (a) Recopiez ou récupérez le programme `scanf_2.c` présenté ci-après :

Remarques / questions : benoit.darties@u-bourgogne.fr

Listing 3 – Programme `scanf_2.c`

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  int main() {
5      int ello ;
6      char latan;
7
8      printf("saisissez un entier et un caractere et appuyez sur
           entree\n");
9      scanf("%i %c", &ello, &latan);
10
11     printf("vous avez saisi comme entier : %i\n", ello);
12     printf("vous avez saisi comme caractere : %c\n", latan);
13
14   exit(0);
15 }
```

(b) Testez le programme plusieurs fois en tapant un entier, suivi d'un espace, suivi d'un caractère, et en appuyant sur entree. Que se passe-t'il si vous mettez plusieurs espaces entre l'entier et le caractère ?

(c) Modifiez ce programme en créant différentes variables et en modifiant le spécificateur de chaines pour que l'utilisateur puisse saisir jusqu'à 5 entiers avec une seule fonction `scanf`.

(d) Récupérer la valeur renvoyée par la fonction `scanf` dans une variable `nbLus` et affichez la valeur de cette dernière à l'écran en fin de programme. Testez plusieurs fois votre programme, en faisant varier le nombre d'éléments que vous saisissez à chaque fois. Quel est le contenu de cette variable `nbLus` ?

3. **Lecture de plusieurs valeurs au milieu d'un flot :** Toute la puissance de `scanf` réside dans la possiblité d'extraire des données dans le flot de caractères entré en utilisant les délimiteurs de son choix, et pas simplement le caractère 'espace'.

(a) Recopiez ou récupérez le programme `scanf_3.c` présenté ci-après :

Listing 4 – Programme `scanf_3.c`

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  int main() {
5      int ello ;
6      int rus;
7      char latan;
8      int nbLus;
9
10     printf("saisissez une chaine et appuyez sur entree\n");
11     nbLus = scanf("123%iabc%cxyz:456%i",&ello,&latan,&rus);
12
13     printf("vous avez saisi comme 1er entier : %i\n", ello);
14     printf("vous avez saisi comme caractere : %c\n", latan);
15     printf("vous avez saisi comme 2er entier : %i\n", rus);
16     printf("nombre de valeurs correctement scannees: %i\n", nbLus);
17
18   exit(0);
19 }
```

(b) Testez le programme plusieurs fois en tapant comme chaines les valeurs suivantes :
— 14 D 28
— 12314iabcDxyz :45628

Remarques / questions : benoit.darties@u-bourgogne.fr

— 123456 D
— 12399iabczxyz :4561
— D
et notez à chaque fois les cas où le nombre d'éléments correctement scannés est supérieur à 0.

(c) Concluez sur l'utilisation de la fonction `scanf` en donnant une chaine, telle qu'à l'affichage, le premier entier saisi soit 50, le caractère soit Y et le second entier saisi soit 60.

2 Génération de nombres et tableaux aléatoires

Les exercices de cette section permettent de se familiariser avec la génération de nombres aléatoires en C avec la fonction `rand()`*, et d'apprendre à générer plusieurs tableaux aléatoires de deux façons différentes. Il est à noter l'importance ici de l'initialisation du générateur de nombres aléatoires, au travers de la fonction* `srand()`*.*

Exercice 3 : *Generation de nombres aleatoires*

Dans cet exercice, nous découvrons l'utilisation de la fonction `rand()`*, ainsi que son appel au sein d'une fonction. Cet exercice présente également les premiers aspects de définition et d'appel d'une fonction de manière intuitive.*

On vous donne le programme `randomNumber.c` suivant :

Listing 5 – Programme `RandomNumber.c`

```
1  #include <iostream>
2  #include <stdio.h>
3  #include <stdlib.h>
4  #include <time.h>
5
6  using namespace std;
7
8  int main() {
9      // declaration de trois variables entieres
10     int a, b, c;
11
12     // ligne suivante  a ecrire une fois seulement
13     // initialise le generateur de nombres aleatoires
14     srand(time(NULL));
15
16     a = rand()  % 100;
17     b = 10 + rand()  % 30;
18     c = -100 + rand()  % 201;
19
20     cout << "a = "<< a << " b = " << b << " et c = "<< c << endl;
21
22     exit(0);
23 }
```

1. Récupérez ou recopiez le programme, et exécutez-le quelques fois. Les variables a, b et c contiennent toutes les trois des nombres aléatoires. Quelles sont les valeurs minimales et maximales que peuvent prendre chacun de ces nombres ?
2. Modifiez ce code de la façon suivante : entre les lignes 7 et 8, ajoutez une fonction `nbAleatoire()` qui prend deux paramètres `min` et `max` de type `int` et qui retourne une valeur de type `int` correspondant à un nombre aléatoire entre `min` et `max` *inclus*. Pour vous aider, on vous donne le squelette de cette fonction, à compléter et à ajouter dans le programme `randomNumber.c`

Remarques / questions : benoit.darties@u-bourgogne.fr

Listing 6 – fonction `nombreAleatoire()` (à completer)

```
1 int nbAleatoire(int ... , int ...) {
2 // @params :  deux nombres
3 // @retour : une valeur aleatoire comprise entre ces deux nombres
4   ...
5   return ...;
6 }
```

(les parties marquées avec trois points `...` doivent être remplacées avec une variable ou une ou plusieurs instructions) :

3. Modifiez alors les instructions des lignes initialisant les variables `a, b` et c, de sorte que ces dernières soient initialisées en appelant la fonction `nbAleatoire()` , en adaptant à chaque fois la valeur des paramètres de sorte à ne pas changer les valeurs min et max des variables.
4. L'importance de la graine :
   (a) Ajoutez une boucle de 10 itérations, contenant *uniquement* les instructions d'affectation et d'affichage des trois variables `a, b` et c, de sorte à exécuter 10 tirages successifs aléatoires (l'instruction d'initialisation du générateur de nombres aléatoires ne doit pas être contenue dans cette boucle). Exécutez et lancez le programme pour afficher dix tirages aléatoires. Recommencez l'opération plusieurs fois pour vérifier que les valeurs sont bien aléatoires.
   (b) Commentez ensuite la ligne `srand(time(NULL);` afin de désactiver l'initialisation du générateur de nombres aléatoires, compilez, et relancez plusieurs fois le programme. Que constatez-vous ?
   (c) Décommentez ensuite la ligne `srand(time(NULL);` et remplacez la par `srand(`*nombre*`);` , où *nombre* est une valeur de votre choix. Compilez, et relancez plusieurs fois le programme. Que constatez-vous ?
   (d) Concluez sur l'importance de la graine : en terme de sécurité, pourquoi n'est-ce pas une bonne chose de choisir comme valeur de départ la valeur `time(NULL)` pour la graine ?

Exercice 4 : *Generer des tableaux aleatoires*

Avec le travail réalisé dans l'exercice précédent, nous disposons des outils permettant de générer des nombres aléatoires entre deux valeurs données. Dans cet exercice, nous allons créer des tableaux contenant des valeurs aléatoires. Ces tableaux seront utilisés par la suite pour appliquer nos algorithmes de tris

Pour générer des tableaux aléatoires, nous présentons ici deux approches :
— La première consiste à générer de manière séquentielle une suite de valeurs aléatoires.
— La seconde consiste à mélanger un tableau existant et dont les valeurs pouvaient être triées. La seconde méthode offre l'avantage de faire des permutations sur des tableaux dont les valeurs sont toutes différentes au départ.

Pour vous aider, on vous donne le programme `tableauxAleatoires.c` présenté ci-après, qu'il conviendra de modifier en complétant les parties en commentaires marquées `[a completer]`.

1. Génération séquentielles de valeurs aléatoires :
   (a) Ecrivez au sein de la fonction `main()` l'instruction permettant de définir un tableau `tab` de 15 valeurs de type `int`. Ajoutez également (à l'extérieur de la fonction `main()`) l'ensemble de la fonction `nbAleatoire()` définie dans l'exercice précédent (à placer avant la fonction `main()`) de sorte à pouvoir appeler cette dernière au sein de votre programme.

Remarques / questions : benoit.darties@u-bourgogne.fr

Listing 7 – programme `tableauxAleatoires.c`

```c
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

// [a completer] Q. 1.a : ajouter fonction nbAleatoire()

int main() {
    int i, u, v, temp;
    // [a completer] Q. 1.a : ajouter definition de tab
    // [a completer] Q. 2.a : ajouter definition de tab2

    // [a completer] Q. 1.b : generateur de nb aleatoires srand()

    // PARTIE 1 : Generation sequentielle de valeurs aleatoires dans un tableau
    // initialisation de tab avec des valeurs aleatoires entre 0 et 30
    for (i=0; i < 15; i++) {
        // [a completer] Q. 1.c : affecter tab[i] avec une valeur aleatoire
    }

    // affichage de tab
    printf("affichage du tableau tab :\n");
    for (i=0; i < 15; i++) {
        // [a completer] Q. 1.d : afficher tab[i]
    }

    // PARTIE 2 : Melange d'un tableau existant
    // creation du tableau [0, 1, 2, ... 14]
    for (i=0; i < 15; i++) {
        // [a completer] Q. 2.a : affecter tab[i] avec la valeur i
    }
    printf("affichage du tableau tab2 avant melange :\n");
    for (i=0; i < 15; i++) {
        // [a completer] Q. 2.a : afficher tab[i]
    }

    // melange du tableau
    for (i=0; i < 100; i++) {
        // [a completer] Q. 2.b : melanger le tableau comme indique
        // indice : utiliser u, v, temp et nbAleatoire
    }

    // affichage de tab2
    printf("affichage du tableau tab2 apres melange :\n");
    for (i=0; i < 15; i++) {
        // [a completer] Q. 2.c : afficher tab[i]
    }

    exit(0);

}
```

(b) Rajoutez une instruction permettant d'initialiser le générateur de nombre aléatoires (n'oubliez pas de rajouter les en-têtes nécessaires)

(c) Ajoutez une boucle qui permet d'initialiser chaque cellule du tableau avec une valeur aléatoire comprise entre 0 et 30, en utilisant la fonction `nbAleatoire()`.

(d) Ajoutez les instructions permettant d'afficher le contenu du tableau `tab` à l'écran (utilisez une boucle pour l'affichage si nécessaire). Vérifiez que les valeurs sont bien aléatoires en exécutant plusieurs fois le programme.

2. Mélange d'un tableau existant :

(a) Ajoutez à votre programme un second tableau `tab2` de 15 valeurs de type `int` non initialisés, puis ajoutez une boucle pour que chaque cellule du tableau contienne comme valeur son propre indice. Le tableau doit alors contenir les valeurs 0 à 14 dans cet ordre. Vérifiez en affichant le tableau avant mélange.

(b) Ajoutez ensuite une boucle de 100 itérations qui effectue les opérations suivantes :
Pour chaque tour de boucle :
— deux valeurs `u` et `v` sont tirées aléatoirement entre 0 et 14 ;
— les valeurs des cellules `u` et `v` sont alors inversées.
On recommence ces opérations à chaque tour de boucle.

(c) Ajoutez les instructions permettant d'afficher le contenu du tableau `tab2` à l'écran (utilisez une boucle pour l'affichage si nécessaire). Vérifiez que les valeurs sont bien mélangées, et qu'il s'agit bien d'une permutation des nombres 0 à 14, en exécutant plusieurs fois le programme.

3 Organisation du code en fonctions

L'exercice de cette section est une synthèse sommaire permettant de comprendre comment le code doit être organisé en fonctions. L'organisation détaillée du code dans différents fichiers, et les mécanismes de compilation associés, seront vus plus tard

Exercice 5 : *Tout organiser autours de fonctions*

L'objectif de cet exercice est de vous familiariser avec le concept d'organisation du code autours de fonctions. Certains aspects des fonctions seront volontairement ignorés afin d'aller au plus simple. Il n'y a pas de piège sur les appels de fonctions

Dans cet exercice, il vous est demandé de reprendre ce qui a été entrepris dans les exercices et de l'organiser désormais dans des fonctions, pour obtenir une version définitive et propre de votre travail. Pour cela on vous donne un squelette de programme présenté ci-après. L'objectif est d'avoir des fonctions qui sont indépendantes de la taille d'un tableau, cette dernière devant être passée en paramètre de chaque fonction, et qui pourront ainsi être utilisées sur des tableaux de tailles différentes. Nous utiliserons dans cet exercice plusieurs tableaux dont les tailles pourront varier. Pour vous aider, on vous donne le programme `fonctionsTableaux.c` qu'il conviendra de modifier en ne changeant que les éléments en commentaires mentionnés `[a completer]`,**et rien d'autre**. Avant d'effectuer toute modification, notez notamment que :

— Nous utilisons une directive de pré-processing. En effet, la taille des tableaux est précisée dans la directive suivante :

```
1 #define TAILLE1 20
2 #define TAILLE2 30
```

Ceci signifie qu'à chaque fois que le mot `TAILLE1` sera rencontré, il sera remplacé par la valeur `20` dans le code, et respectivement pour `TAILLE2` et la valeur `30` ;

— Les fonctions qui ne doivent renvoyer aucune valeur (aussi appelée procédures) sont affectées au type `void`.

Les modifications à opérer dans l'ordre sont présentées ci-après. Chaque modification peut-être testée séquentiellement, si les modifications précédentes ont été correctement effectuées, le programme étant déjà compilable et exécutable.

1. Complétez et testez la fonction `afficherTab()` pour que cette dernière affiche le contenu d'un tableau passé en paramètre.
2. Complétez et testez la fonction `remplirTabValeursAleatoires()` pour que cette dernière remplisse de manière aléatoire le contenu d'un tableau passé en paramètre.
3. Complétez et testez la fonction `remplirTabValeurs1aN()` pour que cette dernière remplisse de manière un tableau passé en paramètre avec les valeurs 1, 2, ... N.
4. Complétez et testez la fonction `melangerTab()` pour que cette dernière mélange les valeurs d'un tableau précédemment rempli.

Listing 8 – programme `fonctionsTableaux.c` (à completer)

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <time.h>
4
5  #define TAILLE1 20
6  #define TAILLE2 30
7
8  // declaration des prototypes de fonctions qui seront
9  // utilisees dans la suite de ce programme
10 // (cette partie ne doit pas etre modifiee)
11 int nbAleatoire(int min, int max);
12 void afficherTab(int tab[], int taille);
13 void remplirTabValeursAleatoires(int tab[], int taille, int min, int max);
14 void remplirTabValeurs1aN(int tab[], int taille);
15 void melangerTab(int tab[], int taille, int nb);
16 // fin de declaration des prototypes de fonctions
17
18 // definition des fonctions
19
20 int nbAleatoire(int min, int max) {
21     // @params : deux nombres
22     // @retour : une valeur aleatoire comprise entre ces deux nombres
23
24     // [a completer] ajouter code de la fonction nbAleatoire()
25 }
26
27 void afficherTab(int tab[], int taille) {
28     // @params : tab : tableau d'entiers, taille :taille du tableau
29     // @action : affiche le tableau a l'ecran
30
31     // [a completer] ajouter boucle pour afficher le tableau
32 }
33
34 void remplirTabValeursAleatoires(int tab[], int taille, int min, int max) {
35     // @params : tab : tableau d'entiers, taille :taille du tableau
36     //           min et max : deux entiers donnes avec min < max
37     // @action : remplit le tableau avec les valeurs aleatoires tirees
          entre
38     //           les valeurs min et max
39
40
41     // [a completer] ajouter boucle pour remplir le tableau
42 }
43
44 void remplirTabValeurs1aN(int tab[], int taille) {
```

```
45      // @params : tab : tableau d'entiers, taille :taille du tableau
46      // @action : remplit le tableau avec les valeurs 1, 2, 3 ... N
47
48      // [a completer] ajouter boucle pour remplir le tableau
49
50
51  }
52
53  void melangerTab(int tab[], int taille, int nb) {
54      // @params : tab : tableau d'entiers, taille :taille du tableau
55      //           nb : nombre d'inversions a faire
56      // @action : inverse nb fois deux valeurs aleatoires du tableau
57
58      // [a completer] ajouter boucle pour remplir le tableau
59  }
60
61
62  int main() {
63      // note : la fonction main() ne doit pas etre modifiee
64      int tab1[TAILLE1];
65      int tab2[TAILLE2];
66      int compteur;
67      int tabTest[6] = {1, 1, 2, 3, 5, 8};
68      int tabTest2[3] = {13, 21, 33};
69
70      srand(time(NULL)); // initialise le generateur de nb aleatoires
71
72      // Test de la fonction d'affichage une fois cette derniere ecrite
73      printf("Test de la fonction d'affichage\n");
74      afficherTab(tabTest, 6); // doit afficher "1 1 2 3 5 8"
75      afficherTab(tabTest, 3); // doit afficher "13 21 33"
76
77      // Test de la fonction de generation de tableau aleatoires
78      printf("Test de la fonction de generation de tableaux aleatoires\n");
79      remplirTabValeursAleatoires(tab1, TAILLE1,0, 100);
80      afficherTab(tab1, TAILLE1);
81
82      // Test de la fonction de remplissage de tableau avec les valeurs 1 a N
83      printf("Test de la fonction pour remplir le tableau de 1 a N\n");
84      printf("affichage du tableau tab2 avant melange :\n");
85      remplirTabValeurs1aN(tab2, TAILLE2);
86      afficherTab(tab2, TAILLE2);
87
88      // Test de la fonction de melange de tableau
89      printf("affichage du tableau tab2 apres melange :\n");
90      melangerTab(tab2, TAILLE2, TAILLE2*TAILLE2);
91      afficherTab(tab2, TAILLE2);
92
93    exit(0);
94  }
```